Parte I

•

IDEAFIX

Instalación y Puesta en Marcha

Sección 1 Instalando IDEAFIX

1

Instalando IDEAFIX

Capítulo 1

Proceso de Instalación

Pasos previos a la instalación de Ideafix

Antes de comenzar la instalación del port de ***Ideafix*** , deberá loguearse al sistema como el usuario "root". Si no se utiliza este usuario, la instalación no podrá realizarse correctamente, pues se necesita de permisos especiales para la misma.

Otro requisito fundamental para poder compilar y/o desarrollar con ***Ideafix*** , es que se encuentre instalado el "System Development" del sistema operativo. Este es un conjunto de utilidades que permite la compilación de programas en el mismo. En algunos *UNIX* este sistema no se incluye y se vende por separado, o requiere ser instalado, pues si se instaló el S.O. con la configuración *"default"* este puede no incluirse.

Para saber si está instalado el "System Development" basta con ejecutar el siguiente comando (preferentemente ejecutarlo como root):

```
# make
```

Si muestra por pantalla un mensaje similar al siguiente:

```
make: *** No targets. Stop.
```

significa que se encuentra instalado; si el mensaje es en cambio:

```
make: command not found o make: not such file or directory
```

éste no se encuentra instalado.

Después de haberse logueado como "root" en el sistema, hay que bajar el port del medio en el cual se encuentra alojado. Este medio por lo general es una cinta o una unidad de CD-ROM.

El port que se distribuye constará (en general, ver *"Particularidades de las diferentes versiones de UNIX"*) de dos archivos tar; por ejemplo:

- Ideafix52-*sistema_operativo*.tar
- gcc*XXX*.tar

El primero es el port de Ideafix correspondiente a la versión del Sistema Operativo (S.O.) *UNIX* que usted requirió. El segundo es el compilador C/C++ de GNU que utiliza el ***Ideafix*** para compilar.

Para extraer los archivos de la unidad de cinta se debe ejecutar el siguiente comando:

```
# cd <directorio temporal>
# tar xvf <device>
```

Donde <device> es el dispositivo de cinta, por ejemplo /dev/rmt0, /dev/rmt/0... etc. Este comando extraerá de la cinta los dos archivos antes mencionados,

Para el caso en que el port este en CD-ROM, existen 2 opciones:

- La primera es montar el CD en la unidad lectora de CD-ROM del equipo *UNIX* con el siguiente comando:

```
# mount <device> <mount-point>
```

Donde <device> es el dispositivo de la lectora, y <mount-point> es el directorio donde se quiere montar, este ultimo debe existir y estar vacío. Luego simplemente se copian los archivos con el comando cp, o se extrae directamente del CD con el comando tar, tal y como se muestra mas abajo.

Por ejemplo si <device> es /dev/cdrom (como en Linux) y <mount-point> /cdrom la sintaxis seria la siguiente:

```
# mount /dev/cdrom /cdrom
```

En ocasiones es necesario indicar el tipo de “file system” del dispositivo, los CD-ROM de *InterSoft* se distribuyen en formato ISO-9660. Para indicarle al comando mount el tipo de “file system” consulte el manual del *UNIX*.

- La segunda, si no se tiene lectora de CD en el equipo *UNIX* y este se encuentra conectado a una red, es utilizar la lectora de una PC y pasar los archivos usando FTP (modo binario).

Instalación del compilador GNU

Para instalar el compilador debe moverse al directorio /usr/local. Si este directorio no existe debe crearlo. Y luego abrimos en archivo del gcc de la siguiente manera:

```
# cd /usr/local
# pwd
/usr/local
# tar xvf /tmp/gccXXX.tar
```

NOTA: Si está realizando una actualización de versión de Ideafix, entonces ya tiene instalado el compilador y puede obviar este paso. Sólo debería verificar que la versión que posee sea la misma que la del nuevo port.

Luego entrar al directorio bin y probar la correcta instalación del gcc de la siguiente manera:

```
# pwd
/usr/local
# cd bin
# pwd
/usr/local/bin
# ./gcc -v
Reading specs from
/usr/lib/gcc-lib/i386-redhat-linux/2.7.2/specs
gcc version 2.7.2
```

Si muestra unas líneas semejantes a las ultimas dos: significa que se realizó correctamente la instalación del gcc y ahora podemos pasar a la instalación del ***Ideafix*** .

NOTA: en el caso del *Linux* 4.x (ya obsoleto) no es necesario realizar la instalación del gcc puesto que viene con el sistema operativo, pero igualmente se incluye el gcc en la distribución de ***Ideafix*** ; y en el caso del Linux 5.x o 6.x se ha realizado un cambio de compilador por lo cual no se requiere la instalación del gcc (ver *"Particularidades de las diferentes versiones de UNIX"*).

Instalación de Ideafix

Ahora vamos a realizar la extracción de los archivos incluidos en el Ideafix52.tar. Para ello vamos a elegir un directorio donde se instalará ***Ideafix*** . Tomemos por ejemplo /usr2/idea52, ahora lo crearemos y abriremos el archivo Ideafix52.tar

```
# mkdir /usr2/idea52
# cd /usr2/idea52
# tar xvf /tmp/idea52.tar
```

NOTA: Si se utilizarán motores Oracle o Informix ver mas abajo la creación de los links.

Ahora deberemos dar los permisos necesarios a los programas utilitarios y binarios del port de ***Ideafix*** , y realizar la activación del mismo para que este pueda ser utilizado. Tener en cuenta que si no se realiza la activación, NO se podrá utilizar ningún utilitario de ***Ideafix*** .

Para dar los permisos necesarios a los utilitarios del port hay que correr un shell script que se encuentra en el directorio "once", a partir de donde se bajo el port, y que se llama "init.ALL". Para correrlo, hay que hacer lo siguiente desde la línea de comandos del shell, estando ubicado en el directorio donde se bajo el port de ***Ideafix*** :

```
# pwd
/usr2/idea52
# sh ./once/init.ALL
```

Cuando este shell se comience a ejecutar, se darán los permisos adecuados a todos los utilitarios de ***Ideafix*** . Una vez que se haya terminado con los permisos, el shell script procederá a realizar preguntas, que servirán para el armado de un profile modelo, que se podrá utilizar para setear un ambiente adecuado para correr los utilitarios de ***Ideafix*** .

Cabe recordar que si es una reinstalación o la instalación de un nuevo Patch; será necesario eliminar el link de las librerías dinámicas /SH520. Debido a que el proceso de instalación abortará.

A continuación se muestran las diferentes etapas del proceso de instalación

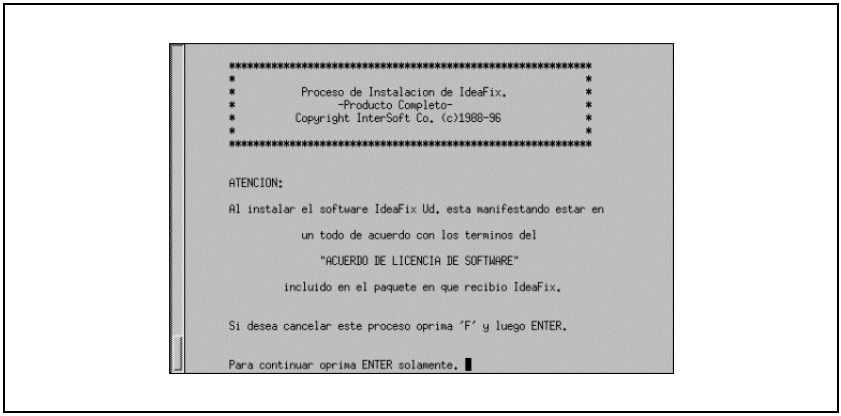


Figura 1.1

En esta instancia sólo es necesario pulsar ENTER.

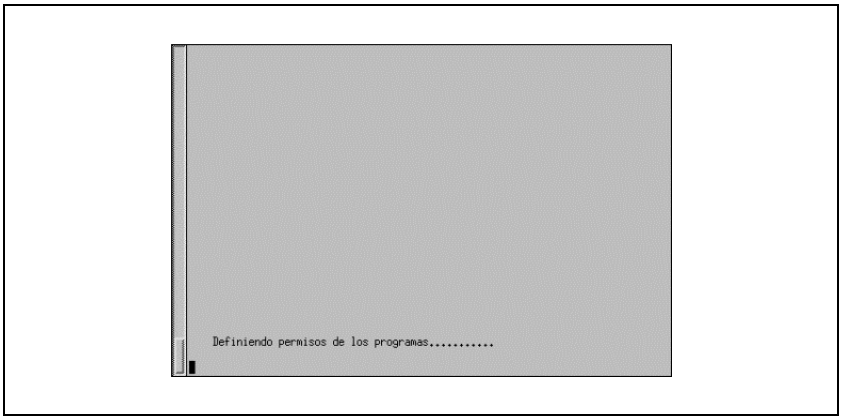


Figura 1.2

En este momento el script de instalación se encuentra seteando los permisos necesarios para todos los programas y utilitarios de ***Ideafix*** . Este proceso puede llevar varios minutos, al finalizar se mostrará la siguiente pantalla:

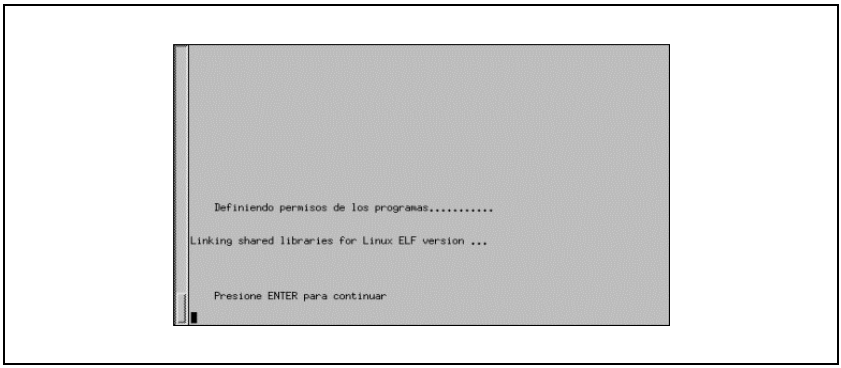


Figura 1.3

Luego de presionar ENTER, aparecerán una serie de preguntas; necesarias para la configuración del perfil de los usuarios de ***Ideafix*** . Si no sabe que contestar; simplemente presione ENTER y el valor “default” será establecido.

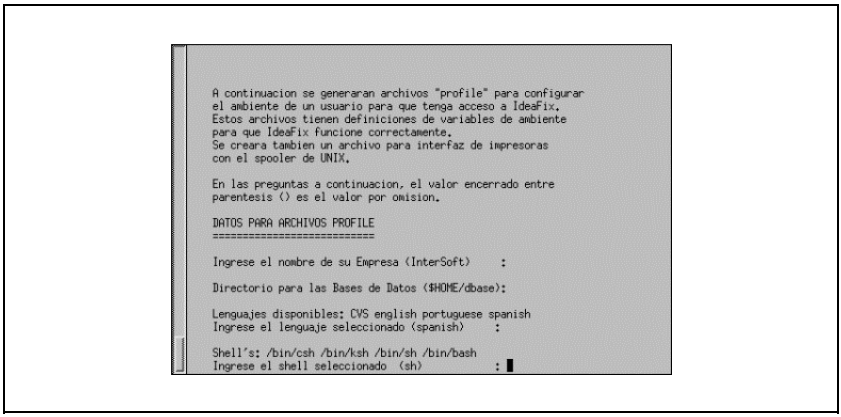


Figura 1.4

Al ingresar el nombre de la empresa, este será establecido en la variable de ambiente “empresa”, el directorio de la base de datos se establecerá en la variable “dbase”. El lenguaje que usted establezca será en el cual ***Ideafix*** mostrará sus mensajes, el shell es el interprete de comandos que usted piensa utilizar.

Todas estas variables y las que siguen pueden ser modificadas o adaptadas luego del proceso de instalación, por lo cual si tiene alguna duda, o si aún no ha pensado alguna de ellas, simplemente presione ENTER y luego podrá setearlas según sus necesidades

Luego será consultado sobre el tipo de impresora que utilizará. El seteo de impresoras y su puesta a punto es un proceso un tanto complejo que se describirá en una seccion posterior de este manual. Simplemente limítese a constestar el modelo de la impresora o presione ENTER.

A continuación usted verá la siguiente pantalla:

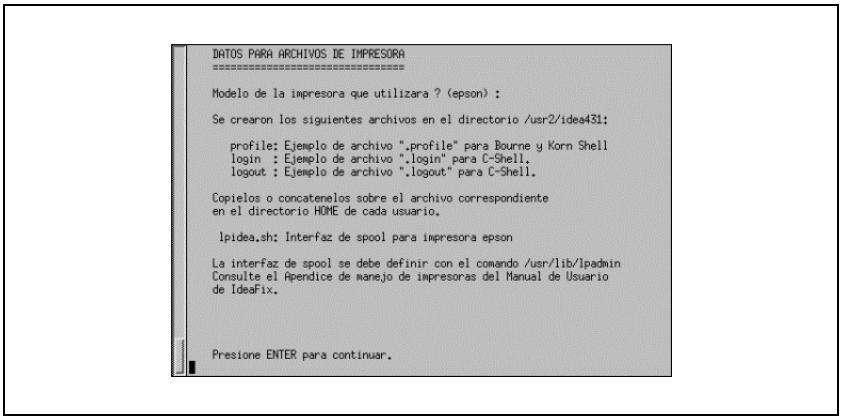


Figura 1.5

Presione ENTER para continuar. Luego le aparecerá una pantalla con los módulos instalados (si ya tenía Ideafix instalado anteriormente o de lo contrario aparecerá una pantalla que le indicará comunicarse con *InterSoft* para obtener su número de instalación (el proceso se describe luego de ésta sección).

Presione ENTER.

A continuación observará una pantalla que indica que el módulo ***Ideafix*** ha sido instalado y le requerirá que presione ENTER para continuar.

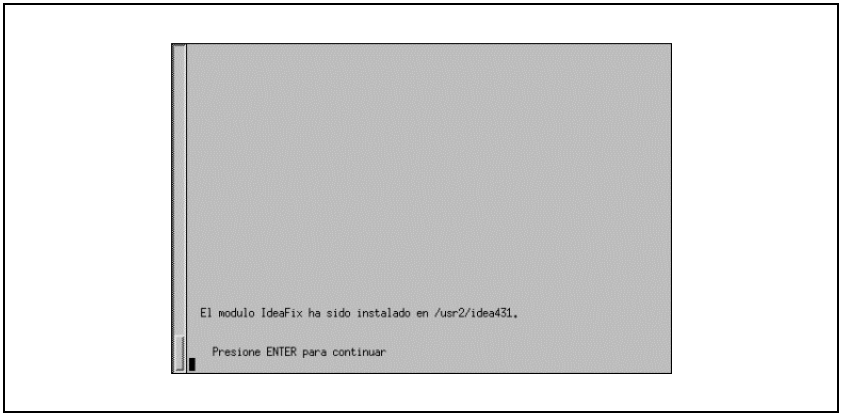


Figura 1.6

Luego verá otra que le preguntará que administrador de versiones utiliza. El administrador de

versiones se utiliza para ambientes de programación con varias personas trabajando sobre los mismos fuentes, si usted utiliza uno en particular; seleccione el adecuado. Si no hace uso de alguno; simplemente presione ENTER.

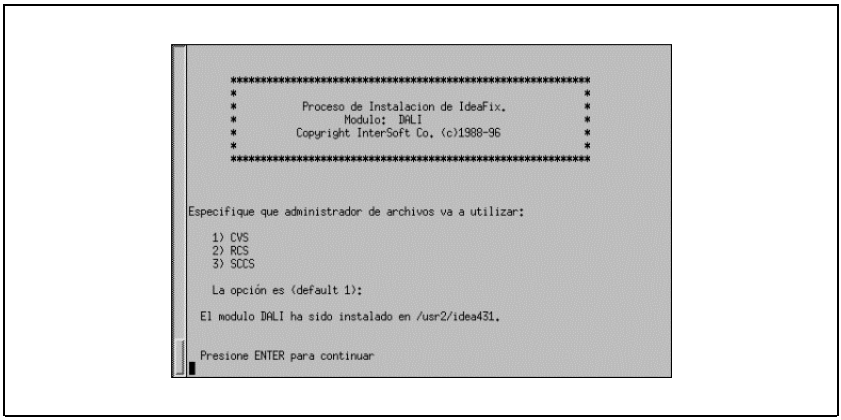


Figura 1.7

Luego de lo cual el modúlo ***Dalí*** habrá sido instalado. ***Dalí*** es un editor de texto que reemplaza al antiguo ***ie*** (*InterSoft* Editor). Tiene una apariencia similar al Edit del D.O.S. y es un muy útil entorno que facilita la programación y la edición de archivos de ***Ideafix*** , como ser los forms y reports.

Encontrará que el ***Dalí*** es un editor simple y fácil de usar, con una interfaz con el usuario un tanto escaza en el ambiente *UNIX*. Y seguramente le brindará a usted una buena herramienta de trabajo.

Luego verá la última pantalla en la que se indica que el módulo ***Essentia*** ha sido instalado. ***Essentia*** es el motor de base de datos propio de *InterSoft*, de tecnología simple y elegante que le brindará algunas prestaciones que están reservadas para motores de bases de datos de mayor costo.

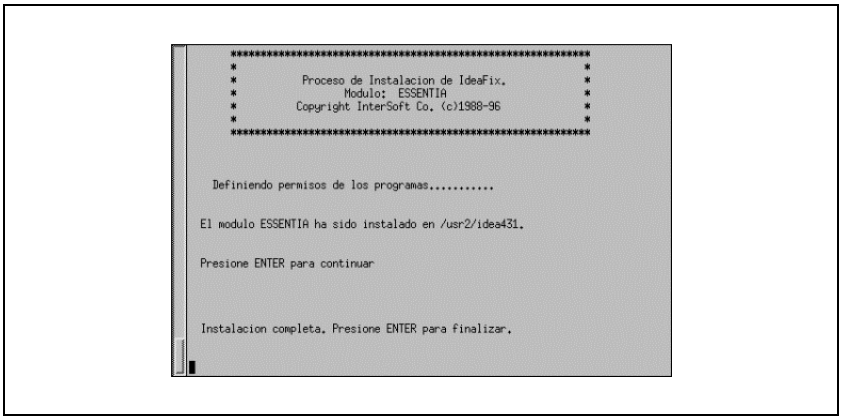


Figura 1.8

Para mayor información sobre Essentia, consulte la Parte IV - Sección 1 de este manual.

De esta manera queda completa la instalación.

Activación de Ideafix

Para realizar la activación (luego de haber dado los permisos a los utilitarios de ***Ideafix***), hay que correr un utilitario provisto con el ***Ideafix*** . Este binario se llama "ixset", y se encuentra en el directorio "bin", a partir de donde se instaló el port de ***Ideafix*** en el disco (En nuestro ejemplo: /usr2/idea52).

Para proseguir con la activación deberá contactarse con *InterSoft* S.A. y pedir un número de instalación. Una vez que se le haya asignado un número de instalación, deberá hacer lo siguiente desde la línea de comandos y desde donde se había bajado el port de ***Ideafix*** : (con usuario ROOT)

```
# ./bin/ixset -f <número de instalación asignado>
Instalation: 999999 Serial number:
123456789-1234567890-123456789
```

Este comando le devolverá un numero de serie, que usted deberá comunicar a la persona encargada de las instalaciones en *InterSoft* S.A. Con este número de serie, usted obtendrá las claves de activación para cada uno de los paquetes del port de ***Ideafix*** que desee instalar y/o utilizar.

Para la activación de cada uno de los paquetes en los que usted estará interesado, deberá hacer lo siguiente desde la línea de comandos:

```
# cd bin
```

```
# ./ixset <número de paquete> - <clave de activación del
paquete> - <meses> - <usuarios>
```

Si desea saber cuales son los paquetes que usted puede instalar y/o el estado de activación de los mismos, puede hacer lo siguiente desde la línea de comandos:

```
# ./ixset -h
```

Este comando le mostrara la siguiente tabla con todos los paquetes disponibles, y su estado de activación, que puede ser "activo" o no.

Una vez que estas dos tareas fueron realizadas, se puede dar por completa la instalación y activación del port de ***Ideafix*** .

Ideafix sobre motores SQL (Oracle/Informix)

Si se va a utilizar un motor SQL como base de datos, hay que crear una base llamada metadata que hace las veces de los archivos ".sco" de Ideafix52 o los $*.* de Essentia. Para realizar esto, se debe tener ya instalado el motor a utilizar. Entonces realizamos lo siguiente:

```
# pwd
/usr2/idea52/
# cd include/sqldb/metadata
# pwd
/usr2/idea52/include/sqldb/metadata
```

En este directorio encontraremos unos archivos con extensión sql que tienen las sentencias SQL necesarias para crear el metadata.

Si se utilizará un motor **Oracle**, loggearse como usuario oracle y hacer lo siguiente:

```
# sqlplus system/manager
SQL>@createmetadata (esto crea el usuario metadata)
SQL>@metadata (esto crea la base metadata)
SQL>quit
```

Se deben crear dos links para que las herramientas de Ideafix puedan ser ejecutadas. Uno en $IDEAFIX/lib y el otro en $IDEAFIX/SH520, ambos con el usuario root. Por lo tanto:

En $IDEAFIX/lib ejecutar:

```
# ln –s liboracledriver.a libsqlimpl.a
```

En $IDEAFIX/SH520 ejecutar:

```
# ln –s liboracleimpl51_s.so libsqlimpl51_s.so
```

De utilizar un motor **Informix**, loggearse como usuario informix y hacer lo siguiente:

1. Crear un dbspace.

2. Moverse al directorio $IDEAFIX/etc; en ese directorio hay un archivo llamado informixdriver que contiene dos líneas como estas:

```
TEMP_DBSPACE=

DATA_DBSPACE=
```

Estas líneas deben ser completadas con el nombre del dbspace creado en el punto 1.

3. Volver al directorio $IDEAFIX/include/sqldb/metadata

4. Correr el dbaccess; elegir Query-language, Choice, metadataifx, Run. (esto crea la base metadata)

5. Crear los links

En $IDEAFIX/lib ejecutar:

```
# ln –s libifxdriver.a libsqlimpl.a
```

En $IDEAFIX/SH520 ejecutar:

```
# ln –s libifximpl52_s.so libsqlimpl52_s.so
```

Si el motor está en otro servidor y se va a utilizar un cliente, hay que pasar los archivos .sql correspondientes para realizar la creación de la metadata.

Capítulo 2

Proceso de Configuración

Configuración del ambiente de los usuarios

En el lugar en el cual se bajo el port, habrá tres archivos que podrán ser utilizados para setear un ambiente adecuado, para usar los utilitarios de ***Ideafix*** . Estos shell scripts se encargan de setear variables de ambiente necesarias por los utilitarios de ***Ideafix*** . Para mayor información sobre las variables de ambiente referirse al manual del usuario de ***Ideafix*** . Estos tres archivos son:

- "profile": para korn, bourne y bash Shell.
- "login" y "logout": para "sh" shell.

Si se va a usar un motor SQL como base de datos, hay que trabajar un poco más. Dentro de estos archivos (profile o login) hay que descomentar y/o completar los datos referentes a cada motor. Esto es:

1. Variables de ambiente correspondientes a cada motor (por ejemplo, ORACLE_HOME, INFORMIXSERVER, etc)

2. Agregar el path de las librerías del motor en la variable LD_LIBRARY_PATH

3. Moverse a $IDEAFIX/bin y correr un programa llamado gensqlfw.exe; este programa genera un archivo de configuración en el home del usuario. Debe haber uno para cada usuario que quiera acceder a un esquema creado en un motor SQL.

Al ejecutarlo se abrirá un formulario donde se deben ingresar los siguientes datos:

- User id (con F1 aparece la lista de usuarios).
- DSN: I (informix) u O (oracle).

- El nombre de la instancia.
- El usuario.
- La password del usuario. (se guarda encriptada en el archivo)

Al presionar F5, en el home del usuario, se creará un archivo llamado .sqlfwrc.uid, donde uid es el id del usuario. Sin este archivo no se puede hacer nada sobre el motor.

4. En $IDEAFIX/etc existe un archivo llamado isoftdb.conf en el cual se debe especificar para cada esquema a qué base de datos pertenece (ideafix, essentia o sql). Debe ir un esquema por línea. Este archivo debe existir pero puede estar vacío, en cuyo caso debe haber en el ambiente una variable llamada ISOFTDB conteniendo lo mismo con la siguiente estructura:

```
esquema=base:esquema=base: ....
```

Es decir, los ":" es el separador.

Ejemplo:

```
# export ISOFTDB=aurus=essentia:tesor=sql:sumin=essentia
```

Configuración de los Servicios de Ideafix

Para el uso de todos o algunos de los siguientes productos de *InterSoft* S.A., a saber: ***GraphicInterface*** , ***ODBC/JDBC*** , o simplemente para utilizar el ***Iql Server*** y/o los servers ***Essentia*** en forma remota; se deberán configurar los servicios que permiten el uso de los mismos.

Para lo cual se deben agregar las siguientes líneas en el archivo /etc/services:

- Para el uso de ***IxGIF*** :

```
iexec 7140/tcp # GIF
```

- Para el uso de servers ***Essentia*** remotos:

```
sha-man 6100/tcp # Essentia shadow
```

- Para el uso del ***Iql Server*** y ***ODBC/JDBC*** :

```
Environ 7100/tcp # Iql Server
```

Luego es necesario agregar las siguientes líneas al archivo /etc/inetd.conf:

```
sha-man<\t>stream<\t>tcp<\>nowait<\t>root<\t>/<idea-dir>/bin/shadow<\t>
!e /etc/shadow.env –b –f /tmp/shadow.err
Environ<\t>stream<\t>tcp<\t>nowait<\t>root<\t>/<idea-dir>/rbin/iqlsrv<\
!e /etc/iqlsrv.env –b –f /tmp/iqlsrv.err
```

```
iexec<\t>stream<\t>tcp<\t>nowait<\t>root<\t>/<idea-dir>/bin/irexecd<\t>
!e /etc/iexec.env –b –f /tmp/iexec.err
```

Donde <\t> significa que se debe insertar un TAB (tabulador) donde no se indica puede utilizarse un TAB o un espacio; e <idea-dir> es el directorio donde se encuentra instalado el ***Ideafix*** . Es muy importante la sintaxis en este archivo y se debe tener cuidado al momento de su edición.

La opción !e requiere un archivo de configuración para el servicio en cuestión, esta configuración es la default del servicio y luego podrá ser modificado por cada usuario, se debe indicar el path completo hasta dichos archivos.

Los archivos de configuración deben setear las siguientes variables (mínimamente):

- shadow.env

```
IDEAFIX=/usr2/idea52
LANGUAGE=spanish
DATADIR=$IDEAFIX/data
PATH=$IDEAFIX/bin:$IDEAFIX/bin/$LANGUAGE:$IDEAFIX/ixdev/tools:$IDEAFIX/
ESSENTIA=$IDEAFIX/essentia
SERVERS=ianus;aurus
```

- iexec.env:

```
IDEAFIX=/usr2/idea52
LANGUAGE=spanish
PATH=$IDEAFIX/bin:$IDEAFIX/bin/$LANGUAGE:$IDEAFIX/ixdev/tools:$IDEAFIX/
DATADIR=$IDEAFIX/data
LD_LIBRARY_PATH=/SH520
ISOFTDB=demo=ideafix:perms=ideafix . . .
```

- iqlsrv.env:

```
IDEAFIX=/usr2/idea52
LANGUAGE=spanish
DATADIR=$IDEAFIX/data
PATH=$IDEAFIX/bin:$IDEAFIX/bin/$LANGUAGE:$IDEAFIX/ixdev/tools/$LANGUAGE
ESSENTIA=$IDEAFIX/essentia
SERVERS=ianus;iqlsrv
LD_LIBRARY_PATH=/SH520
ISOFTDB=demo=ideafix:perms=ideafix . . .
```

Realizando esto; simplemente se configuran los ports necesarios para el uso específico de un servicio determinado, para realizar las configuraciones específicas de ***GraphicInterface*** , ***ODBC/JDBC*** , ***Iql Server*** , ***Essentia*** recúrrase a sus respectivos manuales.

Particularidades de las diferentes versiones de UNIX

Servicios de Ideafix

Linux 7.x/8/9

Si se trabaja con Linux 7.x o superior, el servicio se denomina “xinetd” y está ubicado en /etc/xinetd.d/.

Bajo este directorio se genera un archivo denominado con el nombre del servicio a instalar, que contiene la configuración del mismo. En este ejemplo el archivo se llama “iexec” y la sintaxis es la siguiente:

```
service iexec
{
socket_type = stream
protocol = tcp
wait = no
user = root
server = /idea52/bin/irexecd
server_args = -b !e /etc/iexec.env
log_type = FILE /tmp/iexec.log
disable = no
}
```

Compiladores

Para Linux RedHat

Para este Unix no se distribuye gcc, pues se utiliza el egcs que viene con la distribución.

A partir de la versión 7.x o superior de RedHat se vuelve al compilador gcc (version 2.95 o superior)

Para UNIX que no sean Linux

Se debe crear el link al gcc, para permitir la correcta compilación de programas vía ***Ideafix.***

```
# pwd
/usr2/idea52
# cd bin
# pwd
/usr2/idea52/bin
# ln –s /usr/local/bin/gcc cc
```

Para comprobar su correcta instalación:

```
# ./cc -v
Reading specs from
/usr/lib/gcc-lib/i386-redhat-linux/2.7.2.3/specs
gcc version 2.7.2.3
```

Si no aparece algo similar a esto hay que borrar el link y volverlo a hacer (notesé que las líneas que regresa son las mismas que las mostradas para verificar la instalación del gcc).

Parte II

•

IDEAFIX

Manual del Usuario

1

Trabajando con IDEAFIX

Capítulo 1

Introducción

IDEAFIX (*InterSoft Development Environment for Applications in UNIX*) es un conjunto de herramientas de programación y utilitarios que conforman un ambiente integrado de desarrollo y uso de software. El objetivo de este ambiente es proveer medios para que analistas, programadores y usuarios finales puedan obtener una máxima performance y productividad de los recursos disponibles.

IDEAFIX puede ser visto desde dos ópticas diferentes:

- La del usuario final, es decir quien utilizará programas desarrollados con IDEAFIX.
- La del Diseñador/Implementador, que es quien debe diseñar e implementar los programas que utilizará el usuario.

Cada una de estas visiones plantea sus propios problemas. IDEAFIX pretende dar una solución integrada a ambos:

Al usuario a través de una interfaz amigable (mediante ventanas, menúes y ayuda en línea permanente) y fácil de utilizar, configurada para su lenguaje natural e independiente del hardware.

Al Diseñador/Implementador con los medios que faciliten al máximo la creación de Sistemas que cumplan con las características mencionadas anteriormente, permitiendo aplicar criterios de diseño tales como Bases de Datos relacionales, programas modulares estructurados y divididos en funciones, listados y diseño de pantallas “WYSIWYG” (del inglés What You See Is What You Get : Lo que ves es lo que obtienes), etc.

Ambas posiciones se benefician con la filosofía de portabilidad y coherencia que ofrece un ambiente integrado como IDEAFIX.

Componentes del Sistema

IDEAFIX está compuesto por un conjunto de módulos o componentes como muestra la siguiente figura:

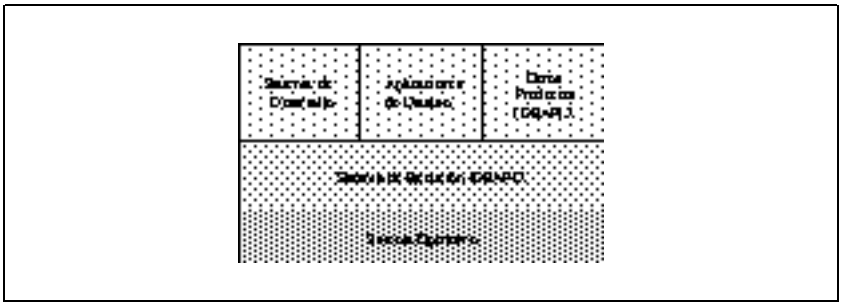


Figura 1.1 - Componentes de IDEAFIX

Sistema de Ejecución

Comprende los programas utilitarios y archivos de configuración necesarios para la utilización de cualquier otro módulo de IDEAFIX, o la ejecución de programas creados con el Sistema de Desarrollo.

Sistema de Desarrollo

Herramientas para el desarrollo de aplicaciones, que incluyen generadores de formularios, listados, programas y una interfaz con el lenguaje C a través de la biblioteca de funciones de IDEAFIX.

Otros Productos IDEAFIX

Dada la filosofía modular con la cual se ha desarrollado IDEAFIX, la familia de productos complementarios está en permanente crecimiento. Estos productos resuelven diversos problemas, tales como consulta de las Bases de datos mediante un lenguaje estándar SQL, procesamiento de Textos, Administración del Sistema Operativo, Planillas de cálculo, etc.

Para agregar uno de estos productos a su ambiente, sólo es necesario contar con el Sistema de Ejecución.

Aplicaciones de Usuario

Son los sistemas desarrollados mediante el Sistema de Desarrollo de IDEAFIX. Para instalar un Sistema de Aplicación en un computador sólo se requiere el Sistema de ejecución, en su versión para dicho computador.

Instalación

El procedimiento de instalación de IDEAFIX en un sistema es sencillo y directo. Consta de dos pasos:

- Copia de los archivos de medio magnético.
- Activación.

La activación significa habilitar un determinado módulo para su funcionamiento mediante una clave propia de cada instalación. Un módulo puede estar copiado, pero si no ha sido activado mediante la clave apropiada, no podrá ser usado.

Las instrucciones detalladas de instalación se dan en la *Guía de instalación*.

Portabilidad

IDEAFIX adhiere a la filosofía de los Sistemas Abiertos. La programación en Lenguaje C lo convierte en un ambiente portable a cualquier tipo de hardware.

La filosofía con la cual IDEAFIX ha sido desarrollado, es cumplir con los estándares más difundidos actualmente, como lo son el Lenguaje “C”, el lenguaje SQL para manejo de Bases de Datos, la interfaz POSIX y SVID y otros, que son respetados por IDEAFIX. Esto asegura que el producto y el software, con él desarrollado queden protegidos en el futuro, y permitan incorporar nuevas tecnologías.

Todo esquema, formulario o código fuente de cualquier otro tipo de archivo de IDEAFIX, corre entre plataformas, virtualmente sin cambios, protegiendo, para un futuro, la investidura del desarrollador.

Sistemas Operativos

IDEAFIX está disponible actualmente para sistemas operativos compa- tibles con UNIX, o compatibles con MS-DOS, corriendo bajo MS-DOS en sí o bajo Microsoft Windows 3.1 o superior (La instalación y uso de la versión de IDEAFIX para Windows está cubierta en la *Guía de Instalación*) .

Para facilitar la lectura, a los sistemas compatibles con UNIX se los denominará genéricamente UNIX, y a los compatibles con MS-DOS de idéntica forma. Asimismo a continuación se listan algunas de las plataformas soportadas por IDEAFIX:

- SCO UNIX
- Unix Ware
- SunSoft's Solaris, Interactive UNIX and SunOS
- Hewlett Packard HP-UX
- IBM AIX
- NCR UNIX System V Release 4
- Unisys V 6000
- DEC's Alpha (arquitectura AXP) OSF/1
- Silicon Graphics' IRIX
- Data General DG-UX
- DEC's Ultrix
- MS-DOS 5.0 o superior, MS-Windows 3.1 o superior, MS-Windows-NT 3.1 o superior y MS-WFW 3.1 o superior.

Acerca de este Manual

El *Manual del Usuario* de IDEAFIX brinda una visión de todos los aspectos necesarios para utilizar Aplicaciones de Usuario o cualquier producto IDEAFIX, cubriendo la óptica del Usuario Final, del Administrador y del Gerente.

Audiencia

Este manual está destinado principalmente a:

- Usuarios finales de aplicaciones.
- Administradores de Sistemas.
- Responsables de área interesados en la mejora de gestión de la empresa.

Para los usuarios finales, este manual contiene la explicación de la interfaz de usuario que presentan los programas de IDEAFIX. Ella se encuentra descripta en los dos primeros capítulos, y es necesaria su lectura por parte de cualquier persona que opere programas.

Los Administradores de Sistema encontrarán el modo de realizar tareas como la definición del entorno de ejecución de los Sistemas de Aplicación, interfaz con el sistema operativo, configuración de terminales e impresoras, procedimientos de respaldo de información (Backups) y otras tareas relativas a la administración de una instalación.

Finalmente, quienes teniendo una responsabilidad gerencial se interesen por las posibilidades de una herramienta poderosa como IDEAFIX, podrán hallar los medios de implementar mejoras en la gestión de la empresa.

Convenciones

A lo largo de este manual se utiliza la siguiente notación:

- < R>: Digitar la tecla RETURN o ENTER en el teclado. El nombre RETURN proviene de las máquinas de escribir o teletipos, y alude al “retorno de carro” que equivale a abrir una nueva línea.
- <Ctrl-A> o ^A: Significa que se debe presionar la tecla CONTROL al mismo tiempo que la tecla siguiente(en este caso <A>). La siguiente tecla debe ser otra letra o carácter especial.
- <NOMBRE>: Indica la tecla ‘NOMBRE' de la terminal virtual de IDEAFIX.
- La información en letra `redondilla` (en inglés courier) muestra lo que el usuario ingresa desde el teclado, o las respuestas que recibirá en pantalla.

Referencias a otros Manuales

Cuando en un Manual o documento de IDEAFIX se recomiende al lector consultar otra documentación, se hará dando el título del manual en bastardilla, referencia acompañada, si fuere necesario, del correspondiente capítulo y sección. Por ejemplo: “... consultar *Manual del Usuario*, sección 1”.

Los manuales de IDEAFIX contienen capítulos destinados a conformar manuales de referencia. El formato de estas páginas se explicita en la introducción de esos capítulos.

Capítulo 2

La Interface del Usuario

Introducción

Entendemos por *interface de usuario* a los aspectos que éste debe tener en cuenta al usar programas de IDEAFIX interactivamente desde una terminal.

Debido a la falta de estandarización, los ambientes de ejecución de programas varían sustancialmente de una instalación a otra. Esto se debe, entre otras cosas, a los diferentes modelos de teclados y pantallas -vale decir, terminales- con que el usuario se enfrenta, inclusive dentro de una misma instalación.

Los programas de aplicación generados con IDEAFIX presentan al usuario una interface uniforme, basada en la terminal 'virtual' de IDEAFIX, aislando de este modo las dependencias del hardware y uniformando el ámbito de trabajo.

La terminal virtual de IDEAFIX, está basada en un display capaz de representar un set de 256 caracteres. De éstos, los 128 primeros corresponden al estándar ASCII, y el resto son usados como caracteres gráficos y de lingüística y como un teclado ASCII de 32 teclas de función. Cada una de estas funciones tiene un nombre que será usado en este manual y en todos los programas y documentación de IDEAFIX.

A lo largo del manual se describirá la interface de usuario en términos de la terminal "virtual". Por ejemplo, una de las teclas de función de la terminal virtual de IDEAFIX se denomina <PROCESAR>. Cuando en el manual se encuentre una mención a ella, significa que el usuario al trabajar en la terminal, deberá oprimir la tecla que en dicha terminal corresponda a <PROCESAR>, por ejemplo F5.

El problema consiste entonces en conocer la correspondencia que existe entre las teclas de la terminal real y la virtual de IDEAFIX.

Como se verá en la próxima sección esto es muy simple ya que el usuario puede pedir al Window Manager que despliegue en la pantalla una tabla con la equivalencia de teclas.

La Terminal Virtual de IDEAFIX

La terminal es el dispositivo que el usuario utiliza para comunicarse con los programas. Mediante ella indica comandos, ingresa información y la visualiza en su pantalla.

Debido a que existen infinidad de modelos distintos de terminales, la forma de operar con ellas varía sustancialmente. Existen diferentes tipos de pantallas y de teclados, por lo que es difícil uniformar este aspecto, para que un operador de programas pueda trabajar sin problemas indistintamente en cualquier modelo de terminal.

IDEAFIX propone una solución a este problema, consistente en definir una terminal “virtual”. Esta terminal posee ciertas características en su teclado y su pantalla. Lo que se hace es adaptar cada terminal real, a las características de la terminal virtual de IDEAFIX.

Conjunto de Caracteres de IDEAFIX

El conjunto de caracteres de IDEAFIX es básicamente el estándar ASCII más 32 caracteres de control -o caracteres de función de IDEAFIX-, y la extensión para alfabetos Europeos y caracteres gráficos. Este juego de caracteres define los símbolos que el usuario verá en la pantalla. Estos símbolos incluyen caracteres acentuados y algunos símbolos gráficos para IDEAFIX. Algunas terminales no pueden desplegar este tipo de símbolos en su pantalla, por lo que IDEAFIX intentará imitarlos con algún carácter similar.

El Teclado

El teclado de la terminal virtual de IDEAFIX posee los caracteres ASCII, más 32 teclas de función usadas en los utilitarios y programas de aplicación para que el usuario indique acciones a realizar.

Los nombres de las teclas de función están indicados en el Apéndice A, donde la primera columna brinda el nombre de la tecla y la segunda su función, aunque ésta varía ligeramente según el programa de IDEAFIX que se esté ejecutando.

Por ejemplo, la tecla <PROCESAR> en una pantalla de datos (formulario) significa grabar en la Base de Datos la información que en ese momento está en la pantalla.

En el editor, <PROCESAR> significa salvar el archivo en disco y si se tratara de algún objeto de IDEAFIX (formulario, reporte, esquema, etc.), compilarlo.

La tecla <CONF_TECL> brinda al usuario la configuración corriente del teclado de la terminal, en cualquier momento en que se esté utilizando el Window Manager de IDEAFIX. Esta tecla despliega en una ventana la ‘correspondencia’ entre la terminal virtual y la real. Normalmente se asigna esta función a <Ctrl-K> en todas las terminales para poder obtener en forma uniforme la configuración.

Las teclas agrupadas bajo el título “Window Manager” son un caso especial ya que son

interpretadas directamente por dicho utilitario. Su significado se explica en la siguiente sección.

El Window Manager (WM)

Introducción

Todo el manejo de ventanas y Entrada/Salida (E/S) de las terminales se hace a través de un único proceso llamado Window Manager . Este administra el teclado y la pantalla, para proveer la misma interface lógica independientemente de los Sistemas Operativos o modelos de terminal. El siguiente diagrama muestra cómo trabaja el Window Manager (WM):

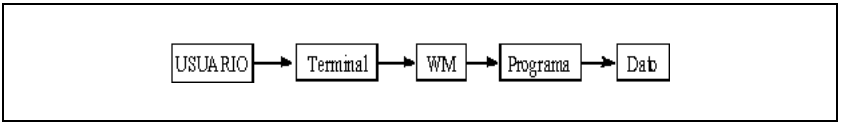


Figura 2.1 - El Window Manager

Todos los utilitarios de IDEAFIX y los programas de aplicación utilizan este proceso para realizar la E/S por teclado y pantalla, que consiste en leer el archivo de configuración para adaptar la terminal “virtual” a la terminal real en uso.

Se utilizan los términos “físico” o “terminal” para la pantalla cuando se refiere al monitor provisto por el hardware.

Ventana es una pantalla lógica que puede ser del mismo tamaño o más pequeña que la pantalla física. Tiene las mismas propiedades que la pantalla de la terminal: puede avanzar o retroceder de a una línea por vez, paginarse hacia arriba o hacia abajo, se puede borrar total o parcialmente, etc.

Una ventana suele ser más pequeña que la pantalla física, como lo muestra la Figura 2.2.

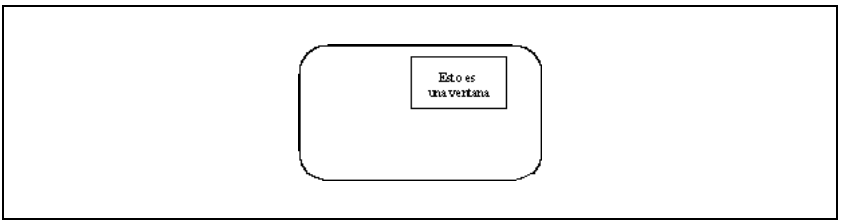


Figura 2.2 - Ejemplo de una ventana

La ventana puede tener en su marco un texto que se denomina una etiqueta. La etiqueta de

una ventana se muestra en el ángulo superior izquierdo de la misma, como se muestra en la Figura 2.3. Las etiquetas son generalmente utilizadas para explicar el propósito principal de una ventana.

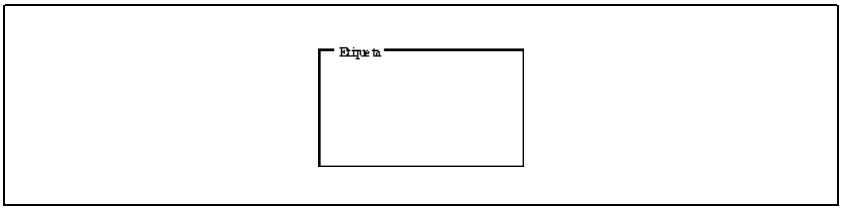


Figura 2.3 - Ejemplo de Etiqueta

Existen dos tipos de ventanas que el usuario utilizará muy a menudo. Se trata de las ventanas de tipo “Menú”, que despliegan una serie de opciones para seleccionar, y las ventanas tipo “Ayuda” para brindar información al usuario. Sus características y operación se detallan en la siguiente sección.

Las Ventanas AYUDA

Las ventanas de tipo “ayuda”' tienen un formato como el mostrado en la Figura 2.4.

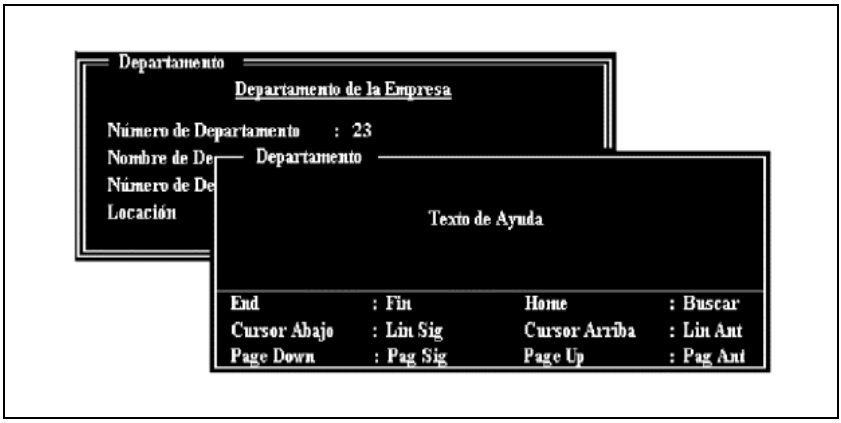


Figura 2.4 - Ventana de Ayuda

Se utilizan para desplegar información de ayuda al usuario en la parte indicada con “Texto de Ayuda”. En esta zona se puede visualizar un texto y paginar sobre él avanzando o retrocediendo por páginas o líneas. El recuadro inferior de la ventana indica las teclas de función que se admiten.

Para que la ayuda sea más efectiva, en lugar de dar el nombre de la tecla de función IDEAFIX, se da el texto con el que la tecla está rotulada en la terminal real en uso. Esta es una característica que se encuentra en todas las facilidades de ayuda de IDEAFIX.

Como se ve en la figura estas teclas de función son:

1. <CURS_ARR> : Retroceder una línea.

2. <CURS_ABA> o <INGRESAR> : Avanzar una línea.

3. <PAG_SIG> : Avanzar una página.

4. <PAG_ANT> : Retroceder una página.

5. <FIN> : Terminar la sesión de ayuda.

6. <META> : Buscar un patrón en el texto de ayuda. Esta función se explica en detalle en la sección correspondiente a la función de búsqueda, luego en este capítulo.

Descripción del Teclado

Un ejemplo de ventana de ayuda es la descripción de la configuración del teclado.

La Figura 2.5 muestra esta ventana en el caso de una terminal VT220 compatible. La primera columna da el nombre de la tecla de función IDEAFIX, y la segunda la/s secuencia/s de tecla/s que hay que oprimir para obtenerla.

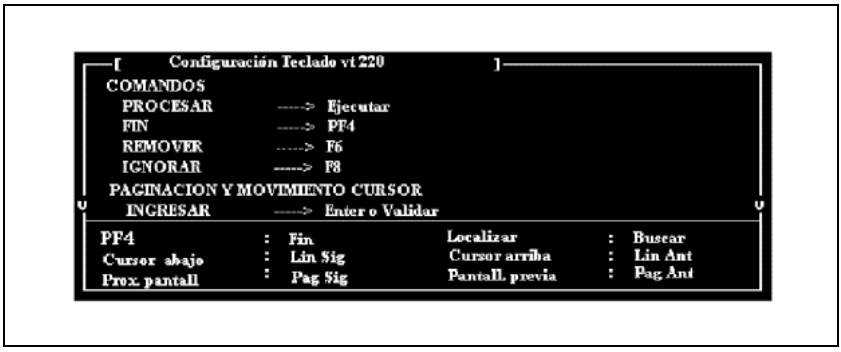


Figura 2.5 - Ejemplo en una terminal VT220

Si se deseara localizar rápidamente una tecla en particular, se puede utilizar la función de búsqueda.

Las Ventanas MENU

Las ventanas de tipo menú se utilizan en el ambiente IDEAFIX toda vez que se requiera que

el usuario haga una selección de una opción entre varias.

Un menú es un cuadro de opciones donde el usuario puede elegir una de ellas. Las opciones pueden ser programas de aplicación (así el usuario no necesita recordar qué programa realiza una función específica), un dato para cargar en un formulario, o inclusive otro menú, el cual en tal caso se denomina usualmente “submenú” o menú secundario.

Una ventana MENU presenta una lista de las opciones existentes. Si hay más cantidad de ellas que filas en la ventana, el cuadro se divide en páginas.

La siguiente figura muestra una ventana menú típica:

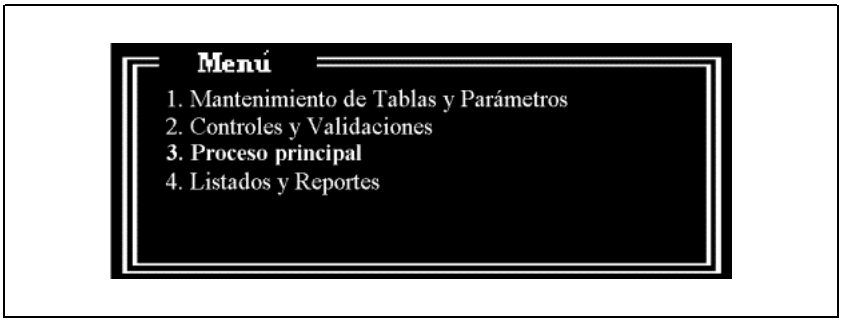


Figura 2.6 - Ventana MENU

La opción “corriente” aparecerá resaltada en la pantalla con video inverso. En la figura precedente ello se ha indicado con negrita (opción tercera).

Frente a una de estas ventanas, se pueden usar las siguientes teclas

de función:

1. <FIN> : Deja la ventana.

2. <INGRESAR> : Selecciona la opción corriente.

3. <CURS_ARR> : Cambia opción corriente a la opción anterior.

4. <CURS_ABA> : Cambia a la opción siguiente. La barra espaciadora equivale a cursor abajo.

5. <PAG_ANT> : Muestra la página anterior del menú, sino se estuviera al comienzo.

6. <PAG_SIG> : Muestra la página siguiente. Estas teclas sólo tienen uso cuando el menú es tan extenso que no entra en una sola pantalla, o bien se lo está desplegando en una ventana de tamaño reducido.

La existencia de más opciones hacia arriba o hacia abajo en la ventana, está indicada por flechas en los bordes de la misma. Si nos encontramos frente a una ventana tal como la

ilustrada en la Figura 2.5, las flechas hacia abajo indican que podremos desplazarnos en tal dirección con <CURS_ABA> o <PAG_SIG> para visualizar las opciones que no están actualmente desplegadas.

Para activar la opción corriente se oprime la tecla <INGRESAR>.

También es posible activar una alternativa digitando un solo carácter. El mismo será buscado a partir de la opción corriente hasta el fin de la ventana como primer carácter de la identificación. Si se produjera una coincidencia, la opción será activada.

Es importante consignar aquí que las opciones de un menú deben designarse con caracteres únicos, ya sean alfabéticos o numéricos, pues el utilitario correspondiente reconoce tan sólo una posición. En otras palabras, no es válido designar a un menú como 10, 11, AA, A1, etc.

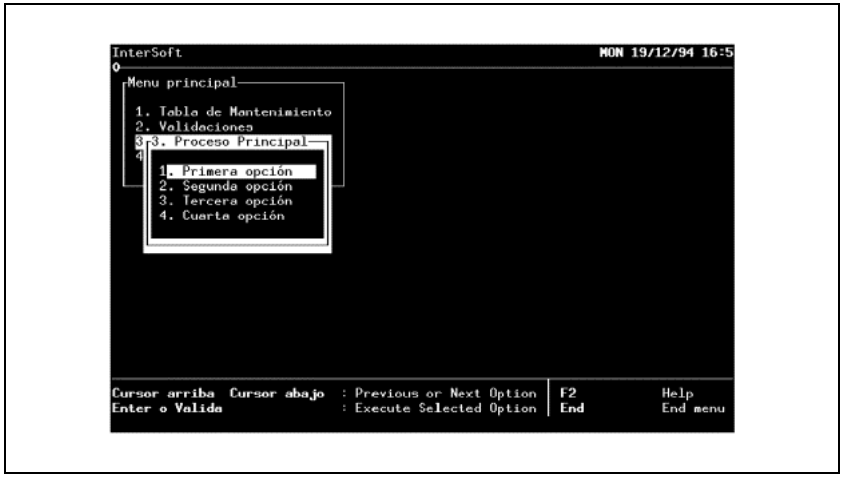


Figura 2.7 - Menú con ventana de Opción

La Función de Búsqueda

En las ventanas del tipo “Ayuda” o “Menú”, es posible realizar búsquedas en el texto de las opciones o la ayuda. Dicha función se activa en ambos casos mediante la tecla <META>, la que al ser oprimida dará lugar a que aparezca una pequeña ventana auxiliar en la parte inferior de la ventana de Ayuda o Menú, que permitirá especificar por teclado el patrón a buscar. Luego de ingresado éste, digitando la tecla de “cursor abajo”, o bien <INGRESAR> , dicho patrón será buscado desde la posición corriente hacia adelante.

Si es encontrado:

- En una ventana menú: partiendo de la opción corriente, aquella cuyo texto contenga la

primera ocurrencia del patrón, quedará como opción corriente y en la primera línea de la ventana.

- En una ventana de ayuda: partiendo de la ubicación corriente, la primera línea de texto que contenga el patrón, quedará al comienzo de la ventana.

En caso de no encontrar el patrón, sonará la señal audible de la terminal y la posición en la ventana no se modificará.

Si el ingreso del patrón finaliza con <CURS_ARR> la búsqueda se realizará hacia atrás, siendo válidas las mismas consideraciones.

El Utilitario MENU

El utilitario `menu` es la interface normal entre los usuarios y las aplicaciones creadas con IDEAFIX.

Los menúes constituyen la herramienta para proporcionar al usuario el acceso ágil y sencillo a un sistema de aplicación, de forma tal que pueda seleccionar rápidamente la función deseada.

IDEAFIX provee dicho utilitario, que interpreta especificaciones de menúes, las cuales son fácilmente diseñadas con cualquier editor ASCII de texto (Ver *Manual del Programador*).

Al invocar el proceso `menu` se presenta una ventana menú de acuerdo a un archivo de descripción. La pantalla se dividirá en tres zonas:

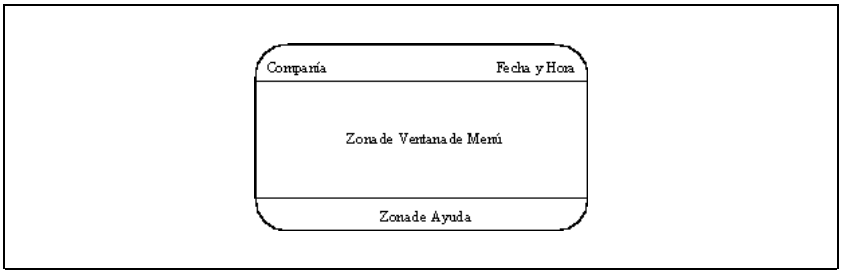


Figura 2.8 - Zonas de Pantalla modelo

La primera ventana menú (el menú principal) aparece en el ángulo superior izquierdo de la zona para las ventanas menú.

Al elegir una opción se provoca la ejecución de un programa o el despliegue de una nueva Ventana Menú. De esta manera, los menúes tienen una estructura jerárquica ya que las ventanas se van apilando unas sobre otras. El primer menú se mantiene en la zona superior y al cancelarlo con la tecla <FIN> se devuelve el control al sistema operativo. En los demás sub-menúes, cuando se los cancela desaparece la ventana y el control vuelve al menú anterior.

La siguiente figura es un ejemplo de la pantalla presentada por el utilitario menu:

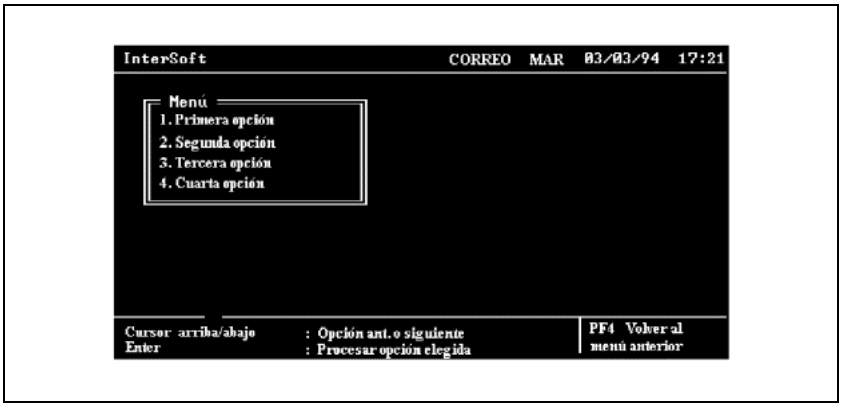


Figura 2.9 - Ejemplo de Menú

La función corriente se muestra resaltada en video inverso. Se puede elegir una opción con las teclas de posicionamiento del cursor y digitando <INGRESAR> para activarla, o tipeando el primer carácter de la opción. Si se supone que el usuario selecciona una cierta función y éste consiste en un sub-menú, la pantalla mostrará el siguiente diseño:

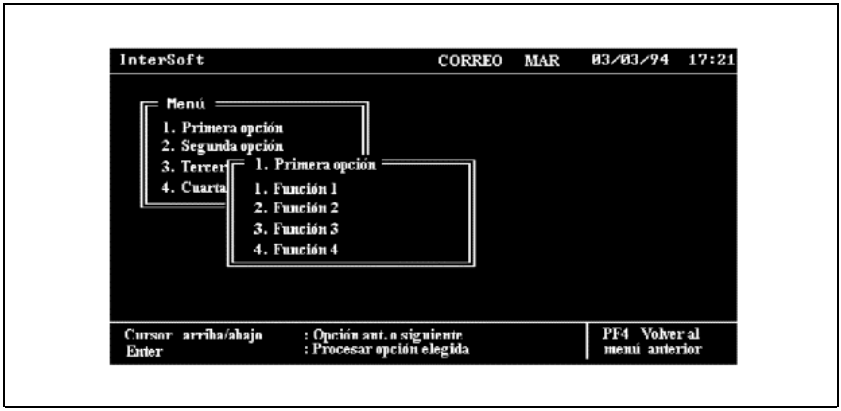


Figura 2.10 - Activando un submenú

Notar que en los sub-menúes se usa el texto de la función seleccionada como etiqueta de la ventana.

Modalidades del Window Manager

El Window Manager (WM) trabaja de dos formas distintas según cómo se lo invoque. En primer lugar, puede ejecutarse en modo residente; es decir, que al ser invocado desde la terminal quedará atendiendo a los procesos que sean luego ejecutados por dicho puesto de trabajo. En tal caso es aconsejable que la invocación se realice desde el `.profile` o el `.login` (según el "shell" que se esté utilizando).

De esta forma se garantiza que cualquier proceso que requiera del WM no aborte por no encontrarlo. Una vez terminada la sesión del usuario, se cancelará la ejecución del WM, ya sea con el comando `wmstop`, en `.logout` o con una sentencia `trap`, en `.profile`.

La otra forma de activarlo es junto con cualquier proceso que lo requiera. De este modo no quedará residente en memoria. El procedimiento a elegir será el que resulte más cómodo y conveniente al usuario; en líneas generales, suele utilizarse la primera alternativa durante el desarrollo de aplicaciones y la segunda para la ejecución por usuario final. Esto debe tomarse como una guía orientativa, y de ningún modo como una regla inflexible.

Funciones del Window Manager

Al estar todas las operaciones de Entrada/Salida centralizadas en un único proceso, es posible implementar una serie de facilidades sin que intervenga en absoluto el programa de usuario. Se produce un diálogo directo entre el usuario y el Window Manager. Estas facilidades se acceden a través de las teclas de función respectivas.

Sin perjuicio de ello, es conveniente ilustrar la función genérica del WM como intermediario entre usuario y sistema, incluyendo la posible existencia de un programa del usuario.

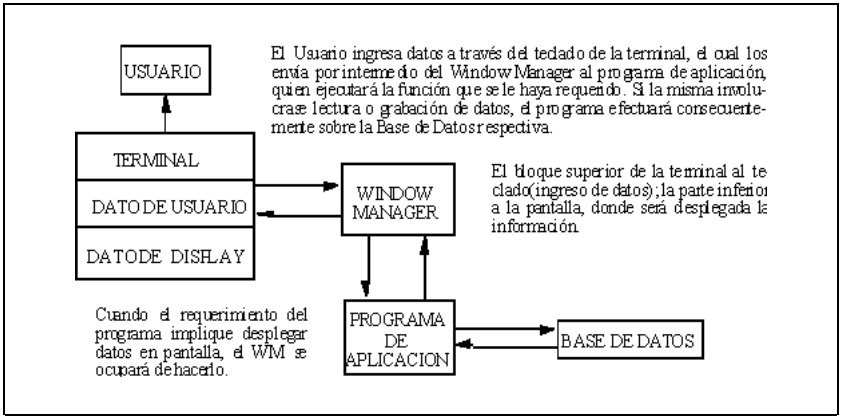


Figura 2.11 - Diagrama Funcional del WM

Descripción del Teclado Corriente

Esta función es obtenida por la tecla de función <CONF_TECL> normalmente asignada a <Ctrl-K>. Esta muestra una ventana tipo “ayuda” con las asignaciones actuales de las Teclas de Función IDEAFIX del teclado, y la forma de ingresar caracteres gráficos, acentos, eñes, y otros caracteres especiales.

Redibujar la Pantalla

Esta función es obtenida por la tecla de función <REDIBUJAR>. Es útil cuando la pantalla se desdibuja por cualquier motivo (caída de tensión en la terminal, error de transmisión, etc.). Mediante esta tecla el usuario puede restaurar el correcto despliegue de la información.

Acceso a los Servicios del Window Manager

Se accede con la tecla <SERVICIOS>. Al oprimirla aparecerá una ventana tipo menú ofreciendo los servicios de los que dispone el usuario. Normalmente estos servicios incluyen:

1. Salida al intérprete de comandos del sistema operativo a través de la opción rotulada como Shell. El programa queda suspendido y se ejecuta el intérprete de comandos. Cuando finaliza la sesión se redibuja la pantalla y se retorna al programa.

2. Impresión de la pantalla. Permite realizar una copia del contenido de la pantalla tal como está al momento de invocar esta opción.

3. Calculadora. En la pantalla aparece el dibujo de una calculadora común de bolsillo, por medio de la cual se pueden efectuar operaciones para determinar el valor de un campo.

4. Conjunto de caracteres de IDEAFIX. Muestra en una ventana tipo “Ayuda” la representación del conjunto de caracteres IDEAFIX en la terminal. Se muestra el código de cada carácter.

5. Correo de usuarios. Facilita el intercambio de mensajes entre ellos.

6. Calendario/Agenda. Esta facilidad permite consultar un calendario, y asociado a él, una agenda de compromisos.

7. Posicionar Ventana. Permite el desplazamiento de las ventanas desplegadas en la terminal.

El Capítulo 10 de este manual brinda una descripción detallada de cómo utilizar cada una de estas funciones.

Suspender el Programa

El Window Manager es capaz de manejar varias tareas --vale decir programas-- y permite al usuario elegir cuál de ellas desea utilizar. La tarea que el usuario utiliza en un determinado

momento se llama tarea activa. Existe una y sólo una tarea activa en cada instante.

Dependiendo de la modalidad utilizada, el Window Manager responderá a la función SUSPENDER de manera diferente.

Suspensión en Modalidad no Residente

Cuando el Window Manager arranca en modalidad no residente dispara una tarea que llamaremos “tarea principal”, la cual suele ser el utilitario `menú`. La Figura 2.12 nos muestra esta situación. En los sucesivos dibujos la tarea activa se indicará con un rectángulo. Notemos que en este caso la tarea principal coincide con la tarea activa.

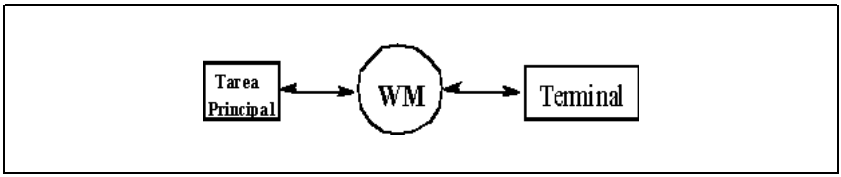


Figura 2.12 - Modalidad no residente

La función SUSPENDER permite crear nuevas copias de la tarea principal como indica la Figura 2.13. Distinguimos cada nueva copia de la tarea principal con una letra.

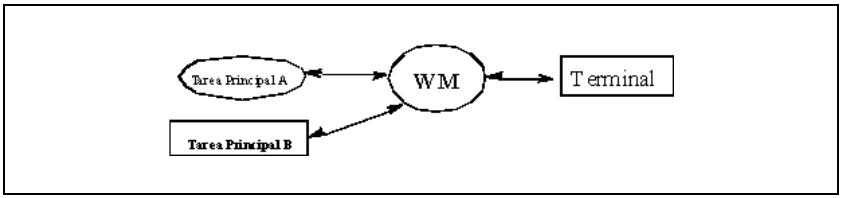


Figura 2.13 - Cambio de tarea activa

Vemos que la Tarea A ha quedado suspendida, y que la recién creada Tarea B pasa a ser la activa.

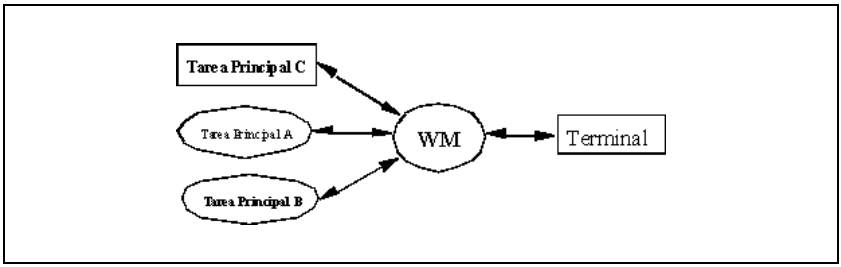


Figura 2.14 - Cambio de la Tarea Activa II

Es decir, que pueden existir varias tareas, pero sólo una de ellas es la activa. Las demás tareas decimos que se encuentran “suspendidas” y las mostramos con elipses (Figura 2.14).

Si la tarea principal permite crear nuevos procesos, a los que llamaremos “subtareas”, tal como lo hace el utilitario menú por ejemplo, podemos tener la situación mostrada en la Figura 2.15. Dicho utilitario tiene la capacidad de crear tareas a las que llamaremos “hijas”'.

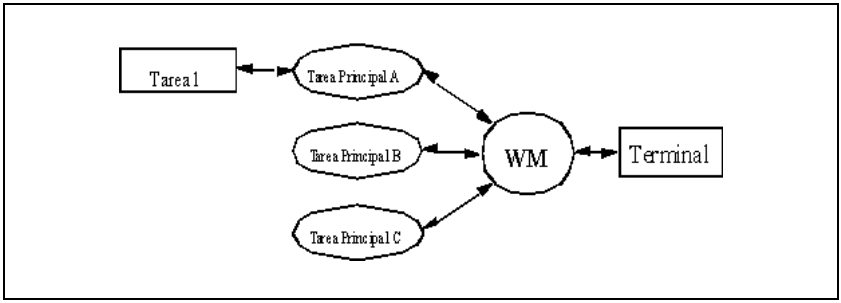


Figura 2.15 - Creación de Subtareas

Teniendo como activa la primera copia (A) de la tarea principal, el usuario ha creado mediante ella, la Tarea 1 pasando ésta a ser la tarea activa. En esta situación es posible decidir suspender momentáneamente la Tarea 1 y pasar a otra tarea existente. Se puede elegir cualquiera de las tareas que no tienen tareas “hijas”, es decir tareas que estén en el final de una cadena. En el ejemplo si suspendemos la Tarea 1 podemos elegir entre las copias B y C de la Tarea principal.

La figura 2.16. nos muestra que el usuario ha elegido la copia C, y con ella ha creado la Tarea 2.

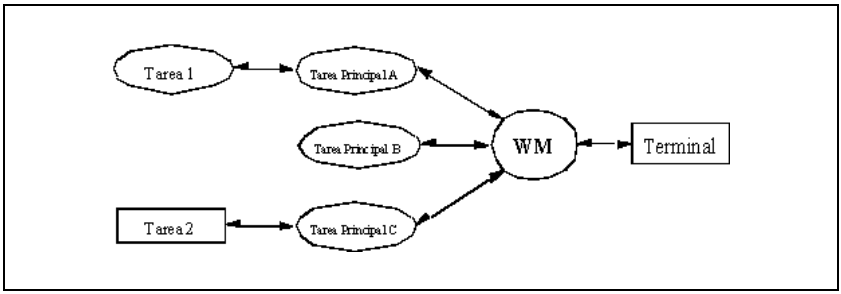


Figura 2.16 - Tareas suspendidas

Si deseáramos suspender en esta situación, podríamos optar entre la Tarea 1 o la copia B de la tarea principal.

Veamos ahora cómo realizar estas operaciones desde el teclado. Cuando se oprime la tecla <SUSPENDER> aparecerá en pantalla una ventana menú con la etiqueta *Tareas Activas*, que ofrece las siguientes opciones:

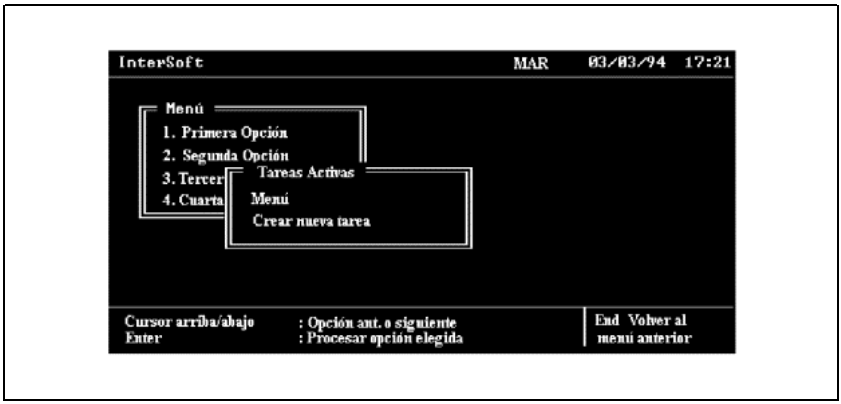


Figura 2.17 - Ventana de Tareas

La primera opción indica siempre la tarea corriente (activa). Se pueden convertir en activas desde la segunda hasta la anteúltima de las tareas existentes, y en último lugar existe la opción de crear una nueva copia de la tarea principal. Si se selecciona la primera opción, lógicamente no se producirá ningún cambio ya que la tarea activa continúa siendo la misma. Desde la segunda hasta la anteúltima, la *tarea corriente* pasará al estado suspendido y tendremos como nueva tarea activa, la seleccionada. Seleccionar la última opción tiene el mismo efecto, salvo que la tarea activa será una nueva copia de la tarea principal.

Cuando se suspende una tarea y se retoma otra que estaba suspendida, la pantalla se redibuja tal como estaba en la tarea ahora reactivada, al momento de suspenderla.

Para aclarar estos conceptos mostraremos la secuencia de comandos que responde a la situación mostrada por las figuras, comenzando desde la Nro.2.13 hasta la 2.17.

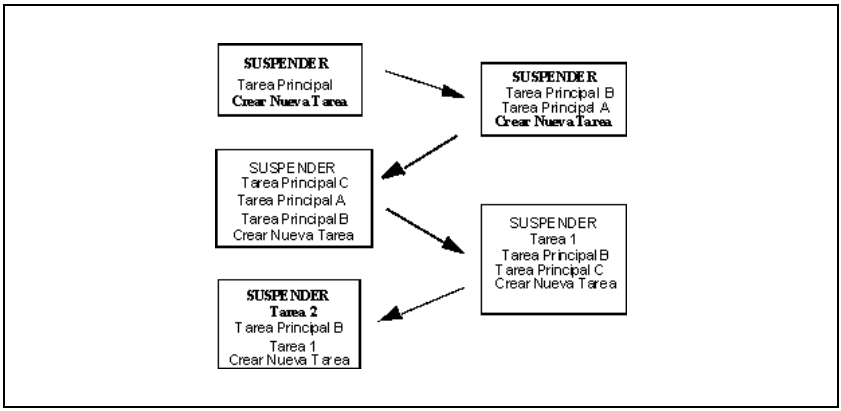


Figura 2.18 - Suspensión de Tareas

Cuando una tarea se finaliza pueden ocurrir dos cosas:

1. Si es una tarea “hija”', la tarea queda eliminada y el control retorna a su tarea madre quedando ésta como activa.

2. Si es una copia de la tarea principal, dicha copia se suprime y el control pasa a la siguiente tarea suspendida (puede ser una copia de la tarea principal o una tarea “hija”). Cuando no existe ninguna otra tarea suspendida (es decir, que nos encontrábamos en la tarea principal original), el control retorna al sistema operativo. De esta forma es imposible que el usuario termine su sesión de trabajo sin dar por terminadas todas sus tareas suspendidas.

Como ejemplo, si el usuario tiene activa la tarea 2 y la da por finalizada,la situación queda como en la Figura 2.19.

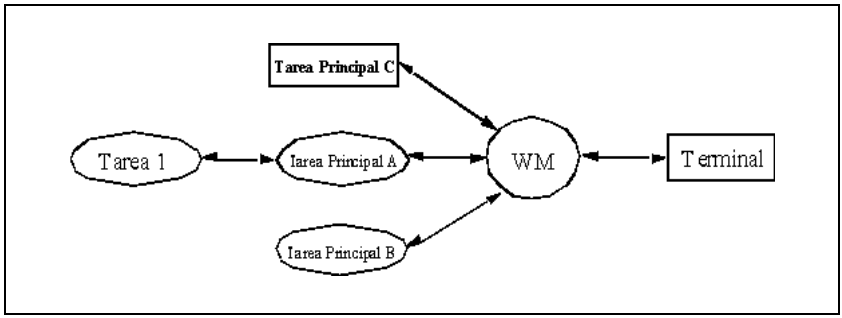


Figura 2.19 - Suspensión de tareas (II)

Existe un límite a la cantidad de copias que puede realizarse de la tarea principal, actualmente configurado en 8.

Suspensión en Modalidad Residente

Utilizando el WM en modo residente, la función SUSPENDER hará que se despliegue una ventana tipo menú, como en el punto anterior, pero en lugar de “Crear Nueva Tarea”, aparecerá “Retornar al Shell”.

Al seleccionar esta opción el usuario suspendará la tarea activa actual y volverá al shell original con el que estaba trabajando (no se crea un sub-shell).

Estando en el shell podrá regresar al WM, bien invocando otra tarea que lo utilice, en cuyo caso ésta quedará como activa; o utilizando el comando fg (Ver Capítulo IX - FG), restaurando el estado activo de la tarea suspendida al ingresar al shell.

Es conveniente aclarar la diferencia entre invocar al shell cuando se lo llama como un servicio, y cuando es activado por la suspensión de tareas.

En el primer caso se crea un “subshell”, mientras que en el segundo se retorna al shell original.

Interrumpir la Tarea Corriente

Se realiza con la tecla <INTERRUMPIR>. Al oprimir esta tecla se despliega una ventana menú que permite seleccionar las siguientes opciones:

• Ignorar (la interrupción). La ventana desaparecerá y el programa seguirá operando normalmente.

• Interrumpir. La tarea será interrumpida y finalizará.

• Abortar. Es similar a interrumpir, pero genera un vuelco de memoria útil para la etapa de

depuración de programas

• Suspender. Suspende la tarea activa para continuar con la que se elija. Con esta opción también se pueden suspender aquellas tareas que no requieran intervención del usuario, en cuyo caso seguirán ejecutándose hasta tanto esta tarea no requiera los servicios del Window Manager

• Terminar. Termina la ejecución del WM incondicionalmente. Sólo debe usarse en casos extremos.

Facilidad de Ayuda

Se accede con la tecla <AYUDA_APL>. Esta función desplegará una ventana de ayuda con un texto relativo a la operación del programa que se está utilizando. El Window Manager simplemente muestra el contenido de un archivo escrito por el diseñador de la aplicación, a efectos de explicar y brindar ayuda sobre el programa que está utilizando el usuario.

Si se está utilizando un utilitario de IDEAFIX (el lenguaje de consulta SQL por ejemplo) el archivo da una descripción de cómo manejar dicho utilitario.

Es muy cómodo utilizar la facilidad de búsqueda en la ventana para ubicar algún tema en particular dentro del texto completo.

El Window Manager buscará, bajo el directorio de ejecución del programa de aplicación ---o en su defecto alguno de los indicados en la variable PATH---, un subdirectorio “/hlp” y dentro de él un archivo con el nombre de dicho programa y extensión “.hlp”.

Capítulo 3

Operación de Formularios

Los formularios son los intermediarios entre el usuario y los datos. Este capítulo explica como operar con los formularios creados con el Generador de Formularios de IDEAFIX.

La Pantalla

En los programas de aplicación la pantalla de la terminal se halla dividida en tres áreas, como muestra la siguiente figura :

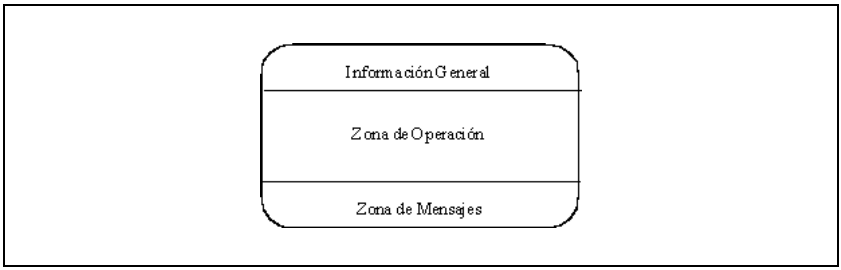


Figura 3.1 - La Pantalla

La línea superior muestra información general (nombre de la empresa, fecha y hora, etc.). La parte central está dedicada a la pantalla de aplicación propiamente dicha. Las dos últimas líneas se reservan para el despliegue de mensajes de error y de ayuda.

Una pantalla está compuesta por campos de ingreso de datos. Los campos pueden ser básicamente de tipo numérico, alfanumérico, fecha, hora y booleano. Esta cualidad define qué tipos de caracteres y qué valores se admiten en ellos.

Cuando se inicia un programa el cursor se posiciona en el primer campo del formulario, que

es el que está más arriba y a la izquierda. El campo en el cual se halla el cursor se reconoce fácilmente ya que se despliega en video inverso.

Durante el ingreso del valor de un campo es posible corregirlo a voluntad con la tecla <RETROCESO> y una vez tipeado el valor definitivo es posible pasar a otro campo adyacente con cualquiera de las teclas de cursor, o bien la tecla <INGRESAR> que equivale a cursor a la derecha.

Las pantallas de aplicación están generalmente divididas en lo que se denomina zona de claves y zona de datos.

La primera incluye los campos a completar, necesarios para poder buscar en la Base de Datos el resto de la información del formulario. La zona de datos está formada por aquellos campos que de alguna forma describen a la zona de clave.

La Figura 3.2. nos muestra un ejemplo donde los campos número y tipo de proveedor definen una clave, en tanto que el resto de los campos constituye la información relativa a él (datos).

Figura 3.2 - Formulario Ejemplo

Cuando los campos de la zona de claves hayan sido completados, el cursor pasa a la zona de datos. Si la clave ingresada, existiera en la Base de Datos, se desplegarán los valores correspondientes y el cursor se posicionará en el “campo de control”. En caso contrario los campos quedarán en blanco y el cursor queda en el primer campo de datos a completar. La Figura 3.3. nos muestra el primer caso:

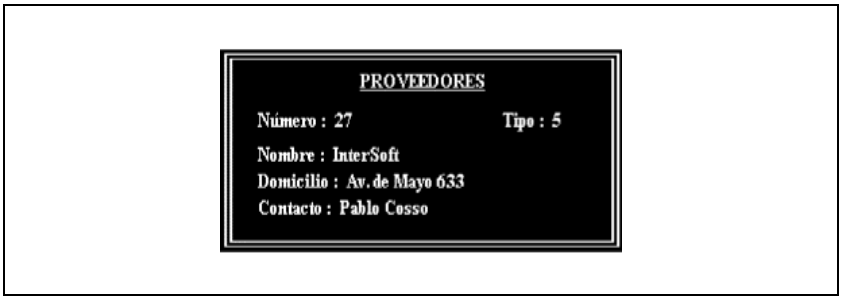


Figura 3.3 - Formulario con datos

La tecla de cursor oprimida para terminar el ingreso de datos en un campo definirá cuál es el siguiente campo de acuerdo a la dirección dada, permitiendo posicionarse de esta manera en cualquier campo de la pantalla (recordar que <INGRESAR> equivale a cursor a la derecha: toma el dato y pasa al campo siguiente). El movimiento sólo está restringido a la "zona" en donde se encuentra el cursor. Es decir, si se está en la zona de claves, el cursor se desplaza dentro de estos campos. Lo propio ocurre en la zona de datos. El movimiento con el cursor pasando sobre campos ya cargados o vacíos, no altera el valor de los mismos.

Validación de Campos

Cuando se intenta salir de un campo por cualquier medio, se verifica que el dato cargado cumpla con las validaciones propias del campo.

La primera validación aplicada es si se ha completado la parte del campo que se considera obligatoria. Por ejemplo, un campo numérico puede tener los primeros 3 dígitos obligatorios, y los restantes optativos. No podrá dejarse el campo a menos que se completen estos 3 primeros lugares. Los campos pueden dejarse vacíos (lo que se denomina "null value"), salvo que se haya establecido lo contrario, en cuyo caso no podrá salirse hasta completar el campo con un dato válido. Los campos fecha y hora se deben completar totalmente para ser considerados como válidos (eventualmente, en el segundo caso, puede cargarse el valor '00' en minutos o segundos).

Una vez completado el campo, se valida el dato contenido en el mismo. En caso de ser erróneo, se desplegará un mensaje de error en la zona de mensajes de la pantalla y el cursor permanecerá en el campo.

Lo conveniente en estos casos es requerir información auxiliar con la tecla <AYUDA>.

Las Funciones de Ayuda

Ayuda de Campo

La primera función de ayuda está dada por la tecla <AYUDA>. Esta puede consistir en:

1. El despliegue de información en la zona de mensajes de la pantalla. En casos sencillos esto puede resultar suficiente para solucionar el problema.

2. La aparición de una ventana de tipo menú, con la lista de valores aceptados en el campo. Al seleccionar una opción, el valor se copiará sobre el campo. Si la lista de valores es extensa, es cómodo usar la función de búsqueda para buscar un dato en particular.

3. Ayudas definidas por el programador, específicas para la aplicación en cuestión.

Ayuda de Aplicación

La tecla <AYUDA_APL> despliega una ventana de ayuda con un texto que explica la forma general de usar el programa. Para ubicar una parte particular del texto de ayuda, se puede usar la función de búsqueda.

Comandos

Las órdenes en cuanto a la acción a ejecutar con el formulario se pueden indicar desde cualquier campo de la pantalla con las siguientes teclas de función:

<PROCESAR>: Grabar en la Base de Datos la información en pantalla. Esto puede implicar el agregado de una nueva clave (*add*), o la modificación de los datos de una ya existente (*update*).

<REMOVER>: Supresión de la Base de Datos del registro con toda su información (*delete*).

<IGNORAR>: Blanquear el Formulario y no realizar la operación.

<PAG_SIG>: Ignorar los datos actuales, y pasar a la clave siguiente. Sólo válida en la zona de claves y en el campo de control.

<PAG_ANT>: Similar a <PAG_SIG> pero con la clave anterior.

<FIN>: Terminar el programa.

Estas operaciones pueden ser inhibidas o bien pedirse confirmación (de acuerdo al diseño de la pantalla). Cuando la operación esté inhibida, al oprimir la tecla correspondiente, sonará la señal audible de la terminal. Si la operación estuviera sujeta a confirmación, aparecerá un mensaje en la zona de mensajes de la pantalla, pidiéndola.

El Campo de Control

En todas las pantallas existe un campo ubicado en el borde inferior derecho de la misma,

llamado “campo de control”. No está destinado a entrada de datos, sino que provee un lugar para descanso del cursor al completar secuencialmente un formulario. Si se pide ayuda con la tecla <AYUDA> en este campo, se presentará una ventana menú con los comandos especificados anteriormente. Lógicamente sólo aparecerán los comandos que no estén inhibidos. Las funciones <PAG_SIG> y <PAG_ANT> sólo se admiten en este campo.

Edición de Campos

En los campos de tipo alfanumérico es posible aplicar comandos avanzados de edición sobre su contenido. Cuando se ingresa en un campo de este tipo, si se digita como primer carácter un espacio, se entra en modo edición. En este modo, las teclas de cursor izquierda y derecha se desplazan dentro del campo, permitiendo posicionar el cursor dentro del mismo. Para salir del campo y terminar la edición, se debe oprimir <INGRESAR>, <CURS_ARR> o <CURS_ABA>.

La siguiente es una lista completa de las teclas aplicables para editar cualquier tipo de campo. Las que están marcadas con (*) sólo son válidas en modo edición, (es decir sólo aplicables en campos alfanuméricos).

<RETROCESO>: Retroceder y borrar el carácter anterior.

<BORRA_CAMPO>: Borrar todo el campo

<DESHACER>: Revertir el valor del campo al que tenía antes de realizar cualquier modificación. El cursor volverá a la primera posición del campo.

<CURS_IZQ> o <CURS_DER>(*): Desplazar el cursor sobre el campo en la dirección indicada sin destruir la información.

<INSERTAR>(*): Insertar un espacio en blanco a la derecha del cursor.

<ELIMINAR>(*): Borrar el carácter bajo el cursor.

Campos Múltiples

Los campos múltiples son conjuntos de campos agrupados en la pantalla en forma de filas. El límite teórico de la cantidad de filas es de 65.535, por lo que cabe desentenderse del tema. Durante el ingreso de datos, al completar una fila automáticamente se pasa a la siguiente.

Un ejemplo común de campo múltiple es un ítem de factura. Este puede constar de los valores: descripción, cantidad, precio unitario e importe; cada uno de los cuales se repite con diferentes contenidos a lo largo del documento -sucesión de filas- y cada uno de los subcampos constituye una columna. Cuando no se desean ingresar más renglones, basta con dejar la primera columna vacía. De ese modo el cursor dejará la matriz y pasará al campo adyacente de acuerdo al comando ingresado. La teclas de cursor permiten recorrer las filas y columnas de la matriz.

La cantidad total de filas de la matriz puede exceder a las que se muestran en pantalla. En ese caso, al completar una página, automáticamente se obtendrá una nueva página en blanco para completar más renglones.

Las siguientes teclas de función se utilizan sobre campos múltiples:

<PAG_ANT>: Pasar a la página anterior.

<PAG_SIG>: Pasar a la página siguiente.

<ELIMINAR>: Borrar la línea del campo múltiple sobre la cual se halla el cursor. Puede estar inhibida según el diseño.

<INSERTAR>: Insertar una línea en blanco sobre la línea donde se encuentra el cursor. Puede estar inhibida según el diseño.

<PAG_IZQ>: Salir del campo múltiple, pasando al siguiente campo hacia la izquierda.

<PAG_DER>: Idem anterior hacia la derecha.

Es posible también inhibir mediante el diseño, la posibilidad de agregar nuevas filas de datos sobre un campo múltiple.

Los Subformularios

Muchas veces la cantidad de información a desplegar en la pantalla excede la capacidad de la misma, de manera que es necesario dividir los datos a solicitar en dos o más formularios. Es común también, que el formato de la información que se requiere luego de un campo, dependa del valor contenido en el mismo.

Para resolver estos problemas existen los sub-formularios. Estos son formularios que aparecen dinámicamente sobre una ventana, se efectúa la entrada de datos, y luego desaparecen retornando a la pantalla el formulario “padre”.

Los siguientes gráficos muestran una secuencia típica.

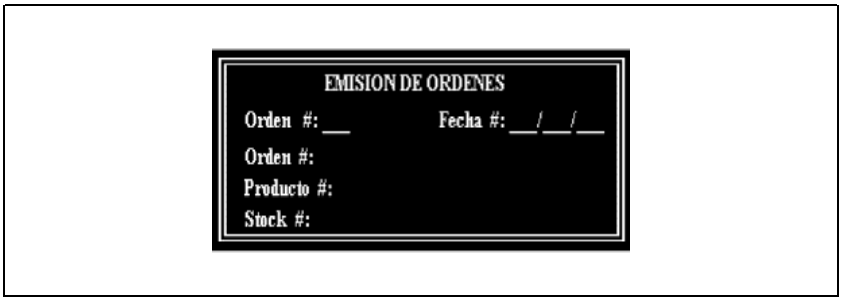


Figura 3.4 - Pantalla de Aplicación

La pantalla mostrada en la Figura 3.4. es una pantalla de aplicación. Cuando el campo llamado Cliente # esté completo, los datos del cliente se mostrarán en una ventana. Cuando el usuario llega a este paso aparecerá lo siguiente:

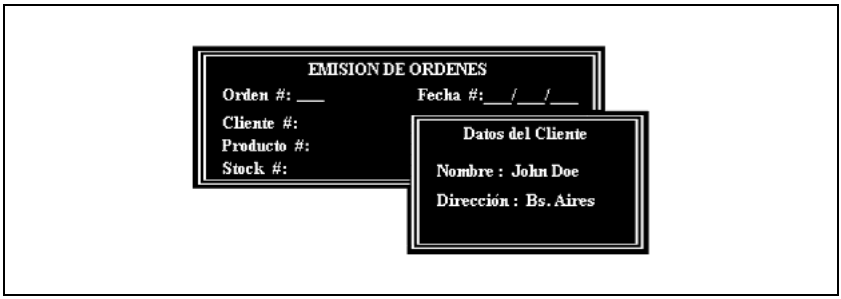


Figura 3.5 - Mostrando una ventana

La nueva ventana puede mostrar solamente datos, o tener campos para que complete el usuario. Cuando éste termina su trabajo con la ventana, la misma desaparece y la pantalla vuelve al estado inicial, como se muestra en la Figura 3.6.

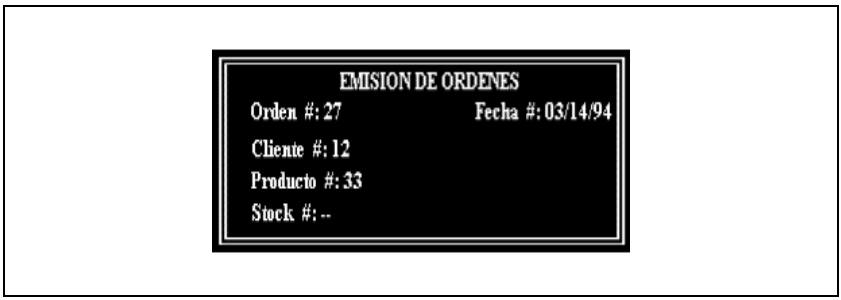


Figura 3.6 - Retornando a la pantalla de aplicación.

El sub-formulario es desplegado solamente cuando se cambia el valor del campo que lo controla y no siempre que se pase sobre el mismo. Si se lo desea visualizar sin cambiar su valor, basta oprimir la tecla <META> (también denominada <HOME>) sobre el campo. Se ingresará así en el sub-formulario correspondiente.

El proceso de ingreso de datos en un sub-formulario es el mismo que en cualquier pantalla. La única diferencia es que al oprimir <INGRESAR> sobre el campo de control, el sub-formulario desaparece y se retorna al formulario principal.

Otras Facilidades

La Calculadora

El manejador de ventanas de IDEAFIX (WM) posee una calculadora incorporada.

Esta calculadora puede ser invocada desde cualquier campo numérico de la pantalla, para poder realizar una serie de operaciones y copiar el resultado sobre éste. Esta función se realiza con la tecla <TAB> y da lugar a que se despliegue un diseño de la máquina de calcular con las teclas que pueden ser accionadas. Una descripción detallada de la calculadora puede encontrarse en el Capítulo 10.

El Calendario

IDEAFIX también incluye un calendario que puede ser llamado desde cualquier campo de datos en la pantalla, para hacer cálculos de plazos y copiar el resultados. Esta función puede ser solicitada con <TAB>, y despliega el diseño del calendario con las teclas disponibles. Para una descripción más detallada, consultar el Capítulo 10.

La Función SUSPENDER

Existe una función del Window Manager llamada SUSPENDER (sólo es válida en sistemas operativos multiusuarios). Al invocarla, se presenta una ventana menú con una lista de programas que han sido previamente activados (Ver Capítulo 2).

La suspensión no puede realizarse en cualquier punto, sino sólo en la primera posición de un campo de ingreso de datos. Es decir, no puede interrumpirse el programa cuando se está por la mitad del ingreso de información.

Mensajes de ERROR

Durante la operación de un formulario pueden obtenerse los siguientes mensajes dados por el Form Manager:

`Permiso denegado`: Este mensaje se despliega porque el usuario no posee permiso para ejecutar una función dada desde un formulario. Por ejemplo, puede darse al intentar borrar un dato (tecla <REMOVER>).

`El registro está siendo accedido desde otra terminal`: Indica que el registro al que se desea acceder está siendo modificado desde otro programa. El registro se encuentra bloqueado y habrá que esperar a que se libere.

`El dato es obligatorio`: El valor del campo no puede ser nulo y deberá ingresarse obligatoriamente un valor válido.

`Valor: valor Ingresado en este campo es incorrecto`: El valor ingresado en el campo no pertenece al rango de valores posibles. Ej.: un número mayor que 12 en el campo “Mes”.

`Digite <Return> para confirmar OPERACION`: Se solicita la confirmación para poder llevar a cabo la operación correspondiente. Por ejemplo en el caso de ingresar valores a una tabla y luego de haber ingresado toda la información necesaria, se debería digitar la tecla <PROCESAR> para grabar dicha información, pero en su lugar se digita la tecla <FIN> por error. Al desplegarse este mensaje, se debe digitar cualquier tecla menos la tecla <INGRESAR> para salvar el error cometido y luego digitar la tecla correcta.

`No hay más datos. Está al final o comienzo de archivo`: Indica que no existen más registros para seguir consultando o modificando si se ha oprimido la tecla <PAG_SIG> o <PAG_ANT>.

`Error: el dato no debe repetirse dentro de la columna`: El valor ingresado en la columna de un campo múltiple ya existe. Se debe ingresar otro distinto.

NOTA: Existen otros mensajes de error, relacionados con el diseño de formularios o la programación “C”. En caso de obtenerse un mensaje no incluido en esta lista, el usuario debería informar al administrador del sistema o al personal idóneo que corresponda, ya que se trata de un error en la programación.

Capítulo 4

Ejecución de programas de aplicación

En este capítulo se describen en forma genérica los elementos que intervienen en la ejecución de un programa de aplicación y la forma de solucionar algunos problemas que pueden presentarse al ejecutarlos.

Componentes de los Programas

El siguiente gráfico muestra la relación entre los distintos componentes de IDEAFIX que intervienen durante la ejecución de un programa de aplicación.

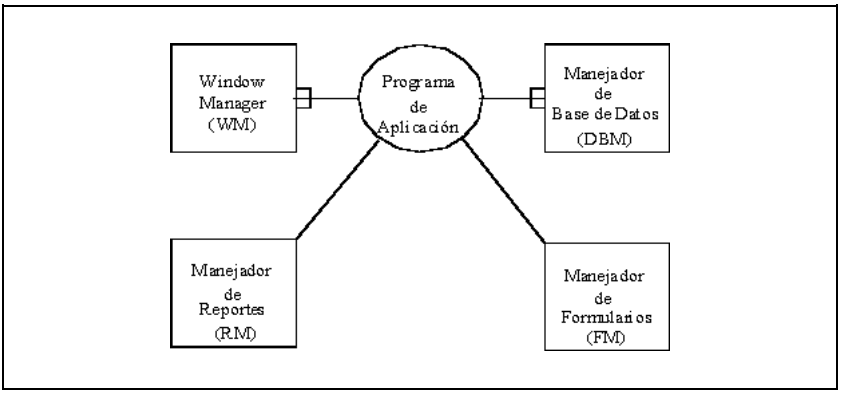


Figura 4.1 - Componentes del Programa

Cada formulario se almacena en un archivo con extensión ".fmo" y es usado por el manejador de formularios (FM -- Forms Manager) cuando un programa lo requiera para que el usuario

opere con él.

Los reportes (listados) residen en archivos con extensión ".rpo". Serán utilizados por el manejador de reportes (RM -- Report Manager) cuando un programa necesite dar formato a su salida a través de un reporte.

Las Bases de Datos residen en un directorio, donde la información se almacena en distintos archivos, *tablas*, organizados según el modelo relacional. Asimismo, las tablas se agrupan en esquemas, definidos en forma similar a los programas, reportes o formularios. Para mayor información sobre todos estos temas, consultar el Manual del Programador de IDEAFIX.

Variables de Ambiente

Al ejecutar un programa de aplicación, éste tiene acceso a su "ambiente" de ejecución. Este ambiente no es más que una colección de variables que permiten controlar ciertos parámetros de ejecución. Las variables de ambiente son entonces necesarias ya que definen valores que el programa necesitará durante su ejecución.

Por ejemplo, los programas consultan la variable de ambiente PATH para obtener una lista de directorios. En estos directorios es donde buscarán los archivos de formularios (.fmo), reportes (.rpo), etc.

Esta variable también es utilizada por el sistema operativo para localizar los comandos que se intentan ejecutar. Es por ello que se debe incluir el directorio "bin" de IDEAFIX y el directorio donde residan los programas de aplicación. Como ejemplo podemos citar:

```
PATH=:/usr/bin:/bin:/usr/ideafix/bin:/usr/sistema
```

El sistema también recurre a otra variable de ambiente, *dbase*, para ubicar el directorio donde se encuentran las Bases de Datos a ser utilizadas por la aplicación.

En el apéndice C "*Variables de Ambiente*" se reliza una especificación detallada de cada una de las variables que se deben definir.

Cuando un programa necesita el valor de una variable de ambiente y no la encuentra, generalmente termina con un mensaje de error informando cuál es la variable faltante. En algún caso, el sistema asigna lo que se denomina "valor por omisión" (default value). Para NFILES, por ejemplo, dicho valor es 12.

Acceso a la Base de Datos

Si un usuario no posee permiso de acceso sobre determinada Base de Datos, el programa avisará de esta situación y terminará la ejecución.

El error puede producirse en dos momentos:

- Si la autorización no permite ninguna operación sobre la Base de Datos que utilizará el programa, el error se producirá en el momento en que se inicie el programa.
- Si el usuario sólo puede realizar ciertas operaciones, el error se producirá en el momento de intentar una operación prohibida, lo que puede ocurrir durante la operación del programa. Un caso típico es la tentativa de actualizar una tabla por parte de un usuario que solamente posee acceso de lectura.

Las Ventanas de Error y Aviso

Los errores o anormalidades producidos al operar un programa se muestran al operador en una ventana. Su etiqueta indica si se trata de un error o un aviso. Así por ejemplo, la siguiente es una ventana de error (Figura 4.3):

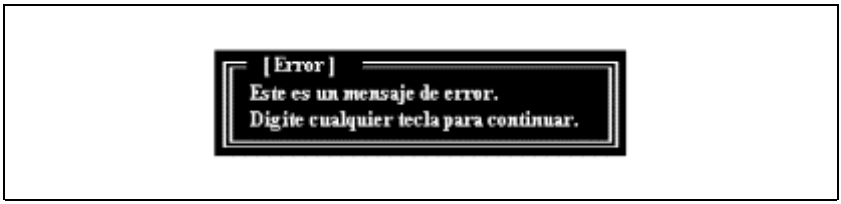


Figura 4.2 - Ventana de Error

Cuando aparezca una ventana de este tipo, al presionar el usuario una tecla cualquiera luego de leer el mensaje, el programa en realidad se cancelará, debido a la imposibilidad de continuar ejecutándose.

En el caso de las ventanas de Aviso, proseguirá normalmente la ejecución del programa. Por ejemplo:

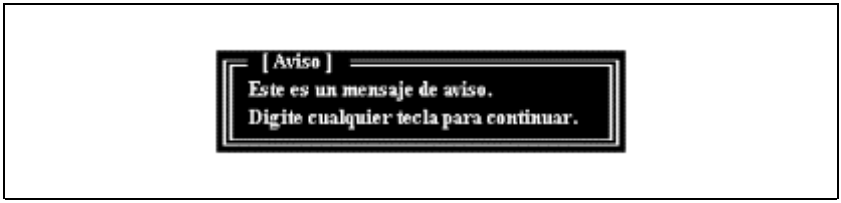


Figura 4.3 - Ventana de Aviso

Existe también otro tipo de ventanas, que simplemente muestran un mensaje informativo, por lo que no tienen etiqueta, tal como:

Figura 4.4 - Ventana Genérica

Problemas de Ejecución

En caso de no poder ejecutar algún programa verifique lo siguiente:

- El valor de las variables de ambiente. En caso de no poder acceder a un formulario, reporte, menú, etc., es probable que la variable PATH no esté definida correctamente. Ello suele significar no tanto que existe un error en su contenido, sino más bien que el mismo se encuentra incompleto.
- Si su sistema soporta Shared Libraries (Bibliotecas Compartidas), verifique la presencia de la que corresponda y eventualmente que Ud. posea autorización para accederla. A partir del UNIX V.3 es posible definir programas que no sean autocontenidos; es decir, que precisen obtener rutinas en tiempo de ejecución. Las bibliotecas que pueden ocasionar este inconveniente son: `/shlib/libidea_s` y `/shlib/lc_s`.
- Permisos sobre los archivos que usa el programa (formularios y reportes deben poseer permiso de lectura para quien intenta utilizarlos).
- Si aparece el mensaje: `No definido mapa input output para terminal "term". Digite cualquier tecla para continuar`. Se debe a que el Window Manager no encuentra la definición de terminal. Al oprimir <INGRESAR>, usará las asignaciones por defecto, con cualquier otra tecla se cancela el programa. Verifique que la varible IDEAFIX contenga el directorio donde se instaló el producto. De este modo se podrán ubicar los subdirectorios "*map*" y "*cap*", en los cuales se encuentran los mapas de terminales e impresoras, respectivamente.
- Activación de módulos. Si al intentar ejecutar un programa de aplicación o utilitario de IDEAFIX, aparece un mensaje que informa que no se tiene autorización para utilizar cierto módulo de IDEAFIX, comuníquese con InterSoft. Este problema puede ocurrir si se reinstala el sistema operativo ya que se necesitan nuevas claves de activación.

Capítulo 5

La Interfaz con Impresora

Este capítulo describe la configuración de impresoras en IDEAFIX.

Introducción

Ya que IDEAFIX provee su propio conjunto de caracteres, debe existir una forma de configuración para obtenerlo en diferentes modelos de impresoras. Esta tarea es efectuada por el utilitario PM (Printer Manager).

Dicho programa provee una traducción para el conjunto de caracteres de IDEAFIX y secuencias para obtener efectos especiales en la salida impresa, tales como la escritura en Negrita, Subrayado, etc.

El Printer Manager es básicamente un filtro que procesa la salida de los programas traduciendo el conjunto de caracteres IDEAFIX de acuerdo a secuencias dadas en un archivo de “capacidades”. Cada modelo de impresora tiene uno de estos archivos con la extensión “.cap”.

Los archivos de capacidades están divididos en dos secciones. La primera define las secuencias que deben ser enviadas a la impresora para obtener cada carácter del conjunto de caracteres de IDEAFIX que necesite un tratamiento especial (por supuesto, los caracteres estándar ASCII no necesitan ser redefinidos).

La segunda parte define las secuencias para obtener efectos especiales. Algunos de ellos (Negrita y Subrayado), pueden ser simulados por el PM si la impresora no los puede manejar.

Utilizando el Printer Manager

El Printer Manager debe ser usado siempre que un programa envíe salida por impresora. El utilitario PM es un programa que lee su entrada estándar, la traduce, y la escribe en su salida estándar.

El PM puede ser utilizado como una interfaz en un sistema spooler (se denomina “spool” al

manejo de dispositivos lentos de salida por medio de un programa administrador de colas, llamado usualmente “scheduler”), o como un filtro que debe ser empleado antes de mandar la salida a impresión. Los siguientes gráficos muestran estas dos opciones:

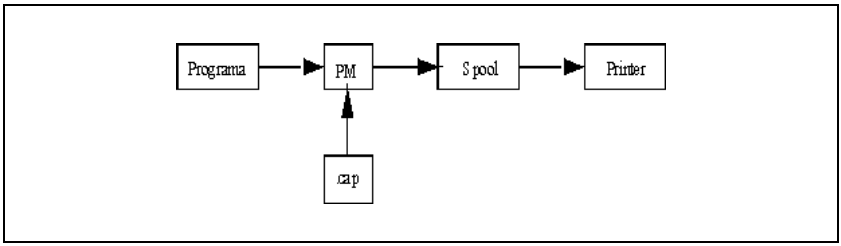


Figura 5.1 - El PM como filtro

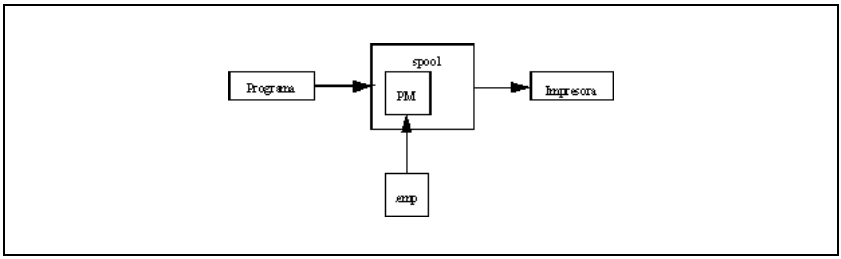


Figura 5.2 - El PM como interfaz con el spool

En caso de configurar el Printer Manager como interfaz en el spool, toda salida impresa por medio del sistema “spooler” será filtrada por él. Los archivos en ASCII no sufrirán ninguna alteración y los que tengan caracteres IDEAFIX serán correctamente traducidos

Si el Printer Manager no se incorpora al spool, se debe tener cuidado antes de imprimir, de filtrar la información.

Capítulo 6

Administrador de Base de Datos

Este capítulo expone los conceptos básicos sobre la composición y organización de las Bases de Datos Ideafix.

Describe también cómo emplear utilitarios que permiten realizar distintas tareas, tales como extraer información, recuperar archivos corruptos, etc.

Se explica asimismo la forma de realizar respaldo (backup) de las tablas que componen una base de datos, y la compatibilidad de los distintos formatos de información.

Si se trabaja con *Essentia* (Servidor de Base de Datos de InterSoft) se deberán considerar los aspectos de configuración propios del mismo. (“Manual del Administrador *Essentia”*)

Introducción

IDEAFIX provee una forma de organizar la información en archivos, a través de un sistema de administración de Base de Datos, que conforma al modelo relacional.

La Figura 6.1. muestra la composición modular del sistema RDBMS (Relational Data Base Management System). La capa exterior pone de manifiesto que existen diversos medios para acceder a la información almacenada.

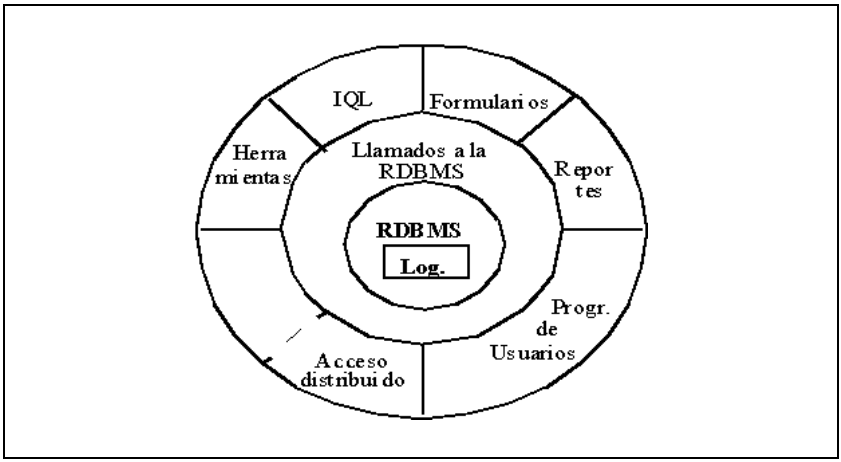


Figura 6.1 - El Manejador de Bases de Datos Relacionales.

El RDBMS es el núcleo del Sistema de Base de Datos. Dicho núcleo se encarga de satisfacer los pedidos de los programas a través del "Método de Acceso InterSoft", que es la parte del RDBMS encargada de acceder físicamente a los dispositivos de almacenamiento. El Método de Acceso InterSoft permite crear múltiples índices, manejar claves duplicadas, etc.

El RDBMS incluye un subsistema de bitácora (logging) transaccional que permite revertir operaciones o restaurar la coherencia de las Bases de Datos frente a fallas de hardware o caídas en el suministro de energía eléctrica.

Como muestra la Figura 6.1., el acceso a la información se puede hacer de varias maneras, lo que convierte a IDEAFIX en un sistema completamente abierto, y capaz de compartir información con otros sistemas. Inclusive está proyectado, entre las futuras extensiones, la posibilidad que el RDBMS sea configurable. Esto significa que podría utilizarse eventualmente como herramienta de diseño de aplicaciones sobre otras bases de datos no provistas por InterSoft, siempre que se ajusten al modelo relacional.

Entre las formas de manejar una Base de Datos, encontramos:

Crear programas en lenguaje "C". La interfaz con el lenguaje "C" provee funciones que implementan los llamados al RDBMS desde programas de aplicación. El IQL, que provee una interfaz interactiva para manipular Bases de Datos.

El utilitario IQL es una implementación del SQL (Lenguaje de consulta estructurado). Permite al usuario realizar operaciones sobre la Base de Datos sin haber creado previamente un programa. Permite resolver una amplia gama de consultas.

El FDL (Form Data Language). A través de formularios, este lenguaje permite relacionar directamente campos de pantalla con campos de la base de datos, y realizar operaciones sobre

la Base de Datos sin necesidad de programar directamente.

Otros utilitarios auxiliares como el *export* e *import*. Los utilitarios `imp` / `exp` permiten extraer información de las Bases de Datos en formato ASCII. Una de sus funciones más obvias es proporcionar los mecanismos de respaldo y restauración (backup/restore).

Componentes de las Bases de Datos

Tablas

Las Tablas son la forma básica de almacenamiento. Están formadas por filas (también denominadas registros) y columnas, como se muestra en la Figura 6.2.

Campo 1	Campo 2	Campo3	Campo 4

Figura 6.2 - Estructura de una tabla

El gráfico muestra que cada registro está formado por un número fijo de campos. Todos los registros en una tabla tienen el mismo número de campos.

IDEAFIX adhiere al modelo relacional, donde existe sólo una estructura de datos - la tabla. De esta uniformidad nace un lenguaje de Base de Datos donde con un comando se puede recuperar un conjunto de registros de una o más tablas existentes, para listarlas, armar una nueva tabla, o reprocesar el conjunto de registros recuperados. Este lenguaje se conoce como SQL (Structured Query Language - Lenguaje de consulta estructurado).

La implementación del SQL ofrecida por IDEAFIX es su producto “IQL”.

Esquema

Un esquema es simplemente una colección de tablas, que normalmente guardan una relación lógica entre sí.

Base de Datos

La Base de Datos es toda la información a la que el usuario tiene acceso en un momento determinado. Para IDEAFIX, una Base de Datos está definida como el conjunto de datos contenidos por todos los esquemas “activos”.

“Activos” significa que el usuario los ha seleccionado a través de una instrucción para operar con ellos.

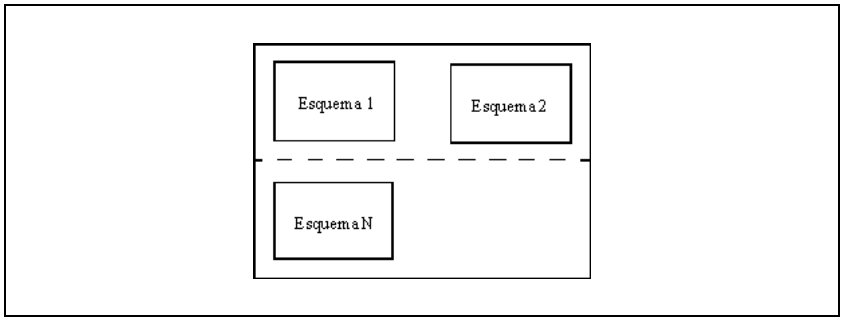


Figura 6.3 - Estructura de una Base de Datos.

Detalles de Implementación

Ubicación Física de los Archivos

Existe una variable de ambiente denominada *dbase* que indica el directorio bajo el cual están los archivos que forman las Bases de Datos. Hay un directorio por cada esquema.

La variable “dbase” apunta a un directorio del cual dependen subdirectorios para cada uno de los esquemas definidos. Esta situación se halla descripta por la siguiente figura:

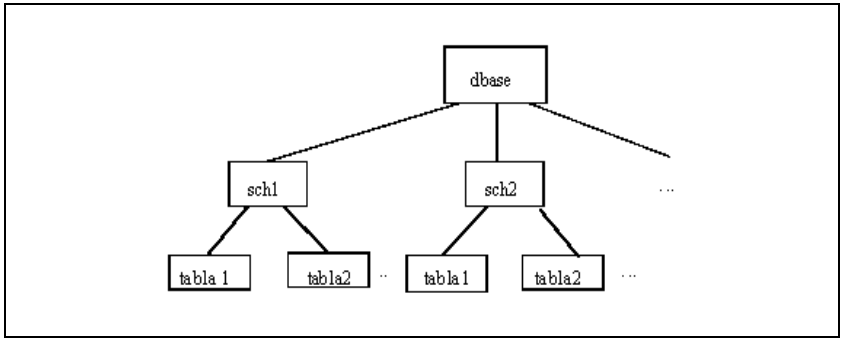


Figura 6.4 - Distribución de la base de datos.

Una tabla se compone de un archivo físico con los datos propiamente dichos (archivo con extensión `.tab`) y uno o más archivos físicos con las claves para acceder a ellos (índices). Cada tabla tiene por lo menos un índice, llamado índice primario, el cual responde a la clave igualmente denominada, que es la utilizada más frecuentemente: clave primaria (primary

key). Dicho índice se identifica mediante la extensión `.i00`.

Para permitir el acceso a la información de una tabla según una clave distinta de la primaria, se pueden organizar índices secundarios, que se van identificando mediante las extensiones `.i01`, `.i02`, etc.

Organización de los Archivos

Los archivos de datos (extensión `.tab`) están organizados como una secuencia de registros de longitud fija, identificándose cada uno por medio de un número de registro "lógico".

El espacio en disco ocupado por los archivos de índices y datos es estrictamente el necesario para guardar la información. No hay información de control que aumente el tamaño de los mismos innecesariamente, salvo un encabezado inicial de 1024 bytes.

A medida que las tablas son utilizadas, su tamaño va creciendo al agregar información. Borrar datos no implica disminuir el espacio físico ocupado por el archivo. Un registro borrado se marca para ser aprovechado cuando más adelante se realice un agregado, de modo tal que antes de incorporar nueva información primero se verifica si hay algún espacio libre. Si lo hay, se lo utiliza; en caso contrario, se añade un nuevo registro al final del archivo de datos, con lo cual aumenta su tamaño.

Esta forma de manejar la supresión de registros se denomina baja lógica, en tanto que la baja física (eliminación real del registro), llamada también reorganización del archivo, se logra mediante los utilitarios ya mencionados `exp` / `imp`. Normalmente esto es innecesario ya que la tendencia natural de cualquier tabla es a crecer, pero si se diera la circunstancia de un file que ha sufrido una fuerte retracción (gran cantidad de bajas) la cual no se prevé vaya a ser compensada en un futuro próximo, existe la forma de recuperar ese espacio no usado.

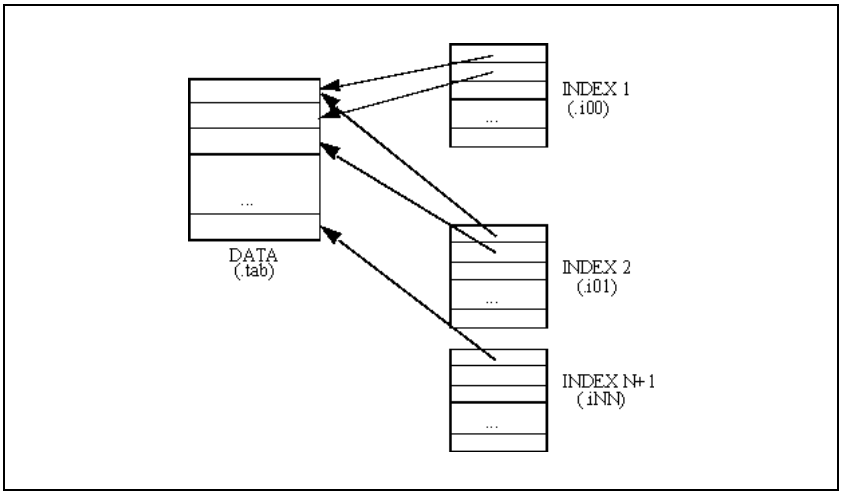


Figura 6.5 - Estructura de una Tabla

Variables de Ambiente

dbase. Esta variable indica un directorio, el cual contiene como subdirectorios los esquemas de base de datos. Por ejemplo si se define:

```
dbase=/usr/datos
```

y se tiene un esquema llamado personal, los archivos con los datos del esquema estarán contenidos en el directorio /usr/datos/personal.

NFILES. En general los sistemas operativos imponen un límite sobre la cantidad de archivos abiertos simultáneamente por un programa de aplicación. Este límite varia entre un sistema y otro. La variable de ambiente NFILES define cual es este límite para los programas de IDEAFIX.

Para averiguar cuantos archivos abiertos simultáneamente admite su sistema, existe el utilitario de IDEAFIX nfiles.

Log de Transacciones

Función

Es el mecanismo que permite eliminar transacciones incompletas, ocasionadas por cortes de energía, caídas de equipo, interrupción del programa por invasión de memoria, errores en punto flotante, etc.

Activación

Para que esté activo es necesaria la existencia de la variable de ambiente LOGDIR, la cual contiene un directorio existente, por ejemplo: `LOGDIR = /usr/ideafix/logs.`

Mecanismo

Cada proceso tiene asociado un archivo en donde se graban todas las transacciones que realiza. Este archivo esta ubicado bajo el directorio especificado en LOGDIR, con un nombre de formato l_xxxxxx (el número xxxxxx identifica unívocamente a cada proceso).

Las funciones IDEAFIX que escriben en el Log son:

• BeginTransaction.

• PutRecord.

• DelRecord, DelAllRecord y similares.

• EndTransaction.

Cuando se produce una interrupción por software del programa, se llama a la función *Stop()*, la cual recorre el archivo de Log desde el final hasta el principio revirtiendo sólo la última transacción que no tenga EndTransaction.

Esto es así para que los datos no queden inconsistentes. Al finalizar, el programa borra su archivo de Log. Si la interrupción del programa no puede ser atendida por software (corte de energía eléctrica, por ejemplo) será necesario ejecutar el comando *rollback* para eliminar la transacción incompleta. Para mayor información sobre este comando consulte la siguiente sección de este manual.

Utilitarios

Explicaremos en esta sección el uso de los programas utilitarios de uso frecuente en relación con la administración de Bases de Datos.

Frecover

Este utilitario permite reconstruir los índices de una tabla, a partir de los registros existentes en el archivo de datos (extensión `.tab`).

Se lo utiliza cuando por algún motivo o eventualidad externa, uno o más archivos índice de una tabla se dañan y quedan inconsistentes (el ordenamiento dado por el índice no coincide con los datos existentes en la tabla); o bien, para poder reconstruir los índices al recuperar de un backup solamente las tablas de datos.

La forma de invocarlo es:

```
frecover esquema[.tabla [indice]]
```

Si no se indica un índice, se recuperarán todos los pertenecientes a la tabla *tabla*. En ausencia del argumento tabla, se hará lo propio con todos los índices de todas las tablas del esquema *esquema*. Como se indica en la sintaxis, sólo es válido colocar el nombre de un índice si antes se especificó el de alguna tabla.

Se debe tener en cuenta que si hay un gran volumen de datos, el proceso de reconstrucción puede insumir bastante tiempo. Es conveniente entonces usar la opción “-v”, que describe con mensajes en la terminal los pasos a medida que los realiza.

El utilitario solo recuperará los índices en la medida en que éstos estén corruptos. Si los índices están correctos no se realiza ningún procesamiento, a menos que se indique la opción **-u**. En ese caso la recuperación de todos los índices de la tabla indicada (o de todas las tablas si se indicó tan sólo el esquema) es incondicional.

Existen otras opciones adicionales, entre ellas una que sirve para resolver el caso de un esquema bloqueado por acceso simultáneo. La opción **-f**, fuerza la recuperación ignorando el

contador de concurrencia del esquema. Para más información sobre este utilitario, consultar el Capítulo 9.

Exp/Imp

Los utilitarios *imp* (import) y *exp* (export) son programas que sirven para convertir la información contenida en archivos con formato ASCII al formato de la base de datos de IDEAFIX y viceversa, respectivamente.

Tanto al uno como al otro se le deben especificar el esquema y la tabla con los cuales se va a trabajar.

El utilitario *imp* toma datos de la entrada estándar en formato ASCII y los almacena en la tabla especificada.

El utilitario *exp* trabaja en forma inversa, toma los datos de una tabla IDEAFIX y los vuelca en ASCII por su salida estándar.

Estos comandos son sumamente útiles cuando se busca migrar de un sistema de base de datos a otro, para preservar los datos cuando se cambia de sistema operativo, o simplemente para emitir listados de consulta.

Forma de uso

Veamos un ejemplo de cómo usar estos utilitarios:

```
exp -d pers emple > exp.out
```

Aquí estamos haciendo un *export* de la tabla *emple* (empleados) del esquema *pers* (personal), grabando la salida en el archivo exp.out. Más adelante explicaremos el significado de la opción “-d”.

Supongamos que la tabla empleados fue creada con el siguiente formato de registro:

nroleg

nombre

categ

fecing.

Esta tabla pertenece a un esquema llamado *pers*.

Para acceder rápidamente a los datos contenidos en ella se han creado dos índices llamados:

nroleg: Dado el número de legajo permite encontrar los datos del empleado. Esta será la clave principal (primary key).

categ: Formado combinando los campos categ y nroleg.

La base de datos no tiene ningún problema en diferenciar los distintos registros que hay en ella. Pero ésto se torna difícil de distinguir en un archivo de texto ASCII, y si pretendemos convertir entre formato ASCII y la Base de Datos de IDEAFIX, debemos convenir en algún modo de separar los registros y los campos.

Para diferenciar en un archivo de texto dónde comienzan y terminan los campos y los registros, es necesario que se intercalen delimitadores. Estos se llaman Field Separator (FS) para fin de campo y Record Separator (RS) para fin de registro. También existe un separador que se usa para distinguir los elementos de los subcampos: Subfield Separator(SS).

Por defecto son tomados los siguientes caracteres como separadores:

- RS: \n (ASCII newline)
- SS: ^A (ASCII Ctrl-A)
- FS: \t (ASCII tab)

al usar la opción **-d** los separadores se convierten en

- FS: ,
- SS: ,
- RS: Newline

En nuestro archivo “exp.out”, en donde hemos guardado la salida del *export*, tendremos los campos separados con comas y los registros con *newlines*. El contenido del archivo podría ser el siguiente.

```
1,”Pablo Marmol”,27,27121988
2,”Pablo Morsa”,15,25061970
3,”Tomás Blanco”,18,12081987
4,”Angel López”,15,12051979
```

Notar que las fechas no están formateadas (barras separadoras). Para ello se debe usar la opción “-f”.

Con esta simple instrucción hemos obtenido un listado del contenido completo de la tabla de empleados. Está ordenado según la clave principal.

Al *export* se le pueden especificar, entre otras cosas, que sólo extraiga un cierto rango de la totalidad de registros en la tabla. Por ejemplo vamos a extraer los registros con los números de legajos entre 2 y 4

```
exp -k 2:4 pers emple > exp.out
```

El flag -k indica que lo que sigue es una especificación de rango de valores para la búsqueda por índice entre los que se desea extraer los datos.

El *export* puede listar los datos ordenados por cualquiera de los índices de una tabla. Para ello

se usa la opción **-i** indicando el nombre del índice a utilizar. Por ejemplo:

```
exp -i categ pers emple <R>
2,"Pablo Morsa",15,25061970
4,"Angel López",15,12051979
3,"Tomás Blanco",18,12081987
1,"Pablo Marmol",27,27121988
```

La opción "-k" es por supuesto aplicable a este caso. Como las claves se forman con pares de valores, si deseamos listar desde la categoría 11 hasta la 30, todos los números de legajo indicamos:

```
exp -d -k 11,0:30,0 -i categ pers emple
```

Estamos indicando los distintos valores del índice separados por comas y que con el flag **-i** le decimos cuál índice debe usar. Como hemos visto si no se especifica el índice toma por defecto el índice principal.

Normalmente se obtienen los valores de todos los campos de los registros seleccionados de la tabla y en la sucesión dada por quien definió la tabla. Si sólo se desea imprimir el valor de algunos campos o bien lograr un ordenamiento distinto, luego del nombre de la tabla se colocan los nombres de los campos que se desean. Al hacerlo así, se listarán sólo los campos dados en el orden indicado. Por ejemplo:

```
exp -d pers emple nroleg nombre
```

Si se le indica la opción **-r** (raw) entenderá que acceda directamente al archivo de datos sin usar índices, y en caso de especificarle un rango éste corresponderá al número de posición física de los registros. Es decir, existe un "índice implícito" para cualquier archivo, tabla o colección física de datos registrados en el sistema, que consiste en el "orden de ubicación". La opción "raw" trae la información en ese orden.

También tanto en el *import* como en el *export* se pueden dar instrucciones para modificar los caracteres delimitadores. Como hemos visto existe una opción que hace que la salida del *export* sea compatible con el formato conocido como "delimitado" usado por la mayoría de los productos de Base de Datos - la opción "-d".

Es posible también indicar de forma más explícita qué caracteres queremos como delimitadores con las siguientes opciones:

```
-Rc RS = caracter 'c'
-Fc FS = caracter 'c'
-Sc SS = caracter 'c'
```

Si usted desea que los números sean impresos en formato exponencial, debe utilizar la opción "-e".

Existe una última opción para que al escribir el contenido de un campo, se le anteponga el nombre que tiene definido en la base de datos, esta es la opción -v.

Si ahora estamos en otra máquina con otra versión de sistema operativo, usando el *import* de

esa máquina recuperamos los datos en esta forma:

```
imp personal2 empl < exp.out
```

Donde personal2 es otro esquema y empl es una tabla que ha sido definida en forma similar.

Procedimientos de Respaldo

Para respaldar la información en tablas de bases de datos es necesario copiar los archivos que la contienen. Es conveniente respaldar cada esquema en forma individual, lo que facilita el procedimiento.

Cada esquema contiene un archivo que indica la estructura del mismo y que puede asimilarse a un diccionario de datos. Este archivo ---que tiene extensión .sco--- debe ser respaldado siempre.

La forma más simple y que requerirá menos espacio en medio magnético para respaldar un esquema, es respaldar solamente el diccionario y los “datos” de las tablas (los archivos con extensión .tab).

La otra posibilidad es respaldar el esquema completo, es decir, el diccionario de datos, los datos en sí mismos y los índices. Estos últimos son archivos que tienen extensión .inn, donde el valor de varía desde 00 hasta 99.

En el primer caso, el beneficio es realizar el respaldo más rápidamente y ocupando menor espacio en el medio magnético. La desventaja es que frente a la necesidad de recuperar la información, se deben copiar los archivos de medio magnético, y luego realizar un procedimiento de recuperación de índices (ver el utilitario frecover).

En el segundo caso copiando los archivos del medio de backup al directorio apropiado, se restaura el esquema inmediatamente.

Veamos entonces un ejemplo, donde se realizará el backup de un esquema llamado “personal”. Lo primero que debe hacerse es ubicar el directorio donde reside este esquema. Recordar que este directorio es el indicado por la variable de ambiente dbase más el nombre del esquema. Entonces si efectuamos:

```
$ echo
$dbase /usr/datos
$ cd /usr/datos
$ cp personal/*.tab personal/*.sco backup
```

hemos respaldado los datos y el diccionario.

Para respaldar el esquema completo se copian todos los archivos:

```
$ cp personal/* backup
```

En este segundo caso, para restaurar la información, solo es necesario realizar el procedimiento inverso:

```
$ cp backup/* personal
```

En el primer caso, donde se han respaldado solo los datos, se copia primero la información, y luego se reconstruyen los índices:

```
$ cp backup/* personal
$ frecover -u -v personal
```

Rollback

Este comando procesa todos los archivos del tipo “l_xxxxxx” que estén en el directorio especificado por LOGDIR (no procesa los Logs activos pues ellos estarán bloqueados por cada proceso), elimina de cada uno de ellos la última transacción incompleta, y por último los borra.

Los nombres de archivo consistentes en una “ele” seguida de un subrayado, identifican los “log_files”, en tanto que los siguientes caracteres identifican el proceso de referencia. La forma de invocar este comando es:

```
rollback
```

Se debe tener en cuenta que para que el comando pueda ser ejecutado, debe estar activo el Log de Transacciones. Para mayor información consulte la sección anterior de este manual.

Compatibilidad de Datos

Los archivos que contienen los datos de las tablas de base de datos tienen la información en formato binario, es decir, una imagen directa de como se ve el dato en la memoria del computador cuando se lo utiliza. Esta característica permite evitar cuando se usa una base de datos los programas tengan que convertir de una representación a otra y además se gana espacio en disco ya que los datos están almacenados de la forma en que menos espacio ocupan.

Dado que los datos están en binario a imagen de memoria, no pueden ser transportados entre ciertas arquitecturas que presentan incompatibilidades. Citamos a continuación los casos más comunes:

Pasaje a un sistema de distinta arquitectura. Cuando se cambia de un equipo Intel a otro Motorola, por ejemplo, los bytes en memoria están ordenados en forma inversa, por lo cual no es posible compartir los datos en formato binario. En caso de trasladar información de un equipo a otro, deberá hacerse en formato ASCII mediante los utilitarios *import*/*export*.

Procesadores con distinto tamaño de palabra de CPU. Es el caso dentro de la línea Intel, donde los procesadores 8086/8, 80186 y 80286 tienen 2 bytes, mientras que los 80386 y 80486 tienen 4 bytes. En estos casos los datos son compatibles, haciendo la salvedad del diccionario de datos (archivo `.sco`). Sólo hace falta entonces regenerar este último archivo.

Para hacerlo, se define la variable de ambiente dbase apuntando a un directorio cualquiera

(por ejemplo `/tmp`) y se genera el esquema a partir de su definición, procedimiento conocido como generación “en seco” y que no debe ser confundido con la “limpieza a seco” o “`rm -f *`” (este comando borra todos los archivos dependientes del directorio en el cual se encuentre ubicado el usuario en ese momento). Luego el diccionario generado se coloca en el directorio del esquema, es decir dbase/esquema.

Capítulo 7

Creación y Administración de Menúes

Este capítulo explica los conceptos necesarios para crear menúes, y controlar los programas a los que tendrá acceso un usuario.

También, se explica como definir los permisos de trabajo de un programa, esto es, las operaciones que podrá realizar sobre los datos desde un formulario.

Especificación de Menúes

Un menú se especifica en un archivo con la extensión .mn. El archivo contiene información de los nombres de las opciones, y qué hacer cuando el usuario elige cada una de ellas. El formato de cada línea es el siguiente:

```
Texto_opcion Tipo_de_opcion Argumentos
```

El texto de la opción es la leyenda que se despliega en la ventana.

El tipo de opción y sus argumentos especifican qué hacer cuando el usuario elige la opción. Hay cinco tipos de opción:

• MENU Cuando se la selecciona, se despliega otro menú. El argumento es el nombre del archivo con las especificaciones del nuevo menú.

• SHELL El argumento especifica el nombre de un comando que ejecutará el sistema operativo. Cuando se termina de ejecutar el comando, el control se devuelve al menú.

• PIPE El argumento especifica el nombre de un comando que ejecutará el sistema operativo, pero la salida del comando se muestra en una ventana. Opcionalmente el tamaño de esta ventana puede ser especificado. Después de la palabra PIPE puede aparecer:

```
PIPE [filas] [columnas] comando argumentos
```

La opción PIPE no está destinada para programas interactivos. Trabaja con programas que sólo envían salida a la pantalla. El típico ejemplo es un comando que lista los contenidos de un directorio.

• WCMD El argumento especifica un programa de aplicación, que debe ser ejecutado con el Window Manager. Todo programa que necesite del WM debe ser indicado en un menú con este tipo de opción.

• FORM *fm* Ejecuta el form *fm* mediante execform. Es equivalente a realizar "WCMD execform *fm"*.

• RWCMD *host program* Ejecuta un programa usando CRACKER de InterSoft. Es equivalente a "WCMD cracker *host program"*.

• SETENV Esta opción permite establecer variables de ambiente desde un menú. Su sintaxis es:

```
'ID' SETENV confarch
```

El archivo confarch tendrá la siguiente salida:

Identificador de un grupo de variables a establecer.

Descripción del identificador.

Pares Variable-Valor

Un ejemplo del contenido de este archivo puede ser:

```
ADMINS,Administration,printer,P:lp -d admin,flength,15
SALES,Sales Dept.,printer,P:lpr -d sales,flength,10
SUPP,Tech. Support,printer,P:lp -d supp,flenght,18
```

Este archivo puede ser configurado para cada uno de los usuarios situados en el directorio "home" de cada uno, o globalmente si está situado en el directorio `$IDEAFIX/data/language`, donde "language" debe ser reemplazada por el valor de la variable de ambiente `LANGUAGE`. El formulario que aparece seleccionando esta opción de menú permitirá al usuario cambiar cada uno de los valores de las variables, antes de establecerlas con la tecla <PROCESO>.

• BUILTIN Esta opción permite invocar un conjunto de funciones incorporadas dentro del Window Manager. Las funciones BUILTIN son las siguientes:

Función	Descripción
GotoShell	Ir al Shell
_print_scr	Imprimir Pantalla
_print_ichset	Juego de Caracteres
calculator	Máquina de Calcular

servmov	Posicionar Ventana

Ejemplo

Se muestra un ejemplo simple de la especificación de un menú:

```
‘1. Listado del Directorio’ PIPE 10 60 ls -lCF
‘2. Calculadora de UNIX’ SHELL bc
‘3. Otro menú’ MENU submn03
‘4. Programa Aplicativo’ WCMD doform carga
```

La primera opción ejecutará el comando del sistema operativo “`ls –lCF`”, y la salida aparecerá en una ventana de 10 filas por 60 columnas.

La segunda posibilidad corre el programa `bc` que provee las funciones de una calculadora. Cuando se termine con bc el control se devuelve al menú original.

La tercera opción invoca a un nuevo menú, cuyas especificaciones están en un archivo llamado otro .mn.

Finalmente, la última alternativa ejecuta un programa definido mediante el generador de formularios “doform”, cuyo nombre es form.

Permisos de los Programas

Cuando se ejecuta un programa desarrollado con IDEAFIX es posible determinar dinámicamente qué operaciones se tendrán habilitadas en los formularios. Estas operaciones son las siguientes:

A - (Add) Agregar nuevos datos.

U - (Update) Modificar datos existentes.

D - (Delete) Borrar datos existentes.

Normalmente un programa tiene habilitadas todas las operaciones. Pero es posible inhibir algunas o todas ellas mediante una indicación dada en la línea de comandos en el momento de lanzar el programa a ejecución.

Esta indicación consta de un signo “!” (usado habitualmente para denotar negación) seguido de una o más de las letras especificadas en la tabla anterior (A, D o U). Por ejemplo

```
cnt101 !D argumento_1 argumento_2 argumento_n
```

invocará el programa cnt101, sin permitir borrar información.

Esta indicación es automáticamente tratada por IDEAFIX y no llega al programa como argumento. Por lo tanto si un programa necesita recibir un argumento en tiempo de ejecución,

estos se suministran luego de la indicación de los permisos.

Esta indicación ---si está presente--- debe ir siempre en primer lugar, ya que de otro modo se tratará como un parámetro más.

Esta característica permite configurar en los menúes de usuario los permisos de trabajo. Por ejemplo, un usuario puede utilizar un programa mediante la siguiente opción de menú:

```
‘1. Cuentas contables'' WCMD cnt101 !ADU
```

pero en tal caso lo que sólo podrá consultar la información existente.

En cambio, quien tenga indicado en el menú:

```
‘1. Cuentas contables'' WCMD cnt10
```

podrá realizar todo tipo de operaciones. Notar que se trata del mismo programa en ambos casos, pero en el primero de ellos se lo ha convertido simplemente en un programa de consulta al limitar los permisos.

Ayuda en los Menúes

Cuando se solicita ayuda de aplicación con la tecla <AYUDA_APL> en un menú, se desplegará una ventana de ayuda que mostrará un archivo con el mismo nombre que aquel que contiene el menú, pero con extensión `.hlp`. Este archivo debe residir en un subdirectorio “hlp” que dependa de alguno de los directorios indicados en la variable de ambiente PATH. Lo usual es que si los programas ejecutables están en un directorio tal como

```
/usr/aplicn/bin
```

el directorio de ayudas para los programas de la aplicación “aplicn” sea

```
/usr/aplicn/bin/hlp
```

Si el archivo no existe se desplegará una ventana que informa de esta situación.

Capítulo 8

Utilitarios del Sistema de Ejecución

Introducción

Este capítulo describe, en orden alfabético, los comandos que forman parte del sistema de ejecución de IDEAFIX. Cada página va encabezada por el nombre del comando, a la manera de ésta que lleva el nombre “Introducción” y consta de una serie de ítems que se detallan más adelante, luego de la sintaxis de los comandos.

A menos que se indique lo contrario, los comandos descriptos en este capítulo aceptan opciones y argumentos de acuerdo con la siguiente sintaxis:

```
nombre [opcion(es)] [argumento(s)]
```

donde:

`nombre`: Es el nombre de un archivo con un programa ejecutable.

`opcion(es)`: noargletra(s), argletra <> opargum, donde <> es un espacio en blanco opcional.

`noargletra`: Una sola letra representando una opción que no necesita argumento.

`argletra`: Una sola letra representando una opción que re quiere de un argumento.

`opargum`: Una cadena de caracteres que es el argumento de la opción argletra.

`argumento(s)`: Cualquier nombre de archivo u otro nombre que no comienza con -, o - solamente que indica la entrada estándar.

Nota: Todos los utilitarios de IDEAFIX admiten la opción “-?’, que significa imprimir la forma de uso del comando, y la opción ‘-b” que indica no emitir el mensaje que identifica el utilitario y su versión.

Cada entrada de este capítulo está presentada usando los siguientes títulos (aunque alguno de

ellos puede ser omitido en ciertos casos):

Sintaxis

Sumariza la forma de usar el comando que está describiendo. Se utilizan las siguientes convenciones:

- Los corchetes “[ ]” alrededor de un argumento indican que dicho argumento es optativo. Cuando el prototipo de un argumento está dado como “nombre” o “archivo” se refiere siempre a un nombre de archivo.
- Los puntos suspensivos se utilizan para indicar que el argumento anterior se puede repetir.

Descripción

Provee una visión general del propósito del comando.

Opciones

Explica el propósito de las opciones soportadas por el comando.

Ejemplo

Esta sección brinda ejemplos cuando resulte pertinente.

Notas

Contiene detalles o comentarios adicionales.

Archivos

Indica nombres de archivos que son usados por el comando.

Diagnósticos

Describe los mensajes de error o indicaciones que pueden ser producidas por el programa durante su ejecución.

Consultar

Brinda referencias a otros documentos o partes del manual donde encontrar información relacionada.

Al finalizar, cada comando retorna un código conocido como status, también denominado “código de retorno”. Este código es 0 para terminación normal, y distinto de 0 cuando durante el proceso se ha encontrado algún error. Es costumbre que el valor numérico del status guarde relación con la gravedad o magnitud del problema (de allí que “cero” equivalga a “sin errores”).

DOFORM

Operar con un formulario.

Sintaxis

```
doform [-l] [-jCampoI=CampoJ[,CampoN=CampoM]] file[.fmo]
```

Descripción

Este utilitario permite usar una especificación de formulario compilada para realizar operaciones sobre tablas de Base de Datos.

Opciones

-l (lock) Bloquear toda la tabla. Todos los registros de las tablas a modificar se bloquean de manera que ningún otro usuario puede modificar datos de dichas tablas.

-j (join) Esta opción se utiliza cuando se está trabajando con dos tablas desde el formulario, donde ambas tablas poseen campos en común.

El utilitario leerá un registro cuando se completen los campos clave del formulario. Si éste utilitario permite modificar o borrar registros existentes, la lectura se realizará bloqueando el acceso a dicho registro para prevenir que se lo modifique desde otros programas.

La clave (índice) que se utiliza para acceder a la tabla es la mínima necesaria donde todos sus campos figuren en la clave dada en la pantalla.

Si el formulario es solamente de consulta ---es decir que ignora las operaciones de modificación (update) y borrado (delete)---, la lectura del registro se realizará sin bloqueo alguno.

Si el formulario tiene campos múltiples, debe caer en alguno de los casos siguientes:

- Los campos del múltiple deben corresponder a un campo vectorizado de la base de datos, salvo que sean skip (ignorar) o display only (sólo para ser desplegados).
- La zona de claves puede no estar completa, pero se completará con las n primeras columnas del campo múltiple.

Cuando se trabaja con dos tablas, donde ambas poseen campos en común de modo que a través de la tabla principal se accede a la tabla del multirenglón, los primeros n campos del múltiple completan la clave de la segunda tabla. En este caso es obligatorio utilizar la opción -j y especificar los campos que deben ser copiados, por ejemplo:

```
-jCampoT2=CampoT1[,CampoT2=CampoT1]...
```

Otras restricciones presentes son:

- Dentro de subformularios, los múltiples deben caer solamente en el caso 1).
- No se manejan llamadas a subformularios dentro de campos múltiples.
- Los casos 2 y 3 de multirenglones sólo se permiten en el formulario principal, y no en subformularios.
- No se pueden actualizar más de dos tablas.
- Si hay más de un multirenglón en el formulario, sólo se pueden combinar los casos 1-3 y 1-2.

Notar que en los campos vector de la Base de Datos, si se los trata con campos múltiples en un formulario, existe la restricción de que un elemento intermedio del vector no puede contener el valor NULL, ya que se lo tomará como última línea del multirenglón.

Ejemplo

Si se ha diseñado un formulario llamado emp, y luego ha sido compilado exitosamente con el utilitario *fgen*, se lo puede utilizar con: doform emp.

Notas

Se emitirán mensajes de aviso cuando un campo del formulario no tenga relación con una tabla de Base de Datos, salvo que el campo sea el primero de la pantalla o que sea campo rector de un multirenglón, donde será obligatorio.

Si no cumple con uno de los casos mencionados de multirenglón el programa terminará con el mensaje de error correspondiente.

Diagnósticos

`No puede bloquearse toda la tabla`. Se ha indicado la opción **-l**, pero algún otro usuario mantiene registros bloqueados sobre la tabla. Espere a que la misma quede liberada y arranque nuevamente el programa.

`Campo NOMBRE no tiene relación con Base de Datos}`.

`No se permiten llamadas a subform dentro de campos múltiples`.

```
Campo NOMBRE debe ser vector para poder manejar el multirenglón.

Campo NOMBRE no completa la clave de la pantalla.

No hay suficientes campos para completar la clave.

En un subform los múltiples deben corresponderse con vectores.

Campo NOMBRE caso no contemplado de multirenglón.

No soporta mas de 2 tablas con multirenglón.

Campo '% s' no pertenece a tabla '% s'.

Con esta pantalla debe usarse la opción -j.

Demasiadas tablas usadas (Máximo 2).

Campo '%s' con diferente dimensión respecto al rector.

No se pudo hallar una clave que concuerde con el form.
```

Consultar

FGEN

DOREPORT

Obtener un reporte a partir de los datos de entrada.

Sintaxis

```
doreport [-b] [-s] [-nNN] [-d| [-Fc] [-Rc]] arch[.rpo]
[parám...]
```

Descripción

Este utilitario lee la entrada estándar, que debe ser una secuencia de registros en formato ASCII. Esta información alimentará la definición del reporte en archivo y de esta forma se obtendrá el reporte requerido.

[archivo] debe ser una especificación RDL compilada.

[parámetros] son valores que se pasan al reporte en el momento de ser ejecutado a través de la línea de comandos. Los parámetros que el reporte necesita son definidos al diseñarlo.

Opciones

Se dispone de las siguientes opciones:

-b No emitir el mensaje identificatorio.

-Rc Definir el carácter “c” como separador de registros. (En caso de omisión es “Newline”: \n). Ver en la sección Notas cómo especificar el carácter.

-Fc Definir el carácter “c” como separador de campos. (En caso de omisión es un TAB). Ver en la sección Notas cómo especificar el carácter.

-d Utilizar el formato delimitador: RS = '\n'; FS = ','.

-s Modo silencioso. No se emite el banner de identificación ni los mensajes de error.

-nNN Define la cantidad NN de copias que se emitirán.

Ejemplo

Suponiendo que se dispone de un lote de datos en un archivo llamado “datos”, se les da formato mediante un reporte con el siguiente comando:

```
$ doreport reporte < datos
```

Notas

Los caracteres RS y FS pueden ser indicados con:

- Caracteres ASCII. Por ejemplo -F,.
- ^ X (X es un carácter entre A y Z ó [y _) para carácteres de control. Por ejemplo -F^ A.
- Especificando un carácter de control con las secuencias del Lenguaje “C”: b\ t\f\n\r . Por ejemplo -R\n.

Con códigos numéricos en octal, decimal o hexadecimal.

- En octal usar \ OOO (O es un dígito octal).
- Para especificar en decimal usar \dDDD (D es un dígito decimal). En ambos casos se pueden especificar de uno a tres dígitos.
- En hexadecimal usar \ xXX (X es un dígito hexadecimal). Se pueden colocar uno o dos dígitos.

Recordar que los caracteres ^ y \ son interpretados por el Shell. Usar alguna forma de escape, por ejemplo precediendo por una barra o encerrando entre comillas.

Diagnósticos

El utilitario puede emitir los siguientes mensajes de error:

`No pudo abrirse para entrada:nombre`: El utilitario no puede acceder a la fuente de datos. nombre puede ser un comando o un nombre de archivo.

`Registro muy grande (max.= NN)`: Un registro en la entrada excede el registro interno. Los datos serán truncados. NN es un número que indica el tamaño configurado del registro de doreport.

`Entrada indefinida (verificar Vars. de Amb. no definidas)`: El reporte no tiene definido de dónde provendrán los datos. Posiblemente falte suministrar parámetros entre los cuales se encuentre el comando o archivo que es la fuente de datos.

`Error en fuente de entrada`: Se produjo un error al intentar leer los datos de entrada.

`Error en la salida`: Se produjo un error escribiendo la salida.

Consultar

EXP, RGEN .

EXECFORM

Permitir entrada de datos y ejecutar un programa

Sintaxis

```
execform [-b] [-r] [-d|-s|-w|-p [ROWS] [xCOLS]] pantalla[.fmo]
[programa]
```

Descripción

Este utilitario permite la entrada de datos mediante el formulario pantalla y luego invoca al proceso programa, con los valores de los campos de la pantalla como argumentos.

Programa es el nombre del proceso que será invocado. En caso de no indicarlo se utiliza el archivo pantalla.sh (El mismo nombre que el formulario con la extensión `.sh`).

Opciones

-b Con esta opción no se muestra el mensaje de identificación del utilitario.

-r Ejecutar el programa y cuando finalice dejar el formulario desplegado para una nueva captura de datos.

-d Modo debbug. No ejecutar la línea de comando, pero sí mostrarla en la pantalla cuando el

usuario digita PROCESAR.

-s Ejecutar el utilitario como un programa del sistema operativo a través del shell.

-w Ejecutar un programa como un usuario del Window Manager. Esta es la opción que se considera en caso de omisión.

-p Ejecutar el programa como un programa del sistema operativo, pero direccionar su salida a una ventana (dimensionada con ROWS y COLS). “Rows” indica la cantidad de filas, y “Cols” la de columnas.

Ejemplo

Supongamos el siguiente formulario (l_libros.fm):

```
Codigo Desde : ____.
Hasta : ____.
Digite ___ para generar el listado.
% form
use biblio;
window border standard;
% fields
desde : libros.codigo, not null, default 1, on help in
libros:titulo;
hasta : libros.codigo, not null, default 9999, on help in
libros:titulo;
ayuda : is help(PROCESAR);
```

Este formulario nos servirá para captar datos que le enviaremos como parámetros a un archivo con sentencias SQL ejecutado mediante el utilitario *iql*.

El archivo con sentencias SQL (`l_libros.sql`) contendrá lo siguiente:

```
use biblio;
select codigo, titulo, autores.nombre, edicion, fecha
from libros, autores
where codigo between $1 and $2
order by autores.nombre, titulo
output to report ‘l_libros’;
```

La sentencia SQL, como se podrá observar, envía la salida a un reporte (`l_libros.rp`) que podría ser el siguiente:

```
% titgen(today, hour, pageno) before page
Fecha : ______
Hora : __:__:__
Pagina : ____.
LISTADO DE LIBROS
Codigo Titulo Autor Ed. Fecha
=========================================================%
linea(codigo,titulo,nombre,edicion,fecha)
____. ___________ _______________ ____. __/__/__
%report
```

La forma de invocar el proceso mediante el utilitario execform sería la siguiente:

```
$ execform -w l_libros iql -b l_libros
```

generando el listado correspondiente.

Si se ejecutara en cambio:

```
$ execform -d l_libros iql -b l_libros
```

se desplegaría la línea de comando ejecutada, que no hubiera aparecido (en caso de no haber especificado la opción -d).

EXP/IMP

Exportar e importar datos en formato ASCII.

Sintaxis

```
exp [-b] [-c] [-v] [-e] [-r] [-k f1, f2, fn:t1, t2,tn] {[-d] |
[Fc] [Rc][Sc]}[-i índice] equema[.]tabla [campo...]
imp [-b] [-d] |[Fc] [Rc] [Sc] equema[.]tabla
```

Descripción

Estos utilitarios permiten al usuario extraer/incorporar datos desde/a tablas de la Base de Datos de IDEAFIX, en formato ASCII y con gran flexibilidad.

La tabla con la cual se operará se especifica con dos nombres separados con blanco (2 argumentos), es decir “squema tabla”, o bien formando un solo argumento usando un punto intermedio “squema.tabla” (ésta es la sintaxis usada en el lenguaje SQL). En este último caso el argumento tabla es omitido.

exp lista los registros seleccionados por la salida estándar. Se puede seleccionar un rango de registros con la opción -k. Los registros se listan según la clave primaria, o bien por otro índice si se usa la opción -i. Se imprimen los valores de todos los campos de los registros de la tabla, salvo que se especifican nombres de campos mediante los argumentos campos. En este caso sólo se imprimen los valores de estos campos, en el orden dado.

imp lee la entrada estándar y según las opciones utilizadas graba o borra dicha información en la tabla de la Base de Datos.

Opciones

Se indican en primer término las opciones aplicables a ambos utilitarios.

-b No emitir el cartel de identificación.

-cn Usa n-kbytes como memoria cache.

- **Rc** RS (separador de registros) = carácter 'c' (default \ n)

-Fc FS (Separador de campos) = carácter 'c' (default TAB)

-Sc SS (Separador de subcampos) = carácter 'c' (default TAB)

-d Formato delimitado: -R \ n -F, -S,

-f archnombre: establece “archnombre” como el archivo de especificación de campo.

Opciones aplicables solamente al utilitario exp.

-v Imprimir el nombre del campo precediendo a cada valor.

-e Convertir números de punto flotante en formato “exponencial”.

-r Realizar lecturas directas del archivo de datos sin usar índices.

-k f1,f2, ... ,fn:t1,t2, ... ,tn

Listar sólo los registros entre las claves f1-fn y t1-tn Si se especifica la opción -r, se debe suministrar un par de números de registros relativos.

-i nombre. Leer utilizando el índice “nombre”.

Opciones sólo aplicables al utilitario *imp*.

-x Los registros que se reciben por la entrada deben ser eliminados de la tabla.

-l No trabar la tabla.

-g Ignorar errores de fuera de rango (asumir null).

-Gn Usa transacciones que encierren n registros.

Notas

Los carácteres RS y FS pueden ser indicados con:

- carácteres ASCII. Por ejemplo -F,.
- ^X (X es un carácter entre A y Z ó [y _) para caracteres de control. Por ejemplo -F^ A.

Especificando un carácter de control con las secuencias del Lenguaje ÒCÓ: \b\ t\ f \ n\ r. Por ejemplo -R\n.

- Con códigos numéricos en octal, decimal o hexadecimal.
- En octal usar \ OOO (O es un dígito octal).
- Para especificar en decimal usar \ dDDD (D es un dígito decimal).
- En decimal y octal se pueden colocar de uno a tres dígitos.

- En hexadecimal usar \ xXX (X es un dígito hexadecimal). Se pueden colocar uno o dos dígitos.

Recordar que los caracteres ^ y \ son interpretados por el Shell. Usar alguna forma de escape, por ejemplo precediendo por una barra o encerrando entre comillas.

Ejemplos

Si existe un esquema llamado pers, que contiene una tabla

```
$ exp pers.emp
```

Si existe un índice para dicha tabla llamado categ y se desea ordenar la salida de acuerdo a él, pero sólo se desea emitir las categorías entre 100 y 200

```
$ exp -i categ -k100:200 pers.emp
```

Si los registros seleccionados en la opción anterior se desean eliminar de la tabla:

```
$ exp -i categ -k100:200 pers.exp | imp -x pers.emp
```

Diagnósticos

Mensajes de error exp/imp:

`esquema invalido: ESQUEMA`: No puede accederse al esquema indicado. Puede ser no existente o bien no tener permiso para accederlo.

`Falta el nombre de la tabla`: No se ha indicado en los argumentos la tabla del esquema.

`Tabla inválida: TABLA`: La tabla indicada no existe en el esquema.

`Campo inválido: CAMPO`:El campo indicado no pertenece a la tabla con la cual se trabaja.

`Indice INDICE inexistente`: No existe un índice con el nombre INDICE para la tabla.

Consultar

Capítulo 6

IXFG

Reactivar la última tarea suspendida (WM).

Sintaxis

```
ixfg [-s]
```

Descripción

Este comando es utilizado para reactivar el último proceso usuario suspendido en modalidad residente.

• Si no hubiera procesos suspendidos desconectará al WM.

• Si los hubiera, reactivará al último suspendido y no permitirá “Retornar al Shell”. Una vez que todos los procesos usuarios hayan terminado, desconectará el WM y retornará al shell.

La invocación a este comando con el argumento “-s”, puede ser sustituido por el comando *wmstop* de idéntico comportamiento.

Consultar

Capítulo 3, Suspensión en Modalidad Residente, *Manual del Programador*.

FRECOVER

Reconstruir índices de una tabla.

Sintaxis

```
frecover [-b][-v][-f][-u][-cN][-dN] esquema[.|][tabla]
[indice...]
```

Descripción

Este programa reconstruye los índices especificados de la tabla dada. Se utiliza en los casos en que se produce algún inconveniente que provoca que la información en los índices no concuerde con la información en el archivo de datos (daño físico a un archivo índice por ejemplo).

esquema puede ser el nombre de un esquema, en cuyo caso se utiliza el argumento *tabla* para completar la definición de la tabla y el esquema que se utilizarán, o bien se especifica “esquema.tabla”, en cuyo caso el argumento *tabla* debe ser omitido. Si la lista de índices está vacía se reconstruyen todos los índices de la tabla dada. Especificando solamente el esquema se reconstruyen todos los índices de todas las tablas que lo componen.

Si la(s) tabla(s) sobre las cuales se recuperarán índices contienen volúmenes considerables de información, el proceso de reconstrucción puede insumir un tiempo apreciable. En tal caso es recomendable usar la opción “-v” para tener información de la evolución del proceso.

Opciones

-b No emitir el mensaje identificatorio.

-v Verbose. Muestra las operaciones a medida que son ejecutadas.

-f Colocar el contador de accesos concurrentes del esquema en 0.

-u Forzar la reconstrucción de los índices aunque no estén marcados como corruptos.

-p Comprime la tabla removiendo registros borrados.

-cN Usar N k-bytes de cache para los índices.

-dN Usar N-kbytes de cache para los datos.

-r[índice] Comprime una tabla usando *index*.

Ejemplo

Se desean reconstruir todos los índices de la tabla “emp” del esquema “personal”:

```
$ frecover -v personal.emp
```

Si sólo se deseara recuperar el índice “categ”:

```
$ frecover -v personal.emp categ
```

Diagnósticos

`Esquema inválido: ESQUEMA`: El esquema indicado no pudo ser abierto. Puede no existir o bien el usuario no tiene permiso para utilizarlo.

`Falta el nombre de la tabla`: No se ha suministrado el nombre de la tabla entre los argumentos del programa.

`Tabla inválida: TABLA`: La tabla indicada no pertenece al esquema.

`Indice inválido: INDICE, ignorado`: El índice no existe. Se ignora la especificación, continuando el proceso de recuperación con el siguiente índice.

IPG

Muestra interactivamente archivos por pantalla (IDEAFIX pg).

Sintaxis

```
ipg [-p proceso] [-wFILASxCOLUMNAS] [archivo...]
```

Descripción

Este comando permite ver un archivo o una salida de otro comando, dentro de una ventana, dando la posibilidad de desplazarse mediante las teclas de cursor o de paginación.

El *ipg* a diferencia de los comandos de Unix *pg* y *more*, representa correctamente los carácteres gráficos de los archivos o procesos seleccionados.

Opciones

-p proceso: Esta opción ejecuta el proceso pasado como parámetro y muestra su salida en una ventana con posibilidad de movimiento por línea y página.

-wFILASxCOLUMNAS: Define la salida en el número de filas y columnas especificados.

-**archivo**: Permite ver de manera interactiva el conjunto de archivos invocados, los comandos soportados son:

n: archivo siguiente.

p: archivo anterior.

q: terminar.

Es posible definir un conjunto de archivos mediante la utilización de metacarácteres y/o colocando la lista de archivos que se desean ver.

Ejemplo

El siguiente ejemplo permite ver un conjunto de archivos con terminación ".fm":

```
$ ipg *.fm
```

En el próximo ejemplo vemos la salida del comando pasado como parámetro dentro de una ventana de veinte filas por 80 columnas:

```
$ ipg -w20x80 -p make sto501
```

Vemos ahora una caso similar al anterior pero capturando la salida a través de un pipe:

```
$ make sto501 | ipg -w20x80
```

Notas

Hemos visto que existen dos formas de obtener la salida de un proceso mediante el comando ipg, se puede realizar con la opción -p, o a través de un pipe.

El comportamiento en ambos casos es el mismo salvo por un punto, el ipg, utilizando la capacidad de suspender procesos (tecla interrumpir) brindada por el Window Manager, permite suspender el proceso corriente y retornar al shell.

Si se encontrara trabajando con la opción -p, al digitar la tecla interrumpir el proceso lanzado quedará suspendido y podrá retornarse al shell.

Sin embargo si la salida fuera capturada mediante un pipe, al digitar la tecla interrumpir el proceso lanzado será abortado y el *ipg* sólo mostrará la salida del proceso hasta ese momento.

Es aconsejable entonces utilizar el pipe únicamente cuando no se desee interrumpir el proceso lanzado.

Consultar

Capítulo 3 - Funciones del Window Manager, *Manual del Programador*.

MENU

Intérprete de especificaciones de menúes.

Sintaxis

```
menu titulo archivo[.mn]
```

Descripción

Este utilitario muestra una ventana menú, de acuerdo a la especificación de menú dada en archivo. El argumento titulo es utilizado como etiqueta de la primera ventana menú. En la línea de información de la pantalla se mostrará el nombre de la empresa que se toma de la variable de ambiente EMPRESA.

El usuario podrá seleccionar una opción de las presentadas, y lanzarla a ejecución.

Notas

Sistemas UNIX o compatibles:

ulimit: El programa *menu* consulta una variable de ambiente llamada ulimit. Si existe, define para los procesos hijos dicho valor como "ulimit".

setuid bit: Para poder modificar el “ulimit” el menú corre con el setuid bit del superuser. Sin embargo esto ocurre en el momento inicial de la carga del programa. Terminada la operación de ulimit, la identificación de programa se redefine a la del usuario que lo invocó.

Ejemplo

Si tenemos el menú principal de un sistema de contabilidad en un archivo llamado cnt000.mn:

```
$ menu "Sistema de Contabilidad'' cnt000
```

Opciones

-c pregunta por la confirmación antes de dejar el menú.

Diagnósticos

`No pudo accederse al menu '\% s'`: No se pudo acceder al archivo con la especificación del menú. Puede no existir, o bien el usuario no posee permiso de lectura.

`Menu MENU: Error de sintaxis en línea NN`: El archivo de especificación de menu tiene un error de sintaxis.

`Menú MENU: Tipo de opción no reconocido en línea NN:TIPO`: La opción encontrada en la línea NN no está dentro de las válidas.

`Menu MENU: No se definieron opciones!`: El archivo de especificación existe pero no posee opciones (está vacío).

`Menu MENU: Demasiadas opciones definidas (Max. NN)`: El archivo de especificación posee más opciones que las permitidas.

Consultar

Capítulo 7, *Manual del Programador*, para consultar el formato de los archivos de especificación ".mn".

NFILES

Cantidad máxima de archivos por proceso.

Sintaxis

```
nfiles
```

Descripción

Este comando permite averiguar la cantidad máxima de archivos que un programa puede

tener abiertos simultáneamente. Es útil para definir con este valor la variable NFILES, que informa tal valor al RDBMS de IDEAFIX. El uso de este programa tiene sentido para IDEAFIX sin InterSoft Essentia.

Ejemplo

```
$ nfiles
```

En Unix es conveniente definir en el archivo de inicio de sesión del sistema (.profile, .login) el valor de la variable NFILES con:

```
NFILES=″34:256″
```

Donde 34 es el número retornado por nfiles, y 256 es el máximo número de “archivos virtuales” que IDEAFIX puede abrir. Se recomienda restarle 5 al número retornado por nfiles. Esto dejará sin efecto problemas innecesarios con archivos que no puedan ser “virtualizados”.

PM

Printer Manager.

Sintaxis

```
pm [-b] [-iNOM_CAP] [-f] [-o archivo] modelo [archivos ...]
```

Descripción

El Printer Manager (pm) es un filtro configurable para impresoras. Este programa lee su entrada y traduce carácter por carácter de acuerdo al archivo de capacidades de la impresora “modelo”.

Si no se especifican archivos, se lee la entrada estándar.

La impresión comienza enviando la secuencia INICIALIZAR. Luego se envían las secuencias especificadas en la opciones -i, si las hay, y finalmente se imprimen los archivos en el orden dado.

Luego de cada archivo se envía la secuencia CANCELATRS, y al finalizar la impresión completa se envía FINALIZAR.

Opciones

-b No emitir el mensaje identificatorio.

-iNOM_CAP Enviar la capacidad NOM_CAP antes de imprimir. Puede ser cualquiera de las

capacidades especificadas en la tabla de atributos del Printer Manager (Ver tabla 7.3).

Varias de estas opciones pueden ser combinadas.

-f Enviar un form feed luego de imprimir cada archivo.

-o archivo Escribir la salida en archivo

Ejemplo

Si el archivo “datos” tiene información en el set de caracteres de IDEAFIX, y se desea enviarlo a una impresora modelo “brand”, pero en letra comprimida (es decir, 16 c.p.i):

```
$ pm -iCPI16 brand | lp
```

El ejemplo está destinado a sistemas donde el Printer Manager no está integrado al sistema de “spooling”.

Diagnósticos

Los siguientes errores son provocados por errores sintácticos en el archivo de descripción de capacidades de la impresora. En todos los casos modelo se refiere al archivo de descripción de dicha impresora, y línea indica la línea del mismo donde se detectó el error.

`modelo(linea): Capacidad NOM_CAP no reconocida`: La capacidad NOM_CAP no es ninguna de las capacidades soportadas por el Printer Manager. Si modelo es la indicación cmdline, la capacidad se ha referenciado en una opción “-i”, y línea es el número de opción “-i” donde está el error.

`modelo(linea): Nombre de carácter NOM_CAR no reconocido`: El carácter NOM_CAR no pertenece al conjunto de caracteres de IDEAFIX.

`modelo(linea): Falta string de asignacion para NOMBRE`: Falta asignar la secuencia de caracteres a enviar correspondiente a la capacidad o al carácter NOMBRE.

Los siguientes errores son durante la ejecución del programa:

`No se encuentra descripción de impresora IMPRESORA`: No puede accederse al archivo con la descripción de IMPRESORA.

`No pudo abrirse archivo '% s' para lectura`: No puede accederse a alguno de los archivos suministrado como parámetro. El archivo es ignorado y se pasa al siguiente de la lista de parámetros.

`No pudo abrirse archivo '% s' para escritura`: No puede abrirse el archivo indicado en la opción “-o”.

Consultar

Capítulo 7.

ROLLBACK

Elimina transacciones incompletas.

Sintaxis

```
rollback
```

Descripción

Este comando permite eliminar transacciones incompletas, ocasionadas por interrupciones que no pueden ser atendidas por software (corte de energía, caídas del equipo, etc.).

Diagnósticos

El utilitario puede emitir los siguientes mensajes:

`LOG inactivo`: El Log no está activo, la variable de ambiente LOGDIR no posee valor.

`No hay archivos de LOG para procesar`: No se han encontrado procesos con transacciones incompletas.

Consultar

Capítulo 7.

WM

Manejador de Ventanas (Window Manager).

Sintaxis

```
wm [-b][-a][-t timeout] [-w archivo] [-r archivo] [tarea
[args]]
```

Descripción

Este programa maneja todas las operaciones de E/S de la terminal para programas de IDEAFIX. Los programas que necesitan del WM para su ejecución son llamados “Programas Usuarios del WM” o simplemente “Programas Usuarios”.

El Window Manager lee la variable de ambiente TERM para determinar el modelo de terminal que utiliza el usuario, y luego lee el archivo ".map" correspondiente para configurar el teclado y la pantalla.

Opciones

-b No emitir mensaje identificador.

-a No permite abortar el programa usuario.

-t timeout: Establece el tiempo de cancelación en segundos, para los procesos usuarios (default = 5).

-r archivo: Lee del archivo *archivo* la lista de instrucciones para trabajar en modalidad demo.

-w archivo: Graba sobre archivo la lista de instrucciones para trabajar en modalidad "demo".

-T *ntareas*: Máximo número de tareas corridas desde el WM.

-d *device*: usa el dispositivo device como terminal.

tarea [args]: Tarea a invocar, con posibles argumentos. Si se invoca al WM sin tarea, correrá en modalidad residente; si se le especificara al menos una, entonces lo hará en modo no residente.

Notas

La opción **-r** simula la ejecución de un programa usuario del WM, pero en lugar de tomar comandos del teclado, los lee del archivo que compone la opción.

La opción **-w** genera el archivo, con el nombre que se le coloque como parámetro. Dicho archivo puede ser creado o modificado mediante un editor de texto.

El archivo de entrada admite el siguiente formato:

Líneas que comienzan con un ".", contienen un comando especial. Los comandos posibles son:

`.SLEEP [n]` Detener el proceso durante "n" segundos.

`.PAUSE [Texto]` Abrir una ventana que muestre el "Texto" y esperar que se oprima una tecla para continuar.

`.DELAY [n]` Define el intervalo de tiempo en "n" segundos entre la ejecución de un comando y otro. El valor inicial es 0.

`.STEP [n]` Define si se ejecuta paso por paso, es decir, luego de cada comando se espera una tecla para ejecutar el siguiente. Si "n" es 0 este modo se cancela, y si es 1 se activa.

`.COMMENT [C]`

```
.................

.. texto ...

.................

.END COMMENT
```

Arma una ventana con el texto indicado. Si se especifica el parámetro “C” el texto se centra línea por línea en la ventana. El tiempo que la ventana se mantiene desplegada es calculado automáticamente a partir de la longitud del texto.

`.STOP` Esta sentencia termina la ejecución

`.TECLA-IDEAFIX` Es el nombre de una tecla de función de IDEAFIX. Por ejemplo: PROCESAR, IGNORAR, etc.

Líneas que no comienzan con “.” tienen texto que será tomado como entrada.

Comentarios, que se indican con una línea que comienza con un “#”. Los comentarios son ignorados en tiempo de ejecución.

`# Comentarios` - No afectan la ejecución de los comandos

Diagnósticos

Los siguientes errores se producen debido a errores indicados en el archivo de definición de terminal de IDEAFIX. En todos los casos modelo indica el archivo donde se produce el error y línea es el número de registro donde se ha detectado.

`modelo(línea): Nombre NOMBRE no reconocido`: El nombre de una tecla o capacidad no es reconocido.

`modelo(línea): Falta secuencia de entrada para tecla TECLA`: No se ha suministrado la secuencia que envía el teclado para definir TECLA.

`modelo(línea): Falta el texto de descripción de tecla TECLA`:

`modelo(línea): Secuencia de tecla TECLA es prefijo de una ya definida`: La secuencia suministrada para TECLA no es válida, ya que sus primeros caracteres corresponden a una tecla ya existente.

`modelo(línea): Secuencia no válida para tecla TECLA`:

`modelo(linea): Descripción de terminal muy grande`: Este error proviene de la descripción en la Base de Datos de terminales del Sistema Operativo (en UNIX termcap o terminfo). La descripción de la terminal es muy grande y no cabe en la memoria asignada.

`modelo(linea): Tecla de INTERRUPCION debe ser un solo carácter`: No se admiten secuencias de más de un carácter para definir la tecla de interrupción. Normalmente se la asigna a Ctrl-| en UNIX y Ctrl-C en DOS.

`modelo(linea): Descripción de terminal incompleta`: La descripción de la terminal existente en la Base de Datos de terminales del Sistema Operativo no contiene suficientes datos para el Window Manager.

`Faltan secuencias necesarias para el manejo de la terminal`: La definición de la terminal en terminfo carece de ciertas secuencias necesarias para el WM.

`No puede ejecutarse el proceso x`: El WM ha lanzado un programa que no existe o que tiene problemas de ejecución.

`WM está corriendo`: Se a lanzado el WM y éste ya está corriendo, si se lo desea ejecutar en modo no residente, debe desconectarse con fg -s el que se encuentra corriendo actualmente, en otro caso se debe ejecutar el programa usuario directamente.

`No se pueden crear los canales de comunicación`: Existen problemas al intentar crear los canales de comunicación para comunicar al WM con los procesos usuarios.

`No se puede instalar WM`: El WM no puede instalarse en modalidad residente, en general esto ocurre cuando se a saturado la cantidad de procesos en Unix.

`Atención: El Wm quedo corriendo`: Se ha salido de la cuenta sin desconectar el WM y teniendo tareas suspendidas (Modo Residente). Se debe entrar a la cuenta nuevamente y ejecutar el comando fg para reactivar las mismas.

`Variable de ambiente “printer” no encontrada`: No se ha definido la varible de ambiente printer y es necesaria para realizar el servicio de impresión de pantalla.

`Error de Instalación: WM debe ser Super-User`: El WM no ha sido instalado como Super-User. Debe cambiarse el modo del WM.

`No hay mas canales de comunicación`: Debe reconfigurar el IPC, comuníquese con Personal Técnico: Se ha excedido la cantidad de canales de comunicación de la instalación, ésta debe reconfigurarse en un número mayor.

`No definido mapa de Input-Output para terminal TERMINAL`: No se dispone del mapa de terminal IDEAFIX para la terminal definida en la variable de ambiente TERM.

`***TERMINAL indefinida en Base de Datos de capacidades de terminal`: No se encuentra en terminfo (base de datos de terminales) la descripción de la terminal indicada en la variable de ambiente TERM.

Errores de Ejecución en modo demo:

`Línea NN: Mandato inválido COMANDO`: Se ha encontrado en el archivo de instrucciones un comando no reconocido.

`No puedo abrir ARCHIVO`: No puede accederse al archivo de instrucciones. Revisar si existe y sus permiso de acceso.

Consultar

Capítulo 2, Modalidades del Window Manager, *Manual del Programador*.

Capítulo 5.

Apéndice D.

Capítulo 9

Archivos de configuración

Este capítulo describe el formato de distintos tipos de archivos utilizados para la configuración de IDEAFIX en distintos ambientes de ejecución. Entre ellos se encuentran: el configurador de terminales e impresoras; las ayudas en línea (helps); archivos de mensajes parametrizados, y la tabla de feriados.

Configuración de Terminales

Introducción

Debido a la falta de estandarización, los ambientes de ejecución de programas varían sustancialmente de una instalación a otra. Esto se debe, entre otras cosas, a los diferentes modelos de teclados y pantallas (es decir terminales) con que el usuario se enfrenta.

Los programas de aplicación generados con IDEAFIX presentan al usuario una Interface uniforme, basada en la terminal “virtual” de IDEAFIX aislando de este modo las dependencias del hardware y uniformizando el ambiente de trabajo.

La terminal “virtual” de IDEAFIX se basa en un display (pantalla) capaz de representar un conjunto de 256 caracteres (siendo los primeros 128 ASCII, y el resto usado para caracteres gráficos y lingüísticos) y un teclado ASCII estándar con 32 teclas de función. Cada una de estas teclas de función tiene un nombre con el que se la denomina en este manual y en todos los programas y documentación de IDEAFIX.

Lógicamente la terminal “virtual” puede diferir de la terminal real, pero permite establecer un estándar de Interface. Cuando un usuario trabaja en una cierta terminal física, se realiza una correspondencia lógica entre ambas terminales.

Hay un archivo de configuración para cada tipo particular de unidad, que define la relación entre la terminal “virtual” de IDEAFIX y el modelo “real” que se utiliza. Mediante este

método se aprovechan todas las características de los modelos de terminales sofisticados (teclas especiales de función, caracteres gráficos), manteniendo la compatibilidad con los modelos más simples.

El Conjunto de Caracteres de IDEAFIX

El conjunto de caracteres de IDEAFIX es básicamente el estándar ASCII con más 32 caracteres de control (llamados caracteres de función de IDEAFIX, y la extensión para alfabetos europeos y caracteres gráficos. La disposición de los códigos está mostrada en la en la tabla del Apéndice B.

Los códigos reservados para las teclas de función (128-159) no tienen representación en la pantalla.

0-31 Carácteres de control ASCII no usados por IDEAFIX

32-127 Juego estándar ASCII

128-159 Teclas de códigos de función de IDEAFIX

160-255 Carácteres gráficos y alfabéticos europeos

El Teclado

El teclado de la terminal virtual de IDEAFIX posee los caracteres ASCII más 32 teclas de función usadas en los utilitarios y programas de aplicación para que el usuario indique acciones a realizar.

Los nombres de las teclas de función están indicados en el Apéndice A, donde la primera columna brinda el nombre de la tecla, y la segunda su función.

Todo el manejo de ventanas y Entrada-Salida de las terminales se hace a través de un œnico proceso llamado Window Manager. Este administra el teclado y la pantalla, para proveer la misma Interface lógica independientemente de los Sistemas Operativos o modelos de terminal. El siguiente diagrama muestra cómo trabaja el Window Manager (WM):

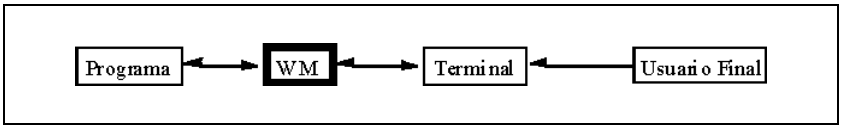


Figura 9.1 - El Manejador de Ventana

Todos los utilitarios de IDEAFIX y los programas de aplicación utilizan este proceso para realizar la E/S por teclado y pantalla.

Este es el proceso que lee el archivo de configuración para adaptar la terminal “virtual” a la terminal real en uso.

El problema para realizar la adaptación es que cada fabricante de terminales tiene su propio estándar de secuencias de control y conjunto de caracteres. Para obtener una Interface uniforme en IDEAFIX, este problema se divide en tres partes:

- Secuencias de control de la terminal (posición del cursor, borrado, etc.)
- Conjunto de caracteres que dispone la pantalla.
- Definición del teclado (teclas de función, caracteres especiales, etc.) Las soluciones adoptadas son:

La base de datos de descripción de terminales estándar provista por el sistema operativo para manejar las capacidades de la terminal.

Un archivo de configuración de IDEAFIX para definir las secuencias que las teclas de función envían al programa y cómo obtener cada carácter del conjunto de caracteres IDEAFIX en la pantalla.

Configuración de la Terminal del Usuario

Las características específicas de cada terminal están definidas en un archivo de configuración con extensión “.map”. Cuando el Window Manager comienza, lee este archivo correspondiente al modelo de terminal corriente para obtener su definición y configuración.

El archivo “`.map`” está dividido en dos secciones: una para el teclado y la otra para la pantalla.

La sección destinada al teclado contiene las asignaciones de las teclas de función de la terminal con la teclas de función de IDEAFIX.

La sección destinada a la pantalla define la forma de obtener cada carácter del conjunto de caracteres de IDEAFIX en la pantalla física.

Estos son los archivos que definen la relación entre un modelo de terminal particular y la terminal virtual de IDEAFIX. Debe existir un archivo por cada modelo de terminal en su sistema, que será leído por el Window Manager, que en el momento de su ejecución tomará el modelo de terminal de la variable de ambiente TERM.

Por ejemplo si se tiene conectada una terminal VT220, la variable TERM tendrá el valor:

```
TERM=vt220
```

En esta situación debe existir un archivo vt220.map que describa las características de las terminales modelo VT220 para el Window Manager, y que reside en un directorio llamado “map”. Este debe ser uno cualquiera de los subdirectorios indicados en la variable de

ambiente PATH.

En el archivo ".map" se definen 2 aspectos:

- Las secuencias que envía el teclado para cada carácter del set de IDEAFIX (sección INPUT)
- Las secuencias que deben enviarse a la pantalla para obtener cada carácter del set de IDEAFIX (sección OUTPUT).

Es estrictamente necesario que exista el archivo .map correspondiente a la terminal. Sin embargo éste puede estar vacío o incompleto, en cuyo caso se usan las asignaciones por defecto.

Sintaxis de los Archivos *.map*

La sintaxis de un archivo "`.map`" es la siguiente:

```
INPUT:
codigo_de_tecla Secuencia Descripcion
OUPUT:
codigo_de_caracter [atributos ... ] codigo
```

Las líneas en blanco son ignoradas, al igual que las que comienzan con un "#" (utilizadas para insertar comentarios).

La sección INPUT

Se especifica para cada tecla de IDEAFIX cuál es la secuencia que recibirá de la terminal. Tomemos por ejemplo la tecla <PROCESAR> y supongamos que deseamos asociarla a una tecla rotulada F5 en nuestra terminal. La secuencia de escape para F5 es "ESC 0 T" (escape-cero-te). Entonces escribimos en el archivo:

```
PROCESAR: \E0T F5
```

Los caracteres \E representan el código asociado a tecla de 'escape': el carácter ASCII (27). El texto descriptivo referente a F5 se mostrará cuando el usuario pida ayuda sobre "Configuración de Teclado".

En general, como "código_de_tecla" se puede indicar:

Un nombre simbólico que indica una tecla de función o un carácter especial. Estos nombres se muestran en la tabla del Apéndice A.

Un código de carácter numérico precedido por la barra inversa o "backslash" ('\'). Se define la tecla por su código numérico.

Por ejemplo, la tecla <PROCESAR> tiene el código 128, por lo que el ejemplo anterior se podría haber escrito:

```
\d128 \E0T F5
```

Esta característica permite redefinir todo un teclado. Si hacemos:

```
\d65 \B B=A
```

esto significa que el código 65 ('A') se obtiene cuando del teclado se recibe la B. Estas posibilidades permiten manejar un teclado EBCDIC en una máquina ASCII.

Un solo carácter. Es similar al caso anterior, sólo que en lugar de un código numérico se indica directamente el carácter. Por ejemplo la siguiente línea redefine la comilla simple:

```
' '' ''
```

Lo cual define que para ingresar una comilla simple o apóstrofo, se deberá oprimir la tecla (') dos veces.

TECLAS DE FUNCION	DESHACER	INTERRUMPIR	IGNORAR
INGRESAR	CURS_ABA	CURS_DER	PAG_ANT
PAG_SIG	DESHACER	INSERTAR	ELIMINAR
BORRA_CAMPO	AYUDA	REDIBUJAR	SERVICIOS
CONF_TECL	META	CTRLX	AF_1, AF_2, AF_3, AF_4
PROCESAR	FIN	CURS_IZQ	CURS_ARR
PAG_DER	RETROCESO	AYUDA_APL	SUSPENDER
TAB			

Figura 9.2 - Tabla de Funciones

CARACTERES GRAFICOS	VERT	HORIZ	ABIZQ
ABAJO	ABDER	DER	CEN
IZQ	ARIZQ	ARRIBA	ARDER
BALA	FLEARR	FLEABA	FLEIZQ
FLEDER	DVERT	D_HORIZ	D_ABIZQ
D_ABAJO	D_ABADER	D_DER	D_CEN
D_IZQ	D_ARRIZQ	D_ARRIBA	D_ARRDER
BLOQUE	DS_ABAJO	DS_DER	DS_IZQ
DS_ARR			

Figura 9.3 - Tabla de carácteres gráficos

La sección OUTPUT

En esta sección se define el código que se debe enviar a la pantalla para obtener un carácter dado. La sintaxis es:

```
codigo_de_caracter [atributos ... ] código
```

codigo_de_caracter sigue las mismas reglas que en la sección INPUT. Por ejemplo si en la terminal para dibujar la letra “á” se debe enviar el código 160, se escribe

```
a_acent \ d160
```

Es posible agregarle atributos a un código. Por ejemplo si los caracteres gráficos requieren pasar la terminal a modo gráfico mediante el atributo “set de caracteres alternativos”:

```
VERT ALTERNATE \d120
```

indica que la línea gráfica vertical (\) se logra en el conjunto alternativo con el código 120. Los nombres de los atributos se encuentran en la tabla de la Figura 9.4. La tercera columna contiene el código numérico para el atributo[1]

Nombre	Código	Significado
NORMAL	0	Letra normal
RESALTADO	1	Intensifica
STANDART	2	Atributo para resaltar
SUBRAYADO	4	Letra subrayada
TITILANTE	5	Titilante
ALTERNATIVO	6	Modo Gráfico
INVERSO	7	Video Inverso

Figura 9.4 - Tabla de Atributos

Configuración de Impresoras (/cap)

[1] Algunos de estos atributos son redundantes porque implementan el mismo atributo en diferentes modos; así que atributos que en teoría harían el mismo trabajo trabajarían en diferentes terminales. El atributo apropiado para cada terminal debe ser determinado por el programador.

Introducción

Los archivos de capacidades definen como obtener el conjunto de caracteres de IDEAFIX y distintos atributos de impresión en cada modelo de impresora. Este archivo será leído por el Printer Manager de acuerdo al modelo de impresora suministrado. Por ejemplo si ejecutamos:

```
pm proprinter
```

debe existir un archivo “proprinter.cap” en un directorio “cap”. Este directorio debe ser un subdirectorio de cualquiera de los indicados en la variable de ambiente PATH.

En el archivo “.cap” se definen dos aspectos:

- La forma de obtener cada carácter del conjunto de caracteres de IDEAFIX en la impresora.
- Cómo seleccionar atributos y determinar parámetros de impresión.

Es estrictamente necesario que exista el archivo “.cap” correspondiente a la impresora a usar. Sin embargo éste puede estar vacío o incompleto, en cuyo caso se usan las asignaciones por defecto.

Sintaxis

La sintaxis de estos archivos es la siguiente:

```
CHARS:
codigo_de_caracter secuencia
ATTRIBUTES:
atributo secuencia
```

Las líneas en blanco son ignoradas, al igual que las que comienzan con un “#” (utilizadas para insertar comentarios)

La sección CHARS

La parte `codigo_de_caracter` responde a las mismas reglas establecidas anteriormente. Se definen los caracteres a enviar a la impresora para obtener cada carácter del conjunto de caracteres IDEAFIX. Si por ejemplo en la impresora la “á” se obtiene enviando el código 160, se indica

```
a_ACENT \d160
```

Si la unidad no tuviera letras acentuadas en su conjunto de caracteres, igualmente podemos definir:

```
a_ACENT a^H'
```

que compone la 'a' acentuada mediante un retroceso (backspace) y la sobreimpresión del acento o comilla que hace sus veces.

La sección ATTRIBUTES

En esta sección se indica la forma de obtener atributos, modalidades de impresión y definir parámetros. La tabla siguiente muestra los nombres de los atributos existentes:

Atributos del Printer Manager	LONGFORM	TAB
CPI15	CP18	CPI10
CPI12	LP16	LP18
INVERSO	RESALTADO	LETCALIDAD
ITALICA	SUBRAYADO	CANCELATRS
INICIALIZAR	FINALIZAR	CAPUS[1234567]

Figura 9.5 - Atributos del Manejador de Impresora.

Archivos de Ayuda en Línea (/hlp)

En directorios con nombre “hlp” dependientes de algœn directorio indicado en la variable PATH se encuentran los archivos con extensión `.hlp` para dar ayuda al usuario.

Existe un directorio provisto por IDEAFIX (`/usr/ideafix/bin/hlp`) que tiene los archivos de ayuda para utilitarios como el editor, el menœ, etc.

Estos archivos pueden tener secuencias para desplegar atributos insertadas con el programa “ie” (IDEFIX Editor) para una mejor apariencia y facilitar la lectura del texto de ayuda.

Archivo de Mensajes Parametrizados

El sistema de archivos de mensajes de error que invocan los comandos y utilitarios de IDEAFIX están disponibles en dos idiomas: Español e Ingles.

Los mensajes se alojan en directorios que dependen del subdirectorio data del directorio raíz de IDEAFIX. Esto quiere decir que si por ejemplo el directorio raíz de IDEAFIX es:

```
/usr/ideafix
```

y el lenguaje utilizado es el castellano, los mensajes de error que invocarán los utilitarios estarán ubicados en el directorio:

```
/usr/ideafix/data/spanish
```

Los utilitarios ubicarán estos mensajes utilizando las variables de ambiente IDEAFIX y LANGUAGE. Esta variables podrán tener, por ejemplo, los siguientes valores:

```
IDEAFIX=/usr/ideafix
LANGUAGE=spanish
```

por lo tanto los utilitarios utilizarán el camino:

```
$ IDEAFIX/data/$LANGUAGE
```

Archivo de Feriados

IDEAFIX permite manejar los feriados del calendario, para cálculos de fechas o intervalos de tiempo en función de días hábiles o días calendario (corridos).

El archivo da información sobre los días feriados, explicitando la fecha, y una descripción del motivo del feriado. Su formato es muy simple, ya que se trata de un archivo de texto, donde cada entrada corresponde a una jornada no laborable. El formato de cada línea es:

```
dd/mm/aa,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

donde `/mm/aa` es la fecha del feriado y `x... x` es una descripción de hasta treinta caracteres.

El `/aa` no es obligatorio. Si se coloca, el feriado sólo será válido en el año indicado, de lo contrario valdrá todos los años.

El nombre de dicho archivo es feriados.dat.

Archivo de Mensajes del Correo

El archivo denominado "`.mlist`" es la ayuda que el usuario dispone para ejecutar el utilitario imail, que permite asociar el nombre de un usuario con la secuencia requerida por dicho utilitario para rutear el mensaje que se desea enviar.

Capítulo 10

Servicios del Window Manager

Este capítulo describe los servicios incorporados al administrador de ventanas (Window Manager).

Activando los Servicios

Se accede mediante la tecla SERVICIOS, la que al ser oprimida desplegará una ventana tipo menú con los servicios de que dispone el usuario, como muestra la figura.

Figura 10.1 - Menú de Servicios.

Usualmente estos servicios incluyen:

Salida al intérprete de comandos del sistema operativo a través de la opción rotulada como Shell. El programa queda suspendido y se ejecuta el intérprete de comandos. Cuando finaliza la sesión, se redibuja la pantalla y se retorna al programa.

Impresión de la pantalla. Permite realizar una copia del contenido de la pantalla tal como está al momento de invocar esta opción.

Calculadora. En la pantalla aparece el dibujo de una calculadora común de bolsillo, por medio de la cual se pueden efectuar operaciones de todo tipo. Invocando esta función desde un campo numérico mediante la tecla <TAB> es factible cargar el resultado en forma directa sobre ese campo.

Conjunto de caracteres de IDEAFIX. Muestra en una ventana tipo "Ayuda" la representación del conjunto de caracteres IDEAFIX en la terminal. Se muestra el código de cada carácter.

Agenda / Calendario. Permite la administración de tareas y la consulta de fechas y plazos.

Correo de usuarios. Facilita el intercambio de mensajes entre usuarios.

Posicionamiento de Ventanas. Brinda la posibilidad de desplazar las ventanas desplegadas en la terminal a la posición deseada.

En las siguientes secciones se explica cada uno de estos servicios.

Es posible configurar el menú de servicios para cada usuario. Esto se debe a que cada usuario tendrá en su directorio HOME un archivo llamado services.mn contemplando las distintas opciones a desplegar en el momento de invocarlos. El archivo tiene el formato de menú estándar (Ver Capítulo 5, del Manual del Programador), permitiendo su modificación e incluso su eliminación.

En los archivos services.mn se podrá usar un tipo de comando de menú llamado BUILTIN. Esta opción permite invocar un conjunto de funciones incorporadas dentro del Window Manager. Dichas funciones (del inglés built-in: internamente construídas), son las siguientes:

Función	Descripción
GoToShell	Ir al Shell
_print_scr	Imprimir Pantalla
_print_ichset	Juego de Carácteres
calculator	Máquina de Calcular
servmov	Posicionar Ventana

Para evitar el ingreso de un usuario al menú de servicios se debe eliminar el archivo services.mn del directorio HOME del usuario. Este archivo tiene originalmente el siguiente formato:

'Shell' BUILTIN GoToShell

'Imprimir Pantalla' BUILTIN _print_scr

'Juego de Carácteres' BUILTIN _print_ichset

'Máquina de Calcular' BUILTIN calculator

‘Agenda/Calendario’ WCMD ical

‘Correo’ WCMD imail

‘Posicionar Ventana’ BUILTIN servmov

Ingreso al Shell

Este servicio permite suspender temporariamente la tarea que se está ejecutando, e ingresar a un intérprete de comandos. Pueden resolverse así tareas puntuales desde el sistema operativo sin necesidad de dar por terminado el programa corriente.

Cuando se seleccione esta opción, se blanqueará la pantalla, y aparecerá el prompt del intérprete de comandos. Una vez ejecutadas las funciones necesarias, cuando termine la sesión la pantalla se restaurará al estado en que estaba antes de pedir los servicios del manejador de ventanas.

Es posible configurar la utilización de este servicio mediante la variable de ambiente SHELL. Esta variable define cuál es el intérprete de comandos que se invocará al seleccionar este servicio. Si la variable no está definida, en UNIX se asume el Bourne Shell, y en MS-DOS el programa command.com, que son los intérpretes de comandos usuales.

Es posible inhibir directamente el uso de este servicio, si a la variable SHELL se le da el valor '0'. Es útil cuando se desea que usuarios de baja prioridad o con escasas nociones del sistema, carezcan de acceso al shell. Si el ingreso al shell está inhibido, el menú de servicios lo mostrará, pero al intentar activarlo sonará la alarma audible de la terminal y no se realizará ninguna acción.

Impresión de la Pantalla

Esta función permite volcar el contenido de la pantalla de la terminal a una impresora o a un archivo. Cuando se selecciona esta opción desde el menú de servicios, se podrá ver al cursor desplazarse rápidamente por la pantalla a medida que va copiando cada línea en la salida.

El resultado de la copia se almacenará según la indicación presente en la variable de ambiente “printer”, que podrá enviar la salida a una de las impresoras del sistema, o bien a un archivo.

Este servicio es útil para realizar documentación de aplicaciones, ya que permite “fotografiar” el estado de la pantalla en un momento dado. También permite a un operador, volcar el estado de la pantalla cuando se produce una falla en un sistema.

La Calculadora

Cuando se selecciona esta opción del menú de servicios, se presentará en pantalla una

calculadora de bolsillo que permite realizar todo tipo de operaciones, cuyo diseño en pantalla es el siguiente :

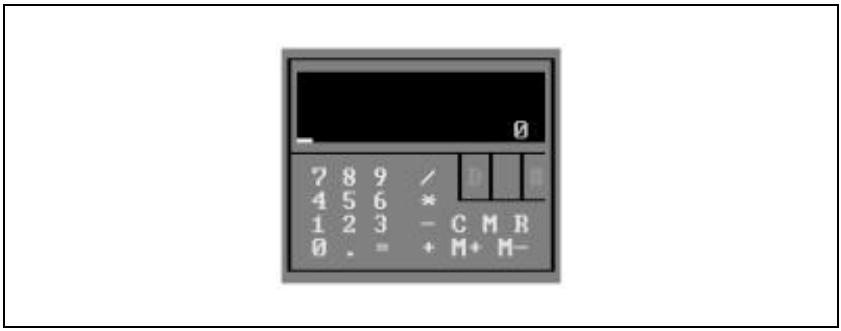


Figura 10.2 - La calculadora

La letra D está indicando que se trabaja en base decimal. El sector donde se encuentra el # contiene siempre el último operador digitado.

La forma de operarla es la forma común de manejar un calculador de bolsillo. Es posible obtener una descripción de las funciones de la calculadora oprimiendo la tecla AYUDA, mediante la cual se obtendrá una ventana con la descripción de los comandos de manejo de memoria, cambio de base, etc. Estos comandos son los que se muestran a continuación:

[\%] Calcular el porcentaje.

[C] Borrar el último número u operación ingresada.

[RETROC] (backspace) Retroceder borrando el último dígito.

[AYUDA] Despliega o elimina la ventana con la descripción de los comandos.

[#] Cambio de base. Las bases soportadas son: Decimal (D), Binario (B), Octal (O) y Hexadecimal.

[-] Cambio de signo del número en display (subrayado o ‘underscore’).

[$] Definir la cantidad de decimales.

[E] Cambiar el formato de los números a exponencial.

[Q] Ingresar un 00.

[W] Ingresar 000.

[!] Calcular el factorial. Se ingresa el número y luego se oprime el !.

[A] Sumar el número en display a la memoria.

[L] Limpiar memoria.

[M] Ingresar el número en display a la memoria.

[R] Recuperar memoria (no restar: ver siguiente comando).

[S] Sustraer el número en display de la memoria.

[X] Intercambiar el contenido del la memoria con el display.

El Juego de Caracteres

Este servicio permite obtener una ventana de tipo ayuda donde se puede visualizar el conjunto de caracteres de IDEAFIX. Es un auxiliar útil para la programación cuando se desea conocer el código de un carácter, ya sea perteneciente a IDEAFIX o bien al conjunto ASCII.

Cuando se active el servicio aparecerá una ventana tipo menú para definir la base en la cual se mostrará el código numérico de cada carácter.

Si se oprime la tecla FIN en esta ventana se cancelará el servicio.

Agenda/Calendario

El servicio de Agenda/Calendario brinda la posibilidad de consultar un calendario contenido en una ventana de IDEAFIX, mostrando un mes completo. El usuario puede recorrerlo de manera rápida y fácil, por día, mes y año.

Integrado al Calendario existe una agenda que permite administrar compromisos y tareas (“La Agenda”), pudiéndose visualizar de forma gráfica el conjunto de tareas correspondientes a una fecha determinada.

En resumen este servicio permite las siguientes funciones:

- Posicionarse en una fecha determinada por el usuario y recorrer el Calendario por día, mes y año.
- Mostrar los feriados y obtener su descripción.
- Dar ingreso a tareas especificando, según el caso, su fecha, duración, fecha límite, título, descripción, etc.
- Ver en forma gráfica las tareas asignadas a un día, mostrando su duración y hora de comienzo.
- Administrar tareas que no tienen su fecha de realización asignada aún (Tareas Pendientes).

Ingreso al Servicio

El ingreso se hace de la misma forma que los demás servicios que provee IDEAFIX desde una aplicación, pero seleccionando, en este caso, la opción Agenda/Calendario.

Una vez que fue seleccionado aparecerá la siguiente ventana:

Figura 10.3 - El Calendario

Mostrando en video inverso el día correspondiente a la fecha corriente. Los días feriados se muestran resaltados.

La Agenda

La agenda de IDEAFIX permite llevar el control de compromisos o “tareas”, mediante la visualización del horario correspondiente a cada día. El horario muestra a lo largo de un día los compromisos y su duración.

La agenda maneja tres tipos de tareas:

Tareas Temporales. Estas tareas son aquellas que al ingresarlas se les asignó su fecha de realización y duración, por lo que, al darlas de alta se podrán visualizar en forma inmediata en el horario.

Tareas Pendientes. Estas tareas son aquellas que al ingresarlas no se conoce su fecha exacta de realización, pero sí su fecha límite, por lo cual no aparecerán agendadas en forma inmediata. Serán administradas por ésto, de otra forma que las anteriores, ésto se explicará luego en Menú de Tareas Pendientes.

Tareas Reiterativas. Estas tareas son aquellas que se realizan con una cierta periodicidad, por ejemplo todos los primeros de mes. Para el cálculo de esta periodicidad se pedirán los datos en el momento de su ingreso.

Componentes del Servicio

Los componentes del servicio de Agenda/Calendario son:

- Calendario.
- Administración de Tareas Temporales.
- Administración de Tareas Pendientes.
- Administración de Tareas Reiterativas.
- Horario.
- Menú de Tareas Pendientes.
- Ayuda.

Calendario

Permite visualizar un mes a elección del usuario, tiene facilidades para el desplazamiento por el mismo:

Las teclas de movimiento de cursor permiten el desplazamiento por día.

Las teclas de paginación el desplazamiento por mes.

A estas teclas puede anteponerse un número de cuatro dígitos como máximo, para obtener desplazamiento rápido por día, semana y mes. En tal caso, las teclas de cursor izquierda y derecha avanzan por día, las de cursor arriba y abajo por semana y las de página arriba y abajo lo hacen por mes. Por ejemplo:

4 CURSOR_LEFT: desplaza el Calendario cuatro días hacia atrás.

2 PAGE_DOWN: desplaza el Calendario dos meses hacia adelante.

9 CURSOR_DOWN: avanza nueve semanas.

La tecla T vuelve a posicionar el Calendario en la fecha corriente (Today).

Administración de Tareas Temporales

Este proceso se activa digitando la tecla PROCESAR desde el Calendario. Los campos de estas tareas son:

Fecha Indica la fecha asignada a la tarea.

Hora Indica la hora de inicio asignada a la tarea.

Duración Indica la duración de la tarea.

Título Título de la tarea. Se mostrará en el horario.

Pend. Indica si la tarea se halla pendiente (o es temporal).

Alarma Indica los minutos antes del comienzo de la tarea que debe sonar la alarma de aviso. En caso de no ingresarle valor alguno la alarma no será activada.

*Descrip*Cada tarea tiene la posibilidad de guardar cien ren glones de descripción, de cincuenta caracteres cada uno.

Las teclas de operación de este proceso están definidas en la ayuda del mismo, como en todas las aplicaciones de IDEAFIX.

A continuación mostramos el formato de la pantalla:

Figura 10.4 - Administración de Tareas

Administración de Tareas Pendientes

Es en todo similar al proceso de administración de tareas temporales, pero con dos diferencias:

- El campo “Pendiente” contendrá una 'S' en lugar de un código 'N' como en el caso anterior. Se activa digitando la tecla p desde el Ca-lendario.
- Los campos de estas tareas son similares a sus correspondientes para las tareas temporales, exceptuando Fecha indica el plazo límite de la tarea, o sea la fecha hasta la cual es posible agendarla como temporal.

El formato de la pantalla es similar al de tareas temporales.

Nota: En ambos procesos (tanto tareas temporales como pendientes) el campo Pendiente indica de qué tipo de tarea se trata. Por defecto es asignado a este campo el valor que corresponda al proceso seleccionado. Este valor puede ser modificado, pero en ese caso, no podrá verse nuevamente ésta tarea dentro del mismo proceso. Si modificáramos el tipo a una tarea temporal, entonces sólo podríamos verla nuevamente dentro del proceso de tareas pendientes o dentro del Menú de Tareas Pendientes. En el caso de las tareas pendientes, deberíamos entrar en el proceso de tareas temporales.

Administración de Tareas Reiterativas

Está orientado a procedimientos de carácter periódico. Se activa mediante la tecla r desde el Calendario. Los campos de estas tareas son:

Número Indica el número de la tarea.

Hora Indica la hora de inicio asignada a la tarea.

Duración Indica la duración de la tarea.

Alarma Indica cuantos minutos antes del comienzo de la tarea debe sonar la alarma de aviso. En caso de no ingresar valor alguno, la alarma no sera activada.

Título Título de la tarea.

Semanal Este campo indica si se han ingresado datos para el cálculo de la periodicidad semanal. Digitando “S”, se pasará a una subventana que permite ingresar los datos para el cálculo de esta periodicidad. Dicha subventana tiene dos campos: el primero indica qué número de semana dentro del mes se repite la tarea; mientras que el otro fija el número de día dentro de la semana seleccionada en el cual se efectúa. Hay que tener en cuenta que la primera semana de un mes es aquella que comienza en día Domingo, y que la última puede terminar, por lo tanto, en el mes siguiente.

Diario Este campo indica si se han ingresado datos para el cálculo de la periodicidad por día. Digitando “S” se dará entrada a una subventana que permite ingresar los datos para el cálculo de esta periodicidad. Contiene un único campo que indica el número de día en el cual se repite la tarea.

Orden/Día Este campo indica si se han ingresado datos para el cálculo de la periodicidad por orden/día. Ingresando “S”, se dará entrada a una subventana que permite especificar los datos para el cálculo de esta periodicidad.

Dicha subventana tiene dos campos: el primero indica el número de orden del día que será seleccionado; el segundo informa el número de día seleccionado en el cual se repite la tarea.

Descrip Similar a la de tareas temporales.

A continuación se muestra el formato de la pantalla:

Figura 10.5 - Tareas Reiterativas

Horario

El Horario permite ver en forma gráfica las tareas agendadas en un determinado día. La tecla TAB activa y desactiva el horario desde el calendario.

Cuando el horario es activado, es desplegado con una ventana contigua al calendario, mostrando el horario correspondiente a la fecha marcada en el calendario.

A continuación se muestra el formato del horario:

Figura 10.6 - Horario de tareas

Menú de Tareas Pendientes

Permite visualizar este tipo de tareas con su correspondiente fecha límite, e incluirla dentro de la fecha que esté seleccionada. Una vez dentro del Menú podremos movernos por cada una de las tareas pendientes y al digitar PROCESAR en una de ellas, se agendará automáticamente en la fecha que esté seleccionada; esta tarea pasará a ser temporal, y si estuviera activo el horario, éste se actualizará en forma inmediata.

Nota sobre Superposición de Tareas

Hay ciertas superposiciones que no son permitidas dentro de la administración de los distintos tipos de tareas. Dos tareas temporales no pueden ser asignadas en una misma fecha y hora, al igual que dos tareas pendientes no puede ser ingresadas con la misma fecha límite y hora de realización. Una tarea pendiente no puede ser asignada dentro del Menú de Tareas Pendientes, en una fecha y hora en la cual exista una tarea temporal.

Ayuda

La tecla de función AYUDA activa y desactiva la ventana de ayuda desde el calendario. En esta ventana están descriptos todos los comandos de este servicio.

Tratamientos de Feriados

Existe la posibilidad desde el calendario de consultar la descripción de un feriado. Posicionado en el calendario sobre un día feriado (el número de día estará resaltado) y digitando la tecla *f*, se abrirá una ventana con la descripción de este feriado.

El Correo entre Usuarios

El *imail* es un utilitario de IDEAFIX que permite enviar, recibir o guardar mensajes entre distintos usuarios.

El comando imail trabaja de dos formas distintas: permite tanto enviar mensajes a otros usuarios del sistema UNIX como leer los mensajes propios.

Una vez que se activa el servicio, se mostrarán los siguientes datos:

- nombre del usuario UNIX
- fecha y hora de envío
- número de mensaje en el buzón

- texto

Por ejemplo, un estado posible de la pantalla será el siguiente:

Figura 10.7 - Correo de usuarios

Para poder examinar los mensajes que hay en el buzón se digitan las opciones A y S o las teclas de paginación, desplegándose en pantalla los datos referidos a los mensajes correspondientes.

Al pie de la ventana, se detallan las operaciones habilitadas. Digitando la primera letra de la operación se realiza la función.

Responder y borrar un mensaje (R).

Copiar un mensaje a un archivo (C).

Limpiar la pantalla (L).

Tabla de Mensajes (T).

Grabar y fin (G).

Enviar correo (E).

Siguiente (S).

Anterior (A).

Borrar (B).

Fin (F).

El resto de esta sección, explicará en forma detallada cada una de ellas.

Enviando Mensajes

Para enviar un mensaje se debe digitar, en el campo control, la opción E.

Se solicitará entonces un nombre de usuario. Se podrá ingresar cualquier nombre de usuario.

El imail posee un archivo de ayuda para el usuario cuyo objetivo es asociar el nombre de un usuario o una descripción, con la secuencia de caracteres que el mail necesita para rutear el mensaje.

Dicho archivo es obligatorio y se denomina “.mlist”, el imail lo trata de leer del directorio indicado en la variable de ambiente HOME o de /usr/mail (en ese orden). El carácter de separación entre ambas descripciones es ':'.

Un ejemplo del archivo “.mlist” es:

Administracion:adm

Bollati:larry!alberto

Arleen Coscarelli:larry!arleen

Demostraciones:demosys

Root:root

En el momento de ingresar el usuario al que se quiere enviar el mensaje, se puede seleccionar el nombre deseado digitando la tecla META en el campo USUARIO. Al digitar dicha tecla, se abrirá una ventana en la que se mostrarán los posibles destinatarios de los mensajes.

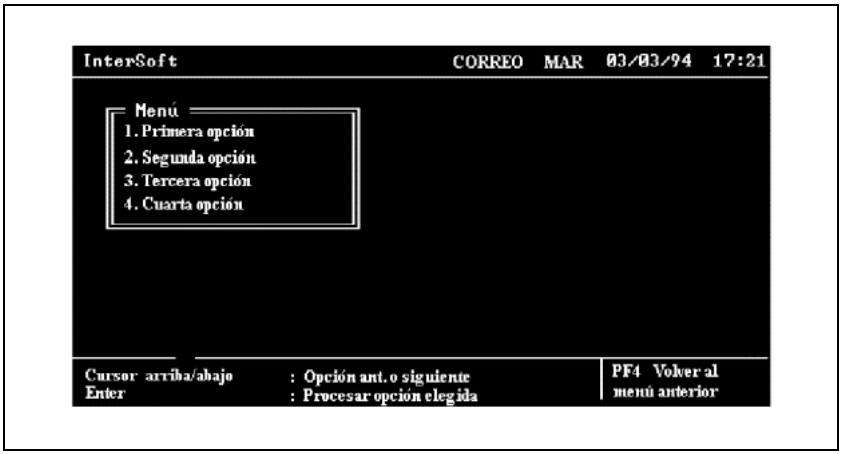


Figura 10.8 - Lista de Usuarios

Una vez que se ha ingresado el mensaje deseado, se debe presionar la tecla PROCESAR para enviar el mensaje.

En caso de desear enviar el mensaje a varios usuarios, sólo será necesario, digitar nuevamente la opción E, modificar el nombre del usuario al que se lo va a mandar y digitar nuevamente la tecla PROCESAR.

Si se desea enviar otro mensaje distinto al anterior, se deberá limpiar previamente la pantalla mediante la opción L, para evitar que se superponga el texto del mensaje anterior con el actual.

El imail utiliza la facilidad de “mailing” de UNIX para enviar mensajes y trabaja con el archivo dado por la variable de ambiente MAIL, que en general es: `/usr/mail/$LOGNAME`.

El imail no permite que el mismo usuario lo utilice simultáneamente desde dos terminales, esto lo resuelve bloqueando el archivo de correo.

Recibiendo Mensajes

Cuando se ingresa al imail, se muestra en pantalla el primer mensaje que haya en el buzón. En caso de no existir ninguno, una ventana indicará tal situación. Si se estuviera en una sesión con el imail y se recibiera un nuevo mensaje, el estado de la pantalla será:

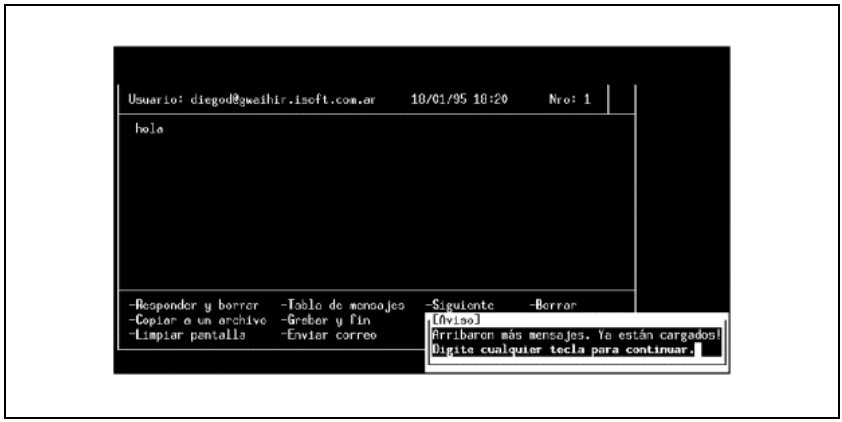


Figura 10.9 - Llegada de mensajes

Con la opción S, se mostrará el mensaje que sigue al corriente; la opción A despliega el mensaje anterior. Con la opción T se obtiene una ventana con la tabla de mensajes que hay en el buzón. Se puede seleccionar alguno de ellos con las teclas de cursor y desplegarlo en pantalla digitando la tecla INGRESAR.

Dicha tabla muestra:

- Número de mensaje.
- Remitente.
- Fecha de envío.

Sigue una pantalla ejemplo.

Figura 10.10 - Pantalla de mensajes

En caso de haber seleccionado, mediante las teclas de posicionamiento, el mensaje número dos, cuyo remitente es “raul”, se desplegará en la pantalla dicho mensaje:

Figura 10.11 - Seleccionando un mensaje

Otras Operaciones con Mensajes

Contestando Mensajes

Mediante la opción R se responde al usuario que envió el mensaje corriente; en este caso se procede de igual manera al envío de un mensaje, con la diferencia de que no permite modificar el nombre del usuario que figura en pantalla, ya que él debe ser el destinatario de la respuesta.

Como ejemplo se muestra la contestación al mensaje del último ejemplo:

Figura 10.12 - Respuesta a mensajes

En este caso se mostrará en la última posición del encabezamiento de la ventana la indicación DEL, que indica que además de estar siendo contestado, será borrado, en caso de digitar la opción G para terminar la sesión.

Borrando Mensajes

Para borrar un mensaje existe la opción B, que provocará un borrado lógico del mensaje.

El mensaje será borrado del archivo de mensajes del mail cuando se digite la tecla G (grabar).

Por supuesto, ya sea que se borró un mensaje con la opción B o con la opción R el estado de la pantalla en caso de digitar la opción T al reingresar será:

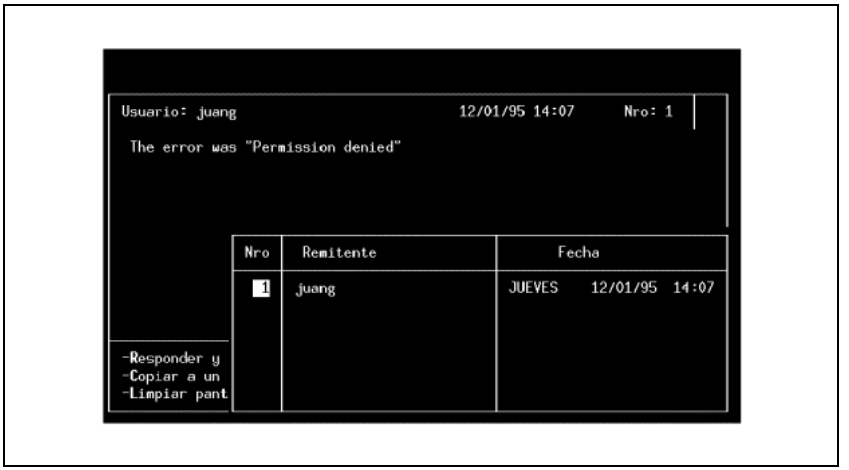


Figura 10.13 - Manejo de mensajes - Borrado

Copiando Mensajes

Mediante la opción C se copia el mensaje que figura en pantalla a un archivo, para ésto se abrirá una subventana para el ingreso del nombre del archivo, una vez hecho ésto, será necesario digitar la tecla INGRESAR, en el campo control de la subventana que se abrió.

Por ejemplo el estado de la pantalla en caso de haber digitado la opción podría ser el siguiente, despúes de haber ingresado el nombre del archivo sobre el que se desea el copiado:

Figura 10.14 - Copia de mensajes

En el archivo “mensaje” se encontrará la copia del mensaje.

Terminando una Sesión con *imail*

La operación F finaliza la sesión sin borrar en el archivo de mail los mensajes que han sido dados de baja, en forma lógica, mediante la operación B.

En caso de querer abandonar la sesión, puede también digitarse la tecla END teniendo el mismo efecto que la operación F.

En caso de querer que se incorporen las modificaciones y finalizar el uso del utilitario, se debe digitar la opción G.

El servicio “Posicionar Ventana” permite al usuario, el desplazamiento de las ventanas (windows) desplegadas en la terminal. Por lo general la apertura de una nueva ventana cubre los datos desplegados en las ventanas abiertas con anterioridad. Para poder visualizar un conjunto de ventanas simultáneamente sin que se solapen, se utiliza este servicio para el desplazamiento de las mismas.

Cuando se selecciona esta opción desde el menú de servicios se colocarán marcas intermitentes (blink) en las esquinas de la última ventana desplegada. Esto nos invita a desplazarnos por las distintas ventanas creadas en la terminal para seleccionar la que deseamos mover. Utilizando las teclas de movimiento de cursor, el usuario recorrerá las ventanas y una vez posicionado sobre la deseada digitará la tecla INGRESAR, lo que provocará la fijación de sus esquinas.

Habiendo seleccionado la ventana, se procederá al desplazamiento de la misma mediante las teclas de movimiento de cursor. Alcanzada la posición deseada se digita nuevamente la tecla INGRESAR y se finaliza la operación.

2

Usando IQL

Capítulo 11

Primeros Pasos

Introducción al SQL

El nombre de SQL (cuya pronunciación oficial es “ess-cue-ell”) es una abreviación para “Structured Query Language” (Lenguaje Estructurado de Consulta). El lenguaje SQL consiste en un grupo de facilidades para definir, acceder y también manejar base de datos relacionales.

Cuando decimos “SQL”, nos referimos al “SQL estándar”. Está es la última versión ISO del lenguaje.

La función principal del lenguaje SQL es soportar la definición, manipulación y control de datos en base de datos relacionales. Una *base de datos relacional* es simplemente una base de datos, la cual es percibida por el usuario como una colección de tablas, en donde una *tabla* es una *colección desordenada de filas*. Algunos consideran a una tabla como un *archivo*, donde las filas representan *registros* y las columnas *campos*; pero atendiendo al estándar, usaremos los términos **tabla**, **fila** y **columna**.

IQL y SQL

Siguiendo al estándar, IQL es un lenguaje no sensible a las mayúsculas. Esto quiere decir que usar mayúsculas cuando se tipea un comando, no tiene un significado especial para el intérprete (por ejemplo, es lo mismo tipear SeLECt o select).

Al escribir números, se usa únicamente el punto decimal (por ejemplo no como separador de miles). Pero cuando se muestren los números en pantalla, el formato actual, depende del valor

de la variable de ambiente LENGUAJE (que será LENGUAJE=spanish, en nuestro caso).

Se pueden usar dos tipos de comentarios:

• Línea simple: empezando con la secuencia // y extendiéndose hasta el final de la línea.

• Líneas Múltiples: Empezando con la secuencia /* hasta la secuencia */. Como lo sugiere su tipo, estos comentarios ignoran las características de la línea y pueden ser usado para encerrar varias líneas de texto.

En la terminología IQL un comando es considerado como una secuencia de expresiones separadas por punto y como. El server IQL acepta comandos como entrada dando filas de datos como salida. Es por esto que, en razón de disminuir el tráfico de red --cuando esta presente una conección de red--, es posible unir varias expresiones para ser procesadas en secuencia, como un comando simple, del lado del server.

A lo largo de este manual se escribirán las palabras claves en estilo itálico (ej., keyword) mientras que los nombres simbólicos que deberá ser reemplazados con nuevos nombres (como los nombres de tabla o campo) aparecerán en estilo normal. Todos ellos aparecerán en curier.

El RDBMS de IDEAFIX

El núcleo del manejo de Bases de Datos se conforma por el RDBMS (Relational Data Base Management System), o Sistema de Manejo de Bases de Datos Relacionales, que se encarga de acceder físicamente a la información. Si el sistema tiene instalado *Essentia*, el Servidor de Bases de Datos de InterSoft, el acceso a la información será mejorado por él, (por ejemplo, *Essentia* reemplazaría el RDBMS IDEAFIX). De todas formas, al nivel de usuario de IQL (del cual nos ocuparemos en esta Parte), no existirán diferencias entre un sistema con Essentia y otro sin (actualmente, la diferencia al nivel de programador, prácticamente tampoco existe).

Conectando al Server IQL

Iniciando una Sesión IQL

El primer requerimiento para iniciar una conección con el Server IQL es tener un cliente compilado con una librería especial. Primero, se debe inicializar la variables de ambiente SERVERS a “iqlsrv”. Segundo, se debe estar seguro que existe un archivo en ESSENTIA/servers bajo el nombre “iqlsrv.rsv” conteniendo la siguiente información:

```
NameSpaces Environ
Host <host>
```

la variable `<host>` indica donde reside el server IQL.

Si este archivo existe, y el valor de SERVERS ha sido inicializado, el server se cargará

automáticamente cuando el cliente es empezado (de hecho, hay otras variables que deben ser configuradas para que el server funcione apropiadamente, como ser, "`/etc/services`" y "`/etc/inetd.conf`"). Para una mayor información de como configurar estos archivos, por favor comunicarse con su administrador de sistemas.

Terminando una Sesión IQL

Una sesión con el server IQL se termina automáticamente al terminar de trabajar con la aplicación cliente. Dependiendo de la configuración de su sistema, un registro de las operaciones procesadas por el server podrán hallarse en el archivo "`/tmp/iqlsrv.log`".

Esquema Ejemplo

Los ejemplos que se encuentran en este manual están basados en el *Esquema Ejemplo*, el cual tiene *cuatros tablas*. Se puede considerar como el esquema de un departamento de personal, lo que sigue es la descripción de cada tabla.

EMP Esta tabla contiene la información de los empleados de la empresa tomada como ejemplo.

DPTO Aquí se almacenan los datos relativos a los departamentos que conforman la empresa.

CARGOS En esta tabla se detallan los posibles cargos que pueden desempear los empleados de la compañía.

FAMI Se registra la información referida a los familiares del personal, en particular de aquellas personas que se encuentran a su cargo.

Las siguientes figuras muestran la información contenida en dichas tablas:

EMP

nroleg	nombre	cargo	jefe	fnac	fingr	sueldo	comis	depno
1	Juan Carlos Suarez	1		10/10/1995	01/01/984			1
2	María Laura Laiso	2		03/04/1953	01/01/1984			1
3	Carlos Gabriel Berilini	3	1	8/06/1960	01/03/1984	4500.00		1
4	Miguel Angel Socos	3		16/12/1954	01/03/1984	4500.00		1
5	Alejandro Sergio	4	1	25/04/1958	01/05/1984	4250.00	2200.00	8

	Darta							
6	Marcela Edith Zarce	4	1	20/07/1962	01/07/1984	4250.00		4
7	Raul Guillermo Caico	4	1	12/11/1956	01/12/1984	4250.00		6
8	Jorge Pablo Felag	4	1	14/12/1960	01/02/1985	4250.00		3
9	Ricardo Marcelo Acol	5	6	21/10/1961	01/05/1985	4000.00		4
10	Nilda Patricia Sovervi	5	5	22/12/1961	01/07/1985	4000.00	900.00	8
11	Marcela Paula Diloropi	6	7	06/02/1959	01/09/1985	3250.00		6
12	César Pablo Taliga	6	9	14/11/1957	01/12/1985	3250.00		4
13	Laura Aída Leriblascoc	6	9	15/07/1960	01/12/1985	3250.00		4
14	Luís Marcos Lapadeo	7	1	03/05/1958	01/01/1986	2750.00		4
15	María Antonieta de las Flores	7	11	29/02/1952	01/01/1986	2750.00		6
16	Estella Maris Newher	7	1	31/03/1957	01/05/1986	2750.00		6
17	Ana María Notón	7	12	07/10/1962	01/09/1986	2750.00		4
18	Julio César Elías	7	13	22/11/1962	01/10/1986	2750.00		4
19	Eduardo Ricardo Estuardo	7	13	09/09/1959	01/12/1986	2750.00		4
20	Clara Nieves Farola	8	1	27/04/1962	01/02/1987	1500.00		1
21	Florio	8	5	16/06/1964	01/04/1987	1500.00	900.00	8

	Manuel Tenorio							
22	Nicolás Atilio Regaluh	9	7	15/05/1963	01/04/1987	850.00		6
23	María Aída Estía	9	9	08/10/1965	01/05/1987	850.00		4
24	Adriana Mariana Crana	7	8	19/08/1964	01/05/1987	2750.00		3
25	Roberto Diego Fla ez	7	8	24/05/1965	01/06/1987	2750.00		3
26	Alejandra Angeles Riveros	7	4	13/01/1964	13/01/1964	2750.00		5
27	Marisa Clarisa Mayo	10	1	12/12/1964	01/09/1987	2250.00		7
28	Fernando López Gabel	10	1	01/06/1965	01/02/1988	2250.00		7
29	Alfredo Mik Ladrón	9	26	22/05/1952	23/08/1987	2000.00		5

DEPTO

depno	nombre	ubic
1	Dirección	Buenos Aires
2	Gerencia	Bs.As./Rosario
3	Software de Base	Buenos Aires
4	Desarrollo	Rosario
5	Documentación	Rosario
6	Mantenimiento	Rosario
7	Administración	Buenos Aires
8	Ventas	Buenos Aires

CARGOS

cargo	descr

1	Presidente
2	Vicepresidente
3	Director
4	Gerente
5	Subgenrente
6	Líder de Proyecto
7	Analista Programador
8	Secretaria
9	Becario
10	Administrativo

FAM

nroleg	nrofam	tipo	nombre
4	1	1	Adriana Gómez
5	1	1	Maria Luisa Cuiralda
5	2	2	Federico Darta
5	3	2	Antonella Darta
6	1	1	Pablo Daniel Martinez
7	1	1	Cecilia Miranda
7	2	2	Santiago Caico
8	1	1	Silvia Hernández
10	1	1	Julio De Caro
12	1	1	Claudia Cristina Correa
12	2	2	Gustavo Daniel Taliga
16	1	2	Nicolás Newher
18	1	1	María Fernanda Luissi
19	1	1	María Emilia Santillán
19	2	2	María Laura Estuardo
19	3	2	María de las Nieves Estuardo
19	4	2	María Vanessa Estuardo
19	5	2	Juan Manuel Estuardo
23	1	2	Natalia Agostina Estía
25	1	1	Virginia Warburg
25	2	2	Vanina Alejandra Flaez

26	1	1	Osvaldo Martínez
26	2	2	Yanina Vanina Martínez
28	1	2	Esteban Gonzalo López Gabel

Relación entre Tablas

La posibilidad de relacionar tablas entre sí, permite organizar los datos en unidades separadas. Se puede entonces manejar la información de los departamentos aparte de la de personal, aunque siempre podemos unir los datos de ambas tablas, por ejemplo: conocer la ubicación (ubic, en depto) para un empleado (en emp). No es necesario que todas las tablas de una Base de Datos estén relacionadas.

A continuación se muestra el código que permite la creación de las tablas de nuestro ejemplo. Las sentencias utilizadas para su definición serán explicadas en capítulos posterior de este mismo manual.

```
table emp descr "Legajos del personal" (
nroleg num(4) descr "Número de Legajo"
primary key,
nombre char(30) descr "Nombre del Empleado",
cargo num(2) descr "Cargo que ocupa"
in cargos:descrip,
jefe num(4) descr "Jefe directo"
in emp:nombre,
fnacim date descr "Fecha de Nacimiento",
fingr date descr "Fecha de ingreso"
<= today,
sueldo num(12,2) descr "Sueldo Mensual",
comis num(12,2) descr "Comisión por Ventas",
depno num(2) descr "Departamento al que pertenece"
in depto:nombre
)
index nombre(nombre not null),
index ingreso(fingr, nroleg);
table cargos descr "Codificador de cargos" (
cargo num(2) descr "Cargo"
primary key,
descrip char(20) descr "Descripción del Cargo"
);
table depto descr "Departamentos de la empresa" (
depno num(2) descr "Número de Dpto."
primary key,
nombre char(20) descr "Nombre del Depto.",
ubic char(20) descr "Ubic. geográfica del depto."
);
table fam descr "Familiares de los empleados" (
nroleg num(4) descr "Nro. de legajo del empleado"
in emp:(nombre),
nrofam num(2) descr "Nro. de familiar",
tipo num(1) descr "Tipo de Parentesco"
```

```
in (1:"Esposo/a", 2:"Hijo/a"),
nombre char(30) descr "Nombre del Familiar"
)
primary key (nroleg, nrofam);
```

Capacidades del Sistema

La tabla a continuación ilustra sobre los límites del sistema.

Esquemas por usuario	Sin límite
Esquemas activos	16
Tablas por esquema	255
Tablas activas por esquema	255
Filas en una tabla,	2147483647
Campos por tabla,	255
Tamaño máximo de un registro en bytes	65535
Caracteres en un campo alfanumérico	65535
Dígitos en un campo numérico	15
Dígitos significativos en un campo numérico	15
Rango de valores en un campo fecha	16/04/1894 al 16/09/2073
Rango de valores en un campo hora	00:00:00 to 23:59:58[a]
Máxima dimensión de un vector	65535
Valores en un campo in	65535
Indices por tabla	255
Campos por índice	255
Longitud de un campo alfanumérico en índice	255
Longitud máxima de nombre de Esquema	10
Longitud máxima de nombre de Tabla	10
Longitud máxima de nombre de Campo	65535

[a] El último horario para un día es 23:59:58, esto se debe a que IDEAFIX tiene una precisión de 2 segundos para los datos del tipo Time.

Capítulo 12

Operando con IQL

Consultando Tablas

El comando *use*

Para el manejo de tablas con IQL es necesario especificarle al sistema cuál es el esquema que los soporta; o, en otros términos, es necesario activarlos. Esto se puede realizar mediante el comando *use*:

```
use schema1, schema 2, ..., schema n;
```

En él se indican los esquemas a utilizar al activarse IQL. La lista de los esquemas se debe separar por comas, y finalizada por punto y coma. Estas dos reglas se aplicarán a todos los comandos de IQL:

• Cuando es válido especificar varios Items del mismo tipo (como ser campos, esquemas y tablas) la lista que los incluya es delimitada con dos palabras clave, o por una palabra clave y el operador de "fin de sentencia", como en este caso.

• Los espacios en blanco no se cuentan (solo es necesario uno, como separador). En nuestro ejemplo los espacios en blanco deben estar entre el comando *use* y el primer esquema.

El primer esquema mencionado será el *esquema corriente*.

El comando *show*

Desplegando esquemas activos

En la sección anterior vimos como activar esquemas con la sentencia use. También mencionamos que en la lista de esquemas hay uno que es llamado corriente. El comando show es usado para desplegar que esquemas están activos, y cuál es el corriente. Su sintaxis es

simple:

```
show;
```

y su salida es la siguiente.

Nombre del esquema	Descripción	Propietario	Permisos
schema 1	descripción esquema1	willy	ALTER
schema 2	descripción esquema2	raul	ALTER
...	...	...	...
schema N	descripción esquemaN	paul	ALTER

Las últimas dos columnas se refieren al propietario del esquema y a los permisos del usuario que ejecuta la sentencia en ese esquema.

Desplegando Tablas y Campos

En caso de no recordar con certeza los nombres de esquemas, tablas y/o campos, el comando *show* , en sus distintas variantes, nos permite averiguarlos. Existen diversas maneras de invocar el comando *show*, según se detalla a continuación:

• sin parámetros se obtiene la lista de esquemas activos (como se especificó anteriormente)

• nombre del esquema se despliega una lista con las tablas que componen ese esquema

• nombre del esquema y tabla se obtine una lista con los campos de la tabla y sus características

Nombre de la Tabla	Descripción	Perm.
emp	Legajos del personal	idus
cargos	Codificador de cargos	idus
depto	Departamentos de la empresa	idus
fam	Familias de los empleados	idus

Nombre de Campo	Tipo	Descripción
nroleg	num(4)	Número de Legajo
nombre	char(30)	Nombre del Empleado
cargo	num(2)	Cargo que ocupa
jefe	num(4)	Jefe directo
fnacim	date	Fecha de Nacimiento

fingr	date	Fecha de Ingreso
sueldo	num(12,2)	Sueldo Mensual
comis	num(12,2)	Comisión por Ventas
depno	num(2)	Departamento al que pertenece

El comando *select*

La operación de mayor uso en IQL es la selección de uno o más campos de uno o más tablas pertenecientes a un esquema dado. La sentencia *select* agiliza este tipo de operaciones. Su sintaxis es:

```
select field1, field2, ..., fieldN
from table;
```

Las sentencias pueden ser escritas en una o más líneas en formato libre. Esto no solo es válido para el selct, sino para los comandos de todos los lenguajes. Como ya se indicó el fin de una sentencia esta dada por el punto y coma.

Especificando el nombre de los campos en la cláusula select es posible ver una tabla.

```
select empno, name, charge
from emp;
```

En este caso, se le pidió a IQL que muestre los campos empno, name y cargo de la tabla emp. La salida de esta consulta es:

nroleg	nombre	cargo
1	Juan Carlos Suarez	1
2	María Laura Laiso	2
3	Carlos Gabriel Berilini	3
4	Miguel Angel Socos	3
5	Alejandro Sergio Darta	4
6	Marcela Edith Zarce	4
7	Raul Guillermo Caico	4
8	Jorge Pablo Felag	4
9	Ricardo Marcelo Acol	5
10	Nilda Patricia Sovervi	5
11	Marcela Paula Diloropi	6
12	César Pablo Taliga	6
13	Laura Aída Leriblascoc	6
14	Luís Marcos Lapadeo	7
15	María Antonieta de las Flore	7

16	Estella Maris Newher	7
17	Ana María Notón	7
18	Julio César Elías	7
19	Eduardo Ricardo Estuardo	7
20	Clara Nieves Farola	8
21	Florio Manuel Tenorio	8
22	Nicolás Atilio Regaluh	9
23	María Aída Estía	9
24	Adriana Mariana Crana	7
25	Roberto Diego Fla ez	7
26	Alejandra Angeles Riveros	7
27	Marisa Clarisa Mayo	10
28	Fernando López Gabel	10
29	Alfredo Mik Ladrón	9

Consultando Campos Vectorizados

Al seleccionar un campo vectorizado de la forma vista hasta ahora, sólo se obtiene el valor de la primer ocurrencia. Si se quisiera recuperar el resto de los valores es necesario subindicar el campo (índice). Para ejemplificar estos casos supongamos un campo definido de la siguiente forma:

```
telef[3] char(10) descr 'Teléfonos'
mask '(3x)2nN-4N'
```

Al ejecutar el siguiente comando:

```
select telef
from tabla;
```

se obtendrá sólo el primer valor cargado. Para obtener todos los valores ejecutaremos:

```
selecta telef[0], telef[1], telef[2]
from tabla;
```

Como se podrá observar en el ejemplo, los elementos de un vector se comienzan a numerar a partir de cero.

Seleccionando Todas las Columnas de una Tabla

Se ha visto que el modo de referenciar las columnas en la tabla de un esquema es refiriendose explícitamente a ellas. Si en una consulta se necesita obtener los valores de todos los campos en una o más tablas, será dificil desplegar los nombres de todas las tablas. Para esto, IQL

cuenta con el operador "*". En su forma más simple, le indicará a IQL que obtenga todos los campos de la mencionada tabla, por ejemplo:

```
select * from dept;
```

De este modo se seleccionarán todos los campos de la tabla *dept*.

depno	nombre	ubic
1	Dirección	Buenos Aires
2	Gerencia	Bs. As. / Rosario
3	Software de Base	Buenos Aires
4	Desarrollo	Rosario
5	Documentación	Rosario
6	Mantenimiento	Rosario
7	Administración	Buenos Aires
8	Ventas	Buenos Aires

Una consulta puede ser mejorada especificando los campos de una tabla en cualquier orden, no sólo en el orden en que están definidas. En la ventana de salida aparecerán en el orden especificado en la consulta.

```
select fingr, cargo, nroleg, nombre
from emp;
```

Desplegará los campos "Día de ingreso" y "Cargo" antes (ej., a la izquierda) de los campos Número de Empleado y Nombre.

ÀY, qué pasa cuando queremos incluir información relacionada con otras tablas?. Existen dos modos de responder esto, que veremos más tarde. Uno es el de usar joins entre tablas, el otro es definir nuevas columnas que son calculadas por medio de expresiones.

Encabezados de Columnas

Muchas veces, cuando se define una tabla, se especifican los campos de la misma usando nombres mnemotécnicos (abreviaciones), los que para una visión final de la consulta dificultan la interpretación de la salida. Como medio de subsanar este inconveniente, IQL permite especificar un título para cada una de las columnas de una consulta. Si se lo incluye, deberá aparecer después del nombre de columna, delimitado entre comillas simples o dobles.

```
select
nroleg "Número de \nEmpleado",
nombre 'Apellido y \nNombre,
cargo "Cargo que \nocupa"
from emp;
```

En este caso se agregaron títulos a todas las columnas. Se observa que aparece la secuencia de

caracteres “\n”. Esta se utiliza para que el título abarque dos líneas diferentes.

Los títulos de las columnas aparecen siempre automáticamente centrados con respeto al área abarcada por el contenido del campo. El ejemplo propuesto deberá mostrar los títulos de esta forma:

Número de Empleado	Apellido y Nombre	Cargo que ocupa
1	Juan Carlos Suarez	1
2	María Laura Laiso	2
3	Carlos Gabriel Berilini	3
4	Miguel Angel Socos	3
5	Alejandro Sergio Darta	4
6	Marcela Edith Zarce	4
7	Raul Guillermo Caico	4
8	Jorge Pablo Felag	4
9	Ricardo Marcelo Acol	5
10	Nilda Patricia Sovervi	5
11	Marcela Paula Diloropi	6
12	César Pablo Taliga	6
13	Laura Aída Leriblascoc	6
14	Luís Marcos Lapadeo	7
15	María Antonieta de las Flores	7
16	Estella Maris Newher	7
17	Ana María Notón	7
18	Julio César Elías	7
19	Eduardo Ricardo Estuardo	7
20	Clara Nieves Farola	8
21	Florio Manuel Tenorio	8
22	Nicolás Atilio Regaluh	9
23	María Aída Estía	9
24	Adriana Mariana Crana	7
25	Roberto Diego Fla ez	7
26	Alejandra Angeles Riveros	7
27	Marisa Clarisa Mayo	10
28	Fernando López Gabel	10
29	Alfredo Mik Ladrón	9

En IQL, como en la mayoría de los lenguajes de computadoras, una expresión aritmética es

una fórmula simbólica que deberá ser resuelta en un valor. Los símbolos usados para las operaciones básicas son: /, *, +, -, y los paréntesis permiten agrupar operaciones rompiendo el orden natural de evaluación.

Es posible agregar columnas a una consulta con una expresión como filtro, como se muestra en el siguiente ejemplo.

```
select nroleg, nombre, comis * 100 / (sueldo + comis)
from emp;
```

Esta consulta mostrará la siguiente lista, representando el porcentaje de la comisión del empleado comparada con su sueldo en la tercer columna.

Número de Empleado	Apellido y Nombre	Porcentaje
1	Juan Carlos Suarez	
2	María Laura Laiso	
3	Carlos Gabriel Berilini	
4	Miguel Angel Socos	
5	Alejandro Sergio Darta	34.11
6	Marcela Edith Zarce	
7	Raul Guillermo Caico	
8	Jorge Pablo Felag	
9	Ricardo Marcelo Acol	
10	Nilda Patricia Sovervi	18.37
11	Marcela Paula Diloropi	
12	César Pablo Taliga	
13	Laura Aída Leriblascoc	
14	Luís Marcos Lapadeo	
15	María Antonieta de las Flores	
16	Estella Maris Newher	
17	Ana María Notón	
18	Julio César Elías	
19	Eduardo Ricardo Estuardo	
20	Clara Nieves Farola	
21	Florio Manuel Tenorio	37.50
22	Nicolás Atilio Regaluh	
23	María Aída Estía	
24	Adriana Mariana Crana	
25	Roberto Diego Fla ez	
26	Alejandra Angeles Riveros	

27	Marisa Clarisa Mayo	
28	Fernando López Gabel	
29	Alfredo Mik Ladrón	

Seleccionando Filas de una Tabla

La Cláusula *where*

Hasta ahora sólo se ha visto una forma de consulta: la búsqueda de registros en la cual no se imponía condición alguna. Por lo tanto, se obtenían todas las filas de la tabla referenciada en la consulta.

Se pueden establecer condiciones de búsqueda, de modo de limitar la misma a registros de las tablas que cumplan con determinadas características. Esto se realiza mediante la cláusula *where*. La sintaxis de la misma es la siguiente:

```
select campo1, campo2, ..., campoN
from tablas
where condición_de_búsqueda;
```

Se puede referenciar cualquiera de las tablas de los esquemas activos. Las tablas del esquema corriente se referencian solamente por su nombre. Cualquier otra tabla debe invocarse explícitamente con el nombre del esquema y el nombre de la tabla.

Especificando Condiciones en la Cláusula *where*

Los operadores de comparación que se utilizan en la cláusula *where* son:

Operador	Significado
=	Igual a
==	Igual a
!=	Distinto de
<>	Distinto de
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
>< ... and ...	Operador *between*
between ... and ...	Operador *between*
like	Patrones de cadenas
in	Lista de valores

in (subquery)	Condición de Subquery
exist (subquery)	Verifica una tabla específica
comp. op. [any|some|all]	Operador de comparación con la condición all-or-any

Los operadores lógicos que complementan las comparaciones son:

Operador	**Significado**
not, !	Negación
and, &	Operador and
or, |	Operador or

Para que una condición se tenga en cuenta debe ser colocada luego del comando *where*, como se dijo anteriormente.

La forma más natural de imponer una condición es cuando el operador de comparación está entre el nombre de una columna y el valor de una constante. Por ejemplo:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg>5;
```

selecciona todos los campos de la tabla emp en donde el número de empleado es mayor a 5. Como se ve en la siguiente tabla:

nroleg	**nombre**	**jefe**	**sueldo**	**depno**
1	Juan Suarez			1
2	María Laiso			1
3	Carlos Berilini		4500.00	1
4	Miguel Socos		4500.00	1
5	Alejandro Darta	1	4250.00	8
6	Marcela Zarce	1	4250.00	4
7	Raul Caico	1	4250.00	6
8	Jorge Felag	1	4250.00	3
9	Ricardo Acol	6	4000.00	4
10	Nilda Sovervi	5	4000.00	8
11	Marcela Diloropi	7	3250.00	6
12	César Taliga	9	3250.00	4
13	Laura Leriblascoc	9	3250.00	4
14	Luís Lapadeo	1	2750.00	4
15	María de las	11	2750.00	6

	Flores			
16	Estella Newher	1	2750.00	6
17	Ana Notón	12	2750.00	4
18	Julio Elías	13	2750.00	4
19	Eduardo Estuardo	13	2750.00	4
20	Clara Farola	1	1500.00	1
21	Florio Tenorio	5	1500.00	8
22	Nicolás Regaluh	7	850.00	6
23	María Estía	9	850.00	4
24	Adriana Crana	8	2750.00	3
25	Roberto Fla ez	8	2750.00	3
26	Alejandra Riveros	4	2750.00	5
27	Marisa Mayo	1	2250.00	7
28	Fernando López Gabel	1	2250.00	7
29	Alfredo Ladrón	26	2000.00	5

otro modo de establecer la comparación es reemplazar la constante por otra columna, o por cualquier otra expresión. En este último caso:

```
select nroleg, nombre, jefe, salario, depno
from emp
where comis>sueldo/2;
```

selecciona a todos los empleados cuyas comisiones son mayores a la mitad de su salario.

nroleg	nombre	jefe	sueldo	depno
5	Alejandro Darta	1	4250.00	8
21	Florio Tenorio	5	1500.00	8

Combinación de Condiciones

Las condiciones de búsqueda vistas dentro de la cláusula *where* evalúan las filas según criterios de verdad, que pueden resultar ciertos o falsos. Los casos que se toman son aquellos en donde la condición se cumple; es decir, el criterio verifica como verdadero. Se pueden formar expresiones compuestas combinando expresiones simples. La forma de hacerlo es mediante los operadores de conjunción o de disyunción. Por compleja que sea una expresión lógica, sólo puede en definitiva arrojar un resultado final binario: “1' (verdadero) ó “’ (falso).

El Operador AND

Si se requiere que se cumpla más de una condición en una consulta, la forma de hacerlo es a través del operador *and* . En este caso se evalúan cada una de las condiciones, debiendo ser verdaderas todas ellas para que la expresión sea verdadera. Por ejemplo:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg <= 15 and jefe = 1;
```

Seleccionará todos los empleados cuyo número sea menor o igual que quince y tengan como jefe al empleado número uno. Esto se denomina conjunción porque las condiciones deben darse conjuntamente. También se la llama producto lógico, ya que todos los factores o términos deben evaluar a 1 para que su producto sea 1: basta la presencia de un solo factor 0 para que el producto se anule (= falso).

nroleg	nombre	jefe	sueldo	depno
5	Alejandro Darta	1	4250.00	8
6	Marcela Zarce	1	4250.00	4
7	Raul Caico	1	4250.00	6
8	Jorge Felag	1	4250.00	3

El Operador OR

Si se consideran varias condiciones de búsqueda pero basta que sea verdadera solamente alguna de ellas, se utiliza el operador or.

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg<3 or nroleg>8;
```

En este caso se obtendrán todos los empleados cuyo número de empleado sea menor que tres o mayor que ocho. Tenemos ahora una disyunción, dado que requerimos que se cumpla una u otra de las posibildades. Esto se denomina suma lógica, ya que ahora la situación es inversa: sólo puede existir un resultado falso si todos los términos son cero. La presencia de más de un elemento igual a uno no produce problemas, ya que las normas del Algebra de Boole hacen que 1 + 1 = 1, pues no tienen sentido los valores mayores que la unidad. El sentido es "Verdadero más verdadero sigue siendo verdadero" (no doblemente verdadero).

nroleg	nombre	jefe	sueldo	depno
1	Juan Suarez			1
2	María Laiso			1
9	Ricardo Acol	6	4000.00	4
10	Nilda Sovervi	5	4000.00	8
11	Marcela Diloropi	7	3250.00	6

12	César Taliga	9	3250.00	4
13	Laura Leriblascoc	9	3250.00	4
14	Luís Lapadeo	1	2750.00	4
15	María de las Flores	11	2750.00	6
16	Estella Newher	1	2750.00	6
17	Ana Notón	12	2750.00	4
18	Julio Elías	13	2750.00	4
19	Eduardo Estuardo	13	2750.00	4
20	Clara Farola	1	1500.00	1
21	Florio Tenorio	5	1500.00	8
22	Nicolás Regaluh	7	850.00	6
23	María Estía	9	850.00	4
24	Adriana Crana	8	2750.00	3
25	Roberto Fla ez	8	2750.00	3
26	Alejandra Riveros	4	2750.00	5
27	Marisa Mayo	1	2250.00	7
28	Fernando López Gabel	1	2250.00	7
29	Alfredo Ladrón	26	2000.00	5

El Operador NOT

Cuando se requiere listar a aquellas filas que no cumplan una cierta condición, se utiliza el operador *not* . Así por ejemplo:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where not nroleg<5;
```

Esta consulta es equivalente a pedir aquellos cuyo número de legajo es mayor o igual que 5. Desde el punto de vista lógico el not equivale a una inversión de valores a los fines de evaluar el resultado final: el uno se convierte en cero y viceversa.

nroleg	nombre	jefe	sueldo	depno
5	Alejandro Darta	1	4250.00	8
6	Marcela Zarce	1	4250.00	4
7	Raul Caico	1	4250.00	6
8	Jorge Felag	1	4250.00	3
9	Ricardo Acol	6	4000.00	4
10	Nilda Sovervi	5	4000.00	8

11	Marcela Diloropi	7	3250.00	6
12	César Taliga	9	3250.00	4
13	Laura Leriblascoc	9	3250.00	4
14	Luís Lapadeo	1	2750.00	4
15	María de las Flores	11	2750.00	6
16	Estella Newher	1	2750.00	6
17	Ana Notón	12	2750.00	4
18	Julio Elías	13	2750.00	4
19	Eduardo Estuardo	13	2750.00	4
20	Clara Farola	1	1500.00	1
21	Florio Tenorio	5	1500.00	8
22	Nicolás Regaluh	7	850.00	6
23	María Estía	9	850.00	4
24	Adriana Crana	8	2750.00	3
25	Roberto Fla ez	8	2750.00	3
26	Alejandra Riveros	4	2750.00	5
27	Marisa Mayo	1	2250.00	7
28	Fernando López Gabel	1	2250.00	7
29	Alfredo Ladrón	26	2000.00	5

Combinación Operadores

Los operadores AND y OR vistos pueden combinarse en una única expresión. Cuando ello ocurre, la expresión se evalúa realizando primero las operaciones AND y luego las OR.

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg<=8 and jefe=1 or sueldo>2000;
```

En este caso se seleccionarán todos los empleados que cumplan alguna de las condiciones (ej., legajo igual o inferior a ocho, cuyo jefe sea el legajo Nro. 1 con sueldo superior a dos mil.)

nroleg	nombre	jefe	sueldo	depno
3	Carlos Berilini		4500.00	1
4	Miguel Socos		4500.00	1
5	Alejandro Darta	1	4250.00	8
6	Marcela Zarce	1	4250.00	4
7	Raul Caico	1	4250.00	6
8	Jorge Felag	1	4250.00	3

9	Ricardo Acol	6	4000.00	4
10	Nilda Sovervi	5	4000.00	8
11	Marcela Diloropi	7	3250.00	6
12	César Taliga	9	3250.00	4
13	Laura Leriblascoc	9	3250.00	4
14	Luís Lapadeo	1	2750.00	4
15	María de las Flores	11	2750.00	6
16	Estella Newher	1	2750.00	6
17	Ana Notón	12	2750.00	4
18	Julio Elías	13	2750.00	4
19	Eduardo Estuardo	13	2750.00	4
24	Adriana Crana	8	2750.00	3
25	Roberto Fla ez	8	2750.00	3
26	Alejandra Riveros	4	2750.00	5
27	Marisa Mayo	1	2250.00	7
28	Fernando López Gabel	1	2250.00	7
29	Alfredo Ladrón	26	2000.00	5

Si se necesita alterar el orden de precedencia, se puede encerrar la expresión entre paréntesis. Veamos qué ocurre cuando especificamos

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg<=5 and (jefe=1 or sueldo>2000);
```

Para interpretar esta expresión, consideremos en primer término a la parte entre paréntesis como una condición que analizaremos luego, y resultará claro entonces que se trata de un *and* que exige por una parte legajos no superiores al cinco, y otro requisito adicional que consiste en que el jefe sea el legajo 1 o bien el sueldo superior a dos mil.

nroleg	nombre	jefe	sueldo	depno
3	Carlos Berilini		4500.00	1
4	Miguel Socos		4500.00	1
5	Alejandro Darta	1	4250.00	8

Seleccionando Filas dentro de Rango

Se vio que una manera de especificar un rango en una cláusula *where* es la de definir la

condición mediante el operador de conjunción.

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg>=3 and nroleg<=8;
```

En este caso se seleccionarán todos los empleados cuyo número de empleado esté comprendido entre tres y ocho.

nroleg	nombre	jefe	sueldo	depno
3	Carlos Berilini		4500.00	1
4	Miguel Socos		4500.00	1
5	Alejandro Darta	1	4250.00	8
6	Marcela Zarce	1	4250.00	4
7	Raul Caico	1	4250.00	6
8	Jorge Felag	1	4250.00	3

Otra manera de estipular la misma condición es a través de la cláusula *between*. La misma se especifica:

```
between límite_inferior and límite_superior
```

Por lo tanto, el ejemplo precedente puede escribirse como:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg between 3 and 8;
```

Semánticamente es equivalente al ejemplo anterior.

nroleg	nombre	jefe	sueldo	depno
3	Carlos Berilini		4500.00	1
4	Miguel Socos		4500.00	1
5	Alejandro Darta	1	4250.00	8
6	Marcela Zarce	1	4250.00	4
7	Raul Caico	1	4250.00	6
8	Jorge Felag	1	4250.00	3

Nótese que los límites se incluyen en la definición del criterio de verdad; es decir, los operadores implícitos son <= y >= , no meramente < y >.

El Operador *in*

Cuando la condición que se desea imponer a una consulta requiere que un campo asuma alguno de entre una serie de valores prefijados, se utiliza el operador *in*. Este permite indicar una lista de valores, y se verificará si su operando es alguno de ellos. Los valores deben ir encerrados entre paréntesis y separados por comas.

Veamos por ejemplo cómo listar los empleados de los departamentos Dirección, Gerencia y Documentación:

```
select nroleg, nombre, depno
from emp
where depno in (1,2,5);
```

Lógicamente esta consulta podría haberse planteado usando el operador de igualdad en combinación con el *or* , pero de esta forma es mucho más sencillo de expresar.

nroleg	nombre	depno
1	Juan Suarez	1
2	María Laiso	1
3	Carlos Berilini	1
4	Miguel Socos	1
20	Clara Farola	1
26	Alejandra Riveros	5
29	Alfredo Ladrón	5

Este operador se aplica a cadena de carácteres, y permite encontrar aquellas que respondan a un cierto patrón. Los patrones están formados por carácteres especiales: “*” (que significa cualquier longitud de cadena) y “?” (que significa solo un carácter).

Si se quiere encontrar los nombres de todos los empleados cuyos nombres empiezan con “A”, se debe ejecutar:

```
select nroleg, nombre, depno
from emp
where name like “A*”;
```

La expresión “A*” indica cualquier cadena que empiece con “A”.

nroleg	nombre	depno
24	Adriana Crana	3
26	Alejandra Riveros	5
5	Alejandro Darta	8
29	Alfredo Ladrón	5
17	Ana Notón	4

Si lo que se quiere es encontrar a alguien que se llame María como primer o segundo nombre:

```
select nroleg, nombre, depno
from emp
where nombre like “*María* “;
```

su salida será:

nroleg	nombre	depno
23	María Aída Estía	4
15	María Antonieta de las Flores	6
17	Ana María Notón	4
2	María Laura Laiso	1

Despliegue Filas Ordenadas - Cláusula *order by*

Cuando se realiza una consulta, no se garantiza en qué orden aparecerán desplegadas las filas de la tabla resultante. Para establecer un ordenamiento de acuerdo a cualesquiera de las columnas de la consulta, se debe especificar la cláusula *order by* . La sintaxis de la misma es como sigue:

```
order by ordenamiento
```

donde ordenamiento puede definirse como:

```
expresión [ asc | desc ], ...expresión [ ascending |
descending ], ...
```

Por ejemplo, si se quiere ordenar una tabla resultante de una consulta en forma ascendente por cargo, se debe hacer:

```
select nroleg, nombre, cargo, jefe, fingr, depno
from emp order by cargo asc;
```

Como se asume ordenamiento ascendente por defecto, se podría haber obviado la cláusula *asc*.

Se pueden establecer más de una condición de ordenamiento:

```
select nroleg, nombre, cargo, jefe, fingr, depno
from emp
order by cargo desc , depno desc, fingr desc ;
```

Cuya salida es:

nroleg	nombre	cargo	jefe	fingr	depno
27	Fernando	10	1	01/02/87	7

	López Gabel				
28	Fernando López Gabel	10	1	01/09/88	7
22	Nicolás Atilio Regaluh	9	7	01/04/87	6
29	Alfredo Mik Ladrón	9	26	23/08/87	5
23	María Aída Estía	9	9	01/05/87	4
21	Florio Manuel Tenorio	8	5	01/04/87	8
20	Clara Nieves Farola	8	1	01/02/87	1
15	María Antonieta de las Flores	7	11	01/01/86	6
16	Estella Maris Newher	7	1	01/05/86	6
26	Alejandra Angeles Riveros	7	4	01/06/87	5
14	Luís Marcos Lapadeo	7	1	01/01/86	4
17	Ana María Notón	7	12	01/09/86	4
18	Julio César Elías	7	13	01/10/86	4
19	Eduardo Ricardo Estuardo	7	13	01/12/86	4
24	Adriana Mariana Crana	7	8	01/05/87	3
25	Roberto Diego Fla ez	7	8	01/06/87	3
11	Marcela Paula Diloropi	6	7	01/09/85	6
12	César Pablo Taliga	6	9	01/12/85	4
13	Laura Aída Leriblascoc	6	9	01/12/85	4
10	Nilda Patricia Sovervi	5	5	01/07/85	8
9	Ricardo Marcelo Acol	5	6	01/05/85	4
5	Alejandro Sergio Darta	4	1	01/05/84	8

7	Raul Guillermo Caico	4	1	01/12/84	6
6	Marcela Edith Zarce	4	1	01/07/84	4
8	Jorge Pablo Felag	4	1	01/02/85	3
3	Carlos Gabriel Berilini	3	1	01/03/84	1
4	Miguel Angel Socos	3		01/03/84	1
2	María Laura Laiso	2		01/01/84	1
1	Juan Carlos Suarez	1		01/01/84	1

JOIN de Tablas

Una característica distintiva de las tablas relacionales es que son susceptibles de unirse o juntarse (tal el significado de la expresión inglesa “join”) a fin de brindar un conjunto de información más completo. Esto implica la posibiildad de incluir en un informe o reporte, campos pertenecientes a distintas tablas. En IDEAFIX, el único requisito es que todas las tablas de las cuales se intenta extraer información se hallen *activas*, es decir, pertenezcan a esquemas activos. Asimismo, para que la operación tenga sentido, es preciso que exista al menos una columna en común entre las tablas que se relacionan. Supongamos una cláusula *where* con una condición que involucra columnas de tablas diferentes, pero con igual significado lógico, tal como:

```
select emp.nroleg, emp.nombre, fam.tipo, fam.nombre
from emp, fam
where emp.nroleg=fam.nroleg;
```

La salida será:

nroleg	nombre	tipo	nombre
4	Miguel Angel Socos	1	Adriana Gómez
5	Alejandro Sergio Darta	1	Maria Luisa Cuiralda
5	Alejandro Sergio Darta	2	Federico Darta
5	Alejandro Sergio Darta	2	Antonella Darta
6	Marcela Edith Zarce	1	Pablo Daniel Martinez
7	Raul Guillermo Caico	1	Cecilia Miranda

7	Raul Guillermo Caico	2	Santiago Caico
8	Jorge Pablo Felag	1	Silvia Hernández
10	Nilda Patricia Sovervi	1	Julio De Caro
12	César Pablo Taliga	1	Claudia Cristina Correa
12	César Pablo Taliga	2	Gustavo Daniel Taliga
16	Estella Maris Newher	2	Nicolás Newher
18	Julio César Elías	1	María Fernanda Luissi
19	Eduardo Ricardo Estuardo	1	María Emilia Santillán
19	Eduardo Ricardo Estuardo	2	María Laura Estuardo
19	Eduardo Ricardo Estuardo	2	María de las Nieves Estuardo
19	Eduardo Ricardo Estuardo	2	María Vanessa Estuardo
19	Eduardo Ricardo Estuardo	2	Juan Manuel Estuardo
23	María Aída Estía	2	Natalia Agostina Estía
25	Roberto Diego Fla ez	1	Virginia Warburg
25	Roberto Diego Fla ez	2	Vanina Alejandra Fla ez
26	Alejandra Angeles Riveros	1	Osvaldo Martínez
26	Alejandra Angeles Riveros	2	Yanina Vanina Martínez
28	Fernando López Gabel	2	Esteban Gonzalo López Gabel

Esto es lo que permite efectuar lo que se denomina un JOIN explícito[1] porque se lo indica en la consulta. Es de notar que en no existe en IDEAFIX una sentencia JOIN, puesto que la acción de juntar las tablas se ejecuta automáticamente cuando el sentido del comando así lo requiere.

Lo que estamos pidiendo al SQL es una lista de los empleados (filas de la tabla emp) que tengan familiares a cargo (filas de la tabla fam), y queremos que el listado contenga el número de legajo del empleado (que es el elemento de unión entre ambas tablas), el nombre

[1] En IDEAFIX, un JOIN *implícito* es el que está definido en la tabla. Para más información ver el Capítulo 17, Sentencias de Definición de Datos.

del empleado, el código que individualiza el tipo de familiar (esposa, hijo, etc.) y por último el nombre del famliar. Debido a que existen campos con el mismo nombre en ambas tablas, se debe prefijar el nombre de tabla al nombre de campo para que la referencia no sea ambigua. El único campo en el que hubiera podido omitirse la calificación es el tipo de famliar, ya que no existe un campo "tipo" en la tabla emp. Véase que por una parte van a existir empleados que no figuran en el reporte, por no tener familiares a cargo, mientras que otros pueden figurar varias veces, tantas como integrantes de la familia se hallen registrados en fam.

Pongamos ahora un caso un poquito más complejo: queremos listar únicamente los cónyuges de los empleados. Para ello debemos introducir una condición adicional en la última cláusula:

```
where emp.nroleg=fam.nroleg and tipo=1;
```

Resumiendo, la cláusula *where* se utiliza tanto para especificar la manera de relacionar las filas de ambas tablas por medio de columnas correspondientes, como para establecer requisitos adicionales.

Referencia a Tablas

El primer ejemplo ha sido sencillo, particularmente en cuanto involucraba el uso de solamente dos tablas. Es posible referenciar varias de ellas y establecer relacionamientos más complejos, incluyendo la unión de una tabla consigo misma, y los denominados "non-equi joins", que determinan uniones por medio de operadores que no implican igualdad. Todo ello se verá más adelante en este mismo capítulo, pero para exponer esos temas con claridad, es preciso introducir el concepto de "alias" o seudónimo para designar tablas.

Asignando Alias a una Tabla

Tanto por conveniencia como por necesidad, surgió la opción de sustituir el nombre de una tabla por una denominación temporaria creada por el usuario. Esto hace más flexible y fácil de usar al SQL, a la par que lo convierte en una herramienta más poderosa, al permitir operaciones que de otro modo no serían factibles. Los alias se especifican en la cláusula from a continuación del nombre de la tabla y antes de la coma que la separa de la tabla siguiente, según el modelo indicado a continuación:

```
select A.campo1, A.campo2, ..., B.campo1, B.campo2, ...,
C.campo1, C.campo2, ..., D.campo1, D.camop2, ..., D.campoN
from tabla1 A, tabla2 B, tabla3 C, tabla4 D
where (condiciones...);
```

Es importante destacar, que al estar los alias definidos en la cláusula from, luego de select tablas, el sistema reconoce los alias y los reemplaza por el correspondiente nombre de tabla.

```
select E.nroleg, E.nombre, tipo, F.nombre
from emp E, fam F
where E.nroleg = F.nroleg and tipo = 1;
```

Siendo la salida la siguiente:

nroleg	nombre	tipo	nombre
4	Miguel Angel Socos	1	Adriana Gómez
5	Alejandro Sergio Darta	1	Maria Luisa Cuiralda
6	Marcela Edith Zarce	1	Pablo Daniel Martinez
7	Raul Guillermo Caico	1	Cecilia Miranda
8	Jorge Pablo Felag	1	Silvia Hernández
10	Nilda Patricia Sovervi	1	Julio De Caro
12	César Pablo Taliga	1	Claudia Cristina Correa
18	Julio César Elías	1	María Fernanda Luissi
19	Eduardo Ricardo Estuardo	1	María Emilia Santillán
25	Roberto Diego Fla ez	1	Virginia Warburg
26	Alejandra Angeles Riveros	1	Osvaldo Martínez

Para tomar todos los campos de una tabla, por ejemplo la de empleados, sigue siendo válido indicar E.*, y en general, es lícito usar el alias en reemplazo del nombre de la tabla en cualquier circunstancia. Existen sí dos limitaciones evidentes, dado que no puede usarse como alias:

• El nombre de otra tabla.

• Una palabra reservada (keyword).

• y, es posible, pero no recomendado que se use el nombre de una tabla.

Unión de una Tabla consigo misma

Supongamos que se quiere listar a uno o varios gerentes, poniendo a continuación de cada uno los empleados que de él dependen. Esto es relativamente sencillo de realizar, asignando dos alias a una misma tabla, de esta forma se puede definir la tabla emp con dos alias. El ejemplo sería:

```
select Jefe.nroleg, Jefe.nombre, nroleg, nombre
from emp, emp Jefe
where jefe = Jefe.nroleg and depno < 3;
```

Siendo la salida obtenida:

nroleg	nombre	tipo	nombre
1	Juan Carlos Suáres	20	Clara Nieves Farola

Nótese que no se han usado nombres de campos como alias, como se recomendó, y por ello en la redefinición de la tabla emp no ha sido "jefe" sino "Jefe" (se ha empleado la mayúscula).

Otros Tipos de Unión

La relación más usual para el JOIN de tablas es igualdad entre dos campos o columnas, uno de cada tabla. Esto es lo que se denomina una "equi-unión" o "equi-join". Es posible especificar cualquier tipo de relación lógica o aritmética, no sólo la igualdad, entre ambas columnas, lo que da origen a las "No equi-uniones" o "Non-equi-joins".

Por otra parte, al efectuar una unión, independientemente de que la condición implique o no una equiunión, puede ocurrir que ciertas filas de alguna tabla no aparezcan en el reporte, por no darse ningún caso en el cual el conjunto de condiciones se verifique. Si por algún motivo es necesario que ésto quede explicitado en el reporte, se recurre al llamado "outer join", o unión externa. Ambos tipos de uniones se analizarán en las secciones que siguen.

Non-equi-joins

Si queremos listar aquellos empleados que ganan menos que una persona dada, o que un grupo determinado, tendremos una no equiunión. En el primer caso, si tomamos como referencia (alias "R") al empleado número 5, tendremos:

```
select R.nombre, R.sueldo, E.nombre, E.sueldo
from emp E, emp R
where E.sueldo < R.sueldo and R.nroleg = 5;
```

nombre	sueldo	nombre	sueldo
Alejandro Sergio Darta	4.250,00	Nilda Patricia Sovervi	4000.00
Alejandro Sergio Darta	4.250,00	Marcela Paula Diloropi	3250.00
Alejandro Sergio Darta	4.250,00	César Pablo Taliga	3250.00
Alejandro Sergio Darta	4.250,00	Laura Aída Leriblascoc	3250.00
Alejandro Sergio Darta	4.250,00	Luís Marcos Lapadeo	2750.00
Alejandro Sergio Darta	4.250,00	María Antonieta de las Flores	2750.00
Alejandro Sergio Darta	4.250,00	Estella Maris Newher	2750.00
Alejandro Sergio Darta	4.250,00	Ana María Notón	2750.00
Alejandro Sergio	4.250,00	Julio César Elías	2750.00

Darta			
Alejandro Sergio Darta	4.250,00	Eduardo Ricardo Estuardo	2750.00
Alejandro Sergio Darta	4.250,00	Clara Nieves Farola	1500.00
Alejandro Sergio Darta	4.250,00	Florio Manuel Tenorio	1500.00
Alejandro Sergio Darta	4.250,00	Nicolás Atilio Regaluh	850.00
Alejandro Sergio Darta	4.250,00	María Aída Estía	850.00
Alejandro Sergio Darta	4.250,00	Adriana Mariana Crana	2750.00
Alejandro Sergio Darta	4.250,00	Roberto Diego Fla ez	2750.00
Alejandro Sergio Darta	4.250,00	Alejandra Angeles Riveros	2750.00
Alejandro Sergio Darta	4.250,00	Marisa Clarisa Mayo	2250.00
Alejandro Sergio Darta	4.250,00	Fernando López Gabel	2250.00
Alejandro Sergio Darta	4.250,00	Alfredo Mik Ladrón	2000.00
Alejandro Sergio Darta	4.250,00	Nilda Patricia Sovervi	4000.00

Para el segundo caso, imaginemos que la referencia es ahora cualquiera de los empleados del departamento número 3. La cláusula de condicionamiento se convierte en:

```
where R.sueldo > E.sueldo and E.depno = 3
and E.depno != R.depno;
```

nombre	sueldo	nombre	sueldo
Jorge Pablo Felag	4.250,00	Carlos Gabriel Berilini	4.500,00
Adriana Mariana Crana	2.750,00	Carlos Gabriel Berilini	4.500,00
Roberto Diego Flaez	2.750,00	Carlos Gabriel Berilini	4.500,00
Jorge Pablo Felag	4.250,00	Miguel Angel Socos	4.500,00
Adriana Mariana Crana	2.750,00	Miguel Angel Socos	4.500,00
Roberto Diego Flaez	2.750,00	Miguel Angel Socos	4.500,00
Adriana Mariana Crana	2.750,00	Alejandro Sergio Darta	4.250,00

Roberto Diego Flaez	2.750,00	Alejandro Sergio Darta	4.250,00
Adriana Mariana Crana	2.750,00	Marcela Edith Zarce	4.250,00
Roberto Diego Flaez	2.750,00	Marcela Edith Zarce	4.250,00
Adriana Mariana Crana	2.750,00	Raul Guillermo Caico	4.250,00
Roberto Diego Flaez	2.750,00	Raul Guillermo Caico	4.250,00
Adriana Mariana Crana	2.750,00	Ricardo Marcelo Acol	4.000,00
Roberto Diego Flaez	2.750,00	Ricardo Marcelo Acol	4.000,00
Adriana Mariana Crana	2.750,00	Nilda Patricia Sovervi	4000.00
Roberto Diego Flaez	2.750,00	Nilda Patricia Sovervi	4000.00
Adriana Mariana Crana	2.750,00	Marcela Paula Diloropi	3250.00
Roberto Diego Flaez	2.750,00	Marcela Paula Diloropi	3250.00
Adriana Mariana Crana	2.750,00	César Pablo Taliga	3250.00
Roberto Diego Flaez	2.750,00	César Pablo Taliga	3250.00
Adriana Mariana Crana	2.750,00	Laura Aída Leriblascoc	3250.00
Roberto Diego Flaez	2.750,00	Laura Aída Leriblascoc	3250.00

El último *and* evita que se comparen entre sí los empleados que pertenecen ambos al departamento 3 ---cosa que en la realidad puede no desearse que ocurra---, para ilustrar cómo a veces pueden escaparse ciertos detalles con las no equi-uniones. Otro aspecto importante de ellas es que debe preverse la posibilidad de que cada fila de una tabla llegue a unirse con una multiplicidad de filas de la otra, dando lugar a un listado de volumen considerable.

Uniones Externas

Tratemos de obtener una lista de empleados de la empresa que tengan familiares, obteniendo el tipo y el nombre del familiar.

```
select emp.nroleg, emp.nombre, tipo, fam.nombre
from emp, fam
where emp.nroleg=fam.nroleg;
```

Obtendremos como resultado:

nroleg	nombre	tipo	nombre

4	Miguel Angel Socos	1	Adriana Gómez
5	Alejandro Sergio Darta	1	Maria Luisa Cuiralda
5	Alejandro Sergio Darta	2	Federico Darta
5	Alejandro Sergio Darta	2	Antonella Darta
6	Marcela Edith Zarce	1	Pablo Daniel Martinez
7	Raul Guillermo Caico	1	Cecilia Miranda
7	Raul Guillermo Caico	2	Santiago Caico
8	Jorge Pablo Felag	1	Silvia Hernández
10	Nilda Patricia Sovervi	1	Julio De Caro
12	César Pablo Taliga	1	Claudia Cristina Correa
12	César Pablo Taliga	2	Gustavo Daniel Taliga
16	Estella Maris Newher	2	Nicolás Newher
18	Julio César Elías	1	María Fernanda Luissi
19	Eduardo Ricardo Estuardo	1	María Emilia Santillán
19	Eduardo Ricardo Estuardo	2	María Laura Estuardo
19	Eduardo Ricardo Estuardo	2	María de las Nieves Estuardo
19	Eduardo Ricardo Estuardo	2	María Vanessa Estuardo
19	Eduardo Ricardo Estuardo	2	Juan Manuel Estuardo
23	María Aída Estía	2	Natalia Agostina Estía
25	Roberto Diego Fla ez	1	Virginia Warburg
25	Roberto Diego Fla ez	2	Vanina Alejandra Fla ez
26	Alejandra Angeles Riveros	1	Osvaldo Martínez
26	Alejandra Angeles Riveros	2	Yanina Vanina Martínez
28	Fernando López Gabel	2	Esteban Gonzalo López Gabel

Y qué pasó con el resto de los empleados? Sencillamente, ocurre que no tienen familiares. Introduciendo el operador outer, que es el "outer join" o unión externa, se agregaría al final del reporte una línea con todos los datos propios de los empleados, conteniendo las columnas relacionadas a los familiares en blanco. El operador de unión externa, que se indica mediante la outer, se especifica en la cláusula from delante de la tabla a la que se le quiere aplicar dicho

operador.

```
from emp, outer fam
where emp.nroleg=fam.nroleg;
```

El sistema actúa como si la tabla de empleados tuviera una fila adicional con todos los valores de sus campos nulos, y la junta con aquellos registros de la tabla de familiares para los cuales no se obtuvo ninguna concordancia de los campos "nroleg". Como se muestra en el ejemplo:

nroleg	nombre	tipo	nombre
1	Juan Carlos Suarez		
2	María Laura Laiso		
3	Carlos Gabriel Berilini		
4	Miguel Angel Socos	1	Adriana Gómez
5	Alejandro Sergio Darta	1	Maria Luisa Cuiralda
5	Alejandro Sergio Darta	2	Federico Darta
5	Alejandro Sergio Darta	2	Antonella Darta
6	Marcela Edith Zarce	1	Pablo Daniel Martinez
7	Raul Guillermo Caico	1	Cecilia Miranda
7	Raul Guillermo Caico	2	Santiago Caico
8	Jorge Pablo Felag	1	Silvia Hernández
9	Ricardo Marcelo Acol		
10	Nilda Patricia Sovervi	1	Julio De Caro
11	Marcela Paula Diloropi		
12	César Pablo Taliga	1	Claudia Cristina Correa
12	César Pablo Taliga	2	Gustavo Daniel Taliga
13	Laura Aída Leriblascoc		
14	Luís Marcos Lapadeo		
15	María Antonieta de las Flores		
16	Estella Maris Newher	2	Nicolás Newher
17	Ana María Notón		
18	Julio César Elías	1	María Fernanda Luissi
19	Eduardo Ricardo Estuardo	1	María Emilia Santillán
19	Eduardo Ricardo Estuardo	2	María Laura Estuardo

19	Eduardo Ricardo Estuardo	2	María de las Nieves Estuardo
19	Eduardo Ricardo Estuardo	2	María Vanessa Estuardo
19	Eduardo Ricardo Estuardo	2	Juan Manuel Estuardo
20	Clara Nieves Farola		
21	Florio Manuel Tenorio		
22	Nicolás Atilio Regaluh		
23	María Aída Estía	2	Natalia Agostina Estía
24	Adriana Mariana Crana		
25	Roberto Diego Fla ez	1	Virginia Warburg
25	Roberto Diego Fla ez	2	Vanina Alejandra Fla ez
26	Alejandra Angeles Riveros	1	Osvaldo Martínez
26	Alejandra Angeles Riveros	2	Yanina Vanina Martínez
27	Marisa Clarisa Mayo		
28	Fernando López Gabel	2	Esteban Gonzalo López Gabel
29	Alfredo Mik Ladrón		

Referencia Campos Esquemas Corrientes

El RDBMS permite definir varios esquemas en una única base de datos. Esto tiene la ventaja de que se mantienen unidades lógicas más pequeas, y permite integrar aplicaciones desarrolladas en forma independiente por distintas personas, o en distintos momentos.

Por ejemplo, sería el caso de un sistema integrado que abarque la inter-acción de varios subsistemas, como podría ser la gestión administrativa y contable de una empresa. En este tipo de sistemas, la Contabilidad actúa como núcleo en el cual los demás sistemas le brindan automáticamente información a través de pases contables en un determinado período. Para este tipo de aplicaciones, la estructura natural es la de tener un esquema por cada uno de los subsistemas presentes. Un subsistema que no sea Contabilidad, como por ejemplo Bienes de Uso o Compras, actúa de manera autónoma resolviendo las funciones administrativas propias. En ese momento, el esquema *corriente* sería el mencionado. A la hora de correr procesos que impliquen una integración contable, se necesita recurrir al esquema de Contabilidad. Aquí es donde se hace una referencia a dicho esquema, que no es el corriente, ya que se está trabajando con Compras.

Cuando se estuviera trabajando con alguna aplicación como las mencionadas en el párrafo

anterior, se definirán todos los esquemas con los que se está trabajando, como esquemas activos. El primero de ellos es el esquema corriente. Las tablas y campos de dicho esquema serán referenciadas directamente, en tanto que los pertenecientes a los demás esquemas activos se referenciarán prefijándoles el nombre del esquema al que pertenecen.

En consecuencia, resulta innecesario ---aunque válido--- usar la expresión: `esqcte.nomtab`, supuesto que "nomtab" sea el nombre de una tabla que pertenece a "esqcte" que es a su vez el esquema corriente. Sí es preciso, en cambio, escribir esquema.clientes, puesto que "esquema" no es el corriente.

El comando *use* se emplea para definir los esquemas activos:

```
use contab, compras, bduso;
```

donde el esquema de contabilidad contab *es el corriente*, utilizándose además los de compras y bienes de uso. Si necesitamos referenciar una columna de la tabla proveed dentro del esquema de compras por ejemplo *nroprov* (número de proveedor), como dicho esquema no es el corriente debemos prefijar su nombre al de la tabla:

```
select nroprov from compras.proveed;
```

Cambiando el orden de las Filas

Características de la Clasificación

La cláusula de ordenamiento contiene dos palabras clave que deben ir seguidas por una lista de expresiones ---puede ser una sola, pero al menos una--- que indican los sucesivos niveles de clasificación. Como se recordará, las listas consisten en una serie de ítems separados por comas --en este caso los ítems son expresiones--; además, es optativo indicar si la clasificación es ascendente o descendente, lo cual se hace mediante las keywords *asc* o *desc* respectivamente. Puesto que se trata de algo optativo, es lícito omitirlo; en este caso IQL supone que se trata de orden ascendente (*asc* por default).

Como ejercicio, listemos los empleados ordenados por departamento, y dentro de cada uno de ellos, en forma descendente por sueldo.

```
select nroleg, nombre, depno, sueldo
from emp
order by depno, sueldo desc;
```

Es importante advertir que la clasificación descendente sólo se aplica a la columna de sueldos. La salida obtenida mediante este comando es:

nroleg	nombre	depno	sueldo
3	Carlos Gabriel Berilini	1	4500.00
4	Miguel Angel Socos	1	4500.00

20	Clara Nieves Farola	1	1500.00
1	Juan Carlos Suarez	1	
2	María Laura Laiso	1	
8	Jorge Pablo Felag	3	4250.00
24	Adriana Mariana Crana	3	2750.00
25	Roberto Diego Fla ez	3	2750.00
6	Marcela Edith Zarce	4	4250.00
9	Ricardo Marcelo Acol	4	4000.00
12	César Pablo Taliga	4	3250.00
13	Laura Aída Leriblascoc	4	3250.00
14	Luís Marcos Lapadeo	4	2750.00
17	Ana María Notón	4	2750.00
18	Julio César Elías	4	2750.00
19	Eduardo Ricardo Estuardo	4	2750.00
23	María Aída Estía	4	850.00
26	Alejandra Angeles Riveros	5	2750.00
29	Alfredo Mik Ladrón	5	2000.00
7	Raul Guillermo Caico	6	4250.00
11	Marcela Paula Diloropi	6	3250.00
15	María Antonieta de las Flores	6	2750.00
16	Estella Maris Newher	6	2750.00
22	Nicolás Atilio Regaluh	6	850.00
27	Marisa Clarisa Mayo	7	2250.00
28	Fernando López Gabel	7	2250.00
5	Alejandro Sergio Darta	8	4250.00
10	Nilda Patricia Sovervi	8	4000.00
21	Florio Manuel Tenorio	8	1500.00

Qué ocurre cuando tanto el departamento como el sueldo son iguales para dos o más empleados? En rigor, el orden de salida es arbitrario, de modo que no existe garantía alguna de que Lapadeo, Notón, Elías y Estuardo (del Depto. 4, sueldo 2750) salgan en orden de legajo -como se ha mostrado- y no en otro orden cualquiera. Para asegurarnos esto habría que agregar al final de la cláusula *order by* el campo *nroleg*.

Cómo se Ordenan los Campos?

Campos Numéricos

En el ejemplo anterior, ambas columnas de clasificación tenían un contenido numérico, y lo que es más, intrínsecamente *positivo*. Los campos numéricos pueden tener en ocasiones valores negativos, y es importante tener en cuenta que lo sistemas en general -no solamente el IQL- respetarán los principios algebráicos, según los cuales los valores negativos son inferiores a cero (y por ende anteriores a él en una clasificación ascendente), y de dos valores negativos, el menor (el que tiene precedencia) es el de mayor valor absoluto.

Los campos de fecha (DATE) y los de hora (TIME) también son numéricos, pero responden a ciertas convenciones que se verán más adelante.

Expresiones

Una columna numérica puede ser el resultado de una expresión, calculada en función de ciertas operaciones aritméticas sobre el contenido de campos de la tabla. Es evidente que si se utiliza dicha columna como elemento de clasificación, valen los principios algebraicos expuestos en el apartado anterior.

Pero una columna definida por una expresión carece de nombre, pues no pertenece a la tabla originaria. Para ordenar por una columna calculada, es preciso repetir la expresión que se formulará en *select*, en la cláusula *order by* , como se ilustra en el ejemplo siguiente:

```
select nroleg, nombre, sueldo + comis "INGRESOS"
from emp
order by sueldo+comis desc ;
```

La salida de esta sentencia es:

nroleg	nombre	Ingresos
1	Juan Carlos Suarez	6.450,00
2	María Laura Laiso	4.900,00
3	Carlos Gabriel Berilini	2.400,00
4	Miguel Angel Socos	
5	Alejandro Sergio Darta	
6	Marcela Edith Zarce	
7	Raul Guillermo Caico	
8	Jorge Pablo Felag	
9	Ricardo Marcelo Acol	
10	Nilda Patricia Sovervi	
11	Marcela Paula Diloropi	

12	César Pablo Taliga	
13	Laura Aída Leriblascoc	
14	Luís Marcos Lapadeo	
15	María Antonieta de las Flores	
16	Estella Maris Newher	
17	Ana María Notón	
18	Julio César Elías	
19	Eduardo Ricardo Estuardo	
20	Clara Nieves Farola	
21	Florio Manuel Tenorio	
22	Nicolás Atilio Regaluh	
23	María Aída Estía	
24	Adriana Mariana Crana	
25	Roberto Diego Fla ez	
26	Alejandra Angeles Riveros	
27	Marisa Clarisa Mayo	
28	Fernando López Gabel	
29	Alfredo Mik Ladrón	

Esta consulta lista los ingresos totales de los empleados (sueldo más comisiones) de mayor a menor, poniendo el título respectivo en la columna calculada. En rigor, no es necesario que la expresión haya figurado en la cláusula *select*: puede utilizarse una expresión incluyéndola directamente en *order by*. Esta implementación, propia del IQL, no está soportada en otras implementaciones SQL

Como se observa en la salida del ejemplo anterior, algunas filas han quedado con valor indefinido para la columna INGRESOS. En la sección *Valores NULL* se explicará el motivo de esto.

Como contrapartida, por ahora no es posible en iql indicar una columna calculada en el order by por su número de ubicación, ya que por una parte este número podría no existir (si la columna no aparece en el listado) y por la otra un número sería tomado por sí mismo como una expresión válida. En una próxima versión de IQL se planea superar esta restricción permitiendo especificar el número de la columna a través de la función *col* , lo que daría lugar, en el ejemplo antes visto, a una cláusula final:

```
order by col(3) desc; (no válida en la actual implementación)
```

Campos de Carácteres

Estos campos pueden contener cualquier tipo de información, la cual en la práctica suele ser alfabética, si bien es lícita la presencia de todo tipo de símbolos: letras, dígitos y caracteres

especiales.

Como es obvio, para que un listado por orden alfabético tenga utilidad, la información debe registrarse dentro de los campos involucrados en forma apropiada (ej., cuando se trate de personas, ubicando en primer término el apellido y después los nombre).

A continuación presentamos un ejemplo:

```
select * from depno
order by ubic, nombre;
```

Los departamentos se listarán en el orden: 2, 7, 1, 3, 8, 4, 5, 6 (como muestra el ejemplo a continuación del parráfo); y, en cada ciudad, se ordenan los nombres de los departamentos alfabéticamente.

depno	nombre	ubic
2	Gerencia	Bs.As./Rosario
7	Administración	Buenos Aires
1	Dirección	Buenos Aires
3	Software de Base	Buenos Aires
8	Ventas	Buenos Aires
4	Desarrollo	Rosario
5	Documentación	Rosario
6	Mantenimiento	Rosario

Los Valores NULL

En el ejemplo para la sección *Expresiones* llama la atención la ausencia de ingresos para varios empleados en la grilla resultante. Esto se debe a que carecían información en alguno de los campos involucrados en el cálculo de la expresión "sueldo + comis". Cuando un campo carece de información se dice que contiene un valor NULO (NULL value), lo cual es distinto de un contenido igual a cero para un campo numérico. Cero es un valor perfectamente definido, mientras que NULL equivale a un valor desconocido, y toda operación efectuada sobre algo desconocido arroja un resultado indefinido. En consecuencia, toda expresión que involucre para un determinado caso un campo cuyo valor es nulo, produce como resultado también un valor NULL.

Por convención, las columnas con valor nulo se ordenan como si se tratara del valor más bajo de todos, de modo que en nuestro ejemplo aparecerán al final porque la clasificación es descendente.

Probemos ahora con:

```
select nroleg, nombre, sueldo+comis, sueldo
```

```
from emp
order by sueldo + comis desc , sueldo desc ;
```

nroleg	nombre	(expr)	sueldo
5	Alejandro Sergio Darta	6.450,00	4.250,00
10	Nilda Patricia Sovervi	4.900,00	4.000,00
21	Florio Manuel Tenorio	2.400,00	1.500,00
3	Carlos Gabriel Berilini		4.500,00
4	Miguel Angel Socos		4.500,00
6	Marcela Edith Zarce		4.250,00
7	Raul Guillermo Caico		4.250,00
8	Jorge Pablo Felag		4.250,00
9	Ricardo Marcelo Acol		4.000,00
11	Marcela Paula Diloropi		3.250,00
12	César Pablo Taliga		3.250,00
13	Laura Aída Leriblascoc		3.250,00
14	Luís Marcos Lapadeo		2.750,00
15	María Antonieta de las Flores		2.750,00
16	Estella Maris Newher		2.750,00
17	Ana María Notón		2.750,00
18	Julio César Elías		2.750,00
19	Eduardo Ricardo Estuardo		2.750,00
24	Adriana Mariana Crana		2.750,00
25	Roberto Diego Fla ez		2.750,00
26	Alejandra Angeles Riveros		2.750,00
27	Marisa Clarisa Mayo		2.250,00
28	Fernando López Gabel		2.250,00
29	Alfredo Mik Ladrón		2.000,00
20	Clara Nieves Farola		1.500,00
22	Nicolás Atilio Regaluh		850,00
23	María Aída Estía		850,00
1	Juan Carlos Suarez		
2	María Laura Laiso		

Si bien quienes carecen de comisión continúan quedando detrás de los que sí la tienen (aún cuando su solo sueldo fuera mayor que la suma de sueldo y comisión para estos últimos), al menos quedan ordenados por el importe del sueldo.

Para obtener un listado completo y ordenado, según el significado vulgar de la expresión "sueldo + comis", donde el resultado de sumar NULL es equivalente a sumar 0, sería preciso usar el operador condicional "?", capacidad avanzada del sistema que se verá en el capítulo Consultas Avanzadas (Capítulo 14).

Los valores NULL en los campos alfabéticos no suelen acarrear problemas de clasificación, pero sí pueden producirlos cuando el campo interviene en una operación lógica. En efecto, NULL no es ni mayor, ni igual, ni menor que una cadena arbitraria de caracteres (string), y nuevamente nos encontramos ante la situación donde el resultado es indefinido. A los fines de la operatoria interna de los programas, el valor nulo se asimila a un string de longitud cero.

Los Campos Numéricos Especiales

Cuando una columna tiene definida la característica de ser DATE, lo cual se especificó en el momento de crear la tabla, el sistema contempla ciertas convenciones en cuanto a formato, operaciones aritméticas y lógicas, funciones, etc. Otro tanto ocurre con la que se define como TIME.

La clasificación ascendente por *fecha* ubica las filas en orden cronológico (ej., primero ordena por ao, luego por mes, y dentro de éste por día.) Si se agrega un segundo nivel de clasificación por valor horario (campo TIME), los registros correspondientes a un mismo día quedarán ordenados por horas, minutos y segundos.

La clasificación descendente entrega los registros de forma tal que aparecen primero los de ocurrencia más reciente, desplegándose luego hacia atrás en el tiempo.

Capítulo 13

Operadores y Funciones

El Operador *like* y los Metacarácteres

A veces ocurre que queremos condicionar un select por el contenido de un cierto campo, tal como un apellido, y no estamos seguros de su escritura exacta. Un par de símbolos denominados *metacarácteres*, en conjunción con el operador *like*, nos permiten hacer una comparación aproximada según las reglas que veremos a continuación.

El operador *like* sustituye al signo igual en una cláusula *where*, en la cual el primer operando puede ser un campo, y el segundo una "seudo constante", que es un string de caracteres con partes fijas y partes variables. El signo de interrogación "?" significa un carácter arbitrario y se puede utilizar cuantas veces y en los lugares que sea necesario. Así CA??N puede equivaler a: CAZON, CASON, CASAN, CATAN, etc. pero no a CARMEN o CAMION, pues sólo hay dos letras incógnitas intercaladas y no tres.

El restante metacarácter es el asterisco "*", que simboliza un string arbitrario de "n" caracteres, donde n puede valer desde cero (NULL string) hasta 65535, que es la máxima longitud admisible para una cadena de caracteres. Probemos ahora un ejemplo concreto:

```
select * from cargos
where descrip like '*ente';
```

El primer asterisco **no es** un metacarácter: es el símbolo adoptado por el IQL para representar todos los campos de la tabla sin tener que detallar sus nombres. La segunda ocurrencia sí es un metacarácter, y lo que hace es pedir la selección de todos los cargos que terminen en 'ente'. El resultado deberá ser:

cargo	descr
1	Presidente
2	Vicepresidente

Operadores *in* y *between*

Permiten efectuar comparaciones extendidas. En el primer caso, contra una lista de valores --operador IN-- de modo que la condición se cumple si el contenido de la columna figura en la lista. Recordando que una lista va encerrada entre paréntesis, con sus elementos separados por comas, probemos el query:

```
select * from cargos
where cargo in (1, 3, 4, 7);
```

La salida obtenida es:

cargo	descr
1	Presidente
3	Director
4	Gerente
7	Analista Programador

Obtendremos, como era de esperar, la información correspondiente a los cargos 1, 3, 4, 7. Como en el caso de like es posible prefijar la negación not al operador in para excluir los casos que figuren en la lista, incluyendo todos los demás. Por consiguiente:

```
select * from cargo
where cargo not in (2, 5, 6) ;
```

La salida a esta línea de comando es:

cargo	descr
1	Presidente
3	Director
4	Gerente
7	Analista Programador
8	Secretaria
9	Becario
10	Administrativo

El operador *between* (que significa "entre") permite incluir o excluir rangos de valores. Debe ir seguido del valor mínimo y el máximo en ese orden, separados por *and* , es decir que su sintaxis es:

```
where columna between valor_min and valor_max
```

Ambos límites se consideran en forma inclusiva; vale decir, la condición se satisface si el campo resulta igual a alguno de ellos, tanto como si queda comprendido entre ambos. Formalmente equivale a la doble condición:

```
where columna >= valor_min and columna <= valor_max
```

Como ejercicio, probemos la consulta:

```
select nroleg, nombre, sueldo, comis, depno
from emp
where sueldo not between 1300 and 1900;
```

cuya salida es:

nroleg	nombre	sueldo	comis	depno
3	Carlos Gabriel Berilini	4500.00		1
4	Miguel Angel Socos	4500.00		1
5	Alejandro Sergio Darta	4250.00	2200.00	8
6	Marcela Edith Zarce	4250.00		4
7	Raul Guillermo Caico	4250.00		6
8	Jorge Pablo Felag	4250.00		3
9	Ricardo Marcelo Acol	4000.00		4
10	Nilda Patricia Sovervi	4000.00	900.00	8
11	Marcela Paula Diloropi	3250.00		6
12	César Pablo Taliga	3250.00		4
13	Laura Aída Leriblascoc	3250.00		4
14	Luís Marcos Lapadeo	2750.00		4
15	María Antonieta de las Flores	2750.00		6
16	Estella Maris Newher	2750.00		6
17	Ana María Notón	2750.00		4
18	Julio César Elías	2750.00		4
19	Eduardo Ricardo Estuardo	2750.00		4

20	Clara Nieves Farola	1500.00		1
21	Florio Manuel Tenorio	1500.00	900.00	8
22	Nicolás Atilio Regaluh	850.00		6
23	María Aída Estía	850.00		4
24	Adriana Mariana Crana	2750.00		3
25	Roberto Diego Fla ez	2750.00		3
26	Alejandra Angeles Riveros	2750.00		5
27	Marisa Clarisa Mayo	2250.00		7
28	Fernando López Gabel	2250.00		7
29	Alfredo Mik Ladrón	2000.00		5
3	Carlos Gabriel Berilini	4500.00		1
4	Miguel Angel Socos	4500.00		1

Por qué no aparece el legajo nro. 1, que no tienen un sueldo comprendido entre los valores dados? Por qué no figuran los empleados con sueldo igual a 1900? Si no logra responder a ésto último, relea este apartado. Si no da con la respuesta a la primera cuestión, repase la siguiente sección sobre NULL values.

Comparación Campos Vacíos

La única operación factible con campos vacíos es determinar si lo son, lo cuál se establece mediante las palabras clave *is null* , o su negación *is not null*. No es válida la expresión "where campo = null" y obviamente tampoco lo es intercalando un *not* . Para poder procesar columnas que contienen valores nulos, asignándoles por ejemplo un valor cero a los fines del cálculo (es lo obvio en casos como el visto en un capítulo anterior, en el cual definíamos: INGRESO = Sueldo + Comisión), es preciso recurrir a las funciones, que son tema del capítulo siguiente.

```
select nroleg, nombre, sueldo
from emp
where sueldo is not null and nroleg<6;
```

nroemp	nombre	sueldo

3	Carlos Gabriel Berilini	4500.00
4	Miguel Angel Socos	4500.00
5	Alejandro Sergio Darta	4250.00

Qué pasa si substituimos el *and* por un *or* .? Pues que ahora se incluyen todos los empleados. Los legajos Nro. 1 y 2 entran por ser menor que 6 aunque tengan vacío el campo de sueldo, ya que el *or* sólo requiere que se cumpla *una* de las condiciones propuestas.

Precedencia de Operadores

Cuando el usuario define una expresión mediante los operadores algebraicos (+, -, /, *, ^), para crear una nueva columna en la grilla o establecer un elemento de comparación en la cláusula *where*, está en realidad creando una función. Desde el punto de vista matemático, una función no es más que una relación entre varias variables, tal que el valor de una de ellas --la llamada variable "dependiente"-- queda definido si se conoce el valor de una o más que son las restantes (llamadas “independientes”). Una función como: `INGRESO=sueldo+comisión', constituye la más simple de las funciones: la lineal. Cuando una expresión se hace más compleja, es importante tener en cuenta lo que se denomina *precedencia* de los operadores algebraicos.

Los operadores en el mismo grupo (suma y resta, multiplicación y división, potencia y radicación) tienen la misma prioridad.

La máxima precedencia la tienen la potenciación y la radicación, la de nivel intermedio es la de multiplicación y división, y la menor prioridad corresponde a suma y resta.

Las Funciones del Sistema

Llamaremos funciones *individuales* a las que se refieren a una fila o registro de la tabla, y permiten obtener un valor a partir de operaciones que se realizan sobre otros valores que pueden ser variables (el contenido de una columna) o constantes (un número explícito).

IQL incluye ciertas operaciones frecuentes que se brindan al usuario "listas para usar", como el truncado de decimales o el redondeo. La primera, que se denomina **trunc** (por “truncate”), toma la parte entera de un valor numérico seguida por la cantidad deseada de decimales. Su sintaxis es:

```
trunc (expresión, número_de_lugares_decimales)
```

La función **round** es similar, pero sólo cuando la fracción decimal es inferior a 0.5. Cuando es igual o superior, se toma el dígito siguiente.

```
round (expresión, número_de_lugares_decimales)
```

La función **abs** (SALDO) devuelve el valor absoluto de la variable: 'saldó, que en el caso de un cuenta bancaria puede se positiva o negativa el valor absoluto de un número positivo es igual a sí mismo, mientras que el de un número negativo se obtienen cambiándole el signo. Como es evidente el valor absoluto de cero es cero, y para todos los demás casos debe ser siempre positivo.

IQL, también brinda al usuario las funciones aritméticas que se detallan a continuación con los operadores correspondientes:

Función	Significado
+	Adición
-	Substracción
*	Multiplicación
/	División
^	Potencia
%	Porcentage

Expresión Condicional (Operadores ?:)

IQL introduce una función que permite elegir entre dos expresiones, según que una expresión resulte verdadera o falsa. El formato de la función es:

```
(expresión ? expr1 : expr2)
```

Y retorna <exp1> si <expresión> fue verdadera y <exp2> si <expresión> fue falsa.

Una aplicación simple e inmediata de esta función es incluir todos los empleados en la grilla INGRESO:

```
select empno, nombre, sueldo, comis, sueldo+(comis is null ? 0
: comis) "INGRESOS"
from emp;
```

nroleg	nombre	sueldo	comis	INGRESO
1	Juan Carlos Suarez			
2	María Laura Laiso			
3	Carlos Gabriel Berilini	4500.00		4500.00
4	Miguel Angel Socos	4500.00		4500.00
5	Alejandro Sergio Darta	4250.00	2200.00	6450.00

6	Marcela Edith Zarce	4250.00		4250.00
7	Raul Guillermo Caico	4250.00		4250.00
8	Jorge Pablo Felag	4250.00		4250.00
9	Ricardo Marcelo Acol	4000.00		4000.00
10	Nilda Patricia Sovervi	4000.00	900.00	4900.00
11	Marcela Paula Diloropi	3250.00		3250.00
12	César Pablo Taliga	3250.00		3250.00
13	Laura Aída Leriblascoc	3250.00		3250.00
14	Luís Marcos Lapadeo	2750.00		2750.00
15	María Antonieta de las Flores	2750.00		2750.00
16	Estella Maris Newher	2750.00		2750.00
17	Ana María Notón	2750.00		2750.00
18	Julio César Elías	2750.00		2750.00
19	Eduardo Ricardo Estuardo	2750.00		2750.00
20	Clara Nieves Farola	1500.00		1500.00
21	Florio Manuel Tenorio	1500.00	900.00	2400.00
22	Nicolás Atilio Regaluh	850.00		850.00
23	María Aída Estía	850.00		850.00
24	Adriana Mariana Crana	2750.00		2750.00
25	Roberto Diego Fla ez	2750.00		2750.00
26	Alejandra Angeles Riveros	2750.00		2750.00
27	Marisa Clarisa Mayo	2250.00		2250.00
28	Fernando López Gabel	2250.00		2250.00
29	Alfredo Mik Ladrón	2000.00		2000.00

Funciones para Campos de Carácteres

En las secciones anteriores hemos visto funciones y operadores aplicables a valores numéricos, que son los más importantes en líneas generales. En esta sección se verá como aplicar funciones a los "strings" (cadena de carácteres).

Para los alcances de este manual, el único operador importante que trabaja sobre cadenas de caracteres se denomina “concatenación”, que no es otra cosa que unir una cadena con otra. Si por ejemplo el campo `letra« tiene el contenido "Alpha" y el campo `ciudad' el valor "ville", la concatenación:

```
letra || ciudad
```

da como resultado: *Alphaville*.

El operador de concatenación se simboliza por dos barras verticales consecutivas.

Un atributo importante de un campo en caracteres es su longitud. Tal valor es el que devuelve la función **length**, que obviamente debe ser un número entero.

Las funciones **lower** y **upper** convierten un campo respectivamente a minúsculas (“lower case”) o mayúsculas (“upper case”). “Case” significa “tipo de imprenta”, y basta recordar que "low" es “bajo” y "upper" significa “superior”. Su sintaxis es:

```
lower(field)
```

ó

```
lower("string")
```

Un par de ejemplos:

```
upper (ciudad) devuelve VILLE
upper ('ciudad') es ... CIUDAD
```

Queda por analizar el operador *substring*. Conviene empezar por definir el concepto de "subcadena": es simplemente una porción del campo total, seleccionada arbitrariamente a partir de una cierta posición y con una determinada longitud. Su sintaxis es:

```
field( posición_de_comienzo, longitud_del_substring)
```

Retomando el ejemplo original de esta sección:

```
letra (0,3)
```

devuelve 'Alp' (siendo letra=’Alpha’), y

```
letra (3,2)
```

devuelve los dos último caracteres: ‘ha’.

Pero existe una segunda forma de usar esta función, aplicando el concepto de separador o *delimitador*. Si dentro de un string existen varios datos separados por un carácter especial -el delimitador- podemos obtener uno de estos datos parciales en la siguiente forma:

```
field ( carácter_delimitador, [1, 0])
```

ahora el primer argumento es el carácter delimitador, y el segundo -siempre numérico entero en ambas implementaciones- es el número de orden de la ocurrencia que nos interesa.

Funciones de Conversión

Las funciones de conversión son muy útiles cuando se desea convertir una expresión a otro tipo. La función *char* toma un argumento de cualquier tipo y lo convierte en una cadena de caracteres. Es muy útil cuando se desea concatenar valores de distintos tipos. Supongamos que se deseara imprimir una fecha en el siguiente formato:

Viernes 15 de Diciembre

usaríamos la expresión:

```
dayname(fingr) || " " || char(day(fingr)) || " de " ||
monthname(month(fingr))
```

La función *day* retorna un valor numérico, por ello hemos utilizado la función *char* para que convierta el valor numérico en una cadena y pueda ser concatenado con los otros componentes de la expresión.

La otra función de conversión es llamada *num* y convierte una expresión cualquiera en un valor numérico. El resultado dependerá del tipo de expresión que se reciba como argumento, siendo estos:

• *Cadena de caracteres*: La cadena de caracteres se convertirá al número que representa. Se toma desde el comienzo de la cadena hasta que se encuentre un carácter que no es un dígito, o el fin de la misma. Por ejemplo num("123a") dará como valor 123.

• *Fecha*: Retornará un número que es el número de días transcurridos desde el 01/01/84.

• *Hora*: Retornará la cantidad de segundos transcurridos desde la hora cero.

Otras Funciones

Restan mencionar cuatro funciones: *major*, *minor*, *descr* y *count*.

• `major(arg1, arg2)` retornará el número más alto entre los dos pasados como argumento. Del mismo modo, minor(arg1, arg2) retornará el menor.

• `descr(campo)` Esta función se aplica sobre un campo que sólo puede tomar un valor de un conjunto, es decir, que tiene asociada lo que se denomina una validación *in.*

• `count(campo)` retorna la cantidad de filas de la columna especificada. Como cualquier otra función, el valor NULL no está incluido. Si en nuestra tabla-ejemplo se quiere saber en número de empleados que pertenecen al departamento 4, la sentencia debería ser:

```
select count(*) from emp where depno=4;
```

Valores FECHA

Los Campos Fecha

Deben ser descriptos con el tipo "date" al momento de definir la tabla de la cual forman parte; esto es, en los DDS (Data Definition Statements) que se verán en el Capítulo 17. El IQL despliega las fecha en el formato usual en nuestro país, denominado "europeo" por los americanos del norte: tres campos de dos dígitos, separados por barras, que suelen simbolizarse "dd/mm/aa". Esto significa que se muestra primero el día, luego el mes y finalmente los dos últimos dígitos del año. Cuando se ingreso por primera vez un campo de fecha, o se modifica una ya existente, el sistema verifica su validez, teniendo en cuenta las diversas duraciones de los meses, los años bisiestos, etc.

Operaciones con Fechas

Para hallar el lapso de días transcurrido entre una fecha y otra, basta restar la menor de la mayor.

La sustracción es la única operación aritmética válida entre dos fechas, pero se puede sumar o restar a una fecha un valor en días, para obtener otra fecha. Resulta lógico que las operaciones lógicas de comparación < , > e = son válidas entre fechas, resultando menor la fecha más antigua y mayor la más reciente. Por consecuencia, es lícito incluirlas en expresiones que involucren el uso de *in* o *between*.

Funciones de Fecha

IQL incluye algunas funciones para trabajar con campos de fechas, y son:

• *today*: retorna la fecha obteniéndola del reloj de la PC.

• *day(*fecha): retorna el número del día de <fecha>.

• *month*(fecha): retorna el número del mes de <fecha>.

• *year*(fecha): retorna el número del año de <fecha>.

• *dayname*(fecha): retorna el nombre del día de <fecha>.

• *monthname*(fecha): retorna el nombre del mes de <fecha>.

Es posible aplicar las funciones *max*(expr) y *min*(expr) a grupos de fechas, las que retornarán la máxima o mínima fecha, respectivamente.

Valores Hora

Los distingue la característica de haber sido definidos con el tipo "time" al momento de especificar la tabla que los contiene.

La función *hour* retornará la hora (obtenida del reloj de la PC) en el momento que se la requiera.

El formato con el cual se despliega un campo de hora puede simbolizarse como "hh:mm:ss", es decir tres campos numéricos de dos dígitos cada uno, separados por el carácter ':' (dos puntos). La hora puede variar de 00 a 23, los minutos de 00 a 58 y los segundos de 00 a 59. Internamente, IQL lleva el tiempo transcurrido del día en segundos, de modo que la diferencia de dos campos de hora es un valor numérico que expresa el intervalo en segundos. Al igual que con las fechas, la resta es la única operación aritmética válida para este tipo de campos.

Capítulo 14

Consultas Avanzadas

Completaremos en este capítulo los conceptos necesarios para resolver consultas más avanzadas. Se verá como agrupar por distintos criterios los resultados de una consulta, y como obtener totales sobre estos grupos. Esto se consigue mediante las cláusulas *group by* y *having*, y las totalizaciones y otras operaciones sobre los grupos con las llamadas funciones *grupales*.

Se explicará también el concepto de *subquery*, que permite realizar consultas que requieren valores calculados como resultado de una consulta, para ser utilizados por otra.

Las Cláusulas *group by* y *having*

Una operación útil --y a veces imprescindible-- consiste en agrupar las filas de una tabla por una o varias columnas, entendiéndose que pertenecen al mismo grupo aquellas filas que contienen valores iguales en el campo o campos especificados.

Esto se logra mediante la cláusula *group by* la cual debe ir después de la *where* si la hay, y en caso contrario a continuación del *from*. La utilidad de agrupar se comprende mejor con relación a las llamadas "funciones de grupo", que devuelven un valor único característico de él, como pueden ser la suma de los valores de una cierta columna -obviamente numérica- o la cuenta de registros pertenecientes al grupo.

El concepto de *grupo* no está restringido al uso de la cláusula *group by* , sino que puede resultar de un condicionamiento a través del *where* pero en este caso el grupo resultante es uno solo y parte de la tabla; *group by* maneja en cambio la totalidad de la tabla trabajando y produciendo resultados con tantos grupos como resulten de acuerdo a las especificaciones usadas. Para ejemplificar usaremos la función *count* (cuenta) que da la cantidad de ocurrencias de valores no nulos en una columna del grupo, o bien la cantidad de filas que lo componen.

Si ejecutamos la consulta:

```
select depno, count(*)
from emp
```

```
group by depno;
```

Obtendremos la cantidad de empleados en cada departamento, conforme a una salida como esta:

depno	(expr)
2	5
7	3
1	9
3	2
8	5
4	2
5	3
6	

La expresión "(*)" le indica a la función que se trata de *cuenta simple* es decir, que no interviene ningún valor de columna.

Muy distinto es el resultado de:

```
select depno, count(comis)
from emp
group by depno;
```

Aquí sólo se toman en cuenta las filas de la tabla cuya columna "comis" no sea nula; es decir, le estamos pidiendo a la función que cuente, para cada departamento, cuántos empleados pueden llegar a recibir comisiones (aunque de momento el importe puede ser cero).

La salida de esta consulta es:

depno	(expr)
2	0
7	0
1	0
3	0
8	0
4	0
5	0
6	3

La cláusula *having* puede colocarse después de *group by* para hacer selecciones dentro de los grupos. Si queremos averiguar cuáles son los cargos para los que existen dos o más empleados:

```
select cargo, count(*)
from emp
group by cargo
having count(*) >= 2;
```

cuya salida es:

cargo	(expr)
3	2
4	4
5	2
6	3
7	9
8	2
9	3
0	2

Véase que la consulta puede casi leerse como si se tratara de inglés corriente; su expresión en castellano sería: "Seleccionar cargos y cantidades de la tabla de empleados agrupados por cargo para aquellos grupos que tengan dos o más empleados".

Una especificación tal como:

```
group by depno, cargo
```

permite obtener información agrupada y ordenada por departamento y cargo ya sea obtenida por medio de *count* u otra de las funciones de grupo. Ahora el grupo está constituido por aquellas filas de la tabla emp que tienen valores respectivamente coincidentes en las columnas depno y cargo.

Las Funciones Grupales

Las funciones grupales son aplicables a un conjunto de valores pertenecientes a una determinada columna; dicho conjunto resulta de considerar ciertas filas de la tabla, que son las que constituyen el grupo, en virtud de cumplir con alguna condición estipulada. Naturalmente, el grupo puede consistir de la totalidad de los registros de la tabla.

Ya hemos visto una función grupal sencilla: *count* . Ahora veremos otras, en principio las que se aplican a campos numéricos. No obstante haremos una revisión de *count* .

La Función *count*

Recordemos que una consulta tal como:

```
select count(comis)
from emp;
```

cuenta la cantidad de filas de la tabla «emp« --aquí el grupo es la tabla completa por no haberse especificado condicionamientos-- para las cuales existe un valor (así fuera cero) del campo comisión, pero no se cuentan los NULL *values*. Esto es válido en general para todas las funciones: los casos nulos no se toman en cuenta.

Veremos que esto tiene importancia en casos como la función *promedio*, que será descripta más adelante. Para hacerlo actuar realmente como una función de grupo, podríamos agregar a la consulta anterior la cláusula:

```
where depno=4
```

Ahora sí el grupo resulta constituido por los empleados pertenecientes al departamento 4. Habíamos visto también que para contar todos los casos del grupo sin discriminación se utiliza el formato *count* (*); es decir:

```
select count(*)
from emp
where depno=4;
```

nos da la cantidad de personal del departamento en cuestión, ya sea que puedan recibir comisión o no.

Las Funciones *max* y *min*

Es algo que suele darse con frecuencia la necesidad de conocer, dentro de un grupo de valores, cuál es el más grande, o bien el más peque–o. En términos matemáticos hablamos de máximo y mínimo respectivamente. Para campos numéricos el significado es obvio, siempre que recordemos que 0 es mayor que cualquier número negativo, y entre dos de éstos el de menor valor absoluto: -3 es mayor que -4.

En forma genérica, podemos definir el máximo M de un conjunto de valores:

A1, A2, A3, ...

a los que designamos globalmente como:

Ai

con i variando entre 1 y n, el valor las dos siguientes condiciones, para cualquier valor de i:

M >= Ai

para cualquier valor posible de i, y

M =Ai

para al menos un valor válido de i.

La primer condición implica que el máximo es mayor o igual que cualquier elemento del

conjunto y la segunda, que pertenece a él, es decir, es igual por lo menos a uno de sus componentes.

Por consiguiente, *max* sueldo nos dará en principio la remuneración más alta dentro de emp ... salvo por el hecho de la existencia de campos NULL.

Como es evidente, todo lo dicho se aplica a la función *min* (mínimo) sin más que cambiar el operador de comparación a <= en la primer condición. Es posible buscar la diferencia de ambas funciones para hallar lo que se denomina rango de variación:

```
select max(sueldo) - min(sueldo) "Rango", depno
from emp
group by depno
having count(*) > 1;
```

El primer campo seleccionado --o sea lo que aparece a continuación del *select* hasta la coma --, es una expresión formada por la diferencia de las dos funciones que hemos visto en este apartado, a la cual adjudicamos un título; el segundo campo es simplemente una columna de la tabla. Agrupamos por departamento ya que eso es lo que indica la cláusula *group by* , y por último hacemos una selección de grupos mediante la cláusula *having* (no es posible usar *where* porque se aplica solo a filas individuales, no a grupos), estipulando que se considerarán solamente aquellos departamentos con varias personas. El resultado es:

cargo	(expr)
3000.00	1
1500.00	3
3400.00	4
750.00	5
3400.00	6
0.00	7
2750.00	8

Es obvio que en los departamentos con un solo integrante, *max* y *min* arroja, resultados cero por ser ambas funciones iguales al único valor existente; no obstante, ello ocurrirá también con los departamentos donde los empleados ganen los mismo.

Es factible aplicar las funciones de máximo y mínimo a columnas de carácteres en cuyo caso se obtendrá como "menor" a aquel campo que comience con el carácter más bajo en la secuencia ASCII; a igualdad del primer carácter decide el segundo, y así sucesivamente. Esto significa que aplicadas a la Guía Telefónica, *min*(apellido) nos da el primer abonado de toda la lista y *max*(apellido) el último.

Estas funciones se pueden aplicar también sobre campos de tipo fecha y hora, y lógicamente el resultado que arrojan es la máxima o mínima fecha u hora del grupo, respectivamente.

Sumatoria: La Función *sum*

Como de costumbre, cuando hablamos de la suma de los valores de una columna de la tabla, estamos usando el término en sentido algebraico; vale decir, teniendo en cuenta el signo de cada campo. Así, la suma de 3, -5 y 2 da por resultado cero. La función *sum* procede en esta forma para cada grupo que se defina en la consulta, y devuelve los valores correspondientes. Por ejemplo:

```
select depno, sum(sueldo), sum(comis)
from emp
group by depno;
```

nos dará el importe total de sueldos y comisiones abonado por cada departamento:

cargo	(expr)	(expr)
1	10500.00	0.00
3	9750.00	0.00
4	26600.00	0.00
5	4750.00	0.00
6	13850.00	0.00
7	4500.00	0.00
8	9750.00	4000.00

Suprimiendo la cláusula *group by*, toda la tabla se convierte en un único grupo, y tendremos los importes totales de sueldos y comisiones pagados por la empesa. Esto es realmente así? No: la consulta produciría un mensaje de error debido a que se le está requiriendo en la cláusula *select* que muestre el departamento, y este valor ha dejado de pertenecer al grupo. Para que la consulta funcione correctamente, es preciso suprimir depno del *select* también.

El ejemplo de error introducido en el párrafo anterior debe hacernos tener presente que no es lícito mezclar en la selección columnas o expresiones correspondientes a niveles de agrupamiento distintos de aquellos que se han especificado. Esto resulta claro si pretendiéramos pedir el nombre o el número de legajo al tiempo que utilizamos una función de grupo: el contenido de tales columnas es individual y propio de cada fila de la tabla, y entonces el sistema no tiene manera de establecer qué legajo debiera mostrar en correspondencia a un grupo de empleados que ostentan, por ejemplo, el mismo cargo.

Las restricciones que surgen de lo expuesto anteriormente pueden en buena medida superarse a través de un "sub-query", concepto que será analizado en el próximo capítulo.

Promedio: La Función *avg*

El nombre de esta función deriva de la palabra "average", que significa promedio. En la terminología estadística se denomina escuetamente "media", dándose por sobreentendido que

se trata de la media aritmética. Esta se define como el cociente entre la sumatoria de un conjunto de valores -en SQL, los que componen el grupo- y la cantidad de valores considerados. En términos prácticos, **avg=sum/count**; referidas todas las funciones como es lógico, a una misma columna de la tabla. El ejemplo más sencillo consiste en calcular el sueldo promedio de toda la empresa:

```
select avg(sueldo)
from emp;
```

El ejemplo devuelve el valor 2.951,85. Dado que hay veintinueve personas registradas en la tabla emp, significa ésto que el monto total de sueldos es 2.951,85 * 29? No, en realidad: solamente 27 empleados han intervenido en el cálculo pues hay dos filas con el campo vacío. De tal modo el total es 2.951,85 * 27 = 79.699,95.

Una vez más, es importante tener en cuenta que los valores nulos no serán considerados, lo que en muchas ocasiones puede introducir una distorsión en los resultados, o más bien en la manera de interpretarlos.

Si queremos equiparar los casos nulos a ceros, recordemos que una función puede aplicarse a una expresión, la que en este caso conviene sea una expresión condicional:

```
select avg(sueldo is null ? 0 : sueldo) "Sueldo \n Promedio"
from emp;
```

La función *avg* se puede aplicar también sobre campos de tipo fecha u hora. En el primer caso el resultado es una fecha calculada como el promedio de todas las fechas del grupo. Si tenemos por ejemplo los valores:

12/01/89; 01/07/89; 14/03/90

nos dará como promedio

29/07/89

En el caso de los campos con valores horarios la operación es similar.

Funciones Acumulativas

El concepto visto en las secciones anteriores de sumatorias, máximos y mínimos y promedio se aplicaba sobre grupos. Existen funciones que permiten realizar estas operaciones pero sobre las columnas de la tabla de salida. Dichas funciones se denominan: *runsum*, *runavg*, *runmin*, *runmax* y *runcount* .

Se pueden indicar en una consulta de la misma forma que cualquiera de las funciones vistas previamente, y el resultado que arrojarán será la sumatoria acumulada para cada fila (en el caso de la función *runsum*).

Concepto de Subquery

Se denomina así a una consulta que es utilizada por una cláusula de otro comando SQL. Generalmente, dicho comando principal es también un query, el cual se vale del comando subordinado -de allí el nombre de subquery- para establecer cierta premisa o determinar un juego de valores seleccionados que será luego objeto de la operación principal. En esta sección veremos únicamente los casos de *subqueries* usados en la cláusula *where* del query principal.

Para comprender el concepto de *subquery* es fundamenteal tener en cuenta que la operación select llevada a cabo sobre una tabla tiene como resultado otra tabla, la grilla en la terminología de iql , que está constituida por una, varias o todas las columnas de la tabla original, y otro tanto ocurre con las filas.

Para ser precisos, entonces, la grilla del subquery puede contener todas, algunas, una o ninguna de las filas de la tabla principal.

Ejemplos de Subquery

Ejemplos Simples

Comencemos por un ejemplo sencillo. Queremos saber quiénes tienen el mismo cargo que un cierto empleado; digamos Tenorio, cuyo número de legajo es 21.

Se podrían hacer dos consultas: una para averiguar el cargo y otra para seleccionar a quienes lo comparten. Es decir:

```
select cargo
from emp
where nroleg=21
```

que nos devolvería el valor 8, luego haríamos:

```
select nroleg, nombre
from emp
where cargo=8;
```

lo que nos daría el resultado final buscado.

nroleg	nombre
20	Clara Nieves Farola
21	Florio Manuel Tenorio

El uso del subquery permite obtener dicho resultado en forma directa, es decir en un solo paso, de la siguiente forma:

```
select nroleg, nombre, cargo
```

```
from emp
where cargo = ( select cargo from emp where nroleg = 21 );
```

La salida de este ejemplo es:

nroleg	nombre	cargo
20	Clara Nieves Farola	8
21	Florio Manuel Tenorio	8

Adviértase que el subquery está encerrado entre paréntesis, y no es otra cosa que el primero de los pasos ejecutados según el método anterior. Podría haber sido escrito de corrido en un solo renglón de la pantalla, pero se ha querido mostrar la técnica llamada "indentado", ampliamente usada en los procesos estructurados y principalmente en todos los lenguajes de programación, que tienen por objeto hacer patente el anidamento ("*nesting*") de procesos. Esto se logra aquí encolumnando el *where* con su correspondiente *select* .

Tal como se ha hecho con el cargo, podríamos igualmente requerir aquellos empleados que trabajen en el mismo departamento que una cierta persona, o que tengan su mismo jefe. Invitamos al lector a proponerse y practicar ejercicios de este tipo. Puede ocurrir que algún caso no sea resoluble por el sistema: empleados cuya comisión sea mayor que la percibida por un cierto número de legajo... correspondiente a alguien que no recibe comisiones. El campo tendrá valor nulo (null value), y no se le podrán aplicar comparaciones aritméticas.

En tales circunstancias se emitirá una advertencia similar a la que se produciría si se requiriera un número de legajo inexistente.

Los subqueries pueden ser múltiples, conectados por los operadores lógicos *and* y *or*. Podemos listar, por ejemplo, los empleados cuya comisión sea mayor o igual que la del empleado número 10, y que pertenezcan a su mismo departamento:

```
select *
from emp
where depno = ( select depno from emp where nroleg= 21) and
comis = ( select comis from emp where nroleg=21);
```

La salida de este ejemplo es:

nroleg	nombre	depno
5	Alejandro Sergio Darta	8
10	Nilda Patricia Sovervi	8
21	Florio Manuel Tenorio	8

Por último, veamos el caso de un subquery de anidado dentro de otro, que involucre además el uso de más de una tabla. Si queremos saber qué empleados ganan más que el promedio de sueldos del departamento de desarrollo, y listar todos sus datos:

```
select *
from emp
where sueldo>(select avg (sueldo) from emp where depno=(select
depno from depto where nombre='Desarr'));
```

Es importante notar dos cosas. Por un lado, el doble cierre de paréntesis al final, debido al anidamiento de subqueries. Por el otro, que la ejecución se realiza "de adentro hacia afuera"; esto es, primero el más interno de los subqueries averigua que el Depto. de Desarrollo es el número 4, luego el de nivel intermedio calcula el promedio de sueldos de dicho departamento, y por último el query principal lista los empleados cuyo sueldo supera dicho valor. La salida de este query es:

nroleg	nombre	sueldo
3	Carlos Gabriel Berilini	4500.00
4	Miguel Angel Socos	4500.00
5	Alejandro Sergio Darta	4250.00
6	Marcela Edith Zarce	4250.00
7	Raul Gullermo Caico	4250.00
8	Jorge Pablo Felag	4250.00
9	Ricardo Marcelo Acol	4000.00
10	Nilda Patricia Sovervi	4000.00
11	Marcela Paula Diloropi	3250.00
12	César Pablo Taliga	3250.00
13	LauraAída Leriblascoc	3250.00

NOTA: Sabemos que **select *** toma todas las columnas, y que una condición tal como **where nroleg > 0** en la tabla emp incluiría todas las filas. En realidad, puede ocurrir que la cláusula *where* no se cumpla nunca (v.gr. **where nroleg is null**), lo cual producirá una grilla vacía para el *subquery* y obviamente otro tanto para el query principal.

La Condición *in*

Esta condición tiene dos casos diferentes, de los cuales el segundo es una forma más efectiva de un uso especial del primero, y el primero es nada más que una forma distinta de llamar a una condición *all* o *any*.

El formato más general del primer caso de la condición *in* es:

```
constructor-de-fila [NOT] in ( expresión-de-tabla )
```

A continuación se presenta un ejemplo de *in*:

```
select emp.nombre
from emp
where emp.depno in ( select dept.deptno from dept where
dept.base='Buenos Aires')
```

El resultado es:

nombre
Juan Carlos Suarez
María Laura Laiso
Carlos Gabriel Berilini
Miguel Angel Socos
Alejandro Sergio Darta
Jorge Pablo Felag
Nilda Patricia Sovervi
Ana María Notón
Julio César Elías
Eduardo Ricardo Estuardo
Clara Nieves Farola
Florio Manuel Tenorio
Adriana Mariana Crana
Roberto Diego Fla–ez
Marisa Clarisa Mayo
Fernando López Gabel

Selecciona a los empleados cuya base está en Buenos Aires. Explicación: El sistema evalúa la expresión *select* dentro del *in*, para seleccionar los números de departamento (1,3,7 y 8). Luego evalúa la expresión *select exterior* para obtener los nombres de los empleados cuyo departamento está contenido dentro del grupo.

Veamos el segundo caso de la expresión in. La sintaxis es de esta forma:

```
expresión-escalar [NOT] in ( expresión-esalar-lista )
```

La lista pasada como parámetro debe estar separada por comas. Esto se puede decir que es semánticamente equivalente a:

x in (a, b, ..., z)

x=a or x=b or ... or x=z

Un ejemplo puede ser:

```
select emp.nroleg
from emp
where emp.cargo in ( 1, 2, 3)
```

El resultado es el siguiente:

nroleg
1
2
3
4

La forma negada puede ser definida intuitivamente; es decir se le debe agregar el operador *not* a la expresión. Su sintaxis es:

```
x not in ( expresión-escalar-lista )
```

La "lista" se debe expresar como en el caso anterior. Y su equivalente semántico es:

not (x in (s-e-c))

Las Condiciones *all* y *any*

La forma general de una condición *all* o *any* es:

```
constructor-de-fila
operador-de-comparación { all | any | some }
( expresión-de-tabla )
```

donde el operador de comparación es cualquiera del grupo usual (=, <, <=, >, >=, o <>), y *some* es solo una forma diferente para *any* . Si la tabla está vacía, la condición all retorna verdadero, y la condición any retorna falso. Un ejemplo:

```
select empx.nombre
from emp as empx
where empx.sueldo > all ( select empy.sueldo from emp as empy
where empy.depno = '8' )
```

El resultado de esto se vería:

nombre
Alejandro Sergio Darta

Explicación: La expresión interior retorna el conjunto de salarios para los empleados del departamento 8. La exterior selecciona y retorna el nombre del empleado cuyo salario es el más alto del conjunto.

La Condición *exists*

La condición *exists* es usada para verificar la existencia de al menos una fila de una tabla

especificada (normalmente se hace con tablas derivadas), es decir si la tabla en cuestión no está vacía. Su sintaxis es:

```
exists ( expresión-de-tabla )
```

La condición evalúa falso si la expresión de tabla evalúa una tabla vacía, de lo contrario retorna verdadero. Un ejemplo podría ser:

```
select emp.nombre
from emp
where exists ( select * from relat where relat.nroleg =
emp.nroleg )
```

"Seleccionar el nombre de los empleados que tienen por lo menos un familiar". La condición *exists*, en este caso, verifica la existencia de filas en la tabla FAM que satisfagan la condición de que su componente **nroleg** tiene igual valor al representado por la condición exterior **emp > nroleg**. Hay dos puntos a tener en cuenta:

1. La cláusula *select* es de la forma “select *”; en la práctica, los argumentos de la condición *exists* son generalmente de esta forma.

2. El argumento de la condición *exists* incluye una referencia exterior (“emp.nroleg”). En la práctica, estos argumentos frecuentemente incluyen por lo menos una referencia exterior.

Capítulo 15

Sentencias de Control

La sentencia *close*

Cuando se cambia de esquema corriente mediante un use o un create, ello no significa que el esquema anteriormente corriente deje de estar activo. Si se quiere suprimir un esquema de la lista de activos, se usa una sentencia tal como:

```
close ALFA;
```

Pero si este esquema era precisamente corriente, quién pasa a serlo ahora? Queda indefinido, el usuario debe indicarlo mediante un *use* .

La cláusula *output*

En modo interactivo, la salida normal del sql es la pantalla, mientras que en modo batch es lo que el sistema operativo (es decir UNIX) tenga definido como salida estándar. Ella es usualmente la terminal.

Cuando se desea alterar el destino de la consulta, podemos recurrir a la cláusula output to destino, que debe ser la última del query, y puede tener los siguientes operandos:

• *terminal*: Permite dirigir la salida de un proceso batch a la pantalla para ver el resultado mediante una grilla.

• *printer*: Direcciona la salida a la impresora (ver variable de ambiente printer).

• 'filename': El resultado de la consulta será escrito en un archivo con el nombre especificado.

• *report 'rnombre_rep*': Esta última opción permite lo que se denomina “formatear” la salida (asignarle un formato), recurriendo a las facilidades que brinda el lenguaje de diseño de formularios RDL de IDEAFIX.

Variables Impresión

Cuando se utiliza *output to printer* , o la impresora resulta ser el dispositivo estándar de salida, puede ser necesario tener en cuenta --y eventualmente modificar-- ciertos parámetros que hacen a la manera en que se imprimirá el listado. Estos parámetros están representados por variables predefinidas en IQL. Los nombres de variable IQL de dichos parámetros, que hacen a las características de impresión, junto con su significado y valores por defecto, se brindan en el cuadro siguiente.

Variable	default	Descripción
topmarg	0	Margen superior libre en cada página, entre el primer renglón posible y el encabezamiento.
botmarg	0	Margen inferior libre entre el pie de página y la última línea de impresión.
heading	$empesa\t\t#D	Tipo ¬string¬. Define el contenido del encabezamiento de página. Puede tener hasta 3 partes.
footing	\t-#P-	String de carácteres que se imprime al pie de página, con hasta tres secciones como el anterior.
flenght	66	Longitud de formulario, expresada en cantidad total de líneas impresas.
fwidth	80	Ancho del formulario, expresado en columnas efectivas de impresión.

Las dos primeras variables, al igual que las dos últimas, son simplemente valores numéricos y no presentan problema. Si queremos modificar -por ejemplo- la longitud del formulario, acortándolo un poco, bastará emitir el comando:

```
set flength=60
```

Las variables de encabezado (heading) y pie de página (footing), en cambio, requieren una explicación. En primer lugar digamos que cada uno de ellos puede constar de varias líneas separadas por " n". Cada línea consta de tres secciones que se separan mediante el carácter "TAB" (Tabulado) que se suele simbolizar con una barra inversa ("\") llamada “backslash”, seguida de la letra "t". La primera sección va ajustada a la izquierda; la segunda centrada, y la última se ajusta a la derecha.

En el caso del encabezado predefinido, vemos que la primera parte es una variable de

ambiente, por ir precedida por el signo “dólar”, que se supone definida como el nombre de la empresa. Es decir, lo que se va a imprimir no es la palabra “empresa”, sino el contenido de la variable así llamada. Si ésta no ha sido definida (o "seteada", según un barbarismo de amplia difusión), quiere decir que equivale a un valor nulo, luego nada se imprimirá en el lado izquierdo del encabezado.

La presencia de dos tabulados consecutivos indica que el defecto para la sección central es efectivamente un “null value”, y finalmente encontramos en la parte final otra expresión simbólica, esta vez predecedida por el símbolo "#" (numeral). El sistema reconoce tres valores de este tipo, que son:

```
#P Número de Página (Page)
#D Fecha del sistema (Date)
#T Hora del sistema (Time)
```

Vemos entonces que el pie de página consiste en su número encerrado entre guiones, ubicado en el centro. Esto se debe a que la especificación comienza con un tabulado, lo convierte a la parte de la izquierda en valor nulo. Como es obvio, al no haber un segundo TAB la sección derecha también es nula.

Datos Compartidos y Seguridad

Con el transcurso del tiempo, las empresas usuarias de equipos de procesamiento de datos han ido tomando conciencia de la imperiosa necesidad de proteger su información.

En consecuencia, los sistemas han sido dotados de medidas de seguridad para evitar este tipo de actividades. Ello implica que no cualquiera puede tener acceso a una tabla; puede ocurrir también que, estando autorizado a leerla -por ejemplo, haciendo un query- no pueda en cambio introducirle modificaciones. El objeto de esta sección será entonces analizar quién y cómo puede tener acceso a los esquemas y tablas, y con qué alcances.

Concesión de Privilegios

Denominamos privilegios a las autorizaciones que recibe un usuario para efectuar determinadas actividades sobre una tabla o esquema.

Las sentencias GRANT y REVOKE permiten manejar los privilegios de acceso de los distintos usuarios a la Base de Datos. El concepto de “usuario” coincide con el implementado por el Sistema operativo.

Existen dos tipos de usuario respecto de un esquema: el dueño, y el resto de los usuarios. El dueño del esquema tiene todos los privilegios sobre el mismo, y la capacidad de decidir qué privilegios tendrán los restantes usuarios. El dueño de un esquema es aquel que lo crea o quien ha sido designado para serlo (mediante la sentencia CHOWN), y todos los archivos sobre el sistema operativo creados con relación al esquema (el directorio del mismo, tablas, índices, etc) pertenecerán a él.

CHOWN

Este comando permite cambiar el dueño de un esquema existente (CHange OWner). La sintaxis es análoga a la del comando chown del Sistema Operativo UNIX:

```
CHOWN usuario lista_esquemas
```

donde usuario es el nombre de algún usuario del sistema y lista_esquemas es la lista de esquemas, separados por comas, a los cuales se les alterará el dueño. Por ejemplo:

```
chown pedro esq1, esq2 ,..., esqN
```

Además del propio dueño, sólo el superusuario de UNIX (o root) tiene permitido implementar este comando.

GRANT

La sentencia GRANT permite al dueño de un esquema definir privilegios de acceso a uno o varios esquemas y/o a una o varias tablas de un esquema. Por lo tanto este comando puede especificarse de dos formas distintas:

```
GRANT privilegio ON SCHEMA lista_esquemas
TO lista_usuarios
```

Mediante esta alternativa se definen los privilegios de acceso del (los) usuario(s) especificado(s) en *lista_usuarios* , al esquema o esquemas indicados en *lista_esquemas* . Los privilegios pueden ser:

Permiso	**Autoriza a ...**
USE	use
TEMP	select ... order by ...
MANIP	insert-update-delete
ALTER	Modificar estructura del esquema [create] schema [create] table, etc.

La lista de usuarios puede especificarse según las siguientes alternativas:

PUBLIC	Todos los usuarios gozan de los privilegios asignados
GROUP nombre_grupo	El grupo de usuarios denominado nombre_grupo goza exclusivamente de los privilegios asignados.

nombre_usuario	El nombre_usuario es el único que goza de los privilegios indicados.

Como se ha mencionado, también se pueden definir los privilegios de acceso de un usuario o de un grupo de usuarios respecto de las tablas pertenecientes a un esquema. Sólo el dueño puede definir todos los privilegios de otros usuarios respecto del esquema en sí. El resto de los usuarios podrá asignar permisos en la medida que estén autorizados a hacerlo. Su sintaxis es:

```
GRANT privilegio-tab ON nombres-tabla
TO {PUBLIC | usuario | lista-usuarios}
[WITH GRANT OPTION]
```

Explicación de la Sintaxis

privilegio-tab	Es uno o más de los siguientes permisos (si se es pecifica más de uno, se debe separar con comas): INSERT: Agregar nuevas filas (registros) a la tabla (se hereda) ADD: Idéntico al anterior. DELETE: Borrar filas de la tabla (se hereda). UPDATE: Modificar datos de filas existentes (se hereda). SELECT: Leer datos de filas existentes. ALL [PRIVILEGES]: Es sinónimo de todos los anteriores. Puede indicarse simplemente como ALL, o bien como ALL PRIVILEGES.
nombres-tab	Es el nombre de la tabla para la cual se están otorgando privilegios de acceso. Debe pertenecer al esquema corriente; o bien a cualquier esquema activo si se indica “esquema.tabla”. La especificación “esquema.*” se refiere a todas las tablas del esquema. Cuando se indica más de un nombre se deben separar con comas.
PUBLIC:	Es el nombre de la tabla para la cual se están otorgando privilegios de acceso. Debe pertenecer al esquema corriente; o bien a cualquier esquema activo si se indica “esquema.tabla”. La especificación “esquema.*” se refiere a todas las tablas del esquema. Cuando se indica más de un nombre se deben separar con comas.
lista-usuarios	Es una lista de nombres de usuarios del sistema separados por comas, a los cuales se

	les otorgan los permisos indicados.
GRANT OPTION:	Indica que cualquiera de los usuarios que haya recibido el permiso tendrá la capacidad de pasárselo a otros usuarios. Si así lo desean, pueden a su vez entregar el permiso con GRANT OPTION, para que el receptor pueda igualmente volver a trans mitirlo.

Ejemplos

```
GRANT ALTER ON SCHEMA personal TO pedro
```

El usuario pedro podrá acceder, modificar y eliminar datos del esquema, e incluso cambiar su estructura.

```
GRANT ALL ON emp TO gloria;
```

Gloria tiene todas las autorizaciones posibles sobre la tabla emp, pero no puede transferirlas a otros.

```
GRANT SELECT ON emp TO pedro WITH GRANT OPTION;
```

Pedro puede consultar todos los campos de emp , y permitir a otros hacerlo.

Visualizando Permisos

En los primeros capítulos se mencionó y ejemplificó la sentencia *show*. Estamos en condiciones de entender la información completa proporcionada por esta sentencia, ya que comprendemos el significado del término dueño de un esquema y los distintos privilegios de acceso.

En el primer caso la sentencia *show* nos mostraba la lista de esquema activos, de la siguiente forma:

Nombre de Esquema	descripción	Dueño	Perm.
esquema 1	Descripción del esquema 1	willy	ALTER
esquema 2	Descripción del esquema 2	raul	ALTER
.....	.....	...	..
esquema N	Descripción del esquema N	pablo	ALTER

La tercera columna nos indica quién es el actual dueño del esquema y la cuarta los permisos

que tiene el usuario que ejecuta el comando sobre el esquema.

El segundo caso de la sentencia *show* nos permitía consultar las tablas de un determinado esquema:

Nombre de Tabla	Descripción	Perm.	Adm.
emp	Legajos del personal	idus	idus
cargos	Codificador de cargos	idus	idus
depto	Departamentos de la empresa	idus	idus
fam	Familias de los empleados	idus	idus

La tercera columna indica los permisos que tiene el usuario sobre las tablas y la cuarta los que puede asignar a otros usuarios. La simbología utilizada se interpreta de la siguiente forma:

Posiciones	1	2	3
4	i (insert)	d (delete)	u (update)
s (select)			

Veamos cómo aparecen en pantalla distintas alternativas de permisos:

Permisos	Descripción
i---	Permiso de insert
id--	Permiso de insert y delete
idu-	Permiso de insert, delete y update
idus	Permiso de insert, delete, update y select

Cualquier combinación de las distintas posiciones es válida. Se utiliza el mismo criterio para la columna titulada grant (permisos que el usuario puede asignar a otros).

REVOKE

La sentencia REVOKE permite quitar permisos sobre esquemas y/o tablas, por lo tanto puede presentarse de dos formas distintas. En el caso de eliminar permisos sobre un esquema la sintaxis es la siguiente:

```
REVOKE privilegio ON SCHEMA lista_esquemas
FROM lista_usuarios
```

Al quitarle privilegios a un usuario o a un grupo de usuario se le asigna a su vez el privilegio

inmediato anterior, siendo el orden creciente:

• USE

• TEMP

• MANIP

• ALTER

La especificación de la lista de esquemas y de la lista de usuarios es idéntica a la de la sentencia GRANT.

La otra alternativa de la sentencia REVOKE permite al dueño de un esquema cancelar los privilegios de acceso de un usuario o grupo de usuarios respecto de las tablas. Su sintaxis es:

```
REVOKE privilegio-tab ON nombres-tab
FROM {PUBLIC | lista-usuarios}
```

Explicación de la Sintaxis

privilegio-tab	Es uno o más de los permisos detallados en el apartado anterior; como antes, si se especifican varios privilegios deben ir separados por comas. INSERT: Adiciona nuevas filas a la tabla. ADD: Un sinónimo de la anterior. DELETE: Suprime filas de una tabla. UPDATE: Modifica el contenido de filas existentes. SELECT: Lee datos de registros existentes. ALL: el juego completo de opciones. Se puede especificar simplemente con la palabra ALL o también con ALL PRIVILEGES.
nombre-tab	Es el nombre de la tabla para la cual se están eliminando privilegios de acceso. Debe pertenecer al esquema corriente si sólo se indica su nombre. Cuando integre cualquier otro esquema activo, se indicará "esquema.tabla".
PUBLIC :	Se utiliza cuando el/los permisos se suprimen para todos los usuarios del sistema.
lista-usuarios	Se aplica un grupo de usuarios, cuyos nombres deberán estar separados por comas.

Notas

• La sentencia REVOKE sólo puede ser aplicada por el dueño del esquema.

• El dueño de un esquema y el super-usuario no pueden quitarse permisos a sí mismos. La sentencia REVOKE en ese caso no realizará cambio alguno.

• Intentar realizar REVOKE de permisos que un usuario no posee resultará en un error de ejecución de la sentencia.

• La sentencia REVOKE no es registrada por el LOG de transacciones. A

• Para realizar una modificación de permisos, se intenta bloquear el esquema. Esto sólo es posible si ningún usuario lo está utilizando.

Ejemplos

```
// // Entregar a Gloria permiso para manejar
// la tabla emp, salvo para selección y
// actualización de registros.
GRANT ALL ON emp TO gloria;
REVOKE SELECT, UPDATE ON EMP TO gloria;
// Pedro puede consultar todos los campos,
// y asimismo permitir a otros hacerlo.
GRANT SELECT ON emp TO pedro WITH GRANT OPTION;
// Ahora Pedro puede continuar consultando, pero ya
// no puede otorgar el privilegio a más usuarios.
REVOKE SELECT ON emp FROM pedro;
GRANT SELECT ON emp TO pedro;
```

Nótese que la opción de GRANT no es revocable por sí misma: es necesario revocar el privilegio concedido con ella, y luego volver a otorgarlo sin la opción de GRANT.

Otras Sentencias

Veremos ahora cuatro sentencias que son utilizadas para evitar salir del IQL cuando necesitemos:

• cambiar de directorio;

• ejecutar comandos del Sistema Operativo, o

• correr procesos usuarios del Window Manager.

Por ejemplo, si necesitamos conocer el contenido del directorio actual podemos usar:

```
shell "ls | more";
```

En este caso la sentencia *shell* blanquea la pantalla, ejecuta el comando y vuelve inmediatamente al IQL.

Es posible ejecutar este mismo comando pero volcando su salida sobre una ventana y esperando el ingreso de una tecla para retorna al lenguaje de consulta. Para ello podemos

emplear la sentencia *pipe* de la siguiente manera:

```
pipe "ls | more";
```

También podemos cambiar el directorio corriente. La sentencia para esta operación es CD , y su sintaxis es la siguiente:

```
cd /usr3/stock/src;
```

ó bien:

```
cd /usr2/acct/manolo;
```

Por último, veamos la ejecución de un proceso usuario del Window Manager. Usando la sentencia *wcmd*, ejecutamos mediante el comando doform el formulario clientes :

```
wcmd "doform clientes";
```

Capítulo 16

Actualizando Información

La sentencia *insert*

El agregado de una fila o registro a una tabla recibe el nombre específico de INSERCION, de allí el nombre de este comando: INSERT (insertar). Su sintaxis es:

```
insert into tablename(lista_de_columnas) values valores;
```

La tabla cuyo nombre hemos simbolizado como "tablename" debe pertenecer al esquema corriente; si no fuera así, podemos optar entre dos caminos: anteponer el nombre del esquema al de la tabla (el esquema debe estar activo) o bien emitir previamente un comando *use* .

La lista de columnas es optativa, pudiendo indicar un subconjunto del total en cuyo caso las omitidas quedarán con valor nulo -siempre que las especificaciones lo permitan-.

Por último nos queda la especificación de los valores a incluir.

La forma más simple consiste en la palabra clave values seguida de otra lista con los datos correspondiente. Así por ejemplo:

```
insert into emp (nroleg, nombre, cargo, fingr, depno)
values(015, "Mónica Gómez", 7, '23/11/89', 01);
```

Los campos jefe, fecha de nacimiento, sueldo y comisiones quedan con el valor por default o con valor NULL. Se supone que sus valores se ingresarán más tarde. De esta manera se incluyen nuevas filas de a una por vez.

En el caso de que no se especifiquen las columnas los valores indicados en *values* deben corresponderse uno a uno con los campos definición de la tabla. Existe otro medio más poderoso -pero que por lo mismo debe utilizarse con cuidado- que es suministrar los valores por medio de una consulta. Si recordamos que el resultado de una consulta es lo que llamamos *grilla*, que en definitiva no es otra cosa que una tabla, esto no debe sorprender. Como es obvio, el *select* deberá elegir las mismas columnas que hemos especificado en la lista del *insert* , tomadas de una tabla diferente de aquella en la cual se hace la inserción. Esto

último significa que no es lícito copiar una tabla sobre sí misma.

La cláusula *where* permitirá copiar filas en forma selectiva, o bien tomar la totalidad de la tabla, si se omite dicho *where*, esto nos da la posibilidad de hacer lo que se llama un "backup" (copia de respaldo) según el procedimiento que sigue:

```
create table tabkup(... -a lista_column=tabla_orig ...-);
insert into tabkup select * from tabla_orig;
```

Resulta evidente que la tabla de backup (tabkup) es una copia fiel (todas las filas, todas las columnas) de la tabla original (tablaorig).

Actualizando Valores: *update*

Para ubicar un ítem o dato específico dentro de una tabla, sabemos que es necesario fijar dos referencias o "coordenadas": la fila y la columna. La primera debe ubicarse a través de un dato que la identifique en forma unívoca -el cual suele ser por ellos la *clave* del registro- como es el número de legajo en la tabla emp. La columna sabemos que se identifica por su nombre.

Por otra parte, es posible que la modificación de un campo quiera hacerse extensiva a un grupo de registros, o aún a toda la tabla. Sería el caso en que se aumentará el sueldo en un porcentaje dado a un departamento, o bien a toda la empresa.

Hechas estas consideraciones previas, veamos la sintaxis del comando *update*:

```
update tablename set lista_de_asignaciones
[where condición_de_búsqueda];
```

La cláusula *where* es optativa: si se utiliza, permite seleccionar uno o varios registros que cumplan con la condición impuesta; de lo contrario, se actualizarán todas las filas de la tabla.

Es convención generalmente adoptada el encerrar entre corchetes los ítems optativos; en ningún caso los corchetes forman parte de la sentencia tal como debe ser leída por el sistema.

La cláusula *set* trabaja por definición con un lista de asignaciones, por lo que no es necesario utilizar paréntesis para delimitarla. El fin de la lista viene dado por lo aparición de la cláusula *where* o bien del punto y coma que señala el final del comando, lo que fuere que ocurra primero.

Las asignaciones deben ir separadas por comas, y cada una de ellas debe observar el siguiente formato:

```
columna=expresión
```

donde la expresión puede ser un valor constante o bien el resultado de operaciones efectuadas con campos pertenecientes a esa misma fila. Esto incluye el valor de la propia columna que se quiere actualizar, como en el caso del incremento de sueldos ya mencionado:

```
update emp set sueldo=sueldo *1.25
where depno=2;
```

que otorga un aumento del 25% a todos los empleados del Departamento de Desarrollo. Si suprimimos la cláusula *where*, para lo que deberíamos colocar el ";" luego del factor 1.25, el incremento se hace extensivo a toda la empresa.

En ciertos casos en que el valor a reemplazar puede depender de alguna condición, es útil recordar la posibilidad del uso de la expresión condicional:

```
(condition ? trueval : falseval)
```

que se evalúa como "trueval" (valor por verdadero) si la condición "condition" se cumple, mientras que en caso contrario toma el valor "falseval" (valor por falso). Para actualizar un campo que contuviera el porcentaje de descuento por Obra Social cuando dependía de la existencia o no de cargas de familia, y suponiendo una variable booleana *cargfam* que lo determinara, la expresión:

```
(cargfam=1 ? 3.0 : 2.0)
```

permite asignar el 3% a quienes tienen cargas y el 2% a los que carecen de ellas.

La sentencia *delete*

Elimina filas completas de una tabla, conforme a la sintaxis:

```
delete from tablename
where condicion_de_baja;
```

donde se borran las filas indicadas en la condición where. Si esta condición se omite, se borrará toda la tabla. Cuando se ejecuta un *delete*, se suprime toda la información pero la tabla en sí como estructura lógica continúa existiendo. La situación es entonces similar a lo que ocurre inmediatamente después de un *create* (cfr. primer párrafo de este capítulo).

Si se ha borrado una tabla completa por error, queda el recurso de restaurar la información apelando a un backup, el cual se recomienda tener disponible hasta tanto se verifique que los datos no sean realmente necesarios. Existe otra posibilidad: ejecutar el comando *rollback*, del cual hablaremos brevemente en la siguiente sección.

El comando *rollback*

Su efecto consiste en volver atrás lo hecho dejando la tabla en su estado original. Para comprender cómo es posible esto, haremos algunas simples consideraciones.

La modificación de datos almacenados en un sistema de computación tiene dos etapas bien definidas: la primera de ellas es la registración en memoria de los cambios efectuados -altas, bajas, modificaciones-; la segunda es la grabación de tales cambios en disco para hacerlos permanentes. El estado anterior de la tabla solamente se pierde cuando se hace efectiva la

segunda operación.

Por ende, si se emite un comando *rollback* antes de producirse la alteración física de la tabla, habremos salvado la dificultad. Como es natural, lo dicho vale tanto para *delete* como para *update* .

El comando *commit work*

Este comando finaliza todas las transacciones pendientes. Es decir efectiviza las modificaciones, bajas, altas, etc. de los registros afectados durante la sesión del IQL. Esto implica que hasta tanto se ejecute este comando, los registros afectados quedarán *bloqueados* para el resto de los usuarios.

Una vez cumplido el comando, sólo se podrá recuperar con *rollback* aquellas operaciones que se realicen con posterioridad.

Esto significa que el *rollback* recuperará el estado anterior de la tabla a partir del último *commit work* que se haya ejecutado. Si nunca se ejecutara en forma explícita el comando *commit work*, se efectivizarán las transacciones cuando se abandone el utilitario IQL. Esto equivale a decir que dicha salida implica un "commit work" implícito.

Algunas Consideraciones Adicionales

Siempre es prudente obtener una copia de respaldo de una tabla antes de hacerle modificaciones, particularmente si van a consistir en actualizaciones y bajas.

Antes de dar por buena una alteración, conviene asegurarse cuidadosamente de que efectivamente las cosas se hicieron bien.

Algunas veces, resulta más conveniente y seguro reemplazar un *update* de condiciones complejas y valores múltiples por el mecanismo de reemplazo *delete/insert*. Es decir, se generan en primer lugar registros con el formato apropiado en una tabla auxiliar, cuyo contenido sea exactamente lo que queremos obtener como resultado final. Si este proceso no produce el resultado querido, puede rehacerse hasta obtenerlo. Luego se suprimen todos los registros correspondientes de la tabla principal, y finalmente se insertan todas las filas tomadas de la tabla auxiliar.

Recalcamos que este es un procedimiento de excepción, y que no debe tomarse como modelo normal de operación.

Capítulo 17

Sentencias de Definición de Datos

Creación de Esquemas y Tablas

Si bien es posible definir, crear y modificar esquemas y tablas de manera interactiva, es recomendable generar archivos que contengan las sentencias DDS y ejecutarlas en modo *batch*.

Cómo Crear un Esquema

Para la creación de un esquema existe la sentencia create schema . Así por ejemplo:

```
create schema personal;
```

dará origen a un esquema llamado "personal" y además lo convertirá en el *esquema corriente*. Vemos que la creación de esquemas implícito un *use*, en tanto fija un nuevo esquema corriente.

Es importante tener presente las consecuencias de lo antedicho: toda operación sobre tablas ejecutada a continuación se referirá a una tabla del esquema corriente mientras no se especifique lo contrario.

Supongamos que el esquema corriente sea uno denominado ALFA. Para poder hacer referencia a una tabla perteneciente a otro esquema, deberá prefijarse el nombre de éste (v.gr. BETA.TABXX); de lo contrario obtendremos un mensaje de error dando a TABXX como inexistente. Por supuesto, si ya no precisamos aludir a las tablas del esquema ALFA, será más práctico emitir un comando

```
use BETA;
```

y luego continuar con nuestro trabajo sobre las tablas de ese esquema.

Definición de Tablas

La sentencia `create table` se utiliza para definir las características de una tabla, para ello se especifica el nombre de la misma, y luego debe seguir una lista encerrada entre paréntensis, con la descripción de las columnas separadas por comas. Antes de entrar en detalle sobre la sintaxis de este comando, es conveniente decir unas palabras sobre la manera de designar esquemas y tablas.

Es útil saber que el concepto de esquema consiste en un conjunto de tablas lógicamente relacionadas. Como es obvio, no puede haber dos esquemas con el mismo nombre, ni dos nombres de tabla iguales dentro del mismo esquema.

Nótese que el sistema sí acepta que haya dos tablas de nombres iguales pertenecientes a distintos esquemas.

Los nombres en sí pueden tener hasta 11 caracteres de longitud en UNIX y 8 en DOS, alfabéticos o numéricos; se acepta como alfabético el carácter subrayado "_". El primero de los caracteres de un nombre de esquema o tabla debe ser alfabético. Son nombres válidos:

PERSON_EXT (Nómina de personal asignado en el exterior)

ESTAD89 (Estadísticas del año 1989)

En cambio no son válidos:

*SALDOS (Primer carácter no alfabético)

SALDOS/88 (Contiene un carácter no alfabético ni numérico)

35_ACUM (Primer carácter no alfabético; es válido en cambio ACUM_35)

3. TRANSACCIONES (Más de 11 carácteres)

No se hace diferencia entre mayúsculas y minúsculas.

La sentencia *create table*

Veamos como ejemplo la manera de crear la tabla emp que nos ha servido para ensayar los ejercicios de los capítulos anteriores.

Si bien la mayor parte de los elementos definitorios de una tabla ha surgido implícitamente con el uso, es conveniente establecerlos en forma completa y detallada.

Tal definición consiste en una lista de datos referentes a las columnas que componen la tabla; para cada columna existe un cierto grupo de características obligatorias, mientras que algunas otras son optativas. Por su parte, la tabla en conjunto tiene como obligatoria la definición de las columnas, pero otras especificaciones son optativas.

Pasemos a detallar entonces la definición de una columna:

nombre

Debe responder a las normas ya indicadas sin restricciones en la cantidad de caracteres.

tipo dato

• CHAR, contiene una secuencia de carácteres de una cierta longitud. El otro tipo char es CHAR VARYING ó VARCHAR y se usa para almacenar secuencias de longitud variable.

• FLOAT, para registrar cifras en la modalidad de "punto flotante". Se utiliza cuando es preciso manejar valores muy grandes o muy pequeños. Un alias para este tipo es REAL.

• DATE, para almacenar fechas en formato interno.

• TIME, para almacenar horas, minutos y segundos en formato interno.

• NUMERIC (numtot [, numdec]); la suma de la parte entera y la parte decimal no debe exceder las 28 posiciones. *numtot* es el número total de posiciones numéricas, y *numdec* es el número de posiciones decimales. Este número se substraerá del anterior, y 4,2 generará un número con 2 posiciones enteras y 2 decimales.

• DECIMAL es un alias para el tipo NUMERIC.

• INTEGER soporta desde -2147483647 hasta 2147483647.

• BIGINT soporta hasta 20 dígitos.

• SMALLINT soporta desde -32767 hasta +32767.

• TINYINT soporta hasta 3 dígitos.

• BOOL, para almacenar valores de verdad. (verdadero ó falso).

• BIT es un alias para BOOL.

• BINARY es un tipo que puede contener datos binarios que usen la capacidad BLOB (Binary Large OBject) de Essentia.

• TEXT es un tipo que puede contener sólo datos de texto que usen la capacidad BLOB de Essentia.

[*longitud*]

Se indica entre paréntesis para CHAR y NUM la cantidad de caracteres o dígitos, respectivamente. No corresponde ponerlo para los campos de *fecha*, *hora* o *float*, ya que tienen un formato predefinido. En un campo NUM puede especificarse una cantidad de posiciones decimales: (8,2) indica que la longitud es ocho dígitos, pero los dos últimos son decimales a la derecha de la coma. Si el valor es entero, la cantidad de decimales es cero, v.gr.:(5,0), pero es más cómodo expresarlo simplemente como (5).

[*validez*]

Esta especificación es optativa, y se refiere a la posibilidad de que el iql controle el contenido

de las columnas para admitir o rechazar los valores que se intenten incorporar. Los criterios de validez permiten especificar operadores relacionales como <, >, etc. y la verificación de que un valor se encuentre en una tabla.

[*otros*]

En la definición de una columna se pueden especificar una descripción del significado del valor almacenado en la misma y un valor por omisión a utilizar en ciertas operaciones.

Ahora estamos en condiciones de comprender bien la siguiente sentencia:

```
create table emp ( // Nómina de empleados
nroleg num(3) not null, // Número de empleado
nombre char(20) not null, // Nombre y Apellido
cargo num (1), // Tarea que desempe a
jefe num (3), // Nro. del gerente
fnac date, // Fecha de nacimiento
fingr date, // Fecha de ingreso
sueldo num (7,2), // Remuneración mensual
comis num (7,2), // Comisiones del mes
depno num(2) // Número de departamento
) primary key (nroleg);
```

Las dos primeras columnas tienen una especificación de validez "not null", que no permite dejar sin un valor estos datos; las restantes sí pueden carecer de valor. Es importante notar la apertura de paréntesis antes del Nro. de legajo, que indica el comienzo de la lista, y el cierre de la misma luego de la especificación de Nro. de Departamento.

Encontramos además una especificación adicional relativa a la tabla, y es que la *clave primaria* (primary key) es el número de empleado. Esto da lugar a que el sistema genere un índice que facilita las tareas de búsqueda. Será en este caso el índice principal o primario, el cual es obligatorio definir al crear la tabla; ello no obsta a la generación de otros índices denominados secundarios, lo cual se analizará en detalle en la sección 3 de este capítulo.

Validación Campos

Hemos visto que la especificación de una columna puede optativamente llevar lo que se denomina "cláusula de integridad" o validación de contenido del campo, cuyo formato más simple es NOT NULL.

Existen otras formas de validar la información, que son las cláusulas *in* , *in table* , *mask* y las validaciones con operadores relacionales. Las dos primeras responden a un concepto similar al de la cláusula *in* de la sentencia *where* : proporcionar una serie de valores elegibles. La diferencia está en que en este último caso se trata de los valores que dan lugar a que un registro se seleccione para la consulta, en tanto que en una cláusula de integridad se establece que son los únicos aceptables para su incorporación a la tabla; los demás son rechazados.

La cláusula de integridad *in* seguida de una lista de valores (que como ya se ha dicho debe ir encerrada entre paréntesis y con sus elementos separados por comas), permite especificar optativamente una descripción, separándola del valor del campo por dos puntos. Así por

ejemplo:

```
create table paises (
ctrycode char (3)
in ( "ARG":"Argentina",
"ESP":"España",
"MEX":"México",
"JAP":"Japón",
"USA":"Estados Unidos"),
......)
```

La tabla de países contiene, además de otros campos posibles que se se- ñalan con puntos suspensivos, un código de país (country code) de tres caracteres, que no puede ser otro que alguno de los cinco valores indicados.

Ahora bien: qué pasa si queremos agregar un valor a la lista, tal como 'URS':"Unión Soviética"?

Tendríamos que redefinir la tabla, es decir, volver a ejecutar un *create table* con la nueva lista, pero teniendo que repetir todas las especificaciones restantes. Esto no sería dificultoso si hemos tenido la precaución de guardar un archivo con la creación de la tabla, pero aún así el proceso insume una cierta cantidad de tiempo y trabajo que puede obviarse si cambiamos la cláusula de integridad en esta forma:

```
ctrycode char(3) in codpais:descr,
```

Ahora la lista de valores está contenida en una tabla llamada "codpais", que no es otra cosa que una tabla cada una de cuyas filas proporciona uno de los elementos de la lista.

De tal manera, al no estar los valores incorporados explícitamente al *create*, la cláusula *in table* permite que la modificación se logre simplemente corrigiendo la tabla auxiliar de validación sin necesidad de redefinir la tabla principal.

La cláusula *mask*

Veamos finalmente y en sección aparte esta cláusula de integridad, que tiene mayor grado de complejidad, aparte de ser una implementación propia del IQL.

Comenzaremos definiendo el concepto de máscara ("mask") como se utiliza ampliamente en diversas implementaciones propias de la computación, particularmente en la casi totalidad de los lenguajes. Una máscara se define como una cadena de caracteres , según la cual se puede especificar:

1. Que las posiciones dentro de un campo se toman tal como vienen, o bien surgen de una sustitución;

2. Que se acepta cualquier valor o sólo aquellos de un determinado tipo, y

3. Que una cierta posición pueda faltar (estar en blanco) o no.

A semejanza de la cláusula *in*, la máscara puede estar directamente especificada en la cláusula

-teniendo presente que ya no se trata de una lista, sino de una cadena de caracteres-, o bien aludir a una referencia externa que en este caso es una variable de ambiente. La misma se especifica precediendo su nombre por el signo "$", como por ejemplo:

```
nroserie char(10) descr, "Número de Serie"mask $serialnr
```

donde la variable ambiental `serialnr' contiene la cadena de carácteres que conforma la máscara para validar un número de serie que debe, por ejemplo, contener carácteres alfabéticos en ciertas posiciones y numéricos en otras.

Los distintos tipos de datos que pueden verificarse mediante la máscara se especifican con los códigos:

a, A Alfabético (en sentido estricto: no se aceptan blancos)

n, N Numérico (dígitos de 0 a 9; único aceptable para num)

x, X Alfanumérico.

h, H Numérico Hexadecimal.

Oculta el carácter.

La diferencia entre el uso de minúscula y mayúscula es que la primera permite la ausencia de un carácter en esa posición -vale decir, la aparición de un blanco-, mientras que en el segundo caso es obligatoria la presencia de un carácter que cumpla con la especificación.

La máscara indica la longitud del campo; vale decir, cada posición de aquella controla la posición correlativa de ésta última. No obstante, esto deja de ser aplicable cuando se utiliza alguno de los caracteres prefijables > (convertir a mayúsculas) o < (convertir a minúsculas , que n son válidos para el código "n" o "N"; o bien si se emplea un factor de repetición (3a en lugar de "aaa").

Si por ejemplo queremos que un campo de cinco posiciones sólo acepte tres caracteres alfabéticos obligatorios, los cuales deben convertirse a mayúsculas, seguidos de dos caracteres numéricos de los cuales sólo el primero es obligatorio, la máscara será:

3>ANn.

Fuera de los caracteres con significado especial, todos los demás se tomarán literalmente. Es decir, los campos date tienen implícita una máscara nN/NN/NN. Esto se menciona a título ilustrativo, ya que en realidad el sistema no permite indicar máscaras para campos *date*, *time* o *float*. Como ejemplo final, daremos las máscaras para números telefónicos en nuestro país y en EE.UU. de Norteamérica. En el primer caso será:

nNN-4N

Tres dígitos, dos de ellos obligatorios, separados por un guión de otros cuatro obligatorios. Para EE.UU. será en cambio:

(3N)3>X-4>X

Hay un código de área de tres dígitos que se encierra entre paréntesis, y luego una característica alfanumérica en la cual las letras deben convertirse a mayúsculas, separada por un guión del número de abonado. Todas las posiciones -diez en total- son obligatorias.

Validaciones con Operadores Relacionales

Los campos podrán tener asociados atributos de validación especificados mediante operadores relaciones. Estos atributos de chequeo, impedirán el ingreso de valores que no concuerden con lo indicado en la definición del campo, centralizando de esta forma, la consistencia de los datos. Su sintaxis genérica es:

```
check (expresión)
```

donde expresión es cualquier expresión conteniendo una o varias relaciones entre variables, campos y constantes.

```
cost num(6,2) descr "Costo"
check (this >0.0 and this<9000.0)
default 100.0
```

Como se observa en el ejemplo, el campo corriente puede ser referenciada por *this* o por su nombre. Otros campos pueden ser referenciados por sus nombres.

Atributo default

El significado de la palabra *default* es: valor por omisión; es decir, el valor que asignará el sistema ante la ausencia de una especificación concreta. Incluyendo este atributo en la definición de una columna que se agrega a una tabla, por ejemplo:

```
vigencia date not null default today,
precio num(7,2) not null default 0,
```

para incorporar a una lista de materiales el precio del proveedor y la fecha de vigencia del mismo, logramos evitar el rechazo. El valor por omisión para un campo cuando no se lo indica es siempre el valor nulo correspondiente a ese tipo de campo.

El atributo *default* es independiente de la cláusula de validación, si bien debe ser coherente con ella y con el tipo de campo. Como se puede observar en el ejemplo, los *default* también puede ser expresiones, incluyendo operaciones aritmética, comparaciones y etcétera, entre campos, variables y constantes.

Modificación y Supresión de Tablas

Una vez creada una tabla, cargados sus datos y habiendo comenzado a utilizarla, con el transcurso del tiempo suele ocurrir que se le deben introducir modificaciones. No nos referimos al hecho de agregar, quitar o alterar filas, sino a la necesidad de añadir o suprimir una columna, o bien modificar alguna de sus características.

Esto implica una modificación del diseño lógico de la tabla y como consecuencia también del esquema al cual ella pertenece. Para facilitar las alteraciones a un esquema, es conveniente haber almacenado, inmediatamente después de crearlo y de definir sus tablas, el conjunto de sentencias usadas para hacerlo.

El comando *store*

Para poder utilizar esta facilidad, es necesario crear un archivo que contenga las sentencias *use*, *create*, etc. destinadas a generar el esquema y las tablas deseadas. Luego de ejecutar dicho archivo en modo batch, y suponiendo que el esquema creado fuera el "epsilon", el comando:

```
store schema epsilon;
```

generará un archivo llamado "epsilon.sc" que contendrá las especificaciones correspondiente según el formato interno que utiliza el sistema.

Cuando surja la necesidad de alterar una tabla perteneciente a epsilon , se usará un editor --que puede ser Dali o el propio de IQL- para introducir las modificaciones en la definición de las columnas, en dicho archivo, y se repetirá el procedimiento. Esto es, se ejecutarán los comandos contenidos en el archivo.

En otras implementaciones del SQL, se emplea un comando alter --inexistente en IQL-- para introducir modificaciones en tablas o clusters, debiendo el programador introducir las alteraciones una a una. La novedad de IDEAFIX es la poder correr a través de todo el onjunto de comandos, con las siguientes ventajas:

• El programador tiene una completa visión de las definiciones de tabla y esquema, esto hace que los cambios sean consistentes con sí mismos y con el resto de las especificaciones (las que no se tocaron).

• El esquema completo es re-generado en un sólo paso, permitiendo al sistema hacer una verificación completa.

Las próximas secciones mencionarán algunos requisitos a en cuenta al modificar esquemas (y sus tablas).

Usando *create* para Modificar

En primer lugar remarcaremos que se entiende por *modificación* de una tabla el hecho de alterar su estructura lógica; ésto es, agregar o suprimir una o varias columnas, o bien cambiar su longitud (incluyendo la cantidad de posiciones decimales de un campo numérico, aún sin modificar el total de dígitos) o las reglas de validación.

Cuando se ejecuta un comando *create table* sobre una tabla ya existente, el sistema emite una advertencia ("warning") en tal sentido, para evitar que por una coincidencia accidental de nombres quien quiere en realidad crear una nueva tabla, modifique otra preexistente. Además,

el mensaje tiene otra función que es poner sobre aviso al usuario de la posible necesidad de modificar la variable "modify".

Las Variables *modify* y *force*

Estas variables controlan la posibilidad de modificar las estructuras de datos. La variable *modify* da el primer nivel de posibilidad de modificación. Su existencia (es decir si está definida, no importando cual es su valor) permite la alteración del esquema en sí, es decir, el agregado o la supresión de una tabla, En el nivel de tablas permite el agregado de nuevas columnas y la modificación de características de las mismas, salvo el cambio del tipo de dato o cualquier cambio de precisión que implique pérdida de información. No permite en cambio la eliminación de columnas de una tabla.

Para realizar operaciones tales como: suprimir o modificar el tipo de dato de una columna y cambiar la precisión de un campo con posible pérdida de información, existe la variable *force*. Lógicamente *force* también permite todos los cambios que permite la variable *modify*.

Para cambiar el estado de estas variable se utiliza el comando set :

```
set modify;
unset modify;
......
```

Alteración de Columnas la Cláusula de Integridad

Cuando se incremente la capacidad de una columna, ello no trae problemas con los datos ya existentes. Sí los hay en cambio cuando se la achica: los campos de caracteres pueden perder algunos en el extremo derecho, mientras que los numéricos pueden ver cercenados dígitos de orden superior, o sea en el extremo izquierdo. Es responsabilidad del usuario tener en cuenta esta posibilidad, pero normalmente no trae consecuencias en lo que hace a la integridad como norma de validación (obviamente sí en cuanto a la integridad entendida en sentido total).

La supresión de una columna acarrea evidentemente la pérdida total de los datos contenidos en ella, pero desde el momento que se la suprime es porque ya no se la considera útil. Esto tampoco incide sobre la cláusula de integridad.

Distinto es lo que ocurre cuando se agrega una nueva columna, o se modifica la propia cláusula de integridad. Esto puede generar incompatibilidades, particularmente cuando la indicación es *not null*, ya que toda columna agregada a una tabla preexistente es llenada por el sistema con valores nulos. Si la columna ya existía, pero tenía valores nulos en algunas filas, la modificación de la cláusula de integridad a *not nul*l plantea la misma incompatibilidad.

Para resolver este problema en la definición de la columna se puede agregar el atributo *default* de manera que al modificar o agregar la nueva columna esta no quede con valor nulo sino con el indicado en dicha cláusula.

La sentencia *create* es *drop* (descartar), que puede aplicarse tanto a esquemas como a tablas.

Su formato es:

```
drop schema nombresq;
```

ó

```
drop table nombretab;
```

según el caso. Nótese que el drop implica la pérdida total de la información contenida en el ente afectado, y además la desaparición de su definición como estructura lógica para el sistema. Si imaginamos tener un esquema llamado HELENO, dentro del cual se hallan definidas las tablas ALPHA, BETA y GAMMA, consideremos la sentencia:

```
drop table beta;
```

El objeto que puede tener este comando es liberar espacio en disco cuando la tabla BETA ha dejado de tener utilidad, sea porque se la deja de usar o bien porque ha sido sustituida por otra, ya sea incorporando sus datos a una tabla más amplia o eventualmente llevando la tabla completa a otro esquema. Recordamos que "heleno" debe ser el esquema corriente para la ejecución del comando. La sentencia:

```
drop schema heleno
```

tiene el efecto de eliminar directamente el esquema íntegro y junto con él a todas las tablas incluidas. Una consecuencia inmediata es que nos quedamos sin esquema activo, por lo cual, si queremos continuar trabajando en iql, será conveniente emitir de inmediato un *use*. Por ello podemos generalizar que, salvo el caso de ser lo último por hacer, todo *drop schema* debiera ir seguido de un *use* .

Breve Resumen

La modificación o supresión de esquemas y tablas es un tema delicado, que debe por consiguiente manejarse con máxima prudencia. Así por ejemplo, es factible editar directamente el file "esquema.sc" en lugar de hacerlo con el archivo correspondiente y luego ejecutarlo, pero no es recomendable.

La idea fundamental es la de reprocesar el file "esquema.sc" conforme a las nuevas especificaciones encontradas en las sentencias create o drop table, dejando como está lo no afectado y corrigiendo aquello que ha sido objeto de alteración.

Al modificar tablas hay que tener en mente las restricciones relativas a la variable "modify", y la coherencia necesaria entre los valores por default y la cláusula de integridad.

Los comandos *lock/unlock*

Una consideración final: que sucede con los demás usuarios mientras alguien (generalmente el dueño del esquema) está introduciendo modificaciones?

Distinguimos dos situaciones:

• Se están modificando estructuras de datos en el esquema, o bien

• se están insertando, borrando o modificando filas en alguna tabla.

En el primer caso nadie más que quien está realizando las modificaciones puede tener acceso al esquema; por ello, cuando se intenta realizar una modificación, se verifica que ningún otro usuario esté usando el esquema. Si esto es así, se procede a bloquear el esquema y realizar la modificación. Si hay otros usuarios accediendo se informará de tal situación con un mensaje que informa de la imposibilidad de bloquear el esquema para acceso exclusivo.

En cuanto a la modificación de datos, desde ya el sistema adopta ciertos recaudos mínimos para impedir que dos usuarios puedan tratar de actualizar la misma tabla al mismo tiempo, pero no inhibe la posibilidad de leerla. Esto significa que un usuario puede en determinado momento acceder a la versión vieja, justo antes de que sea actualizada. Para impedir esta situación, existe el comando *lock* que inhibe el acceso a los demás usuarios, ya sea a una tabla o a un esquema. Así pues:

```
lock schema heleno;
```

impedirá acceso a la totalidad del esquema nombrado, en tanto que

```
lock table alpha;
```

hace lo propio exclusivamente con esa tabla. Para liberarlos y facilitar el acceso a los demás usuarios, lo cual corresponde hacer al incluir las modificaciones, se usa el comando unlock con idéntica sintaxis.

Indexación

El índice de una tabla, y más generalmente de cualquier archivo o base de datos, cumple las mismas funciones que las de un libro o manual. Su objeto principal es el de ubicar la información con mayor rapidez. a.

Creación Indices

El comando para crear un índice es:

```
create index nombre_indx on tablname (column_list);
```

El nombre del índice, que hemos simbolizado como "indxname", debe responder a las mismas normas ya establecidas para los nombres de esquemas y tablas. A los fines prácticos, es conveniente adoptar alguna convención que permita recordar fácilmente los nombres de los índices a partir del nombre de la tabla a la cual indexan, v.gr. empx , empx2, etc. podrían ser los índices de la tabla emp. Pero en lo que hace al sistema la denominación es completamente libre.

La *lista de columnas* permite que la clave, es decir el campo de búsqueda, esté formado por varios elementos distintos de información. En principio, tales componentes están consitituidos por el valor completo de la columna. Sin embargo, cuando se trata de campos de caracteres, es decir definidos como char, es posible usar la función que define una subcadena: campo(10) para tomar los diez primeros caracteres solamente campo(3,8) para ignorar los tres primeros y tomar los ocho siguientes

El atributo unique

Puesto que podemos definir cuantos índices queramos sobre una tabla, y elegir como clave cualquier campo dentro de ella, es natural que aparezcan repeticiones. Esta situación, que se denomina "clave duplicada" (duplicate key), implica que dos o más filas contienen idéntico valor para una cierta columna -o combinación de ellas- que ha sido definida como clave.

Tal circunstancia puede no ser aceptable, en este caso IQL incluye el comando unique, que debe ser añadido a la clave primaria (primary key), o al crear el índice (ej., `create unique index ...`).

Otros atributos de las Claves

Hasta aquí hemos dado por supuesto que la indexación de una tabla se hacía por el orden ascendente de la o las columnas que integraban la clave. Esto es un típico default de IQL, pero puede modificarse especificando a continuación del nombre de una columna dentro de la lista, el atributo *desc*, y también se podrían excluir los campos nulos con la condición *not null*.

Veamos el siguiente ejemplo:

```
create index promind on emplead
(feprom desc not null);
```

Especificación de Indices en *create table*

Vimos que despúes de las especificaciones de columnas, se podía indicar una clave primaria al momento de crear una tabla. El formato general de tal opción es:

```
primary key (column_list)
```

Pero también se puede incluir al final de la sentencia *create table* la cláusula:

```
index indxname (column_list)
```

además de la cláusula primary key .

Supresión de Indices

Se efectúa mediante el comando:

```
drop index indxname on tabname;
```

Es la menos delicada de las opciones del drop , ya que el índice es una estructura separada de la tabla y está en un archivo aparte, su eliminación no produce ningún efecto físico sobre la información propia de la tabla.

Consideraciones Finales

No siempre es provechoso indexar una tabla, pero seguramente lo será si contiene al menos varios centenares de registros. Crear índices para una tabla con menos de cien registros es una pérdida de espacio y de tiempo: en la mayoría de los casos hará que la respuesta empeore en lugar de mejorar.

No suele ser conveniente tampoco crear demasiados índices para una misma tabla: dos o tres debieran ser suficientes si están adecuadamente elegidos. Hay que tener en cuenta que, a cambio de acelerar los queries, una gran cantidad de índices hace más lentos los procesos de *insert*, *update* y *delete*, debido a la necesidad de actualizar los índices afectados por esas columnas aparte de la modificación sufrida por la tabla en sí.

El campo que debería ser usado como clave es, generalmente, el usado por la cláusula *where*.

3

Apéndices

Apéndice A

Teclas de Función de IDEAFIX

Teclas de Función de IDEAFIX	Nombre
Descripción	**COMANDOS**
PROCESAR	Procsar dato
FIN	Fin del proceso corriente
REMOVER	Remover dato
IGNORAR	Ignorar
PAGINACION Y MOVIMIENTO DEL CURSOR	INGRESAR CURSOR_UP CURSOR_LEFT CURSOR_RIGHT CURSOR_DOWN PAG_ARR PAG_IZQ PAG_DER PAG_ABJ
Ingresar una l'nea de dato Mover cursor arriba Mover cursor izquierda Mover cursor derecha Mover cursor abajo	**EDICION**

Mover una página arriba Mover una página izquierda Mover una página derecha Mover una página abajo	
RETROCESO INSERTAR ELIMINAR BORRA_CAMPO TAB DESHACER	Mover hacia atráz y borrar Insertar un carácter Eliminar un carácter Eliminar un campo Mover hasta la próxima tabulación Ignorar el último cambio
AYUDA	AYUDA_APL AYUDA
Ayuda de la aplicación Ayuda sensitiva al contexto	**MANEJADOR DE VENTANAS**
REDIBUJAR ATENCON CONF_TECL SUSPENDER INTERRUMPIR	Redibujar la pantalla Servicios del WM Mostrar la configuración de teclado Suspender la tarea corriente Cancelar la tarea corriente
MICELANEAS	META CTRLX AF_1 AF_2 AF_3 AF_4
Buscar una cierta información Prefijo de tecla en el editor Función de aplicación 1 Función de aplicación 2 Función de aplicación 3 Función de aplicación 4	

Apéndice B

Conjunto de Caracteres de IDEAFIX

Referencia: primer número: octal; segundo número: decimal, tercer número: hexadecimal.

200 128 80	**PROCESAR**	220 144 90	**ELIMINAR**	240 160 a0	³	260 176 b0	º	300 192 c0	**Ë**	320 208 d0	**Á**	340 224 e0	ˆ	360 240 f0	**À**
201 129 81	**FINBORRA_CAMPO**	221 145 91		241 161 a1	**Ä**	261 177 b1	**Í**	301 193 c1	**ç**	321 209 d1	**„**	341 225 e1	‡	361 241 f1	–
202 130 82	**REMOVEAYUDA_AP**	222 146 92		242 162 a2	**À**	262 178 b2	**È**	302 194 c2	**å**	322 210 d2	**ñ**	342 226 e2	‰	362 242 f2	˜
203 131 83	**IGNORARAYUDA**	223 147 93		243 163 a3	**Á**	263 179 b3	**Ê**	303 195 c3	**Ì**	323 211 d3	**î**	343 227 e3	‹	363 243 f3	—
204 132 84	**INGRESAR**	224 148 94	**REDIBUJAR**	244 164 a4	**Ù**	264 180 b4	¼	304 196 c4	**€**	324 212 d4	**ï**	344 228 e4	**Š**	364 244 f4	™
205 133 85	**CURSOR_ARR**	225 149 95	**ATENCION**	245 165 a5	´	265 181 b5	¹	305 197 c5	#	325 213 d5	**Í**	345 229 e5	**Ú**	365 245 f5	›
206 134 86	**CURSOR_ABA**	226 150 96	**CONF_TECLA**	246 166 a6	**Å**	266 182 b6	**Î**	306 198 c6	®	326 214 d6	**…**	346 230 e6	¾	366 246 f6	**š**
207 135	**CURSOR_DER**	227 151	**IMPRIMIR**	247 167	**Ã**	267 183	**Ì**	307 199	**’**	327 215	°	347 231	#	367 247	**Ö**

87		97		a7		b7		c7		d7		e7		f7	
210 88	**CURSOR_IZQ**	230 98	**SUSPENDER**	250 a8	**Ú**	270 184 b8	**É**	310 200 c8	**é**	330 216 d8	**¯**	350 232 e8	**#**	370 248 f8	**¿**
211 89	**PAG_ARR**	231 99	**AF1**	251 169 a9	**Â**	271 185 b9	**Ë**	311 201 c9	***f***	331 217 d9	**ô**	351 233 e9	**#**	371 269 f9	**#**
212 8a	**PAG_IZQ**	232 9A	**AF2**	252 170 aa	**¿**	272 186 ba	**»**	312 202 ca	**æ**	332 218 da	**ò**	352 234 ea	**#**	372 270 fa	**ú**
213 8b	**PAG_ABJ**	233 9B	**AF3**	253 171 ab	**·**	273 187 bb	**÷**	313 203 cb	**è**	333 219 db	**ó**	353 235 eb	**‘**	373 271 fb	**#**
214 8c	**PAG_DER**	234 9C	**AF4**	254 172 ac	**-**	274 188 bc	**Ï**	314 204 cc	**í**	334 220 dc	**†**	354 236 ec	**“**	374 272 fc	**Ÿ**
215 8d	**RETROCESO**	235 9D	**AF5**	255 173 ad	**¯**	275 189 bd	**¶**	315 205 cd	**ê**	335 221 dd	**Å**	355 237 ed	**’**	375 273 fd	**Æ**
216 8e	**EDITAR**	236 9E	**AF6**	256 174 ae	**¬**	276 190 be	**Ç**	316 206 ce	**ë**	336 222 de	**×**	356 238 ee	**”**	376 274 fe	**Ê**
217 8f	**INSERTAR**	237 9F	**AF7**	257 175 af	**®**	277 191 bf	**Ñ**	317 207 cf	**ì**	337 223 df	**§**	357 239 ef	**•**	377 275 ff	**Ø**

Apéndice C

Variables de Ambiente

Variables de Ambiente IDEAFIX

Muchas de las características del entorno en el cual se ejecutan los programas de aplicación se pueden configurar a través de mecanismo de variables de ambiente provisto por el sistema operativo.

Brindan la comodidad de determinar en forma flexible y rápida ciertos parámetros. A continuación se muestra una lista de las variables usadas por IDEAFIX y su significado. Las marcadas con un (*) deben existir obligatoriamente para ejecutar cualquier programa:

PATH Indica para IDEAFIX una lista de directorios donde buscar archivos de tipo ".fmo", ".rpo", ".mn" y subdirectorios con nombre cap, map y hlp. El valor que se asigna es una sucesión de directorios separados con un carácter separador (":" en UNIX y ";" en DOS).

IDEAFIX Indica el directorio raíz de **IDEAFIX** y será utilizado por los utilitarios para ubicar los archivos que necesiten para su ejecución. Por ejemplo el utilitario *cfix* necesita los archivos de **include** de **IDEAFIX** que se encuentran el directorio:

```
$IDEAFIX/include
```

LANGUAGE Indica el lenguaje que se utilizará con **IDEAFIX**. Actualmente están disponibles:

HOME Indica el directorio HOME del usuario. En este directorio se encontrará el menú de servicios de **IDEAFIX** para cada usuario.

printer (*) Indica dónde y cómo se debe enviar la salida por impresora de los programas. Su valor consiste en dos campos separados por el carácter ":" (dos puntos). El primer campo puede contener la letra "F" o la letra "P". La primera, indica que el valor del segundo campo es el nombre de un archivo donde se debe grabar la salida. La letra "P" indica que el segundo campo es el nombre de un proceso UNIX. La salida del programa se enviará a la entrada estándar de dicho proceso. En el nombre del proceso se puede poner la indicación "el

programa. Para la definición en DOS consultar el Apéndice E.

TERM (*) Indica para **IDEAFIX** el nombre del modelo de terminal del usuario.

dbase Indica un directorio bajo el cual se crearán los esquemas. Ejemplo:

```
dbase=/usr/ideafix/datos
```

SHELL Indica el nombre del programa intérprete de comandos. Será usado por el Window Manager cuando se pida el Servicio de Ingresar al Shell.

empresa Desplegado por el utilitario menú en la línea de información de la pantalla.

NFILES Está dividida en dos partes separadas por el carácter ":" (dos puntos). La primera parte indica el número de archivos que un programa puede tener abiertos simultáneamente. Si no existe se asume por defecto 12. La segunda, es el número de archivos virtuales que puede tener abierto el manejador de Bases de Datos. En este caso si no se especifica se asumen 100, tanto en UNIX como en MS-DOS. Ejemplo:

```
NFILES=10:15
```

SCHEMA_SUFFIX Esta variable permite especificar el nombre del sufijo de un esquema. Cada vez que se abra o se cree un esquema (A travéz de la funciones OpenSchema, CreateSchema o FindSchema) al nombre de ese esquema se le agregará el sufijo indicado por esta variable, y el nombre resultante será usado con todas las funciones.

DBTAR Se especifican las opciones relativas a los utilitarios *exp* y *imp*, que vaya a utilizar el comando *tar*. Ejemplo:

```
export DBTAR = '-t –g'
```

En el ejemplo se especifica en la variable de ambiente las opciones "-t" y "-g". Si el comando tar ejecuta el utilitario exp o imp, lo hará usando estas dos opciones.

MULTI Esta variable define operación en red multiusuario.

Definición del Ambiente

Generalmente las variables de ambiente de cada usuario se definen en un archivo, y se define su valor antes que el usuario comience la ejecución de un programa. Según el sistema operativo utilizado se tendrán las siguientes características:

UNIX

Lo usual es emplear un archivo leído por el intérprete de comandos cuando el usuario realiza el login. Puede ser *.profile* (para Bourne y Korn Shell) o bien *.login* (para C Shell). Notar que las variables PATH, TERM y SHELL son utilizadas por otros utilitarios de UNIX o por el intérprete de comandos. Ejemplo (para Korn Shell):

```
set export IDEAFIX=/usr/idea
set LANGUAGE=spanish
set PATH=":$IDEAFIX/bin:$PATH:/usr/dos/bin"
set SHELL=/bin/ksh
set dbase=/usr/idea/dbase/
set empresa="InterSoft Argentina S.A."
set printer=F:x.pr
```

Se supone que las variables TERM y empresa han sido definidas por un archivo leído antes del `.profile` del usuario (en este caso `/etc/profile`) ya que estos parámetros son genéricos y no dependen de cada usuario.

MS-DOS

Las variables de ambiente se definen con el comando 'set' del intérprete de comandos. Notar que la variable PATH es usada por el intérprete de comandos de DOS.

Puede crearse un procedimiento que las defina en un archivo que llamaremos por ejemplo setenv.bat:

```
set PATH=;c:\edos\ebin;c:\eusr\eideafix\ebin
set IDEAFIX=\eusr\eideafix
set LANGUAGE=spanish
set HOME=\eusr\eideafix
set SHELL=command
set PRINTER=F:x.pr
set TERM=ibmpc
set DBASE=c:\eusr\edatos\e
set EMPRESA=InterSoft
set MULTI=0
set NFILES=15:200
```

IDEAFIX en MS-DOS brinda la posibilidad de definir variables de ambiente en un archivo, para evitar el problema de la falta de espacio para variables de ambiente. Consultar el Apéndice E.

Apéndice D

Capacidades de Terminal en UNIX

Capacidades de Terminal

Estas son las capacidades utilizadas en los sistemas UNIX y compatibles.

Descripción	Cupname
Movimiento del cursor	cup
Borrar hasta fin de línea	el
Borrar pantalla	clear
Insertar una línea	il1
Borrar una línea	dl1
Definir región de scroll	csr
Scroll inverso	ri
Inicializar la terminal	is1
Hacer cursor invisible	civis
Hacer cursor muy visible	cvvis
Hacer cursor visible	cnorm
Atributo ALTERNATIVO	smacs
Atributo TITILANTE	blink
Atributo RESALTADO	bold
Atributo INVERSO	rev
Atributo SUBRAYADO	smul
Atributo STANDARD	smso
Atributo NORMAL	sgr0

Variables Booleanas

hz - No puede imprimir circunflexiones

Números

cols - número de columna

lines - número de línea

xmc - espacio dejado en blanco por el atributo de *video reverso*

Ver la sección de TERMINFO(4) del *Manual de Referencia del Programador de UNIX*, o el equivalente para su instalación. Se incluyen un grupo de archivos en el directorio “tic” con el sistema de ejecución de IDEAFIX. Estos archivos (con extensión “.tic”) contienen descripciones de terminfo ya probadas en IDEAFIX.

Apéndice E

Interfaz con el SPOOL de UNIX

Con el sistema de ejecución de IDEAFIX se entrega un archivo denominado pmintrf.sh que es un procedimiento shell que puede ser instalado como interfaz para el sistema spooler "lp" de UNIX. Los comandos necesarios para hacerlo son:

```
$ /usr/lib/lpadmin -pimpre -ipmintrf.sh
```

impre es el nombre de la impresora a la que se desea agregar el programa de interfaz. Conviene que la impresora haya sido previamente creada con este mismo comando (si no es as', se efectuará la generación, pero los parámetros asumidos pueden no ser los correctos para su sistema). Para más información sobre el *lpadmin*, consultar la documentación en la página LPADMIN(1M) del *User's Reference Manual* de UNIX.

El archivo *pmintrf.sh* está comentado de modo de explicar las operaciones que realiza. De todos modos agregamos que las opciones "-o" que se indiquen con el comando "lp" serán recibidas por el PM.

Apéndice F

Mensajes de Error y Advertencia

Se detallan aquí los distintos tipos de mensajes susceptibles de ser emitidos por el sistema. Se clasifican en:

- Mensajes de Error (Errores en Tiempo de Compilación y de Ejecución)
- Mensajes de Advertencia.

Mensajes de Error

Estos mensajes se pueden dividir en dos grandes grupos: aquellos que son detectados durante la compilación, y los que se presentan al momento de la ejecución.

Mensajes en Tiempo de Compilación

El rango de nœmeros dado para estos errores es del 1 al 500. Los casos que pueden detectarse son los siguientes:

1 Error de Sintaxis.

Este mensaje indica que ha habido un especificación incorrecta en una sentencia.

Por ejemplo: falta el punto y coma (";") al final de la sentencia; no se intercaló una coma como separador de campos; falta alguna especificación en la cláusula (p.ej. "from"); palabra clave mal escrita (v.gr. were por where), etc.

2 Carácter octal inválido %o .

Existe un carácter que es ilegal en esa posición, o bien es un carácter de control.

3 Cadena de caracteres sin cerrar.

Este mensaje aparecerá cuando se especifique un "string" de caracteres al cual le falta la indicación de cierre, ya sea " ' " or " " ". Por ejemplo: el nombre de una columna se especifica como ["Nombre del Empleado] faltan las comillas dobles se–alando el fin de la cadena.

4 Archivo no encontrado.

Se intenta leer un archivo inexistente con la sentencia *exec*, o bien no existe el archivo especificado en la línea de comandos.

Recomendación: verificar que el nombre esté correctamente deletreado. "Clientes" no es lo mismo que "clientes", ni "Deptos" igual a "Dptos".

5 Tabla de Macros excedida.

Se han ejecutado demasiadas sentencias define .

6 Sentencia errónea.

El error debiera ser lo suficientemente grueso como para ser detectado a simple vista.

7 Falta la especificación de clave primaria (PRIMARY KEY).

No se ha establecido el campo de clave para acceso básico a la tabla.

8 Error de definición de Macro.

9 Redefinición de la CLAVE PRIMARIA.

El atributo "primary key" ha sido definido para más de un campo, o bien se ha agregado una especificación de índice primario inconsistente.

10 Combinación no soportada de condiciones mœltiples de verificación.

Especificación de más de una validación como atributo de campo.

11 Valor %s incompatible con el tipo de campo.

Se especifica un valor que no corresponde al tipo de campo en cuestión.~ V.gr.: una fecha que contiene caracteres no numéricos.

12 Longitud (%d) excede valor de longitud del campo (%d).

Se intenta asignar un valor por defecto cuya longitud es mayor que la del campo.

13 Valor con demasiadas (%d) posiciones enteras. Máx.: %d

Se especifica un valor con mayor cantidad de posiciones enteras que las definidas.

14 Valor con demasiadas (%d) posiciones decimales. Máx.: %d.

Se especifica un valor con mayor cantidad de posiciones decimales que las definidas.

15 Fecha (DATE) inválida: %s.

Se ha indicado una fecha no válida (por ejemplo: 29/02/90).

16 Hora (TIME) inválida : %s.

Se ha indicado una hora no válida (por ejemplo 15:69:42).

17 Tabla %s no encontrada.

Se ha referencia en una operación a una tabla no definida.

18 Tabla %s no declarada.

Se ha mencionado una tabla (por ejemplo, en una validación in table) que no pertenece al esquema corriente.

19 Campo no válido.

Se hace referencia un campo inexistente. Como en el caso anterior y otros análogos, verificar que el nombre esté correctamente deletreado.

20 Cantidad errónea de valores.

21 Campo '%s' no definido.

22 Tabla '%s' an '%s' no definida.

23 Uso incorrecto del Alias '%s' en '%s'.

24 Redefinición del Alias '%s'.

Se ha intentado redefinir un alias en la sentencia from .

25 Variable ambiental '%s' indefinida.

Se ha hecho referencia a una variable de ambiente que no está definida en el momento de la compilación.

26 Nœmero de dígitos no debe exceder de 15.

Se ha definido un campo numérico con más de 15 dígitos (se debe computar la totalidad: cantidad de enteros más posiciones decimales).

Mensajes en Tiempo de Ejecución

La numeración de este grupo de mensajes comienza con el Nro. 501, y ellos son:

501 Sentencia no válida el modo corriente.

Se intenta ejecutar una sentencia de manipulación de datos en el utilitario dgen .

502 Esquema '%s' no encontrado.

Se ha hecho referencia en una operación a un esquema no activo (o incorrectamente deletreado).

503 Tabla '%s' no enconttrada.

Una operación ha hecho referencia a una tabla no definida, o no perteneciente a ninguno de los esquemas activos.

504 Campo '%s' no encontrado.

Se ha hecho referencia en una operación a un campo no definido.

505 Indice '%s' no encontrado.

Se ha hecho referencia en una operación a un índice no definido.

506 No se pudo abrir el esquema '%s'.

Este error se puede producir si el usuario:

¥ Hace referencia a un esquema inexistente.

¥ No tiene permisos de acceso al esquema.

¥ Alude a un esquema que sí existe pero está corrupto.

¥ Intenta usar un esquema bloqueado para uso exclusivo de otro usuario.

507 Falló operación de grabación de definición del esquema '%s'.

En la sentencia store schema , no se pudo abrir el archivo de salida donde se debía grabar la definición.

508 No se pudo abrir archivo HEADER para esquema '%s'.

Mismo caso del mensaje anterior para sentencia store schema header .

509 No se pudo crear la Tabla '%s'.

510 No se pudo crear el Indice '%s'.

511 No se pudo crear el Esquema '%s'.

512 No puede agregarse el Campo '%s'.

Los errores anteriores son susceptibles de producirse al intentar crear o modificar un esquema.

513 Cantidad errónea de campos clave en la cláusula ' in '.

Se ha definido la clave de acceso en un atributo " in " con mayor cantidad de campos que la correspondiente.

514 Demasiadas columnas de salida.

Este error se produce cuando se intenta incluir un nœmero excesivo de campos en la grilla de salida, lo que lleva a superar la capacidad de manejo del IQL.

515 Este nœmero de mensaje no ha sido usado.

516 No puede crearse el esquema '%s': ya existe.

Se intenta crear un esquema dándole el nombre de otro ya generado con anterioridad.

517 No puede crearse la tabla `%s' : ya existe.

Similar al mensaje anterior.

518 No es posible convertir tabla '%s' : campo '%s' carece de equivalente.

Al intentar modificar una tabla, la nueva especificación omite un campo previamente existente.

519 No es posible convertir tabla '%s' : campo '%s' tiene diferentes tipos.

En la modificación de una tabla, un campo ya existente está siendo especificado de distinto tipo.

520 No puede convertirse tabla '%s' : campo '%s' tiene menor longitud.

Auto-explicativo. Ello implicaría pérdida de información.

521 No puede convertirse tabla '%s' : campo '%s' tiene menos decimales.

Similar al anterior. La menor cantidad de decimales implica pérdida de precisión.

522 No puede convertirse tabla '%s' : campo '%s' tiene menos ocurrencias.

El campo es cuestión es repetitivo (vector), y la nueva cantidad de elementos es menor que la original.

523 No puede convertirse tabla '%s' : campo '%s' tiene diferente máscara.

Al modificar una tabla, se intenta asignar a un campo una máscara diferente a la que ya tenía.

524 Nœmero de mensaje no utilizado.

525 No puede borrarse (drop) '%s'.

526 No pudo crearse una tabla temporaria.

527 Redefinición del Alias : %s.

Se ha intentado redefinir un alias en la sentencia from .

528 Demasiados Alias.

529 No se pudo crear un Alias.

530 Fecha (DATE) no válida : %s.

Se ha indicado una fecha errónea (por ejemplo: 31/06/90).

531 Hora (TIME) no válida : %s.

Se ha indicado una hora no válida (por ejemplo: 25:32:57).

532 La función `restó (Remainder) no puede aplicarse a nœmeros no enteros.

Se ha aplicado la función *rem* a un campo con decimales.

533 La operación '%s' no es válida para el tipo '%s'.

534 Operación '%s' : Tipos de dato incompatibles '%s' y '%s'.

Se está realizando una operación entre tipos de dato no compatibles, como por ejemplo dividiendo una fecha por nœmero entero (sí es lícito sumar o restar un valor entero a una fecha, que será interpretada como una cantidad de días de corrimiento). También puede ocurrir cuando no se colocan comillas para encerrar la fecha.

535 No puede cambiarse el tipo de variable .

536 El valor de la longitud (%d) excede la del campo (%d).

537 Inserción cancelada. Registro ya existente.

Se intentaba dar de alta un registro cuya clave ya figura en el índice.~ Probablemente se trate de una modificación; o bien se debe corregir alguno de los valores que componen la clave, para diferenciarla de la existente.

538 No puede bloquearse el esquema '%s'.

El sistema no está en condiciones de bloquear "LOCK") el esquema indicado, pues está siendo utilizado por otro proceso.

539 No pudo abrirse el esquema '%s' : está bloqueado por otro proceso.

Para poder usar el esquema, es necesario esperar a que el proceso que lo tiene bloqueado lo libere; mientras tanto, no puede ser abierto.

540 Campo %s no cumple la condición NOT_NULL (no nulo).

El valor del campo es nulo pese a existir una especificación de que tal condición no es válida. Debe asignarse un valor, así fuere cero o blanco.

541 Campo: %s, Valor: %s no cumple condición IN .

El contenido del campo no figura dentro del conjunto de valores definidos en la cláusula IN. Si se requiere incluir nuevos posibilidades, es preciso redefinir la lista de códigos o datos admisibles.

542 Campo %s, Valor: %s no cumple condición NOT IN (no en).

El contenido del campo está incluido en el conjunto de valores prohibidos que define la cláusula NOT IN .

543 Campo %s, Valor: %s no cumple condición IN TABLE (en tabla).

El valor del campo no está incluido en el conjunto de datos contenidos en la tabla referida en la cláusula IN TABLE . Se el valor fuera efectivamente correcto, debe actualizarse previamente la tabla en cuestión para que sea aceptado.

544 Campo %s, Valor: %s no cumple condición NOT IN TABLE (no en tabla).

Caso simétrico al anterior. La tabla contiene ahora los valores prohibidos .

545 Campo %s, Valor: %s no cumple la condición BETWEEN (entre).

El valor del campo no está comprendido en el rango de valores aceptables definidos por el atributo BETWEEN.

546 Campo %s, Valor: %s no verifica la condición NOT BETWEEN.

Similar al caso anterior, sólo que ahora el rango no es de validez sino de exclusión.

547 Campo %s, Valor: %s no verifica la condición EQUAL (igual).

El valor del campo no corresponde a lo definido en la condición EQUAL.

548 Campo %s, Valor: %s no cumple condición NOT EQUAL (no igual).

El valor del campo es igual al definido en la condición NO IGUAL.

549 Campo %s, Valor: %s no cumple condición GREATER THAN (mayor que).

550 Campo %s, Valor: %s no cumple condición LOWER THAN (menor que).

551 Field %s, Value: %s no cumple la condición GREATER or EQUAL (mayor o igual a).

552 Campo %s, Valor: %s no cumple condición LOWER or EQUAL (menor o igual a).

553 %s : La Tabla %s no existe.

554 %s : Los nombres de Tabla deben diferir.

No pueden existir definiciones de tablas con el mismo nombre.

555 %s : La Tabla %s ya existe.

556 Las nuevas tablas (como '%s') deben agregarse al final del esquema.

Si se modifica un esquema por adición de tablas, éstas œltimas deben agregarse al final del esquema en caso de modificarlo.

557 Tabla '%s' fuera de secuencia.

558 No hay esquema corriente.

559 %s : Los nombres de campo deben diferrir.

560 %s : El Campo %s ya existe.

561 No puede borrarse el Directorio '%s' .

562 Falló operación de borrado (Drop) para el esquema '%s'.

563 La cláusula Having requiere especificar la cláusula group by .

Siempre que quiera usarse la cláusula ' having ', deberá especificarse conjuntamente ' group by '.

564 Cláusula 'Order by' inválida para 'one-row-select' (fila œnica).

565 Expresión nœmero %d tiene diferente nivel que la precedente.

566 Expresión de campo en 'one-row-select' (fila œnica).

567 Este nœmero de mensaje de error no fue utilizado.

568 Nivel erróneo de expresión para cláusula 'having' .

569 Nivel erróneo de expresión en cláusula 'order by' .

570 Expresión nœmero %d no incluida en cláusula 'group by' .

571 Demasiados niveles de funciones agrupadas en expresion %d.

572 Expresión con subíndice para campo œnico '%s'.

573 Subíndice fuera de rango para Campo '%s' Valor %d.

574 Descripción inválida de operando para campo '%s'.

575 Imposible convertir tabla '%s': campo '%s' no nulo y carece de default.

En una conversión de tabla, se está asignando el atributo "not null" a un campo que no lo tenía con anterioridad. Esto es válido solamente si se especifica un valor por defecto (default value) para sustituir a los posibles valores nulos preexistentes.

576 La Table '%s' no puede ser bloqueada.

El sistema no está en condiciones de bloquear ("LOCK") la tabla en cuestión, pues la está utilizando otro proceso.

577 Esquema '%s' bloqueado.

Para poder usar el esquema, es necesario esperar a que el proceso que lo tiene bloqueado lo libere; mientras tanto, no puede ser usado.

578 Esquema '%s' creado por una CPU incompatible.

Existen diferencias insalvables de hardware entre el equipo que generó la Base de Datos y el que ahora pretende procesarla.

Mensajes de Advertencia

Comentarios sin cerrar al fin de la entrada.

El final de la sentencia involucra el cierre de cualquier ítem que aœn estuviera abierto, lo que obviamente incluye los comentarios. Pero cualquier otro tipo de cláusula no cerrada dará lugar a error , y no a warning .

Cláusula "in table" ignorada.

Si la tabla aludida no existe, la pretensión de usarla como verificación no da lugar a error (se crea igualmente el esquema), pero se omite la cláusula de validación ÔIn tableÕ .

Expresión de campo '%s' no incluida en la cláusula 'group by'.

Para efectuar agrupamientos, es necesario indicar qué papel cumple cada uno de los campos seleccionados (incluidos en la sentencia select). En caso contrario, el SQL no sabe qué hacer con ellos: no se acumulan, y tampoco cortan control.

Apéndice G

Sistema de Control de Accesos a Aplicaciones Ideafix (Ixsystem)

Introducción al sistema de control de accesos

Objetivos y alcance del sistema

Ixsystem es una herramienta destinada a administrar el control de acceso de los usuarios a los diferentes componentes de un sistema desarrollado en Ideafix. Mediante este sistema se puede controlar el modo de operación permitido a cada perfil de usuario sobre cada unidad funcional presente en un sistema administrativo. Por ejemplo:

Sobre el programa pgm.exe

• el perfil A tiene permiso para realizar cualquier operación.

• el perfil B tiene permiso solo para dar altas.

• el perfil C no tiene permisos.

• el perfil D tiene solo permiso de efectuar consultas.

Con Ixsystem solo se controlan los accesos a las unidades funcionales (programas, etc), no se controlan accesos a las entidades de la base de datos.

Composición

Usuarios

Representan las personas que interactúan con los programas que integran la aplicación Ideafix sobre la que se controlarán los accesos.

Grupos

Conjuntos que agrupan usuarios que comparten un perfil común de acceso a las unidades funcionales. Llamamos perfil de acceso al conjunto de permisos que tiene un grupo sobre todas las unidades que componen la aplicación.

Unidades

Son componentes de la aplicación a ser accedidas con el proposito interactuar con el sistema. La mas común de las unidades es el programa pero tambien los forms y los menues son unidades. Sobre las unidades se definirán los permisos.

Permisos

Control de acceso a verificarse al ser accedida determinada unidad por un usuario perteneciente a un grupo arbitrario. A cada unidad y para cada grupo se puede otorgar uno de los siguientes permisos:

• Cualquier operación

• Sin permisos

• Solo consultas

• Solo altas

• Solo bajas

• Solo modificaciones

• Modificaciones y bajas

• Modificaciones y altas

• Altas y bajas

Cómo utilizar el systema de control de accesos

Administración de usuarios y grupos

Los usuarios de un sistema desarrollado con Ideafix sobre los que se pretende efectuar algún tipo de control de acceso deben agruparse según sean los perfiles que la empresa define en función de sus responsabilidades. Una vez definidos estos grupos, deben registrarse en la base

de datos de Ixsystem (esquema perms.sc). Esto es darlos de alta mediante el ABM de grupos donde debe especificarse una descripción y si el perfil tiene o no permisos por defecto. Que el grupo tenga permisos por defecto significa que tiene accesos a todas las unidades del sistema salvo aquellas a las que se le ha restringido especificamente. Si el grupo no tiene permisos por defecto pasa exactamente lo contrario.

Finalizado el registro de los grupos pueden darse de alta los usuarios en cada perfil utilizando el ABM de usuarios.

```
   [img00.bmp] [img10.bmp]
```

Generación de unidades

Cada unidad funcional debe ser dada de alta en la base de datos para poder administrar accesos sobre la misma. Como esta tarea puede resultar tediosa, se dispone de un utilitario que permite registrar las unidades automáticamente. Si se ejecuta la opción de menú correspondiente al generador automático se despliegará un formulario donde deben especificarse los parámetros necesarios para que ésta herramienta cumpla su objetivo. Lo primero que se requiere es el menú principal del módulo al que se le administrarán los accesos. Lo siguiente es el número de sistema, cuya información es solamente descriptiva. Luego debe especificarse el número de módulo al que pertenece el menu principal registrado en primera instancia. Este número es particularmente importante porque determina el código de la primera unidad del módulo a ser registrada (ver ejemplo). El último parámetro indica si se desea limpiar todas las unidades y los permisos registrados con anterioridad o solo los comprendidos en el rango perteneciente al módulo en cuestión. Por ejemplo, supongamos que tenemos un sistema que se compone de tres módulos o subsistemas. El menú principal del sistema es sistema.mn y los menues principales de cada módulo son: modulo1.mn, modulo2.mn y modulo3.mn. Asumamos tambien que el número que identifica al sistema es el 902 y los números de los módulos son 903, 904 y 905 respectivamente. Es posible generar las unidades de una sola vez ingresando los siguientes parámetros en el formulario del generador:

Menú Padre : sistema.mn

Número de Sistema : 902

Número de Módulo: 902

Desea borrar los registros previos: Si

[img20.bmp]

Pero ésta no es una buena práctica. La explicación es la siguiente: El número de módulo multiplicado por 100.000 determina el código de la primera unidad registrada y a partir de ese valor se registrarán las otras a medida que se recorren los menues en profundidad incrementando en uno el código en cada alta de unidad. Si el módulo uno cuenta con 203 unidades, el módulo dos con 157 y el módulo tres con 321, las 681 unidades se habrán registrado mezcladas en el rango 90.200.000 al 90.200.680.

Lo conveniente es ejecutar el generador una vez por cada módulo de manera de separar los rangos de códigos. De esta forma es posible generar una y otra vez las unidades de un módulo sin tener que afectar (borrar), los datos cargados referidos a otros módulos. En nuestro caso se sugiere ejecutar tres veces el generador con los siguientes parámetros:

[img30.bmp]

Menú Padre : modulo1.mn

Número de Sistema : 902

Número de Módulo: 903

Desea borrar los registros previos: No

Menú Padre : modulo2.mn

Número de Sistema : 902

Número de Módulo: 904

Desea borrar los registros previos: No

Menú Padre : modulo3.mn

Número de Sistema : 902

Número de Módulo: 905

Desea borrar los registros previos: No

Habiendo generado los siguientes rangos de códigos de unidades:

Módulo 1 del 90.300.000 al 90.300.202.

Módulo 2 del 90.400.000 al 90.400.156.

Módulo 3 del 90.500.000 al 90.500.320.

Administración de accesos

Los accesos a las unidades deben especificarse mediante el ABM de permisos siguiendo la lógica de la definición de los perfiles de usuarios. En este ABM se ingresa para los pares unidad-grupo de interés alguno de los posibles valores de permisos indicados arriba.

```
[img40.bmp]
```

Compilación de la información de accesos

La información de permisos presente en la base de datos debe procesarse (compilarse) a los efectos de generar archivos binarios que serán consultados al momento de iniciar cada

ejecutable Ideafix. Estos archivos se encuentran en un directorio creado especialmente para contenerlos que está referenciado por la variable de ambiente AUTHDIR. Supongamos que tenemos dos perfiles de usuarios, grp_A y grp_B, y que la variable AUTHDIR apunta al directorio /perfiles (en Windows NT puede ser d:\perfiles), entonces la estructura de $AUTHDIR será:

```
/perfiles
/perfiles/grp_A
/perfiles/grp_B
```

Nota: Estos directorios deben crearse en foma manual. En caso de no existir el directorio del grupo involucrado en la ejecución de gensys, no se generarán los archivos correspondientes al perfil.

El utilitario necesario para compilar los permisos es el ejectutable "gensys", que es invocado desde la línea de comandos y recibe dos parámetros. El primero es el nombre del menú ppal. del módulo y el segundo es el nombre del grupo sobre el que se compilará la informanción. Para nuestro caso se debe correr seis veces :

```
$ gensys modulo1.mn grp_A
$ gensys modulo1.mn grp_B
$ gensys modulo2.mn grp_A
$ gensys modulo2.mn grp_B
$ gensys modulo3.mn grp_A
$ gensys modulo3.mn grp_B
```

Como respuesta a cada ejecución exitosa debemos esperar un mensaje como el siguiente:

```
$ System 'moduloX.mn' for group 'grp_X' was saved ok.
```

y deben haberse creado los siguientes archivos:

/perfiles/groups

/perfiles/grp_X/units

/perfiles/grp_X/modulo1.mn.mno

/perfiles/grp_X/modulo2.mn.mno

/perfiles/grp_X/modulo3.mn.mno

Problemas usuales

Programas invocados a través de Cracker no se respetean los permisos compilados

Es muy probable que la variable AUTHDIR no este persente en el archivo .crackerrc del usuario.

Can not save system 'moduloX.mn' for group 'grp_X' fue el mensaje emitido tras ejecutar gensys

Seguramente grp_X no tiene permisos por defecto, con lo cual es un perfil totalmente restrictivo. Pero si grp_X debe poder ejecutar la unidad uni_Y no basta con darle permiso a ésta sino que debe tambien dársele permiso a todos los menues padres de uni_Y a los efectos de que exista un camino desde el menú principal hasta uni_Y. Gensys genera subconjuntos del menú correspondiente al sistema entero. En el caso de grupos restrictivos deben indicarse explícitamente los caminos hasta la unidad involucrada dando permiso de cualquier operación a todos los menues padres de la misma.

Se permite operar a determinado grp_X con la unidad uni_Y y despues de compilar los permisos los usuarios de grp_X continúan sin porder ejectuar uni_Y

Si un perfil es restrictivo por defecto y la unidad que se desea ejecutar es de la forma "doform form_Y" no alcanza con dar permiso a form_Y, tambien hay que dar de alta la unidad doform.exe y permitir sobre ésta cualquier operación. Otro caso similar corresponde a las invocaciones de la forma "execform -b form_Y "iql -b iql_Y.sql"" donde no solo hay que habilitar la unidad execform.exe, sino tambien la unidad iql.exe.

No se puede administrar los permisos sobre la unidad setenv

El motivo de este problema es que el generador automático da de alta la unidad respectiva como "setenv" y debe registrarse como "setenv.exe". Basta con editar la unidad setenv y modificar su nombre para solucionar el problema.

Parte III

•

IDEAFIX

Manual del Programador

1

Vademecum

Capítulo 1

Notación y sintaxis

La sintaxis utilizada para la descripción de los distintos componentes de IDEAFIX (DDS, FDL, RDL, IQL, etc.) es la siguiente:

```
Palabras en courier.
```

Las palabras que se encuentran con este tipo de letra, deberán ser ingresadas exactamente como se muestra. Excepto en la especificación de las funciones de biblioteca, en las que solamente el nombre de la función debe ser escrito igual.

Menor-Mayor.

Las palabras que aparecen escritas entre < > indican que son nombres por lo tanto deberán ser substituidas por el correcto valor, en el momento de utilizarlas.

Ejemplo:

```
create table <nombre>
```

En el ejemplo la palabra nombre podrá ser reemplazada por cualquier nombre, en el momento que se esté definiendo una tabla.

Corchetes.

Los “[ ]” indican que el argumento encerrado entre ellos es opcional. Cuando el prototipo de un argumento está dado como ”nombre” o “archivo” se refiere a un nombre de archivo.

Llaves { }

Cuando se especifiquen varios ítems entre llaves se deberá indicar solo uno de ellos.

Los ítems se separan con un "|", no escribir las llaves ni el pipe. Por ejemplo:

```
default {valor | $nombre }
```

En el ejemplo dado se define un valor en ausencia un "valor" fijo o una "variable de ambiente".

Paréntesis

Se presentan dos casos en el uso de paréntisis:

- sum (param)

En este caso se deberán especificar los paréntisis siempre que aparezcan

- (before | after)

En este caso, se deberán indicar tanto los paréntisis, como uno de los ítems indicados before ó after.

Palabras reservadas

Las palabras reservadas no pueden ser utilizadas como nombre de tablas, esquemas, campos o cualquier otro nombre que quede a elección del usuario.

Las palabras reservadas son las siguientes:

abs	add	after	all	alter
and	as	ascending	asterisk	at
autoenter	autowrite	avg	background	bar
before	between	binary	blink	blue
bold	bool	border	botmarg	by
cat	cd	char	check	chown
clear	close	commands	commit	confirm
control	count	create	cyan	date
day	dayname	debug	default	define
delete	delimited	descending	descr	description

digit	display	distinct	double	drop
eject	end	eof	error	exec
exit	false	field	flddec	flddef
flength	float	fnumber	formfeed	from
full	get	grant	graph	green
group	having	header	help	hour
id	if	ignore	in	index
inout	input	insert	internal	into
inverted	is	key	label	language
left	leftmarg	length	like	line
lineno	lock	low	lower	magenta
major	manip	manual	mask	max
messages	min	minor	module	month
monthname	next	no	not	null
num	number	numeric	on	only
option	or	order	origin	out
outer	output	page	pageno	pie
pipe	prev	primary	print	printer
privileges	public	recover	red	relop
rename	report	reverse	revoke	right
rollback	round	rows	runavg	runcount
runmax	runmin	runsum	schema	screen
select	set	shell	show	single
skip	stackedbar	standard	status	stdout
store	str	string	subform	sum
table	temp	temporary	terminal	text
time	to	today	top	topmarg
true	trunc	unique	unset	update
upper	use	values	view	wcmd
when	where	white	width	window
with	without	work	xy	year
yellow	zeros			

Generalidades

Las funciones de biblioteca CFIX de IDEAFIX se dividen en los siguientes grupos:

Grupo de funciones	Abreviatura

Base de Datos	DB
Formularios	FM
Generales	GN
Reportes	RP
Cadena de Caracteres	ST
Fecha y Hora	TM
Ventanas	WI
Manejo del tipo NUM	NM

Los tipos de datos definidos en los esquemas de IDEAFIX tienen la siguiente correspondencia con los utilizados en la interfaz de programación CFIX.

CFIX	Base de Datos	Tipo	Identificador	bool	num (<=4)	num (<=9)	num (<=15)	num (<=28)	float
char	date	bool	I	x					
		int	I	x	x				
		long	L		x	x			
		double	F		x	x	x		x
		NUM	N		x	x	x	x	x
		char	S						
x		DATE	D						
	x	TIME	T						

Los archivos de encabezamiento .fmh y .rph definen los campos involucrados en el formulario y en el reporte respectivamente. Estos campos tienen asociado un comentario indicando su tipo. El siguiente cuadro muestra la correspondencia entre los tipos de estos archivos y los definidos en la base de datos:

Archivo	Base de Datos	Tipo	bool	num (<=4)	num (<=9)	num (<=15)	num (<=28)	float
char	date	integer	x	x				
		long			x			
		double				x		x
		NUM					x	
		string						
x		DATE						
	x	TIME						

Capítulo 2

DDS (Data Definition Statements)

Para crear un esquema:

[CREATE] SCHEMA <nombre_esquema>

[DESCR[IPTION] <cadena>]

[LANGUAGE <cadena>]

Para crear una tabla:

[CREATE] TABLE <nombre_tabla> [[dimensión]]

[DESCR[IPTION]] <cadena>] {

campo [[dimensión]] <tipo> [lista_atributo],

campo ...

} [clave ...];

El tipo de campo puede ser uno de los siguientes:

num[eric] [(digitos [, dec])]

char [(long)]

time

date

float

bool

binary (sólo cuando se utiliza Essentia)

text (sólo cuando se utiliza Essentia)

El atributo es uno de los siguientes:

descr[iption] <cadena>

primary key [[separ]]

not null

default <valor>

mask <cadena>

check digit

check digit “/”

check digit “-”

< valor

> valor

= valor

<= valor

>= valor

!= valor

[not] between <valor> and <valor>

[not] in (<valor> [:<cadena>] [...])

[not] in <table> [by <índice>] [(<campo_clave> [,..])]

[: (campo [,...])]

check (expresión)

La expresión de un atributo check se puede formar con los siguientes operandos y operadores:

( )

+ - * /

= > < >0 != <=

and

or

not

in (<valor>, <valor>,...)

between <valor> and <valor>

num (expresión)

float (expresión)

date (expresión)

time (expresión)

this

<nombre de campo>

<valor>

<cadena>

today

hour

year

day

month

true

false

null

La clave de una tabla puede ser una de las siguientes:

primary key <espec_índice>

[unique] index <nombre> <espec_indice>

El espec_indice tiene la siguiente forma:

(campo [(inicio [,long])] [opción [opción...]]) [[separ]]

Las opciones de espec_índice pueden ser:

not null

asc[ending]

desc[ending]

El valor óptimo de separ se puede calcular con esta fórmula:

Para crear un índice:

[create] [unique] index <nombre> on <espec_índice>;

Para borrar archivos:

drop schema <nombre>;

drop table <nombre>;

drop index <nombre> on <tabla>;

Para generar la definición de un esquema a partir del archivo compilado:

store schema [<nombre>] [header];

Otras sentencias:

recover <índice> on <table>;

recover * on <tabla>;

rename table <tabla> as <nombre>;

rename field <campo> as <nombre>;

clear table <tabla>;

lock schema <esquema>;

lock table <tabla>;

El nombre de una tabla es uno de los siguientes:

nombre

esquema.nombre

El nombre de un campo es uno de los siguientes:

nombre

tabla.nombre

esquema.tabla.nombre

En cualquiera de las sentencias, se puede usar una variable de ambiente, escribiendo “$nombre”.

Biblioteca “C” de Base de Datos (DB)

Funciones para manejo de Esquemas

schema OpenSchema (char* name, UShort mode)

Abre un esquema.

const char* GetSchemas (void)

Obtiene una lista de los esquemas que están disponibles para la aplicación.

void CloseSchema (schema sch)

Cierra un esquema abierto.

void CloseAllSchemas ()

Cierra todos los esquemas abiertos por el programa.

schema FindSchema (char* name)

Devuelve el descriptor del esquema cuyo nombre se indica.

schema FindNextSchema (schema sch)

Devuelve el descriptor del siguiente esquema de la tabla de esquemas activos.

schema SwitchToSchema (schema sch)

Cambia el esquema corriente.

int LockSchema (schema sch, int mode)

Bloquea un esquema para poder modificarlo.

void FreeSchema (schema sch)

Libera un esquema bloqueado previamente. No libera tablas ni registros.

schema CurrentSchema ()

Devuelve el descriptor del esquema actual.

bool DbCheckSum (schema esq)

Devuelve el checksum del esquema.

void DbVerifyCheckSum (schema esq, long chksum)

Verifica que el checksum del esquema coincida con el especificado (es útil para verificar que el archivo ".sch" está actualizado).

bool DbCheckPoint (schema sch)

Esta función realiza un *checkpoint* en el server al cual pertenece el esquema sch.

char *GetSchDescr (schema sch)

Retorna una cadena con la descripción del esquema.

char *GetTables (schema sch)

Retorna una cadena con los nombres de las tablas del esquema, separados por tabuladores (\t).

Funciones para manejo de Tablas

void SetRelation (dbtable root, int level, int flag_err)

Activa la lectura automática de las tablas relacionadas con el atributo in table.

int GetRelDepth (dbtable tab)

Devuelve el nivel del último SetRelation.

int GetRelErrFlag (dbtable tab)

Devuelve el tercer parámetro (flag_err) del último SetRelation.

void CopyTable (dbtable target, dbtable source, dbfield vec[],

int cpy_flag)

Copia una tabla sobre otra.

dbtable CreateAlias (dbtable t)

Crea un alias a una tabla.

void DeleteAlias (dbtable t)

Borra un alias

void DeleteAllAlias (schema sch)

Borra todos los alias de todas las tablas de un esquema.

dbtable FindDbTable (schema sch, char* name)

Devuelve el descriptor de la tabla cuyo nombre se indica.

int LockTable (dbtable tab, int mode)

Bloquea una tabla para poder modificarla.

void FreeTable (dbtable tab)

Libera una tabla bloqueada previamente. También libera todos los registros de esa tabla.

void FlushTable (dbtable tab)

Graba físicamente los datos relativos a la tabla y libera la memoria asociada.

long TabActualSize (dbtable tab)

Obtiene el número de registros físicos de una tabla.

long TabNRecords (dbtable table)

Obtiene el número de registros lógicos de una tabla.

char *GetTabDescr (dbtable tab)

Retorna una cadena con la descripción de la tabla.

char *GetTabFlds (dbtable tab)

Retorna una cadena con los nombres de los campos de la tabla, separados por tabuladores (\t).

int SetTableCache(dbtable tbl, long n)

Establece un cache de n registros para la tabla tbl.

Funciones para manejo de Registros

long GetRecord (dbind nd, find_mode lmode, int bmode[, long n])

Lee un registro de datos.

void FreeRecord (dbindex ind, find_mode mode[,long recno])

Libera un registro bloqueado.

long FindRecord (dbindex ind, find_mode mode [,Ushort nparts])

Busca un registro dada la clave, sin copiar su contenido al buffer.

void PutRecord (dbtable tab)

Graba un registro en la tabla. Si ya existía lo modifica.

void AddRecord (dbtable tab)

Graba un registro en una tabla. Si ya existía da error.

void AddRecordTS (dbtable tab, TIMESTAMP crts, TIMESTAMP modts)

Graba un registro en la tabla, con los time stamps de creación y modificación indicados.

long AddRecordTest (dbtable tab)

Tiene la misma funcionalidad que AddRecord, pero no corrompe la transacción si el registro existe.

TIMESTAMP CTimeStamp (dbtable tab)

Obtiene el time stamp de creación del registro corriente.

TIMESTAMP MTimeStamp (dbtable tab)

Obtiene el time stamp de modificación del registro corriente.

void DelRecord (dbtable tab)

Borra un registro de una tabla.

int DelAllRecords (dbtable tab)

Borra todos los registros de una tabla.

void InitRecord (dbtable tab)

Pone los valores por omisión en los campos del registro corriente.

void PushRecord (dbtable tab)

Almacena el registro del buffer de la tabla en una pila.

long PopRecord (dbtable tab)

Copia en el buffer de la tabla el último registro introducido en la pila.

void DiscardRecord (dbtable tab)

Elimina el último registro introducido en la pila.

void DiscardAllRecords (dbtable tab)

Elimina todos los registros de la pila.

long RecordStackSize (dbtable tab)

Devuelve la cantidad e registros que hay en la pila.

Funciones para manejo de campos

void SetFld (dbfield fn, char* buf[, int row])

Da un valor a un campo, convirtiéndolo si es necesario.

void SetIFld (dbfield fn, short buf[, int row])

Da un valor de tipo short a un campo.

void SetLFld (dbfield fn, long buf[, int row])

Da un valor de tipo long a un campo.

void SetFFld (dbfield fn, double buf[, int row])

Da un valor de tipo double a un campo.

void SetDFld (dbfield fn, DATE buf[, int row])

Da un valor de tipo DATE a un campo.

void SetTFld (dbfield fn, TIME buf[, int row])

Da un valor de tipo TIME a un campo.

void SetNFld (dbfield fn,0 NUM* val[, int row])

Da un valor de tipo NUM a un campo.

void CopyFld (dbfield source, dbfield target [,int row])

Copia el valor de un campo del buffer de la base de datos a otro campo.

int CopyAliasRecord (dbtable target, dbtable source)

Copia información entre los buffers de una tabla y un alias, o entre los buffers de dos alias de una tabla.

void GetDspFld (dbfield f, char* buf[, int row])

Obtiene el valor de un campo como una cadena de caracteres, en formato de despliegue. Antes de llamar a esta función, se debe reservar suficiente memoria en buf.

void GetFld (dbfield f, char* buf[,int row])

Obtiene el valor de un campo como una cadena de caracteres. Antes de llamar a esta función se debe reservar suficiente memoria en buf.

int LenDspFld (dbfield fn)

Obtiene la longitud de un campo.

char* SFld (dbfield f[,int row])

Obtiene el valor de un campo de la base de datos, produciendo un resultado de tipo char*.

short IFld (dbfield f[,int row])

Obtiene el valor de un campo de la base de datos, produciendo un resultado de tipo short.

long LFld (dbfield f[, int row])

Obtiene el valor de un campo de la base de datos, produciendo un resultado de tipo long.

double FFld (dbfield f[,int row])

Obtiene el valor de un campo de la base de datos, produciendo un resultado de tipo double.

DATE DFld (dbfield f[,int row])

Obtiene el valor de un campo de la base de datos, produciendo un resultado de tipo DATE.

TIME TFld (dbfield f[,int row])

Obtiene el valor de un campo de la base de datos, produciendo un resultado de tipo TIME.

NUM NFld (dbfield fn [, int fila])

Obtiene el valor de un campo de la base de datos, produciendo un resultado de tipo NUM.

bool IsNull (dbfield fld[, int row])

Devuelve TRUE si el contenido del campo es NULL y FALSE en caso contrario.

dbfield AlFld (dbtable t, dbfield f)

Devuelve el descriptor de un campo alias en una tabla.

dbfield FindDbField (dbtable tab, char* name)

Devuelve el descriptor del campo cuyo nombre se indica.

char* InDescr (dbfield fld, char* val)

Obtiene la descripción asociada a un valor en una cláusula IN.

char *GetFldDesc (dbfield fld)

Devuelve la descripción del campo.

char *GetFldName (dbfield fld)

Devuelve el nombre del campo.

type GetFldType (dbfield fld)

Devuelve el tipo del campo.

int GetFldLen (dbfield fld)

Devuelve la longitud del campo.

int GetFldNDec (dbfield fld)

Devuelve la cantidad de decimales del campo.

void UpdateRecord (dbtable tab, dbfield fld, ...)

Permite actualizar campos de una tabla, sin necesidad de leer/grabar todo el registro. Se puede utilizar sólo con Essentia.

Funciones para manejo de índices

void SetKey (dbindex ind, value[, value][, ...])

Asigna valores a los campos que forman la clave.

void SetKeyFld (dbindex ind, int n, char* val)

Asigna un valor a un campo de la clave.

void GetKeyFld (dbindex ind, int n, char* buf)

Obtiene el valor de un campo de la clave. Antes de llamar a esta función, se debe reservar suficiente memoria buf.

dbindex AlInd (dbtable t, dbindex ind)

Devuelve el descriptor de un índice para un alias.

dbindex FindDbIndex (dbtable tab, char* name)

Devuelve el descriptor del índice cuyo nombre se indica.

dbfield KeyDbField (dbindex ind, int n)

Devuelve el descriptor de un campo de la clave.

long FindRecord (dbindex ind, find_mode mode [, Ushort nparts])

Busca una clave en un índice.

char *GetIndFlds (dbindex ind)

Retorna un cadena de carácteres con los nombres de los campos del índice dado, separado con tabuladores (\t).

char *GetIndName (dbindex ind)

Retorna una cadena de carácteres con el nombre del índice.

char *GetIndices (dbtable tab)

Retorna una cadena con los nombres de los índices de la tabla, separados por tabuladores (\t).

Funciones para manejo de cursores

dbcursor CreateCursor (dbindex ind, int mode)

Crea un cursor.

void SetCursorFrom (dbcursor cursor, [,value...])

Establece la clave para el límite inferior de un cursor.

void SetCursorTo (dbcursor cursor, [,value...])

Establece la clave para el límite superior de un cursor.

void SetCursorFromFld (dbcursor cursor, int n, char* val)

Establece el valor de un campo de la clave para el límite inferior de un cursor.

void SetCursorToFld (dbcursor cursor, int n, char* val)

Establece el valor de un campo de la clave para el límite superior de un cursor.

void DeleteCursor (dbcursor cursor)

Elimina un cursor.

void DeleteAllCursor (schema sch)

Elimina todos los cursores asociados al esquema sch. Sólo con Essentia.

long CountCursor (dbcursor cursor)

Obtiene la cantidad de registros en el rango de un cursor.

long FetchCursor (dbcursor cursor)

Lee el siguiente registro dentro de un cursor.

long FetchCursorPrev (dbcursor cursor)

Lee el registro anterior dentro de un cursor.

long FetchCursorThis (dbcursor cursor)

Inserta en el registro corriente de la tabla asociada al cursor, el último registro leído con *FetchCursor/FetchCursorPrev*.

void MoveCursorFirst (dbcursor cursor)

Mueve un cursor al principio del rango de registros.

void MoveCursorLast (dbcursor cursor)

Mueve un cursor al final del rango de registros.

int SetCursorFlds (dbcursor cursor, ..)

Recibe la lista de los descriptores de campos para pasársela al registro corriente cada vez que un FetchCursor/FetchCursorPrev sea ejecutado. El final está indicado con cero.

int SetCursorVFlds (dbcursor cursor, dbfield *fld)

Igual que SetCursorFlds, pero recibe un arreglo de descriptores de campo finalizado con el descriptor 0.

Funciones para manejo de Versiones

long VersionNumber (char* name, DATE f)

Devuelve el número de versión de un esquema que estaba activo en una fecha dada. Sólo se puede utilizar con Essentia

long SchemaVersion (schema sch)

Obtiene el número de versión asociado al descriptor de esquema sch. Sólo cuando Essentia está corriendo.

schema OpenSchemaVersion (char* name, UShort mode, long version)

Abre una versión de un esquema. Sólo se puede utilizar con Essentia

schema FindSchemaVersion (char* name, long version)

Obtiene el descriptor correspondiente a una versión dada de un esquema. Sólo se puede utilizar con Essentia.

Funciones para manejo de objetos de texto y binarios

Todas las funciones que se describen a continuación utilizan las capacidades BLOB de Essentia.

int GetBinaryToFile (dbfield f, char *path)

Copia los contenidos del campo *f* binario al archivo indicado en el path.

int GetBinaryToMem (dbfield f, void *buffer, long size)

Copia los contenidos del campo *f* binario al buffer de memoria de tamaño *sz*.

int StoreBinaryFromFile (dbfield f, char *path)

Copia los contenidos del campo indicado en el *path* al campo *f* binario.

int StoreBinaryFromMem (dbfield f, void *buffer, long sz)

Copia los contenidos del buffer de memoria de tamaño sz al campo *f* binario.

int DiscardBinary (dbfield f)

Borra (deja con valor 0) el contenido del campo binario *f*.

int GetTextToFile (dbfield f, char *path)

Copia los contenidos del campo *f* texto al archivo indicado en el path.

int GetTextToMem (dbfield f, void *buffer, long size)

Copia los contenidos del campo *f* texto al buffer de memoria de tamaño *sz*.

int StoreTextFromFile (dbfield f, char *path)

Copia los contenidos del campo indicado en el *path* al campo *f* texto.

int StoreTextFromMem (dbfield f, void *buffer, long size)

Copia los contenidos del buffer de memoria de tamaño sz al campo *f* texto.

int DiscardText (dbfield f)

Borra (deja con valor 0) el contenido del campo texto *f*.

Funciones para manejo de transacciones

void BeginTransaction ()

Comienza una transacción

void EndTransaction ()

Termina una transacción.

bool TransOk (void)

Obtiene el estado de la transacción activa.

int RollBack ( )

Deshace todos los cambios hechos en la transacción activa.

DoTransaction { ...(texto que incluye la transacción

para ejecutarse)...}

Ejecuta el bloque de sentencias como una transacción completa, reintentando hasta tener éxito.

FPLCPCP SetDbHandler (FPLCPCP f)

Establece el manejador de errores de entrada / salida.

DoSyncTransation { }

Es equivalente a DoTransaction pero ejecuta un ENDSyncTransaction en lugar de un EndTransaction cada ciclo. Sólo con Essentia

bool EndSyncTransation ()

Es equivalente a EndTransaction, además garantiza el *sync* (bajada a disco) del log de transacciones. Sólo con Essentia.

int dbmsType ()

Devuelve el DBMS con el cual se está trabajando (DBMS_IDEAFIX ó DBMS_ESSENTIA).

Otras funciones de Bases de Datos

schema CreateSchema (char* name, char* descr)

Crea un esquema.

dbindex CreateIndex (dbtable tbl, char* name, key_flags flags,

UChar separ)

Crea un índice nuevo para una tabla.

void BuildIndex (dbindex ind, int flags)

Construye un índice, pasando todos los registros de la tabla. Si no se usa la opción IO_VERBOSE, los últimos parámetros son NULL.

long AddKey (dbindex ind)

Agrega al índice la clave que está en el buffer de la tabla.

int CompleteIndex (dbindex ind)

Construye un índice, sin pasar ningún registro de la tabla.

int AddIndField (dbindex ind, dbfield campo, k_field_flags flags,

UShort n, Uhort m)

Agrega campos a un índice.

dbtable CreateTable (schema schema, char*name, char* descr,

tab_flags flags, Ulong size, UChar n_fields)

Crea una nueva tabla. El último parámetro debe ser > = 1.

dbfield AddTabField (dbtable tab, char* fname, char* descr,

UShort dim, type ty, sqltype sql_ty, UShort len,

UChar ndec, field_flags flags, char* def,

char* mask)

Agrega un campo a una tabla.

int DropSchema (schema sch)

Borra un esquema.

int DropTable (dbtable tab)

Borra una tabla.

int DropIndex (dbindex ind)

Borra un índice.

dbtable FindTabReference (dbtable t)

Busca referencias a una tabla.

int TabNFields (dbtable tab)

Devuelve el número de campos de una tabla.

int IndNFields (dbtable ind)

Devuelve el número de campos de un índice.

Tipos de la base de datos

schema: descriptor de esquema.

dbtable: descriptor de tabla.

dbfield: descriptor de campo.

dbindex: descriptor de índice.

Valores de los parámetros

Modos de búsqueda (find_mode):

- THIS_KEY, búsqueda por clave exacta.
- PREV_KEY, busca la clave anterior.
- NEXT_KEY, busca la clave siguiente.
- FIRST_KEY, busca la primera clave del índice.
- LAST_KEY, busca la última clave del índice.
- PARTIAL_KEY, compara solamente una parte de la clave.

Modo de bloqueo:

- IO_NOT_LOCK , siempre lee el registro, por más que esté bloqueado por otro programa.
- IO_LOCK, lee el registro y lo bloquea.
- IO_PLOCK, lee el registro y lo bloquea en forma persistente (se mantiene cuando termina la transacción). Desaparece sólo cuando se lo libera ó el proceso termina.
- IO_TEST, verifica si el registro está bloqueado.
- IO_EABORT, si se produce un error, aborta el programa. No se puede usar con CreateCursor.
- IO_READ , permite que un proceso pueda asegurar que ese registro no es modificado mientras mantenga su lock sobre el.

Modos para **OpenSchema**:

- IO_DEFAULT, si ocurre un error, devuelve ERROR.

- IO_EABORT , si se produce un error, aborta el programa.
- ... | IO_SYMBOLS , carga la tabla de símbolos del esquema.

Modos para **CreateCursor**:

- IO_NOT_LOCK, siempre lee el registro, por más que esté bloqueado por otro programa.
- IO_LOCK, lee cada registro y lo bloquea.
- IO_TEST, verifica si el registro está bloqueado.
- IO_CONTROL_BREAK, verifica cortes de control.
- IO_DESCENDING, el cursor se mueve en el índice que es creado de forma reversa a la que se movería si el flag no estuviera presente.
- IO_KEEP_LOCKS, (tiene sentido cuando el cursor es creado para bloquear) hace que el cursor no libere automáticamente los bloqueos que están siendo creados. Si el cursor es creado dentro de una transacción, el registro será liberado cuando esta termine, en cualquier otro caso, es responsabilidad del programador liberarlos (por ejemplo, con FreeTable o FreeRecord).
- IO_KEY_FIELDS, el cursor sólo traerá los campos que pertenecen al índice por el cual fue creado el cursor..
- IO_READ, crea un bloqueo de lectura. Se define añadiendo (no reemplazando) IO_READ a IO_LOCK. Varios bloqueos de lectura pueden coexistir en un momento dado en un registro, pero un bloqueo de lectura no puede coexistir con un bloqueo de escritura (el bloqueo normal de IDEAFIX).
- IO_PLOCK, crea un bloqueo persistente a través de transacciones; desaparecerá sólo cuando se lo libere explícitamente o cuando termine el proceso.

Modos para **SetRelation**:

- IO_EABORT, si se produce un error, aborta el programa.
- IO_TEST, para notificar indicando las tablas que no pueden ser relacionadas.
- IO_DEFAULT, sino puede relacionar dos tablas, lo ignora.

Flags para **CopyTable**:

- CT_NORMAL: significa que la copia será una copia normal.
- CT_COPY_STAMPS: este modificador hará que CopyTable borre la tabla destino y luego use AddRecordTS para copiar los registros de una tabla a la otra, para mantener los timestamps de los registros. debido al uso de AddRecordTS, la persona que ejecute esta función con este indicador, debe ser el dueño del esquema.

Modos para **BuildIndex**:

- IO_CLEAR_INDEX, si ya existe ese índice, lo borra y lo vuelve a crear.
- IO_VERBOSE, muestra información adicional: nombre del índice y de la tabla, porcentaje de registros procesados, y cantidad total de registros.

Modos para **CreateIndex**:

- K_DUP, acepta claves duplicadas.
- K_UNIQUE, no acepta claves duplicadas.
- K_TEMP, el índice es temporario (se borra al cerrar el esquema).
- IO_EABORT, aborta la ejecución del programa si se detecta un error.

Modos para **AddIndField**:

- K_ASCENDING, orden ascendente.
- K_DESCENDING, orden descendente.
- K_NOT_NULL, el índice no puede contener entradas nulas.

Modos para **CreateTable**:

- TAB_NORMAL, la tabla es normal (queda almacena da hasta que el usuario la borre).
- TAB_TEMP, la tabla es temporaria (se borra automáticamente al cerrar el esquema).
- IO_EABORT, aborta la ejecución del programa si se detecta un error.

Modos para **AddTabField**:

- F_OPTIONAL, el campo puede quedar vacío.
- F_NOT_NULL, el campo no puede ser ignorado, debe contener algo.

Tipos de campos para **AddTabField**:

- TY_NUMERIC, Entero.
- TY_FLOAT, Punto flotante.
- TY_STRING, Alfanumérico.
- TY_DATE, Fecha.
- TY_TIME, Hora.

- TY_BOOL, Booleano (verdadero o falso).

Operaciones para **UpdateRecord**:

- U_ADD, suma un valor determinado al campo.
- U_SUB, resta un valor al campo.
- U_MUL, multiplica el campo por un número.
- U_DIV, divide el campo por un valor.
- U_SET, establece el valor del campo.
- U_COPY, copia un campo sobre otro.

Capítulo 3

FDL (Form Description Language)

La estructura de un archivo “.fm” es la siguiente:

Imagen de la pantalla

%form (parámetros)

[especificaciones generales del formulario]

%fields

[especificaciones de los campos]

La imagen de la pantalla es una copia del formulario que se quiere mostrar.

Los campos se indican de la siguiente forma:

Campos alfanuméricos _ _ _ _ _ _ _ _ _ _ _ _ _ _

Campos numéricos [-] _ _ [,] _ _ _. [_ _ _ _]

Campos tipo fecha _ _ / _ _ / _ _ [_ _]

Campos tipo hora _ _ : _ _ [: _ _]

Campos numéricos con punto flotante [-] _ _ _ _ _ _e

Campos booleanos _ _ ?

Para definir un **campo múltiple**, se lo debe encerrar entre corchetes [ ].

Para definir un **campo agrupado**, se lo debe encerrar entre llaves { }.

Para definir **campos polimórficos**, los campos referenciados deben ser todos de tipo internal, estar juntos dentro de la zona %fields, aparecer luego del campo de referncia y estar comprendidos en forma creciente dentro de la lista de la cláusula reference.

En los formularios el fin de zona de claves se indica con un pipe |.

Si se quiere usar una variable de ambiente, se la debe escribir en la forma “\$nombre”.

La secuencia ´\0´ actúa como separador de campos en la imágen del formulario. Esto permite que dos campos diferentes aparezcan como si fueran uno solo.

Para agregar el atributo check digit a un campo numérico sin decimales, se coloca un signo numeral # al lado del punto final del campo.

Los parámetros que se indican a continuación del %form solamente se usan en subformularios, para indicar que datos reciben desde el formulario principal. Se escriben separados por comas y cada uno lleva delante in, out o inout, según sea de entrada, salida o de entrada y salida. Si no se indica ninguno, se asume in.

Las especificaciones generales del formulario pueden ser:

use esquema [, esquema....];

language “c”;

ignore tecla [, tecla...];

confirm operación [, operación...];

messages nombre: cadena [, nombre: cadena ....];

window atributo [, atributo...];

without control field;

autowrite;

display status;

Las teclas de la cláusula ignore pueden ser:

add

update

ignore

delete

next

prev

end

Las operaciones de la cláusula confirm pueden ser:

add

update

ignore

delete

end

Los atributos de la cláusula window pueden ser:

label cadena

full screen

origin fila [, columna]

border atributo [, atributo ...]

background color

Los atributos de la cláusula border pueden ser:

Atributo de borde	Tipo de carácteres	Límites del borde	Colores del borde
blink bold reverse	single double asterisk	top low left right	red green blue yellow magenta cyan white

La zona “especificaciones de los campos” contiene una línea por cada campo de la pantalla, de esta forma:

nombre_campo: [[esquema.]tabla] [, atributo[, atributo..]

Los atributos pueden ser:

not null

descr[iption] <msg>

descr[iption] <expresión> (Versión 4.3.1)

display only

display only when <expresión>

skip

skip when <condición>

[manual] subform (<nombre> [, ...])

display only + is (si “is” retorna void el campo no será skip sino read only)

has [after|before] when <expresión>

Sólo para campos simples:

in tabla [by indice [descending][(valor..)]][:(campo1..)]

En la versión 4.3.1 se cambia la posición de *descending:*

in tabla [by indice][(valor..)][:(campo1..)]descending

internal tipo

internal campo

prev campo

next campo

auotoenter

on help {<MSG>|manual}

on error <MSG>

on help <expresion> (Versión 4.3.1)

on error <expresion> (Versión 4.3.1)

mask <string>

mask <expresion> (Versión 4.3.1)

mask {<expresion>|<string>}, in table; el campo aparecerá enmascarado dentro del PopUp.(Versión 4.3.1)

default <valor>

length<valor>

autoenter

<campo_tabla>

[manual] subform(<cadena> [, ...])

check

is <expresión>

El tipo de atributo internal puede ser uno de los siguientes:

num[eric] [(long [, decs])]

char [(long)]

time

date

float

bool

Los atributos de **check** que permiten especificar una validación sobre el valor ingresado en el campo, son los siguientes:

Operadores relacionales (>, <, >=, <=, !=, ==).

Operador [not] between.

[not]in ... (valores)...

[not]in tabla [descendente]

row() Retorna la fila del campo sobre el cual se está ejecutando la expresión (Versión 4.3.1)

Los atributos de **check** que permiten especificar validaciones sobre el valor de los campos en pantalla son:

relop {<valor>|campo}

[not] between <valor> and <valor>

[not] in (<valor>[:<cadena>] [,<valor>...])

[not] in table

Los campos de ayuda de un atributo **in** son una lista de campos separados por comas (entre paréntesis si hay más de uno). además, se puede hacer referencia a campos de otras tablas, siempre que estén vinculadas a la tabla corriente con otro in: se usa la notación “campo-que -tiene-el-inf(CB -> campo-que-se-quiere-ver”.

La expresión de un atributo **is** se puede formar con los siguientes operandos y operadores y si se indica una condición se puede usar cualquiera de los indicados para check:

( ) + - * / ? :

nombre de campo valor

today hour

sum(campo) avg(campo) max(campo) min(campo) count(campo)

descr(campo) help(nombre de tecla)

num(expresión) float(expresión) date(expresión) time(expresión)

when condición

void

Expresiones para el atributo **has after when**:

changed() valor de retorno bool, indicando si el campo a sido modificado o no. (Versión 4.3.1)

Sólo para campos múltiples:

rows <número>

display <número>

ignore [delete] [add] [insert]

Sólo para campos numéricos (sin decimales):

check digit check digit "/" check digit "-"

Para campos dentro de un campo múltiple:

unique

Para campos dentro de un campo agrupado:

check after <campo>

Además, para campos agrupados se pueden colocar expresiones de check sobre estos.

Biblioteca "C" de Formularios (FM)

Funciones para manejo de formularios y subformularios

form OpenForm (char* name, int flags)

Abre un formulario

void CloseForm (form fmd)

Cierra un formulario.

void CloseAllForms ()

Cierra todos los formularios.

fm_cmd DoForm (form fmd, fm_status_fp before, fm_status_fp after)

Opera con un formulario.

fm_cmd DoSubform(form fmd, fm_status_fp before, fm_status_fp after,

fmfield fld, UShort subf_n[, int row])

Opera con un subformulario.

void DisplayForm (form fmd)

Despliega un formulario.

void ClearForm(form fmd)

Saca de pantalla un formulario.

form UseSubform (form fmd, fmfield fld, UShort subf_n[, int row])

Obtiene el descriptor del subformulario asociado a un campo.

int FmSubformId (form fmd)

Devuelve el identificador de un subformulario.

void FmRecalc (form fmd, UShort row)

Recalcula los valores de los campos con atributo is.

UShort FmSubfRow (form fmd)

Devuelve el número de renglón corriente del subformulario.

form FmFather (form fmd)

Devuelve el descriptor del formulario padre.

long FmCheckSum (form fmd)

Devuelve el checksum del formulario

void FmVerifyCheckSum (form fmd, long chksum)

Verifica que el checksum del formulario coincida con el especificado (es útil para verificar que el archivo ".fmh" esté actualizado).

fmfield FmAssocFld (form subf)

Permite ejecutar diferentes *after/before* cuandose llama al mismo subformulario desde distintos campos. (Versión 4.3.1)

form FmCallBacks (form fmd, fmdfield fno, const char *sfname,

fm_status_fp before_fp,fm_status_fp after_fp)

Cuando se utilizan subforms automáticos los *after/before* de los forms que los contienen, deben estar diseñados para tal propósito (generalmente usando la función FmSubFormId). Para estos casos se recomienda el uso de esta nueva función, evitando así *after/before* complejos. (Versión 4.3.1)

FmSetErrorFld (form fm, fmfield fld, int rowno=0)

Esta función (sólo disponible en C++) posiciona el cursor en la fila indicada por rowno, del

campo fld del formulario fm y establece el estado del formulario en error. (Versión 4.3.1)

Funciones para manejo de campos

void FmShowFlds (form fmd, fmfield from, fmfield to[, int row])

Muestra en la pantalla uno ó mas campos de un formulario.

void FmShowAllFlds (form fmd)

Muestra en la pantalla todos los campos de un formulario.

void FmClearFlds (form fmd, fmfield from, fmfield to[, int row])

Pone en blanco uno o más campos de un formulario.

void FmClearAllFlds (form fmd)

Pone en blanco todos los campos de un formulario.

void FmGetFld (char* buf, form fmd, fmfield fn[, int fila])

Obtiene el valor de un campo de l formulario como una cadena de caracteres. Antes de llamar a esta función, se debe reservar suficiente memoria en buf.

char* FmSFld (form fmd, fmfield fld[,int fila])

Obtiene el valor de un campo del formulario, produciendo un resultado de tipo char *.

int FmIFld (form fmd, fmfield fld[,int fila])

Obtiene el valor de un campo del formulario, produciendo un resultado de tipo int.

long FmLFld (form fmd, fmfield fld[,int fila])

Obtiene el valor de un campo del formulario, produciendo un resultado de tipo long.

double FmFFld (form fmd, fmfield fld[,int fila])

Obtiene el valor de un campo del formulario, produciendo un resultado de tipo double.

TIME FmTFld (form fmd, fmfield fld[,int fila])

Obtiene el valor de un campo del formulario, produciendo un resultado de tipo TIME.

DATE FmDFld (form fmd, fmfield fld[,int fila])

Obtiene el valor de un campo del formulario, produciendo un resultado de tipo DATE.

NUM FmFld (form fmd, fmfield fn [, int fila])

Obtiene el valor de un campo del formulario, produciendo un resultado de tipo NUM.

void FmSetFld (form fmd, fmfield fld, char* buf[, int fila])

Establece el valor de un campo del formulario, convirtiéndolo si es necesario.

void FmSetIFld (form fmd, fmfield fld, int val[, int fila])

Da un valor de tipo int a un campo del formulario.

void FmSetLFld (form fmd, fmfield fld, long val[, int fila])

Da un valor de tipo long a un campo del formulario.

void FmSetFFld (form fmd, fmfield fld, double val[, int fila])

Da un valor de tipo double a un campo del formulario.

void FmSetDFld (form fmd, fmfield fld, DATE val[, int fila])

Da un valor de tipo DATE a un campo del formulario.

void FmSetTFld (form fmd, fmfield fld, TIME val[, int fila])

Da un valor de tipo TIME a un campo del formulario.

void FmSetNFld(form fmd, fmfield fn, NUM* val[, int row])

Da un valor de tipo NUM a un campo del formulario.

void FmToDb (form fmd, fmield from, fmfield to[, int row])

Copia campos del formulario a la base de datos, validando los atributos “in table”

void DbToFm (form fmd, fmfield from, fmfield to[,int row])

Copia campos de la base de datos al formulario.

char* FmFldPrev (form fmd)

Devuelve el valor anterior del campo corriente. Sólo se usa en funciones after.

int FmFldLen (form fmd, fmfield fld)

Devuelve la longitud de un campo en caracteres. Si el campo es múltiple, devuelve el número de filas.

bool FmChgFld (form fmd)

Devuelve TRUE si el campo corriente ha sido modificado. Sólo se usa en funciones after.

bool FmIsNull (form fmd, fmfield fld[, int row])

Devuelve TRUE si el campo contiene el valor NULL y FALSE en caso contrario.

fmfield FindFmField (form fm, const char *name)

Obtiene el descriptor del campo cuyo nombre se indica.

fmfield FmInMult (form fmd fmfield fld[, int row])

Devuelve el descriptor del campo virtual del multirrenglón al que pertenece un campo.

bool FmIsDisplayOnly (form tbl, fmfield fld, [int row])

Retorna TRUE si el campo fld del formulario fmd tiene atributo display-only. El último parámetro corresponde a la versión 4.3.1.

Funciones Generales

fm_status FmStatus (form fmd)

Devuelve el estado actual de un formulario.

void FmSetStatus (form fmdd, fm_status state)

Establece el estado de un formulario.

int FmNextFld (form fmd, fmfield fno, [int nofila])

Establece el siguiente campo al que irá el cursor.

void FmSetError (form fmd, fmfield fn, ...)

Establece el estado de un formulario en ERROR, pone el cursor en un campo determinado y muestra un mensaje de error.

fm_status FmErrMsg (form fmd, int msg, ...)

Despliega un mensaje. Siempre devuelve FM_ERROR.

void FmSetKeyCode (form fmd, UChar code)

Establece la última tecla de función presionada en el formulario.

UShort FmInOffset (form fmd)

Devuelve el índice IN seleccionado en el campo actual. Sólo se usa en funciones after.

void FmSetInOffset (form fmd, UShort n)

Selecciona un valor del IN del campo actual. Sólo se usa en funciones after.

void FmSetDisplayOnly (form fm, fmfield from, fmfield to, bool value)

Establece el atributo “display only” de un rango de campos.

void FmSetMask (form fmd, fmfield fld, char* mask)

Coloca una máscara a un campo.

char* FmInValue (form fmd, fmfield fld, int n)

Devuelve un valor de la cláusula IN del campo actual.

char* FmInDescr (form fmd, fmfield fld, int n)

Devuelve la descripción asociada a un valor de la cláusula IN del campo actual.

int FmNKeys (form fm)

Retorna un entero con el número de campos que conforman la clave del formulario.

void FmOnKey (form fm, long fkey, fm_status_fp fp, fmfield from,

fmfield to)

Relaciona una tecla a una función que modificará un grupo de campos en un formulario.

fmfield FmRefFld (form f, fmfield fld, ...)

Retorna el descriptor de campo que corresponde al tipo corriente en un momento dado.

void SetAddPerm (bool v)

Permiten al programador establecer el permiso de alta del formulario.

void SetUpdatePerm (bool v)

Permiten al programador establecer el permiso de modificación del formulario.

void SetDelPerm (bool v)

Permiten al programador establecer el permiso de baja del formulario.

Tipos de los Formularios

form: descriptor de formulario.

fmfield: descriptor de campo de formulario.

fm_cmd:

FM_EXIT, se presionó la tecla FIN.

FM_DELETE, se presionó la tecla REMOVER.

FM_IGNORE, se apretó la tecla IGNORAR.

FM_READ, la clave está completa.

FM_READ_NEXT, la clave contiene el registro anterior al que se desea procesar.

FM_READ_PREV, la clave contiene el registro siguiente al que se desea procesar.

FM_ADD, se presionó la tecla PROCESAR y el registro no existe en la tabla.

FM_UPDATE, se presionó la tecla PROCESAR y el registro ya existía en la tabla.

fm_status, para las funciones after y before:

FM_OK, continuar el procesamiento normal del campo.

FM_SKIP, saltear el campo y pasar al siguiente. Sólo se usa en la función before.

FM_REDO, volver a pedir la entrada del campo. Sólo se usa en la función after.

FM_LOCKED mostrar un mensaje indicando que el registro está bloqueado.

fm_status, para DoForm:

FM_NEW, blanquea el formulario y posiciona el cursor en el primer campo.

FM_USED, despliega en la pantalla los datos del formulario y posiciona el cursor en el campo de control.

FM_LOCKED, muestra un mensaje de error indicando el bloqueo y posiciona el cursor en el primer campo.

FM_EOF, muestra un mensaje de error indicando el fin de archivo y posiciona el cursor en el primer campo.

FM_ERROR, despliega los datos del formulario y posiciona el cursor en el campo indicado por FmSetError.

Modos para OpenForm:

FM_NORMAL, abre el formulario y lo despliega en la pantalla.

FM_NODISPLAY, abre el formulario, pero no lo muestra en la pantalla.

FM_EABORT, si ocurre un error, aborta el programa.

FM_SYMBOLS, carga en memoria la tabla de símbolos del formulario.

Código de teclas:

K_PROCESS	Procesar
K_ENTER	Enter/Return/Intro.
K_CURS_UP	Flecha hacia arriba.
K_CURS_LEFT	Flecha hacia izquierda.
K_CURS_RIGHT	Flecha hacia derecha.
K_CURS_DOWN	Flecha hacia abajo.
K_PAGE_UP	Página Arriba/Page Up.
K_PAGE_DOWN	Página Abajo/Page Down.
K_PAGE_LEFT	Página Izquierda.
K_PAGE_RIGHT	Página Derecha.
K_BACKSPACE	Retroceso/Backspace.
K_TAB	Tabulador.
K_HELP	Ayuda (<dg>).
K_META	Meta (<dg>).

K_CTRLX	Ctrl+X.

Capítulo 4

RDL (Report Description Language)

La estructura de un archivo “.rp” es la siguiente:

definición de zonas del reporte

%report

especificaciones generales del reporte

%fields

especificaciones de los campos

Cada zona del reporte se define de la siguiente forma:

% <nombre de zona> ([parámetros]) [opciones]

imagen de la zona

Las opciones posibles son:

before report

before page

before <campo>

after report

after page

after <campo>

if <condición>

group with <zona>

eject

eject before

eject after

at line <número de línea>

no print

La condición de una cláusula if se arma con los siguientes operandos y operadores:

< > <= >= = !=

<nombre de campo>

<constante>

sum(param)

avg(param)

count(param)

min(param)

max(param)

runsum(param)

runavg(param)

runcount(param)

runmin(param)

runmax(param)

day(campo)

month(campo)

year(campo)

dayname(campo)

monthname(campo)

date()

time()

today

hour

pageno

lineno

module

Expresiones dentro de la definición de campos de una zona:

%zoneX (expr [:expid],...)..

El parámetro expr es una expresión completa que puede contener llamadas a funciones y campos combinados en expresiones aritméticas.

Las expresiones en at line, flenght, botmarg, topmarg, leftmarg, width, output to, input from, mask no pueden contener referencias a expresiones definidas en una zona.

La imagen de una zona es una copia del renglón que se quiere mostrar.

Los campos se indican de la siguiente forma:

Campos alfanuméricos _ _ _ _ _ _ _ _ _ _ _ _ _

Campos numéricos [-] _ _ [,] _ _ _. [_ _ _ _]

Campos tipo fecha _ _ / _ _ / _ _ [_ _]

Campos tipo hora _ _ : _ _ [: _ _]

Campos numéricos con punto flotante [-] _ _ _ _ _ _e

Campos booleanos _ _ ?

Si se quiere usar una variable de ambiente, se la debe escribir en la forma “\$nombre”.

Para agregar el atributo check digit a un campo numérico sin decimales, se coloca un signo numeral “#” al lado del punto final del campo.

Las especificaciones generales del reporte pueden ser:

use <esquema> [, <esquema>];

language “c”;

flength = <valor>; // Largo de página

topmarg = <valor>; // Margen superior

botmarg = <valor>; // Margen inferior

leftmarg = <valor>; // Margen izquierdo

[no] formfeed;

output to [

archivo

| printer

| pipe comando

| terminal

| stdout

]

input from <fuente>;

<fuente>: <cadena>

| <variable_de_ambiente> [...]

| pipe <cadena>

| <variable_de_ambiente>

| terminal

Las especificaciones de los campos contiene una línea por cada campo de las imágenes de las zonas, de esta forma:

campo: [[[esquema].tabla].campo] [,atributo [,...]]

Los atributos pueden ser:

mask <cadena>

null zeros

[no] check digit

check digit “/”

check digit “-”

Biblioteca “C” de Reportes (RP)

Funciones para manejo de Reportes

report OpenReport (char* name, int flags[, int ncopies, char* str])

Abre un reporte.

int CloseReport (report rpd)

Cierra un reporte.

void CloseAllReports ()

Cierra todos los reportes.

void RpSetOutput (report rpd, rp_output out_to, char* arg)

Cambia la salida del reporte a punto de comenzar (abrir el reporte con RP_NOBEGIN y usar la función antes de BeginReport).

int BeginReport (report rpd, int ncopies, char* str)

Comienza un reporte. (si se abrió con RP_NOBEGIN)

int EndReport (report rpd)

Termina un reporte.

long RpCheckSum (report rpd)

Calcula el “checksum” de la definición de un reporte.

void RpVerifyCheckSum (report rpd, long chksum)

Verifica que el checksum del reporte coincida con lo especificado.

Funciones para manejo de zonas

int DoReport (report rpd, UShort zone)

Envía una zona a la salida.

void RpClearZone (report rpd, int zone)

Inicializa todos los campos de una zona.

Funciones para manejo de campos

void RpSetFld (report rpd, rpfield fn, char* val)

Asigna un valor a un campo del reporte, convirtiéndolo si es necesario.

void RpSetIFld (report rpd, rpfield fn, int val)

Da un valor de tipo int a un campo del reporte.

void RpSetLFld (report rpd, rpfield fn, long val)

Da un valor de tipo long a un campo del reporte.

void RpSetFFld (report rpd, rpfield fn, double val)

Da un valor de tipo double a un campo del reporte.

void RpSetDFld (report rpd, rpfield fn, DATE val)

Da un valor de tipo DATE a un campo del reporte.

void RpSetTFld (report rpd, rpfield fn, TIME val)

Da un valor de tipo TIME a un campo del reporte.

void RpSetNFld (report rpd, rpfield fn, NUM* val)

Da un valor de tipo NUM a un campo del reporte.

void DbToRp (report rpd, rpfield from, rpfield to [,int row])

Copia los valores de la base de datos al reporte.

int RpIFld (report rpd, rpfield fn)

Obtiene ciertos valores especiales de un reporte.

int RpFldLen (report rp, rpfield fld)

Retorna un entero con la longitud del campo *fld* del reporte *rp*.

Funciones para manejo de grillas

Output OpenTermOutput (int f_org, int c_org, int height,

int width, attr_type backgr, IFP getcmd)

Obtiene un descriptor para enviar la salida del reporte a la terminal.

Output OpenRpOutput (char* rpname, int ncopies, char* arg)

Obtiene un descriptor para enviar la salida a un reporte.

Output OpenDelimOutput (int dest, char* arg, char* sep, int ncopies)

Obtiene un descriptor para enviar la salida a un archivo o un “pipe” a otro proceso, con los separadores indicados.

Output OpenFmtOutput (int dest, UChar* argv, UChar* head ing,

UChar* footing, int fwidth, int flen,

int topm, int botm, int leftm, int ncopies)

Obtiene un descriptor para enviar la salida a un archivo o a otro proceso, con los separadores estándar.

UShort NColsOutput (Output out)

Devuelve la cantidad de columnas de la salida indicada.

UShort CurrColOutput (Output out)

Devuelve la columna corriente de la salida indicada.

UShort PageWithOutput (Output out)

Devuelve el ancho de página de la salida indicada.

void SetRpOutput (Output out, tp_output out_to, char* arg)

Cambia la salida de un reporte asociado a punto de comenzar. (abrir el reporte con RP_NOBEGIN y usar la función antes de BefinReport).

int SetOutputColumn (Output out, char* title, int wide, type t[,

int long, int ndec])

Define el título y el ancho mínimo para la siguiente columna del reporte.

int ColumnWidth (Output out, int col)

Devuelve el ancho de una columna del reporte.int

OutputColumn (Output out, char* bp)

Coloca un dato en la siguiente columna del reporte.

void CloseOutput (Output out)

Cierra la salida cuyo descriptor se indica.

Funciones Generales

void RpEjectPage (report rpd)

Fuerza un salto de página en un reporte.

void RpSkipLines (report rpd, int n)

Deja una cantidad de líneas en blanco en un reporte.

void RpDrawZ (report rpd, UChar n)

Cancela las últimas líneas de la página.

Tipos de los reportes

Descriptor	Tipo de Variable
reporte	report
campo	rpfield
grilla	Output
Puntero a una función que devuelve un entero	IFP

Colores para el fondo de una ventana (**attr_type**):

A_NORMAL, el color por omisión.

A_REVERSE, fondo en video inverso.

A_RED_BG, rojo.

A_BLUE_BG, azul.

A_GREEN_BG, verde.

A_YELLOW_BG, amarillo.

A_CYAN_BG, cyan.

A_MAGENTA_BG, magenta.

A_WHITE_BG, blanco.

Campos posibles para **RpIFld** (también se pueden utilizar en RpSetFld):

PAGENO, número de página.

LINENO, número de línea dentro del reporte.

FLENGHT, largo de la página.

WIDTH, ancho de la página.

BOTMARG, margen inferior.

TOPMARG, margen superior.

LEFTMARG, margen izquierdo.

Modos para **OpenReport:**

RP_NORMAL, abre el reporte en la forma normal.

RP_SYMBOLS, carga la tabla de símbolos de los esquemas asociados.

RP_COPIES, toma en cuenta los parámetros ncopias y str.

RP_EABORT, si se produce un error al abrir el reporte, aborta el programa.

RP_NOBEGIN, no imprime las zonas que tienen la condición before report. Este flag se debe indicar si se usan las funciones *RpSetOutput* y *SetRpOutput*.

Salidas posibles:

RP_IO_FILE, envía la salida al archivo arg.

RP_IO_PIPE, envía la salida al proceso arg.

RP_IO_DEFAULT, envía la salida al lugar indicado en la variable de ambiente “printer”.

RP_IO_TERM, envía la salida a la terminal.

RP_IO_REPORT, envía la salida a un reporte.

Capítulo 5

Biblioteca “C” de Ventanas (WI)

Funciones para la creación y borrado de ventanas.

window WiCreate (window parent, int f_org, int c_org, int nfile,

int ncols, attr_type border, char* label,

attr_type backg)

Crea una ventana dentro de la pantalla.

void WiDelete (window wi);

Borra una ventana.

void WiDeleteAll ();

Borra todas las ventanas del proceso actual.

window WiSwitchTo (window wi)

Cambia la ventana actual.

Funciones de borrado dentro de ventanas

void WiEraCol ()

Borra hasta el final del renglón.

void WiEraEop ()

Borra hasta el final de la página.

void WiEraLine ()

Borra toda la línea actual.

void WiErase ()

Borra toda la ventana actual.

void WiDelChar (int n, int pos)

Borra una cantidad de caracteres desde la posición actual.

void WiDelLine (int n)

Borra una línea de la ventana actual.

Funciones para manejo de atributos

attr_type WiGetAttr ();

Devuelve el atributo de la ventana actual.

void WiSetAttr (attr_type at);

Pone un atributo a la ventana actual.

void WiSetBackGr (attr_type at);

Pone un atributo al fondo de la ventana actual.

attr_type WiInAttr (int row, int col);

Devuelve el atributo de un carácter de la ventana actual.

void WiSetBorder (attr_type at, char* label)

Pone un atributo y un título a la ventana actual.

Funciones para obtener parámetros de ventanas

int Wicol (window wi)

Devuelve el número de columna donde está el cursor dentro de una ventana.

int WiHeight (window wi)

Devuelve la cantidad de filas que ocupa una ventana.

int WiLine (window wi)

Devuelve el número de fila donde está el cursor dentro de una ventana.

int WiOrgCol (window wi)

Devuelve el número de columna donde empieza una ventana.

int WiOrgRow (window wi)

Devuelve el número de fila donde empieza una ventana.

window WiParent (window wi)

Devuelve la ventana padre de una ventana dada.

int WiWidth (window wi)

Devuelve la cantidad de columnas que ocupa una ventana.

Funciones para moverse dentro de una ventana

void WiMoveTo (short f, short c);

Mueve el cursor a una posición determinada de la ventana actual.

void WiRMoveTo (short f, short c)

Desplaza el cursor una cantidad de filas y columnas a partir de la posición actual.

Funciones para manejar “scrolling” en ventanas

void WiSetScroll (int top, int bot, int left, int right)

Define la región de la ventana actual donde se realizará el scroll.

void WiScroll (int n)

Hace un scroll de la ventana actual en una cantidad de filas hacia arriba o hacia abajo.

Funciones para manejo de menúes

int PopUpMenu(int nrows, int ncols, char* label,

CPFPCPI get_optx,char* garg, FPCPPI execf,

char* earg[], int attrib);

Despliega un menú popup y devuelve el número de opción elegida; el texto de las opciones lo suministra la función get_optx.

int PopUpVMenu (int nrows, int ncols, char* label, char* argv[])

Despliega un menú popup y devuelve el número de opción elegida; el texto de las opciones está guardado en el vector argv.

int PopUpLMenu (int nrows, int ncols, char* label, ...)

Despliega un menú popup y devuelve el número de opción elegida; el texto de cada opción se indica como un parámetro en la llamada.

int PopUpDbMenu (int height, int width, char *label, dbcursor c,

int nparts, IFP func, FPCP dspFunc)

Despliega un menú popup con una lista de registros. func es la función usada para validar los registros. dspFunc es la función usada para desplegar cada línea del menú.

Funciones para manejo de mensajes

int WiMsg (char* fmt, ...)

Imprime un mensaje dentro de una ventana, espera a que se apriete una tecla y borra la ventana.

int WiVMsg (int flag, char* fmt, va_list ap)

Imprime los elementos de una va_list dentro de una ventana. Luego espera a que se apriete una tecla o deja el mensaje en la pantalla.

void DisplayMsg (bool wait_conf, char* fmt, ...)

Despliega un mensaje en la última línea de la pantalla.

void DisplayVMsg (bool wait conf, char* fmt, va_list ap)

Despliega los elementos de una va_list en la última línea de la pantalla.

void ClearMsg ()

Borra el último mensaje de la pantalla.

wdflag WiDialog (wdflag param, wdflag default, char * ti tle,

char * fmt, ...)

Abre una ventana de diálogo.

Funciones para ejecutar comandos del window manager

int WiExecCmd (char* titulo, char* cmd)

Ejecuta un comando usuario del window manager.

int ExecPipe (char* cmd, int nfils, int ncols, attr_type border,

char* label)

Ejecuta un comando a través de un “pipe”, mostrando la salida en una ventana.

int ExecShell (char* cmd)

Ejecuta un comando a través del shell.

int ExecMenu (char* titulo, char* name, int flags)

Ejecuta un menú en una ventana.

Funciones para desplegar archivos

void WiDispFile (char* filename, int nfile, int ncols, char* label)

Muestra un archivo en una ventana.

void WiHelpFile (char* filename, char* label)

Muestra un archivo con extensión “.hlp” en una ventana, buscándolo en un directorio “.../hlp” que pertenezca al PATH.

Funciones para capturar información en ventanas

UChar WiGetc ()

Lee un caracter del teclado; si es un comando del WIndow Maneger, lo procesa y lee otro.

UChar* WiGets (UChar* s)

Lee una cadena de caracteres en la ventana actual. Antes de llamar a esta función, se debe reservar suficiente memoria s.

int WiGetField (type typ, UChar* buff, UShort opt, short length,

short olength, short ndec, char* tst mask,

char* omask);

Define un campo dentro de la ventana actual y espera hasta que se complete su valor. Antes e llamar a esta función, se debe reservar suficiente memoria buff.

UChar WiInChar (int row, int col)

Devuelve el carácter que se encuentra en una posición determinada de la ventana actual.

Funciones para desplegar carcteres y cadenas de caracteres

int WiPutc (UChar c)

Escribe un carácter en la ventana actual.

void WiPuts (char* s)

Escribe una cadena de carcteres en la ventana actual.

void WiPrintf (char* fmt, ...)

Escribe una lista de parámetros (al estilo printf) en la ventana actual.

void WiInsChar (int n, int pos)

Inserta una cantidad de espacios en la posición actual del cursor. El parámetro pos no se utiliza.

void WiInsLine (int n)

Inserta una línea en blanco antes de una fila de la ventana actual.

Funciones para manejo de ayudas

void WiSetAplHelp (char* mod, char* ver)

Establece la ayuda para la aplicación actual.

void WiAplHelp ()

Despliega la ayuda para la aplicación actual según lo indicado con WiSetAplHelp.

void WiHelp (int nfils, int ncols, char* label, CPFPCPI get_txt,

char* garg)

Despliega la ayuda dentro de una ventana.

UChar* WiKeyHelp (UChar k, UChar* s)

Devuelve la descripción de una tecla de Ideafix.

Funciones Generales

void WiBeep ()

Emite un sonido de advertencia.

void WiRefresh ()

Actualiza el contenido de la pantalla con los últimos cambios.

void WiRedraw ()

Vuelve a dibujar toda la pantalla.

window WiCurrent ()

Devuelve el descriptor de la ventana actual.

void WiCursor (bool f)

Muestra u oculta el cursor en la ventana actual.

void WiInterrupts (bool f)

Permite o impide la interrupción del proceso actual.

window WiDefPar ()

Devuelve el descriptor de la ventana padre por defecto.

void WiSetDefPar (window wi)

Establece la ventana padre por defecto.

void WiStatusLine (int f)

Activa o desactiva la línea de estado.

void WiSetTab (int tab)

Establece la medida del tabulador.

int WiGetTab ()

Devuelve la medida actual del tabulador.

int WiSettty ()

Prepara la terminal para trabajar en modo Window Manager.

void WiResettty ()

Devuelve la terminal a su modo normal.

void WiWrap (bool f)

Habilita o deshabilita el modo “wrap”.

int WiGetPassword (char *user, char *password)

Despliega una ventana en la que permite ingresar el usuario y su password. (Versión 4.3.1)

Tipos de las Ventanas

window: descriptor de una ventana.

Atributos (attr_type):

A_NORMAL normal

A_UNDERLINE subrayado

A_BLINK parpadeante

A_BOLD resaltado

A_REVERSE inverso

A_INVISIBLE color del fondo

A_GREEN verde

A_BLUE azul

A_YELLOW amarillo

A_CYAN cyan

A_MAGENTA magenta

A_WHITE blanco

A_RED rojo

A_RED_BG rojo

A_BLUE_BG azul

A_GREEN_BG verde

A_YELLOW_BG amarillo

A_CYAN_BG cyan

A_MAGENTA_BG magenta

A_WHITE_BG blanco

Atributos de Background:

A_NORMAL A_INVISIBLE

A_REVERSE A_RED_BG

A_BLUE_BG A_GREEN_BG

A_YELLOW_BG A_CYAN_BG

A_MAGENTA_BG A_WHITE_BG

Bordes:

Los caracteres de borde son:

SLINE_TYPE Línea simple.

DLINE_TYPE Línea doble.

ASTSK_TYPE Asteriscos.

BLANK_TYPE Blanco.

STAND_TYPE Delineado estándar. (*)

(*) Dependiendo de la capacidad de la terminal será DLINE_TYPE o BLANK_TYPE.

Los indicadores de zona bordeada son:

NO_BORDER No posee borde.

TOP_BORDER Borde superior.

LOW_BORDER Borde inferior.

LEFT_BORDER Borde izquierdo.

RIGHT_BORDER Borde derecho.

ALL_BORDER Posee los cuatro bordes.

Existen bordes definidos por IDEAFIX, estos son:

STAND_BORDER ALL_BORDER+A_REVERSE+STAND_TYPE

SLINE_BORDER ALL_BORDER+SLINE_TYPE

DLINE_BORDER ALL_BORDER+DLINE_TYPE

ASTSK_BORDER ALL_BORDER+ASTSK_TYPE

Atributos para los menúes **PopUp**:

POP_STATIC, No borra la ventana del menú cuando se ejecuta execf.

POP_VOLATIL, Borra la ventana del menú cuando se ejecuta execf.

POP_BORDER, Cambia el borde de la ventana cuando se ejecuta execf.

POP_DEFAULT, Equivale a POP_STATICO| POP_BORDER.

Tipos de campo para **WiGetField**:

TY_NUMERIC, numérico

TY_FLOAT, numérico con punto flotante

TY_DATE, fecha

TY_TIME, hora

TY_STRING, alfanumérico

TY_BOOL, booleano (verdadero o falso)

Opciones para **WiGetField:**

TY_NUMERIC, numérico

TY_FLOAT, numérico con punto flotante

TY_DATE, fecha

TY_TIME, hora

TY_STRING, alfanumérico

TY_BOOL, booleano (verdadero o falso)

Modos para **ExecMenu:**

MENU_MENU, deja la ventana del menú en la pantalla al llamar a otro programa de aplicación.

MENU_PGM, borra la ventana del menú de la pantalla al llamar a otro programa de aplicación.

MENU_MSGERR, si no existe el archivo indicado, muestra un mensaje de error.

MENU_DEFAULT, equivale a MENU_MENU|MENU_MSGERR.

Modos para **WiVMsg**:

MSG_WAIT, espera hasta que se apriete una tecla, y luego borra la ventana.

MSG_PRESERVES, deja la ventana en la pantalla.

MSG_RELEASE, quita la ventana anterior de la pantalla.

Los Modos para los parámetros param y default de la función **WiDialog** son:

Modificadores Normales:

- WD_OK
- WD_YES
- WD_NO
- WD_ABORT
- WD_CANCEL
- WD_RETRY
- WD_IGNORE

Modificadores Especiales:

- WD_ERROR, el fondo se torna rojo. En Windows aparece un ícono de STOP.
- WD_PRESERVE, la ventana activa es preservada hasta la próxima llamada a WiDialog. Esta opción no puede combinarse con modificadores normales, ni puede alternar con la ventana padre.
- WD_RELEASE Cierra una ventana abierta previamente con el modificador WD_PRESERVE.
- WD_HERE la ventana aparece en una posición que depende de la posición del cursor, en vez de estar centrada.

Capítulo 6

Funciones Generales (GN)

void* Alloc (unsigned n);

Obtiene una zona de memoria libre.

FPCPIU SetAllocHandler (FPCPIU f)

Define la función que llama Alloc cuando no queda más memoria libre.

char* BaseName (char* path)

Separa el nombre de archivo de un path completo.

void SetCheckFactor (long n)

Asigna el factor verificador para CheckDigit y TestCheckDigit.

int CheckDigit (long n);

Calcula el dígito verificador para un número.

FPCP SetConvHandler (FPCP f)

Establece el manejador de error de conversión de tipos.

IFPICCPCCP SetMessager (IFPICCPCCP f)

Define la función que utilizan Error y Warning para mostrar los mensajes.

void Error (char* msg[, args ...]);

Muestra un mensaje y aborta un programa.

void Warning (char* msg[, args ...])

Muestra un mensaje y espera hasta que se apriete una tecla.

FILE* FopenPath (char* name, char* ext, char* mode)

Abre un archivo buscando según la variable de ambiente PATH.

int Locate (char* data, char* table, UShort ne1, UShort size, FP rut)

Busca en forma binaria en una tabla en memoria.

char* CUserId (char* s)

Obtiene el nombre de login del usuario actual. Antes de llamar a esta función, se debe rservar suficiente memoria en s.

int GetUid ()

Obtiene el número del usuario actual.

int GetPWEntry (int userid, char *s)

Busca un usuario en el archivo /etc/passwd. Antes de llamar a esta función, se deb reservar suficiente memoria en s.

char* UserName (int uid)

Obtiene el nombre de login de un usuario, dado su número.

char* FullUserName (int uid)

Obtiene el nombre de login de un usuario, dado su número.

int UserId (char* name)

Obtiene el número de un usuario, dado su nombre de login.

FILE* OpenPrinter (int ncopies, char* x)

Abre un canal de salida por impresora.

void ClosePrinter (FILE* p)

Cierra un canal de slaida por impresora.

FPCPCPI SetReadEnvHandler (FPCPCPI f)

Establece el manejador de errores en la lectura de una variable de ambiente.

void OnStop (FP fun)

Agrega una función a la lista de funciones que ejecutará Stop.

void Stop (int state)

Termina el programa, llamando antes a todas las funciones indicadas con OnStop.

long TestCheckDigit (char* s)

Comprueba que el número guardado en una cadena de caracteres contiene un dígito verificador válido.

long ItoL (short i)

Convierte un short en un long.

double ItoF (short i)

Convierte un short en un double.

double LtoF (long i)

Convierte un long en un double.

short LtoI (long i)

Convierte un long en un short.

short FtoI (double i)

Convierte un double en un short.

long FtoL (double i)

Convierte un double en un long.

long FreeMem (void)

Devuelve la cantidad de memoria disponible en bytes.

void FreeMemWi (void)

Muestra en una ventana la cantidad de memoria disponible, y espera hasta que se presione una tecla.

bool GetUpdatePerm()

bool GetAddPerm()

bool GetDelPerm()

Retornan TRUE si el proceso tiene permiso de actualización, inserción y borrado, respectivamente.

int ModuleIsInstalled (int mo)

Retorna ERROR si el módulo no existe; FALSE si el módulo existe y no está instalado, y TRUE si el módulo existe y está instalado.

int ModuleExists (int mod, char *esquema)

La función puede retornar ERROR si el módulo no existiera (en este caso el contenido del string quedará indefinido), u OK si el módulo existiese.

bool ModuleOk (int modid)

Retorna TRUE si el módulo efectivamente está instalado y el esquema testigo de dicho módulo puede ser accedido por el programa, y FALSE en el caso que el módulo no esté instalado o esté instalado de forma incorrecta.. (Versión 4.3.1)

char **ProcArgs()

Retorna argv.

Int ProcNArgs()

Retorna argc.

TIME ProcInitTime()

Retorna la hora de comienzo del proceso.

DATE ProcInitDate()

Retorna la fecha de comienzo del proceso.

Int ProcPid()

Retorna el “id” del proceso.

bool ProcSaveMem()

Retorna TRUE si el proceso esta en modo “save memory”

Int ProcSig()

Retorna el número de señales que hacen que el proceso aborte.

const char *ProcTtyName()

Retorna el nombre de la terminal donde está corriendo el proceso.

const char *ProcTty()

Retorna el nombre de la terminal donde el proceso realiza I/O con el **wm**.

UShort ProcUserGid()

Retorna el “group id” del proceso.

UShort ProcUserId()

Retorna el “user id” del proceso.

bool CrackerClient()

Retorna TRUE si el programa está corriendo en modalidad crackery FALSE en caso contrario. (Versión 4.3.1)

const char *NMethods (const char *server)

Esta rutina se utiliza cuando es necesario conocer la cantidad de mensajes que un cliente

ejecuta sobre Essentia, en general con propósitos de *debug*. (Versión 4.3.1).

Capítulo 7

Funciones para Fecha y Hora (TM)

Las fechas se representan con una variable de tipo DATE que almancena el número de días transcurridos desde el 01/01/84 (fecha cero).

El valor puede ser negativo, para indicar una fecha al 01/01/84.

Una hora se almacena en una variable de tipo TIME que tiene la cantidad de segundos transcurridos en el día, dividido dos.

Funciones para manejo de fechas

void SetDateFmt(int n)

Establece el formato para las fechas.

int GetDateFmt()

Devuelve un entero, que indica el formato de fecha en uso.

char* DayName(DATE fec)

Obtiene el día de la semana de una fecha.

char* MonthName(int n)

Obtiene el nombre de un mes.

DATE StrToD(char* s)

Convierte una cadena de caracteres en una fecha.

DATE StrNXToD(char* s)

Convierte una cadena de caracteres en una fecha, sin expandir TODAY_STR.

void DToStr(DATE d, char* dest, int fmt)

Convierte una fecha en una cadena de carcteres. Antes de llamar a esta función, se debe reservar suficiente memoria en dest.

DATE DMYToD(short day, short month, short year)

Convierte un día, mes y año en una fecha.

void DToDMY(DATE d, short* day, short* month, short* year)

Separa el día, mes y año de una fecha.

DATE FirstMonthDay(DATE d)

Obtiene el primer día del mes de una fecha.

int HalfMonth84(DATE d)

Obtiene el número de quincena desde 01/01/1984.

DATE Today()

Obtiene la fecha corriente.

DATE LastMonthDay(DATE d)

Obtiene el último día del mes de una fecha.

long MonthDiff(DATE a, DATE b)

Obtiene la diferencia entre los meses de dos fechas.

long Week(DATE fec)

Obtiene el número de semana y el año de una fecha.

DATE AddMonth(DATE d, int n)

Suma una cantidad de meses a una fecha.

int Day(DATE fec)

Obtiene el número de día de una fecha.

int Month(DATE fec)

Obtiene el número de mes de una fecha.

int Year(DATE fec)

Obtiene el año de una fecha.

char* IsHoliday(DATE d)

Comprueba si una fecha es feriado.

Funciones para manejo de horas

TIME Hour()

Obtiene la hora actual.

TIME StrToT(char* s)

Convierte una cadena de caracteres en una hora en formato interno.

TIME StrNXToT(char* s);

Convierte una cadena de caracteres en una hora en formato interno, sin expandir HOUR_STR.

void TToStr(TIME t, char* dest, int fmt);

Convierte una hora en una cadena de carateres. Antes de llamar a esta función se debe reservar suficiente memoria en dest.

Funciones para manejo de time stamps

TIMESTAMP StrToTS (char* s)

Convierte una cadena de caracteres en el formato “ddmmyyyy-hhmmss-nsec- uid” en un valor de tipo TIMESTAMP.

void TSToStr (TIMESTAMP ts, char* s)

Convierte un valor de tipo TIMESTAMP en una cadena de caracteres.

Tipos para fechas y horas

Formatos para SetDateFmt y GetDteFmt:

DFMT_INTERNAT, formato internacional (dd/mm/aa)

DFMT_AMERICAN, formato americano (mm/dd/aa)

Formatos para TToStr:

TFMT_SEPAR, usa “:” como separador de horas, minutos y segundos.

TFMT_SECONDS, incluye los segundos en el resultado.

TFMT_HS12, indica la hora de 1 a 12 y agrega “AM” o ‘‘PM”.

Formatos para **DToStr**:

DFMT_SEPAR, usa “/” como separador del día, mes y año.

DFMT_YEAR4, expresa el año con cuatro dígitos.

DFMT_STANDART, es equivalente a DFMT_SEPAR.

DFMT_LDAYNAME, agrega el nombre completo del día.

DFMT_LMONTHNAME, agrega el nombre completo del mes.

DFMT_DAYNAME, agrega una abreviatura del nombre del día: ‘‘LUN", “MAR”, “MIE”, etc. a

DFMT_MONTHNAME, Agrega una abreviatura del nombre del mes: “ENE”, “FEB”, “MAR”, etc.

Capítulo 8

Funciones para manejo de Strings (ST)

int StrCmp (char* s1, char* s2)

Compara dos cadenas de caracteres.

int StrNCmp (char* s1, char* s2, int n)

Compara los primeros caracteres de dos cadenas.

void SetStrCmp (IFPCPCP funct)

Define la función utilizada por StrCmp para comparar.

void SetStrNCmp (IFPCPCPI funct)

Define la función utilizada por StrCmp para comparar.

int StrDspLen (char* s)

Obtiene la longitud de despliegue de una cadena de caracteres.

void IToStr (int val, char* s)

Convierte un int en una cadena de caracteres. Antes de llamar a esta función se debe reservar suficiente memoria en s.

void LToStr (long val, char* s)

Convierte un long en una cadena de caracteres. Antes de llamar a esta función se debe reservar suficiente memoria en s.

void FToStr (double val, char* s)

Convierte un double en una cadena de caracteres. Antes de llamar a esta función se debe reservar suficiente memoria en s.

void FmtNum (char* output, char* input, int long, int ndec, int sep)

Formatea un número con puntos y comas. Antes de llamar a esta función se debe reservar suficiente memoria en output..

void StrToUpper (char* s)

Convierte una cadena de caracteres a mayúsculas.

void StrToLower (char* s)

Convierte una cadena de caracteres a minúsculas.

void NumToTxt (double m, void* x, int f, int c, bool final,

char* front, char* below)

Convierte un número en su descripción literal.

short StrToI (char* s)

Convierte una cadena de caracteres en un short.

long StrToL (char* s)

Convierte una cadena de caracteres en un long.

double StrToF (char* s)

Convierte una cadena de caracteres en un double.

int CompileMask (char* mask, char* tstmask, char* omask)

Compila una máscara de testeo de cadenas de caracteres.

int SizeOfMask (char *mask)

Obtiene la longitud real de un campo enmascarado (que posee el atributo ´mask´). También se puede usar para obtener la longitud de los *buffers* utilizados por la función CompileMask. (Versión 4.3.1)

char* ReadEnv (char* name)

Lee una variable de ambiente.

char* StrTxt (char* s, int n)

Separa en partes una cadena de caracteres.

bool RexpMatch (UChar* rexp, UChar* s, int flag)

Compara una cadena de caracteres con una expresión regular limitada.

Capítulo 9

Funciones para manejo de NUM (NM)

NUM* NumAdd (NUM* a, NUM* b)

Suma un valor de tipo NUM a una variable de tipo NUM.

NUM* NumAddL (NUM* a, long b)

Suma un valor de tipo long a una variable de tipo NUM.

NUM* NumAddF (NUM* a, double b)

Suma un valor de tipo double a una variable de tipo NUM.

NUM* NumSub (NUM* a, NUM* b)

Resta un valor de tipo NUM a una variable de tipo NUM.

NUM* NumSubL (NUM* a, long b)

Resta un valor de tipo long a una variable de tipo NUM.

NUM* NumSubF (NUM* a, double b)

Resta un valor de tipo double a una variable de tipo NUM.

NUM* NumMul (NUM* a, NUM* b)

Multiplica una variable de tipo NUM por un valor de tipo NUM.

NUM* NumMulL (NUM* a, long b)

Multiplica una variable de tipo NUM por un valor de tipo long.

NUM* NumMulF (NUM* a, double b)

Multiplica una variable de tipo NUM por un valor de tipo double.

NUM* NumDiv (NUM* a, NUM* b)

Divide una variable de tipo NUM por un valor de tipo NUM.

NUM* NumDivL (NUM* a, long b);

Divide una variable de tipo NUM por un valor de tipo long.

NUM* NumDivF (NUM* a, double b);

Divide una variable de tipo NUM por un valor de tipo double.

void SetNFld (dbfield fn, NUM* val[, int row])

Da el valor val (de tipo NUM) al campo fn de la base de datos.

long NumCmp (const NUM* a, const NUM* b)

Compara dos valores de tipo NUM.

long NumCmpL (const NUM* a, const long b)

Compara un valores de tipo NUM con otro de tipo long.

long NumCmpF (const NUM* a, const double b)

Compara un valores de tipo NUM con otro de tipo double.

void IToNum (short val, NUM* n)

Convierte un valor de tipo short a otro de tipo NUM.

void LToNum (long val, NUM* n)

Convierte un valor de tipo long a otro de tipo NUM.

void FToNum (double val, NUM* n)

Convierte un valor de tipo double a otro de tipo NUM.

void StrToNum (const char* s, NUM* n)

Convierte una cadena de caracteres en un valor de tipo NUM.

long NumToL (const NUM*n)

Convierte un valor de tipo NUM a otro de tipo long.

double NumToF (const NUM*n)

Convierte un valor de tipo NUM a otro de tipo double.

const char* NumToStr (const NUM*n, int tamaño, int prec)

Convierte un valor de tipo NUM en una cadena de caracteres.

Capítulo 10

Manejo de Mensajes (Versión 4.3.1)

Formato de los Mensajes

Los mensajes deben tener el siguiente formato:

“Es un mensaje, para insertar #0 se debe digitar #1”

Luego los caracteres ‘#’ serán reemplazados según los parámetros pasados como argumento, correspondiendo al número que lo acompaña (ej.: #0 y #1).

Los ‘#?’ pueden estar en cualquier orden dentro de los mensajes y estos serán reemplazados en forma correcta.

La cantidad máxima de parámetros soportada es de 10, es decir desde #0 hasta #9.

NOTA: La única característica, del estilo printf, soportada por los mensajes es %K_TECLA, cualquier otra característica (%d, %f, etc) tendrá resultado indefinido. La impresión literal del carácter “%”se logra colocando dos símbolos (%%).

Mensajes en forms

En la sección %form se puede colocar un conjunto de tablas de mensajes. Los mensajes pueden ser usados más tarde, desde las expresiones de los forms. La sintaxis es:

msgtable tab(“tab” [, “sec”]) [, ...];

donde “tab” y “sec” son los nombres de la tabla y la sección, respectivamente. En caso de no especificar el nombre de la sección, dicho nombre será igual al del form (por omisión).

Funciones para Mensajes

Funciones Disponibles en C

const char *GetMessage(const char *tab, const char *section,

const char *msgid)

Esta rutina retorna el mensaje correspondiente al identificador de mensaje ¨*msgid*¨ dentro de la tabla de mensajes ¨*tab*¨ sección ¨*sec*¨ para el lenguaje sobre el cual se está trabajando (especificado en $LANGUAJE).

void MessageDisplay (const char *tab, const char *section,

const char *label, const char *msgid, ...)

Muestra una ventana de diálogo con el *label* pasado como parámetro, con un mensaje cuyo identificador es ¨*msgid*¨ de la tabla de mensajes ¨*tab*¨ sección ¨*section*¨ para el lenguaje definido por la variable de ambiente ´LANGUAGE´, pasándole los argumentos opcionales que se encuentran detrás del argumento ¨*msgid*¨.

void MessageDisplayError (const char *tab,

const char *section,

const char *msgid, ...)

Muestra una ventana de diálogo que es similar a la rutina *Error*, pero sin abortar el programa, que incluye un mensaje.

void MessageFatalError (const char *tab, const char *section,

const char *msgid, ...)

Esta función muestra una ventana de diálogo similar a la rutina *Error*, abortando el programa.

Funciones Disponibles en C++

MsgTable tabn(“tab”, “sec”);

El primer parámetro‘*tab*’ indica el nombre de la tabla y ‘*sec*’ la sección. Si no se especifica la sección, la función asume como sección por omisión ¨*main*¨, de este modo se genera un objeto tabn y para obtener un mensaje se puede ejecutar:

tabn(“MSG”)(params);

donde los parámetros params son los argumentos que se utilizan para reemplazar los ´#?´ dentro del mensaje.

Al objeto se le podrán aplicar funciones para su despliegue, como ser:

- tabn (“MSG”)(params).display();

Muestra una ventana de diálogo similar a la de la rutina *Warning* pero sin *label* en la ventana, debido a que la función display no recibe parámetro.

- tabn (“MSG”)(params).display(tab(“WARNING”));

Muestra una ventana de diálogo similar a la de la rutina *Warning* con *label* igual al mensaje pasado como parámetro (WARNING), que puede ser obtenido de otra tabla de mensajes.

- tabn (“MSG”)(params).displayError();

Muestra una ventana de diálogo similar a la de la rutina *Error* pero no aborta el programa. La función displayError no recibe parámetro.

- tabn (“MSG”)(params).fatalError();

Se comporta igual que la rutina *Error*. La función fatalError no recibe parámetro.

Los mensajes pueden concatenarse entre sí y con constantes, como así también recibir parámetros de diferentes tipos.

Capítulo 11

Caminos de desarrollo en IdeaFix

Utilitarios de Desarrollo

DGEN

Creación y Modificación de esquemas de Base de Datos.

Sintaxis

dgen [-b] [-c 'comando'] [-f] [-h] [-m] [-v] [-p] [-s] [-n] [-l] [-k] archivo[.sc] ...

Descripción

Este utilitario reconoce un subconjunto de IDEAFIX Query Language (IQL, o Lenguaje de Consulta IDEAFIX) llamado DDS (Sentencias de Definición de Datos). Estas sentencias son las que permiten crear y modificar las estructuras de bases de datos.

Opciones

-b no despliega el mensaje identificatorio

-c genera el encabezado (*header*) para un programa en COBOL que usará IDEAFIX o Essentia.

-c2 lo mismo que -c, pero agrupando todas las tablas en un registro con el nombre del esquema.

-f modifica de manera incondicional un esquema existente, lo que puede ocacionar pérdida de información.

-h se genera un archivo cabecera (con extensión ".sch") para ser incluido en programas desarrollados en ¨C¨ que utilizan esquemas ya genera- dos y que trabajen con Ideafix ó con

Essentia.

-m modifica un esquema existente. Si no se puede convertir toda la infor mación, se detiene el programa.

-v *verbose*. Mediante esta opción se muestran las operaciones a medida que son ejecutadas.

-p genera o modifica el esquema asignando todos los permisos a todos los usuarios.

-s toma la entrada de stdin y envía la salida a stdout.

-n abre todos los esquemas sin las descripciones, para ahorrar memoria.

-l modifica el esquema existente aunque esté bloqueado.

-k efectúa un *checkpoint* sobre cada esquema creado.

FGEN

Compila archivos fuentes FDL.

Sintaxis

fgen [-b] [-h] [-t] [-w] archivo[.fm] [archivo ...]

Descripción

Este programa convierte la definición en FDL de un formulario a la representación usada por IDEAFIX en tiempo de ejecución.

La definición de un formulario está contenida en un archivo de definición con la extensión ".fm", conocido como archivo fuente FDL.

Opciones

-b No se muestra el mensaje de identificación del utilitario ("banner").

-h Genera un archivo cabecera ("header") con extensión ".fmh", para ser incluido en los programas "C" que utilicen el formulario. Dicho archivo también será generado automáticamente se incluye la claúsula langua ge en el formulario.

-t Esta opción indica que se envíen los mensajes a la salida estándar.

-w Permite obtener mensajes de advertencia ("warnings"), tales como diferencias entre campos del formulario y los de la Base de Datos, superpo sición de atributos, etc.

RGEN

Compila archivos fuente en RDL

Sintaxis

rgen [-b][-h][-w][-f] archivo[.rp] [archivo...]

Descripción

La especificación de un reporte está contenida en un archivo con extensión ".rp", conocido como archivo fuente RDL (Report Definition Language).

El utilitario *rgen* genera un archivo con estensión ".rpo" y en forma opcional uno con extensión ".rph", que será el archivo cabecera que deberá ser incluido en los programas ¨C¨ que utilicen el reporte.

Opciones

-b no se muestra el mensaje identificatorio del utilitario.

-*h* se genera un archivo cabecera

-w advertencias acerca del reporte que está siendo compilado.

-f se imprime la lista de los campos del reporte que no han sido decla rados.

IQL

Sintaxis

iql [-b] [-c] [-f] [-h] [-m] [-v] [-p] [-s] [-n] [-l]

[-k] <archivo> [<archivo>]

Descripción

Este utilitario interpreta las sentencias del lenguaje SQL estándar en su totalidad, más algunas extensiones muy útiles provistas por IDEAFIX.

El *iql* puede ser invocado de dos formas distintas:

- Interactivamente, donde por medio de un editor similar al *Dalí* de IDEAFIX el usuario introduce los comandos, o
- ejecutando el utilitario con un archivo conteniendo sentencias SQL.

Opciones

-b mediante esta opción no se muestra el mensaje de identificación del utilitario.

-c se ejecuta, la cadena de caracteres siguiente a esta opción, como un comando.

-f fuerza la modificación de esquemas existentes.

-h si la opción -h es indicada en la línea de comandos, se generarán los archivos cabecera de los esuqemas.

-m indica que se están modificando esquemas existentes.

-v se muestran las distintas operaciones a medida que se van ejecuntando.

-p genera (o modifica) el esquema asignando todos los permisos a todos los usuarios.

-s toma la entrada de stdin y envía la salida a stdout.

-n abre todos los esquemas sin las descripciones, para ahorrar memoria.

-l modifica el esquema existente aunque está bloqueado.

-k efectuar un checkpoint sobre cada esquema creado.

TESTFORM

Prueba la especificación de un form.

Sintaxis

testform form

Descripción

Este utilitario permite la prueba de un diseño de formulario, es decir, se obtiene un prototipo del comportamiento de la pantalla, de acuerdo a lo especificado en la definición del formulario.

No se pueden consultar datos existentes,y como es lógico, ni actualizar las tablas asociadas al formulario.

GENFM

Generador de formularios

Sintaxis

genfm [-b] [-c] [-f] [-g] [-i indice] [-o archivo] [-l leng] esquema[.]tabla.SE

Descripción

El utilitario genfm toma una tabla de un esquema existente, y genera una especificación de formularios en FDL (Form Definition Language) para realizar operaciones sobre dicha tabla. El archivo se genera con el nombre de la tabla especificada mediante “esquema.tabla” y se le agregará la extensión “.fm”, a menos que se use la opción “-o” para indicar en qué archivo se dejará la especificación generada.

Opciones

-b Mediante esta opción no se muestra el mensaje de identificación.

-f Dibuja los campos numéricos sin punto de miles.

-c Permite sobre-grabar el archivo destino si éste existiera.

-g Esta opción sirve para indicar que se procese el formulario generado con el utilitario *fgen*.

-i índice: Indica que se utilicen los campos que forman parte de la cla- ve índice como claves del formulario que se está generando.

-o archivo: El formulario generado se grabará en ``archivo.fm".

-l leng: Incluye en las especificaciones generales del formulario la cláusula language, con el lenguaje indicado en leng.

GENCF

Generador de programa “C”

Sintaxis

gencf [-a] [-b] [-f] [-p] [-i] [-jcampoI=campoJ[,cam poN=campoM]] [-o archivo] form

Descripción

El utilitario *gencf* permite generar un programa en “C” a partir de un formulario compilado, es decir que utilizará el archivo con extensión “fmo”, obtenido con el utilitario *fgen*.

Se pueden generar dos tipos de programas:

- Listador. Cuando el formulario no tiene zona de claves.
- ADM (ver abajo). Cuando el formulario tiene una zona de claves.

Opciones

-a Incluye rutina after field (ex postcond de PREFIX).

-b Incluye rutina before field (ex precond de PREFIX).

-f Fuerza la reescritura del archivo destino si éste existiera.

-g Genera las funciones con prototipos.

-i No imprime el mensaje de identificación.

-j *campoI=campoJ [,campoN=campoM]*

Esta opción indica que el programa trabajará con dos tablas relaciona das entre sí mediante los campos indicados. De ellos, campoI y cam poN pertenecen a una de las tablas; campoJ y campoM pertenecen a la otra. archivo.

-o archivo Genera el programa como `archivo.c'. En caso de no informar la op ción “-o”, el nombre del programa generado es el mismo del formula rio con extensión “.c”.

Los campos de tipo SKIP o DISPLAY ONLY no darán lugar a entradas en los “switch” (programación CFIX) de las rutinas after-field y before-field. Las sentencias especificadas en el formulario que sean del tipo: ignore, add, delete, update, etc., no generarán la entrada correspondiente en el “switch” del cuerpo del programa.

CFIX

Compila y encadena archivos fuentes en C.

Sintaxis

cfix [-b] [-f] [-g] [-n] [-c] [-e] [-o nombre] [-s] [-I directorio] [-x opciones] [-y opciones] [-E] [-S] modulo... [-l biblioteca...]

Descripción

Este utilitario permite compilar y encadenar módulos de programas. Si la extensión de los módulos es .c primero los compila y luego los vincula ("linkedit") junto a los que tengan extensión .o.

Opciones

-b No muestra identificación de versión.

-f Fuerza compilación de los programas ¨C¨ aún si el módulo fuente no ha sido modificado. Esto puede ser necesario si se han alterado archi - vos de encabezamiento (v.gr.: .fmh de formularios) que al contener constantes cuyo valor ha cambiado obligan a la recompilación del mó- dulo.

-g Compilar para debugging.

-n Sólo mostrar los comandos sin ejecutar.

-c No realizar la etapa de link edición.

-e No agregar la extensión .exe al programa ejecutable.

-o *nombre* El archivo ejecutable resultante se llamará ¨nombre¨. Para que incluya la extensión .exe debe especificársela directamente al dar el nombre; es de dir, debe indicarse nombre.exe. No cabe aquí por consiguiente el uso de la opción **-e**.

-s No utilizar la biblioteca compartida. Esto toma en cuenta que ella se en cuentra operativa sólo en sistemas con “shared libraries”.

-l *directorio* Se indica el camino de un directorio de donde se obtendrán archivos de include.

-y *opciones* Se pasa una cadena que se utilizará como parámetro del linkeditor.

-x *opciones* Se pasa una cadena que se utilizará como parámetro del compilador.

-E Realiza el link con la biblioteca de acceso a Essentia (el server de base de datos de InterSoft Argentina).

-S Realiza el link con la biblioteca de acceso a IDEAFIX.

-l *librería* Utiliza la biblioteca de funciones especificada.

DBDES

Diseñador interactivo de bases de datos.

Sintaxis

dbdes

Descripción

Este utilitario permite la creación o modificación de un esquema de una forma muy amigable, haciendo uso intensivo de menúes para guiar al desarrollador en cada paso.

Esto es sólo para IDEAFIX, no funciona con Essentia.

IDEAFIX

Ambiente de desarrollo guiado por menúes 4GL.

Sintaxis

ideafix

Descripción

Este comando ingresa al Ambiente de Desarrollo de IDEAFIX: un sistema de menúes que permite acceder a todas las herramientas que componen dicho ambiente.

Utilitarios del sistema de ejecución

DOREPORT

Obtiene unreporte de los datos de entrada.

Sintaxis

doreport [-b] [-s] [-n número] [-F caracter] [-R caracter] [-d] <archivo[.rpo]>

Descripción

Doreport es un utilitario que lee una secuencia de registros de datos por su entrada estándar. Esos datos son los que alimentan la definición del reporte que se le pasa como parámetro.

Opciones

-b no muestra el mensaje de identificación

-s no muestra el mensaje de identificación ni los mensajes de error.

-n *número* emite una cantidad determinada de copias.

-F *caracter* define el caracter separador de campos (por omisión “,”)

-R *caracter* define el caracter separador de registro (por omisión “\n”).

-d define “,” como separador de campos y “\n! como separador de registros.

EXECFORM

Recibe la entrada y ejecuta el programa.

Sintaxis

execform [-w] [-p] [-d] [-s] <archivo>[.fmo] [<programa>]

Descripción

Este utilitario muestra un formulario en la pantalla, espera a que se completen los datos y llama a otro programa, pasándole los valores de los campos parámetros.

Opciones

-b No muestra el mensaje de identificación

-r Vuelve al formulario después de ejecutar el programa.

-w El programa se ejecutará como un proceso usuario del Window Manager. Es el valor tomado en caso de omisión.

-p [FILAS] [xCOLS] El programa será ejecutado a través del shell, pero capturando su salida en una ventana. Las dimensiones de la misma pueden definirse optativamente a continuación de la letra p, donde FILAS y COLS indican la cantidad de filas y columnas de dicha ventana. Por defecto se asumirá el espacio disponible en pantalla en cada caso.

-d Este modo se denomina *debugging* ya que no se invoca al programa, sino que se muestran los argumentos que se le pasarían al mismo en la parte inferior de la pantalla cuando el usuario digita la tecla .

-s Se ejecuta el programa como si fuese uno del sistema operativo.

WM

Window Manager.

Sintaxis

wm [-b] [-a] [-t timeout] [-T ntasks] [{-w |-r} <archivo>]

cmd [arg....]

Descripción

Este utilitario maneja todas las operaciones de entrada/salida de laterminal en los programas

de aplicación. cmd y args son el nombre y los argumentos del programa que requiere la interfaz de IDEAFIX para el manejo de ventanas.

Opciones

-b No muestra el mensaje de identificación.

-a Suprime la opción abortar del menú que aparece al digitar la tecla interrumpir.

-s Permite el uso de una impresora esclava sobre la misma linea de la terminal.

-T n Limita el número de tareas simultáneas a “n”.'

-S Usa la capacidad 'sgr´ de terminfo

-f Corre en foreground.

-d dev Usa 'dev' como dispositivo de i/o en lugar de /dev/tty

-t secs Setea el timeout en 'secs' segundos. El tiempo por defecto es de cinco segundos.

{-w| -r} f Las opciones -w y -r permiten crear o leer un archivo que contenga instrucciones para ejecutar el proceso cmd en forma automática (por ejemplo, en una demo).

FG

Reactiva la última tarea suspendida del WM.

Sintaxis

fg [-s]

Descripción

Este comando reactiva el último proceso de usuario suspendido, si se está usando el WM en modo residente.

Opciones

-s Establece el comportamiento del WM de la siguiente forma: si hay procesos suspendidos, reactiva el último, sin permitir retornar el shell; si no, desconecta el WM. Una vez que todos los procesos del usuario hayan terminado, desconectará el WM y retornará al Shell.

EXP

Exporta datos en formato ASCII.

Sintaxis

exp [flags] esquema tabla [campo ...]

Descripción

Extrae datos de la base de datos de IDEAFIX, en formato ASCII. Lista los registros de la tabla en la salida estándar. Si se indica una lista de campos, se listan sólo ellos.

Opciones

Los flags pueden ser los siguientes:

-b Suprimir identificación

-c n Usar 'n' k-bytes como memoria cache

-D n Usar 'n' k-bytes como cache de datos

-R c Usar c como separador de registro

-F c Usar c como separador de campo

-S c Usar c como separador de subcampo

-L Usar registros de longuitud fija (tomando la long. de la definición del esquema).

-d Equivalente a -R\n -F, -S,

-f *nomarch* Establece 'nomarch' como archivo de especif. de campos

-e Convierte los doubles en formato 'e'.

-v Cada campo va precedido de su nombre

-i *name* Accede a través del índice 'name'

-V Usar la versión de base de datos especificada (solo para ESSENTIA).

-k d1,d2,..dn:h1,h2,..,hn lista solo desde d1-dn hasta h1-hn

-t Incluye campos de tipo TIMESTAMP.

Output = stdout

IMP

Importa datos en formato ASCII.

Sintaxis

imp [flags] esquema tabla [campo ...]

Descripción

Agrega datos en la base de datos de IDEAFIX. Lee su entrada estándar y escribe en la tabla de la base de datos.

Opciones

Los flags pueden ser los siguientes:

-b Suprimir identificación

-c *n* Usar 'n' k-bytes como memoria cache

-D *n* Usar 'n' k-bytes como cache de datos

-R *c* Usar c como separador de registro

-F *c* Usar c como separador de campo

-S *c* Usar c como separador de subcampo

-L Usar reg. de long. fija (tomando long. de la def. del esquema).

-d Equivalente a -R\n -F, -S,

-f *nomarch* Establece 'nomarch' como archivo de especif. de campos

-x Dar de baja los registros importados.

-l No bloquear la tabla

-g Ignorar errores para numeros fuera de rango (dejando en nulo).

-G *n* Usar transacciones que encierren n registros.

Input = stdin

DOFORM

Opera con un formulario

Sintaxis

doform [-b] [-l] [-jCampoI=CampoJ [,...]] formulario

Descripción

Este utilitario toma un formulario co,pilado y realiza operaciones en la base de datos con él.

Opciones

-b No imprime el mensaje de identificación

-l Bloquea la tabla

-j *CampoI=CampoJ [,...]* Establece los campos “join” cuando se tiene un ABM a dos tablas con multirrenglón.

FRECOVER

Reconstruye los índices de las tablas.

Sintaxis

frecover [-b][-v][-f][-u][-cN][-dN] esquema[.|][tabla] [indice...]

Descripción

Este comando se utiliza para reconstruir las tablas y sus índices, de un esquema de base de datos, frente a situaciones tales como la pérdida de índices.

Opciones

-b No emitir el mensaje identificatorio.

-v Verbose. Muestra las operaciones a medida que son ejecutadas.

-f Colocar el contador de accesos concurrentes del esquema en 0.

-u Forzar la reconstrucción de los índices aunque no estén marcados como corruptos.

-p Comprime la tabla removiendo registros borrados.

-cN Usar N k-bytes de cache para los índices.

-dN Usar N-kbytes de cache para los datos.

-r[índice] Comprime una tabla usando *index*.

IXFG

Reactivar la última tarea suspendida (WM).

Sintaxis

ixfg [-s]

Descripción

Este comando es utilizado para reactivar el último proceso usuario suspendido en modalidad residente.

- Si no hubiera procesos suspendidos desconectará al WM.
- Si los hubiera, reactivará al último suspendido y no permitirá “Retornar al Shell”. Una vez que todos los procesos usuarios hayan terminado, desconectará el WM y retornará al shell.

La invocación a este comando con el argumento “-s”, puede ser sustituido por el comando *wmstop* de idéntico comportamiento.

MENU

Intérprete de especificaciones de menúes.

Sintaxis

menu titulo archivo[.mn]

Descripción

Este utilitario muestra una ventana menú, de acuerdo a la especificación de menú dada en archivo. El argumento titulo es utilizado como etiqueta de la primera ventana menú. En la línea de información de la pantalla se mostrará el nombre de la empresa que se toma de la variable de ambiente EMPRESA.

El usuario podrá seleccionar una opción de las presentadas, y lanzarla a ejecución.

DBCHECK

Chequea la consistencia de los datos.

Sintaxis

dbcheck [-b] [-s] esquema[.] {tabla,*}

Descripción

Este comando verifica la consistencia de los datos cargados en distintas tablas de la base de datos, contra lo especificado en la definición de las mismas. Recorre cada uno de los registros de las tablas que se indican, controlando que se verifiquen los operadores: not null, [not] in table, [not] in, [not] between, operadores relacionales (=, !=, <, >, <= y >=) presentes en la base de datos.

Opciones

-s Silencioso: no muestra los resultados de las operaciones.

-b No imprime el mensaje de identificación.

IPG

Muestra interactivamente archivos por pantalla (IDEAFIX pg).

Sintaxis

ipg [-p proceso] [-wFILASxCOLUMNAS] [archivo...]

Descripción

Este comando permite ver un archivo o una salida de otro comando, dentro de una ventana, dando la posibilidad de desplazarse mediante las teclas de cursor o de paginación.

El *ipg* a diferencia de los comandos de Unix *pg* y *more*, representa correctamente los carácteres gráficos de los archivos o procesos seleccionados.

Opciones

-p *proceso* Esta opción ejecuta el proceso pasado como parámetro y muestra su salida en una

ventana con posibilidad de movimiento por línea y página.

-w *FILASxCOLUMNAS*: Define la salida en el número de filas y columnas especificados.

-**archivo** Permite ver de manera interactiva el conjunto de archivos invocados, los comandos soportados son:

n: archivo siguiente.

p: archivo anterior.

q: terminar.

Es posible definir un conjunto de archivos mediante la utilización de metacarácteres y/o colocando la lista de archivos que se desean ver.

ROLLBACK

Elimina transacciones incompletas.

Sintaxis

rollback

Descripción

Este comando permite eliminar transacciones incompletas, ocasionadas por interrupciones que no pueden ser atendidas por software (corte de energía, caídas del equipo, etc.).

PM

Printer Manager.

Sintaxis

pm [-b] [-iNOM_CAP] [-f] [-o archivo] modelo [archivos ...]

Descripción

El Printer Manager (pm) es un filtro configurable para impresoras. Este programa lee su entrada y traduce carácter por carácter de acuerdo al archivo de capacidades de la impresora “modelo”.

Si no se especifican archivos, se lee la entrada estándar.

La impresión comienza enviando la secuencia INICIALIZAR. Luego se envían las secuencias especificadas en la opciones -i, si las hay, y finalmente se imprimen los archivos en el orden dado.

Luego de cada archivo se envía la secuencia CANCELATRS, y al finalizar la impresión completa se envía FINALIZAR.

Opciones

-b No emitir el mensaje identificatorio.

-iNOM_CAP Enviar la capacidad NOM_CAP antes de imprimir. Puede ser cualquiera de las capacidades especificadas en la tabla de atributos del Printer Manager (Ver tabla 7.3).

Varias de estas opciones pueden ser combinadas.

-f Enviar un form feed luego de imprimir cada archivo.

-o archivo Escribir la salida en archivo

WMTTYKEY

(Versión 4.3.1)

Sintaxis

wmttykey [tty]

Descripción

Este utilitario permite obtener las claves con las cuales son creadas las colas de mensajes y el semáforo utilizado por el Window Manager, en una terminal dada.

Para ello, el utilitario retorna las claves de la terminal pasada como parámetro y, en caso de omitirse el argumento, retorna las claves de la terminal sobre la cual se corre.

Utilitarios para Mensajes

MSGED

Editor de la tabla de configuración de mensajes. (Versión 4.3.1)

Sintaxis

msged

Descripción

La tabla posee los siguientes campos:

- *File*: es el nombre del módulo que contendrá los mensajes. Esta tabla se encuentra en alguno de los directorios incluidos dentro de la variable de ambiente DATADIR.
- *Section*: Indica la sección dentro de la tabla de mensajes que se desea cargar.
- *Language*: es el correspondiente al mensaje de dicho *módulo/sección*.

Estos tres son los campos de la clave. Una vez ingresados se podrá realizar alta, baja o modificación del *módulo/sección* para el lenguaje seleccionado. Para ello se dispone de un multirrenglón en el cual se permite cargar los diferentes mensajes que componen al *módulo/sección*. Dicho multirrenglón posee dos campos:

- *Key*: es la clave del mensaje. Acepta una cadena de hasta 32 caracteres, por la cual será posible obtener el mensaje.
- *Message*: es el texto del mensaje. Puede contener hasta 400 líneas de 128 caracteres cada una.

MSGIMP

Importador de mensajes. (Versión 4.3.1)

Sintaxis

msgimp modulo lenguaje seccion < msgs.txt

Descripción

Permite importar los mensajes a partir de un archivo ASCII para un *módulo/sección/lenguaje* dado.

El formato del archivo ASCII de entrada debe ser:

@(#)@

MSGKEY1

Linea 1

Linea 2

Linea 3

@(#)@

MSGKEY2

Linea 1

Linea 2

Linea 3

....

Donde la secuencia '@(#)@' indica la separación de mensajes, la línea posterior al separador es tomada como clave del mensaje y luego aparecerán en forma consecutiva todas las líneas que pertenecen al mensaje.

MSGEXP

Exportador de mensajes. (Versión 4.3.1)

Sintaxis

msgexp [-k] modulo language seccion key|* msgs.txt

Descripción

Este comando permite exportar a un archivo con formato ASCII los mensajes de un *módulo/sección/lenguaje*, para una clave o para todo el grupo de mensajes.

El formato de salida es el mismo que el explicado para el utilitario antes mencionado.

Opciones

-k indica exportar solamente la clave de cada uno de los mensajes.

Capítulo 12

DMS (Data Manipulation Statements)

INSERT

Agrega registros a una tabla.

insert into tablename(lista_de_columnas) values valores;

DELETE

Borra registros de una tabla.

delete from tablename

where condicion_de_baja;

UPDATE

Modifica el contenido de los registros de una tabla.

update tablename set lista_de_asignaciones

[where condición_de_búsqueda];

SELECT

Selecciona campos y registros de una tabla.

select campo1, campo2, ..., campoN

from tablas

[where condición_de_búsqueda]

[having condición_de_búsqueda]

[order by expresión [asc | desc], ...expresión [ascending | descending], ...]

;

Las condiciones de las cláusulas where y having se forma con los siguientes operadores y operandos:

Operador	Significado
=	Igual a
==	Igual a
!=	Distinto de
<>	Distinto de
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
>< ... and ...	Operador *between*
between ... and ...	Operador *between*
like	Patrones de cadenas
in	Lista de valores
in (subquery)	Condición de Subquery
exist (subquery)	Verifica una tabla específica
comp. op. [any|some|all]	Operador de comparación con la condición all-or-any

Los operadores lógicos que complementan las comparaciones son:

Operador	Significado
not, !	Negación
and, &	Operador and
or, |	Operador or

CHOWN

Cambia el dueño de un esquema.

CHOWN usuario lista_esquemas

GRANT y REVOKE

Asigna y quita permisos sobre esquemas o tablas.

GRANT privilegio ON SCHEMA lista_esquemas

TO lista_usuarios

Los privilegios pueden ser:

Permiso	Autoriza a ...
USE	use
TEMP	select ... order by ...
MANIP	insert-update-delete
ALTER	Modificar estructura del esquema [create] schema [create] table, etc.

La lista de usuarios puede especificarse según las siguientes alternativas:

PUBLIC	Todos los usuarios gozan de los privilegios asignados
GROUP nombre_grupo	El grupo de usuarios denominado nombre_grupo goza exclusivamente de los privilegios asignados.
nombre_usuario	El nombre_usuario es el único que goza de los privilegios indicados.

GRANT privilegio-tab ON nombres-tabla

TO {PUBLIC | usuario | lista-usuarios}

privilegio-tab	Es uno o más de los siguientes permisos (si se es pecifica más de uno, se debe separar con comas): INSERT: Agregar nuevas filas (registros) a la tabla (se hereda) ADD: Idéntico al anterior. DELETE: Borrar filas de la tabla (se hereda). UPDATE: Modificar datos de filas existentes (se hereda). SELECT: Leer datos de filas existentes.

	ALL [PRIVILEGES]: Es sinónimo de todos los an teriores. Puede indicarse simplemente como ALL, o bien como ALL PRIVILEGES.
nombres-tab	Es el nombre de la tabla para la cual se están otor gando privilegios de acceso. Debe pertenecer al es quema corriente; o bien a cualquier esquema activo si se indica “esquema.tabla”. La especificación “es quema.*” se refiere a todas las tablas del esquema. Cuando se indica más de un nombre se deben sep arar con comas.
PUBLIC:	Es el nombre de la tabla para la cual se están otor- gando privilegios de acceso. Debe pertenecer al es quema corriente; o bien a cualquier esquema activo si se indica “esquema.tabla”. La especificación “es quema.*” se refiere a todas las tablas del esquema. Cuando se indica más de un nombre se deben sep arar con comas.
lista-usuarios	Es una lista de nombres de usuarios del sistema separados por comas, a los cuales se les otorgan los permisos indicados.
GRANT OPTION:	Indica que cualquiera de los usuarios que haya re cibido el permiso tendrá la capacidad de pasárselo a otros usuarios. Si así lo desean, pueden a su vez entregar el permiso con GRANT OPTION, para que el receptor pueda igualmente volver a trans mitirlo.

Otras sentencias

exit [código];

Termina la ejecución del iql.

close esquema [,esquema];

Elimina un esquema de la lista de esquemas activos.

commit work;

Termina todas las transacciones pendientes.

define identificador as cadena;

Define un símbolo que será reemplazado por una cadena de caracteres.

exec cadena;

Ejecuta los comandos SQL contenidos en un archivo.

set;

Muestra todas las variables e identificadores definidos.

set output=expresión;

Define la salida de los comandos SQL.

set variable=expresión;

Asigna un valor a una variable del usuario o del iql. Las variables del iql son:

botmarg debug flenght

footmarg fwidth heading

show;

Muestra la lista de esquemas activos;

show esquema[.tabla];

Muestra la estructura de un esquema o de una tabla.

use esquema [, esquema...] [not descr];

Activa uno o más esquemas y convierte al primero en el esquema corriente.

unset identificador;

Elimina una variable o identificador.

wcmd proceso;

Ejecuta un proceso usuario del WM.

shell proceso;

Ejecuta un comando del shell, retornando inmediatamente.

pipe comando;

Ejecuta un comando del shell, mostrando su salida en una ventana.

cd directorio;

Cambia el directorio actual.

Capítulo 13

Las Variables de Ambiente

Variables	Usada por	Propósito
PATH	DBM, RM, FM, WM	Ubicar utilitarios, menúes, etc.
IDEAFIX	El Ambiente en General	Ubicar mapas, capacidades, etc.
LANGUAGE	El Ambiente en General	Ubicar archivos de mensajes.
printer	RM,RM	Definir salida de impresión
TERM	RM	Modelo de Terminal
dbase	DBM	Ubicación de los archivos de BD
SHELL	WM	Intérprete de comandos
empresa	menu	Nombre del licenciatario
NFILES	DBM	Max. número de archivos abiertos
MULTI	DBM	Define operación en red multiusuario
DBTAR	Utilitario tar	Se especifican las opciones relativas a los utilitarios exp ó imp, que vaya a utilizar el comando tar.
SCHEMA_SUFFIX	Open_Schema, Create_Schema, Find_Schema	Permite especificar el nombre del sufijo de un esquema.
HOME		Indica el directorio home del usuario.
DATADIR		Indica los directorios donde se almacenan las tablas según el idioma. Los directorios se pueden separar con ":".

2

Manual Básico

Capítulo 14

Visión General

Introducción

Básicamente, los elementos de IDEAFIX que soportan la tarea de desarrollo e implementación son:

- Manejo de Base de datos, a través del lenguaje SQL de IDEA-FIX.
- Diseño de reportes y formularios.
- Utilización de la interfaz de programación con Lenguaje "C" cuando sea necesario.

Estos elementos se apoyan sobre el Sistema de Ejecución de IDEAFIX, y fundamentalmente sobre el Window Manager, que se encarga de brindar al usuario la interfaz con las herramientas y programas de aplicación generados con IDEAFIX.

También se deben mencionar las Facilidades dadas por el entorno de desarrollo de software brindado por el sistema operativo. Unix se destaca, por ejemplo, por su amplia gama de herramientas para el desarrollo de software (Sistemas para control de versión -RCS-, mantenimiento de programas -Make-, editores -vi, ed-, etc.). IDEAFIX se integra perfectamente con ellas para aprovechar al máximo la combinación de:

Sistema Operativo + Ambiente de desarrollo de aplicaciones

Objetos que se Manejan en IDEAFIX

En la implementación con IDEAFIX se manejan distintos tipos de entidades: Bases de datos, Formularios, Reportes, etc. Estas entidades u "objetos" se definen mediante un lenguaje apropiado que permite especificar sus propiedades. IDEAFIX provee los siguientes lenguajes:

- SQL: Structured Query Language. Es el lenguaje que permite definir Bases de Datos con algunas extensiones provistas por I-DEAFIX.
- FDL: Form Definition Language. Es un lenguaje que permite especificar formularios; esto significa un acceso a tablas vía pantalla y teclado.
- RDL: Report Definition Language. Es un lenguaje para definir reportes impresos, incluyendo cálculos, control de página, etc.

A ellos se debe sumarle el Lenguaje de programación "C", que se utiliza en ciertos casos para programar operaciones especiales. Cuando se programa en "C" se utiliza la biblioteca de funciones de IDEAFIX y todo un ambiente que facilita el uso de este lenguaje. Este ambiente es denominado CFIX.

Cada "objeto" tiene su definición escrita en un archivo. La extensión del nombre del archivo indica qué tipo de objeto contiene. Los objetos que se pueden manejar son:

.sql	Consultas de Tablas en SQL
.sc	Definiciones SQL de un esquema
.fm	Definición de Form
.rp	Definición de Reporte
.mn	Definición de Menú
.c	Programa fuente C

Figura 1.1 - Extensiones de Archivos Fuente

A partir de ellos se generan -salvo los menúes y consultas SQL[1]- los objetos "compilados".

.sco	Definición compilada de esquema
.fmo	Definición compilada de formulario
.rpo	Definición compilada de reporte compilada
.o	Programa compilado C

[1] Los archivos de Query y menú no se compilan debido a que se ejecutan directamente por medio de un intérprete (por ejemplo, los utilitarios de menú o iql), de esta forma quedan siempre en "código fuente" durante la operación.

Figura 1.2 - Extensiones de Archivos Objeto

Además, pueden generarse archivos auxiliares que se utilizan solamente cuando estos objetos se manejan mediante la programación en CFIX:

.sch	Encabezado de definición de esquema
.fmh	Encabezado de definición de formulario
.rph	Encabezado de definición de reporte

Figura 1.3 - Extensiones de Archivos de Encabezado

Los Caminos de Desarrollo

El desarrollo de aplicaciones en IDEAFIX se realiza generalmente siguiendo un camino que parte de la definición de la Base de Datos. La Figura 1.4 muestra los distintos caminos a seguir para obtener funciones de ABM (programas para altas, bajas, modificaciones y consultas) y reportes o consultas impresos.

Sin embargo con IDEAFIX siempre existen alternativas para obtener una solución, por lo tanto, partir de la base de datos no es la única posibilidad. Es posible por ejemplo crear y depurar un prototipo de un sistema, como se verá en la siguiente sección. Dado que el caso más común es contar con la definición de la base de datos, la tomaremos como punto de partida.

IDEAFIX permite que el desarrollo de software se base en tratar naturalmente los casos generales, pero dejando abiertas las posibilidades para los casos particulares o muy complejos, que requieran tratamiento especial. En este sentido la interfaz de programación con CFIX, permite atacar cualquier problema por complejo que sea.

Los caminos indicados implican una serie de pasos que se cumplen mediante utilitarios de IDEAFIX, o bien mediante la intervención de un programador, lo cual se denota con el círculo marcado EDIT.

La línea de trazo punteado representa el camino automático, ya que no se requiere ninguna intervención para generar la función. Cuando por su complejidad, una función no se puede generar automáticamente, van apareciendo distintos grados de intervención, desde la simple modificación con el editor de un archivo generado automáticamente, hasta la programación en CFIX.

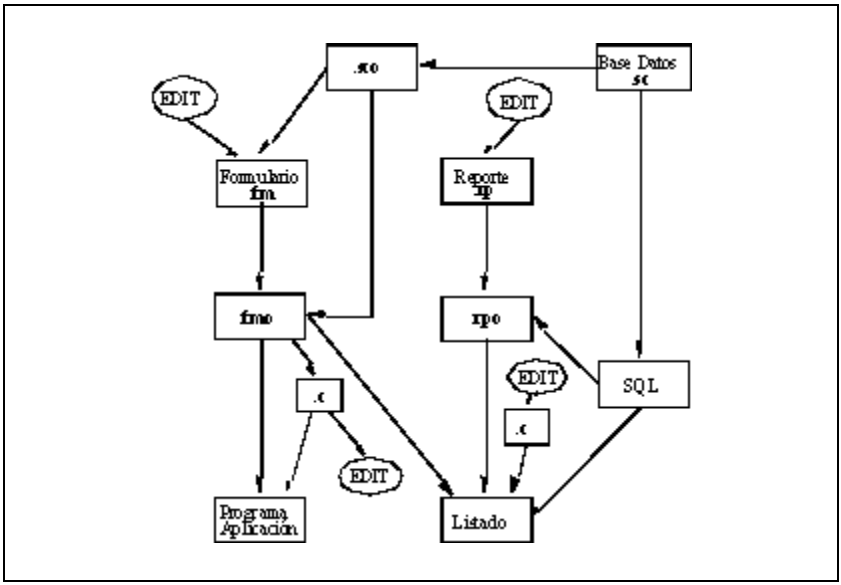


Figura 1.4 - Caminos del Desarrollo

Esta intervención se basa en programar por "excepción". Cuando algo no puede realizarse automáticamente, se recurre a la programación. Como es lógico, se programa **solamente** aquello que NO pudo hacerse en forma automática.

Analizemos por ejemplo la creación de una función de Altas, Bajas y Modificaciones (ABM) sobre datos contenidos en una tabla. En primer lugar, la base de datos ha sido procesada por el utilitario "*dgen*" -cuyo nombre proviene de Data base GENeration-, el cual lee un archivo con la definición de una base de datos y genera su representación en IDEAFIX.

A partir de esta base de datos, el utilitario "genfm" -GENerate ForM specification- permite generar una especificación de formulario. Sólo es necesario indicarle la tabla sobre la cual el formulario debe grabar los datos. Como resultado tendremos un archivo en FDL, que se procesa con "fgen" -Form GENeration-, que es el utilitario que a partir de una definición de formulario genera su representación intermedia. Con esto se completa el proceso, ya que para ejecutar el ABM sólo es necesario invocar el utilitario "doform", que presentará la pantalla permitiendo realizar las operaciones sobre la base de datos.

Si se desea modificar algunos detalles del formulario generado, es posible hacerlo editando el archivo ".fm" que se ha obtenido como resultado de "genfm". De allí la indicación EDIT en la Figura 1.4.

Diseño con Prototipos

Como se ha mencionado, el diseño partiendo de una Base de Datos no es la única alternativa posible, ya que es muy simple en IDEAFIX crear un prototipo de un sistema a partir de los formularios y menúes que presentará al usuario. En forma muy simplificada, el proceso de desarrollo se muestra en la siguiente figura:

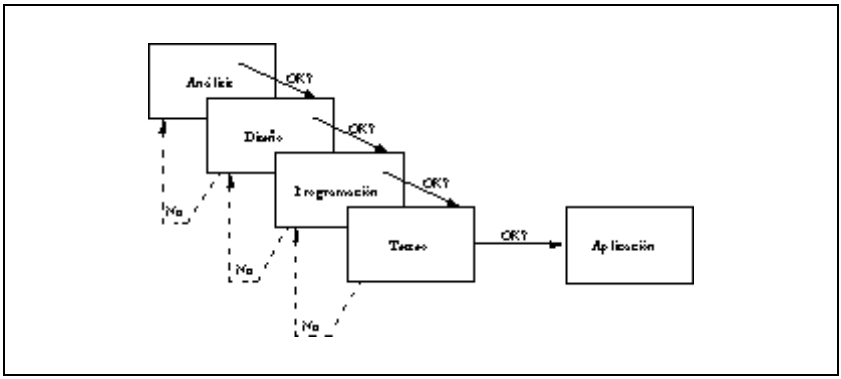


Figura 1.5 - Diseño con Prototipos

Las líneas punteadas que vinculan cada etapa de desarrollo con la anterior son consecuencia del resultado de cada una de las revisiones efectuadas para verificar la consistencia de lo diseñado, programado, etc., con los requerimientos del usuario.

Los menúes y formularios que formarán el prototipo pueden diseñarse rápidamente y simular la ejecución del sistema a través de los utilitarios *menu* y *testform*.

El *menu* es un utilitario que interpreta definiciones de menúes, las cuales están contenidas en archivos muy simples de escribir.

El *testform* permite simular el proceso de carga de datos con un formulario.

Con estos elementos se podrá presentar una primera versión del sistema que ejemplifique la interfaz del usuario. En esta etapa se trabaja sin datos, simplemente simulando la carga en las pantallas.

Si luego se diseña y se crea la base de datos, puede continuarse la prueba del prototipo, pero ahora pudiendo realizar cargas y consultas sobre datos reales. En lugar de usar *testform*, se utiliza *doform* que puede acceder a la base de datos. En este punto pueden crearse reportes mediante el SQL u otros utilitarios que permiten extraer información de la base.

El proceso de refinamiento del diseño puede continuar, y la base de datos seguir sufriendo modificaciones (sin perder la información), cuantas veces sea necesario hasta lograr el sistema

definitivo.

Herramientas

Encontramos cuatro áreas en que podemos dividir las herramientas de implementación:

- Manejador de Base de Datos
- Manejador de Formularios
- Manejador de Reportes
- Manejador de Lenguaje SQL

Es importante que el usuario conozca con el mayor grado de detalle las distintas posibilidades y opciones que tiene disponible para desarrollar sus aplicaciones. Para ello se dará un breve repaso a las facilidades que brinda IDEAFIX al respecto.

Base de Datos

Como hemos establecido, generalmente se diseña partiendo de la base de datos. IDEAFIX brinda una implementación del lenguaje SQL estándar, llamado IQL, para manejar todo lo relativo a la definición de estructura y manipulación de datos.

Sobre el estándar SQL, IDEAFIX ofrece algunas extensiones importantes que facilitan el diseño y la implementación. Estas son:

- Posibilidad de definir los criterios de consistencia de los datos directamente en la definición de la base de datos. Esto posibilita centralizar los criterios de validación, y evitar repetirlos en cada formulario o programa que actualice una tabla.
- Posibilidad de definir "joins implícitos" entre tablas. El término join proviene del inglés y significa juntar. Se entiende como "join entre tablas" que estas tengan columnas (campos) en común. IDEAFIX permite establecer este tipo de relaciones directamente al definir la Base de Datos.
- Posibilidad de trabajar con varios esquemas simultáneamente. Aunque el estándar no define nada al respecto, en la mayoría de las implementaciones actuales de bases de datos relacionales, un programa sólo puede operar con un esquema por vez. Esto complica o impide operaciones tales como joins entre tablas de distintos esquemas y movimientos de datos de una tabla a otra en distinto esquema. Lógicamente entonces, el diseñador opta por englobar todos sus datos en un solo esquema, lo cual no siempre es posible. Pero esto conspira contra la modularidad y la performance de los sistemas creados. En IDEAFIX, al poder trabajar con varios esquemas simultáneamente, permite realizar diseños modulares, sin penalidad de performance.
- Sentencias para permitir incorporar una descripción de cada elemento de datos (esquema, tabla o campo) que luego facilitará la generación de documentación.

Formularios

IDEAFIX ofrece un poderoso manejador de formularios. Su diseño se realiza, mediante Dalí (el editor de IDEAFIX) o cualquier otro editor, conforme al criterio conocido como WYSIWYG (*What You See Is What You Get*). Para describir los formularios se utiliza una herramienta denominada FDL (Form Definition Language, o Lenguaje de Definición de Formularios), que permite especificar sus características, desde la ventana sobre la cual se lo mostrará, hasta las validaciones que se realizarán sobre cada campo.

El FDL permite asociar campos de formularios con campos de tablas de bases de datos. Allí entonces, se aprovecha la centralización de validaciones mediante el mecanismo de "herencia": el campo del formulario hereda las validaciones y otros atributos del campo de la base de datos.

Con sólo diseñar un formulario, puede obtenerse una función de ABM sobre una tabla de base de datos. En realidad el formulario puede ser generado automáticamente mediante un utilitario.

El FDL ofrece además:

- Subformularios
- Campos múltiples
- Campos agrupados

Reportes

La filosofía de trabajo para los reportes es muy similar a la de los formularios. Se los diseña en forma WYSIWYG usando cualquier editor, mediante el lenguaje de definición de reportes denominado RDL (Report Definition Language). Permite especificar aspectos de los informes tales como:

- Cortes de control por página, por campo, y por inicio o fin del reporte.
- Asociación con campos de la base de datos.
- Funciones como sumatoria, promedio, máximos, mínimos y otras sobre los valores impresos en los campos del reporte.

El manejador de reportes de IDEAFIX puede usarse para cualquier tipo de aplicación, ya que se comporta como un verdadero "filtro" al estilo de UNIX.

También se integra naturalmente con el intérprete "IQL". Un reporte se obtiene fácilmente seleccionando los registros deseados mediante el SQL, y filtrando la salida mediante el manejador de reportes, que se ocupa de darle formato, imprimir número de página, realizar los cortes de control, etc.

IQL

IDEAFIX ofrece una implementación de lenguaje estándar SQL (Structured Query Language) denominada IQL, que facilita algunas extensiones provistas por IDEAFIX, para trabajar interactivamente realizando consultas y modificación de datos.

Los procedimientos mediante SQL permiten resolver en la mayoría de los casos tanto las consultas por pantalla como las salidas impresas de un sistema, según se mostró en la figura (Figura 4.1) que describe los caminos de desarrollo.

Integración de las Herramientas

Una conclusión importante que se puede destacar habiendo presentado los componentes que conforman el ambiente IDEAFIX, es el nivel de integración que ofrecen entre ellos.

A pesar de que cada uno se puede utilizar separadamente, al integrarlos y trabajar en conjunto se potencian todas las capacidades.

Por ejemplo, al estar definidas todas las validaciones y criterios de consistencia de datos, en la misma definición de la Base de Datos se logra que:

1. Los formularios y reportes a los cuales se les asocien campos con la base de datos, "hereden" de esta última las validaciones necesarias para el ingreso de datos (formularios), o aspectos sobre el despliegue del valor (reportes). Se evita tener que repetir las validaciones en cada formulario, ahorrando tiempo, y lo que quizás es aún más importante, se minimiza la cantidad de errores.

2. En la eventualidad de tener que introducir modificaciones a un desarrollo ya implementado, la modularidad e integridad de su concepción facilitará sobremanera dicha tarea.

Capítulo 15

Bases de Datos

Introducción

IDEAFIX permite manejar las bases de datos de acuerdo al modelo relacional. En este modelo, la estructura de datos fundamental es la tabla, las cuales pueden agruparse en "esquemas", como se verá en las siguientes secciones.

Existe una gran variedad de operaciones que es necesario realizar en relación con las bases de datos. Algunas de las más usuales son:

• Creación y modificación de la estructura de una tabla.

• Consulta o actualización de información existente en una o varias tablas.

Todas estas operaciones son realizadas por un componente de IDEAFIX que se denomina RDBMS (Relational Data Base Management System - Sistema de Manejo de Base de Datos Relacional). Este subsistema es el núcleo administrador de bases de datos, y como tal, el encargado de resolver las operaciones planteadas anteriormente.

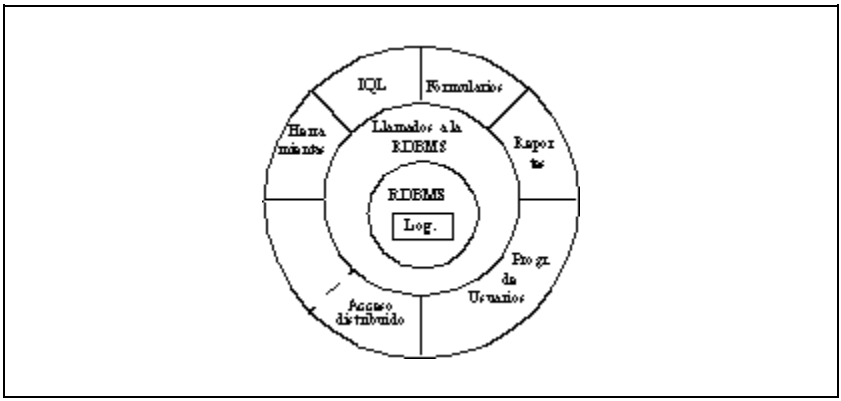


Figura 2.1 - Funciones del RDBMS

Los pedidos de operaciones al RDBMS se pueden realizar de diferentes maneras. Estas diversas formas ofrecen alternativas de acceso a la información, tal como mostraba la figura anterior.

El RDBMS está ubicado en el centro del gráfico, para destacar que es en definitiva el encargado de resolver los pedidos de operaciones sobre la base de datos de los utilitarios, los programas de usuario y cualquier otra aplicación o programa IDEAFIX.

Los pedidos de operaciones se realizan a través de la capa denominada "Llamados al RDBMS". Estos llamados se hacen en la forma de funciones en lenguaje "C", que se hallan disponibles en una biblioteca de programación.

Los compartimientos que rodean al núcleo son las distintas formas de acceder a la información que utiliza finalmente el usuario o el programador.

La forma más simple es a través de los "utilitarios". Estos pueden mostrar información, modificarla, etc.- Los utilitarios incluyen una variada gama de herramientas como se ve en el gráfico: El lenguaje SQL, la interfaz de formularios y reportes, y otros utilitarios auxiliares que se utilizan directamente sin necesidad de programar.

El compartimiento "programas de usuario" engloba aquellas aplicaciones que se escriben mediante el Lenguaje "C", y acceden a la base de datos con las funciones de la biblioteca de programación directamente haciendo llamados al RDBMS.

Definición de Bases de Datos

El diseño y la manipulación de las estructuras de base de datos se realiza a través de las sentencias del lenguaje estándar SQL, con algunas extensiones muy útiles, provistas por IDEAFIX.

El SQL provee un conjunto de sentencias que pueden dividirse en tres grupos según su función:

DDS: Data Definition Statements - Sentencias de definición de datos. Son las que permiten definir la estructura de una base de datos.

DMS: Data Manipulation Statements - Sentencias de Manipulación de Datos. Permiten operar con los datos existentes, por ejemplo realizar consultas, actualizar información, borrar datos existentes, etc.

CS: Control Statements - Sentencias de Control. Permiten realizar operaciones varias como activar un esquema, reconstruir índices, etc.

En este capítulo se describirán las sentencias DDS y CS, ya que son las que el programador o el analista utilizarán para diseñar la estructura de la base de datos.

Para realizar estas tareas existe el utilitario *dgen*, que interpreta estas sentencias (DDS + CS) y

se utiliza en las etapas de desarrollo.

El utilitario *iql* interpreta la totalidad de las sentencias SQL y puede ser utilizado para los mismos fines que dgen, pero además permite armar consultas (reportes) y otras funciones.

IDEAFIX Query Language (IQL) es la versión del lenguaje SQL que integra el sistema global, pero se contrata como un módulo separado.

Estructura de las Bases de Datos

Tablas

Las Tablas son la forma básica de almacenamiento. Están formadas por filas (también denominadas registros), y columnas, como se muestra en la siguiente figura. Las columnas son los campos que forman el registro.

campo 1	campo 2	campo 3	...	campo n

Figura 2.2 - Estructura de una Tabla

La figura muestra que cada registro está formado por un número fijo de campos. Todos los registros en una tabla tienen el mismo número de campos. En lo sucesivo, los términos "fila/registro" y "columna/campo" se utilizarán en forma intercambiable, ya que de hecho constituyen sinónimos a los fines prácticos.

IDEAFIX se suma al modelo relacional, donde existe sólo una estructura de datos: la tabla. De esta uniformidad nace un lenguaje de Base de Datos donde, con un comando, se puede recuperar un conjunto de registros de una o más tablas existentes, para listarlas, armar una nueva tabla, o reprocesar el conjunto de registros recuperados. Este lenguaje SQL es tan poderoso, que se necesita solamente especificar qué hacer, no cómo hacerlo.

En IDEAFIX, el intérprete de este lenguaje se denomina iql. Se incluye en el paquete iql, no formando parte del Sistema de Desarrollo.

El utilitario *dgen* se utiliza para la definición de las estructuras de datos, ya que interpreta un sub-conjunto de la totalidad del lenguaje SQL, compuesto por las sentencias para definición de datos (DDS) y otras auxiliares.

Esquemas

Un esquema es simplemente una colección de tablas, que normalmente guardan una relación lógica entre sí.

En IDEAFIX la división en esquemas no es una restricción, dado que el RDBMS permite trabajar con varios esquemas simultáneamente, pudiendo relacionar datos contenidos en

tablas de diferentes esquemas. Por lo tanto, para lograr un buen diseño es conveniente estudiar cuidadosamente cómo agrupar las distintas tablas de datos en los distintos esquemas. Las ventajas no son solamente de claridad en la definición lógica de los datos, sino también de orden práctico.

Bases de Datos

Para un usuario, la Base de Datos es toda la información a la que tiene acceso en un momento determinado. En IDEAFIX, una Base de Datos se define como el conjunto de todos los datos contenidos en todos los esquemas *activos*. Estos son aquellos que se han seleccionado con la operación apropiada, que puede asumir diferentes formas según el contexto en el cual se está trabajando, es decir, un formulario un reporte, la interfaz con lenguaje "C" u otro elemento.

Para que un esquema pueda ser activado, el usuario debe poseer los permisos apropiados para realizar tal operación. Aún contando con un esquema activo, un usuario tiene ciertas restricciones respecto de las operaciones que puede realizar con los datos contenidos en el esquema.

Esquema Corriente

Entre todos los esquemas *activos*, existe uno que se denomina *corriente*. Este esquema se distingue de los demás porque:

• Es el que se utiliza cuando no se indica explícitamente con cuál de los esquemas activos se quiere trabajar.

• Existen ciertos tipos de operaciones, por ejemplo, la de crear una nueva tabla, que sólo pueden hacerse sobre el esquema corriente.

Definición de una Base de Datos

Definición de Esquemas y Tablas

El lenguaje SQL provee diferentes tipos de sentencias como se explicó anteriormente en este capítulo. Las Sentencias de Definición de Datos (Data Definition Statements - DDS) se utilizan para definir tablas y esquemas. Estas permiten la creación, borrado y modificación de las mismas.

Se mostrarán ahora algunos ejemplos de las DDS y posteriormente se dará una referencia completa de los comandos correspondientes.

Como ejemplo de aplicación, se escribirán las sentencias necesarias para crear una base de datos que almacenará la información de una biblioteca simplificada. Para una mejor documentación de las estructuras creadas se recomienda editar un archivo que contenga todas las sentencias de definición de datos. Los nombres de archivos que reconoce el utilitario de IDEAFIX que interpreta las DDS (dgen), pueden tener las siguientes terminaciones:

.sc dgen lee definiciones de un archivo con esta terminación si se lo invoca explícitamente, o dando como argumento solamente la raíz del nombre, sin la terminación *.sc*. La desinencia deriva de schema (esquema).

.sql Un archivo con esta terminación se puede utilizar también para tener DDS's, pero es más general, ya que también se utiliza para las sentencias de manipulación de datos (Data Manipulation Statements - DMS). En este caso, con dgen se lo debe invocar explícitamente.

Si se utiliza el comando *iql* en forma interactiva se evita la necesidad de editar el archivo, ya que provee su propio editor (similar al *ie*), de lo contrario *iql* lee definiciones de un archivo con las características mencionadas en el párrafo anterior.

Creación de Esquemas y Tablas

Lo primero que se debe hacer es crear un esquema. Para ello se utiliza la sentencia CREATE SCHEMA:

```
CREATE SCHEMA biblio;
```

Se ha creado un esquema denominado *biblio*. No se hace distinción entre mayúsculas y minúsculas, por lo que se podría haber especificado:

```
create schema biblio;
```

Esta sentencia tiene como efecto colateral hacer *corriente* al esquema que se está creando. Una vez creado el esquema, vamos a necesitar crear tablas. Para ello contamos con la sentencia CREATE TABLE:

```
create table libros descr "Biblioteca" {
codigo num(4) descr "Código del libro"
not null,
titulo char(30) descr "Título del libro"
not null
mask "30>x",
autor num(4) descr "Código del autor"
not null
in autores:nombre,
edicion num(2) descr "Número de edición"
not null,
fecha date descr "Fecha de edición"
not null
<= today,
}
primary key (codigo),
index titulo (titulo not null);
```

Con CREATE TABLE se indica cómo se llama la tabla (*libros* en el ejemplo). Luego se especifican los nombres de las columnas (campos) de la tabla creada y se describen los tipos de datos y los atributos que cada columna contendrá.

Los Campos de la Tabla Ejemplo

Antes de explicar los campos contenidos en la tabla del ejemplo, es conveniente aclarar que NULL significa "sin valor", y no debe confundirse con el valor "0" (cero).

• *codigo* es el código del libro. El tipo de dato que contendrá el campo es numérico (NUM) de cuatro dígitos. (Este código podría ser el ISBN del libro, pero para ello deberíamos ampliar la cantidad de dígitos soportados por el campo). El atributo de esta columna especifica que no se aceptan valores nulos (not null), por lo tanto el dato es obligatorio.

• *titulo* es el nombre del libro. Esta columna contendrá cadenas de caracteres (char) de longitud menor o igual a 30. No acepta valores nulos (en este caso una cadena vacía) para un nombre. La máscara indica que se convierta a mayúsculas todas las entradas y que los valores aceptados sean alfanuméricos no obligatorios.

• *autor* es el código del autor del libro. El campo es numérico de cuatro posiciones y no acepta entradas nulas. El atributo *in* indica que el valor será validado, verificando que se encuentre en la tabla autores del mismo esquema.

• *edicion* indica cuál es el número de la edición del libro. El campo es numérico de dos dígitos y en este caso tampoco se aceptan entradas nulas.

• *fecha* es la de edición del libro (o pie de imprenta); luego el dato que contendrá es de tipo DATE (fecha) y se indica que el valor ingresado debe ser menor o igual a la fecha del día.

Aquellos campos en los que no se especifique el atributo *not null*, no requerirán obligatoriamente que se les defina un valor.

Especificación de claves

PRIMARY KEY indica que los campos entre paréntesis -si fueran varios la lista se separa por comas- contienen la información (clave primaria) para identificar una fila (registro) según el acceso más comúnmente utilizado. Esta clave debe ser *unívoca*, es decir, que no pueden existir dos registros con la misma clave.

Tal indicación creará un índice con estas características (es decir, sin duplicados). A dicho índice se le asignará el nombre del primer campo usado para formar la clave primaria.

Con la especificación INDEX *nombre*, se indica que se creará un índice cuya clave de acceso será el campo *nombre*. Estos índices se denominan *secundarios*, y se les puede dar un nombre distinto del que corresponde al primer campo de la clave. Así podría ser:

```
index obras (autor, titulo);
```

cuya significación esperamos resulte evidente.

No se estableció explícitamente a qué esquema debe pertenecer la tabla que se creó. Esto se debe a que por definición de las normas, al definir una tabla, ésta se agrega como parte del esquema corriente.

Tablas adicionales

Se supone que el esquema *biblio* contiene todas las tablas relacionadas con los datos de la biblioteca. Se agregará ahora una tabla con los códigos de los autores de los libros, junto con algunos datos adicionales. Para esto se necesita otra sentencia CREATE TABLE:

```
create table autores descr "Autores de libros" {
codigo num(4) descr "Código del autor"
not null
primary key,
nombre char(30) descr "Nombre del autor"
not null
mask "30>x",
nacion num(2) descr "Nacionalidad del autor"
not null
in paises:descrip,
};
```

Nótese que en este caso se omitió la cláusula PRIMARY KEY, ya que se la especificó en cambio como un atributo de la columna *código*. La explicación de la cláusula *in paises* se verá seguidamente. Otra tabla que se crea en este esquema es la tabla paises, que en principio contiene las nacionalidades de los autores:

```
create table paises descr "Paises de origen" {
codigo num(2) descr "C"digo del país"
not null
primary key,
descrip char(15) descr "Descripción del país"
not null
mask "15>x",
};
```

La verificación "in paises" se refiere por defecto a la clave primaria de la tabla, es decir al *código de país*, y la especificación `:' seguida de un campo de la tabla de verificación -*descrip* en este caso- indica que se debe desplegar el contenido de dicho campo cuando se requiera ayuda para llenar el contenido del campo original *nacion* en la tabla autores.

Las sentencias que se acaban de presentar, se escribieron sobre un archivo llamado *biblio.sc* con la ayuda del editor *Dalí* de IDEAFIX. La sintaxis completa de estas sentencias se detallan más adelante.

La definición de tablas y esquemas se realiza por medio del utilitario *dgen*, antes mencionado. Dicho utilitario trabaja de la siguiente forma:

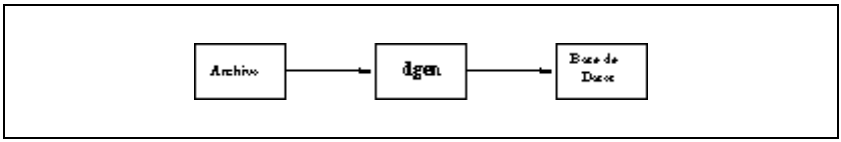


Figura 2.3 - Generación de un esquema

Generación del Esquema

Para crear el esquema y sus tablas asociadas puede ejecutarse:

```
$ dgen biblio
```

o bien

```
$ dgen biblio.sc
```

El utilitario *dgen*, en este capítulo, está presentado en su forma más simple. Para un mayor detalle debe consultar el capítulo sexto de este manual (Utilitarios).

Como se ve, por defecto el sistema asume la extensión *.sc*. En cambio, si se hubieran escrito las definiciones en un archivo con terminación .sql, se debería ejecutar:

```
$ dgen biblio.sql
```

Tener la definición de los esquemas en archivos es útil para la documentación, y aconsejable si se crean posteriormente programas en "C" para manejar las tablas. Si no se dispusiera de dicha definición, mediante el uso de la sentencia STORE SCHEMA, se puede obtener un archivo con sentencias de definición de datos (DDS) a partir de un esquema existente, el que tendrá terminación *.sc*. Por ejemplo:

```
$ iql -c "use biblio; store schema;"
```

generará un archivo llamado *biblio.sc* que contendrá las siguientes sentencias:

```
/* biblio 12/07/90 15:00 */
schema biblio;
table libros descr "Biblioteca"
{
codigo num(4) descr "Código del libro",
titulo char(30) descr "Título del libro"
mask "30>x",
autor num(4) descr "Código del autor"
in autores:(nombre),
edicion num(2) descr "Número de edición",
fecha date descr "Fecha de edición"
<= TODAY,
}
primary key (codigo),
index titulo (titulo not null);
table autores descr "Autores de libros"
{
codigo num(4) descr "Código del autor",
nombre char(30) descr "Nombre del autor"
mask "30>x",
nacion num(2) descr "Nacionalidad del autor"
in paises:(descrip),
}
primary key (codigo);
```

```
table paises descr "Paises"
{
codigo num(2) descr "Código del país",
descrip char(15) descr "Descripción del país"
mask "15>x",
}
primary key (codigo);
```

Por más detalle sobre el utilitario iql consultar el *Manual del Usuario, Parte II.*

Indices

Cuando se utiliza un Sistema Administrador de Base de Datos relacional (Data Base Manager System - DBMS), se opera siempre sobre datos que parecen estar almacenados en tablas bidimensionales simples; pero ello se debe a que el DBMS separa *lo que se ve* de la manera en que los datos están registrados en realidad en el dispositivo de almacenamiento de la computadora. Según como esté almacenada físicamente la información y las características de arquitectura de la base de datos, se tiene un mayor o menor impacto en la performance, conforme a las consideraciones siguientes:

• Cuando se necesita consultar una tabla con varias filas, y el registro que se quiere direccionar está lejos del comienzo del archivo, se debe buscar en la tabla completa, operación que consume tiempo.

• El punto anterior se extiende también a los programas de aplicación. En éstos, el programador necesita algún tipo de clave para direccionar un registro dado en una tabla y operar con él. En IDEA-FIX, una clave puede ser cualquier combinación de columnas, inclusive partes de ellas.

• La creación de índices ayuda al RDBMS de IDEAFIX a direccionar registros específicos en las tablas. Un índice es un archivo auxiliar relacionado con una tabla, que contiene claves ordenadas que *apuntan* (esto es, *direccionan físicamente* dentro de un disco) a los registros que efectivamente contienen la información.

• Los índices se crean con la Sentencia de Definición de Datos CREATE INDEX, y pueden generarse y borrarse en forma dinámica. Hay dos clases de índices:

UNIQUE No se permiten claves duplicadas.

NOT UNIQUE Este tipo de índice permite claves duplicadas, y es el utilizado en caso de omitirse la especificación. Esto es lo que se llama óvalor por defectoó (default value), concepto que se aplica a toda clase de atributos o características de los diversos elementos o conceptos empleados en computación. En este caso particular, el tipo de índice definido es *not unique* cuando no se lo aclara en forma explícita.

Cuando una clave es *not unique*, pueden existir distintas filas para el mismo valor de clave, y el programa de procesamiento deberá verificar de algún modo que se seleccione el registro correcto. Una tabla puede tener varios índices, y el RDBMS actualiza todos los índices asociados con una tabla siempre que ésta tenga un cambio.

Descripción de Sentencias de Control

use

Esta sentencia permite activar esquemas. Se da la lista de esquemas que se quiere activar:

```
use esquema1 [, esquema2, ...];
```

Después de la palabra clave USE se puede ingresar uno o varios nombres, separados por comas, quedando como esquema corriente el primero que se incluyó en la lista. Así por ejemplo:

```
USE biblio;
```

Descripción de Sentencias DDS

Esta sección ofrece una referencia completa de la sintaxis de las sentencias de definición de datos. El SQL de IDEAFIX no hace distinción entre las palabras claves en mayúsculas o minúsculas.

create schema

```
[CREATE] SCHEMA <nombre_esquema>
[DESCR[IPTION] <cadena>]
[LANGUAGE <cadena>]
```

Crea un esquema llamado *nombre_esquema*. Este esquema se agrega a la nómina de esquemas activos, quedando como corriente. Esto equivale a una sentencia USE implícita contenida en el CREATE. La descripción se utiliza con fines de documentación. La opción LANGUAGE, indica al utilitario *dgen* que genere el archivo de encabezado del esquema, archivo con el mismo nombre que el esquema y extensión .sch. Este archivo será incluido en los programas fuente de CFIX que utilicen el esquema.

create table

```
[CREATE] TABLE <nombre_tabla> [[dimensión]]
[DESCR[IPTION]] <cadena>] {
campo [ [dimensión] ] <tipo> [ lista_atributo ],
campo ...
} [clave ...];
```

La sentencia crea la tabla *nombre_tabla*. Si se indicó el número de ófilasó para la tabla ([dimensión]), ésta admitirá como máximo la cantidad de registros indicados. En caso contrario la tabla es dinámica y admite tantos registros como se deseen grabar. En muchos casos la cantidad de registros está acotada por el tamaño de los campos de la clave. Es decir que si la clave primaria está formada por un campo, y ese campo es numérico de dos

posiciones, la cantidad máxima de registros que podrá contener la tabla es de 200 (Números positivos, negativos y NULL). En este caso, por una cuestión de buena organización del disco, sería conveniente especificar explícitamente la cantidad máxima de registros que puede contener la tabla.

La cláusula DESCRIPTION da una descripción de los propósitos de la tabla y se utiliza cuando el usuario requiere información acerca de la tabla con el comando SHOW.

La estructura de la tabla se define especificando el nombre de cada columna (*campo*), y el tipo de dato (*tipo*) que cada columna contendrá. Los tipos de datos que pueden contener los campos son los siguientes:

NUM Se especifica como: num [(digitos [, dec])]. La columna puede contener un número de dígitos digitos y un número de decimales especificado en dec. Si no se completa el número de dígitos decimales se asumirá que es cero. El número máximo de dígitos es 28.

CHAR Se especifica como: char [(n)].Para cadenas de caracteres de longitud *n*. Si *n* no se especifica se asumirá 1.

DATE La columna contendrá fechas.

TIME La columna contendrá horas.

FLOAT Para números representados en notación científica.

BOOL La columna puede contener el valor: verdadero (Si, Yes, etc) o falso (No).

Los Atributos de Campo

Los atributos son ciertas propiedades que se pueden dar a las columnas, y pueden ser:

```
not null
```

La columna requiere el ingreso obligatorio de datos.

```
description <cadena>
```

Permite asignar una descripción literal del propósito del campo. Su función es documental.

```
default <valor>
```

El valor por omisión de la columna será *valor*.

```
primary key
```

Para esta columna se creará un índice sin duplicados, que será la clave primaria de la tabla. Este atributo puede aparecer en una columna solamente; si no aparece en ninguna, se puede usar la cláusula primary key en la parte clave

```
check digit
```

Indica que se va a usar un dígito verificador en el campo, separado por el carácter indicado.

```
mask <cadena>
```

Con esta cláusula se especificará una máscara que contendrá los caracteres válidos que puede tener un campo. La especificación es para campos de tipo char, y la máscara es una cadena de caracteres, pudiendo estar definida de dos formas:

- Explícitamente.

- Mediante una variable de ambiente. La misma se referenciará con el signo $ seguido por el nombre de la variable.

Los caracteres que se utilizan para las máscaras y que tienen significado especial pueden ser caracteres de conversión, de ocultamiento, de especificación de tipo o factores de repetición. Los caracteres de conversión son los siguientes:

> Convertir a mayúsculas.

< Convertir a minúsculas.

El carácter de ocultamiento es:

Oculta el carácter.

Los caracteres de especificación de tipo son:

h Numérico Hexadecimal.

H Numérico Hexadecimal obligatorio.

a Alfabético.

A Alfabético obligatorio.

x Alfanumérico.

X Alfanumérico obligatorio

n Numérico.

N Numérico obligatorio.

Un factor de repetición se define antecediendo un número a los otros caracteres especiales. Cualquier otro carácter en la máscara se imprimirá en la salida tal como fue definido. Si se desea imprimir uno de los caracteres especiales, se lo debe preceder por una barra invertida ("\e") para que pierda su significado especial.

A continuación se presentan algunos ejemplos de uso de máscaras:

```
mask "AANNXXHH"
```

En este caso la entrada exige que los dos primeros caracteres sean alfabéticos, que los dos siguientes sean numéricos, luego se deberán ingresar dos caracteres alfanuméricos y por

último dos hexadecimales. En todos los casos los datos son obligatorios.

```
mask "30>x"
```

En este ejemplo se deben ingresar treinta o menos caracteres alfanuméricos que serán convertidos a mayúsculas.

```
mask "15#h"
```

En este ejemplo se deben ingresar quince o menos caracteres numéricos hexadecimales que serán ocultados cuando se realice el display. Cada carácter ingresado será reemplazado por "#", ocultando de esta forma los carácteres.

```
mask "nn.nn.NNN"
```

Esta máscara requiere siete caracteres numéricos de los cuales los primeros cuatro no son obligatorios y los últimos tres sí. Nótese también que los caracteres están separados, en dos oportunidades por un punto.

```
mask "3>#A.4n.hhh"
```

Esta máscara requiere tres caracteres que sean alfabéticos obligatorios, que serán ocultados y convertidos a mayúscula, luego cuatro caracteres numéricos no obligatorios, y por último tres caracteres hexadecimales no obligatorios.

```
<oprel> <valor>:
```

Los valores que no verifiquen la condición no serán aceptados, y se tratarán como errores. Los operadores relacionales son:

Operador Relacional	Descripción
>	Mayor
<	Menor
>=	Mayor o Igual
<=	Menor o Igual
!= <>	Diferente
==	Igual

Figura 2.4 - Operadores Relacionales.

```
[not] between <valor> and <valor>:
```

Es una forma abreviada de colocar operadores relacionales de forma tal que establezcan un rango. Se validan las entradas dentro de ese rango de valores que incluye los extremos.

```
[not] in (<valor>[:<cadena>] [,...]):
```

Permite especificar un conjunto finito de valores que debe asumir la columna para que el registro sea admitido. Si se indica "not" la validación es a la inversa. Cada valor del conjunto puede tener asociada una cadena de caracteres que sirve como documentación del mismo y para ser desplegado en las ventanas de ayuda, asociadas a este campo, en los formularios.

```
[not] in <table> ...:
```

El valor de la columna debe existir en la tabla. Usando "not" la validación es a la inversa. Esta validación se explica en detalle más adelante en esta sección debido a que será necesario introducir algunos conceptos previos para comprender íntegramente su uso.

```
check (expresión)
```

Permite indicar una condición que deben cumplir los valores para ser aceptados. Esta condición puede involucrar otros campos, operadores relacionales, valores, etc.

Superposición de condiciones

Las validaciones que implican un chequeo sobre el valor del campo se las conoce como "atributo de chequeo" y son las siguientes:

Operadores relacionales (>,<,>=,<=,==,!=).

```
[not] between.
[not] in (valores).
[not] in tabla.
check (expresion)
```

El atributo *check* puede tener una expresión que incluya las cláusulas *between* e in (valores). Estas cláusulas se mantienen en su forma aislada por compatibilidad con las versiones anteriores de IDEAFIX.

Especificaciones de Indices

Si se especifica la cláusula PRIMARY KEY en la parte *clave* indicada en la sintaxis general de la sentencia, el efecto es el mismo que para el atributo de campo *primary key* (clave primaria).

Nótese que esta cláusula permite una clave primaria formada con una combinación de columnas, mientras que el atributo de campo *primary key* puede ser especificado sólo para una columna.

Las especificaciones de índice permiten agregar índices alternativos a una tabla:

```
clave: primary key espec_indice
[, [unique] index nombre espec_indice
[,...]] ;
```

donde espec_indice puede estar dado por las siguientes alternativas:

espec_indice: `(nom_campo [rango] [opciones] [,...]) [[separ]]`

rango: `(desde [,longitud])`

opciones: `not null`

`asc[ending]`

`desc[ending]`

Seguidamente se explican los distintos elementos que intervienen en la especificación de un índice:

• nom_campo es el nombre de una columna.

• El *rango* se puede indicar para campos alfanuméricos y permiten tomar una parte de la columna indicada por el "desde" (posición inicial a partir de 0) y una *longitud*. Por ejemplo si un campo es alfanumérico de treinta posiciones (char(30)) y se lo quiere indexar por las primeras cinco, el rango se indicara "(0, 5)".

• Las opciones para cada componente de la clave son:

`not null`: No se indexan las entradas con valor nulo.

`ascending`: Ordenar en forma ascendente (usado por defecto).

`descending`: Ordenar en forma descendente.

• El valorsepar se utiliza como "separación" para el método de acceso (Ver Parámetros de Configuración de los Indices, en este capítulo).

Un Ejemplo de Clave Primaria Compuesta

Continuando con el ejemplo presentado en este capítulo (Biblioteca), vamos a colocarle un campo más a la tabla de paises de forma tal que pueda contener idiomas. Entonces le agregamos el campo tipo de la siguiente forma:

```
tipo num(1) descr "Tipo de Registro"
not null
in (1:"Paises",2:"Idiomas"),
```

Con esta modificación puede deducirse fácilmente que la clave primaria no puede contener un solo campo, por lo tanto debe especificarse al final de la tabla, como se indica a continuación:

```
primary key (tipo, codigo);
```

La validación *in table*

En IDEAFIX se conoce esta validación como "join implícito", porque permite definir una relación entre columnas de distintas tablas directamente en la definición de una de ellas. Consiste en verificar que el valor que se le da a un campo en una fila de una tabla, exista en

una columna dada de otra tabla. La verificación se hace realizando una búsqueda por clave sobre la tabla donde debe existir el valor. En el caso más simple la lectura se hace por la clave primaria. El valor que se le da a la clave a buscar es el que tiene el campo que se está validando. Supondremos en primera instancia que el valor del campo se puede usar directamente para realizar una búsqueda.

El resultado de la validación es entonces el resultado de la búsqueda, es decir, si la búsqueda es exitosa la validación se verifica.

La sintaxis completa para expresar esta validación es:

```
[not] in <tabla> [by <indice>][(<campo_clave>
[,...])][:(<nom_campo>[,...])]
```

• *tabla* es la tabla donde se hace la búsqueda.

• *índice* es el índice por el cual se accede a la tabla. Esto permite que la búsqueda pueda hacerse por un índice alternativo, no siendo necesario expresarlo en el caso de la clave primaria.

• *campo_clave* son los valores necesarios para formar la clave, salvo el último que será el que se quiere validar. Estos valores pueden ser los que contengan otros campos, variables de ambiente o bien constantes. En este punto es necesario destacar que sólo se podrán hacer validaciones sobre campos que se ubiquen en último término en la especificación del índice <indice> de la tabla <tabla>.

• *nombre_campo* son campos de la tabla donde se está realizando la búsqueda. Los valores de estos campos son los que se despliegan en las ventanas de ayuda de los formularios, cuando se relacionen campos de éstos con campos de la base de datos. En el capítulo referente a formularios se explica detalladamente este aspecto ya que no tiene ninguna influencia en la validación en sí.

Un ejemplo simple de su aplicación está dado en la tabla *libros* cuando se le asigna este atributo al campo *autor*.

```
autor num(4) ... in autores:nombre,
```

En este caso cualquier ingreso que se intente hacer sobre este campo, será validado contra el campo *codigo* de la tabla *autores* ya que éste es el único que forma parte de la clave primaria.

Para poder ejemplificar adecuadamente las otras alternativas posibles de este atributo, sería conveniente introducir algunas modificaciones en el esquema de biblioteca que antes creamos.

La primer modificación la vamos a efectuar sobre la tabla `libros` en el campo `titulo`, y le vamos a agregar el atributo:

```
not in libros by titulo
```

Nótese que se tuvo que especificar el índice por el cual se debe acceder a la tabla, ya que no

es la clave primaria. Lo que va a validar esto, es que las entradas de títulos no sean repetidas. Obviamente esto se podría haber evitado colocándole la especificación de UNIQUE al índice *titulo*.

Una validación *in table* usando un índice compuesto debe especificar de qué modo completar la clave. Siempre se usará como último componente de la misma el campo a validar. Para los valores anteriores se puede especificar:

• Un valor constante.

• Un valor de otro campo de la tabla

• Una variable de ambiente.

Entonces en el campo *nacion* de la tabla *autores* el atributo *in table* debe conformarse como sigue:

```
in paises(1):descrip
```

Recordar que se le agregó un campo a la tabla *paises* y que la clave primaria quedó formada por tipo y codigo, por lo tanto el número 1 sirve para completar el primer campo de la clave primaria indicando que se trata de un país y no de un idioma. En este caso se utilizó un valor constante. Si se hubiera tenido que usar una variable de ambiente, simplemente al nombre de la variable hay que anteponerle un signo "$":

```
in paises($TIPO_PAIS):descrip
```

En el caso de completar la clave con el valor de otro campo, se coloca el nombre del mismo. Por ejemplo en la tabla *grales*, al campo *codigo* podríamos agregarle el atributo :

```
not in grales(tipo)
```

Arreglos

Es probable que en una tabla se tengan que definir dos o más campos con las mismas características y que contendrán el mismo tipo de datos. Por ejemplo, en la definición de una tabla de clientes donde se necesite tener más de un teléfono. En este caso se puede optar entre dos alternativas: crear tantos campos como teléfonos se quieran tener, o bien, crear un campo vectorizado.

La forma de crear este tipo de campos es colocándole al final del nombre del mismo la dimensión encerrada entre corchetes ([ ]). De este modo, una solución práctica para nuestro ejemplo sería:

```
telef[3] char(10) mask "(3n)nnN-4N",
```

donde se admiten tres teléfonos por cada registro de cliente.

Relación con los Formularios

Como se verá al explicar el generador de formularios, se pueden asociar campos de una tabla con una pantalla. Se produce entonces lo que en IDEAFIX se denomina "herencia" y es el proceso por el cual un campo de pantalla hereda atributos del campo de la Base de Datos. Los atributos sujetos a este proceso son:

• not null

• default

• mask

• operadores relacionales (<, >, <=, >=, !=, ==)

• between

• [not] in

• [not] in table

• check (expresión)

drop table

```
DROP TABLE <nombre_tabla>
```

Borra la tabla *nombre_tabla*, la que debe pertenecer a uno de los esquemas activos. Todos sus índices asociados también se borran.

store schema

```
STORE SCHEMA [nombre_esquema] [HEADER]
```

Almacena la definición del esquema nombre_esquema en un archivo denominado *nombre_esquema.sc*. Si no se especifica nombre_esquema, se utiliza el último esquema activado.

Si se especifica la cláusula HEADER, se genera un archivo llamado *nombre_esquema.sch*, el que contiene la definición de las constantes de "C" que necesitan los programas de aplicación.

drop schema

```
DROP SCHEMA nombre_esquema
```

Borra el esquema *nombre_esquema*. Solamente tiene permiso para esta operación el dueño del esquema. No puede borrarse un esquema activo.

create index

```
[CREATE] [UNIQUE] INDEX nombre_indice
[ON <nombre_tabla>] espec_indice
```

Crea un índice para *nombre_tabla* para las columnas indicadas en *spec_indice*. Si no se especifica *nombre_tabla*, se utiliza la última tabla creada. Si se indica UNIQUE, el índice no permitirá duplicados.

La especificación del índice es idéntica a la mencionada en la creación de tablas y presenta las siguientes alternativas:

espec_indice: `(nom_campo [rango] [opciones] [,...]) [[separ]]`

rango: `(desde [,longitud])`

opciones: `not null`

`asc[ending]`

`desc[ending]`

Por ejemplo si se quisiera crear un índice para la tabla *libros* el que nos sirva para ubicar libros por autor, se ejecutaría:

```
create unique index autores
```

En este caso se crea un índice unívoco cuya clave de acceso es código de autor y el código del libro, siendo el nombre del índice *autores*.

drop index

```
DROP INDEX nombre_indice ON nombre_tabla
```

Borra el índice *nombre_indice* de la tabla *nombre_tabla* del esquema corriente. Si se quisiera borrar un índice que no fuera del esquema corriente, simplemente hay que especificarlo. Por ejemplo:

```
DROP INDEX .... ON esquema.tabla.
```

La clave primaria de una tabla no puede borrarse.

recover

```
RECOVER {nombre_indice | * } ON tabla
```

Recupera un índice o todos (en caso de indicar *) de la tabla *tabla*. El o los índices se reconstruyen a partir de los datos existentes.

Esta operación puede realizarse también con el utilitario *frecover*.

rename table

```
RENAME TABLE [esquema.]tabla AS nombre
```

Renombra la tabla *tabla* con el nombre *nombre*. Si no se especifica el esquema se asume el corriente. La modificación del archivo que contiene las sentencias de definición (en nuestro ejemplo, biblio.sc), queda como tarea del programador. Esta modificación puede hacerse tanto editando el archivo, como también ejecutando la sentencia STORE SCHEMA. Para una mejor comprensión, veamos un ejemplo:

```
rename table paises as grales
```

Otra forma de especificar esto podría ser:

```
rename table biblio.paises as grales
```

rename field

```
RENAME FIELD [esquema.]tabla.campo AS nombre
```

Renombra el campo *campo* con el nombre *nombre*. En el caso de no especificarse esquema, se asume el corriente. Por ejemplo:

```
rename field libros.codigo as codlib
```

o bien,

```
rename field biblio.autores.codigo as codlib
```

Para mantener el archivo de sentencias de definición del esquema actualizado, podemos editar el archivo y modificarlo o bien ejecutar la sentencia STORE SCHEMA.

Usando Bases de Datos

En IDEAFIX existen varias maneras de manejar una Base de Datos:

• A través del utilitario *iql*, con las sentencias de manipulación del *iql*.

• Mediante los utilitarios que manejan formularios, con los que se puede cargar, modificar, consultar y eliminar datos de una o varias tablas de cualquier base de datos. Estos utilitarios serán presentados en el siguiente capítulo (Formularios).

• Por medio de los utilitarios *exp* o *imp*, que permiten convertir los datos que se encuentran almacenados en formato interno de IDEAFIX a un formato ASCII, transportable o viceversa.

• Escribiendo programas en "C", utilizando las funciones provistas por la biblioteca de interfaz en "C". El sistema de desarrollo IDEAFIX provee la Biblioteca de Funciones "C" para crear programas de aplicación.

Utilitario IQL

Este utilitario interpreta las sentencias del lenguaje SQL estándar en su totalidad, más algunas extensiones muy útiles provistas por IDEAFIX.

El *iql* puede ser invocado de dos formas distintas:

• Interactivamente, donde por medio de un editor similar al *Dalí* de IDEAFIX el usuario introduce los comandos, o

• ejecutando el utilitario con un archivo conteniendo sentencias SQL.

El subconjunto de sentencias utilizado para el manejo de datos, se denomina DMS - Data Manipulation Statements, (Sentencias de Manipulación de Datos) y abarca todas aquellas sentencias necesarias para:

• Consultar los datos contenidos en una o más tablas de uno o más esquemas. Esta consulta nos permite colocar restricciones tanto sobre las filas como sobre las columnas, agrupar filas, aplicar funciones y/u operaciones aritméticas sobre los campos seleccionados, especificar criterios de orden, redireccionar la salida de resultados (la salida estándar es por pantalla), etc.. La sentencia que maneja estas consultas se denomina **Error! Bookmark not defined.**SELECT y un ejemplo de su aplicación podría ser el siguiente:

```
select codigo, titulo, autores.nombre
from libros, autores
where autor = autores.codigo and
codigo between 1 and 100
output to terminal;
```

Aquí se están seleccionando las columnas de *codigo* y titulo de la tabla *libros* y la columna *nombre* de la tabla *autores*.

Nótese que para la primera tabla no fue necesario indicar explícitamente el nombre de la misma. Esto se debe a que en caso de omisión se entiende que es la primer tabla indicada en la cláusula **Error! Bookmark not defined.** *from* de la sentencia.

Con la cláusula **Error! Bookmark not defined.** *where* se colocan restricciones sobre las filas a seleccionar. En nuestro ejemplo se indica que el código de autor de la tabla *libros* debe coincidir con el código de autor de la tabla *autores*. En SQL esta condición se conoce como *condición de unión* ("join condition") y es aquella mediante la cual se establece la relación de selección entre las tablas involucradas. Siempre van a existir condiciones de *join* cuando se seleccionan datos de más de una tabla; en caso contrario se obtendrá el producto cartesiano de los registros de las tablas. También es válido colocar restricciones independientes sobre las tablas como en el ejemplo. Mediante el conector lógico *and* se pretende además que el código de libro de la tabla *libros* pertenezca al rango de valores especificado. Por último, la salida se envía a pantalla.

• Eliminar registros (filas) de una o más tablas de uno o más esquemas. Esta sentencia también posee la cláusula *where* con la que se puede restringir la eliminación de registros. Un

ejemplo de su utilización podría ser el siguiente:

```
delete from libros where codigo >= 10;
```

donde se eliminan los registros cuyos códigos de libros sean mayores o iguales a diez.

• Agregar nuevos registros a una tabla. Mediante la sentencia INSERT, podemos insertar nuevas filas en una tabla especificando explícitamente los valores de las columnas, o bien, indicando que el resultado de una sentencia *select* proporcionará los valores que serán agregados. Un ejemplo de esta sentencia es el siguiente:

```
insert into paises values (1,1,"ARGENTINA");
```

En este caso se agrega a la tabla *paises* un registro con los valores de campo especificados entre paréntesis.

Modificar valores de campos de registros existentes. En muchos casos es necesario modificar el contenido de las tablas de la Base de Datos. Mediante la sentencia UPDATE se pueden realizar estas operaciones, ejemplificadas a continuación:

```
update autores set nombre = "BORGES J.L."
where codigo = 86;
```

Se modifica el valor del campo *nombre* de la tabla *autores* indicando que se efectúe en el registro con código de autor igual a 86.

Existen otras sentencias interpretadas por el utilitario *iql*, pero que escapan al objetivo de este capítulo. Para una mayor referencia y profundización de las sentencias aquí presentadas consultar el *Manual del Usuario, Parte II*.

Utilitario EXPORT/IMPORT

Este comando permite migrar datos que se encuentran en formato interno de IDEAFIX a un formato ASCII, y viceversa. Si se invoca como *exp* imprime sobre la salida estándar datos del esquema y la tabla que se indiquen, en un formato de caracteres alfanuméricos. Ejecutándolo como *imp* toma de la entrada estándar caracteres alfanuméricos y los convierte al formato con que IDEAFIX maneja los datos. Por supuesto, se pueden redireccionar a través de los mecanismos del shell tanto la entrada como la salida estándar para leer o grabar sobre un archivo cualquiera.

A continuación se presentan algunos ejemplos de su aplicación. La sintaxis completa de este utilitario se puede hallar en el *Manual del Usuario* en el Capítulo 8.

```
$ exp biblio.autores
```

Al ejecutar esta sentencia desde la línea de comandos se obtendrá:

```
1->BORGES, JORGE LUIS->
2->SHAKESPEARE, WILLIAM->
```

```
4->VERNE, JULIO->
5->SHAW, BERNARD->
```

donde el símbolo "->" indica que el separador de campos utilizado en este caso es el tabulador. El utilitario cuenta con una serie de opciones que permiten modificar los separadores de campo y de registro; la clave por la cual se accede a los datos; el rango de valores para una determinada clave que se quiere exportar o importar; la memoria cache utilizada para los datos y/o índices, y otras características que hacen al proceso de migración de datos.

Interfaz con el Lenguaje C

Los programas en lenguaje "C" necesitan información acerca de las tablas que manejarán, para pasársela a las funciones de la biblioteca. Esta información se toma del archivo cabecera que debe incluirse en el programa y que se genera junto con el esquema, de la siguiente forma:

```
CREATE SCHEMA biblio LANGUAGE "C";
```

Si el esquema se genera correctamente, entonces también se generará un archivo de encabezado llamado *biblio.sch*, conteniendo las constantes apropiadas de "C" que se requieren en los programas de aplicación que usen el esquema *biblio*. Otra forma de obtener este archivo de encabezado es mediante el utilitario *dgen*, colocándole la opción **-h**, o mediante el utilitario *iql* utilizando la sentencia STORE SCHEMA esquema HEADER. El siguiente gráfico muestra el proceso:

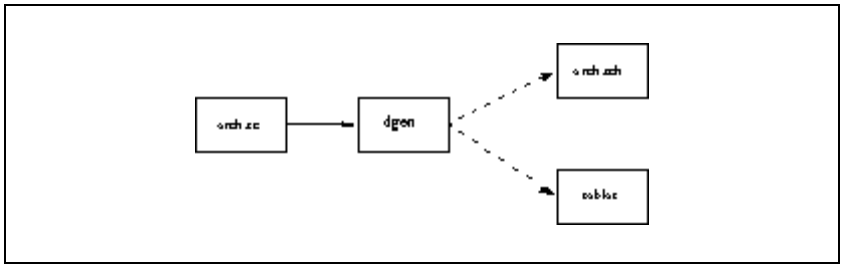


Figura 2.5 - Generación de Esquemas y Tablas

Las constantes son necesarias en el momento de la compilación de los programas de aplicación. Desde un programa "C", se abren los esquemas mediante la función de la biblioteca de interface de IDEAFIX OpenSchema, todos los esquemas que se abren desde un programa con esta función son esquemas activos, quedando como corriente aquel que se haya abierto último.

Mantenimiento de Bases de Datos

Modificación de una Base de Datos Existente

Por varias razones, a veces se hace necesario **Error! Bookmark not defined.**modificar un esquema de una base de datos. Ello puede pasar, por ejemplo, cuando se elimina o se crea una nueva tabla, o se modifican campos de una tabla en un esquema determinado. El utilitario **Error! Bookmark not defined.** *dgen* puede utilizarse para tal finalidad.

Por ejemplo, si se modificó una tabla del esquema *biblio*, editando el archivo *biblio.sc*, previamente creado, y se ejecuta:

```
$ dgen biblio
```

dgen nos informará que hay un error indicandonos que el esquema ya existe. Por esta razón este utilitario tiene la opción **-m**, debiendose ejecutar el comando:

```
$ dgen -m biblio
```

para que se pueda modificar un esquema ya creado. En este caso, *dgen* sólo realiza las operaciones necesarias para convertir el esquema a su nueva definición intentando conservar los datos existentes.

La búsqueda y correspondencia de los campos y tablas es por "nombre". Esto significa que si se cambia el orden de definición de los campos en alguna tabla, *dgen* automáticamente cambiará el orden en la base de datos, manteniendo la consistencia de los mismos en dicha tabla. Si se está usando la interfaz con el lenguaje "C" y se alteró el orden de los campos de alguna tabla, se deben *recompilar* todos los programas que referencien a la tabla que fue modificada, debiendo haber generado previamente el archivo de encabezado del esquema en el que se encuentre la tabla. Es conveniente aclarar que *sólo en este caso* es necesaria la recompilación de los programas.

En caso de que se agreguen campos, o se amplíe la precisión de los existentes, *dgen* convierte los datos. Si se borra la definición de un campo de una tabla determinada, y se ejecuta `dgen -m`, se obtendrá un mensaje de error y el programa abortará, dado que si efectuara la operación se perderían los datos asociados al campo eliminado. Lo mismo sucede en otros casos en que se puede perder información, por ejemplo cuando se disminuye la longitud de un campo, lo que provocaría el truncamiento de algunos datos. La opción **-f** permite realizar esta tarea, aunque por las características de la operación se debe estar muy seguro de lo que se va a realizar, ya que puede ocasionar pérdida de información.

La opción **-v** de dgen va mostrando los resultados de las operaciones a medida que las va realizando. Por ejemplo:

```
$ dgen -v biblio
Ejecutando File: biblio.sc.
Creando Esquema 'biblio'.
Creando Tabla 'libros'.
Creando Tabla 'autores'.
```

```
Creando Tabla 'paises'.
Hecho.
Generando archivo de encabezamiento para esquema 'biblio'.
Hecho.
$... (promp esperando para el próximo comando)
```

Acceso Concurrente

Cada vez que se abre un esquema para utilizarlo, se incrementa un contador en el mismo. De esta manera se tiene control de cuántos procesos están utilizando un esquema. Para poder modificar un esquema existente este contador debe estar en 0 (cero), para estar seguro de que ningún proceso esté utilizando sus datos.

Si esto no fuera así, el proceso de modificación informará al usuario que no puede bloquearse el esquema para uso exclusivo.

En caso de poder hacerlo, el esquema permanecerá bloqueado durante el proceso de modificación. Cualquier proceso que intente abrir el esquema mientras se esté ejecutando la sentencia, dará un error.

Esta característica implica que si un programa concluye anormalmente sin cerrar el esquema, el contador no quedará nunca en 0 (cero) y por lo tanto no permitirá bloquear el esquema para su modificación. Para remediar esta situación se debe utilizar el utilitario *frecover*, que se explica a continuación.

Recuperación de Archivos

Si un proceso termina anormalmente, cabe la posibilidad de que los índices de las tablas que estuviera modificando queden desactualizados respecto de los datos contenidos en dichas tablas. En este caso se dice que el índice está *corrupto*. Si un esquema tiene índices corruptos la operación de apertura del mismo retornará error. Esta situación se resuelve aplicando el utilitario *frecover* sobre el índice corrupto.

Utilitario FRECOVER

Este comando se utiliza para **Error! Bookmark not defined.**reconstruir las tablas y sus índices, de un esquema de base de datos, frente a situaciones tales como la pérdida de índices.

Hay distintas maneras de invocar a *frecover*:

```
$ frecover esquema
```

reconstruye todos los índices de todas las tablas del esquema indicado, que estén corruptos, y

```
$ frecover esquema.tabla
```

reconstruye todos los índices corruptos de la tabla *tabla*.

En los casos anteriores se recuperan todos los índices. Si se desea recuperar algún índice en

particular, se lo debe indicar como:

```
$ frecover esquema.tabla [indice]
```

La sintaxis completa del utilitario se encuentra en el *Manual del Usuario*, Capítulo 8.

Chequeo de Consistencia

Utilitario DBCHECK

Este comando verifica la **Error! Bookmark not defined.**consistencia de los datos cargados en distintas **Error! Bookmark not defined.**tablas de la base de datos, contra lo especificado en la definición de las mismas. Recorre cada uno de los registros de las tablas que se indican, controlando que se verifiquen los operadores:

• not null

• [not] in table

• [not] in

• [not] between

• Operadores Relacionales (=, !=, <, >, <= y >=) presentes en la base de datos.

Un ejemplo de su aplicación podría ser el siguiente:

```
$ dbcheck biblio
```

daría como resultado por ejemplo:

```
Schema biblio
Verificando tabla libros...
En clave: 1
Campo: autor, Valor: 0 no verifica condicion IN TABLE
En clave: 2
Campo: autor, Valor: 0 no verifica condicion IN TABLE
Hecho.
Verificando tabla autores...
Hecho.
Verificando tabla paises...
Hecho.
Errores detectados: 2.
```

El R.D.B.M.S.

Ubicación y Distribución Física de los Archivos

Existe una variable de ambiente denominada *dbase* que contiene los directorios bajo los

cuales se encuentran los archivos que forman las Bases de Datos. Cada esquema que se cree, tendrá un único directorio asociado, que será un subdirectorio de alguno de los directorios especificados en la variable de ambiente base.

La forma de dar el valor a la variable *dbase*, si se está trabajando en Bourne Shell (sh), es por ejemplo:

```
$ dbase=/usr/datos:/usr/sistemas/datos
$ export dbase
```

Si se trabaja con "C-shell" (csh) se debe ejecutar:

```
$ setenv dbase /usr/datos:/usr2/sistemas;
```

Los usuarios que usen "Korn Shell" (ksh) pueden hacer lo mismoque lo indicado para el Bourne, o bien

```
$ export dbase=/usr/datos:/usr/sistemas/datos;
```

Notar que los dos puntos (:) separan los distintos directorios, de la misma forma en que se indican los directorios en la variable de ambiente PATH.

En MS-DOS se define de la siguiente forma:

```
C>SET DBASE=A:\eDATOS;C:\eUSR\eIDEAFIX\eDATOS
```

En este caso los directorios se separan con punto y coma (;) de la misma forma en que se definen en la variable de ambiente PATH.

Es común que estas definiciones se especifiquen en los archivos que se leen al inicio de sesión `/etc/profile` y `.profile` si se trabaja con *ksh* o *sh*, y *.login* si el shell de trabajo es *csh*). En el caso de MS-DOS se definirá en el archivo AUTOEXEC.BAT o en el archivo ENV.

Si se trata de ejecutar *dgen*, sin especificar *dbase*, dará como mensaje de error:

```
Variable de ambiente dbase inexistente.
```

En la siguiente figura se muestra la distribución de los archivos en disco, para un directorio de base de datos llamado /usr/datos.

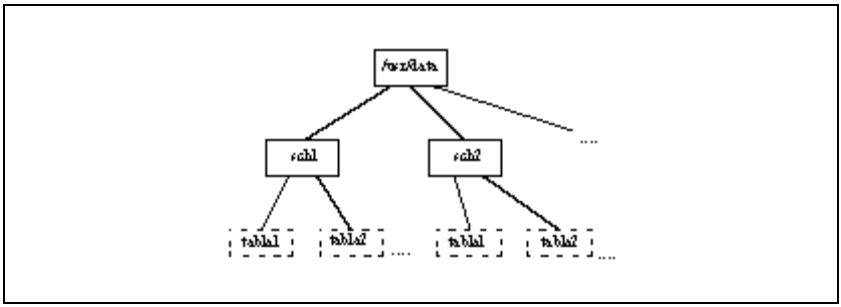


Figura 2.6 - Distribución de la Base de Datos

El directorio /usr/datos/esq1 es el directorio que contiene toda la información del esquema esq1. En él existen archivos para las tablas, índices y el diccionario de datos. En el ejemplo de la biblioteca este directorio se llama *biblio* y contiene los siguientes archivos:

Archivo	Contenido
biblio.sco	Definición del esquema compilada
autores.tab	Datos de la tabla "autores"
autores.i00	Clave primaria de la tabla "autores"
libros.tab	Datos de la tabla "libros"
libros.i00	Clave primaria de la tabla "libros"
libros.i01	Indice de acceso alternativo a la tabla "libros"
paises.tab	Datos de la tabla "libros"
paises.i00	Clave primaria de la tabla "paises"

Una *tabla* se compone de un archivo físico con los datos propiamente dichos (archivo con extensión ".tab"), y uno o más archivos físicos con las claves para acceder a ellos (índices). Cada tabla tiene por lo menos un índice, llamado índice primario, que se especifica mediante la cláusula *primary key* (archivo con extensión ".i00" el resto de los índices se irán numerando ".i01", "i02" y así sucesivamente).

Estructura Interna de los Archivos

Los registros almacenados en el archivo que forman una tabla poseen estructuras determinadas por el formato de cada uno de los campos que lo componen.

Existe una estructura externa, que es la establecida por la definición de tablas en un archivo con extensión *.sc*, y una estructura interna, que se corresponde con la definición de tipos de datos en formato binario, necesaria para almacenar físicamente la información. La estructura

interna tiene los mismos campos que la externa, más un campo agregado al comienzo llamado "pool", que se utiliza para individualizar los registros libres.

En realidad la designación "pool", que en este caso puede traducirse como área común, corresponde en rigor a la lista de registros vacantes, pero se usa también para el campo que permite direccionarla. La estructura interna responde entonces al siguiente diagrama:

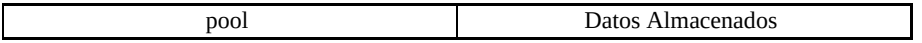


Figura 2.7 - Almacenamiento de datos

El campo *pool* contiene:

• La longitud del registro, si está ocupado (valor positivo).

• Si está libre, el número del siguiente registro libre cambiado de signo.

De esta manera se forma la llamada "lista de libres", donde se encadenan todos los registros que no están usados. Cuando se borra un registro, se lo agrega a la cabeza de la lista de libres para que su lugar sea utilizado cuando se incorpore un nuevo registro.

Cuando se agrega un nuevo registro, primero se explora la lista de libres. Si ésta no está vacía se toma el registro que está a la cabeza, y se lo marca como usado. Si la lista está vacía, se agrega un nuevo registro al final del archivo.

La correspondencia entre los tipos de datos internos y externos es la siguiente:

Externo	Interno
num(1) to num(2)	1 byte
num(3) to num(4)	2 bytes
num(5) to num(9)	4 bytes
num(10) to num(15)	8 bytes con punto flotante
num(16) to num(28)	16 bytes
float	8 bytes con punto flotante
char(n)	n+1 bytes
date	2 bytes
time	2 bytes
bool	1 byte

Figura 2.8 - Cantidad de Bytes para cada tipo de dato

Mecanismo de Locking

Para la implementación en sistemas multiusuarios se usa el mecanismo estándar de UNIX a través de las funciones de Biblioteca lockf(3) o bien de la llamada al sistema operativo

(System Call) locking(2).

En principio, se utiliza como norma general el *advisory locking*, y en aquellos casos que fuera necesario, IDEAFIX maneja internamente el *mandatory locking*. Es de destacar que en el nuevo motor de base de datos desarrollado por **InterSoft**, *Essentia*, se brindan soluciones aún más amplias al problema del bloqueo de registros.

Parámetros de Configuración de los Indices

La estructura de los archivos de índices es la siguiente:

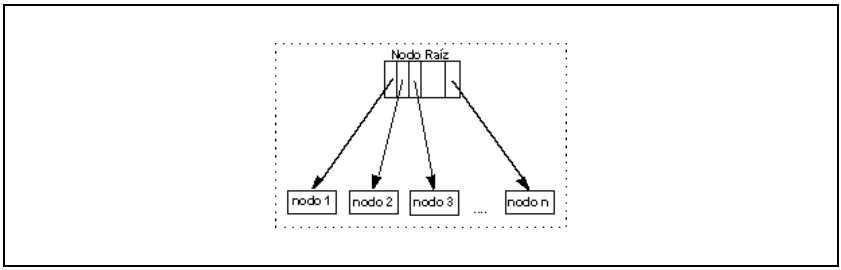


Figura 2.9 - Estructura de los archivos Indice

El nodo raíz contiene claves ordenadas, y asociado a cada clave un puntero al nodo terminal que contiene las claves mayores que ella pero menores que la siguiente. Los nodos terminales son los marcados en el dibujo como **nodo 1 ,nodo 2, ..., nodo n**.

En los nodos terminales las claves tienen asociado un número de tipo *long* (4 bytes) con el puntero al registro de datos. En los nodos terminales siempre existen por lo menos la mitad de las entradas ocupadas con claves, para mantener una estructura balanceada. El único caso en que un nodo terminal puede tener menos del 50% ocupado es cuando es el único nodo existente, y no hay todavía suficientes claves para crear un nuevo nodo terminal.

Los tamaños en bytes de las claves, los nodos terminales, y el nodo raíz están dados por las relaciones que se mostrarán, donde se usa la siguiente nomenclatura:

• *size* Maximo posible número de entradas (claves) .

• *separ* Número de bloques de 1Kb ocupado por un nodo terminal.

El parámetro *size* se define cuando se quiere imponer un máximo a la cantidad de registros de una tabla. Si se deja indefinido la tabla no tendrá límite en cuanto a la cantidad de registros.

Las magnitudes *separ* y *size* se pueden indicar en el archivo ".sc". La manera de hacerlo es a través de la definición de tablas e índices, encerrando los valores entre corchetes. Si se quisiera definir a *size* como 1000 y a *separ* como 2, para especificar el índice *titulo*, podría procederse como en el siguiente ejemplo:

```
create table libros [1000]
(
codigo num(4) not null,
titulo char(30) not null,
autor num(2) not null,
edicion num(2),
fecha date,
)
primary key (codigo),
index titulo (titulo not null) [2];
```

Si no se indican estos valores, asumirá 1 para *separ*, y el parámetro *size* queda indefinido, es decir que la tabla podrá crecer en tamaño indefinidamente (obviamente limitada por el espacio físico en disco).

Las distintas relaciones entre todas las magnitudes se expresan mediante las fórmulas que se brindan seguidamente:

• Longitud de clave:

Kl = å size of(*componentes de clave*)

• Longitud de una entrada en el archivo índice:

Il = *Kl* + size of (*long*)

• Número de entradas en un nodo terminal:

• Número de entradas en un nodo raíz:

• Tamaño del root en bytes:

Rs = *Nr* x *Il*

• Tamaño del nodo terminal en bytes:

Ts = *Ntmax*x *Il*

En memoria se mantiene permanentemente una copia del nodo raíz, y del nodo terminal que contiene las claves sobre las que se está trabajando. Por lo tanto, el espacio ocupado en memoria para una tabla es la suma de los espacios de memoria requeridos por cada uno de sus archivos índices. Para un índice puede calcularse de la siguiente forma:

Memoria Total = *Rs* + *Ts*

El parámetro *separ* permite dimensionar el tamaño de memoria ocupada, es decir, variando su valor cambia el espacio requerido por el uso del índice. Una fórmula aproximada, pero simple, para calcular rápidamente un valor óptimo para la separación de los índices, es la siguiente :

Capacidades Máximas

Esquemas por base de datos	Unlimited
Esquemas activos	16
Tablas por esquema	256
Tablas activas por esquema	256
Registros por tabla	2147483648
Campos por tabla	255
Tamaño máximo de un registro	65535
Caracteres en un campo alfanumérico	65535
Dígitos en un campo numérico	28
Dígitos significativos en un campo numérico	28
Rango de valores en un campo fecha	16/04/1894 to 16/09/2073
Rango de valores en un campo hora	00:00:00 to 23:59:58[a]
Máxima dimensión de un vector	65535
Valores en un campo in	65535
Indices por tabla	256
Campos por índice	256
Longitud de un campo alfanumérico en índice	256
Máximo nombre de Esquema	8
Máximo nombre de Tabla	8
Máximo nombre de Campo	16

a El último horario para un día es 23:59:58, esto se debe a que IDEAFIX tiene una precisión de 2 segundos para los datos del tipo Time.

Capítulo 16

Formularios

Introducción

Uno de los aspectos más importantes en un programa de aplicación es la interfaz con el usuario. IDEAFIX provee una forma para diseñar fácilmente las pantallas de formularios de manera amigable para el operador, proporcionando:

• Definición de características de los campos;

• Criterios de consistencia para aplicar a los datos cargados;

• Mensajes de error y ayuda, y

• Completo control sobre la imagen de la pantalla, incluyendo atributos y ventanas.

Los programas de aplicación interactivos son formularios electrónicos desplegados en pantalla. Un formulario es la imagen de un documento, sobre la cual pueden realizarse las mismas acciones que las que se realizarían sobre un trozo de papel, como ser:

1. Archivar datos en un archivo.

2. Leer de un archivo, y luego modificarlo (o actualizarlo).

3. Remover los datos de un archivo.

Para archivar un formulario, como en 1), existen dos operaciones asociadas: AGREGAR (cuando el formulario es nuevo) o ACTUALIZAR (cuando se ha cambiado un formulario existente previamente, como en 2).

Para desechar un formulario (caso 3.), se realiza una operación llamada BORRAR.

Para ignorar lo que se ha escrito en un formulario y dejarlo en el estado anterior a la modificación, se realiza una operación denominada IGNORAR.

Todas las operaciones definidas anteriormente se ejecutan mediante una Tecla de Función específica, como ser <PROCESAR>, o <REMOVER>.

La siguiente tabla describe la correspondencia entre las operaciones y las Teclas de Función.

Operación	Tecla de Función
AGREGAR	<PROCESAR>
ACTUALIZAR	<PROCESAR>
REMOVER	<REMOVER>
IGNORAR	<IGNORAR>
FIN	<FIN>

Figura 3.1 - Operaciones de Formularios

Luego se verá, como un formulario puede tener restringidas ciertas operaciones, o bien pedir una confirmación antes de ejecutarla.

La interfaz de cada programa con el usuario se establece a través de formularios. Se considera cada formulario como una ventana en una pantalla física.

Los formularios pueden diseñarse con cualquier editor, aunque *Dalí* de IDEAFIX posee ciertas características que facilitan mucho esta tarea, alguna de ellas imprescindible, como por ejemplo la posibilidad de representar signos propios del español (vocales acentuadas, eñes, etc.) y caracteres gráficos.

Para definir un formulario se utiliza el FDL, Forms Definition Language (Lenguaje de Definición de Formularios). El FDL permite dibujar la imagen de la pantalla tal como se la desea, en un modo denominado WYSIWYG (What You See Is What You Get, es decir, lo que ves es lo que obtienes); o sea que al ejecutar el programa, la pantalla aparecerá tal como se la ha dibujado.

Finalizado el dibujo, se definen los nombres de campos y sus atributos. Cuando el archivo de especificación del formulario se ha completado, se lo compila con *fgen*, el utilitario de IDEAFIX para generación de formularios.

Generando el Formulario

Como se mencionó anteriormente, cada formulario está contenido en un archivo de definición con la extensión ".fm", conocido como archivo fuente FDL. Este archivo será procesado con *fgen*, para generar un archivo con la extensión ".fmo", y opcionalmente un *archivo de cabecera* con la extensión ".fmh", como se muestra en el siguiente gráfico:

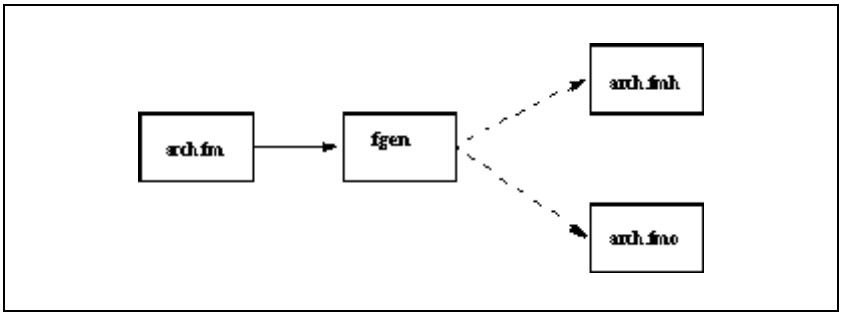


Figura 3.2 - Generación de un Formulario

El archivo ".fmo" contiene el *formulario compilado*, el cual es usado por los utilitarios que permiten emplearlo para distintos fines (*testform*, *execform*, *doform*, etc) o por un programa de aplicación escrito en "C", en el momento de su ejecución.

El archivo de cabecera debe incluirse en los programas de aplicación en "C" que utilicen el formulario. Contiene valores de constantes simbólicas que representan los distintos campos, y otras constantes auxiliares necesarias para la compilación.

El Lenguaje FDL

Un archivo de formulario escrito en FDL se divide en tres secciones:

• La primera parte contiene la *imagen de la pantalla*.

• La segunda parte (iniciada por la palabra clave `%form`), contiene información general sobre el formulario. Esta sección es opcional.

• La tercera parte (que se inicia con la palabra clave `%fields`), tiene como fin definir los nombres de los campos y sus atributos (validaciones, ayudas, etc.).

Se explicará seguidamente el formato de cada sección.

Imagen de la Pantalla

Comienza al principio del archivo, y muestra cómo se verá la pantalla cuando se utilice el formulario. Hay cuatro clases de datos en la imagen:

• Texto común. Se despliega directamente.

• Indicadores de Atributos de Pantalla. Consisten en una secuencia "Esc [n m", donde "Esc [" simboliza <HOME>, y el valor de n corresponde al atributo:

Atributo	Número (n)
NORMAL	0
NEGRITA	1
STANDOUT	2
SUBRAYADO	4
BLINK	5
REVERSO	7

Figura 3.3 - Atributos de Pantalla

El atributo se activa desde el momento en que se inserta en la imagen hasta que otra indicación modifique su vigencia. El editor ie permite manejar estos atributos de manera muy cómoda, digitando ~n a, donde n es el dígito indicado en la segunda columna. Se recomienda no usar el número 6, ya que provoca una alteración en el juego de caracteres.

• Valores de variables de ambiente. El valor de una variable de ambiente puede obtenerse con la indicación `\$name`, donde name es el nombre de la variable. Por ejemplo: la indicación `\$usrname` desplegará el contenido de la variable de ambiente usrname al momento de desplegar el formulario, en la posición de pantalla donde ha sido escrito.

• Campos de Pantalla. Son los campos de datos donde el usuario ingresará y/o visualizará información. Pueden ser de tipo fecha (DATE), hora (TIME), numérico (NUM), punto flotante (FLOAT), alfanumérico (CHAR) y booleano (BOOL). Los campos también se conocen con el nombre steps (pasos) porque cuando se utiliza el formulario, el proceso de ingreso de datos recorre cada campo de la pantalla "paso por paso". Cada campo dibujado en la imagen debe ser bautizado con un nombre en la sección `%fields`. Los campos se enumeran de izquierda a derecha, y de arriba hacia abajo según aparecen en la pantalla y deben nombrarse en ese mismo orden en la sección `%fields`.

Los campos que el usuario debe completar con datos, están marcados en la pantalla con el carácter de subrayado (_). Se dibujan de la siguiente manera, de acuerdo al tipo de dato que ellos aceptan y/o muestran:

1. ALFANUMERICO. Está formado por una secuencia de uno o más caracteres de subrayado (_):

_ _ _ _ _ _ _ _ _ _ Alfanumérico de 10 caracteres.

En este campo, el usuario puede ingresar cualquier carácter imprimible. Es posible dar validaciones adicionales sobre el tipo de caracteres que se admiten mediante las especificaciones de máscara que se describirán más adelante. Debe tenerse en cuenta que conforme a las normas del trazado WYSIWYG los "underscores" aparecerán uno a continuación del otro formando un trazo continuo, y no separados como aquí se ilustran.

2. NUMERICO. Es también una secuencia de subrayados, pero termina con un punto decimal. Opcionalmente puede especificarse la coma (",") como separador de unidades de

mil. Luego del punto pueden indicarse posiciones para decimales. Se usa el punto decimal y la coma para los millares, a fin de uniformizar la especificación de pantalla; sin embargo, en el momento de usar el formulario, los números se mostrarán según el país. En el caso del español se usará la coma como separador decimal y el punto para indicar los millares.

_ _ _. Entero de 3 dígitos.

_ _ _ _ _._ 5 dígitos enteros, 1 decimal.

_ _ _,_ _ _._ _ 6 dígitos enteros con separador de miles, 2 decimales.

En caso de que el campo tenga posibilidad de ser negativo, se indicará anteponiendo el signo correspondiente, de la siguiente manera:

-_ _ _,_ _ _._ _ 6 dígitos enteros, 2 decimales, separador de miles y signo.

Si se quiere usar dígito verificador en el campo, se agrega un signo numeral después del punto (el campo debe ser entero):

_ _ _ _.#

Al ingresar los datos, sólo se permiten dígitos y el signo menos, si ha sido indicado.

3. DATE. En este caso las barras definen el formato utilizado para la fecha, que puede ser alguno de los siguientes:

_ _/_ _/_ _ (dd/mm/aa)

_ _/_ _/_ _ _ _ (dd/mm/aaaa)

La cantidad de posiciones después de la segunda barra determina si se especifica el año completo, o solamente los dos últimos dígitos. Las fechas se validan en el momento en el que el usuario las digita, rechazando las que sean erróneas.

4. TIME. El carácter dos puntos ":" es el que marca este tipo de campo. La hora puede especificarse en dos formatos:

_ _:_ _ (HH:MM)

_ _:_ _:_ _ (HH:MM:SS)

La hora se valida en el momento en que se ingresa, rechazando los valores sin sentido.

5. FLOAT. Este tipo de campo se identifica mediante el carácter "e" al final de una secuencia de subrayados. Al igual que para los campos numéricos, si se antepone el signo menos, se permitirá el ingreso del mismo durante la carga. Veamos ahora su sintaxis:

_ _ _ _ _ _ e 7 dígitos en punto flotante sin signo.

-_ _ _ _ _ _ _ _ e 9 dígitos en punto flotante con signo.

En el ingreso de este tipo de datos, sólo se permitirán:

• dígitos numéricos;

• el signo menos si estubiera especificado, o

• el carácter e para el ingreso en formato exponencial.

Es importante notar que debido a este último formato la longitud mínima de este tipo de campo es siete.

6. BOOL. Este tipo de dato está formado por cero o mas caracteres subrayados seguidos por el carácter "?". El formato es el siguiente:

_ _ ? Booleano de 3 caracteres.

Los valores lícitos para este tipo de dato son: verdadero (Si, Yes, etc) o falso (No, etc). Solo hará falta ingresar el primer carácter, por ejemplo "Y" o "S" para el valor verdadero, desplegándose automáticamente el resto del campo, "YES" en el primer caso y "SI" en el segundo. La configuración de los valores dependerá de la variable de ambiente LANGUAGE.

Las características y validaciones aplicadas en cuanto al tipo de dato aceptado fueron indicadas al dibujar cada campo. Pueden presentarse casos más complejos donde sean necesarias otras restricciones a los caracteres aceptados. IDEAFIX provee las llamadas "máscaras" que permiten especificar validaciones sobre un campo alfanumérico, restringiendo el tipo de carácter que puede ingresarse en cada posición. Estas posibilidades se obtienen mediante el atributo mask. Para más información se puede consultar la página de la función mask(ST) en la *Referencia de Funciones de* IDEAFIX.

Continuando con el ejemplo introducido en el capítulo anterior (esquema de biblioteca), podríamos definir la imagen del formulario de manejo de datos de la tabla libros, de la siguiente forma:

```
LIBROS
Código del Libro: __________.
Título de la Obra: ______________________________
Código del Autor: __________.
Edición: _____
Fecha: _____/_____/_____
```

Existen también posibilidades más avanzadas:

• **Campos Múltiples**: Permiten formar una matriz con una serie de campos dibujados en una fila.

• **Campos Agrupados**: Conjunto de campos que se relacionan para efectuar validaciones cruzadas entre sus valores.

• **Subformularios**: Capacidad de presentar un formulario en forma dinámica. Cuando el usuario completa un campo, se despliega el subformulario, permitiendo cargar datos en él. Cuando se completa a su vez el subformulario, éste desaparece de la pantalla, continuándose en el formulario original.

Separador de Campos

Es posible usar el carácter NULL (ej., '\0') como separador de Campo, el cual hará que dos campos separados por él en una imagen aparezcan juntos en la pantalla. Esto puede ser útil cuando se quiera desplegar varios campos en pantalla, porque usando el separador NULL se ahorra el espacio entre los campos numéricos, los cuales ya tienen un separador (ej., el punto del final).

La Sección %form

Esta sección es opcional, y especifica ciertas características generales de un formulario. Está formada por una serie de sentencias finalizadas por punto y coma (";"). Pueden insertarse comentarios de la siguiente forma:

• Precediéndolos por dos barras ("/ /"). El comentario se extiende desde allí hasta el fin de línea.

• Encerrándolos entre los pares de caracteres barra-asterisco ("/*") y asterisco-barra ("*/"). Esta notación es análoga a la empleada en lenguajes de programación como "C" y PL/I:

```
/* Esto es un comentario */
```

En esta sección se admiten las siguientes sentencias:

`use`

Especifica los esquemas de la base de datos que se usarán cuando se relacionen campos de tablas de la Base de Datos con los de pantalla (luego se verá más sobre este tema). Los esquemas se especifican por una lista de nombres separados por comas:

```
// Esquemas a utilizar:
use biblio,alpha, sueldos;
```

`language 'c'`

Esta sentencia indica al compilador (fgen) que se genere el archivo de encabezado (el que lleva extensión `.fmh`). Este archivo es necesario en el momento de la compilación de los programas CFIX que invoquen a este formulario.

`ignore`

Las operaciones a ser ignoradas van en una lista, separadas por comas. Pueden ser: *add*, *update*, *ignore*, *delete*, *next* , *prev* y *end*. Por ejemplo:

```
/* Ignorar operaciones de agregado y borrado */
ignore delete, add;
```

`confirm`

Las operaciones de formulario que deben confirmarse antes de su ejecución pueden ser: *add*,

update, ignore, delete y *end*. Su sintaxis es la misma que para el `ignore`.

```
// Confirmar actualización y fin
confirm update, end;
```

`window`

Esta sentencia controla los parámetros de la ventana que contendrá al formulario.

`messages`

La próxima subsección explica las cláusulas admitidas. En esta opción se especifican los mensajes de error o ayuda que serán asociados a los campos de la pantalla, y desplegados en el momento en que se digite la tecla de función <AYUDA>, o se produzca un error en tales campos. La lista de mensajes está separada por comas (",") y finaliza con punto y coma (";").

```
messages
HELP1: "Código numérico del Libro",
ERROR: "El título ya existe";
```

Al desplegar un mensaje desde un programa en CFIX (Lenguaje "C" + Biblioteca de funciones IDEAFIX) pueden pasarse como argumentos del mismo valores obtenidos durante la ejecución del programa. De esta forma se logrará que los mensajes varíen según las circunstancias. La forma de especificar la posición donde se colocarán los valores pasados al mensaje, es similar a la del printf del "C", por lo tanto las siguientes alternativas son válidas:

```
%d : short (d = Decimal corto)
%ld : long (ld = Long Decimal)
%f : double (f = Full capacity = Double length)
%g : float ("f" ya se usó, por eso emplea "g")
%s : char (s = string de caracteres)
%D : date
%T : time autoexplicativos
%B : bool <S3>
```

Por ejemplo el mensaje ERROR podría completarse de la siguiente forma:

```
ERROR:"El título %s ya existe";
```

Cuando se despliegue el mensaje desde el programa se le pasará como parámetro el título del libro que se haya repetido.

Nótese también que si se desea colocar el signo 'porcentaje' (%) en un mensaje será necesario anteponerle otro signo %, por ejemplo:

```
MENSAJE:"La suma supera el 100%%."
```

con lo que en pantalla aparecerá:

```
La suma supera el 100%.
```

`without control field`

Mediante esta sentencia se evita que se incluya en la pantalla el campo de control.

`autowrite`

Esta cláusula permite indicar que se grabe el contenido de la pantalla automáticamente, cuando se pase por el campo de control. Esto tendrá efecto exclusivamente cuando se trate de un alta. Si el registro existiera para realizar una nueva grabación será necesario digitar la tecla de IDEAFIX.

`display status`

Esta cláusula indica mediante una palabra al pie de la pantalla el estado del registro en proceso: alta, baja o modificación.

La Cláusula Window

Como ya se ha mencionado, esta cláusula admite una serie de opciones que se describen a continuación:

`label`

Esta opción permite la definición de una etiqueta que aparecerá en el borde de la ventana que contendrá al formulario. La leyenda se indica con un string a continuación de la palabra clave, como muestra el ejemplo:

```
/* Etiqueta del formulario */
window label "Libros de la Biblioteca";
```

`fullscreen`

Esta cláusula dimensiona la ventana que contendrá el formulario a la totalidad de la pantalla (generalmente 80 columnas x 24 filas).

`origen`

Permite especificar las coordenadas del borde superior izquierdo de la ventana. Puede indicarse la fila solamente, o la fila y la columna:

```
window origin (4);
window origin (2, 10);
```

`border`

Define el atributo y los caracteres utilizados para armar el borde de la ventana. Las características del mismo se definen mediante las posibilidades ilustradas en la tabla siguiente:

Atributo de borde	Tipo de carácteres	Límites del borde	Colores del borde
blink bold reverse	single double asterisk	top low left right	red green blue yellow magenta cyan white

Figura 3.4 - Características de borde

Cuando no se pone ninguna de estas indicaciones, el borde se hace en los cuatro laterales de la ventana. Los atributos de la ventana pueden combinarse entre sí para obtener distintos tipos de borde. Por ejemplo:

```
window border (double, re<hy>verse);
```

Existe un tipo de borde llamado 'standard' que intenta dibujar un recuadro de acuerdo a las posibilidades de la terminal, y que en terminales con carácteres gráficos de línea equivale al ejemplo anterior.

`background`

Determina el color de fondo de la ventana creada para el formulario. Los colores posibles son:

```
red green yellow blue magenta cyan white
```

Estamos ahora en condiciones de agregarle a la pantalla del ejemplo, la sección %form, como se muestra en la ilustración siguiente:

```
LIBROS
Código del Libro: __________.
Título de la Obra: ______________________________
Código del Autor: __________.
Edición: _____
Fecha: _____/_____/_____
%form
// Esquema a utilizar
use biblio;
language C;
confirm end, update;
messages HELP1: Código del libro;
ERROR: El título ya existe;
window border standard,
label Libro de la Biblioteca;
```

Figura 3.5 - Añadiendo la sección %form

La Sección %fields

Esta parte del archivo FDL es obligatoria. Define los nombres de los campos incluidos en la pantalla, y sus atributos. Los nombres deben estar ordenados de acuerdo con la posición del campo en la pantalla, recordando que se enumeran de izquierda a derecha y de arriba hacia abajo.

La sintaxis de esta sección es una lista de nombres de campos seguidos por una lista opcional de atributos:

```
nombre_campo [: atributos];
```

Debe haber tantas sentencias como campos en la pantalla. La especificación de atributos para un campo es una lista de sentencias separadas por comas.

Los atributos permiten definir aspectos tales como relación con campos de bases de datos, mensajes de ayuda y error, validaciones a realizar sobre el valor del campo, etc. La siguiente sintaxis se aplica a ítems encerrados entre los signos de "mayor" y "menor":

<*string*>

Es cualquier cadena de caracteres encerrada entre comillas dobles.

```
"Soy una cadena" "This is a string"
```

<value>

Es una constante del tipo adecuado según la naturaleza del campo de formulario:

• Alfanumérico: Se admite como <valor> una cadena de caracteres.

• Numérico: Admite un número. La cantidad de dígitos y los decimales deben ser compatibles con el dibujo dado al campo en la imagen de la pantalla.

• Fecha: Una cadena de caracteres que represente una fecha válida. Por ejemplo:

```
fecha: between "01/01/86" and "29/02/88";
```

• Hora: Una cadena de caracteres que representa una hora válida, por ejemplo:

```
hora: between "06:00" and "17:30:45";
```

Las constantes especiales today y hour se refieren a la fecha y hora corrientes respectivamente, y se pueden usar como <valor> para campos de tipo fecha y hora respectivamente. También se puede hacer referencia a una variable de ambiente indicándola con $var, la que debe estar definida en el momento de ejecución.

• *Referencia*: este tipo especial de campo permite la definición de ciertos campos como "polimórficos". Esto significa que el campo puede aceptar diferentes tipos de datos en forma

nativa, por ejemplo:

```
_____
____.
____. _____ _____
%form
%fields
a;
b: reference(r1..r4), is cod
r1: internal char(10), check this != "hola"
r2: internal num(4);
r3: internal time;
r4: internal date;
...
```

donde *b* es el campo de referencia el que puede cambiar su tipo en tiempo de ejecución a uno de los especificados dentro de la lista de campos de la cláusula de referencia. En este caso los campos son los comprendidos en el rango "r1 .. r4". Todos estos campos deben ser *internals* y deben aparecer luego del campo de referencia.

b es un campo numérico de cuatro posiciones (en correspondencia a lo que se definió en la imagen del form) e indica el campo, dentro del rango de campos *internals*, que en algún momento es usado como definición.

Por Ejemplo: si el campo tiene valor 1, esto significa que cuando se ingrese este campo, será del tipo char(10) y también deberá elegir la condición de *check* específica (puede ser igual al string "hola"); si el campo toma valor 2, se verá como un campo numérico de cuatro posiciones, si su valor es tres se verá como un campo de hora y finalmente si es cuatro, se verá como un campo de fecha.

El campo *b* tomará el valor necesario manualmente (por medio de la función FmSetFld), o automáticamente (por medio de la expresión *is*). El uso más comúnes el de asociarle al campo una expresión *is*, la que le dará su valor y definirá su tipo dinámicamente.

Se debe tener en cuenta que el campo b tendrá que contener cualquiera de los tipos correspondientes a la lista de campos de la cláusula de referencia, así que el espacio que se le otorga en la imagen del form debe ser aquella del tipo de mayor longitud.

Como podemos ver, cada campo perteneciente al rango puede tener su propio atributo, que será respetado cada vez que el campo tome su forma.

Restricciones en campos polimórficos

El campo de referencia (en el ejemplo era b) no puede ser asociado a la base de datos. Los campos de referencia tienen las mismas restricciones que los campos de tipo *internals*.

Finalmente, es bueno repetir que los campos de referencia deben ser siempre de tipo internal, estar juntos en una zona de campo, aparecer luego del campo de referencia y aparecer en forma ascendente dentro de la lista de la cláusula de referencia.

Cuando se usa un valor al definir un atributo de *check*, se verifica que sea compatible con el

campo de formulario, y en el caso del atributo default, que verifique el atributo de check si es que hay alguno.

En el caso de usar una variable de ambiente como valor, no puede hacerse ningún control al momento de compilar el formulario, por lo que el mismo queda pospuesto hasta que el formulario sea utilizado.

Como se ha mencionado anteriormente, existen campos de tipo múltiple y agrupado con lo que se tienen básicamente campos:

• Simples.

• Múltiples.

• Agrupados.

Los campos simples son aquellos en donde se realiza ingreso de datos, y que contendrán un valor. Los campos múltiples son matrices formadas por columnas, que contienen conjuntos de valores; cada una de ellas corresponde a un campo. Los campos agrupados se utilizan para realizar validaciones cruzadas entre distintos componentes de un formulario.

Los campos múltiples y los agrupados están siempre formados por uno o más campos simples.

Atributos Válidos para Todo Tipo de Campo

`not null`

El campo debe tener un valor.

`descr[iption] MSG`

Define un mensaje que aparecerá en el borde inferior de la pantalla toda vez que el usuario esté sobre el campo. MSG debe ser un mensaje definido en la tabla de mensajes definida con el atributo MESSAGES de la sección `%form`.

`display only`

Los contenidos del campo no deben ser modificados. El usuario puede posicionarse sobre dicho campo, pero no le está permitido alterar su valor.

`display only + is`

Cuando estas cláusulas están combinadas, cuando el if no retorna valor void, el campo no será *skip* (permite posicionarse sobre el mismo), pero será *read only* (no se podrá modificar). Por ejemplo:

```
   a: display only, is b==1? c: void;
```

`display only when expr`

Esta cláusula hará que el campo sea display only cuando la condición en expr sea verdadera. Por ejemplo:

```
a: display only whenb==1;
```

Nota: esta cláusula funcionara como una abreviación de:

```
a: display only, is b==1 ? this : void;
```

`skip`

Saltea el campo en el proceso de ingreso de datos (salta al próximo campo). Esto también funciona para campos agrupados y múltiples, salteando completamente la estructura de campo.

`skip when cond`

Implementa un skip cuando la condición retorna verdadero. Por ejemplo, para campos simples y agrupados:

```
_______.
{ _______. _____. }
_______.
%form
%fields
a;
agrup: skip when a > 1;
b;
c;
d: skip when a > 2;
```

En el caso en que *a* sea 2, el próximo campo al cual saltará el cursor será *d*.

Atributos para Campos Simples

`on help {<MSG>|manual}`

Si el usuario digita la tecla <HELP> mientras está posicionado en este campo, muestra el texto asociado a MSG definido en la cláusula messages de la sección `%form` o devuelve el control al programa en el caso de que se especifique manual.

Esto permite al programador incluir rutinas que requieran mayor elaboración que las ofrecidas por IDEAFIX. La opción también permite especificar títulos para las ventanas generadas por las ayudas. Un ejemplo de su aplicación se verá más adelante cuando se presenten las cláusulas que generan ventanas.

Cuando no se especifica mensaje de ayuda para un campo y se la solicita, se despliega un mensaje que indica el tipo del campo y la cantidad de posiciones que ocupa.

`on error <MSG>`

Se despliega el texto asociado a MSG, definido en la cláusula *messages* de la sección `%form`, como mensaje de error cuando el usuario ingresa un dato inválido en el campo.

`mask <string>`

Realiza una operación de máscara sobre el campo. Ver *Capítulo 2-Descripción de Sentencias DDS*, en la subsección CREATE TABLE, mask, de este Manual.

`default <valor>`

El valor por omisión para un campo será el valor indicado. Puede usarse una variable de ambiente mediante la indicación:

```
   default $VARIABLE
```

`length<valor>`

Un campo tipo "char" o "num" posee dos longitudes asociadas: una real, que es la verdadera longitud del campo y otra de salida, que es una longitud virtual dependiente de la imagen especificada en el form. La longitud real debe ser siempre menor o igual que la de salida.

Cuando el campo no está asociado con la base de datos entonces el atributo *length* permite definir la longitud real. Si ésta última es diferente a la de salida, en el ingreso de estos dos tipos de campos se podrán utilizar la teclas de movimiento de cursor para hacer "scroll", y poder ver el campo completamente.

`autoenter`

Mediante este atributo, se establece que no será necesario digitar para pasar de campo, sino que cuando se complete la cantidad de posiciones del mismo, se efectúa un pase de campo automático.

`<campo_tabla>`

El campo de pantalla hereda los atributos del campo de base de datos especificado. Los atributos en campos de tablas tienen el mismo significado en los formularios. La única diferencia es que no tienen significado en el formulario ni la clave primaria, ni la cadena de descripción. Si los tamaños de los campos no coinciden pueden suceder las siguientes alternativas:

• Si el campo definido en el formulario tiene una longitud mayor que la del campo de la Base de Datos, se toma la longitud del segundo.

• Si el campo definido en el formulario tiene una longitud menor que la del campo de la Base de Datos y el tipo de dato no es ni "char" ni "num", en el momento de la compilación

aparecerá un mensaje de error que dice:

```
"pant.fm", line nn, (48). La longitud del campo 'campo' es
incorrecta. Debe ser mayor o igual que la de la base de datos.
```

En caso que el dato sea de tipo "num" o "char", la longitud definida en la base de datos establecerá la longitud real, mientras que la de salida quedará determinada por la longitud en el form (ver atributo *length*).

• Si el campo definido en el formulario tiene una longitud menor que la del campo de la Base de Datos y tiene un atributo skip, se toma la longitud del campo del primero.

• Si el campo definido en el formulario tiene una longitud menor que la del campo de la Base de Datos y el último tiene asociado un atributo mask, en el momento de la compilación aparecerá un mensaje de error:

```
"pant.fm", line nn, (10). La máscara "cadena"
no coincide con la longitud del campo.
```

La opción de *fgen* **-w** permite informar cuándo se han hecho ajustes.

El esquema al cual pertenece la tabla debe haber sido indicado en la sentencia use. La forma de especificar la asociación entre campo de base de datos y de formulario es la siguiente:

```
esquema.tabla.campo
```

Si es el esquema corriente (es decir el primero de la sentencia use se puede omitir la especificación de esquema:

```
tabla.campo
```

Si además el campo del formulario tiene el mismo nombre que el campo en la tabla, la indicación se reduce a:

```
tabla.
```

El siguiente es un ejemplo:

```
%form
use biblio;
%fields
codigo: libros.;
```

Se dan más detalles sobre la herencia de atributos de base de datos en la sección *Interfaz con la Base de Datos*.

`[manual] subform(<cadena> [, ...])`

Asocia uno o más formularios. Se explica en detalle en la sección dedicada a subformularios.

`check`

Los atributos de "check" (verificación) permiten especificar una validación sobre el valor ingresado en el campo, y son los siguientes:

• Operadores relacionales (>, <, >=, <=, !=, ==).

• Operador [not] between.

• `[not]in ... (valores)...` . Se especifica una lista de valores que se admiten en el campo. También puede indicarse la validación contraria (not), es decir que el campo no puede contener ninguno de los datos indicados.

• `check expresión`. Combina las anteriores, pudiendo indicarse condiciones que involucren a otros campos del formulario.

• `[not]in tabla [descendente]`. Esta validación permite controlar que un valor exista en una tabla de base de datos. Con *not* se excluyen los valores sobrantes en la tabla.

Estos atributos se explican más en detalle en la siguiente sección *Atributos de Check*.

`is ...`

Este atributo permite definir campos virtuales. Su especificación completa se detalla en la sección *Atributo IS* de este capítulo.

Atributos de Check

Estos atributos permiten especificar validaciones sobre el valor de los campos de pantalla.

`relop {<valor> | campo}`

Los valores que no verifican la condición no serán aceptados y se los tratará como errores. La condición se puede expresar con cualquiera de los siguientes operadores relacionales:

> < >= <= == !=

Por ejemplo se puede especificar:

```
codigo: > 10;
```

Es posible también realizar operaciones lógicas entre campos del formulario, siempre y cuando el campo contra el cual se compara esté definido previamente. Por ejemplo:

```
%fields
. . . . .
desde ;
hasta : >= desde;
```

`[not] between <valor> and <valor>`

Igual que en el punto anterior, pero la condición da un rango de validez, incluidos los extremos. valor1 debe ser menor que valor2. Por ejemplo:

```
codigo: between 10 and 100;
```

`[not] in (<valor>[: <cadena>] [, <valor> ...])`

Los valores aceptados deben pertenecer al conjunto indicado. Opcionalmente se puede indicar una descripción asociada a cada valor. Este atributo da al campo la característica de que cuando se pide ayuda sobre el mismo, se despliegue una ventana de tipo menú, mostrando como opciones los valores del conjunto, y su descripción asociada si es que la hay. El siguiente es un ejemplo:

```
tipo: in (1:"Paises", 2:"Idiomas");
```

que aparece en pantalla de la siguiente forma:

Si se desea cambiar la etiqueta por omisión de la ventana ("Opción a Seleccionar") se puede combinar esta cláusula con la on help, de la siguiente forma:

```
. . . . .
%form
messages TITULO: "Cód. Descripción";
%fields . . . . . tipo: in (1: "Paises", 2: "Idiomas"),
on help TITULO;
```

En este caso la etiqueta de la ventana pasará a ser la cadena de caracteres contenida en el mensaje "TITULO". Cuando el atributo está prefijado por not, la validación es a la inversa, es decir, el valor se rechaza si pertenece al conjunto.

`check expresión`

Este atributo permite construir condiciones más complejas que involucren el campo actual del formulario (indicado con *this*) y cualquier otro campo (indicado con su nombre). La expresión se puede armar con cualquiera de los operadores anteriores. Por ejemplo:

```
precio: check (this<=1000 and cantid*this<=100000);
```

en este ejemplo se especifica que los precios no deben exceder los mil pesos, y el total no debe exceder los cien mil pesos. Notar el campo cantidad debe ser definido previamente.

`[not] in table`

Esta validación permite verificar que el valor ingresado en el campo exista (o no, si se usa *not*) en una tabla de base de datos.

En caso de solicitar ayuda en un campo de un formulario que tenga asociada esta cláusula, (ya sea explícitamente o bien heredada de la Base de Datos) se desplegará una ventana similar a la de las cláusulas in, vistas en el punto anterior. Esta ventana incluirá todas las alternativas posibles de selección para ese campo, efectuando a su vez la validación luego del ingreso de datos.

Como en el caso anterior, puede especificarse un título a la ventana generada, combinando esta opción con la cláusula *on help*.

Debido a la gran variedad de alternativas que ofrece esta cláusula, se explica en profundidad en la siguiente sección.

También, los atributos check pueden referirse a un grupo de campos. Por ejemplo:

```
{ ______. ______. }
%form
%field
agrup: check a > b;
a;
b;
```

Atributo *in table*

Consiste en verificar que el valor que se le da a un campo de un formulario exista en una columna de una tabla de base de datos. La verificación se hace realizando una búsqueda por clave sobre la tabla donde debe existir el valor. En el caso más simple la lectura se realiza por la clave primaria. A la clave que se está buscando se le da el valor El valor que se le da a la clave a buscar es el que tiene el campo que se está validando. Supondremos en primera instancia que el valor del campo se puede usar directamente para realizar una búsqueda.

El resultado de la validación es entonces el resultado de la búsqueda, es decir, si la búsqueda es exitos a la validación verifica.

Cuando un campo tiene un atributo 'in tabla' y se pide ayuda sobre el mismo se desplegará una ventana menú que muestra el contenido de la tabla para permitir la selección de uno de sus valores. En dicha ventana se muestran los valores posibles para el campo, y asociado a cada uno de ellos una descripción, de la misma forma que en la validación 'in' con un conjunto de valores.

La sintaxis para expresar esta validación es:

```
[not] in tabla [by indice [(valor,...)][:(campo1,...)]
```

donde *tabla* es aquella donde se hace la búsqueda, y debe pertenecer a alguno de los esquemas indicados en la sentencia use. Los campos: campo1, campo2, ... pertenecen a dicha tabla, y se los denomina "campos de descripción". En el formulario usado como ejemplo, existe el campo autor. Su contenido debe existir en la tabla autores, del esquema biblio, por

tanto:

```
autor: in autores:(nombre)
```

Para efectuar la validación se toma el valor dado a 'autor' en el formulario y se lo usa como clave primaria para realizar una búsqueda en autores. Ahora supongamos que el campo sobre el cual hacemos una validación se puede utilizar para hacer una búsqueda por clave, pero que no es la clave primaria, sino otro índice. Entonces se debe indicar el nombre del mismo -supongamos sea *titulo*- de la siguiente forma:

```
titulo: not in libros by titulo;
```

Ahora bien, es frecuente que el campo sobre el cual se realiza la validación no baste por sí solo para hacer la búsqueda por índice, debido a que se desea usar una clave compuesta por varios campos.

Una validación *in table* usando un índice compuesto debe especificar de qué modo completar la clave. Siempre se usará como último componente de la misma el campo de formulario a validar. Para los demás valores de la clave se puede especificar:

• Un valor constante. La forma de definirlo es la misma que para definir <valor>, como se mostró anteriormente al explicar los atributos de check.

• Un valor de otro campo del formulario.

• Una variable de ambiente.

• Un campo de la Base de Datos.

Por ejemplo, la tabla *grales* tiene un índice formado por dos campos numéricos. El primero es el tipo de registro y el segundo un código. En el formulario se quiere validar que los campos tipo y código existan en la tabla grales, para lo que se indica:

```
tipo:;
codigo: in grales(tipo):descrip;
```

Con esta especificación, cuando se valide el valor del campo código se usará el valor del campo tipo del formulario como primer componente de la clave, completándola con el valor de código.

Si la verificación se hiciera con un valor constante, podríamos tener por ejemplo:

```
codigo: in grales(1):descrip;
```

Aquí se usará el valor constante 1 (uno) como primer campo de la clave, y el valor del campo a validar en último lugar.

Con una variable de ambiente se procede de manera similar, pero debe ir precedida por un símbolo '$'. Por ejemplo:

```
codigo: in grales($VARAMB):descrip;
```

donde 'VARAMB' es el nombre de una variable ambiental, el cual debe escribirse en mayúsculas o minúsculas según haya sido definida dicha variable (no es indistinto aquí el uso de uno u otro tipo).

Finalmente, con un campo de la Base de Datos se especifica:

```
codigo: in grales(tabla.campo):descrip;
```

Donde *tabla.campo* es la referencia al campo *campo* de un registro, de la tabla *tabla*, que fue accedido, previamente, por otro campo del formulario mediante un atributo in table.

Resumiendo, véase la diferencia entre las distintas notaciones posibles para una misma designación:

empresa Alude al campo así llamado dentro de la tabla a la que pertenece el campo que está siendo validado.

$empresa Se refiere a la variable de ambiente denominada 'empresa', que contiene habitualmente el nombre o razón social del cliente que utiliza el sistema.

emps.empresa El campo 'empres' ya no pertenece a la tabla sobre la cual se está trabajando, sino a otra llamada 'emps'.

Los Campos Descripción

Los campos del registro accedido por el atributo in table, pueden ser copiados automáticamente sobre campos del formulario si se cumplen las siguientes condiciones:

• Los campos del formulario están asociados a la tabla accedida.

• Estos campos están a continuación del campo que tiene el atributo 'in tabla'.

• Los campos tienen el atributo skip, o bien display only.

En el siguiente ejemplo:

```
autor: libros., in autores:(nombre);
desc1: autores.nombre,display only;
```

cuando se cargue el valor del campo autor, y este se encuentre en la tabla autores, se copiará el valor del campo nombre del registro encontrado sobre el campo de formulario *desc1*.

Atributo *is*

Este atributo permite definir campos virtuales, es decir campos cuyo valor resulta de alguna expresión. Puede implementarse de la siguiente forma:

```
campo : is expr;
```

donde expr puede ser:

```
exp_valor { +, -, *, / } exp_valor | (exp_valor) |
{ campo | constante | $var_ambiente |
función | today | hour }
función : sum (campo)
| avg (campo)
| max (campo)
| min (campo)
| count (campo)
| descr (campo)
| help (nombre_tecla)
| num (expr)
| date (expr)
| time (expr)
```

La asignación de un atributo *is* a un campo, lo convierte a su vez en uno el tipo *skip* para que el operador no pueda modificar su valor.

Es obvio que esto debe ser así puesto que se le está asignando al sistema la responsabilidad de fijar el contenido del campo. Las funciones *sum*, *avg*, *max*, *min* y *count* sólo aceptan como parámetros campos pertenecientes a un múltiple. A su vez, *sum* acepta solamente campos numéricos (en sentido restringido) y *avg* no acepta campos alfanuméricos, pero sí por ejemplo del tipo "time" podría utilizarse para determinar la hora promedio de llegada o de salida de un grupo de empleados. El tipo del resultado de cada función se detalla a continuación:

• *sum*, *count* y *num* son siempre numéricas.

• *descr* y *help* son alfanuméricas.

• *date* es de tipo fecha.

• *time* es de tipo hora.

• *avg*, *max* y *min* son del mismo tipo que su argumento.

Las expresiones combinadas deben estar compuestas por términos que tengan el mismo tipo. De lo contrario se obtendrá un mensaje de error indicando que se están mezclando valores de tipos incompatibles. A continuación se explican las distintas funciones, acompañadas de algunos ejemplos de sus aplicaciones. Supongamos un formulario de carga de asientos, semejante al siguiente:

```
Cuenta Descripcion Debe Haber
[ ____. _______________ _,___,___.__ _,___,___.__ ]
. . . .
Total Debe Total Haber
_,___,___.__ _,___,___.__
Diferencia _,___,___.__
```

La función *descr* asocia a un campo con las descripciones de un atributo *in* de otro campo, por ejemplo:

```
Tipo: __.
Descripción: ____________
```

```
. . . . .
%fields
tipo: in (1:"Proveedor", 2:"Cliente");
desc0: is descr(tipo);
. . . . .
```

En el campo `desc0` de la pantalla se copiará automáticamente la descripción asociada con el valor ingresado en el campo tipo.

La función *help* devuelve la tecla de función asociada con un nombre de función. Su argumento debe ser un nombre de tecla válido. El resultado de la función variará según la terminal. Un ejemplo de su aplicación podría ser:

```
. . . . .
Presione ___ para generar el listado.
. . . . .
%fields
. . . . .
ayuda : is help (PROCESAR);
```

y en pantalla podrá aparecer lo siguiente:

```
Presione F5 para generar el listado.
```

Las funciones *num*, *date* y *time* convierten, si es posible, la expresión pasada como parámetro al tipo de la función.

Por ejemplo:

```
campo : is date ($fecha);
```

donde la variable de ambiente fecha contiene una cadena de carácteres que puede ser convertida a un valor de tipo date.

Las teclas que se pueden utilizar son las que se detallan en la tabla ilustrada en el *Apéndice A, del Manual del Usuario.*

Atributo *on help in table*

Es posible aprovechar las facilidades de selección de un valor mediante una ventana menú para un campo, pero sin obligar a que se realice la validación. Esto permite que el usuario examine los valores obrantes en la tabla y luego decida adoptar uno de ellos, o bien otro que no se halle incluido en la misma.

Esto se logra con la cláusula *on help* precediendo al atributo *in help*:

```
autor: on help in autores:nombre;
```

Con esta cláusula la validación no se realiza, pero si el usuario pide ayuda sobre el campo se mostrará la ventana menú con los valores existentes. Sigue vigente la propiedad de copia de los campos de descripción.

Interfaz con la Base de Datos

La asociación de un campo de formulario con el de una tabla de base de datos tiene una serie de implicancias que se verán a continuación. Debe tenerse en cuenta que el proceso de herencia de los atributos se produce en el momento de procesar la definición del formulario con el utilitario *fgen*. Por lo tanto un cambio de atributos en la base de datos no implicará que se refleje en los formularios hasta que se reprocesen éstos. Esto puede producir efectos indeseados, como cambiar un valor previo válido en uno no válido; en realidad esto debe ser cuidadosamente tenido en cuenta antes de introducir dicha alteración.

Atributos Heredados

Los campos especificados en un formulario heredan de la base de datos los siguientes atributos:

flags La especificación de not null.

longitud La longitud que predomina es la especificada en la base de datos. Si la indicada en el formulario es menor que la del Data Base, entonces se señala que existe un error en la es pecificación, a menos que el campo sea display only o skip, en cuyo caso el valor podrá aparecer truncado en el formu lario. En caso que sea mayor, la longitud se ajustará a la de la base de datos (esto es indicado cuando se compila el for mulario con la opción -w del fgen).

máscara Si en la base de datos ha sido definida una máscara para un campo, esta también es heredada al formulario.

default Los valores definidos por defecto; es decir, cuando no se explicita un contenido y se asume un valor por el sistema.

check Los atributos de check (check expresión, between, opera- dores relacionales, in tabla o in con un conjunto de valores) son heredados por el campo de formulario.

Compatibilidad

Para poder relacionar un campo de formulario con uno de base de datos, es necesario que sus tipos sean compatibles. Seguidamente se muestran las combinaciones válidas:

Base de	Formularios	Datos	CHAR	TIME	DATE	NUM
FLOAT	BOOL	CHAR	x			
		TIME		x		
		DATE			x	
		NUM	x[*]			x
		FLOAT				
x		BOOL				

	x					

Figura 3.6 - Correspondencia de Tipos DB/Form

En caso de que el campo sea numérico y tenga decimales, se valida que la cantidad de decimales especificados en el formulario sea exactamente igual a la de la base de datos.

Las compatibilidades indicadas con [*] son válidas sólo si el campo tiene una máscara completamente numérica. A continuación se explica esta característica con más detalle.

Máscaras Numéricas

En caso de tener un campo en un formulario con una máscara definida y que ésta sea toda numérica, se puede tratar a dicho campo como un campo numérico, es decir, que se pueden hacer operaciones tales como obtener su valor desde un programa "C" mediante la función de interface de IDEAFIX FmIFld, aunque esté especificado como una cadena de caracteres.

Sin embargo en el ingreso de datos no se obtendrá justificación a derecha como ocurre en los campos numéricos, sino que se manejará mediante la máscara indicada.

Cuando se compila el formulario con el utilitario *fgen*, se analizan todos los atributos definidos en el campo para encontrar una definición de máscara, antes de indicar que existe incompatibilidad de tipos entre la definición del formulario y de la base de datos, en caso de que se haya indicado al campo como una cadena de caracteres, y en la base de datos como un numérico.

El siguiente ejemplo muestra un formulario completo con un campo alfanumérico con máscara numérica:

```
INGRESO DE AUTORES
Código del Autor : ____ |
Nombre del Autor : _____________________________
Nacionalidad : __._______________
%form
use biblio;
window label "Autores", border standard;
%fields
codigo : autores., mask "nn-NN" ;
nombre : autores. ;
nacion : autores. ;
descrip : grales., skip;
```

En este caso se define como una cadena de caracteres al campo codigo, aunque en la base de datos este definido como numérico, debido a que se define la máscara toda numérica "nn-NN", que quiere decir que habrá cuatro dígitos, los primeros dos no obligatorios y los últimos dos si, separados por un guión ("-").

Superposición de Atributos

Cuando en un campo se repite un atributo, queda como efectivo el último de ellos. Por ejemplo si se indica dos veces el valor default, se usará el último.

Un campo de formulario puede poseer atributos definidos en la definición FDL, o bien recibirlos por herencia desde la base de datos. Este es un caso de potencial superposición si en el formulario se define un atributo que ya tenía el campo de base de datos.

Predominará siempre la definición en el formulario ya que la asociación con la base de datos va siempre como primer atributo. Un ejemplo de tal situación es:

```
Base de datos:
fecha date descr "Fecha de edición"
default today,
Formulario:
fecha: libros., default "01/01/84";
```

Se tomará como default el valor definido en el formulario (01/01/84).

Para averiguar cuándo se producen superposiciones de atributos, utilizar la opción **-w** del utilitario *fgen*.

Subformularios

Como se señaló anteriormente, es posible asociar uno o más subformularios a un campo simple (los campos pertenecientes a un múltiple o a un agrupado pueden tener asociados subformularios). El subformulario se diseña de la misma manera que cualquier formulario. Se despliega una vez que el campo al cual está asociado se completa. En ese momento se crea una ventana, se despliega el subformulario y se realiza la captura de datos. Cuando el proceso termina, la ventana con el subformulario desaparece.

Los subformularios resuelven básicamente dos problemas:

1. Permitir el ingreso de un conjunto de datos relacionados con un campo determinado, pero que no hace a la esencia del contenido del formulario principal y por lo tanto no es necesario su despliegue permanente. También es el caso en el cual el formulario es demasiado complejo y requiere una gran cantidad de datos, por lo que no cabe en una única ventana, o bien la información queda muy abigarrada.

2. Permitir ingresar datos con diferente formato, de acuerdo a los distintos valores que puede asumir un determinado campo.

Esta asociación se especifica con el atributo subform. El argumento que acompaña a esta palabra clave contiene el nombre de un subformulario. Por ejemplo:

```
codigo: subform "cod1";
```

En este caso, se invocará al subformulario `cod1` cada vez que se altere el campo `codigo`; o aún sin modificarse, cada vez que se digite la tecla <META> sobre el campo `codigo`.

Los subformularios pueden estar anidados hasta ocho niveles. Esta restricción se debe a que sólo se pueden tener abiertos esta cantidad de formularios o subformularios simultáneamente. En el caso de que un campo tenga asociados más de un subformulario, la especificación será:

```
codigo: subform ("cod1", "cod2", "cod3");
```

Se invocará a los subformularios *cod1*, *cod2* o *cod3*, de acuerdo al valor que tome el campo código, que debe obligatoriamente tener un atributo in cuya cantidad de valores admitidos coincida con la cantidad de subformularios. Si algunas de las opciones del atributo in no se corresponde con ningún subformulario, se lo debe especificar con *null*. Por ejemplo:

```
codigo: subform ("cod1", null, "cod3");
```

Supongamos que un campo de un formulario tiene asociado un subformulario y el atributo *default*. Cuando se está en un alta (FM_NEW) y se pasa por dicho campo, aunque no se cambie el valor del mismo, se desplegará el subformulario. Esto es necesario ya que por tratarse de un alta el subformulario no tiene todavía datos cargados. Si se tratara de una modificación (FM_USED) esto no ocurre. Será necesario modificar el valor del campo para que se despliegue el subformulario o bien digitar la tecla sobre el mismo.

Campos Múltiples

Un conjunto de campos puede agruparse para formar un campo múltiple. Dicho conjunto se repetirá en una cierta cantidad de filas, formando de este modo una matriz. La primera columna de esta matriz se denomina "campo rector". Las dimensiones de los campos múltiples son: tantas columnas como campos diferentes comprenda y tantas filas como se especifique con el atributo *rows*.

Hay teclas de función que permiten paginar en un campo múltiple, para consultar la información desplegada de una manera confortable, así como eliminar e insertar filas, etc.

Para definir un campo múltiple se encierran entre corchetes los campos de pantalla que se incluirán en la matriz. Para ejemplificar este caso, nos vamos a alejar un poco del esquema de la biblioteca, de forma tal que se pueda ver claramente el sentido de los campos múltiples. Así por ejemplo:

```
nrocuenta escripción débitos créditos
[______._ ________ ___,___.__ __,___.__]
%fields
grancampo: rows 30, display 5;
nrocuenta:;
descrip:;
debitos:;
creditos:;
```

En este ejemplo se define con grancampo el nombre del multirenglón, especificando que tendrá treinta filas (rows 30), y que se desplegarán de a cinco por pantalla (display 5). La cantidad de filas totales (rows) debe ser múltiplo de la cantidad de filas que se desplieguen

(display).

Atributos de Campos Múltiples

Se verá primero qué significado tienen los atributos que se aceptan en cualquier tipo de campo en el caso de aplicarlo sobre un campo múltiple.

`not null`

Cuando se realiza el ingreso de datos, debe ingresarse por lo menos una fila de la matriz si se especificó este atributo sobre el campo múltiple. De ser especificado sobre el campo rector (primero después de la definición del múltiple, en el ejemplo "nrocuenta"), se ignora.

Si aparece sobre cualquier otro campo definido dentro de un múltiple, será obligatorio si se cargó el campo "rector" de la fila correspondiente.

`display only`

Todos los campos comprendidos en el múltiple son de despliegue exclusivamente.

skip when condición

En caso de cumplirse la condición, realiza un skip de todos los campos del múltiple.

Los campos que forman el múltiple podrán tener atributos de campos simples, con la excepción del campo rector que no podrá tener asociado el atributo default.

Atributo Unique

Sobre un campo que forma parte de un múltiple se puede indicar el atributo "unique" con dicha palabra clave. Mediante este atributo se indica que se verifique que el valor ingresado no esté repetido en alguna fila anterior o posterior.

Atributos Específicos de Campos Múltiples

Los atributos que se indican a continuación se pueden especificar solamente en la definición del campo múltiple, es decir, en el campo virtual que lo nombra.

display <número>

Es la cantidad de filas, sobre el total de la matriz, que se mostrarán simultáneamente en la pantalla.

rows <número>

Es obligatoria su presencia ya que define la cantidad de filas totales que tiene la matriz. Debe ser un múltiplo de la cantidad indicada en *display*.

```
ignore [delete] [add] [insert]
```

La operación indicada no es permitida al usuario sobre la matriz.

• *delete* se refiere a la posibilidad de borrar una fila completa.

• *add* se refiere a la posibilidad de agregar nuevas filas a continuación de la última.

• *insert* es la posibilidad de insertar una fila en una posición cualquiera dentro de la matriz.

Campos Agrupados

Existen casos en los cuales las validaciones sobre datos en los campos de pantalla se realizan teniendo en cuenta valores de otros campos. Un caso típico es el ingreso de dos fechas, donde la segunda debe ser mayor o igual que la primera. Sobre este campo (el de la segunda fecha) no hay forma de especificar una validación, dejándola entonces a cargo de un programa "CFIX". Este verificará que cada vez que se ingrese una fecha en el segundo campo sea mayor o igual que la primera. Sin embargo, dadas las posibilidades de movimiento del cursor a través de los campos en pantalla, el programa debería tener en cuenta (y eventualmente restringir) determinados movimientos, para poder efectuar fácilmente la verificación.

Los campos agrupados simplifican la lógica de este problema. Al igual que los múltiples, los campos agrupados son un conjunto de campos sucesivos de pantalla que generan un campo virtual. Se definen en la pantalla encerrando sus componentes entre llaves:

```
Fecha Desde Fecha Hasta
=========== ===========
{ __/__/__ __/__/__ }
```

Al igual que en los campos múltiples, esta definición da origen a un campo virtual que debe ser declarado. Los campos agrupados pueden anidarse y pueden estar dentro de un campo múltiple. En la sección que explica la interfaz con el lenguaje "C" se da un ejemplo del uso de campos agrupados.

Atributos para Campos Agrupados

Además de los atributos especificados en "Atributos para todo tipo de campo" se pueden colocar expresiones de checks sobre los campos agrupados. Ejemplo:

```
agrup: check a > b;
a:;
b:;
```

Atributos para Campos dentro de un Agrupado

En un campo incluido dentro de un agrupado se puede indicar el atributo:

```
check after nombre_campo
```

donde *nombre_campo* designa a un campo agrupado que contiene a aquél al cual se le está indicando esta validación. Esto tiene como efecto posponer la aplicación de los atributos de check de dicho campo hasta que se salga del campo agrupado "nombre_campo". Supongamos, a título de ejemplo, tener los siguientes campos:

```
agrup;
tipo :;
codigo : in grales(tipo);
```

Con estas definiciones, si hemos ingresado el valor 1 en el campo tipo, y se está posicionado luego en el campo codigo, y no existiese ningún registro con valor 1 en tipo, la validación in no se verificará nunca, y no se podrá salir del campo.

En cambio agregando:

```
codigo : in grales(tipo), check after agrup;
```

la validación se aplicará cuando se salga del agrupado, es decir que se puede retroceder con el cursor para cambiar el valor del campo tipo sin que se valide el valor en codigo. El *check* recién se aplicará cuando se salga del agrupado, es decir cuando se oprima <INGRESAR> o <CURS_ABA> en el campo codigo, o saliendo por cualquier otra dirección. Esto también soluciona el problema que se presentaría si el campo agrupado estuviera dentro de un campo múltiple. Si se modifican los valores de uno de los campos incluidos en el campo agrupado, pero en lugar de recorrer toda la fila del campo múltiple, se desplaza el cursor por las columnas, la validación se realizará de la misma forma, ya que se efectúa al salir del campo agrupado.

Campos Agrupados dentro de un Campo Múltiple

Es posible definir campos agrupados dentro de un campo múltiple. Esto permite hacer validaciones cruzadas en cada fila del multirrenglón y extender la unicidad de las filas a más de un campo. Por ejemplo, si consideramos el siguiente formulario de carga de comprobantes:

```
. . . . .
Tipo Comp. Nro. Comp. Fecha Importe
[ __. _,___,___. __/__/__ _,___,___.__]
. . . . .
%fields
. . . . .
multip : rows 100, display 5; /* Campo virtual múltiple */
tipcmp : comps.;
nrocmp : comps.;
feccmp : comps.;
impcmp : comps.;
```

En este caso sería conveniente validar que no se repitan los comprobantes en las columnas. Con nuestros conocimientos hasta este momento, esta tarea no la podríamos realizar automáticamente. Podríamos utilizar el atributo unique pero sólo en una de las columnas lo que no produciría el efecto deseado. Recurriendo a los campos agrupados deberíamos hacer lo

siguiente:

```
. . . . . .
Tipo Comp. Nro. Comp. Fecha Importe
[{ __. _,___,___.} __/__/__ _,___,___.__]
. . . . . .
%fields
. . . . . .
multip : rows 100, display 5; /* Campo virtual múltiple */
agrup : unique; /* Campo virtual agrupado */
tipcmp : comps.;
nrocmp : comps.;
feccmp : comps.;
impcmp : comps.;
```

Esto nos asegura que no se repitan los comprobantes en las líneas del multirrenglón. Se declara como unique al agrupamiento de los campos *tipcmp* y *nrocmp* que es muy distinto a declarar independientemente *unique* a cada uno de los campos.

Formularios con Zona de Clave

Cuando se vieron las operaciones que se realizaban con los formularios, se estableció que una de ellas era la de traer un formulario existente.

Debe existir algún modo de individualizar los datos que se buscan. Para ello es posible definir en la imagen de un formulario una primera parte (compuesta por un número cualquiera de campos), llamada "**Error! Bookmark not defined.**zona de claves".

Cuando un formulario tiene *zona de claves*, se lo puede imaginar dividido en dos secciones. La primera se comporta como un formulario clave de entrada, de modo que una vez completado el último campo se realizará la operación denominada LEER.

En el formulario electrónico se busca que el usuario complete inicialmente el formulario clave. Luego se utiliza la información ingresada para buscar el formulario completo en la base de datos. De no encontrarse, la sección de datos aparece en blanco para ser completada y archivada.

Hay otras dos operaciones muy similares a LEER, llamadas LEER_SIGUIENTE y LEER_PREVIO. Especifican respectivamente el formulario con la clave siguiente, o la anterior, al que se tiene en pantalla.

Operación	Tecla de Función
Zona de Datos	AGREGAR ACTUALIZAR REMOVER IGNORAR
<PROCESAR> <PROCESAR>	*Zona de Claves o Campo de Control*

<REMOVER> <IGNORAR>	
LEER_SIG LEER_ANT	<PAG_SIG> <PAG_ANT>
Zona de Claves	LEER
Completar zona de claves	Cualquier Posición
FIN	<FIN>

Figura 3.7 - Validez de las Operaciones de Formularios

La tabla está dividida según las zonas del formulario, indicando el significado de cada tecla de función activada sobre dicha zona. En el caso de la zona de claves, el solo hecho de completar sus campos provoca la operación LEER.

Una vez dentro de la zona de datos, no es posible retornar a la zona de claves hasta tanto no se realice una operación.

Luego de ordenar las operaciones ACTUALIZAR, AGREGAR, REMOVER o IGNORAR, el formulario de datos se blanqueará, y el cursor pasará al primer campo clave.

El formulario de la figura 3.8., se utiliza con el esquema `biblio` (el mismo que se utiliza como ejemplo en el capítulo de Bases de Datos), donde la tabla `autores` se indexa por la columna `codigo`. En otras palabras, es fácil encontrar un formulario por el valor del campo codigo. Entonces se modificará el formulario, agregando la zona delimitadora de clave, indicando tal efecto mediante una barra vertical de Unix), como se muestra a continuación:

```
FORMULARIO DE LIBROS
--------------------
Código de Libro : _,___. |
Título del Libro :_____________________________
Autor : ____._____________________________
Número de Edición : __.
Fecha de Edición : __/__/__
%form
use biblio;
window label "Libros", border standard;
%fields
codigo : libros.;
titulo : libros.;
autor : libros., in au<hy>
tores : nombre;
nombre : autores., skip;
edicion: libros.;
fecha : libros.,<= today;
```

Figura 3.8 - Figura 3.9. - Formulario de Libros

En el campo `autor` se ha agregado la cláusula "in", indicando en este caso que el código del autor que se ingrese se validará contra lo grabado en la tabla autores, no permitiendo otro valor distinto a los que figuren en esa tabla.

En el diseño del formulario aparece el campo `nombre`. Es un campo de despliegue (lo especifica la sentencia skip). Además, está relacionado con el campo nombre de la tabla autores, por lo que se desplegará el valor del campo sobre el formulario, cuando se ingrese el código del autor.

El campo `fecha` tiene un atributo que indica que se lo debe completar con una fecha menor o igual que la del sistema.

Utilitarios

Existen diversas maneras de utilizar un formulario, una vez procesado con *fgen*. En líneas generales, se puede decir que hay dos formas:

• Mediante un utilitario de IDEAFIX.

• Escribiendo un programa en "C", usando rutinas para manejo de formularios (FM), provistas por la biblioteca de IDEAFIX.

En esta sección se describirán los utilitarios de IDEAFIX que se relacionan con los formularios. En la sección siguiente se describirá en líneas generales la interfaz de programación con el lenguaje "C".

TESTFORM

Este utilitario permite la prueba de un diseño de formulario, es decir, se obtiene un prototipo del comportamiento de la pantalla, de acuerdo a lo especificado en la definición.

No interactúa con la base de datos en el momento de la lectura o la grabación, es decir que no permite consultar datos existentes, ni como es lógico, actualizar las bases de datos asociadas al formulario. Sin embargo, se aplican todas las validaciones indicadas en la definición del formulario, aún aquellas que impliquen consultar una tabla de la base de datos (el atributo *in tabla*). La forma de utilizarlo a través de la línea de comandos (o shell) es:

```
$ testform nombre_formulario
```

GENFM

El utilitario *genfm* toma una tabla de un esquema existente, y genera una especificación de formularios en FDL (Form Definition Language) para realizar operaciones sobre dicha tabla. La forma de utilizarlo es:

```
genfm esquema.tabla
```

Se obtendrá como resultado un archivo llamado "tabla.fm" con una definición de un formulario para operar sobre dicha tabla. El nombre del archivo resultante se puede cambiar mediante la opción "**-o** nombre" para indicar en qué archivo se dejará la salida.

La zona de claves del formulario se construirá en base a la clave primaria de la tabla. Es posible indicar cualquier otro índice de la misma mediante la opción "**-i** indice". Por ejemplo:

```
genfm -i indice esquema.tabla
```

DOFORM

doform permite utilizar un formulario para consultar y modificar información de la base de datos. Para poder trabajar correctamente, el formulario que se utilizará debe cumplir ciertas condiciones:

1. Debe poseer una zona de claves. Esta zona de claves debe contener campos que coincidan con algún índice unívoco de la tabla, o bien caiga dentro de alguno de los casos de multirrenglón que se detalla más adelante.

2. Si tiene campos múltiples deben cumplir con alguno de los casos que se especifican más adelante.

3. Todos los campos de la zona de datos del formulario deben estar asociados con campos de base de datos, de la tabla que se usó en la zona de claves, o de tablas que estén relacionadas mediante atributos "in table". Si así no se hiciere, la información cargada sobre los campos no asociados se perderá. Cuando hay campos no asociados con la base de datos, el utilitario pone un mensaje de advertencia.

4. El formulario sólo debe involucrar campos de a lo sumo dos tablas. Así mismo, cualquier número de tablas puede ser usado para procedimientos de validación.

La manera de utilizar doform es la siguiente:

```
$ doform nombre_formulario
```

Doform maneja la estrategia de bloqueo de registros de la siguiente manera:

Al leer el registro lo bloquea hasta que se indique una operación, cumplida la cual el registro será liberado.

Existe una opción en la línea de comandos: **-l**. Si está presente, se bloqueará toda la tabla al iniciar el proceso y se la liberará recién al terminar el mismo. Si no puede bloquearse debido a que otro proceso tiene algún registro ya bloqueado, *doform* abortará con un mensaje de error.

Modo de lectura de los registros

Si el formulario permite modificar o borrar registros existentes doform realizará la lectura en modo TEST y LOCK (ver *GetRecord* (DB), *Referencia del Programador*).

Un formulario de consulta es aquel que ignora las operaciones *delete* (borrado) y *update* (modificación), en estos formularios *doform* hará la lectura sin bloquear.

Formularios con campos Multirrenglón

El utilitario doform permite manejar formularios que tengan campos multirrenglón, siempre y cuando caigan dentro de alguno de los siguientes casos:

1. Cuando los campos (columnas) del multirrenglón están relacionados con un vector de la base de datos. En este caso todas las columnas (excepto aquellas que sean SKIP o DISPLAY ONLY) deben ser vectores y de igual dimensión.

```
......
Vec1 Desc(campo skip) Vec2
[ ___. ____________________ _____ ]
```

2. Cuando la clave de la pantalla no coincide completamente con un índice de la tabla y los primeros campos del multirrenglón completan la clave. Este es el caso de la Edición de Form de *n* registros por vez.

```
ABM DE ASIENTOS
fecha : __/__/__
tipo : __.
nro : _,___.|
nreng descrip Cuenta Importe
[ ___. ____________________ __. _,___,___.__ ]
```

y el archivo tiene una clave de la forma:

```
... (fecha, tipo, nro, nreng);
```

3. Edición usando dos tablas con una relación de 1 a *n*, es decir que para un registro de la tabla principal hay n de la tabla asociada. En este caso se debe usar la opción **-j** para indicar qué campos son comunes a ambas tablas.

```
nrosub : ___. |
nombre : ____________________________
Tipos de Comprobante
tcom descrip
[ __. ____________________]
```

donde las tablas son:

```
table SUBCTA {
nrosub num(3),
nombre char(30)
};
table TCOMP {
nro2 num(3) in SUBCTA:nombre,
tcom num(2),
descrip char(20)
}
primary key(nro2, tcom);
```

En este caso la llamada al doform debe ser de la forma:

```
doform -jnro2=nrosub pantalla
```

Los campos que utilizará el utilitario doform para acceder a la segunda tabla, deben formar la clave primaria de la misma.

Casos no contemplados:

1.No se permiten llamadas a subform dentro de campos múltiples.

2.En un subform los múltiples deben corresponderse con vectores.

3.No soporta la actualización por Edición de Formularios de más de 2 tablas.

GENCF

Este utilitario permite generar automáticamente un programa en CFIX a partir de un formulario compilado, es decir que utilizará el archivo con extensión "fmo", obtenido con el utilitario *fgen*. Desde la línea de comandos se invoca mediante:

```
gencf formulario
```

Como resultado del utilitario se obtendrá un programa llamado `formulario.c`. Es posible dar un nombre distinto al programa objeto resultante mediante la opción "**-o** nombre".

Se pueden generar dos tipos de programas dependiendo del tipo de formulario que se vaya a utilizar:

• Se genera cuando el formulario no tiene zona de claves.

• Se produce cuando el formulario tiene una zona de claves.

Listadores: Estos programas consisten solamente en la toma de datos del fomulario, y la captura de un comando (tecla de función) dada por el usuario. Si ésta es distinta de <PROCESO> simplemente terminará. En caso contrario se ejecutará el código que se agregue en el lugar del programa generado indicado como proceso central.

ABM: Cuando el formulario posee zona de claves se intenta generar un programa de altas bajas y modificaciones. Es factible generarlos para formularios que trabajan con una o dos tablas, contemplando multirenglones y subformularios. En el programa generado se incluirán rutinas de chequeo pre-campo y post-campo sólo si se pide explícitamente a través de las opciones:

• Incluye rutina after field.

• Incluye rutina before field.

Por ejemplo:

```
gencf -b formulario
```

Formularios con Multirenglones

El utilitario *gencf* contempla los mismos casos de multirrenglón que el *doform*.

EXECFORM

Este comando permite utilizar un formulario para pedir al usuario en una forma amigable parámetros que se utilizarán para invocar otro proceso. Los valores que se ingresen en los campos, serán pasados como parámetros al invocar el proceso.

Por ejemplo:

```
$ execform prog donde prog es un formulario
```

definido de la siguiente manera:

```
Parametro 1 : _______.
Parametro 2 : _______________
```

Cuando se digite la tecla de función será invocado un programa llamado prog.sh con los valores cargados en los campos del formulario como parámetros. Por ejemplo si el usuario ingresó 1000 en el primer campo, y dejó en blanco el segundo se ejecutará:

```
prog.sh "1000" ""
```

Como se mostró, *execform* asume que el programa a ejecutar tiene el mismo nombre que el formulario, pero con extensión ".sh".

Es posible indicar cualquier otro nombre de programa, poniéndolo a continuación del nombre del formulario.

Por ejemplo:

```
$ execform prog prog1.exe
```

Existe otro parámetro importante para este utilitario que es el modo en que se ejecutará el programa. Si no se especifica otra cosa, se asume que es un programa a ejecutar a través del shell, pero existen otros modos que se indican como opciones en la línea de comandos, que son los siguientes:

• **-r** vuelve al formulario después de ejecutar el programa.

• **-w** El programa se ejecutará como un proceso usuario del Window Manager. Es el valor tomado en caso de omisión.

• **-p** [FILAS] [xCOLS] El programa será ejecutado a través del shell, pero capturando su salida en una ventana. Las dimensiones de la misma pueden definirse optativamente a continuación de la letra p, donde FILAS y COLS indican la cantidad de filas y columnas de dicha ventana. Por defecto se asumirá el espacio disponible en pantalla en cada caso.

• **-d** Este modo se denomina *debugging* ya que no se invoca al programa, sino que se muestran los argumentos que se le pasarían al mismo en la parte inferior de la pantalla cuando el

usuario digita la tecla .

• **-s** Se ejecuta el programa como si fuese uno del sistema operativo.

Interfaz C

Compilando Programas

IDEAFIX provee la facilidad de escribir programas en lenguaje "C" que utilicen todas las posibilidades de manejo de formularios vistas anteriormente. Para este fin se dispone de una poderosa biblioteca de funciones para usar en los programas.

Los programas en C pueden ser generados en forma automática mediante el utilitario gencf a partir de una definición de formulario. La manera de compilar un programa escrito en "C" con IDEAFIX es:

```
$ cfix form1.c
```

Este comando imprimirá en la pantalla el siguiente mensaje:

```
$ cc -Iinclude form1.c -o form1 -lidea
```

donde include es el directorio donde se encuentran los archivos de encabezamiento de IDEAFIX. Este directorio es localizado por medio de la variable de ambiente IDEAFIX. Con esta instrucción se compilará un programa llamado `form1.c` y se lo dejará como archivo ejecutable con el nombre form1.

Si se trabaja con sistemas que tienen gran cantidaad de funciones, es imprescindible utilizar algún programa que pueda, ante la definición de dependencias entre programas objetos y fuentes, realizar automáticamente las tareas de compilación y linkedición. make, de UNIX, es una herramienta que cumple con ese cometido, y por lo tanto se recomienda su uso.

La manera de referenciarse a los diversos campos de formulario en un programa en "C" es a través del uso de constantes simbólicas o constantes manifiestas. Para poder generar estas constantes (que se expresan mediante sentencias `#define` del preprocesador de "C") se debe incluir en la sección `%form` de la definición del formulario, la siguiente sentencia:

```
language C;
```

Otra forma de obtener estas constantes, es desde la línea de comandos, ejecutando:

```
$ fgen -h form1.fm
```

En este caso, la opción **-h** indica que además de generar el "formulario compilado", se genere un archivo de encabezados con todas las definiciones comentadas anteriormente. El archivo se genera con extensión *.fmh*; por lo tanto en el ejemplo anterior se creará el archivo form1.fmh. El mismo debe incluirse en el programa fuente como:

```
#include <ideafix.h>
#include "form1.fmh"
```

La Función *DoForm*

La función DoForm es el núcleo de toda aplicación que se implemente en lenguaje "C" utilizando formularios. Permite el ingreso de datos sobre los campos definidos en un formulario realizando validaciones, las cuales pueden estar especificadas:

De acuerdo al tipo de campo. Por ejemplo, en un campo de tipo *fecha*, sólo se permitirá ingresar datos que representen fechas válidas, es decir que no se aceptará el 29 de febrero de un año no bisiesto.

De acuerdo a lo indicado en el formulario. Por ejemplo, si un campo se definió como not null, no se podrá pasar ese campo si no se le ingresa un dato.

Mediante programación, a través de funciones pre y post campo.

DoForm se mueve a través de todos los campos de un formulario hasta que el usuario digite una tecla de función determinada, en cuyo caso retorna un valor que representa el estado del formulario. Las operaciones y los valores retornados son:

Operación	Valor Retornado
ADD	FM_ADD
UPDATE	FM_UPDATE
REMOVE	FM_DELETE
IGNORE	FM_IGNORE
END	FM_EXIT

Figura 3.9 - La función *DoForm*

Cuando el formulario tiene "zona de claves" se agregan otros estados adicionales. Estos estados son:

FM_READ: Se devuelve cuando se ha ingresado el último campo de la zona de claves o campo control.

FM_READ_NEXT: En caso que se haya digitado la tecla página siguiente en la zona de claves o campo control.

FM_READ_PREV: En caso que se haya digitado la tecla página previa en la zona de claves o campo control.

Condiciones Pre-Campo y Post-Campo

Anteriormente se mencionó que se realizan validaciones mediante funciones pre y post campo

(se las denominará before y after field respectivamente). DoForm llama a estas funciones antes (before) y después (after) de realizar la entrada de datos.

Desde un punto de vista técnico, la manera de indicar a qué función se debe invocar es pasando como argumentos de la misma un puntero a la función que se quiera ejecutar, o NULLFP (NULL File Pointer) cuando no se desea realizar dicha validación.

El utilitario *gencf* incluye rutinas after y before field si se le indican las opciones -a y -b respectivamente. El proceso completo de ingreso y validación de campos es entonces el siguiente:

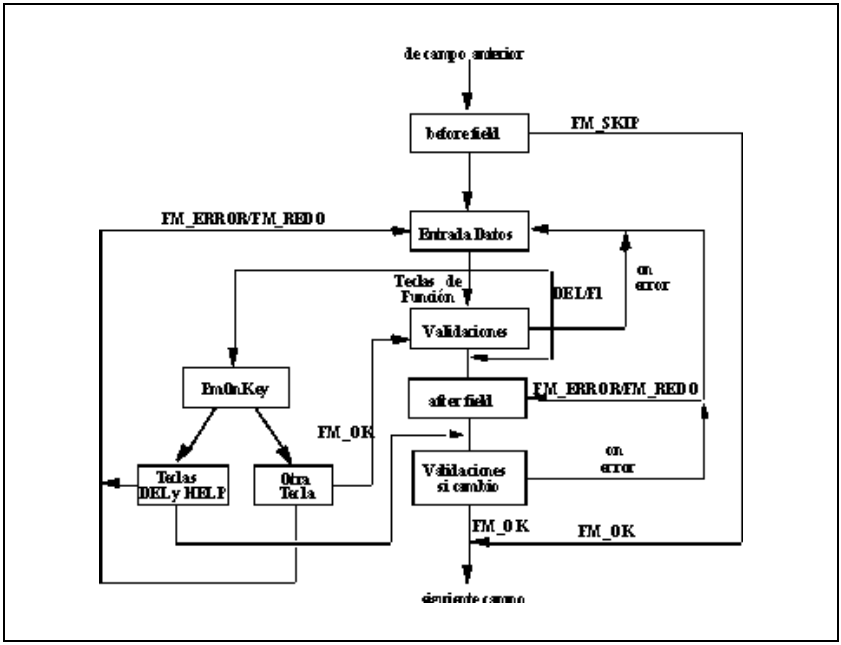


Figura 3.10 - Validaciones Pre- y Post- campo.

Un ejemplo muy simple de uso de before field sería el que se muestra a continuación. Lo que se desea hacer es saltear el campo DESCRIP pues es un campo de despliegue solamente. En rigor no es necesario realizar esta operación en un programa CFIX pues se podría indicar directamente como atributo del campo de formulario: se muestra tan sólo a modo de ejemplo.

```
private fm_status before(form fm, fmfield campo, it fila)
{
}
switch (campo) {
...
case DESCRIP:
```

```
return FM_SKIP;
break;
}
return FM_OK;
}
```

En este caso el before del campo DESCRIP, indica que se debe ejecutar un "skip".

Se debe tener en cuenta que las teclas de función: <REMOVER>, <IGNORAR> y <FIN> no pasan por el *after*; es decir, que las validaciones que sean necesarias realizar para estas teclas de función deberán ser hechas en el cuerpo principal del programa, pudiendo eventualmente informar situaciones de error con la rutina *FmSetError*.

has [after|before] when *expr*

Estas cláusulas (las que deberán ser especificadas en la sección %field del formulario) definirán si un campo dado tiene o no tiene la condición after o before dependiendo del valor de verdad de la expresión expr. Por ejemplo:

```
a: has after when a != null;
```

ejecutará el after del campo a cuando a sea diferente de NULL.

Condiciones Pre- y Post- Campo en campos múltiples y agrupados

Los campos múltiples y los agrupados también pueden utilizar las funciones *before* y *after field*. Si el "nombre de campo" referenciado es un campo agrupado (obtenido del .fmh correspondiente), se realizará una validación antes de entrar o de después de salir de él, según corresponda. Esto es de suma utilidad en los casos en los cuales se necesita una validación cruzada entre varios campos (habitualmente dos, como sería el caso de valores "desde" y "hasta", incluyendo campos tipo fecha, en los que debe verificarse que una de ellas sea anterior o posterior a otra.

A continuación se presenta un ejemplo de formulario para la carga de un rango de códigos de autor, con el objeto de listar los libros asociados a ellos.

```
LISTADO DE LIBROS POR CODIGO DE AUTOR
Autor Desde : { ____. _____________________________
Autor Hasta : ____. _____________________________ }
%form
use biblio;
window label "Listado de "Libros", border standard;
messages MAYOR:"El código desde debe ser menor que el hasta";
%fields
AGRUP:
desde : libros.autor;
desc0 : autores.nombre, skip;
hasta : libros.autor;
desc1 : autores.nombre, skip;
```

Figura 3.11 - Figura 3.12 - Formulario de listado de libros

El código CFIX para verificar que desde sea menor que hasta es:

```
private fm_status after(form fm, fmfield nro_campo) ;
{
switch (nro_campo) {
case AGRUP:
if (FmIFld(fm, DESDE) > FmIFld(fm, HASTA))
return FmErrMsg(fm, M_MAYOR);
break;
}
return FM_OK; }
```

En este caso, AGRUP es el nombre simbólico del campo agrupado definido en el formulario, que permite verificar que el campo HASTA sea mayor que el DESDE. Los valores ingresados en estos campos se obtienen mediante la función FmIFld de la biblioteca. Si no se cumple la condición, se mostrará en pantalla el mensaje MAYOR mediante la función de biblioteca FmErrMsg, definido en la opción messages de la zona `%form`.

Los campos múltiples también pueden ser objeto de validaciones, de manera similar a la explicada para los agrupados.

Suformularios Manuales

Ya se ha indicado la posibilidad de asociar, a un campo de un formulario, subformularios automáticos o manuales. La sección *Subformularios* de este capítulo explica el caso de los primeros. Para asociar subformularios manuales se debe indicar el atributo manual subform según la sintaxis:

```
campo: manual subform ("formulario");
```

La diferencia entre un subformulario automático y uno manual es que éste último debe ser desplegado por el programa en el momento apropiado. Los subformularios se activan mediante una función de la biblioteca de interfaz (ver *DoSubform(FM), en Referencia del Programador*). El programa CFIX debe incluir los archivos de encabezado que correspondan.

La Biblioteca de Formularios

Para una descripción completa de la biblioteca de formularios, consultar la parte II de este manual, capítulo Interfaz con Formularios (FM).

Capacidades Máximas

Número de Formularios abiertos	8

simultáneamente Número de Esquemas abiertos simultáneamente	4
Campos en un Formulario	9999
Caracteres en un campo alfanumérico	65535
Dígitos en un campo numérico	15
Dígitos significativos en un campo numérico	15
Rango de valores en un campo fecha	16/04/1894 to 16/09/2073
Rango de valores en un campo hora	00:00:00 to 23:59:58[a]
Dimensión de un vector	65535
Valores en un campo in	65535
Nombres de Campo	Unlimited
Cantidad de Mensajes	64
Cantidad de caracteres de un identificador	16
Campos clave en un "in table"	32

[a] El último horario para un día es 23:59:58, esto se debe a que IDEAFIX tiene una precisión de 2 segundos para los datos del tipo Time.

Capítulo 17

Reportes

En este capítulo se describen las capacidades de los reportes de IDEA-FIX, el lenguaje utilizado para su definición (RDL) y los utilitarios que permiten manejarlos.

Introducción

Los informes impresos (listados) son denominados en IDEAFIX "reportes", terminología tomada de la palabra inglesa *reports*.

IDEAFIX permite diseñar reportes en forma muy simple, a fin de poder indicar aspectos tales como:

• Características de los campos;

• Relación con Bases de Datos;

• Funciones como sumatorias y promedios;

• Cortes de control, y otros.

Los reportes pueden diseñarse con cualquier editor, pero es altamente recomendable el uso de uno de los provistos por IDEAFIX, ya que poseen ciertas características que facilitan mucho esta tarea; siendo algunas de ellas imprescindibles, como por ejemplo la posibilidad de escribir en español (vocales acentuadas, eñes, etc.) y carácteres gráficos.

Para definir un reporte se utiliza el RDL, o *Report Definition Language* (Lenguaje de Definición de Reportes). El RDL permite dibujar la imagen del listado tal como se desea su impresión, en un modo llamado WYSIWYG (*What You See Is What You Get*: lo que ves es lo que obtienes), es decir que al ejecutar el programa, el listado se genera con el formato que se le ha diseñado.

Cuando el archivo de definición del reporte está completo, se lo compila con *rgen*, el utilitario de IDEAFIX para generación de reportes.

Generando Reportes

La especificación del reporte está contenida en un archivo con extensión ".rp", conocido como archivo fuente RDL. Este se procesa con el utilitario *rgen* de IDEAFIX, para generar un archivo con extensión ".rpo".

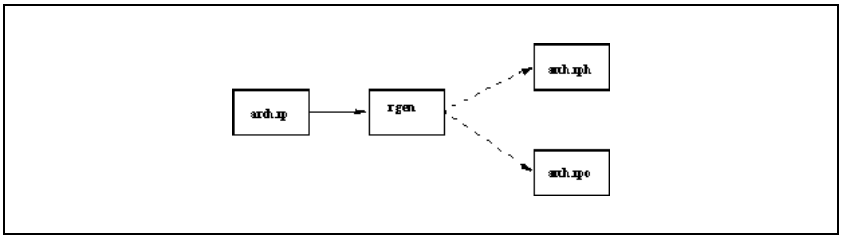


Figura 4.1 - Generación de Reportes

La figura anterior muestra este proceso de generación, incluyendo en forma optativa crear otro con extensión ".rph", este archivo contiene el *reporte compilado*. Este archivo es utilizado tanto por el utilitario *doreport* como por un programa de aplicación escrito en "C", en el momento de su ejecución.

El archivo de cabecera ".rph" debe incluirse en los programas de aplicación en "C" que utilicen el reporte. Contiene valores de constantes simbólicas que representan los distintos campos, y otras constantes auxiliares necesarias para la compilación.

El Lenguaje RDL

Un archivo de especificación RDL se divide en tres secciones.

• La primera contiene la imagen del listado. En esta sección se dibuja el reporte, como se desea que éste aparezca en la salida. La imagen se compone de campos del listado que se llenan con la información provista por el usuario, y cadenas de carácteres constantes. Esta sección comienza al principio del archivo, y en ella se describen las distintas zonas que componen el listado.

• La segunda sección se inicia con la palabra clave %report. Se compone de sentencias que terminan en punto y coma, dando información acerca de los requerimientos generales del listado, tales como longitud del papel, esquemas de base de datos a utilizar, márgenes, destino del reporte, etc.

• La tercera sección comienza con la palabra clave %fields y define los nombres de los campos del listado y sus atributos.

Imagen del Reporte

La imagen del reporte se divide en zonas del reporte que son líneas de detalle. Una zona del listado tiene un nombre, y está formada por uno o más campos, o inclusive por ninguno. En el diseño es posible imponer condiciones para definir cuándo debe imprimirse una zona.

La definición de una zona se inicia con un "%" seguida por su nombre, los nombres de los campos de la zona, y opcionalmente por la condición para imprimirla. La figura muestra el esqueleto de una especificación de reporte.

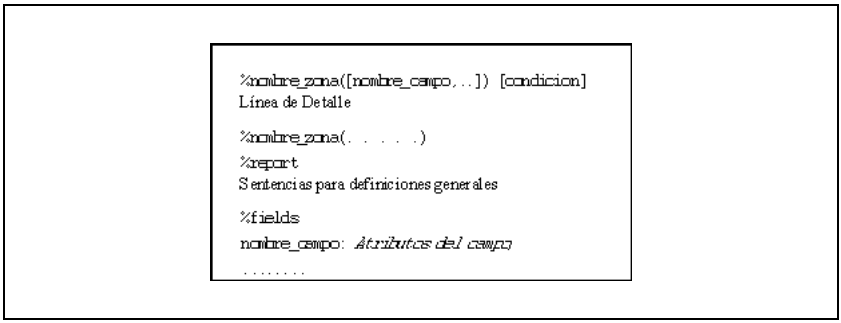


Figura 4.2 - Estructura de un Reporte

La salida del reporte se compone de páginas, cada una de las cuales se divide en zonas, tal como se las dibuja en el archivo de definición (.rp). Sobre el dibujo de la zona se especifican los campos que serán reemplazados por información.

La zona tiene un nombre, y se indican uno o más parámetros que son los valores a reemplazar en los campos de la zona. Opcionalmente se puede especificar una condición que controla cuándo se imprimirá la zona.

La sintaxis completa para definir una zona es:

```
%zoneX(expr [: expid], ...) ...
```

donde:

nombre es el nombre de la zona.

expr es una expresión completa que puede contener llamados a función y campos combinados en expresiones aritméticas.

Definiendo Expresiones

Las expresiones definen los valores que se imprimirán en los campos correspondientes de la

zona.

También, una expresión puede ser identificada con un nombre, para ser referenciada directamente por su nombre luego en el reporte.

Por ejemplo:

```
%zoneA (n1, n2, (n1+n2)*200 : b)
%zoneB ((runsum(b)+n1+n2)/800)
```

El id de la expresión (b, en este caso), también puede ser usado para tener dos entradas distintas en la sección %fields, por ejemplo:

```
%zoneA(a : c1, a : c2)
%fields
c1: atributos para c1;
c2: atributos para c2;
```

Los valores que conforman una expresión puede ser un campo de un reporte, un función, una variable, una constante, o cualquier combinación de ellos. A continuación se presenta una explicación detallada de cada valor:

Funciones:

sum(param): Se imprimirá la suma de los valores que haya tomado *param*.

avg(param): Se imprimirá el promedio de los valores que haya tomado *param*.

count(param): Se imprimirá la cantidad de los valores que haya tomado *param*.

min(param): Se imprimirá el mínimo de los valores que haya tomado *param*.

max(param): Se imprimirá el máximo de los valores que haya tomado *param*.

runsum(param): Se imprimirá la suma de los valores que haya tomado el parámetro indicado, a lo largo de todo el reporte.

runavg(param): Se imprimirá el promedio de los valores que haya tomado el parámetro indicado, a lo largo de todo el reporte.

runcount(param): Se imprimirá la cantidad de valores que haya tomado el parámetro indicado, a lo largo de todo el reporte.

runmin(param): Se imprimirá el mínimo de los valores que haya tomado el parámetro indicado, a lo largo de todo el reporte.

runmax(param): Se imprimirá el máximo de los valores que haya tomado el parámetro indicado, a lo largo de todo el reporte.

Las funciones *runsum*, *runavg*, *runcount*, *runmin* y *runmax* , se aplican cuando se desea evaluar los valores que obtuvo un determinado parámetro hasta el momento en que se la invoca.

Variables:

• *today~*: Contiene la fecha corriente.

• *hour*: Contiene la hora corriente.

• *pageno*: Contiene el número de página.

• *lineno*: Contiene el número de línea.

• *module*: Contiene el nombre del módulo.

Constantes:

• Puede ser cualquier constante numérica: 1, 2, . . ., 50, . . .738, . . .

Para una mayor comprensión, tomaremos un nuevo ejemplo, que intenta abarcar la mayor cantidad de variantes posibles. Para ello utilizaremos las siguientes tablas de un esquema llamado personal:

```
table emp descr "Legajos del personal" {
empno num(4) not null
primary key,
nombre char(30)not null,
cargo num(1)
in cargos:descrip,
jefe num(4) in emp:nombre,
fingr date <=today,
sueldo num(12,2),
depto num(2) in depto:nombre
};
table depto descr "Departamentos" {
depno num(2) not null
primary key,
nombre char(20) not null
};
table cargos descr "Cargos de la Empresa" {
cargo num(2) not null
primary key,
descrip char(20) not null
};
```

Como ejemplo de utilización de funciones, remitimos al lector a la figura siguiente, en la cual se define un reporte utilizando las tablas del esquema que acabamos de definir. Pero antes, a los fines de un mejor aprovechamiento del espacio, dejamos ilustrada una consulta por *iql*, consistente en el "query":

```
use personal;
select ~depto, nombre, sueldo, fingr
from ~emp, depto
where emp.depto = depto.depno
output to report ~personal;
```

Aquí se han representado en bastardilla los nombres y operandos variables que debe ingresar el programador.

```
%tit(pageno, today) before page
Página : __. Fecha: __/__/__
Listado de Empleados
====================
Depto. Empleado Sueldo Fecha Ingreso
============================================================
%linea(depto, nombre, sueldo, fingr)
__. _____________________ ___________. __/__/__
%suma(sum(sueldo)) after depto
Cálculos por departamento: ------------------------
Suma de sueldos por departamento : ___________.
%prom(avg(sueldo)) after depto
Promedio de sueldo por departamento : ___________.
%mini( min(sueldo) ) after depto
Mnimo sueldo del departamento : ___________.
%maxi(max(sueldo)) after depto
Máximo sueldo del departamento : ___________.
%totsuma( sum(sueldo) ) after report
Cálculos Totales: -------------------
Suma de los sueldos de la empresa : ___________.
%totprom( avg(sueldo) ) after report
Promedio de sueldo de la empresa : ___________.
%totmini( min(sueldo) ) after report
Sueldo mínimo de la empresa : ___________.
%totmaxi( max(sueldo) ) after report
Sueldo máximo de la empresa : ___________.
%report
use personal;
%fields
depto: ; nombre: ;
sueldo: ;
fingr: ;
```

Figura 4.3 - Reporte con Funciones

```
Página: 1 Fecha : 03/08/89
Listado de Empleados
====================
Depto. Empleado Sueldo Fecha Ingreso
============================================================
1 Juan Gatica 950 14/01/82
1 Hugo Riveros 1400 23/04/80
Cálculos por departamento:
-------------------------
Suma de sueldos por departamento : 2350
Promedio de sueldo por departamento : 1175
Mínimo sueldo del departamento : 950
Máximo sueldo del departamento : 1400
2 Charlie Parker 2600 23/06/87
2 Albert Colby 1150 07/08/92
Cálculos por departamento:
-------------------------
Suma de sueldos por departamento : 3750
Promedio de sueldo por departamento : 1875
Mínimo sueldo del departamento : 1150
Máximo sueldo del departamento : 2600
3 John Holmes 1000 27/08/84
Página: 2 Fecha : 03/08/89
Cálculos Totales:
-------------------
```

```
Suma de los sueldos de la empresa : 7100
Promedio de sueldo de la empresa : 1420
Sueldo mínimo de la empresa : 950
Sueldo máximo de la empresa : 2600
```

Figura 4.4 - Listado cn Funciones

Nótese que se han usado expresiones simples (aquellas que sólo una referencia), para ver su uso a continuación se presentan algunos ejemplos.

```
%zoneX ((runsum(b)+n1+n2)/8000)
```

Aquí, b, n1 y n2 pueden ser campos pertenecientes a la misma zona o a zonas previamente definidas. Nótese que la definición de campos es un caso particular de esta sintaxis.

```
%zoneX(n)
%zoneY(a, b, a+b, sum(n), avg(n)*100)
```

En este caso, n es una expresión que retorna el valor de un campo n, a es una expresión que retorna el valor del campo a, y lo mismo pasa con b. La expresión a+b retorna el valor de a y de b, la expresión sum(n) retorna la sumatoria del campo n y la expresión avg(n)*100 retorna el promedio del campo n por 100.

La expresión a+b permite realizar operaciones "horizontales" (esto es poner en una columna la suma de otras dos) dentro de reportes.

Opciones de Impresión

Las zonas de los reportes pueden ser impresas según condiciones que se indican junto con la definición de las mismas.

Cortes de Control

```
{before, after} {report, page, campo,
( [report] [,page] [,campo] [,...] )}
```

Estas opciones controlan aspectos de la impresión de una zona. Las condiciones *before* (antes) y *after* (después) implementan cortes de control, que pueden ser efectuados según:

• *report*: El comienzo o fin del reporte. Esta opción puede usarse en conjunción con la page, según se verá más adelante.

• *page*: El comienzo o fin de la página. Es de señalar que la opción before page no imprime la zona en la primera página, en tanto que after page no lo hace en la última.

• *campo*: La impresión se realiza antes o después de que cambie el valor del campo mencionado.

• *condiciones*: La zona se imprime cuando se cumpla un subconjunto de las condiciones de corte. Es decir que si el conjunto de condiciones está formado por las opciones page, report y

campo, la zona se imprimirá cuando se cumpla cualquiera de las condiciones individuales o cuando se cumplan dos o más simultáneamente. El conjunto de opciones puede estar formado por varios campos, en cuyo caso la zona se imprimirá antes o después (*before* o *after*) del cambio de valores de uno o más de ellos.

Un ejemplo de la aplicación de la combinación de opciones de corte de control, sería el caso en que el programador necesitara imprimir una zona al comienzo de todas las páginas, la primera inclusive, y otra al final de todas ellas, incluyendo la última. Si colocara simplemente las opciones *before report* y *after page* no conseguiría esto, ya que en el primer caso no se imprimiría la zona en la primera página y en el otro no se imprimiría en la última. Combinando estas opciones con *before page* y *after report* de la manera apropiada, es posible obtener el resultado deseado. Por ejemplo:

```
%zonaM( param [,...] ) before (page, report)
%zonaN( param [,...] ) after (page, report)
```

Por otro lado, u grupo de especificaciones como:

```
%zoneB1( param1 ) before page
%zoneB2( param1, param2 ) before report
%zoneB1( param3 ) after page
%zoneB2( param3, param4 ) after report
```

Los siguientes son algunos ejemplos de aplicación de las opciones de corte de control:

```
%titulo( today, hour, pageno ) before page
Fecha : __/__/__
Hora : __:__:__
Página : ___.
%total ( sum(sueldo) ) after (depto, seccion, report)
TOTAL : _,___,___.__
```

En el primer caso la zona se imprimirá antes de cada página, salvo la primera, y los campos indicados tomarán los valores de las variables pasadas como parámetros. La segunda zona del ejemplo se imprimirá cuando alguno de los parámetros *depto* y/o *seccion* cambien de valor o bien termine el reporte. Si se diera el caso de que se cumplan dos o tres de las condiciones simultáneamente la zona sólo se imprimirá una vez.

Impresión Condicionada - *if*

```
if <param> <condicion> <param>
```

Esta opción permite controlar una determinada zona, de forma tal que si se cumple <*condicion*>, se imprime la zona de impresión asociada.

```
%linea() if campo1 = campo2
```

es un caso simple. Usando expresiones sería de este tipo:

```
%zoneY(a, b) if a + 10 > b + sum(n)
```

de esta forma la opción if se torna más potente.

Agrupación

```
group with zona
```

La opción group permite agrupar zonas para que sean impresas en la misma página. Por ejemplo si una zona de totales desea ser impresa junto con otra de subtotales, para que dado el caso no quede la zona de totales impresa en una hoja separada, debería especificarse:

```
%subtot(sum(sueldo)) after (depto, seccion, report)
Total : _,___,___.__
%total(sum(sueldo), max(sueldo), min(sueldo), avg(sueldo))
after report, group with subtot
Totales : _,___,___.__
Sueldo Máximo : _,___,___.__
Sueldo Mínimo : _,___,___.__
Sueldo Promedio : _,___,___.__
```

Avance de página

```
eject {before, after}
```

La opción *eject* provoca un salto de página antes (*eject before*) o después (*eject after*) de imprimirse la zona. Si no se especifica el momento, se obtendrá el mismo efecto que un *eject after*. Esto equivale a decir que ÔafterÕ es el valor por defecto.

Zonas no Impresas

```
no print
```

En ocasiones no se desea que una determinada zona sea impresa. Ello ocurre cuando se quiere imprimir el nombre del mes de una fecha pasada como parámetro: es necesario ubicar la fecha en alguna zona para que quede definido el tipo de dicho parámetro. En tal caso la fecha puede estar definida en una zona no imprimible, de forma tal de no entorpecer el diseño del reporte. Veamos un ejemplo:

```
%fecaux(fecha) no print __/__/__
%titulo( day(fecha), monthname(fecha), year(fecha) ) before
report
__. de __________ de _,___.
. . . . .
```

Impresión en una posición fija

```
at line NN
```

La opción at *line NN* permite imprimir una zona siempre en la misma posición de la hoja.

```
piepag(param1, ... ) at line 53
```

En este ejemplo la zona de "pie de página" piepag siempre se imprimirá a partir del renglón 53 de cada página del reporte. Es importante resaltar que el usuario es responsable de verificar que las zonas previas no sobrepasarán el número de línea especificado: dado que el sistema no interrumpirá la impresión de una zona que involucre varias líneas, en este caso el pie de página aparecerá en una línea inferior (la primera disponible luego de que la zona previa este completada).

Las expresiones pueden ser usadas incluyendo esta opción, como en

```
%zoneZ(c, d) at line $PLINE + 1
```

Así mismo, las expresiones usadas en esta cláusula no podrán contener referencias a expresiones definidas en una zona.

Definición de Campos

Los campos de los listados se definen siempre dentro de las zonas, y se los especifica igual que en los formularios. La única diferencia es que a los campos se les da un nombre dentro de la especificación de la zona, y en la sección *%fields*, se especifican las características de los mismos.

La expresión "\$varname" se reemplaza por el valor de la variable de ambiente *varname*. El texto que aparecerá en la salida es el contenido de dicha variable de ambiente cuando se imprime el reporte.

La Sección *%report*

Esta sección puede contener las siguientes sentencias:

use esquema [, esquema];

especifica el o los esquemas de la Base de Datos a utilizar. Por su parte las especificaciones

```
flength = <valor>; // Largo de página
topmarg = <valor>; // Margen superior
botmarg = <valor>; // Margen inferior
leftmarg = <valor>; // Margen izquierdo
[no] formfeed;
```

indican los valores para la longitud del formulario de papel (flength) y para los márgenes superior, inferior e izquierdo de la página (topmarg, botmarg y leftmarg respectivamente). La longitud de la línea se supone de 80 caracteres. Si se indica la opción formfeed, se cumplirán los saltos de página al imprimir el reporte; en caso contrario el listado saldrá en forma continua.

<valor> puede ser un número, o un valor que se toma del ambiente con una indicación `$varname`. En este caso, el valor es aquel contenido en la variable de ambiente cuando se

imprime el listado. Los valores por defecto son:

flength: 66

width: 80

leftmarg: 0

topmarg: 2

botmarg: 2

```
output to {
archivo
| printer
| pipe comando
| terminal
| stdout
}
```

Esta sentencia permite la especificación del destino de la salida, que puede ser un archivo, la impresora, otro comando, la terminal o la salida normal (standard output).

archivo Esta opción permite enviar los resultados del reporte a un archivo

printer Esta opción envía la salida del reporte al destino especificado en la variable de ambiente printer (Consultar *Manual del Usuario*, Apéndice A). En el sistema operativo MS-DOS, la variable de ambiente PRINTER podría estar definida de la siguiente forma:

```
PRINTER = P:pm -o PRN
```

pipe command Si se usa esta alternativa, los resultados del reporte son enviados al proceso indicado por el comando *cmnd*, sirviendo de input para la ejecución del mismo.

terminal La salida por terminal genera una ventana con los resultados del reporte, permitiendo paginar vertical y lateralmente. La ventana generada tendrá tantas columnas como la línea más ancha del reporte. La cantidad de columnas máxima de la ventana creada nunca excederá el ancho de la terminal en uso, por lo tanto el usuario tendrá la posibilidad de paginar lateralmente. El largo de ventana también se calcula automáticamente, en base a la longitud de página definida en el reporte y la cantidad de filas disponibles en la terminal. En tal caso, el usuario podrá paginar verticalmente.

stdout Esta opción envía los resultados del reporte a la salida estándar, tal como ésta esté definida.

En caso de no especificarse esta cláusula se considera como salida la opción *output to printer* (default value).

```
input from <fuente>;
<fuente>: <cadena>
| <variable_de_ambiente> [...]
| pipe <cadena>
| <variable_de_ambiente>
```

```
| terminal
```

Esta sentencia se utiliza cuando el reporte es utilizado por doreport, permitiendo definir de dónde se leerá la entrada.

La primer opción de <fuente>, es utilizada para indicar la entrada mediante un archivo de datos. Por ejemplo, si en el archivo datos, se tiene almacenada la información que debe ser entrada al utilitario doreport, se puede especificar como:

```
input from "datos";
input from $datos;
```

haciendo primero referencia a un string de caracteres, y luego a una variable de ambiente; ésta debe existir al momento de ejecución.

En cambio:

```
%report
use personal;
input from pipe "iql -b -c
select depto, nombre, sueldo, fingr
from emp, depto
where emp.depto = depto.depno
output delimited;"
```

se usa para indicar, mediante la especificación *input from pipe(<dg>)*, que la entrada será la salida de un comando, mientras que input from terminal utiliza la *entrada estándar*. Cuando no está presente, doreport la emplea por defecto, siendo necesario indicar en el comando la opción ``-d", o cuál es la forma en que vienen separados los campos y los registros que entran al utilitario *doreport (<dg>)*. Finalmente, con la sentencia

```
language "C";
```

se informa que el reporte será usado desde la interfaz de programación, de modo que cuando sea procesado con el utilitario rgen se genere siempre el archivo de encabezamiento (archivo.rph), sin necesidad de indicarlo explícitamente.

La Sección %fields

Esta sección es semejante a la de un archivo de formulario. Pueden especificarse atributos para cada nombre de los campos de una zona. El orden en que los campos se nombran en esta sección, es aquel en el cual espera encontrarlos doreport.

La sintaxis de esta sección es una lista de los nombres de los campos seguidos por una lista opcional de atributos:

```
campo: opciones;
```

La especificación de opciones está formada por una lista de sentencias separadas por comas, que pueden ser:

<campo_tabla> Los atributos inherentes al campo en la especificación en la tabla de campos de la base de datos a la que se hace referencia. Es factible especificar atributos adicionales o derivados de otros ya existentes.

mask <cadena> Realiza una operación de máscara como el atributo *mask* definido en el *Capítulo II*, (Descripción de Sentencias DDS - CREATE TABLE). Como ser:

```
%zoneX(a)
_____
%fields
a: mask ($MASK != null ? $MASK : "NNNN");
```

Nótese que esta expresión no puede contener referencias a expresiones definidas en una zona.

null zeros Aplicable solamente sobre campos numéricos. Si el campo tiene valor "0", se dejará en blanco (" ") en lugar de imprimirlo.

Se explican a continuación las distintas sintaxis posibles, aplicables en cada caso a los anteriores ítems.

<campo_tabla> La forma de especificarlo puede ser:

```
esquema.tabla.campo
```

Esta es una especificación completa y definida. Se refiere a un campo de una tabla de un esquema. Las restantes definiciones implican un cierto grado de ambigüedad.

```
tabla.campo
```

La tabla debe pertenecer a uno de los esquemas activos, establecidos en la sentencia use. Se utiliza la primera de ellas que concuerde con el nombre tabla.

```
tabla.
```

Valen las consideraciones efectuadas para el caso anterior con respecto a *tabla*. El campo usado es aquel con el mismo nombre que el del campo de la pantalla.

<cadena> Una *cadena* es cualquier conjunto de caracteres encerrado entre comillas dobles (character string).

```
"Hola, mundo!"
```

<valor> Un valor puede ser cualquiera de los tipos válidos para las columnas de las tablas, que son: *numeric*, *character string*, *date*, *time*, *float* o *bool*.

Las variables especiales *today* y *hour* corresponden a la fecha y hora corriente. Cualquiera sea la cantidad de campos que se empleen con el utilitario doreport, el orden en que se especifican debe coincidir con el orden en que fueron colocados para su impresión.

Funciones y Variables

Cabe hacer las siguientes consideraciones con respecto al tipo de función (formato) que corresponde a cada una de las funciones permitidas en un reporte.

• Las funciones son creadas dinámicamente, cada vez que aparece una función en la definición de un reporte, se crea un nuevo campo virtual. Esto permite que la misma función sea impresa con diferentes formatos.

• Las funciones pueden ser usadas en expresiones (if a < sum(h)). En este caso se le asigna un tipo interno (el cual está señalado con una "x" en la Tabla 6), pues el tipo interno permanece indefinido. Cuando hay más de una "x", se eligen el tipo que haga válida la expresión.

• El tipo de una función está definido de dos formas:

1.Por el tipo seleccionado en la imagen del reporte.

2.Por el uso del mismo dentro de una expresión.

FUNCION	NUMERICO	NO-NUMERICO		SHORT	LONG	DOUBLE
DATE	TIME	CHAR	SUM	x	x	x
			AVG	x	x	x
x	x		COUNT	x	x	x
			MIN	x	x	x
x	x		MAX	x	x	x
x	x		DAY-MONTH-YEAR	x	x	x
			DAYNAME			
		x	MONTHNAME			
		x	TODAY			
x			HOUR			
	x					

Figura 4.5 - Funciones y Tipos Internos

Las variables tienen un tipo fijo asignado, y no puede ser modificado. La única opción posible es agregar short o long al valor numérico, como se puede observar en la siguiente tabla:

VARIABLE	NUMERICO	NO-NUMERICO		Char	Short	Long
Double	Date	Time	pageno		x	x
			lineno		x	x
			today			
	x		hour			
		x	flength		x	x
			botmarg		x	x
			topmarg		x	x

			leftmarg		x	x
			width		x	x

Figura 4.6 - Variables y Tipos Internos

Para completar esta descripción, veamos un diagrama de compatibilidad entre las funciones y los tipos de argumentos que aceptan.

FUNCION	NUMERICO	NO-NUMERICO		Char	Short	Long
Double	Date	Time	SUM		x	x
x			AVG		x	x
x	x	x	COUNT	x	x	x
x	x	x	MIN		x	x
x	x	x	MAX		x	x
x	x	x	DAY			
	x		MONTH			
	x		YEAR			
	x		DAYNAME			
	x		MONTHNAME			
	x					

Figura 4.7 - Funciones y Tipos de Argumentos Válidos

Definición del tipo de un campo:

1. Si se lo va a imprimir y no está definido aún, se le asigna el tipo de diseño de impresión (si es compatible en caso de ser una función).

2. Si ya tiene un tipo definido, pueden darse los siguientes casos:

• Son iguales: no existen problemas. Es el caso de una función con un solo lugar donde imprimir.

• Son distintos (ocurre cuando un campo debe imprimirse en distintos lugares):

• Válido sólo en los casos numéricos.

• Si hay decimales deben coincidir., entre la definición original (pe., la Base de Datos) y el diseño de impresión.

Expresiones:

• Si ambos campos están definidos, se verifica la compatibilidad.

• Si está definido uno solo, se define al otro de ese mismo tipo.

• Si ambos campos se hallan indefinidos, se produce ERROR.

Interfaz entre la Base de Datos y los Reportes

En la siguiente tabla se especifica la compatibilidad entre tipos de campo de la base de datos y del reporte:

BASE de DATOS	REPORTE		Char	Numeric	Time
Date	Float	CHAR	x		
		NUMERIC	x[a]	x	
		TIME			x
		DATE			
x		FLOAT			
	x	BOOL	x		

Figura 4.8 - Correspondencia de Tipos: BD/Reporte

Debe tenerse en cuenta que:

• La longitud del campo en el reporte puede ser cualquiera.

• En caso de que se halla indicado la opción "-w" al efectuar la compilación del reporte con el utilitario rgen, se avisa si los decimales no coinciden.

Listado Ejemplo

Como ejemplo, se construirá un listado del esquema personal que se utilizó en este capítulo. El listado debe dar para cada departamento una lista de sus miembros, imprimiendo el nombre del empleado y la fecha de ingreso. Se desea también, al final del listado, la cantidad total del "staff" de la empresa. La figura muestra un listado que tiene esas especificaciones, y está almacenado en un archivo llamado `emplo.rp`.

```
%reptitulo() before report
LISTADO DEL PERSONAL
====================
%pgtitulo(pageno) before page
Listado del Personal Page: ____.
=========================================
```

[a] Sólo si la máscara es completamente numérica.

```
%dep(dnum,ddes) before dnum
Departamento: __. _____________________________
NUMERO NOMBRE INGRESO
%person(pnum,pnombre,pfingr)
_____ ____________________ __/__/__
%tot(count(pnum)) after report
=========================================
Total Staff : _____. Empleados.
%report
%fields
dnum;
ddes;
pnum;
pnombre;
pfingr;
```

Figura 4.9 - Listado Simple

El listado se compila con el utilitario *rgen*:

```
$ rgen emplo.rp
```

Una vez que el listado fue compilado, se lo puede correr con doreport. Pero se le debe preparar una entrada en ASCII. Es fácil de lograr con la sentencia SELECT del iql. Para eso se prepara un archivo llamado list.sql con los comandos que siguen:

```
use personal;
select depno, depto.nombre,
empno, emp.nombre, fingr
from emp, depto
where emp.depto = depto.depno
order by depno
output delimited;
```

Se puede ver que la sentencia SELECT lista todos los empleados ordenados por número de departamento. De esta forma, cuando cambia el valor de depno se satisface la condición *before dnum*, y se imprime el encabezamiento. La estructura del registro tiene los campos en el orden que los espera el listado, en el orden especificado en la sección fields.

El listado se obtiene con los siguientes comandos:

```
$ iql list.sql | doreport emplo
```

En primer lugar se invoca al utilitario *iql*, pasándole como parámetro un archivo que contiene sentencias DMS (Data Manipulation Statements). Mediante el símbolo "|" (pipe) se indica que la salida que se obtenga del comando se tome como entrada del proceso que le sigue. En nuestro caso, los resultados arrojados al ejecutarse las sentencias incluidas en el archivo "list.sql" serán tomados por el utilitario doreport como parámetros del reporte. En el sistema operativo MS-DOS esta es la forma de simular las opciones *input from pipe* y *output to pipe*, de las especificaciones generales del reporte.

La salida impresa se muestra en el siguiente gráfico:

```
LISTADO DEL PERSONAL
====================
Listado del Personal al 03/01/88 Pag: 1
>=============================================
Departamento: 1 Desarrollo
NUMERO NOMBRE INGRESO
1 Juan Gatica 01/03/88
4 Hugo Riveros 01/04/86
Departamento: 2 Software de Base
NUMERO NOMBRE INGRESO
2 Charlie Parker 03/12/60
5 Albert Colby 03/11/70
Listado del Personal al 03/01/88 Pag: 2
=============================================
Departamento: 3 Documentación
NUMERO NOMBRE INGRESO
9 John Holmes 25/02/75
3 Sam Pepper 25/02/75
7 Peter Pan 30/10/66
Departamento: 4 Administración
NUMERO NOMBRE INGRESO
6 Piccolino Capri 11/09/87
=========================================
Total del Personal: 8 Empleados
```

Figura 4.10 - Listado de Salida

Otra opción, es incluir en la sentencia de iql, a qué report se debe direccionar la salida:

```
use personal;
select depno, departamento.nombre,
empno, emp.nombre, fingr
from emp, depto
where emp.depto = depto.depno
order by depno
output to report "emplo";
```

Nótese que el nombre del reporte no necesita llevar la extensión ".rp", dado que será asumida por el sistema. Con este cambio, el listado será producido por el comando:

```
$ iql list.sql
```

Usando Reportes

Existen distintas formas de utilizar un reporte, una vez que ha sido procesado con el utilitario rgen:

• Mediante el utilitario "doreport" de IDEAFIX.

• Desde un programa de aplicación en lenguaje "C", mediante rutinas de manejo de reportes, provistas por la biblioteca de IDEAFIX.

• Direccionando la salida de una sentencia SELECT de IQL.

Se verá a continuación cuales son los métodos existentes.

DOREPORT

Doreport es un utilitario que lee una secuencia de registros de datos por su entrada estándar. Esos datos son los que alimentan la definición del reporte que se le pasa como parámetro.

Se indica de donde tomará la entrada de datos especificándolo en la sección `"%report"` de la definición del reporte (ver Lenguaje RDL).

Este utilitario trabaja como un filtro, lee su entrada, la formatea, y escribe el resultado en la salida. La entrada debe ser en ASCII, admitiendo distintos formatos de delimitación de campos, en forma similar a los utilitarios imp y exp (ver *Capítulo 2 - Usando Bases de Datos /Utilitario export-import*).

Esto permite generar reportes en forma sumamente rápida y sencilla a través de la combinación de estos utilitarios como muestra la figura:

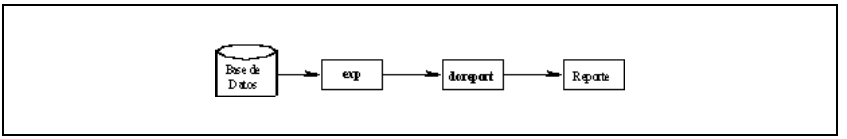


Figura 4.11 - Generación de Reportes

Para poder trabajar en forma correcta con este utilitario debe asegurarse que el orden de los campos en el lote de datos de entrada es el que se ha definido en el reporte.

La forma de utilizar doreport es la siguiente:

```
a) $ doreport -Fc -Rc reporte
```

Mediante estas opciones se indica que:

Rc: Define a *c* como el carácter separador de registros. Por defecto es el carácter "newline" (\n).

Fc: Define a c como el carácter separador de campos. Por defecto es la coma (,).

```
b) $ doreport -d reporte
```

-d Mediante esta opción se indica que se toman los delimitadores de campos y registros como los estándares, es decir que se espera como separador de campos a las "," y al de los registros al "\n".

```
c) $ doreport reporte param1 param2 ...
```

Los parámetros que se pasan al utilitario serán tomados como valores que saldrán impresos en el reporte en el lugar donde se lo haya referenciado en el archivo de especificación del reporte con \$1, \$2, etc..

```
d) $ doreport -nNN reporte
```

Emite NN copias del reporte.

Interfaz C

Las distintas aplicaciones con reportes pueden ser desarrolladas mediante la aplicación del utilitario "doreport" y con las instrucciones del ``iql", al cual se ha hecho referencia en este mismo capítulo.

En caso de que algún requerimiento no pueda ser satisfecho con las herramientas antes mencionadas, puede recurrirse a la Interfaz de Programación (ver *Capítulo 1*, de *Referencia del Programador*). Esta permite el desarrollo de reportes de distinto nivel de complejidad, para cualquier aplicación. La Biblioteca de IDEAFIX posee un subconjunto de funciones para manejo de reportes, entre las cuales se destaca la función DoReport. Esta función junto con las funciones *DbToRp*, *RpSetFld* y otras, permiten el desarrollo de las aplicaciones antes mencionadas.

La función *DoReport*, es la que produce la salida de una zona que se haya definido en el archivo de especificación del reporte. En caso de que dicha zona contenga campos, debe haberse dado valores a los mismos mediante la función *RpSetFld* o alguna de sus variantes. Ellas tienen por función específica la asignación de contenidos a los distintos campos definidos en el reporte, existiendo una variante de "Set field" para cada tipo de formato (float, date, time, char, etc.).

DbToRp (Data base to Report) copia valores de una tabla de la base de datos a los campos del reporte que se le indiquen.

Para tener más información de estas funciones remitirse a al parte II de este manual.

En un programa en lenguaje "C" desarrollado para trabajar con un reporte, también puede ser utilizada la función DoForm en caso de que se esté utilizando una pantalla para referenciar la información que se desea esté contenida en el reporte. Por ejemplo, asociar una pantalla a un reporte para que ciertos datos de la pantalla se impriman en el reporte. Se puede obtener más información sobre ésto en la sección Generador de Formularios.

Por ejemplo, si se quiere obtener un listado, de los empleados con el siguiente formato:

```
%tit(pageno, today) before page
Página: __.
Fecha : __/__/__
Listado de Empleados
====================
Depto. Empleado Sueldo F. Ingreso
=========================================================
%linea(rdepto, nombre, sueldo, fingr)
__. ____________________________ _______,__. __/__/__
%report
use personal;
%fields
rdepto: emp.depto;
```

```
nombre: emp.;
sueldo: emp.;
fingr : emp.;
```

Figura 4.12 - Especificación de Reporte

Se puede obtener mediante el acceso a los datos pertinentes con las sentencias del iql o el *doreport*, utilizando a éste último como formateador.

```
/*****************************************************
* DENOMINACION:Listado de Empleados *
*PANTALLAS:No se usan *
*REPORTES:rep.rp *
*****************************************************/
#include <ideafix.h>
#include "rep.rph"
#include "personal.sch"
wcmd (rep, 1.1 08/08/89)
{
report rp;
rp = OpenReport("rep", RP_EABORT);
while(GetRecord(EMP,NEXT_KEY, IO_NOT_LOCK)!=ERROR){
DbToRp(rp, RDEPTO, FINGR);
DoReport (rp, LINEA);
}
}
```

Figura 4.13 - Programa en "C"para manejo de Reportes

Este no es un caso de gran complejidad, pero su objeto es mostrar un ejemplo de programa desarrollado en "C", empleando algunas de las funciones de la biblioteca de interface de IDEAFIX para obtener el listado.

Como se ve, en el programa se deben incluir los archivos cabecera resultantes de compilar el reporte rep.rp con la opción "-h" del rgen, habiendo creado el archivo cabecera del esquema.

Mediante este programa se abre un reporte, y por cada registro de datos obtenido se realiza la transferencia de dichos datos desde el buffer de la base de datos al área del reporte; luego, mediante la función DoReport, se provoca la salida al destino indicado mediante la variable de ambiente printer, que puede ser un device o un archivo. Remitimos al lector al capítulo de Funciones para comprender el funcionamiento de cada una de las que se han utilizado en esta porción de código. El listado tendría el siguiente aspecto:

```
Página: 1
Fecha : 23/08/92
Listado de Empleados
====================
Depto. Empleado Sueldo F. Ingreso
=================================================
1 Juan Gatica 1900,00 01/03/88
1 Hugo Riveros 1090,00 01/04/86
2 Charlie Parker 900,00 12/10/60
2 Albert Colby 1550,00 11/03/70
3 John Holmes 1000,00 25/02/75
```

```
Página: 2
Fecha : 23/08/92
Listado de Empleados
====================
Depto. Empleado Sueldo F. Ingreso
=================================================
3 Sam Pepper 1350,00 25/02/75
3 Peter Pan 1200,00 30/10/66
3 Piccolino Capri 1200,00 09/11/87
```

Figura 4.14 - Ejemplo de Impresión de Reporte

Capacidades Máximas

Número de Reportes abiertos simultaneamente	8
Número de Esquemas abiertos simultaneamente	4
Carácteres en un campo alfanumérico	65535
Dígitos en un campo numérico	15
Dígitos significativos en un campo numérico	15
Rango de valores en un campo fecha	16/04/1894 to 16/09/2073
Rango de valores en un campo hora	00:00:00 to 23:59:58[a]
Nombres de Esquema	10
Nombres de Tabla	10
Nombres de Campo	Unlimited
Campos por Reporte	128
Zonas por Reporte	128

[a] El último horario para un día es 23:59:58, esto se debe a que IDEAFIX tiene una precisión de 2 segundos para los datos del tipo Time.

Capítulo 18

Menúes

Especificación de Menúes

Un menú se especifica en un archivo con la extensión ".mn". El archivo contiene información de los nombres de las opciones, y qué hacer cuando el usuario elige cada una de ellas. El formato de cada línea es el siguiente:

```
Texto_opcion Tipo_de_opcion Argumentos
```

`Texto_opcion` es la leyenda que se despliega en la ventana. Si la leyenda incluye espacios en blanco se la debe encerrar entre comillas dobles.

`Tipo_de_opcion` y sus argumentos especifican la acción a ejecutar cuando el usuario elige la opción. Existen seis tipos de opción:

• MENU. Cuando se la selecciona, se despliega otro menú. El argumento es el nombre del archivo con las especificaciones del nuevo menú.

• SHELL. El argumento especifica el nombre de un comando que ejecutará el sistema operativo. Cuando se termina de ejecutar el comando, el control se devuelve al menú.

• PIPE. El argumento especifica el nombre de un comando que ejecutará el sistema operativo, pero la salida del comando se muestra en una ventana. Opcionalmente el tamaño de esta ventana puede ser especificado. Después de la palabra PIPE puede aparecer:

```
PIPE [filas] [columnas] comando argumentos.
```

La opción PIPE no está destinada a procesos interactivos. Trabaja con programas que sólo envían salida a la pantalla. El típico ejemplo es un comando que lista los contenidos de un directorio.

• WCMD. El argumento especifica un programa de aplicación, que debe ser ejecutado con el Window Manager. Todo programa que necesite del WM debe ser indicado en un menú con este tipo de opción.

• BUILTIN. El argumento especifica una función incorporada al Window Manager. Las funciones BUILTIN son las siguientes:

Función	Descripción
GoToShell	Va al Shell
_print_scr	Imprime la pantalla
_print_ichset	Imprime el juego de carácteres
Calculator	Despliega una calculadora
Servmov	Posiciona una ventana

Ejemplo

Se muestra un ejemplo simple de la especificación de un menú:

```
"1. Listado del Directorio" PIPE 10 60 ls -lCF
"2. Calculadora de UNIX" SHELL bc
"3. Otro menu" MENU otro
"4. Programa Aplicativo" WCMD doform formen4
```

La primera opción ejecutará el comando `ls -lCF` y la salida aparecerá en una ventana de 10 filas por 60 columnas.

La segunda posibilidad corre el programa `bc` que provee las funciones de una calculadora. Cuando se termine con `bc` el control se devuelve al menú.

La tercera opción invoca a un nuevo menú, cuyas especificaciones están en un archivo llamado `otro.mn`.

La cuarta alternativa ejecuta el programa *doform* con el formulario *formen4*.

Permisos de los Programas

Cuando se ejecuta un programa desarrollado con IDEAFIX es posible determinar dinámicamente que operaciones tendrá habilitadas en los formularios. Estas operaciones son las siguientes:

A - (Adicionar) Agregar nuevos datos.

U - (del inglés Update) Modificar datos existentes.

D - (del inglés Delete) Borrar datos existentes.

Normalmente un programa tiene habilitadas en principio todas las operaciones. Pero es posible inhibir algunas o todas ellas mediante una indicación en la línea de comandos al momento de lanzar el programa a ejecución.

Esta indicación consta de un signo "!" (usado habitualmente como operador lógico para denotar negación) seguido de una o más de las letras indicadas en la tabla anterior (A, D o U). Por ejemplo

```
cnt101 !D
```

invocará el programa `cnt101`, pero sin permitir borrar información.

Esta indicación es automáticamente tratada por IDEAFIX y no llega al programa como argumento. Por lo tanto, si un programa necesita un argumento en tiempo de ejecución, éstos se suministran luego de la indicación sobre los permisos.

Esta indicación, si está presente, debe ir siempre en primer lugar, de otro modo se tratará como un parámetro más.

Esta característica permite configurar en los menúes de usuario los permisos de trabajo. Por ejemplo, un usuario puede utilizar un programa mediante el siguiente menú:

```
"1. Cuentas contables" WCMD cnt101 !ADU
```

por lo que sólo podrá consultar información existente. En cambio, quien tenga indicado en el menú:

```
"1. Cuentas contables" WCMD cnt101
```

podrá realizar todo tipo de operaciones. Notar que se trata del mismo programa en ambos casos, pero en el primero se lo ha convertido simplemente en un programa de consulta.

Ayuda en los Menúes

Cuando se solicita ayuda de aplicación con la tecla <AYUDA_APL> en un menú, se desplegará una ventana de ayuda que mostrará un archivo con igual nombre que el archivo que contiene el menú, pero con extensión ".hlp". Este archivo debe residir en un subdirectorio "hlp" que dependa de alguno de los directorios indicados en la variable de ambiente PATH. Es usual que si los archivos ejecutables están en un directorio como

```
/us/aplicn/bin
```

el directorio de ayuda para la aplicación "aplicn" es

```
/usr/aplicn/bin/hlp
```

Si el archivo no existe se desplegará una ventana informando la situación.

Capítulo 19

Utilitarios de Desarrollo

Este capítulo consta de la descripción de los utilitarios de desarrollo de IDEAFIX. Para cada uno existe una página que lo describe en forma esquemática indicando las opciones que acepta.

Las páginas están ordenadas en forma alfabética según el nombre de cada utilitario. La primera página se denomina INTRO y explica el formato de las páginas de esta sección de referencia.

La siguiente es la lista de utilitarios del sistema de desarrollo:

ideafix Ambiente de desarrollo guiado por menúes.

cfix Compilador y encadenador de archivos fuente en "C".

dgen Crear o modificar un Esquema de Base de Datos.

fgen Compilador de archivos FDL (definiciones de formularios).

gencf Generador automático de programas "CFIX".

genfm Generador automático de formularios.

dali Editor de textos.

rgen Compilador de archivos RDL (definiciones de reportes).

tar Utilitario de almacenamiento.

testform Simulador de ejecución de formularios.

Introducción

Este capítulo describe en orden alfabético, los comandos que forman parte del sistema de desarrollo de IDEAFIX. Cada página va encabezada por el nombre del comando, a la manera de ésta que lleva el nombre Intro.

A menos que se indique lo contrario, los comandos descriptos en este capítulo aceptan opciones y argumentos de acuerdo con la siguiente sintaxis:

```
nombre [opcion(es)] [argumento(s)]
```

donde:

nombre Es el nombre de un archivo con un programa ejecutable.

opción noargletra(s), argletra<>opargum, donde <> es espacio en blanco opcional. Los significados son los siguientes:

noargletra: Opción de una sola letra que no necesita argumento.

argletra: Una letra representando una opción que requiere un argumento.

opargum: Cadena de caracteres argumento de la opción *argletra*.

argumento Cualquier nombre de archivo u otro nombre que no comience con un guión, o bien "-" solamente, que indica la entrada estándar.

Todos los utilitarios de IDEAFIX admiten la opción "-?", que significa imprimir la forma de uso del comando, al igual que la opción "-b", por la cual se indica no emitir el mensaje que identifica el utilitario y su versión.

Cada entrada del capítulo está presentada usando los títulos normalizados que se describen seguidamente (aunque algunos de ellos puedan omitirse para ciertas entradas).

Sintaxis

Sumariza la forma de usar el comando que está describiendo. Se han empleado las siguientes convenciones:

Los corchetes "[ ]" indican que el argumento encerrado en ellos es optativo. Cuando el prototipo de un argumento está dado como "nombre" o "archivo" se refiere siempre a un nombre de archivo.

Los puntos suspensivos ". . ." indican que el argumento anterior puede repetirse.

Descripción

Provee una visión general del propósito del comando. Usualmente incluye un subtítulo

Opciones

Se explican las posibles opciones que admite el comando.

Ejemplo

Esta sección brinda casos ilustrativos o aclaratorios.

Notas

Contiene detalles o comentarios adicionales.

Archivos

Indica nombres de archivos usados por el comando.

Diagnósticos

Describe los mensajes de error o advertencia que pueden ser producidos por el programa durante su ejecución.

Consultar

Brinda referencias a otros documentos o partes del manual donde encontrar información relacionada.

Al finalizar cada comando retorna un código conocido como "status", también denominado "código de retorno". Este código es "0" para terminación normal y distinto de cero cuando durante el proceso se ha encontrado algún error. Es costumbre que el valor numérico del status guarde relación con la gravedad o magnitud del problema (de allí que "cero" equivalga a "sin errores").

IDEAFIX

Ambiente de desarrollo guiado por menúes 4GL.

Sintaxis

```
ideafix
```

Descripción

Este comando ingresa al Ambiente de Desarrollo de IDEAFIX: un sistema de menúes que permite acceder a todas las herramientas que componen dicho ambiente.

CFIX

Compila y encadena archivos fuente en C.

Sintaxis

```
cfix [-b] [-f] [-g] [-n] [-c] [-e] [-o nombre] [-s] [-I
directorio] [-x opciones] [-y opciones] [-E] [-S] modulo...
[-l biblioteca...]
```

Descripción

Este utilitario permite compilar y encadenar módulos de programas. Si la extensión de los módulos es .c primero los compila y luego los vincula ("linkedit") junto a los que tengan extensión.

En MS-DOS las extensiones son *.obj* para los módulos objeto y *.lib* para las bibliotecas de funciones.

Este utilitario invoca internamente al compilador y al vinculador del sistema operativo, por lo tanto estos programas ("C" compiler, linkage editor) deben estar instalados para poder utilizar la interfaz de programación.

El módulo ejecutable tendrá el mismo nombre que el primer módulo de la lista de argumentos, pero agregándole la extensión *.exe*. Si se quisiera cambiar el nombre del módulo ejecutable, debe apelarse a la opción "-o".

Cuando se toma un módulo fuente, antes de compilarlo se verifica si ha sido modificado respecto a su correspondiente módulo objeto, si es que éste existe. De ser así, se lo compila; en caso contrario se utiliza el módulo objeto ya existente.

Opciones

-b No muestra identificación de versión.

-f Fuerza compilación de los programas "C" aún si el módulo fuente no ha sido modificado. Esto puede ser necesario si se han alterado archivos de encabezamiento (v.gr.: .fmh de formularios) que al contener constantes cuyo valor ha cambiado obligan a la recompilación del módulo.

-g Compilar para debugging.

-n Sólo mostrar los comandos sin ejecutar.

-c No realizar la etapa de link edición.

-e No agregar la extensión .exe al programa ejecutable.

-o *nombre* El archivo ejecutable resultante se llamará "nombre". Para que incluya la extensión .exe debe especificársela directamente al dar el nombre; es decir, debe indicarse nombre.exe. No cabe aquí por consiguiente el uso de la opción **-e**.

-s No utilizar la biblioteca compartida. Esto toma en cuenta que ella se en cuentra operativa sólo en sistemas con "shared libraries".

-l *directorio* Se indica el camino de un directorio de donde se obtendrán archivos de include.

-y *opciones* Se pasa una cadena que se utilizará como parámetro del linkeditor.

-x *opciones* Se pasa una cadena que se utilizará como parámetro del compilador.

-E Realiza el link con la biblioteca de acceso a Essentia (el server de base de datos de InterSoft Argentina).

-S Realiza el link con la biblioteca de acceso a IDEAFIX.

-l *librería* Utiliza la biblioteca de funciones especificada.

Notas

Usa la variable de ambiente IDEAFIX para obtener el directorio de "include".

Ejemplo

```
cfix -f sto2011.c sto2012.o sto.a
```

En este caso el utilitario compila el módulo sto2011 para generar el módulo objeto, y luego encadena los módulos sto2011.o y sto2012.o junto con la biblioteca de funciones sto.a. La opción **-f** fuerza a recompilar el módulo, pese a que no haya sufrido alteraciones desde la última compilación.

DGEN

Creación y Modificación de esquemas de Base de Datos.

Sintaxis

```
dgen [-b] [-c 'comando'] [-f] [-h] [-m] [-v] [-p] [-s] [-n]
[-l] [-k] archivo[.sc] ...
```

Descripción

Este utilitario reconoce un subconjunto del IDEAFIX Query Language (IQL, o Lenguaje de Consulta IDEAFIX) llamado DDS (Sentencias de Definición de Datos). Estas sentencias son las que permiten crear y modificar las estructuras de bases de datos.

Este utilitario lee el o los archivos indicados como parámetros y ejecuta las sentencias contenidas en ellos. En el caso más simple, dicho archivo contiene las sentencias para crear

una nueva base de datos. En tal caso se invoca el utilitario para proceder a la creación de esa base de datos. Para generar el esquema y sus tablas asociadas puede ejecutarse:

```
$ dgen esquema
```

Cuando es necesario modificar un esquema para eliminar o crear nuevas tablas, o si se modifican o eliminan campos de una o más de ellas, se debe informar esta situación con la opción **-m**. Por ejemplo, si se modificó una tabla de un esquema, editando el archivo esquema.sc, que ya ha sido creado, y se ejecuta:

```
$ dgen esquema
```

dgen dará un error avisando que no es posible crearlo puesto que ya existe. Para esta situación existe la opción -m (modificación). Por lo tanto, el comando debe ejecutarse con el formato:

```
$ dgen -m esquema
```

Al realizar modificaciones que impliquen cambios de formato, los datos preexistentes se convertirán a su nueva definición sin que se produzca pérdida de información. Sin embargo, hay algunos casos en los que se puede presentar dicha posibilidad. Ellos son:

• Se reduce la precisión de un campo, ya sea en los decimales, en la cantidad total de dígitos, o en la cantidad de posiciones de un campo alfanumérico.

• Se elimina un campo de una tabla.

En estos casos el *dgen* avisa que la operación requerida implica pérdida total o parcial de la información contenida en tales campos y no la ejecuta. Empero, si efectivamente se quiere realizar la modificación aún a costa de perder datos, ello debe indicarse mediante la opción **-f** (forzar).

Los mensajes de error que emite hacen referencia a la línea donde comienza la definición de la tabla.

Opciones

-b no despliega el mensaje identificatorio

-c genera el encabezado (*header*) para un programa en COBOL que usará IDEAFIX o Essentia.

-c2 lo mismo que -c, pero agrupando todas las tablas en un documento con el nombre del esquema.

-f modifica de manera incondicional un esquema existente, lo que puede ocacionar pérdida de información.

-h se genera un archivo cabecera (con extensión ".sch") para ser incluido en programas desarrollados en "C" que utilizan esquemas ya generados.

-m modifica un esquema existente. Si no se puede convertir toda la infor mación, se detiene el programa.

-v *verbose*. Mediante esta opción se muestran las operaciones a medida que son ejecutadas.

-p genera o modifica el esquema asignando todos los permisos a todos los usuarios.

-s toma la entrada de stdin y envía la salida a stdout.

-n abre todos los esquemas sin las descripciones, para ahorrar memoria.

-l modifica el esquema existente aunque esté bloqueado.

-k efectúa un *checkpoint* sobre cada esquema creado.

Reconfiguración de Indices

Cada índice de una tabla tiene un parámetro conocido como "separ" que afecta la cantidad de memoria que utiliza el índice. Ese parámetro se define en la especificación SQL, de la siguiente manera:

```
primary key (campo, campo, ....) [separ]
```

o bien

```
index indice (campo, campo, .....) [separ]
```

separ es un número usualmente comprendido entre 1 y 10. Cuando no se especifica este valor, se asume en ausencia el valor 1.

El valor de separ se puede estimar para minimizar la cantidad de memoria y obtener buena performance, en función de la cantidad de registros que contendrá la tabla de la siguiente manera:

```
separ = ((lc + 4)*V(size))/724
```

donde las variables utilizadas tienen el siguiente significado:

size Cantidad máxima esperada de registros en la tabla.

Longitud de la clave. Es la suma de los bytes ocupados por cada uno de los campos que la componen.

A los efectos del cálculo, resulta útil aplicar la tabla dada a continuación.

Externo	Interno
num(1), num(2), bool	1 byte
num(3), num(4), date, time	2 bytes
num(5) to num(9)	4 bytes
num(10) to num(15), float	8 bytes (punto flotante)

num(16) to num(28) char(x)	s16 bytes x bytes

Figura 6.1 - Tamaños de los componentes de la clave

Ejemplo

Supongamos que se prevé un volumen de 25.000 registros para la tabla que responde a la definición siguiente:

```
table stock
{
itemno num(4),
serno num(7),
ordno num(4),
prvta num(14,2),
fecomp date,
prcomp num(14,2)
}
primary key (itemno, serno);
```

El valor de separ se calcula teniendo en cuenta que los campos que forman la clave primaria son itemno (2 bytes) y serno (4 bytes):

```
lc = 2 + 4 = 6(tam. clave)
separ
```

Para cambiar la separación de un índice ya existente, sólo hace falta agregar en el archivo .sc del esquema que lo contiene la indicación [separ] con el valor apropiado, y luego ejecutar.

```
$ dgen -v -m esquema
```

Diagnosticos

Los mensajes de error se pueden dividir en tres grandes grupos: aquellos que son detectados durante la compilación; los que se producen en tiempo de ejecución, y los mensajes de advertencia (warnings). Estos grupos se numeran del 1 al 500, del 501 al 1000, y del 1001 en adelante, respectivamente.

Errores en Tiempo de Compilación

1 `Error de Sintaxis.`: Indica que la especificación de una sentencia es incorrecta. Por ejemplo: falta el ";" al final de la sentencia; no se empleó una coma para la separación entre campos; falta algún operando obligatorio de una cláusula (from); error de escritura de alguna palabra clave, etc.

2 `Carácter octal inválido %o.`: Existe un carácter que es ilegal en esa posición o bien es un carácter de control.

3 `Cadena de caracteres sin cerrar.`: Se ha omitido el carácter de cierre de una cadena de caracteres, ya sea """ ó """". Por ejemplo: Cuando se le coloca nombre a una columna seleccionada ["Nombre del Empleado] (no se cerró la cadena con ").

4 `Archivo no encontrado.`: Con la sentencia exec o bien desde la línea de comandos, se intenta leer un archivo que:

a)No existe.

b)Tiene el nombre mal deletreado.

c)Pertenece a un directorio no accesible.

5 `Tabla de Macros excedida.`: Se han ejecutado demasiadas sentencia define.

6 `Sentencia Errónea.`

7 `Falta especificación de CLAVE PRIMARIA.`: No se ha definido clave primaria de acceso para una tabla.

8 `Error de definición de Macro.`

9 `Redefinición de CLAVE PRIMARIA.`: Se ha definido el atributo "primary key", para más de un campo.

10 `No se admiten condiciones múltiples de verificación.`: Se ha especificado más de una validación como atributo de un campo.

11 `Valor %s incompatible con el tipo de campo.`: Se especifica un valor que no corresponde con el tipo de campo dado.

12 `Longitud (%d) excede valor de longitud del campo (%d).`

13 `Valor con demasiadas (%d) posiciones enteras. Máx: %d.`

14 `Valor con demasiadas (%d) posiciones decimales. Máx: %d.`

15 `Fecha (DATE) inválida: %s.`: Se ha indicado una fecha no válida (por ejemplo: 29/02/83).

16 `Hora (TIME) inválida: %s.`: Se ha indicado una hora no válida (por ejemplo: 35:45:89).

17 `Tabla %s no encontrada.`: Se ha hecho referencia en una operación a una tabla no definida.

18 `Tabla %s no declarada.`: Se ha mencionado una tabla (por ejemplo en una validación in table) que no pertence al esquema.

19 `Campo no válido.`: Se hace referencia a un campo inexistente. Cantidad errónea de valores.

20 `Número erroneo de valores.`

21 `Campo '%s' no definido.`

22 `Tabla '%s' en '%s' no definida.`

23 `Uso incorrecto del Alias '%s' en '%s'.`

24 `Redefinición del Alias '%s'.`: Se ha redefinido un alias en la sentencia from.

25 `Variable de ambiente '%s' indefinida.`: Se ha hecho referencia a una variable de ambiente que no está definida en el momento de la compilación.

26 `Número de dígitos no debe exceder a 28.`: Se ha definido un campo numérico con más de 28 dígitos.

27 `Cantidad de decimales debe ser < cant. dígitos.`: Se ha definido un campo con cantidad de dígitos en parte entera menor que en la parte decimal.

28 `Tabla de Variables excedida.`: Se han definido demasiado variables con la sentencia set.

29 `Función de Grupo '%s' no utilizable en esta cláusula.`

30 `Lenguaje '%s' No Soportado.`

31 `'%s' Nombre de usuario inexistente.`

32 `'%s' Nombre de grupo inexistente.`

33 `Nombre demasiado largo (maximo: %d caracteres).`

37 `Error en Default '%s'.`

Errores en Tiempo de Ejecución

501 `Sentencia no válida en el modo corriente.`: Se intenta ejecutar una sentencia de manipulación de datos en el utilitario *dgen*.

502 `Esquema '%s' no encontrado.`: Se ha hecho referencia en una operación a un esquema no activo.

503 `Tabla '%s' no encontrada.`: Se ha hecho referencia en una operación a una tabla no definida, o no perteneciente a ninguno de los esquemas activos.

504 `Campo '%s' no encontrado.`: Se ha hecho referencia en una operación a una campo no definido.

505 `Indice '%s' no encontrado.`: Se ha hecho referencia en una operación a una índice no definido.

506 `No se pudo abrir el esquema '%s'.`: Este error se puede producir si el usuario no tiene permisos de acceso al esquema o bien el esquema no existe, o el esquema existe y está corrupto, o está bloqueado para uso exclusivo para otro usuario.

507 `Falló operación de grabación de definición de esquema '%s'.`: En la sentencia store schema no se pudo abrir el archivo de salida donde se debía grabar la definición.

508 `No se pudo abrir archivo HEADER para esquema '%s'.`: Idem anterior para la sentencia store schema header.

509 `No se pudo crear la Tabla '%s'.`

510 `No se pudo crear el Indice '%s'.`

511 `No se pudo crear el Esquema '%s'.`

512 `No puede agregarse el campo '%s'.`: Los errores anteriores se producen al intentar crear o modificar un esquema.

513 `Cantidad errónea de campos clave en la cláusula 'in'.`: Se ha definido la clave de acceso en un atributo "in" con mayor cantidad de campos que la correspondiente.

514 `Demasiadas columnas de salida.`: Este error se produce cuando hay demasiadas columnas en la grilla de salida. No ha sido utilizado.

516 `No puede crearse esquema '%s' : ya existe.`: Se ha intentando crear un esquema que fue creado con anterioridad.

517 `No puede crearse tabla '%s' : ya existe.`: Se ha intentado crear una tabla, creada con anterioridad.

525 `No puede borrarse (drop) '%s'.`

526 `No puede crearse tabla temporaria.`

527 `Redefinición de Alias : %s.`: Ha sido redefinido un alias en la sentencia from.

530 `Fecha (DATE) no válida: %s.`: Se ha indicado una fecha no válida (por ejemplo: 29/02/83).

531 `Hora (TIME) no válida: %s.`: Se ha indicado una hora no válida (por ejemplo: 35:45:89).

532 `No puede aplicarse 'resto' (Remainder) a números no enteros.`: Se ha aplicado la función rem a un campo numérico no entero.

533 `Operación '%s' no válida para el tipo '%s'.`

534 `Operación '%s' : Tipos de datos incompatibles '%s' & '%s'.`: Se está realizando una operación entre tipos de datos incompatibles, por ejemplo dividiendo una fecha por un número entero. También ocurre cuando no se colocan las comillas para encerrar una fecha.

535 `No puede cambiarse el tipo de variable.`

536 `Valor de longitud (%d) excede la del campo (%d).`

537 Inserción cancelada. Registro ya existente.

538 Imposible bloquear el esquema '%s' : otros procesos lo están utilizando.: No puede bloquearse el esquema indicado, pues está siendo utilizado o bloqueado por otro proceso.

539 No puede abrirse esquema '%s': está bloqueado para uso exclusivo.: El esquema no puede ser abierto, pues otro proceso lo está bloqueando con alguna operación.

540 Campo %s Valor:%s no cumple condición '%s'.

553 %s : La Tabla %s no existe.

554 %s : Los nombres de Tabla deben diferir.: No pueden existir definiciones de tablas con el mismo nombre.

555 %s : La Tabla %s ya existe.

556 Las nuevas tablas (como '%s') deben agregarse al final del esquema.: Las tablas deben agregarse al final de la definición del esquema en caso de modificarlo.

557 Tabla '%s' fuera de secuencia.

558 No hay esquema corriente.

559 %s : Los nombres de campo deben diferir.

560 %s : El campo %s ya existe.

561 No puede borrarse el directorio '%s'.

562 Falló operación borrado (drop) para esquema '%s'.

563 Cláusula 'Having' requiere especificar cláusula 'group by'.

568 Nivel erróneo de Expresión para cláusula 'having'.

569 Nivel erróneo de Expresión en cláusula 'order by'.

570 Expresión numero %d no incluida en cláusula 'group by'.

571 Demasiados niveles de funciones agrupadas en Expresión %d.

572 Expresión con subíndice para campo único '%s'.

573 Subíndice fuera de rango para Campo '%s'. Valor %d.

574 Descripción de Operando no válida para campo '%s'.

575 Imposible convertir table '%s': campo '%s' no nulo y carece de default.

576 No puede bloquearse la tabla '%s'.

577 Esquema '%s' bloqueado para uso exclusivo por otro proceso.

578 La definicion encontrada del esquema '%s' no es valida.: Las causas posibles son:

1) Creada para CPU con formato de datos incompatible.

2) Fue creada por una versión previa de IdeaFix. Asegurese de haber actualizado la base de datos a la version actual.

579 No tiene permiso para %s (tabla %s.%s). (Dueño del esquema: %s).

580 Campo %s: El valor excede la longitud del mismo (%d).

581 Esquema '%s': No tiene permiso para USE. (Dueño del esquema: %s).

582 Funcion '%s': INOPERANTE en modelo comprimido.

583 Esquema '%s': No tiene permiso para crear indices o tablas temporarias (TEMP). (Dueño del esquema: %s).

584 '%s': Esquema '%s', no tiene permiso para efectuar instrucciones manipulativas (MANIP). (Dueño del esquema: %s).

585 No tiene permiso para modificar la estructura del esquema '%s'.(ALTER). (Dueño del esquema: %s).

586 Debe ser 'root' o dueño del schema para ejecutar 'chown'. (Dueño del esquema: %s).

587 Falló la operación: 'chown %s %s'.

588 Comando no encontrado.

589 Directorio invalido.

593 Panel muy chico para mostrar todas las barras.

594 No hay rangos 'Y' en grafico XY.

595 Valores negativos en grafica apilada.

596 Valores negativos en grafico torta.

597 No se pudo escribir en el archivo.

598 No hay rango X en grafico XY.

599 Versión del esquema no existente.

Mensajes de Advertencia

1001 Comentarios sin cerrar al fin de la entrada.

1002 Clausula 'in table' ignorada.

```
1003 Expresion de campo '%s' no incluida en clausula 'group by'.

1004 Expresion numero %d no incluida en clausula 'group by'.

1005 Tabla %s. Tamaño máximo menor a la cantidad actual de registros.
Fue ajustado a %ld.

1006 Intentando desbloquear el esquema '%s'.
```

FGEN

Compila archivos fuentes FDL.

Sintaxis

```
fgen [-b] [-h] [-t] [-w] archivo[.fm] [archivo ...]
```

Descripción

Este programa convierte la definición en FDL de un formulario a la representación usada por IDEAFIX en tiempo de ejecución.

La definición de un formulario está contenida en un archivo de definición con la extensión ".fm", conocido como archivo fuente FDL. El utilitario fgen genera un archivo con la extensión ".fmo" a partir del fuente FDL, y opcionalmente un archivo de cabecera con la extensión ".fmh". Este último (ForM Header) se generará sólo si se incluye la claúsula language en el formulario o si se especifica la opción **-h** al invocar el utilitario fgen. Si el archivo con extensión ".fmh" existiera, el programa verifica si se ha modificado el orden de los campos, o si se agregaron o quitaron algunos de ellos, antes de crearlo nuevamente.

fgen verifica que no haya inconsistencias cuando se asocian campos del formulario con la Base de Datos.

Si la longitud de un campo especificada en el formulario es menor que la de la base de datos, será aceptada. En caso de ser mayor, la longitud se ajustará a la especificación de la base de datos.

Los mensajes de error que el utilitario envía son auto-explicativos. Tener siempre en cuenta que un error simple puede desencadenar una cascada de errores espúreos, es decir, corregido el primer error los demás quedan subsanados. Por ello es importante prestar suma atención al o los primeros mensajes de error cuando la lista es muy larga.

Opciones

-b No se muestra el mensaje de identificación del utilitario ("banner").

-h Genera un archivo cabecera ("header") con extensión ".fmh", para ser incluido en los programas "C" que utilicen el formulario. Dicho archivo también será generado automáticamente se incluye la claúsula language en el formulario.

-t Esta opción indica que se envíen los mensajes a la salida estándar.

-w Permite obtener mensajes de advertencia ("warnings"), tales como diferencias entre campos del formulario y los de la Base de Datos, superposición de atributos, etc.

Diagnosticos

La siguiente es la lista de los errores que pueden ser arrojados por el utilitario:

Errores en Tiempo de Ejecución

1 `Uso: fgen [-b][-h][-t][-w][-?] archivo [archivos ...].`: Indica que hay un argumento no válido, o no se ha especificado al menos un archivo.

2 `La extensión del archivo 'nomarch' es ilegal.`: El archivo que contiene la definición del formulario en FDL, debe tener la extensión ".fm".

3 `Error de escritura en el archivo 'nomarch'.`

4 `No es posible abrir el archivo 'nomarch'.`: El usuario no tiene permiso de lectura sobre el archivo de sentencias FDL. Recurrir al comando chmod de UNIX.

5 `El nombre del archivo 'nomarch' es demasiado largo.`: El nombre del archivo, sin la extensión ".fm", debe tener a lo sumo 8 (ocho) caracteres. No ha sido utilizado.

6 `No se puede incluir parcialmente un campo multilinea o agrupado dentro de la zona de claves.`: No ha sido utilizado.

10 `La máscara 'máscara' no coincide con la longitud del campo.`

11 `El carácter 'c' es ilegal.`: Este error se producirá cuando en las zonas %form o %fields aparezca un carácter no válido.

12 `Carácter 'o' octal ilegal.`

13 `No existe mensaje asociado al valor del atributo 'in'.`: Uno o varios de los valores pertenecientes a un atributo in no tienen asociados un mensaje. (Ver *Atributos de Check* , Manual del Programador).

14 `El atributo 'mask' no tiene sentido en este tipo de campo.`: El atributo mask es válido exclusivamente en campos alfanuméricos.

15 `Existen n campos menos en la seccion 'fields' que en la imagen del form.`: En la sección %fields deben estar definidos todos los campos incluidos en la imagen de la pantalla.

16 `La especificación del campo no es correcta.`: Corrija este error antes que los demás.

17 `El valor 'valor' es incompatible con el tipo del campo.`: Algún atributo de check contiene un valor que no es compatible con el tipo del campo (p.ej., campo numérico y valor cadena de caracteres).

18 `El '[' extra es ignorado.`: Los corchetes ([ ]) que aparezcan dentro de los delimitadores del campo múltiple serán ignorados.

20 `Existen demasiados decimales en 's' para un campo numérico.`

21 `El valor: 'valor' del tipo DATE es inválido.`

22 `El valor: 'valor' del tipo TIME es inválido.`

23 `El valor 'valor' es demasiado largo para este campo.`

24 `Error de chequeo.`: El valor 'default' (valor) no verifica la condición.

25 `Existen más campos en la sección 'fields' que en la imagen del form.`: Verifique la concordancia entre las cantidades de campos definidos en cada caso.

26 `Error de sintaxis.`: Son casos típicos: sentencia de FDL mal escrita, falta el punto y coma (;) al final de una sentencia, carácter o palabra sin sentido, omisión de la coma separando atributos de un campo, etc.-

27 `El atributo 'subform' está redefinido.`

28 `El campo múltiple 'nombre' no tiene el atributo 'rows'.`: Este atributo es obligatorio para especificar la cantidad de filas que contendrá el campo múltiple.

29 `El atributo 'atributo' es inválido para el campo 'campo'.`

30 `La parte entera de 'campo' es demasiado larga para este campo.`

31 `El campo 'campo' está definido más de una vez.`

32 `El valor 'default' no verifica el atributo 'mask' (verifica hasta: %s <--).`

33 `Overflow de la tabla de símbolos.`: El máximo corriente es n.

34 `El símbolo 'símbolo' está indefinido.`

35 `La especificación de la base de datos debe ser el primer atributo.`: Cuando se asocia un campo del formulario con uno de la base de datos, debe indicarse como primer atributo de dicho campo.

36 `No es posible encontrar el campo 'esquema.tabla.campo'.`: El campo asociado de la base de datos no pertenece a la tabla indicada; o bien la tabla referenciada no pertenece al esquema, ya sea el corriente u otro especificado explícitamente.

37 `No es posible abrir el esquema 'esquema'.`: El esquema indicado en la sentencia use adolece de alguno de los siguientes problemas: no existe (nombre mal escrito); está siendo bloqueado por otro usuario para uso exclusivo; su índice está corrupto, o se carece de los permisos adecuados para abrirlo.

38 `El esquema 'esquema' no definido en la sentencia 'use'.`: Se está asociando un campo del formulario con un campo de un esquema no activo.

39 `El máximo número de esquemas excede el máximo (n)).`

40 `El símbolo 'símbolo' está redefinido.`

41 `El campo 'campo' en el atributo 'in' del campo 'campo' no está definido.`

42 `El campo de referencia para 'campo' es el mismo.`

43 `El campo de referencia para 'campo' es un campo múltiple.`

44 `La posición del campo de referencia para 'campo' es inválida.`

45 `El campo de referencia para 'campo' es inválido.`: Debe ser alfanumérico y 'skip' o 'display only'

46 `Los valores del atributo 'between' son inválidos.`: El primer valor debe ser menor que el segundo. Esto es válido para campos numéricos, alfanuméricos, y de tipo fecha u hora.

47 `El número de decimales en la base de datos es distinto.`: Existe una discrepancia entre el campo imagen del formulario y el campo de la base de datos al cual está asociado.

48 `La longitud del campo 'campo' no es correcta.`: Debe ser mayor o igual que la de la base de datos.: La definición de un campo en la imagen del formulario nunca puede implicar una longitud menor que la de su asociado en la base de datos.

49 `Existe incompatibilidad de tipos entre la base de datos y el campo del form.`: La definición del campo en la imagen del formulario es inadecuada para representar la información asociada en la base de datos.

50 `Hay un número excesivo de campos agrupados anidados.`: La cantidad de campos agrupados en la imagen del formulario excede el máximo.

51 `El atributo 'mask' está redefinido.`: Este error ocurre cuando se le asigna un atributo mask a un campo del formulario que está asociado con un campo de la base de datos, que a su vez tiene otro atributo mask.

52 `El número de campos es incorrecto para formar la clave (el número necesario es n).`: Revise la estructura de la clave para el atributo in table. (Ver *Capítulo 3 , Manual del Programador*).

53 `No se encuentra el índice 'índice' de la tabla 'tabla''.`: El índice especificado en el atributo in table no pertenece a la tabla indicada.

54 `El campo 'campo' no está definido.`

55 `No es posible encontrar la tabla 'esquema.tabla'.`: Es posible que la tabla indicada esté mal escrita o que no pertenezca al esquema especificado.

56 `Campo 'campo'.`: no puede ser a la vez NOT NULL y DISPLAY ONLY: La especificación de estos dos atributos simultáneamente no es compatible.

57 `El número de subforms es diferente al especificado en la cláusula 'in'.`: Cuando se asocian subformularios a un campo y éste a su vez tiene un atributo in, es necesario que haya una correspondencia entre los valores del atributo in y la cantidad de subformularios especificados. (Ver *Subformularios, Capítulo 3, Manual del Programador*).

58 `El subform está redefinido.`

59 `El valor 'valor' debe ser sin signo.`

60 `El string no está cerrado.`

61 `En el campo múltiple 'campo' el número de filas debe ser un múltiplo del atributo 'display'.`: El número de filas especificado en el atributo rows de un campo múltiple, debe ser un múltiplo del valor especificado en el atributo display del mismo campo.

62 `El campo 'campo' no está definido.`

63 `El atributo 'atributo' es para campos dentro de un 'múltiple|agrupado'.`

64 `El campo 'campo' no está dentro del campo agrupado especificado.`

65 `El campo 'campo' no es del tipo 'agrupado|multiple'.`

66 `No se permiten llamadas al mismo form.`: Al menos un campo de la clave debe ser NO-Skip.

67 `No existe un schema corriente (falta instrucción USE).`: NO es obligatorio colocar la sentencia use, pero en caso de querer asociar campos del formulario con campos de base de datos es necesario.

69 `Campo 'campo': cláusula IS con tipos incompatibles.`: Las valores o expresiones involucradas en un atributo is deben ser de idénticos tipos.

70 `Campo 'campo': Operando 'n' tipos incompatibles.`

71 `Campo 'campo': Debe ser numérico (operando 'n').`

72 `Campo 'campo': Debe estar dentro de un multirrenglón.`

73 `Campo 'campo': No tiene cláusula 'in'.`

74 `Campo 'campo': Nombre de tecla inválida 'nombre'.`: El nombre de tecla pasada como parámetro de la función help del atributo is no existe.

75 `Campo 'campo': Debe ser alfanumérico (operando 'n').`

76 `Lenguaje 'lenguaje' No Soportado.`: El lenguaje indicado en la sentencia language no está dentro de los soportados por IDEAFIX.

77 `La clausula IS es incompatible con clausula DEFAULT.`

78 `Demasiados campos definidos (maximo %d).`

79 `Nombre de identificador demasiado largo (max. %d).`

80 `La longitud de salida del campo es mayor que su longitud real.`

81 `Atributo invalido (LENGTH), el campo tiene relacion con la Base de Datos.`

82 `La longitud de la mascara no coincide con la del campo.`

83 `El atributo check digit solo puede aplicarse a campos numericos.`

84 `Error de Sintaxis en Check '%s'.`

85 `Error de Sintaxis en Default o Is '%s'.`

Mensajes de Advertencia

1 `La longitud del campo '%s' se ajusto de acuerdo al campo correspondiente de la base de datos.`

2 `Los decimales del campo '%s' se ajustaron de acuerdo al campo correspondiente de la base de datos.`

3 `Un campo de la base de datos esta siendo usado como campo clave en una cláusula 'in table'.`

4 `Comentarios no cerrados al final del archivo.`

5 `Redefinicion de 'check'. Se uso la ultima definicion.`

6 `El atributo 'not null' se ignoro en el campo %s (primero de un campo multiple).`

7 `Existe una redefinicion del valor 'default'.`

8 `La clausula "in tabla" requiere indice univoco. El indice usado (%s) no lo es. Reformule la validacion usando un indice univoco.`

9 `Caracter de separacion de check digit invalido ('-' o '/' unicamente). Se asume guion ('/').`

```
10 Redefinicion de 'is'. Se uso la ultima definicion.
```

GENCF

Generador de programa "C"

Sintaxis

```
gencf [-a] [-b] [-f] [-p] [-i] [-jcampoI=campoJ[,cam
poN=campoM]] [-o archivo] form
```

Descripción

El utilitario *gencf* permite generar un programa en "C" a partir de un formulario compilado, es decir que utilizará el archivo con extensión "fmo", obtenido con el utilitario *fgen*.

Se pueden generar dos tipos de programas:

• Listador. Cuando el formulario no tiene zona de claves.

• ADM (ver abajo). Cuando el formulario tiene una zona de claves.

Listadores

Los programas generados como listadores consisten solamente en la toma de datos del fomulario, y la captura de un comando (tecla de función) dada por el usuario. Si ésta es distinta de <PROCESAR> el proceso simplemente terminará. En caso contrario se ejecutará el código inserto en la parte del programa generado, que se indica como proceso central.

ABM (Altas, Bajas y Modificaciones)

Cuando el formulario posee zona de claves se intenta generar un programa de altas bajas y modificaciones. Es posible generarlos para formularios que trabajan con una o dos tablas, contemplando multirenglones y subformularios. Consultar el *Capítulo 3 del Manual del Programador* para más detalles sobre los casos manejados.

Opciones

-a Incluye rutina after field (ex postcond de PREFIX).

-b Incluye rutina before field (ex precond de PREFIX).

-f Fuerza la reescritura del archivo destino si éste existiera.

-g Genera las funciones con prototipos.

-i No imprime el mensaje de identificación.

-jcampoI=campoJ Esta opción indica que el programa trabajará con dos tablas re [,campoN=campoM] lacionadas entre sí mediante los campos indicados. De ellos, campoI y campoN pertenecen a una de las tablas; campoJ y campoM pertenecen a la otra. archivo.

-o archivo Genera el programa como 'archivo.c'. En caso de no informar la opción "-o", el nombre del programa generado es el mismo del formulario con extensión ".c".

Los campos de tipo SKIP o DISPLAY ONLY no darán lugar a entradas en los "switch" (programación CFIX) de las rutinas after-field y before-field. Las sentencias especificadas en el formulario que sean del tipo: ignore, add, delete, update, etc., no generarán la entrada correspondiente en el "switch" del cuerpo del programa.

Ejemplo

```
gencf -o libro.c libros
```

generará el siguiente archivo con nombre "libro.c":

```
#include <ideafix.h>
#include "libros.fmh"
#include "biblio.sch"
/* Funciones privadas */
private void Lectura();
/* Declaraciones globales */
form fm0;
/* Programa principal */
wcmd(libros, 1.2 30/04/90)
{
fm_cmd cmd;
fm0 = OpenForm("libros", FM_EABORT);
while ((cmd = DoForm(fm0, NULLFP, NULLFP)) != FM_EXIT)
switch (cmd) {
case FM_READ: Lectura(cmd, THIS_KEY);
break;
case FM_READ_NEXT: Lectura(cmd, NEXT_KEY);
break;
case FM_READ_PREV: Lectura(cmd, PREV_KEY); break;
case FM_ADD: InitRecord(LIBROS);
case FM_UPDATE: BeginTransaction();
FmToDb(fm0,0,CONTROL_FLD,0);
PutRecord(LIBROS);
EndTransaction();
break;
case FM_DELETE: BeginTransaction();
FmToDb(fm0, 0, CODIGO);
DelRecord(LIBROS);
EndTransaction();
break;
case FM_IGNORE: FreeTable(LIBROS);
break;
}
}
/* Rutina de lectura y pasaje a pantalla */
```

```
private void Lectura(cmd, mode)
fm_cmd cmd;
find_mode mode;
{
FmToDb(fm0, 0, CODIGO);
switch(GetRecord(LIBROSbyCODIGO,mode, IO_LOCK|IO_TEST))
{
case IO_LOCKED:
FmSetStatus(fm0, FM_LOCKED);
(void)FindRecord(LIBROSbyCODIGO,mode); DbToFm(fm0, 0, CODIGO);
FmShowFlds(fm0, 0, CODIGO);
return;
case ERROR:
FmSetStatus(fm0,cmd==FM_READ ? FM_NEW:FM_EOF);
return;
}
DbToFm(fm0, 0, CONTROL_FLD, 0);
FmShowFlds(fm0, 0, CODIGO);
}
```

Siendo "libros.fm", el siguiente:

```
Código del libro: _,___. |
Título del libro: ____________________________
Código del autor: _,___. ____________________
Número de edición: __.__
Fecha de la edición: __/__/__
%form
use biblio;
window label "libros", border standard;
%fields
codigo: libros. ;
titulo: libros. ;
autor: libros. ;
desc0: autores.nombre, skip;
edicion: libros. ;
fecha: libros., default today ;
```

Diagnósticos

La siguiente es la lista de mensajes de error que puede generar el utilitario:

0 `%s: Form invalido: %s.fm.`

1 `%s: No puede abrirse '%s' para escritura.`

2 `Campo '%s' debe ser vectorizado.`

3 `Campo '%s' debe tener igual dimension que el rector.`

4 `Campo '%s' no completa la clave de la pantalla.`

5 `Con esta pantalla debe usarse la opcion -j.`

6 `Campo '%s' no pertenece a tabla '%s'. a 12 : No han sido utilizados.`

13 `%s: '%s' ya existe (Opcion -f sobre escribe).`

14 `Aviso : No existe una clave en la base de datos que coincida con la del form.`

15 `El formulario no tiene zona de campos, caso no con templado.`

16 `El manejo automatico de multirenglones en subforms debe estar relacionado con un campo vectorizado.`

Consultar

GENFM

FGEN

GENFM

Generador de formularios

Sintaxis

```
genfm [-b] [-c] [-f] [-g] [-i indice] [-o archivo] [-l leng]
esquema[.]tabla.SE
```

Descripción

El utilitario genfm toma una tab la de un esquema existente, y genera una especificación de formularios en FDL (Form Definition Language) para realizar operaciones sobre dicha tabla. El archivo se genera con el nombre de la tabla especificada mediante en "esquema.tabla" y se le agregará la extensión ".fm", a menos que se use la opción "-o" para indicar en qué archivo se dejará la especificación generada.

Opciones

-b Mediante esta opción no se muestra el mensaje de identificación.

-f Dibuja los campos numéricos sin punto de miles.

-c Permite sobre-grabar el archivo destino si éste existiera.

-g Esta opción sirve para indicar que se procese el formulario generado con el utilitario *fgen*.

-i índice: Indica que se utilicen los campos que forman parte de la clave índice como claves del formulario que se está generando.

-o archivo: El formulario generado se grabará en "archivo.fm".

-l leng: Incluye en las especificaciones generales del formulario la cláusula language, con el

lenguaje indicado en leng.

Diagnósticos

1 `%s: Esquema invalido: %s`

2 `%s: Falta el nombre de la tabla.`

3 `%s: Tabla invalida: %s`

4 `%s: Indice '%s' inexistente. %s:`

5 `No puede abrirse '%s' para escritura.`

6 `Aviso: aun no contempla vectores (campo %s). no ha sido utilizado.`

8 `%s: '%s' ya existe (Opcion -f sobre escribe).`

9 `genfm: Campo '%s' de longitud %d dibujado de %d posiciones.`

Al ingresar datos podrá desplazarlo lateralmente para completarlo.

Consultar

FGEN

RGEN

Compila archivos fuente en RDL

Sintaxis

```
rgen [-b][-h][-w][-f] archivo[.rp] [archivo...]
```

Descripción

La especificación de un reporte está contenida en un archivo con extensión ".rp", conocido como archivo fuente RDL (Report Definition Language).

El utilitario *rgen* genera un archivo con estensión ".rpo" y en forma opcional uno con extensión ".rph", que será el archivo cabecera que deberá ser incluido en los programas "C" que utilicen el reporte.

Los mensajes de error son auto-explicativos. Recordar la posibilidad de que un error simple pueda desencadenar una cascada de errores espúreos; es decir, corregido el primer error los demás quedan subsanados. Por ello es importante prestar atención a los primeros mensajes de error cuando la lista es extensa.

Opciones

-b no se muestra el mensaje identificatorio del utilitario.

-*h* se genera un archivo cabecera

-w advertencias acerca del reporte que está siendo compilado.

-f se imprime la lista de los campos del reporte que no han sido declarados.

Diagnósticos

1 `Uso: rgen [-h][-w][-b][-f] archivo [archivos ...].`: Este mensaje aparecerá cuando se coloque un argumento no válido o cuando no se especifique, al menos, el nombre de algún archivo.

2 `La extensión del archivo 'nomarch'es ilegal.`: El archivo que contiene la definición del reporte en RDL, debe tener la extensión ".rp".

3 `Error de escritura en el archivo 'nomarch'.`

4 `No es posible abrir el archivo 'nomarch'.`: El usuario no tiene permiso de lectura del archivo que contiene las sentencias FDL. Para salvar este inconveniente debe recurrir al comando chmod (change mode) de UNIX.

5 `El nombre del archivo 'nomarch' es demasiado largo.`: El nombre del archivo, sin la extensión ".rp", debe tener a lo sumo 8 (ocho) caracteres.

7 `Texto ilegal. Sólo se admiten líneas en blanco en encabezado.`

8 `Campo no declarado: '%s'.`

9 `Declaración duplicada del campo '%s'.`

10 `La condición '%s' está redefinida.`

11 `La máscara '%s' no coincide con la longitud del campo.`: Verifique la especificación de la máscara

12 `El carácter '%c' es ilegal.`: Este error se producirá cuando en las zonas %report o %fields aparezca un carácter no válido.

13 `El carácter '%o' (octal) es ilegal.`

14 `La sentencia no es reconocida.`

15 `La redefinición del tipo para '%s' es incompatible.`

16 `Existen más argumentos que posiciones de campo en la zona '%s'.` Ambas cantidades deben coincidir. Ver mensaje Nro. 19.

17 `El tipo del campo '%s' es erroneo.`

18 `El campo '%s' no está definido.`

19 `Existen más posiciones de campo que argumentos en la zona '%s'.`

20 `La zona '%s' no está definida.`

21 `El campo '%s' es mostrado con diferente número de decimales.`

22 `El valor '%s' del tipo DATE es inválido.`

23 `El valor '%s' del tipo TIME es inválido.`

24 `La combinación de tipos en la expresión es erronea.`

25 `No es posible determinar el tipo de la expresión.`

26 `Redefinición de la zona '%s'.`

27 `Error de sintaxis.`: Son casos típicos: sentencia de RDL mal escrita, falta el punto y coma (;) al final de una sentencia, carácter o palabra sin sentido, omisión de la coma separando atributos de un campo, etc.-

28 `Las condiciones deben estar ordenadas. Todas las zonas 'before' deben estar antes que las zonas 'after'.`

29 `Se produjo un overflow de espacio de zona al crear la zona '%s'.`

30 `Se produjo un overflow de espacio de campo al crear el campo '%s'.`

31 `El campo '%s' no es usado.`: Se definió un campo en la sección %fields que no fue utilizado en ninguna de las zonas.

32 `El atributo 'mask' sólo es válido para campos alfanuméricos.`

33 `La longitud de la máscara es superior a la longitud del campo.`

34 `Solamente se permite una sentencia 'use'.`

35 `El atributo 'mask' se ha redefinido.` (Puede haber sido heredado de la base de datos).

36 `La especificación de la base de datos debe ser el primer atributo.`: Cuando se asocia un campo del formulario con uno de la base de datos, ello debe indicarse en primer término en la lista de atributos de dicho campo.

37 `No es posible encontrar el campo '%s.%s.%s'.`: El campo asociado de la base de datos no pertenece a la tabla indicada; o bien la tabla referenciada no pertenece al esquema, ya sea el corriente u otro especificado explícitamente.

38 `No es posible abrir el esquema '%s'.`

39 `El esquema '%s' no esta definido en la sentencia 'use'.`: Se está asociando un campo del formulario con otro campo perteneciente a un esquema no activo.

40 `El máximo número de esquemas excede el máximo (%d).`

41 `El tipo del campo de reporte es incompatible con el tipo del campo de la base de datos.`: La definición del campo en la zona del reporte no es apropiada para representar el formato del campo asociado de la base de datos.

42 `Es diferente el número de decimales en la base de datos que en el reporte.`

43 `El string es demasiado largo.`

44 `Overflow de la tabla de valores.`

45 `Lenguaje '%s' No soportado.`

46 `Nombre demasiado largo (max. %d).`

47 `Atributo 'check digit' solo aplicable a campos numericos.`

Mensajes de Advertencia

1 `Diferente numero de decimales en el campo de base de datos.`

2 `Distinta longitud del campo en base de datos.`

3 `Comentarios no cerrados al final del archivo.`

4 `Mascara '%s' no coincide con longitud del campo.`

5 `Se utilizo el orden de entrada de campos en ausencia.`

TAR

Utilitario de almacenamiento.

Sintaxis

```
tar {-D|+data-base} esquema.[tabla] [=nomarch]
```

Descripción

Esta herramienta es, en esencia, el utilitario tar de GNU, el cual ha sido modificado para aceptar la opción `-D` o `+basedatos`. Con esta opción, es posible importar y exportar esquemas seleccionados y tablas, directamente desde un archivo *.tar*. Los archivos pueden ser comprimidos usando la opción de *tar* **: -z**.

Si el *tar* llama a los utilitarios *exp* o *imp*, es posible pasarles parámetros usando la variable de ambiente DBTAR.

Ejemplo

```
#tar cDf /dev/fd0 auus.astos impuesto.ganan aurus.emps
```

Exporta tres tablas.

```
#tar -c +basedatos -f /dev/fd0 aurus impuesto aurus
```

Exporta tres esquemas

```
#tar xDf /dev/fd0 aurus.astos impuesto.ganan aurus.emps
```

Importa tres tablas.

```
#tar -xf /dev/fd0 +basedatos aurus impuestos aurus
```

Importa tres esquemas

```
#tar xDf /dev/fd0
```

Importa todas las tablas o esquemas.

```
#tar xDf /dev/fd0 aurus.dat=aurus.astos xx.zz=taxes.win
```

Importa aurus.dat como aurus.astos y xx.zz como taxes.win.

Además, este utilitario puede correr en forma remota. Esto significa que el comando puede ser ejecutado desde un server usando los datos de otro. Por ejemplo:

```
#tar cDf idefix:/dev/fd0 aurus impuestos
```

Exporta dos esquemas al dispositivo de host *idefix*.

TESTFORM

Prueba la especificación de un form.

Sintaxis

```
testform form
```

Descripción

Este utilitario permite la prueba de un diseño de formulario, es decir, se obtiene un prototipo del comportamiento de la pantalla, de acuerdo a lo especificado en la definición del formulario.

No se pueden consultar datos existentes, ni como es lógico, actualizar las tablas asociadas al formulario.

Al utilizar un formulario con testform se aplican todas las validaciones indicadas en la definición del formulario, aún aquellas que impliquen consultar una tabla de base de datos (mediante el atributo "in table" especificado en los campos del formulario).

Capítulo 20

Interfaz de Programación

Este capítulo ofrece una visión general de la interfaz de IDEAFIX con el Lenguaje de programación "C", a la cual se la denominará genéricamente CFIX.

Se exponen los conceptos básicos sobre el manejo de Base de Datos, Formularios y Reportes desde el Lenguaje "C".

Introducción

La interfaz de programación se realiza básicamente a través de una Biblioteca, que provee rutinas en lenguaje "C" para acceder a las facilidades de IDEAFIX. Entre ellas encontramos las funciones necesarias para manejar bases de datos, formularios, reportes, y otras rutinas de propósitos generales para programar en CFIX.

CFIX es el nombre dado en IDEAFIX a la programación en "C", y que constituye un *super-set* de dicho lenguaje, ya que al programar en CFIX se prescinde de muchas de las asperezas de "C", y se dispone de Macrodefiniciones de alto nivel y un repertorio de funciones que facilitan la tarea de programación.

Es importante aclarar que cuando se programa en CFIX y se utilizan Bases de Datos, formularios o reportes, se aprovechan **todas** las características de IDEAFIX que se utilizaron al diseñar éstos.

Pongamos un ejemplo. Si se diseña un formulario de ingreso de datos, que en uno de sus campos requiere una validación que está fuera del alcance del lenguaje de definición de formularios de IDEAFIX (FDL), se recurre a programar en "C" *solamente dicha validación*. Todas las otras indicaciones, validaciones y atributos del formulario siguen siendo manejadas automáticamente sin que el programador deba ocuparse de ellas en absoluto. El programa puede ser generado automáticamente a partir de un formulario mediante el utilitario **gencf** (generar CFIX). Luego, sobre el programa generado se agrega la modificación en cuestión.

La secuencia típica para desarrollar un programa interactivo en IDEAFIX es:

- Generar automáticamente el formulario (**genfm**), modificando algún detalle si fuera necesario.
- Generar automáticamente el programa CFIX (**gencf**) a partir del formulario.
- Agregar validaciones u opciones sobre el programa sólo para los casos no contemplados ya en el diseño del formulario.

Básicamente para programar en CFIX se necesita:

- La Biblioteca de funciones de IDEAFIX.
- Archivos de encabezamiento (header files) con definiciones.

La Biblioteca C de IDEAFIX

La biblioteca de funciones está copiada en uno de los directorios estándar de Unix para contener bibliotecas de programación. El nombre de este directorio y de la biblioteca puede variar de acuerdo al sistema operativo. Mostramos aquí algunos de los nombres más comunes:

UNIX-AIX: `/usr/lib/libidea.a`. Si su sistema soporta shared libraries encontrará también: `/usr/lib/libidea_s.a`

XENIX 386: `/lib/386/Slibidea.a`

MS-Windows (en el directorio bajo el directorio sobre el cual este instalado IDEAFIX): `idea.lib`

Para facilitar el manejo de las rutinas, la biblioteca está dividida en grupos de funciones según la tarea que realizan. Los grupos son los siguientes:

DB Interfaz con las Bases de Datos. Permiten acceder a la información en tablas, realizar búsquedas por índices, agregar registros y otras.

FM Interfaz con formularios. Permiten utilizar formularios diseñado con FDL para capturar datos.

RP Es la interfaz con los reportes. Permiten utilizar reportes diseñados con RDL.

GN Son rutinas de tipo general: manejo de errores, despliegue de mensajes en la pantalla, y otras.

TM Estas funciones permiten manejar fechas y horas en forma to IDEAFIX.

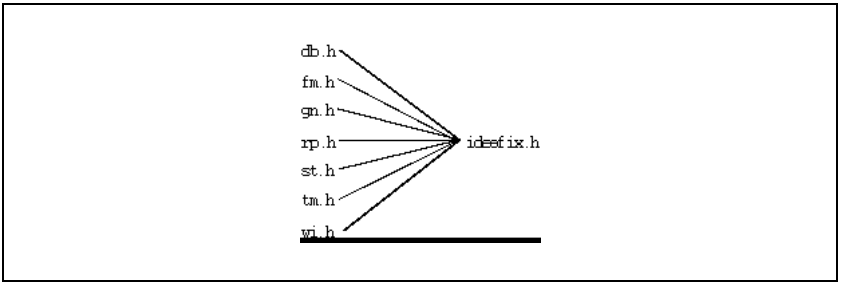
ST Ejecución de operaciones con cadenas de carácteres (strings).

WM Funciones de manejo de ventanas y otras tareas del Win dow Manager.

NM Funciones de manejo del tipo NUM.

Los Archivos de Encabezamiento

Existen ciertos archivos que los programas de aplicación necesitan incluir, para obtener la definición de constantes y tipos de datos que IDEAFIX agrega al Lenguaje "C". Para facilitar el manejo de las rutinas, la biblioteca está dividida en grupos de funciones. Cada grupo dentro de la biblioteca tiene su propio archivo de encabezamiento; ellos son:



Estos archivos están almacenados en el directorio include donde se haya instalado IDEAFIX. Un archivo de encabezamiento (header file) llamado ideafix.h incluye todos estos archivos automáticamente, como se verá en el ejemplo que se presenta más adelante.

La Función *wcmd*

La función wcmd (Window Manager Command) es para CFIX equivalente al main del Lenguaje "C". Realiza una serie de tareas necesarias para el inicio de la ejecución de un programa, como comenzar el diálogo con el Manejador de Ventanas (Window Manager).

Todo programa en CFIX que desee realizar entrada/salida por terminal mediante ventanas y usando la terminal virtual de IDEAFIX, debería utilizar esta función. Además, wcmd terminará el programa correctamente, es decir, cuando el control llegue a la llave que termina la función, se avisará al Manejador de Ventanas que terminó la sesión, y se realizarán todas las tareas relativas a la terminación de un programa, como cerrar y bajar a disco todos los cambios realizados en tablas de base de datos

Veamos un programa muy elemental, llamado hello.c, que consiste en el siguiente código:

```
#include "ideafix.h"
wcmd(hello, %I% %G%) {
WiMsg("Hola, mundo\n");
}
```

La línea con la sentencia #include "ideafix.h" indica la inclusión del archivo estándar de encabezamiento de IDEAFIX.

La macro-función wcmd debe ser llamada con dos argumentos que son simplemente texto,

pero que NO deben ser encerrados entre comillas. El primer texto es el nombre del programa y debe ser una sola palabra sin contener caracteres de puntuación ni otros símbolos, sólo letras y números (debe comenzar con una letra). Se utiliza para informar al WM el nombre del programa. Entre otras cosas el WM lo usará para ubicar el archivo de ayuda que desplegará cuando el usuario pida ayuda con la tecla; dicho archivo tendrá igual nombre que el programa seguido por la extensión .hlp.

El segundo argumento es la identificación de la versión del programa. Será desplegado en la etiqueta de la ventana de ayuda mencionada en el párrafo anterior, para identificar rápidamente qué versión de un programa está en ejecución. Este segundo parámetro puede consistir en varias palabras.

Usualmente los parámetros de wcmd se indican mediante algún sistema de control de versión. En Unix existe un sistema denominado SCCS (Source Code Control System, o Sistema de Control de Código Fuente) que permite controlar la evolución y los cambios que se realizan sobre todo tipo de programas fuente; las letras entre símbolos "porciento" responden a la codificación del SCCS.

El SCCS permite colocar ciertas indicaciones dentro del código fuente a controlar, que serán reemplazadas automáticamente. En nuestro ejemplo los parámetros del wcmd, al ser procesados por el SCCS, serán reemplazados por:

```
#include <ideafix.h>
wcmd(hola, 1.1 10/10/89) {
WiMsg("Hola, mundo\n");
}
```

La rutina WiMsg (Window Message) permite mostrar un mensaje en la pantalla, conteniéndolo en una ventana. La forma de utilizarla es similar a printf.

Cuando se ejecute el programa, se verá en la terminal una ventana con el mensaje, y se esperará una tecla para finalizar.

```
$ hola.exe
```

Figura 7.1 - El Programa "hola"

Cómo Compilar un Programa

Para compilar un programa en CFIX se debe contar con la biblioteca de IDEAFIX, los archivos de encabezamiento y el compilador "C" para su equipo. La forma más fácil de compilar es mediante el utilitario cfix. En Unix por ejemplo, se podría compilar un programa con el comando:

```
$cfix hello.c
```

Este comando arroja el siguiente mensaje:

```
cc -c -I/usr2/ideafix/include -O hello.c
cc -s hello.o -o hello.exe -lidea
```

Nótese que el directorio */usr/ideafix/include* es el contenido de la variable de ambiente IDEAFIX más la palabra "include", e indica al compilador dónde buscar los archivos de encabezado de IDEAFIX. El utilitario cfix invoca al comando cc de Unix quien a su vez invoca automáticamente al compilador y vinculador, y en este caso compilará y encadenará el programa fuente hello.c, dejando el resultado en un programa ejecutable llamado hello.exe. Si se quiere suprimir la extensión .exe, usar la opción -e del comando cfix. (Para más detalles, consultar el Capítulo 6).

La indicación -lidea indica al compilador que utilice la biblioteca de IDEAFIX. Como está copiada sobre un directorio estándar, el compilador no tendrá problemas en ubicarla.

Bases de Datos, Formularios y Reportes

La filosofía de manejo de bases de datos, reportes y formularios mediante la Biblioteca "C" de IDEAFIX permite aislar el programa de dependencias de diseño de estos elementos, como por ejemplo el tipo de dato de un campo, el orden de estos, etc.

Ya se ha remarcado que cuando se programa en CFIX y se utilizan Bases de Datos, formularios o reportes, se aprovechan todas las características asignadas al diseñarlas. Esto significa que si se necesita hacer una validación que está fuera del alcance del lenguaje de definición de formularios de IDEAFIX (FDL), se recurre a programar solamente dicha validación. Todas las demás indicaciones siguen siendo manejadas en forma automática. Lo dicho es también válido para los reportes y las Bases de Datos.

Los tres conjuntos de funciones (formularios, reportes y bases de datos) comparten una filosofía común, que es manejar un ``buffer" interno que contiene información. Este buffer sólo es accesible al programador mediante las funciones apropiadas. De este modo se evita el contacto con datos en forma "cruda", lo cual reduce el riesgo de errores.

Cada objeto tiene su propio buffer, es decir que cada formulario dispone de un área de trabajo exclusiva, al igual que cada reporte y cada tabla de base de datos usada por un programa. Como ya queda implícitamente expresado, un buffer es una sección de memoria reservada a un fin específico, en la cual se almacena:

1. Para una base de datos, la imagen de un registro de una tabla. Cada tabla tiene su propio

buffer.

2. Para un formulario, los valores de todos los campos que conforman ese formulario.

3. Para un reporte, se registran los valores de todos los campos contenidos en él.

Existen funciones que permiten volcar el contenido del buffer al medio físico adecuado:

1. Grabar un buffer de una tabla de base de datos en disco, con la consecuente actualización de índices.

2. Trasladar los valores de los campos en un buffer de formulario a la representación del formulario en pantalla.

3. Pasar los valores de los campos de una línea de detalle de un reporte, a la impresora.

IMPORTANTE: Cuando se trabaja con esquemas, formularios y reportes simultáneamente, o con alguna combinación de ellos, puede suceder que en el momento de la compilación se genere una advertencia similar a la siguiente:

```
./esquema.sch: 16: CAMPO redefined
```

o bien:

```
./pantalla.fmh: 16: CAMPO redefined
./reporte.rph: 16: CAMPO redefined
```

Esto significa que se está usando simultáneamente el nombre CAMPO para definir una tabla del esquema y el campo de un formulario o reporte; o bien asignando el mismo nombre a dos campos diferentes.

Este error debe ser corregido, ya que quedará como válida la última definición y cuando hagamos referencia a la tabla o campo definido anteriormente, obtendremos un valor indeseado.

Relaciones entre los Distintos Objetos

Los campos de formularios y reportes pueden relacionarse con campos de base de datos, a través de referencias en los lenguajes FDL y RDL. Esta relación tiene dos implicancias:

1. Herencia de atributos: Los campos de reporte o formulario "heredan" los atributos del campo de base de datos. Ellos incluyen las validaciones (atributos de check), el valor por defecto (default) y las máscaras.

2. Se crea un "puente" entre los buffers de formulario (o reporte) y la tabla de base de datos a la que pertenezca el campo con el cual se relacionó. Este "puente" permite pasar fácilmente datos entre un buffer y otro. (Ver Figura 7.2.)

Para que estos efectos se produzcan, el campo del formulario o reporte debe haber sido explícitamente relacionado con algún campo de base de datos. Desde un programa, el acceso al buffer que contiene los valores de los campos se realiza a través de funciones. Por ejemplo,

en el caso de la base de datos, para tomar un campo de tipo entero (int en "C") se usa la función IFld. Esta retorna en formato entero el valor almacenado el buffer del campo.

Hay una función para cada tipo de dato en que se quiera tomar el valor del campo. Si el formato con que el dato es pedido no es coherente con el tipo de dato del campo de base de datos, la función desplegará un mensaje de error.

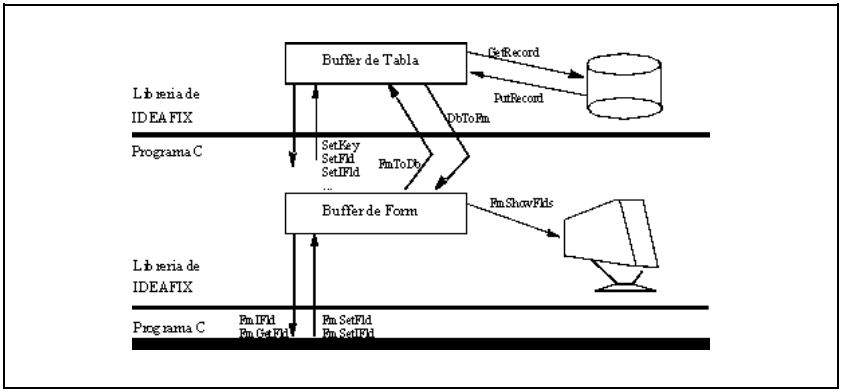


Figura 7.2 - Manejo de Buffers

Cuando un buffer se ha modificado, y se desea actualizar sobre el dispositivo físico, se usa también una función. En el caso de las Bases de Datos, la función *PutRecord* actualiza en disco el contenido del buffer de una tabla. Cuando se trata de un formulario, la función *FmShowFlds* actualiza el valor de cada campo de pantalla con el correspondiente contenido del buffer. La figura precedente ilustra gráficamente estas relaciones entre buffers.

Una capacidad muy importante que provee la biblioteca de funciones es la de permitir el pasaje de valores entre campos relacionados. Por ejemplo, si hay campos de un formulario relacionados con campos de una tabla de base de datos, la función *DbToFm* copia desde el buffer de la tabla hacia el buffer de formulario, para el rango de campos que se indique. La función *FmToDb* realiza el procedimiento contrario. Es posible relacionar en un mismo formulario distintos campos con distintas tablas. Igual consideración vale para los reportes, pero en este caso sólo existe la función *DbToRp*, que copia desde el buffer de la tabla hacia el buffer del reporte.

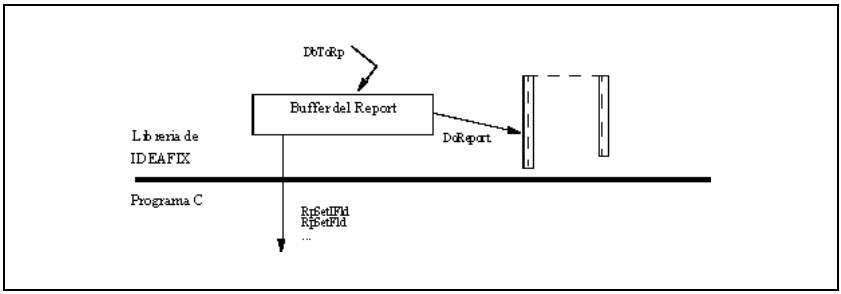


Figura 7.3 - Buffer de Reporte

Punto Fijo en Campos Numéricos

Los campos numéricos con punto decimal definidos en esquema, formularios y reportes se guardan internamente como campos enteros. Por ejemplo, si el campo peso está definido en la base de datos como:

```
table items {
. . . .
peso num(8,4),
. . . .
}
```

se considerarán los valores almacenados como de tipo long.

En el caso de manejar valores numéricos desde un programa en CFIX y el pasaje de datos se realiza entre los buffers de las distintas entidades, la operación es transparante para el usuario.

Si se asignan valores del buffer de la base de datos o de un formulario a una variable de programa, se ignorará el punto decimal. Esto quiere decir que si en nuestro ejemplo, en el campo peso tenemos almacenado el valor 120,0000 (ciento veinte), al asignarlo a una variable de tipo long o double el valor que obtendremos será 1.200.000 (un millón doscientos mil). Utilizando algunas de las funciones de biblioteca de IDEAFIX consideremos el caso recién planteado:

```
wcmd(prog, %I% %G%) {
/* Obtención del registro */
WiMsg("1: %ld", LFld(ITEMS_PESO));
WiMsg("2: %8.4f", FFld(ITEMS_PESO)/10000);
WiMsg("3: %8.4f", LFld(ITEMS_PESO)/10000);
WiMsg("4: %8.4f", LFld(ITEMS_PESO)/10000.0);
}
```

Veamos las salidas por pantalla que se obtienen para las distintas alternativas planteadas. Para la primera de ellas:

En este caso se obtiene el valor del buffer de la base de datos mediante la función de biblioteca LFld (retorna un valor de tipo long) y se la imprime en pantalla con la función WiMsg. Como se trata al dato como si fuera de tipo long, se ignora el punto decimal.

Veamos ahora qué ocurre con la segunda opción.

Se obtiene el valor con la función FFld, que retorna un valor numérico con punto flotante, y se lo divide por una constante entera que desplaza la coma decimal a la posición deseada. Con la función de biblioteca WiMsg se lo muestra en pantalla especificándose que el el valor está formado por 8 (ocho) dígitos de los cuales 4 (cuatro) son decimales. Hemos obtenido en consecuencia un despliegue correcto.

En el tercer caso, en cambio:

Esta vez hemos obtenido el valor del buffer con la función *LFld*, que retorna un long, y se lo divide por una constante entera. En la función *WiMsg* se especifica que el valor pasado como parámetro será un número tipo float de 8 (ocho) dígitos de los cuales 4 (cuatro) son decimales. El resultado de la operación realizada es un valor de tipo entero (por lo tanto no del tipo esperado por la función *WiMsg*), y como resultado obtenemos un valor cero.

Finalmente, la cuarta opción produce:

Este último caso, como el segundo, retorna el valor deseado. Se pasa como segundo argumento de la función *WiMsg* un valor del tipo especificado en el primer argumento. La función *LFld* retorna un valor entero tipo long, pero al ser dividido por una constante con punto decimal, el resultado pasa a ser de tipo float.

Ejemplo

Continuando con nuestro ejemplo de la biblioteca, analizaremos el fuente en CFIX que se genera al aplicar el utilitario **gencf** al formulario libros, previamente creado mediante el utilitario **genfm**. La forma de invocar al utilitario gencf para generar este archivo fuente, es la siguiente:

```
$ gencf -a -b libros
```

y el resultado quedará en un archivo llamado libros.c ya que no redireccionamos la salida. Las opciones **-a** y **-b** se indican para que se incluyan las funciones *after* y *before* respectivamente. Estas funciones son utilizadas para realizar validaciones y otras operaciones imposibles de efectuar desde el formulario en forma automática.

El código a analizar se detalla seguidamente.

```
#include <ideafix.h>
#include "libros.fmh"
#include "biblio.sch"
/* Funciones privadas */
private fm_status before(form, fmfield, int);
private fm_status after(form, fmfield, int);
private void Lectura(fm_cmd, find_mode);
```

```
/* Declaraciones globales */
form fm0;
/* Programa principal */
wcmd(ejemplo, %I% %G%)
{
fm_cmd cmd;
fm0 = OpenForm("libros", FM_EABORT);
while ((cmd=DoForm(fm0, before, after)) != FM_EXIT)
switch (cmd) {
case FM_READ: /* Si hay que leer, buscar el registro */
Lectura(cmd, THIS_KEY);
break;
case FM_READ_NEXT:
Lectura(cmd, NEXT_KEY);
break;
case FM_READ_PREV:
Lectura(cmd, PREV_KEY);
break;
case FM_ADD: /* Alta: preparar registro en blanco */
InitRecord(LIBROS);
case FM_UPDATE:/* Alta/Actualizac.:grabar registro */
BeginTransaction();
FmToDb(fm0,0, CONTROL_FLD, 0);
PutRecord(LIBROS);
EndTransaction();
break;
case FM_DELETE: /* Baja: borrar el registro */
BeginTransaction();
FmToDb(fm0, 0, CODIGO);
DelRecord(LIBROS);
EndTransaction();
break;
case FM_IGNORE: /* Liberar contenido
del buffer */
FreeTable(LIBROS); break;
}
}
/* Rutina de lectura y pasaje a pantalla */
private void Lectura(fm_cmd cmd, find_mode mode) {
FmToDb(fm0, 0, CODIGO);
switch(GetRecord(LIBROSbyCODIGO,mode ,IO_LOCK|IO_TEST)) {
case IO_LOCKED:
FmSetStatus(fm0, FM_LOCKED);
(void)FindRecord(LIBROSbyCODIGO, mode);
DbToFm(fm0, 0, CODIGO);
FmShowFlds(fm0, 0, CODIGO);
return;
case ERROR:
FmSetStatus(fm0, cmd==FM_READ ? FM_NEW : FM_EOF);
return;
}
DbToFm(fm0, 0, CONTROL_FLD, 0);
FmShowFlds(fm0, 0, CODIGO);
}
/* Rutina que se ejecuta antes de entrar a cada campo */
private fm_status before(form fm, fmfield fno, int row)
{ switch (fno) {
case CODIGO:
break;
case TITULO:
break;
case AUTOR:
```

```
   break;
   case EDICION:
   break;
   case FECHA:
   break;
   }
   return FM_OK;
   }
```

En primer término se incluyen los archivos de encabezado de IDEAFIX y los correspondientes al formulario y al esquema. Estos archivos contienen las definiciones de constantes simbólicas necesarias en el momento de la compilación.

Antes de comenzar el programa con la función *wcmd*, hay una declaración de funciones y variables globales que se describen más adelante.

Dentro del cuerpo principal del programa hay una nueva declaración de variables, denominadas locales, que podrán ser utilizadas exclusivamente por el mismo. Es necesario destacar que se pueden definir variables locales en cualquiera de las funciones del programa. En nuestro caso se declara una variable de tipo fm_cmd la cual tomará los valores de retorno de la función *DoForm*.

Para poder utilizar un formulario desde un programa en CFIX, es necesario abrirlo previamente. Para ello, la biblioteca de funciones de IDEAFIX cuenta con la función *OpenForm* que es la que se encarga de abrir el formulario que le indiquemos. Esta función devuelve un descriptor que se asigna a una variable de tipo form (en nuestro ejemplo fm0, variable global).

Una vez abierto el formulario la función *DoForm* lo ejecuta. Esta función es similar al utilitario **doform**, con la salvedad de que la primera simplemente devuelve los códigos de las teclas de función, cuando el usuario las digita, sin resolver en definitiva las operaciones.

El programa se mantiene en un ciclo (while) que no se interrumpe hasta que el usuario digite la tecla de función <FIN>.

Los códigos de teclas de función que pueden ser devueltos por el utilitario *DoForm*, están contemplados en la sentencia switch. Según sea su valor se ejecutan las operaciones asociadas con cada alternativa. En nuestro ejemplo las tres primeras opciones (FM_READ, FM_READ_PREV, FM_READ_NEXT) invocan a la función Lectura, pero cada una con diferencia en sus parámetros.

Luego se contempla el caso en que se quiera dar de alta un registro (FM_ADD). Para ello se inicializa a valores por defecto (si existen) el buffer de la tabla de la Base de Datos. En este caso no se interrumpe el switch, porque las sentencias que siguen son comunes tanto para las altas como para las modificaciones (FM_UPDATE). Se copian los valores del buffer del formulario al buffer de la Base de Datos (*FmToDb*) y se agrega el registro mediante la función *PutRecord*.

En el caso de las bajas (FM_DELETE) se pasan los valores de la clave desde el buffer del formulario al buffer de la Base de Datos. Luego se ejecuta la función *DelRecord* que elimina

el registro con la clave indicada. Por último tenemos la alternativa FM_IGNORE que ignora toda modificación que se le haya hecho al registro accedido y libera todos los registros de la tabla utilizada (FreeTable).

La función *Lectura*, engloba las funciones necesarias para realizar el acceso a la Base de Datos. En primer lugar, mediante la función *FmToDb*, se copian los valores del buffer de la pantalla al buffer de la Base de Datos. Esto es necesario para tener la clave por la cual se va a acceder a los registros de la tabla en cuestión. La función de biblioteca *GetRecord*, es la que se encarga de efectivizar la recuperación de los registros, devolviendo un código de estado del registro accedido. Una vez recuperado un registro es preciso copiar los valores del buffer de la Base de Datos al buffer del formulario, operación que se realiza por medio de la función *DbToFm*. Para poder visualizar los resultados que se hallan en el buffer del formulario, se debe invocar la función *FmShowFlds*.

Las funciones *before* y *after* incluirán, **exclusivamente**, todas aquellas validaciones y operaciones que no puedan realizarse en forma automática desde el formulario. Estas funciones devuelven al *DoForm* el estado del formulario (FM_OK, FM_ERROR, FM_SKIP, FM_REDO, etc.), por ello se declaran del tipo fm_status. Al ser generadas por el utilitario gencf, incluyen todos los campos del formulario, de forma tal que el usuario pueda introducir validaciones u otras operaciones a ser realizadas antes de entrar o después de salir de los mismos, según sea la función. No es necesario contemplar en estas funciones aquellos campos que no requieran un tratamiento especial, razón por la cual se devuelve el estado FM_OK al final de las mismas.

Este programa fuente puede ser compilado con el utilitario **cfix** (presentado en este capítulo) obteniéndose un programa ejecutable. Si no se introducen validaciones y/u operaciones adicionales, este último nos brindará las mismas prestaciones que el utilitario **doform**. Por esta razón todo lo nuevo que se le agregue al programa, debería tender a perfeccionar los resultados ya obtenidos en la forma automática, sin tener que retroceder al punto de partida.

Para una mayor referencia de las funciones aquí presentadas, consultar *Funciones de Biblioteca*, en este libro.

3

Manual Avanzado

Capítulo 21

Biblioteca

El capítulo comienza con una introducción a los distintos aspectos de la interfaz de programación de IDEAFIX y luego se presenta un compendio de las funciones de biblioteca. Cada función esta documentada en una página que muestra los parámetros que recibe, el valor retornado y la descripción de la tarea que realiza.

En algunos casos se incluyen notas especiales, diagnósticos y ejemplos aclaratorios.

Introducción

Este capítulo está organizado como un Manual de Referencia de las funciones que componen la Interfaz con el Lenguaje "C" de IDEAFIX. Estas funciones están a disposición del programador para implementar programas de aplicación en CFIX.

CFIX es el nombre dado en IDEAFIX a la programación en "C", y que constituye un super-set de dicho lenguaje, ya que al programar en CFIX se prescinde de muchas de las asperezas de "C", y se dispone de macrodefiniciones de alto nivel que facilitan la tarea de programación, que son consideradas como un super-grupo del lenguaje C.

En este capítulo, cada función tiene su explicación al estilo de una página del manual de UNIX, a pesar de que en más de un caso, una página puede contener más de una función documentada, en particular cuando una función es similar a otra, o cuando se encuentran íntimamente relacionadas. Para facilitar la búsqueda, las funciones están divididas en grupos. A cada grupo se lo nombra con dos letras definiendo la siguiente tabla:

Grupo de Funciones	Abreviatura
Base de Datos	DB
Formularios	FM
Generales	GN
Reportes	RP
Cadenas de Caracteres	ST
Fecha y Hora	TM
Ventanas	WI
Manejo del tipo NUM	NM

Figura 8.1 - Grupos de Funciones

Dentro de cada sección se ordenan las páginas alfabéticamente.

El formato de cada página es el mismo para todas: el detalle de los parámetros de las funciones (en el formato aceptado el ANSI estándar de "C"), *Valor Retornado*, y una *Descripción* de la función. En algunos casos, se detallan *Ejemplos*, *Notas* o información de *Diagnóstico*. Generalmente, las secciones de función terminan con una subsección*Conssultar*, detallando otras secciones o manuales para buscar información detallada.

Archivos de Inclusión

Existen algunos *archivos de cabecera* que los programas de aplicación necesitan incluir, para obtener la definición de constantes y tipos de datos. Cada grupo de la biblioteca tiene su propio archivo ".h", que son:

```
db.h -----|
fm.h -----|
gn.h -----|
rp.h -----|---------- ideafix.h
st.h -----|
tm.h -----|
wi.h -----|
```

Estos archivos están almacenados en el directorio *include* donde se ha instalado IDEAFIX. Un archivo de encabezamiento (header file) llamado ideafix.h incluye todos estos archivos automáticamente.

Manejadores de Error

Es muy común que cuando se detecta un error, se llame a una función "manejadora" del mismo (terminología proveniente del inglés: error handler). Esto permite una gran flexibilidad en el manejo de errores. En todos los casos existe una acción por omisión, y una función específica que permite al programador cambiar a una rutina propia. Los manejadores

existentes son:

- Manejador de conversión de campo, ver SetConvHandler(GN).
- Manejador de ubicación en memoria, ver SetAllocHandler(GN).
- Manejador de error de Entrada Salida, ver SetDbHandler(DB).
- Lectura de una variable de ambiente, ver SetReadEnvHandler(GN).

Nota para Argumentos Variables

Las funciones que aceptan argumentos variables son usualmente aquellas que imprimirán algo, como ser printf, fprintf, Warning o WiMsg. Todas estas funciones aceptan que se les pase varios tipos de IDEAFIX.

También es posible imprimir la descripción de teclas del Window Manager, como por ejemplo:

```
WiMsg("Presione %K_META para ver el menu");
```

Este llamado dará como salida un mensaje usando la tecla correcta perteneciente a la terminal activa. Suponiendo que la tecla sea *Home*, mostrará:

```
Presione Home para ver el menu.
```

Además, los atributos comunes de IDEAFIX para formateo, (como %4.2%K_META) también funcionarán, produciendo el resultado correcto.

Constantes de IDEAFIX

IDEAFIX tiene predifinido un conjunto de constantes útiles en el manejo de Base de Datos, Formularios, Reportes, etc., que se detalla a continuación. El resto de las constantes predefinidas de IDEAFIX se presenta junto con las funciones que hacen uso de ellas.

Constante	Significado
NULL_BYTE	Valor "null" para numéricos de 1 o 2 dígitos o alfanuméricos de 1 posición.
NULL_SHORT	Valor "null" para numéricos de 3 o 4 dígitos.
NULL_LONG	Valor "null" para numéricos de 5 a 9 dígitos.
NULL_DOUBLE	Valor "null" para numéricos de más de 9 dígitos.
NULL_STR	Valor "null" para cadenas de caracteres.
NULL_DATE	Valor "null" para fechas.
NULL_TIME	Valor "null" para horas.

HIGH_VALUE	Máximo valor para una cadena de caracteres.
LOW_VALUE	Mínimo valor para una cadena de caracteres.

Figura 8.2 - Constantes de IDEAFIX

Sumario de Tipos de IDEAFIX

schema DB descriptor de esquema.

dbtable DB descriptor de tabla.

dbfield DB descriptor de campo de la base de datos.

dbindex DB descriptor de índice.

find_mode DB modos de búsqueda para la función GetRecord.

form FM descriptor de form.

fmfield FM dscriptor de campo de form.

fm_cmd FM Comandos retornados por DoForm y DoSubForm.

fm_status FM estatus del form.

report RP descriptor de reporte.

rpfield RP descriptor de campo de reporte.

Output RP descriptor de grilla de salida.

window WI descriptores de ventanas.

attr_type WI atributo de ventana.

DATE TM fecha.

TIME TM hora.

SUM NM numérico (hasta 28 dígitos).

Capítulo 22

Interfaz con Base de Datos (DB)

Las funciones de Base de Datos proveen al programador los medios para realizar operaciones con tablas de la base de datos, desde los programas CFIX.

Para poder operar con los datos de un esquema, se lo debe abrir (o "activar" en la terminología IDEAFIX) con la función *OpenSchema*, que lo agregará a la lista de esquemas activos, y lo convertirá en el esquema corriente.

Un programa puede tener varios esquemas abiertos llamando repetidas veces a *OpenSchema*, pero sólo uno es el corriente (el esquema corriente es aquel en el que se realizan las operaciones cuando no se menciona explícitamente con cual trabajar). Existe una función llamada *SwitchToSchema* que permite cambiar el esquema corriente a cualquiera de los activos.

Cada tabla de un esquema posee un "buffer" interno que puede almacenar una fila de dicha tabla. Por ejemplo cuando se ordena leer un registro con la rutina *GetRecord*, el registro leido se copia sobre dicho buffer.

El programador puede acceder a los campos de este buffer a través de funciones, que permiten tomar un valor del buffer y almacenarlo en una variable del programa, o bien copiar sobre el buffer el valor de una variable. A las funciones que toman el valor de un campo del buffer se las denomina genericamente *GetField*, y a las que cargan el valor de un campo en el buffer *SetField*.

Existe un conjunto de rutinas "Get" y un conjunto de rutinas "Set". Dentro del conjunto se diferencian por el "tipo de campo" al que pueden acceder. Por ejemplo existe una función para tomar campos enteros, otra para los campos long, etc. Estas funciones se diferencian mediante un prefijo (en las funciones Set a ese prefijo se le prefija a su vez la sílaba Set).

CFIX	Base de Datos	TipoIdentificador	bool	num (#4)	num (#9)	num(#15)	num(#28)	float

char	**date**	bool	I	x					
		int	I	x	x				
		long	l		x	x			
		double	F		x	x	x		x
		NUM	N		x	x	x	x	x
		char	S						
x		DATE	D						
	x	TIME	T						

Figura 9.1 - Correspondencia de Tipos

En el cuadro anterior se presentan las correspondencias entre los tipos de las variabls de IDEAFIX y los de "C". La siguiente es una lista de algunas de las funciones que acceden a los buffer de tablas:

Set	**Get**	**Tipo de Campo**
SetIFld(campo,valor)	IFld(campo)	short
SetLFld(campo,valor)	LFld(campo)	long
SetFFld(campo,valor)	FFld(campo)	double
SetNFld(campo,valor)	NFld(campo)	NUM
SetFld(campo,valor)	SFld(campo)	char*
SetDFld(campo,valor)	DFld(campo)	date
SetTFld(campo,valor)	TFld(campo)	time

Figura 9.2 - Funciones de Acceso de Tablas

Las rutinas del grupo Get, no tienen el prefijo "Get" como parte del nombre.

La compatibilidad de los tipos de IDEAFIX con los de "C" para las funciones de tipo Set es la siguiente:

	int/short	long	double	NUM	date	time
SetFld	x	x	x	x	x	x
SetIFld	x	x	x	x		
SetLFld		x	x	x		
SetFFld			x	x		
SetNFld				x		
SetDFld					x	
SetTFld						x

Figura 9.3 - Compatibilidad de Funciones y Formatos

La función SetFld es válida para cualquier tipo de campo, tal como se lo indica en el cuadro anterior. Es por ello que las constantes HIGH_VALUE y LOW_VALUE (presentadas en la sección 4-Constantes de Uso General. de este capítulo) definidas para fijar valores extremos de cadenas de caracteres, pueden ser aplicadas sobre cualquier tipo de campo.

Descriptores

Para identificar los elementos de las bases de datos se utilizan los descriptores. Existen los siguientes tipos de descriptores:

Descriptor	Variable type
schema	schema
table	dbtable
index	dbindex
field	dbfield

Figura 9.4 - Descriptores y variables

Un esquema es identificado dentro de un programa CFIX con una variable de tipo schema, llamada "descriptor de esquema". Estos descriptores se obtienen con la función *OpenSchema*, y se usan como parámetros al invocar las funciones que permiten establecer sobre qué esquema, tabla u otro elemento se trabajará.

Por ejemplo, al leer un registro se le pasa a la función GetRecord como parámetro un descriptor de índice, que está definiendo qué tabla y por cuál índice debe acceder para hacer la lectura (notar que un descriptor de índice define una tabla + un índice de la misma).

En cuanto a la definición del esquema con el que se trabajará, de no indicarse otra cosa, se asume el esquema corriente; es decir que cuando se referencie una tabla se supone que pertenece a dicho esquema. Es posible referirse a tablas que no pertenezcan al esquema corriente, siempre el otro esquema esté activo. Para ello se realiza una operación sobre su descriptor. Por ejemplo: sea una tabla llamada DATA propia del esquema S1, de la cual se necesita el campo C1, y otra denominada TABLA que pertenece al esquema S2, de la que se precisa el campo C2; para acceder a ambos valores:

```
int c1, c2;
schema s1 = OpenSchema("S1", IO_EABORT);
schema s2 = OpenSchema("S2", IO_EABORT);
/* S2 queda como corriente */
c1 = IFld(DATA_C1);
c2 = IFld(s1 | TABLA_C2);
```

Los descriptores de esquema se obtienen con la función *OpenSchema*, una de las nuevas facilidades de la cual dispone la versión actual de IDEAFIX.

Para obtener los otros tipos de descriptores se utiliza el archivo ".sch". Este archivo es generado mediante el lenguaje SQL (ver sentencia STORE del grupo de las sentencias DDS o la opción "language" en la creación de esquemas), o bien por el utilitario **dgen**(1D)). En él existen constantes simbólicas que forman los descriptores del siguiente modo:

Descriptor	Symbol
Table	TABLENAME
Index	TABLENAMEbyINDEXNAME
Field	TABLENAME_FIELDNAME

Figura 9.5 - Descriptores y Símbolos.

Cada descriptor de tabla tiene asociado un buffer, que contiene el último registro leído o escrito. Este dato se denomina «registro corriente". Los descriptores de tablas se obtienen de dos maneras:

1. De constantes definidas en un archivo cabecera ".sch", como se ha indicado.

2. Creándolos por medio de un llamado a la función CreateAlias.

CreateAlias se utiliza cuando se necesita más de un buffer para la misma tabla, por ejemplo cuando se precisa procesar más de un registro al mismo tiempo. Cuando se realiza una operación, el descriptor de tabla utilizado está especificando su buffer de tabla; esto es, si se tienen muchos alias, se tienen muchos buffers. Siempre se cuenta al menos con el asociado con el descriptor de tabla que está en el archivo ".sch".

Indices

Los datos en una tabla pueden ser accedidos por medio de diferentes claves. El índice permite la búsqueda de una clave específica, y también operaciones secuenciales.

Estos índices generalmente se definen mediante las sentencias de definición de datos del IQL (DDS, Data Definition Statements) exsistiendo la posibilidad de hacerlo también desde la interfaz de programación (en este caso los índices suelen estar definidos como temporales).

El Archivo *.sch*

Este archivo puede ser generado con los utilitarios **dgen** o **iql**. Contiene constantes para los "descriptores de tablas", "descriptores de índices", y "descriptores de campos". Dichos descriptores se utilizan en los programas CFIX para indicar una tabla, un campo o una tabla y su índice.

En el archivo .sch, se encontrará un comentario a continuación de cada campo que indica el tipo interno de dato del campo en cuestión. Esto permitirá elegir correctamente el tipo de función Set o Get que debe usarse sobre dicho campo.

Lista de Funciones

La siguiente tabla muestra las funciones documentadas en la sección dedicada a Bases de Datos.

Sección	Tarea que Realiza
Add	Agrega campos a una tabla o índice.
Alias	Crea, borra y obtiene alias.
BLOBs	Maneja grandes objetos de texto y binarios
BuildIndex	Construye un índice.
Close	Cierra esquemas.
Copy	Copia un campo de una tabla a otra, o de un buffer de DB a otro.
Create	Crea nuevos esquemas, tablas e índices.
CreateCursor	Crea y borra cursores.
DbCheckPoint	Realiza un checkpoint en un server
DbCheckSum	Calcula el ÔÔchecksum" de la definición de un esquema.
Delete	Borra registros.
Drop	Borra un esquema, tabla o índice.
FetchCusor	Lee registros del cursor
Fields	Devuelve la cantidad de campos de una tabla o índice.
Find	Obtiene descriptores de campo, índice, tabla o esquema.
FindRecord	Busca una clave en un índice.
FindTabReference	Busca referencias a una tabla.
FlushTable	Graba y libera una tabla.
Free	Libera registros, tablas o esquemas bloqueados.
GetFld	Obtiene valores de campos de una tabla.
GetRecord	Lee un registro de una tabla.
GetNames	Obtiene nombres y descripciones de tablas y esquemas
GetRecord	Lee un registro de una tabla
KeyDbField	Obtiene el descriptor de un campo clave.
LenDspFld	Obtiene la longitud de un campo.
Lock	Bloquea un esquema o una tabla.
MoveCursor	Desplazan un cursor.
OpenSchema	Abre un esquema.

PopUpDbMenu	Despliega un menú en la pantalla
PushRecord	Almacena y recupera registros en una pila.
PutRecord	Graba un registro en una tabla.
Set	Establece el valor de los campos del buffer de una tabla.
SetDbHandler	Establece el manejador de errores de entrada/salida.
Relation	opera con el estado de la lectura automática de tablas relacionadas con el flag del atributo in table
SwitchToSchema	Cambia el esquema corriente.
TabSize	Obtiene la cantidad de registros que hay en una tabla.
Transaction	Comienza y termina transacciones.
Values	Obtiene valores asociados a un campo de la base de datos.

Referencia de Funciones

Add

Agrega campos a una tabla o índice.

Sintaxis

```
dbfield AddTabField (dbtable tab, char* fname,
char* descr, UShort dim, type ty,
sqltype sql_ty, UShort len, UChar ndec,
field_flags flags, char* def, char* mask);
int AddIndField (dbindex ind, dbfield campo,
k_field_flags flags, UShort n, Uhort m);
```

Valor Retornado

AddTabField

• un descriptor de campo.

AddIndField

• IO_LOCKED, si el esquema, la tabla o algún registro de la tabla fueron bloqueados por algún usuario.

• OK, si la operación pudo concretarse satisfactoriamente.

• ERROR, si la tabla o el índice no existen.

Descripción

AddTabField agrega campos a la tabla que se indica como primer parámetro. Si la tabla no pertenece al esquema corriente, hay que indicar el nombre del esquema antes del nombre de la tabla, separando ambos con un símbolo "|".

Los distintos parámetros nos permiten especificar el tipo de campo, el tama–o, los atributos, etc. A continuación se detallan las alternativas:

tab nombre de la tabla a la que se le agrega el campo.

fname nombre del campo a agregar. Por ejemplo "CAMPO1".

descr descripción del campo.

dim dimensión del campo, si es un campo vectorizado.

ty tipo de dato que contendrá el campo:

TY_NUMERIC Entero.

TY_FLOAT Punto flotante.

TY_STRING Alfanumérico.

TY_DATE Fecha.

TY_TIME Hora.

TY_BOOL Booleano (verdadero o falso).

sql_ty tipo de dato de campo SQL:

TY_SQL_DEFAULT este parámetro se utiliza para compatibilizar totalmente el lenguaje SQL de InterSoft con el SQL estándar.

len longitud del campo, es decir, cuántos caracteres va a con-tener.Si el campo es numérico, es la cantidad total de dígitos, incluyendo los decimales.

ndec cantidad de dígitos decimales, si es un campo numérico.

flags sus posibles valores son:

F_OPTIONAL, el campo puede quedar vacío.

F_NOT_NULL, el campo no puede ser ignorado, debe contener algo.

def el valor por defecto del campo ("default").

mask máscara del campo, si tiene.

AddIndField agrega campos a un índice. Los parámetros son:

ind descriptor del índice al que se agregará el campo.

field descriptor del campo que se va a agregar.

flags sus valores posibles son:

K_ASCENDING, orden ascendente.

K_DESCENDING, orden descendente.

K_NOT_NULL, el índice no puede contener entradas nulas.

n para un campo alfanumérico, indica la posición del primer carácter dentro del campo que se tomará en cuenta para formar la clave. Las posiciones se numeran a partir de cero.

m este parámetro es complementario del anterior e indica la cantidad de caracteres a partir del indicado en el parámetro anterior, que se utilizarán para formar la clave.

Ejemplo

El siguiente ejemplo crea un campo numérico en la tabla, y lo agrega al índice primario. El atributo default para el campo nuevo es el triple del valor del CAMPO2, que ya existía antes.

```
include <ideafix.h>
include "example.sch"
Program (example, %I% G%)
{
dbfield field;
OpenSchema ( "example" , IO_EABORT);
field = AddTabField (TABLE, "NEWFIELD", " Field for Add
test",0,
TY_NUMERIC, 6, 2, F_NOT_NULL," 3 * FIELD2 ",NULL);
AddIndField (TABLEbyFIELD1,field,K_ASCENDING, 0, 0);
}
```

El segundo ejemplo crea un campo de caracteres, y le asigna como default la cadena de caracteres "PRUEBA". Observe las comillas adicionales, precedidas de una "\". Si no se colocan, IDEAFIX asume que "PRUEBA" es el nombre de un campo.

```
include <ideafix.h>
include "example.sch"
Program (example, %I% G%)
{
dbfield field;
OpenSchema ( "example" , IO_EABORT);
field = AddTabField (TABLE, "NEWFIELD",
"Field for Add test",0, TY_STRING, 6, 0,
F_NOT_NULL," \"TEST \" ",NULL);
AddIndField(TABLEbyFIELD1, field, K_ASCENDING, 0, 0);
}
```

Consultar

CreateTable(DB)

CreateIndex(DB)

Alias

Crea, borra y obtiene alias.

Sintaxis

```
dbtable CreateAlias (dbtable t);
void DeleteAlias (dbtable t);
void DeleteAllAlias (schema sch);
dbfield AlFld (dbtable t, dbfield f);
dbindex AlInd (dbtable t, dbindex ind);
```

Valor de Retorno

CreateAlias

• devuelve un nuevo descriptor de tabla, o ERROR si no puede crear el alias.

AlFld y *AlInd*

• devuelven un descriptor de campo e índice, respectivamente.

Descripción

Un alias es un nuevo descriptor de una tabla, que se obtiene con CreateAlias a partir de uno ya existente. Estos descriptores pre-existentes, llamados "originales", están definidos en el archivo de encabezamiento ".sch".

CreateAlias retorna un nuevo descriptor de tabla para la tabla t. Como cada descriptor de tabla posee su propio buffer, los "alias" permiten disponer de buffers adicionales para tener copias de varios registros de una misma tabla a la vez. El nuevo descriptor de tabla obtenido puede utilizarse en cualquier lugar en que se requiera un descriptor de tabla.

DeleteAlias borra un alias previamente creado con CreateAlias. La memoria asociada con este alias (es decir, el buffer) es liberada. Si se especifica un descriptor "original" de la tabla (no un alias), la función no realiza ninguna operación.

DeleteAllAlias borra todos los alias creados en el esquema pasado como parámetro.

AlFld es utilizada para referirse a un campo de un buffer perteneciente a un "alias". La función retorna un descriptor de campo para el campo cuyo descriptor original es f, pero en el buffer del alias t. El valor retornado se puede utilizar en cualquier lugar en que se requiera un descriptor de campo.

AlInd devuelve un descriptor de índice para el índice cuyo descriptor original es ind, pero en el alias t.

Ejemplo

```
#include < ideafix.h >
#include "example.sch"
Program (example, % I% % G%)
{
dbtable t1;
long result;
OpenSchema ( "example" , ++IO_EABORT);
/* t1 es un alias de la tabla * /
CreateAlias (TABLE);
/* lee el primer registro * /
t= GetRecord (TABLEbyFIELD1, FIRST_KEY,IO_NOT_LOCK);
/* lee en el alias el ultimo registro */
result = GetRecord (AlInd (t1, TABLEbyFIELD1),
LAST_KEY, IO_NOT_LOCK);
/* compara el campo CAMPO2 del primer y ultimo registro */
if(LFld(TABLE_FIELD2)==LFld(AlFld(t1,TABLE_FIELD2)))
printf ( "Are equal \ n" );
else
printf ("Are different \ n" );
}
```

Cuando un programa utiliza muchos alias, es más cómodo trabajar con constantes simbólicas, de esta manera:

```
#include <ideafix.h>
#include "example.sch"
#define T1byFIELD1 AlInd(T1, TABLEbyFIELD1)
#define T1_FIELD2 AlFld(T1, TABLE_FIELD2)
Program(example, %I% %G%)
{
dbtable T1;
long result;
OpenSchema("example", IO_EABORT);
/* T1 is an alias of the table "TABLE'' */
T1 = CreateAlias(TABLE);
/* reads the first record */
result=GetRecord(TABLAbyFIELD1,FIRST_KEY,IO_NOT_LOCK);
/* read in the alias the last record */
result=GetRecord(T1byFIELD1, LAST_KEY, IO_NOT_LOCK);
/* compare the field FIELD2 of he first and last records */
if (LFld(TABLE_FIELD2) == LFld(T1_FIELD2))
printf("Are equal\n");
else
printf("Are different\n");
}
```

BLOBs

Maneja grandes objetos de Texto y Binarios usando las capacidades BLOB de Essentia.

Sintaxis

```
int GetBinaryToFile(dbfield f, char *path);
int GetBinaryToMem(dbfield f, void *buffer, long size);
int StoreBinaryFromFile(dbfield f, char *path);
int StoreBinaryFromMem(dbfield f, void *buffer, long sz);
int DiscardBinary(dbfield f);
int GetTextToFile(dbfield f, char *path);
int GetTextToMem(dbfield f, void *buffer, long size);
int StoreTextFromFile(dbfield f, char *path);
int StoreTextFromMem(dbfield f, void *buffer, long size);
int DiscardText(dbfield f);
```

Descripción

Estas funciones dan al programador un completo control sobre grandes objetos de Texto y Binarios, como archivos personales, memos, fotografías o también programas ejecutables.

Get[Binary|Text]ToFile copia los contenidos del campo *f* [binario|texto] al archivo indicado en el path.

Get[Binary|Text]ToMem copia los contenidos del campo *f* [binari|texto] al buffer de memoria de tama–o *sz*.

Store[Binary|Text]FromFile copia los contenidos del campo indicado en el *path* al campo *f* [binario|texto].

Store[Binary|Text]FromMem copia los contenidos del buffer de memoria de tama–o sz al campo *f* [binario|texto].

Discard[Binary|Text] borra (deja con valor 0) el contenido del campo [binario|texto] *f*.

Ejemplo

Debido a la gran cantidad de datos que estos tipos pueden llegar a abarcar, la transferencia de datos del server al cliente es llevada a cabo en la demanda, por ejemplo, puede ser posible obtener un dato binario desde Essentia luego de realizar el correspondiente GetRecord, y, similarmente, será posible guardar datos binarios luego del correspondiente PutRecord. El código a coninuación es un ejemplo:

```
void function()
{
...
st=GetRecord(PLANETSbyID_PLANEA,THIS_KEY, IO_DEFAULT);
if (st != ERROR)
GetBinaryToFile(PLANETS_PHOTO,"/tmp/pho to.bmp");
...
PutRecord(PLANETS);
StoreBinaryFromFile(PLANETS_PHOTO, "/tmp/pho to.bmp");
}
```

BuildIndex

Construye un índice.

Sintaxis

```
void BuildIndex(dbindex ind, int flags)
long AddKey(dbindex ind);
int CompleteIndex(dbindex ind);
```

Descripción

Estas funciones construyen el índice que le especificamos como parámetro. Cada vez que se construye un índice con CreateIndex, es obligatorio usar alguna de estas funciones a continuación.

BuildIndex incluye todos los registros de la tabla en el índice.

AddKey incluye en el índice el registro que está en el buffer de la tabla correspondiente y devuelve el número de registro dentro dentro del índice.

CompleteIndex no pasa ningún registro al índice. Generalmente se usan para construir índices temporarios en los programas CFIX. Los parámetros que reciben son los siguientes:

ind descriptor del índice que se va a construir.

flags posibles valores de flags:

IO_CLEAR_INDEX, si ya existe ese índice, lo borra y lo vuelve a crear.

IO_VERBOSE, muestra información adicional: nombre del índice y de la tabla, porcentaje de registros procesados, y cantidad total de registros.

Ejemplo

```
#include <ideafix.h>
#include "example.sch"
Program(example, %I% %G%)
{
dbindex TABLEbyFIELD2;
/* Open the schema */
OpenSchema("ejemplo", IO_EABORT);
/* Create the index structure */
TABLEbyFIELD2=CreateIndex(TABLE, "INDEX", K_TEMP, 1);
/* Indicate the key field */
AddIndField(TABLEbyFIELD2, TABLE_FIELD2, K_ASCENDING, 0, 0);
/* Build the index */
BuildIndex(TABLEbyFIELD2, IO_VERBOSE, printf, "%s");
}
```

Consultar

CreateIndex(DB)

Close

Cierra esquemas.

Sintaxis

```
void CloseSchema(schema sch);
void CloseAllSchemas();
```

Descripción

CloseSchema cierra un esquema cuyo descriptor es sch. Toda la memoria utilizada para este esquema es liberada. Las operaciones de actualización de índices que estén pendientes serán volcadas a disco.

CloseAllSchemas cierra todos los esquemas activos.

Diagnostico

Despúes de cerrar un esquema, su descriptor (sch) no es válido hasta que un *OpenSchema* le asigne un nuevo valor. Si se cierra el esquema corriente, el esquema corriente quedará "indefinido".

Copy

Copia el contenido de un campo del buffer de la Base de Datos a otro, o de una tabla a otra.

Sintaxis

```
void CopyFld(dbfield source, dbfield target [,int row]);
int CopyAliasRecord(dbtable target, dbtable source);
void CopyTable(dbtable target, dbtable source,
dbfield vec[], int cpy_flag);
```

Descripción

CopyFld copia el contenido de un campo del buffer de la base de datos a otro. Se debe especificar el campo origen, el campo destino y -si es un campo vectorizado- el número de fila.

Esta función convierte, si es posible, el tipo de dato del campo origen al tipo de dato del campo destino. Por ejemplo, si el campo origen es de tipo char y el campo destino es de tipo

DATE, la copia se realizará satisfactoriamente si la cadena contenida en el primer campo es convertible a una fecha válida.

CopyAliasRecord copia información entre el buffer de una tabla y el buffer de un alias de la misma tabla, o entre los buffers de dos alias de una misma tabla. Si la copia es exitosa, devuelve OK. Si los descriptores de tabla destino y origen no pertenecen a la misma tabla o a uno de sus alias, devuelve ERROR.

CopyTable copia todos los registros de una tabla en otra. El vector vec[] debe contener tantos elementos como campos tenga la tabla origen: en cada uno de ellos, se debe guardar el identificador del campo de la tabla destino en donde se va a copiar. Si no se quiere copiar un campo de la tabla origen en la tabla destino, se debe poner la constante EQ_NO_FIELD en la posición que corresponda dentro de vec[]. El modificador cpy_flag puede tomar dos valores:

• `CT_NORMAL`: significa que la copia será una copia normal.

• `CT_COPY_STAMPS`: este modificador hara que *CopyTable* borre la tabla destino y luego use *AddRecordTS* para copiar los registros de una tabla a la otra, para mantener los timestamps de los registros. debido al uso de *AddRecordTS*, la persona que ejecute esta función con este indicador, debe ser el due–o del esquema.

```
#include <ideafix.h>
#include "librar.sch"
wcmd(example, %I% %G%)
{
OpenSchema("librar", IO_EABORT);
if(GetRecord(BOOKS,FIRST_KEY, IO_DEFAULT) != ERROR) {
CopyFld (BOOKS_AUTHOR, AUTHORS_CODE, 0);
GetRecord (AUTHORS, THIS_KEY, IO_EABORT);
WiMsg ("The author's name is %s",
SFld(AUTHORS_NAME));
} else {
WiMsg ("The reading wasn't satisfactory");
}
}
```

Este programa siempre intenta recuperar el primer registro de la tabla libros. Si la búsqueda es exitosa, copia el valor del campo autor de la tabla libros al campo código de la tabla autores. Luego realiza una lectura de la tabla autores con la clave indicada y muestra el nombre en pantalla invocando a WiMsg. Si la lectura de la tabla libros no es exitosa, despliega un mensaje indicándolo.

Este segundo ejemplo copia los campos clieno, nombre e iva de todos los registros de la tabla clientes a la tabla backup.

La macro FLDTOI, que está incluida en el archivo "dbdefs.h", convierte un descriptor de campo en un número entero.

Create

Crea nuevos esquemas, tablas e índices.

Sintaxis

```
dbindex CreateIndex(dbtable tbl, char* name,
key_flags flags, UChar separ);
dbtable CreateTable(schema schema, char*name,
char* descr, tab_flags flags,
Ulong size, UChar n_fields);
schema CreateSchema(char* name, char* descr);
```

Valor de Retorno

CreateIndex

• un descriptor de índice.

CreateTable

• un descriptor de tabla.

CreateSchema

• un descriptor de esquema.

Descripción

CreateIndex define la estructura de un índice, que luego se debe construir con *BuildIndex*, *AddKey* o *CompleteIndex*. Habitualmente se usa para crear índices temporarios en un programa CFIX. Los parámetros son:

tbl identificador de la tabla sobre la cual se generará el índice.

name nombre que se le asignará al índice.

flags posibles valores de flags:

K_DUP, acepta claves duplicadas.

K_UNIQUE, no acepta claves duplicadas.

K_TEMP, el índice es temporario (se borra al cerrar el esquema).

IO_EABORT, aborta la ejecución del programa si se detecta un error.

separ número de bloques de 1 Kbyte que ocupa el nodo terminal (Ver Sección II.7.3 de este manual, "Parámetros de configuración de los índices").

CreateTable crea una nueva tabla. Sus parámetros son:

schema descriptor del esquema al que pertenecerá la nueva tabla.

name nombre de la tabla.

descr descripción que se le asocia a la tabla a modo de documentación.

flags posibles valores de flags:

TAB_NORMAL, la tabla es normal (queda almacenada hasta que el usuario la borre).

TAB_TEMP, la tabla es temporaria (se borra automáticamente al cerrar el esquema).

IO_EABORT, aborta la ejecución del programa si se detecta un error.

size cantidad de registros que albergará la tabla. Este número debe ser mayor que cero.

n_fields cantidad de campos que tendrá la tabla. Debe ser un número mayor que cero.

IMPORTANTE:

Si se desea crear una tabla que no sea temporaria, en un esquema existente, es necesario bloquear el esquema previamente con LockSchema.

CreateSchema crea un nuevo esquema. Los parámetros que recibe son:

name nombre del esquema.

descripción que se la asocia a modo de documentación.

Ejemplo

Ejemplo 1: Creación de esquemas, tablas e índices.

```
#include <ideafix.h>
Program(example, %I% %G%)
{
schema bib;
dbtable books;
dbindex authors;
dbfield fld1,fld2;
/* Create the schema*/
bib = CreateSchema("librar", "Library Schema");
/* Create the table*/
books = CreateTable(bib, "books", "Library's Books",
TAB_NORMAL, 1000, 2);
/* Add the fields to the table */
fld1 = AddTabField(books, "code", "Book's code", 0,
TY_NUMERIC, 4, 0, F_NOT_NULL, NULL_STR, NULL_STR);
fld2 = AddTabField(books, "title", "Book's title", 0,
TY_STRING, 30, 0, F_NOT_NULL,NULL_STR,NULL_STR);
/* Create the index structure */
authors = CreateIndex(books, "code", K_UNIQUE, 1);
/* Add the key field to the index */
AddIndField(authors, fld1, K_ASCENDING, 0, 0);
/* Build the index */
BuildIndex(authors, IO_CLEAR_INDEX);
}
```

Ejemplo 2: Creación de una tabla en un esquema existente.

```
#include <ideafix.h>
wcmd(example, %I% %G%)
{
schema sch;
dbtable tab1;
dbindex ind1;
dbfield fld1,fld2,fld3;
/* Open the schema */
sch = OpenSchema("ejemplo", IO_EABORT | IO_SYMBOLS);
/* Lock the schema to add a table */
if (LockSchema(sch, IO_TEST | IO_LOCK)==IO_LOCKED)
Error("The schema is locked");
/* Create the table */
if((tab1=CreateTable(sch,"test","Test table", TAB_NORMAL, 100,
3)) == ERROR)
Error("Can't create the table");
/* Add fields to the table */
fld1 = AddTabField(tab1, "code", "Code Field", 0, TY_NUMERIC,
4, 0, F_NOT_NULL,
NULL_STR, NULL_STR);
fld2=AddTabField(tab1,"name","Description field", 0,
TY_STRING, 30, 0, F_OPTIONAL, NULL_STR, NULL_STR);
fld3 = AddTabField(tab1, "addres", "Address field",
0,TY_STRING, 30, 0, F_OPTIONAL, NULL_STR, NULL_STR);
/* Create the index structure */
ind1 = CreateIndex(tab1, "code", K_UNIQUE, 1);
/* Add the key field to the index */
AddIndField(ind1, fld1, K_ASCENDING, 0, 0);
/* Build the index */
BuildIndex(ind1, IO_CLEAR_INDEX);
}
```

Consultar

AddIndFields(DB)

AddTabFields(DB)

LockSchema(DB)

BuildIndex(DB)

CreateCursor

Crea y borra cursores.

Sintaxis

```
dbcursor CreateCursor(dbindex ind, int mode);
void SetCursorFrom(dbcursor cursor, [,value...]);
void SetCursorTo(dbcursor cursor, [,value...]);
void SetCursorFromFld(dbcursor cursor, int n, char* val);
void SetCursorToFld(dbcursor cursor, int n, char* val);
void DeleteCursor(dbcursor cursor);
int SetCursorFlds(dbcursor cursor, ...);
```

```
int SetCursorVFlds(dbcursor cursor, dbfield *fld);
```

Valor de Retorno

Si se produce un error, *CreateCursor* devuelve ERROR; si no, devuelve el descriptor del cursor creado.

Descripción

Estas funciones se usan para crear y borrar cursores, es decir, punteros a un conjunto de registros de una tabla.

CreateCursor crea un cursor sobre el índice ind. El parámetro modo define la acción de bloqueo que se realizará cada vez que se lea un registro con las funciones FetchCursor y FetchCursorPrev. Los valores posibles son:

IO_NOT_LOCK	siempre lee el registro, por más que esté bloqueado por otro programa.
IO_LOCK	si el registro no está bloqueado, lo lee y lo bloquea. Si no, espera a que el registro sea liberado, y después lo lee y lo bloquea.
IO_TEST	si el registro está bloqueado, devuelve el valor IO_LOCKED inmediatamente. Si no, lo lee sin bloquearlo.
IO_DESCENDING	el cursor se mueve en el índice que es creado de forma reversa a la que se movería si el flag no estuviera presente.
IO_KEEP_LOCKS	(tiene sentido cuando el cursor es creado para bloquear) hace que el cursor no libere automáticamente los bloqueos que estan siendo creados. Si el cursor es creado dentro de una transacción, el registro será liberado cuando esta termine, en cualquier otro caso, es responsabilidad del programador liberarlos (por ejemplo, con FreeTable o FreeRecord).
IO_READ	crea un bloqueo de lectura. Se define a–adiendo (no reemplazando) IO_READ a IO_LOCK. Varios bloqueos de lectura pueden coexistir en un momento dado en un registro, pero un bloqueo de lectura no puede coexistir con un bloqueo de escritura (el bloqueo normal de IDEAFIX). Nota: Funciona solo con Essentia. Cuando el RDBMS recibe este parámetro lo ignorara.
IO_PLOCK	crea un bloqueo que es persistente a través de las transacciones; el que puede desaparecer cuando se lo libere explícitamente o cuando termina el proceso. Puede ser combinada con

	IO_READ para crear un bloqueo de lectura persistente. IO_PLOCK reemplaza IO_LOCK, así que no pueden ser combinadas entre sí. Nota: Funciona solo con Essentia. Cuando el RDBMS recibe este parámetro lo ignorara.
IO_CONTROL_BREAK	permite indicar que el cursor verifique cortes de control cuando se usan las rutinas FetchCursor() y FetchCursorPrev().

SetCursorFrom y *SetCursorTo* definen los límites del cursor, es decir, el primer y el último registro al que accederá. El primer parámetro es el descriptor del cursor. Los parámetros siguientes son los valores de los campos claves para el índice sobre el que se definió el cursor, en el mismo orden en que se definió la clave. Los valores especificados deben ser compatibles con el tipo de cada campo, y se debe indicar valores para todos los campos.

SetCursorFromFld y *SetCursorToFld* definen el valor val del n-ésimo campo de la clave para el límite inferior y superior del cursor, respectivamente.

DeleteCursor elimina el cursor cuyo descriptor se pasa como parámetro.

SetCursorFlds recibe la lista de los descriptores de campos para pasarsela al registro corriente cada vez que un FetchCursor/FetchCursorPrev sea ejecutado. El final está indicado con cero.

SetCursorVFlds al igual que SetCursorFlds, pero recibe un arreglo de descriptores de campo finalizado con el descriptor 0.

Consultar

FetchCursor(DB)

DbCheckPoint

Realiza un *checkpoint* en un server.

Sintaxis

```
bool DbCheckPoint(schema sch);
```

Descripción

Esta función realiza un *checkpoint* en el server al cual pertenece el esquema esq. Para ejecutar un *checkpoint* en el server del esquema corriente, se puede realizar el siguiente llamado:

```
DbCheckPoint (Current Schema())
```

DbCheckSum

Calcula el "checksum" de la de un esquema.

Sintaxis

```
bool DbCheckSum(schema esq);
void DbVerifyCheckSum(schema esq, long chksum);
```

Descripción

Cuando se compila un esquema con **dgen -h** o se incluye la cláusula *language* en su definición, IDEAFIX genera un archivo con el mismo nombre del esquema y extensión ".sch". En este archivo, define una constante llamada IO_esq_CHKSUM (donde esq es el nombre del esquema) cuyo valor depende de los campos, tablas, etc. que lo componen. Si se modifica la definición del esquema, pero no se actualiza el archivo ".sch", se pueden producir errores en los programas. Para detectar esta situación, se utilizan las funciones *DbCheckSum* y *DbVerifyCheckSum*.

DbCheckSum devuelve el valor que debería tener IO_esq_CHKSUM según la definición actual del esquema. Comparando la constante con el valor obtenido, se podrá saber si el archivo ".sch" está actualizado o no.

DbVerifyCheckSum realiza la comparación directamente, y si el archivo está desactualizado, emite un mensaje informando el error. El segundo parámetro debe ser la constante IO_esq_CHKSUM.

Ejemplo

```
#include <ideafix.h>
#include "librar.sch"
Program(example, %I% %G%)
{
schema sch;
sch = OpenSchema("librar", IO_EABORT);
DbVerifyCheckSum(sch, IO_LIBRAR_CHKSUM);
}
```

Delete

Borra registros.

Sintaxis

```
void DelRecord(dbtable tab);
int DelAllRecords(dbtable tab);
```

Valor de Retorno

DelAllRecords devuelve uno de los siguientes valores:

• IO_LOCKED si el o los registros están bloqueado.

• ERROR si se produce un error al leer la tabla o el índice (por ejemplo, si no se tiene los permisos adecuados).

• OK si no hubo problemas.

Descripción

DelRecord borra el registro de la tabla cuya clave está en el buffer.

DelAllRecords borra todos los registros de una tabla dada.

Ejemplo

La tabla TABLA tiene como clave primaria a CAMPO1 que es un campo entero. Se desea borrar el registro cuya clave es 20:

```
#include "example.sch"
Program(example, %I% %G%)
{
OpenSchema("example", IO_EABORT);
SetFld(TABLE_FIELD1, "MicroDevelopments Inc.");
DelRecord(TABLE);
}
```

Consultar

Alias(DB)

Drop

Borra un esquema, tabla o índice.

Sintaxis

```
int DropSchema(schema sch);
int DropTable(dbtable tab);
int DropIndex(dbindex ind);
```

Valor de Retorno

DrpSchema, DropTable y *DropIndex* retornan uno de los siguientes valores enteros:

• OK, si la operación fué completada.

• ERROR, si la operación falló. La causa del error puede ser obtenida con GetIoErrno.

• IO_LOCKED, si el esquema, tabla o índice estaba en uso por otro usuario.

Descripción

DropSchema borra las definiciones y contenidos de todas las tablas e índices del esquema pasado como parámetro, incluyendo el directorio que los contenía.

DropTable elimina la definición y el contenido de la tabla pasada como parámetro y todos los índices que tuviera asociados. Si no se especifica el esquema, se asumirá el corriente.

DropIndex da de baja la definición y el contenido del índice pasado como parámetro.

Estas funciones realizan la misma tarea que los comandos *drop schema*, *drop table* y *drop index* del lenguaje SQL.

Ejemplo

```
#include <ideafix.h>
#include "test1.sch"
#include "test2.sch"
Program(example, %I% %G%)
{
schema sch1,sch2;
sch1 = OpenSchema ("test1", IO_EABORT);
sch2 = OpenSchema ("test2", IO_EABORT);
/* Delete an index from the table TABLE2
of the test2 schema*/
DropIndex (TABLE2byFIELD1);
/* Delete an index from the table TABLE1
of the test1 schema*/
DropIndex (sch1 | TABLE1byFIELD1);
/* Delete the table TABLE4 from schema test2 */
DropTable (TABLE4);
/* Delete table TABLE5 from schema test1 */
DropTable (sch1 | TABLE5);
/* Delete schema test1 */
DropSchema (sch1);
/* Delete schema test2 */
DropSchema (sch2);
/* Delete schema test3 */
DropSchema (OpenSchema("test3", IO_EABORT));
}
```

FetchCursor

Lee un registro de un cursor.

Sintaxis

```
long FetchCursor(dbcursor cursor);
long FetchCursorPrev(dbcursor cursor);
long FetchCursorThis(dbcursor cursor);
long CountCursor(dbcursor cursor);
```

Valor de Retorno

FetchCursor y *FetchCursorPrev* retornan uno de los siguientes valores:

• IO_LOCKED, si el registro está bloqueado.

• ERROR, si se produjo un error.

• OK, si el registro fue leido sin problemas.

Descripción

Estas funciones desplazan el cursor especificado y leen el registro correspondiente de la tabla, en forma análoga a los modos de acceso NEXT_KEY y PREV_KEY de *GetRecord*.

FetchCursor lee el siguiente registro dentro del cursor.

FetchCursorPrev lee el registro anterior dentro del cursor.

En todos los casos, la acción de bloqueo realizada dependerá de lo que se haya indicado al crear el cursor con *CreateCursor*.

FetchCursorThis inserta en el registro corriente de la tabla asociada al cursor, el último registro leído con *FetchCursor/fetchCursorPrev*.

CountCursor recibe como parámetro un cursor con los rangos previamente fijados y retorna la cantidad de registros en su rango. Si los límites no fueron especificados, retornará la cantidad de registros de toda la tabla, debido a que el valor por defecto de un cursor es la dimensión de la tabla.

Consultar

CreateCursor(DB)

MoveCursor(DB)

GetRecord(DB)

Fields

Devuelve la cantidad de campos de una tabla o índice.

Sintaxis

```
int TabNFields(dbtable tab);
int IndNFields(dbtable ind);
```

Valor de Retorno

TabNFields devuelve la cantidad de campos de una tabla.

IndNFields devuelve la cantidad de campos de un índice.

Descripción

TabNFields e *IndNFields* se utilizan para saber la cantidad de campos que conforman una tabla y la cantidad de campos que conforman un índice, respectivamente.

Consultar

AddTabFields(DB)

AddIndFields(DB)

CreateTable(DB)

CreateIndex(DB)

Find

Obtiene descriptores de campo, índice, tabla o esquema.

Sintaxis

```
dbfield FindDbField(dbtable tab, char* name);
dbindex FindDbIndex (dbtable tab, char* name);
dbtable FindDbTable(schema sch, char* name);
schema FindSchema(char* name);
schema FindNextSchema(schema sch);
schema CurrentSchema();
```

Valor de Retorno

FindDbField devuelve el descriptor del campo cuyo nombre se indica, perteneciente a la tabla tab.

FindDbIndex devuelve el descriptor del índice cuyo nombre se indica, perteneciente a la tabla tab.

FindDbTable devuelve el descriptor de la tabla cuyo nombre se indica, perteneciente al esquema esq.

FindSchema devuelve el descriptor del esquema cuyo nombre se indica, y que ya había sido abierto con OpenSchema.

FindNextSchema devuelve el descriptor del siguiente esquema de la tabla de esquemas activos.

CurrentSchema devuelve el descriptor del esquema actual.

Descripción

Dado el nombre del campo, índice, tabla o esquema, las funciones *FindDbField*, *FindDbIndex*, *FindDbTable* y *FindDbSchema* devuelven el descriptor correspondiente. En todos los casos, si los parámetros son erróneos, las funciones retornarán un código de error.

Para poder utilizar estas funciones, es necesario abrir el o los esquemas con la opción IO_SYMBOLS.

FindNextSchema devuelve el descriptor del esquema que haya sido abierto inmediatamente después del que se indica como parámetro. Si no se han abierto más esquemas, la función retorna un código de error.

CurrentSchema devuelve el descriptor del esquema actual.

Ejemplo

```
#include <ideafix.h>
#include "test1.sch"
Program(example, %I% %G%)
{
dbfield field;
dbtable table;
schema sch;
sch = OpenSchema ("test1", IO_EABORT | IO_SYMBOLS);
tabla = FindDbTable (sch, "table1" );
campo = FindDbField (tabla, "field1" );
/* etc... */
}
```

En este ejemplo, primero se declaran las variables campo, tabla y esq de tipo dbfield, dbtable y schema, respectivamente. Luego se abre el esquema prueba1 y se guarda el descriptor en la variable esq.

En la variable tabla se guarda el descriptor de la tabla tabla1, que se obtiene con *FindDbTable*.

Por último, se guarda en la variable campo el descriptor del campo tabla1, que se obtiene con *FindDbField*. Los parámetros pasados son el descriptor de la tabla a la cual pertenece el campo y el nombre del campo que se busca.

Consultar

OpenSchema(DB)

Version(DB)

FindRecord

Busca una clave en un índice.

Sintaxis

```
long FindRecord(dbindex ind,
find_mode mode [, Ushort nparts]);
```

Valor de Retorno

FindRecord devuelve:

• ERROR, si la clave buscada no existe en el índice.

• un número, si la clave fue encontrada.

Descripción

Esta función busca una clave utilizando el índice ind. Se utiliza para determinar si la clave existe o no, ya que el registro de datos no es leído sobre el buffer, aunque la búsqueda sea exitosa. En este caso, simplemente devuelve el número de registro físico donde se encuentra la clave. Este número se puede utilizar luego para leer el registro de datos con GetRecord y la opción BY_RECNO.

Antes de realizar la búsqueda, debe cargarse en el buffer de la tabla el valor de la clave que se quiere buscar.

ind es el descriptor del índice sobre el cual se realizará la operación de búsqueda.

Modos de Búsqueda

Las claves de un índice pueden estar ordenadas en forma ascendente o descendente, según lo

que se haya indicado en el momento de crear el índice. Por eso, no se habla de claves "mayores" o "menores", sino de "clave anterior" y "clave siguiente".

El parámetro modo de *FindRecord* indica la forma en que se buscará la clave:

• THIS_KEY, búsqueda por clave exacta.

• PREV_KEY, clave anterior.

• NEXT_KEY, clave siguiente.

• FIRST_KEY, primera clave del índice.

• LAST_KEY, última clave del índice.

THIS_KEY busca una clave utilizando el índice ind. Los campos que forman la clave deben ser cargados con *SetKey* o *SetKeyFld*, o con la función "Set" apropiada. Si el registro es encontrado, los campos clave permanecen inalterados en el buffer y se retorna la posición física del registro dentro de la tabla. De otro modo, estos campos contendrán la clave anterior a la buscada, y el valor devuelto por *FindRecord* será ERROR.

NEXT_KEY y PREV_KEY buscan la siguiente clave o la anterior a la contenida en el buffer, respectivamente. Si la clave dada es la primera o anterior a la primera y se usa PREV_KEY, o si es la última o siguiente a la última y se usa NEXT_KEY, FindRecord devolverá ERROR y los campos de la clave permanecerán inalterados.

FIRST_KEY y LAST_KEY buscarán la primera y la última clave del índice, respectivamente. No es necesario cargar previamente en el buffer ningún valor. Como resultado de la función, quedará la primera o última clave en los campos del buffer. Si la tabla está vacía, *FindRecord* devolverá ERROR.

Modos Compuestos

Es posible combinar los modos de búsqueda de la siguiente manera:

• THIS_KEY | NEXT_KEY, busca la clave dada o la siguiente.

• THIS_KEY | PREV_KEY, busca la clave dada o la anterior.

En estos modos, FindRecord devolverá ERROR solamente si la clave dada es siguiente a la última (THIS_KEY | NEXT_KEY), o anterior a la primera (THIS_KEY | PREV_KEY).

Busqueda por Clave Parcial

El modo PARTIAL_KEY se puede agregar a cualquiera de los anteriores para buscar una clave compuesta, pero tomando no la totalidad de la clave, sino los npartes primeros campos.

Para unir PARTIAL_KEY a los otros modos, se usa el símbolo " | ":

• THIS_KEY | PARTIAL_KEY

• THIS_KEY | NEXT_KEY | PARTIAL_KEY

El efecto de este modo es el siguiente: cuando se lee de la base de datos, se compara el registro leído con el que existía previamente en el buffer. Si los primeros npartes campos de la clave son iguales, la búsqueda se considera exitosa.

Ejemplo

La tabla TABLA tiene como clave primaria los campos CAMPO1 (un campo de caracteres) y CAMPO2 (un campo entero).

```
#include <ideafix.h>
#include "example.sch"
Program(example, %I% %G%)
{
long result;
int n;
OpenSchema("example", IO_EABORT);
/* Busca una clave exacta */
SetFld(TABLE_FIELD1, "MicroMachines Inc.");
SetFld(TABLE_FIELD2, "234");
if ((result=FindRecord(TABLE, THIS_KEY)) == ERROR)
printf("The key does not exist\n");
else
printf("The key exists\n");
/* cuenta los registros que responden a la cond. FIELD1=7 */
n=0;
SetIFld(TABLE2_FIELD1, 7);
while (FindRecord(TABLE2, NEXT_KEY | PARTIAL_KEY, 1)
!= ERROR)
n++;
printf("There are %d with the fist key field 7\n",n);
}
```

Consultar

GetRecord(DB)

CreateAlias(DB)

Set(DB)

FindTabReference

Busca referencias a una tabla.

Sintaxis

```
dbtable FindTabReference(dbtable t);
```

Descripción

FindTabReference busca referencias a la tabla pasada como parámetro en otras tablas del mismo esquema. Si encuentra una referencia (por ejemplo, en una cláusula IN), devuelve el descriptor de la tabla que la contiene. Si no, devuelve ERROR.

DropTable llama a esta función para verificar que no hay relaciones que dependan de la tabla que se quiere borrar.

FlushTable

Graba y libera una tabla.

Sintaxis

```
void FlushTable(dbtable tab);
```

Descripción

FlushTable graba físicamente los datos de la tabla indicada, y libera toda la memoria asociada a ella. Es útil en sistemas con poca memoria, cuando se desea disponer de más memoria libre.

Free

Libera registros, tablas o esquemas bloqueados.

Sintaxis

```
void FreeSchema(schema sch);
void FreeTable(dbtable tab);
void FreeRecord(dbindex ind,
find_mode mode [, long recno]);
```

Descripción

Estas funciones liberan registros, tablas o esquemas que han sido bloqueados para prevenir el acceso simultáneo.

FreeSchema libera un esquema bloqueado con LockSchema. Se elimina sólo el bloqueo impuesto con esta última función, sin liberar otros bloqueos sobre tablas o registros del esquema que el programa haya definido.

FreeTable libera todos los registros bloqueados de una tabla, y si existía, el bloqueo total sobre la tabla impuesto con *LockTable*.

FreeRecord libera un registro bloqueado con el modo IO_LOCK de *GetRecord*, sin quitar otro tipo de bloqueos.

En el caso de FreeRecord, se debe indicar el índice por el que se buscará el registro (ind) y el modo de búsqueda (modo).

Los modos de búsqueda posibles son:

THIS_KEY busca la clave guardada en el buffer.

PREV_KEY busca la clave anterior a la del buffer.

NEXT_KEY busca la clave siguiente a la del buffer.

FIRST_KEY busca la primera clave del índice.

LAST_KEY busca la última clave del índice.

THIS_KEY|PRV_KEY busca la clave del buffer o la anterior.

THIS_KEY|NEXT_KEY busca la clave del buffer o la siguiente.

...|PARTIAL_KEY compara solamente los n primeros campos de las claves.

BY_RECNO busca el registro que ocupa la n-ésima posición física dentro de las tablas.

Consultar

GetRecord(DB)

Lock(DB)

FindRecord(DB)

GetFld

Obtiene valores de campos de una tabla.

Sintaxis

```
char* SFld(dbfield f[,int row]);
short IFld(dbfield f[,int row]);
long LFld(dbfield f[, int row]);
double FFld(dbfield f[,int row]);
DATE DFld(dbfield f[,int row]);
TIME TFld(dbfield f[,int row]);
NUM NFld(dbfield f[,int row]);
void GetFld(dbfield f, char* buf[,int row]);
void GetDspFld(dbfield f, char* buf[, int row]);
void GetKeyFld(dbindex ind, int n, char* buf);
char *GetFldDesc(dbfield fld);
char *GetFldName(dbfield fld);
```

```
type GetFldType(dbfield fld);
int GetFldLen(dbfield fld);
int GetFldNDec(dbfield fld);
```

Valor de Retorno

- *SFld, IFld, FFld, DFld, TFld y NFld*, valor del campo.
- *GetFldDesc*, descripción del campo.
- *GetFldName*, nombre del campo.
- *GetFldType*, tipo del campo.
- *GetFldLen*, longitud del campo.
- *GetFldNDec*, número de lugares décimales.

Descripción

Estas funciones permiten tomar valores de campos del buffer de una tabla. f es un descriptor de campo. Los descriptores de campos se obtienen del archivo cabecera ".sch", o por medio de alias.

Si el campo es vectorizado, el parámetro fila indica qué elemento del vector debe ser tomado.

Las funciones Get retornan el valor del campo en diferentes tipos de datos del lenguaje "C". Si el valor de un campo no puede ser convertido al tipo de dato seleccionado (por ejemplo, si un campo alfanumérico es accedido con *GetIFld*) se invoca al manejador de error de conversión de tipos. Si éste retorna el control a la función, devolverá el valor NULL correspondiente a ese tipo.

SFld devuelve un puntero a un registro interno de IDEAFIX, que contiene una cadena de caracteres con el valor del campo.

GetDspFld copia en buf el valor del campo tal como aparecería en la pantalla, es decir, aplicándole la máscara (si la tiene) y colocándole la coma decimal y el punto de miles si el campo es numérico. buf debe tener suficiente espacio para contener la cadena de caracteres resultante (se puede conocer esta longitud con la función *LenDspFld(DB)*).

GetFld copia el valor del campo en ASCII sobre la cadena de caracteres apuntada por buf, sin hacer ningún tipo de formateo. Antes de llamar a esta función, se debe reservar suficiente memoria en buf.

GetKeyFld copia el contenido del n-ésimo campo de la clave del índice ind en la cadena de caracteres apuntada por buf. Antes de llamar a la función, se debe reservar suficiente memoria en buf.

Consultar

SetConvHandler(GN)

GetIndex

Obtine el nombre de un índice y campos.

Sintaxis

```
char *GetIndFlds(dbindex ind);
char *GetIndName(dbindex ind);
```

Valor de Retorno

GetIndFlds, retorna un cadena de carácteres con los nombres de los campos del índice dado, separado con tabuladores (\t).

GetIndName, retorna una cadena de carácteres con el nombre del índice.

Notas

Para usar estas funciones el esquema debe ser abierto con el modificador `IO_SIMBOLS`.

GetNames

Obtiene nombres y descripciones de esquemas y tablas.

Sintaxis

```
char *GetSchDescr(schema sch);
char *GetTabDescr(dbtable tab);
char *GetTables(schema sch);
char *GetTabFlds(dbtable tab);
char *GetIndices(dbtable tab);
```

Valor de Retorno

• *GetSchDescr*, retorna una cadena con la descripción del esquema.

• *GetTabDescr*, retorna una cadena con la descripción de la tabla.

• *GetTables*, retorna una cadena con los nombres de las tablas del esquema, separados por tabuladores (\t).

• *GetTabFlds*, retorna una cadena con los nombres de los campos de la tabla, separados por tabuladores (\t).

• *GetIndices*, retorna una cadena con los nombres de los indices de la tabla, separados por tabuladores (\t).

Notas

Para usar estas funciones el esquema debe ser abierto con el modificador `IO_SYMBOLS`.

GetRecord

Lee un registro de una tabla.

Sintaxis

```
long GetRecord (dbind nd, find_mode lmode,
int bmode[, long n]);
```

Valor de Retorno

• ERROR, si no pudo encontrar el registro.

• IO_LOCKED, si el registro estaba bloqueado por otro programa.

• un número, si la operación de lectura fue exitosa; el valor devuelto es el número de registro físico leído.

Descripción

Esta función lee un registro, buscando por el índice ind. Si lo encuentra, lo copia en el buffer de la tabla.

El parámetro *lmodo* define el modo de búsqueda, y puede ser una de las siguientes constantes:

THIS_KEY busca la clave guardada en el buffer.

PREV_KEY busca la clave anterior a la del buffer.

NEXT_KEY busca la clave siguiente a la del buffer.

FIRST_KEY busca la primera clave del índice.

LAST_KEY busca la última clave del índice.

THIS_KEY|PREV_KEY busca la clave del buffer o la anterior.

THIS_KEY|NEXT_KEY busca la clave del buffer o la siguiente.

...|PARTIAL_KEY compara solamente los n primeros campos de las claves.

BY_RECNO busca el registro que ocupa la n-ésima posición física dentro de las tablas.

El parámetro *bmodo* define la acción de bloqueo que se realizará si el registro es encontrado.

Los posibles valores son:

IO_NOT_LOCK siempre lee el registro, por más que esté bloqueado por otro programa.

IO_LOCK si el registro no está bloqueado, lo lee y lo bloquea. Si no, espera a que el registro sea liberado, y después lo lee y lo bloquea.

IO_TEST si el registro está bloqueado, devuelve el valor IO_LOCKED inmediatamente. Si no, lo lee sin bloquearlo. si el registro no existe, aborta el programa.

IO_READ	crea un bloqueo de lectura. Se define a–adiendo (no reemplazando) IO_READ a IO_LOCK. Varios bloqueos de lectura pueden coexistir en un momento dado en un registro, pero un bloqueo de lectura no puede coexistir con un bloqueo de escritura (el bloqueo normal de IDEAFIX).Nota: Funciona sólo con Essentia. Cuando el RDBMS recibe este parámetro lo ignorará.

Los usos más comunes de estos modos son:

IO_NOT_LOCK listados y consultas.

IO_LOCK bloqueo durante una actualización.

IO_TEST no suele usarse; devuelve el estado en que está un registro.

IO_TEST | IO_LOCK utilizado para realizar altas, bajas, modificaciones y consultas al mismo tiempo.

Diagnósticos

Si se produce un error de acceso físico, se invoca al manejador de errores correspondiente, que se establece con *SetDbHandler*.

Por omisión, el manejador muestra un mensaje de error en la pantalla y pregunta si se quiere abortar o reintentar la operación.

Para leer registros dentro de un cursor, se usan las funciones *FetchCursor* y *FetchCursorPrev*.

Consultar

SetDbHandler(DB)

FindRecord(DB)

FetchCursor(DB)

GetSchemas

Obtiene los nombres de los esquemas disponobles para una aplicación.

Sintaxis

```
const char* GetSchemas(void);
```

Descripción

GetSchemas retorna una cadena con los nombres de todos los esquemas que pueden ser abiertos desde una aplicación. Los nombres de los esquemas son separados por tabuladores (\t). Si no hay esquemas disponibles, retornará `NULL_STRING`.

KeyDbField

Obtiene el descriptor de un campo clave.

Sintaxis

```
dbfield KeyDbField(dbindex ind, int n);
```

Valor de Retorno

Descriptor del n-ésimo campo de la clave de un índice.

Descripción

Esta función retorna el descriptor del campo número n dentro de la clave del índice ind.

Se la utiliza cuando por algún motivo no se dispone del descriptor. Por ejemplo, en el caso de un programa parametrizado que maneja distintas bases de datos, y no puede utilizar un archivo ".sch" para obtener los descriptores de los campos.

LenDspFld

Obtiene la longitud de un campo.

Sintaxis

```
int LenDspFld(dbfield fn);
```

Valor de Retorno

LenDspFld devuelve la longitud del campo formateado.

Descripción

LenDspFld retorna un entero con la longitud máxima que ocupará un valor de ese campo convertido a una cadena de caracteres, formateado con máscara si la tiene, y con coma decimal y punto de miles si es un campo numérico.

Lock

Bloquea un esquema o una tabla.

Sintaxis

```
int LockSchema(schema sch, int mode);
int LockTable(dbtable tab, int mode);
```

Valor de Retorno

• IO_LOCKED, si la tabla o el esquema están bloqueados por otro programa.

• OK, si la operación fue realizada con éxito.

Descripción

LockTable bloquea todos los registros de una tabla, imponiendo un bloqueo general sobre la misma. Desde el momento en que la tabla queda bloqueada, ningún otro programa podrá:

• bloquear un registro de esa tabla con la opción IO_LOCK de *GetRecord*.

• bloquear la tabla con *LockTable*.

Para que esta operación se pueda realizar, ningún otro proceso debe tener registros bloqueados sobre la tabla.

LockSchema bloquea el acceso a un esquema. Para que la operación se pueda realizar, ningún proceso debe tener abierto el esquema que se desea bloquear. Mientras un esquema está bloqueado, ningún proceso puede activarlo con *OpenSchema*.

En ambas funciones, el parámetro modo define la estrategia de bloqueo.

Los valores posibles son:

IO_TEST simplemente averigua si la operación se podría realizar. Si es posible, se devolverá OK, y en caso contrario IO_LOCKED.

IO_LOCK bloquea la tabla o esquema. Si no puede hacerlo, espera hasta que sea posible. IO_TEST | IO_LOCK" intenta bloquear la tabla o esquema.

Si no puede hacerlo, devuelve inmediatamente IO_LOCKED.

Consultar

Free(DB)

MoveCursor

Desplazan un cursor.

Sintaxis

```
void MoveCursorFirst(dbcursor cursor);
void MoveCursorLast(dbcursor cursor);
```

Descripción

MoveCursorFirst lleva el cursor al primer registro del conjunto en el que fue definido.

MoveCursorLast lleva el cursor al último registro del conjunto en el que fue definido.

Consultar

CreateCursor(DB)

FetchCursor(DB)

OpenSchema

Abre un esquema.

Sintaxis

```
schema OpenSchema (char* name, UShort mode);
```

Valor de Retorno

El descriptor del esquema, si se pudo abrir, y ERROR en caso contrario.

Descripción

Esta función abre un esquema. nombre es el nombre con que está grabado en el disco, y también puede incluir el directorio que contiene el archivo ".sco".

Si la operación tiene éxito, *OpenSchema* devuelve el descriptor del esquema abierto, lo agrega a la lista de esquemas *activos* y lo marca como el esquema *corriente*.

Los esquemas *activos* son aquellos a los que se puede acceder en un determinado momento, porque ya fueron abiertos con *OpenSchema*. El esquema *corriente* es aquel que se usa por defecto cuando en una operación sobre una tabla no se indica el esquema al que pertenece. El esquema corriente cambia al usar las funciones *OpenSchema*, *SwitchToSchema*, *FindNextSchema* y *OpenSchemaVersion*.

El parámetro *modo* puede ser una de estas constantes:

IO_DEFAULT es una operación normal: si ocurre un error, devuelve ERROR.

IO_EABORT si se produce un error, aborta el programa.

... | IO_SYMBOLS carga la tabla de símbolos del esquema. Esta tabla es usada -entre otras funciones- por FindDbField e InDescr.

Ejemplo

En UNIX:

```
dbschema scd;
scd = OpenSchema("biblio", IO_EABORT);
```

o bien

```
scd = OpenSchema("/usr/ideafix/datos/biblio", IO_EABORT);
```

En MS-DOS:

```
scd = OpenSchema("biblio", IO_EABORT);
scd = OpenSchema("\\usr\\idea\\datos\\biblio", IO_EABORT);
scd = OpenSchema("c:\\usr\\idea\\datos\\biblio", IO_EABORT);
```

Consultar

SwitchToSchema(DB)

FindNextSchema(DB)

CloseSchema(DB)

CloseAllSchemas(DB)

Version(DB)

PushRecord

Almacena y recupera registros en una pila.

Sintaxis

```
void PushRecord(dbtable tab);
long PopRecord(dbtable tab);
void DiscardRecord(dbtable tab);
void DiscardAllRecords(dbtable tab);
long RecordStackSize(dbtable tab);
```

Valor de Retorno

Si se produce algún error, PopRecord devuelve ERROR.

Descripción

Estas funciones permiten guardar registros en una pila LIFO (ÔÔLast In, First Out").

PushRecord copia en la pila el registro que está en el buffer de la base de datos.

PopRecord copia en el buffer de la base de datos el último registro que fue introducido en la pila.

DiscardRecord elimina el último registro que fue introducido en la pila. El contenido de ese registro se pierde.

DiscardAllRecords elimina todos los registros que fueron introducidos en la pila, dejándola vacía. El contenido de esos registros se pierde.

RecordStackSize devuelve la cantidad de registros que quedan en la pila.

PutRecord

Graba un registro en una tabla.

Sintaxis

```
void PutRecord (dbtable tab);
```

```
void AddRecord (dbtable tab);
void AddRecordTS(dbtable tab, TIMESTAMP crts,
TIMESTAMP modts);
long AddRecordTest (dbtable tab);
```

Valor de Retorno

Si durante la operación ocurre algún error (ej. el registro ya existía) la transacción se corrompe. Esto se puede verificar mediante la función TransOk.

Si el registro ya existe, AddRecordTest retorna ERROR .

Descripción

PutRecord graba el contenido del buffer de la tabla tab en el disco. También actualiza todos los índices de la tabla con la clave del buffer, agregándola si no existía o modificándola en caso contrario.

AddRecord graba el contenido del buffer de la tabla tab en el disco, siempre y cuando no exista un registro con la misma clave.

NOTA: Cuando se usa IDEAFIX con Essentia, luego de un PutRecord o un *AddRecord*[TS], los campos del tipo *stamp* (por ejemplo, cuid, muid, cdate, mdate, ctime, mtime) no son actualizados en el buffer de la tabla. Para acceder correctamente a estos campos, es necesario usar *GetRecord* luego de *Add* o *Put*.

AddRecordTest tiene la misma funcionalidad que AddRecord, pero no corrompe la transacción si el registro existe.

Ejemplo

```
#include <ideafix.h>
#include "example.sch"
Program(example, %I% %G%)
{
OpenSchema("example", IO_EABORT);
/* Create a blank record */
InitRecord(TABLE);
/* Set the values for the fields */
SetFld(TABLE_FIELD1, "MicroMach Co.");
SetFld(TABLE_FIELD2, "223");
/* Store the record */
PutRecord(TABLE);
}
```

Consultar

FmToDb(DB)

TransOk(DB)

Relation

Activa la lectura automática de las tablas relacionadas con el atributo *in table*.

Sintaxis

```
void SetRelation(dbtable root, int level, int flag_err);
int GetRelDepth(dbtable tab);
int GetRelErrFlag(dbtable tab);
```

Descripción

El atributo *in table*, que se a–ade al campo en la definición del esquema, crea una relación entre dos tablas. Esta relación puede ser cambiada, formando un sistema como el que se muestra a continuación.

Cuando *GetRecord* es usada en una tabla y la lectura de tablas relacionadas esta activada, no solo el registro de esa tabla es leído, sino también todas las que dependan de ella.

SetRelation activa la lectura automática de las tablas que dependan de la raíz.

El parámetro **level** indica la profundidad de la relación, por ejemplo, cuantos in table serán considerados desde la tabla raíz. Si level es cero, la lectura automática es desactivada para la tabla indicada.

El parámetro **flag_err** indica la acción a realizar si GetRecord no encuentra el valor buscado en la tabla relacionada:

• IO_EABORT, aborta el programa.

• IO_TEST, para notificar indicando las tablas que no pueden ser relacionadas.

• IO_DEFAULT, ignorar.

Niveles:

Figura 9.6 - Tablas Relacionadas

En esta figura la tabla A tiene el atributo in B en un campo y el in C en otro campo. La tabla B tiene un in D en uno de sus campos, como C tiene un in E.

El parámetro **level** es usado como en los siguientes ejemplos:

```
SetRelation(A, 2, flag);
```

Para cada lectura en la tabla A, lee los registros relacionados (a travéz del "in") de las tablas B

y C, y también lee los registros de las tablas D y E relacionadas por campos en las tablas C y B respectivamente. En consecuencia, GetRecord actualiza cinco (5) buffers.

```
SetRelation(A, 1, flag);
```

Para cada lectura en la tabla A, lee los registros relacionados de las tablas B y C. En este caso, sólo se actualizan tres (3) buffers, debido a que las tablas D y E no son leidas.

```
SetRelation(A, 1, flag1);
SetRelation(C, 1, flag2);
```

Para cada lectura en la tabla A, lee los registros relacionados de las tablas B y C, y también lee el registro de la tabla E relacionado con el campo en la tabla C. Se actualizan cuatro buffers con solo un GetRecord, porque la tabla E no es leida. Como se puede ver en el último ejemplo, el *level* establecido en la tabla A no altera el de la tabla B, a no ser que el *level* de A indique leer más tablas que el *level* de B.

GetRelDepth retorna la profundidad de la SetRelation sobre la tabla pasada como parámetro.

GetRelErrFlag retorna el modificador implantado con SetRelation.

Ejemplo

```
#include <ideafix.h>
#include "librar.sch"
wcmd(ejemplo, %I% %G%)
{
fin_mode mode;
OpenSchema("librar", IO_EABORT);
SetRelation(BOOKS, 1, IO_EABORT);
for(mode = FIRST_KEY;
GetRecord(BOOKS, mode,I_DEFAULT) != ERROR;
mode = NEXT_KEY)
WiMsg("The author of the book %s is %s", SFld(BOOKS_TITLE),
SFld(AUTHORS_NAME));
}
```

Este programa muestra el nombre del autor de cada uno de los libros de la tabla de libros. Como el campo autor tiene el atributo `in authors`, la tabla de autores es leida usando el valor de ese campo como clave. Si el registro no existe en la tabla, el programa termina desplegando un mensaje de error, debido al modificador IO_EABORT de *SetRelation*.

Consultar

GetRecord(DB)

Set

Establece el valor de los campos del buffer de una tabla.

Sintaxis

```
void SetFld(dbfield fn, char* buf[, int row]);
void SetIFld(dbfield fn, short buf[, int row]);
void SetLFld(dbfield fn, long buf[, int row]);
void SetFFld(dbfield fn, double buf[, int row]);
void SetDFld(dbfield fn, DATE buf[, int row]);
void SetTFld(dbfield fn, TIME buf[, int row]);
void SetKey(dbindex ind, value[, value][, ...]);
void SetKeyFld(dbindex ind, int n, char* val);
void InitRecord(dbtable tab);
int SetTableCache(dbtable tbl, long size);
```

Descripción

Estas funciones colocan datos en el buffer de una tabla. fn es el descriptor del campo que se desea modificar: puede obtenerse del archivo cabecera ".sch" o mediante un alias.

El parámetro **fila** se agrega cuando el campo es un vector, e indica qué elemento de éste debe utilizarse.

SetFld copia la cadena de caracteres apuntada por buf en el campo indicado. Los caracteres son copiados hasta que la cadena termina, o el campo no tiene más espacio.

Las siguientes funciones Set colocan un valor en el campo indicado del buffer de la tabla. El valor se suministra en el parámetro val. Si se utiliza una función que no es compatible con el tipo del campo, se invoca al manejador de error de conversión de tipos (ver SetConvHandler(DB)).

SetKey pone los valores de la clave en el buffer de una tabla. El primer parámetro es un descriptor de índice, obtenido del archivo cabecera ".sch" o de un alias. Los parámetros siguientes son los valores de los campos claves, en el mismo orden en que la clave fue definida. Los valores suministrados deben ser compatibles con el tipo de cada campo, y se debe indicar valores para todos los campos.

SetKeyFld asigna un valor al n-ésimo campo de la clave. InitRecord prepara un registro nuevo en el buffer de la tabla indicada, poniendo NULL en cada campo o -si lo tiene definido- su valor por defecto.

SetTableCache establece un cache de n registros para la tabla tbl.

Cada vez que se ejecute GetRecord sobre la tabla tbl (con parámetros THIS_KEY y IO_NOT_LOCK) la biblioteca de Essentia buscará el registro primero en este cache, y si no lo encuentra ahí, accederá al server. Si el registro es encontrado, pasará a ser registro corriente de la tabla y en el cache, ,para permitir futuros GetRecords (con THIS_KEY|IO_NOT_LOCK) para encontrarlo en el cache, evitando el acceso al server.

Esta función está dise–ada especialmente para acceder a aquellas tablas donde no es necesario obtener la última información grabada. (por ejemplo, tablas codificadoras).

SetTableCache sólo trabaja cuando Essentia está instalado.

Notas

Cuando se usa *SetTableCache*, si un registro está en el cache y el registro es modificado en la base de datos, la información en el cache no será actualizada, y estará fuera de sincronismo.

Consultar

SetConvHandler(GN)

SetDbHandler

Establece el manejador de errores de entrada/salida.

Sintaxis

```
FPLCPCP SetDbHandler(FPLCPCP f);
```

Valor de Retorno

Un puntero al manejador previo, es decir, aquel que estaba establecido antes de invocar a esta función.

Descripción

SetDbHandler establece el manejador de errores de entrada/salida. Este manejador es invocado cuando las funciones de lectura/escritura que trabajan a bajo nivel no pueden cumplir su cometido.

El manejador recibirá tres parámetros:

```
int handler(dbtable tab, int errno, char* desc)
```

Estos parámetros indican la tabla en la que se estaba trabajando cuando se produjo el error (tab), el código de error (errno) y una breve descripción de la operación que lo produjo (desc).

El manejador por defecto muestra un mensaje en la pantalla y pregunta si se desea abortar o reintentar la operación.

SwitchToSchema

Cambia el esquema corriente.

Sintaxis

```
schema SwitchToSchema(schema sch);
```

Valor de Retorno

El esquema corriente antes de cambiarlo. Si el esquema corriente no estaba definido, devuelve ERROR.

Descripción

SwitchToSchema cambia el esquema corriente al esquema indicado por sch.

TabSize

Obtiene la cantidad de registros que hay en una tabla.

Sintaxis

```
long TabActualSize(dbtable tab);
long TabNRecords(dbtable table);
```

Valor de Retorno

TabActualSize devuelve el número de registros físicos existentes en la tabla indicada.

TabNRecords devuelve el número de registros lógicos existentes en la tabla indicada.

Descripción

Estas dos funciones devuelven el número de registros de una tabla, pero mientras *TabActualSize* devuelve el número de registros físicos, *TabNRecords* devuelve el número de registros lógicos.

Un registro físico existe en la medida en que su espacio está reservado dentro de la tabla, aun cuando sus datos hayan sido borrados. *TabActualSize* puede usarse para calcular el espacio en disco que ocupa una tabla, multiplicando el valor retornado por la cantidad de bytes que ocupa cada registro, y sumándole 1024 bytes del encabezado de la tabla.

Un registro lógico es aquel que contiene datos. *TabNRecords* informa entonces cuántos registros de datos tiene una tabla.

Estas funciones no alteran el buffer de la tabla.

Ejemplo

```
#include <ideafix.h>
#include "librar.sch"
wcmd(example, %I% %G%)
{
OpenSchema("librar", IO_EABORT);
WiMsg("Number of physical records: %ld\n",
TabActualSize(BOOKS));
WiMsg("Number of books: %ld\n", TabNRecords(BOOKS));
}
```

Si la salida de este programa es -por ejemplo- 9 y 8, significa que en algún momento se dio de baja uno o más registros, y que las altas que se dieron luego, no fueron suficientes para volver a cubrir el espacio reservado para la tabla en el disco.

TimeStamp

Obtiene el *time stamp* de creación y modificación del registro corriente de una tabla.

Sintaxis

```
TIMESTAMP CTimeStamp (dbtable tab);
TIMESTAMP MTimeStamp (dbtable tab);
```

Valor de Retorno

CTimeStamp retorna en una variable de tipo TIMESTAMP los valores cdate, ctime y cuid que es el time stamp de creación que tiene el registro corriente.

MTimeStamp retorna en una variable de tipo TIMESTAMP los valores mdat, mtime y muid que constituyen el time stamp de modificación del registro corriente.

Descripción

CTimeStamp obtiene el *time stamp* de creación del registro corriente.

MTimeStamp obtiene el time stamp de modificación del registro corriente.

En caso que la tabla no posea uno de estos campos el valor correspondiente a la variable de tipo TIMESTAMP es indeterminado.

Transaction

Comienza y termina transacciones.

Sintaxis

```
void BeginTransaction ();
void EndTransaction ();
bool TransOk (void);
DoTransaction { ...(texto que incluye la transaccion
para ejecutarse)... }
```

Descripión

BeginTransaction indica a IDEAFIX que las operaciones siguientes sobre la base de datos (hasta que se encuentre una *EndTransaction* u otra *BeginTransaction*) deben ser procesados como una sola transacción lógica.

Es necesario llamar a la rutina EndTransaction luego de realizar una serie de actualizaciones (con PutRecord o con DelRecord) para que IDEAFIX actualice los índices en disco, y permita el acceso de otros programas a la información actualizada.

Las transacciones no pueden anidarse (por ejemplo, si una *BeginTransaction* es declarada, cualquier otra *BeginTransaction* luego de esta primera, será ignorada. De la misma forma si una *EndTransaction* es declarada sin declarar una *BeginTransaction* antes, será ignorada - y en cualquiera de estos casos no se mostraran errores.)

TransOk no recibe ningún parámetro y retorna TRUE o FALSE, dependiendo del estado de la transacción que se está realizando. Por este motivo, esta función debe ser insertada entre una secuencia de acción BeginTransaction / EndTransaction, o dentro de la macro DoTransaction. Una transacción se puede corromper debido a que trata de borrar o actualizar un registro bloqueado por otro proceso, o como ejemplo por el ingreso a un *deadlock*.

Para los programas que usan Essentia, la maco *DoTransaction* establece una transacción de la misma forma que las funciones *BeginTransaction* y *EndTransaction*, con la diferencia que la transacción será ejecutada incondicionalmente. Así, que si la transacción es corrompida, (por alguno de los motivos explicados anteriormente) realizará automáticamente un rollback y retirará la operación hasta que la transacción se realizada correctamente.

Ejemplo

Un ejemplo de la macro DoTransaction es el que sigue:

```
BeginTransaction();
...
PutRecord(MOV) //Update stock movements
...
GetRecord(MAE, THIS_KEY, IO_LOCK);
SetLFld(MAE_REST, LFld(MAE_REST+LFlds(MOV_QTY));
PutRecord(MAE) //Update Master Book
...
EndTransaction();
```

Será especificada como:

```
DoTransaction {
...
PutRecord(MOV) //Update stock movements
...
GetRecord(MAE, THIS_KEY, IO_LOCK);
SetLFld(MAE_REST, LFld(MAE_REST+LFlds(MOV_QTY));
PutRecord(MAE) //Update Master Book
...
}
```

Values

Obtiene valores asociados a un campo de la base de datos.

Sintaxis

```
char* InDescr (dbfield fld, char* val);
bool IsNull (dbfield fld[, int row]);
```

Valor de Retorno

InDescr devuelve la desripción asociada a un valor en la cláusula "in" de un campo. Si se produce algún error, devuelve NULL.

IsNull devuelve TRUE si el campo contiene algún valor y FALSE en caso contrario.

Descripción

InDescr devuelve la descripción asociada al valor val en la cláusula "in" del campo fld. El resultado está almacenado en un registro interno de IDEAFIX.

Devuelve NULL en los siguientes casos:

• si el campo fld no tiene cláusula "in".

• si el valor suministrado no es ninguno de los valores aceptados.

• si el esquema se abrió sin cargar su tabla de símbolos (sin la opción IO_SYMBOLS).

IsNull permite saber si un campo tiene almacenado un valor o si por el contrario está vacío. Los parámetros que recibe son el descriptor del campo que se está analizando (fld) y -si es un campo vectorizado- el número de elemento del vector (fila).

Para poder usar InDescr, se debe abrir el esquema cargando su tabla de símbolos. Esto se puede hacer de estas tres formas distintas:

```
scd = OpenSchema("esquema", IO_SYMBOLS ...)
```

```
fm = OpenForm("formulario", FM_SYMBOLS ...)
rp = OpenReport("reporte", RP_SYMBOLS ...)
```

Consultar

OpenSchema(DB)

OpenForm(FM)

OpenReport(RP)

Version

Manejo de versiones.

Sintaxis

```
long VersionNumber(char* name, DATE f);
long SchemaVersion(schema sch);
schema OpenSchemaVersion(char* name, UShort mode,
long version);
schema FindSchemaVersion(char* name, long version);
```

Valor de Retorno

VersionNumber y *SchemaVersion* devuelven el número de versión del esquema, o la constante CURRENT_VERSION si se trata de la versión actual.

OpenSchemaVersion devuelve un descriptor de esquema, o la constante ERROR si no puede realizar la tarea.

FindSchemaVersion devuelve un descriptor de esquema, o la constante ERROR si la versión buscada del esquema no ha sido abierta.

Descripción

Una versión de una base de datos es una copia del contenido de la base en una fecha determinada. Cada vez que se genera una nueva versión en el server de base de datos Essentia, se guarda el estado de las tablas en ese momento, de forma tal que en el futuro se pueda consultar la información a esa fecha, sin los cambios que se hayan hecho más adelante. Solamente se puede realizar modificaciones sobre la versión actual de un esquema, no sobre las anteriores.

VersionNumber devuelve el número de versión del esquema nombre que estaba activa en la fecha f. Si no se creó ninguna versión después de la fecha indicada, la función devuelve CURRENT_VERSION.

SchemaVersion obtiene el número de versión asociado al descriptor de esquema esq. Si el descriptor pertenece a la versión actual del esquema, devuelve CURRENT_VERSION.

OpenSchemaVersion abre la versión indicada del esquema nombre. Si el parámetro version es la constante CURRENT_VERSION, abre la última versión del esquema (la que se puede modificar).

Si la operación tiene éxito, *OpenSchemaVersion* devuelve el descriptor del esquema abierto, lo agrega a la lista de esquemas "activos" y lo marca como el esquema "corriente".

Los esquemas "activos" son aquellos a los que se puede acceder en un determinado momento, porque ya fueron abiertos con *OpenSchemaVersion*. El esquema "corriente" es aquel que se usa por defecto cuando en una operación sobre una tabla no se indica el esquema al que pertenece. El esquema corriente cambia al usar las funciones *OpenSchema*, *SwitchToSchema*, *FindNextSchema* y *OpenSchemaVersion*.

El parámetro **modo** puede ser una de estas constantes:

IO_DEFAULT es una operación normal: si ocurre un error, devuelve ERROR.

IO_EABORT si se produce un error, aborta el programa.

... | IO_SYMBOLS carga la tabla de símbolos del esquema. Esta tabla es usada -entre otras funciones- por FindDbField e InDescr.

FindSchemaVersion obtiene el descriptor correspondiente a la versión indicada del esquema nombre. Si el parámetro version es la constante CURRENT_VERSION, obtiene el descriptor de la versión actual del esquema (la que se puede modificar). La versión buscada debe haber sido abierta con *OpenSchemaVersion*.

Importante:

El manejo de versiones de una base de datos solamente se puede realizar cuando se está trabajando con Essentia, el server de base de datos de InterSoft Argentina.

Consultar

OpenSchema(DB)

FindSchema(DB)

Capítulo 23

Interfaz con Formularios (FM)

Las funciones para manejo de formularios le dan al programador, un completo control de las operaciones realizadas con un formulario compilado (recordemos que los mismos se compilan con el utilitario FGEN).

Los formularios una vez diseñados y compilados pueden ser ejecutados mediante una función de la biblioteca. Esta función permite al usuario moverse dentro de los campos y modificarlos con las teclas de cursor.

Los criterios de validación ingresados cuando el formulario fue diseñado, son aplicados interactivamente, es decir que los campos son validados "on line".

Teclas de Función

El proceso de ingreso de datos es controlado con las teclas de función. Estas están divididas en varios grupos de acuerdo a su uso. El primer grupo controla al proceso:

<PROCESAR> Procesa datos en el formulario.

<IGNORAR> Ignora los datos en el formulario, y muestra un nuevo formulario blanqueado.

<REMOVER> Realiza un operación de eliminación con los datos del formulario.

<FIN> Termina las operaciones con el formulario.

Se dan más detalles sobre estas teclas en *DoForm(FM)*.

El grupo siguiente permite el movimiento de campos. Si se está en el modo EDIT, algunas claves tienen diferentes significados. En este caso, si se establece explícitamente, el significado de la clave es el mismo a pesar del modo EDIT.

El modo EDIT es iniciado con la clave EDIT, y es indicado cambiando la forma del cursor. Cuando se deja un campo, automáticamente se sale del modo EDIT.

<PROCESAR> Se termina el ingreso en el campo corriente. El cursor se posiciona en el siguiente campo.

<CURSOR_ARR> Se va del campo en el que está y se posiciona en el campo previo. en un campo múltiple, mueve el cursor a la línea previa. Si es la primera línea, se mueve una línea hacia abajo.

<CURSOR_ABA> Se posiciona el cursor en el campo siguiente al corriente. En un campo múltiple, mueve el cursor a la próxima línea.

<CURSOR_IZQ> En el modo EDIT: mueve el cursor una posición a la izquierda, sin borrar el carácter. Si no estamos en modo EDIT, se posiciona en el siguiente campo de la derecha.

<CURSOR_DER> En modo EDIT: mueve el cursor una posición a la derecha. De otro modo, se posiciona en el siguiente campo de la izquierda.

<PAG_ANT> En campos múltiples, mueve hacia arriba una página completa.

<PAG_SIG> En campos múltiples, mueve hacia abajo una página completa.

Las claves utilizadas cuando se edita información en un campo son:

<EDIT> Cambio de modo de edición. Cuando se está en el modo EDIT, se indica mediante el cambio de la forma del cursor.

<INS> Inserta un carácter en modo EDIT . Si no se está en modo de edición y está posicionado en un campo múltiple, inserta una línea en blanco. En otro momento se ignora.

<DEL> Borra un carácter en modo EDIT. Si no se está en modo de edición y está posicionado en un campo múltiple, borra una línea. En otro momento se ignora.

<BORRA_CAMPO> Borra el campo completo.

<RETROCESO> Mueve el cursor hacia la izquierda un carác ter, borrando el previo.

Finalmente, hay teclas de función para ayuda:

<AYUDA> Despliega un mensaje de ayuda acerca del campo corriente.

<AYUDA_APL> Despliega un mensaje de error acerca de la aplicación corriente.

Debe recordarse que hay otras teclas de función manejadas directamente por el Manejador de Ventanas (Window Manager), que trae la configuración del teclado, facilidades de Calculadora, Agenda y Calendario, Interrupción, Suspensión, etc.

El Archivo .*fmh*

Este archivo puede ser generado con el utilitario FGEN (ver la sección *Generando el Formulario*) o bien indicándolo en las especificaciones generales de la definición del formulario (%form).

Contiene constantes de tipos fm_field que son utilizados para referenciar un campo en

particular con las funciones de formularios.

Como un ejemplo, se muestra el archivo cabecera generado para el formulario libros presentado en el *Capítulo 3 del Manual de este Manual*:

```
/* Field names */
# define CODIGO (fmfield) 0 /* Integer */
# define TITULO (fmfield) 1 /* String */
# define AUTOR (fmfield) 2 /* Integer */
# define DESC0 (fmfield) 3 /* String */
# define EDICION (fmfield) 4 /* Integer */
# define FECHA (fmfield) 5 /* Date */
/* Subform names */
/* Symbol names for messages */
/* ========== End of fmh file ========== */
```

Descriptores de Formularios

Para realizar cualquier tipo de operación en un formulario, es necesario asociar a la función apropiada un descriptor de formulario. Un descriptor de formulario es una variable de tipo form. Este tipo es definido en una sentencia *typedef* en el archivo cabecera *fm.h*.

Los descriptores de formularios se obtienen con la función *OpenForm*, y son la clave para acceder a todos los datos contenidos en un formulario. Estos datos se mantienen internamente en registros de almacenaje interno direccionados, y la única forma de acceder a ellos es a través de funciones especiales. Para identificar los elementos de un formulario existen dos tipos de descriptores:

Descriptor	Tipo de Variable
formulario	form
campo	fmfield

Mensajes de Formularios

Los mensajes definidos en la zona %form de un formulario pueden ser desplegados en pantalla mediante funciones especiales (ej.: FmErrMsg). Para referenciarlos es necesario anteponerles el prefijo "M_" (por ejemplo, M_NOMBRE).

Entonces, habiendo definido un mensaje:

```
.....
%form use biblio;
messages ERRCOD:"El código de libro ingresado no
existe.";
.....
```

desde el fuente CFIX será necesario invocarlo como M_ERRCOD.

Subformulario

De modo similar al anterior, los subformularios dentro del archivo con extensión *.fmh*, se identifican anteponiéndoles el prefijo SF_. Habiendo definido:

```
%fields
.....
campo : subform ("bib105", "bib106", "bib107");
```

en el archivo de encabezado se denominarán SF_BIB105, SF_BIB106 y SF_BIB107 respectivamente.

Comandos de Formularios

Las funciones de formulario retornan en algunos casos un "comando de formulario". Estos son valores que reflejan las acciones del operador. Un tipo enumerativo llamado fm_cmd describe los valores posibles.

FM_EXIT	se presionó la tecla FIN
FM_DELETE	se presionó la tecla REMOVER
FM_IGNORE	se presionó la tecla IGNORAR
FM_READ	la clave está completa
FM_READ_NEXT	la clave contiene el registro anterior al que se desea procesar.
FM_READ_PREV	la clave contiene el registro siguiente al que se desea procesar.
FM_ADD	se presionó la tecla PROCESAR y el registro no existe en la tabla.
FM_UPDATE	se presionó la tecla PROCESAR y el registro ya existía en la tabla.

Archivos de Inclusión

Un archivo cabecera llamado *fm.h* contiene algunas constantes necesarias y definiciones de tipos. En algunos casos otros archivos de inclusión son necesarios. El encabezamiento ideafix.h incluye automáticamente todos los archivos cabecera necesarios de IDEAFIX.

Lista de Funciones

El primer cuadro muestra las páginas documentadas referentes a Formularios.

Sección	**Tarea Realizada por la sección de la**

	función
Close	Cierra un formulario.
DisplayForm	Muestra y oculta un formulario.
DoForm	Opera con un formulario.
FmCheckSum	Calcula el ÔÔchecksum" de la definición de un formulario.
FmOnKey	Actúa en un grupo de campos, al requerirse por el usuario.
FmRecalc	Recalcula los valores de los campos con atributo is.
FmSetDisplayOnly	Cambia el atributo display only de un rango de campos.
FmToDb	Copia valores entre un formulario y una base de datos.
Get	Obtiene el valor de un campo del formulario.
InOffSet	Obtiene/establece el índice IN que será utilizado.
OpenForm	Abre un formulario.
Set	Establece el valor de un campo del formulario.
Show	Muestra en la pantalla uno o más campos de un formulario.
Status/Error	Establece/Obtiene el estado/error de un formulario.
Subform	Obtiene el descriptor o el identificador de un subformulario.
Values	Obtiene o establece varios valores asociados con un formulario.

Referencia de Funciones

Clear

Blanquea los valores de los campos de un formulario.

Sintaxis

```
void FmClearFlds (form fmd, fmfield from, fmfield to[, int
row]);
void FmClearAllFlds (form fmd);
```

Descripción

FmClearFlds pone en blanco uno o más campos del buffer de un formulario. Esta función actúa solamente sobre la memoria: para que los cambios aparezcan en la pantalla, hay que usar *FmShowFlds*.

Los parámetros de *FmClearFlds* son:

fmd es el descriptor del formulario que se obtuvo con *OpenForm*.

from descriptor del primer campo que se va a blanquear.

until descriptor del último campo que se va a blanquear.

string si es un campo múltiple, indica la fila afectada por la opera ción.

Cabe recordar que los campos de un formulario se enumeran de izquierda a derecha y de arriba hacia abajo.

Un campo múltiple se blanquea completo si está incluido entre los campos desde y hasta. Si desde y hasta están dentro de un campo múltiple, se debe usar el parámetro fila para indicar qué fila se quiere dejar en blanco.

FmClearAllFlds blanquea todos los campos del formulario indicado.

Consultar

FmShowFlds(FM)

CloseForm

Cierra un formulario.

Sintaxis

```
void CloseForm (form fmd);
void CloseAllForms ();
```

Descripción

CloseForm cierra el formulario cuyo descriptor es fmd. Toda la memoria asignada a ese formulario se libera.

CloseAllForms cierra todos los formularios que estén abiertos.

DisplayForm

Muestra y oculta un formulario.

Sintaxis

```
void DisplayForm (form fmd);
void ClearForm(form fmd);
```

Descripción

DisplayForm muestra en la pantalla el formulario cuyo descriptor es fmd. Es útil cuando el formulario fue abierto con el modo FM_NODISPLAY. Si el formulario tiene el estado FM_USED, los datos que contiene serán mostrados; si no, se desplegará un formulario en blanco.

ClearForm saca de la pantalla el formulario fmd. Se usa para ocultar un formulario que fue desplegado con DisplayForm.

DoForm

Opera con un formulario.

Sintaxis

```
fm_cmd DoForm (form fmd, fm_status_fp before, fm_status_fp
after);
fm_cmd DoSubform (form fmd,fm_status_fp before,fm_status_fp
after, fmfield fld, UShort subf_n[, int row]);
```

Valor de Retorno

Una de las siguientes constantes:

FM_EXIT si se apretó la tecla FIN.

FM_DELETE si se apretó la tecla REMOVER.

FM_IGNORE si se apretó la tecla IGNORAR.

FM_READ si se completó los campos que forman la clave.

FM_READ_NEXT si los campos que forman la clave contienen el registro anterior al que se desea procesar.

FM_READ_PREV si los campos que forman la clave contienen el registro siguiente al que se desea procesar.

FM_READ_PREV si se apretó la tecla PROCESAR y el registro no existe en la tabla.

FM_UPDATE si se apretó la tecla PROCESAR y el registro ya existía en la tabla.

Descripción

DoForm es una rutina que permite la carga de datos en forma interactiva, utilizando un formulario que fue compilado con el utilitario fgen. El parámetro fmd es el descriptor del formulario, que se obtiene con *OpenForm*.

DoForm asume que el formulario fue desplegado en la pantalla.

Los parámetros before y after son punteros a funciones que serán invocadas antes de entrar y después de salir de cada campo, respectivamente. Estas funciones se suelen usar para realizar todas aquellas validaciones que no pueden efectuarse directamente desde el formulario. Si no se va a usar las funciones before y after, estos dos parámetros deben ser NULLFP (esta constante está definida en el archivo "gn.h").

DoSubform realiza la misma tarea que *Doform*, pero sobre un subformulario asociado a un campo del formulario principal. fmd es el descriptor del formulario padre, fld es el descriptor del campo al que está vinculado el subformulario y subf_n es el número de subformulario asociado a ese campo (los subformularios se numeran a partir de cero). El parámetro fila se usa si el campo fld forma parte de un multirrenglón, para indicar qué fila se debe tomar.

El Estado del Formulario

Cada formulario tiene un estado asociado, y puede ser uno de los siguientes:

FM_NEW si el formulario aún no ha sido usado.

FM_USED si ya se modificó el contenido de uno o más campos.

FM_ERROR si se produjo algún error.

FM_PAGE_DOWN se apretó la tecla PAGE_DOWN para pasar a otro registro.

FM_PAGE_UP se apretó la tecla PAGE_UP para pasar a otro registro.

FM_LOCKED si el registro que contiene el formulario está bloqueado por otro programa.

Según el estado de un formulario, DoForm devuelve distintos valores cuando se aprieta una tecla de función.

El estado de un formulario puede ser modificado por medio del comando *FmSetStatus*:

```
FmSetStatus(fmd,valor)
```

IDEAFIX también utiliza el estado del formulario en el momento de iniciar el proceso de carga de datos, de la siguiente forma:

FM_NEW blanquea el formulario y posiciona el cursor en el primer campo.

FM_USED despliega en la pantalla los datos del formulario y posiciona el cursor en el campo de control.

FM_LOCKED muestra un mensaje de error indicando el bloqueo y posiciona el cursor en el primer campo.

FM_EOF muestra un mensaje de error indicando el fin de archivo y posiciona el cursor en el primer campo.

FM_ERROR despliega los datos del formulario y posiciona el cursor en el campo indicado por FmSetError.

FmSetError se usa de la siguiente forma:

```
FmSetError(fmd, fld, errno)
```

fmd es el descriptor del formulario, fld es el descriptor del campo en el que se va a posicionar el cursor, y errno es el número de mensaje de error que se muestra.

Formularios sin Area de Clave

La primera vez que se invoca a la función *DoForm*, ésta posiciona el cursor en el primer campo, y comienza el proceso de carga, hasta que se digite una de las teclas correspondientes.

Si se invoca a DoForm por segunda vez, se blanquea todos los campos del formulario y se repite el mismo proceso de carga de datos (posicionamiento en el primer campo, etc.).

Si un campo tiene definida una máscara, la entrada se validará a medida que se escribe. El resto de las validaciones (in, ÔÔ>", ÔÔ<", etc.) se realizan al terminar de ingresar el campo.

El operador puede moverse entre los campos con las teclas de movimiento del cursor y editarlos, como se explicó en la introducción de esta sección.

Formularios con Area de Clave

Si el formulario tiene definida un área de claves (Ver *Capítulo 3, Formularios con Zona de Claves*), *DoForm* espera hasta que se complete la clave, y después retorna con el valor FM_READ. La única tecla que hace retornar a *DoForm* antes de que se complete la clave es la tecla END, que devuelve el valor FM_EXIT.

Cuando el cursor sale de la zona de claves y pasa a la zona de datos, el comportamiento de *DoForm* es el mismo que el explicado en el apartado anterior.

Procesamiento de Campos

Cada campo es procesado de la siguiente forma:

1. Se llama a la función before. Si esta función devuelve FM_SKIP, IDEAFIX saltea el campo y pasa al siguiente. Si no, continúa con el paso 2.

2. Se acepta la entrada del campo. En ese momento, se realizan algunas validaciones (por

ejemplo, un campo numérico acepta sólo dígitos, se chequea la consistencia de las fechas, etc.)

3. Se realizan otras validaciones especificadas en la definición del formulario. Si no se verifican, IDEAFIX vuelve al paso 2.

4. Se llama a la función after, siempre y cuando no se haya apretado la tecla <FIN>,< REMOVER> o <IGNORAR>.

Las funciones *before* y *after* tienen la siguiente sintaxis:

```
fm_status before (form fmd, fmfield fld, int fila)
fm_status after (form fmd, fmfield fld, int fila)
```

Las funciones reciben el descriptor del formulario en fmd y el descriptor del campo en fld. El tercer parámetro es el número de fila si es un campo múltiple, o cero en cualquier otro caso.

La función before tiene que devolver una de las siguientes constantes, según la acción que se quiera realizar:

FM_SKIP saltear este campo y pasar al siguiente. Si es un campo multirrenglón, lo saltea completo.

FM_OK continuar el procesamiento normal de este campo.

La función after debe devolver una de las siguientes constantes:

FM_OK aceptar la entrada y pasar al siguiente campo.

FM_REDO rechazar la entrada y volver al paso 2.

FM_ERROR es equivalente a FM_REDO.

FM_LOCKED rechazar la entrada, desplegar el mensaje de error predefinido y volver al paso 2.

Ejemplo

```
#include <ideafix.h>
#include "books.fmh"
#include "librar.sch"
/* Private functions */
private fm_status before(form, fmfield, int);
private fm_status after(form, fmfield, int);
private void Lectura(fm_cmd, find_mode);
/* global declarations */
form fm0;
/* main program */
wcmd(example, %I% %G%)
{
fm_cmd cmd;
fm0 = OpenForm("books", FM_EABORT);
while ((cmd = DoForm(fm0, before, after)) != FM_EXIT)
switch (cmd) {
```

```
case FM_READ:
/* search for the record */
Lectura(cmd, THIS_KEY); break;
case FM_READ_NEXT:
Lectura(cmd, NEXT_KEY); break;
case FM_READ_PREV:
Lectura(cmd, PREV_KEY); break;
case FM_ADD:
/* If it is an add, prepare blank record*/
InitRecord(BOOKS);
case FM_UPDATE:
/* If it is an update, save the record */
BeginTransaction();
FmToDb(fm0, 0, CONTROL_FLD, 0);
PutRecord(BOOKS);
EndTransaction();
break;
case FM_DELETE:
/* If it is a delete, delete the record*/
BeginTransaction();
FmToDb(fm0, 0, CODE);
DelRecord(LIBROS);
EndTransaction();
break;
case FM_IGNORE:
/* If we must ignore, free the contents of the buffer */
FreeTable(BOOKS);
break;
}
}
/* read-and-display routine */
private void Lectura(fm_cmd cmd, find_mode mode)
{
FmToDb(fm0, 0, CODIGO);
switch(GetRecord(BOOKSbyCODE,mode, IO_LOCK|IO_TEST))
{
case IO_LOCKED:
FmSetStatus(fm0, FM_LOCKED);
(void)FindRecord(BOOKSbyCODE, mode);
DbToFm(fm0, 0, CODE);
FmShowFlds(fm0, 0, CODE);
return;
case ERROR:
FmSetStatus(fm0, cmd==FM_READ ? FM_NEW : FM_EOF);
return;
}
DbToFm(fm0, 0, CONTROL_FLD, 0);
FmShowFlds(fm0, 0, CODE);
}
/* Routine executed before entering each field*/
private fm_status before(form fm, fmfield fno, int row)
{
switch (fno) {
case CODE:
break;
case TITLE:
break;
case AUTHOR:
break;
case EDITION:
break;
```

```
case DATE:
break;
}
return FM_OK;
}
/* Routine executed when leaving each field */
private fm_status after(form fm, fmfield fno, int row)
{
switch (fno) {
case CODE:
break;
case TITLE:
break;
case AUTHOR:
break;
case EDITION:
break;
case DATE:
break;
}
return FM_OK;
}
```

FmCheckSum

Calcula el "checksum" de la definición de un formulario.

Sintaxis

```
long FmCheckSum (form fmd);
void FmVerifyCheckSum (form fmd, long chksum);
```

Descripción

Cuando se compila un formulario con fgen -h o se incluye la cláusula language en su definición, IDEAFIX genera un archivo con el mismo nombre del formulario y extensión ".fmh". En este archivo, define una constante llamada FM_fm_CHKSUM (donde fm es el nombre del formulario) cuyo valor depende de los campos, tablas, etc. que lo componen. Si se modifica la definición del formulario, pero no se actualiza el archivo ".fmh", se pueden producir errores en los programas.

Para detectar esta situación, se utilizan las funciones *FmCheckSum* y *FmVerifyCheckSum*.

FmCheckSum devuelve el valor que debería tener FM_fm_CHKSUM según la definición actual del formulario. Comparando la constante con el valor obtenido, se podrá saber si el archivo ".fmh" está actualizado o no.

FmVerifyCheckSum realiza la comparación directamente, y si el archivo está desactualizado, emite un mensaje informando el error. El segundo parámetro debe ser la constante FM_fm_CHKSUM.

Ejemplo

```
#include <ideafix.h>
#include "books.fmh"
wcmd (example, %I% %G%)
{
form fm0;
fm0 = OpenForm ( "books" , FM_EABORT);
FmVerifyCheckSum (fm0, FM_BOOKS_CHKSUM);
}
```

FmOnKey

Relaciona una tecla a una función que modificará un grupo de campos en un formulario.

Sintaxis

```
void FmOnKey(form fm, int fkey, fm_status_fp fp,
fmfield from, fmfield to);
```

Descripción

Esta función relaciona una tecla a la ejecución de una rutina para un grupo de campos dentro de un formulario. La rutina es del mismo tipo que las rutinas after/before. Su comportamiento es similar al del after, debido a que si se retorna FM_REDO, se mantendrá en el campo donde estaba el doform, y si se retorna FM_SKIP el cursor se mueve al próximo campo, etc.

Esta función puede ser aplicada al codificar la ejecución de una función, en al *after* para un grupo de campos, cuando el *FmKeyCode* retorna una tecla predefinida.

FmRecalc

Recalcula los valores de los campos con atributo is.

Sintaxis

```
void FmRecalc (form fmd, UShort row);
```

Descripción

Los campos con atributo is se recalculan automáticamente al pasar por una función after. Si por alguna razón hay que recalcularlos en otro momento, se usa *FmRecalc*.

Una situación en que sería necesario usar *FmRecalc* es la siguiente:

Se tiene un campo del formulario ("suma") que es la suma de una columna de un multirrenglón. Si en la función before se modifica el valor de un campo del multirrenglón y se devuelve FM_SKIP, IDEAFIX no ejecutará la función after, y por lo tanto el campo "suma" quedará desactualizado. Por eso, es necesario invocar a FmRecalc antes de salir de la función before.

FmSetDisplayOnly

Cambia el atributo display only de un rango de campos.

Sintaxis

```
void FmSetDisplayOnly (form fm, fmfield from, fmfield to, bool
value);
```

Descripción

FmSetDisplayOnly cambia el atributo display only de todos los campos del formulario fm comprendidos entre desde y hasta, ambos inclusive.

Si value es verdadero, pone el atributo. Si es falso, lo saca.

FmToDb

Copia valores entre un formulario y una base de datos.

Sintaxis

```
void FmToDb (form fmd, fmfield from, fmfield to[, int row]);
void DbToFm (form fmd, fmfield from, fmfield to[, int row]);
```

Descripción

Estas funciones copian información entre un formulario y el buffer de las tablas asociadas.

FmToDb copia los datos de los campos indicados del formulario (todos los que están entre desde y hasta) a los buffers de tabla respectivos.

DbToFm copia los datos de los campos indicados del buffer de una tabla a los campos respectivos del formulario.

En ambos casos, los campos que no tienen ninguna relación con la base de datos (o con el formulario) permanecen inalterados.

Si se está copiando un campo múltiple, hay que indicar la fila que se desea copiar al buffer de la tabla.

Get

Obtiene el valor de un campo del formulario.

Sintaxis

```
char* FmSFld (form fmd, fmfield fld[,int fila]);
int FmIFld (form fmd, fmfield fld[,int fila]);
long FmLFld (form fmd, fmfield fld[,int fila]);
double FmFFld (form fmd, fmfield fld[,int fila]);
TIME FmTFld (form fmd, fmfield fld[,int fila]);
DATE FmDFld (form fmd, fmfield fld[,int fila]);
NUM FmNFld (form fmd, fmfield fld[,int fila]);
void FmGetFld (char* buf, form fmd, fmfield fn[, int fi la]);
fmfield FindFmField (form fm, const char *name);
int FmNextFld (form fmd, fmfield fno, [int nofila]);
```

Descripción

Estas funciones devuelven el contenido de un campo del formulario. fmd es el descriptor del formulario y fld es el descriptor del campo.

El parámetro fila es utilizado en los campos múltiples para indicar qué fila debe ser utilizada.

FmGetFld y *FmSFld* obtienen el valor del campo como una cadena de caracteres. *FmGetFld* copia esta cadena en la dirección apuntada por buf (que debe tener suficiente espacio para contenerla). *FmSFld* copia esta cadena en un registro interno de IDEAFIX, y devuelve su dirección; este registro interno se sobreescribe en cada llamada a la función.

Las demás funciones *Get* devuelven el valor del campo en diferentes tipos de datos del lenguaje C. Si el valor de un campo no puede ser convertido al tipo de dato (por ejemplo, si se lee un campo alfanumérico con *FmIFld*) se invoca al manejador de error de conversión de tipos. Si éste retorna el control a la función, devolverá el valor NULL correspondiente a ese tipo.

FindFmField recibe como parámetros un descriptor de formulario y el nombre de un campo que se pueda encontrar en la sección %fields del mismo, retornando el correspondiente descriptor de campo. Es similar a *FindDbField*.

FmNextFld rutea el camino del cursor para permitir especificar cuál es el próximo campo que será tomado como entrada en un formulario. Puede retornar ERROR si el campo especificado es inválido o del tipo «*skip*«, y OK de cualquier otra forma.

Consultar

SetConvHandler(GN)

InOffSet

Obtiene/establece el índice IN que será utilizado.

Sintaxis

```
UShort FmInOffset (form fmd);
void FmSetInOffset(form fmd, UShort n);
```

Descripción

FmInOffset devuelve un número que indica cuál de los valores de un IN ha sido seleccionado en el campo actual.

FmSetInOffset selecciona el n-ésimo valor de un IN para el campo actual.

OpenForm

Abre un formulario.

Sintaxis

```
form OpenForm (char* name, int flags);
```

Valor de Retorno

Si puede abrir el formulario, devuelve su descriptor. De lo contrario, devuelve ERROR.

Descripción

OpenForm abre un formulario, cuyo nombre se indica en una cadena de caracteres apuntada por nombre. IDEAFIX buscará un archivo llamado "nombre.fmo" en los directorios indicados en la variable de ambiente PATH. Este archivo debe contener el formulario compilado con el utilitario fgen.

Si la operación fue ejecutada con éxito, *OpenForm* devuelve el descriptor del formulario, que se usará en todas las funciones que trabajen sobre ese formulario.

El parámetro flag puede ser:

FM_NORMAL abre el formulario y lo despliega en la pantalla.

FM_NODISPLAY abre el formulario, pero no lo muestra en la pantalla.

FM_EABORT si ocurre un error, aborta el programa.

FM_SYMBOLS carga en memoria la tabla de símbolos del for q mulario.

La opción FM_SYMBOLS es obligatoria si se va a utilizar ciertas funciones especiales, como por ejemplo InDescr.

Ejemplo

```
#include < ideafix.h >
wcmd (example,% I%% G%)
{
form fmd;
fmd = OpenForm ( "librar", FM SYMBOLS | FM EABORT);
/* etc... */
}
```

Consultar

DisplayForm(FM)

ClearForm(FM)

Set

Establece el valor de un campo del formulario.

Sintaxis

```
void FmSetFld (form fmd, fmfield fld, char* buf[, int fi la]);
void FmSetIFld (form fmd, fmfield fld, int val[, int fi la]);
void FmSetLFld (form fmd, fmfield fld, long val[, int fi la]);
void FmSetFFld (form fmd, fmfield fld, double val[, int
fila]);
void FmSetDFld (form fmd, fmfield fld, DATE val[, int fi la]);
void FmSetTFld (form fmd, fmfield fld, TIME val[, int fi la]);
void FmSetMask (form fmd, fmfield fld, char* mask);
```

Descripción

Estas funciones colocan un dato en un campo del buffer de un formulario. fmd es el descriptor del formulario, fld es el descriptor del campo, y fila es el número de fila dentro del campo, si éste es un campo múltiple.

FmSetFld copia la cadena de caracteres apuntada por buf en el campo seleccionado. Los caracteres son copiados hasta que la cadena termina, o hasta que el campo no tiene más espacio.

Las siguientes funciones *Set* colocan el valor val en el campo, y sólo se diferencian entre sí por el tipo de dato que manejan. Si val no puede ser convertido al tipo del campo (por ejemplo, si se usa *FmSetIFld* con un campo alfanumérico), se invoca al manejador de error de conversión de tipos.

FmSetMask coloca la máscara mask al campo fld.

Consultar

DoForm(FM)

Get(FM)

SetConvHandler(GN)

Show

Muestra en la pantalla uno o más campos de un formulario.

Sintaxis

```
void FmShowFlds (form fmd, fmfield from, fmfield to[, int
row]);
void FmShowAllFlds (form fmd);
```

Descripción

FmShowFlds despliega el contenido de los campos indicados del buffer del formulario en la pantalla.

fmd es el descriptor del formulario (que se obtuvo con *OpenForm*). desde y hasta especifican el rango de campos que van a ser desplegados: todos los campos entre desde y hasta -ambos inclusive- aparecerán en la pantalla.

Un campo múltiple es desplegado si está entre desde y hasta, o si los dos parámetros son iguales al número que lo representa. Si desde y hasta están dentro del campo múltiple, el parámetro fila indica qué fila será desplegada.

FmShowAllFlds muestra todos los campos del formulario.

Status

Establece/Obtiene el estado/error de un formulario.

Sintaxis

```
void FmSetStatus (form fmdd, fm_status state);
void FmSetError (form fmd, fmfield fn, ...);
fm_status FmStatus (form fmd);
fm_status FmErrMsg (form fmd, int msg, ...);
```

Descripción

Estas funciones se utilizan junto con *DoForm*, para poner el estado del formulario antes de entrar a esa función, o para ver en qué estado quedó al salir de ella.

FmSetStatus establece el estado de un formulario, que luego será utilizado por *DoForm*.

El estado puede ser uno de los siguientes:

FM_NEW si el formulario aún no ha sido usado.

FM_USED si ya se modificó el contenido de uno o más campos.

FM_ERROR si se produjo algún error.

FM_LOCKED si el registro que contiene el formulario está bloqueado por otro programa.

FM_PAGE_DOWN se apretó la tecla PAGE_DOWN para pasar a otro registro.

FM_PAGE_UP se apretó la tecla PAGE_UP para pasar a otro registro.

FmStatus devuelve el estado actual del formulario.

FmErrMsg emite el mensaje asociado a msj (que está en la definición del form fmd) y devuelve FM_ERROR.

En las funciones *FmSetError* y *FmErrMsg*, se puede indicar una serie de parámetros adicionales, que se usarán para completar el mensaje mostrado. Por ejemplo, si se definió el siguiente mensaje en el archivo ".fm":

```
%form
messages ENT_INVALIDA: "El campo %s no puede aceptar ese
dato."
```

Para mostrarlo con FmErrMsg, habría que hacer:

```
FmErrMsg(fm0,M_ENT_INVALIDA,"autor")
```

Consultar

DoForm(FM)

SubForm

Obtiene el descriptor o el identificador de un subformulario.

Sintaxis

```
form UseSubform (form fmd, fmfield fld, UShort subf_n[, int
row]);
int FmSubformId (form fmd);
```

Valor de Retorno

UseSubform devuelve el descriptor del subformulario.

FmSubformId devuelve el identificador del subformulario.

Descripción

FmSubformId devuelve el identificador del subformulario fmd, es decir, el número de subformulario dentro del formulario principal. Se usa generalmente en las funciones *after* y *before*, para saber a qué formulario pertenece el campo que se está procesando.

UseSubform se utiliza para obtener el descriptor de un subformulario asociado a un campo.

La función mostrará un mensaje de error si el campo fld no tiene subformularios asociados.

El tercer parámetro, *subf_n*, indica cuál de los subformularios asociados al campo fld se quiere usar. Los subformularios se numeran para cada campo, empezando en 0 (cero). Si el subformulario está asociado a un campo que está dentro de un multirrenglón, se debe indicar también el número de fila. Esto es así porque los subformularios se asocian a cada renglón del campo múltiple por separado.

Para cada campo, se reserva la cantidad de memoria correspondiente al subformulario de mayor longitud.

UseSubform devuelve el descriptor del subformulario abierto, y prepara el buffer correspondiente.

Ejemplo

```
#include <ideafix.h>
#include "librar.fmh"
#include "librar1.fmh"
wcmd(example, %I% %G%)
{
form fm1, fm2;
fm1 = OpenForm("librar", FM_EABORT);
fm2 = UseSubform(fm1, EDITION, 0);
if (!FmIsNull(fm1, EDITION))
FmToDb(fm2, DATE, DATE);
/* etc... */
}
```

En este ejemplo, se tiene un formulario principal (biblio) y un subformulario (biblio1) asociado al campo EDICION del formulario principal. Con *UseSubform* se abre el subformulario asociado.

Si el campo EDICION hubiese tenido más de un subformulario asociado, se debería haber indicado cuál de ellos había que abrir, escribiendo el número en lugar del cero.

```
#include <ideafix.h>
#include "librar.fmh"
#include "librar1.fmh"
#include "librar.sch"
/* private functions */
private fm_status after(form, fmfield, int);
/* global declarations */
form fm0, fm1;
/* main program */
wcmd(librar, %I% %G%)
{
fm_cmd cmd;
fm0 = OpenForm("librar", FM_EABORT);
cmd = DoForm(fm0, NULLFP, after);
/* etc... */
}
private fm_status after(form fm, fmfield fno, int row)
{
if (FmSubformId(fm) == SF_LIBRAR1) {
switch (fno) {
case DATE
break;
}
return FM_OK;
}
switch (fno) {
case CODE:
break;
case TITLE:
break;
case AUTHOR:
break;
case EDITION:
break;
}
return FM_OK;
}
```

En este segundo ejemplo, se utiliza la función FmSubformId dentro de la función after, para verificar si el formulario corriente es el subformulario biblio1. Si es así, se realiza la acción correspondiente a cada uno de sus campos.

Nótese que el valor de SF_BIBLIO1 no es necesariamente el número de subformulario asociado al campo. Casualmente pueden coincidir, ya que el primero es el número de subformulario declarado en el formulario principal, contando de izquierda a derecha y de arriba a abajo.

Values

Obtiene o establece varios valores asociados con un formulario.

Sintaxis

```
char* FmFldPrev (form fmd);
int FmFldLen (form fmd, fmfield fld);
bool FmChgFld (form fmd);
void FmSetKeyCode (form fmd, UChar code);
UChar FmKeyCode (form fmd);
char* FmInValue (form fmd, fmfield fld, int n);
char* FmInDescr (form fmd, fmfield fld, int n);
bool FmIsNull (form fmd, fmfield fld[, int row]);
fmfield FmInMult (form fmd fmfield fld[, int row]);
UShort FmSubfRow (form fmd);
form FmFather (form fmd);
int FmNKeys(form fm);
bool FmIsDisplayOnly (form tbl, fmfield fld);
void SetUpdatePerm(bool v);
void SetAddPerm(bool v);
void SetDelPerm(bool v);
fmfield FmRefFld(form f, fmfield fld, ....);
```

Descripción

Estas funciones obtienen y establecen varios valores asociados con el formulario cuyo descriptor es fmd.

FmFldPrev devuelve un puntero a un registro interno de IDEAFIX, que contiene el valor previo del campo corriente. Solamente se usa en el post-condicional (la función *after*).

FmFldLen devuelve la longitud de un campo en caracteres, tal como está definido en la pantalla. Si el campo es múltiple, devuelve la cantidad de filas que tiene.

FmChgFld devuelve TRUE si el campo fue modificado en la última pasada. Unicamente es válido en la función after.

FmSetKeyCode establece a código como la última tecla presionada en el formulario. Sólo se usa en la función after.

FmKeyCode devuelve el código de la última tecla de función apretada. Si se usa en la función after, indica qué tecla hizo salir el cursor del campo, y provocó la llamada a esa función.

Los códigos de teclas utilizados por las funciones *FmSetKeyCode* y *FmKeyCode* son los siguientes:

K_PROCESS Procesar

K_ENTER Enter/Return/Intro.

K_CURS_UP Flecha hacia arriba.

K_CURS_LEFT Flecha hacia izquierda.

K_CURS_RIGHT Flecha hacia derecha.

K_CURS_DOWN Flecha hacia abajo.

K_PAGE_UP Página Arriba/Page Up.

K_PAGE_DOWN Página Abajo/Page Down.

K_PAGE_LEFT Página Izquierda.

K_PAGE_RIGHT Página Derecha.

K_BACKSPACE Retroceso/Backspace.

K_TAB Tabulador.

K_HELP Ayuda (<dg>).

K_META Meta (<dg>).

K_CTRLX Ctrl+X.

FmInValue devuelve un puntero a un registro interno de IDEAFIX, que contiene el n-ésimo valor de la cláusula IN del campo fld.

FmInDescr devuelve un puntero a un registro interno de IDEAFIX, que contiene la descripción asociada al n-ésimo valor de la cláusula IN del campo fld.

FmIsNull devuelve TRUE si el campo fld está vacío, y FALSE si contiene algo. El parámetro fila se usa si fld es un campo múltiple.

FmInMult devuelve el descriptor del campo virtual del multirrenglón al que pertenece el campo fld. Si ese campo no está incluido dentro de un multirrenglón, devuelve ERROR.

FmSubfRow devuelve el renglón actual del subformulario.

FmFather devuelve el descriptor del formulario "padre" (el formulario del cual depende un subformulario).

FmNKeys retorna un número entero con la cantidad de campos que conforman la clave del formulario.

FmIsDisplayOnly retorna TRUE si el campo fld del formulario fmd tiene atributo display-only.

SetUpdatePerm, *SetAddPerm* y *SetDelPerm*, permiten al programador establecer los permisos del formulario, usualmente dados por la línea de comando, para la actualización, alta o baja de registros (por ejemplo, para restringir algunas de estas operaciones en un formulario dado).

FmRefFld retorna el descriptor de campo que corresponde al tipo corrinte en un momento dado.

Capítulo 24

Generales (GN)

La biblioteca de aplicaciones incluye funciones generales, que realizan ciertas tareas tales como:

• Operaciones iniciales al comienzo de un programa.

• Manejo de Memoria.

• Otras operaciones generales.

Lista de Funciones

El siguiente cuadro muestra las páginas documentadas en la sección dedicada a Funciones Generales.

Sección	Tareas Realizadas por la sección de la Función
Alloc	Obtiene un espacio de memoria libre.
BaseName	Separa el nombre de archivo de un path completo.
Check	Establece/Obtiene el dígito verificador.
Conv	Convierte datos de diversos tipos.
CUserId	Obtiene el nombre de login de un usuario.
Error	Maneja el despliegue de la información de errores.
FopenPath	Abre un archivo buscando según la variable de ambiente PATH.
FreeMem	Obtiene la cantidad de memoria libre.
GetPWEntry	Busca un usuario en el archivo /etc/passwd.
Locate	Busca en forma binaria en una tabla en memoria.

Modules	Modula información.
OpenPrinter	Abre una impresora.
Proc	Atributos de proceso
SetConvHandler	Activa el manejador de errores de conversión de tipos.
SetReadEnvHandler	Activa el manejo de errores de lectura en variables.
Stop	Termina un proceso.

Referencia de Funciones

Alloc

Obtiene un espacio de memoria libre.

Sintaxis

```
void* Alloc (unsigned n);
FPCPIU SetAllocHandler (FPCPIU f);
```

Valor de Retorno

Alloc devuelve un puntero a una zona de memoria libre de n bytes.

SetAllocHandler devuelve un puntero al manejador previo.

Descripción

Alloc obtiene del sistema operativo una zona de memoria libre, para almacenar n bytes.

En UNIX, esta función equivale a alloc.

En DOS, si no hay suficiente memoria disponible, se invoca a la función indicada con *SetAllocHandler*. El manejador por defecto aborta la ejecución con un mensaje de error apropiado.

SetAllocHandler indica a IDEAFIX qué función debe llamar (f) cuando se usa Alloc y no queda más memoria libre.

La función f recibirá el mismo parámetro que recibió Alloc, y si no aborta el programa, el valor devuelto por f será el valor devuelto por Alloc.

Nota

Alloc obtiene memoria para almacenar cualquier clase de objeto. En algunas computadoras, el hardware impone límites al alineamiento en la dirección de algunos objetos. Por ejemplo, un char puede ser almacenado en cualquier dirección, pero un **int** no. Generalmente, el objeto más restrictivo en la alineación es el double. Es por esta razón que *Alloc* fue declarada como un puntero a double.

En algunos compiladores, existe el tipo **void***, que se interpreta como un puntero al tipo más restrictivo para la alineación.

SetAllocHandler sólo se usa en MS-DOS.

Consultar

malloc(3) en el manual de Unix

BaseName

Separa el nombre de archivo de un path completo.

Sintaxis

```
char* BaseName (char* path);
```

Valor de Retorno

Devuelve un puntero a una cadena de caracteres, que es el nombre del archivo dentro del path que se dio como argumento.

Descripción

BaseName recibe un path completo (directorios + nombre de archivo) y devuelve un puntero al nombre del archivo.

No hace falta reservar memoria para el resultado, porque BaseName devuelve un puntero a una posición dentro de la cadena de caracteres pasada como argumento.

Ejemplo

```
#include <ideafix.h>
Program (example, %I% %G%)
{
char * c;
```

```
c = BaseName ( "/usr/aplicat/stock/sto01.fm");
printf ( "The result is % s\n" , c);
}
```

La variable c quedará apuntando al string "sto01.fm".

Check

Establece/Obtiene el dígito verificador.

Sintaxis

```
int CheckDigit (long n);
void SetCheckFactor (long n);
long TestCheckDigit (char* s);
```

Valor de Retorno

CheckDigit devuelve el dígito verificador.

TestCheckDigit devuelve el valor numérico de la cadena s si el dígito verificador es correcto; si no, devuelve ERROR.

Descripción

CheckDigit calcula el dígito verificador en módulo 10, para el número n, utilizando el factor previamente asignado con *SetCheckFactor*. El factor por defecto es 1397139713.

TestCheckDigit verifica la cadena s, que debe contener un número seguido de su dígito verificador.

SetCheckFactor asigna el factor verificador para CheckDigit y TestCheckDigit.

Ejemplo

```
#include < ideafix.h >
Program (example,% I%% G%)
{
SetCheckFactor (1313131313);
printf ( "%d \ n" , CheckDigit (12345));
printf ( "% ld% ld \ n" , TestCheckDigit ( "123457"),
TestCheckDigit ("123456 "));
}
```

Este código tendrá como salida:

7 12345 -1

Conv

Convierte datos de diversos tipos.

Sintaxis

```
long ItoL (short i);
double ItoF (short i);
double LtoF (long i);
short LtoI (long i);
short FtoI (double i);
long FtoL (double i);
```

Descripción

ItoL recibe un valor de tipo short como parámetro y lo devuelve convertido en un long.

ItoF recibe un valor de tipo short como parámetro y lo devuelve convertido en un double.

LtoF recibe un valor de tipo long como parámetro y lo devuelve convertido en un double.

LtoI recibe un valor de tipo long como parámetro y lo devuelve convertido en un short.

FtoI recibe un valor de tipo double como parámetro y lo devuelve convertido en un short.

FtoL recibe un valor de tipo double como parámetro y lo devuelve convertido en un long.

CUserId

Obtiene el nombre de login de un usuario.

Sintaxis

```
char* CUserId (char* s);
char* UserName (int uid);
char* FullUserName(int uid);
```

Valor de Retorno

Un puntero a una cadena de caracteres con el nombre de login del usuario.

Descripción

CUserId obtiene el nombre de login del usuario que arrancó el programa, y lo copia en s. Antes de llamar a esta función, se debe reservar suficiente memoria en s.

UserName devuelve un puntero a un registro interno de IDEAFIX, que contiene una cadena de caracteres con el nombre de login del usuario cuyo número es uid.

FullUserName retorna el nombre completo de un usuario, dado su número, buscándolo en el /etc/passwd. Esta función es complementaria de UserName.

Nota

Estas funciones sólo son operativas en sistemas multiusuarios.

Error

Maneja el despliegue de la información de errores.

Sintaxis

```
void Error (char* msg[, args ...]);
void Warning (char* msg[, args ...]);
IFPICCPCCP SetMessager (IFPICCPCCP f);
```

Descripción

Warning muestra un mensaje en la pantalla y *Error* muestra un mensaje y aborta el programa. Los argumentos de las dos funciones siguen la forma del printf del lenguaje C.

Además, se permite imprimir la descripción de una tecla del Window Manager. El formato es %tecla y puede estar modificado por los atributos estándares (longitud, alineación, etc.)

SetMessager indica qué función es llamada por Error y Warning para mostrar los mensajes. Si no se ha designado ninguna rutina para mostrar los mensajes, se llama a una rutina por defecto, cuya salida es estándar.

SetMessager devuelve un puntero al manejador de mensajes previo.

Consultar

Stop (GN)

FopenPath

Abre un archivo buscando según la variable de ambiente PATH.

Sintaxis

```
FILE* FopenPath (char* name, char* ext, char* mode);
```

Valor de Retorno

El valor devuelto es el descriptor del archivo abierto, o NULL si se produjo un error (no se encontró el archivo, no tenía permisos para acceder a él, etc.).

Descripción

FopenPath abre el archivo cuyo nombre se indica en la cadena de caracteres apuntada por nombre.

La extensión del archivo se puede incluir dentro de la cadena de caracteres apuntada por nombre, o se la puede indicar por separado en la cadena de caracteres apuntada por ext.

Si el nombre de archivo empieza con "/" (en DOS, "\"), IDEAFIX busca el archivo solamente en el directorio indicado. Si no, lo busca en todos los directorios señalados por la variable de ambiente PATH.

El parámetro **modo** se pasa como segundo parámetro a la función *fopen* de la biblioteca estándar del lenguaje C. Las funciones de la biblioteca de IDEAFIX que abren archivos (*OpenReport*, *OpenForm*, etc) utilizan *FopenPath*.

Si un programa termina con un mensaje de error que indica que un archivo no pudo ser abierto, es posible que la variable de ambiente PATH esté mal configurada.

Ejemplo

Suponiendo que la variable PATH está configurada de la siguiente manera en Unix:

```
PATH = :/usr/bin:/bin:/usr/sistemas
```

al hacer la llamada:

```
FILE *f;
f = FopenPath ("arch", ".usr", "r");
```

la función buscará en el siguiente orden:

```
./arch.usr /usr/bin/arch.usr
/bin/arch.usr
/usr/sistemas/arch.usr
```

FopenPath devolverá el descriptor de archivo para el primer archivo que pueda ser abierto. Si no puede abrir ninguno, devolverá NULL.

Consultar

fopen(3) en el manual de Unix

FreeMem

Obtiene la cantidad de memoria libre.

Sintaxis

```
long FreeMem (void);
void FreeMemWi (void);
```

Descripción

FreeMem devuelve la cantidad de memoria disponible en bytes.

FreeMemWi muestra la cantidad de memoria disponible dentro de una ventana, y espera hasta que se apriete una tecla.

Nota

Estas funciones sólo están disponibles en la versión para MS-DOS.

GetPWEntry

Busca un usuario en el archivo /etc/passwd.

Sintaxis

```
int GetPWEntry (int userid, char* s);
```

Valor de Retorno

GetPWEntry devuelve 0 si el usuario indicado existe, y -1 si el número de usuario no corresponde a ningún usuario del sistema.

Descripción

GetPWEntry toma el valor pasado en userid y lo busca en la tabla de usuarios el sistema. Si la búsqueda es exitosa, copia en s la línea del archivo /etc/passwd correspondiente a ese usuario.

Nota

Solo es operativa en sistemas multiusuarios.

GetUid

Obtiene el número de un usuario.

Sintaxis

```
int GetUid ();
int UserId (char* name);
```

Descripción

GetUid devuelve el número de usuario correspondiente al usuario que arrancó el programa.

UserId devuelve el número del usuario cuyo nombre de login se indica.

Nota

Estas funciones sólo son operativas en sistemas multiusuarios.

Locate

Busca en forma binaria en una tabla en memoria.

Sintaxis

```
int Locate (char* data, char* table, UShort ne1, UShort size,
FP rut);
```

Valor de Retorno

Si encuentra el elemento buscado, Locate devuelve la posición que ocupa en la tabla (las posiciones se numeran a partir de 0).

Si no lo encuentra, devuelve un número negativo n. El valor (-n-1) es la posición que debería ocupar el elemento dentro de la tabla.

Locate realiza una búsqueda binaria del elemento dato en la tabla indicada. Los argumentos que recibe son:

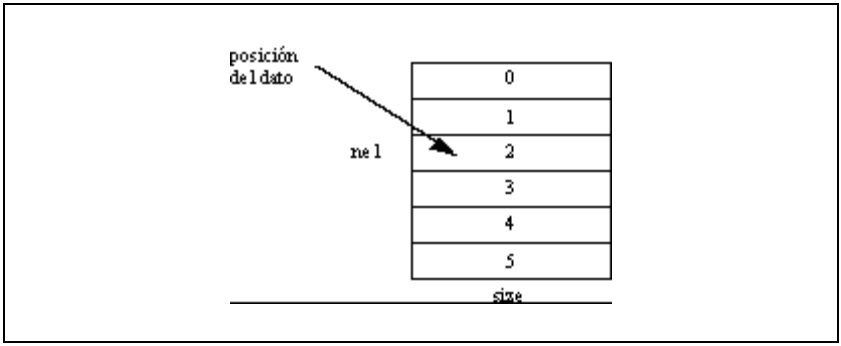
data es la cadena de caracteres que se está buscando.

table es la dirección de memoria donde comienza la ta bla.

nel es el número de elementos en la tabla. es la longitud en bytes de cada elemento.

rut es un puntero a la función que se usará para comparar los valores. Recibirá como primer argumento, el dato que se está buscando, y como segundo argumento, el elemento de la tabla con el que se quiere comparar. Debe devolver 0 si son iguales, un valor negativo si dato es menor y uno positivo si dato es mayor.

La estructura de la tabla es la siguiente:



En principio, la tabla debe estar ordenada en forma ascendente, pero si la función rut compara en forma inversa, habrá que ordenar la tabla en forma descendente.

Ejemplo

El siguiente programa crea una tabla con cuatro entradas, y usa la función Locate para buscar tres datos distintos. La función para comparar dos valores -tabcmp- toma en cuenta los tres primeros caracteres de cada cadena.

```
#include <ideafix.h>
char tab [] [10] = { "aaa", "bbbb", "dddddd", "eeee" };
int tabcmp();
Program(example, %I% %G%)
{
printf("For aaa, Locate returns %d\n", Loca te("aaa",tab[0],4,
10,tabcmp));
printf("For ddd, Locate returns %d\n",Locate("ddd", tab[0],4,
10,tabcmp));
printf("For ccc, Locate returns %d\n",Locate("ccc",
tab[0],4,10,tabcmp));
}
int tabcmp(char *s1,char *s2)
{
return strncmp(s1,s2,3);
}
```

Modules

Obtiene información de módulos instalados.

Sintaxis

```
int ModuleIsInstalled(int mo);
int ModuleExists(int mod, char *esquema);
```

Descripción

ModuleIsInstalled recive como parámetro el número del módulo y retorna ERROR si el módulo no existe; FALSE si el módulo existe y no está instalado, y TRUE si el módulo existe y está instalado.

ModuleExists recive como parámetros el número del módulo y un buffer donde estaría situado el esquema de ejemplo si el módulo existiera. La función puede retornar ERROR si el módulo no existiera (en este caso el contenido del string quedará indefinido), u OK si el módulo existiese, retornando en schema el producto del esquema ejmplo.

Consultar

Profile System, Installation Guide.

OpenPrinter

Abre una impresora.

Sintaxis

```
FILE* OpenPrinter (int ncopies, char* x);
void ClosePrinter (FILE* p);
```

Valor de Retorno

OpenPrinter devuelve un puntero al archivo de salida, que puede ser un archivo en el disco o una impresora.

Descripción

OpenPrinter abre el archivo de salida, teniendo en cuenta la cantidad de copias deseadas, y el sistema operativo en el que se está trabajando:

• en MS-DOS, abre el archivo temporario indicado en la variable de ambiente TEMPRINT.

• en UNIX, esta función se fija en el contenido de la variable de ambiente "printer", que habitualmente será un nombre de archivo, o el comando lp seguido de dos argumentos (ncopias y x).

ClosePrinter cierra un canal de salida abierto con *OpenPrinter*. El argumento que recibe es el puntero que se obtuvo al abrir el canal.

Proc

Atributos de proceso

Sintaxis

```
bool GetUpdatePerm();
bool GetAddPerm();
bool GetDelPerm();
bool ProcSaveMem();
UShort ProcUserId();
UShort ProcUserGid();
Int ProcPid();
const char *ProcTtyName();
const char *ProcTty();
Int ProcSig();
char **ProcArgs();
Int ProcNArgs();
TIME ProcInitTime();
DATE ProcInitDate();
```

Valor de Retorno

GetUpdatePerm, *GetAddPerm* y *GetDelPerm* retornan TRUE si el proceso tiene permiso de actualización, inserción y borrado, respectivamente.

ProcSaveMem retorna TRUE si el proceso esta en modo "save memory" (por ejemplo, tratando de usar la mínima cantidad de memoria posible, solo en Windows).

ProcUserId y *ProcUserGid* retornan el "user id" y el "group id" del proceso, respectivamente.

ProcPid retorna el "id" del proceso.

ProcTtyName retorna el nombre de la terminal donde está corriendo el proceso.

ProcTty retorna el nombre de la terminal donde el proceso realiza I/O con el **wm**.

ProcSig retorna el número de señales que hacen que el proceso aborte.

ProcArgs retorna argv.

ProcNArgs retorna argc.

ProcInitTime retorna la hora de comienzo del proceso.

ProcInitDate retorna la fecha de comienzo del proceso.

SetConvHandler

Establece el manejador de error de conversión de tipos.

Sintaxis

```
FPCP SetConvHandler (FPCP f);
```

Valor de Retorno

Un puntero al manejador previo.

Descripción

SetConvHandler establece el manejador invocado por las funciones que convierten los tipos de datos de los campos, cuando encuentran un error. Todas las funciones que obtienen y asignan valores a campos de bases de datos, formularios y reportes, invocan a este manejador cuando se intenta hacer una conversión ilegal.

El manejador por defecto muestra un mensaje de error en la pantalla, que incluye el nombre de la función, y aborta el programa.

f es un puntero al manejador, que recibirá dos cadenas de caracteres como argumentos: el nombre de la función que se estaba ejecutando y la causa del error.

SetReadEnvHandler

Establece el manejador de errores en la lectura de una variable de ambiente.

Sintaxis

```
FPCPCPI SetReadEnvHandler(FPCPCPI f);
```

Valor de Retorno

Un puntero al manejador previo.

Descripción

SetReadEnvHandler establece el manejador que es invocado por la función *ReadEnv* cuando no puede encontrar una variable de ambiente.

f es un puntero al manejador, que recibirá tres argumentos: una cadena de caracteres con el nombre de la variable de ambiente que no se encontró, otra cadena de caracteres con el nombre del programa, y un entero con el número de línea del programa que se estaba ejecutando.

Stop

Termina un proceso.

Sintaxis

```
void OnStop (FP fun);
void Stop (int state);
```

Descripción

Stop cierra todos los esquemas, formularios y reportes abiertos, llama a una serie de funciones definidas por el usuario y realiza otras tareas necesarias antes de dar por terminado un programa. Luego, sale con el código de error que indica estado.

Las funciones *Program* y *wcmd* (que reemplazan a la función main) hacen siempre un *Stop* antes de dar por terminada la ejecución del programa. Por eso, si se usa una de esas funciones, no es necesario incluir Stop.

OnStop agrega la función *f* a la lista de funciones definidas por el usuario que se deben ejecutar cuando IDEAFIX encuentre un Stop.

Capítulo 25

Manejo de Números

Estas funciones realizan operaciones sobre datos de tipo numérico. Las operaciones posibles incluyen suma, resta, multiplicación y división; poner y obtener el valor de un campo de tipo NUM, y realizar conversiones a otros tipos de datos.

Lista de Funciones

Esta tabla muestra las páginas documentadas en la sección dedicada a Manejo del tipo NUM.

Sección	Tareas Realizadas por la Sección de ls Función
Add	Suma un valor a una variable de tipo NUM.
Sub	Resta un valor a una variable de tipo NUM.
Mul	Multiplica una variable de tipo NUM por un valor.
Div	Divide una variable de tipo NUM por un valor.
Set	Establece el valor de un campo tipo NUM.
Get	Obtiene el valor de un campo tipo NUM.
Cmp	Compara una variable de tipo NUM con un valor.
Conv	Convierte valores al tipo NUM y viceversa

Referencia de Funciones

Add

Suma un valor a una variable de tipo NUM.

Sintaxis

```
NUM* NumAdd(NUM* a, NUM* b);
NUM* NumAddL(NUM* a, long b);
NUM* NumAddF(NUM* a, double b);
```

Valor de Retorno

Devuelven un puntero a la variable a, donde se almacena el resultado.

Descripción

NumAdd suma dos variables de tipo NUM, y deja el resultado en la primera.

NumAddL suma un valor de tipo long a una variable de tipo NUM, en la que deja el resultado.

NumAddF suma un valor de tipo double a una variable de tipo NUM, en la que deja el resultado.

Sub

Resta un valor a una variable de tipo NUM.

Sintaxis

```
NUM* NumSub(NUM* a, NUM* b);
NUM* NumSubL(NUM* a, long b);
NUM* NumSubF(NUM* a, double b);
```

Valor de retorno

Devuelven un puntero a la variable a, donde se almacena el resultado.

Descripción

NumSub resta el valor de una variable b de tipo NUM a una variable a, que también es de tipo NUM. El resultado queda en a.

NumSubL resta un valor de tipo long a una variable de tipo NUM, en la que deja el resultado.

NumSubF resta un valor de tipo double a una variable de tipo NUM, en la que deja el resultado.

Mul

Multiplica una variable de tipo NUM por un valor.

Sintaxis

```
NUM* NumMul(NUM* a, NUM* b);
NUM* NumMulL(NUM* a, long b);
NUM* NumMulF(NUM* a, double b);
```

Valor de Retorno

Devuelven un puntero a la variable a, donde se almacena el resultado.

Descripción

NumMul multiplica dos variables de tipo NUM, y deja el resultado en la primera.

NumMulL multiplica un valor de tipo long por una variable de tipo NUM, en la que deja el resultado.

NumMulF multiplica un valor de tipo double por una variable de tipo NUM, en la que deja el resultado.

Div

Divide una variable de tipo NUM por un valor.

Sintaxis

```
NUM* NumDiv(NUM* a, NUM* b);
NUM* NumDivL(NUM* a, long b);
NUM* NumDivF(NUM* a, double b);
```

Valor de Retorno

Devuelven un puntero a la variable a, donde se almacena el resultado.

Descripción

NumDiv divide una variable a de tipo NUM por una variable b, que también es de tipo NUM. El resultado queda en a.

NumDivL divide una variable de tipo NUM por un valor de tipo long, y deja el resultado en a.

NumDivF divide una variable de tipo NUM por un valor de tipo double, y deja el resultado en a.

Set

Establece el valor de un campo tipo NUM.

Sintaxis

```
void SetNFld(dbfield fn, NUM* val[, int row]);
void FmSetNFld(form fmd, fmfield fn, NUM* val[, int row]);
void RpSetNFld(report rpd, rpfield fn, NUM* val);
```

Descripción

SetNFld da el valor val (de tipo NUM) al campo fn de la base de datos. El parámetro fila se usa si el campo es vectorizado, para indicar qué elemento debe utilizarse.

FmSetNFld da el valor val (de tipo NUM) al campo fn del formulario fmd. El parámetro fila se usa para un campo multirrenglón, e indica qué fila se desea usar.

RpSetNFld da el valor val (de tipo NUM) al campo fn del reporte rpd. En todos los casos, el valor de tipo NUM se indica a través de un puntero.

Cmp

Compara una variable de tipo NUM con un valor.

Sintaxis

```
long NumCmp (const NUM* a, const NUM* b);
long NumCmpL (const NUM* a, const long b);
long NumCmpF (const NUM* a, const double b);
```

Valor de Retorno

Devuelven un número positivo si *a* es mayor que *b*, un número negativo si *a* es menor que *b*, y cero si son iguales.

Descripción

NumCmp compara dos variables de tipo NUM.

NumCmpL compara un valor de tipo NUM con otro de tipo long.

NumCmpF compara un valor de tipo NUM con otro de tipo double.

Conv

Convierte valores al tipo NUM y viceversa.

Sintaxis

```
void IToNum (short val, NUM* n)
void LToNum (long val, NUM* n)
void FToNum (double val, NUM* n)
void StrToNum (const char* s, NUM* n)
long NumToL (const NUM*n)
double NumToF (const NUM*n)
const char* NumToStr (const NUM* n, int tamaño, int prec)
```

Valor de Retorno

NumToL y NumToF retornan el valor convertido al tipo long y double, respectivamente.

NumToStr devuelve un puntero a una cadena de caracteres que contiene el valor convertido.

Descripción

IToNum, LToNum y FToNum convierte nel valor val (de tipo short , long y double respectivamente) a otro de tipo NUM.

StrToNum convierte una cadena de caracteres en un valor de tipo NUM.

NumToL y NumToF devuelven el valor de tipo NUM convertido al tipo long y double respectivamente.

NumToStr convierte un valor de tipo NUM en una cadena de caracteres. El parámetro *tamaño* indica el tamaño del buffer de conversión (si vale NTSDEF, se usará el buffer más chico posible) y *prec* es la cantidad de decimales que se desea obtener en el resultado.

El buffer de conversión usado por NumToStr es interno a IDEAFIX.

Capítulo 26

Interfaz con Reportes (RP)

Las funciones para reportes le dan al programador, un completo control de las operaciones realizadas con un reporte compilado mediante el utilitario RGEN. Los reportes son diseñados mediante la definición de los campos, y estableciendo la relación con la base de datos en caso de que ésta existiera.

El Archivo .*rph*

Este archivo puede ser generado con el utilitario RGEN (Ver Capítulo 4, *de este Manual*) o bien indicándolo en la zona de especificaciones generales del reporte (Ver *Capítulo 4 de este Manual*, Sección %report). Contiene constantes de tipos rpfield que son utilizados para referenciar un campo en particular con las funciones de reportes. Como un ejemplo, se muestra el archivo cabecera generado para el reporte emplo.rp, presentado en Capítulo 4, Listado Ejemplo :

```
/* Field names */
# define DNUM (rpfield) 1 /* Integer */
# define DDES (rpfield) 2 /* String */
# define PNUM (rpfield) 3 /* Long */
# define PNOMBRE (rpfield) 4 /* String */
# define PFINGR (rpfield) 5 /* Date */
/* Zone names */
# define REPTITULO (int) 0
# define DEP (int) 1
# define PERSON (int) 2
# define TOT (int) 3
```

Descriptores de Reportes

Para que sea posible efectuar operaciones con un reporte, se debe asociar mediante una de las funciones de IDEAFIX un descriptor de reporte, el cual es una variable de tipo report.

Estos descriptores de reportes se obtienen mediante la función *OpenReport* de IDEAFIX, y es el descriptor el que permite dar valores a un reporte determinado. La forma de realizar ésto es mediante funciones especiales para manejo de reportes. Existe otro tipo de descriptor en la familia de los reportes denominado Output. Este descriptor se utiliza para las salidas por grillas, tanto a terminal, impresora o archivo como a otro proceso. La forma de obtenerlo es mediante las funciones Open de manejo de grillas. Podemos resumir estos descriptores en la siguiente tabla:

Descriptor	Tipo de Variable
reporte	report
campo	rpfield
grilla	Output

Archivos de Inclusión

El archivo rp.h contiene constantes y definiciones de tipos, ambos necesarios para el manejo de reportes. El encabezamiento ideafix.h incluye automáticamente todos los archivos de cabecera necesarios de IDEAFIX.

Lista de Funciones

El primer cuadro muestra las páginas documentadas en la sección dedicada a Reportes y el segundo muestra todas las funciones incluidas y la página en la que se encuentran.

Sección	Tares Realizadas por la sección de la Función
BeginReport	Comienza un reporte.
Close	Cierra el reporte.
DbToRp	Copia los valores de la base de datos al reporte.
DoReport	Envía una zona a la salida.
EndReport	Termina un reporte.
OpenReport	Abre un reporte.
Output	Abre y cierra la salida de un reporte.
RpCheckSum	Calcula el "checksum" de la definición de un reporte.
RpClearZone	Blanquea o borra cero todos los campos de una zona.
RpDrawZ	Cancela las últimas líneas de la página.
RpEjectPage	Fuerza un salto de página.

RpSkipLines	Deja cierta cantidad de líneas en blanco.
RpSetOutput	Cambia la salida de un reporte asociado a una grilla.
Set	Asigna un valor a los campos del reporte.
Values	Obtiene valores de un reporte.

Referencia de Funciones

BeginReport

Comienza un reporte.

Sintaxis

```
int BeginReport(report rpd, int ncopies, char* str);
```

Valor de Retorno

Si todo funcionó bien, devuelve OK. Si se produjo un error, devuelve ERROR.

Descripción

BeginReport inicia el reporte, abriendo la salida del mismo (un archivo o la impresora, según el contenido de la variable de ambiente "printer") y poniendo en cero todos los valores del reporte.

También prepara para imprimir todas las zonas que tengan la condición before report (que se imprimirán la próxima vez que se imprima una zona).

Cuando se abre un reporte con *OpenReport* y no se indica el modo RP_NOBEGIN, IDEAFIX llama automáticamente a esta función.

Los parámetros **ncopias** y str serán pasados en el mismo orden a la función *OpenPrinter*.

Consultar

OpenPrinter(GN)

OpenReport(RP)

Closereport

Cierra el reporte.

Sintaxis

```
int CloseReport(report rpd);
void CloseAllReports();
```

Valor de Retorno

Si se produce un error, CloseReport devuelve ERROR; de lo contrario, devuelve OK.

Descripción

CloseReport cierra el reporte cuyo descriptor es rpd, liberando la zona de memoria que utilizaba.

CloseAllReports cierra todos los reportes abiertos.

Antes de cerrar los reportes, estas dos funciones imprimen todas las zonas que tengan la condición after report.

DbToRp

Copia los valores de la base de datos al reporte.

Sintaxis

```
void DbToRp(report rpd, rpfield from, rpfield to [,int row]);
```

Descripción

Esta función copia todos los campos entre desde y hasta del buffer de la base de datos al buffer del reporte cuyo descriptor es rpd. Si el campo que se está copiando es un campo vectorizado, hay que indicar el número de fila que se desea copiar.

DoReport

Envía una zona a la salida.

Sintaxis

```
int DoReport(report rpd, UShort zone);
```

Valor de Retorno

El valor devuelto por esta función tiene sentido únicamente si la salida del reporte es output to terminal, y es uno de los siguientes:

• ERROR, cuando el usuario aprieta la tecla FIN.

• OK, en cualquier otro caso.

Descripción

DoReport imprime la zona indicada del reporte, teniendo en cuenta las validaciones que se hayan indicado en la definición del mismo (before page, after field, etc.).

Ejemplo

```
#include <ideafix.h>
#include "library.rph"
wcmd (example, %I% G%)
{
report rp;
rp = OpenReport_("library", RP_EABORT);
/*..............* /
DoReport (rp, LINE);
}
```

Este programa imprimirá la zona LINEA, definida en el reporte biblio.

Endreport

Termina un reporte.

Sintaxis

```
int EndReport(report rpd);
```

Valor de Retorno

Esta función devuelve OK si todo funcionó correctamente, o ERROR en caso contrario.

Descripción

EndReport finaliza el reporte cuyo descriptor es rpd, imprimiendo todas las zonas que tengan la condición after report.

CloseReport llama a *EndReport* en forma automática.

Consultar

OpenPrinter(GN)

OpenReport(RP)

OpenReport

Abre un reporte.

Sintaxis

```
report OpenReport(char* name, int flags[, int ncopies, char*
str]);
```

Valor de retorno

Devuelve el descriptor del reporte que se abrió.

Descripción

OpenReport abre el reporte indicado en nombre, buscándolo en todos los directorios incluidos en la variable de ambiente PATH. También abre los esquemas que estén asociados al reporte.

Los flags indican la forma en que se quiere abrir el reporte, y pueden ser:

RP_NORMAL, abre el reporte en la forma normal.

RP_SYMBOLS, carga la tabla de símbolos de los esquemas asociados.

RP_COPIES, toma en cuenta los parámetros ncopias y str.

RP_EABORT, si se produce un error al abrir el reporte, aborta el programa.

RP_NOBEGIN, no imprime las zonas que tienen la condición before report. Este flag se debe indicar si se usan las funciones *RpSetOutput* y *SetRpOutput*.

Para comprender el uso de los parámetros ncopias y str, es necesario aclarar el uso de la variable de ambiente "printer". Esta variable indica cómo se debe enviar la salida a la impresora, y generalmente utiliza el comando lp de UNIX:

```
export printer='P:lp -n%d -o%s'
```

El comando **lp** admite diversas opciones, pero las que más se usan son "**-n**" (número de copias a imprimir) y "**-o**" (opciones de la impresora). Cada una de estas opciones lleva un número o una cadena de caracteres a continuación: esos datos son los que se suministran en los parámetros *ncopias* y *str*.

Por ejemplo, si la variable de ambiente "printer" está definida como se indicó más arriba y se llama a OpenReport así:

```
OpenReport( "repo1", RP_COPIES, 3, "CPI16");
```

la información se enviará a la impresora con el comando ‘lp -n3 -oCPI16", que imprimirá tres copias del reporte con letra comprimida.

Los parámetros de OpenReport a partir del tercero se pasan en el mismo orden al comando **lp** de UNIX.

Ejemplo

Suponiendo que la variable de ambiente "printer" está definida de la siguiente forma:

```
export printer='lp -s -n%d'
```

el siguiente programa abre el reporte biblio, indicando a la impresora que haga dos copias del mismo:

```
#include <ideafix.h>
wcmd (example, %I% %G%)
{
report rp;
rp = OpenReport ( "library", RP_EABORT|RP_COPIES, 2);
/* etc...* /
}
```

OpenReport devolverá un descriptor de reporte, que será asignado a la variable rp. Si no encuentra el reporte biblio en ninguno de los directorios indicados en la variable de ambiente PATH, *OpenReport* abortará el programa.

Consultar

BeginReport(RP)

RpSetOutput(RP)

Output

Maneja grillas.

Sintaxis

```
Output OpenTermOutput(int f_org, int c_org, int height,
int width, attr_type backgr, IFP getcmd);
Output OpenRpOutput(char* rpname, int ncopies, char* arg);
Output OpenDelimOutput(int dest, char* arg,
```

```
char* sep, int ncopies);
Output OpenFmtOutput(int dest, UChar* argv, UChar* head ing,
UChar* footing, int fwidth, int flen,
int topm, int botm, int leftm, int ncopies);
void CloseOutput(Output out);
int SetOutputColumn(Output out, char* title, int wide,
type t[, int long, int ndec]);
int ColumnWidth(Output out, int col);
int OutputColumn(Output out, char* bp);
UShort NColsOutput (Output out);
UShort CurrColOutput (Output out);
UShort PageWidthOutput (Output out);
```

Valor de Retorno

Las funciones *Open...* devuelven un descriptor de salida. Este descriptor se usa para identificar la salida en las funciones que envían datos.

SetOutputColumn retorna ERROR u OK, si aparece o no un error, respectivamente.

ColumnWidth retorna el tamaño del ancho de la columna de un reporte.

Output Column retorna ERROR si el usuario presiona la tecla FIN, de cualquier otra forma retorna OK.

NColsOutput retorna la cantidad de columnas del output «out«.

CurrColOutput retorna la columna corriente del mismo.

PageWidthOutput retorna el ancho de página del output «out«.

Descripción

OpenTermOutput obtiene un descriptor para enviar la salida del reporte a la terminal. Los parámetros son:

f_org fila origen de la grilla.

c_org columna origen de la grilla.

height altura de la grilla. Si es cero, ocupará el tamaño máximo que le permita la terminal en uso.

width ancho de la grilla. Si es cero, ocupará el ancho máximo que le permita la terminal en uso.

backgrgrid«s color del fondo de la grilla. Los colores posibles son:

A_NORMAL, el color por omisión.

A_REVERSE, fondo en video inverso.

A_RED_BG, rojo.

A_BLUE_BG, azul.

A_GREEN_BG, verde.

A_YELLOW_BG, amarillo.

A_CYAN_BG, cyan.

A_MAGENTA_BG, magenta.

A_WHITE_BG, blanco.

getcmd nombre de la función que se usará para leer el teclado. Si es NULL, se usa WiGetc.

OpenRpOutput obtiene un descriptor para enviar la salida a un reporte. Se debe indicar el nombre del archivo ".rpo", la cantidad de copias y los demás argumentos para la variable de ambiente "printer".

OpenDelimOutput obtiene un descriptor para enviar la salida a un archivo, o a un "pipe" a otro proceso, con los separadores indicados. Los parámetros son los siguientes:

dest una de las siguientes constantes:

RP_IO_FILE, envía la salida al archivo arg.

RP_IO_PIPE, envía la salida al proceso arg.

RP_IO_DEFAULT, envía la salida al lugar indicado en la variable de ambiente "printer".

arg nombre del archivo o proceso al que se desea enviar la salida.

Sep una cadena de dos caracteres, que son los separadores de campo y de registro. Si es NULL, se toma como separador de campo el tabulador y como separador de registro el carácter newline, es decir, equivale a escribir "\t\n".

ncopies cantidad de copias que se emitirán.

OpenFmtOutput obtiene un descriptor para enviar la salida a un archivo o a otro proceso, con separadores estándar ("\t" como separador de campo y "\n" como separador de registro), y con las demás opciones de formato indicadas. Los parámetros son:

dest, una de las siguientes constantes:

RP_IO_FILE, envía la salida al archivo arg.

RP_IO_PIPE, envía la salida al proceso arg.

RP_IO_DEFAULT, envía la salida al lugar indicado en la variable de ambiente "printer".

arg, nombre del archivo o proceso al que se desea enviar la salida.

heading, cadena de caracteres que se va imprimir en la parte superior de cada hoja.

footing, cadena de caracteres que se va imprimir en la parte inferior de cada hoja.

fwidth, ancho de cada renglón.

flen, cantidad de renglones por página.

topm, margen superior de la página (número de líneas que se dejan en blanco entre el heading y el texto del reporte).

botm, margen inferior de la página (número de líneas que se dejan en blanco entre el texto del reporte y el footing).

leftm, margen izquierdo de la página (número de caracteres que se dejan en blanco entre el borde de la hoja y el principio de cada renglón del reporte).

ncopies, cantidad de copias. Sólo se usa si se especificó el modo RP_IO_DEFAULT.

SetOutputColumn define el título o el encabezamiento y el ancho mínimo para las columnas de la grilla. Sus parámetros son los siguientes:

out, descriptor de salida obtenido con una de estas funciones: OpenTermOutput, OpenRpOutput, OpenDelimOutput y OpenFmtOutput.

title, título o encabezamiento de la columna. Si no se desea insertar ningún título, este parámetro debe ser NULL.

wide, ancho mínimo de las columnas. Si el título ocupara más espacio, se tomará el ancho del título en reemplazo de este valor.

t, tipo de dato de la columna:

TY_NUMERIC, numérico.

TY_FLOAT, numérico con punto flotante.

TY_STRING, alfanumérico.

TY_DATA, fecha.

TY_TIME, hora.

TY_BOOL, booleano (verdadero o falso).

long, ancho del dato (en carácteres).

ndec, número de dígitos decimales, si el campo es numérico.

ColumnWidth retorna el ancho de las columnas de la grilla. Los parámetros son:

out, descriptor de salida, obtenido con alguna de estas funciones: OpenTermOutput, OpenRpOutput, OpenDelimOutput, OpenFmtOutput.

col, número de la columna de la que se quiere saber su ancho. Las columnas se empiezan a numerar de cero.

OutputColumn almacena datos en la siguiente columna de la grilla. Si todas las columnas

fueron completadas, se mueve a la siguiente fila. Para completar una grilla, esta función debe ser llamada con todos los valores, tomados de izquierda a derecha, y desde arriba hacia abajo. Los parámetros son:

out, descriptor de salida obtenido con alguna de estas funciones: OpenTermOutput, OpenRpOutput, OpenDelimOutput, OpenFmtOutput.

bp, cadena de carácteres con los datos de la columna.

NColsOutput, CurrColOutput y PageWidthOutput tienen utilidad en el momento de crear listados dinámicos a través de Grillas.

Ejemplo

El siguiente programa envía la salida según el valor que recibe como argumento.

```
#include <ideafix.h>
#include "biblio.sch"
wcmd (example, %I% %G%)
{
Output out;
OpenSchema ( "biblio", IO_EABORT);
switch (StrToI (argv [1])) {
case 1:
WiMsg ( "Sending the output to the terminal... \n ");
out = OpenTermOutput (1, 1, 0, 0,
TO_NORMAL, NULL);
break;
case 2:
WiMsg ( "Sending the output to a fi le... \n ");
out = OpenDelimOutput
(RP_IO_FILE,"x.pr", NULL, 1);
break;
case 3:
WiMsg ( "Sending the output to a pro cess... \n ");
out = OpenDelimOutput (RP_IO_PIPE, "ipg", NULL, 1);
break;
case 4:
WiMsg ( "Sending the output to the printer... \n ");
out = OpenDelimOutput (RP_IO_DEFAULT, NULL, NULL, 1);
break;
case 5:
WiMsg ("Sending the output to a
report. .. \n");
out = OpenRpOutput ( "output", 1, NULL);
break;
default:
WiMsg("Sending the output to a file with format..\n");
out = OpenFmtOutputRP_IO_FILE,"x.pr", "List's Header",
"Message of list's end" , 120, 72, 2, 1, 5, 1);
break;
}
/* Create the grid */
SetOutputColumn (out,"Book" , 4, TY_NUMERIC);
SetOutputColumn(out, "Title", 30, TY_STRING);
/* Fill the grid */
while (GetRecord (BOOKS, NEXT_KEY, IO_DEFAULT)! = ERROR) {
```

```
charxaux [5];
GetFld (CODE BOOKS,xaux);
if (OutputColumn (out,xaux) == ERROR) break;
if (OutputColumn(out,SFld(TITLE_BOOKS)) ==ERROR)
break;
}
CloseOutput (out);
}
```

Consultar

WiGet(WI)

Output(RP)

RpCheckSum

Calcula el "checksum" de la definición de un reporte.

Sintaxis

```
long RpCheckSum(report rpd);
void RpVerifyCheckSum(report rpd, long chksum);
```

Descripción

Cuando se compila un reporte con **rgen -h** o se incluye la cláusula language en su definición, IDEAFIX genera un archivo con el mismo nombre del reporte y extensión ".rph". En este archivo, define una constante llamada RP_rep_CHKSUM (donde rep es el nombre del reporte) cuyo valor depende de los campos, tablas, etc. que lo componen. Si se modifica la definición del reporte, pero no se actualiza el archivo ".rph", se pueden producir errores en los programas.

Para detectar esta situación, se utilizan las funciones *RpCheckSum* y *RpVerifyCheckSum*.

RpCheckSum devuelve el valor que debería tener IO_rep_CHKSUM según la definición actual del reporte. Comparando la constante con el valor obtenido, se podrá saber si el archivo ".rph" está actualizado o no.

RpVerifyCheckSum realiza la comparación directamente, y si el archivo está desactualizado, emite un mensaje informando el error. El segundo parámetro debe ser la constante IO_rep_CHKSUM.

Ejemplo

```
#include <ideafix.h>
#include "biblio.rph"
```

```
wcmd (example,% I%% G%)
{
reportrp;
rp = OpenReport ( "biblio", RP EABORT | RP COPIES, 2);
RpVerifyCheckSum (rp, RP BIBLIO CHKSUM);
}
```

RpClearZone

Limpia o blanquea todos los campos de una zona.

Sintaxis

```
void RpClearZone(report rpd, int zone);
```

Descripción

RpClearZone pone en blanco todos los campos de una zona del reporte.

rpd es el descriptor del reporte y zona es el descriptor de la zona que se quiere blanquear.

RpDrawZ

Cancela las últimas líneas de la página.

Sintaxis

```
void RpDrawZ(report rpd, UChar n);
```

Descripción

RpDrawZ cancela las *n* últimas líneas del reporte rpd. Fuerza un salto de página.

RpEjectPage

Fuerza un salto de página.

Sintaxis

```
void RpEjectPage(report rpd);
```

Descripción

RpEjectPage produce un salto de página en el reporte cuyo descriptor es rpd. La próxima información que se envíe al reporte aparecerá en una página nueva.

RpSkipLines

Deja una cantidad de líneas en blanco

Sintaxis

```
void RpSkipLines(report rpd, int n);
```

Descripción

RpSkipLines deja n líneas en blanco en el reporte rpd.

RpSetOutput

Cambia la salida de un reporte asociado a una grilla.

Sintaxis

```
void RpSetOutput(report rpd, rp_output out_to, char* arg);
void SetRpOutput(Output out, tp_output out_to, char* arg);
```

Descripción

Estas funciones cambian la salida de un reporte. Recibe como parámetro el descriptor de un reporte; *SetRpOutput* recibe el descriptor de la salida de la grilla.

out_to es una de las siguientes constantes.

RP_IO_FILE, envía la salida al archivo arg.

RP_IO_PIPE, envía la salida al proceso arg.

RP_IO_TERM, envía la salida a la terminal. arg debe ser NULL.

RP_IO_REPORT, envía la salida a un reporte. arg debe ser NULL.

RP_IO_DEFAULT, envía la salida al lugar indicado en la variable de ambiente "printer". arg debe ser NULL.

Para usar estas funciones, el reporte debe ser abierto en modo RP_NOBEGIN. Esta función debe ser usada antes de *BeginReport*.

Set

Asigna un valor a los campos del reporte.

Sintaxis

```
void RpSetFld(report rpd, rpfield fn, char* val);
void RpSetIFld(report rpd, rpfield fn, int val);
void RpSetLFld(report rpd, rpfield fn, long val);
void RpSetFFld(report rpd, rpfield fn, double val);
void RpSetDFld(report rpd, rpfield fn, DATE val);
void RpSetTFld(report rpd, rpfield fn, TIME val);
```

Descripción

Este grupo de funciones sirve para darle el valor val al campo fn del reporte *rpd*.

Cada función se utiliza para un tipo de dato distinto.

Values

El valor pedido del reporte.

Sintaxis

```
int RpIFld(report rpd, rpfield fn);
int RpFldLen(report rp, rpfield fld);
```

Valor de Retorno

RpIFld retorna el valor requerido del reporte.

RpFldLen retorna un entero con la longitud del campo *fld* del reporte *rp*.

Descripción

RpIFld obtiene ciertos valores especiales de un reporte. Los parámetros son:

rpd, descriptor del reporte.

fn, valor que se quiere obtener:

PAGENO, número de página.

LINENO, número de línea dentro del reporte.

FLENGHT, largo de la página.

WIDTH, ancho de la página.

BOTMARG, margen inferior.

TOPMARG, margen superior.

LEFTMARG, margen izquierdo.

Capítulo 27

Cadena de Caracteres (ST)

Las funciones de manejo de cadena de caracteres realizan operaciones sobre "character strings". Tales operaciones pueden consistir en convertir números a letras, copiar zonas de memoria, ordenar alfabéticamente, etc. Dichas conversiones se realizan conforme al conjunto de caracteres de IDEAFIX; es decir, teniendo en cuenta aspectos modificatorios tales como acentos, diéresis, etc.

Lista de Funciones

El primer cuadro muestra las páginas documentadas en la sección dedicada a Manejo de Cadenas de Caracteres.

Sección	Tarea Realizada por la Sección de la Función
CompileMask	Compila una máscara de testeo de cadenas de caracteres.
Conv	Convierte números a cadenas ASCII y viceversa.
FmtNum	Formatea un número.
NumToTxt	Convierte un número en su descripción literal.
ReadEnv	Lee una variable de ambiente.
RexpMatch	Compara un string con una expresión regular limitada.
StrCase	Convierte cadenas de caracteres a mayúsculas o minúsculas.
StrCmp	Compara cadenas de caracteres.
StrDspLen	Obtiene la longitud de despliegue de un string.
StrTxt	Separa en partes una cadena de caracteres.

Referencia de Funciones

CompileMask

Compila una máscara de testeo de cadenas de caracteres.

Sintaxis

```
int CompileMask(char* mask, char* tstmask, char* omask);
```

Valor de Retorno

Si todo funcionó bien, devuelve OK. Si no, devuelve ERROR (por ejemplo, si la longitud de la máscara es mayor que 512).

Descripción

CompileMask compila una máscara de testeo de cadenas de caracteres, que luego será utilizada con *WiGetField*. Copia en tstmask la máscara de bits de chequeo y en omask la máscara de formateo de salida.

La longitud de tstmask y omask debe ser igual a la longitud real de la máscara más un espacio para el código de fin de la cadena (\0). Por ejemplo, si se tiene la máscara "9A", la longitud de tstmask y omask debe ser igual a 10. El espacio de memoria para tstmask y omask debe reservarse antes de llamar a CompileMask.

Si no se desea usar *tstmask* u *omask*, se puede escribir NULL en su lugar.

Ejemplo

```
#include <ideafix.h>
UChar buff[10];
char omask[10], tstmask[10];
wcmd(example, %I% %G%)
{
/* Create a work window */
WiCreate(SCREEN, 10, 10, 10, 30, STAND_BORDER, "Label",
A_NORMAL);
/* Generate the mask for data input */
CompileMask("aa.aa.aaa", tstmask, omask);
/* Test the mask in a field */
while(WiGetField(TY_STRING, buff, 0, 9, 9, 0, tst mask,omask)
!= K_END);
}
```

Consultar

WiCreate(WI)

WiGetField(WI)

Conv

Convierte números a cadenas ASCII y viceversa.

Sintaxis

```
void IToStr(int val, char* s);
void LToStr(long val, char* s);
void FToStr(double val, char* s);
short StrToI(char* s);
long StrToL(char* s);
double StrToF(char* s);
```

Descripción

IToStr, *LToStr* y *FToStr* convierten el número val a una cadena de caracteres, dejando el resultado en s.

Antes de llamar a esta función, se debe reservar suficiente memoria en s. *StrToI*, *StrToL* y *StrToF* convierten la cadena de caracteres s en un número, que es el valor retornado por cada función.

Si se intenta convertir un valor o una cadena de caracteres nula, estas funciones darán como resultado NULL.

FmtNum

Formatea un número.

Sintaxis

```
void FmtNum(char* output, char* input, int long, int ndec, int
sep);
```

Descripción

FmtNum toma una cadena de caracteres apuntada por input, que contiene un número, la formatea de acuerdo a los parámetros long, ndec y sep, y copia el resultado en output. Antes

de llamar a esta función, se debe reservar suficiente memoria en output.

Los parámetros que controlan el formato son:

long, longitud final de la cadena convertida. Si el resultado es más corto, se rellena con espacios a la izquierda.

ndec, cantidad de dígitos decimales.

Sep, si este parámetro tiene valor verdadero, se colocará el separador de miles cada tres dígitos enteros. Por ejemplo, en lugar de "1234567,2" aparecerá "1.234.567,2".

Ejemplo

```
#include <ideafix.h>
Program(example,% I%% G%)
{
charout [15];
FmtNum (out, "11273459", 15,2,1);
printf ( "The converted string is %s\n", out);
}
```

El resultado será:

```
La cadena convertida es 112.734,59
```

NumToTxt

Convierte un número en su descripción literal.

Sintaxis

```
void NumToTxt(double m, void* x, int f, int c, bool
final,char* front, char* below);
```

Descripción

NumToTxt convierte un número en punto flotante m en su descripción literal. Los dígitos decimales son ignorados.

El texto resultante se escribe en una matriz de caracteres de *f* filas por *c-1* columnas, cuya dirección inicial es x. Antes de llamar a la función, se debe reservar suficiente memoria en x.

El argumento final se usa en idiomas como el español: si tiene un valor verdadero, las unidades terminarán con la palabra "uno", y si tiene un valor falso, terminarán con la palabra "un".

Las cadenas de caracteres front y below se copiarán antes y después de la descripción literal, respectivamente.

El tamaño máximo del resultado es de 400 caracteres.

Nota

En cada fila, se debe reservar una posición más para el código "\0" que marca el fin de la cadena.

Ejemplo

```
#include <ideafix.h>
#define STRINGS 5
#define COLS 20+1
Program (example, %I% G%)
{
char res[STRINGS][COLS];
int i;
NumToTxt (1234.56, res, STRINGS, COLS, TRUE, "It's",
"dollars.");
for (i=0; i < STRINGS; i++)
printf ( "%s\n" , res[i]);
}
```

La salida será:

```
Son mil doscientos treinta y cuatro pesos.
```

ReadEnv

Lee una variable de ambiente.

Sintaxis

```
char* ReadEnv(char* name);
```

Valor de Retorno

Un puntero a una cadena de caracteres con el valor de la variable. Si no la puede encontrar, devuelve NULL.

Descripción

ReadEnv busca la variable nombre en el ambiente del sistema operativo. Si la encuentra, devuelve un puntero a un registro interno de IDEAFIX, que contiene el valor de la variable como una cadena de caracteres.

Si no encuentra la variable de ambiente indicada, *ReadEnv* llama al manejador definido con *SetReadEnvHandler*. El manejador por defecto aborta la ejecución del programa con un

mensaje de error.

Consultar

SetReadEnvHandler(GN)

RexpMatch

Compara una cadena de caracteres con una expresión regular limitada.

Sintaxis

```
bool RexpMatch(UChar* rexp, UChar* s, int flag);
```

Descripción

RexpMatch devuelve TRUE si la cadena s se corresponde con la expresión regular rexp. En caso contrario, devuelve FALSE.

Descripción

RexpMatch compara la cadena de caracteres s con la expresión regular rexp. Esta expresión regular puede contener los siguientes caracteres especiales:

"*" cualquier conjunto de caracteres.

"?"un carácter cualquiera.

"." es equivalente a "?".

"^"si está al principio de la expresión regular, indica que la misma debe coincidir con el segmento inicial de la cadena.

"$" si está al final de la expresión regular, indica que la misma debe coincidir con el segmento final de la cadena.

Para que *RexpMatch* considere los símbolos "^" y "$", el parámetro flag debe ser la constante REXP_EMBEDDED.

StrCase

Convierte una cadena de caracteres a mayúsculas o minúsculas.

Sintaxis

```
void StrToUpper(char* s);
void StrToLower(char* s);
```

Descripción

Estas funciones toman la cadena de caracteres indicada en s y convierten todas sus letras a mayúsculas (*StrToUpper*) o a minúsculas (*StrToLower*).

Cualquier carácter de la cadena que no sea una letra permanece inalterado.

StrCmp

Compara cadenas de caracteres.

Sintaxis

```
int StrCmp(char* s1, char* s2);
int StrNCmp(char* s1, char* s2, int n);
void SetStrCmp(IFPCPCP funct);
void SetStrNCmp(IFPCPCPI funct);
```

Descripción

StrCmp compara las cadenas de caracteres s1 y s2, y devuelve un entero menor, igual o mayor que 0, según sea s1 "alfabéticamente" menor, igual o mayor que s2, respectivamente.

"Alfabéticamente" significa que no se considera el código ASCII de cada letra, sino su posición dentro del alfabeto: "ll" está después de "lz", "ch" está después de "cz", etc.

StrNCmp compara los n primeros caracteres de las cadenas s1 y s2, produciendo un resultado similar al de StrCmp. SetStrCmp y SetStrNCmp definen las funciones utilizadas para realizar las comparaciones.

Ejemplo

```
#include <ideafix.h>
Program (example,% I%% G%)
{
char* s1 = "Rain" ;
char* s2 = "Rest" ;
int n;
n = StrCmp (s1, s2);
printf ( "n is worth %d\n", n);
}
```

El resultado será un valor mayor que 0. Nótese que la comparación es alfabética y no por orden ASCII: si así fuera, el resultado sería el opuesto.

StrDspLen

Obtiene la longitud de despliegue de una cadena de caracteres.

Sintaxis

```
int StrDspLen(char* s);
```

Descripción

StrDspLen devuelve la longitud de una cadena de caracteres, sin contar los códigos de control que pueda contener.

Ejemplo

```
#include <ideafix.h>
Program (example,% I%% G%)
{
char * s = "\033[1mBold\033[0m";
int n;
n = StrDspLen (s);
printf ( "n is worth %d\n" , n);
}
```

El valor de la variable n será 9. Si se hubiera usado la función estándar de "C" strlen, el resultado habría sido 17.

StrTxt

Separa en partes una cadena de caracteres.

Sintaxis

```
char* StrTxt(char* s, int n);
```

Valor de Retorno

Un puntero a una cadena de caracteres que no tiene más de n caracteres, o el valor NULL si la cadena ya no se puede dividir más.

Descripción

StrTxt divide la cadena apuntada por s en partes de n caracteres como máximo. Para dividir la

cadena, tiene en cuenta los espacios en blanco, de forma tal que una palabra no puede quedar separada en dos partes.

Si se indica una cadena de caracteres en s, *StrTxt* devuelve la primera parte de esa cadena. Si s vale NULL, *StrTxt* devuelve la siguiente parte de la cadena que se indicó la última vez.

Cuando ya no quedan más partes, *StrTxt* devuelve NULL.

No es necesario reservar memoria para ningún dato, porque todas las operaciones se hacen en un registro interno de IDEAFIX.

Ejemplo

```
#include <ideafix.h>
Program (example,% I%% G%)
{
chartext [40] =This is a test string for StrTxt ";
char * s;
s = StrTxt (text, 18);
do {
puts (s);
} while (*(s=StrTxt (NULL, 18)));
}
```

Capítulo 28

Fecha y Hora (TM)

Las funciones que manejan fechas permiten a los programas de aplicación realizar operaciones tales como obtener la fecha del día, calcular el número de quincena, etc.

Para realizar estas operaciones de una manera uniforme, se define el tipo de dato "date" (en tm.h). Toda fecha es convertida al formato "date" (llamado formato interno), utilizando la función *StrToD*. El formato interno es el número de días entre 01/01/84 y la fecha a convertir, por lo tanto "date" representa un número entero. Este número posee signo a fin de poder manejar fechas anteriores al 01/01/84. Cabe recalcar que la diferencia en días entre dos fechas, se obtiene restando ambas en formato "date". Y si se desea obtener, por ejemplo, la fecha de diez días posteriores a una fecha dada, se debe sumar el entero 10 a la fecha en formato "date". Ejemplos:

```
date f1,f2; int dif;
...... dif= f2-f1;
/* dif es la diferencia en días entre f1 y f2 */
f1= f2+10;
/* f1 es la fecha 10 días después de f2 */
```

Las relaciones de orden son también válidas, por ejemplo comparaciones como:

```
f1 > f2 f1 >= f2 f1 == f2
```

retornan el resultado correcto.

Existe un valor especial para variables de tipo "date", llamado NULL_DATE que significa un valor indeterminado para fechas. Este valor se obtiene usando la función *StrToD* para convertir una cadena vacía al formato interno. La relación inversa también es válida: si se convierte NULL_DATE a una cadena con la función *DToStr*, se obtiene un NULL_STRING. Los valores de una variable "date" permiten, mediante la operación "módulo 7", obtener el día de la semana correspondiente a la fecha, si el día es domingo retorna cero (0).

También se ha definido el tipo de dato "time". Se convierte la hora a dicho formato (llamado formato interno), utilizando la función StrToT. NULL_TIME es un valor especial para variables de este tipo, para el cual valen similares consideraciones que para el caso anterior,

incluyendo la reciprocidad de conversión entre una cadena vacía y el valor NULL_TIME.

Lista de Funciones

Siguen en el cuadro las páginas documentadas referentes al Manejo de Fechas y Horas.

Sección	Tarea Realizada por la Sección de la Función
AddMonth	Suma una cantidad de meses a una fecha.
DayName	Obtiene el día de la semana de una fecha.
DMYToD	Convierte día, mes y año a formato interno, y viceversa.
FirstMonthDay	Obtiene el primer día o el último del mes.
GetDateFmt	Obtiene o establece el formato de la fecha.
HalfMonth84	Obtiene el número de quincena.
Hour	Obtiene la hora actual.
IsHoliday	Comprueba si una fecha es feriado.
MonthDiff	Obtiene la diferencia entre los meses de dos fechas.
MonthName	Obtiene el nombre de un mes.
Numbers	Obtiene el número de día, semana, mes o año de una fecha.
StrToD	Convierte una cadena de caracteres al formato interno de fecha.
StrToT	Convierte una cadena de caracteres al formato interno de hora.
Today	Obtiene la fecha corriente.

Referencia de Funciones

AddMonth

Suma un número de meses a una fecha.

Sintaxis

```
DATE AddMonth(DATE d, int n);
```

Descripción

AddMonth suma n meses a la fecha d, y devuelve la fecha correspondiente al primer día del mes del resultado. La variable e contendrá la fecha 01/10/89.

Ejemplo

```
#include <ideafix.h>
Program (example, %I% G%)
{
DATE d, e;
/* put the date on the first variable */
d = StrToD ( "05/12/94");
/* add five months */
e = AddMonth (d, 5);
}
```

La variable e contendrá la fecha 01/10/94.

DayName

Obtiene el día de la semana de una fecha.

Sintaxis

```
char* DayName(DATE fec);
```

Descripción

DayName devuelve un puntero a un registro interno de IDEAFIX, que contiene una cadena de caracteres con el nombre del día de la semana para la fecha fec (que debe estar en formato interno).

Ejemplo

```
#include <ideafix.h>
#include "library.sch"
wcmd (example, %I% %G%)
{
DATES date;
/* open the schema and read the first record */
OpenSchema( "librar", IO_EABORT);
GetRecord(BOOKSbyCODE, FIRST_KEY, IO_NOT_LOCK);
/* show day of the week of the edition date*/
date = DFld(DATE_BOOKS);
WiMsg ( "Day: %s",DayName (date));
}
```

DMYToD

Convierte un día, mes y año al formato interno de fecha, y viceversa.

Sintaxis

```
DATE DMYToD(short day, short month, short year);
void DToDMY(DATE d, short* day, short* month, short* year);
```

Descripción

DMYToD toma la fecha indicada con día, mes y año, la convierte al formato interno de IDEAFIX y devuelve ese número. El formato interno de una fecha es la cantidad de días que han transcurrido desde el 1 de enero de 1984.

DToDMY realiza la operación inversa: toma la fecha en formato interno indicada en d y separa sus componentes, dejando los resultados en día, mes, y año.

Ejemplo

```
#include <ideafix.h>
wcmd (example,% I%% G%)
{
DATES d;
short day, month, year;
/* create the first date */
day = 12; month = 8; year = 1990;
d = DMYTOD (day, month, year);
/* add 180 */
d = d + 180;
/* return the separate the components of the date */
DToDMY (d, &day, &month, &year);
/* show the result */
WiMsg ("%d/%d/%d\n", month, day, year);
}
```

Este programa toma la fecha 12/08/90, la convierte al formato interno, le suma 180 días y la vuelve a separar en día, mes y año. El resultado será la fecha 08/02/91.

FirstMonthDay

Obtiene el primero o el último de los días del mes.

Sintaxis

```
DATE FirstMonthDay(DATE d);
DATE LastMonthDay(DATE d);
```

Valor de Retorno

Estas dos funciones devuelven una fecha en formato interno.

Descripción

FirstMonthDay devuelve la fecha correspondiente al primer día del mes de la fecha d, que está en formato interno.

LastMonthDay devuelve la fecha correspondiente al último día del mes de la fecha d, que también está en formato interno.

GetDateFmt

Obtiene o establece el formato de la fecha.

Sintaxis

```
int GetDateFmt();
void SetDateFmt(int n);
```

Valor de Retorno

GetDateFmt devuelve un entero, que indica el formato de fecha en uso.

Descripción

GetDateFmt obtiene el formato de fecha que se está utilizando, mientras que *SetDateFmt* establece un nuevo formato de fecha. Los posibles formatos (el parámetro **n** de *SetDateFmt* y el valor devuelto por *GetDateFmt*) son:

• DFMT_INTERNAT, formato internacional (dd/mm/aa)

• DFMT_AMERICAN, formato americano (mm/dd/aa)

Estas constantes simbólicas están definidas en el archivo "tm.h".

HalfMonth84

Obtiene el número de quincena.

Sintaxis

```
int HalfMonth84(DATE d);
```

Valor de Retorno

Un entero que indica el número de quincenas desde el 01/01/84.

Descripción

HalfMonth84 calcula el número de quincenas transcurridas desde el 1 de enero de 1984 hasta la fecha d indicada como parámetro (en formato interno).

HalfMonth84 es útil para calcular pagos.

Ejemplo

```
#include <ideafix.h>
wcmd(example, %I% %G%)
{
DATE d;
int q1, q2;
/* create date */
d = StrToD("2/12/90");
/* Obtain the number of fortnight since 1984 */
q1 = HalfMonth84(d);
/* Obtain the number of fortnights in that year */
q2 = HalfMonth84(d) % 24;
WiMsg("%d\n%d\n", q1, q2);
}
```

Hour

Obtiene la hora actual.

Sintaxis

```
TIME Hour();
```

Descripción

Hour obtiene la hora actual del sistema operativo y la devuelve en el formato interno de IDEAFIX.

IsHoliday

Comprueba si una fecha es feriado.

Sintaxis

```
char* IsHoliday(DATE d);
```

Descripción

IsHoliday busca la fecha d en el archivo de feriados de IDEAFIX. Si la encuentra, devuelve un puntero a un registro interno de IDEAFIX, que contiene una cadena de caracteres con la descripción del feriado. Si no, devuelve NULL.

El archivo de feriados tiene el nombre "holliday.dat" y está ubicado en los siguientes directorios, según el idioma utilizado:

```
$IDEAFIX/data/english
```

o,

```
$IDEAFIX/data/spanish
```

El archivo puede ser editado para agregar, modificar o suprimir feriados. Cada renglón debe tener la fecha (en el formato dd/mm), una coma o espacio y la descripción del feriado.

La fecha "P" apunta al día del Domingo de Resurrección, que se calcula automáticamente para cada año. A partir de ahí, se pueden definir otras fechas sumándole o restándole una cantidad de días (por ejemplo, "P-2" es el Viernes Santo).

Ejemplo

```
#include <ideafix.h>
Program (example,% I%% G%)
{
DATES d;
d = StrToD ( "25/12/89");
printf ( "%s\n" , IsHoliday (d));
}
```

En versiones anteriores de IDEAFIX, IsHoliday se llamaba IsHolliday. Esta última función está aún definida por razones de compatibilidad.

MonthDiff

Obtiene la diferencia entre los meses de dos fechas.

Sintaxis

```
long MonthDiff(DATE a, DATE b);
```

Descripción

MonthDiff calcula la diferencia entre los meses de la fecha b y la fecha a (ambas en formato interno). El resultado será positivo si la fecha a es posterior a la fecha b.

Ejemplo

```
#include <ideafix.h>
#include "biblio.sch"
wcmd (example,% I%% G%)
{
DATES date;
/* open the schema and read the first record * /
OpenSchema( "biblio", IO_EABORT);
GetRecord(LIBROSbyCODIGO, FIRST_KEY, IO_NOT_LOCK);
/* show the name of the month of date of edition */
date = DFld(DATE_BOOKS);
WiMsg ("Month: %s ",MonthName(Month(date)));
}
```

El resultado de este ejemplo será 12.

MonthName

Obtiene el nombre de un mes.

Sintaxis

```
char* MonthName(int n);
```

Descripción

MonthName devuelve un puntero a un registro interno de IDEAFIX, que contiene una cadena de caracteres con el nombre del n-ésimo mes del año. Si n es menor que 1 o mayor que 12, MonthName devuelve NULL.

Ejemplo

```
#include <ideafix.h>
#include "biblio.sch"
wcmd (example,% I%% G%)
{
DATES date;
/* open the schema and read the first record * /
OpenSchema( "biblio", IO_EABORT);
GetRecord(LIBROSbyCODIGO, FIRST_KEY, IO_NOT_LOCK);
/* show the name of the month of date of edition */
date = DFld(DATE_BOOKS);
WiMsg ("Month: %s ",MonthName(Month(date)));
}
```

Numbers

Obtiene el número de día, semana, mes o año de una fecha.

Sintaxis

```
int Day(DATE fec);
int Month(DATE fec);
int Year(DATE fec);
long Week(DATE fec);
```

Valor de Retorno

Todas estas funciones devuelven un valor entero, que indica el número de día, semana, mes o año de una fecha dada.

Descripción

Day devuelve el número de día dentro del mes.

Month devuelve el número de mes.

Year devuelve el año de la fecha indicada.

Week devuelve un número que representa el número de semana y el año. Se calcula con siguiente fórmula:

```
valor devuelto = semana * 10000 + año
```

Si la fecha es anterior al primer lunes del año, Week devuelve la semana 53 del año anterior. En todos los casos, fec debe estar en formato interno.

Ejemplo

```
#include <ideafix.h>
#include "biblio.sch"
Program (example, %I% %G%)
{
DATES date;
/* To open the plan and to read the first record */
OpenSchema( "biblio", IO_EABORT);
GetRecord(LIBROSbyCODIGO, FIRST_KEY, IO_NOT_LOCK);
/* To separate the components of the date */
date = DFld(DATE_BOOKS);
printf( "Day: %d\n ", Day(date));
printf( "Month: %d\n ", month(date));
printf( "Year: %d\n ", Year(date));
/* obtain the number of week within year */
printf ( "Number of week: %ld\n",Week (date));
```

```
}
```

StrToD

Convierte una cadena de caracteres al formato interno de fecha, y viceversa.

Sintaxis

```
DATE StrToD(char* s);
DATE StrNXToD(char* s);
void DToStr(DATE d, char* dest, int fmt);
```

Valor de Retorno

StrToD y StrNXToD devuelven la fecha en formato interno. Si la fecha es inválida, devuelven NULL_DATE.

Descripción

Estas funciones convierten fechas utilizando el formato indicado con *SetDateFmt*.

StrToD convierte la fecha contenida en la cadena de caracteres apuntada por s a su valor en formato interno.

s puede estar en formato "dd/mm/aa" o "dd/mm/aaaa". El separador también puede ser nulo, o un carácter no numérico.

Si se estableció el formato americano con *SetDateFmt*, se debe indicar primero el mes y después el día.

Si s es la cadena de caracteres vacía, *StrToD* devuelve NULL_DATE.

Si s es la cadena de caracteres TODAY_STR, *StrToD* devuelve la fecha actual, en formato interno.

StrNXToD funciona igual que StrToD, con una única diferencia: si s es la cadena de caracteres TODAY_STR, *StrNXToD* devuelve la constante TODAY_DATE.

DToStr toma la fecha en formato interno d y la convierte en una cadena de caracteres, que deja en la dirección apuntada por *dest*. Antes de llamar a esta función, se debe reservar suficiente memoria en *dest*.

El parámetro *fmt* indica qué aspecto debe tener el resultado, y es una combinación de las siguientes constantes:

DFMT_SEPAR, usa "/" como separador del día, mes y año.

DFMT_YEAR4, expresa el año con cuatro dígitos.

DFMT_STANDART, es equivalente a DFMT_SEPAR.

DFMT_LDAYNAME, agrega el nombre completo del día.

DFMT_LMONTHNAME, agrega el nombre completo del mes.

DFMT_DAYNAME, agrega una abreviatura del nombre del día: ÔÔLUN", "MAR", "MIE", etc. a

DFMT_MONTHNAME, Agrega una abreviatura del nombre del mes: "ENE", "FEB", "MAR", etc.

d también puede ser una de las siguientes constantes:

NULL_DATE, DToStr devuelve la cadena de caracteres vacía.

TODAY_DATE, DToStr convierte la fecha actual del sistema.

Ejemplo

StrToD se usa de la siguiente forma:

```
#include <ideafix.h>
Program (example,% I%% G%)
{
DATES f1;
f1 = StrToD ( "09/07/86");
/* etc.. .* /
}
```

DToStr se usa así:

```
#include <ideafix.h>
wcmd (example, %I% G%)
{
char date [30];
DToStr (TODAY_DATES, date, DFMT_SEPAR);
WiMsg ( "Date of the day: %s", date);
DToStr (TODAY_DATE, date,
DFMT_LDAYNAME|DFMT_LMONTHNAME|DFMT_YEAR4);
WiMsg ( "Date of the day: %s", date);
}
```

El resultado de este programa será semejante al siguiente:

```
Fecha del día: 27/03/90
Digite cualquier tecla para continuar.
Fecha del día: MARTES 27 de MARZO de 1990
Digite cualquier tecla para continuar.
```

Consultar

SetDateFmt(TM)

StrToT

Convierte una cadena de caracteres al formato interno de hora, y viceversa.

Sintaxis

```
TIME StrToT(char* s);
TIME StrNXToT(char* s);
void TToStr(TIME t, char* dest, int fmt);
```

Valor de Retorno

StrToT y *StrNXToT* devuelven la hora en formato interno. Si la hora es inválida, devuelven NULL_TIME.

Descripción

StrToT convierte la hora contenida en la cadena de caracteres apuntada por s a su valor en formato interno.

s puede estar en formato "hh:mm" o "hh:mm:ss". El separador también puede ser nulo, o un carácter no numérico. También se puede indicar "AM" o "PM". Si s es la cadena de caracteres vacía, *StrToT* devuelve NULL_TIME. En cambio, si s es la cadena de caracteres HOUR_STR, devuelve la hora actual, en formato interno.

StrNXToT funciona igual que *StrToT*, excepto por una única diferencia: si s es la cadena de caracteres HOUR_STR, entonces *StrNXToT* devuelve la constante HOUR_TIME.

TToStr toma la hora en formato interno t y la convierte en una cadena de caracteres, que deja en la dirección apuntada por dest. Antes de llamar a esta función, se debe reservar suficiente memoria en dest.

El parámetro *fmt* indica qué aspecto debe tener el resultado, y es una combinación de las siguientes constantes:

TFMT_SEPAR, usa ":" como separador de horas, minutos y segundos.

TFMT_SECONDS, incluye los segundos en el resultado.

TFMT_HS12, indica la hora de 1 a 12 y agrega "AM" o "PM".

t también puede ser la constante HOUR_TIME, en cuyo caso *TToStr* convierte la hora actual del sistema.

StrToTS

Convierte una cadena de caracteres en el formato TIMESTAMP y viceversa.

Sintaxis

```
void TSToStr (TIMESTAMP ts, char* s);
TIMESTAMP StrToTS (char* s);
```

Descripción

TSToStr pasa una vaiable de tipo TIMESTAMP a su representación en string. El formato correspondiente es: "ddmmyyyy-hhmmss-nsec-uid", donde "ddmmyyyy" es el día, mes y año; "hhmmss" son hora, minutos y segundos, "nsec" es un campo resevado para una futura implementación, y "uid" corresponde al valor del UserId del usuario.

StrToTS realiza al trabajo contrario, es decir, toma un string y lo devuelve en una variable de tipo TIMESTAMP.

Today

Obtiene la fecha corriente.

Sintaxis

```
DATE Today();
```

Descripción

Today obtiene la fecha actual del sistema operativo y la devuelve en formato interno.

Capítulo 29

Interfaz con Ventanas (WI)

Esta sección contiene la documentación de las funciones para el manejo de ventanas. Los programas que utilicen estas funciones deberán ser corridos estando el Window Manager activado, de lo contrario un mensaje de error aparecerá en pantalla. Como es lógico, esto vale también para los editores de IDEAFIX, ya se trate del ie o del dali, ya que utilizan el wm para su propio manejo interno de pantallas. En este caso, el mensaje es explícito: "El wm no está corriendo".

Operación

Las funciones del Window Manager proveen las interfaces necesarias para las Entradas/Salidas por terminal. Existen rutinas para leer caracteres, leer y desplegar cadenas de caracteres, fijar atributos de pantalla, etc. Algunas de ellas se asemejan a las rutinas del lenguaje "C" estándar (por ejemplo WiPrintf, WiGetc, etc.)

Conceptos Básicos

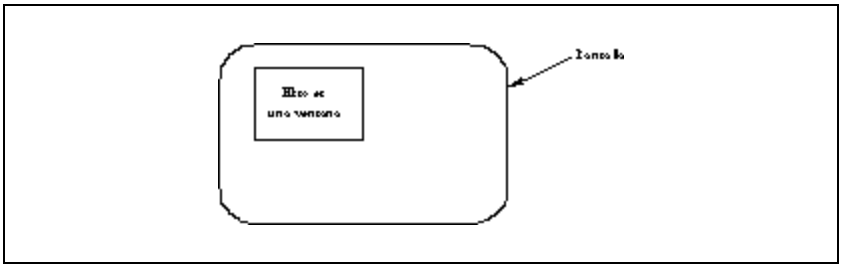
Definiremos a continuación algunos de los conceptos utilizados en las funciones de procesamiento de pantalla:

• La pantalla física es la pantalla completa que el usuario ve en su terminal.

• Una ventana es una pantalla lógica que puede ser del mismo tamaño o menor que la pantalla física. Tiene las mismas propiedades que la pantalla física: puede blanquearse total o parcialmente, scrolear, etc.

• Una propiedad importante es que pueden superponerse, sin que ello implique pérdida de la información; al cambiar de una a otra (lo que se denomina swap) reaparece la información propia de cada una dentro del área común.

Normalmente una ventana es más pequeña que la pantalla física como se muestra en la siguiente figura:



Características

Las ventanas pueden aparecer y desaparecer dinámicamente, por ejemplo pueden ser creadas y borradas en cualquier momento. Las ventanas están organizadas jerárquicamente en forma de árbol y poseen las siguientes características:

Existe desde el comienzo, la ventana SCREEN (Pantalla), que es la raíz de la estructura. Esta ventana posee la misma dimensión que la pantalla. El programador no accede a la ventana SCREEN directamente.

Existe también una ventana sobre la cual se está trabajando denominada "ventana corriente" y será referenciada dentro de la estructura como WICURRENT. Cada ventana creada será hija de alguna ventana existente, es posible utilizar SCREEN o WICURRENT como padre de nuevas ventanas.

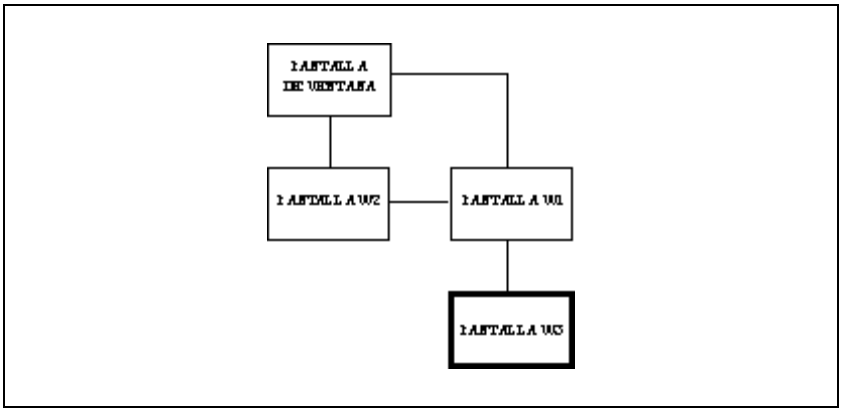
Una ventana hija no podrá superar nunca en dimensión a su padre, si así fuera la rutina de creación de ventanas hará los ajustes necesarios, para respetar esta condición.

Se puede colocar como activa cualquier ventana que no posea ventanas hijas.

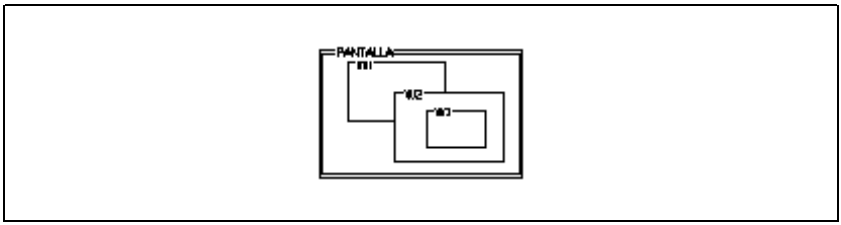
Hay operaciones que se llevan a cabo sobre la ventana corriente, mientras que hay otras que necesitan que se les pase como parámetro la ventana sobre cual se desea realizar la operación.

Para comprender estos conceptos, veamos un ejemplo donde suponemos haber creado tres ventanas, de las cuales la primera y segunda fueran hijas de la ventana SCREEN, y la tercera hija de la primera. Si fueron creadas en este orden (1,2,3) la ventana número tres estaría activa.

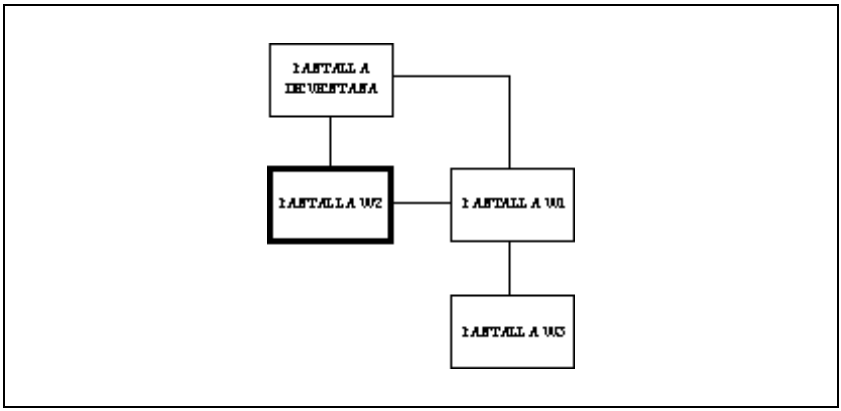
Esquemáticamente esto se vería de la siguiente forma:



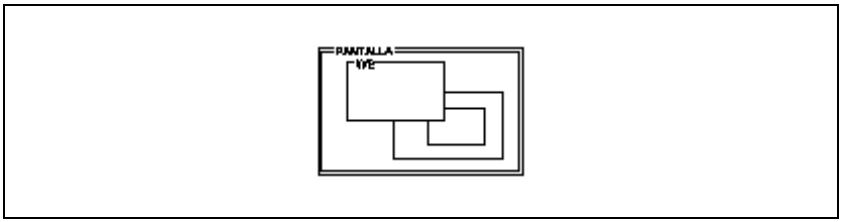
La ventana corriente está siempre en el extremo inferior derecho de la estructura. El nexo entre las ventanas uno y dos indica que éstas son hermanas; dicha relación se desprende del hecho de que ambas poseen un mismo padre. Esto podría visualizarse en pantalla de la siguiente manera:



Si cambiaramos la ventana corriente a la dos el esquema resultaría:



Y la pantalla cambiaría de la siguiente forma:



Definiciones

Padre: Toda ventana posee un padre dentro de la estructura. Una ventana creada nunca podrá superar en dimensión a su padre. Como se mencionó anteriormente, es posible utilizar a SCREEN y a WICURRENT como padres de nuevas ventanas.

Origen: Define la posición del extremo superior izquierdo de la ventana. Este origen es relativo a la ventana padre. Dimensiones:~ Las ventanas tiene una dimensión en filas (alto) y otra en columnas (ancho). Ambas están medidas en caracteres, equivalentes a una posición de pantalla. Las dimensiones no incluyen los bordes.

Atributo: Es el elemento que establece la forma en la que se desplegarán los caracteres en la ventana. El atributo de la ventana corriente puede ser asignado o modificado por la función WiSetAttr. Los atributos de IDEAFIX son:

A_NORMAL Normal

A_UNDERLINE Subrayado

A_BLINK Intermitente

A_BOLD Resaltado

A_RED Rojo

A_GREEN Verde

A_BLUE Azul

A_YELLOW Amarillo

A_CYAN Azul de Prusia

A_MAGENTA Magenta

A_WHITE Blanco

A_INVISIBLE Sin fondo

A_REVERSE Video inverso

A_RED_BG Fondo rojo

A_BLUE_BG Fondo azul

A_GREEN_BG Fondo verde

A_YELLOW_BG Fondo amarillo

A_CYAN_BG Fondo Prusia

A_MAGENTA_BG Fondo magenta

A_WHITE_BG Fondo blanco

Borde: Las ventanas generalmente tienen un borde que puede ser alterado dinámicamente. Los bordes están compuestos por tres componentes: atributo, indicador de borde e indicador de zona bordeada. Los caracteres de borde son:

SLINE_TYPE Línea simple.

DLINE_TYPE Línea doble.

ASTSK_TYPE Asteriscos.

BLANK_TYPE Blanco.

STAND_TYPE Delineado estándar. (*)

(*) Dependiendo de la capacidad de la terminal será DLINE_TYPE o BLANK_TYPE.

Los indicadores de zona bordeada son:

NO_BORDER No posee borde.

TOP_BORDER Borde superior.

LOW_BORDER Borde inferior.

LEFT_BORDER Borde izquierdo.

RIGHT_BORDER Borde derecho.

ALL_BORDER Posee los cuatro bordes.

Existen bordes definidos por IDEAFIX, estos son:

STAND_BORDER ALL_BORDER+A_REVERSE+STAND_ TYPE

SLINE_BORDER ALL_BORDER+SLINE_TYPE

DLINE_BORDER ALL_BORDER+DLINE_TYPE

ASTSK_BORDER ALL_BORDER+ASTSK_TYPE

Etiqueta: Si una ventana tiene un borde puede tener a su vez una etiqueta de identificación (window label). Las etiquetas de ventana se despliegan en el borde superior justificadas a izquierda. Generalmente se utilizan para explicar el propósito principal de la ventana.

Background: Es el atributo que define el fondo de la ventana. Los atributos válidos para Background son:

A_NORMAL A_INVISIBLE

A_REVERSE A_RED_BG

A_BLUE_BG A_GREEN_BG

A_YELLOW_BG A_CYAN_BG

A_MAGENTA_BG A_WHITE_BG

Al crear una ventana se inicializa mediante un parámetro el fondo de la misma. Para cambiar el fondo una vez creada la ventana, se utiliza la función WiSetBackGr.

Descriptores de Ventanas: Cuando se crea una ventana mediante la función WiCreate la misma retorna un descriptor de ventana. Este descriptor es utilizado cuando se realizan operaciones sobre esa ventana. Por ejemplo, para cambiar la ventana corriente, la función WiSwitchTo recibe el descriptor de ventana que se quiere hacer corriente y devuelve el descriptor de la ventana que hasta ese momento era la corriente.

Lista de Funciones

El cuadro muestra las páginas documentadas en la sección dedicada a Ventanas.

Sección	Tarea Realizada por la Sección de la Función
PopUp	Despliega menúes del tipo "pop up".
WiAttr	Maneja atributos de la ventana actual.
WiEra	Funciones de borrado dentro de la ventana actual.
WiExec	Ejecuta procesos.
WiFile	Despliega un archivo en una ventana.
WiGet	Captura caracteres dentro de la ventana actual.
WiGral	Funciones generales para manejo de ventanas.
WiHelps	Maneja textos y mensajes de ayuda.
WiMessages	Despliega mensajes.
WiMove	Mueve el cursor.
WiOpers	Crea, borra y cambia de ventana.
WiOutput	Despliega caracteres en una ventana.
WiParam	Obtiene parámetros de una ventana.
WiScroll	Maneja el "scroll" de la ventana.

Referencia de Funciones

PopUp

PopUpMenu despliega un menú de tipo "popup".

Sintaxis

```
int PopUpMenu(int nrows, int ncols, char* label,
CPFPCPI get_optx,char* garg, FPCPPI execf, char* earg[], int
attrib);
int PopUpVMenu(int nrows, int ncols, char* label,
char* argv[]);
int PopUpLMenu(int nrows, int ncols, char* label, ...);
int PopUpDbMenu (int nfils, int ncols, char *label, dbcursor
c,
int nparts, IFP valfunc, FPCP dspFunc);
```

Descripción

Crea una ventana dejando dos líneas en blanco, muestra el texto de las opciones, espera a que se elija una y devuelve el número de opción elegida. Los parámetros son:

nrows número de filas que ocupa la ventana.

ncols número de columnas que ocupa la ventana.

label título de la ventana.

get_optx puntero a una función que debe devolver la descripción de cada opción. Esta función recibirá dos parámetros: la dirección donde debe poner el texto de la descripción y el número de opción que se está buscando. Si se pide una opción inexistente, debe devolver NULL.

garg primer argumento para get_optx; es el buffer donde co locará la descripción de cada opción.

execf puntero a la función que se ejecutará al seleccionar la opción. Esta función recibirá dos parámetros: la dirección de una lista de cadena de caracteres (como el char *argv[] de una función main) y el número de opción que se seleccionó. Si no se desea usar esta función, execf debe ser NULLFP.

earg primer argumento para execf. Si no se usa, debe valer NULL.

POP_STATIC No borra la ventana del menú cuando se ejecuta execf.

POP_VOLATIL Borra la ventana del menú cuando se ejecuta execf.

POP_BORDER Cambia el borde de la ventana cuando se ejecuta execf.

POP_DEFAULT Equivale a POP_STATICO| POP_BOR DER.

PopUpVMenu también se usa para mostrar un menú en la pantalla y esperar hasta que el usuario elija una opción. El valor que devuelve es el número de opción elegida por el usuario. Los parámetros son:

nfils número de filas que ocupa la ventana.

ncols número de columnas que ocupa la ventana.

label título de la ventana.

argv arreglo de punteros a cadenas de caracteres, que con tienen el texto de las opciones del menú. El último puntero del arreglo debe ser NULL.

PopUpLMenu es similar a *PopUpVMenu*, pero en lugar de indicar el texto de las opciones mediante un arreglo, se indica cada opción como un argumento en la llamada. El último argumento siempre debe ser NULL. Los parámetros son los mismas que los de la función descripta anteriormente (*PopUpVMenu*).

PopUpDbMenu despliega una ventana con una lista de registros. Los parámetros son:

nfils número de filas que ocupa la ventana.

ncols número de columnas que ocupa la ventana.

label título de la ventana.

c cursor con rangos seteados que permite recorrar la tabla.

nparts cantidad de partes del cursor que se quiere que no sean duplicados. Si está en cero no elimina duplicados; sino, especifica la cantidad de campos hasta la cual se desea elimminar duplicados. Por ejemplo, si se especifica 1, se saltean los registros que tengan hasta el primer campo duplicado(del índice sobre el cual fue creado el cursor), si se especifica 2 hasta el segundo, etc. Para poder usar esta opción (nparts > 0) es obligatorio haber creado el cursor utilizado con el flag IO_CONTROL_BREAK.

valfunc función para validar los registros obtenidos por el cursor.

dspfunc función para armar la lista a desplegar por el menú. La misma puede tomar los datos de la tabla del cursor o de cualquiera que esté asociada a ella a través de un SetRelation.

Ejemplo

```
#include <ideafix.h>
/* PopUp configuration data */
# defines POP_LEN 5
# defines POP_WID 30
# defines POP_LABEL "Menu title"
char temp [256];
/* PopUp feed text */
char * text []= { "first row","second row", "third row",
"fourth row", "fifth row"};
void Printf (char * [],int);
char * GetText (char *, int);
wcmd (example,% I%% G%)
{
int row;
/* show and execute the PopUp * /
while ((row = PopUpMenu (POP_LEN, POP_WID,POP_LABEL,
GetText, temp, Printf, text, POP_DEFAULT))! = ERROR);
}
char * GetText (char * temp, int n)
/* Routine that feeds the PopUp */
{
if (n < 1 || n > 5) return NULL;
/* verify the option number */
return strcpy (temp, text [n-1]);
}
/* Routine executed when an option is selected */
void Printf (char *arg[], int n)
{
WiMsg ( "has been selected the% s" , arg [n]);
}
```

Consultar

WiMsg(WI)

WiAttr

Maneja atributos de la ventana actual.

Sintaxis

```
attr_type WiGetAttr();
void WiSetAttr(attr_type at);
void WiSetBackGr(attr_type at);
attr_type WiInAttr(int row, int col);
void WiSetBorder(attr_type at, char* label);
```

Descripción

WiGetAttr devuelve el atributo de la ventana actual.

WiInAttr devuelve el atributo del carácter que se encuentra en la fila fil y columna col de la ventana actual. Si fil y col valen DEFAULT, devuelve el atributo del carácter que está sobre el cursor.

WiSetAttr pone el atributo at a la ventana actual. Los valores posibles para at son:

A_NORMAL normal

A_UNDERLINE subrayado

A_BLINK parpadeante

A_BOLD resaltado

A_REVERSE inverso

A_INVISIBLE color del fondo

A_GREEN verde

A_BLUE azul

A_YELLOW amarillo

A_CYAN cyan

A_MAGENTA magenta

A_WHITE blanco

A_RED rojo

WiSetZackGr pone el atributo **at** al fondo de la ventana actual. Los valores posibles para **at** son:

A_NORMAL normal

A_INVISIBLE color del frente

A_INVERSE inverso

A_RED_BG rojo

A_BLUE_BG azul

A_GREEN_BG verde

A_YELLOW_BG amarillo

A_CYAN_BG cyan

A_MAGENTA_BG magenta

A_WHITE_BG blanco

WiSetBorder pone el atributo at y el título label al borde de la ventana actual. Los valores posibles para at son:

NO_BORDER no dibuja ninguna línea.

TOP_BORDER dibuja sólo el borde superior.

LOW_BORDER dibuja sólo el borde inferior.

LEFT_BORDER dibuja sólo el borde izquierdo.

RIGHT_BORDER dibuja sólo el borde derecho.

ALL_BORDER dibuja el recuadro completo.

SLINE_TYPE dibuja con líneas simples.

DLINE_TYPE dibuja con líneas dobles.

ASTSK_TYPE dibuja con asteriscos.

BLANK_TYPE dibuja con blancos.

STAND_TYPE equivale a DLINE_TYPE.

STAND_BORDER equivale a ALL_BORDER|A_REVERSE |STAND_TYPE.

SLINE_BORDER equivale a ALL_BORDER|SLINE_TYPE.

DLINE_BORDER equivale a ALL_BORDER|DLINE_TYPE.

ASTSK_BORDER equivale a ALL_BORDER|ASTSK_TYPE.

WiEra

Funciones de borrado dentro de la ventana actual.

Sintaxis

```
void WiEraCol();
void WiEraEop();
void WiEraLine();
void WiErase();
void WiDelChar(int n, int pos);
void WiDelLine(int n);
```

Descripción

WiEraEol borra hasta el final del renglón.

WiEraEop borra hasta el final de la página.

WiEraLine borra toda la línea actual.

WiErase borra toda la ventana actual.

WiDelChar borra n caracteres desde la posición actual del cursor; el parámetro pos no se utiliza.

WiDelLine borra la n-ésima línea y cubre el espacio, desplazando un lugar hacia arriba todas las demás líneas.

WiExec

Ejecuta procesos.

Sintaxis

```
int WiExecCmd(char* titulo, char* cmd);
int ExecPipe(char* cmd, int nfils, int ncols,
attr_type border, char* label);
int ExecShell(char* cmd);
int ExecMenu(char* titulo, char* name, int flags);
```

Valor de Retorno

Estas cuatro funciones devuelven el mismo código de salida que devuelve el proceso ejecutado.

Descripción

WiExecCmd ejecuta un comando usuario del Window Manager, volviendo al proceso actual cuando termina. Los parámetros son:

title título de la tarea lanzada, que aparecerá en el menú de tareas activas al digitar la tecla SUSPENDER.

cmd cadena de caracteres con el comando a ejecutar.

ExecPipe ejecuta un comando a través de un "pipe", mostrando la salida en una ventana, que se crea en la posición actual del cursor. Los parámetros son:

cmd cadena de caracteres con el comando a ejecutar.

nfils número de filas que ocupará la ventana.

ncols número de columnas que ocupará la ventana.

border atributo para el borde de la ventana:

NO_BORDER no dibuja ninguna línea.

TOP_BORDER dibuja sólo el borde superior.

LOW_BORDER dibuja sólo el borde inferior.

LEFT_BORDER dibuja sólo el borde izquierdo.

RIGHT_BORDER dibuja sólo el borde derecho.

ALL_BORDER dibuja el recuadro completo.

SLINE_TYPE dibuja con líneas simples.

DLINE_TYPE dibuja con líneas dobles.

ASTSK_TYPE dibuja con asteriscos.

BLANK_TYPE dibuja con blancos.

STAND_TYPE equivale a DLINE_TYPE.

STAND_BORDER equivale a ALL_BORDER|A_REVERSE |STAND_TYPE.

SLINE_BORDER equivale a ALL_BORDER|SLINE_TYPE.

DLINE_BORDER equivale a ALL_BORDER|DLINE_TYPE.

ASTSK_BORDER equivale a ALL_BORDER|ASTSK_TYPE.

label título de la ventana.

ExecShell ejecuta un comando a través del shell. El comando recibe toda la pantalla limpia, y cuando termina, se vuelve a poner lo que estaba en la pantalla. cmd es una cadena de caracteres que contiene el comando a ejecutar.

ExecMenu ejecuta un menú en una ventana. El menú debe estar guardado en un archivo llamado "nombre.mnÓ. Si el archivo no existe o tiene errores de sintaxis, ExecMenu devuelve ERROR. Los parámetros son:

title título para la ventana.

name nombre del archivo del menú (se asume la extensión ".mn")

flags los posibles valores son:

MENU_MENU deja la ventana del menú en la pantalla al llamar a otro programa de aplicación.

MENU_PGM borra la ventana del menú de la pantalla al llamar a otro programa de aplicación.

MENU_MSGERR si no existe el archivo indicado, muestra un mensaje de error.

MENU_DEFAULT equivale a MENU_MENU|MENU_MSGERR.

Ejemplo

```
#include <ideafix.h>
#include <gndefs.h>
wcmd (example,% I%% G%)
{
/* create a work window * /
WiCreate (SCREEN, 0, 0, WiHeight (SCREEN), WiWidth (SCREEN),
SLINE_BORDER, NULL_STR, TO_NORMAL);
WiPrintf ( "This is a test" );
WiRefresh ();
sleep (3);
/* execute an O.S. command */
ExecShell ( "ls -l" );
WiRefresh ();
sleep (3);
}
```

En este ejemplo se crea una ventana, se imprime un mensaje sobre la misma y luego se usa ExecShell para ejecutar el comando ls -l de UNIX, que lista el directorio actual.

Una vez finalizada la ejecución del comando, se refresca la pantalla y se puede volver a ver lo que había antes.

En el siguiente ejemplo, se realiza la misma tarea, pero utilizando *ExecPipe* en lugar de *ExecShell*. De esta forma, el comando **ls -l** se ejecuta dentro de una ventana, y no en pantalla completa.

```
#include <ideafix.h>
#include <gndefs.h>
wcmd (example, %I% G%)
{
/* create a work window * /
WiCreate (SCREEN, 0, 0, 22, 78, SLINE_BORDER, "",TO_NORMAL);
WiPrintf ( "This is a test" );
WiRefresh ();
sleep (2);
/* execute an O.S. command trapped in a window */
ExecPipe ( "ls - l" , 15, 78, STAND_BORDER,
```

```
   "Current Directory" );
   WiRefresh();
   sleep (2);
   }
```

Consultar

WiOpers(WI)

WiMessages(WI)

WiGral(WI)

WiParam(WI)

WiFile

Despliega un archivo en una ventana.

Sintaxis

```
   void WiDispFile(char* filename, int nfile, int ncols,
   char* label);
   void WiHelpFile(char* filename, char* label);
```

Descripción

WiDispFile muestra el contenido de un archivo dentro de una ventana, dando la posibilidad de moverse con las flechas y las teclas PAGE_UP y PAGE_DOWN. Los parámetros son:

filename nombre del archivo a buscar.

nfilsnumber cantidad de filas que ocupará la ventana.

ncols cantidad de columnas que ocupará la ventana.

label título de la ventana.

WiHelpFile muestra el contenido de un archivo, asumiendo que está dentro de un directorio "... /hlp" y que tiene extensión ".hlp". La ventana donde aparecerá el archivo ocupará toda la pantalla. Los parámetros son:

filename nombre del archivo (si se indica una extensión, se ignora).

label título de la ventana.

WiGet

Captura caracteres dentro de la ventana actual.

Sintaxis

```
UChar WiGetc();
UChar* WiGets(UChar* s);
int WiGetField(type typ, UChar* buff, UShort opt,
short length, short olength, short ndec,
char* tst mask, char* omask);
UChar WiInChar(int row, int col);
```

Descripción

WiGetc lee un carácter del teclado. Si es un comando del Window Manager, lo procesa y lee otro carácter.

WiGets lee una cadena de caracteres ingresada sobre la ventana actual y la copia sobre s. La dirección que devuelve apunta al resultado. Antes de llamar a WiGets, hay que reservar suficiente memoria en s.

WiGetField define un campo dentro de la ventana actual, y espera hasta que se complete su valor. Los parámetros son:

typ uno de los siguientes tipos de campos:

TY_NUMERIC numérico

TY_FLOAT numérico con punto flotante

TY_DATE fecha

TY_TIME hora

TY_STRING alfanumérico

TY_BOOL booleano (verdadero o falso)

buf buffer donde se almacena el contenido del campo.

opt atributo del campo; puede ser 0 o alguno de los siguientes:

WGET_DSPONLY el campo no puede ser editado, sólo muestra su contenido.

WGET_COMMA coloca el separador de miles cada tres digitos enteros.

WGET_AUTOENTER no inicializa el campo antes de reci bir la entrada.

WGET_CDIGIT incluye el dígito verificador.

WGET_CDIGIT_DASH incluye el dígito verificador, separado por un guión.

WGET_CDIGIT_SLASH incluye el dígito verificador, separa do por una barra.

lenght longitud real del campo.

olenght longitud del campo en la pantalla.Si es menor que length, se hará un "scroll".

ndec cantidad de decimales (si el campo es numérico).

tstmask máscara de validación del campo, obtenida con Com pileMask. Si no se usa, vale NULL.

omask máscara de salida del campo, obtenida con Compile Mask. Si no se usa, vale NULL.

Antes de llamar a *WiGetField*, se debe reservar suficiente memoria en buff.

Si hay algún tipo de error, *WiGetField* devuelve ERROR. De lo contrario, devuelve el código de la tecla que se apretó.

WiInChar devuelve el carácter que se encuentra en la fila fil y columna col de la ventana actual. Si *fil* y *col* valen DEFAULT, devuelve el carácter que está sobre el cursor.

Ejemplo

```
#include <ideafix.h>
#defines LENGTH 10
#defines CANT 4
#defines MASK "aa.aa.aaa"
#defines FOREVER for (;;)
UChar buff[CANT][LENGTH];
char omask[LENGTH], tstmask[LENGTH];
wcmd (example, %I% %G%)
{
/* create a work window */
WiCreate (SCREEN, 10, 10, 10, 30, STAND_BORDER,
"Label" ,TO_NORMAL);
/* generate a mask for the third field */
CompileMask(MASK, tstmask, omask);
FOREVER {
/* get the first field in row 2 column 3 */
WiMoveTo (2, 3);
if (WiGetField (TY_STRING, buff[0], 0,
LENGTH-1,LENGTH-1,0, NULL, NULL) == K_END)
break;
/* get the second field in row 4 column 3 */
WiMoveTo (4, 3);
if (WiGetField (TY_NUMERIC, buff[1], 0,
LENGTH-1,LENGTH-1, 2, NULL, NULL) ==K_END)
break;
/* get the third field in row 6 column 3 */
WiMoveTo (6, 3);
if (WiGetField (TY_STRING, buff [2], 0,
LENGTH-1, LENGTH-1,0, tstmask, omask) == K_END)
break;
/* get the fourth field in row 4 column 15*/
WiMoveTo (4, 15);
if (WiGetField (TY_DATES, buff [3], 0,
LENGTH-2, LENGTH-2, 0, NULL, NULL) == K_END)
break;
}
```

```
}
```

Consultar

WiMoveTo(WI)

WiCreate(WI)

CompileMask(ST)

WiGral

Funciones generales para manejo de ventanas.

Sintaxis

```
void WiBeep();
void WiRefresh();
void WiRedraw();
window WiCurrent();
void WiCursor(bool f);
void WiInterrupts(bool f);
window WiDefPar();
void WiSetDefPar(window wi);
void WiStatusLine(int f);
void WiSetTab(int tab);
int WiGetTab();
int WiSettty();
void WiResettty();
void WiWrap(bool f);
```

Descripción

WiBeep emite un sonido de advertencia.

WiRefresh actualiza el contenido de la pantalla con los cambios realizados desde el último *WiRefresh*. Cada vez que se realiza un ingreso de datos, se llama automáticamente a esta función.

WiRedraw borra toda la pantalla y la vuelve a generar. Es equivalente a la tecla REDIBUJAR del Window Manager.

WiCurrent devuelve el descriptor de la ventana actual.

WiCursor muestra u oculta el cursor, dependiendo del valor de *f* (TRUE o FALSE).

WiInterrupts permite o impide la interrupción del proceso actual, según el valor de f (TRUE o FALSE).

WiDefPar devuelve el descriptor de la ventana padre por defecto (generalmente es SCREEN).

WiSetDefPar establece a wi como la ventana padre por defecto.

WiStatusLine activa o desactiva la línea de estado, dependiendo del valor de f (TRUE o FALSE).

WiSetTab establece cuánto mide el tabulador.

WiGetTab devuelve la medida actual del tabulador.

WiSettty prepara la terminal para trabajar en modo Window Manager.

WiResettty devuelve la terminal a su modo normal.

WiWrap habilita o deshabilita el modo "wrap", dependiendo del valor de f (TRUE o FALSE). Cuando está habilitado, al llegar al borde derecho de la ventana, el cursor baja a la siguiente línea. Si el modo "wrap" está deshabilitado, se desplaza el contenido de la ventana para seguir escribiendo hacia la derecha.

WiHelps

Maneja textos y mensajes de ayuda.

Sintaxis

```
void WiSetAplHelp(char* mod, char* ver);
void WiAplHelp();
void WiHelp(int nfils, int ncols, char* label,
CPFPCPI get_txt, char* garg);
UChar* WiKeyHelp(UChar k, UChar* s);
```

Descripción

WiSetAplHelp establece la ayuda para la aplicación actual. El archivo que contiene el texto de ayuda debe tener el mismo nombre que la aplicación, con extensión ".hlp", y debe estar ubicado en un directorio "... /hlp" dentro del PATH actual.

Los parámetros de *WiSetAplHelp* se usan para poner el título de la ventana de ayuda, y son los siguientes:

mod módulo del programa de aplicación.

ver versión del programa de aplicación.

WiAplHelp despliega la ayuda para la aplicación actual según lo indicado con *WiSetAplHelp*.

WiHelp despliega la ayuda dentro de una ventana, que empieza en la posición actual del cursor. Los parámetros son:

nfils número de filas que ocupará la ventana.

ncols número de columnas que ocupará la ventana.

label título de la ventana.

get_txt puntero a una función que debe devolver cada renglón de la ayuda. Esta función recibirá dos parámetros: la dirección donde debe poner el texto del renglón y el número de renglón que se está buscando. Si se pide un renglón inexistente, debe devolver NULL.

garg primer argumento para get_txt; es el buffer donde colocará la descripción de cada opción.

WiKeyHelp devuelve en s la descripción de la tecla de IDEAFIX cuyo código se indica en k. Si s vale NULL, WiKeyHelp devuelve un puntero a un buffer interno de IDEAFIX, cuyo contenido se sobreescribe en cada llamada.

Ejemplo

```
#include <ideafix.h>
#defines LARG_HELP 3
#defines ANCH_HELP 30
#defines LABEL_HELP "Help Title"
/* Text to feed the help * /
char *text[]= { "first row"," second row ", " third row",
"fourth row", " fifth row "};
char text [256];
char *Gtext (char *, int);
wcmd (example,% I%% G%)
{
iHelp (LARG_HELP, ANCH_HELP, LABEL_ HELP, Gtext, text);
}
char * Gtext (char *text, int n)
/* Routine that feeds the PopUp */
{
if (n < 1 || n > 5) return NULL;
/* verify the option number */
return strcpy (text, text [n-1]);
}
```

To show the description of a key, it must be made:

```
#include <ideafix.h>
wcmd (example,% I%% G%)
{
UChar buf [256];
WiKeyHelp (K_TAB, buf);
WiMsg ( "The description of the key is% s", buf);
}
```

Consultar

WiMsg(WI)

WiMessages

Despliega mensajes.

Sintaxis

```
int WiMsg(char* fmt, ...);
int WiVMsg(int flag, char* fmt, va_list ap);
void DisplayMsg(bool wait_conf, char* fmt, ...);
void DisplayVMsg(bool wait conf, char* fmt, va_list ap);
void ClearMsg();
wdflag WiDialog(wdflag param, wdflag default,
char * title, char * fmt, ...);
```

Valor de Retorno

WiDialog retorna el modificador correspondiente al botón seleccionado. Si el modificador no tiene valor de retorno (por ejemplo, cuando se establecen WD_PRESERVE o WD_RELEASE, vease la descripción a continuación) retornará WD_OK.

Descripción

WiMsg imprime un mensaje dentro de una ventana, espera que se apriete una tecla, borra la ventana y devuelve el código de la tecla apretada. Los argumentos se indican de la misma forma que para printf.

NOTA: Se recomienda discontinuar el uso de esta función para realizar ventanas de diálogo y en su reemplazo usar WiDialog.

WiVMsg imprime una serie de elementos contenidos en una va_list dentro de una ventana, y según el primer parámetro, espera que se apriete una tecla o vuelve dejando la ventana en la pantalla. También se puede usar para quitar una ventana que se había dejado antes.

El parámetro *flag* es una de las siguientes constantes:

MSG_WAIT espera hasta que se apriete una tecla, y luego borra la ventana.

MSG_PRESERVES deja la ventana en la pantalla.

MSG_RELEASE quita la ventana anterior de la pantalla.

DisplayMsg coloca un mensaje en la última línea de la pantalla. Si el primer parámetro (wait_conf) vale verdadero, espera hasta que se apriete una tecla. El resto de los parámetros funciona igual que para printf.

DisplayVMsg es similar a *DisplayMsg*, pero recibe los elementos a imprimir dentro de una va_list.

ClearMsg borra el último mensaje de la pantalla.

WiDialog abre una ventana de diálogo.

param: este parámetro especifica el modificador para el botón y el modo de la ventana. Estos modificadores deben ser separados con un or de bits (por ejemplo, con un pipe "|"). Los valores posibles son:

default: especifica el botón que estará activo cuando se abra la ventana. Sus posibles valores son:

title: título de la ventana.

fmt, varargs, ... : texto de la ventana. Este conjunto de parámetros debe ser pasado con formato de printf.

Además, se permite imprimir la descripción de una tecla del Window Manager. El formato es %tecla y puede estar modificado por los atributos estándares (longitud, alineación, etc.)

Los modificadores que pueden ser pasados a param y default son:

Modificadores Normales:

• WD_OK

• WD_YES

• WD_NO

• WD_ABORT

• WD_CANCEL

• WD_RETRY

• WD_IGNORE

Modificadores Especiales:

• WD_ERROR Con esta opción el fondo se torna rojo. En Windows aparece un ícono de STOP.

• WD_PRESERVE Este modificador permite que la ventana activa sea preservada hasta la próxima llamada a WiDialog. Esta opción no puede combinarse con modificadores normales, ni puede alternar con la ventana padre.

• WD_RELEASE Cierra una ventana abierta previamente con el modificador WD_PRESERVE.

• WD_HERE Este modificador permite que la ventana aparezca en una posición que dependa de la posición del cursor, en vez de estar centrada.

Consultar

WiOpers(WI)

WiGral(WI)

WiOutputs(WI)

WiMove

Mueve el cursor.

Sintaxis

```
void WiMoveTo(short f, short c);
void WiRMoveTo(short f, short c);
```

Descripción

WiMoveTo lleva al cursor a la fila f y columna c de la ventana actual.

WiRMoveTo desplaza el cursor f filas y c columnas a partir de la posición actual del cursor.

WiOpers

Crea, borra y cambia de ventana.

Sintaxis

```
window WiCreate(window parent, int f_org, int c_org, int
nfile,
int ncols, attr_type border, char* label, attr_type backg);
void WiDelete(window wi);
void WiDeleteAll();
window WiSwitchTo(window wi);
```

Descripción

WiCreate crea una nueva ventana, que pasa a ser la ventana actual. Devuelve el descriptor de la ventana creada. Los parámetros son:

parent descriptor de la ventana padre (aquella de la que dependerá la ventana creada). Si vale SCREEN, la nueva ventana dependerá directamente de la pantalla. Si vale WICURRENT, dependerá de la ventana actual.

f_org fila donde empezará la ventana.

c_org columna donde empezará la ventana.

nfils cantidad de filas que ocupará la ventana.

ncols cantidad de columnas que ocupará la ventana.

border tipo de borde para la ventana creada. Los valores posi bles son:

NO_BORDER no dibuja ninguna línea.

TOP_BORDER dibuja sólo el borde superior.

LOW_BORDER dibuja sólo el borde inferior.

LEFT_BORDER dibuja sólo el borde izquierdo.

RIGHT_BORDER dibuja sólo el borde derecho.

ALL_BORDER dibuja el recuadro completo.

SLINE_TYPE dibuja con líneas simples.

DLINE_TYPE dibuja con líneas dobles.

ASTSK_TYPE dibuja con asteriscos.

BLANK_TYPE dibuja con blancos.

STAND_TYPE equivale a DLINE_TYPE.

STAND_BORDER equivale a ALL_BORDER|A_ REVER SE|STAND_TYPE.

SLINE_BORDER equivale a ALL_BORDER |SLINE_TYPE.

DLINE_BORDER equivale a ALL_BORDER|DLINE _TYPE.

ASTSK_BORDER equivale a ALL_BORDER|ASTSK _TYPE.

label título de la ventana.

backg atributo para el fondo de la ventana creada. Los valores posibles son:

A_NORMAL normal

A_INVISIBL Ecolor del frente

A_REVERSE inverso

A_RED_BG rojo

A_BLEU_BG azul

A_GREEN_BG verde

A_YELLOW_BG amarillo

A_CYAN_BG cyan

A_MAGENTA_BG magenta

A_WHITE_BG blanco

WiDelete borra la ventana cuyo descriptor es wi, y todas las ventanas que dependían de esa.

WiDeleteAll borra todas las ventas creadas por el proceso actual.

WiSwitchTo cambia la ventana actual a la indicada en wi. Si hay algún problema, devuelve ERROR. De lo contrario, devuelve el descriptor de la ventana que estaba activa antes de hacer el cambio.

La ventana que se desea activar no puede tener ventanas "hijas".

Ejemplo

Ejemplo 1:

```
#include <ideafix.h>
#include <gndefs.h>
wcmd(example, %I% %G%)
{
window w1, w2;
/* create a window */
w1 = WiCreate (SCREEN, 10, 10, 10, 30, STAND_BORDER,
"This is a label" , A_NORMAL);
WiRefresh ();
sleep (2);
/* create other window */
w2 = WiCreate (w1, 2, 2, 4, 10, SLINE_BORDER, NULL_STR,
A_NORMAL);
WiRefresh ();
sleep (3);
}
```

Ejemplo 2:

```
#include <ideafix.h>
wcmd(example, %I% %G%)
{
window w1, w2;
/* create a work window * /
w1 = WiCreate (SCREEN, 1, 1, 10, 30, STAND_BORDER,
"Window 1", A_NORMAL);
/* create other work window * /
w2 = WiCreate (SCREEN, 1, 50, 10, 30, STAND_BORDER,
"Window 2", A_NORMAL);
/* change to the first window * /
WiSwitchTo (w1);
WiPrintf ( "I am in the first window" );
WiGetc ();
/* change to the second window * /
WiSwitchTo (w2);
WiPrintf ( "I am in the second window" );
WiGetc ();
}
```

Consultar

WiGral(WI)

WiOutputs(WI)

WiGet(WI)

WiOutput

Despliega caracteres en una ventana.

Sintaxis

```
int WiPutc(UChar c);
void WiPuts(char* s);
void WiPrintf(char* fmt, ...);
void WiInsChar(int n, int pos);
void WiInsLine(int n);
```

Descripción

WiPutc escribe el carcter c en la ventana actual.

WiPuts escribe una cadena de caracteres en la ventana actual.

WiPrintf escribe una lista de parámetros (al estilo printf) en la ventana actual.

WiInsChar inserta n espacios de la posición actual el cursor. El parámetro pos no se usa.

WiInsLine inserta una linea en blanco antes de una fila en la ventana corriente. Si es necesario, realiza un scroll para hacerlo visible.

WiParam

Obtiene parámetros de una ventana.

Sintaxis

```
int Wicol (window wi);
int WiHeight (window wi);
int WiLine (window wi);
int WiOrgCol (window wi);
int WiOrgRow (window wi);
window WiParent (window wi);
int WiWidth (window wi);
```

Descripción

WiCol devuelve el número de columna donde está el cursor dentro de la ventana wi.

WiHeight devuelve la cantidad de filas que ocupa la ventana wi.

WiLine devuelve el número de fila donde está el cursor dentro de la ventana wi.

WiOrgCol devuelve el número de columna donde empieza la ventana wi.

WiOrgRow devuelve el número de fila donde empieza la ventana wi.

WiParent devuelve el descriptor de la ventana padre de la ventana wi.

WiWidth devuelve la cantidad de columnas que ocupa la ventana wi.

WiScroll

Maneja el "scroll" de la ventana.

Sintaxis

```
void WiSetScroll (int top, int bot, int left, int right);
void WiScroll (int n);
```

Descripción

WiSetScroll define la región de la ventana actual donde se realizará el "scroll". Los parámetros son:

top fila superior de la región.

bot fila inferior de la región.

left columna izquierda de la región.

right columna derecha de la región.

WiScroll hace un scroll de la ventana actual en n filas. Si n es positivo, el scroll se hace hacia arriba. Si n es negativo, se hace hacia abajo.

4

Apéndices

Apéndice A

Interfaz Ideafix-Cobol

En este apéndice se describirá el modo de acceder desde Cobol a las funciones de Ideafix y Essentia.

Concepto general

Durante la ejecución, un módulo Cobol es normalmente interpretado, por un módulo llamado RUNCOBOL, que es el que se invoca verdaderamente.

Básicamente el mecanismo es usar rutinas implementadas en el módulo RUNCOBOL, que deberán invocarse al momento de corrida, pues la manera de invocar un programa ejecutable Cobol (.COB) es:

```
RUNCOBOL [prog]
```

donde [prog] es el nombre del programa.

El módulo RUNCOBOL es necesario ya que provee los procedimientos invocados por el programa en Cobol durante el tiempo de corrida.

Cómo usar la interfaz

La invocación de una función de acceso a la base de datos (ya sea vía Ideafix o vía Essentia) desde un programa o subprograma Cobol, debe hacerse mediante el verbo CALL. Nótese que esto no implica la llamada a un subprograma; el procesamiento es análogo al realizado al ejecutarse un verbo cualquiera de Cobol, invocándose una función de biblioteca adecuada,

que se encuentra dentro del módulo RUNCOBOL. Las funciones antedichas están *linkeditadas* a este módulo y por ello se invocarán automáticamente.

Contenido y modificaciones de RUNCOBOL

El módulo de runtime, RUNCOBOL, incluye todos los procedimientos usuales ejecutados con los verbos primitivos del lenguaje, a los que se agregan otros específicos para Ideafix/Essentia. Existe un RUNCOBOL para Ideafix y otro para Essentia, debiendo utilizarse el necesario en cada caso según cómo vaya a funcionar la aplicación correspondiente.

Cómo invocar un comando de Ideafix/Essentia

La manera de llamar a un comando (nuevo) es la siguiente:

```
CALL "[nombre de comando]" USING [parametros]
```

donde [nombre de comando] es una cadena de caracteres (string), y [parametros] es una lista, cuyos elementos van separados por comas, cada uno de los cuales tiene alguna de las siguientes tres formas:

- BY CONTENT [id]
- BY REFERENCE [id]
- [id]

Siguiendo el estilo de Cobol, estas tres formas permiten pasar un parámetro por valor, por referencia, o de la manera *default*, respectivamente. Como es habitual en Cobol, el *default* es pasarlos por referencia.

Ejemplo

Un ejemplo típico de llamada a una función es el siguiente:

```
CALL "GetRecord" USING Mi-Archivo, 2, "indice1", FIRST-KEY,
TEST-LOCK, estado
```

lo cual se traduce en: invocar la rutina GetRecord, con los parámetros indicados. Para una explicación de las constantes FIRST-KEY y otras, véase más abajo.

Constantes usuales

Siempre deberá incluirse un archivo *idea.cpy* que contiene los nombres y valores de "constantes" muy usadas. Por consiguiente, las variables aquí declaradas no deberían, en general, ser modificadas por el programador así como tampoco se les asignarán valores ex profeso.

```
01 READ-MODE.
02 THIS-KEY PIC 99 VALUE 1.
02 NEXT-KEY PIC 99 VALUE 2.
02 PREV-KEY PIC 99 VALUE 4.
02 FIRST-KEY PIC 99 VALUE 8.
02 LAST-KEY PIC 99 VALUE 16.
02 THIS-OR-NEXT-KEY PIC 99 VALUE 3.
02 THIS-OR-PREV-KEY PIC 99 VALUE 5.
01 LOCK-MODE.
02 NOT-LOCK PIC 9(5) VALUE 0.
02 WAIT-LOCK PIC 9(5) VALUE 1.
02 TEST-LOCK PIC 9(5) VALUE 3.
02 WAIT-READ-LOCK PIC 9(5) VALUE 16385.
02 TEST-READ-LOCK PIC 9(5) VALUE 16387.
02 WAIT-PLOCK PIC 9(5) VALUE 8192.
02 TEST-PLOCK PIC 9(5) VALUE 8194.
02 WAIT-READ-PLOCK PIC 9(5) VALUE 24576.
02 TEST-READ-PLOCK PIC 9(5) VALUE 24578.
01 RETURN-VALUES.
02 OK PIC S9 VALUE 0.
02 ERR PIC S9 VALUE -1.
02 IO-LOCKED PIC S9 VALUE -2.
```

que definen registros adecuados.

Constantes de locks

Al utilizar locks se emplean los mismos valores de constantes que en el código CFIX (ver secciones correspondientes). Cabe aclarar que los nombres de las constantes son diferentes en COBOL. La siguiente es una equivalencia entre dichos nombres:

```
NOT-LOCK = IO_NOT_LOCK
WAIT-LOCK = IO_LOCK|IO_WRITE o IO_LOCK
WAIT-READ-LOCK = IO_LOCK|IO_READ
TEST-LOCK = IO_LOCK|IO_WRITE|IO_TEST o IO_LOCK|IO_TEST
TEST-READ-LOCK = IO_LOCK|IO_READ|IO_TEST
WAIT-PLOCK = IO_PLOCK|IO_WRITE o IO_PLOCK
WAIT-READ-PLOCK = IO_PLOCK|IO_READ
TEST-PLOCK = IO_PLOCK|IO_WRITE|IO_TEST o IO_PLOCK|IO_TEST
TEST-READ-PLOCK = IO_PLOCK|IO_READ|IO_TEST
```

Table 1: equivalencias de tipos COBOL/CFIX

Ejemplo comentado

El siguiente es un programa completo en Cobol que hace uso de las rutinas mas comunes. Comienza con los encabezados típicos:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "clientes".
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. RMCOBOL-85.
OBJECT-COMPUTER. RMCOBOL-85.
```

En la DATA DIVISION colocamos:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
copy "idea.cpy".
copy "demo.cpy".
```

para incluir o "copiar" los archivos explicitados.

Declaración de datos y estructuras del programador

El segundo archivo incluido, demo.cpy, contiene la definición completa del esquema a utilizar. Esto se realiza definiendo el registro correspondiente a cada tabla (archivo) del esquema. En primer lugar, definimos una tabla con datos de clientes:

```
01 CLIENTES.
06 TAB-ID PIC 9(5) BINARY VALUE 512.
06 CLIENTES-DATA.
11 CLIENO-KEY.
16 CLIENO PIC S9(4) VALUE ZEROES.
11 NOMIDX-KEY.
16 NOMBRE PIC X(30) VALUE SPACES.
11 DIREC PIC X(30) VALUE SPACES.
11 PROV PIC X(1) VALUE SPACES.
11 LOCAL PIC S9(2) VALUE ZEROES.
11 COD-P PIC X(4) VALUE SPACES.
11 FEALTA PIC 9(6) VALUE ZEROES.
11 IVA PIC S9(2)V9(2) VALUE ZEROES.
```

que es el registro de la tabla CLIENTES. El siguiente es el registro de otra tabla que almacenará con datos de movimientos:

```
01 MOVIM.
06 TAB-ID PIC 9(5) BINARY VALUE 1024.
06 MOVIM-DATA.
11 ORDERNO-KEY.
16 ORDERNO PIC S9(4) VALUE ZEROES.
16 ITEMNO PIC S9(4) VALUE ZEROES.
11 CANTID PIC S9(4) VALUE ZEROES.
11 PRVTA PIC S9(12)V9(2) VALUE ZEROES.
```

que es el registro de MOVIM para almacenar cada movimiento. Es importante destacar que el uso de estos registros reemplaza (y hace innecesaria) su definición en la FILE SECTION del programa. Vale decir, no se deben usar definiciones en dicha sección cuando se trata de registros de tablas de Ideafix. Nótese la aparición de la "constante" TAB-ID, un número identificador de tabla, que el utilitario DGEN genera automáticamente. Este campo normalmente no debe ser modificado por el programador, puesto que se utilizará (ver más adelante, en la sección relacionada con el uso del DGEN).

Continuando, inicializamos los siguientes titulos y mensajes de error:

```
01 title pic X(60) value
"Clientes".
```

```
01 MSG1 pic X(60) value
"Error opening schema.".
01 MSG2 pic X(60) value
"El cliente solicitado no existe.".
01 schema pic X(30) value "demo".
01 sch pic S9(8) binary.
01 stat pic S9(4) value 0.
01 option pic 9 value 0.
01 dummy pic X value spaces.
```

Veamos ahora la PROCEDURE DIVISION. El programa principal puede comenzar con el rótulo MAIN:

```
PROCEDURE DIVISION.
MAIN.
call "OpenSchema" using schema, sch.
```

abre un esquema llamado "demo" (ver la definición de la variable schema), retornando el resultado de la operación en la variable sch. Se puede controlar así si hubo error o no:

```
if sch = ERR
then
perform clear-scr
display MSG1, low, line 2, position 1
go to exit-func
end-if.
```

A continuacion, se invoca la funcion que llamamos do-form hasta que la opción elegida sea salir.

```
perform do-form until option = 4.
```

Acto seguido, se cierra el esquema y se finaliza la ejecucion del programa.

```
call "CloseSchema" using sch.
go to exit-func.
```

Veamos la "función" do-form.

```
do-form.
perform menu.
perform clear-scr.
if option < 4 and option > 0
then
perform get-key
if option = 1 then perform get-rec end-if
if option = 2 then perform put-rec end-if
if option = 3 then perform del-rec end-if
perform wait
perform clear-buff
end-if.
```

lo que esto hace es mostrar el menú principal, que permite al usuario seleccionar una opción, luego borra la pantalla, y por último decide qué operación ejecutar según la opción elegida.

Las funciones antedichas, y otras útiles, pueden implementarse así:

```
clear-scr.
display " " erase.
display title, high, line 1.
clear-buff.
move zero to CLIENO.
move spaces to NOMBRE.
move spaces to DIREC.
move spaces to PROV.
move zero to LOCAL.
move spaces to CP.
move spaces to FEALTA.
move zero to IVA.
```

Rutina de espera de una tecla:

```
wait.
display "Presione enter para continuar.", low,
line 20, position 1.
accept dummy, line 20 position 40 no beep.
```

Rutina de lectura de nro. de cliente:

```
get-key.
display "Cliente :", low , line 3, position 1.
accept CLIENO, line 3, position 12, TAB no beep.
```

Rutina de lectura de un registro.

```
get-rec.
call "GetRecord" using CLIENTES stat.
if stat = ERR
then
display MSG2, low, line 3, position 1
else
perform form
perform show
end-if.
```

Rutina de grabado de un registro.

Notar que se pide poner lock al registro puesto que se lo va a modificar. Primero se lo lee:

```
put-rec.
call "GetRecord" using CLIENTES, WAIT-LOCK.
perform form.
perform show.
```

luego el usuario ingresan los datos del cliente:

```
accept NOMBRE, high, line 4 position 12 update no beep.
accept DIREC, high, line 5 position 12 update no beep.
accept PROV, high, line 6 position 12 update no beep.
accept LOCAL, high, line 7 position 12 update no beep.
```

```
accept CP, high, line 8 position 12 update no beep.
accept FEALTA, high, line 9 position 12 update no beep.
accept IVA, high, line 10 position 12 update no beep.
```

Por último, el registro es grabado:

```
call "PutRecord" using CLIENTES.
```

Rutina de borrado de un registro.

Acá también se hace necesario usar locks:

```
del-rec.
call "GetRecord" using CLIENTES, WAIT-LOCK, stat.
if stat = ERR
then
display MSG2, low, line 3, position 1
else
call "DelRecord" using CLIENTES
end-if.
```

Rutinas para mostrar en una pantalla los datos del cliente.

Los nombres de los campos:

```
form.
display "Cliente :", low, line 3, position 1.
display "Nombre :", low, line 4, position 1.
display "Direcc. :", low, line 5, position 1.
display "Prov. :", low, line 6, position 1.
display "Local. :", low, line 7, position 1.
display "C.P. :", low, line 8, position 1.
display "Ingreso :", low, line 9, position 1.
display "IVA :", low, line 10, position 1.
```

Los contenidos de los campos:

```
show.
display CLIENO, high, line 3, position 12 convert.
display NOMBRE, high, line 4, position 12.
display DIREC, high, line 5, position 12.
display PROV, high, line 6, position 12.
display LOCAL, high, line 7, position 12 convert.
display CP, high, line 8, position 12.
display FEALTA, high, line 9, position 12.
display IVA, high, line 10, position 12 convert.
```

El menú principal es:

```
menu.
perform clear-scr.
display "1. Consulta ", low, line 4 position 10.
display "2. Alta/Modif.", low, line 5 position 10.
display "3. Baja ", low, line 6 position 10.
display "4. Salir ", low, line 7 position 10.
```

```
display "Especifique la operacion", low,
line 10 position 20.
accept option, line 10, position 47, tab no beep.
```

Notar que, siguiendo el estilo de los programas clásicos en Cobol, se definen secciones, dándoles rótulos adecuados, una para cada opción del menú.

Por último, para terminar la ejecución, se puede utilizar la sección:

```
exit-func.
exit program.
stop run.
```

Sintaxis general

A continuación damos reglas sintácticas para los llamados a las funciones típicas de Ideafix/Essentia. Nótese que no colocamos aquí las cláusulas BY CONTENT ni BY REFERENCE, aún cuando éstas pueden usarse, puesto que al omitirlas el pasaje de los parámetros se realiza de la manera sugerida por la función en cada caso.

Manejo de esquemas

`CALL "OpenSchema" USING [schema-name] [sch-id]`

Parámetros:

- [schema-name] : string, nombre de esquema, pasa por valor
- [sch-id] : pasa por referencia, en esta variable se devuelveel identificador del esquema para ser usado luego}

Abre el esquema schema-name. Si se especifica [sch-id], se copiará el identificador del esquema en dicha variable. En caso de error al abrirlo, se devuelve en dicha variable el valor ERR. Si no se especifica esta variable, entonces en caso de error el programa aborta.

Una vez abierto un esquema, éste queda establecido como esquema corriente.

`CALL "CloseSchema" [ USING [sch-id]`

Parámetro:

- [sch-id] : identificador de esquema obtenidocon OpenSchema, pasa por valor}

Si [sch-id] es especificado se cierra el esquema correspondiente; si no, se cierra el esquema corriente.

Manejo de Tablas

```
CALL "SetTableCache" USING [sch-id] [record] [rec-amount] [ret-val]
```

Reserva memoria cache para el esquema, para una cantidad de registros igual a la cantidad especificada, para la tabla dada.

Parámetros:

- [sch-id] : identificador del esquema, pasa por valor
- [record] : registro con estructura de la tabla correspondiente, pasa por valor}
- [rec-amount] : numérico sin signo, cantidad de registros para los cuales crear el cache, pasa por valor}
- [ret-val] : valor de retorno, pasa por referencia

Esta función permite definir el tamano de la memoria cache a utilizar para una tabla dada dentro de un esquema dado.

El valor default de [sch-id] es el esquema corriente.

Manejo de registros

```
CALL "GetRecord" USING [sch-id] [record] [index-name]
[find-mode] [lock-mode] [ret-val]
```

Parámetros:

- [sch-id] : identificador de esquema obtenido con OpenSchema, pasa por referencia}
- [record] : registro con estructura de la tabla correspondiente, pasa por referencia}
- [index-name] : cadena, nombre del índice a usar, pasa por valor
- [find-mode] : el modo de búsqueda (ver partes anteriores), pasa por valor
- [lock-mode] : modo de lock (ver partes anteriores), pasapor valor}
- [ret-val] : valor de retorno, pasa por referencia

Copia en [record] el registro (buffer) según los valores de los campos que conforman la clave correspondiente. Si se le pasa la variable [ret-val] , se devuelve en ella el estado que resulta de la operación (para referencia de los estados, véanse las partes anteriores).

Los valores default son:

- [index-name] : primary key
- [find mode] : THIS-KEY

- [lock mode] : NOT-LOCK

`CALL "PutRecord" USING [sch-id] [record]`

Parámetros:

- [sch-id] : identificador de esquema obtenido con OpenSchema, pasa por valor}
- [record] : registro (buffer), pasa por valor

Modifica/Agrega el contenido del registro [record] en la tabla correspondiente. El valor default de [sch-id] es el esquema corriente.

`CALL "DelRecord" USING [sch-id] [record]`

Parámetros:

- [sch-id] : identificador de esquema obtenido con OpenSchema, pasa por valor
- [record] : registro (buffer), pasa por valor

Elimina el registro [record] de la tabla correspondiente, según los valores en [record] de la clave primaria. El valor default de [sch-id] es el esquema corriente.

`CALL "AddRecord" USING [sch-id] [record] [ret-val]`

Permite agregar un registro dado a una tabla.

Parámetros:

- [sch-id] : identificador de esquema obtenido con OpenSchema, pasa por valor
- [record] : registro (buffer), pasa por valor
- [ret-val] : valor de retorno, pasa por referencia

Agrega el contenido del registro [record] en la tabla correspondiente. El valor default de [sch-id] es el esquema corriente. Si un registro con la misma clave ya existe en dicha tabla, se produce un error. Notar la diferencia con PutRecord. Si [ret-val] no fue especificado, entonces la transacción corriente quedará corrupta, pudiendo comprobarse el error usando TransOk (ver TransOk). Si, en cambio, se especificó [ret-val], entonces la transacción corriente no quedará corrupta sino que se devuelve en dicha variable el error correspondiente (valor ERR).

`CALL "BindRecord" USING [sch-id] [record] [table-name] [ret-val]`

Asocia un registro a una tabla.

Parámetros:

- [sch-id] : identificador de esquema obtenido con OpenSchema, pasa por valor
- [record] : registro (buffer), pasa por referencia
- [table-name] : variable indicando nombre de la tabla a usar
- [ret-val] : valor de retorno, pasa por referencia

Esta función permite asociar un registro a una tabla, de modo que futuras referencias a ese registro (e.g., usando Get Record, Put Record, CreateCursor, etc.) impliquen acceso a dicha tabla. Esto se puede hacer necesario cuando la tabla a usar no es siempre la misma, i.e., dependa de una condición específica durante la ejecución.

El registro a ser asociado debe mantener el mismo formato (tipos en cada campo) del registro correspondiente a la tabla (el generado por el DGEN). El campo TAB-ID no necesita tener valor.

En caso de error en la operación, se devuelve en [ret-val] el valor ERR. Si no se especifica esta variable, entonces en caso de error el programa aborta.

Por ejemplo, el siguiente cambio puede incorporarse al ejemplo de programa Cobol anterior:

```
get-rec.
call "BindRecord" using CLONE "clientes" stat
if stat = ERR
then
display MSG3, low, line 3, position 1
else
call "GetRecord" using CLONE stat
...
```

y agregando a la WORKING-STORAGE SECTION algo como lo siguiente (definición de registro a asociar y "constante" mensaje de error correspondiente):

```
01 CLONE.
06 TAB-ID PIC 9(5) BINARY.
06 CLIENTES-DATA.
11 CLIENO-KEY.
16 CLIENO PIC S9(4).
11 NOMIDX-KEY.
16 NOMBRE PIC X(30).
11 DIREC PIC X(30).
11 PROV PIC X(1).
11 LOCAL PIC S9(2).
11 COD-P PIC X(4).
11 FEALTA PIC 9(6).
11 IVA PIC S9(2)V9(2).
01 MSG3 pic X(60) value
"Error: imposible asociar registro.".
```

Manejo de transacciones

`CALL "BeginTransaction"`

Inicia una transacción. Las transacciones consisten en operaciones que pueden ser hechas todas, o ninguna, es decir, puede ser deshecha. (Para más información, véanse partes anteriores.)

```
CALL "EndTransaction"
```

Termina (confirma) una transacción, llevándola a cabo.

```
CALL "RollBack"
```

Cancela (aborta) la transacción corriente (es decir, la última iniciada).

```
CALL "TransOk" USING [bool-var]
```

Verifica el estado de una transacción.

Parámetro:

- [bool-var] : variable string, en donde se guarda elresultado de la n corriente, pasa por referencia}

Pregunta si la transacción corriente (es decir, la última declarada) se lleva a cabo sin error. Retorna para esto "T" (verdadero) o "F" (falso) en la variable [bool-var].

Manejo de cursores

Las funciones de manejo de cursores consisten en crear un cursor, eliminar un cursor, definir registros límite de un cursor y leer registro próximo y anterior según un cursor. Una vez creado un cursor, éste queda establecido como corriente.

```
CALL "CreateCursor" USING [sch-id] record [index-name]
[lock-mode] [cur-id]
```

Crea un cursor, para el esquema [sch-id], según las características especificadas.

Parámetros:

[sch-id] : identificador de esquema obtenido con OpenSchema, pasa por valor

- [record] : registro (buffer), pasa por referencia
- [index-name] : nombre del índice asociado, pasa por valor
- [lock-mode] : modo de lock, pasa por valor
- [cur-id] : identificador del cursor, pasa por referencia

Los valores default son:

- [sch-id] : esquema corriente
- [index-name] : primary key
- [lock-mode] : NOT-LOCK

En caso de error al crear el cursor, se devuelve en [cur-id] el valor ERR. Si no se especifica esta variable, entonces en caso de error el programa aborta.

`CALL "DeleteCursor" USING [cur-id]`

Elimina el cursor [cur-id].

Parámetro:

[cur-id] : identificador del cursor, pasa por valor

Por default, se usa el cursor corriente.

`CALL "SetCursorFrom" USING [cur-id]`

Define el primer registro asociado al cursor [cur-id].

Parámetro:

- [cur-id] : identificador del cursor, pasa por valor

Por default, se usa el cursor corriente.

`CALL "SetCursorTo" USING [cur-id]`

Define el último registro asociado al cursor [cur-id].

Parámetro:

- [cur-id] : identificador del cursor, pasa por valor

Por default, se usa el cursor corriente.

`CALL "MoveCursorFirst" USING [cur-id]`

Posiciona el cursor [cur-id] al comienzo del conjunto de registros asociados a él.

Parámetro:

- [cur-id] : identificador del cursor, pasa por valor

Por default, se usa el cursor corriente.

`CALL "MoveCursorLast" USING [cur-id]`

Posiciona el cursor [cur-id]al final del conjunto de registros asociados a él.

Parámetro:

- [cur-id] : identificador del cursor, pasa por valor

Por default, se usa el cursor corriente.

```
CALL "FetchCursor" USING [cur-id] [ret-val]
```

Lee el registro siguiente según un cursor.

Parámetros:

- [cur-id] : identificador del cursor, pasa por valor
- [ret-val] : valor de retorno, pasa por referencia

Lee el próximo registro a partir del cursor [cur-id]. Por default, se usa el cursor corriente. En [ret-val] se devuelve un valor según el resultado de la operación: OK, si no hubo problemas; ERR, si hubo un error, o IO_LOCKED, si no pudo leerlo por lock. Estas constantes están definidas en idea.cpy .

El valor default para [sch-id] es el esquema corriente.

```
CALL "FetchCursorPrev" USING [cur-id] [ret-val]
```

Lee el registro previo según un cursor.

Parámetros:

- [cur-id] : identificador del cursor, pasa por valor
- [ret-val] : valor de retorno, pasa por referencia

Lee el registro anterior según el cursor [cur-id]. Por default, se asume el cursor corriente. Si se especifica [ret-val], se devuelve en dicha variable un valor según el resultado de la operación: OK, si no hubo problemas; ERR, si hubo un error, o IO_LOCKED, si no pudo leerlo por haber un lock. Estas constantes están definidas en idea.cpy .

El valor default para [sch-id] es el esquema corriente.

```
CALL "SetCursorFlds" USING [cur-id] "field-name" [ "field-name" ... ]
```

Define los campos asociados a un cursor determinado.

Parámetros:

- [cur-id] : identificador del cursor, pasa por valor

- "field-name" ... : cadenas, nombres de los campos del cursor, pasan porvalor

Esta función define cuáles son campos que estarán ligados a un cursor creado previamente. Para más información acerca de cursores, ver la sección correspondiente del manual.

El valor default para [cur-id] es el cursor corriente.

Manejo de Locks

`CALL "LockSchema" USING [sch-id] [lock-mode] [ret-val]`

Pone un lock a un esquema.

Parámetros:

- [sch-id] : identificador de esquema, pasa por valor
- [lock-mode] : modo de lock, pasa por valor; sólo puede ser WAIT-LOCK o TEST-LOCK}
- [ret-val] : valor de retorno, pasa por referencia

Los valores default son:

- [sch-id] : esquema corriente
- [lock-mode] : WAIT-LOCK

`CALL "LockTable" USING [sch-id] [table-name] [lock-mode] [ret-val]`

Pone un lock a una tabla.

Parámetros:

- [sch-id] : identificador de esquema, pasa por valor
- [table-name] : cadena, nombre de la tabla, pasa por valor
- [lock-mode] : modo de lock, pasa por valor; sólo puede serWAIT-LOCK o TEST-LOCK
- [ret-val] : valor de retorno, pasa por referencia

Los valores default son:

- [sch-id] : esquema corriente
- [lock-mode] : WAIT-LOCK

`CALL "FreeSchema" USING [sch-id]`

Elimina el lock de un esquema.

Parámetro:

- [sch-id] : identificador del esquema, pasa por valor

El valor default de [sch-id] es el esquema corriente.

`CALL "FreeTable" USING [sch-id] [table-name]`

Elimina el lock de una tabla.

Parámetros:

- [sch-id] : identificador del esquema, pasa por valor
- [table-name] : cadena, nombre de la tabla, pasa por valor

El valor default de [sch-id] es el esquema corriente.

`CALL "FreeRecord" USING [sch-id] [record] [index-name] [find-mode]`

Elimina el lock de un registro.

Parámetros:

- [sch-id] : identificador del esquema, pasa por valor
- [record] : registro (buffer), pasa por valor
- [index-name] : cadena, nombre del índice a usar, pasa por valor
- [find-mode] es el modo de búsqueda (ver partes anteriores), pasa por valor

Los valores default son:

- [sch-id] : esquema corriente
- [find-mode] : THIS-KEY
- [index-name] : primary key

Tipos de datos usados

Como referencia general, los tipos de datos específicos para los parámetros descriptos en la sección de sintaxis general son los siguientes:

```
[schema-name] : string
[sch-id] : S9(8) binary
[record] : con el formato (y en el orden) generado por DGEN
(p.ej. demo.cpy)
[index-name] : string
[find-mode] : valor según idea.cpy
[lock-mode] : valor según idea.cpy
[cur-id] : S9(8) binary
[ret-val] : S9, valor según idea.cpy
```

Equivalencia entre los tipos de la base de datos y tipos en Cobol

Los tipos en Cobol que representan los tipos usuales de datos de la base de

datos deben asumirse según las siguientes equivalencias:

date	9(6)
time	9(8)
bool	X, valiendo "TÓ (verdadero), "FÓ (falso) o " " (nulo)
num(a,b)	S9(a-b)V9(b)
char(n)	X(n)

Table 2: CFIX COBOL

donde tipo es uno de los 5 tipos básicos, es decir, date, time, bool, num() o char(), y tipocobol es su equivalencia en COBOL según esta tabla. Un ejemplo de esta última equivalencia es el siguiente:

CFIX	COBOL
[n]num(a,b)	S9(a-b)V9(b) OCCURS n TIMES

declarando un array de n valores numéricos de a dígitos incluyendo b dígitos a la derecha de la coma decimal.

Uso del DGEN

El utilitario DGEN se puede usar ahora para generar archivos de encabezado en formato correspondiente a definiciones Cobol. La extensión por default es .cpy . Las siguientes son maneras de generarlos:

- Con DGEN, usar los switches -c o -c2 . Si se usa -c, se generará un registro por cada tabla de esquema usado, juntando todo en un archivo de n (.cpy).

Ejemplo:

```
01 ITEMS.
06 TAB-ID PIC 9(5) BINARY VALUE 256.
06 ITEMS-DATA.
11 ITEMNO-KEY.
16 ITEMNO PIC S9(4) VALUE ZEROES.
11 DSC PIC X(45) VALUE SPACES.
11 COST PIC S9(12)V9(2) VALUE ZEROES.
11 PRVTA PIC S9(12)V9(2) VALUE ZEROES.
11 CANTID PIC S9(4) VALUE ZEROES.
01 CLIENTES.
06 TAB-ID PIC 9(5) BINARY VALUE 512.
06 CLIENTES-DATA.
11 CLIENO-KEY.
16 CLIENO PIC S9(4) VALUE ZEROES.
11 NOMIDX-KEY.
16 NOMBRE PIC X(30) VALUE SPACES.
11 DIREC PIC X(30) VALUE SPACES.
11 PROV PIC X(1) VALUE SPACES.
11 LOCAL PIC S9(2) VALUE ZEROES.
11 COD-P PIC X(4) VALUE SPACES.
11 FEALTA PIC 9(6) VALUE ZEROES.
11 IVA PIC S9(2)V9(2) VALUE ZEROES.
01 ORDENES.
06 TAB-ID PIC 9(5) BINARY VALUE 768.
06 ORDENES-DATA.
11 ORDERNO-KEY.
16 ORDERNO PIC S9(4) VALUE ZEROES.
11 FECHA PIC 9(6) VALUE ZEROES.
11 CLIENO PIC S9(4) VALUE ZEROES.
11 VALOR PIC S9(12)V9(2) VALUE ZEROES.
11 MIVA PIC S9(12)V9(2) VALUE ZEROES.
11 MENTR PIC S9(1) VALUE ZEROES.
11 DIREC PIC X(50) VALUE SPACES.
01 MOVIM.
06 TAB-ID PIC 9(5) BINARY VALUE 1024.
06 MOVIM-DATA.
11 ORDERNO-KEY.
16 ORDERNO PIC S9(4) VALUE ZEROES.
16 ITEMNO PIC S9(4) VALUE ZEROES.
11 CANTID PIC S9(4) VALUE ZEROES.
11 PRVTA PIC S9(12)V9(2) VALUE ZEROES.
01 PROVIN.
06 TAB-ID PIC 9(5) BINARY VALUE 1280.
06 PROVIN-DATA.
11 CODPRO-KEY.
16 CODPRO PIC X(1) VALUE SPACES.
11 NOMBRE PIC X(20) VALUE SPACES.
01 LOCAL.
06 TAB-ID PIC 9(5) BINARY VALUE 1536.
06 LOCAL-DATA.
11 CODPRO-KEY.
16 CODPRO PIC X(1) VALUE SPACES.
16 CODLOC PIC S9(2) VALUE ZEROES.
11 NOMBRE PIC X(20) VALUE SPACES.
```

- Si se usa -c2, se hará lo mismo, pero agrupando todas las tablas en un registro con el nombre del esquema. Esto es útil si por ejemplo se trabaja con s de un esquema en el

mismo programa, y tablas distintas de distintos esquemas puedan tener un mismo nombre. Lo que se con el ejemplo anterior es algo como:

```
01 DEMO
06 ITEMS.
...
06 CLIENTES.
...
06 ORDENES.
...
06 MOVIM.
...
06 PROVIN.
...
06 LOCAL.
...
```

Usar la cláusula *language "Cobol"* en la definición del esquema correspondiente (.sc). El siguiente ejemplo genera encabezados en Cobol y en C:

```
create schema demo descr "Demostration Schema"
language "Cobol", language "C";
table items ...
table clientes ...
```

Observación

Notar que en el encabezado generado se enumeran las claves de cada tabla del siguiente modo:

```
06 CLIENTES-DATA.
11 CLIENO-KEY.
16 CLIENO PIC S9(4) VALUE ZEROES.
11 NOMIDX-KEY.
16 NOMBRE PIC X(30) VALUE SPACES.
```

vale decir, se genera un nivel más de campos para cada clave, dándole a éstas un nombre identificatorio formado por el nombre del índice especificado seguido de -KEY. Cuando se trata de la clave primaria, como no se especifica ningún nombre de índice, el campo toma el nombre del primer campo de la clave.

Esto último ocurre en el caso de que:

- o bien los campos de la clave de la tabla estén definidos en forma consecutiva, y éstos no se superponen con los campos de otra clave definida previamente,
- o bien los campos de la clave de la tabla están incluidos dentro delos campos de otra clave

En estos casos, se agrupa los campos de la clave dentro de otro con el nombre antedicho.

Apéndice B

Vocabulario

A

Almacenar

esquema 46

B

Borrar

esquema 46

índice 47

tabla 46

C

Campos

renombrar 48

tipos de datos 37

Campos agrupados 97

atributos 98

Campos de formularios

atributos

descr 78

display only 78

display only when 78

in table 85

is 78

not null 78

on help in table 90

skip 78

skip when 79

Campos de tablas

arreglos 45

Atributos 38

between 40

check 41

check digit 38

default 38

description 38

in (<valor>) 40

in table 41, 43

mask 38

not null 38

primary key 38

Campos múltiples 95

atributos 96

condiciones pre y post-campo 112

Cfix funciones

DB

AddIndField 209

AddKey 216

AddRecord 252

AddRecordTest 252

AddRecordTS 252

AddTabField 209

AlFld 212

AlInd 212

BeginTransaction 261

BuildIndex 216

CloseAllSchemas 218

CloseSchema 218

CompleteIndex 216

CopyAliasRecord 218

CopyFld 218

CopyTable 219

CountCursor 231

CreateAlias 212

CreateCursor 224

CreateIndex 220

CreateSchema 220

CreateTable 220

CTimeStamp 261

CurrentSchema 233

DbCheckPoint 227

DbCheckSum 227

FreeRecord 239

FreeSchema 239

FreeTable 239

GetBinaryToFile 215

GetBinaryToMem 215

GetDspFld 241

GetFld 241

GetFldDesc 241

GetFldLen 241

GetFldName 241

GetFldNDec 241

GetFldType 241

GetIndFlds 242

GetIndices 243

GetIndName 242

GetKeyFld 241

GetRecord 244

GetRelDepth 254

GetRelErrFlag 254

GetSchDescr 243

GetSchemas 246

GetTabDescr 243

GetTabFlds 243

GetTables 243

GetTextToFile 215

GetTextToMem 215

IFld 241

InDescr 263

IndNFields 233

InitRecord 257

IsNull 263

KeyDbField 247

LenDspFld 247

LFld 241

LockSchema 248

LockTable 248

MoveCursorFirst 249

MoveCursorLast 249

MTimeStamp 261

NFld 241

OpenSchema 250

OpenSchemaVersion 264

PopRecord 251

PushRecord 251

PutRecord 252

RecordStackSize 251

SchemaVersion 264

SetCursorFlds 224

SetCursorFrom 224

SetCursorFromFld 224

SetCursorTo 224

SetCursorToFld 224

SetCursorVFlds 224

SetDbHandler 258

SetDFld 256

SetFFld 256

SetFld 256

SetIFld 256

FreeMem 305

FreeMemWi 305

FtoI 301

FtoL 301

FullUserName 302

GetAddPerm 310

GetDelPerm 310

GetPWEntry 305

GetUid 306

GetUpdatePerm 310

ItoF 301

ItoL 301

Locate 307

LtoF 301

LtoI 301

ModuleExists 309

ModuleIsInstalled 309

OnStop 313

OpenPrinter 309

ProcArgs 310

ProcInitDate 310

ProcInitTime 310

ProcNArgs 310

ProcPid 310

ProcSaveMem 310

ProcSig 310

ProcTty 310

ProcTtyName 310

ProcUserGid 310

NumSubF 316

NumSubL 316

NumToF 319

NumToL 319

NumToStr 319

RpSetNFld 318

SetNFld 318

StrToNum 319

RP

BeginReport 323

CloseAllReports 324

CloseOutput 329

CloseReport 324

ColumnWidth 329

CurrColOutput 329

DbToRp 325

DoReport 325

EndReport 326

NColsOutput 329

OpenDelimOutput 329

OpenFmtOutput 329

OpenReport 327

OpenRpOutput 329

OpenTermOutput 329

OutputColumn 329

PageWithOutput 329

RpCheckSum 335

RpClearZone 336

RpDrawZ 336

StrToI 343

StrToL 343

StrToLower 347

StrToUpper 347

StrTxt 350

TM

AddMonth 352

Day 360

DayName 353

DMYToD 354

DToDMY 354

DToStr 362

FirstMonthDay 355

GetDateFmt 356

HalfMonth84 356

Hour 357

IsHoliday 358

LastMonthDay 355

Month 360

MonthDiff 359

MonthName 360

SetDateFmt 356

StrNXToD 362

StrNXToT 364

StrToD 362

StrToT 364

StrToTS 365

Today 366

TSToStr 365

Cobol

Consistencia

D

E

M

P

R

S

T

U

V

Variables de ambientes

dbase 56

W

wcmd 182

Parte IV

•

IDEAFIX

Base de Datos

Sección 1 Essentia

Sección 2 Drivers

Sección 3 Apéndices

Capítulo 1

Presentación

InterSoft S.A. pone a disposición de sus clientes el server de Base de Datos InterSoft Essentia, desarrollado sobre una nueva tecnología de diseño: la tecnología RISE.

Mientras otras Bases de Datos compiten en función de tamaño y complejidad, Essentia pone en primer plano la utilidad, la sencillez y la potencia.

La Tecnología

RISE (Reduced Instruction Set Engine - Motor con Conjunto de Instrucciones Reducido) es un concepto análogo al de la tecnología RISC para microprocesadores. La idea básica es tener un proceso (el motor) que provee una cantidad de servicios (de allí el nombre de server - servidor) a otros procesos llamados clientes. Algunos ejemplos de servers son los manejadores de ventanas, los servers de comunicaciones, etc.

La comunicación entre el server y los clientes se realiza por medio de una interface perfectamente establecida, pero de dimensiones reducidas. Esta pequeña cantidad de mensajes, permite, sin embargo, armar funciones para realizar tareas muy complejas, sin perder simplicidad o velocidad.

El Server de Base de Datos

ESSENTIA, por su diseño, puede implementar distintos modelos de base de datos; en la actualidad el Modelo Relacional y el Modelo Orientado a Objetos. Al mismo tiempo, esta tecnología permite mantener (y aumentar) dos características fundamentales: eficiencia y

seguridad.

La tecnología RISE elimina la complejidad del server para permitir que los distintos clientes extiendan las funciones provistas por el mismo. Estos procesos clientes pueden ser implementados por el usuario o bien pueden ser provistos por InterSoft. En particular, este manual trata de la implementación de IDEAFIX (la base de datos relacional de InterSoft Argentina S.A.) que trabaja sobre el server ESSENTIA.

La comunicación entre ESSENTIA e IDEAFIX es totalmente transparente para el usuario y para el programador, de forma tal que un programa funciona exactamente igual cuando trabaja con el server ESSENTIA que cuando lo hace con el propio manejador de IDEAFIX.

Los principales servicios que ofrece ESSENTIA son los referidos a la creación, recuperación y borrado de registros. Sin embargo, el hecho de que los servicios sean pocos no debe interpretarse de forma errónea. El server realiza (por cuenta propia o a pedido de un cliente) una enorme cantidad de funciones administrativas y de resguardo de la información.

Entre éstas, las más importantes son :

- *Chequeo de consistencia automático*: cada vez que arranca el server, se verifica el estado de las transacciones. Así se asegura que el estado de la base de datos es consistente hasta la última transacción completa. Para lograr esto, ESSENTIA utiliza un mecanismo combinado de checkpoints y journals.
- *Versiones de la base de datos* : este mecanismo -inexistente en los demás DBMS tradicionales- permite mantener el estado de cualquier base de datos a una fecha dada. Así, un proceso puede consultar la base de datos al 31/12/95 mientras otro ingresa los datos actuales, sin por esto duplicar el tamaño y la información.
- *Backup incremental* : cuando se desea hacer una copia del contenido de la base de datos, ESSENTIA ofrece la posibilidad de ir haciendo backups incrementales, es decir, ir guardando las modificaciones realizadas desde el último backup. Esta tarea es llevada a cabo por un proceso cliente más, no por el server, eliminando no sólo los antiguos backups completos de datos, sino también permitiendo que el mismo se realice mientras se está trabajando con la Base de Datos.
- *Mirroring* : se puede mantener una imagen instantánea de la base de datos, en forma paralela a la base de datos real. Así, si un server cae, otro server puede continuar el trabajo sobre la imagen de la base de datos.
- *Shadowing* : es una técnica que permite manejar con seguridad la actualización de la información de la base de datos: primero se crea una copia de la información que se va a modificar (shadow), se trabaja sobre la copia y finalmente, cuando es seguro, se convierte la copia en el objeto definitivo.

La arquitectura "client/server" de ESSENTIA permite que los procesos se ejecuten en distintas máquinas. Por ejemplo, el proceso server puede estar en Buenos Aires, mientras que el cliente que realiza la consulta puede estar en Río Negro, comunicándose a través de una

línea telefónica.

La única condición es que las distintas máquinas formen parte de una red con alguno de los protocolos soportados por ESSENTIA. Además -vale la pena insistir-, la ubicación de los procesos es totalmente transparente para el programador.

Por otra parte, Essentia optimiza el manejo de locks y colisiones, haciendo uso de su propio esquema de locks, soportando locks de lectura, escritura o modificación a nivel de registro, tabla y esquema.

En los últimos años la tecnología client/server ha sido aplicada a los más diversos problemas en computación. En este esquema, se parte de la base que en toda aplicación compleja hay una cantidad de tareas específicas que pueden ser separadas del proceso principal. De esta manera, uno o varios procesos independientes (los servers) se ocupan de estas tareas, simplificando la estructura original de la aplicación.

Los clientes ejecutan diferentes aplicaciones que acceden a la información de la base de datos e interactúan con el usuario. Cada server maneja las funciones para atender pedidos de aplicaciones específicas y de cualquier otro proceso que conozca el "lenguaje" o interface necesaria para comunicarse.

De esta forma, un server no queda ligado a una aplicación concreta, sino a un recurso o actividad específica.

Algunos ejemplos comunes de servers son:

- *Un manejador de ventanas* : La función de este tipo de servers es administrar el uso de la pantalla para una cierta terminal, ocupándose de crear y mover ventanas, mostrar textos y gráficos, y leer información del teclado. La ventaja de este tipo de enfoque es que las aplicaciones ya no deben preocuparse de las características físicas de la terminal, porque se manejan con un nivel de abstracción más alto.
- *Un server de comunicaciones* : Este tipo de servers administra todas las funciones de comunicación entre procesos o máquinas. Estas funciones incluyen la de establecer el vínculo, enviar y recibir mensajes, "multiplexar" y "demultiplexar" el uso de un recurso compartido, etc. De esta manera, se consigue compartir eficientemente un recurso y a la vez, ocultar los detalles de la línea de comunicación a los clientes.
- *Un server de base de datos* : En este caso, el server administra la información contenida en una base de datos, atendiendo pedidos de grabación y consulta de datos. Si el server es el único proceso que accede a la información, es posible implementar un sistema de bloqueo eficiente e independiente del sistema operativo.

Otra ventaja del modelo utilizado por ESSENTIA es que permite separar las tareas entre el server y los clientes. Así resulta más natural la sustitución de la comunicación entre procesos que corren en un mismo equipo por la comunicación entre procesos que corren en distintas máquinas.

Por otra parte, la tecnología RISE permite que la interface entre el server y los clientes sea

mínima. El server ofrece un conjunto de servicios seguros y eficientes, en base a los cuales los clientes pueden realizar tareas más complejas. Esto no supone una limitación en el manejo de los datos, porque el server provee un nivel de abstracción y una cantidad de servicios suficientes para implementar en forma natural cualquier operación deseada. Como resultado, ESSENTIA ofrece un alto grado de concurrencia, integridad de datos y performance a sus aplicaciones clientes.

En una red, los clientes y el server de la base de datos están ubicados en diferentes computadoras. Este tipo de configuración permite una división de tareas entre las computadoras clientes y las computadoras servers. Los servers requieren de memoria y espacio en disco para administrar la base de datos. Los clientes por otro lado sólo necesitan memoria para ejecutar la aplicación. Esta separación de procesamiento entre diferentes computadoras es llamado procesamiento distribuido.

Los beneficios que la arquitectura client/server ofrece en un entorno de procesamiento distribuido como el implementado en ESSENTIA incluyen :

- Bajo costo de las máquinas donde se ejecutan los procesos clientes, las cuales pueden acceder a los datos remotos de un server.
- Los clientes pueden optimizarse para la presentación de los datos Ej., gráficos o mouse), en cambio los hosts servers pueden ser optimizados para el procesamiento y almacenamiento de datos (Ej., cantidad de memoria RAM y espacio en disco.).
- Las aplicaciones clientes procesan y envían pedidos al server. Una vez recibidos estos pedidos, los mismos son procesados. El server retorna sólo los datos pedidos. Las aplicaciones clientes, por lo tanto, no procesan más información que la necesaria.

Capítulo 2

Instalación

Para instalar y trabajar con Essentia es necesario disponer de las siguientes facilidades :

- **El server, Máquina con sistema operativo POSIX compatible**, que permita manejar recursos de IPC (semáforos y colas de mensajes). Si se trabaja en red, Runtime y Development de TCP/IP ya que es necesario disponer del server "inetd", para la correspondiente conexión.
- **El cliente**, **Máquina con sistema operativo POSIX compatible, Windows o DOS** (estos dos últimos requieren, además, un servicio de comunicación TCP/IP).
- **La red, Cualquier red que trabaje con protocolo TCP/IP**.
- **IDEAFIX**, en su versión correspondiente.

Cada server es el único responsable del manejo de uno o más esquemas. Por lo tanto, el primer paso es decidir qué esquemas van a ser atendidos por cada server. Es posible que esta distribución involucre a más de un nodo de una red.

Variables de Ambiente

Se deben definir cuatro variables de ambiente:

`Essentia`

Es el directorio donde se guardan los distintos archivos de configuración de los servers. Todos los usuarios que trabajan sobre un mismo nodo deben tener el mismo valor en esta variable.

Ejemplo:

```
Essentia = $IDEAFIX/essentia
```

`DATADIR`

Es el directorio donde se guardan algunos archivos de datos usados por los ejecutables de Essentia, como por ejemplo los archivos que contienen los mensajes de error. El valor usual de esta variable es el siguiente:

```
DATADIR = $IDEAFIX/data
```

SERVERS

Es la lista de los nombres de los servers a los que pueden acceder los usuarios, separados por ";". Cada uno de estos servers debe tener un archivo de configuración en el directorio $Essentia/servers, con el mismo nombre que el server y la extensión ".sv" o ".rsv".

Ejemplo:

```
SERVERS = "aurus;denarius"
```

PATH

Esta variable de UNIX indica la lista de directorios en los que el sistema operativo busca los programas ejecutables. Debe incluir los directorios `$IDEAFIX/rbin` y `$IDEAFIX/bin`, en ese orden, porque en el primero se guardan las versiones nuevas de algunos utilitarios de IDEAFIX, que fueron modificadas para trabajar con Essentia, y en el segundo los utilitarios de IDEAFIX y Essentia.

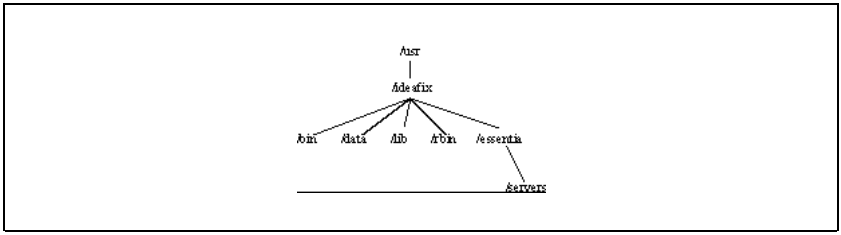
Ejemplo:

```
PATH=":$IDEAFIX/rbin:$IDEAFIX/bin:/usr/local/bin:/bin"
```

Configuración de la Instalación

Existen diferentes criterios que pueden ser utilizados para instalar el producto Essentia. Aquí vamos a referirnos a un formato de instalación standard.

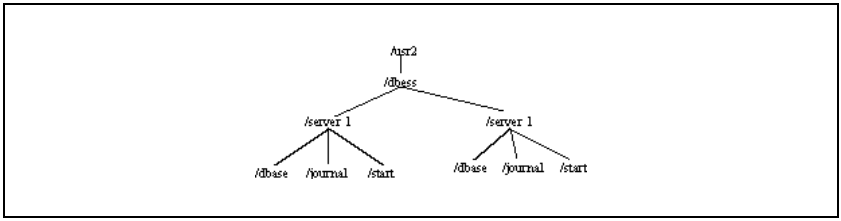
El valor por omisión en una instalación de Essentia es el mismo que IDEAFIX (`/usr/ideafix`). Por lo tanto si tuviéramos una vista de la estructura de la instalación la misma sería :



Para la configuración de los datos pueden emplearse diferentes criterios, cada uno de acuerdo a la distribución de los sistemas de archivos del host. Aquí vamos a especificar un ejemplo de posible distribución de la base de datos funcionando bajo el esquema de Essentia.

La distribución empleada es muy sencilla y fácil de implementar. Para ello en un sistema de archivos, en este caso "`usr2`", creamos un directorio **dbess** donde vamos a colocar los diferentes esquemas de nuestra aplicación. Dentro del mismo se hace una segunda división discriminada por "server" con lo cual cada directorio deberá poseer el mismo nombre del server para identificarlo con facilidad.

Una vez creados los directorios para cada server podemos configurarlos, para lo que creamos dentro de cada uno los siguientes directorios :



- `dbase` : Guarda la información de la Base de Datos.
- `journal` : Guarda el journal de transacciones.
- `start` : Guarda archivos de log de startup y shutdown.

***Nota** : Nunca se debe cambiar el reloj de hardware del equipo hacia atrás, ya que Essentia no arrancará cuando detecte un tiempo menor al último tiempo guardado en la base de datos. Además, si se cambia la hora mientras un server Essentia está corriendo, se generará inconsistencia en los datos*

Servers Locales y Remotos

Un server local es aquel que reside en el mismo nodo de la red; un server remoto corre en otro nodo. Para que un nodo de la red reconozca un server determinado, debe haber un archivo con ese nombre en el directorio `$Essentia/servers`. Si el server es local, el nombre del archivo debe terminar en ".sv"'; si el server es remoto, debe terminar en ".rsv".

Nota: Por ningún motivo se debe modificar manualmente el archivo de configuración de un server mientras éste está corriendo, puesto que es posible que el mismo no se corresponda exactamente con el que tiene en memoria el server Essentia que está ejecutando.

El archivo de configuración de un server remoto (un archivo ".rsv") tiene la siguiente

estructura:

```
Host < nombre del nodo del server o su dirección IP >
Schemas < lista de esquemas >
```

Las sentencias tienen los siguientes significados:

Host : Nombre del nodo de la red en el que corre el server.

Schemas : Lista de esquemas que maneja el server (optativamente, con sus respectivos directorios). El formato es el siguiente:

```
esquema,esquema, ...
```

Esta línea no debe modificarse a mano en la máquina local, ya que cuando se crea un esquema con el utilitario **dgen**, se actualiza automáticamente en el próximo checkpoint. Si se usa la opción "-k", el checkpoint se realiza de inmediato, y el esquema queda disponible. Sin embargo, en el resto de las máquinas que contengan referencia a ese server (con un archivo .rsv) sí se deberá modificar a mano, con una sola excepción más: si el **dgen** se ejecutó remoto, el utilitario se encargará de actualizar también los .rsv de la máquina desde donde el **dgen** se ejecutó, si los hubiera.

Nota: Si en una instalación se encuentran archivos .sv y .rsv para un mismo server, Essentia leerá el primero.

Cada archivo de configuración de un server local (un archivo ".sv") tiene la siguiente estructura:

```
ServerId < número de identificación del server >
Schemas < lista de esquemas >
NextSchemaId < próximo identificador >
CacheSize < tamaño >
CkPointFreq < frecuencia en segundos >
CkPointLowBound < número de segundos >
CkPointUpBound < número de segundos >
CorruptionTolerant < modificador >
Dbase < directorio valor >
HandleVersions < modificador >
HKeepingMsgBound < frecuencia >
Locks < cantidad >
LocksPerClient < cantidad >
LockUnifyBound < cantidad >
LockUnifyPercent < porcentaje >
LockWait < tiempo >
PendLockNotifTime < tiempo >
Mirroring < modificador >
Backup < modificador >
Notifies < cantidad >
RunLogFile < archivo >
StateFile < archivo >
JournalBuffer < tamaño >
MatureTime < tiempo >
MaxAttachsPerClient < cantidad de relaciones >
JournalDir < directorio >
JournalFlushFreq < frecuencia >
```

```
TransTimeOut < tiempo >
ObjectRepository < directorio >
DiskConfiguration < archivo >
```

Las sentencias tienen los siguientes significados:

ServerId

Número de identificación del server. Debe ser único en cada nodo de la red. No se debe modificar esta línea, puesto que este número también identifica los datos del server.

Nota: La repetición de este número identificador en distintos servers activos producirá problemas de ejecución y datos en las aplicaciones.

Schemas

Lista de esquemas que maneja el server, con sus respectivos directorios. El formato es el siguiente:

```
esquema:directorio:id_esquema,esquema:directorio:id_esquema,
...
```

No se debe modificar esta línea a mano. Cuando se crea un esquema con el utilitario **dgen**, esta línea se actualiza automáticamente en el próximo checkpoint. Si se usa la opción "-k", el checkpoint se realiza de inmediato, y el esquema ya queda disponible.

NextSchemaId

Próximo número identificador que el server va a asignar a los esquemas que va generando. Esta línea no existe al arrancar el server por primera vez. La misma es actualizada automáticamente en conjunto con el parámetro anterior.

CacheSize

Tamaño del caché de tablas e índices, en Kbytes. El mismo es utilizado por el server para optimizar su funcionamiento y acceso a índices y datos. El valor por omisión es 256, pero es conveniente que este valor sea lo más alto posible en relación con la memoria RAM de la instalación. El valor mínimo posible es 64 Kbytes.

Nota: La asignación desproporcionada de memoria al server o la utilización de varios servers concurrentemente con demasiado tamaño de cache, puede generar swap *del equipo con la consecuente pérdida de performance.*

CkPointFreq

Cantidad de segundos que transcurrirán entre los distintos intentos de hacer un checkpoint. El valor por omisión es 3600 segundos, es decir, una hora.

CkPointLowBound

Cantidad máxima de segundos que se puede adelantar un checkpoint para aprovechar la inexistencia de transacciones activas y ejecutar un checkpoint a nivel de transacción (ver Capítulo 3). El valor por omisión es 300 segundos, es decir, 5 minutos.

CkPointUpBound

Lapso (en segundos) tras el cual se dejará de intentar un checkpoint a nivel de transacción y se ejecutará uno a nivel de cache. El valor por omisión es 15.

CorruptionTolerant

Si este modificador vale 0, cuando un cliente es abortado y el journal está deshabilitado, se marca el estado del server como "corrupto" y no se aceptan más checkpoints (excepto con la opción "-f ").

Si el modificador vale 1, ese tipo de situaciones son ignoradas. El valor por omisión es 1.

Dbase

Directorio donde se grabarán los esquemas que se creen en este server.

Nota: los esquemas creados por Essentia no son binario- compatibles con los esquemas creados por IDEAFIX cuando no está corriendo con el server.

HandleVersions

Este modificador con valor 1, habilita la posibilidad de pedidos de nuevas versiones. Si es 0, no se aceptan pedidos de generación de versiones. El valor por omisión es 0.

Nota: La cantidad máxima de versiones posibles a generar en una base de datos es 122.

HKeepingMsgBound

Frecuencia (medida en cantidad de mensajes) con la que se realizan tareas administrativas y de limpieza del server. (por ejemplo, verificación del estado de los clientes, verificación de la necesidad de efectuar checkpoints, etc.). El valor por omisión es 100. Es necesario aclarar que el server siempre realiza estas tareas durante el tiempo de ocio. Este parámetro permite pedirle que haga lo antedicho aún cuando no disponga de estos tiempos.

Locks

Cantidad máxima de locks admitidos por el server. El valor por omisión es 1024.

***Nota** : establecer un lock sobre una tabla cuenta como un sólo lock, no como una cantidad de locks igual a la cantidad de registros de la tabla.*

LocksPerClient

Cantidad máxima de locks disponibles -limitados por la variable anterior- para un mismo

cliente. El valor por omisión es 256.

LockUnifyBound

Cantidad máxima de locks de un mismo cliente sobre una misma tabla después de la cual se intentará bloquear la tabla entera. El valor por omisión es 200.

LockUnifyPercent

Este parámetro funciona en forma similar a *LockUnifyBound* pero en lugar de indicar una cantidad fija de registros (como este último) se indica un porcentaje de la cantidad de registros que posee la tabla. Su valor en 0 (cero) indica que esta facilidad está deshabilitada.

En el caso en el que el cliente supere su cantidad de locks asignados *(LocksPerClient)* Essentia tratará de unificar los locks sobre alguna tabla (convirtiendo todos los locks en un único lock de tabla) que cumpla con alguna de estas dos condiciones: *a)* la cantidad de locks supera al *LockUnifyBound* o, *b)* el porcentaje de registros lockeados en la tabla supera al número especificado en *LockUnifyPercent*.

LockWait

Cantidad de tiempo (expresada en segundos) que se debe esperar por un lock en una transacción que involucra más de un server. Si un proceso cliente estuviera esperando más del tiempo especificado en este parámetro la transacción abortará con un mensaje de *deadlock*. Essentia existen dos maneras de detectar *deadlocks*. Una de ellas es cuando un cliente tiene locks en un solo server, y la otra cuando los tiene en más de uno.

En el primer caso el server será capaz de detectar los *deadlocks* antes que se produzcan y abortar a uno de los clientes que provocan el ciclo (el de la transacción más reciente).

En el segundo de los casos, el server establece un tiempo máximo de espera (que es el especificado en este parámetro) sobre cada lock, y una vez transcurrido ese tiempo se asume que la causa de dicha espera es un *deadlock*. El valor por omisión es 120.

PendLockNotifTime

Cantidad de tiempo (expresada en segundos) que un server espera para avisarle a un cliente que su lock está esperando por causa de otro cliente.

El server es el encargado de avisarle al cliente cuándo el dueño del lock cambia (una vez transcurrido el tiempo '*PendLockNotifTime*'). El valor por omisión es 0, lo que implica que el server no avisará nunca a los clientes.

Notifies

Cantidad máxima de locks de notificación disponibles. Esta facilidad aún no está disponible para IDEAFIX. El valor por omisión es 1024.

Mirroring

La asignación de este modificador en 1, habilita la posibilidad de realizar mirroring de la base de datos. Si vale 0, se deshabilita la misma. La función del mismo es dejar que el server mantenga los archivos de journal para permitir realizar mirroring, con lo cual si esta capacidad no es utilizada se recomienda deshabilitar esta opción, puesto que el espacio en disco que los archivos de journal ocupan es proporcional a la información modificada. Una vez utilizados los archivos para realizar el mirroring (Ver Configuración y Uso de Mirroring) , estos son borrados. El valor por omisión es 0.

Backup

El valor de este modificador determina si se encuentra habilitada (valor 1) o no (valor 0) la posibilidad de realizar resguardos incrementales de la base de datos. Este modificador actúa de manera similar al del mirroring permitiendo que se mantengan los archivos de journal una vez realizados los checkpoints hasta tanto los utilice el proceso de backup. El mismo trabaja en conjunto con el mirroring para eliminar el archivo de journal una vez que ambos (mirroring y backup) lo han utilizado. El valor por omisión es 1.

RunLogFile

Archivo en el que se guarda la información de *debugging* de una corrida del server. Para la habilitación del mismo el server debe arrancarse con la opción "-w". Para su utilización una vez generada la información, se debe arrancar el server con la opción "-r". Este archivo contiene una lista de todas las operaciones que realiza el server. Al reproducirlo con la opción "-r", el server vuelve a ejecutar cada una de las operaciones realizadas anteriormente, para lo que se debe asegurar que el estado completo de la base de datos al momento de ejecutar "`essentia -w`" esté disponible. De esta forma, se puede reproducir un problema del server en forma muy sencilla. El valor por omisión es "`/tmp/rise.log`".

StateFile

Archivo en el que se guarda información sobre el estado del server (versiones, checkpoints, etc.). Este archivo es utilizado para recuperar un server luego de una caída, por lo tanto no debe estar en el directorio "`/tmp`", pues ese directorio generalmente se borra al levantar el sistema. El nombre de este archivo debe ser único en cada nodo de la red y no puede cambiarse una vez que fue creado. ***Advertencia*** *: Este archivo es tan importante como los datos del server*.

JournalDir

Directorio donde se graba el journal de transacciones del server. El nombre de los archivos de un journal comienzan con "rlog". Si no se usa esta sentencia, esta capacidad queda deshabilitada, lo cual implica que si un cliente aborta por alguna condición anormal, la base de datos quedará corrupta a nivel transaccional puesto que no tiene journal habilitado para realizar el *rollback*. Los archivos de journals se borran automáticamente si no se usan las

opciones de mirroring y backup incremental en el momento que se realiza un checkpoint.

Advertencia : Se debe asegurar que el sistema de archivos al que pertenece este directorio tenga suficiente espacio libre como para alojar los archivos de journal, ya que los mismos pueden tomar un tamaño importante dependiendo del intervalo con el cual se definieron los checkpoint y las condiciones de mirroring y backup. En caso de llenado del "file system" el server queda en estado de espera hasta que el administrador del sistema le otorgue espacio en el mismo.

JournalBuffer

Tamaño en Kbytes del buffer de memoria para el journal. Cuando este buffer se llena, se baja el contenido del mismo al archivo de journal activo. El valor por omisión (que también es el valor mínimo) es 32.

JournalFlushFreq

Frecuencia medida en segundos con la cual se realiza un flush del buffer del journal. Este parámetro trabaja en relación con el anterior (JournalBuffer) ya que el buffer es bajado al archivo de journal de acuerdo a cuál de las siguientes condiciones se completa primero: *a)* se llena el buffer o, *b)* transcurre el tiempo especificado en este parámetro.

El valor por omisión es 10.

Nota : No es recomendable utilizar un valor menor al por omisión puesto que se puede ver afectada la performance del server.

MatureTime

Este parámetro indica el tiempo (en segundos) que debe pasar entre la última modificación a una tabla o índice para que su *header* sea grabado en disco. Esto permite realizar la actualización en forma incremental de los datos que tienen que ser grabados, lo cual disminuye el tiempo que tarda el checkpoint siguiente.

El valor de este parámetro depende de las condiciones de uso (escritura) sobre una tabla; por ejemplo, si debido a los requerimientos sobre un determinado sistema los datos se ingresan en "ráfagas" (como en el caso de stock que se actualiza constantemente) cada día, sería adecuado dejar el valor por omisión (600 = 10 minutos) ya que sería común que cuando no se hayan ingresado datos en diez minutos signifique que se ha terminado con la actualización y que no se ingresarán más datos hasta el día siguiente.

MaxAttachsPerClient

Este parámetro indica el número máximo de objetos (esquemas, tablas y cursores) con cuales se permitirá a cada cliente relacionarse. Este parámetro se puede utilizar como un control para programas con errores que, por ejemplo, generen una cantidad demasiado grande de cursores, y por lo tanto consuman demasiada memoria.

Una buena forma para determinar el número que se utilizará es hacer correr los programas más exigentes (aquellos que utilicen gran número de objetos) y, utilizando el comando **dbps -f**, verificar la cantidad de objetos a los cuales se relacionan. A este número se le puede sumar un margen razonable (un 20-30% más del número más grande). Si un programa intenta relacionarse con más objetos que los permitidos, abortará con un mensaje aclaratorio.

El valor por omisión de esta variable es 0 (cero), lo que significa que no se ponen límites a la cantidad de objetos con los cuales un cliente puede relacionarse.

TransTimeOut

Cantidad de segundos que el server espera a un cliente con una transacción activa sin que envíe mensajes, tras lo cual Essentia interpretará que el cliente está inactivo y procederá a abortar la transacción. El valor por omisión es 0, lo cual implica que una vez iniciada una transacción la misma queda abierta hasta tanto el cliente la finalice, sin intervención alguna del server.

ObjectRepository

Este parámetro de configuración permite definir el directorio donde se almacenarán datos binarios y de texto masivos.

Es importante que aquellos servers que no utilicen datos binarios dejen este parámetro con valor nulo, ya que se degradará la performance innecesariamente.

DiskConfiguration

El archivo indicado en este parámetro guarda la configuración que será utilizada para las diferentes tablas (*packed*, *ranged* o *round- robin*). Para más información acerca de esto, Ver el capítulo 5, “Fragmentación de Tablas”.

Configuración de un Server Essentia Remoto

Como ya se ha mencionado, el server Essentia puede ser ejecutado en múltiples computadoras o nodos de la red. En algunos casos la información de la base de datos de un server Essentia está relacionada con otra base de datos almacenada en otros servers. Para que diferentes clientes puedan comunicarse con Essentia a través de una red, es necesario configurar algunos archivos del sistema operativo para lograr tal fin.

El *shadow* es el proceso encargado dentro del entorno de Essentia de atender y controlar las comunicaciones provenientes de clientes remotos enviando y recibiendo los datos a través de una red. El uso del shadow permite a cualquier aplicación cliente independizarse de la comunicación a través de una red si la misma existiera. Por lo tanto, es necesario incluir este programa para que el mismo tome control en el momento que se ejecuta un pedido a un server ubicado en un host remoto de la red.

La instalación de este proceso shadow se realiza de la siguiente forma :

- Incluir en el archivo "`/etc/services`" una línea como la siguiente:

```
.............
sha-man 6100/tcp # Essentia shadow (Comentario)
.............
```

La misma puede ser insertada al final del archivo.

Advertencia: Deberá controlarse que el port 6100 no esté asignado a otro servicio, en cuyo caso se deberá cambiar por otro valor y utilizar el mismo en todas las máquinas en las cuales se vayan a utilizar.

- Incluir en el archivo "`/etc/inetd.conf`" una línea como la siguiente :

```
.............
sha-man stream tcp nowait root $IDEAFIX/bin/shadow shadow
.............
```

donde `$IDEAFIX` no se debe poner como se muestra, sino que se debe expandir a su contenido real, por ejemplo, si `$IDEAFIX` está instalado en `/usr/idea` la línea anterior debería quedar de la siguiente forma :

```
.............
sha-man stream tcp nowait root /usr/idea/bin/shadow shadow
.............
```

Además, si se desea utilizar varias líneas, se debe insertar una "\" al final de cada una a excepción de la última. El proceso shadow utiliza un archivo de configuración, del cual lee los valores de cada una de las variables para configurar su entorno. Para ello al conectarse un cliente remoto , el shadow utilizará dicho archivo, que se especifica con la opción `-e envfile`, donde *envfile* es reemplazado por el path completo del archivo que se quiere utilizar como archivo de configuración de entorno del proceso shadow. De no existir esta opción en la línea de configuración anteriormente mencionada, Essentia no asumirá ninguno por omisión. Un ejemplo del contenido de este archivo sería :

```
Essentia=/usr/ideafix/essentia
DATADIR=/usr/ideafix/datacpp
LANGUAGE=english
```

***Nota** : Si no se especifica la opción -e el shadow no podrá comunicarse con el server Essentia correspondiente, evitando así que los procesos clientes remotos también lo realicen.*

Existe otra opción también utilizada por el proceso shadow para almacenar información de debugging, que es `-f debugfile` donde *debugfile* es reemplazado por el path completo del archivo en el cual se grabará dicha información.

Un ejemplo completo de configuración del archivo `/etc/inetd.conf` se vería entonces de la siguiente forma :

```
.............
sha-man stream tcp nowait root /usr/ideafix/bin/shadow
shadow !e/usr/ideafix/adm/shadow.env
-f /usr/ideafix/adm/shadow.dbg
.............
```

***Nota** : Para que un usuario se comunique adecuadamente con un server Essentia remoto, debe tener cuenta (de usuario) en el host en que corre Essentia y su máquina (la máquina cliente) debe estar declarada en* `$HOME/.rhosts` *o en* `/etc/hosts.equiv`. *Esto es importante para evitar problemas de negación a aceptar conexiones por parte del host donde corre Essentia.*

Sincronización de Relojes de Hosts Remotos

Estar seguros que la hora de los diferentes hosts en una red o en un grupo de redes interconectadas donde ejecuta Essentia es la misma es una tarea muy importante. Una razón por la cual debe tenerse en cuenta esto es que Essentia utiliza *timestamps* como parte de su procesamiento.

Para lograr este objetivo, todos los hosts UNIX deben tener definido correctamente su *Timezone*. Normalmente la definición del Timezone está ubicada en el file `/etc/TIMEZONE` que puede contener, por ejemplo, la siguiente definición :

```
TZ=ARG3
export TZ
```

Algunos Sistemas Operativos no utilizan esta variable de ambiente. En estos casos suelen utilizarse archivos como `/etc/localtime` o `/usr/ lib/zoneinfo/localtime` que usualmente son links simbólicos a un archivo `/usr/lib/zoneinfo/GMT-3`.

Además de esto, cada equipo UNIX debe ejecutar el proceso **timed**, un *daemon* que permite sincronizar los relojes de hardware de las máquinas. Para cada subred se definen hosts maestros y esclavos. Cada subred debe tener al menos un host maestro.

Para realizar la sincronización se deben utilizar los siguientes comandos:

En los hosts maestros:

```
/etc/timed -M o /usr/sbin/in.timed -M.
```

En los hosts esclavos:

```
/etc/timed o /usr/sbin/in.timed.
```

Algunos UNIX no poseen el comando **timed**; en estos casos se pueden utilizar comandos para configurar la hora local acorde con lo de un host remoto. Estos comandos que no son *daemons* existen en algunos Sistemas Operativos UNIX (como por ejemplo Esix o SunOS), y podemos hacer referencia al comando **rdate**. En este momento Linux tiene un error en el daemon **timed**, pero posee un comando análogo denominado **netdate** (sólo en la Distribución Slackware).

El comando **timed** permite una sincronización con una tolerancia de 1 segundo de diferencia.

Para realizar esta tarea llamada *time synchronization* con mayor precisión, se puede utilizar el protocolo NTP (*Network Time Protocol*) disponible en diferentes Sistemas Operativos que utiliza el **xntpd** daemon. Para su configuración se deberá hacer referencia al manual de TCP/IP, puesto que su complejidad escapa a los alcances de este manual.

Startup y Shutdown Automático de Essentia

Para finalizar la configuración de Essentia, debemos dejar instalado correctamente el Startup y Shutdown automático de los servers que se ejecutan en el equipo, de manera tal que el arranque y detención de los mismos sea transparente para el usuario y se ejecute en el momento de encender o apagar el host respectivamente. Para ello es que existe en Essentia la posibilidad de realizar esta tarea mediante la utilización de un shell script, que ubicado en el directorio correspondiente y con un nombre adecuado, permite lograr tal fin. Cabe aclarar que existen diferentes formas de inicialización entre los diferentes Unix existentes hasta hoy, con lo cual este ejemplo puede variar principalmente en cuanto a la ubicación y nombre de los archivos nombrados en el manual.

Para el ejemplo tomaremos como guía la configuración del Sistema Operativo Unix SCO Versión 3.2.

El shell script que incluye Essentia para el arranque y detención automático, está ubicado en el directorio `$IDEAFIX/bin` con el nombre `rcshell`, que a continuación mostramos :

```
---------------------- cut here --------------------
##########
#
# Si no es invocado con al menos 1 argumento, asumir que
# estamos en una maquina pre-SVR3.0. Chequearemos nuestro
# nombre para ver que hacemos. De otra forma, el primer
# argumento dira que hacemos.
##########
if [ $# = 0 ]
then
case $0 in
*/ess-start )
state=start
;;
*/ess-stop )
state=stop
;;
esac
else
state=$1
fi
set ‘who -r‘
if [ $8 != "0" ]
then
exit
fi
IDEAFIX=/usr/ideafix
DATADIR=$IDEAFIX/data
```

```
Essentia=$IDEAFIX/essentia
SERVERS="server1"
export IDEAFIX
case $state in
'start')
if [ $9 = "2" -o $9 = "3" ]
then
exit
fi
echo "Levantando Essentia..."
rm /usr2/dbess/server1/start/out.*
rm /usr2/dbess/server1/start/err.*
$IDEAFIX/bin/essentia -bw -e0 server1 > /usr2/dbess/
server1/start/out.$DT
2> /usr2/dbess/server1/start/err.$DT
;;
'stop')
echo "Deteniendo Essentia..."
$IDEAFIX/bin/dbstop -wbf server1
;;
esac
------------------------ cut here -------------------
```

El shell anteriormente mostrado debe ser adaptado según la configuración de Essentia, así como los servers con los que ejecute. El proceso de inicialización de UNIX SCO se encuentra en el directorio `/etc` y es ejecutado por el script general llamado **rc**. Este proceso inicializa los diferentes *daemons* pertenecientes a los diferentes estados de UNIX.

Para ejecutar Essentia, es necesario que el host UNIX se encuentre en "**State 3**" (Remote File Sharing State) para permitir la correcta ejecución del server en modo remoto. Como consecuencia de esto, deben ejecutarse los scripts pertenecientes al directorio `/etc/rc2.d` (State 3) entre ellos el más importante para nuestra configuración es el de inicialización de TCP/IP. Estos scripts son ejecutados durante la inicialización por el proceso `/etc/rc2`. Luego de la inicialización del mismo, estaríamos en condiciones de agregar nuestro script de Startup de Essentia.

- Startup : Copiar el archivo arriba especificado con el nombre de **S99ess** en el directorio antes mencionado `/etc/rc2.d`. Las razones por la cual es requerido tal nombre depende de dos puntos : La "S" especifica que es de Start-Up, mientras que el "**99**" especifica que será el último proceso de Start-Up en ejecutarse, debido a que es necesario que otros procesos como el TCP/IP estén corrientes anteriormente, y en este caso se ejecutan teniendo en cuenta dicha numeración ordenado de menor a mayor (los de menor numeración ejecutan primero), .
- Shutdown : Posicionados ahora en el directorio `/etc/rc0.d` realizar un link con el script ubicado en `/etc/rd2.d/S99ess` denominando a este nuevo script K01ess. También en este caso el nombre es de vital importancia puesto que la "K" especifica Kill y el "01" que éste se ejecutará primero, o sea el orden inverso al proceso de Startup.

Una vez instalados estos scripts y configurados internamente para que las variables IDEAFIX, Essentia y SERVERS se encuentren inicializadas correctamente, el Startup y Shutdown de los

servers se realizará en forma automática sin intervención alguna del usuario.

Log De Errores De Essentia

Essentia, al igual que otros procesos de UNIX, utiliza el registro de mensajes del Sistema Operativo cuyo daemon es conocido como "**syslogd**" y que permite la visualización en un archivo de texto de los diferentes errores ocurridos en el sistema. Para la correcta visualización de los errores producidos por Essentia, se debe modificar el archivo de configuración de este *daemon* en `/etc/syslog.conf` insertando la siguiente línea :

```
...
local0.err;local0.warning;local0.notice /usr/adm/messages
...
```

Esta modificación se debe a que la facilidad `local0` es la utilizada por Essentia para enviar mensajes de Error, Advertencia o Notificación. Todos estos niveles de severidad pueden ser especificados en la misma línea separados por ";" y enviados a un mismo destino, en nuestro caso el archivo `/usr/adm/messages`, con lo cual la visualización del mismo permitirá conocer los diferentes mensajes enviados por Essentia y de ser un ERROR poder conocer el origen del mismo.

Los errores, advertencias o notificaciones son variados, y van desde errores del server hasta avisos de checkpoint y demás tareas que el server desea informar.

Capítulo 3

Seguridad e Integridad de los Datos

Seguridad

En un sistema local, el manejo de la seguridad queda exclusivamente a cargo del server.

En un sistema remoto, se deben verificar además las condiciones usuales para trabajar en equipos remotos: el usuario debe tener una cuenta en el host donde reside el server, y el nombre del host del cliente debe figurar en el archivo `/etc/hosts.equiv` del equipo donde corre el server o del `$HOME/.rhosts` del usuario en cuestión. Para información sobre la estructura de estos archivos `/etc/hosts.equiv` ver el Manual de Administrador de UNIX, sección configuración de red.

En ambos casos, el server maneja la parte de la seguridad que le corresponde haciendo uso de los permisos para usuarios y grupos que provee el sistema operativo UNIX, y también con los permisos asignados en la definición de cada base de datos.

Integridad (Checkpoints)

Un checkpoint es un proceso realizado por el server (a intervalos regulares o a pedido del cliente) que asegura la consistencia de los datos. Essentia maneja dos tipos de checkpoints:

- Checkpoints de consistencia de cache.
- Checkpoints de consistencia de transacción.

La consistencia de cache asegura que toda la información guardada en disco (tablas e índices) es correcta. Como pueden existir transacciones incompletas, este tipo de checkpoints puede ser usado sólo cuando el journal está también activo.

La consistencia de transacción es mas sólida, porque asegura que todas las transacciones se

completan y se guardan en disco. Este tipo de checkpoints puede realizarse solamente cuando no hay transacciones ejecutándose.

Essentia puede ser configurado para realizar checkpoints de consistencia de transacción a intervalos regulares utilizando algunos de los parámetros del archivo .sv. Para mas información acerca de este archivo, ver el Capitulo 2, Instalación.

Los checkpoints se pueden también realizar manualmente utilizando la herramienta **chkpoint**. Si no se le pasan parámetros, se realiza checkpoint de consistencia de cache. Si se le pasa el parámetro "-t" se asegura la consistencia al nivel de transacciones. Si el journal se encuentra desactivado, el único tipo de checkpoint posible es el de consistencia de transacciones.

Finalmente, es importante mencionar que algunas herramientas de IDEAFIX poseen parámetros para realizar un checkpoint una vez concluida su tarea. De éstas, las más importantes son **dgen** e **iql**, que poseen la opción "-k".

Cada vez que se crea un nuevo esquema, la nueva información no estará disponible para los usuarios hasta que se realice un checkpoint. La opción "-k" asegura que los esquemas estén disponibles inmediatamente luego de actualizados.

Capítulo 4

Backup

Essentia permite realizar dos tipos diferentes de backups: completos e incrementales. Si bien el backup completo es el más utilizado, presenta las siguientes desventajas:

- Durante cada backup se copia la base de datos completa, ocupando así un espacio considerable.
- Si se desea restaurar la base de datos se obtendrá su estado a la última copia de seguridad, perdiéndose las modificaciones efectuadas a partir de ese momento.

El backup incremental permite resguardar los datos a medida que se van modificando, aportando así las siguientes ventajas:

- Sólo se guardan las modificaciones; no hay replicación innecesaria de datos.
- En caso de una falla puede reconstruirse el estado de la base de datos hasta el punto en que la falla ocurrió.

Por estas cualidades, es recomendable utilizar el sistema de backup incremental. Durante el resto del capítulo nos ocuparemos de los pasos a seguir para realizar backups incrementales y de como recuperar los datos cuando sea necesario.

Funcionamiento

La copia de seguridad incremental es efectuada por un proceso cliente especial de Essentia (*secondary server*) *denominado "dbibup"*.

El **dbibup** consulta periódicamente el *journal* en busca de nuevas transacciones, permitiendo:

- Que los clientes continúen trabajando mientras se realiza el backup.
- Que Essentia no muestre un impacto de performance por realizar esta operación.

Creación del backup

A grandes rasgos, los pasos a seguir para para habilitar el backup incremental son:

- Poner en valor 1 la opción de backup incremental en el archivo de configuración (opción *Backup* igual a 1).
- Realizar un backup completo de la base de datos mediante el utilitario **dbbup**.
- Ejecutar el backup incremental utilizando el utilitario **dbibup**.

Para restaurar los datos, se deberá recuperar primero el backup inicial y luego los incrementales en orden cronológico.

Pasos

A continuación se explican los pasos en forma detallada, asumiendo que el server a resguardar se denomina “sisint”.

Primer paso

Modificar el archivo de configuración del server para habilitar el backup incremental. Se debe editar “sisint.sv” y en el parámetro *Backup* debe colocarse el valor 1 (uno).

Este parámetro le indica al server que no se deben borrar los archivos del journal hasta que hayan sido resguardados por el **dbibup**.

En caso de no utilizarse backup incremental, *Backup* debera tener valor 0 (cero).

Segundo paso

Realizar un backup completo del server mediante el utilitario **dbbup**. Este se puede realizar en un archivo o sobre un dispositivo.

En el primer caso, el utilitario se ejecutará de la siguiente forma:

```
dbbup -d /usr3/backup/sisint.bup sisint
```

donde *-d* indica el archivo donde se guardaran los datos. Al ejecutar este comando, se imprimirán por la salida estándar los nombres de los archivos resguardados. Estos son:

- Tablas e índices administrados por el server.
- Archivo de estado.
- Log de transacciones.
- Archivo de configuración del server (“.sv”).

En el segundo caso (cuando, por ejemplo se desea guardar en cinta y el dispositivo de la cinta es `/dev/tape`), los parámetros para **dbbup** son los siguientes:

```
dbbup -d /dev/tape sisint
```

Si se está utilizando el utilitario **tar** de IDEAFIX (utilizado por omisión por **dbbup** y **dbibup** para realizar las copias), se podrá especificar el equipo en el que se desea realizar la copia de resguardo:

```
dbbup -d guest@gandalf:/dev/tape sisint
```

Como se puede ver, el destino es el dispositivo "`/dev/tape`" del *host* "`gandalf`". Para realizar la copia, se accederá al otro equipo utilizando el usuario "`guest`" . Cuando el usuario actual posea cuenta en ambos equipos (el local y el remoto) y pueda realizar copias remotas(rcp) entre ambos, bastará ejecutar:

```
dbbup -d gandalf:/dev/tape sisint
```

Para poder realizar copias remotas entre dos equipos se debe agregar al archivo ".rhosts" en el *home directory* del usuario los nombres de ambos equipos.

Advertencia: los utilitarios para realizar backup deben utilizar el comando **tar** de IDEAFIX para poder realizar dispositivos remotos.

Tercer Paso

Ejecutar el backup incremental. Este puede tambien utilizar las dos variantes: backup a un archivo o backup a un dispositivo.

En el primer caso se ejecuta el backup incremental de la siguiente forma:

```
dbibup -d /usr3/backup/sisint.ibup -t 600 -r sisint
```

Como se puede ver el ejemplo, el **dbibup** quedará corriendo en *background* hasta que finalice la ejecución del server del cual depende. Para verificar que este proceso está funcoinando correctamente, se puede utilizar el comando **dbps,** ya que el **dbibup** es un cliente más de Essentia.

La opción "-r" de este ejemplo especifica que este último es un archivo común y que **dbibup** debe agregarle una nueva estensión cada vez que resguarda transacciones nuevas, evitando así borrar datos antiguos. **dbibup** irá entonces agregando como extensión un número correlativo creciente entre los archivos. La opción "-t 600" indica que el backup incremental consultará cada 600 segundos el *journal* en busca de nuevas transacciones. En caso de no estar especificado , el **dbibup** realizará un único backup del log de transacciones y luego finalizará su ejecución.

Para suspender temporalmente el backup incremental, deberá ejecutarse el siguiente comando:

```
dbshut dbibup sisint
```

En el segundo caso (backup incremental sobre dispositivo) el comando a ejecutar es el siguiente:

```
dbibup -d guest@gandalf:/dev/tapenr -t 600 sisint
```

El dispositivo a utilizar en este caso es "`/dev/tapenr`", ya que ese dispositivo no rebobina la cinta luego de cada copia, permitiendo así que las sucesivas copias del journal se graben una a continuación de otra.

Advertencia: los utilitarios para realizar backup deben utilizar el comando **tar** de IDEAFIX para poder realizar backups a dispositivos remotos.

Advertencia: El dueño de los utilitarios **dbbup** y **dbibup** debe ser *root*, y ambos programas necesitan tener permiso para ser ejecutados como dicho usuario. Esto es necesario ya que ambos utilitarios acceden a los archivos del *journal* y datos del server, sobre los cuales únicamente *root* tiene permisos de acceso. Si los programas no tienen correctamente definidos estos permisos, esto se puede hacer con el siguiente comando:

```
chmod ugo+s dbbup dbibup
```

Recuperación de los backups

Para recuperar la base de datos y el server en base a los backups se deben realizar los siguientes pasos:

Primer Paso

Verificar que Essentia no esté corriendo, y, si lo está, realizar un *shutdown*. Luego recuperar el backup completo teniendo en cuenta lo siguiente:

- El backup se restaurará con el comando **tar** de IDEAFIX.
- el **tar** deberá ser ejecutado desde el usuario *root*.
- En el momento de realizar la restauración el usuario deberá estar ubicado en el directorio raíz del filesystem objetivo.

El comando para recuperar un backup en archivo es:

```
/usr3/idea43/bin/tar xvf /usr3/backup/sisint.bup
```

y para un dispositivo:

```
/usr3/idea43/bin/tar xvf guest@gandalf:/dev/tape
```

Segundo paso

Recuperar cada uno de los backups incrementales *en orden cronológico*. El comando para restaurar un backup de archivos es:

```
tar xvf /usr3/backup/sisint.ibup.n
```

donde *n* es el número de backup incremental del server. Recordemos que esté número era asignado automáticamente por **dbibup** al realizar las copias.

Para obtener los datos desde un distositivo el comando sería:

```
/usr3/idea43/bin/tar xvf guest@gandalf:/dev/tapenr
```

En este caso, se utiliza el dispositivo "`/dev/tapenr`" para no rebobinar la cinta luego de recuperar los datos parciales. El comando deberá ejecutarse tantas veces como sea necesario para recuperar las diferentes porciones del journal resguardadas por el **dbibup**.

Tercer Paso

Relizar un *startup* de Essentia normalmente:

```
essentia sisint
```

Cuando el server comience a trabajar ejecutará un *checkpoint* transaccional. Durante el mismo reflejará en la base de datos las operaciones contenidas en los journals, resguardados por el **dbibup**, restaurando así la base de datos al estado del último backup incremental.

Capítulo 5

Mirroring

Escenarios con necesidades de performance on-line críticas y rápida recuperación son cada vez más comunes. El servidor ESSENTIA brinda para ello la posibilidad de espejar la totalidad de la base de datos, permitiendo ante una falla de hardware del server con los datos reales, reiniciar el trabajo en otro server con la base de datos espejada. Este espejado o *mirroring* es logrado gracias a la utilización de un *Secondary Server* de Mirroring, quien se encarga de replicar los datos utilizando el journal de transacciones que maneja ESSENTIA, la base de datos ya sea en forma local o remota. Este proceso de Mirroring no implica en lo absoluto una pérdida de performance, puesto que el uso de Secondary Servers evita *overhead* en ESSENTIA.

Funcionamiento

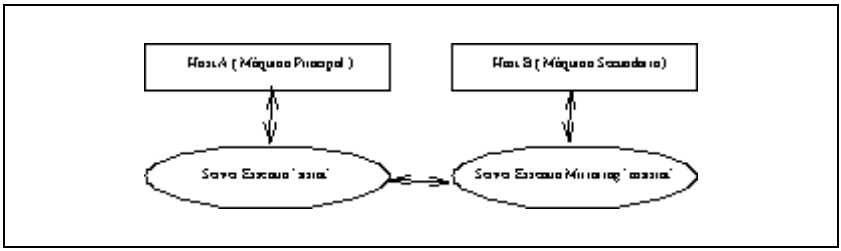
La configuración de ESSENTIA para realizar mirroring es sencilla. Para su funcionamiento debe existir por supuesto un server original, que deberá tener configurado su archivo .sv con la opción Mirroring con valor 1 (uno) lo cual implica que el mismo no borrará el journal de transacciones luego de cada checkpoint, puesto que será utilizado para replicar la base y luego de esto serán borrados. También debe existir otro server que contenga la misma estructura de server y datos que el original, al que llamaremos en adelante "mirror essentia server". Es importante destacar que los esquemas, tablas y datos deben ser exactamente iguales que los del server original antes de comenzar con el proceso de mirroring. Este server puede ser utilizado como cualquier otro server pero *read-only*, debido a que rechazará operaciones de modificación o creación de registros.

En la misma máquina donde corre el server original, debe ejecutarse un proceso denominado **dbmirror**, que es un "secondary server" o sea un server en un nivel inferior a ESSENTIA (vemos aquí los múltiples niveles del server), quien será el encargado de copiar periódicamente de los archivos del journal del server original al directorio que corresponda del "mirror essentia server". Para copiar estos archivos el programa **dbmirror** utiliza un comando provisto con ESSENTIA denominado **dbmcopy**, que es ejecutado en la máquina donde corre el *mirror essentia server* a través de la función de UNIX **rcmd**. **dbmcopy** es casi

equivalente al comando **cp** de UNIX pero permite copias incrementales, mejorando así la performance.

Para comprender estos conceptos con mayor claridad veamos un ejemplo de una sesión de instalación de un server ESSENTIA original con su correspondiente "mirror server" :

Ejemplo:



Supongamos que nuestro server original tiene el nombre "sisint" y corre en el Host A mientras que el "Essentia Mirror Server" se llama "msisint" y ejecuta en el Host B.

Configuración

Primer Paso

Instalar el server "sisint" en el Host A y ponerlo en funcionamiento normalmente.

Instalar el server "msisint" en el Host B y asegurarse que el server "msisint" tenga los mismos esquemas que el server original "sisint" sin ninguna modificación y que tengan exactamente los mismos datos. Puede haber además otros esquemas en el server "msisint" pero un Mirror Essentia Server sólo puede actuar como mirror de un único Server ESSENTIA (en nuestro caso "sisint").

Segundo Paso

Realizar un *shutdown* de ambos servers y luego borrar los archivos de journal que hayan quedado en los directorios de journal de ambos Servers ESSENTIA.

Una vez que los datos y estructura de la base de datos coincide en ambos Servers, y que la opción Mirroring del archivo de configuración Essentia tiene valor 1:

En el Host A ejecutar :

```
# essentia sisint
```

En el Host B :

Asegurarse que las variables del archivo `/etc/ess.env` tienen los valores correctos (Ver Capítulo 2 - Instalación y Configuración de Essentia Remoto).

Deshabilitar el parámetro de Mirroring en el `msisint.sv` asignándole valor 0.

Ejecutar :

```
# essentia -m msisint
```

Tercer Paso

En el Host A ejecutar :

```
# dbmirror -t 30 -i B.msisint sisint
```

Este proceso es el que realmente se encarga de realizar el proceso de mirroring y replicar la base de datos de un server a otro. La opción `-t 30` del **dbmirror** indica que el proceso de mirroring verificará la aparición de nueva información en los archivos de journal cada 30 segundos.

Si en algún momento se desea suspender temporalmente el proceso de mirroring, se puede usar el comando:

```
# dbshut dbmirror
```

Y reanudar nuevamente con el mismo comando:

```
# dbmirror -t 30 -i B.msisint sisint
```

Si en algún momento se desea comenzar a usar el server de mirroring como server real (ante una falla del server original), se debe parar el server de mirroring mediante el comando **dbstop** y levantarlo nuevamente sin el modificador "-m" para permitirle trabajar en modo normal.

Advertencias

- El usuario que lanza el comando **dbmirror** debe tener cuenta en ambos equipos y poder ejecutar el comando *rcp* entre ellos sin problemas.
- Se debe realizar un *link* del comando **dbmcopy** en alguno de los directorios que se incluyen automáticamente en el `PATH` de un Remote Comand (rcmd), por ejemplo en `/bin`.
- El *SchemaId* de los esquemas en el server original y en el de mirror deben coincidir. Para esto es conveniente que se copie el directorio dbase entero desde el Server

Original en el Host A al Server de Mirroring en el Host B, y también que se copie la línea Schemas del archivo de configuración .sv del Server Original al archivo de configuración ".sv" del Server de Mirroring.

Capítulo 6

Fragmentación de tablas

Esta versión de Essentia introduce modificaciones que permiten a las tablas, junto a sus índices, estar repartidas en más de un archivo. Estos archivos pueden pertenecer a distintos file systems y, por ende, a discos diferentes.

Esto da solución al problema del límite de 2 Gb para tamaño de archivo, usualmente encontrado en los sistemas UNIX de 32 bits.

En adelante, al referinos a una “tabla” nos referiremos a un conjunto de tablas físicas necesario para el almacenamiento de los datos (junto a los índices asociados).

Tipos de tabla

Las tablas de un esquema dado pueden ser de tres tipos diferentes: *packed*, *ranged* y *round-robin*.

- **Packed:** son aquellas que tienen una estructura compacta. Este es el formato de las tablas usadas en versiones previas de Essentia.
- **Ranged:** son aquellas que pueden ser fragmentadas en dos o más partes, que son llenadas en orden y en tanto se necesite espacio. Este es el tipo de tabla que crece en forma lineal, y que usan otros directorios o *filesystems* a medida en que se llenan con información. Por lo tanto, existe la posibilidad de crear nuevos directorios en donde continuar el almancenamiento de la tabla cuando el directorio original de la tabla no admite más lugar. Este tipo de tablas constituye el formato más flexible en lo que concierne al manejo de espacio de disco, pero requiere que el administrador revise periódicamente si el tamaño está por exceder el máximo permitido en el file system que se está usando, y agregar un nuevo directorio para la tabla en caso de ser necesario.
- **Round Robin**: son aquellas que están limitadas al espacio original que se les asigne en un principio, pero con la ventaja de poder repartirse en varios directorios. Esto último hace que potencialmente no tengan límite de tamaño. A diferencia del tipo

ranged, estas guardan un bloque de datos en cada directorio, distribuyéndolos en forma alternativa. Como sucede con las del tipo packed, el tamaño de estas tablas no puede cambiar o redefinirse fácilmente después de haber sido establecido la primera vez.

Archivo de descripción de distribución física de datos

Este archivo describe la fragmentación de cada tabla del esquema manejado por el server. El formato de este archivo es el siguiente:

```
esq.tab<tab>definición
esq.tab<tab>definición
...
```

donde

- *esq.tab* es el nombre del esquema (*esq*) y tabla (*tab*) cuyo tipo será establecido.

La sintaxis de una definición PACKED es la siguiente:

```
PACKED<tab>directorio
```

donde

- *directorio* es el directorio que contiene las tablas. Esto no puede ser fácilmente modificado una vez que el sistema está en funcionamiento, por lo que se debe considerar el máximo espacio posible de la tabla para evitar problemas de espacio.

La sintaxis de una definición RANGED es la siguiente:

```
RANGED<tab>directorio[:tam,directorio[:tam,...,directorio]]
```

donde

- directorio[:tam, directorio[:tam,...,directorio]] es la lista de directorios que serán usados para alojar la tabla. El último de estos no deberá tener límite de tamaño, de modo que la lista siempre termina con un nombre de directorio. El límite de tamaño puede ser especificado en el formato `n[K|M|G]`, donde *n* es el número de unidades que se dispone como límite, y K, M y G son las unidades a usar (K=kilobytes, M=megabytes, G=gigabytes). Si no se especifica la unidad, se usa K por omisión.

Por último, la sintaxis de una definición ROUNDROBIN es la siguiente:

```
ROUNDROBIN<tab>tambl<tab>directorio[,directorio,...]
```

donde

- *tambl* es el tamaño del bloque que Essentia usará para almacenar cada uno de los directorios especificados, en el formato n[K|M|G], donde *n* es el número de unidades que se dispone como límite, y K, M y G son las unidades a usar (K=kilobytes, M=megabytes, G=gigabytes). Si no se especifica la unidad, se usa K por defecto.
- *directorio[, directorio, ...]* es la lista de directorios en donde se almacenarán las tablas.Esto no puede ser modificado fácilmente, por lo que se debe considerar el máximo espacio posible de la tabla para evitar problemas de espacio.

Nota: es fundamental el hecho de que los caracteres "<tab>"que aparecen en las descripciones anteriores deben ser insertados exactamente como se indica. También, la línea no puede tener caracteres de más (espacios, tabulados u otros) al final, o Essentia abortará con un mensaje de error de sintaxis.

Ejemplo

El siguiente es un ejemplo que muestra cómo especificar la distribución de una tabla de un esquema:

```
aurus.$*<tab>PACKED<tab>/disk1/dbess/
aurus.cbtes<tab>RANGED<tab>/disk2/dbess/:1M,/disk3/dbess/
aurus.astos<tab>roundrobin<tab>2k<tab>/disk2/dbess,disk3/dbess/
aurus.*<tab>PACKED<tab>/disk1/dbess/
```

El contenido de este archivo significa lo siguiente:

- La primera línea define las tablas de definición del esquema como PACKED (obligatorio) y las aloja bajo /disk1/dbess. Hay en Essentia un tipo especial de tabla llamada *tabla de descripción de esquemas*, que contiene la distribución de registros del esquema. Pueden identificarse por tener un signo pesos ($) al comienzo, y *deben* ser de tipo packed. Es común, por ende, que la primera línea del archivo de distribución sea

```
esq.$*<tab>PACKED<tab>/usr/tabsdsc
```

en donde *esq* identifica el esquema deseado del server, y $* identifica *todas* las tablas de descripción de esquemas del esquema.

- La segunda línea define la tabla *cbtes* como RANGED, dejando 1 megabyte bajo /disk/dbess y el resto de la tabla bajo /disk3/dbess.
- La tercera línea define que los archivos físicos pertenecientes a la tabla *astos* serán alojados bajo /disk2/dbess y /disk3/dbess, usando un tamaño de bloque de 2 kbytes y el esquema de almacenamiento *roundrobin*.
- Las últimas líneas definen que el resto de las tablas del esquema *aurus* será almacenado (usando el tipo packed) bajo /disk1/dbess.

Nota: si una misma tabla se define más de una vez, Essentia considerará sólo la primera de estas definiciones.

Compatibilidad con versiones previas

Cuado se usa con Essentia 3.3.1 una tabla de Essentia 3.3 o anterior, Essentia creará automáticamente un archivo de descripción de distribución usando la información del archivo '.sv'. Para que esta conversión sea hecha en forma automática, la entrada *DiskConfiguration* deberá establecerse correctamente en el archivo de configuración (Ver el capítulo 2).

Ejemplo

Si el archivo '.sv'tiene la siguiente línea para la definición del esquema:

```
Schemas<tab>esq1:dir_esq1:id_esq1,esq2,dir_esq2:id_esq2
```

la distribución física que Essentia generará será la siguiente:

```
esq1.$*<tab>PACKED<tab>dir_esq1
esq1 *<tab>PACKED<tab>dir_esq1
esq2.$*<tab>PACKED<tab>dir_esq2
esq2 *<tab>PACKED<tab>dir_esq2
```

y la línea que define el esquema será convertida al formato de Essentia 3.3.1, quedando como sigue:

```
Schemas<tab>esq1:id_esq1,es12:id_esq2
```

Creación de nuevos esquemas

Para la creación de un esquema nuevo, Essentia continuará utilizando el contenido de la línea que define *Dbase* en el archivo '.sv'. Cuando un esquema nuevo aparece en esta línea, Essentia automáticamente generará nuevas entradas en el archivo de distribución física y en la línea que define *Schemas*.

Cambios de tipos de tablas

Para cambiar la especificación de un tipo de tablas, es necesario realizar un *shutdown* del server. Luego de esto, puede modificarse el archivo de distribución física sin riesgo alguno.

Si el esquema cuyas tablas cambiarán de tipo no está vacío, será necesario exportar sus datos, para luego hacer una operación de borrado de tabla (*clear table*) antes de realizar el *shutdown* de Essentia. A continuación, después de levantarlo, será necesaria la operación de **import**ación de datos para restaurar el contenido.

Los pasos necesarios para cambiar un tipo de tabla pueden resumirse del siguiente modo:

- Exportar los datos de la tabla usando el comando **export**
- Ejecutar un *clear table* de la tabla correspondiente
- Realizar un *shutdown* del server
- Cambiar el archivo de descripción de distribución física
- Realizar un *startup* del server
- Importar los datos de la tabla usando el comando **import**

Capítulo 7

Herramientas

Essentia provee un número de utilitarios para facilitar la administración y el mantenimiento de un servidor de bases de datos fácil. Estas herramientas, utilizadas junto con las de IDEAFIX, proveen un completo control sobre la base de datos.

ESSENTIA

Sintaxis

```
essentia [opciones] <server>
```

Descripción

Ejecuta el server de base de datos.

Opciones

-V, --version : Imprime la versión del programa

-b, --hide-banner : oculta el mensaje de identificación.

-r, --read-log : Reproduce las operaciones guardadas en el *RunLogFile* especificado en la archivo de configuración

-w, --write-log num : Guarda todas las operaciones realizadas en el *RunLogFile* especificado en el archivo de configuración. Para aquellos equipos que soportan *memory mapped files* (MMFs) tanto la lectura como la grabación del log de Essentia se realizará utilizándo dicha facilidad. Esto disminuye sustancialmente el impacto en la performance de Essentia cuando se utiliza este modificador.

El *n* es la cantidad de KBytes que ocupará la página utilizada para el *mapping* del log en

memoria. Cuando la instalación no soporte MMFs, se recomienda pasar valor cero (`-w 0`).

-v, --verbose : Verborrágico. Muestra información adicional en el estado y las operaciones realizadas por el server.

-u, --verbose-level num : nivel de Verbose (1es equivalente a -v)

-f, --foreground : Corre el servidor en *foreground* (modo dedicado).

-p, --check-previous-cp : chequea el *checkpoint* previo.

-e, --check-level num : Chequeo de índices exhaustivo. (n=0, solo al comienzo; n=1, antes de cada checkpoint.

-a, --administrator num : Indica el nombre del único usuario habilitado para conectarse con Essentia. Este modificador es útil cuando se realiza el mantenimiento de un server.

-i, --ignore str : Procesar el journal ignorando las transacciones con id especificados.

-m, --mirror : Corre el server como un server de mirroring.

-t, --trans-query-behavior num : Comportamiento de una transacción cuando más de un server está trabajando. Los valores posibles son:

0: *EndTransaction* con *two-phase commit* (2PC)

1: *AbortTransaction* con *two-phase commit* (2PC)

2: Fuerza un *EndTransaction* para aquellas transacciones con 2PC.

3: Fuerza un *AbortTransaction* para aquellas transacciones con 2PC.

-H Muestra la sintaxis y opciones del comando.

<server> Archivo del configuración del server.

DBPS

Sintaxis

```
dbps [opciones][server]
```

Descripción

Obtiene información (con diferentes niveles de detalle) sobre los clientes de un server.

Opciones

Si no se pasan opciones, lista el número de proceso de cada cliente, la terminal en la que

corre, y el tiempo que el servidor ha estado corriendo. Si no se especifica ningún servidor, la herramienta usa el valor de la variable de ambiente SERVERS. Si en esta línea hay más de un server especificado, **dbps** utilizará el primero.

-b, --hide-banner: Suprimir mensaje de identificación.

-H, --help : Muestra la sintaxis y opciones del comando.

-V, --version : imprime la versión del programa.

-l, --long: Muestra, además, información en el nombre del *host*, el nombre del usuario, la hora de comienzo del proceso.

-f, --full: Muestra toda la información de "-l" más el nombre del grupo del usuario, el número de objetos que está usando, el ID del cliente y su estado de lock.

DBSHUT

Sintaxis

```
dbshut [opciones][secserver][server]
```

Descripción

Detiene uno o todos los *secondary servers*.

Opciones

-b: --hide-banner: Suprimir mensaje de identificación.

-H, --help: Imprime ayuda acerca de las opciones del comando.

-a, --all: realiza un shutdown de todos los *secondary servers.*

-V, --version; imprime la versión del programa.

secserver secondary server a detener, perteneciente a *server*.

A la fecha, los únicos *secondary servers* existentes son **dbmmirror** y **dbibup**.

DBSTOP

Sintaxis

```
dbstop [opciones][server]
```

Descripción

Detiene un server.

Opciones

-H, --help : Suprimir mensaje de identificación.

-V, --version : Imprime la versión del programa.

-b: --hide-banner: Suprimir mensaje de identificación.

-f, --force : Detiene el server aún cuando haya clientes corriendo o el server se encuentre en estado corrupto.

-u, --unconditional : Detiene el server incondicionalmente. Se utiliza cuando el server ha entrado en un *loop* y no responde a los mensajes de los clientes (p.ej., cuando se ha agotado el espacio en disco, o no tiene permisos de acceso).

-w, --wait : Espera a la finalización del server para devolver el *prompt*.

-t, --time num : Número de segundos que espera para detenerse luego de enviar avisos a los secondary servers (default: 20 s).

-H, --help: Muestra la sintaxis y opciones del comando.

Aviso: Las opciones "-f"y "-u" deben ser utilizadas con cuidado, debido a que fuerzan el server a detenerse incondicionalmente y sin consultar a los clientes que lo utilizan. Esto no afecta a la base de datos, pero interrumpe los procesos que estaban corriendo.

DBBUP

Sintaxis

```
dbbup [opciones][server]
```

Descripción

Realiza un *backup* completo de los esquemas atendidos por un server.

Opciones

-H, --help : Muestra la sintaxis y opciones del comando.

-V, --version : Imprime la versión del programa.

-d, --device str : Especifica el archivo o dispositivo a utilizar. El valor por defecto es "`/dev/tape`".

-p, --preserve : No borra los archivos copiados del Journal.

-c, --command str : Este modificador permite especificar el comando exacto que se utilizará para resguardar los datos. *str* es una cadena de caracteres de la siguiente forma: "`gtar -T %F -zcvf %D`"

Donde:

%F es es archivo que contiene la lista de archivos a resguardar

%D es el dispositivo a utilizar (especificado en la opción -d dispositivo)

El valor por omisión de esta variable es "`tar cvf %D 'cat %F'`". Es importante notar que este comando puede tener problemas dependiendo del sistema operativo en el que se usa si la lista de archivos es muy grande (es decir, cuando se obtiene el error 'arg list too long').

-d *disp* Especifica el archivo o dispositivo a utilizar. El valor por defecto es "/dev/tape".

-b, --hide-banner: Suprimir mensaje de identificación.

Notas

Para que este servicio pueda utilizarse es necesario que la Variable *Backup* del Archivo ".sv" tenga valor 1.

DBIBUP

Sintaxis

```
dbibup [opciones][server]
```

Descripción

Realiza un *backup* incremental de los esquemas atendidos por un server.

Opciones

-H, --help : No borra los archivos del Journal copiados.

-V, --version : Imprimir la versión del programa.

-d, --device str : Especifica el archivo o dispositivo a utilizar. El valor por defecto es "`/dev/tape`".

-t, --*time num* : Realiza backups incrementales cada "n" minutos.

-r, --regular : Implica que el archivo indicado con la opción "-d" es común y no de carácter. De esta forma, cada vez que un nuevo backup es creado, al archivo se le agregará una nueva extensión para evitar borrar los datos antiguos.

-p, --preserve : No borra los archivos del Journal copiados.

-f, --foreground : Corre el proceso de backup en foreground (modo dedicado).

-c, --*command str* : Este modificador permite especificar el comando exacto que se utilizará para resguardar los datos. com es una cadena de caracteres de la siguiente forma: "`gtar -T %F -zcvf %D`"

Donde:

%F es es archivo que contiene la lista de archivos a resguardar

%D es el dispositivo a utilizar (especificado en la opción -d dispositivo)

El valor por omisión de esta variable es "`tar cvf %D | 'cat %F'`". Es importante notar que este comando puede tener problemas dependiendo del sistema operativo en el que se usa si la lista de archivos es muy grande (es decir, cuando se obtiene el error 'arg list too long').

-b, hide-banner: Suprimir mensaje de identificación.

Notas

Para que este servicio pueda utilizarse es necesario que la Variable Backup del Archivo ".sv" valga 1. Este flag en 1 no implica que el comando **dbibup** pueda ejecutarse; sólo indica si deben borrarse o no los archivos del journal.

No puede invocarse a **dbibup** sin antes invocar a **dbbup**.

CHKPOINT

Sintaxis

```
chkpoint [opciones][server]
```

Descripción

Realizar un *checkpoint* en un server.

Opciones

-H, --help : Muestra la sintaxis y opciones del comando.

-V, --version : Imprime la versión del programa.

-b, --hide-banner : Suprimir mensaje de identificación.

-t, --tcc : Realiza un checkpoint de consistencia de transacción, en lugar de un checkpoint de consistencia de cache.

-f, --force : Fuerza el *checkpoint*, aún cuando el server esté corrupto.

DBMON

Sintaxis

```
dbmon [opciones][server]
```

Descripción

Analiza el estado de un server, mostrando información estadística en el número de transacciones y métodos procesados, el número de locks activos y el estado del cache.

Opciones

-b, --hide-banner : Suprimir mensaje de identificación.

-t n, --time n : Tiempo en segundos que el programa esperará entre actualizaciones de información. Si esta opción no se utiliza, el programa actualizará la información cada vez que se presione una tecla.

-H, --help : Muestra la sintaxis y opciones del comando.

DBMIRROR

Sintaxis

```
dbmirror [opciones][server]
```

Descripción

Lanza un proceso de *mirroring*.

Opciones

-H, --help : Muestra la sintaxis y opciones del comando.

-V, --version : Imprime la versión del programa.

-t, --time num : La frecuencia en segundos a esperar para hacer el mirroring.

- ***c, --command str*** : Usar el comando especificado para copiar cada archivo.

-p, --preserve : Preserva los archivos de *journal* (no los borra).

-i host, --intelligent host : Hacer copias inteligentes al host especificado.

-f, --foreground : Lanza el server en *foreground* (modo dedicado)

-b, hide-banner: Suprimir mensaje de identificación.

NEWVERS

Sintaxis

```
newvers [opciones][server]
```

Descripción

Realiza una nueva versión sobre el server pasado como parámetro, o sobre el primer server de la variable de ambiente SERVERS en caso de omisión.

Opciones

-b, --hide-banner: Suprimir mensaje de identificación.

-H, --help: Muestra la sintaxis y opciones del comando.

-V, --version : Número de versión

Capítulo 8

Utilitarios de IDEAFIX con Essentia

CFIX

La compilación de un programa con Essentia o IDEAFIX se realiza invocando al utilitario **cfix** de la siguiente forma:

Si se utiliza el modificador -S, **cfix** usará las bibliotecas comunes de IDEAFIX.

Invocando **cfix** con el modificador -E serán utilizadas las bibliotecas de IDEAFIX que utilizan Essentia.

En el caso de estar definida la variable de ambiente ESSENTIA, al compilar no es necesario utilizar el modificador -E.

DGEN

El utilitario **dgen** incluye la opción -k que realiza un *checkpoint* en forma automática una vez procesado el esquema. Esto habilita a otros procesos a utilizarlo.

EXP

En este utilitario se agregó la opción -V, que permite exportar una versión en particular . Supóngase que a lo largo del tiempo se crearon tres versiones diferentes:

Fecha 1/1/921/1/941/1/95

Versión 123

En este caso la utilización del modificador -V tendría el siguiente efecto:

Modificador utilizado	Versión exportada
exp	última
exp -V3	tercera (en este ejemplo la última)
exp -V1	primera
exp -V0	última
exp -V-1	segunda (versión anterior a la última)
exp -V-2	primera (versión antepenúltima)
exp -V2/1/92	primera
exp -V1/1/94	segunda

IMP

Cambió el comportamiento por omisión de este utilitario. La tarea de **imp** es considerada (por omisión) como una única transacción, por lo cual se bloquea la tabla para evitar que se supere la cantidad de locks por cliente configurada para el server Essentia con el que se trabaja. De esto se deduce que cualquier error (o interrupción del usuario con Ctrl-C por ejemplo) hace que la transacción aborte lo cual puede resultar inconveniente cuando la cantidad de registros a importarse es considerable ya que el server deberá volver atrás quizás millones de altas.

La opción "-l" desactiva el bloqueo de la tabla y realiza una transacción por cada `Put/DelRecord` con lo cual si se presenta algún error los registros importados (o borrados) hasta ese momento quedan grabados (borrados) en la base.

La opción "`-G n`" realiza una transacción por cada `Put/DelRecord`.

Para *imports* masivos se recomienda desactivar el journal del server Essentia, con lo cual se logra mayor rapidez y se reduce el espacio necesario en disco.

IQL

Se agregó también una nueva sentencia IQL:

```
use esquema version [nro|fecha] as nombre;
```

Esta sentencia permite abrir una versión indicando el número o fecha de la misma. Una vez abierta se referenciará la misma a través del nombre.

Capítulo 9

Funciones de Biblioteca (CFIX)

En esta sección se verán todas las funciones de CFIX relacionadas con el manejo de ESSENTIA.

Manejo de versiones

FindSchemaVersion

```
schema FindSchemaVersion(char *name, long version);
```

Obtiene el descriptor de la versión indicada del esquema *name*. Si el parámetro `version` es la constante `CURRENT_VERSION`, se obtiene el descriptor de la versión corriente del esquema.

OpenSchemaVersion

```
schema OpenSchemaVersion(char *name, UShort mode, long
version);
```

Esta función abre el esquema de la versión pasada como parámetro. En caso de pasarse como parámetro la constante `CURRENT_VERSION` se abrirá la última. Si la operación tiene éxito, la función `OpenSchemaVersion` devuelve el descriptor del esquema, lo coloca en la lista de "esquemas activos" y lo marca como esquema corriente.

SchemaVersion

```
long SchemaVersion(schema sch);
```

Devuelve el número de versión asociado al descriptor del esquema. Si el descriptor pertenece a la versión actual, entonces devuelve `CURRENT_VERSION`.

VersionNumber

```
long VersionNumber(char *name, DATE d);
```

Esta función devuelve el número de versión del esquema que estaba activa en la fecha *d*, o la constante `CURRENT_VERSION` si no se creó otra versión luego de la fecha indicada.

Manejo de Transacciones

BeginTransaction

```
void BeginTransaction();
```

Ver el *Manual del Programador IDEAFIX*.

DoSyncTransaction

```
DoSyncTransaction { }
```

Es equivalente a `DoTransaction` pero ejecuta un `EndSyncTransaction` en lugar de un `EndTransaction` cada ciclo.

DoTransaction

```
DoTransaction { }
```

Esta sentencia permite al usuario reintentar la transacción encerrada entre llaves hasta que la misma termine exitosamente.

Ejemplo

```
/*
dentro del DoTransaction va el cuerpo de la transacción
que se repetirá hasta que la misma termine en forma
exitosa.
*/
int NroIntento = 0;
DoTransaction {
GetRecord(...);
GetRecord(...);
if (...)
PutRecord(...);
else
PutRecord(...);
/*
el programador decide no esperar más y sale
del ciclo
```

```
*/
if (!TransOk() && ++NroIntento>20) {
AbortTrans();
...
break;
}
```

EndSyncTransaction

```
bool EndSyncTransaction();
```

La utilización de las funciones `BeginTransaction` y `EndTransaction` garantiza que se ejecutarán completamente todas las operaciones implicadas en la transacción, o que no se ejecutará ninguna. Si se desea asegurar además que la transacción quede registrada en caso de que se produzca una caída del sistema, se deberá utilizar la función `EndSyncTransaction`, ya que la misma garantiza el *sync* (bajada a disco) del log de transacciones.

EndTransaction

```
bool EndTransaction();
```

Esta función tiene un funcionamiento semejante a la `EndTransaction` de IDEAFIX. Devuelve `TRUE` si la transacción terminó correctamente y `FALSE` en caso contrario.

RollBack

```
int RollBack();
```

Esta función se encarga de deshacer las operaciones realizadas en la última transacción.

TransOk

```
bool TransOk();
```

Retorna `FALSE` o `TRUE` dependiendo de si la transacción actual ha abortado o no.

Manejo de Cursores

```
dbcursor CreateCursor(dbindex ind, int mode);
void SetCursorFrom(dbcursor cursor, ...);
void SetCursorTo(dbcursor cursor, ...);
void MoveCursorFirst(dbcursor cursor);
void MoveCursorLast(dbcursor cursor);
void SetCursorFromFld(dbcursor cursor, int nfield, char
*value);
void SetCursorToFld(dbcursor cursor, int nfield, char *value);
long FetchCursorPrev(dbcursor cursor);
```

```
long FetchCursor(dbcursor cursor);
void DeleteCursor(dbcursor cursor);
```

Estas funciones aparecen en el Manual del Programador de IDEAFIX. Debe considerarse que `CreateCursor` recibe dos parámetros adicionales:

- **`IO_CONTROL_BREAK`** : este modificador hace que `FetchCursor` y `FetchCursorPrev` indiquen el número del primer campo de la clave que cambia, comparando de izquierda a derecha. En caso de no cambiar ningún campo, devuelven cero.
- **`IO_KEY_FIELDS`** : cuando se usa este modificador, `FetchCursor` o `FetchCursorPrev`, el cursor sólo traerá los campos que pertenecen al índice por el cual fue creado el cursor.
- **`IO_READ`** : Este modificador obtiene un lock de lectura sobre el registro. Esto permite más flexibilidad, ya que es posible definir múltiples locks de lectura sobre un mismo registro.
- **`IO_PLOCK`** : Este modificador define locks persistentes a través de transacciones. Esto implica que el lock se mantendrá al finalizar la transacción y desaparecerá sólo cuando el usuario [el programador] lo libere explícitamente o el proceso termine.
- **`IO_KEEP_LOCKS`** : Este modificador (que sólo tiene sentido cuando se crea el cursor para bloquear) hace que el cursor no libere automáticamente los locks que se van creando. Posteriormente será responsabilidad del programador liberarlos, salvo que el cursor se recorra dentro de una transacción, con lo cual se liberarán al finalizarla.

CountCursor

```
long CountCursor(dbcursor cursor);
```

Devuelve la cantidad de registros existentes dentro del rango por el cual fue creado el cursor .

DeleteAllCursors

```
void DeleteAllCursors(schema sch);
```

Borra todos los cursores asociados al esquema sch.

FetchCursorThis

```
long FetchCursorThis(dbcursor cursor);
```

Coloca en el registro corriente de la tabla asociada al cursor el último registro leído con `FetchCursor/FetchCursorPrev`.

SetCursorFlds

```
int SetCursorFlds(dbcursor cursor, ...);
```

Recibe la lista de descriptores de campos que se desean traer al registro corriente cada vez que se ejecuta en `FetchCursor/FetchCursorPrev`, el final se indica con 0.

Ejemplo

```
SetCursorFlds(cur, TABLE_CAMPO1, TABLE_CAMPO4, 0);
```

SetCursorVFlds

```
int SetCursorVFlds(dbcursor cursor, dbfield *fld);
```

Idem a `SetCursorFlds` pero ésta recibe un vector de descriptores de campo terminado con el descriptor 0.

Ejemplo

```
dbfield flds[] = { TABLE_CAMPO1, TABLE_CAMPO4, 0 };
SetCursorVFlds(cur, flds);
```

Otras Funciones

CompleteIndex

```
int CompleteIndex(dbindex ind);
```

Para crear un índice es obligatorio, luego del `CreateIndex` (y de los AddIndField correspondientes), ejecutar `BuildIndex`, `AddKey` o `CompleteIndex`.

Usar esta nueva función tiene sentido sólo cuando la tabla (aún) no tiene registros para incluir en el índice.

DbCheckPoint

```
bool DbCheckPoint(schema scg);
```

Esta rutina ejecuta un checkpoint a pedido sobre el esquema pasado como parámetro. Si el checkpoint se pudo ejecutar en forma correcta retorna `TRUE`, de lo contrario retorna `FALSE`. Para ejecutar un checkpoint sobre el esquema corriente, podría utilizarse:

```
DbCheckPoint(CurrentSchema());
```

dbmsType

```
int dbmsType();
```

Esta rutina retorna el DBMS con el cual se está trabajando:

- **DBMS_IDEAFIX** : la aplicación está corriendo utilizando Ideafix para el acceso a la base de datos.
- **DBMS_ESSENTIA** : la aplicación está corriendo utilizando Essentia para el acceso a la base de datos.

GetRecord

Esta versión de Essentia agrega dos nuevos modificadores a esta función:

- **IO_READ** : Este modificador obtiene un lock de lectura sobre el registro. Esto permite más flexibilidad, ya que es posible definir múltiples locks de lectura sobre un mismo registro.
- **IO_PLOCK** : Este modificador define locks persistentes a través de transacciones. Esto implica que el lock se mantendrá al finalizar la transacción y desaparecerá sólo cuando el usuario [el programador] lo libere explícitamente o el proceso termine.

SetTableCache

```
int SetTableCache(dbtable tbl, long n);
```

Define un cache de `n` registros para la tabla `tbl`.

Cada vez que un `GetRecord` se ejecuta en la tabla `tbl` con parámetros `THIS_KEY` e `IO_NOT_LOCK`, Essentia buscará primero en este cache por el registro pedido, y, si no lo encontrara, entonces accedería al server. Si el dato se encuentra, se definirá como el registro corriente en la tabla y en el cache. Esto permitirá que subsecuentes `GetRecord` (con `THIS_KEY/ IO_NOT_LOCK`) encuentren el registro en el cache, evitando el acceso a Essentia.

UpdateRecord

```
void UpdateRecord(dbtable tab, ...);
```

Esta rutina permite ejecutar operaciones de actualización sobre campos de una tabla, sin necesidad de leer/grabar todo el registro.

Operaciones:

- **U_ADD** : Suma un valor determinado al campo.
- **U_SUB** : Resta un valor al campo.
- **U_MUL** : Multiplica el campo por un número.
- **U_DIV** : Divide el campo por un valor.
- **U_SET** : Establece el valor del campo.
- **U_COPY** : Copia un campo sobre otro.

Ejemplo

```
UpdateRecord(tab, TABLA_CAMPO1, U_SET, 0.0,TABLA_CAMPO2,
U_ADD, 10.0, TABLA_CAMPO3, U_COPY, TABLA_CAMPO4);
```

Esta sentencia establece el campo `TABLA_CAMPO1` con valor 0.0, al campo `TABLA_CAMPO2` se le suma el valor 10.0 y se copia el valor del campo TABLA_CAMPO4 sobre el campo TABLA_CAMPO3.

En el caso de usar campos vectorizados, es necesario indicar el rango del vector sobre el cual se desea realizar la operación.

Ejemplo

```
UpdateRecord(tab, TABLA_CAMPOV, U_MUL, 0, 10, 20.0);
```

Esta sentencia multiplica por 20.0, pero sólo para los elementos entre 0 y 10, en el campo vectorizado `TABLA_CAMPOV` .

Capítulo 10

Trucos y Consejos

La mayoría de los trucos y consejos que se desarrollarán en esta sección están orientados a minimizar la comunicación entre el server de base de datos y el cliente, ya que el tiempo consumido por la misma es importante.

La comunicación entre el server y el cliente varia básicamente en función de dos variables. La primera es la cantidad de instrucciones (*métodos*) que el cliente pide al server sean ejecutadas. La segunda es conocida con el nombre de “buffering”, lo que significa mandar al server m‡s de un método (y la información que cada uno necesita) en el mismo mensaje.

Solo son “buferizables” los métodos de los cuales el cliente no espera respuesta, que son:

```
[Put|Del|Add|Update]Record
SetCursor[From|To]
Free[Record|Table|Schema]
```

Es conveniente, en lo posible, tratar de ejecutar la mayor cantidad de métodos “buferizables” juntos, hasta que el buffer se llene o haya que ejecutar uno “no-buferizado”.

Usar Cursores

El uso de cursores es muy importante porque brinda la posibilidad de traer páginas de registros. Cada mensaje hacia el server trae un grupo de registros, en vez de uno solo como la función `GetRecord`.

La cantidad de registros que se leen un una página está dada por la longitud de la página y del registro de la tabla. Como la longitud de página es de 1024 bytes, no se recomienda su uso para registros de más de 512 bytes.

Usar IO_KEY_FIELDS

El uso del modificador `IO_KEY_FIELDS` permite traer sólo aquellos campos de la clave por los cuales se creó el cursor, minimizando así los datos transportados y maximizando la

cantidad de información que viene en cada una de la páginas leídas.

Usar *SetCursorFlds*

La rutina `SetCursorFlds` permite traer sólo aquellos campos de la tabla que se especifican, minimizando los datos transportados de registro y maximizando la cantidad de información que viene en cada una de la páginas leídas.

Usar *UpdateRecord*

La rutina `UpdateRecord` permite realizar operaciones sobre uno o más campos de una tabla sin necesidad de transmitir el registro completo.

Usar *SetTableCache*

`SetTableCache` está pensada especialmente para el acceso a aquellas tablas donde no es necesario obtener la última información grabada (Ej.: tablas de codificadores). Minimiza la cantidad de mensajes al server manteniendo en memoria los registros y haciendo una búsqueda local (en el cliente) de los mismos.

num **(9) vs.** *num* **(8)**

Existe una tradición entre los programadores IDEAFIX con respecto a los campos numéricos:

- Si se necesita un campo numérico de 3 se coloca un numérico de 4 en la base de datos porque internamente son del mismo tipo.
- Si se necesita un campo numérico de 5,6,7,8 se coloca un numérico de 9 ya que internamente también son del mismo tipo.

Este tipo de presunciones no son ciertas trabajando con Essentia, ya que los campos numéricos internamente se incrementan de a 4 dígitos. Un campo numérico de 8 dígitos no posee la misma representación interna que un numérico de 9, es mas, se necesitan 2 bytes mas para representarlo. Esto quiere decir que por la conexión viajan 2 bytes más cada vez que se opera con el registro.

Otra solución posible es utilizar el tipo INTEGER, que permite utilizar valores enteros de 32 bits.

Uso del *SetRelation*

Al utilizar esta rutina se ejecutarán varios `GetRecord` por cada lectura que se realice. Por lo tanto se debe tener *mucho* cuidado a al hora de utilizarla. Si el programador no conoce

realmente la cantidad de accesos que implica el `SetRelation` que está definiendo, puede degradar la performance debido a la gran cantidad de lecturas innecesarias.

Uso de Vectores

Se debe tener mucho cuidado cuando se definen vectores dentro de la base de datos, ya que pueden incrementar en forma no deseada la longitud de registro de la tabla, trayendo serios problemas de performance.

Transacciones (*Locks*, *PutRecord*)

Si se trabaja con Essentia como motor de base de datos, la utilización correcta del mecanismo de transacciones es de vital importancia. Las aplicaciones deben estar diseñadas teniendo esto en cuenta. Una *transacción* es una unidad de programa cuya ejecución conserva la consistencia de la base de datos. Para garantizar esto las transacciones deben ser atómicas, es decir, se ejecutan completamente todas las instrucciones implicadas en la misma o no se ejecuta ninguna. Nótese que el programador es el encargado de definir los programas de forma tal que utilicen el mecanismo de transacciones en forma correcta.

Si el programador no define las transacciones, cada función que modifique la base de datos (`PutRecord`, `DelRecord`, `UpdateRecord`, `BuildIndex`, etc.) será tomada como una transacción independiente, es decir, la biblioteca **cfix** de acceso a Essentia realizará en forma automática el `BeginTransaction` y el `EndTransaction` antes y después de la operación a ejecutar.

Esto, que generalmente no es el comportamiento deseado, implica dos desventajas: mayor cantidad de métodos que viajan a Essentia y restricción de la posibilidad de "buferizar" al máximo los mismos ya que el `BeginTransaction` y `EndTransaction` no son "buferizables".

Veamos un ejemplo en donde un simple olvido de un `BeginTransaction`/`EndTransaction` puede incrementar la cantidad de mensajes al server:

```
BeginTransaction(); // 1er. mensaje
PutRecord(...);
PutRecord(...);
PutRecord(...); // 2do mensaje
PutRecord(...); // con el EndTransaction
EndTransaction();
Total de mensajes: 2.
PutRecord(...); // 1ros. 2 mensajes (Begin/Put-End)
PutRecord(...); // 2dos. 2 mensajes (Begin/Put-End)
PutRecord(...); // 3ros. 2 mensajes (Begin/Put-End)
PutRecord(...); // 4tos. 2 mensajes (Begin/Put-End)
PutRecord(...); // 5tos. 2 mensajes (Begin/Put-End)
Total de mensajes: 10.
```

Esta diferencia en la cantidad de mensajes puede no notarse mucho cuando el cliente corre local, pero cuando se habla de una red local la diferencia puede ser notoria (sobre todo si esta

secuencia de instrucciones se encuentra dentro de un *loop*). Este efecto es aún más claro cuando se trabaja sobre una red donde un mensaje (*ping*) tarda 2 o 3 segundos.

Locks

En Essentia, para preservar la consistencia de la base de datos, los bloqueos obtenidos durante una transacción no son realmente liberados hasta el final de la misma. Esto puede hacer que programas mal diseñados, con transacciones largas y bloqueos masivos (que llenen la tabla de locks), no puedan ejecutarse.

Este tipo de transacciones deben rediseñarse para realizar bloqueos a nivel de tabla (en vez de registro) o bien ser divididas en transacciones más pequeñas.

PutRecord

Cada `PutRecord` que se realiza sobre la base de datos genera un bloqueo sobre el registro actualizado, permitiendo a Essentia mantener la consistencia lógica de las transacciones. Esto significa que las transacciones que trabajen realizando PutRecord tienen los problemas antes mencionados y se deben tener en cuenta las mismas consideraciones.

2

Drivers

Capítulo 11

Introducción a ODBC

Introducción

ODBC (Open Database Connectivity) es una interfase de programación (API) ampliamente aceptada para el acceso a bases de datos. Está basado en la especificaciones Call-Level Interfase (CLI) de X/Open y ISO/IEC para APIs para bases de datos y usa SQL (Structured Query Language) como lenguaje para el acceso a las mismas.

ODBC está diseñado para obtener máxima *interoperatibilidad* –esto es, la habilidad de una sola aplicación de acceder a diferentes sistemas de manejo de bases de datos (DBMS) con el mismo código fuente. Las aplicaciones de bases de datos llaman a funciones de la interfase ODBC, las cuales están implementadas en módulos específicos para cada base de datos llamados *drivers*. El uso de drivers aísla a las aplicaciones de llamadas a funciones específicas de cada base de datos de la misma manera que los drivers para impresoras aíslan a los programas procesadores de palabras de comandos específicos para cada impresora. Debido a que los drivers son cargados en tiempo de ejecución, un usuario solo tiene que agregar un nuevo driver para acceder a un nuevo DBMS; no es necesario recompilar o relinkear la aplicación.

Por qué fue creado ODBC?

Históricamente, las compañías usaban un único DBMS. Todos los accesos a la base de datos eran hechos mediante el front-end de ese sistema o a través de aplicaciones escritas para trabajar exclusivamente con ese sistema. Sin embargo, a medida que el uso de computadoras

creció y más software y hardware estuvo disponible, las compañías comenzaron a adquirir diferentes DBMS. Las razones fueron muchas: La gente compró lo que era más barato, lo que era más rápido, lo que ya conocían, lo último que aparecía en el mercado, lo que funcionaba mejor para un determinado tipo de aplicación. Otras razones fueron reorganizaciones y fusiones, donde departamentos que antes contaban con un único DBMS ahora contaban con varios.

El tema creció aún más en complejidad con la llegada de las computadoras personales. Estas computadoras trajeron consigo diversas herramientas para realizar consultas, análisis, y mostrar información, junto con un número de bases de datos sin costo y fáciles de usar. Desde entonces, una única organización usualmente contaba con información diseminada a través de varios desktops, servers y mini computadoras, almacenada en una variedad de bases de datos incompatibles, y accedida por un vasto número de herramientas diferentes, pocas de las cuales eran capaces de obtener toda la información disponible.

El desafío final llegó con el advenimiento de la computación client/server, cuyo objetivo es hacer el uso más eficiente de los recursos de computadoras. Computadoras personales de bajo costo (los clientes) yacen en los escritorios y proveen tanto un front-end gráfico para la información como un número de herramientas de bajo costo como hojas de cálculo, programas de diseño de esquemas (charts) y generadores de reportes. Las mini computadoras y los mainframes (los servers) contienen los DBMSs, usando su poder de operación y locación central para proveer acceso rápido y coordenado a los datos. Entonces, ¿cómo conectar el software que hace de front-end a las diferentes bases de datos?

Ante un problema similar se encontraron los vendedores independientes de software (ISVs). Los que escribían software de base de datos para mini computadoras y mainframes usualmente se veían forzados a escribir una versión de una aplicación para cada DBMS o escribir código específico para cada DBMS a la que querían acceder. Los que escribían software para computadoras personales tuvieron que escribir rutinas para acceso a los datos para cada DBMS diferente con la cual querían trabajar. Esto usualmente significaba una enorme cantidad de recursos gastados en escribir y mantener las rutinas de acceso, en vez de las aplicaciones, y las aplicaciones usualmente no vendían según su calidad sino según a que tipo de DBMS brindaban acceso.

Lo que ambos tipos de desarrolladores necesitaban era una manera de acceder datos en diferentes DBMS.

El primer grupo necesitaba una manera de mezclar datos provenientes de diferentes DBMS en una sola aplicación, mientras que el segundo grupo necesitaba esta posibilidad y una forma de escribir una única aplicación que fuera independiente del tipo de DBMS. En resumen, ambos grupos necesitaban un modo interoperable de acceder a los datos; necesitaban conectividad abierta a bases de datos.

Qué es ODBC?

Existen varios conceptos acerca de ODBC en el mundo de la computación. Para el usuario

final, es un ícono en el Panel de Control de Windows. Para el programador de aplicaciones, es una librería que contiene rutinas para el acceso de datos. Para muchos otros, es la respuesta a todos los problemas sobre acceso a bases de datos que siempre se imaginaron.

Ante todo, ODBC es una especificación para un API de bases de datos. Este API es independiente de cualquier tipo de DBMS, lenguaje de programación y sistema operativo. El API de ODBC está basado en las especificaciones CLI de X/Open y ISO/IEC. ODBC 3.0 implementa completamente estas dos especificaciones –versiones anteriores de ODBC estaban basadas en versiones preliminares de estas especificaciones pero no las implementaban totalmente- y agrega algunas funciones necesitadas usualmente por los desarrolladores de aplicaciones de bases de datos, como por ejemplo los scrollable cursors (cursores escrolables).

Las funciones en el API ODBC son implementadas por drivers específicos a cada tipo de DBMS. Las aplicaciones llaman a las funciones de estos drivers para acceder a los datos en una manera independiente al DBMS. Un Driver Manager maneja las comunicaciones entre aplicaciones y drivers.

Aunque Microsoft provee un Driver Manager para computadoras corriendo Windows NT/2000/XP y Windows 95/98/ME, ha escrito varios drivers ODBC y llama funciones ODBC desde algunas de sus aplicaciones, cualquiera puede escribir aplicaciones y drivers ODBC. De hecho, la gran mayoría de aplicaciones y drivers ODBC disponibles para computadoras corriendo Windows NT/2000/XP y Windows 95/98/ME son producidas por otras compañías aparte de Microsoft.

Es más, existen aplicaciones y drivers ODBC en Macintosh y en una gran variedad de plataformas UNIX.

Accediendo a los datos: Data Sources (Fuentes de Datos)

Un *Data Source (fuente de datos)* es simplemente el lugar de donde son extraídos los datos. Puede ser un archivo, una base de datos en particular en un DBMS, o inclusive una alimentación dinámica de datos. Por ejemplo, un data source pude ser un DBMS Oracle corriendo bajo el sistema operativo OS/2, accedido por Novell Netware, un DBMS DB2 de IBM accedido a través de un gateway, una colección de Xbase en un directorio del servidor, una base de datos Ideafix corriendo bajo unix, o un archivo local de la base de datos Microsoft Access.

Tipos de Data Sources

Existen dos tipos de data sources: machine data sources y file data sources. Aunque ambos contienen información similar acerca de la fuente de datos, difieren en la forma en que almacenan esa información. Debido a estas diferencias, son usados en formas diferentes.

Machine Data Sources

Los ***Machine Data Sources*** son guardados en el sistema con un nombre definido por el usuario. Asociado al nombre del data source se encuentra toda la información que el Driver Manager y el mismo driver necesitan para conectarse a la fuente de datos. Por ejemplo, para una base de datos Ideafix, estos datos son el nombre del esquema a utilizar y el host en donde reside la base de datos que contiene al mismo.

File Data Sources

Los File Data Sources son guardados en un archivo y permiten que la información para la conexión sea usada repetidamente por un mismo usuario o compartida entre varios usuarios. Cuando se usa un ***File Data Source*** , el Driver Manager realiza la conexión utilizando la información contenida en un archivo ***.dsn*** . Este archivo puede ser manipulado como cualquier otro archivo. Un ***File Data Source*** no tiene un nombre de fuente de datos (***Data Source Name, DSN***), como sí lo tiene un ***machine data source*** , y no está registrado para con ningún usuario o máquina.

Un ***File Data Source*** facilita el proceso de conexión, ya que el archivo ***.dsn*** contiene el string de conexión que de otro modo tendría que ser construído para luego llamar a la función **SQLDriverConnect**. Otra ventaja del archivo ***.dsn*** es que puede ser copiado a cualquier máquina, por lo que data sources idénticos pueden ser usados por varias máquinas siempre y cuando tengan instalado el driver necesario. Un ***File Data Source*** también puede ser compartido por las aplicaciones. Un ***File Data Source*** puede ser puesto como archivo compartido en una red, y ser usado simultáneamente por varias aplicaciones.

Un archivo ***.dsn*** puede también ser incompartible. Un archivo ***.dsn*** incompartible reside en una única computadora, y apunta a un machine data source. Estos archivos existen principalmente para facilitar la conversión de un ***machine data source*** a un ***file data source*** . Cuando el Driver Manager recibe la información en un archivo ***.dsn*** incompartible, se conecta al ***machine data source*** al que el archivo ***.dsn*** apunta.

Usando Data Sources

Los ***Data Sources*** son generalmente creados por el usuario final o por un técnico usando un programa llamado ***ODBC Administrator*** . El ODBC Administrator le pregunta al usuario cuál es el driver que desea usar y luego llama a ese driver. El driver muestra un dialog box con diferentes datos necesarios para conectarse a la fuente de datos que el usuario debe completar. Luego que el usuario ingresa esta información, el driver la almacena en el sistema.

Más tarde, la aplicación llama al Driver Manager y le suministra el nombre de un ***machine data source*** o el path de un archivo conteniendo un ***file data source*** . Cuando le pasa un nombre de un machine data source, el Driver Manager busca por el sistema el driver que se necesita usar para acceder a esa fuente de datos. Luego carga el driver y le pasa a éste el nombre del ***data source*** . El driver usa el nombre del ***data source*** para encontrar la

información que necesita para conectarse a la fuente de datos. Finalmente, se conecta a la fuente de datos, típicamente preguntándole antes al usuario su user ID y password, los que usualmente no son almacenados por cuestiones de seguridad.

Cuando se le pasa un ***file data source*** , el Driver Manager abre el archivo y carga el driver especificado. Si el archivo también contiene un string de conexión, el mismo le es pasado al driver. Usando la información en el string de conexión, el driver se conecta al ***data source*** . Si no se le pasó ningún string de conexión, el driver generalmente le pregunta al usuario la información necesaria.

Ejemplo de Data Source

En las computadoras corriendo Microsoft Windows NT/2000/XP o Microsoft Windows 95/98/ME, la información sobre los machine data source es almacenada en el registry. Dependiendo bajo que key del registry la información se encuentra almacenada, el ***data source*** es conocido como un ***user data source*** o un ***system data source*** . Los ***User Data Sources*** se encuentran guardados bajo el key `HKEY_CURRENT_USER` y están disponibles sólo para el usuario actual. Los ***System Data Sources*** están guardados bajo el key `HKEY_LOCAL_MACHINE` y pueden ser usados por más de un usuario en una misma máquina. También pueden ser usados por diferente servicios ubicados a lo largo del sistema, los cuales luego pueden obtener acceso al ***data source*** aún si ningún usuario se encuentra logueado en la máquina.

Supóngase que un usuario cuenta con tres ***user data sources*** : Personal e Inventario, los cuales usan un DBMS Oracle, y Sueldos que usa el DBMS Microsoft SQL Server. Los valores de las claves del registry para estos data sources serían:

```
HKEY_CURRENT_USER
SOFTWARE
ODBC
ODBC.INI
ODBC Data Sources
Personal : REG_SZ : Oracle
Inventario : REG_SZ : Oracle
Sueldos : REG_SZ : SQL Server
```

y los valores del registry para el data source Sueldos serían:

```
HKEY_CURRENT_USER
SOFTWARE
ODBC
ODBC.INI
Sueldos
Driver : REG_SZ : C:\WINDOWS\SYSTEM\SQLSRV.DLL
Server : REG_SZ : SERVER1
FastConnectOption : REG_SZ : No
UseProcForPrepare : REG_SZ : Yes
OEMTOANSI : REG_SZ : No
LastUser : REG_SZ : erik
Database : REG_SZ : Sueldos
Language : REG_SZ : English
```

Ideafix ODBC Driver

InterSoft cuenta con un driver ODBC que permite el acceso a las bases de datos Essentia e Ideafix para las plataformas Windows y Unix. A continuación se detalla su instalación.

Instalación del Ideafix ODBC Driver en Windows

El proceso aquí descripto bajo Windows XP Professional es válido también para Windows NT/2000 y Windows 95/98/ME, haciéndose notar cuando sea necesario las diferencias que puedan llegar a existir.

- Descomprimir el archivo ***psqlodbc.zip*** que contiene el archivo de instalación del driver ODBC de Ideafix ***psqlodbc.exe*** .
- Hacer doble click sobre ***psqlodbc.exe*** para iniciar el proceso de instalación.
- A continuación se verá en pantalla la ventana de instalación del driver ODBC de Ideafix. (Figura 11.1)

Figura 11.1

- Presionar el botón ***Next*** para comenzar la instalación de los componentes del driver. (Figura 11.2)

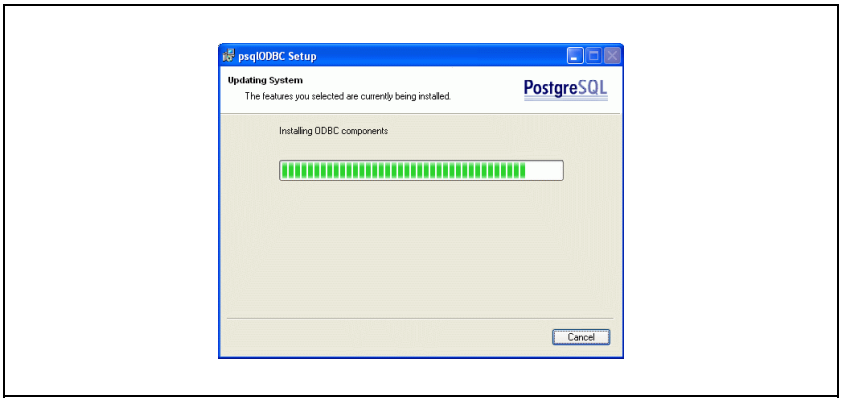


Figura 11.2

Figura 11.3

- Presionar el botón ***Finish*** para terminar la instalación. (Figura 11.3)

Configurando Data Sources para el Ideafix ODBC Driver

Para crear diferentes ***data sources*** para acceder a diferentes bases de datos y esquemas es necesario recurrir al ODBC Administrator. El mismo puede ser ubicado en ***Inicio->Panel de Control->Herramientas administrativas***, o ***Mi PC->Panel de Control*** bajo Windows NT/2000 y Windows 95/98/ME.(Figura 11.4)

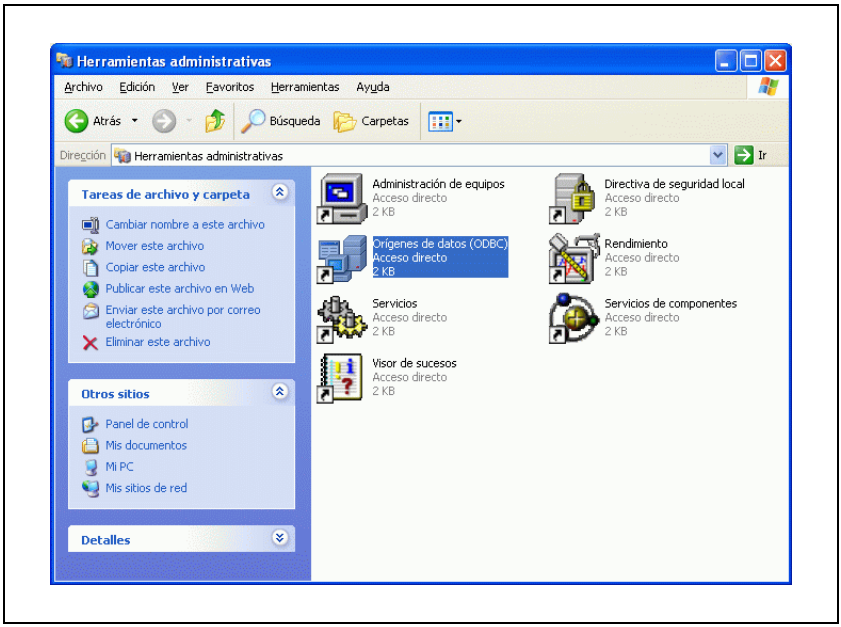


Figura 11.4

Creando User y System Data Sources

Crear un ***User DSN*** y un ***System DSN*** es muy simple, ambos casos se manejan de forma idéntica.

- Presionar el botón ***Agregar*** para crear un nuevo ***User DSN*** . (Figura 11.5)

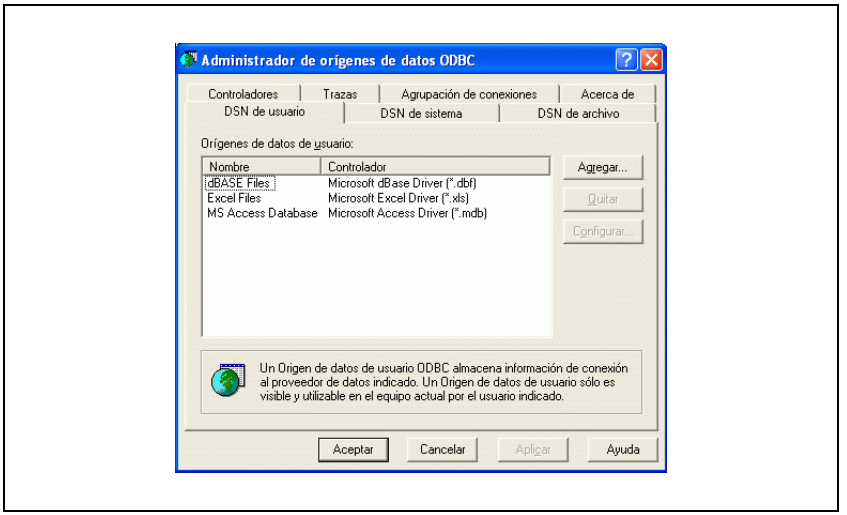


Figura 11.5

- A continuación se deberá elegir el driver a usar para con el ***User DSN*** a crear. En este caso debemos seleccionar el driver de Postgres, que se encuentra disponible bajo el nombre *PostgresSQL* y luego presionar el botón ***Finalizar*** . (Figura 11.6)

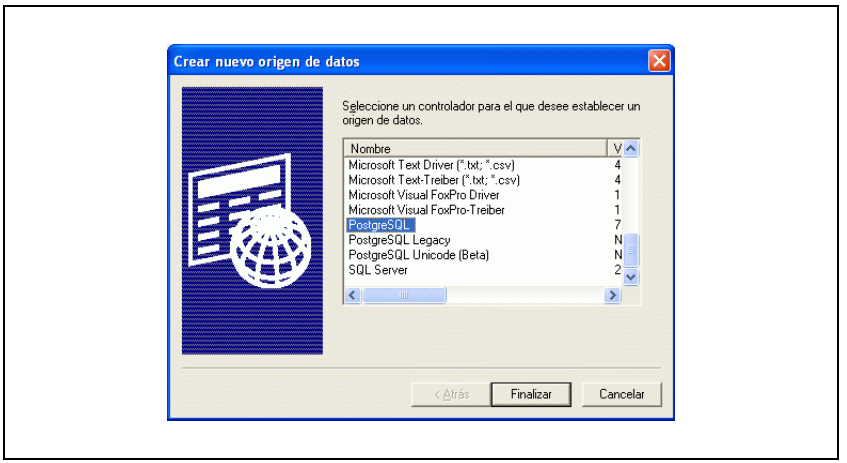


Figura 11.6

- Aparecerá un cuadro de diálogo mostrado por el Ideafix ODBC Driver, requiriendo que el usuario ingrese determinados datos (Figura 11.7), el sentido de los cuales se explica a continuación.

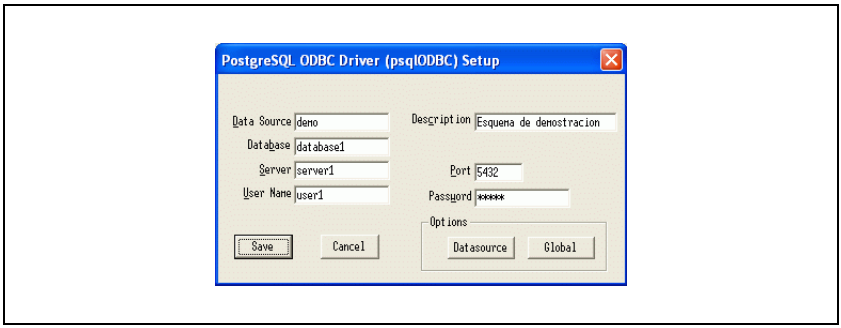


Figura 11.7

Data Source: Nombre del esquema al cual se desea acceder. Este campo es obligatorio.

Description: Una pequeña descripción acerca del data source, o del esquema etc. Este campo es optativo.

Database: Nombre de la base de datos al cual se desea conectar.

Server: Nombre o dirección IP de la máquina en la cual está ubicado el server Essentia o la base de datos Ideafix. Este campo es obligatorio.

Port: Número de puerto por el cual el servidor de base de datos espera la conexión. El valor por defecto es 5432.

User Name: Nombre del usuario de conexión a la base de datos.

Password: Contraseña del usuario de conexión a la base de datos.

Una vez presionado el botón ***Save*** en el cuadro de diálogo anterior, el proceso se da por terminado y el ODBC Administrator muestra en pantalla el nuevo ***User DSN*** que acabamos de crear.(Figura 11.8)

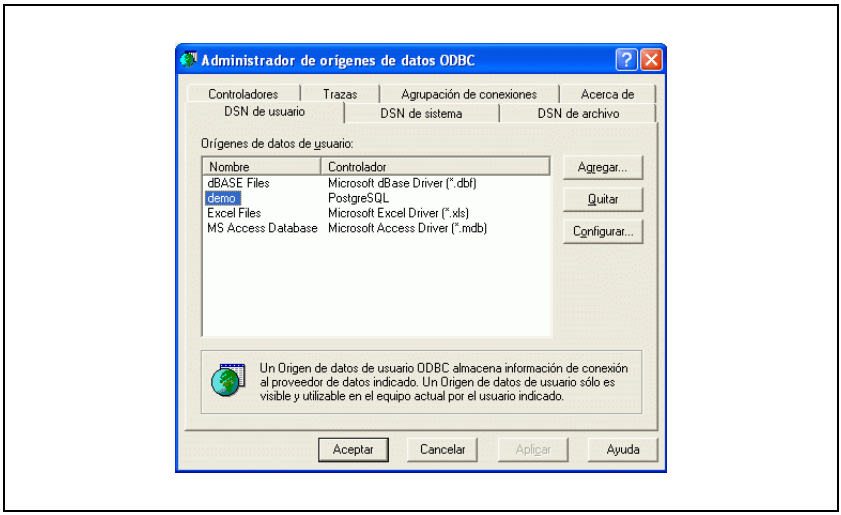


Figura 11.8

Creando File Data Sources

Los ***File Data Sources*** son creados también, como los ***User*** y ***System Data Sources*** , haciendo uso del ODBC Administrator. Estos son los pasos a seguir:

- Abrir el ODBC Administrator. Seleccionar el Tab de ***DSN de archivo***, presionar el botón ***Agregar*** y se nos presentará una lista de drivers a elegir.
- Seleccionar ***PostgresSQL*** y a continuación presionar el botón ***Avanzadas*** .
- Aquí es necesario agregar una línea de texto que tiene el siguiente formato(Figura 11.9):

Database=<nombre_del_esquema_a_utilizar> , en este caso estamos creando un ***File data source*** para acceder al esquema ***demo*** , por lo tanto agregamos la línea ***Database=database1*** . Es importante respetar las mayúsculas y minúsculas, es decir, se debe ingresar ***Database*** , y no DATABASE, o DATAbase, o similar, y el esquema debe estar completamente en letra minúscula.

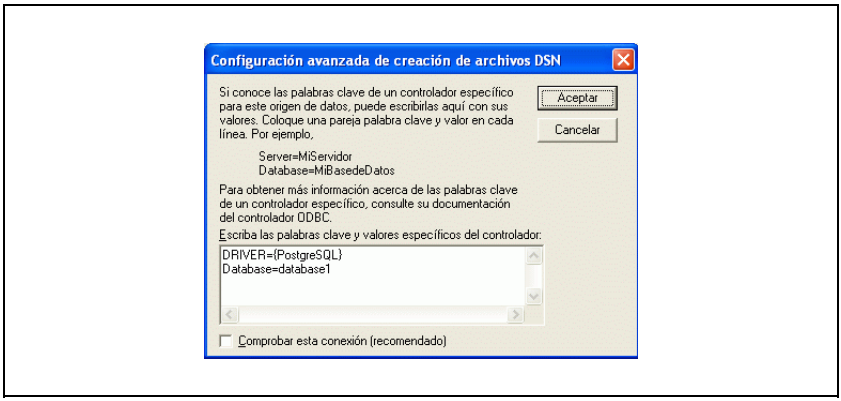


Figura 11.9

- Destildar el check box rotulado ***Comprobar esta conexión (recomendado)*** y presionar el botón ***Aceptar*** (Figura 11.9).
- Nos volverá a aparecer la ventana anterior, presionar el botón ***Siguiente*** .
- Nos aparecerá una nueva ventana en la cual debemos ingresar el nombre del archivo ***.dsn*** que contendrá esta información. En este ejemplo ***demo_filedsn*** (no es necesario agregar la extensión ***.dsn***). (Figura 11.10)

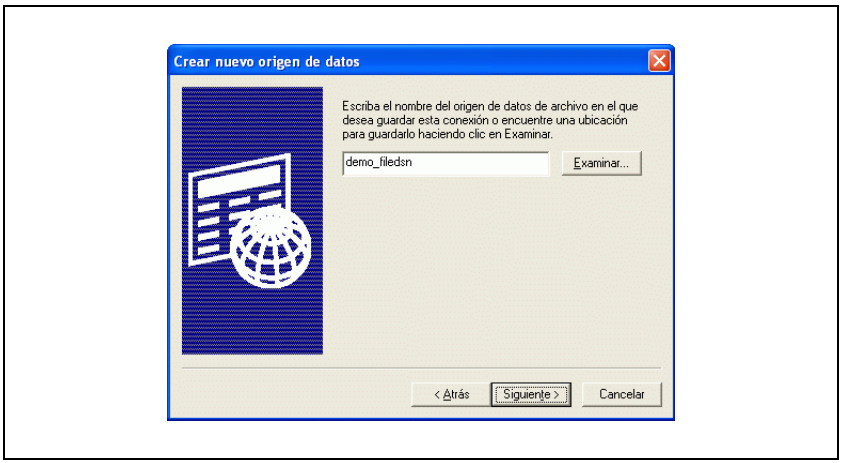


Figura 11.10

- Luego de esto nos aparecerá una ultima ventana que nos muestra el resumen de todos los datos que ingresamos anteriormente(Figura 11.11), presionando el botón ***Finalizar*** ya habremos creado nuestro nuevo ***File data source*** .

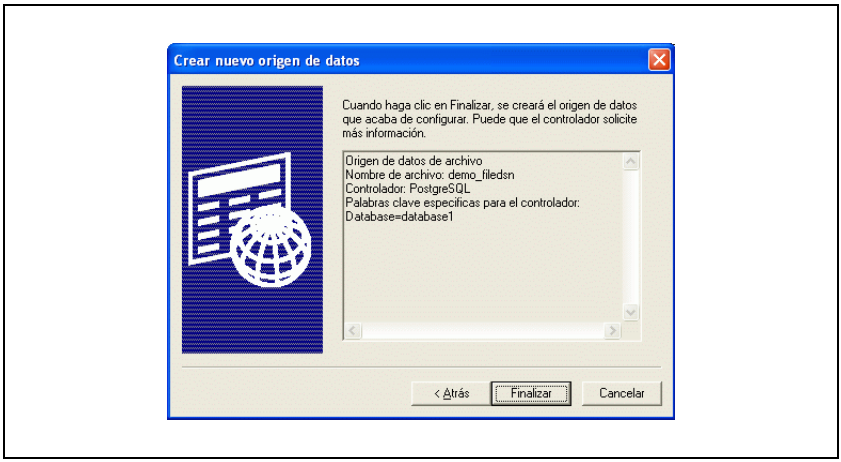


Figura 11.11

Todos los archivos ***.dsn*** son almacenados en el directorio ***C:\Archivos de Programa\Archivos Comunes\Odbc\Data Sources*** por defecto. Para cambiarlo solo es necesario presionar el botón ***Directorio*** (Figura 11.12), también es conveniente antes de crear el primer ***File data source*** presionar este botón para que el ODBC Administrator guarde en el ***registry*** el directorio por defecto y así el Driver Manager sepa donde encontrar los archivos ***.dsn*** .

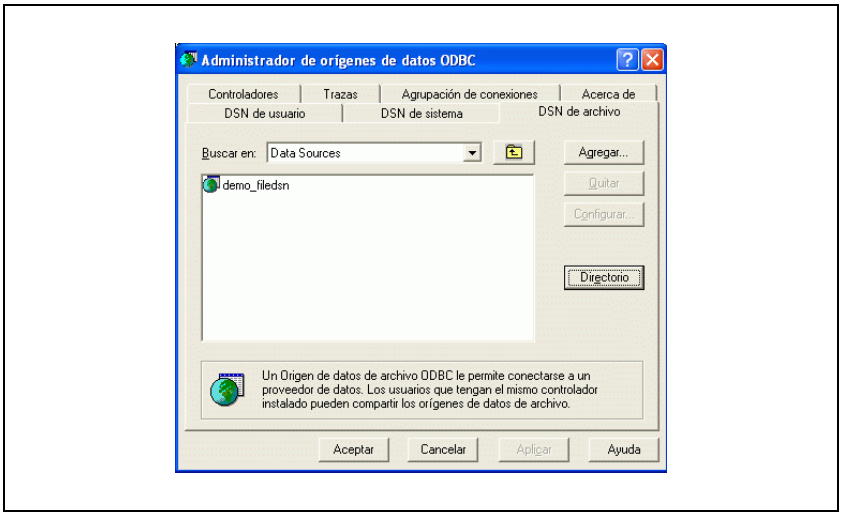


Figura 11.12

Ejemplos de usos del Ideafix ODBC Driver

A continuación se detallan algunos de los programas y entornos de desarrollo con los cuales es posible utilizar el ***Ideafix ODBC Driver*** y ejemplos de como utilizar algunos de ellos:

- Excel/Microsoft Query
- Access
- Seagate Crystal Reports
- Visual Basic
- Visual C++
- Visual Fox Pro
- PowerBuilder

etc.

Usando Microsoft Query

El ***Microsoft Query*** nos permite realizar cualquier tipo de consultas SQL sobre cualquier base de datos ***Essentia*** e ***Ideafix*** mediante el ***Ideafix ODBC Driver*** .

Existen dos formas básicas de realizar consultas utilizando el MSQuery: utilizando el asistente para consultas que proporciona el MSQuery o escribiéndolas uno mismo en SQL. La diferencia no es mucha y el asistente se explica por sí solo, por lo tanto a continuación se detalla como realizar consultas manualmente.

- Lo primero es crear una nueva consulta. Para eso existen dos caminos: Ir hacia el menú ***Archivo*** y elegir ***Nuevo*** , o presionar el primer botón de la extrema izquierda de la barra de herramientas del MSQuery(Figura 11.13).

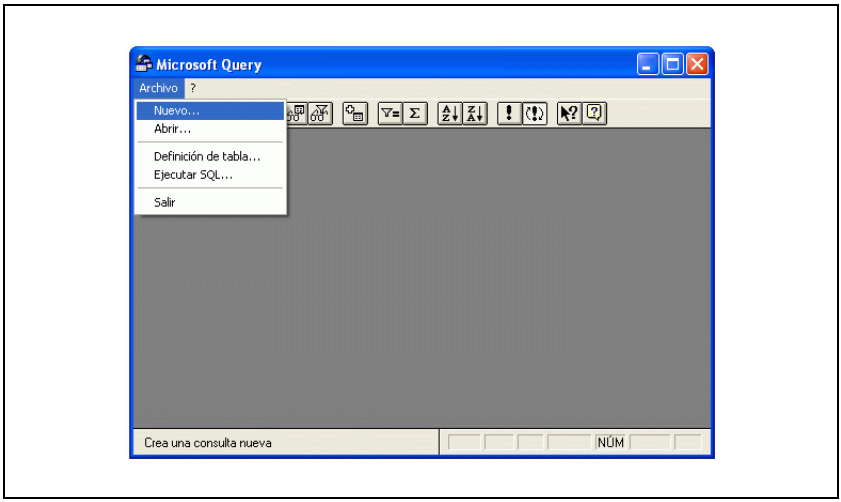


Figura 11.13

- A continuación es necesario elegir un ***file data source*** de entre la lista que el ***MSQuery*** nos presenta. En nuestro caso pretendemos acceder al esquema ***demo*** y nuestro ***file data source name*** es ***demo_filedsn*** . Seleccionamos el mismo y presionamos el botón ***Aceptar*** . Cabe remarcar que debido a que no estamos usando el Asistente para consultas, el checkbox que se encuentra en la parte inferior de la pantalla deberemos desactivarlo(Figura 11.14).

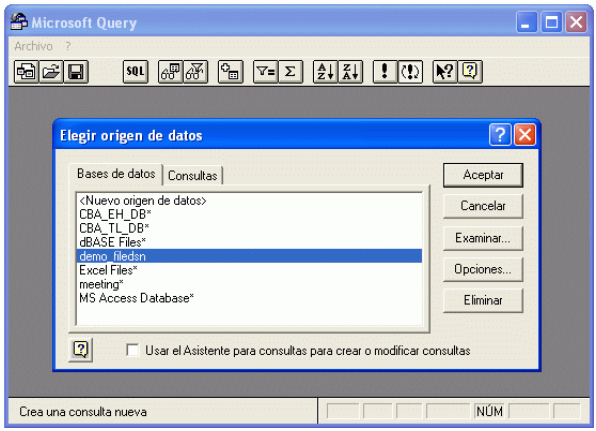


Figura 11.14

- Una vez elegido nuestro ***file data source*** , se nos presenta una caja de diálogo en donde debemos ingresar el nombre de la base de datos, el nombre o dirección IP del servidor, el nombre de usuario y password para acceder a la base de datos (Figura 11.15).

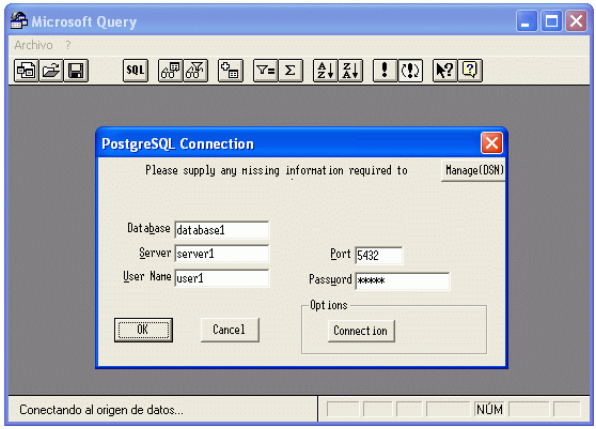


Figura 11.15

- Una vez ingresados correctamente los datos requeridos, la conexión ya está realizada con la base de datos. El MSQuery automáticamente nos presentará una lista de todas las tablas disponibles en el esquema que estamos utilizando (Figura 11.16). A partir de aquí existen dos caminos para realizar la consulta:

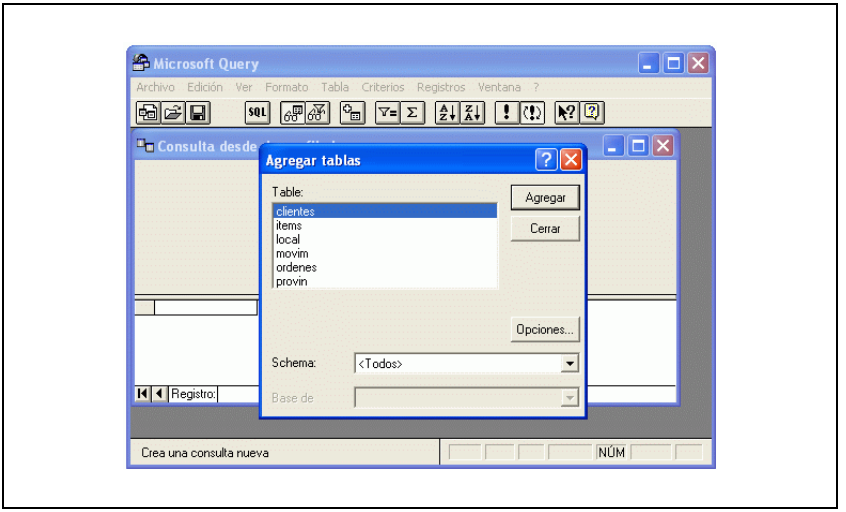


Figura 11.16

1.Escribir directamente la sentencia SQL que deseamos ejecutar:

Para realizar nuestra consulta de esta manera se deben seguir los siguientes pasos:

- En la lista de tablas, presionar el botón ***Cerrar*** .
- Presionar el botón ***SQL*** de la barra de herramientas rotulado.
- En la caja de diálogo que se nos presenta, introducir la sentencia SQL que se desea ejecutar, por ejemplo: ***SELECT * FROM clientes*** y presionar el botón ***Aceptar*** (Figura 11.17).

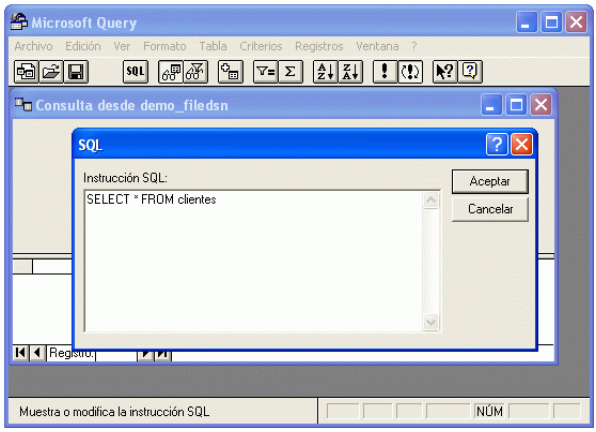


Figura 11.17

En este momento la consulta es enviada por el Ideafix ODBC Driver al IQL Server quien será el encargado de llevarla a cabo y devolver los resultados de la misma al driver.

- Luego de un breve lapso de tiempo (dependiendo de la cantidad de información que resulta de nuestra consulta) , el MSQuery nos mostrará la información solicitada (Figura 11.18)

Utilizando las barras de desplazamiento podemos movernos por toda la grilla donde reside la información, o también presionando los botones adyacentes a la ventana que contiene el texto *Registro* en la parte inferior de la extrema izquierda de la ventana podemos adelantarnos o retroceder de a un registro por vez.

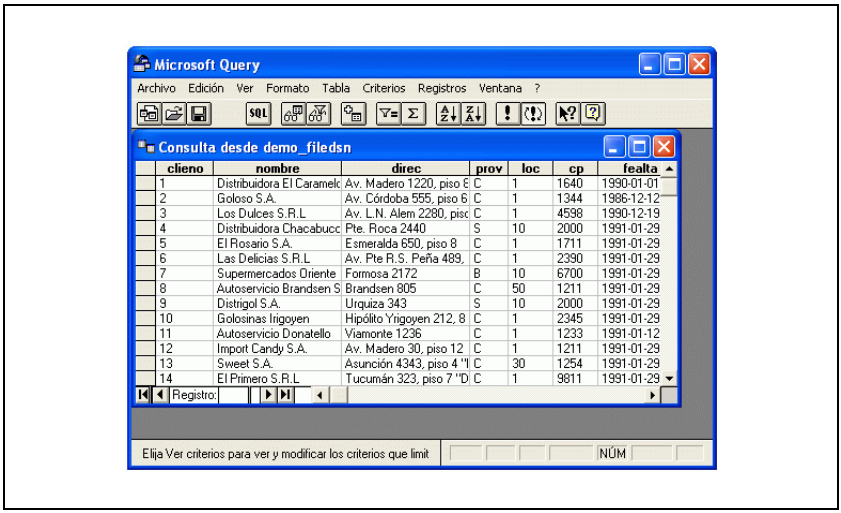


Figura 11.18

Para realizar más consultas de este tipo solo repetimos los pasos detallados anteriormente.

2.Realizar una consulta "gráficamente":

Esto se refiere a la utilización de botones y menúes en una forma mas intuitiva, donde se necesitan conocer nociones mínimas de la lógica del lenguaje SQL.

- En la lista de tablas, seleccionamos, haciendo un doble-click sobre su nombre, las tablas de las cuales pretendemos obtener datos, en nuestro ejemplo *clientes* y *ordenes*.

El MSQuery nos mostrará una ventana por cada tabla que seleccionemos, conteniendo la lista de los campos pertenecientes a la misma. Además se presenta el símbolo * que referencia al conjunto de todos los campos de esa tabla. El campo clave primario de esa tabla estará resaltado en negrita, y si las tablas seleccionadas poseen JOINs que las vinculan, los campos a través de los cuales se da el JOIN también estarán escritos en negrita y de ellos partirá una línea recta conectándolos con el campo correspondiente de la otra tabla o tablas.

Haciendo click sobre el nombre de cualquier de los campos el MSQuery mostrará la información correspondiente a ese campo contenida en la tabla (equivalente a ***SELECT <campo> from <tabla>***).

Si clickeamos en el símbolo * el MSQuery obtendrá de una sola vez, todos los datos correspondientes a todos los campos de la tabla (equivalente a ***SELECT * from <tabla>***)

Veamos a un ejemplo práctico:

Supongamos que se desea obtener el número de cliente y el nombre de todos aquellos clientes que posean ordenes con un valor mayor a $1000.

- El número de cliente está representado por el campo *clieno* en la tabla *clientes*, hacemos click en el mismo y el MSQuery nos traerá los datos correspondientes a ese campo.

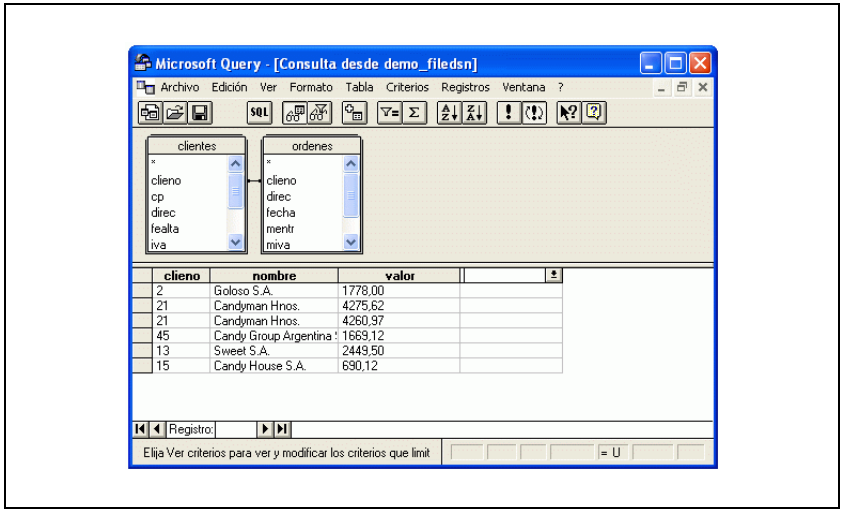


Figura 11.19

- Hacemos lo mismo con el campo *nombre* de la tabla *clientes* y también con el campo *valor* de la tabla *ordenes* (Figura 11.19).

Una vez que ya tenemos en la grilla los datos que deseamos obtener, es momento de especificar el criterio de nuestra consulta, es decir, queremos restringir todos los datos que aparecen en la grilla a aquellos que posean ordenes mayores a $1000. Para lograrlo hacemos lo siguiente:

- Hacemos click en el menú ***Criterios->Agregar criterios*** .
- Nos aparecerá una ventana en donde podemos seleccionar todas los filtros que usualmente usamos en lenguaje SQL con directivas como ***WHERE*** , ***GROUP BY*** , etc. pero presentados de una forma mucho más intuitiva y natural (Figura 11.20).

En nuestro caso el criterio que deseamos agregar es “ ***ordenes.valor sea mayor que 1000*** ”. Lo hacemos de la siguiente manera:

- En la lista bajo el nombre ***Campo*** seleccionamos el campo que nos interesa, ***ordenes .valor*** .
- En ***Operador*** seleccionamos ***es mayor que*** .
- Y finalmente, en *Valor*, ingresamos 1000.

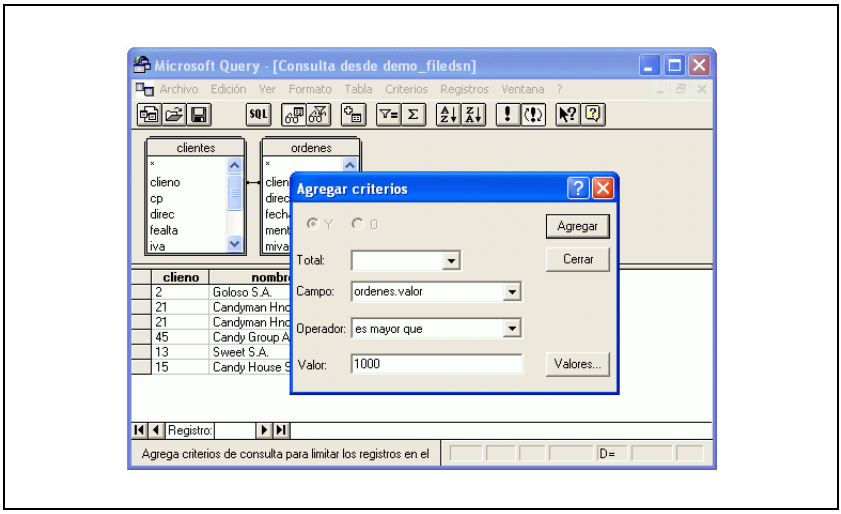


Figura 11.20

De esta manera nuestro criterio nos queda armado (Figura 11.20), presionamos el botón ***Agregar*** y ya la información que nos muestre el MSQuery estará sujeta al criterio de selección que acabamos de agregar (Figura 11.21).

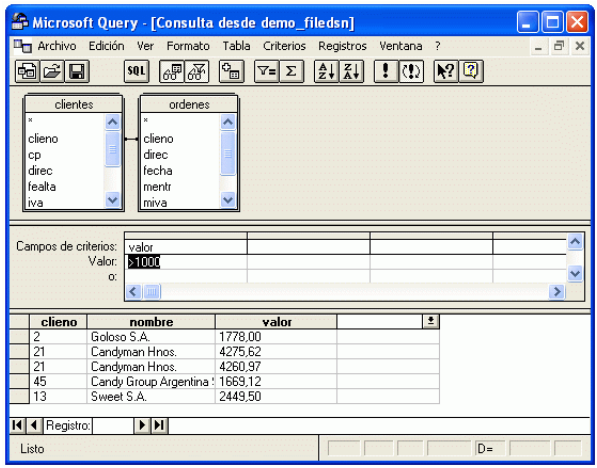


Figura 11.21

Una vez terminados estos pasos, podemos también presionar sobre el botón ***SQL*** en la barra de herramientas y el MSQuery nos mostrará la sentencia SQL que acabamos de generar "gráficamente".

Guardando las consultas:

Una vez que hayamos realizado nuestra consulta, ya sea "gráficamente" o escribiéndola en lenguaje SQL, tenemos la posibilidad de guardarla en un archivo en disco para luego poder recuperarla y utilizarla tantas veces como queramos sin pasar por todo el proceso de creación de la misma.

La forma de hacerlo es muy sencilla, solo es necesario una vez realizada la consulta pretendida, dirigirse al menú ***Archivo->Guardar*** o ***Archivo->Guardar Como*** , elegir el nombre del archivo y la consulta se grabará en un archivo con extensión ***.dqy,*** archivo este que luego podrá ser cargado con el mismo MSQuery para ejecutar la consulta nuevamente.

Usando el Microsoft Excel

El Microsoft Excel utiliza el MSQuery para importar datos desde bases de datos a través de ODBC.

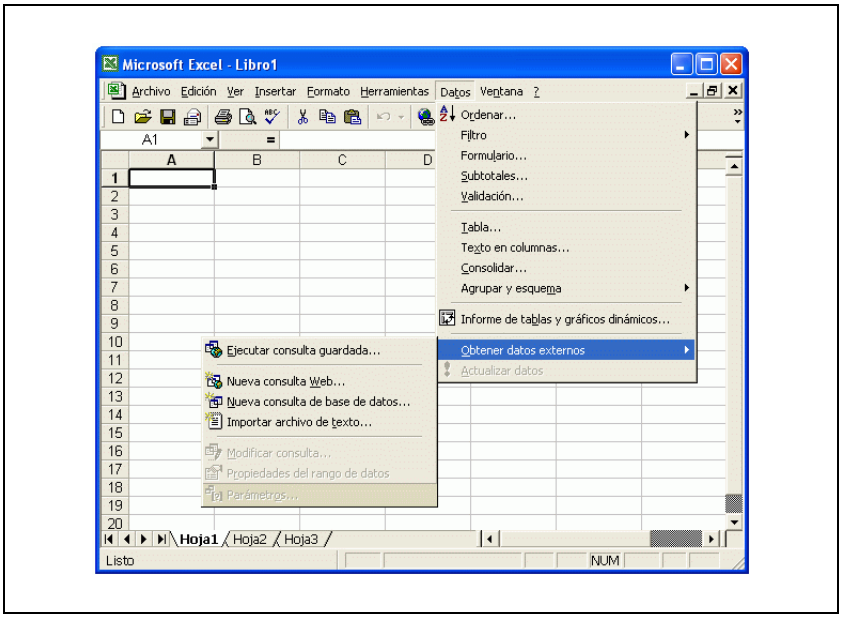


Figura 11.22

Para poder llevar a cabo esta operación, tendremos que dirigirnos al menú ***Datos->Obtener datos externos*** . Se desplegará otro menú, aquí podemos tomar principalmente dos caminos (Figura 11.22):

- ***Ejecutar consulta guardada*** : Excel nos presentará una ventana de la cual podremos elegir un archivo .dqy con una consulta previamente grabada, la cual ejecutará utilizando el MSQuery y luego transferirá los datos a la planilla.
- ***Nueva consulta de base de datos*** : Excel nos dejará seleccionar una fuente de datos y a continuación ejecutará el MSQuery para que realicemos nuestra consulta.

Lo más conveniente la primera vez que creemos una planilla con información importada de alguna base de datos será seleccionar *Nueva consulta de base de datos*, utilizar el MSQuery para crear la consulta, y luego grabarla en un archivo .dqy, de esta manera la próxima vez que utilicemos la planilla y queramos actualizar los datos importados, solo necesitaremos seleccionar *Ejecutar consulta guardada* indicando el nombre del archivo donde previamente grabamos la consulta.

Usando el Microsoft Access

Es posible crear tablas en Microsoft Access importando datos de otras bases de datos y también vincular tablas existentes en otras bases de datos dentro de una base Access.

Una vez abierta una base de datos Access existente o una nueva, presionando el botón *Nuevo* nos aparecerá una nueva ventana de la cual podemos seleccionar varias opciones (Figura 11.23). Las que nos interesan son:

- *Importar Tabla:* Nos permite importar datos de una tabla vía ODBC y volcarlos en una tabla local de Access.
- *Vincular Tabla:* Nos permite vincular toda una tabla encontrada en otra base de datos vía ODBC a nuestra base Access.

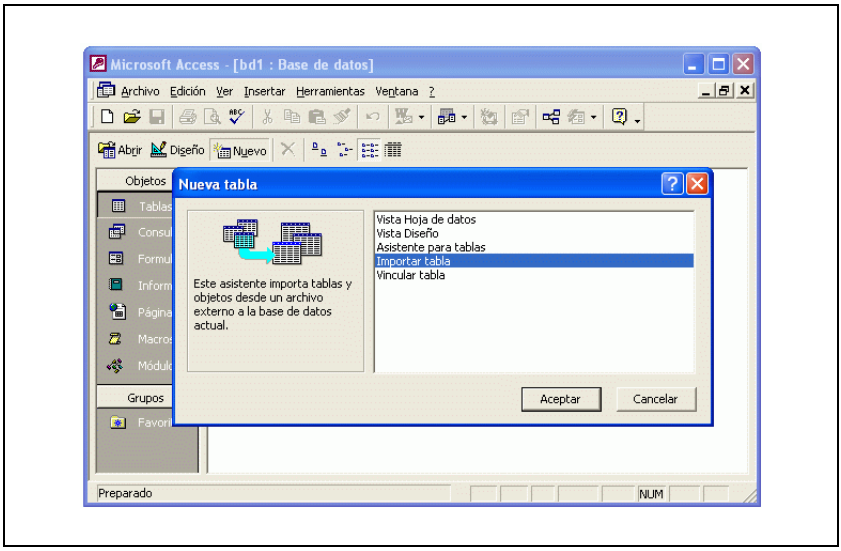


Figura 11.23

La diferencia entre importar y vincular está en que la importación de datos es estática, se obtienen los datos vía ODBC, y se guardan en una tabla local y no existe de aquí en adelante ninguna relación establecida con el data source de donde se obtuvieron los datos.

En cambio, al vincular una tabla, estamos creando un vínculo perpetuo con el ***data source*** , es decir, cada vez que se requieran los datos de esa tabla vinculada, ***Access*** se comunicará vía ODBC con el ***data source*** especificado y transferirá los datos.

El procedimiento para ambos casos es idéntico, por eso mostraremos solamente como importar una tabla.

- Seleccionamos *Importar Tabla* y presionamos el botón *Aceptar* .
- En la siguiente ventana, seleccionamos en ***Tipo de Archivo*** la opción ***ODBC Databases*** (Figura 11.24).

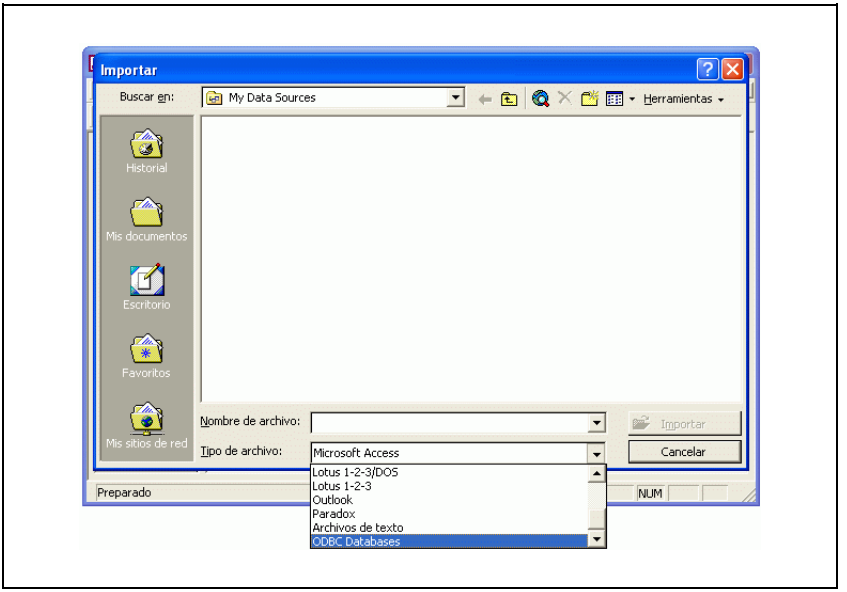


Figura 11.24

- Luego de seleccionar el ***Data Source*** , nos aparecerá una lista de todas las tablas disponibles, seleccionamos todas las tablas que queremos importar y presionamos el botón ***Aceptar*** (Figura 11.25).

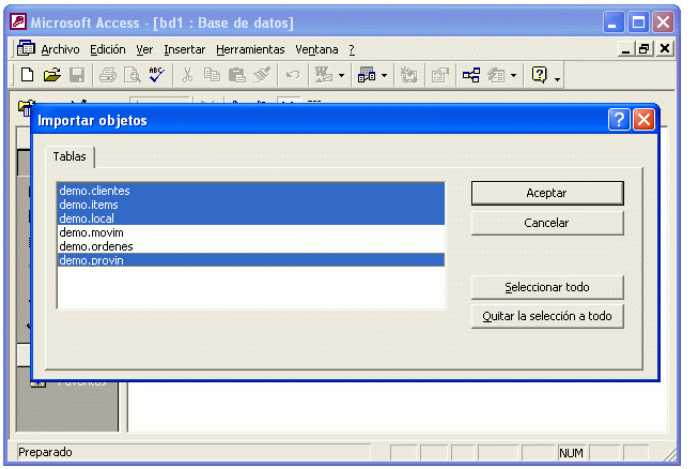


Figura 11.25

Access transferirá los datos desde la base de datos vía ODBC y los mostrará en la base de datos local Access en la que estamos trabajando.

Capítulo 12

Traceo de llamadas ODBC

Bajo Windows 95/98/ME y NT/200/XP el Driver Manager de Microsoft provee una opción de *traceo de llamadas ODBC* que resulta muy útil al momento de determinar la fuente de cualquier tipo de problema o error que se tenga con un driver ODBC, cualquiera sea éste.

Este *traceo de llamadas ODBC* consiste en almacenar en un archivo, configurable por el usuario, todas las llamadas que se hacen al Driver Manager (y que éste en última instancia hace al driver que se esté utilizando) junto con sus parámetros y códigos de retorno.

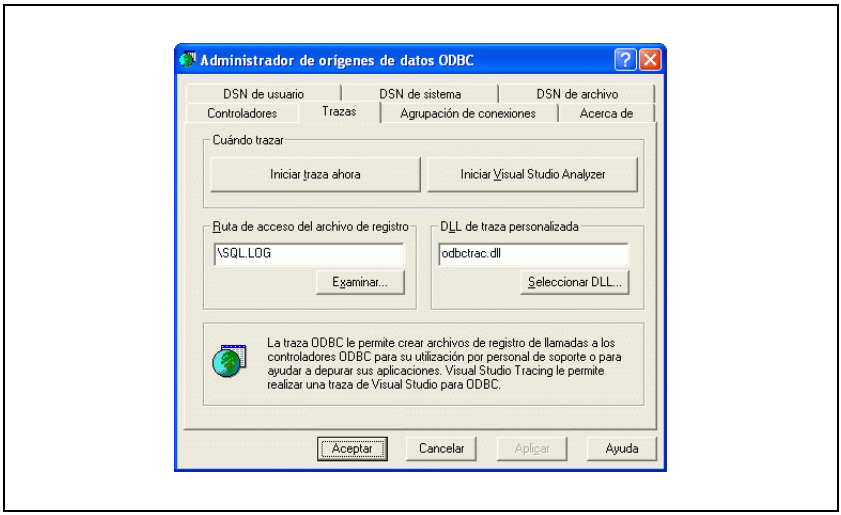


Figura 12.1

Esta facilidad se activa desde el ***Administrador de ODBC*** , en la solapa titulada ***Trazas*** . A continuación se detallan las opciones disponibles y su significado:

- ***Iniciar traza ahora*** *:* Activa el traceo dinámico el cual es llevado a cabo mientras la ventana del Administrador de ODBC permanezca abierta. El traceo dinámico puede ser activado ya sea que una conexión haya sido hecha o no. Luego de que es clickeado, el botón de ***Iniciar traza ahora*** es reemplazado por el botón de ***Detener traza ahora*** . Cuando el botón ***Detener traza ahora*** es clickeado se desactiva el traceo dinámico.
- ***Iniciar Visual Studio Analyzer*** : Habilita el Visual Studio Analyzer, que permanece habilitado hasta que se hace clic en ***Detener Visual Studio Analyzer*** . Visual Studio Analyzer es una herramienta que puede utilizar para depurar y analizar una aplicación distribuida.
- ***Ruta de acceso del archivo de registro*** *:* Muestra el nombre de directorio y el nombre de archivo donde la información de traceo será guardada. Es posible especificar un archivo nuevo con solo escribir su nombre de archivo y directorio, o seleccionando el botón rotulado ***Examinar*** . Este botón permite seleccionar el archivo al ir navegando por los directorios de la máquina.
- ***DLL de traza personalizada*** : Este control permite seleccionar una DLL de traceo que no sea ***odbctrac.dll*** .

El archivo ***odbctrac.dll*** forma parte del ***ODBC SDK*** y puede ser reemplazado por otro ***DLL*** que el usuario elija. Ingrese el nombre de directorio y el nombre de archivo de la DLL a usar, o haga click en el botón rotulado ***Seleccionar DLL*** para navegar por los directorios y buscar la ***DLL*** .

Cuando una ***DLL*** nueva es seleccionada, su nombre aparece en la ventana de texto del control ***DLL de traza personalizada*** .

- ***Aceptar*** : Acepta los cambios realizados al traceo y cierra la ventana del ODBC Administrator.
- ***Cancelar*** *:* Cierra la ventana del ODBC Administrator sin aceptar los cambios al traceo.
- ***Aplicar*** : Acepta cualquier cambio que se haya hecho a las opciones de traceo sin cerrar la ventana del Administrador de ODBC. El botón ***Aplicar*** se encuentra desactivado si no se han producido cambios a la configuración.
- ***Ayuda*** *:* Muestra una pantalla de ayuda

3

Apéndices

Apéndice A

Mensajes de Error

En este capítulo se detallan los mensajes de error del sistema, juntamente con sus causas más comunes y soluciones.

- **El archivo 'file' con extensión 'ext' no tiene path asociado en el archivo de distribución 'arch’.**

Causa:

Este mensaje aparece cuando existe un archivo físico, de nombre 'file' y extensión 'ext', que pertenece a una de las tablas de uno de los esquemas atendidos por el server que no se encuentra compredido dentro de las líneas del archivo de configuración de la distribución física llamado 'arch'.

Solución:

El nombre del archivo del mensaje está asociado al nombre lógico de la tabla que posee el problema, chequear que esa tabla esté comprendida en alguna de las líneas del archivo de distribución física.

- **El último directorio debe tener tamaño nulo, nombre lógico = 'tabla', lista de directorios = 'dir1:nbytes,...'.**

Causa:

Este mensaje aparece cuando, en caso de estar trabajando con tablas de tipo Ranged, el último directorio tiene especificado la cantidad de bytes.

Solución:

No colocar el parámetro *cantidad de bytes* sobre el último directorio.

- **En el archivo de configuración '...' la línea 'n': error de sintaxis o valor nulo no permitido.**

Causa:

Este mensaje aparece cuando la línea 'n' del archivo de distribución física tiene error de sintaxis, o cuando se le establece un valor nulo a un parámetro que no lo permite.

Solución:

Chequear el archivo de descripción de distribución física, colocando en editor el modo de despliegue de caracteres de control, y verificar que cada una de las líneas cumplan la sintaxis de cada uno de los tipos de tablas.

- **No se pudo grabar el archivo de configuración '...'.**

Causa:

Este mensaje indica que hubo problemas al intentar grabar el archivo de descripción de la distribución física.

Solución:

Se debe chequear la ubicación y los permisos que posee dicho archivo.

- **No se pudo leer el archivo de configuración de distribución.**

Causa:

La razón es que el archivo es inválido o la entrada *DiskConfiguration* no existe dentro del archivo '.sv', o bien el archivo indicado es inválido o no posee permisos de lectura.

Solución:

Chequear el parámetro '*DiskConfiguration*' dentro del archivo de configuración '.sv'.

- **Directorio no encontrado en archivo de configuración de disco, pattern 'esquema.tablas' directorio '/dir'.**

Causa:

Existe un directorio dentro del archivo de descripción de la distribución física llamado '/dir' que es inválido (no existe o no es un directorio) para las tablas 'esquemas.tablas'.

Solución:

Chequear que el directorio '/dir' para la entrada 'esquema.tablas' dentro de la distribución física exista y sea un directorio.

- **En archivo de configuración de disco, la entrada 'n'con pattern 'pattern' no pertenece a ningun esquema.**

Causa:

Este mensaje aparece cuando la línea 'n' dentro del archivo de distribución física posee un *pattern* que no pertenece a ningún esquema de los atendidos por el server. Los patterns usados para la especifición de tipo de tablas tienen que tener el siguiente formato:

```
esquema, expressión_regular_de_tablas
```

Ejemplo:

```
aurus.saldos # tabla saldos
aurus.cbtes # tabla cbtes
aurus.mov* # todas las tablas que impiezan con 'mov'
aurus.* # todas las tablas de aurus
```

Este mensaje evita entradas genéricas que no esten asociadas a ningún esquema, como por ejemplo: '*'.

Solución:

Verificar que el pattern de la línea 'n' dentro del archivo de distribución física, pertenezca a alguno de los esquemas del server.

- **Las tablas de estructuras solo pueden ser de tipo packed, chequee la entrada dentro del archivo de configuración de file para la tabla de estructura '$tabla'.**

Causa:

Este mensaje aparece cuando se declara una tabla de tipo no permitido. Como se mencionó anteriormente todas la tablas de definición del esquema deben ser de tipo packed.

Solución:

Chequear dentro del archivo de la distribución física que todas las tablas de estructuras sean de tipo packed.

Se recomienda colocar como primera línea para un esquema dentro de la distribución física la siguiente línea:

```
schema.$*<TAB>PACKED<TAB>/dir
```

Lo cual garantizará que todas las tablas de estructuras sean de tipo packed.

- **En archivo de configuración de la distribución física, 2147483646 bytes es la máxima cantidad de bytes permitidos para una fragmentación.**

Causa:

Este mensaje aparece cuando se pide una longitud de fragmentación demasiado grande. La longitud máxima no puede superar los 2147483646 bytes (aproximadamente 2 Gb).

Solución:

Se debe encontrar la entrada dentro del archivo de configuración que describe esta partición y bajar la cantidad de bytes a menos de 2 Gb.

- **En archivo de configuración de la distribución física, 'c' no es una unidad permitida, las unidades permitidas son: 'K' para Kb., 'M' para Mb. y 'G' para Gb.**

Causa:

La unidad especificada no existe. Según se vio antes, las dimensiones de una partición pueden ser expresadas en kbytes, megabytes o gigabytes.

Solución:

Chequear que las dimensiones dadas a las fragmentaciones dentro de la distribución física estén expresadas en alguna de estas tres unidades.

- **La hora reportada por el sistema operativo es inconsistente.**

Causa:

El último *timestamp* de la base de datos es mayor que la fecha actual retornada por el sistema operativo.

Solución:

Chequear los timers del sistema operativo o del hardware.

- **Archivo de journal inválido, incompatible con la versión actual.**

Causa:

Essentia detectó un archivo de Journal incompatible.

Solución:

Para recuperar los datos se debe levantar la versión correcta del server.

- **No se puede leer archivo de log '...'.**

Causa:

Essentia intenta leer un archivo de log (opción -r) que no puede ser encontrado o no tiene permisos de lectura apropiados.

Solución:

Encontrar el archivo, e indicar en dónde está éste, y/o modificar sus permisos de lectura apropiadamete.

- **Número mágico inválido para archivo de log '...'.**

Causa:

Essentia intenta leer un archivo de log que fue creado con otra versión.

Solución:

Utilizar la versión correcta del server.

- **No se puede conectar con el server: '...'.**

Causa:

En medio de una transacción del tipo 2PC (two phase commit), un server intenta conectarse con su principal (master) pero falla.

- **El tamaño del cache es demasiado chico, debe ser >=64K**

Causa:

El tamaño del espacio cache reservado no es lo suficientemente grande. El mínimo permitido es 64K.

Solución:

Cambiar a 64K (o más) el espacio cache reservado.

- **Ocurrió un error inicializando tabla de lock**

Causa:

El máximo número de locks especificado en el archivo de configuración es muy grande.

Solución:

Reducir la cantidad de locks especificada en el archivo de configuración.

- **`No se pudo encontrar el directorio '...' para el archivo de journal.`**

Causa:

El directorio para el journal, indicado en el archivo '.sv', no existe o no está en el lugar indicado.

Solución:

Asegurarse de que este directorio exista y que se encuentre en el lugar indicado.

- **`No se pudo encontrar el directorio '...' para la base de datos.`**

Causa:

El directorio para la base de datos, indicado en el archivo '.sv', no existe o no está en el lugar indicado.

Solución:

Asegurarse de que este directorio exista y que se encuentre en el lugar indicado.

- **`Log de corrida generado por arquitectura endian diferente. Debe recompilar Essentia con opción DEBUG.`**

Causa:

El archivo de log de corrida que se intenta usar fue generado (opción -r) en una máquina basada en procesador con arquitectura endian diferente.

Solución:

Llamar al departameto de soporte técnico de Intersoft.

- **`Problema de instalación, el server debe correr como root.`**

Causa:

El archivo binario Essentia tiene un *uid* (identificador de usuario) incorrecto o bien un *suid* (identificador de grupo de usuarios) incorrecto.

Solución:

Ambos *uid* y *suid* deben ser root.

Apéndice B

User Servers

Introducción

La biblioteca "libus.a" provee un user-server frame-work.

Básicamente, un user-server, tiene la finalidad de agrupar o centralizar un grupo de operaciones que conforman un sistema o subsitema que mantiene su base de datos en Essentia (si bien esto último no es una condición excluyente).

La ventaja principal es que, corriendo en el mismo equipo que Essentia, brinda la posibilidad de conexión remota de su cliente para requerir cualquiera de estas operaciones y, de esta manera, reducir considerablemente el tráfico en la red y, por lo tanto, el tiempo de procesamiento.

Es importante tener en cuenta que la reducción del tiempo depende fundamentalmente de la forma en que se escriban las operaciones que el cliente pide a su user-server, es decir, que las mismas deben agrupar la mayor cantidad posible de instrucciones a ejecutar localmente sobre Essentia.

La relación client-server es 1 a 1. Cada vez que un cliente quiera conectarse, el user-server será automáticamente ejecutado.

Configuración

En el equipo donde corre el server es necesario configurar un servicio y hacer que el ***inetd*** "escuche" en él; a modo de ejemplo llamemos "aurus" al service (con el identificador unívoco 7130) donde "atenderá" el user-server "`Saurus`":

archivo /etc/service:

```
aurus 7130/tcp
```

archivo /etc/inetd.conf:

```
aurus stream tcp nowait root /ideafix/bin/saurus saurus
!e/etc/saurus.env -u 3 -f /tmp/saurus.log
```

archivo /etc/saurus.env:

En este archivo deben estar definidas las variables que `Saurus` necesita en su ambiente (en la forma ENVAR=path); éstas son, como mínimo, DATADIR y LANGUAGE, para encontrar mensajes comúnmente utilizados, y ESSENTIA y SERVERS para comunicarse con el server de base de datos correspondiente.

Si fuera necesario adjudicar a estas variables, o cualquier otra, un valor determinado para un usuario, puede crearse, en el directorio $HOME del usuario, un archivo `.saurusrc` conteniendo los valores correctos.

En el equipo del cliente bastará con agregar en el archivo /etc/service una línea exactamente igual a la del equipo del server.

Construcción de un user server

Para obtener un server de este tipo será necesario crear una clase (que defina el comportamiento del server) llamada, por ejemplo, `Saurus` y que cumplirá con los siguientes requisitos:

a) Ser derivada de ShadowObject

b) Que la primera línea en la declaración en el header, sea la macro DeclareNameSpace, que necesita los siguientes include:

<usrsrv/shadobj.h>

<ifound/ns.h>)

Incluir (al implementar sus métodos) el header que denominaremos (b1) <ifound/namespac.h>, y el correspondiente al punto (d), `.h` , y luego las macros:

```
CatalogClass(Saurus, ShadowObject) (b2)
DefineNameSpace(Saurus) (b3)
```

c) Definir las operaciones que deba realizar como métodos, de la siguiente forma:

```
void Saurus::operName(void)
```

d) Crear un header (.h), que será usado tanto por `Saurus` como por su cliente, en el que todos los servicios (operaciones) que `Saurus` brinde quedarán explícitos; de la siguiente manera:

```
#define SaurusMethods \
{ \
&Saurus::oper1, \
&Saurus::oper2, \
&Saurus::oper3, \
..... \
&Saurus::operN, \
```

```
}
enum SaurusMessage {
Saurus_oper1,
Saurus_oper2,
Saurus_oper3,
.......
Saurus_operN,
Saurus_lastmsg,
};
```

donde `Saurus_lastmsg` debe existir siempre y ser el œltimo.

El constructor default será el que se ejecute para crear a `Saurus`; es decir, al momento de recibir un mensaje DBM_ATTACH al mismo.

La función de las macros es lograr que el frame-work reconozca, con sólo linkeditar, estas clases "inesperadas" (name-spaces) y luego, ante los requerimientos de un proceso cliente, las cree y deje actuar.

Vale la pena saber que el frame-work, sin ningún server linkeditado, es el cuerpo, y se comporta, como el shadow (proceso-server que se encarga de comunicar procesos -cfix-remotos con Essentia en forma transparente).

El o los objetos (.o) obtenidos deberán ser linkeditados con:

- `libus.a` (User Server frame-work),
- `libesdb.a` (para el manejo de objetos binarios),
- `libiext.a` (funciones misceláneas),
- `libidea.a` (funciones misceláneas)

De esta forma, el ejecutable obtenido se comportará como `Saurus` y como cada una de las clases (servers) definidas en esta forma, cuando un cliente se "atache" al mismo y le efectúe requerimientos.

Un server puede generar objetos ("ShadowObject"s) a los cuales el cliente podrá requerirles operaciones. Estos objetos deberán cumplir con los puntos (a), (b1), (b3), (c) y (d).

Un server puede redefinir cualquiera de las operaciones DBM_ (y debe pasarle como parámetro "Client *", aunque no se usa).

Construcción de un cliente

Debe incluir los siguientes headers <ifound.h>, <essentia/dbconn.h>, <essentia/dbconst.h>, <essentia/rconnect.h> más el header resultante del punto (D) antes mencionado. Para linkeditar necesitará libidea.a .

Forma de obtener la conexión

El cliente tiene tres formas de obtener la conexión según donde reside el user-server:

A) user-server conviviendo con el shadow, en el service sha-man.

```
DbConnection *scon = RemoteConnection(host, NULL_STRING,
DbConnection::abortOnRun,
NULL_STRING);
if (scon->status() != ERR)
scon->connect(true);
if (scon->status() != ERR) {
// El server esta esperando requerimientos !
...
}
```

B) user-server solo o acompañado, en el servicio que sea (ej: "serviceName").

```
DbConnection *scon = RemoteConnection(host, NULL_STRING,
DbConnection::abortOnRun,
"serviceName");
if (scon->status() != ERR)
scon->connect(true);
if (scon->status() != ERR) {
// El server esta esperando requerimientos !
...
}
```

C) user-server solo o acompañado, en un servicio cuyo nombre es el de uno de ellos.

```
DbConnection *scon = DbConnection::findSchema("Saurus",
true,DbConnection::abortOnRun)
if (scon->status() != ERR) {
// El server esta esperando requerimientos !
...
}
```

El cliente necesita, de este modo, un .rsv (NameSpace) para identificar a su server.

En los casos B y C, para tener *debug*, debe crearse una clase que derive de Shadow (con el efecto colateral de poder redefinir el comportamiento de Shadow) para redefinir service() y un *main* para instanciarla en lugar de Shadow.

Una vez obtenida exitosamente la conexión, hay que "atacharse" al server deseado (recordar que varios servers pueden convivir en el mismo ejecutable, en el mismo servicio), de esta forma:

```
Int srvid = (*scon)(DBM_ATTACH) << SCHEMA
<< bool(false)
<< bool(false)
<< "Saurus";
```

Donde los tres primeros parámetros, constantes, deben ser siempre SCHEMA, false, false.

La variable `srvid` será usada, luego, para enviar mensajes al server `Saurus`.

Al terminar main() debe liberarse la conexión:

```
DbConnection::dispose(scon);
```

El server es creado cuando el cliente lo necesita, por lo cual ya posee un puntero a la conexión llamado "`scon`".

Forma de comunicación

El cliente ejecuta requerimientos de la siguiente forma:

```
Int param1;
String param2;
int result = (*scon)(srvid, DBM_OBJMETHODEXEC) << Saurus_oper2
<< param1
<< param2;
```

y recibe el resultado de la operación en *result* y *valores*, si los hay, de esta manera:

```
Int valInt;
String valStr;
(*scon) >> valInt >> valStr;
```

El server especifica el resultado de la operación a través del método "`prepare(Int)`" y envía datos, si los hay, de la siguiente manera:

```
Int valInt;
String valStr;
scon->prepare(OK);
(*scon) << valInt << valStr;
scon->reply();
```

y obtiene valores (parámetros) de esta manera

```
(*scon) >> param1 >> param2;
```

Jerarquía de objetos

ShadowObject

|

SaurusObject

| |

SaurusServer SaurusBatch SaurusInter ...

Que corresponde a la jerarquía sugerida:

ShadowObject

|

...Object

| | |

...Server ...Batch ...Inter ...

Código de Ejemplo

A continuación se presenta un simple user server llamado *test*, que guarda una cadena de caracteres en un archivo, con su correspondiente aplicación cliente.

Archivo test.h

```
// test.h
// test US header
class Prueba : public ShadowObject {
// macro
DeclareNameSpace
FILE *f;
public:
Test();
~Test();
void printthis(void);
};
// list of US methods
#define TestMethods \
{ \
&Test::printthis, \
}
```

Archivo testcom.h

```
// testcom.h
// list of method identifiers
enum TestMessage {
Test_printthis,
Test_lastmsg,
};
```

Código del server

```
// test.cc
// example of a simple US
// necessary include files
#include <usrsrv/shadobj.h>
#include <ifound/ns.h>
#include <ifound/namespac.h>
#include <ifound.h>
#include <essentia/dbconn.h>
#include <essentia/dbconst.h>
#include <essentia/rconnect.h>
// header for the US
```

```
#include <test.h>
#include <testcom.h>
// other include files
#include <stdio.h>
// Test constructor.
Test::Test()
{
f = fopen("/tmp/test.dat","w");
fprintf(f, "Welcome\n"); fflush(f);
}
// Test destructor.
Test::~Test()
{
fprintf(f, "Goodbye\n"); fflush(f);
fclose(f);
}
// main Test method
void Test::printthis(void) {
String s;
fprintf(f, "printed\n"); fflush(f);
// get string
(*scon) >> s;
fprintf(f, "%s\n", toCharPtr(s)); fflush(f);
scon->prepare(OK);
scon->reply();
}
// macros for Test
CatalogClass(Test, ShadowObject)
DefineNameSpace(Test)
```

Código del cliente

```
// testcl.cc
// client of the US test.cc
// necessary include files
#include <ifound.h>
#include <ifound/str.h>
#include <essentia/dbconn.h>
#include <essentia/rconnect.h>
#include <testcom.h>
int main(int argc, char **argv) {
startUp("testcl", argc, argv);
String myString = "This is about to be printed";
// obtain connection, must specify the name of the machine
DbConnection *scon = new RemoteConnection("idefix",
NULL_STRING,
DbConnection::abortOnRun,
"dealer");
Int result; // to store the result of the US requests
printf("Initializing connection...\n");
if (scon->status() != ERR)
scon->connect(true);
else {
printf("Creation resulted in error\n");
}
if (scon->status() != ERR) {
// attach client to server
Int srvid = (*scon) (DBM_ATTACH)
<< SCHEMA
```

```
<< bool(false)
<< bool(false)
<< String("Test");
printf("Using test sample\n");
result = (*scon) (srvid, DBM_OBJMETHODEXEC) << Test_printthis
<< myString;
}
// in case of ERRor
else printf("An error occured.\n");
// liberate connection
DbConnection::dispose(scon);
}
```

Archivo IBuild

Para compilar este ejemplo se puede utilizar el siguiente archivo (`test.ib`). Es importante notar que los dos componentes se especifican en el mismo archivo IBuild: el cliente y el user server.

```
project $ISRC(Essentia,ExtLib);
//test.ib
CPPFLAGS = $CPPFLAGS + " -I$ISRC/essentia/usrsrv/include";
LIBS = "-L$ESSENTIA/usrsrv -lus";
test {
test.cc;
}
testcl {
testcl.cc;
}
```

Para compilar, es necesario invocar ibuild de la siguiente forma

```
ib -u test
```

lo que creará dos archivos binarios, `test` y `testcl`. Para ejecutar el cliente, correr:

```
testcl
```

y el US test será ejecutado automáticamente cuando se lo requiera (ya que el sistema utiliza los archivos `inted.conf` y `services` configurados para el caso).

Un Repartidor de Cartas

La idea de este ejemplo es simular la tarea de repartir cartas de un mazo. Todos los datos necesarios sobre las cartas se guardan en el objeto Dealer en la forma de un user server.

Archivo de Header dealer.h

```
// dealer.h
// list of dealer methods
#define DealerMethods \
{ \
```

```
&Dealer::init, \
&Dealer::deal, \
&Dealer::shuffle, \
}
```

Archivo de Header dealercom.h

```
// dealercom.h
// list of method identifiers
enum DealerMessage {
Dealer_init,
Dealer_deal,
Dealer_shuffle,
Dealer_lastmsg,
};
```

Código del Server

```
// server.h
// necessary include files
#include <usrsrv/shadobj.h>
#include <ifound/ns.h>
#include <ifound/namespac.h>
#include <ifound.h>
#include <essentia/dbconn.h>
#include <essentia/dbconst.h>
#include <essentia/rconnect.h>
#include <dealer.h>
#include <dealercom.h>
// other include files
#include <stdio.h>
#include <math.h>
// class definitions
class Card {
int _rank;
int _suit;
public:
int rank();
int suit();
void setRank(int r);
void setSuit(int s);
};
// constants
#define ncards 52
#define nranks 13
// dealer user server
class Dealer : public ShadowObject {
// macro
DeclareNameSpace
Card deck [ncards];
int currentCard;
public:
void init(void);
void deal(void);
void shuffle(void);
};
// Card methods
```

```
int Card::rank() {
return _rank;
}
int Card::suit() {
return _suit;
}
void Card::setRank(int r) {
_rank = r;
}
void Card::setSuit(int s) {
_suit = s;
}
// Dealer methods
void Dealer::init(void) {
int r, s, cc;
for (s=0; s<4; s++)
for (r=0; r<nranks; r++) {
cc = s*13+r;
deck[cc].setRank(r+1);
deck[cc].setSuit(s);
}
currentCard = 0;
}
void Dealer::shuffle(void) {
Int i, j, r, randomizer;
Card aux;
// get randomizer
(*scon) >> randomizer;
for (j=0; j<randomizer; j++)
for (i=0; i<ncards; i++) {
r = random() % ncards;
aux = deck[i];
printf("Shuffling - Nro. : %d, palo : %d\n", aux.rank,
aux.suit);
deck[i] = deck[r];
deck[r] = aux;
}
currentCard = 0;
}
void Dealer::deal(void) {
currentCard++;
if ((++currentCard) >= ncards) shuffle();
// "returns" current card
scon->prepare(OK);
(*scon) << (String) (deck[currentCard]); //cast the Card to a
String
scon->reply();
}
// macros for Dealer
CatalogClass(Dealer, ShadowObject)
DefineNameSpace(Dealer)
```

Código del cliente

```
// dealer client */
#include <ifound.h>
#include <essentia/dbconn.h>
#include <essentia/rconnect.h>
#include <dealercom.h>
// main program
```

```
int main() {
Card c; // player's last card
float cardValue; // player's last card value 1..7 or 1/2
int cardRank; // player's last card rank 1..10
float score1, score2; // scores obtained
char answer; // player's decision
// connection
DbConnection *scon = RemoteConnection("idefix", NULL_STRING,
DbConnection::abortOnRun, "dealer");
int result; // result for user server requests
printf("Initializing connection...\n");
if (scon->status() != ERR)
scon->connect(true);
if (scon->status() != ERR) {
// attach client to server
int srvid = (*scon) (DBM_ATTACH)
<< SCHEMA
<< bool(false)
<< bool(false)
<< String("dealer");
printf("Initializing dealer pepe\n");
result = (*scon) (srvid, DBM_OBJMETHODEXEC) << Dealer_init;
printf("Shuffling - about to play 7 1/2\n");
result = (*scon) (srvid, DBM_OBJMETHODEXEC) << Dealer_shuffle
<< 3 ;
// player 1 plays
printf("\n\nPlayer 1\n");
bool busted = false; // not busted yet
bool hit = true; // will get a card
score1 = 0;
while (hit && !busted) {
printf("Dealing a card for player 1\n");
result = (*scon) (srvid, DBM_OBJMETHODEXEC) << Dealer_deal;
((Card) (*scon)) >> c; //cast to Card because a String is sent
cardRank = (int) (c.rank);
printf("Card : %d , score : %d\n", cardRank, score1);
if (cardRank > 7) cardValue = 0.5; else cardValue = cardRank;
score1 += cardValue;
if (score1 > 7.5) {
busted = true;
score1 = 0;
}
else {
printf("Another card, player 1 ?\n");
scanf("%c", &answer);
if (answer == 'y') hit = true; else hit = false;
}
}
printf("\nPlayer 1: score = %s\n", score1);
// player 2 plays
printf("\n\nPlayer 2\n");
busted = false; // not busted yet
hit = true; // will get a card
score2 = 0;
while (hit && !busted) {
printf("Dealing a card for player 2\n");
result = (*scon) (srvid, DBM_OBJMETHODEXEC) << Dealer_deal;
(*scon) >> c;
cardRank = (int) (c.rank);
printf("Card : %d , score : %d\n", cardRank, score2);
if (cardRank > 7) cardValue = 0.5; else cardValue = cardRank;
```

```
score2 += cardValue;
if (score2 > 7.5) {
busted = true;
score2 = 0;
}
else {
printf("Another card, player 2 ?\n");
scanf("%c", &answer);
if (answer == 'y') hit = true; else hit = false;
}
}
printf("Player 2: score = %s\n", score2);
// game result
if (score1 > score2)
printf("\nPlayer 1 wins.\n");
else if (score1 < score2)
printf("\nPlayer 2 wins.\n");
else
printf("\nIt's a tie - congratulations to nobody.\n");
}
// ERR
else print("Server not ready.\n");
}
```

Parte VI

•

IxGIF

Interface Gráfica

Sección 1 Instalación y Configuración

1

Instalación y Configuración

Capitulo 1

IxGIF

Requerimientos de Software

El Servidor

Sistema Operativo Unix / Windows NT – Plataforma válida sobre la cual estén compilados los productos de InterSoft S.A.

Ideafix 431 patch level 24 o superior (pl23 en caso de Windows NT)

Servicio de TCP/IP correctamente instalado.

Máquinas Clientes

Minimo: Procesador tipo Pentium II 300 Mhz – 64 Mb Ram.

Recomendado: Procesador tipo Pentium III 450 Mhz - 128 Mb Ram (o superior).

Sistema Operativo Windows 9.x – NT – 2000 con servicio de TCP/IP.

En caso de ser Unix, se requiere el entorno gráfico funcionando correctamente. (Gnome – Kde – etc)

Máquina Virtual Java (JRE) Versión 1.3.x o superior.

Distribuciones Disponibles

Graphic Interface se distribuye en dos diferentes formatos:

- IxGIF_Setup_NN.exe : Para instalar en Windows. (NN = versión del producto)
- IxGIF_Setup.tgz : Para instalar en Unix.

Instalación del Cliente Graphic Interface (Windows)

Ejecutar el archivo IxGIF_Setup_NN.exe (Instalación).

Esto creará el directorio "GraphicInterface" con su correspondiente estructura.

Para iniciar Graphic Interface existe un archivo ejecutable "GIFstartup.exe".

Configuración Básica

Opciones – Opciones Generales

Solapa *Conexión*

Servidor: dirección IP del host (servidor) a conectarse.

Puerto: puerto de conexión (generalmente 7140)

Obtener menu desde servidor: indica si los menues serán leídos en forma local o desde el servidor. Por default no debe estar seleccionada. Cuando los menúes son centralizados en el servidor, se estará utilizando el IxSCA (Sistema de Control de Accesos de Ideafix) y esta opción se setea automáticamente.

Obtener ambiente desde el servidor: indica si se desea transmitir el ambiente actual del usuario en el server (conjunto de variables) a la máquina local.

Obtener archivos de ayuda desde servidor: indica si los archivos de ayuda .hlp serán leídos desde el server o localmente. Si se leen desde el server deben estar ubicados en algún directorio seteado en la variable PATH. Si están en forma local, deben ser ubicados en un directorio HLP bajo el directorio de instalación del Graphic Interface.

Conexión segura con el servidor: utiliza protocolo seguro (Ej.: SSH). Por default no está seteada, porque aún no está soportada.

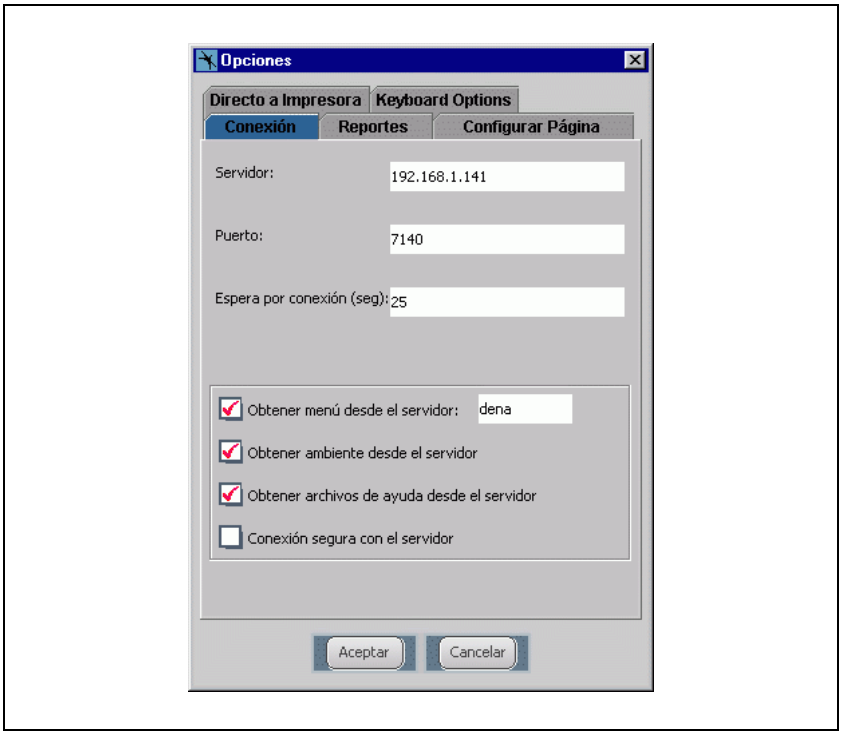


Figura 1.1

Solapa *Reportes*

Opción para que los reportes aparezcan con líneas o sin ellas.

Solapa *Configurar Página*

Opciones para la impresión local.

El resto de las solapas no se utilizan aún en esta versión.

Uso de Menúes Locales

Existe la posibilidad de instalar los menúes de la aplicación o menúes particulares localmente para luego invocar a los ejecutables desde el servidor. Para ello crear un directorio menúes bajo el directorio de instalación de Graphic Interface. También crear un ícono de acceso

directo hacia el GIFstartup.exe y en las **propiedades** pasarle como parámetro el menu principal, utilizando el parámetro ***–menu.***

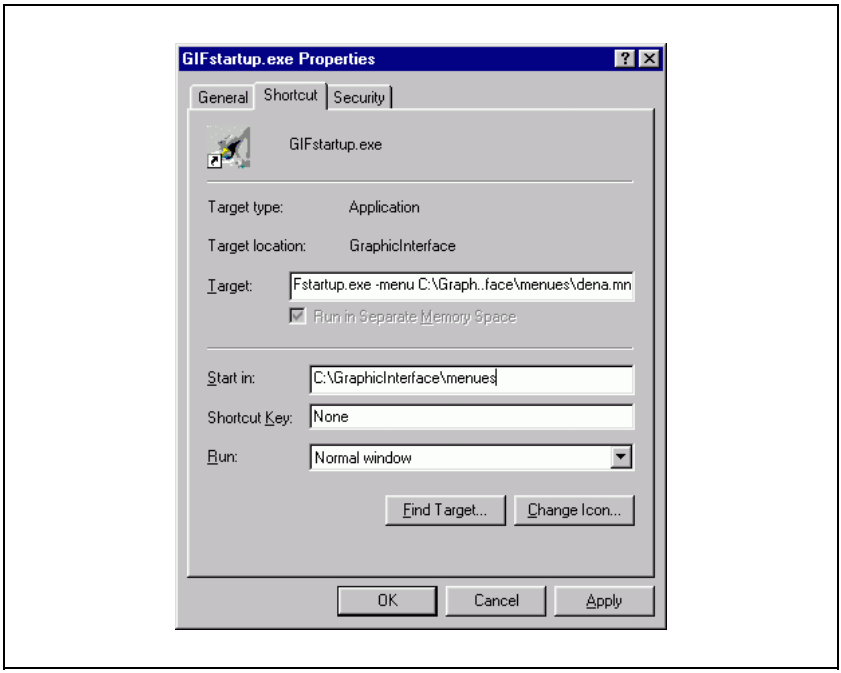


Figura 1.2

Por ejemplo, en Target o Destino colocar:

```
C:\GraphicInterface\GIFstartup.exe –menu
C:\GraphicInterface\menues\menuppal.mn
```

En Start in o Iniciar en: C:\GraphicInterface\menues

Se puede utilizar el parámetro ***–title "Título del menu"*** para colocar un título a la ventana del Graphic Interface.

Configuración para la Conexión Cliente / Servidor

Configurar el Demonio 'irexecd' en el Servidor

Se debe agregar el servicio "iexec" al archivo de servicios TCP/IP.

Para ello editar el archivo /etc/services y colocar al final del mismo:

```
iexec<><>7140/tcp (donde <> es un tabulador)
```

Luego será necesario modificar la configuración del "inetd" de modo que permita al 'irexecd' (daemon) arrancar. Para completarlo agregar la siguiente línea al archivo /etc/inetd.conf (notar que es solo una línea y deben usarse **TABS** para separar los ítems).

```
!e /etc/iexec.env -b -f /tmp/iexec.log
```

/usr2/idea51: directorio de instalación de Ideafix.

/tmp/iexec.log: archivo al que se redirige la salida estándar de errores. /etc/iexec.env: archivo de configuración de los clientes de Graphic Interface (común a todos los clientes).

Por ejemplo, contiene las variables de entorno utilizadas en Ideafix con la siguiente sintaxis:

```
VARIABLE=valor
```

Ej: IDEAFIX, DATADIR, LANGUAGE, PATH, ESSENTIA, SERVERS (si se usa), etc.

Una vez modificado se debe reinicializar el proceso 'inetd' enviándole un kill con la señal 1 (relectura de la configuración).

```
kill -1 <pid>
```

donde <pid> es el "process id" del proceso inetd que se obtiene utilizando el comando 'ps'.

```
ps -ax | grep inetd
```

IMPORTANTE: ver sintaxis del comando kill ya que puede variar con cada sistema operativo, verificar que se envía correctamente la señal 1.

NOTA: Si se trabaja con Linux 7.x o superior, el servicio se denomina “xinetd” y está ubicado en /etc/xinetd.d/

Bajo este directorio se genera un archivo denominado con el nombre del servicio a instalar, que contiene la configuración del mismo. En este caso el archivo se llama “iexec” y la sintaxis es la siguiente:

```
service iexec
{
socket_type = stream
protocol = tcp
wait = no
user = root
server = /idea51/bin/irexecd
server_args = -b !e /etc/iexec.env
log_type = FILE /tmp/iexec.log
disable = no
}
```

NOTA: para el correcto funcionamiento de IxGIF se requiere que el ejecutable **crexec.exe**

debe estar compilado para la plataforma correcta y ubicado en $IDEAFIX/bin.

Configuración del Perfil de Usuario

El perfil de usuario se configura mediante el archivo .irexecdrc que se encuentra en el directorio HOME del usuario.

El formato y el significado de los parámetros del archivo es igual a la del /etc/iexec.env

La existencia de este archivo es necesaria y permite 'personalizar' la configuración agregando variables particulares de cada usuario, aunque no elimina los seteos del /etc/iexec.env, en todo caso a igualdad de variables tendrá prioridad la del archivo .irexecdrc

IMPORTANTE: la sintaxis de este archivo de configuración debe ser exclusivamente VARIABLE=VALOR, omitiendo usar comillas y la palabra “export” ya que no es necesaria.

Para separar distintos valores para una variable, por ejemplo PATH, se debe usar ‘:’ (UNIX) o ‘;’ (Windows)

Instalación del Cliente Graphic Interface (Unix)

Descomprimir el archivo IxGIF_Setup.tgz (tar xvfz wif.tgz).

Esto creará el directorio "ixgif" con su correspondiente estructura.

Se debe editar el GIFstartup.sh y verificar el correcto seteo de las variables.

Por ejemplo:

```
JDK_HOME= {directorio de la maquina virtual java}
CL= {directorio de instalación}
SRV_HOST= {ip del host del las aplicaciones}
SRV_PORT= {puerto del irexecd en el servidor}
```

Para iniciar Graphic Interface ejecutar "GIFstartup.sh”

```
./GIFstartup.sh & (para background)
```

Capitulo 2

El archivo de Configuración .lax

El archivo .lax

En el directorio de instalación del Graphic Interface (IxGIF) existe un archivo con el mismo nombre que el ejecutable que inicia la aplicación pero con extension ".lax" (generalmente GIFStartup.lax).

Este archivo contiene las propiedades iniciales y los parámetros por default del IxGIF.

Se pueden generar varias copias de este archivo (con su correspondiente copia del .exe) y parametrizar c/u de acuerdo a que host se desea conectar o a que menu se quiere acceder, etc.

Por ejemplo copiar el GIFstartup.exe a PGM1.exe y copiar el GIFstartup.lax a PGM1.lax. Editar el PGM1.lax y cambiarle las opciones para que se conecte a otro host, levante otro menu, etc.

Parámetros mas relevantes

(Notar que los PATHS se indican con \\ )

- **`lax.application.name=GIFstartup.exe`**

Nombre del ejecutable que inicia la aplicación. Si se generan copias del archivo .lax, como también se debe cambiar el nombre del ejecutable, se debe cambiar esta línea.

- **`lax.class.path=classes/ixgif_b015.jar;classes/jcert.jar; classes/jnet.jar;classes/jsse.jar;classes/rbac.jar;lax.jar`**

Setea el CLASSPATH de la aplicación, fundamental para cargar las clases de Java. Por

default carga todos los archivos ".jar" que están en el directorio classes.

- **lax.command.line.args=$CMD_LINE_ARGUMENTS$ -e –debug**

Indica los parámetros que recibe la aplicación a traves de la línea de comandos.

-e : obligatorio por los after y before.

-debug : indica que se envíen todos los mensajes de debug a la salida de errores (stderr - ver mas abajo).

- **lax.nl.current.vm=jre\\bin\\java.exe**

Indica desde donde invoca a java para iniciar la aplicación. En este caso es un subdirectorio debajo de la instalación de IxGIF. Otro ejemplo podría ser:

`lax.nl.current.vm=C:\\JavaSoft\\JRE1.3.1\\bin\\java.exe`

- **lax.root.install.dir=C:\\GraphicInterface**

Indica directorio raíz de la aplicación.

- **lax.stderr.redirect=C:\\GraphicInterface\\debug.txt**

Indica que la salida de errores sea redireccionada a un archivo. Esta opción se ejecuta solo si se invocó al IxGIF con " –debug ".

Parámetros Adicionales

A medida que se agrega funcionalidad al IxGIF, se crean nuevos parámetros con el objeto de poder adaptar cada instalación según la necesidad del usuario.

Estos parámetros son opcionales y pueden ir en cualquier parte del archivo (se recomienda al inicio del mismo).

- **cracker.host=< dir IP o nombre_host >**

Indica a que host se va a conectar IxGIF.

- **cracker.port=< 7140 >**

Indica el puerto del servicio "irexecd". Generalmente es 7140.

(ver /etc/services en el servidor). Esta opción modifica el seteo del item "Port" en el IxGIF

(Opciones – Opciones Generales)

- **cracker.remote.menu=< nombre_menu >**

Si se generó un sistema de permisos para utilizar IxGIF, se puede indicar el nombre del menu generado. Esta opción modifica el seteo del item “Obtener menu del servidor” en el IxGIF (Opciones – Opciones Generales).

- **cracker.debug.level= < nro >**

Indica el nivel de display de mensajes de debug. El mayor nivel es ‘15’ (recomendado). Esta opción se ejecuta solo si se invoca al IxGIF con la opción “ –debug ”

- **cracker.screen.factor=< nro_factor >**

Indica el factor para el tamaño de las letras para los reportes en pantalla y el display de la ayuda. Este valor se debe modificar de acuerdo a la resolución (pixeles) de la pantalla.

Valores recomendados:

640x480 -> 1.4

800x600 -> 1.6

1024x768 -> 1.8

- **cracker.screen.applyFactor=< true | false >**

Indica si se tomará o no en cuenta los valores seteados en screen.factor

Parte VIII

•

IxTOOLS

Herramientas y Utilitarios

Sección 1 Dalí

Sección 2 IQL

Capítulo 1

Introducción

Qué es Dalí

Dalí es fundamentalmente un entorno de desarrollo que permite realizar virtualmente todo tipo de operaciones sobre programas y documentos, presentando una interface amigable de menúes y ventanas.

Objetivos de Dalí

Dalí nació como editor de Ideafix, el administrador de bases de datos relacionales de Intersoft. Posteriormente, se incrementó la cantidad de funciones disponibles, convirtiendo a Dalí en una potente herramienta para el programador, dado que cuenta con la facilidad de IBuild y varias más.

Qué distingue a Dalí de los demás editores

Como editor de texto, Dalí brinda muchas facilidades en su operación, posee potentes funciones para dar formatos varios al texto y ofrece ayuda para la programación. La interface ha sido especialmente diseñada para ser amigable y de fácil manejo, con una estructura de pantalla muy similar a la que está disponible en los programas para PCs. Existen muchas herramientas que un editor común no ofrece. Dalí tiene una interface de operación con IBuild (un “compilador universal” de Intersoft) y con gdb (el debugger de la Free Software Foundation), además un acceso automático a los administradores de versiones (RCS y SCCS),

como ejemplos de tales herramientas..

Características de Dalí

Las facilidades que Dalí ofrece al programador son, principalmente:

- Manejo de versiones
- Manejo de tags
- Ayuda de funciones de Ideafix y comandos del sistema operativo
- Búsqueda rápida de funciones y expresiones
- Interface de debugging con gdb
- Otras facilidades varias para la edición

Manejo de versiones

Dentro de los menúes de Dalí, hay opciones para trabajar con archivos administrados. Esto permite ir guardando versiones de un archivo a medida que se va modificando. Las versiones anteriores quedaran eventualmente guardadas y bloqueadas de toda posible modificación ulterior, de forma tal que se puede ver el estado del archivo en un momento dado con sólo elegir esa versión de una lista. También, dado que Dalí permite acceder al chequeo de diferencias entre archivos cualesquiera, se facilita la comparación de versiones mediante el ver sus diferencias.

Además, cuando un archivo está siendo modificado por varios usuarios, Dalí asegura que sólo uno de ellos pueda cerrar (administrar) la versión corriente o última: así se evita que dos usuarios modifiquen copias separadas del archivo y que uno de ellos elimine los cambios del otro al cerrar la versión.

Para realizar todas estas funciones, Dalí utiliza un conjunto de shell scripts propios, que a su vez invocan comandos del administrador RCS de GNU. Opcionalmente, se puede instalar shell scripts para trabajar con el administrador estándar de Unix, el Source Code Control System. Más información acerca de esto, ver el capítulo correspondiente en este manual.

Manejo de tags

Dalí provee un manejo natural de tags. Un tag es una referencia a la definición de un bloque o módulo determinado, generalmente relacionado con un lenguaje de programación. Lo más común es utilizar un lenguaje como C o C++ en donde existan clases, funciones, módulos, etc. Usualmente se usan estos tags para hacer referencia a estas construcciones. Esas referencias se guardan en un archivo, que puede ser generado por Dalí. Cuando el archivo de tags ha sido generado y está disponible, se puede indicar el nombre de la clase o función y

Dalí mostrará una lista de todas las referencias en que aparece. Con sólo elegir una de ellas, Dalí abrirá el archivo donde se encuentra la definición y pondrá el cursor sobre ella. También pueden usarse expresiones regulares sobre los tags.

Ayuda de funciones de Ideafix y comandos del sistema operativo

Dalí ofrece un sistema de ayuda on-line que satisface los requerimientos o dudas del usuario sin abandonar la edición. Cuando se está escribiendo un programa en CFIX (la extensión del lenguaje “C" para acceder a Ideafix), se puede consultar on-line las páginas del Manual del Programador correspondientes a cualquiera de las funciones de biblioteca. Si el comando o función sobre el que se pide ayuda no pertenece a la biblioteca de Ideafix, Dalí invoca al comando man del sistema operativo o algún otro comando que se especifique en la configuración.

Búsqueda rápida de funciones y expresiones

Para buscar funciones y expresiones en C o C++, existen dos opciones de menúparticularmente útiles para el programador: Hojear archivo y Hojear expresión. La primera lista en una ventana todos los prototipos de funciones que hay en el archivo corriente. Se puede elegir uno de ellos y Dalí llevará el cursor a la línea correspondiente. Si no se trata de un programa, se puede configurar esta opción para que muestre determinadas líneas en lugar de los prototipos de funciones (por ejemplo, en un archivo de definición de esquema de Ideafix, se puede hacer que aparezca la lista de todas las definiciones de tablas contenidas en ese archivo).

La segunda opción lista en una ventana todas las ocurrencias de una determinada función o expresión. En cualquiera de estas dos opciones, se puede elegir una instancia de la lista y llevarautomáticamente el cursor al lugar del programa correspondiente.

Interface de debugging con gdb

Dalí posee un modo de debugging que permite seguir paso a paso la ejecución de un programa. Además de ver la secuencia de instrucciones que se van ejecutando, se puede ver el estado de las variables, poner breakpoints, etc. Esta técnica es muy útil a la hora de depurar errores en los programas, porque permite hallar exactamente el lugar donde se produce el error, así como también las circunstancias en que aparece el problema. Para llevar a cabo esto, Dalí usa el debugger gdb de la Free Software Foundation, mostrando su salida en múltiples ventanas.

Otras facilidades

La indentación automática es un ejemplo de otras facilidades que ofrece Dalí. Cuando se está escribiendo un programa, y la línea actual está indentada (es decir, tiene un margen a la izquierda), al pulsar la tecla [INGRESAR] el cursor aparecerá al principio de la siguiente línea, pero conservando el mismo margen que tenía la línea de arriba. Además, se puede cambiar la indentación (aumentándola o disminuyéndola) para todas las líneas de un bloque a la vez.

Algunas operaciones posibles

- Hacer / deshacer
- Caracteres gráficos y acentuados
- Manipulación de bloques en forma interactiva
- Búsqueda y reemplazo usando expresiones regulares
- Atributos de visualización
- Dibujo de líneas y recuadros

Hacer / Deshacer

Dalí puede volver hacia atrás, deshaciendo los comandos ejecutados después de la última grabación. El hacer y rehacer son posibles mediante un fácil “navegado”. Con deshacer se van deshaciendo sucesivamente las operaciones hechas, mientras que con rehacer se vuelven a hacer. Esto se puede reiterar tantas veces como haga falta.

Caracteres gráficos y acentuados

Dalí posee el conjunto de caracteres gráficos con los que opera Ideafix. Este conjunto de caracteres contiene los caracteres ASCII, 32 caracteres de control, caracteres específicos para los alfabetos europeos (español, alemán, etc.) y caracteres de dibujo. La tabla completa se encuentra en el anexo de este manual.

Manipulación de bloques en forma interactiva

Para seleccionar un bloque, se lo va marcando con las flechas de movimiento del cursor, “pintando" en la pantalla la parte del archivo que interesa. Una vez marcado el bloque, se puede realizar las operaciones comunes sobre bloques: cortar, copiar, pegar, y otras más avanzadas tales como aplicar filtros, indentar (deslizar texto a izquierda o a derecha), aplicar atributos varios, etc.

Búsqueda y reemplazo

Dalí permite buscar un texto en la forma tradicional, usando expresiones regulares y también en forma incremental, es decir al momento de tipear cada carácter componente del texto a buscar.. Además, se puede hacer sustituciones, en un bloque o en todo el documento, con o sin confirmación.

Atributos visuales

Se puede aplicar un formato particular a un bloque de texto, de forma tal que aparezca resaltado, parpadeante o en video inverso en la pantalla, o que aparezca en negrita, subrayado o en itálica en el papel.

Dibujo de líneas y recuadros

Se puede dibujar líneas simples, dobles o de asteriscos, presionando un par de teclas y luego moviéndose en la dirección deseada con las flechas del cursor. El modo de dibujo también tiene una opción de borrado.

Otras facilidades de operación

Otras facilidades de Dalí son:

- Menúes pull-down y pop-up
- Formularios y ventanas de diálogo
- Mecanismo de historia
- Múltiples ventanas
- Operaciones sobre ventanas
- Menú de ayuda
- Edición simultánea de archivos

Menúes pull-down y pop-up

La mayoría de los comandos que tienen asociada una tecla también están disponible a través de alguno de los menúes. De esta forma, no es necesario recordar la combinación de teclas para cada comando, y es posible operar con Dalí usando exclusivamente estos menúes. Es posible además moverse entre los menúes utilizando las flechas del cursor y la tecla [FIN] en la forma usual y natural.

Formularios y ventanas de diálogo

Cada vez que Dalí necesita más información para ejecutar un comando, presenta al usuario un formulario o ventana de diálogo, en los que se puede seleccionar o escribir la respuest1a rápidamente y con comodidad.

Mecanismo de historia

En muchas de las ventanas de diálogo donde se debe llenar información en un campo, se puede ver una lista de los últimos valores de ese campo presionando la tecla [AYUDA].

Luego se puede elegir uno de esos valores con la tecla [INGRESAR], y si es necesario, editarlo presionando la tecla [ESPACIO].

Múltiples ventanas

Dalí permite abrir una cantidad grande (configurable) de ventanas. Cada ventana puede tener un documento distinto abierto. De esta manera, en cada ventana se puede ver una sección dada de un archivo dado, incluso un mismo archivo en distintas ventanas. Además, ciertos comandos del menú crean automáticamente una ventana y envían allí sus resultados.

Operaciones sobre ventanas

Se puede cambiar la disposición de las ventanas en la pantalla, cambiándolas de lugar y de tamaño en forma manual, llevándolas al máximo tamaño posible, o distribuyéndolas en forma de cascada o una al lado de la otra estilo mosaico.

Menú de ayuda

Dalí tiene una ventana de ayuda con información general sobre el uso del editor y un índice para solicitar ayuda sobre los temas más importantes. También brinda ayuda sensible al contexto y por indice.

Edición simultánea de archivos

Dalí permite editar varios archivos al mismo tiempo, dando la posibilidad de moverse de uno a otro con un par de teclas. Así, se puede copiar información entre archivos en forma rápida y eficiente.

Otras facilidades más avanzadas

- Archivos preservados
- Bloqueo de archivos
- Window Manager de Ideafix
- Configuración por el usuario
- Calculadora

Archivos preservados

Dalí mantiene una copia del archivo que se está editando, lo que le permite recuperar los últimos cambios realizados ante una caída del equipo. Al salir de Dalí, esta copia se borra en forma automática. Al re iniciar después de la caída del sistema, Dalí mostrará al usuario todos los nombres de los archivos preservados de esta manera, para que éste elija el que desee.

Bloqueo de archivos

Cada vez que se abre un archivo, Dalí lo bloquea usando las facilidades provistas por el sistema operativo. De esta forma, si un usuario abre un archivo que está siendo editado por otro, podrá abrirlo, pero pondrá el aviso de que esta con lock indicando además el nombre del usuario que lo bloquea. No podrá luego grabar las modificaciones a dicho archivo.

Window Manager de Ideafix

Dalí trabaja sobre el Window Manager (wm) de Ideafix (en la versión 3.6 o posterior), lo que le permite correr sobre virtualmente cualquier sistema operativo que tenga instalado Ideafix o, al menos, este wm (con la excepción de DOS/Windows).

Configuración por el usuario

Algunas de las opciones de menú tienen parámetros que son configurables por el usuario o por el administrador del sistema: por ejemplo, se puede configurar el directorio de trabajo, la cantidad máxima de ventanas que se pueden abrir simultáneamente, los directorios donde Dalí busca los archivos que se pide abrir, etc.

Calculadora

La calculadora del Window Manager de Ideafix está disponible a través de una opción de menú de Dalí. De esta forma, el usuario puede hacer sus cuentas on-line, es decir sin abandonar el editor.

Capítulo 2

Operación Básica

En este capítulo se explican las nociones básicas de operación de Dalí, las teclas más usadas y la forma de usar los menúes.

Entrando a Dalí

Para ingresar al editor, se debe escribir en el shell del sistema operativo lo siguiente:

```
dalí [-t <nombre de tag>] [-w <nombre de workspace>] [archivo
1 [archivo 2...]]
```

Si se escribe el nombre de uno o más archivos, Dalí los abre. En caso de que alguno no exista, crea la ventana correspondiente para dicho archivo, el cual será creado cuando se pida grabarlo. La opción `-t <nombre de tag>` activa Dalí cargando un workspace determinad. La opción `-w <nombre de workspace>` activa Dalí mostrando un tag determinado (ver más adelante).

Descripción de la pantalla principal

La pantalla principal de Dalí está compuesta por un menú desplegable a lo largo de la línea superior de la pantalla y una ventana con recuadro doble que ocupa el resto de la pantalla.El recuadro doble delimita la ventana activa, esto es, aquella en donde aparecerá escrito el texto que el usuario vaya tipeando o las modificaciones que éste vaya haciendo. En los bordes del recuadro, pueden aparecer diversos indicadores. Los principales (leídos de izquierda a derecha, de arriba a abajo) son los siguientes:

nombre de archivo

es el nombre del archivo que se está editando.

[nombre de función]

es el nombre de la función dentro de la cual está el cursor; es configurable, y se usa en programas C o CFIX.

*****

indica que el archivo ha sido modificado desde la última grabación.

Ins

indica si se está trabajando en el modo de inserción o de sobre escritura.

_ línea:columna

indica la posición actual del cursor dentro del archivo.

_ Bloque

indica que se está marcando un bloque de texto para realizar con él alguna operación de borrado, copiado, etc.

_ Admin

indica que el contenido de la ventana ha sido extraído de la versión administrada del archivo.

_ Sólo lectura

indica que el usuario no tiene permisos de escritura para ese archivo.

_ Bloqueado: [nombre de otro usuario]

indica que el otro usuario está accediendo desde otra terminal a ese archivo.

_ Control

indica que se está mostrando en la pantalla los caracteres de control (por ejemplo, los signos “$" que indican el final de línea).

_ TROFF

indica que el modo de edición para el procesador troff está activo.

TeX

indica que el modo de edición para el procesador Tex o LaTeX está activo.

_ Dibujo

indica que el modo de dibujo está activo.

Desplazamiento del cursor

Para desplazar el cursor dentro del archivo, existen varios comandos que se acceden mediante una o dos letras:

Para ir:	La tecla es
Un carácter a la izquierda:	[CURS_IZQ]
Un carácter a la derecha:	[CURS_DER]
A la línea de arriba:	[CURS_ARR]
A la línea de abajo:	[CURS_ABA]
Al principio de la línea:	[META][CURS_IZQ
Al final de la línea:	[META][CURS_DER]
A la palabra anterior:	[CTRLX][CURS_IZQ]
A la palabra siguiente:	[CTRLX][CURS_DER]
Una página hacia arriba:	[PAG_ANT]
Una página hacia abajo:	[PAG_SIG]
Al principio del archivo:	[META][PAG_ANT]
Al final del archivo:	[META][PAG_SIG]
Al principio de la pantalla:	[META][CURS_ARR]
Al final de la pantalla:	[META][CURS_ABA]
Al medio de la pantalla	[META]m
A la función anterior:	[META]'['
A la función siguiente	[META]']'
Al principio del ámbito	[PAG_IZQ]

Al final del ámbito:	[PAG_DER]

Modos de tipeo: inserción y sobre escritura

Existen dos modos distintos de ingresar texto en un documento. En el modo de inserción (cuando aparece “Ins" en el borde inferior de la ventana), a medida que se va escribiendo,se va haciendo lugar, corriendo el resto de la línea hacia la derecha. En el modo de sobre escritura (cuando aparece “Sob" en el borde inferior de la ventana), los caracteres que se tipea van “pisando" los caracteres que había antes. Para pasar de un modo a otro, se usa la tecla [INSERTAR].

Operaciones básicas de inserción y borrado

Los comandos para insertar y borrar texto son los siguientes:

Para	La tecla es
Borrar toda la línea del cursor:	[META]d
Borrar hasta el final de la línea:	[META]e
Borrar palabra	[META]-w
Borrar el carácter del cursor	[ELIMINAR]
Borrar un carácter a la izquierda:	[RETROCESO]
Borrar un tabulador a la izquierda:	[META][RETROCESO]
Insertar una línea en blanco arriba:	[META]O
Insertar una línea en blanco debajo:	[META]o
Insertar un tabulador	[TAB]
Unir la siguiente línea	[META]j o [ALT]-j
Insertar un par de llaves:	[META]b

Nota

Si se borra un tabulador con la tecla [RETROCESO], en lugar de eliminar todo el espacio ocupado por el tabulador (por ejemplo, 4 posiciones), Dalí convertirá el tabulador en lacantidad de espacios correspondiente, y quitará uno de ellos (en el mismo ejemplo, quedarían 3 espacios). La longitud de un tabulador es configurable por el usuario. Las teclas [META]-b son muy útiles -por ejemplo- en la programación en C o CFIX. Si se escribe el prototipo de una función y se pulsan a continuación dichas combinaciones de teclas, y Dalí colocará las llaves que delimitan el cuerpo de la función, dejando el cursor en el medio, y con la indentación adecuada.

Caracteres especiales

Los caracteres especiales son aquellos caracteres o símbolos que no se encuentran disponibles en los teclados comunes, pero que sí forman parte del juego de caracteres del Window Manager de Ideafix. Los caracteres españoles son:

Para escribir	Presione
á	'a
é	'e
í	'i
ó	'o
ú	'u
ñ	'n
!	'!
?	'?
u:	:u

En forma análoga, se pueden obtener los caracteres especiales para otros idiomas (`a, `e, 'c, 's, etc.)

Caracteres de doble tipeo

Algunos caracteres deben tipearse dos veces en el teclado para que aparezcan en la pantalla:

Para escribir	Pulsar
\e	\e\e
:	::
~	~~
'	''
`	“

Cómo se trabaja con el menú desplegable

El menú principal esta compuesto por varios menúes que se despliegan a partir del aquél. Cada uno de éstos tiene resaltada una letra.

Para acceder a uno de los sub menúes, se debe presionar la tecla [CTRLX]-t, donde t es la

letra que aparece resaltada. Aparecerá entonces un sub menú con el aspecto de una ventana más chica que la principal, tal como se muestra aquí. Dentro de un menú, se puede elegir un comando presionando la letra que aparece resaltada, o también con las flechas de movimiento del cursor y la tecla [INGRESAR]. A la derecha del nombre del comando, aparece un par de teclas, señalando el método de acceso rápido a ese comando. Al presionar esas teclas, se ejecutará el comando, sin necesidad de ingresar a ninguno de los menúes. En el ejemplo que aparece más arriba, para abrir un archivo puede usarse las teclas [CTRLX]-A (para ir al menú Archivo) y luego la tecla A (para seleccionar el comando Abrir). También se puede pulsar directamente la tecla F3.

Al final de este manual se detalla la consultar la tabla completa de los métodos de acceso rápido a cada comando.

Ventanas de diálogo (paneles)

Una ventana de diálogo, o panel, es una ventana que se abre en el centro de la pantalla cuando se selecciona un comando y éste requiere más información del usuario para poder realizar su cometido.

Por ejemplo, el comando Abrir archivo necesita el nombre del archivo a abrir. Por eso, al elegir este comando, aparecerá la siguiente ventana de diálogo (siempre y cuando el modo Comando no esté activo - ver más adelante). Dentro de una ventana de diálogo hay algunas teclas con funciones específicas. La tecla [TAB] lleva el cursor del campo donde está posicionado al campo siguiente. Las teclas de movimiento del cursor mueven el cursor de campo en campo, o dentro de un campo que está siendo editado. La tecla [INGRESAR] da por terminado el ingreso de datos, y pasa a ejecutar el comando. La tecla [FIN] cancela el comando sin ejecutarlo.

Cómo se abandona el editor

Para salir de Dalí, se debe presionar las teclas [CTRLX]-X o elegir la opción Salir del menú Archivo. Si no se ha grabado las últimas modificaciones, aparecerá una ventanapreguntando si se desea grabar el archivo antes de salir.

Menúes

A continuación describen los diferentes comandos que pueden ser accedidos a través de los menúes de Dalí. El menú principal tiene ocho menúes que se descuelgan de él, cada uno de los cuales agrupa un conjunto de funciones específicas. A continuación se describe brevemente la división de los diferentes menúes:

Sistema

Este menú -el primero desde la izquierda- no posee un título específico, y se accede con la combinación de teclas [CTRLX]-[ESPACIO]. Contiene opciones para ejecutar comandos del sistema operativo, acceder a algunos servicios del Window Manager de Ideafix y una opción que muestra la versión de Dalí con la que se está trabajando.

Archivo

Este menú incluye los comandos relativos al manejo de archivos: abrir un archivo, grabar su contenido, grabarlo con otro nombre, crear uno nuevo, insertar un archivo dentro de otro, cambiar de directorio, salir de Dalí, lectura de tags, workspace, etc..

Editar

Este menú abarca los comandos que facilitan la edición de un documento: comandos que permiten seleccionar un bloque de texto, comandos que permiten mover, copiar y borrar el bloque seleccionado y comandos que permiten agregarle atributos (resaltado, subrayado, etc.)

Búsqueda

Bajo este menú, encontramos los comandos que facilitan la búsqueda de una palabra o texto a lo largo del documento y eventualmente también su reemplazo por otra palabra o texto. También contiene los comandos que permiten hojear un archivo o las ocurrencias de una determinada expresión regular.

Opciones

Este menú contiene los comandos para activar y desactivar los modos de Dalí, y para manejar caracteres especiales.

Herramientas

Debajo de ese título se encuentran los comandos de ayuda para el programador, que incluyen el manejo de versiones de archivos, interface de debugging con gdb, interface con IBuild y manejo de tags. También una interface con los utilitarios grep, egrep y diff.

Ventana

En este menú están todos los comandos que nos permiten operar sobre las ventanas abiertas en la pantalla: pasar de una otra, cambiar su tamaño, reubicarlas, cerrarlas, etc.

Ayuda

Este menú brinda distintos tipos de ayuda sobre la forma de operación de Dalí y sus comandos. También se puede obtener ayuda sobre comandos de Unix y funciones de la biblioteca de Ideafix.

En las secciones siguientes se describe en profundidad qué opciones brinda cada menú, comenzando con el primero desde la izquierda (el menú Sistema) y recorriendo todos los demás hacia la derecha.

Capítulo 3

Menú del sistema

El menú del sistema posee comandos de propósito general y de vínculo hacia el sistema operativo.

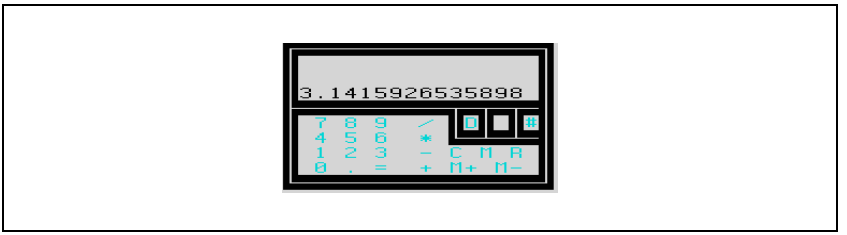
Opción: Acerca de ...

Esta opción despliega en pantalla información sobre la versión de Dalí que se está usando.

Opción: Calculadora

Cuando se selecciona esta opción del menú de servicios, se presentará en pantalla una calculadora de bolsillo que permite realizar todo tipo de operaciones. Es la misma calculadora del ambiente de desarrollo de Ideafix:

La letra “D" indica que se está trabajando en base decimal. La “M" indica que hay un número guardado en la memoria. El “#" es el último operador que se digitó.



Para hacer una operación, se procede de la misma forma que con una calculadora de bolsillo. Presionando la tecla [AYUDA], se puede obtener una ventana con la descripción de todos los comandos de la calculadora. Estos comandos son los que se muestran a continuación:

%

Calcula un porcentaje. Se ingresa primero el monto, luego se oprime la tecla % y por último, se escribe el porcentaje del monto que se desea obtener.

C

Borra el último número u operación ingresada.

[RETROCESO]

Borrando el último dígito ingresado.

[AYUDA]

Despliega o elimina la ventana con la descripción de los comandos.

#

Cambia la base de numeración. Las bases contempladas y los indicadores que aparecen en pantalla son: decimal (“D"), binaria (“B"), octal (“O") y hexadecimal (“H").

–

Cambia el signo (positivo/negativo) del número que está en la pantalla.

E

Muestra los números en formato exponencial. Si se presiona de nuevo, vuelve al formato normal.

Q

Equivale a escribir 00.

W

Equivale a escribir 000.

!

Calcula el factorial. Primero se debe escribir el número y después presionar !.

A

Sumar a la memoria el número que está en la pantalla.

L

Borra la memoria.

M

Pone en la memoria el número que está en la pantalla.

R

Recupera el contenido de la memoria.

S

Sustrae de la memoria el número que está en la pantalla.

X

Intercambia el contenido de la memoria con el número que está en la pantalla.

Opción: Imprimir pantalla

Este comando imprime la pantalla tal como se ve en el momento de invocarlo. Se puede mandar la impresión a la impresora o guardarla en un archivo (ver la configuración de la variable de ambiente printer en los manuales de Ideafix).

Opción: Ver juego de caracteres

Este comando muestra el juego completo de caracteres del Window Manager de Ideafix. Se puede seleccionar la base de numeración (octal, decimal o hexadecimal) en la que se desea ver los códigos de los caracteres.

Opción: Redibujar

Esta opción borra la pantalla y vuelve a dibujarla, para eliminar cualquier carácter extrañoque haya podido quedar superpuesto al ejecutar un comando. El comando rápido asociado es la combinación de teclas [META]-R.

Opción: Ejecutar comando del wm (window manager)

Con esta opción se ejecuta cualquier comando del sistema operativo, y se captura su salida en una ventana de Dalí. El contenido de esta ventana puede editarse y grabarse en el disco con el comando Salvar como del menú Archivo.

Los comandos que se ejecutan con esta opción no deben leer datos de la entrada estándar. En ese caso, hay que usar la opción siguiente.

El comando rápido asociado es la tecla [SUSPENDER].

Opción: Ejecutar comando del S.O.

Con esta opción se ejecuta cualquier comando del usuario del WM. El comando rápido asociado es la combinación de teclas [HOME]-F4.

Opción: Ir al shell

Esta opción es similar a la anterior, con la diferencia de que no captura la salida del comando en una ventana, sino que abre un shell donde el usuario puede escribir directamente todos los comandos que desee. Para volver a Dalí, se debe escribir “exit".

El comando rápido asociado es la combinación de teclas [META]-z.

Capítulo 4

Menú Archivo

Este menú es el segundo contando desde la izquierda en el menú principal. Se ingresa a él por medio de las teclas [CTRLX]-a. Contiene diversas opciones relativas al manejo de archivos, que son las siguientes:

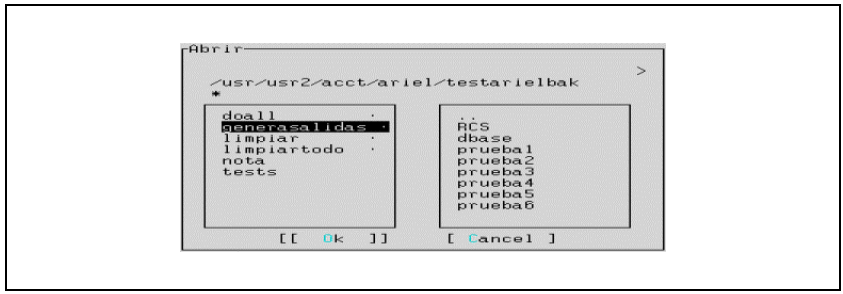
Opción: Nuevo

Esta opción abre una ventana vacía para editar un archivo nuevo (que no existe en el disco). El nombre del archivo nuevo será noname0, hasta tanto se modifique ese nombre con alguno de los comandos de grabación.

Opción: Abrir

Este comando abre una ventana con el archivo cuyo nombre se indica. Si ese archivo no existe, la ventana aparecerá vacía, pero con el nombre de archivo que corresponde. Si el modo comando está inactivo, aparecerá una ventana de diálogo para que especifiquemos qué

archivo deseamos abrir:



En el modo comando, se puede acceder al selector de archivos pulsando la tecla TAB. Dentro de esta ventana de diálogo, se puede escribir el nombre del archivo que se desea abrir, o también se puede elegir de la lista de archivos que aparece en el recuadro de la izquierda.

Si el archivo está en otro directorio, se lo puede buscar eligiendo el directorio correcto dentro del recuadro de la derecha.

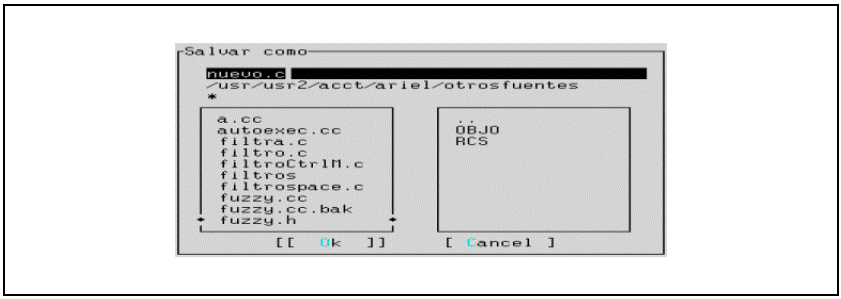
En todos los casos, para elegir un archivo o directorio, se debe usar las teclas de movimiento del cursor y la tecla [INGRESAR] (o la tecla [ESPACIO] si se desea abrir más de un archivo). Para ir de un directorio dado al directorio que está encima de él (su directorio “padre"), se debe elegir “.." en el recuadro de la derecha.

El comando rápido asociado es la tecla [SERVICIOS] o F3.

Hay dos comandos rápidos asociados: la tecla [AYUDA_APL] (F2) graba la ventana corriente solamente si ha sido modificada desde la última grabación; la combinación de teclas [META]-[AYUDA_APL] ([HOME]-F2) graba la ventana corriente aún cuando no haya sido modificada después de la última grabación.

Opción: Salvar como

Esta opción permite grabar el archivo corriente con un nombre distinto o en otro directorio. La versión anterior del archivo no se pierde, sino que se crea una copia con el nombre y la ubicación que se indique. Al elegir este comando, aparecerá una ventana de diálogo similar a la del comando Abrir:



Si se graba un archivo con el mismo nombre que otro ya existente, aparecerá un mensaje como el siguiente:

Si se responde “Sí", se borrará el archivo anterior, y quedará solamente el nuevo. Si se responde “No" o “Cancelar", no se grabará el archivo nuevo, y el anterior quedará intacto.

El comando rápido asociado es la combinación de teclas [META]-s ([HOME]-s).

Nota: Al pedir grabar, no podrá hacerse en un archivo que está siendo editado por Dalí en esa terminal en ese momento.

Opción: Salvar todo

Esta opción graba el contenido de todos los archivos abiertos (estén o no en la pantalla) que hayan sido modificados.

Opción: Insertar archivo

Esta opción inserta un archivo completo en la posición actual del cursor. Al seleccionarla, aparecerá una ventana de diálogo similar a las anteriores, que permite al usuario elegir el archivo a insertar.

El comando rápido asociado es la combinación de teclas [META]-i.

Opción: Cambiar de directorio

Esta opción permite cambiar el directorio corriente para que todas las operaciones con archivos trabajen en un directorio distinto. A partir de ese momento, cada vez que se abra una ventana de diálogo referida al manejo de archivos, el contenido del nuevo directorio aparecerá en el recuadro de la izquierda.

Al seleccionar esta opción, aparece en pantalla la siguiente ventana de diálogo mostrada aquí.

Se puede escribir el path completo para el nuevo directorio, o elegirlo con las flechas y la tecla [INGRESAR] en el recuadro de abajo.

El comando rápido asociado es la combinación de teclas [META]-c.

Opción: Abrir un tag

Esta opción busca una expresión regular en la lista de tags definidos. Si la encuentra, abre el archivo correspondiente en la posición indicada por el mismo archivo de tags.

Al elegir este comando, aparecerá una ventana de diálogo pidiendo la expresión regular que se desea buscar. Si se halla más de un tag asociado, aparecerá una ventana con un menú de tags y del que se puede elegir la entrada deseada. Presionando [INGRESAR] se abrirá el archivo correspondiente a dicha entrada. Para que esto funcione, debe estar definida la entrada DaliTags en el archivo de configuración de Dalí ("$HOME/.dali/dalirc"). El formato de esta variable es el siguiente:

```
DaliTags=directorio1:directorio2:...
```

O bien esta forma:

```
DaliTags=directorio1
DaliTags=directorio2:directorio3
DaliTags=directorio4
...
```

Cuando se pide buscar un tag, Dalí busca un archivo llamado "tags" en cada uno de esos directorios. Cada línea de ese archivo debe tener la forma:

```
Tag [TAB] clase [TAB] archivo [TAB] /expresión de búsqueda/
```

El nombre de la clase solamente se usa para aquellos tags que están en un archivo fuente de C++. En los demás casos, simplemente se escribe el tag y el archivo separados por dos tabuladores. Dentro de Dalí se pueden generar tags en forma automática, a través del

comando Generar tags del menú Herramientas. Desde el shell de Unix puede usarse el comando *bldtags*. Este shellscript está ubicado en en $IDEAFIX/bin. Sirve para generar los tags para Dalí. Por default este comando toma en cuenta sólo los archivos o fuentes que se encuentren administrados.

El comando rápido asociado es la combinación de teclas [META]-T.

Opción: Buscar un tag

Esta opción es similar a la anterior, con la única diferencia de que no pide la expresión regular a buscar, sino que busca el tag sobre el que está posicionado el cursor. El comando rápido asociado es la combinación de teclas [META]-t.

Opción: Ir a archivo

Esta opción abre el archivo en cuyo nombre dentro del texto está posicionado el cursor. Si además está seguido de un número de línea, coloca el cursor en esa posición. Es útil cuando se tiene una ventana con la salida de un comando, y este comando ha arrojado errores en la forma "usual", como por ejemplo:

```
prueba.c:10: wmtops.h: No such file or directory
```

En ese caso, si se coloca el cursor al principio del nombre o encima de éste y se llama a la opción Ir a archivo, Dalí abrirá el archivo "prueba.c" y colocará el cursor en la línea 10, en el primer carácter distinto de blanco. El comando rápido asociado es la combinación de teclas [META]-g / [META]-y.Sólo abre el archivo pero no activa la ventana.

Opción: Abrir ambiente

Un "ambiente" es la disposición de la pantalla de Dalí en un momento dado, es decir: qué ventanas están abiertas, nombre del archivo de trabajo en cada una de ellas, así como los tamaños de éstas, etc. Véase la opción Grabar Ambiente.

La información que se graba incluye las ventanas que están abiertas, el contenido de cada una, su tamaño y posición en la pantalla. Si alguna ventana está abierta pero no aparece en la pantalla, también queda registrada.

Esta opción pide el nombre del ambiente que se desea recuperar, lo busca en el disco y deja la pantalla tal como estaba en el momento de grabarlo con la opción Salvar ambiente.

Opción: Salvar ambiente

Este comando graba la disposición actual de la pantalla de Dalí, generando un archivo con extensión ".wsp" en el directorio "$HOME/.Dalí". También se guarda la marca que puede

haber en cada ventana, y el nombre del proyecto que se está usando.

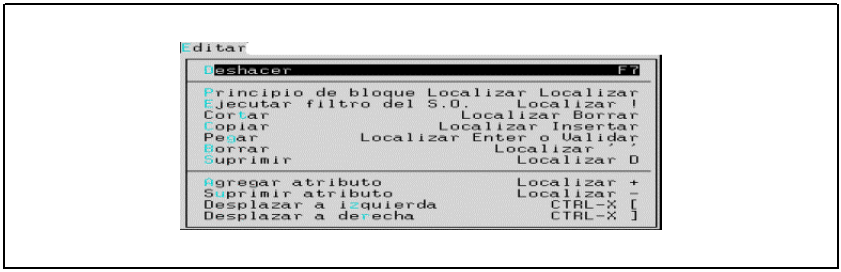
Opción: Salir

Con esta opción, se cierran todas las ventanas, se graba el ambiente default (si así lo indica la variable de configuración SaveDesktop) y se sale de Dalí. Si alguna de las ventanas contiene modificaciones que no han sido grabadas, Dalí preguntará si se las desea grabar antes de salir. El comando rápido asociado es la combinación de teclas [CTRLX]-x .

Capítulo 5

Menú Editar

Este menú es el tercero contando desde la izquierda en el menú principal. Se ingresa a él por medio de las teclas [CTRLX]-e. Contiene varias opciones relativas a la edición del archivo, que son las siguientes:



Opción: Deshacer

Esta opción permite deshacer los cambios realizados al archivo desde la última vez que se grabó o se cambió de ventana. Aparecerá el indicador de modo en el borde inferior de la ventana, juntamente con dos números que indican la posición actual dentro de la “historia” de comandos.

A partir de ese momento (hasta que se vuelva a elegir la opción Deshacer), se puede retroceder en la edición del archivo con las teclas [CUR_ARR] y [PAG_ANT]. También se puede volver a hacer lo que se había deshecho, con las teclas [CUR_ABA] y [PAG_SIG]. El comando rápido asociado es la tecla [DESHACER].

Opción: Principio de bloque

Esta opción permite marcar un bloque de texto para realizar sobre él diversas operaciones: moverlo, copiarlo, borrarlo, ponerle un atributo, etc.

Después de elegir esta opción, se puede usar las teclas y comandos usuales de movimiento del cursor para ir pintando el bloque que se desea.

El fin del bloque no se marca: se considera que es la posición actual del cursor en el momento de ejecutar la operación sobre el bloque.

Si se desea desmarcar el bloque sin ejecutar ninguna operación, se puede presionar la tecla [FIN].

El comando rápido asociado es la combinación de teclas [META]-[META].

Opción: Ejecutar filtro del S.O.

Esta opción toma el bloque que está seleccionado, lo pasa como entrada estándar a un comando y finalmente reemplaza el bloque por la salida de ese comando.

Es útil para ejecutar ciertas operaciones que no están disponibles en Dalí, pero sí lo están a través de algún comando del sistema operativo u otro programa. Por ejemplo, se puede ordenar alfabéticamente un bloque filtrándolo a través del comando sort, y se puede numerar un conjunto de líneas filtrando ese bloque a través del comando nl.

Cuando se elige esta opción (después de haber seleccionado un bloque de texto), aparece una ventana de diálogo, donde se debe escribir el nombre del comando o filtro que se desea ejecutar.

El comando rápido asociado es la combinación de teclas [META]-!.

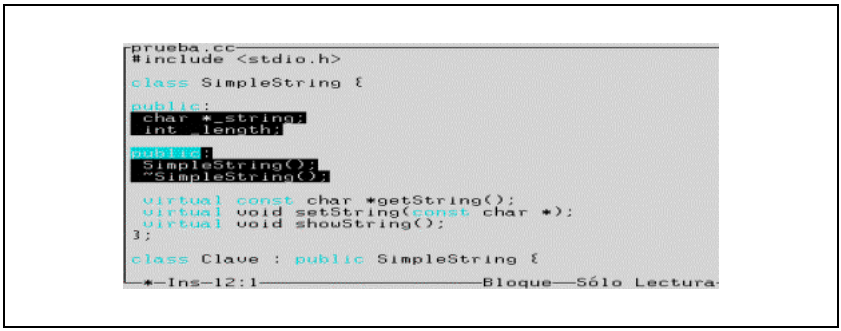
Ejemplos: c++filt, sort, indent, cb

Opción: Cortar

Esta opción copia el bloque seleccionado al portapapeles de Dalí (un buffer de almacenamiento temporario) y luego lo saca del documento.

Generalmente se usa para mover un bloque de texto, cortándolo con esta opción y volviéndolo a colocar en otro lugar del texto con la opción Pegar.

El comando rápido asociado es la combinación de teclas [META]-[SUPRIMIR].



Opción: Copiar

Esta opción copia el bloque seleccionado al portapapeles de Dalí, sin sacarlo del documento. El bloque copiado puede repetirse en otras posiciones del documento con la opción Pegar de este mismo menú. El comando rápido asociado es la combinación de teclas [META]-[INSERTAR].

Opción: Pegar

Esta opción inserta en la posición actual del cursor el texto que había sido guardado en el portapapeles de Dalí (con las opciones Cortar o Copiar). El contenido del portapapeles no se modifica, de forma tal que se puede volver a pegar la misma porción de texto en otra posición. El comando rápido asociado es la combinación de teclas [META]-[INGRESAR].

Opción: Borrar

Esta opción elimina el bloque de texto seleccionado, sin cubrir el espacio que ocupaba,y sin pasarlo al portapapeles. El comando rápido asociado es la combinación de teclas [META]-[ESPACIO].

Opción: Suprimir

Esta opción elimina el bloque de texto seleccionado, sin pasarlo al portapapeles, pero cubriendo el espacio que ocupaba (desplazando hacia arriba el resto del archivo). El comando rápido asociado es la combinación de teclas [META]-D.

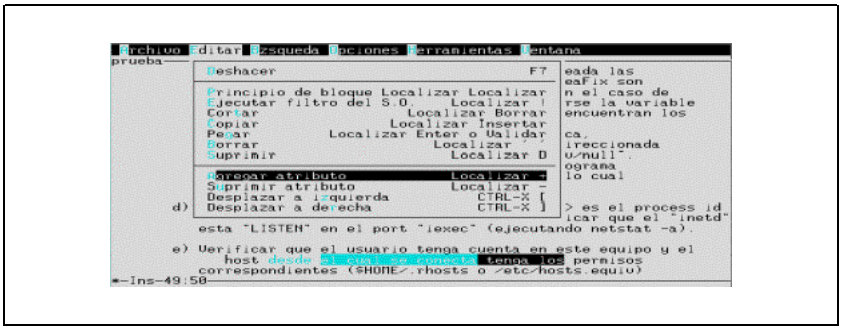
Opción: Agregar atributo

Esta opción permite agregar al texto seleccionado un atributo. Los atributos disponibles son los siguientes:

- resaltado
- subrayado
- parpadeo
- video inverso
- cualquiera de los colores disponibles

Cuando se elige esta opción (después de haber seleccionado el bloque de texto) aparece en una esquina de la pantalla la lista de los atributos que se puede agregar:

Algunos atributos no son admitidos en ciertas terminales e impresoras. Por ejemplo, cierto tipo de terminales con plaquetas VGA muestran el subrayado como video inverso, y la mayoría de las impresoras láser muestra el subrayado como “itálicas”.



El comando rápido asociado es la combinación de teclas [META]-'+'.

Opción: Suprimir atributo

Esta opción quita un atributo al bloque de texto seleccionado. Al elegirla, aparece la misma lista de atributos que en la opción Agregar atributo. El comando rápido asociado es la combinación de teclas [META]-'-'.

Opción: Desplazar a izquierda

Esta opción disminuye la sangría o margen izquierdo del bloque de texto seleccionado, corriéndolo hacia la izquierda el espacio ocupado por un tabulador. Puede accionarse

repetidas veces hasta obtener el espacio deseado. El comando rápido asociado es la combinación de teclas [CTRLX]-'['.

Opción: Desplazar a derecha

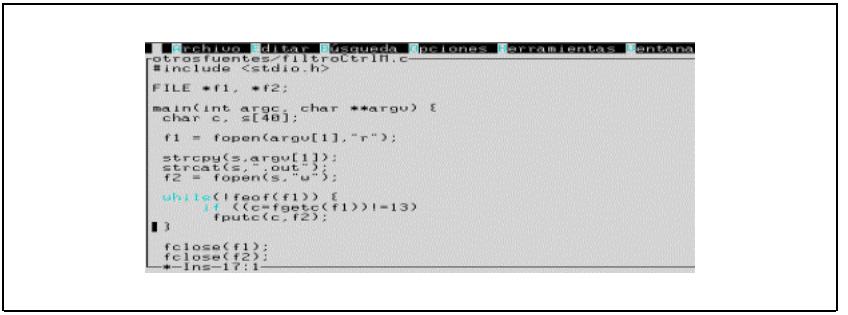
Esta opción aumenta la sangría o margen izquierdo del bloque de texto seleccionado, corriéndolo hacia la derecha el espacio ocupado por un tabulador.

Puede accionarse repetidas veces hasta obtener el espacio deseado.

A continuación, se muestra un ejemplo de esta opción, a través de dos pantallas con el estado del archivo antes y después de usarla. El comando rápido asociado es la combinación de teclas [CTRLX]-']'.

Antes:

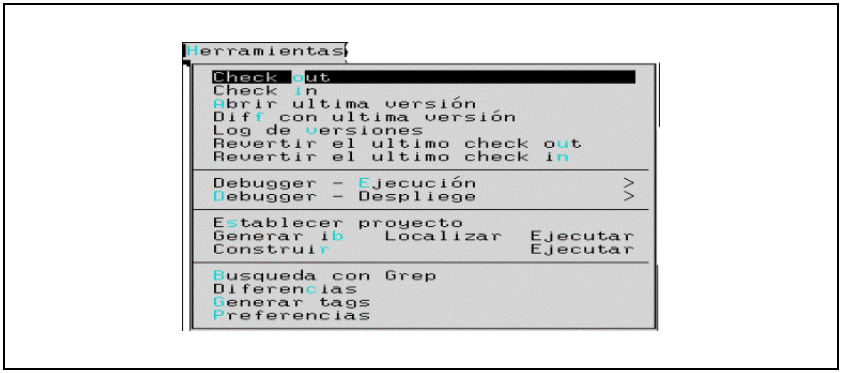
Después:



Capítulo 6

Menú Herramientas

Este menú es el sexto contando desde la izquierda en el menú principal. Se ingresa a él por medio de las teclas [CTRLX]-h. Contiene varias opciones relacionadas con la interface entre Dalí y otros utilitarios, que son las siguientes:



Opción: Check out

Esta opción crea una nueva versión editable del archivo contenido en la ventana corriente. Para esto, llama al shell script *checkout* con la opción “-e”, el cual se encarga de hacer el trabajo. Si el archivo ha sido modificado antes de extraer la nueva versión, aparecerá una ventana pidiendo confirmación para la operación.Existen varias versiones de estos shellscript, dependiendo del administrador de versiones que se use (RCS o SCCS, siendo RCS el default), así como la modalidad con que se usen estos administradores (eg, dónde guardan los archivo de control). Lo más común es tener andando el RCS que utiliza un directorio con el nombre RCS. El SCCS guarda en cambio estos archivos de control en el mismo directorio que el texto original. Lo más recomendable es utilizar un sub directorio del original, independientemente

de que se use RCS o SCCS.

Para más información, se sugiere consultar el capítulo sobre manejo de archivos administrados.

Opción: Check in

Esta opción cierra la última versión del archivo contenido en la ventana corriente. Para esto, llama al shell script *checkin* con la opción -k, el cual se encarga de administrarlo y da el atributo de read-only (sólo lectura) al archivo original. Si se usa la opcion -c, se puede especificar en la línea de comando el comentario asociado a la versión. Aparece luego una ventana de diálogo para pedir el comentario que se desea incluir en el archivo administrado para la versión que se está cerrando.

Véase la opción Check out. Para más información, se sugiere consultar el capítulo sobre manejo de archivos administrados.

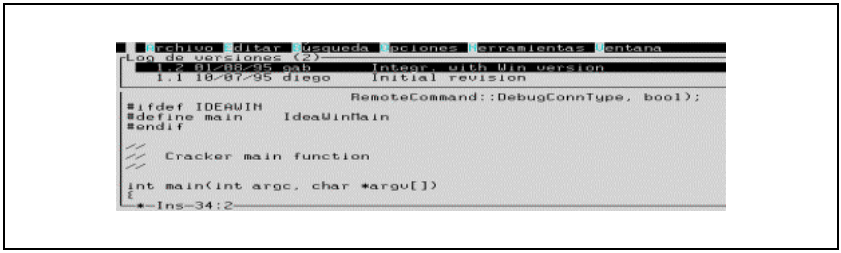
Opción: Abrir última versión

Esta opción abre la última versión disponible del archivo administrado que está en la ventana corriente. La versión que aparece no es editable.

Para más información, consultar el capítulo sobre manejo de archivos administrados.

Opción: Log de versiones

Esta opción muestra en una ventana una lista de todas las versiones disponibles del archivo administrado que está en la ventana corriente, de esta manera.



Se puede elegir una de ellas con la tecla [INGRESAR], y Dalí abrirá una nueva ventana con la versión indicada del archivo.

Opción: Revertir el último check out

Esta opción deshace el check out que se había realizado antes. dejando al archivo administrado cerrado en la última versión. Para esto, llama al shell script checkout con la opción -u.

Para más información, consultar el capítulo sobre manejo de archivos administrados.

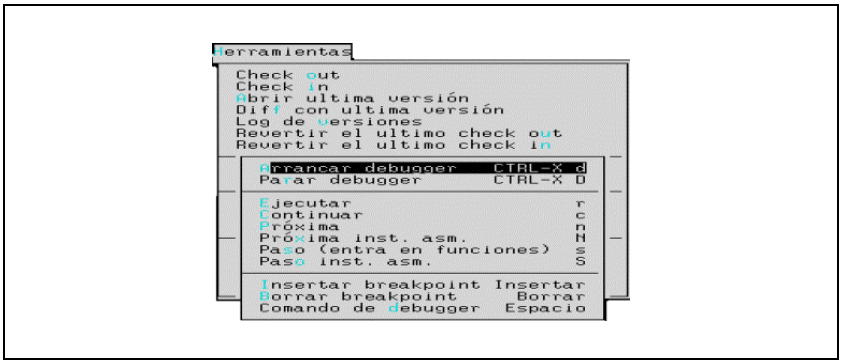
Opción: Revertir el último check in

Esta opción deshace el último check in que se había ejecutando, quitando la última versión del archivo administrado. Para esto, llama al shell script checkin con la opción -u. No se puede revertir el último check in si se ha hecho un check out.

Para más información, consultar el capítulo sobre manejo de archivos administrados.

Opción: Debugger - Ejecución

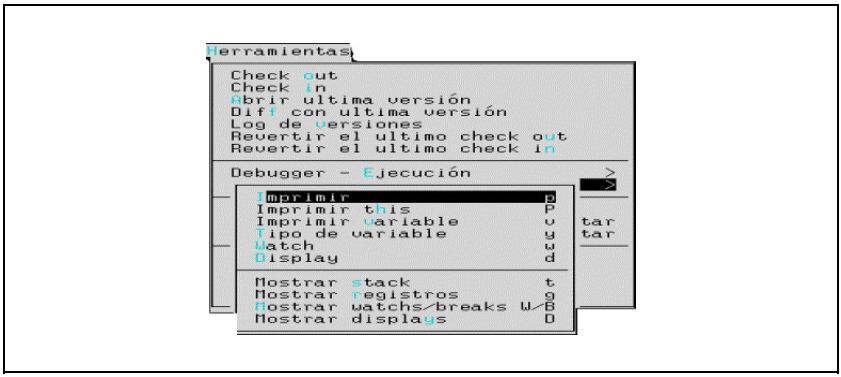
Esta opción abre un sub menú que contiene los comandos para arrancar y parar el debugger y para ejecutar paso a paso un programa.



Las opciones de este sub menú están explicadas en un capítulo separado. Para más información, consultar el capítulo sobre manejo de la interface de debugging con gdb.

Opción: Debugger - ‘

Esta opción abre un sub menú que contiene los comandos para mostrar el valor de una expresión mientras se está ejecutando un programa paso a paso, y el estado del debugger en un momento dado.



Las opciones de ese sub menú están explicadas en un capítulo separado. Para más información, consultar el capítulo sobre manejo de la interface de debugging con gdb.

Opción: Establecer proyecto

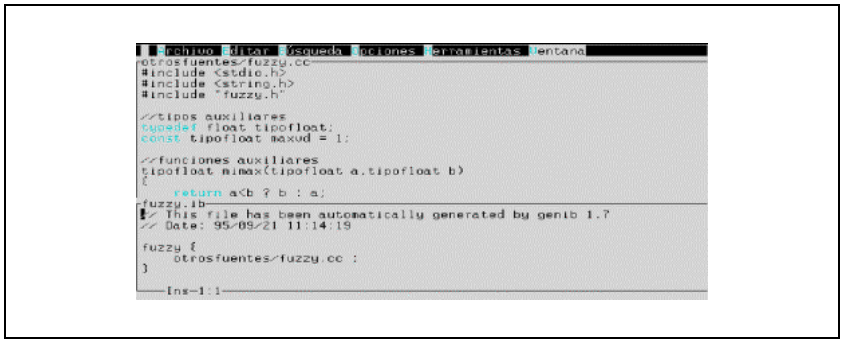
Esta opción se usa para indicar a IBuild en qué directorio debe buscar el archivo de configuración, así como el nombre de éste. El nombre puede ser cualquiera, sólo que debe terminar en .ib . Si sólo se especifica un directorio se asume por default el archivo “$IPROJECT/ibrules.ib”, cao contrario se ignora el contenido de la variable de ambiente IPROJECT.

Para más información, se sugiere consultar el capítulo sobre el utilitario IBuild.

Opción: Generar ib

Esta opción crea un archivo “.ib” para el programa fuente que está abierto en la ventana corriente. El shell script genib -que es invocado por esta opción- establece las dependencias del archivo según las sentencias “#include” que contenga.

Por ejemplo, la siguiente es la pantalla de Dalí después de haber generado un archivo “.ib” para el programa en C que se está editando.



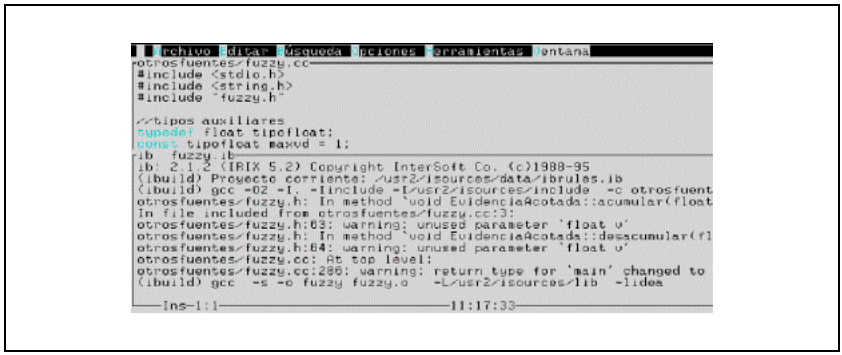
El comando rápido asociado es la combinación de teclas [META]-[PROCESAR].

Para más información, consultar el capítulo sobre el utilitario IBuild.

Opción: Construir

Esta opción construye el archivo que está en la ventana corriente. Para ello, llama al utilitario IBuild, pasándole como parámetros el contenido de la variable de configuración IbFlags y el nombre del archivo a construir. La salida generada por el utilitario IBuild aparecerá en una ventana en la parte inferior de la pantalla.

Por ejemplo, la siguiente es la pantalla de Dalí después de haber construido el archivo ".ib" que está en la figura anterior (por razones de compaginación, se ha eliminado los mensajes que generan los comandos invocados por IBuild):



El comando rápido asociado es la tecla [PROCESAR].

Opción: Preferencias

Esta opción hace que Dalí relea el archivo de configuración “dalirc”. Es útil cuando se ha modificado ese archivo de configuración usando el mismo Dalí.

Opción: Generar tags

Esta opción llama al shell script bldtags, que arma un archivo de tags buscando texto a partir de determinadas expresiones regulares en todos los archivos administrados que están debajo de un directorio dado. Vale decir, si se decide generar tags para un directorio dado, todos los archivos de los sub directorios de éste último serán analizados y serán generados los tags correspondientes para ellos.

Al elegir esta opción, Dalí pedirá a través de un cuadro de diálogo confirmación para realizar la operación por cada uno de los directorios que aparecen en la variable de configuración *DaliTags*, permitiendo así al usuario elegir qué tags desea generar.

Si se responde con Sí, Dalí creará un archivo con el nombre “tags” en el directorio para el cual se están creando junto a todos sus subdirectorios..

Capítulo 7

Menú Ventana

Este menú es el anteúltimo contando desde la izquierda en el menú principal. Se ingresa a él por medio de las teclas [CTRLX]-v. Contiene diversas opciones para el manejo de ventanas, que son las siguientes:

Opción: Cambiar tamaño

Esta opción permite cambiar el tamaño de la ventana corriente. Al elegirla, aparecerán unas marcas en las esquinas de la ventana, indicando que puede usarse las teclas de movimiento del cursor para ajustar el tamaño. Para fijar el nuevo tamaño, se debe presionar la tecla [INGRESAR]. Para cancelar el comando sin cambia la ventana, se puede presionar la tecla [FIN].

El comando rápido asociado es la combinación de teclas [CTRLX]-Z.

Opción: Mover

Esta opción permite cambiar la posición de la ventana corriente dentro de la pantalla. Al

elegirla, aparecerán unas marcas en las esquinas de la ventana que se está moviendo.

Se puede usar las teclas de movimiento del cursor para desplazar la ventana, la tecla[INGRESAR] para dejar la ventana en la nueva posición y la tecla [FIN] para anular el comando y no se habrán producido cambios.

El comando rápido asociado es la combinación de teclas [CTRLX]-M.

Opción: Agrandar/achicar

Esta opción expande la ventana corriente para que ocupe toda la pantalla. Si la ventana ya estaba expandida, la vuelve a poner en su tamaño anterior.

El comando rápido asociado es la tecla [IGNORAR].

Opción: Próxima

Esta opción lleva el cursor a la próxima ventana abierta, convirtiéndola en la ventana corriente. La ventana anterior queda abierta.

El comando rápido asociado es la combinación de teclas [CTRLX]-n.

Opción: Anterior

Esta opción lleva el cursor a la ventana anterior dentro de la lista de ventanas abiertas, convirtiéndola en la ventana corriente. La ventana en la que estaba el cursor en el momento de ejecutar este comando queda abierta.

El comando rápido asociado es la combinación de teclas [CTRLX]-p.

Opción: Cerrar

Esta opción cierra la ventana corriente. Si el archivo que contiene ha sido modificado después de la última grabación, aparecerá una ventana pidiendo confirmación.

El comando rápido asociado es la tecla [REMOVER].

Opción: Cerrar todo

Esta opción cierra todas las ventanas abiertas. Si alguna de ellas contiene un archivo que ha sido modificado después de la última vez que fue grabado, aparecerá una ventana similar a la de la opción Cerrar.

El comando rápido asociado es la combinación de teclas [META]-[REMOVER].

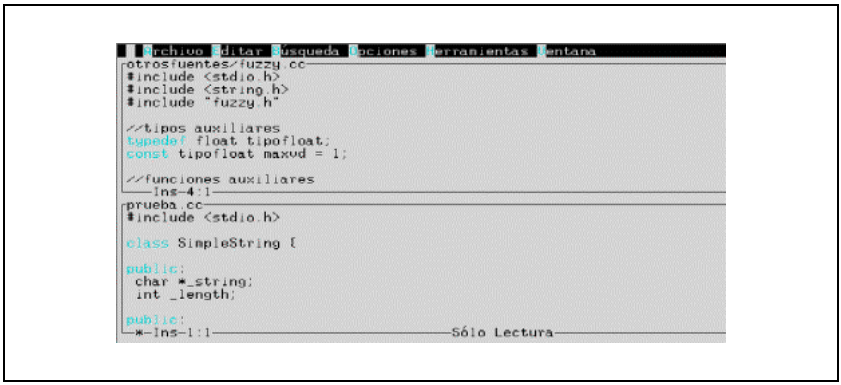
Opción: Seleccionar

Esta opción permite pasar de una ventana a otra seleccionándola directamente de la lista de ventanas abiertas. En la lista de ventanas, las ventanas que están activas (i.e. además de estar abiertas, aparecen en la pantalla) tienen una marca a la izquierda del nombre.

El comando rápido asociado es la combinación de teclas [CTRLX]-l.

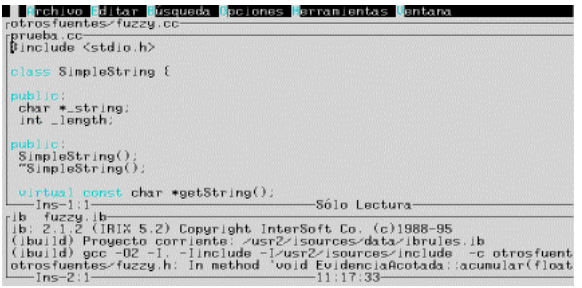
Opción: Mosaico

Esta opción distribuye las ventanas activas en la pantalla, asignando una porción más o menos parecida a cada una de ellas: La cantidad máxima de ventanas que distribuye Dalí es la indicada en la variable de configuración WindowsTile. El comando rápido asociado es la combinación de teclas [CTRLX]-m.



Opción: Cascada

Esta opción distribuye las ventanas activas en la pantalla, superponiéndolas en forma de cascada o “escalera”:

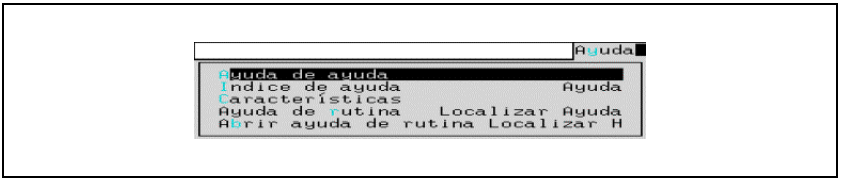


El comando rápido asociado es la combinación de teclas [CTRLX]-c.

Capítulo 8

Menú Ayuda

Este menú, el último desde la izquierda, presenta opciones de ayuda en línea y general para el usuario. Se ingresa a él por medio de las teclas [CTRLX]-y. Contiene algunas opciones relacionadas con la ayuda que Dalí ofrece al usuario, que son las siguientes:



Opción: Ayuda de ayuda

Esta opción muestra una ventana con indicaciones sobre cómo se accede a las diferentes ayudas que brinda Dalí.

Para desplazar el texto, se usa las teclas de movimiento del cursor. Para salir, se usa la tecla [FIN].

Opción: Indice de ayuda

Esta opción muestra una lista de todos los temas sobre los que se puede pedir ayuda. Al elegir uno de ellos con la tecla [INGRESAR], aparecerá la ventana de ayuda para ese tema.

El comando rápido asociado es la tecla [AYUDA].

Opción: Características

Esta opción brinda información general acerca de Dalí, las posibilidades que brinda y las principales características que tiene.

Opción: Ayuda de rutina

Esta opción busca ayuda para la rutina cuyo nombre está debajo del cursor. Si es el nombre de una función de la biblioteca de Ideafix, mostrará las páginas del Manual del Programador correspondientes. En caso contrario, Dalí invocará al comando indicado por la variable de configuración DaliMan, que generalmente será el comando man del sistema operativo.

El comando rápido asociado es la combinación de teclas [META]-[AYUDA].

Opción: Abrir ayuda de rutina

Esta opción es similar a la anterior, pero en lugar de usar el nombre de rutina que está debajo del cursor, pide al usuario que lo ingrese.

El comando rápido asociado es la combinación de teclas [META]-H

Capítulo 9

Administración de archivos

Introducción

La administración de archivos es una herramienta muy útil para el programador, que le permite mantener una “historia” de la evolución de un archivo a lo largo del tiempo.

Cada vez que se crea una nueva versión de un archivo, Dalí busca las modificaciones que se realizaron y las marca en un archivo especial. También graba información sobre quién hizo los cambios, en qué fecha y un breve comentario ingresado por el usuario.

Este archivo especial (que se guarda en un sub directorio específico debajo del directorio al que pertenece el archivo modificado) contiene la versión original del archivo y todas las modificaciones que se fueron haciendo en las sucesivas versiones. Así, Dalí puede

obtener el estado de un archivo cuando se cerró una versión determinada, sin las modificaciones que se hicieron más tarde.

Todo esto se realiza sin duplicación de la información: no se guarda una copia de cada versión del archivo, sino que se guarda una sola copia y la lista de todas las modificaciones que se hicieron en cada una de las versiones. Así se evita que el programador tenga varios archivos distintos (“ordenes.c”, “ordenes2.c”, “ordenes3.c”, etc.) para mantener diferentes versiones de un mismo programa. Todo queda grabado dentro del archivo administrado o dentro del archivo de control.

Cuando el archivo va a ser modificado por un usuario, éste abre una versión “editable” del mismo. A partir de ese momento, el archivo especial queda bloqueado, y los demás usuarios sólo podrán leer versiones del mismo, pero no podrán crear nuevas versiones. Así se evita que dos usuarios modifiquen simultáneamente un mismo archivo, y que al cerrar uno de ellos la versión, “pise” los cambios realizados por el otro usuario.

Cómo implementa Dalí esta técnica

Las opciones para manejo de archivos administrados están en el menú Herramientas. Estas opciones llaman a un conjunto de shell scripts provistos con Dalí, los que a su vez invocan a los comandos específicos del utilitario de administración de archivos.

Este utilitario de administración de archivos es un software de dominio público: el Revision

Control System (RCS) de GNU. Opcionalmente, se puede configurar Dalí para que trabaje con el utilitario de administración de archivos que es estándar en Unix: el Source Code Control System (SCCS).

La correspondencia entre las opciones de menú, los shell scripts y los utilitarios de administración es la siguiente:

- Comandos de administración de versiones

Opción del menú	Shell script	Comando RCS	Comando SCCS
Check out	checkout -e	co -l	get -e
Check in	checkin	ci	delta
Abrir última versión	checkout	co	get
Log de versiones	verslog	rlog	prs
Revertir el último check out	checkout -u	co -u	unget
Revertir el último check in	checkin -u	rcs -o	rmdel

Los archivos administrados se guardan en un sub directorio, debajo del directorio que contiene el archivo original. El nombre de ese sub directorio es “RCS” si se está trabajando con el Revision Control System, o “SCCS” si se está trabajando con el Source Code Control System.

Al hacer el check in de un archivo por primera vez, Dalí se fija de alguna manera si existe el directorio de administración. Si no existe, lo crea automáticamente. En realidad, lo hace a través del RCS si se está usando este administrador de verisones, o bien lo hace a través de algun shellscript incluido con Dalí si se está usando el SCCS.

Capítulo 10

Debugging

Introducción

Se llama debugging al proceso de depuración o eliminación de errores dentro de un programa. Un debugger es una herramienta que permite seguir paso a paso la ejecución del programa, para ver dónde se produce el error y en qué circunstancias.

Dalí ofrece al programador un conjunto de opciones de debugging muy completo, que permite -entre otras- las siguientes posibilidades:

Seguimiento paso a paso: se puede ejecutar el programa de a una instrucción por vez, viendo en una ventana qué funciones son llamadas, cómo se recorren los bucles, etc.

Niveles de detalle: el tamaño de cada "paso" puede ser elegido por el usuario: se puede ejecutar todo el programa de una sola vez, toda una función, una sola instrucción del lenguaje C o C++, o incluso una sola instrucción de lenguaje máquina.

Breakpoints: un breakpoint es una marca que se hace en el código fuente del programa para que el debugger detenga la ejecución del programa al llegar a ese punto.

Evaluación de expresiones: en cualquier momento, se puede ver el contenido de una variable o el resultado de una expresión, eligiendo la opción adecuada del menú de Dalí. También se puede evaluar expresiones que involucren llamadas a otras funciones, sean del programa o de una biblioteca como la de Ideafix.

Monitoreo de variables: cuando se desea que Dalí muestre el contenido de una variable cada vez que ésta cambia de valor, se coloca un watch. Cuando se desea que Dalí muestre el valor de la variable en cada paso (aunque no cambie), se pone un display. En ambos casos, se puede trabajar con el valor de una expresión en lugar del contenido de una variable.

Monitoreo del sistema: una de las opciones de Dalí muestra en una ventana el contenido del stack (qué funciones fueron llamadas, dónde y con qué parámetros). Otra opción muestra el contenido de los registros de la CPU.

Monitoreo del debugger: también hay opciones que muestran en una ventana la lista de breakpoints, watchs y displays que fueron colocados por el usuario.

Cómo implementa Dalí esta técnica

Dalí trabaja con gdb -el debugger de GNU- , software de dominio público.

Además de las posibilidades ya mencionadas (que están disponibles directamente a través de opciones de menú de Dalí), este debugger ofrece una cantidad adicional de comandos, que pueden elegirse con la opción Comando de debugger del menú Debugger - Ejecución. Para que un programa pueda ser depurado, el único requisito que debe cumplir es que haya sido compilado para debugging, porque de esa forma se guarda la información sobre los símbolos del programa dentro del ejecutable.

Además, si el programa a depurar es usuario del Window Manager de Ideafix (por ejemplo, si usa formularios), se necesitará una segunda terminal sobre la cual Dalí mostrará la salida del programa. En la primera terminal aparecerá el código fuente del programa, junto con todas las salidas del debugger.

Opciones del menú Debugger - Ejecución

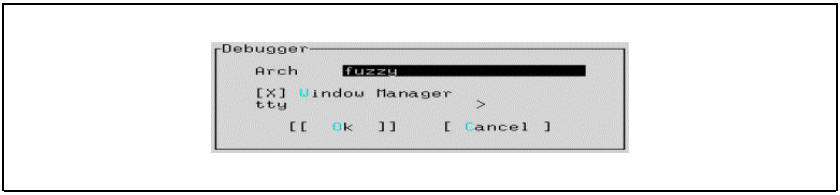
Las opciones disponibles son las siguientes.

Opción: Arrancar debugger

Esta opción prepara el debugger para ejecutar un programa. No hace falta que el programa esté abierto en una ventana, puesto que Dalí buscará el código fuente y lo mostrará en la pantalla sin intervención del usuario.

Al elegir esta opción, aparece una ventana de diálogo similar a la mostrada aquí.

En el primer campo, se debe escribir el nombre del programa que se desea ejecutar, seguido de los mismos argumentos que recibiría en la línea de comando.



Los otros dos campos solamente se usan si el programa es usuario del Window Manager de Ideafix: en el primero se marca esta condición y en el segundo se escribe el nombre de device de la terminal a la que se quiere enviar la salida del programa a ejecutar (por ejemplo, “/dev/tty02").

El comando rápido asociado es la combinación de teclas [CTRLX]-d.

Opción: Parar debugger

Esta opción sale del modo de debugging y cierra la ventana de salida del mismo. La ventana del archivo fuente (y cualquier otra que pudiera haber abierto el usuario) quedan abiertas, para que se pueda modificar su contenido.

El comando rápido asociado es la combinación de teclas [CTRLX]-D.

Opción: Ejecutar

Esta opción vuelve a ejecutar el programa desde el principio, hasta llegar al primer breakpoint o watchpoint. Es útil cuando el usuario pasó por encima del error sin darse cuenta, y quiere volver a empezar.

La opción Arrancar debugger llama automáticamente a esta opción, después de haber fijado un breakpoint en la primera línea del programa para que se ejecute el programa paso a paso de allí en más.

El comando rápido asociado es la tecla 'r'.

Opción: Continuar

Esta opción reanuda la ejecución del programa (a partir de donde estaba detenido), deteniéndose en el próximo breakpoint o watchpoint, o cuando el programa termina.

El comando rápido asociado es la tecla 'c'.

Opción: Próxima

Esta opción ejecuta la siguiente instrucción de lenguaje C o C++, y luego se detiene. Sila instrucción que se ejecuta es una llamada a función, se ejecuta toda la función de una sola vez antes de detenerse.

El comando rápido asociado es la tecla 'n'.

Opción: Próxima instrucción asm.

Esta opción ejecuta la siguiente instrucción de lenguaje máquina, y luego se detiene. Si la instrucción que se ejecuta es una llamada a una subrutina, se ejecuta toda la subrutina de una sola vez antes de detenerse.

El comando rápido asociado es la tecla 'N'.

Opción: Paso

Esta opción ejecuta la siguiente instrucción de lenguaje C o C++, y luego se detiene. Si la instrucción que se ejecuta es una llamada a función, se ejecuta solamente la llamada, y el cursor queda al principio de la función invocada.

El comando rápido asociado es la tecla 's'.

Opción: Paso inst. asm.

Esta opción ejecuta la siguiente instrucción de lenguaje máquina, y luego se detiene. Si la instrucción que se ejecuta es una llamada a una subrutina, se ejecuta solamente la llamada, y la ejecución se detendrá en la primera instrucción de la subrutina invocada.

Como una instrucción de lenguaje C o C++ abarca varias instrucciones de lenguaje máquina, es posible que una sola invocación a esta opción no produzca ningún efecto visible en la pantalla.

El comando rápido asociado es la tecla 'S'.

Opción: Insertar breakpoint

Esta opción coloca un breakpoint en la línea del código fuente donde está el cursor, obligando al debugger a detenerse en esa instrucción la próxima vez que deba ejecutarla.

El comando rápido asociado es la tecla [INSERTAR].

Opción: Borrar breakpoint

Esta opción elimina todos los breakpoints que estuviesen puestos sobre la línea del cursor.

Si hay varios breakpoints en la misma posición y sólo se quiere borrar uno de ellos, se debe usar la opción Mostrar watchs/breaks del menú Debugger - Despliegue.

El comando rápido asociado es la tecla [SUPRIMIR].

Opción: Comando de debugger

Esta opción se usa para invocar comandos del debugger que no estén disponibles en el menú de Dalí.

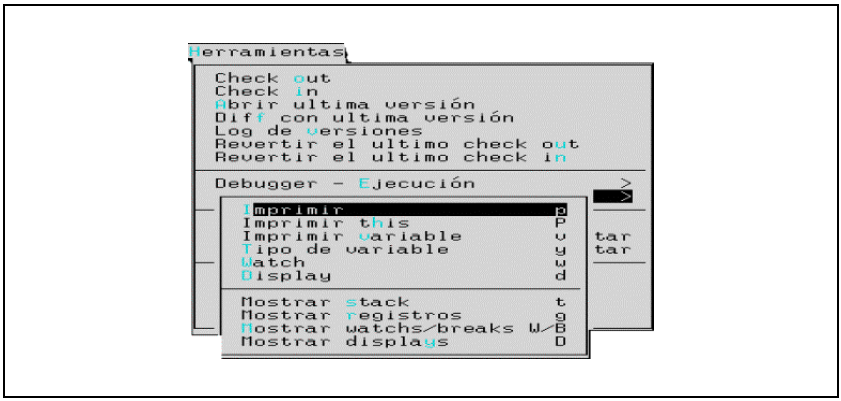
Al elegirla, se requerirá al usuario tipear el comando deseado. Una vez escrito éste y se presione la tecla [INGRESAR], Dalí pasará el comando al debugger y mostrará su salida en la ventana inferior (la ventana del debugger).

En uno de los anexos de este manual, se incluye una copia de la cartilla del gdb, el debugger usado por Dalí.

El comando rápido asociado es la tecla [ESPACIO].

Opciones del menú Debugger - Despliegue

Las opciones de este menú son las siguientes.



Opción: Imprimir

Esta opción muestra el valor de una variable o expresión, que el usuario escribe en la ventana de diálogo que aparece.

Si se escribe una expresión que incluye una llamada a función, Dalí evalúa toda la expresión ejecutando todas las funciones que sean necesarias. Así, es posible escribir expresiones como “FmSFld(fm0, NOMPROV)” y obtener el resultado correcto.

El comando rápido asociado es la tecla 'p'.

Opción: Imprimir this

Esta opción muestra el contenido del puntero a objeto “this”, y es usada solamente en programas escritos en C++.

El comando rápido asociado es la tecla 'P'.

Opción: Imprimir variable

Esta opción muestra el contenido de la variable cuyo nombre está debajo del cursor. Es una forma más rápida de la opción Imprimir, que se puede usar cuando se quiere inspeccionar una variable del programa y no una expresión.

El comando rápido asociado es la tecla 'v'.

Opción: Tipo de variable

Esta opción muestra el tipo (“int”, “char *”, etc.) de la variable cuyo nombre está debajo del cursor.

El comando rápido asociado es la tecla 'y'.

Opción: Watch

Esta opción pone un watchpoint sobre una variable o expresión. Cada vez que se reanude la ejecución del programa, Dalí se fijará si cambia el valor de esa variable o expresión: en cuanto cambie, detendrá la ejecución del programa, indicando el valor anterior y el valor nuevo de la variable que cambió.

El comando rápido asociado es la tecla 'w'.

Opción: Display

Esta opción pone un display sobre una variable o expresión. Cada vez que se avance en la ejecución del programa, se mostrará el valor actual de esa variable o expresión.

El comando rápido asociado es la tecla 'd'.

Opción: Mostrar stack

Esta opción abre una ventana con la lista de todas las llamadas a funciones que hay en el stack, indicando para cada una de ellas dónde se produjo la llamada y con qué parámetros.

El comando rápido asociado es la tecla 't'.

Opción: Mostrar registros

Esta opción muestra en una ventana el contenido de todos los registros de la CPU.

El comando rápido asociado es la tecla 'g'.

Opción: Mostrar watchs/breaks

Esta opción muestra en una ventana una lista de todos los breakpoints y watchpoints que están definidos.

Se puede borrar cualquiera de ellos presionando la tecla [INGRESAR] sobre la línea correspondiente.

Si se desea salir sin borrar ningún breakpoint ni watchpoint, se debe usar la tecla [FIN].

Hay dos comandos rápidos asociados: las teclas 'W' y 'B'.

Opción: Mostrar displays

Esta opción muestra en una ventana una lista de todos los displays que están definidos.

Se puede borrar cualquiera de ellos presionando la tecla [INGRESAR] sobre la línea correspondiente. Si se desea salir sin borrar ninguno, se debe usar la tecla [FIN].

El comando rápido asociado es la tecla 'D'.

Capítulo 11

IBuild

Introducción

IBuild es un utilitario que facilita al programador la tarea de mantenimiento de proyectos, especialmente cuando éstos están formados por una gran cantidad de archivos. Forma parte del entorno de desarrollo Dalí, y se lo puede considerar como un administrador de proyectos o compilador universal.

Todo proyecto abarca un conjunto de archivos fuentes que se van construyendo para formar módulos o archivos ejecutables. A su vez, estos archivos dependen entre sí, de forma tal que si se modifica uno de los archivos fuentes, se debe reconstruir todos los archivos que dependían de él.

Con IBuild, el programador no necesita recordar las dependencias entre los archivos, ni la forma en que se van construyendo (esto es, con qué secuencia de comandos). Cuando se modifica uno o más fuentes, simplemente hay que invocar a IBuild, y éste se encarga de reconstruir todos los archivos que dependan de los fuentes modificados, en caso de que haga falta, y así mantener al día un sistema de manera transparente al usuario.

Por cada proyecto, se tiene un archivo con extensión “.ib”, que contiene indicaciones sobre las dependencias entre los diversos archivos del proyecto. En otro archivo se guardan las reglas para construir los diversos tipos de archivos reconocidos por IBuild.

Lo que en definitiva IBuild provee es un programa de control que ejecuta tareas en tanto éstas se vuelven necesarias y siguiendo un orden determinado según la precedencia. Por ende, puede usarse para desarrollo de proyectos de envergadura. Dentro de las ventajas de esta herramienta están la sintaxis simple y de fácil comprensión y acostumbramiento, y la portabilidad 100% de los archivos .ib entre todas las plataformas para que IBuild contempla.

Construcción de módulos simples

Llámase módulo simple a un archivo que especifica la generación de un archivo ejecutable

individual. La forma general para especificar este tipo de módulos es:

```
nombre del módulo simple [ : lista de dependencias ] ;
```

Se indica de este modo que se procese un archivo solo. Opcionalmente se pueden establecer dependencias, indicadas luego de los dos puntos (:) en la línea anterior (hace falta separarlos por blancos del nombre del módulo y de la lista de dependencias).

Construcción de módulos compuestos

Llámase módulo compuesto a un archivo que especifica la generación de un conjunto de archivos ejecutables o bien biblioteca. La forma general para especificar este tipo de módulos es:

```
nombre del módulo compuesto
{
archivo1 [ : lista de dependencias ] ;
[ archivo2 [ : lista de dependencias ] ; ...]
}
```

Se indica de este modo que se procese el conjunto de archivos encerrados entre las llaves. Opcionalmente se pueden establecer dependencias, indicadas luego de los dos puntos (:) en las líneas necesarias, del mismo modo que en el caso de módulos simples.

Ejemplo:

```
CheckDeps=true;
ejemplo.exe {
a.fm ;
a.c : a.h a.fmh a1.c;
biblio1.a;
}
```

De esta manera se indica que el programa "ejemplo.exe" va a ser re-armado cada vez que se modifique alguno de los archivos "a.fm", "a.c" o "biblio.a".

A su vez, "a.c" debe ser recompilado si se modifica "a.h", "a.fmh" o "a1.c".

La función de la línea:

```
CheckDeps=true;
```

en el ejemplo, es la de indicar a IBuild que debe chequear las dependencias.

Debido a que esta característica puede ser deshabilitada seteando el valor "false" a la variable CheckDeps, es conveniente explicitar que se desea que las dependencias sean verificadas. De esta manera se hace un "override" del valor que pudiera tener esta variable anteriormente (por ejemplo en el archivo de reglas).

Una opción alternativa es setear la variable CheckDeps en el archivo $IDEAFIX $LANGUAGE/ibflags. Cuando IBuild comienza su ejecución determina la exis- tencia de

este archivo, y setea las variables que allí se encuentran.

Este archivo tiene el formato:

```
variable: :valor booleano:descripción 1
```

donde valor booleano es 1(true) o 0 (false). De esta manera es posible agregar la siguiente línea si se desea chequear las dependencias:

```
CheckDeps: :1:Chequear dependencias
```

o, en caso contrario:

```
CheckDeps: :0:Chequear dependencias
```

Recordar que estas variables pueden ser redefinidas posteriormente en las reglas o en algún otro archivo ".ib".

Es de destacar que los archivos .fm (en el ejemplo, el a.fm) se deben declarar antes que los módulos que dependan de dichos archivos .fm (en el ejemplo, el a.c).

Sentencias de inclusión

Las sentencias de inclusión de archivos permiten el procesamiento de otro u otros archivos a partir de una posición dada del archivo. Tienen la forma:

```
[ source ] fuente_ibuild [@ “directorio”]
```

Esto permite procesar el archivo indicado. Si se especifica el directorio, IBuild pasará a leer de dicho directorio específico ese archivo, así como subsiguientes archivos en tanto no se especifique otro directorio. Si se especifica *source*, se procesará el archivo fuente según los valores de variables del ambiente corriente (del mismo modo que lo hace el agregado de un punto “.” dentro del shell de Unix).

Ejemplo:

```
...
otro1.ib;
otro2.ib;
...
source variables.ib;
```

Sentencias de asignación

Se puede definir variables para utilizarlas luego en otra parte del archivo “.ib”. Para ello, se escribe una línea de la siguiente forma:

```
variable = expresión;
```

Para hacer referencia al contenido de una variable, se debe utilizar la sintaxis *“$variable”* o *“$(variable)”*.

Hay un conjunto de variables que están definidas con un valor por default. Estas variables que pueden redefinirse en los archivos “.ib” son las siguientes:

Variable	Descripción
OS	Sistema operativo
VCFILE	archivo para administración de versiones
SEARCHPATH	Directorios en los que IBuild busca los archivos pedidos

Figura 11.1 - Variables predefinidas

Comentarios

Se puede agregar comentarios de la misma forma que en las extensiones usuales del ANSI-C, es decir, usando // al comienzo de línea o /* ... */ encerrando el texto a comentar.

Si se usa “//”, todo el resto de la línea se considera como un comentario:

```
// Este es un comentario de una sola línea
```

Si se comienza una línea con usa “/*”, se considera como comentario todo el texto hasta encontrar el próximo “*/”. Por ejemplo,

```
/*
Este es un comentario
que ocupa varias líneas
de texto
*/
```

Tratamiento de extensiones

IBuild permite procesar archivos de acuerdo a las extensiones de estos. Para eso, existen reglas para el de tratamiento de extensiones. Estas tienen la siguiente forma:

```
on [extension] [simple | compound | ar | archive | list] “
extensión ”
[filename “archivo ”]
[checking | exporting “archivo ”]
[where bloque de asignación de flags ]
[run bloque de comandos ];
```

- *filename* indica el archivo a ...

- *checking* verifica que exista un archivo dado
- *exporting* indica el nombre de archivo a generar
- *where* permite asignar valores a flags
- *run* permite ejecutar comandos del shell de UNIX como por ejemplo archivos ejecutables

En estas reglas pueden utilizarse las siguientes macros:

Macro	Significado
%f	nombre del archivo
%p	path completo del archivo
%b	nombre del archivo sin el path ni la exten sión
%e	extensión del archivo, incluyendo el punto (.)
%d	nombre de destino del archivo (dado por la cláusula checking o exporting)
%l	la lista de todos los destinos de los módu los incluidos por un módulo compuesto
%v	lista de todos los destinos exportados
%t	lista de todos los archivos fuentes del módulo
%u	la lista de todos los destinos que fueron actualizados

Figura 11.2 - Macros para IBuild

Ejemplo de archivo .ib

Este es un ejemplo que construye un módulo ejecutable (a.exe) dependiente de un fuente en C (b.c), otro en Assembler (c.asm), y una biblioteca (d.lib) que a su vez se compone de dos programas en C (e.c y f.c), y uno en Assembler (g.asm).

```
a.exe {
b.c ;
c.asm ;
d.lib {
e.c ;
f.c : f.h ;
g.asm ;
}
}
```

Otro ejemplo, definiendo tratamiento de las extensiones “c” y “exe”:

```
on extension simple “c”
exporting “%b.o”
where
{
Debug is “-O -g DDEBUG”;
Optimize is “-O2";
}
run “$CC $CFLAGS %(Debug) %(Optimize) %f -o %d”;
on extension simple “exe”
checking “%f”
where
{
Debug is “-g”;
Optimize is “-s”;
}
run ”$CC $LDFLAGS %(Debug) %(Optimize) %f”;
```

Argumentos en la línea de comando

La sintaxis del utilitario IBuild en la línea de comando es la siguiente:

```
ib [ flags ] archivo ...
```

Los posibles flags (que pueden abreviarse usando la primera o más letras siempre que no resulte ambiguo) son los siguiente:

debug	Construye para debugging.
-flag [*flags*]	Construye usando la lista de flags indicada (cada una puede ser true o false)
profile	Construye para profiling.
noflag	Niega los flags indicados (hasta el próximo blanco). Sólo se puede usar con los flags que figuran en la parte superior de la tabla.
-check	Chequea las dependencias
continue	Continúa la construcción aunque se produzca un error.
-project [proyecto]	Construye usando el proyecto indicado
-show	Muestra los comandos sin ejecutarlos
-unconditional	Construye todo, incondicional mente
-verbose [num]	Nivel verborrágico (de 1 (default) a 3)
version versión	Construye una versión específica

Figura 11.3 - Opciones de invocación de IBuild

Se puede utilizar la primera letra de cada opción precedida por el guión (-). Así, se puede invocar a Ibuild pidiendo compilación incondicional del archivo *uno.ib* usando el proyecto *primero*, mostrando verborragia de nivel 1, con la siguiente línea de comando:

```
ib -v -p primero uno
```

etcétera.

Variables de ambiente

IBuild busca los valores de nombres de directorio y proyecto en variables de ambiente:

- al correr, siempre procesa el archivo ibrules.ib en el directorio de instalación del IBuild, indicado en la variable de ambiente $DATADIR.
- si la variable de ambiente “IPROJECT” está definida, busca en el directorio al que apunta.

Notar que IPROJECT es una variable que puede señalar un archivo o bien un directorio.

Otras sentencias

La sentencia *echo* imprime en las salida standard, y es de la forma:

```
[echo | abort] “texto”;
```

Si se utiliza echo, se produce la salida del texto indicado e IBuild continúa con los comandos siguientes. Si en cambio se utiliza abort, Ibuilld imprime el texto indicado y para la ejecución. Esta última manera es muy usual para definir mensajes de error.

La sentencia *shell* produce la ejecución de un comando del sistema operativo. La forma de dicha sentencia es la siguiente:

```
shell [ verbose | abort ] “comando”;
```

Si se usa la opción *verbose*, se muestra por la salida standard el comando a ejecutar. Si no, no se muestra. Con la opción *abort*, se interrumpe la ejecución de Ibuild en caso de que el comando indicado arroje algún error.

Esqueleto típico y recomendaciones

Usualmente, la estructura general de un archivo “.ib” suele ser la siguiente:

```
[ sentencias de inclusión; ]
[ sentencias de asignación; ]
[ sentencias de ejecución; ]
módulo simple [ : lista de dependencias ] ;
módulo compuesto {
módulo simple [ : lista de dependencias ] ;
.
```

```
.
.
[ módulo simple [ : lista de dependencias ] ;]
};
```

Es recomendable no incluir más que un módulo compuesto por archivo .ib .

Proyecto

Para que se tomen las reglas default, en la primera línea del archivo .ib debe figurar necesariamente una línea de proyecto, de la siguiente forma:

```
project [directorio | path completo] (lista de flags) ];
```

Un ejemplo de archivo de proyecto es el siguiente, en el que se contemplan varias extensiones típicas en una instalación de Ideafix y GCC bajo Unix:

```
/*Ejemplo */
AR = "ar"; // flags usados más abajo
ARFLAGS= "ruv";
CC = "gcc";
CPP = "gcc";
LD = "gcc";
LEX = "flex";
OBJ ="OBJO";
YACC = "bison";
YACCFLAGS = "-y -d";
SEARCHPATH = ":include:$ISRC/include";
STDINCLUDES = "-I. -Iinclude -I$ISRC/include";
VCFILE = "%pRCS/%b%e,v";
objdir = true;
if ($WinApp == true) {
ExtLib = true;
}
if ($Essentia == true and $Ideafix == true)
abort "ERROR: Redefinición simultánea de Essentia e Ideafix";
if ($Debug == true) {
Optimize = false;
E = "g";
OBJ = "OBJG";
}
if ($install == true)
if ($Essentia == true)
BIN = "$ISRC/rbin/"; // elige directorio apropiado
y else
BIN = "$ISRC/bin/";
TCPLIBS = "-linet"; // más opciones
ISRCLIBS = "-lidea$E";
on "h"; // declaraciones ...
on "fmh"; // que serán definidas como reglas después
on "rph";
on "o"
exporting "%f";
on "fm"
checking "$BIN%P/%b.fmo" // chequeael time stamp; y si dan
bien, ...
run {
```

```
"fgen %f";
"mv %b.fmo $BIN%P || true";
};
on "c"
where { // definiciones de variables locales
objdir is "$OBJ/"; // directorio para el .o
Debug is "-g -O -DDEBUG "; // opciones de debug y optimización
Optimize is "-O2 ";
exporting "%(objdir)%b.o"
run "$CC %(Optimize)%(Debug)$STDINCLUDES $CFLAGS -c %f -o
%(objdir)%b.o";
on "cc"
where {
objdir is "$OBJ/";
Optimize is "-O2 ";
Debug is "-g -O -DDEBUG ";
}
exporting "%(objdir)%b.o"
run "$CPP %(Optimize)%(Debug)$STDINCLUDES $CPPFLAGS -c %f -o
%(objdir)%b.o";
on "l"
where {
objdir is "$OBJ/";
Debug is "-g -O -DDEBUG ";
Optimize is "-O2 ";
}
exporting "%(objdir)%b.o"
run {
"$LEX $LEXFLAGS %f";
"mv lex.yy.c %b.cc";
"$CPP %(Optimize)%(Debug)$STDINCLUDES $CPPFLAGS -c %b.cc -o
%(objdir)%b.o"; rm -f lex.yy.c %b.cc";
};
on "y" // cómo tratar un .y de yacc
where {
objdir is "$OBJ/";
Debug is "-g -O -DDEBUG ";
Optimize is "-O2 ";
}
exporting "%(objdir)%b.o"
run {
"$YACC -p %b_ $YACCFLAGS %f";
"cmp -s y.tab.h %b.h || mv y.tab.h %b.h";
"mv y.tab.c %b.cc";
"$CPP %(Optimize)%(Debug)$STDINCLUDES $CPPFLAGS -c %b.cc -o
%(objdir)%b.o"; "rm -f y.tab.[ch] %b.cc";
}
on "sh" // cómo tratar un .sh (shellscript)
exporting "$BIN%f"
run {
"cp %f $BIN%b";
"chmod 777 $BIN%b";
};
on compound "" // extensión nula
where {
Optimize is "-s ";
Debug is "-g ";
Shared is "_s";
Ideafix is "-lixdb$E%(Shared) ";
Essentia is "-lesdb$E%(Shared) ";
}
exporting "$BIN%f$E" // tratar como archivo ejecutable
```

```
run {
"$LD $LDFLAGS %(Optimize)%(Debug)-o %d %l $LIBS
-L$ISRC/lib%(Ideafix)%(Essentia) $ISRCLIBS %(tcp)$STDLIBS";
};
on compound "exe" // otros ejecutables
where {
Optimize is "-s ";
Debug is "-g ";
Shared is "_s";
Ideafix is "-lixdb$E%(Shared) ";
Essentia is "-lesdb$E%(Shared) ";
root is "su root -c 'chown root %d; chmod ugo+s %d'"; }
exporting "$BIN%f$E"
run {
"$LD $LDFLAGS %(Optimize)%(Debug)-o %d %l $LIBS -L$ISRC/lib
%(Ideafix)%(Essentia)$ISRCLIBS %(tcp)$STDLIBS";
"%(root)";
};
on ar "a" // cómo tratar un .a
exporting "%p%b$E.a"
run $AR $ARFLAGS %d %u";
on archive "lib" // cómo tratar un .lib
exporting "%p%b$E.a"
run"$LIBRARIAN $LIBFLAGS %d %u";
```

Este utilitario permite generar automáticamente un archivo .ib (especificación de módulos para IBuild) a partir de archivos fuentes indicados. La sintaxis es las siguiente:

```
genib [ -h | -d | -v [ | -i ] ] módulo_compuesto
módulo_simple1 módulo_simple2 ...
```

Este comando envía el resultado por default a la salida estándar del sistema operativo. Si se desea grabar en un archivo, debe redireccionarse de la manera usual (usando >).

El significado de los flags es el siguiente:

- -h o -? : muestra la sintaxis del comando
- -d : pone la lista de dependencias para cada módulo simple

Si se usa el administrador de versiones SCCS, los siguientes flags están también disponibles:

- -v: obtiene el número de versión de cada módulo
- -i : ignora aquellos módulos de los cuales no puede obtener el número de versión; esta opción sólo tiene sentido si se especifica el flag -v.

Capítulo 12

Comandos rápidos

Menú del sistema

Castellano	Inglés	Tecla
Acerca de Dalí	About Dalí	Por menú
Calculadora	Calculator	Por menú
Imprimir pantalla	Print screen	Por menú
Ver juego de caracteres	Show character set	Por menú
Redibujar pantalla	Repaint desktop	Por menú
Ejecutar comando del S.O.	Execute O.S. command	[SUSPENDER]
Ir al shell	Go to shell	[META]-z

Para entrar al menú del sistema, usar [CTRLX]-[ESPACIO].

Manejo de archivos

Castellano	Inglés	Tecla
Archivo nuevo	New file	Por menú
Abrir	Open	[SERVICIOS]
Re abrir	Reopen	[META]-[SERVICIOS]
Salvar	Save if modified	[AYUDA_APL]
Salvar incondicional	Save always	[META]-[AYUDA_APL]
Salvar como ...	Save as ...	[META]-s
Salvar todo	Save all	Por menú
Insertar archivo	Insert file	[META]-i
Cambiar de directorio	Change dir	[META]-c
Abrir un tag	Open tag	[META]-T

Ir a archivo	Go to file	[META]-g
Seguir archivo	Trace file	[META]-y
Abrir ambiente	Open workspace	Por menú
Salvar ambiente	Save workspace	Por menú
Salir de Dalí	Exit	Por menú

Edición

Castellano	Inglés	Tecla
Deshacer	Undo	[DESHACER]
Principio de bloque	Begin block	[META]-[META]
Dessmarcar bloque	Unmark block	[META]-[FIN]
Ejecutar filtro del S.O.	Execute O.S. filter	[META]-'!'
Cortar	Cut	[META]-[ELIMINAR]
Copiar	Copy	[META]-[INSERTAR]
Pegar	Paste	[META]-[INGRESAR]
Borrar	Clear	[META]-[ESPACIO]
Suprimir	Delete	[META]-D
Agregar atributo	Set attribute	[META]-'-'
Suprimir atributo	Delete attribute	[META]-'-'
Desplazar a izquierda	Shift left	[ALT]-'['
Desplazar a derecha	Shift right	[ALT]-']'
Borrar línea	Delete line	[META]-d
Borrar hasta fin de línea	Delete rest of line	[META]-e
Borrar palabra	Delete word	[META]-w
Borrar tabulador a izquierda	Delete tab	[META]-[RETROCESO]
Principio de línea	Beginning of line	[META]-[CURS_IZQ]
Fin de línea	End of line	[META]-[CURS_DER]
Palabra anterior	Previous word	[ALT]-[CURS_IZQ]
Próxima palabra	Next word	[ALT]-[CURS_IZQ]
Página anterior	Previous page	[PAG_ANT]
Próxima página	Next page	[PAG_SIG]
Principio de archivo	Beginning of file	[META]-[PAG_ANT]
Fin de archivo	End of file	[META]-[PAG_SIG]
Principio de pantalla	Beginning of screen	[META]-[CURS_ARR]
Fin de pantalla	End of screen	[META]-[CURS_ABA]
Al medio de la pantalla	Center of screen	[META]-m
Función anterior	Previous section	[META]-'['

Función siguiente	Next function	[META]-’]’
Principio de ámbito	Beginning of scope	[PAG_IZQ]
Fin de ámbito	End of scope	[PAG_DER]
Insertar llaves	Insert braces	[META]-b
Unir la siguiente línea	Join line	[META]-j
Insertar/sobreescribir	Insert/overwrite	[INSERTAR]
Insertar línea arriba	Insert line above	[META]-O
Isnertar línea debajo	Insert line below	[META]-o
Insertar un tabulador	Insert tab	[TAB]

Búsquedas y reemplazos

Castellano	Inglés	Tecla
Búsqueda	Find	[META]-f
Repetir búsqueda	Find again	[META]-n
Búsqueda incremental	Incremental find	[ALT]-i
Ocurrencia anterior	Previous occurence	[META]-p
Reemplazo	Replace	[META]-r
Repetir reemplazo	Replace again	[CTRLX]-r
Ir a línea número	Go to line number	[ALT]-g
Hojear archivo	Source browser	[ALT]-’/’
Hojear expresión en contexto	Context expression browser	[ALT]-’.’
Hojear una expresión	Expression browser	[ALT]-’>’
Poner marca	Set mark	[META]-X
Ir a la marca	Go to mark	[META]-x

Opciones

Castellano	Inglés	Tecla
Modo dibujar	Draw mode	[ALT]-G
Modo Troff	Troff mode	[META]-F
Modo comando	Command mode	[ALT]-C
Caracteres de control	Show control chars	[META]-l
Insertar carácter ASCII	Insert ASCII char	[ALT]-’#’

Herramientas

Castellano	Inglés	Tecla
Check out	Check out	Por menú
Check in	Check in	Por menú
Abrir última versión	Open last version	Por menú
Log de versiones	Version log	Por menú
Revertir el último check out	Revert last check out	Por menú
Revertir el último check in	Revert last check in	Por menú
Debugger - ejecución	Run debugger	Por menú
Debugger - despliegue	Display debugger	Por menú
arrancar debugger	Start debugger	[ALT]-d
Parar debugger	Stop debugger	[ALT]-D
Construir	Build	[PROCESAR]
Generar ib	Generate IBuild file	[META]-[PROCESAR]
Preferencias	Preferences	Por menú
Generar tags	Build tags	Por menú

Debugger - Ejecución

Castellano	Inglés	Tecla
Ejecutar	Run	r
Continuar	Continue	c
Próxima	Next	n
Próxima Inst. Asm	Next Asm. Instruction	n
Paso (entra en funciones)	Step (over functions)	s
Paso Inst. Asm	Step Inst. Asm	S
Insertar Breakpoint	Insert Breakpoint	[INSERTAR]
Borrar breakpoint	Delete Breakpoint	[BORRAR]
Comando de Debugger	Debugger Command	[ESPACIO]

Debugger - Despliegue

Castellano	Inglés	Tecla
Imprimir	Print	p

Imprimir ’this’	Print ’this’	P
Imprimir variable	Print variable	v
Tipo de variable	Type var.	v
Watch	Watch	w
Display	Display	d
Mostrar stack	Stack frame	t
Mostrar registros	Display registers	g
Mostrar watchs/breaks	Display watchs/breaks	W/B
Mostrar displays	Displays	D

Ventanas

Castellano	Inglés	Tecla
Cambiar tamaño	Size	[ALT]-Z
Mover	Move	[ALT]-M
Agrandar/achicar	Zoom	[IGNORAR]
Próxima ventana	Next	[ALT]-n
Ventana anterior	Previous	[ALT]-p
Cerrar	Close	[REMOVER]
Cerrar todo	Close all	[META]-[REMOVER]
Seleccionar ventana	Select	[ALT]-l
Mosaico	Mosaic	[ALT]-m
Cascada	Cascade	[ALT]-c

Ayuda

Castellano	Inglés	Tecla
Ayuda de ayuda	Help on help	Por menú
Indice de ayuda	Help index	[AYUDA]
Características	Characteristics	Por menú
Ayuda de rutina	Routine help	[META]-[AYUDA]
Abrir ayuda de rutina	Open routine help	[META]-H

Capítulo 13

Apéndice A

Glosario de términos más comunes

En este apéndice se ofrece una lista de las palabras y expresiones más comunes que fueran usados en el presente manual, y por ende que tengan relación con el entorno de desarrollo Dalí. Se da también la equivalencia en inglés a continuación de cada voz. (En el caso en que ésta no difiera, no se repite.)

Archivo (file)

Información o conjunto de datos relacionados que se mantiene almancenada de manera independiente de la memoria de una computadora. Lo más común es guardar archivos en disco rígido, diskettes o cintas, para su posterior recuperación. Dalí graba y lee archivos con tal de que éstos sean de formato ASCII.

ASCII

Siglas de American Standard Code for Information Interchange, o código estándar americano para intercambio de información. Es un código que asigna un número distinto a cada carácter que pueda aparecer en la pantalla, es decir, a las (letras, dígitos, símbolos, e incluso otros códigos especiales comom fin de transmisión, etc). Un archivo se encuentra en formato ASCII cuando la secuencia de bytes que lo componen reflejan literalmente la secuencia de caracteres del archivo. Dalí graba y lee textos en este formato, lo cual hace posible el intercambio de información con otros editores o utilitarios..

Bloque (block)

Secuencia de caracteres y líneas en un texto que se está editando en un momento dado. El bloque abarca pues todo lo que hay desde un punto hasta otro del texto., Dalí marca el bloque activo sobresaltándolo con otro color respecto del resto del texto, y permite operaciones de

bloque tales como borrar, copiar y pegar bloques.

C

Lenguaje de programación de mediano nivel, muy popular. El C fue ideado por B. Kernighan y D. Ritchie en 1969 y se suele utilizar hoy en día para el desarrollo de utilitarios, manejo de bases de datos y aplicaciones administrativas. Dalí provee de un interface con C a través de su módulo IBuild.

Compilador (compiler)

Utilitario que permite a partir de una especificación en lenguaje de alto nivel, obtener código ejecutable en un lenguaje de bajo nivel (lenguaje de máquina). Desde el entorno Dalí se puede acceder a virtualmente cualquier compilador que esté instalado en la computadora.

Debugger

Módulo o utilitario que permite ejecutar un programa mientras se observan valores del contenido de la memoria de la máquina y registros de la unidad central de proceso, así como poder ejecutar las instrucciones paso a paso, con el objeto de analizar y corregir posibles errores en la programación. Dali provee una interfaz al debugger gdb de GNU.

Dependencia (dependency)

Relación entre dos archivos que establece que uno debe ser compilado o procesado antes que el otro. Con el IBuild se pueden especificar dependencias de un archivo con otro u otros.

Editar (to edit)

La tarea de modificar un documento en una o todas sus partes. Los documentos que Dalí maneja son textos de cualquier índole (escritos, programas, etc). El editor de Dalí constituye una gran parte del entorno. Un editor no es otra cosa que un programa utilitario que permite o facilita la tarea de editar un documento.

IBuild

Utilitario incluido en el entorno Dalí que permite construir módulos o archivos variados según las especificaciones del usuario o programador, tales como lista de componentes, dependencias de qué módulos deben generarse antes, etc.

Ideafix

Nombre del administrador de bases de datos relacionales de Intersoft. El entorno Ideafix permite definir y usar tablas, forms o formularios, informes, menúes y consultas asociados

todos a una base de datos dada. El entorno Dalí desde el comienzo fue un accesorio a Ideafix, hasta constituir un entorno independiente por motivos de crecimiento propio. Provee maneras de vínculo con Ideafix, por ejemplo, a través de IBuild.

Instrucción (instruction)

Tarea elemental ejecutable por la computadora. Generalmente la ejecución de un programa entero o aplicación involucra la ejecución de millones o billones de instrucciones elementales. Dalí permite seguir paso a paso las instrucciones assembler así como en lenguaje de alto nivel, a través de la interface al gdb (debugger de GNU).

Interface

Dícese del módulo o rutina que provee de la comunicación e interacción usuario/ sistema, es decir, la parte ejecutable de la aplicación que se ocupa de tomar los pedidos del usuario y mostrar en pantalla las respuestas o resultados. Un ejemplo de interfaz es el menú desplegable, usado por Dali.

Lenguaje de programación (programming language)

Código simbólico que permite especificar programas para ser finalmente ejecutados por una computadora. El entorno Dalí ofrece facilidades específicas para trabajar con algunos de ellos en el sentido de que es un editor orientado al programador que use alguno de dichos lenguajes.

Lenguaje de programación de alto nivel (high level programming language)

Dícese de los lenguajes de programación cuyas instrucciones o primitivas se vinculan más con el pensamiento del programador o la mayoría de éstos, y por ende necesita ser transformado en otro más comprendible por la computadora (lenguaje de máquina). Entre los lenguajes de programación más conocidos y usados están el C, C++, Pascal y Basic. Dalí provee de un debugger que permite trabajar paso a paso con las instrucciones en el lenguaje de alto nivel compilado.

Lenguaje de máquina/de bajo nivel (machine language/low-level language)

Dícese del lenguaje específico comprendido directamente por la unidad central de proceso (CPU) que permite ejecutar programas. El lenguaje de máquina se compone exclusivamente por secuencias de números binarios (es decir, escritos en base de numeración 2). Una manera un poco más simbólica y comprensible más fácilmente por los programadores es el lenguaje Assembler. Dalí permite trabajar con un debugger que procesa instrucciones a nivel lenguaje de máquina.

Menú (menu)

Lista de opciones destinada a ser mostrada en pantalla para que el usuario pueda seleccionar la opción deseada en cada momento, tal como alinear texto a izquierda, grabar un archivo o abrir una nueva ventana. El entorno Dalí trabaja con opciones que aparecen agrupadas en menúes para su mejor y más fácil uso.

Módulo (module)

Parte importante de un programa o aplicación, generalmente destinada a una tarea específica. Por ejemplo, el debugger, el editor, IBuild, son módulos de Dalí.

Red de comunicación (communications network)

Conjunto de computadoras interconectadas. El entorno Dalí funciona en equipos individuales como bajo redes ya que funciona bajo el sistema operativo Unix.

Sistema operativo (operating system)

Programa o módulo que se carga inicialmente en la memoria de la computadora, totalmente o por partes, que permite que los progrramas escritos en lenguajes de alto o bajo nivel puedan usar de una forma estándar los recursos de ésta. Entre los sistemas operativos más comunes de hoy en día están Unix, DOS, Windows, OS/ 2 y Mac. Dalí está preparado para funcionar en casi cualquier versión de Unix

Tag

Entrada de manera de palabra en un diccionario, empleada para el fácil o rápido acceso a las distintas porciones de un documento dado. El editor de Dalí ofrece un manejo de tags orientado fundamentalmente a la edición de programas.

Unidad central de proceso (central processing unit, CPU)

Parte física de la computadora que procesa instrucciones a nivel de lenguaje de máquina, y se ocupa por ende de dirigir todo el procesamiento del equipo.

Ventana (window)

Porción de la pantalla destinada a un uso específico tal como mostrar o permitir la edición de un texto, despliegue de errores en un programa, etc. El entorno Dalí permite tener abiertas varias ventanas a la vez, con un mismo o distintos documentos.

Capítulo 14

Apéndice B

Problemas comunes

Esta es la sección de problemas comunes y posibles soluciones. Probablemente algunos problemas no estén contemplados aquí. Para más información se recomienda consultar el archivo ‘leame’ que se provee con Dalí.

PROBLEMA

No lee correctamente el archivo 'dalirc'.

SOLUCION

Asegúrese de que no figuren blancos a la derecha de cada entrada. Si alguna entrada es rechazada por Dali.

PROBLEMA

Las funciones de delimitación de ambito ([PAG_IZQ] y [PAG_DER]) no funcionan.

SOLUCION

Asegúrese de que en su archivo 'dalirc' figure la sentencia 'MatchTab'.

Asegúrese de que este definida correctamente.

La sintaxis es la siguiente:

```
MatchTab=a1 c1:a2 c2 ...
```

- a1 y a2 son los caracteres con los que comienza un ambito.

- c1 y c2 son los caracteres con los que termina un ambito.

Ejemplo:

```
MatchTab=( ):[ ]:{ }
```

PROBLEMA

La función 'Hojear archivo' ([CTRLX]-'/') no hace nada

SOLUCION

Asegúrese de que en su archivo 'dalirc' figure la sentencia 'SourceBrowser'.

Asegúrese que exista una entrada para la extension del archivo que esta editando.

Asegúrese de que este definida correctamente; recuerde que la sintaxis es la siguiente:

```
SourceBrowser=rexp:titulo:exp1:fmt1:exp2:fmt2
```

donde

- rexp: extension de archivo para la cual se quiere configurar esta opcion. Si se quiere configurar para mas de una extension a la vez, se puede usar una expresion regular que las incluya.
- titulo: titulo que aparecera en la ventana.
- exp1 y exp2: expresiones regulares que se desea buscar.
- fmt1 y fmt2: expresiones que se desea mostrar en la ventana. Si no se indican, Dali mostrara la linea completa donde encontro exp1 o exp2.

PROBLEMA

El panel para seleccionar archivos se blanquea

SOLUCION

Esta situación se presenta en circunstancias aun no determinadas, y sólo se conoce que suceda en Interactive Unix. Aunque el panel este en blanco, igual se puede seleccionar un archivo escribiendo el path completo.

Este error desaparece cuando se sale de Dalí y se vuelve a entrar.

PROBLEMA

Despues de hacer un checkin o un checkout, la ventana no actualiza la leyenda 'Solo Lectura' de la línea de status.

SOLUCION

Esto ocurre en algunos sistemas operativos (p.ej, SCO Unix) únicamente. Sólo basta re abrir el archivo para actualizar la linea de status. ([META]-F3) (Este problema fue corregido en la version 1.5.5)

PROBLEMA

El history de cada comando aparece con entradas de otros comandos y/o con menos entradas de las que suele tener o directamente el comando no tiene history.

SOLUCION

Esto puede ocurrir si se cambia la version de Dali, ya que es posible que entre distintas versiones, el archivo donde se guardan los ultimos valores de los comandos sea incompatible. Para solucionar este problema, basta borrar el archivo "history" del subdirectorio ".dali" de cada usuario.

PROBLEMA

El panel de selección de archivos no contiene todos los archivos.

SOLUCION

Los archivos que no aparecen fueron modificados o creados en una terminal diferente, o por un usuario diferente. Basta con cerrar el panel y volver a abrirlo ya que la actualización se hizo después de ser abierto éste.

Capítulo 15

Apéndice C

Historia de versiones

En esta sección se hace una revisión de las principales características introducidas en las últimas versiones de Dalí.

Dali v1.5.7

En esta versión se corrigieron varios bugs, ademas se cambió el comportamiento del comando 'Token Completion'. Tambien se cambió la combinación de teclas utilizadas para invocar a este comando y al comando 'Unir lineas'. Las nuevas combinaciones son [CTRLX]-TAB y [CTRLX]-j, respectivamente.

Dali v1.5.6

Versión de Dali, liberada junto con la versión 4.3 de Ideafix. Es posible que en esta versión, y probablemente desde la v1.5.5, exista un bug relacionado a los archivos preservados.

Dali v1.5.5.3

Comando 'Token Completion'. Con esta opción se muestran las posibles terminaciones de la palabra escrita hasta el momento. [META]-[TAB]

Dali v1.5.5.2

Correción de varios bugs. Ahora el FileSelector funciona bien.

Dali v1.5.5

Modo TeX. Esta opción se encuentra en el Menu de Herramientas y permite leer y grabar archivos con extensiones '.tex' o '.latex'.

En el archivo de configuración de Dali, se cambio la opción 'IbFlags' por BuildFlags, y

ademas se agreó la opción 'BuildCmd', la cual especifica el comando que se debe utilizar para la construcción de proyectos (por omisión se invoca al utilitario 'ibuild')

En el archivo de configuración de Dali, se agreó la opción 'DiffCmd', la cual especifica el comando que sera utilizado por los comandos que efectuan diferencias entre archivos. Por omisión se invoca al utilitario 'diff' sin argumentos.

Dali v1.5.4

Comando 'diferencias'. Esta opción figura en el menu de Herramientas. Con esta opción es posible observar las diferencias entre dos archivos. El comando toma las dos ventanas mas recientes del editor, y entra en un modo en el que con [CURS_ARR] y [CURR_ABA] se puede navegar entre las diferencias. Con [FIN] se abandona este modo.

Ahora, ademas de indicar que un archivo esta bloqueado, se informa que usuario lo esta bloqueando.

Dali v1.5.3

Ahora es posible ejecutar comandos y programas usuarios del Window Manager desde Dali. Esta opción figura en el menu del sistema, y el comando rapido asociado es [META]-[SUSPENDER]

La opción "Hojear expresión" ahora muestra, para fuentes C/C++, la función en la que se encuentra cada línea que satisface la expresión. Este comportamiento es configurable mediante la cláusula FuncExpBrowser en el archivo de configuración "dalirc" (sus valores posibles son on y off).

Dali v1.5b

La interfaz con el usuario se ha modificado en los siguientes aspectos.

Grep de archivos.

Se agrega la opción de diff con la ultima version administrada.

Dali v1.5

El modo "vi" ya no esta disponible, porque fue reemplazado por funciones de edicion de Dali.

Dali v1.4

Debe crearse el directorio '$HOME/.dali' para cada usuario.

En versiones anteriores de Dali, se debia definir las variables de ambiente DALITAGS y DALIMAN. Ahora ya no son necesarias, porque es posible configurar estas opciones en el archivo dalirc.

Si ya existia una version anterior de Dali, se deben borrar los archivos .dalirc, .dalihist y

.dalidsktop que pueden haber quedado en el directorio $HOME de los usuarios.

Capítulo 16

Introducción

Generalidades

Este manual explica en que consiste y como se utiliza el lenguaje standard "**SQL"** (Structured Query Language), y las características propias de su implementación dentro del contexto de **IDEAFIX**, denominada *iql* (**IDEAFIX** Query Language). Si bien el término *query* significa "consulta", esta herramienta hace mucho mas que eso: tiene por objeto facilitar el manejo integral de Bases de Datos relacionales (relational Data Bases), incluyendo su creación, mantenimiento y control de acceso; agregado, supresión y modificación de datos, y su presentación en forma de reportes, ya sea impresos o desplegados por pantalla.

Estructura

Esta guía se halla dividida en tres partes, de las cuales la primera -que abarca la Introducción y los cuatro primeros capítulos- presenta un panorama general, proveyendo los conceptos fundamentales que capaciten al usuario para seleccionar y presentar información obtenida de bases de datos ya existentes. La segunda parte, que consta de los capítulos 21 y 22, introduce conceptos avanzados del lenguaje de consulta, dando por ende la posibilidad de un manejo más elaborado de los daros, como por ejemplo la obtención de totales y subtotales. En la última parte (caps. 23 al 26) se explica como manipular datos; es decir, agregar o eliminar información a las tablas, como crear nuestras propias Bases de Datos, trabajar con índices, usar la interfaz de edición de consultas, etc.

Las tres partes conforman una progresión de los temas más sencillos hacia los más complejos,

y se recomienda al lector poner en practica los conocimientos a medida que se vayan adquiriendo, a través de la ejercitación sobre las tablas ejemplo que se proveen con el sistema. De tal manera se asegurará la cabal compresión de cada tema antes de proseguir con el siguiente.

Algunas consideraciones iniciales

Bajo la denominación genérica de “Sistemas de Información”, y muy frecuentemente con la sola palabra “sistema” , se suele aludir a una gama de conceptos muy variada, susceptible de confundir con cierta frecuencia al lego .... y alguna vez también vez también al experto.

Comencemos reconociendo el significado original de “sistema”, venerable vocablo de origen griego -era inevitable, ?verdad?- que designa a un conjunto de elementos dispuestos con arreglo a ciertas leyes. Así un sistema solar, a cuerpos celestes, y el sistema cardiovascular a los órganos encargados de proveer irrigación sanguínea. Los sistemas de información existen desde mucho antes de la invención de la computadora: una biblioteca lo es, y de los más antiguos. Resulta obvio entonces que lo que se quiere decir hoy en día al referirnos a *sistemas* es a **sistemas mecanizados de información**, o dicho de otra manera, al Procesamiento Electrónico de Datos (EDP, o “Eletronic Data Processing”).

Dentro de nuestro sistemas -ya queda aclarado cuales son los nuestro- existen dos grandes grupos de componentes: aquello que se denomina *hardware* (literalmente “ferretería”), constituido por las unidades físicas, tales como la Unidad Central de Proceso o CPU, las terminales, acunado por contraposición de los adjetivos “soft” (blando) y “hard” (duro) en ingles. El *software* es el conjunto de estructuras *lógicas* compuestas por instrucciones o comandos que gobiernan el funcionamiento y la intercomunicación de los componentes del hardware; esto es, lo que generalmente llamamos *programación*.

Existen muchos lenguajes de programación, y Guía se refiere a uno de ellos, llamado **SQL** (Structured Query Language), específicamente diseñado para facilitar el acceso a la información almacenada en el sistema; manipularla de la forma que nos convenga (v.gr. seleccionando ciertos datos, reorganizándolos, etc.), y por último disponerla para ser vista por el usuario, ya sea en una pantalla o un listado impreso.

Al igual e los idiomas de uso cotidiano, los *lenguajes de programación* deben ajustarse a ciertas reglas de *sintaxis*, que tienen que ver con el uso y la disposición de determinadas *palabras clave* (en ingles *keywords*) y símbolos de puntuación, a los cuales se les asigna un significado especial. Dichas palabras y símbolos se agrupan en *sentencias*, cuyo componente más importantes es el termino que designa la acción (el “verlo” o nombre de comando) que implica ejecutarlo. Una vez más, vemos la analogía que existe entre un lenguaje de computación y el uso cotidiano, donde el verbo es la parte más importante de la oración.

El propósito fundamental del SQL es *interrogar* al sistema (tal el significado de “query” en ingles: interrogación o consulta), fijando tanto los aspectos formales de la pregunta (reglas de sintaxis) como las características propias de la respuesta (disposición y ordenamiento de los datos de salida). Es el momento de señalar que como suele ocurrir con la mayoría de los

lenguajes, es SQL implementado en **IDEAFIX** (IQL) consta de una gran estructura básica que abarca los aspectos normalizados, con más algunas facilidades adicionales propias, que pueden diferir de las implementaciones en otros entornos. No obstante, los conceptos esenciales son siempre validos, y las discrepancias se refieren a cuestiones de detalle.

La consulta tiene por sujeto un conjunto de información contenido en lo que se denomina **Base de Datos** (DB), de la cual se procura extraer solamente aquello que resulte de nuestro interés, prescindiendo de los datos que *en esta oportunidad* no nos sean útiles; ello no quita que puede ser necesarios en otros casos. La información dentro de un DB se halla organizada en archivos o tablas, que pueden tener diversas características; pero dentro de una tabla en particular los datos se hallan agrupados en **registros**, de modo tal que todos ellos responden a idéntica estructura lógica. Esto significa que cada uno de ellos contienen una determinada cantidad de **campos**, en un cierto orden, y que cada campo tiene una **longitud** y un **formato** prefijados.

Un campo es un dato básico tal como un nombre, un código, un importe, una descripción. El conjunto de campos referente a un determinado item o rubro constituye un registro, y un conjunto de registros de similares características conforman una tabla o archivo (file). Si nos remontamos a las épocas heroicas en las cuales el único medio de almacenamiento era la tarjeta perforada, esta se asimilaba a la “ficha” de un *archivo manual* (gaveta o fichero = file), en la que se registraba -por ejemplo- la información de un cliente.

En cosa de un cuatro de siglo, el avance tecnológico nos ha permitido desentendernos de elementos tales como la tarjeta perforada, una de cuyas limitaciones más serias era la necesidad del proceso secuencial. Esto significa que para acceder aun registro (tarjeta) era necesario pasar por todos los que lo precedían.

Así mismo, la clasificación (reordenamiento) de lotes voluminosos era un proceso largo, tedioso y expuesto a errores. La moderna tecnología de dispositivos de acceso directo (discos y diskettes) brinda la herramienta desde el punto de vista **físico** (hardware) para superar esas y otras limitaciones; los lenguajes avanzados orientados al usuario, como el IQL, brindan el instrumento **lógico** para sacarle el máxima provecho.

Audiencia

La Audiencia a la cual está dirigido este manual consiste en personal administrativo y técnico, profesionales y programadores que deseen interiorizarse de las posibilidades del **IQL**, la implementación de INTERSOFT del lenguaje **SQL**.

Aunque no es un prerrequisito excluyente, conviene que le lector tenga ciertos conocimientos básicos acerca del procesamiento de datos, y también del equipos del cual dispone en su instalación.

Capítulo 17

Conceptos Básicos

Las estructuras lógicas utilizadas para organizar las Bases de Datos se definen con unos pocos conceptos que se explican en este capítulo.

Se verán también los aspectos relativos a la operación y uso del intérprete SQL de **IDEAFIX**, cómo iniciar una sesión, cómo finalizarla, y las convenciones relativas al teclado de la terminal.

Que es una Base de Datos?

Una base de Datos es toda la información que un usuario tiene disponible en un momento determinado. A través de una serie de funciones, que se pueden realizar operaciones de búsqueda, modificación, agregado o eliminación de datos.

El enfoque Relacional

IDEAFIX permite manejar las bases de datos de acuerdo al modelo relacional. Para este modelo, la única estructura de datos existente es la tabla. A mero titulo ilustrativo, digamos que existen también otros modelos de Bases de Datos, tales como las jerárquicas, organizadas en torno a diversos tipos de archivos, segmentos de longitudes variables y ocurrencia múltiples, etc.

El RDBMS de IDEAFIX

El núcleo del manejo de Bases de Datos se conforma por el **RDBMS** (Relation Data Management System), o Sistema de Manejo de Bases de Datos Relacionales, que se encarga de acceder físicamente a la información.

El diseño y la manipulación de las estructuras de base de datos se realiza a través de las sentencias del lenguaje Estándar SQL (Structured Query Language), con algunas extensiones muy útiles, proporcionadas por **IDEAFIX**, que conforman la implementación practica de

dicho lenguaje, denominada **iql**.

Tablas

Las tablas son la forma básica de almacenamiento. Están formadas por **filas** (también llamadas *registro*) y **columnas** (asimismo llamadas *campos*), como se muestra en la Figura 17.1. A lo largo de este manual se emplearan en forma intercambiable los términos fila/registro, y campo/columna.

field1	field2	field3	field4

Figura 17.1 - Estructura de una Tabla

La figura sugiere que cada registro esta formado un numero fij de campos, y efectivamente es así: *todos* los registros en una tabla tienen el mismo numero de campos.

Nótese que las columnas:

- Se despliegan verticalmente en la pantalla.
- Contienen datos todos del mismo tipo.
- Tienen un nombre que les designa.

En cambio las filas:

- Se despliegan horizontalmente.
- Contienen distintas clases de datos sobre un cierto item. Entendemos por tal una persona, empresa, producto, transacción, etc.
- Carecen de nombre, pero tienen un numero relativo de registro.

IDEAFIX se suma al modelo relacional, donde existe solo una estructura de datos: la *tabla*. De esta uniformidad nace un lenguaje de Base de Datos donde, con un comando, se puede recuperar un conjunto de registros de una o mas tablas existentes, para listarlas, armar una nueva tabla, o bien reprocesar todos los registros recuperados. El lenguaje SQL es tan poderoso, que solo necesita que se le especifique **que** debe hacer, y no **como** hacerlo.

En **IDEAFIX**, el interprete de este lenguaje se denomina **iql**.

El utilitario **dgen** se utiliza para la definición de las estructuras de datos, ya que interpreta un subconjunto de la totalidad del lenguaje **SQL**, compuesto por las sentencias de definición de

datos (**DDS**, o *Data Definition Statements*) y otras auxiliares.

Esquemas

Un *esquema* es simplemente una colección de tablas, que normalmente guardan una relación lógica entre si.

Esta división en esquemas no implica restricción, dado que el RDBMS permite trabajar con varios esquemas simultáneamente, pudiendo relacionar datos contenidos en tablas de diferentes esquemas. Por lo tanto, es conveniente para lograr un buen diseño estudiar cuidadosamente como agrupar las diversas tablas de datos en los distintos esquemas. Las ventajas no son solamente de claridad en la definición lógico de los datos, sino también de orden practico (v.gr.: performance).

Bases de Datos

Se había mencionado que para un usuario, la Base de Datos es toda la información a la cual tiene acceso en un momento determinado. En **IDEAFIX**, una base de datos se define como el conjunto de datos contenido en todos los esquemas **activos**. Estos son aquellos que se han seleccionado con la operación apropiada, que puede asumir diferentes formas según el contexto en el cual se este trabajando.

Para que un esquema pueda ser activado, el usuario debe poseer los permisos adecuados para realizar tal operación. Aun contando con un esquema activo, un usuario tiene ciertas restricciones respecto a las operaciones que puede realizar con los datos contenidos en el.

Esquema Corriente

Entre todos los esquemas activos, existe uno que se denomina *corriente.* Es el esquema con el cual:

- Se trabaja en ausencia de una especificación explícita.
- Se realizan ciertos tipos de operaciones, por ejemplo, la de crear una nueva tabla.

Cualquier esquema activo puede seleccionarse en un determinado momento como corriente. Sin embargo, en un instante dado no puede existir mas que **un solo** esquema corriente. Esto implica que al pasar a un diferente esquema al cual se define como corriente, el que lo era con anterioridad permanece activo pero ya no es el corriente.

Tablas Ejemplo

A lo largo de este manual se dará una serie de ejemplos que explicaran todas las posibilidades que brinda el SQL de **IDEAFIX**. Estos ejemplos trabajaran con las cuatro tablas que se indican a continuación:

emp : Esta tabla contiene la información de los empleados de la empresa tomada como ejemplo.

depto : Aquí se almacenan los datos relativos a los departamentos que conforman las empresa.

cargos : En esta tabla se detallan los posibles cargos que pueden desempeñar los empleados de la compañía.

fam : Se registra la información referida a los familiares del personal, en particular de aquellas personas que se encuentran a su cargo.

Las siguientes figuras muestran la información contenida en dichas tablas:

EMP

nroleg	nombre	cargo	jefe	fnac	fingr	Sueldo	comis	depno
1	Juan	1		10/01/1953	01/01/1984	4500.00		1
2	Carlos Suárez	2	1	03/04/1953	01/01/1984	4500.00		1
3	Mar'a Laura	3	1	18/06/1960	01/03/1984	4250.00		1
4	Laiso	3	1	16/12/1954	01/03/1984	4520.00		1
5	Carlos Gabriel	4	1	25/041958	01/05/1984	4520.00		8
6	Berilini	4	6	20/07/1962	01/07/1984	4520.00		4
7	Miguel Angel	4	5	12/11/1956	01/12/1984	4000.00		6
8	Socos	4	7	14/12/1960	01/02/1985	4000.00		3
9	Alejandro	5	9	21/10/1961	01/05/1985	3250.00		4
10	Sergio Darta	5	9	22/12/1961	01/07/1985	3250.00		8
11	Marcela Edith	6	12	06/02/1959	01/09/1985	3250.00	2200.00	6
12	Zarce	6	11	14/11/1957	01/12/1985	2750.00		4
13	Raul	6	11	15/07/1960	01/12/1985	2750.00	900.00	4
14	Guillermo Caico	7	12	03/05/1958	01/01/1986	2750.00		4
15	Jorge Pablo	7	13	29/02/1952	01/01/1986	2750.00		6
16	Felag	7	13	31/03/1957	01/05/1986	2750.00	900.00	6
17	Ricardo	7	1	07/10/1962	01/09/1986	2750.00		4
18	Marcelo Acol	7	5	22/11/1962	01/10/1986	1500.00		4
19	Nilda Patricia	7	7	09/09/1959	01/12/1986	1500.00		4
20	Sovervi	8	9	27/02/1962	01/02/1987	850.00		1
21	Marcela	8	8	16/06/1964	01/04/1987	850.00		8
22		9	8	15/05/1963	01/04/1987	2750.00		6
23		9	4	08/10/1965	01/05/1987	2750.00		4
24		7	1	19/08/1964	01/05/1987	2750.00		3
25		7	1	24/05/1965	01/06/1987	2250.00		3
26		7	26	13/01/1964	01/06/1987	2250.00		5

	paula Dilopori							
	César pablo Taliga							
	Laura Aída Leriblascoc							
	Luís Marcos Lapadeo							
	Mar’a Antonieta de las flores							
	Estella Maris Newher							
	Ana María Notón							
	Julio César							
27	Elías	10		12/12/1964	01/09/1987			7
28	Eduardo	10		01/06/1965	01/02/1988	2000.00		7
29	Ricardo Estuardo	9		22/05/1952	23/08/1987			5
	Clara Nieves Farola							
	Florio Manuel Tenorio							
	Nicolás Atilio Regaluh							
	María Aída Estía							
	Adriana Mariana Crana							
	Roberto Diego Flañez							
	Alejandra Angeles Riveros							
	Marisa							

	Clarisa Mayo Fernando López Gabel Alfredo Mik Ladrón							

Figura 17.2 - Tabla de Personal EMP

DEPTO

depno	nombre	ubic
1	Dirección	Buenos Aires
2	Gerencia	Bs.As./Rosario
3	Software de Base	Buenos Aires
4	Desarrollo	Rosario
5	Documentación	Rosario
6	Mantenimiento	Rosario
7	Administración	Buenos Aires
8	Ventas	Buenos Aires

Figura 17.3 - Tabla de Departamentos DEPTO

CARGOS

cargo	descrip
1	Presidente
2	Vicepresidente
3	Director
4	Gerente
5	Subgerente
6	Líder de Proyecto
7	Analista Programador
8	Secretaria
9	Becario
10	Administrativo

Figura 17.4 - Tabla CARGOS

FAM

nroleg	nrofam	tipo	nombre
			Adriana Gómez
			Maria Luisa Cuiralda
4	1	1	Federico Darta
5	1	1	Antoniella Darta
5	2	2	Pablo Daniel Martinez
5	3	2	Cecilia Miranda
6	1	1	Santiago Caico
7	1	1	Silvia Hernández
7	2	2	Julio De Caro
8	1	1	Claudia Cristina Correa
10	1	1	Gustavo Daniel Taliga
12	1	1	Nicolás Newher
12	2	2	María fernanda Luissi
16	1	2	María Emilia Santillán
18	1	1	María Laura Estuardo
19	1	1	María de las Nieves Estuardo
19	2	2	María Vanessa Estuardo
19	3	2	Juan Manuel Estuardo
19	4	2	Natalia Agostina Estía
19	5	2	Virginia Warburg
23	1	2	Vanina Alejandra Flañez
25	1	1	Osvaldo Martínez
25	2	2	Yanina Vanina martínez
26	1	1	Esteban Gonzalo López Gabel
26	2	2	
28	1	2	

Figura 17.5 - Tabla FAM

El Comando use

Para poder manejar tablas por medio del **iql**, es necesario advertir previamente al sistema que el esquema que las contiene va a ser usado; o dicho en términos propios del lenguaje, es

preciso *activarlo.*

La sentencia **use** permite al usuario lograr esto:

```
use esquema1, esquema2,........, esquemaN;
```

indica al **iql** que los esquemas mencionados deben pasar al estado de *activos.* La lista de esquemas va separada por comas, y finalizada por " punto y coma" (;), lo cual ejemplifica dos reglas de sintaxis aplicables a todo comando: cuando es valido especificar varios items de la misma especie (tales como campos, tablas o esquemas) la lista que los incluye queda delimitada pir dos **palabras clave** (*keywords*), o bien por una *keyword* y el "fin de sentencia" (;) como en este caso; por otra parte, la lista de items se construye intercalando comas entre ellos. Los espacios en blanco no son tenidos en cuenta -salvo cuando al menos uno es necesario como separador- y quedan a gusto del usuario. En el ejemplo dado, se precisa uno de tales espacios entre el verbo **use** y el primer termino de la lista **esquema1.**

Volviendo al comando en si, el primero de los esquemas mencionados tiene una categoría especial para el **iql** y se lo denomina esquema *corriente*. Mas adelante se vera la importancia que esto tiene.

Como Visualizar Esquemas Activos

En el apartado anterior vimos como activar esquemas mediante la sentencia **use**. Dentro de las listas de esquemas activos mencionamos la existencia de uno de ellos denominado *corriente*. Para conocer esta lista de esquemas activos y cual de ellos es el corriente, existe la sentencia **show**. La forma de invocarla es sencilla:

```
show;
```

El resultado de este comando tendrá un aspecto similar al siguiente:

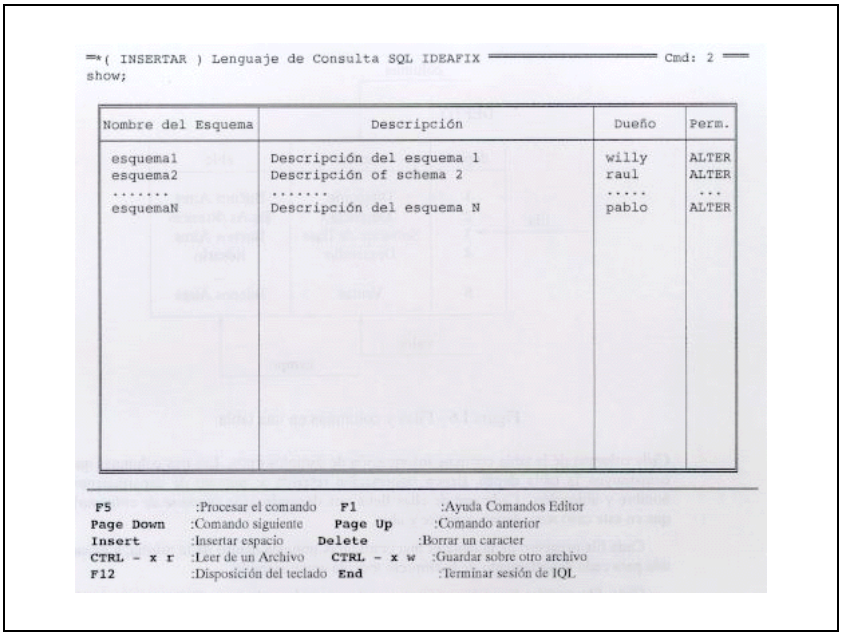


Figura 17.6

Las ultimas dos columnas se refieren al dueño del esquema y a los permisos que tiene el usuario que ejecuta esta sentencia sobre ese esquema. Mas adelante en este manual se explican en detalle los significados en ambas columnas.

Tablas, Filas y Columnas

Las columnas de la Base de Datos **IDEAFIX** están formadas por *filas y columnas*, como se indica el ejemplo de la Figura 17.7:

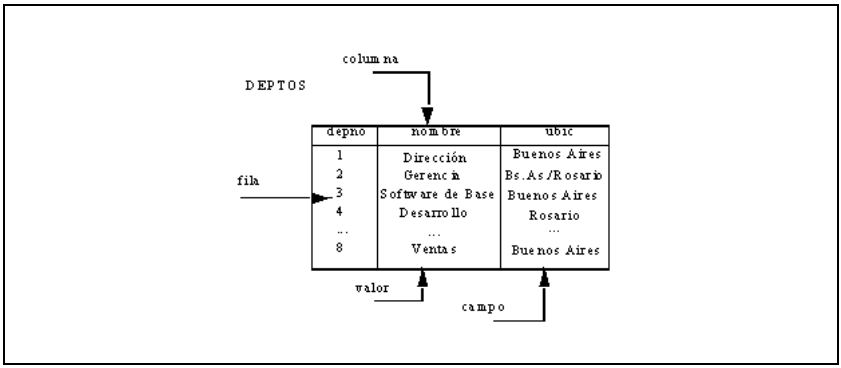


Figura 17.7 - Filas y columnas en una tabla

Cada columna de la tabla contiene información de distintos tipos. Las tres columnas que constituyen la tabla **depto.** tienen información referida a: numero de departamento, nombre y ubicación. Cada una de ellas lleva una denominación (*nombre de columna)*, que en este caso son: depto, nombre y ubic.

Cada fila o registro de la tabla es una ocurrencia unívoca dentro de la misma, y existe una para cada departamento de la empresa tomada como ejemplo.

Cada fila consta de varios *campos*, uno para cada *columna*. Cada campo puede contener un único *valor*. Por ejemplo, el campo *ubic* para la fila *Ventas* tiene el valor *Buenos Aires*.

Cada columna definida en la tabla puede contener un único tipo de datos, a veces llamado *formato*, siendo los más comunes:

char : En una columna de este tipo se puede registrar información de cualquier índole, tal como letras, números o símbolos especiales: +,-, (,*, %, etc.

num : Las columnas de tipo numérico contienen un valor aritmético incluyendo signo y punto decimal (nu una coma).

date : En este tipo de columna se almacena una fecha (en formato interno)

time : Permite registrar un valor de horas, minutos y segundos, asimismo en formato interno.

float : Este tipo de columna almacena valores numéricos en punto flotante de doble precisión (*mantisa y exponente*).

Relaciones entre Tablas de Datos

La información de una tabla puede estar relacionad con la de otra tabla de esquema. En el ejemplo que se muestra, la tabla **emp**, esta relacionada con la tabla **depto** mediante el campo *depno.* Este ultimo contiene el mismo tipo de datos en ambas tablas. Dicha información

permite tener entradas en ambas tablas que puedan ser relacionadas.

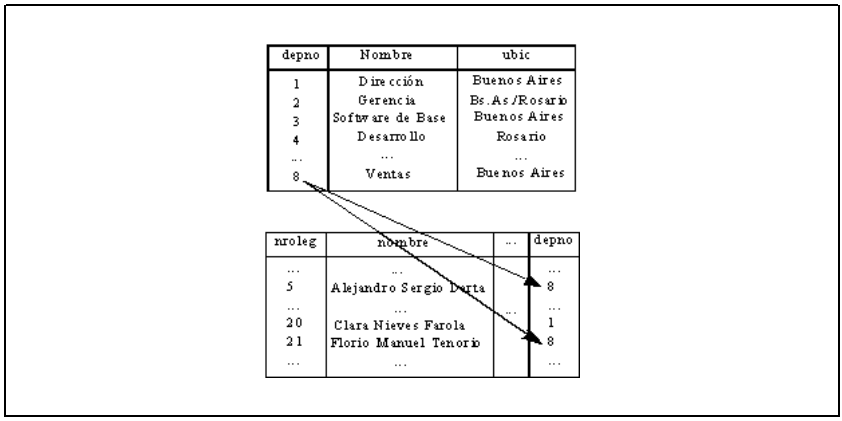


Figura 17.8 - Relación entre Tablas

La posibilidad de relacionar tablas entre si, permite organizar los datos en unidades separadas. Se puede entonces manejar la información de los departamentos aparte de la de personal, aunque siempre podemos unir los datos de ambas tablas, por ejemplo: conocer la ubicación (ubic, en **depto**) para un empleado (en **emp**). No es necesario que **todas** las tablas de una Base de Datos estén relacionadas.

A continuación se muestra el código que permite la creación de las tablas del ejemplo que se ve en las Figuras 17.2/3/4/5. Las sentencias utilizadas para su definición serán explicadas en capítulo posterior de este mismo manual.

```
create schema personal descr “Esquema del Personal de la
Empresa”;
table emp descr “Legajos del personal” {
nroleg num (4) descr “Numeros de Legajo”
primary key,
nombre char (30) descr “Nombre del Empleado”,
cargo num (2) descr “Cargo que ocupa” in cargos: descrip,
jefe num (4) descr “Jefe Directo” in emp: nombre,
fnacim date descr “Fecha de Nacimiento”,
fingr date descr “Fecha de Ingreso” <= today,
sueldo num (12,2) descr “Sueldo Mensual”
comis num (12,2) descr “Comision por Ventas”,
depno num (12,2) descr “Departamento al que pertenece” in
depto: nombre
}
index nombre (nombre not null),
index ingreso (fingr, nroleg);
table cargos descr “Codificadores de cargos”{
cargo num (2) descr “Cargo” primary key,
descrip char (20) descr “Descripcion del Cargo”
};
```

```
table depto descr “Departamentos de la empresa” {
depno num (2) descr “Numero de Dpto.” primary key,
nombre char (20) descr “Nombre del Depto.”
ubic char (20) descr “Ubic. geografica del depto.”
};
table fam descr “Familiares de los empleados”{
nroleg num (4) descr “Nro. de legajos del empleado” in emp:
(nombre),
nrofam num (2) descr “Nro. de familiar”,
tipo num (1) descr “Tipo de Parentesco” in (1: “Esposo/a”, 2:
“Hijo/a”),
nombre char (30) descr “Nombre del Familiar”
}
primary key (nroleg, nrofam);
```

Inicio de una Sesión con IQL

Dado que el SQL es un lenguaje fundamentalmente interactivo, es preciso disponer de una terminal, o sea un dispositivo de comunicación con la unidad central de proceso. Consta de dos unidades claramente diferenciadas: el teclado, por medio del cual se digitan las instrucciones o comandos que el usuario quiere ejecutar, y la pantalla, semejante a un televisor en el cual se representan los mensajes del sistema, información dirigida al usuario, etc.-

La terminal debe estar conectada a un equipo que trabaje bajo UNIX, y tenga instalado además el software de **IDEAFIX**. Esto implica que dispondrá también de la Base de Datos ejemplo que acompaña al modulo IQL de **IDEAFIX**, con la cual podrá practicar los ejercicios propuestos.

Las sentencias o comandos del Lenguaje SQL pueden dividirse en tres grupos:

- Las de manipulación de datos (DMS).
- Las de definición de datos (DDS).
- Un conjunto variado que llamaremos sentencias de control (CS).

Entre estas ultimas vamos a encontrar algunas que son extensiones al Estándar, propias de la implementación IQL de **IDEAFIX**.

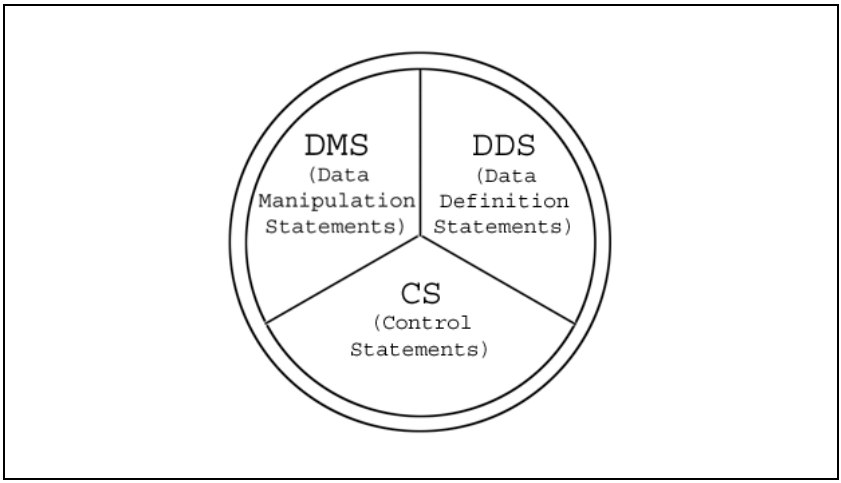


Figura 17.9 - La pizza de Sentencias IQL

Para iniciar una sesión interactivo con **iql**, desde el nivel del sistema operativo, que se identifica con la señal de espera (*prompt*, indicada por el símbolo $ en el ejemplo), se debe ejecutar el comando:

```
$iql
```

El sistema responderá con algo semejante a lo siguiente:

```
iql. 91/06/19. Cpoyright Intersoft Co. (c) 1988-89
```

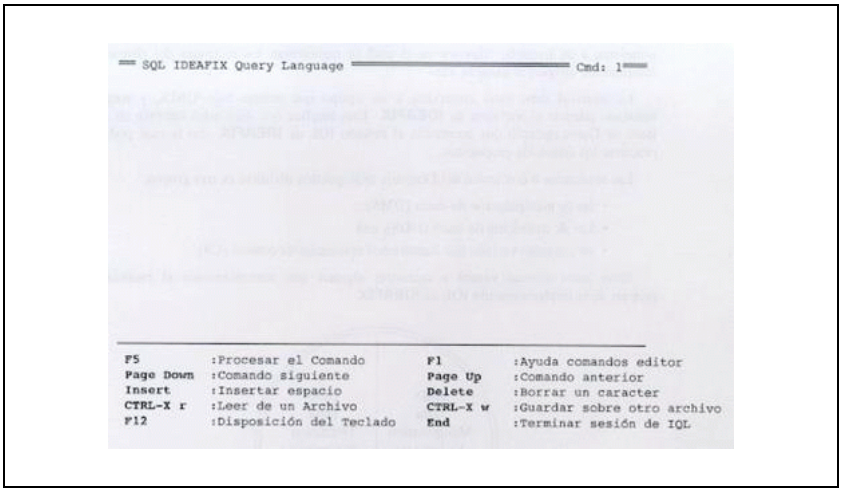


Figura 17.10

Teclas usadas

En la parte inferior de la pantalla ilustrada por la figura anterior, aparecen las principales teclas de función disponibles. No obstante, existen muchas posibilidades dado que el **iql** trabaja dentro del entorno del editor de **IDEAFIX**, de modo que se puede obtener una ventana auxiliar que detalle todas las funciones utilizable oprimiendo la tecla HELP, o bien recurriendo a “Ctrl-K” (oprimir simultáneamente las teclas “ctrl.” y “k”), que proporciona instrucciones en cualquier teclado. En el capitulo X se amplia la información sobre el editor.

Sin perjuicio de lo anteriormente expuesto, debe puntualizarse aquí que la variedad de modelos de terminal existentes en plaza, hace a veces dificultoso el ubicar físicamente una tecla determinada dentro del teclado. Si ello ocurriera, pida ayuda al personal especializado de su instalación, quienes lo instruirán al respecto. Existe lo que en UNIX se denomina una **variable de ambiente** *(environment variable,* llamada *TERM* (apócope de terminal), que describe las características de la unidad que usted usa; y que entre otras cosas establece la relación lógica entre cada posición de teclado, según el nombre que la identifica, y la función que el **iql** interpreta como requerimiento del usuario (v. gr.: desplazar el cursor, avanzar o retroceder una pantalla, desplegar un comando anterior, etc.)

El Editor IQL

El interprete de IQL permite editar y ejecutar comandos disponiendo de todas las características propias de las aplicaciones desarrolladas con **IDEAFIX** (*calculadora en lines,*

Print Screen, Multitareas, etc).

Las sentencias IQL se ejecutan mediante la tecla PROCESAR PAG ANT y PAG SIG se puede avanzar y retroceder en la secuencia de comandos para repetir consultas sin necesidad de digitarlas de nuevo. Con las teclas de cursor, ELIMINAR e INSERTAR se pueden editar los comandos.

Si se quiere guardar la información generada en forma de comandos para hacer mas tarde uso de ella, se digita ^X-W (esto es, primero las teclas Ctrl-X simultáneamente, y luego la W por write). con lo cual el sistema pedirá el nombre del archivo en el cual va a grabar ese conjunto de comandos.

Para insertar líneas grabadas previamente, dentro del texto que se esta elaborando, se ingresa ^X-R (ahora la “R” es por “read”), y ante el pedido del editor, se le da el nombre asignado al archivo en el cual se almaceno oportunamente la información, para traerla ahora en pantalla.

Mensajes de Ayuda y Error

Los mensajes emitidos por el sistema pueden dividirse en dos grandes grupos: errores de sintaxis y errores de ejecución. Los primeros están numerados del 1 al 500 (numero de mensaje) y además se identifica el numero de l’nea en el cual se produjo. Esto es útil cuando un comando se escribe utilizando varios renglones, y mas aun cuando se encadenan varios comandos en lo que se denomina *exec* (por “execution procedure”), como se vera en un capitulo posterior.

Los errores de sintaxis típico son tales como los siguientes:

- Falta de una coma o punto y coma, o está mal ubicado.
- Paréntesis sin balancear, o mal ubicados.
- Palabra clave (keyword) mal escrita, como por ejemplo WERE en lugar de WHERE.
- Palabra clave usada como nombre de campo (v. gr. no debe llamarse ORDER al numero de orden o pedido, ya que ORDER BY es el subcomando de clasificación).

Los errores de ejecución, en cambio, se identifican con los números del 501 en adelante, y se refieren a cuestiones tales como:

- Tabla o esquema inexistente (verificar que el nombre este correctamente escrito).
- Falta de permiso de acceso.
- No hay esquema corriente.

Terminando una Sesión con IQL

Para terminar una sesión con **iql** se debe presionar la tecla FIN con el editor esperando entrada, o bien digitar EXIT.

Se pedirá confirmación para terminar la sesión.

Capacidades del Sistema

La tabla inserta a continuación ilustra sobre los limites del sistema.

Esquema por usuario Sin Límite Esquema activos 8 Tablas por esquema 255 Tablas activas por esquema 255
Filas en una tabla 21474836447 Campos por tabla 255 Tamaño máximo de un registro en bytes 65535 Caracteres en un campo alfanumérico 65535
Dígitos en un campo numérico 15 Dígitos significativos en un campo numérico 15 Rango de valores en un campo fecha 16/04/1894 a 16/09/2073 Rango de valores en un campo hora 00:00:00 a 23:59:58
Máxima dimensión de un vector 65535 Valores en un campo in 65535 Indices por tabla 255 Campos por índice 255
Longitud de un campo alfanumérico en índice 255 Longitud máxima de nombre de Esquema 10 Longitud máxima de nombre de Tabla 10 Longitud máxima de nombre de Campo 65535

Capítulo 18

Consultando Tablas

En este capítulo se muestra como realizar una de las operaciones más comunes en SQL: obtener reportes de información de las tablas de base de datos. Esta operación es conocida como **select**.

Antes de comenzar a trabajar con casos concretos de selección, conviene establecer ciertas convenciones que son las que utiliza el SQL/IQL a través de su editor, tal como se halla implementado en **IDEAFIX**, que se especifican en el siguiente apartado.

Convenciones del SQL/IQL

Es indistinto el uso de mayúsculas o minúsculas para escribir los comandos, ya que se efectúa una conversión automática interna para uniformar el tipo de letra.

Cuando se escriben números se usa el punto como separador decimal. Sin embargo cuando se los muestre en pantalla o impresora se les dará formato de acuerdo al país (v.gr. en nuestro caso el separador será la coma y el punto oficiara de indicador de millares).

Pueden agregarse comentarios a las sentencias, de dos maneras:

- Precediéndolos con // a partir de cualquier columna. El resto de la línea se ignora a los fines del proceso.
- Encerrándolos entre “/*” para comenzar y “*/” para terminar, como se ilustra seguidamente: /* **Esto es un comentario** */, a semejanza del lenguaje de programación. “C”. De esta forma puede usarse el resto de la línea para continuar el comando.

Por ultimo, adoptaremos la convención usual de escribir en el manual las palabras clave -es decir, que deben escribirse tal como están- en letra **resaltada**, mientras que irán en letra normal los nombres simbólicos que deben sustituirse por los valores que efectivamente correspondan. Un caso típico de esto ultimo es **tabla**, que representa el nombre de alguna tabla de una base de datos.

El Comando select

La operación más común en el lenguaje de consulta es la de seleccionar uno o más campos de una o varias tablas de un esquema determinado. La sentencia **select** es la que permite realizar este tipo de operación. En su forma más simple su sintaxis es:

```
select campo1, campo2, ..., campoN
from tabla;
```

Las sentencias pueden escribirse en una o varias líneas, en formato libre. Esto es valido no solo para el **select**, sino para todos los comandos del lenguaje. El fin de una sentencia se indica mediante un "punto y coma".

El significado de este comando, será algo así como:

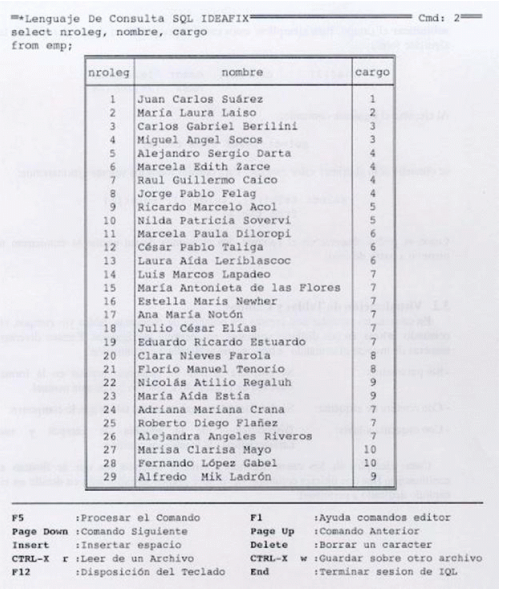
seleccionar campo1 campo2 ... campoN de "tabla" , con lo que se obtendrá como resultado una lista de todos los registros de la tabla en cuestión, pero viéndose solamente los campos indicados.

Seleccionando Columnas Específicas de una Tabla

Una de las maneras de referenciar a los campos de la tabla o las tablas que se quieren consultar es especificando cada uno de dichos campos. Por lo tanto, en una sentencia **select** se podrá especificar cuales son las columnas que se quieren obtener en la consulta.

```
select nroleg, nombre, cargo
from emp;
```

En este caso se especificó que se quieren obtener los campos **nroleg, nombre** y **cargo** de la tabla **emp** del esquema corriente. La salida de este comando es:



```
=*Lenguaje De Consulta SQL IDEAFIX========== Cmd: 2===
select nroleg, nombre, cargo
from emp;
```

nroleg	nombre	cargo
1	Juan Carlos Suárez	1
2	María Laura Laiso	2
3	Carlos Gabriel Berilini	3
4	Miguel Angel Socos	3
5	Alejandro Sergio Darta	4
6	Marcela Edith Zarce	4
7	Raul Guillermo Calco	4
8	Jorge Pablo Felag	4
9	Ricardo Marcelo Acol	5
10	Nilda Patricia Soverví	5
11	Marcela Paula Diloropi	6
12	César Pablo Taliga	6
13	Laura Aída Leriblascoc	6
14	Luis Marcos Lapadeo	7
15	María Antonieta de las Flores	7
16	Estella María Newher	7
17	Ana María Notón	7
18	Julio César Elías	7
19	Eduardo Ricardo Estuardo	7
20	Clara Nieves Farola	8
21	Florio Manuel Tenorio	8
22	Nicolás Atilio Regaluh	9
23	María Aída Estía	9
24	Adriana Mariana Crana	7
25	Roberto Diego Flañez	7
26	Alejandra Angeles Riveros	7
27	Marisa Clarisa Mayo	10
28	Fernando López Gabel	10
29	Alfredo Mik Ladrón	9

```
F5        :Procesar el Comando      F1        :Ayuda comandos editor
Page Down :Comando Siguiente        Page Up   :Comando Anterior
Insert    :Insertar espacio         Delete    :Borrar un caracter
CTRL-X  r :Leer de un Archivo       CTRL-X  w :Guardar sobre otro archivo
F12       :Disposición del Teclado  End       :Terminar sesion de IQL
```

Figura 18.1

Consultando Campos Vectorizados

Al seleccionar un campo vectorizado de la forma vista hasta ahora, solo se obtiene el valor de la primer ocurrencia. Si se quisiera recuperar el resto de los valores es necesario **subindicar** el campo. Para ejemplificar estos casos supongamos un campo definido de la siguiente forma:

```
telef [3] char (10) descr “Telefonos”
mask “ (3x) 2nN-4N”
```

Al ejecutar el siguiente comando:

```
select .. telef..from.. tabla;
```

se obtendrá solo el primer valor cargado. Para obtener todos los valores ejecutaremos:

```
select .. telef [0], telef [1], telef [2]
from tabla;
```

Como se podrá observar en el ejemplo, los elementos de un vector se comienzan a numerar a partir de cero.

Visualización de Tablas y Campos

En caso de no recordar con certeza los nombres de esquemas, tablas y/o campos, el comando **show**, en sus distintas variantes, nos permite averiguarlos. Existen diversas maneras de invocar el comando **show**, según se detalla a continuación:

- Sin parámetros: Se obtendrá la lista de esquemas activos en la forma indicada en el Capítulo 17 de esta Sección
- Con nombre de esquema: Se detallaran los nombres de las tablas que lo componen.
- Con esquema y tabla: Dispondremos de la nómina de campos y sus características, de la tabla en cuestión.

Como ejemplos de los casos segundo y tercero, tenemos los que se ilustran a continuación (las dos últimas columnas del primer gráfico se explicaran en detalle en el capítulo dedicado a permisos):

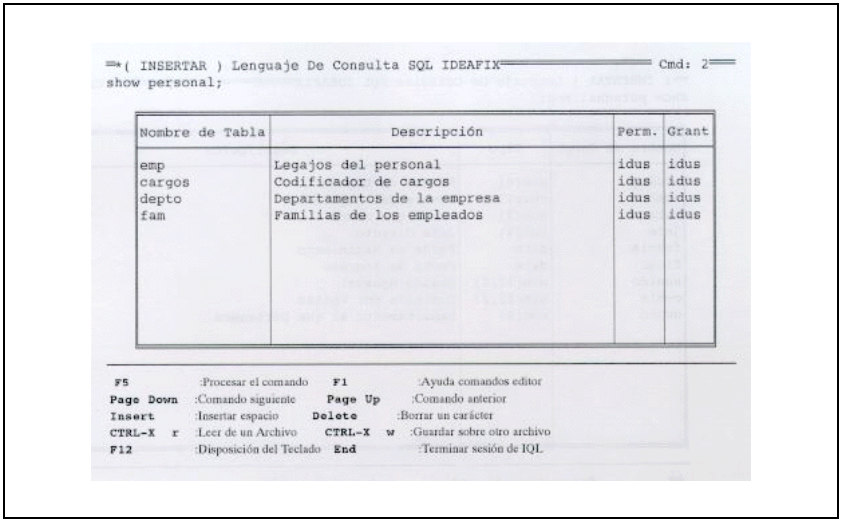


Figura 18.2

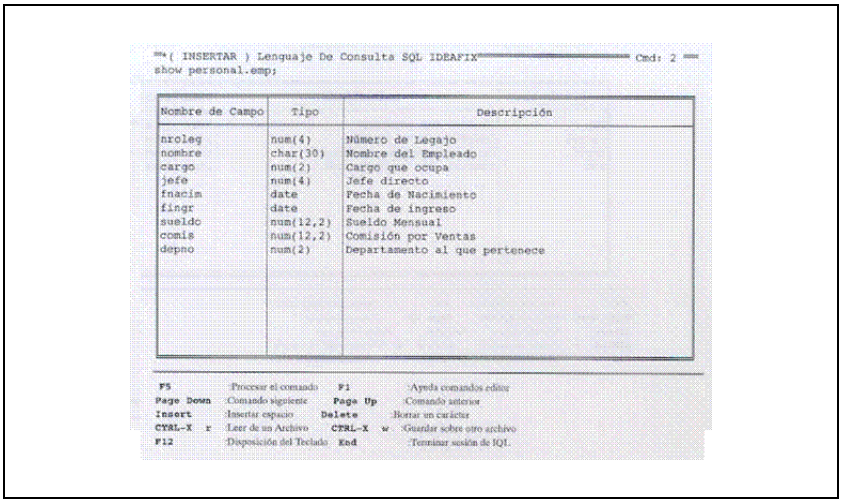


Figura 18.3

Referencia a Campos de Esquemas no Corrientes

El RDBMS **IDEAFIX** permite definir varios esquemas en una única base de datos. Esto tiene la ventaja de que se mantienen unidades lógicas más pequeñas, y permite integrar aplicaciones desarrolladas en forma independiente por distintas personas, o en distintos momentos.

Por ejemplo, seria el caso de un sistema integrado que abarque la interacción de varios subsistemas, como podría ser la gestión administrativa y contable de una empresa. En este tipo de sistemas, la Contabilidad actúa como núcleo en el cual los demás sistemas le brindan automáticamente información a través de pases contables en un determinado periodo. Para este tipo de aplicaciones, la estructura natural es la de tener un esquema por cada uno de los subsistemas presentes. Un subsistema que no sea Contabilidad, como por ejemplo Bienes de Uso o Compras, actúa de manera autónoma resolviendo las funciones administrativas propias. En ese momento, el esquema **corriente** seria el mencionado. A la hora de correr procesos que impliquen una integración contable, se necesita recurrir al esquema de Contabilidad. Aquí es donde se hace una referencia a dicho esquema, que no es el corriente, ya que se esta trabajando con Compras.

Cuando se estuviera trabajando con alguna aplicación como las mencionadas en el párrafo anterior, se definirán todos los esquemas con los que se esta trabajando, como **esquemas activos**. El primero de ellos es el **esquema corriente**. Las tablas y campos de dicho esquema serán referenciadas directamente, en tanto que los pertenecientes a los demás esquemas activos se referenciarán prefijándoles el nombre del esquema al que pertenecen.

En consecuencia, resulta innecesario -aunque valido- usar la expresión: esqcte. nomtab, supuesto que “nomtab” sea el nombre de una tabla que pertenece a “esqcte” que es a su vez el esquema corriente. Si es preciso, en cambio, escribir esquema. clientes, puesto que “esquema” no es el corriente.

El comando use se emplea para definir los esquemas activos:

```
use contab, compras, bduso;
```

donde el esquema de contabilidad contab es el corriente, utilizándose además los de compras y bienes de uso.

Si necesitamos referenciar una columna de la tabla **proved** dentro del esquema de compras por ejemplo *nroprov* (numero de proveedor), como dicho esquema no es el corriente debemos prefijar su nombre al de la tabla:

```
select nroprov from compras.proveed;
```

Seleccionando Todas las Columnas de una Tabla

La forma vista hasta ahora para referenciar a columnas de una tabla de un esquema es la de definir explícitamente las mismas. Si en una consulta se necesitan obtener los valores de todos los campos de una o más tablas, se deberían escribir los mismos, con el consiguiente esfuerzo que ello implica, sobre todo si la tabla contiene muchos campos. Existe una manera de hacer esta tarea menos difícil, y es a través del operador “*”. El mismo indicara. Por ejemplo:

```
select * from depto;
```

De esta forma se seccionan todas las columnas de la tabla **depto.**

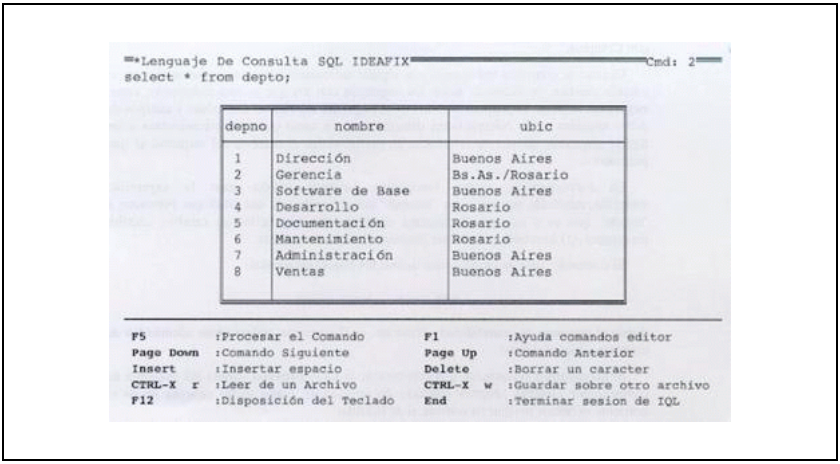


Figura 18.4

Controlando el Orden de las Columnas

Una consulta puede realizarse especificando los campos de una tabla en cualquier orden, no solamente por el orden en que estos están definidos. En la salida aparecerán en el orden determinado por la consulta.

```
select fingr, cargo, nroleg, nombre
from emp;
```

Desplegara los campos “Fecha de Ingreso” y “Cargo” antes (es decir, a la izquierda) de los campos de numero de empleado y nombre.

Y que pasa cuando queremos incluir información que no figura en la tabla **emp**, pero nos seria muy útil contar con ella? Existe dos formas posibles de superar este inconveniente, que veremos más adelante. Una consiste en el *join*, por el cual se obtienen datos de una o más tablas diferentes. La otra consiste en definir columnas nuevas que se calculan por medio de expresiones.

Encabezados de Columnas

Muchas veces, cuando se define una tabla, se especifican los campos de la misma usando nombres mnemotécnicos, los que para una visión final de la consulta dificultan la interpretación de la salida. Como medio de subsanar este inconveniente, se puede especificar un titulo para cada una de las columnas de una consulta. Si se lo incluye, deberá aparecer después del nombre de columna, delimitado entre comillas simples o dobles.

```
select
nroleg “Numero de\nempleado”,
nombre “Apellido\ny Nombre”,
cargo “Cargo que\nocupa”
from emp;
```

En este caso se agregaron títulos a todas las columnas. Se observa que aparece la secuencia de caracteres “\n”. Esta se utiliza para que el titulo abarque dos líneas diferentes, ya que lo que el iql interpreta es que debe insertar un símbolo “new-línea” (fin de línea).

Los títulos de las columnas aparecen siempre automáticamente centrados con respeto al área abarcada por el contenido del campo. El ejemplo propuesto deberá mostrar los títulos de esta forma:

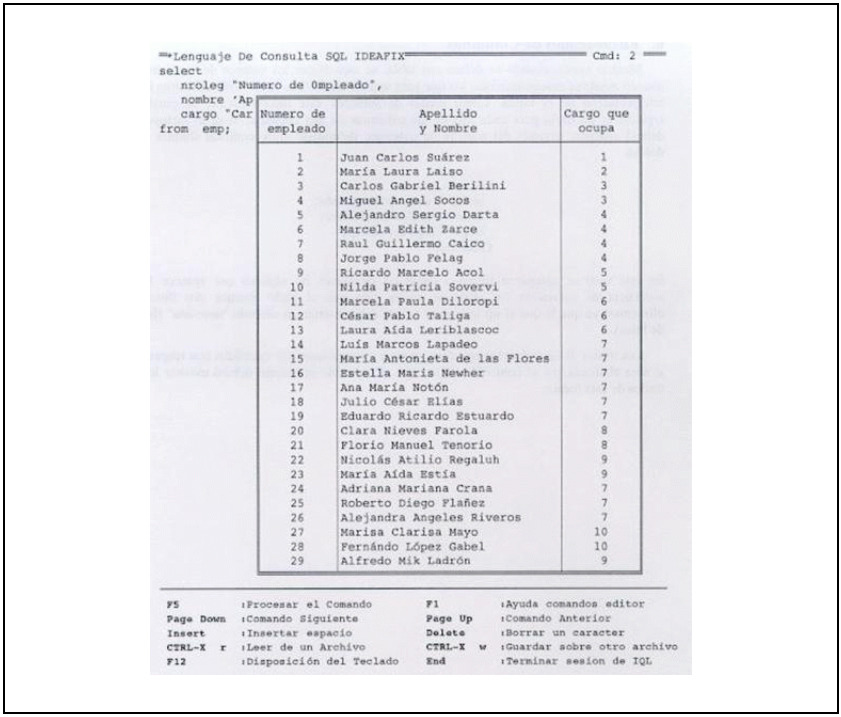


Figura 18.5

según el esquema y datos propuestos en el Capítulo 17 de esta Sección.

Formato Estándar de Salida

Según la manera normal de trabajar con **iql**, la salida se realiza a travésde una grilla en la que

aparecen todos los campos que son objeto de la consulta. La grilla es parte de una matriz que consiste de tantas filas como registros a desplegar, con más de una adicional de encabezamiento al comienzo para contener los títulos, y tantas columnas como resulte necesario conforme al requerimiento. Esto equivale, en definitiva, a una columna por cada campo seleccionado.

Es útil comprender el concepto de *grilla*, que es una suerte de "trozo" de información que se despliega de una vez. La salida total de un comando puede comprender muchas filas y columnas, en cantidad que habitualmente excede la capacidad de la pantalla para desplegar la matriz resultante. Vamos a suponer una salida que conste de 8 columnas -que ocuparan distintas longitudes según la naturaleza y el formato de cada una- y una cantidad arbitraria **N** de filas.

Se da entonces la situación descripta en la figura siguiente, donde la grilla viene a ser una especie de **ventana** que brinda acceso a una cierta cantidad de filas y columnas, y puede desplazarse en distintos sentidos. La suponemos posicionada luego de ciertos desplazamientos.

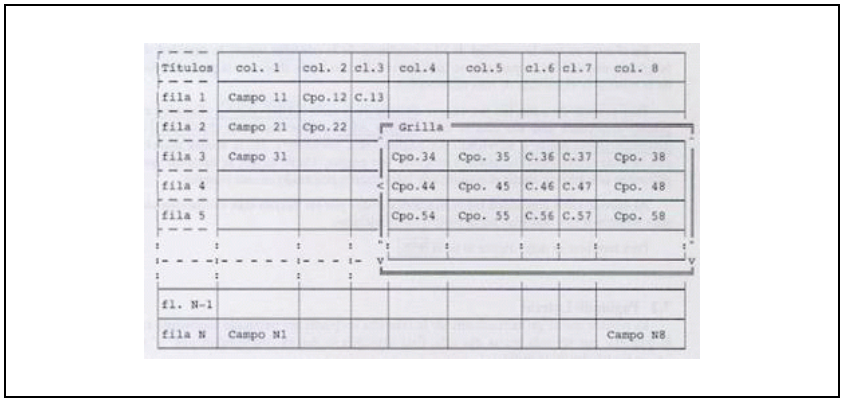


Figura 18.6

La grilla abarca cinco columnas, desde la cuarta hasta la octava inclusive (que es la ultima), y una cantidad de filas arbitraria, pero en todo caso menor que "N". Vemos que las flechitas indican la posibilidad de movimiento por ejemplo hacia la izquierda, pero no a la derecha (no hay más columnas para mostrar en esa dirección); en cambio puede desplazarse tanto hacia arriba como hacia abajo.

Los campos se identifican por su numero de fila y de columna: Campo 38 es el correspondiente a la tercera fila, octava columna, de la matriz.

iql busca las filas de la o las tablas que satisfacen la expresión de consulta y llena la matriz. Cuando la misma haya sido completada, esta en condiciones de cargarlas sobre la grilla; es decir se pueden ejecutar comandos para visualizar la información. Cuando se piden las líneas

o paginas siguientes, se buscaran estas y se agregaran a la grilla.

Con la grilla se pueden efectuar ciertas operaciones para desplegar la información de manera más cómoda. Las operaciones que se permiten son:

- Avance/retroceso
- Paginado lateral
- Bloqueo de columnas
- Fin de la consulta

Avance y Retroceso

En el caso en que la cantidad de filas resultante de la consulta supere la cantidad de renglones disponibles en pantalla, se indicara con flechas hacia abajo -en la parte inferior de la misma- la existencia de más información.

Para avanzar de a una fila por vez, se dispone de la tecla CURS ABA. También es posible retroceder una fila con la tecla CURS ARR. Con las teclas PAG SIG o INGRESAR se puede ir moviendo por página hacia adelante (*). De la misma manera con la tecla PAG ANT se puede ir moviendo por página. Desde luego, se entiende que *página* es la forma de aludir al conjunto de información contenido en una pantalla.

Al utilizar estos comandos las indicaciones dadas por las flechas irán evolucionando para mostrar si hay más filas hacia atrás o hacia adelante.

Para terminar se debe digitar la tecla FIN.

Paginado Lateral

Es posible que la grilla resultante de la consulta no pueda ser mostrada totalmente en la pantalla, por ser más ancha que ella. Esta situación se denota con indicaciones “>” o “<” en los bordes de la grilla.

En el caso en que se quisieran observar campos que se encuentren en los extremos de la grilla y que no entraran simultáneamente en la pantalla por la razón mencionada. Se puede paginar hacia la derecha con la tecla CURS DER. Realizando esta operación, se paginara por columna, corriéndose la grilla una columna hacia la derecha. Esta operación se podrá repetir hasta llegar a la ultima columna, que quedara alineada a la derecha.

Se puede También paginar por columna hacia la izquierda, con la tecla CURS IZQ.

También es posible desplazar la grilla hacia la izquierda o la derecha tantas columnas como entren en la pagina con las teclas PAG IZQ y PAG DER.

Al realizar las operaciones de paginación lateral las indicaciones van evolucionando para mostrar si hay más columnas hacia la derecha o la izquierda.

Bloqueo de Columnas

Si se pagina hacia la derecha, según lo explicado en el apartado anterior, se van perdiendo de la grilla las primeras columnas de la consulta. Muchas veces las primeras columnas son las que permiten identificar a cada fila de dicha consulta. Por ejemplo,

```
select * from emp;
```

obtendrá una matriz con todos los datos de la tabla **emp**, en la cual las primeras columnas son **nroleg** (el numero de empleado) y **nombre** (el nombre del mismo). Por lo tanto, para tener una mejor referencia de los datos que se obtienen en la grilla, se pueden suprimir columnas de la misma de modo que al desplazarse con las teclas de cursor o paginación hacia la derecha, una determinada cantidad de ellas permanezcan inmóviles Para realizar la operación de bloqueo, se oprime un numero del 1 al 9, que determina la cantidad de columnas a bloquear. Para volver al modo normal basta con oprimir el 0.

Fin de la Sesión de IQL

Si se desea finalizar la ejecución de **iql**, con la tecla FIN se podrá dar por terminada la misma. Antes de concluir, el programa preguntara al usuario si realmente es esa su intención, para prevenir una terminación indeseada debido a oprimir inadvertidamente la tecla FIN.

También se puede terminar la sesión ejecutando la sentencia é**xit.** En tal caso no se pide confirmación -no se concibe como accidental que el usuario haya digitado una secuencia de cuatro letras cuyo significado es muy especifico- y la sesión finaliza directamente.

Expresiones

Para el SQL, al igual que para la mayoría de los lenguajes de computación, una expresión es una formulación simbólica que responde a las reglas del álgebra ordinaria, y debe resolverse en un valor numérico. Los símbolos usados para las cuatro operaciones fundamentales son: +,-, * y /; y los paréntesis permiten agrupar operaciones de la manera deseada.

Es posible incorporar columnas a una tabla a través de una expresión, lo cual se comprenderá mejor con un ejemplo.

```
select nroleg, nombre,
comis * 100 / (sueldo + comis)
from emp;
```

nos listará tres columnas de la tabla de empleados:

- La que contiene su **numero de legajo**.
- La del **nombre y apellido** del empleado.

- El **porcentaje** que las comisiones representan sobre su **total de ingresos**.

La tercera columna es una expresión, que debe ser evaluada en cada caso a partir del salario y la comisión de un empleado; valores estos obrantes en la tabla. Si no se indica explícitamente otra cosa, tales columnas aparecen con el titulo (expr). Conviene por ello asignarles un titulo, que en el ejemplo propuesto podría ser “Porcentaje\n de comision”.(rfr. apartado "Encabezados de Columnas" de este capítulo).

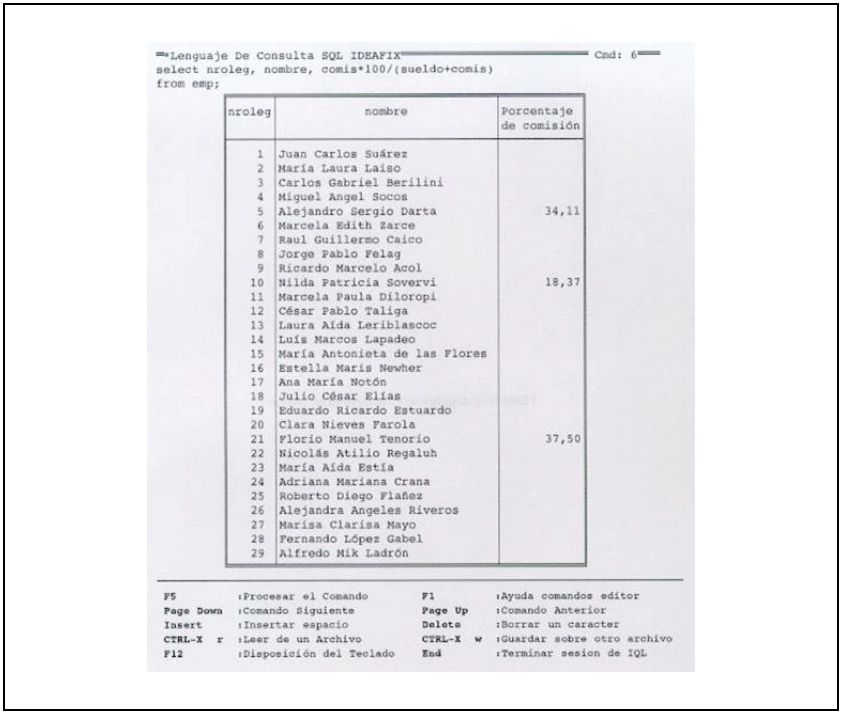


Figura 18.7

Capítulo 19

Seleccionando Filas de una Tabla

En el capítulo anterior se han explicado los fundamentos de la sentencia **select**. En este capítulo se exponen capacidades mas avanzadas de la misma, tales como la cláusula **where** para restringir el rango de la consulta, **order by** para especificar en que orden se debe listar el resultado de la misma, y los operadores lógicos que permiten formar expresiones.

Se mostrara también la forma de obtener información de varias tablas en una sola consulta, operación conocida como **join** en el lenguaje relacional.

La Cláusula where

Hasta ahora solo se ha visto una forma de consulta: la búsqueda de registros en la cual no se imponía condición alguna. Por lo tanto, se obtenían todas las filas de la tabla referenciada en la consulta.

Se pueden establecer condiciones de búsqueda, de modo de limitar la misma a registros de las tablas que cumplan con determinadas características. Esto se realiza mediante la cláusula **where**. La sintaxis de la misma es la siguiente:

```
select campo1, campo2, ..., campoN
from tablas
where condicion _de_busqueda;
```

Se puede referenciar cualquiera de las tablas de los esquemas activos. Las tablas del **esquema corriente** se referencian solamente por su nombre. Cualquier otra tabla debe invocarse explícitamente con el nombre del esquema y el nombre de la tabla.

Especificando Condiciones en la Cláusula where

Los operadores de comparación que se utilizan en la cláusula **where** son:

Operador	Significado
=	Igual a
= =	Igual a
!=	Distinto de
<>	Distinto de
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
>< **... and ...**	Operador **between**
between ... and	Operador **between**
like	Patrones en cadenas
in	Lista de valores

Los operadores lógicos que complementan las comparaciones son:

Operador	Significado
not, !	Negación unaria
and, &	Operador de conjunción (Y)
or, |	Operador de disyunción (O)

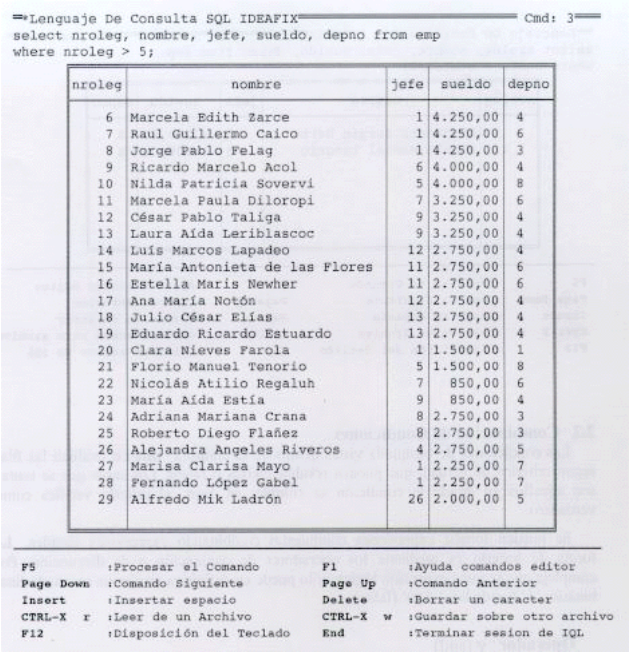
Especificando una Condición

Para establecer una condición, simplemente se la debe indicar después de la cláusula **where**, como se definió anteriormente.

La forma mas natural de imponer una condición es aquella en la cual el operador de comparación esta entre un nombre de columna y un valor constante. Por ejemplo:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg > 5;
```

Selecciona todos los campos de la tabla **emp** donde el numero de empleado sea mayor que 5. Como se ve en el siguiente gráfico:



```
=>Lenguaje De Consulta SQL IDEAFIX                                Cmd: 3
select nroleg, nombre, jefe, sueldo, depno from emp
where nroleg > 5;
```

nroleg	nombre	jefe	sueldo	depno
6	Marcela Edith Zarce	1	4.250,00	4
7	Raul Guillermo Caico	1	4.250,00	6
8	Jorge Pablo Felag	1	4.250,00	3
9	Ricardo Marcelo Acol	6	4.000,00	4
10	Nilda Patricia Soverví	5	4.000,00	8
11	Marcela Paula Diloropi	7	3.250,00	6
12	César Pablo Taliga	9	3.250,00	4
13	Laura Aida Leriblascoc	9	3.250,00	4
14	Luis Marcos Lapadeo	12	2.750,00	4
15	María Antonieta de las Flores	11	2.750,00	6
16	Estella Maris Newher	11	2.750,00	6
17	Ana María Notón	12	2.750,00	4
18	Julio César Elías	13	2.750,00	4
19	Eduardo Ricardo Estuardo	13	2.750,00	4
20	Clara Nieves Farola	1	1.500,00	1
21	Florio Manuel Tenorio	5	1.500,00	8
22	Nicolás Atilio Regaluh	7	850,00	6
23	María Aida Estía	9	850,00	4
24	Adriana Mariana Crana	8	2.750,00	3
25	Roberto Diego Flañez	8	2.750,00	3
26	Alejandra Angeles Riveros	4	2.750,00	5
27	Marisa Clarisa Mayo	1	2.250,00	7
28	Fernando López Gabel	1	2.250,00	7
29	Alfredo Mik Ladrón	26	2.000,00	5

```
F5          :Procesar el Comando          F1          :Ayuda comandos editor
Page Down   :Comando Siguiente            Page Up     :Comando Anterior
Insert      :Insertar espacio             Delete      :Borrar un caracter
CTRL-X  r   :Leer de un Archivo           CTRL-X  w   :Guardar sobre otro archivo
F12         :Disposición del Teclado      End         :Terminar sesion de IQL
```

Figura 19.1

Otra manera de establecer la comparación es reemplazando la constante por otra columna, o por una expresión cualquiera. En este último caso:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where comis > sueldo/2;
```

Selecciona todos los empleados cuyas comisiones exceden a la mitad de su sueldo.

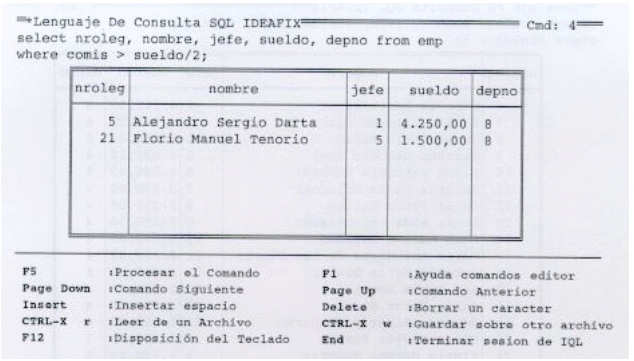


Figura 19.2

Combinación de condiciones

Las condiciones de búsqueda vistas dentro de la cláusula **where** evalúan las filas según criterios de verdad, que pueden resultar ciertos o falsos. Los casos que se toman son aquellos en donde las condición se cumple; es decir, el criterio verifica como verdadero.

Se pueden formar expresiones compuestas combinando expresiones simples. La forma de hacerlo es mediante los operadores de conjunción o de disyunción. Por compleja que sea una expresión lógica, solo puede, en definitiva arrojar un resultado final binario: “1” (verdadero) o “0” (falso).

Operador y (and)

Si se requiere que se cumpla mas de una condición es una consulta, la forma de hacerlo es a través del operador **and**. En este caso se evalúan cada una de las condiciones debiendo ser verdaderas todas ellas para que la expresión sea verdadera.

Por ejemplo:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg <= 15 and jefe = 1;
```

Seleccionara todos los empleados cuyo numero sea menor o igual que quince y tengan como jefe al empleado numero uno. Esto se denomina *conjunción* porque las condiciones deben darse conjuntamente. También se la llama *producto lógico*, ya que todos los factores o términos deben evaluar a 1 para que su producto sea 1: basta la presencia de **un solo factor 0** para que el producto se anule (=falso).

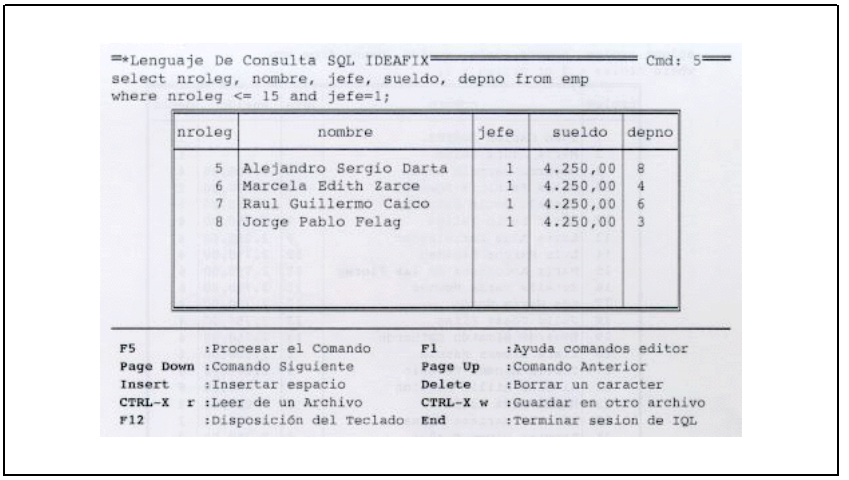


Figura 19.3

Operador o (or)

Si se consideran varias condiciones de búsqueda pero basta que sea verdadera solamente alguna de ellas, se utiliza el operador **or**.

```
selecet nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg < 3 or nroleg > 8;
```

En este caso se obtendrán todos los empleados cuyo número de empleado sea menor que tres o mayor que ocho.

Tenemos ahora una disyunción, dado que requerimos que se cumpla una u otra de las posibilidades. Esto se denomina suma lógica, ya que ahora la situación es inversa: sólo puede existir un resultado falso si todos los términos son cero. la presencia de más de un elemento igual a uno no produce problemas, ya que las normas del álgebra de **Boole** hacen que 1+1=1, pues no tienen sentido los valores mayores que la unidad. El sentido es “Verdadero más verdadero sigue siendo verdadero” (no doblemente verdadero).

Figura 19.4

Operador not (negación)

Cuando se requiere listar a aquellas filas que **no** cumplan una cierta condición, se utiliza el operador **not**. Así por ejemplo:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where not nroleg < 5;
```

Esta consulta es equivalente a pedir aquellos cuyo número de legajo es mayor o igual que 5. Desde el punto de vista lógico el not equivale a una inversión de valores a los fines de evaluar el resultado final: el uno se convierte en cero y viceversa.

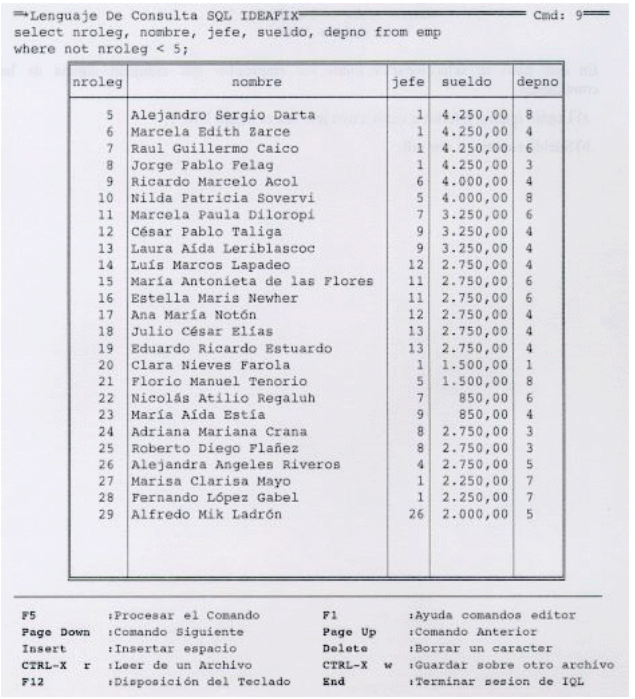


Figura 19.5

Combinación de Operadores

Los operadores de conjunción y disyunción vistos pueden combinarse en una única expresión. Cuando ello ocurre, la expresión se evalúa realizando primero las operaciones de conjunción y luego las de disyunción.

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg <= 8 and jefe = 1
or sueldo > 2000;
```

En este caso se seleccionarán todos los empleados que cumplan alguna de las condiciones:

a) Legajo igual o inferior a ocho, cuyo jefe sea el legajo Nro.1;

b) Sueldo superior a dos mil.

Figura 19.6

Si se necesita alterar el orden de precedencia, se puede encerrar la expresión entre paréntesis. Veamos qué ocurre cuando especificamos.

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg <= 5 and (jefe = 1 or sueldo > 2000);
```

Para interpretar esta expresión, consideremos en primer término a la parte entre paréntesis como una condición que analizaremos luego, y resultará claro entonces que se trata de un and que exige por una parte legajos no superiores al cinco, y otro requisito adicional que consiste en que el jefe sea el legajo 1 o bien el sueldo superior a dos mil.

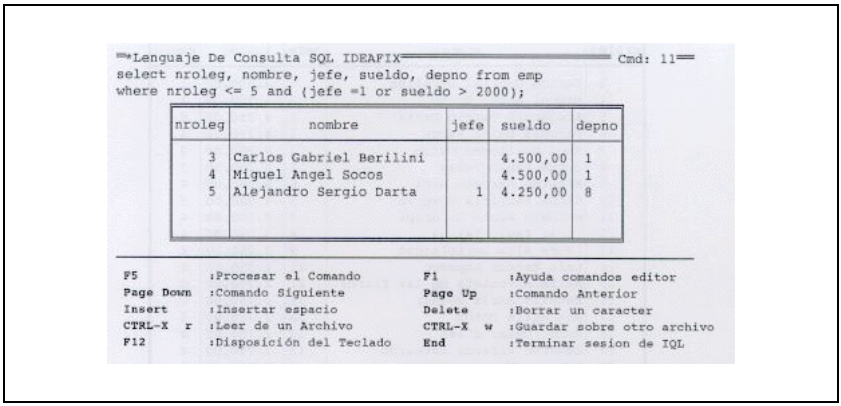


Figura 19.7

Seleccionando Filas dentro de un Rango

Se vio que una manera de especificar un rango en una cláusula where es la de definir la condición mediante el operador de conjunción.

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg >= 3 and nroleg <= 8;
```

En este caso se seleccionarán todos los empleados cuyo número de empleado esté comprendido entre tres y ocho.

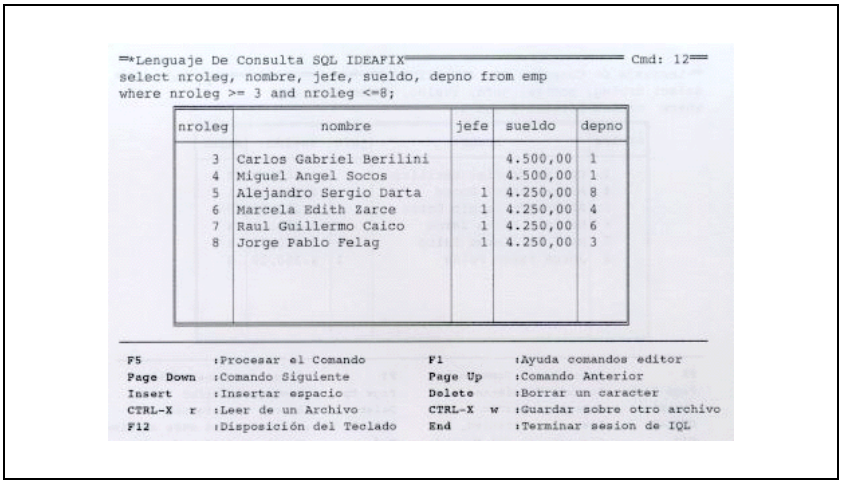


Figura 19.8

Otra manera de estipular la misma condición es a través de la cláusula **between**. La misma se especifica:

```
between límite_inferior and límite_superior
```

Por lo tanto, el ejemplo precedente puede escribirse como:

```
select nroleg, nombre, jefe, sueldo, depno
from emp
where nroleg between 3 and 8;
```

Semánticamente es equivalente al ejemplo anterior.

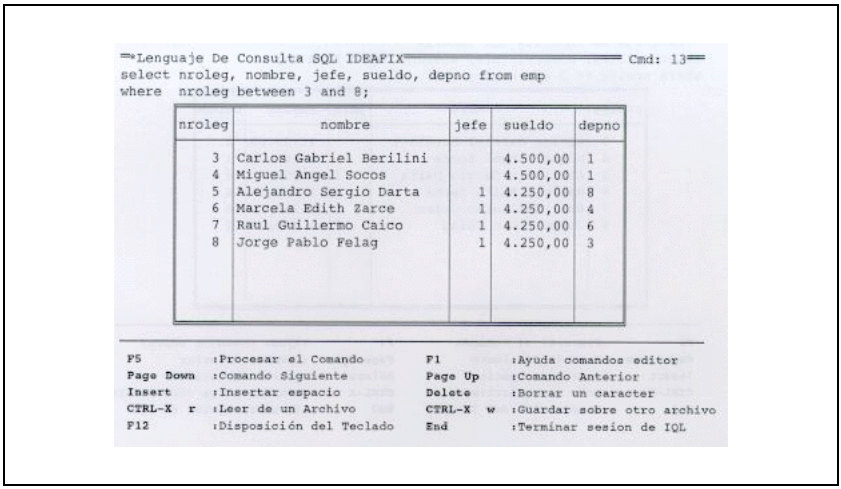


Figura 19.9

Nótese que los límites se incluyen en la definición del criterio de verdad; es decir, los operadores implícitos son<= y >=, no meramente < y >.

El operador in

Cuando la condición que se desea imponer a una consulta requiere que un campo asuma alguno de entre una serie de **valores prefijados**, se utiliza el operador **in**. Este permite indicar una lista de valores, y se verificará si su operando es alguno de ellos. Los valores deben ir encerrados entre paréntesis y separados por comas.

Veamos por ejemplo cómo listar los empleados de los departamentos Dirección, Gerencia y Documentación:

```
select nroleg, nombre, depno
from emp
where depno in (1,2,5);
```

Lógicamente esta consulta podría haberse planteado usando el operador de igualdad en combinación con el **or**, pero de esta forma es mucho más sencillo de expresar.

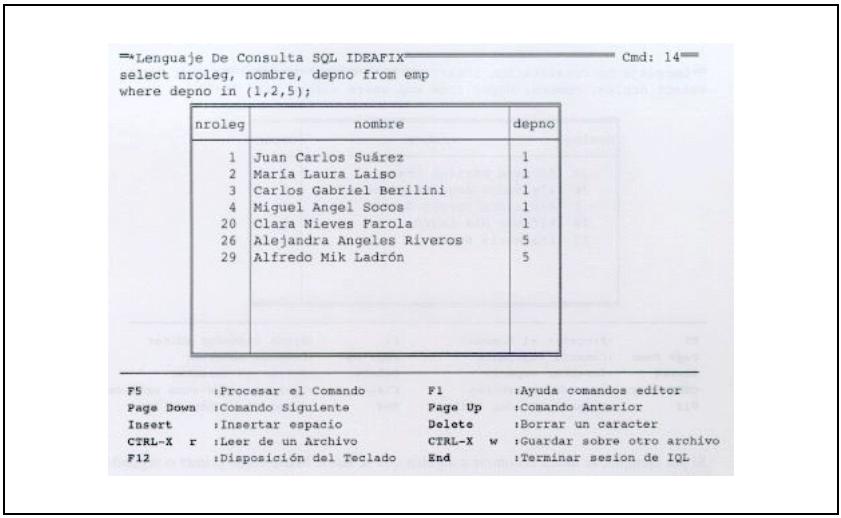


Figura 19.10

El Operador like

Este operador se aplica sobre cadenas de caracteres, y permite encontrar aquéllas que verifiquen un cierto patrón. Los patrones se forman con dos de los caracteres que tienen significado especial en UNIX (llamados metacaracteres), usados en el **like**: el "*", que significa cualquier sucesión de caracteres (incluyendo una cadena vacía), y el "?", que significa un carácter cualquiera, pero uno y sólo uno.

Si por ejemplo deseamos encontrar todos aquellos empleados cuyos nombres comienzan con A, ejecutamos:

```
select nroleg, nombre, depno
from emp where nombre like "A*";
```

La expresión "A*" se verificará con una cadena que tenga una A en la primera posición, y luego puede seguir cualquier cadena de caracteres. Como muestra la salida ejemplo:

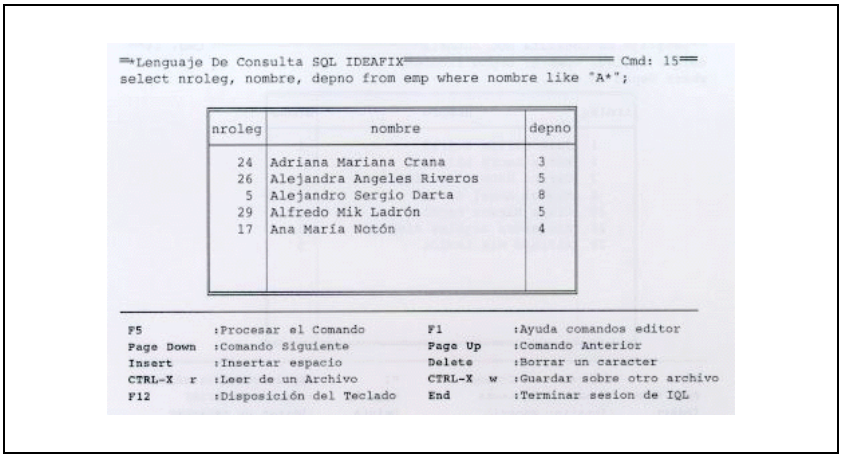


Figura 19.11

Si por ejemplo, se desea encontrar a alguien que se llame María como primer o segundo nombre:

```
select nroleg, nombre, depno
from emp where nombre like "*María*";
```

ya que el nombre puede estar precedido de otros caracteres.

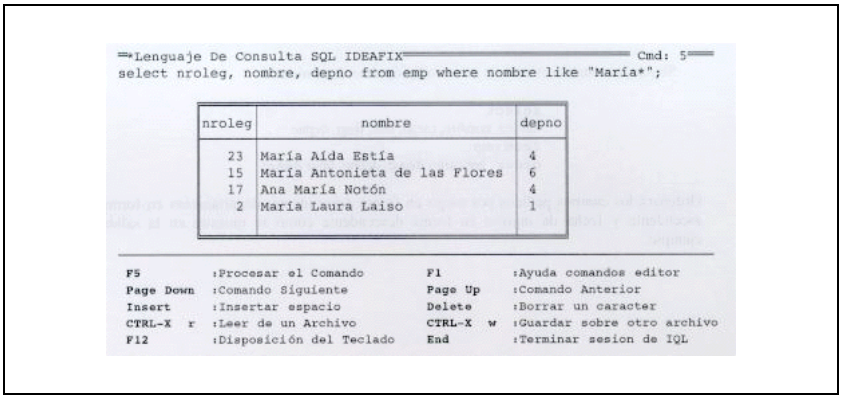


Figura 19.12

Despliegue de Filas Ordenadas. Cláusula order by

Cuando se realiza una consulta, no se garantiza en qué orden aparecerán desplegadas las filas de la tabla resultante. Para establecer un ordenamiento de acuerdo a cualquiera de las columnas de la consulta, se debe especificar la cláusula **order by**.

La sintaxis de la misma es como sigue:

```
order by ordenamiento
```

donde ordenamiento puede definirse como:

```
expresión [asc | desc ],...
expresión[ ascending | descending ],...
```

Por ejemplo, si se quiere ordenar una tabla resultante de una consulta en forma ascendente por cargo, se debe hacer:

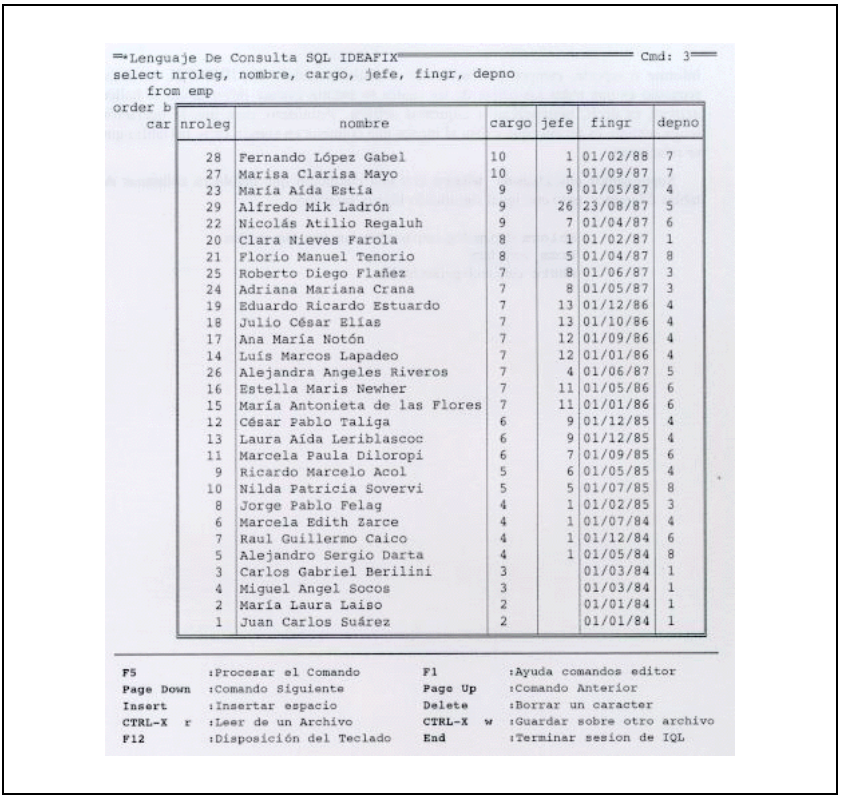
```
selecet nroleg, cargo, jefe, fingr, depno
from emp
order by cargo asc;
```

Como se asume ordenamiento ascendente por defecto, se podría haber obviado la cláusula asc.

Se pueden establecer más de una condición de ordenamiento:

```
select nroleg, nombre, cargo, jefe, fingr, depno
from emp
order by cargo desc, depno, fingr desc;
```

Ordenará los campos pedidos por cargo en forma descendente, departamento en forma ascendente y fecha de ingreso en forma descendente como se muestra en la salida ejemplo.



nroleg	nombre	cargo	jefe	fingr	depno
28	Fernando López Gabel	10	1	01/02/88	7
27	Marisa Clarisa Mayo	10	1	01/09/87	7
23	María Aída Estía	9	9	01/05/87	4
29	Alfredo Mik Ladrón	9	26	23/08/87	5
22	Nicolás Atilio Regaluh	9	7	01/04/87	6
20	Clara Nieves Farola	8	1	01/02/87	1
21	Florio Manuel Tenorio	8	5	01/04/87	8
25	Roberto Diego Flañez	7	8	01/06/87	3
24	Adriana Mariana Crana	7	8	01/05/87	3
19	Eduardo Ricardo Estuardo	7	13	01/12/86	4
18	Julio César Elías	7	13	01/10/86	4
17	Ana María Notón	7	12	01/09/86	4
14	Luís Marcos Lapadeo	7	12	01/01/86	4
26	Alejandra Angeles Riveros	7	4	01/06/87	5
16	Estella Maris Newher	7	11	01/05/86	6
15	María Antonieta de las Flores	7	11	01/01/86	6
12	César Pablo Taliga	6	9	01/12/85	4
13	Laura Aída Leriblascoc	6	9	01/12/85	4
11	Marcela Paula Diloropi	6	7	01/09/85	6
9	Ricardo Marcelo Acol	5	6	01/05/85	4
10	Nilda Patricia Sovervi	5	5	01/07/85	8
8	Jorge Pablo Felag	4	1	01/02/85	3
6	Marcela Edith Zarce	4	1	01/07/84	4
7	Raul Guillermo Caico	4	1	01/12/84	6
5	Alejandro Sergio Darta	4	1	01/05/84	8
3	Carlos Gabriel Berilini	3		01/03/84	1
4	Miguel Angel Socos	3		01/03/84	1
2	María Laura Laiso	2		01/01/84	1
1	Juan Carlos Suárez	2		01/01/84	1

Figura 19.13

JOIN de Tablas

Una característica distintiva de las tablas relacionales es que son suceptibles de unirse o juntarse (tal el significado de la expresión inglesa “join”) a fin de brindar un conjunto de información más completo. Esto implica la posibilidad de incluir en un informe o reporte, campos pertenecientes a distintas tablas. En IDEAFIX, el único requisito es que todas las tablas de las cuales se intenta extraer información se hallen activas; es decir, pertenezcan a esquemas activos. Asimismo, para que la operación tenga sentido, es preciso que exista al menos una columna en común entre las tablas que se relacionan.

Supongamos una cláusula where con una condición que involucra columna de tablas diferentes, pero con igual significado lógico, tal como:

```
select emp. nroleg, emp.nombre, fam.tipo, fam.nombre
from emp, fam
where emp.nroleg = fam.nroleg;
```

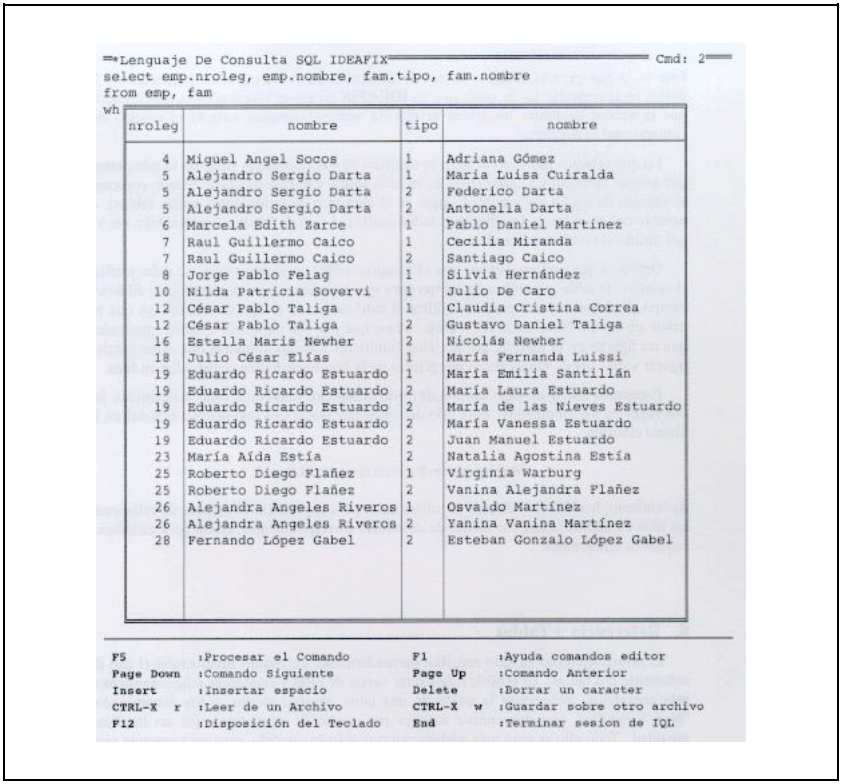


Figura 19.14

Esto es lo que permite efectuar lo que se denomina un JOIN explícito ( ) porque se lo indica en la consulta. Es de notar que en **IDEAFIX** no existe una sentencia JOIN, puesto que la acción de juntar las tablas se ejecuta automáticamente cuando el sentido del comando as' lo requiere.

Lo que estamos pidiendo al SQL es una lista de los empleados (filas de la tabla **emp**) que tengan familiares a cargo (filas de la tabla **fam**), y queremos que el listado contenga el número de legajo del empleado (que es el elemento de unión entre ambas tablas), el nombre del empleado, el código que individualiza el tipo de familiar (esposa, hijo, etc.) y por último el nombre del familiar.

Debido a que existen campos con el mismo nombre en ambas tablas, se debe prefijar el

nombre de tabla al nombre de campo para que la referencia no sea ambigua. El único campo en el que hubiera podido omitirse la calificación es el tipo de familiar, ya que no existe un campo "tipo" en la tabla **emp**. Véase que por una parte van a existir empleados que no figuran en el reporte, por no tener familiares a cargo, mientras que otros pueden figurar varias veces, tantas como integrantes de la familia se hallen registrados en **fam**.

Pongamos ahora un caso un poquito más complejo: queremos listar únicamente los cónyuges de los empleados. Para ello debemos introducir una condición adicional en la última cláusula:

```
where emp.nroleg = fam.nroleg and tipo=1;
```

Resumiendo, la cláusula where se utiliza tanto para especificar la manera de relacionar las filas de ambas tablas por medio de columnas correspondientes, como para establecer requisitos adicionales.

Referencia a Tablas

El primer ejemplo ha sido sencillo, particularmente en cuanto involucraba el uso de solamente dos tablas. Es posible referenciar varias de ellas y establecer relaciones más complejas, incluyendo la unión de una tabla consigo misma, y los denominados "Non Equijoins", que determinan uniones por medio de operadores que no implican igualdad. Todo ello se verá más adelante en este mismo capítulo, pero para exponer esos temas con claridad, es preciso introducir el concepto de "alias" o seudónimo para designar tablas.

Asignando un Alias a una Tabla

Tanto por conveniencia como por necesidad, surgió la opción de sustituir el nombre de una tabla por una denominación temporaria creada por el usuario. Esto hace más flexible y fácil de usar al SQL, a la par que lo convierte en una herramienta más poderosa, al permitir operaciones que de otro modo no serían factibles.

Los alias se especifican en la cláusula **from** a continuación del nombre de la tabla y antes de la coma que la separa de la tabla siguiente, según el modelo indicado a continuación:

```
select A.campo1, A.campo2, ..., B.campo1, B.campo2, ...,
C.campo1, C.campo2, ..., D.campo1, D.campo2, ..., D.campoN
from tabla1 A, tabla2 B, tabla3 C, tabla4 D
where (condiciones...);
```

Conviene empezar el análisis por la cláusula **from**. En ella indicamos que la tabla llamada "tabla1" va a ser designada como "A", que la "tabla2" va a llamarse "B", la "tabla3" será "C" y por último "tabla4" se convierte en "D". Es por eso que el comando **select** alude a los distintos campos con prefijos que no corresponden en realidad a tablas existentes; no obstante, el sistema substituye los alias por los nombres verdaderos, como por ejemplo C.campo2 se convierte en tabla3.campo2, gracias a lo cual puede cumplir lo requerido sin inconvenientes.

El ejercicio del apartado "La Cláusula where", valiéndose de alias, se formula entonces:

```
select E.nroleg, E.nombre, tipo, F.nombre
from emp E, fam F
where E.nroleg = F.nroleg and tipo = 1;
```

Siendo la salida la siguiente:

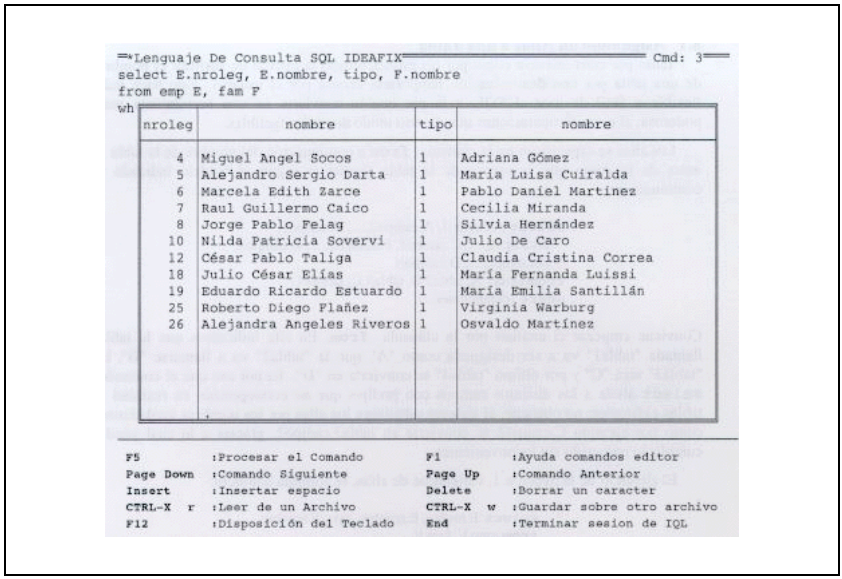


Figura 19.15

Para tomar todos los campos de una tabla, por ejemplo la de empleados, sigue siendo válido indicar E.*, y en general, es lícito usar el alias en reemplazo del nombre de la tabla en cualquier circunstancia. Existen sí dos limitaciones evidentes, dado que no puede usarse como alias:

a. El nombre de otra tabla.

b. Una palabra reservada (keyword).

Si bien no es una imposibilidad desde el punto de vista de vista del lenguaje, es altamente recomendable por razones de claridad no utilizar como alias el nombre de un campo de otra tabla.

Unión de una Tabla consigo misma

Supongamos que se quiere listar a uno o varios gerentes, poniendo a continuación de cada uno

los empleados que de él dependen. Esta operación, relativamente sencilla de hacer para cualquier empleado administrativo por un simple examen cuidadoso de la tabla **emp**, resulta poco menos que imposible de definir al sistema a menos que se recurra al artificio de la redefinición múltiple. Asignando dos alias a una misma tabla, podemos simular la existencia de dos tablas, diferencias; o más exactamente, de los ejemplares de aquélla. Puesto que ambos ejemplares son idénticos tanto en estructura como en contenido, su unión resulta en definitiva un JOIN de la tabla original consigo misma:

```
select Jefe.nroleg, Jefe.nombre, nroleg, nombre
from emp, emp Jefe
where jefe = Jefe.nroleg and depno < 3;
```

Siendo la salida obtenida:

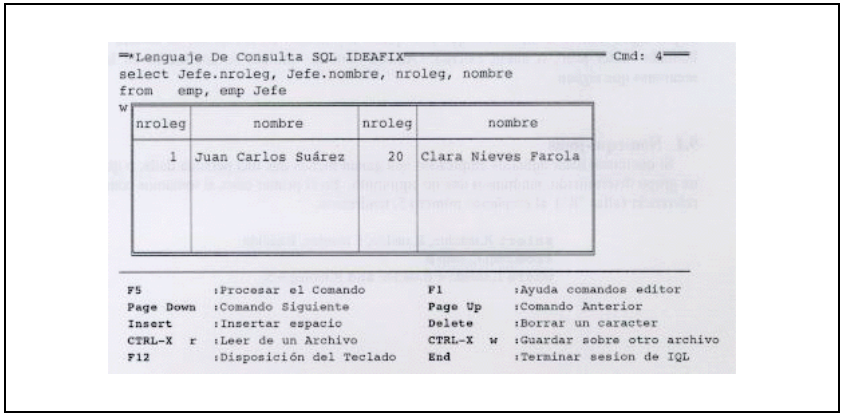


Figura 19.16

¿Qué es lo que hace este query”? Por lo pronto, utiliza dos denominaciones distintas para la tabla de personal: su propio nombre emp y el alias Jefe, que será utilizado para identificar los campos pertenecientes al gerente, mientras que el nombre original se referirá a los campos propios del empleado. Los campos a desplegar son: el número y nombre del jefe, y el legajo y nombre del empleado, pero solamente cuando se cumplan dos requisitos: que el número contenido en el campo “jefe” del empleado coincida con el legajo del jefe, y que el número de departamento del empleado sea menor que tres.

Téngase en cuenta que esta última condición no excluye los casos en que el jefe pertenezca a un departamento mayor o igual a 3, o incluso carezca de departamento (valor **null** en la última columna de **Jefe**), ya que lo que interesa es el número de departamento del empleado. Por supuesto, hubiera podido condicionarse el query al campo Jefe.depno -número de departamento del jefe-, en cuyo caso el resultado sería algo diferente.

Finalmente, nótese que hemos puesto en práctica la recomendación de no usar como alias el

nombre de un campo: por ello la redefinición de la tabla **emp** no ha sido "jefe" -aunque hubiera sido lícito- sino que se ha empleado la mayúscula.

Otros Tipos de Unión

La relación más usual para el JOIN de tablas es igualdad entre dos campos o columnas, uno de cada tabla. Esto es lo que se denomina una "equiunión" o "equi-join". Es posible especificar cualquier tipo de relación lógica o aritmética, no sólo la igualdad, entre ambas columnas, lo que da origen a las "No equi-uniones" o "Non-equi-joins".

Por otra parte, al efectuar una unión independientemente de que la condición implique o no una equiunión, puede ocurrir que ciertas filas de alguna tabla no aparezcan en el soporte, por no darse ningún caso en el cual el conjunto de condiciones se verifique. Si por algún motivo es necesario que ésto puede explicado en el reporte, se recurre al llamado "outer join", o unión externa. Ambos tipos de uniones se analizarán en las secciones que siguen.

Non-equi-joins

Si queremos listar aquellos empleados que ganan menos que una persona dada, o que un grupo determinado, tendremos una no equiunión. En el primer caso, si tomamos como referencia (alias "R") al empleado número 5, tendremos:

```
select R.nombre, R.sueldo, E.nombre, E.sueldo
from emp E, emp R
where E.sueldo < R.sueldo and R.nroleg = 5;
```

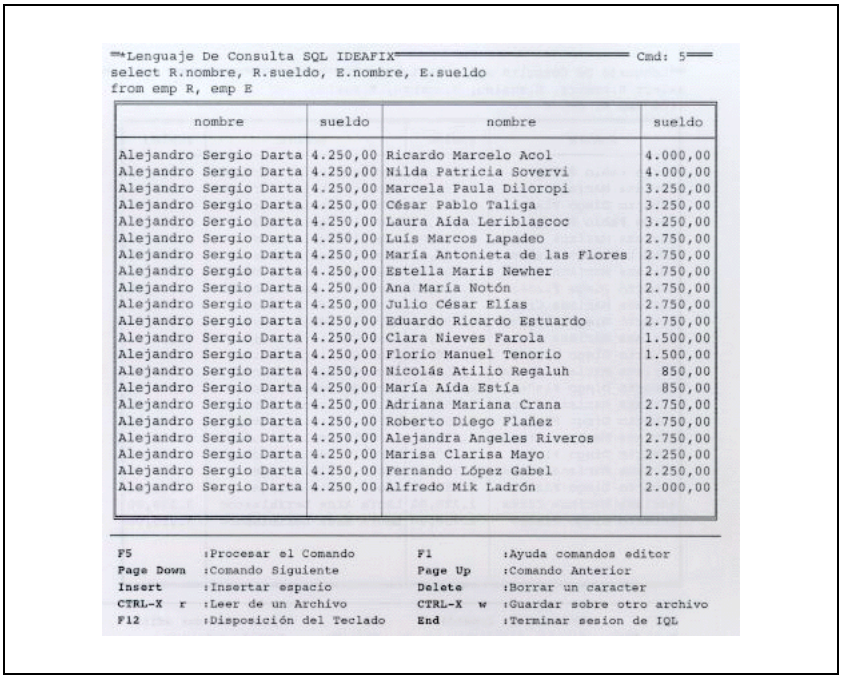
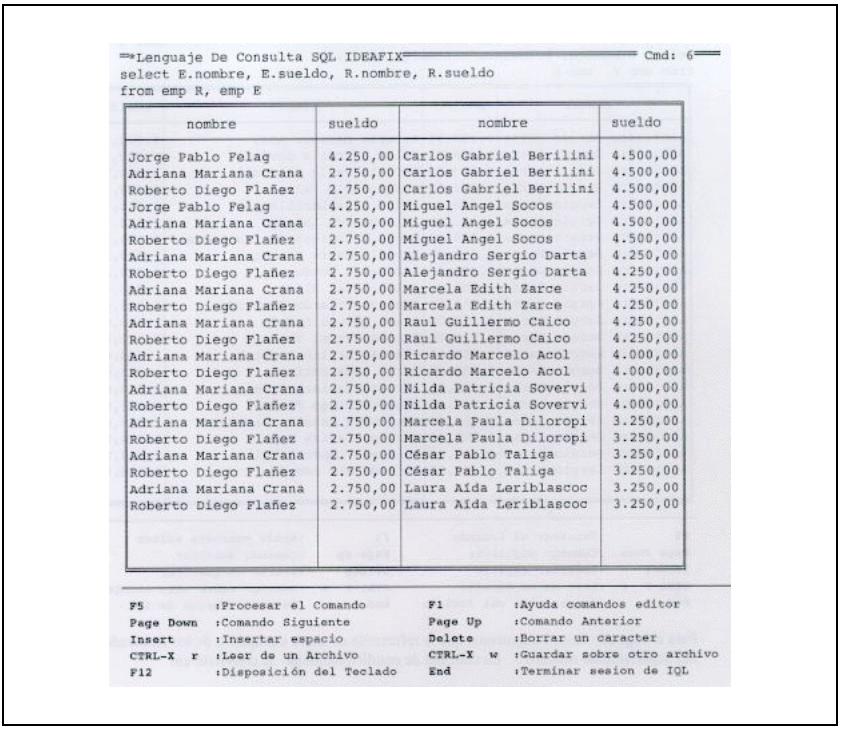


Figura 19.17

Para el segundo caso, imaginemos que la referencia se ahora cualquiera de los empleados del departamento número 3. La cláusula de condicionamiento se convierte en:

```
where R.sueldo > E.sueldo and E.depno = 3
and E.depno != R.depno;
```



nombre	sueldo	nombre	sueldo
Jorge Pablo Felag	4.250,00	Carlos Gabriel Berilini	4.500,00
Adriana Mariana Crana	2.750,00	Carlos Gabriel Berilini	4.500,00
Roberto Diego Flañez	2.750,00	Carlos Gabriel Berilini	4.500,00
Jorge Pablo Felag	4.250,00	Miguel Angel Socos	4.500,00
Adriana Mariana Crana	2.750,00	Miguel Angel Socos	4.500,00
Roberto Diego Flañez	2.750,00	Miguel Angel Socos	4.500,00
Adriana Mariana Crana	2.750,00	Alejandro Sergio Darta	4.250,00
Roberto Diego Flañez	2.750,00	Alejandro Sergio Darta	4.250,00
Adriana Mariana Crana	2.750,00	Marcela Edith Zarce	4.250,00
Roberto Diego Flañez	2.750,00	Marcela Edith Zarce	4.250,00
Adriana Mariana Crana	2.750,00	Raul Guillermo Caico	4.250,00
Roberto Diego Flañez	2.750,00	Raul Guillermo Caico	4.250,00
Adriana Mariana Crana	2.750,00	Ricardo Marcelo Acol	4.000,00
Roberto Diego Flañez	2.750,00	Ricardo Marcelo Acol	4.000,00
Adriana Mariana Crana	2.750,00	Nilda Patricia Sovervi	4.000,00
Roberto Diego Flañez	2.750,00	Nilda Patricia Sovervi	4.000,00
Adriana Mariana Crana	2.750,00	Marcela Paula Diloropi	3.250,00
Roberto Diego Flañez	2.750,00	Marcela Paula Diloropi	3.250,00
Adriana Mariana Crana	2.750,00	César Pablo Taliga	3.250,00
Roberto Diego Flañez	2.750,00	César Pablo Taliga	3.250,00
Adriana Mariana Crana	2.750,00	Laura Aida Leriblascoc	3.250,00
Roberto Diego Flañez	2.750,00	Laura Aida Leriblascoc	3.250,00

Figura 19.18

El último **and** evita que se comparen entre sí los empleados que pertenecen ambos al departamento 3 -cosa que en la realidad puede no desearse que ocurra-, para ilustrar cómo a veces pueden escaparse ciertos detalles con las no equi-uniones. Otro aspecto importante de ellas es que debe preverse la posibilidad de que cada fila de una tabla llegue a unirse con una multiplicidad de filas de la otra, dando lugar a un listado de volumen considerable.

Uniones Externas

Tratemos de obtener una lista de empleados de la empresa que tengan familiares, obteniendo el tipo y el nombre del familiar.

```
select emp.nroleg, emp.nombre, tipo, fam.nombre
from emp, fam
where emp.nroleg = fam.nroleg;
```

Obtendremos como resultado:

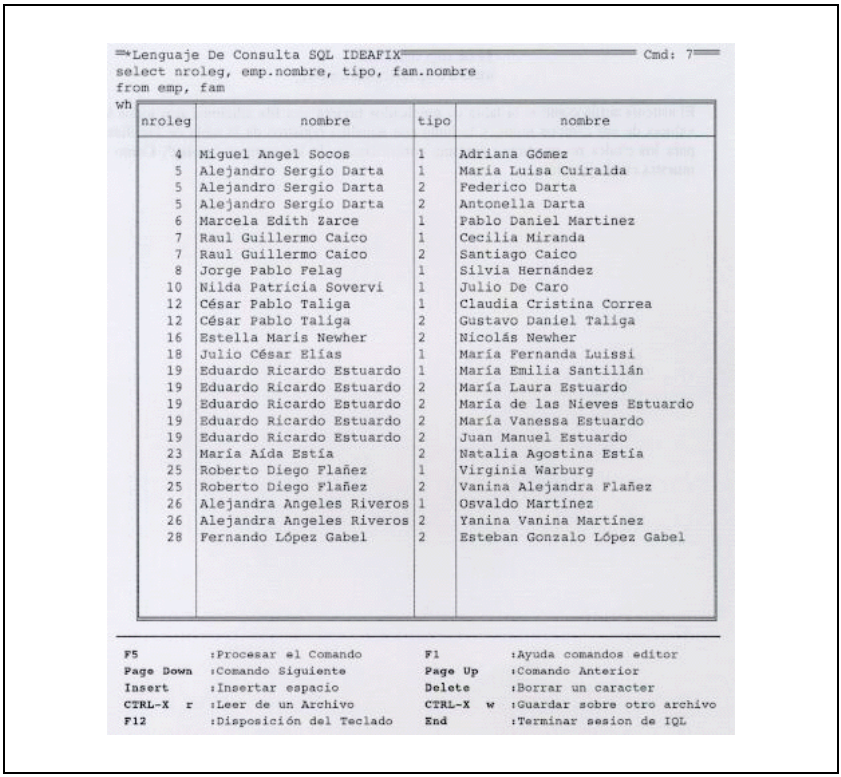


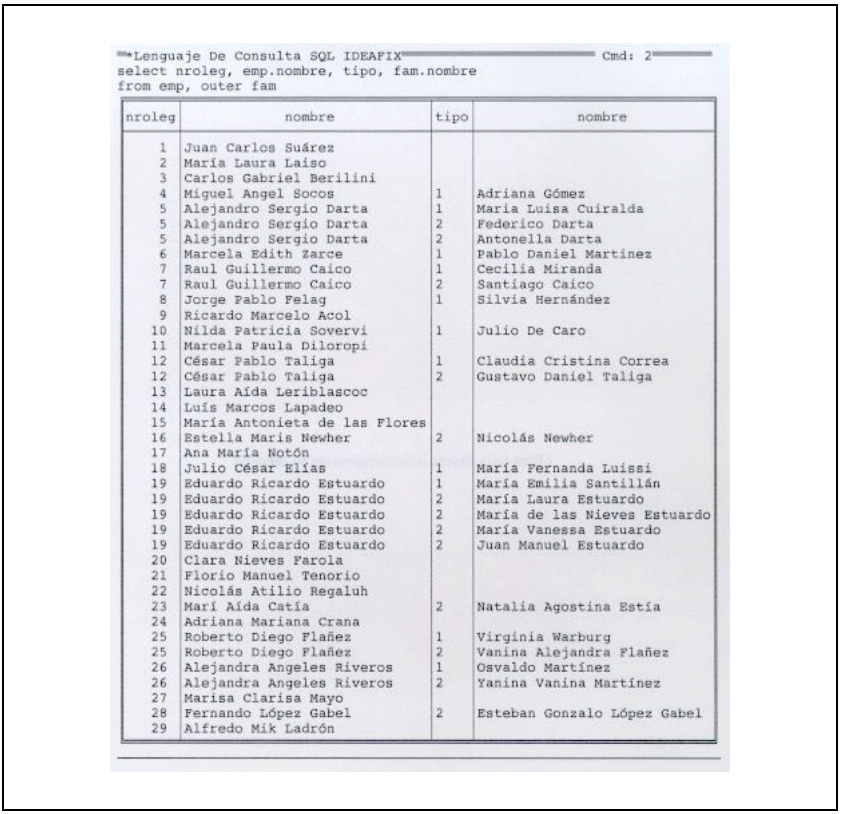
Figura 19.19

Y qué pasó con el resto de los empleados? Sencillamente, ocurre que no tienen familiares. Introduciendo el operador outer, que es el “outer join” o unión externa, se agregaría al final del reporte una línea con todos los datos propios de los empleados, conteniendo las columnas relacionadas a los familiares en blanco.

El operador de unión externa, que se indica mediante la **outer**, se especifica en la cláusula **from** delante de la tabla a la que se le quiere aplicar dicho operador.

```
from emp, outer fam
where emp.nroleg = fam.nroleg;
```

El sistema actúa como si la tabla de empleados tuviera una fila adicional con todos los valores de sus campos nulos, y la junta con aquellos registros de la tabla de familiares para los cuales no se obtuvo ninguna concordancia de los campos “nroleg”. Como se muestra en el ejemplo:



Lenguaje De Consulta SQL IDEAFIX Cmd: 2

```
select nroleg, emp.nombre, tipo, fam.nombre
from emp, outer fam
```

nroleg	nombre	tipo	nombre
1	Juan Carlos Suárez		
2	María Laura Laiso		
3	Carlos Gabriel Berilini		
4	Miguel Angel Socos	1	Adriana Gómez
5	Alejandro Sergio Darta	1	Maria Luisa Cuiralda
5	Alejandro Sergio Darta	2	Federico Darta
5	Alejandro Sergio Darta	2	Antonella Darta
6	Marcela Edith Zarce	1	Pablo Daniel Martínez
7	Raul Guillermo Caico	1	Cecilia Miranda
7	Raul Guillermo Caico	2	Santiago Caico
8	Jorge Pablo Felag	1	Silvia Hernández
9	Ricardo Marcelo Acol		
10	Nilda Patricia Sovervi	1	Julio De Caro
11	Marcela Paula Diloropi		
12	César Pablo Taliga	1	Claudia Cristina Correa
12	César Pablo Taliga	2	Gustavo Daniel Taliga
13	Laura Aída Leriblascoc		
14	Luis Marcos Lapadeo		
15	María Antonieta de las Flores		
16	Estella Maris Newher	2	Nicolás Newher
17	Ana María Notón		
18	Julio César Elías	1	María Fernanda Luissi
19	Eduardo Ricardo Estuardo	1	María Emilia Santillán
19	Eduardo Ricardo Estuardo	2	María Laura Estuardo
19	Eduardo Ricardo Estuardo	2	María de las Nieves Estuardo
19	Eduardo Ricardo Estuardo	2	María Vanessa Estuardo
19	Eduardo Ricardo Estuardo	2	Juan Manuel Estuardo
20	Clara Nieves Farola		
21	Florio Manuel Tenorio		
22	Nicolás Atilio Regaluh		
23	Marí Aída Catía	2	Natalia Agostina Estía
24	Adriana Mariana Crana		
25	Roberto Diego Flañez	1	Virginia Warburg
25	Roberto Diego Flañez	2	Vanina Alejandra Flañez
26	Alejandra Angeles Riveros	1	Osvaldo Martínez
26	Alejandra Angeles Riveros	2	Yanina Vanina Martínez
27	Marisa Clarisa Mayo		
28	Fernando López Gabel	2	Esteban Gonzalo López Gabel
29	Alfredo Mik Ladrón		

Figura 19.20

Capítulo 20

Cambiando el Orden de las Filas

Hasta aquí hemos supuesto que la secuencia en la cual aparecían los registros de la tabla principal -usualmente la de empleados, en los ejemplos propuestos- era satisfactoria a los fines buscados. Pero muchas veces, en las tareas de procesamiento, es preciso alterar el **orden** en el cual se despliega la información. Esto se logra en los quieres introduciendo la cláusula **order by** la cual debe ir después de la cláusula **where** si ésta aparece, o bien a continuación de la cláusula **from**. En otras palabras, si se la utiliza debe ser la última cláusula del query (excepción hecha de la cláusula output to).

Para quienes tengan conocimientos previos de computación, esta cláusula es equivalente al comando o función sort en otros sistemas o lenguajes, ya que produce un reordenamiento de los registros (filas) de la tablas.

Características de la Clasificación

La cláusula de ordenamiento contiene dos palabras clave que deben ir seguidos por una lista de expresiones -puede ser una sola, pero al menos una que indican los sucesivos niveles de clasificación. Como se recordará, las listas consisten en una serie de items separados por comas en este caso los items son expresiones-; además, es optativo indicar si la clasificación es ascendente o descendente, lo cual se hace mediante las keywords **asc** o **desc** respectivamente. Puesto que se trata de algo optativo, es lícito omitirlo; en este caso el sistema supone que se trata de orden ascendente. Aquí tenemos un ejemplo de lo que se denomina “default value”, o valor por defecto: en general, vale decir para cualquier circunstancia y tipo de lenguaje, el valor por defecto es lo que el sistema va a suponer como informado, cuando se ha omitido hacerlo en forma expresa.

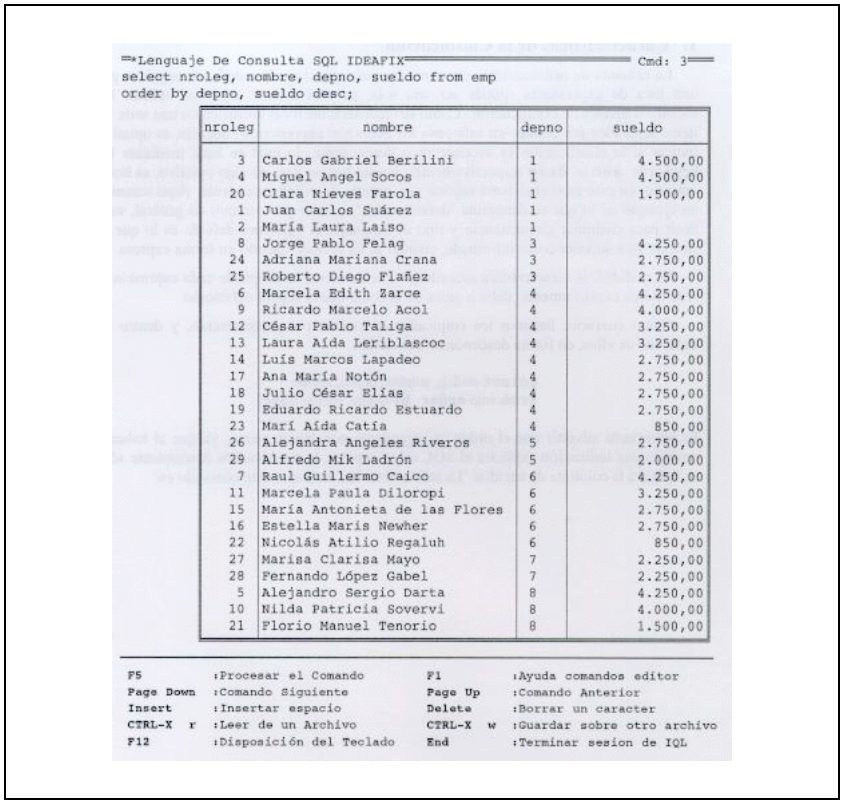
En realidad, la característica ascendente o descendente es propia de cada expresión, y si se indica explícitamente, debe ir antes de la coma que la separa del campo.

Como ejercicio, listemos los empleados ordenados por departamento, y dentro de cada uno de

ellos, en forma descendente por sueldo.

```
select nroleg, nombre, depno, sueldo
from emp orter by depno, sueldo desc;
```

Es importante advertir que el orden es ascendente por departamento, ya que al haberse omitido una indicación explícita el SQL así lo supone: la clasificación descendente sólo se aplica a la columna de sueldos. La salida obtenida mediante este comando es:



```
=*Lenguaje De Consulta SQL IDEAFIX=========== Cmd: 3===
select nroleg, nombre, depno, sueldo from emp
order by depno, sueldo desc;
```

nroleg	nombre	depno	sueldo
3	Carlos Gabriel Berlini	1	4.500,00
4	Miguel Angel Socos	1	4.500,00
20	Clara Nieves Farola	1	1.500,00
1	Juan Carlos Suárez	1	
2	María Laura Laiso	1	
8	Jorge Pablo Felag	3	4.250,00
24	Adriana Mariana Crana	3	2.750,00
25	Roberto Diego Flañez	3	2.750,00
6	Marcela Edith Zarce	4	4.250,00
9	Ricardo Marcelo Acol	4	4.000,00
12	César Pablo Taliga	4	3.250,00
13	Laura Aída Leriblascoc	4	3.250,00
14	Luís Marcos Lapadeo	4	2.750,00
17	Ana María Notón	4	2.750,00
18	Julio César Elías	4	2.750,00
19	Eduardo Ricardo Estuardo	4	2.750,00
23	Marí Aída Catía	4	850,00
26	Alejandra Angeles Riveros	5	2.750,00
29	Alfredo Mik Ladrón	5	2.000,00
7	Raul Guillermo Caico	6	4.250,00
11	Marcela Paula Diloropi	6	3.250,00
15	María Antonieta de las Flores	6	2.750,00
16	Estella Maris Newher	6	2.750,00
22	Nicolás Atilio Regaluh	6	850,00
27	Marisa Clarisa Mayo	7	2.250,00
28	Fernando López Gabel	7	2.250,00
5	Alejandro Sergio Darta	8	4.250,00
10	Nilda Patricia Sovervi	8	4.000,00
21	Florio Manuel Tenorio	8	1.500,00

```
F5          :Procesar el Comando      F1          :Ayuda comandos editor
Page Down   :Comando Siguiente        Page Up     :Comando Anterior
Insert      :Insertar espacio         Delete      :Borrar un caracter
CTRL-X  r   :Leer de un Archivo       CTRL-X  w   :Guardar sobre otro archivo
F12         :Disposición del Teclado  End         :Terminar sesion de IQL
```

Figura 20.1

Es importante notar un aspecto que a veces se omite considerar: ¿qué ocurre cuando tanto el departamento como el sueldo son iguales para dos o más empleados? En rigor, el orden de salida es arbitrario, de modo que no existe garantía alguna de que Lapadeo, Notón, Elías y Estuardo (del Depto. 4, sueldo 2750) salgan en orden de legajo como se ha mostrado y no en otro orden cualquiera. Para asegurarnos esto habría que agregar al final de la cláusula **order by** el campo nroleg.

Cómo se Ordenan los Campos?

Campos Numérico

En el ejemplo anterior, ambas columnas de clasificación tenían un contenido numérico, y lo que es más, intrínsecamente positivo. Los campos numéricos pueden tener en ocasiones valores negativos, y es importante tener en cuenta que lo sistemas en general no solamente el SQL respetarán los principios algebraicos, según los cuales los valores negativos son inferiores a cero (y por ende **anteriores** a él en una clasificación ascendente), y de dos valores negativo, el **menor** (el que tiene precedencia) es el de mayor absoluto. Tomando como referencia los números enteros, el esquema de ordenamiento ascendente es:

```
... -3 -2 -1 0 1 2 3 ...
```

Existen campos de contenido numérico pero que tienen un significado especial: son los campos de fecha (DATE) y los de hora (TIME), que responden a ciertas convenciones que se verían más adelante.

Expresiones

Una columna numérica puede ser el resultado de una expresión, calculada en función de ciertas operaciones aritméticas sobre el contenido de campos de la tabla. Es evidente que si se utiliza dicha columna como elemento de clasificación, valen los principios algebraicos expuestos en el apartado anterior.

Pero una columna definida por una expresión carece de nombre, pues no pertenece a la tabla originaria, y no es válido referirla por el encabezado o título que pueda asignársele, pues eso no es un nombre para el SQL. Para ordenar por una columna calculada, es preciso repetir la expresión que se formulará en **select**, en la cláusula **order by**, como se ilustra en el ejemplo siguiente:

```
select nroleg, nombre, sueldo+comis “INGRESOS”
from emp
order by sueldo + comis desc;
```

La salida de esta sentencia es:

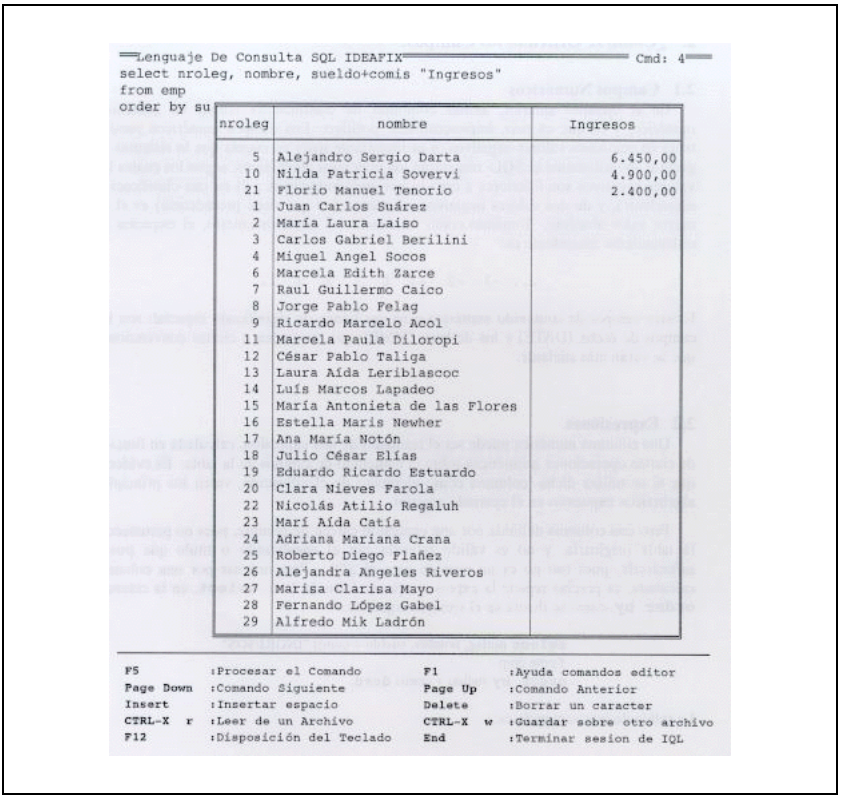


Figura 20.2

Este query lista los ingresos totales de los empleados (sueldo más comisiones) de mayor a menor, poniendo el título respectivo en la columna calculada. En rigor, no es necesario que la expresión haya figurado en la cláusula **select**: puede utilizarse una expresión incluyéndola directamente en **order by**. Esta implementación, propia del **iql**, no está soportada en otras implementaciones (v.gr. el SQL de IBM) y brinda una facilidad importante, como es la capacidad de ordenar un listado por una o varias expresiones que no es necesario hacer figurar como columnas a la salida, dejando espacio para otra información.

Como se observa en la salida del ejemplo anterior, algunas filas han quedado con valor indefinido para la columna INGRESOS. En el apartado de "Los Valores NULL" se explicará el motivo de esto.

Como contrapartida, por ahora no es posible en **iql** indicar una columna calculada en el **order by** por su número de ubicación, ya que por una parte este número sería tomado por si mismo como una expresión válida. En una próxima implementación del **iql** se planea superar esta

restricción permitiendo especificar el número de la columna a través de la función col, lo que daría lugar, en el ejemplo antes visto, a una cláusula final:

```
   order by col(3) desc; (no válida en la actual implementación)
```

Campos de Caracteres

Estos campos pueden contener cualquier tipo de información, la cual en la práctica suele ser alfabética, si bien es lícita la presencia de todo tipo de símbolos: letras, dígitos y caracteres especiales.

Como es obvio, para que un listado por orden alfabético tenga utilidad, la información debe registrarse dentro de los campos involucrados en forma apropiada: cuando se trate de personas, ubicando en primer término el apellido y después los nombre. En el caso de mujeres casadas, existen las alternativas:

LOPEZ de GARCIA, Susana

GARCIA, Susana López de

Según se desee ubicarlas por el apellido de soltera o de casada.

La implementación **iql** de **IDEAFIX** ubica correctamente las eñes entre la “N” y la “O”, la “CH” entre la “C” y la “D”. Esta es una característica importante, de la cual carece generalmente el software de origen estadounidense o europeo.

Como ejercicio, hagamos lo siguiente:

```
   select * from depno
order by ubic, nombre;
```

Los departamentos se listarán en el orden: 2, 7, 1, 3, 8, 4, 5, 6 (como muestra el ejemplo a continuación del párrafo); esto es, primero todos los ubicados en Buenos Aires y luego los de Rosario, y dentro de cada ciudad, alfabéticamente por orden del departamento.

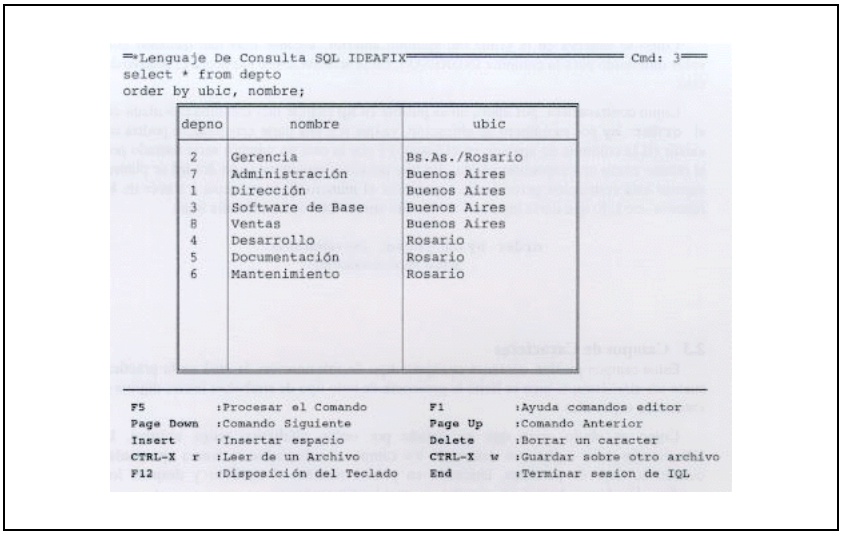


Figura 20.3

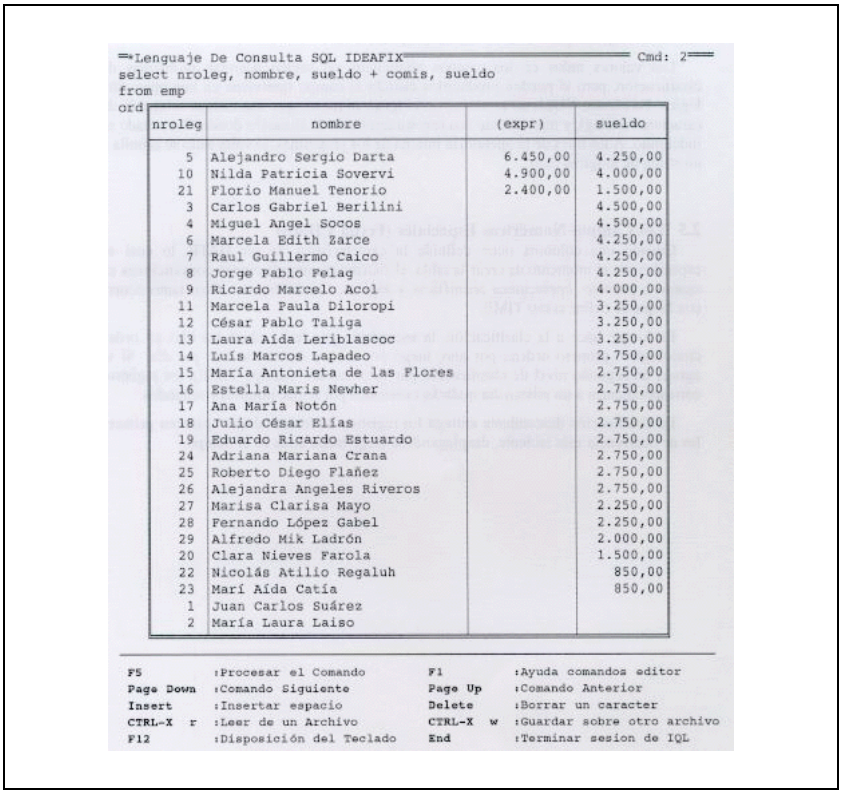
Los Valores NULL

Si usted se tomó el trabajo de ejecutar el ejercicio del apartado "Expresiones", le habrá llamado la atención la ausencia de ingresos para varios empleados en la grilla resultante. Esto se debe a que carecían de información en alguno de los campos involucrados en el cálculo de la expresión “sueldo + comis”. Cuando un campo carece de información se dice que contiene un valor NULO (NULL value), lo cual es distinto de un contenido igual a cero para un campo numérico. Cero es un valor perfectamente definido, mientras que NULL equivale a un valor desconocido, y toda operación efectuada sobre algo **desconocido** arroja un resultado indefinido. En consecuencia, toda expresión que involucre para un determinado caso un campo cuyo valor es nulo, produce como resultado también un valor NULL.

Por convención, las columnas con valor nulo se ordenan como si se tratara del valor más bajo de todos, de modo que en nuestro ejemplo aparecerán al final porque la clasificación es descendente.

Probemos ahora con:

```
select nroleg, nombre, sueldo+comis, sueldo
from emp
order by sueldo + comis desc, sueldo desc;
```



nroleg	nombre	(expr)	sueldo
5	Alejandro Sergio Darta	6.450,00	4.250,00
10	Nilda Patricia Sovervi	4.900,00	4.000,00
21	Florio Manuel Tenorio	2.400,00	1.500,00
3	Carlos Gabriel Berilini		4.500,00
4	Miguel Angel Socos		4.500,00
6	Marcela Edith Zarce		4.250,00
7	Raul Guillermo Caico		4.250,00
8	Jorge Pablo Felag		4.250,00
9	Ricardo Marcelo Acol		4.000,00
11	Marcela Paula Diloropi		3.250,00
12	César Pablo Taliga		3.250,00
13	Laura Aída Leriblascoc		3.250,00
14	Luís Marcos Lapadeo		2.750,00
15	María Antonieta de las Flores		2.750,00
16	Estella Maris Newher		2.750,00
17	Ana María Notón		2.750,00
18	Julio César Elías		2.750,00
19	Eduardo Ricardo Estuardo		2.750,00
24	Adriana Mariana Crana		2.750,00
25	Roberto Diego Flañez		2.750,00
26	Alejandra Angeles Riveros		2.750,00
27	Marisa Clarisa Mayo		2.250,00
28	Fernando López Gabel		2.250,00
29	Alfredo Mik Ladrón		2.000,00
20	Clara Nieves Farola		1.500,00
22	Nicolás Atilio Regaluh		850,00
23	Marí Aída Catía		850,00
1	Juan Carlos Suárez		
2	María Laura Laiso		

Figura 20.4

Si bien quienes carecen de comisión continuarán quedando detrás de los que sí la tienen (aún cuando su solo sueldo fuera mayor que la suma de sueldo y comisión para estos últimos), al menos quedan ordenados por el importe del sueldo.

Para obtener un listado completo y ordenado, según el significado **vulgar** de la expresión “sueldo + comis”, donde el resultado de sumar NULL es equivalente a sumar 0, sería preciso usar el operador condicional “?”, capacidad avanzada del sistema que se verá en la segunda parte de este manual.

Los valores nulos en los campos alfabéticos no suelen acarrear problemas de clasificación, pero si pueden producirlos cuando el campo interviene en una operación lógica. En efecto, NULL no es ni mayor, ni igual, ni menor que una cadena arbitraria de caracteres (string), y nuevamente nos encontramos ante la situación donde el resultado es indefinido. A los fines de la operatoria interna de los programas, el valor nulo se asimila a un string de longitud cero.

Los Campos Numéricos Especiales (Fecha y Hora)

Cuando una columna tiene definida la característica de ser DATE, lo cual se especificó en el momento de crear la tabla, el sistema contempla ciertas convenciones en cuanto a formato, operaciones aritméticas y lógicas, funciones, etc. Otro tanto ocurre con la que se define como TIME.

En lo que hace a la clasificación, la ascendente por fecha ubica las filas en orden cronológico: primero ordena por año, luego por mes, y dentro de éste por día. Si se agrega un segundo nivel de clasificación por valor horario (campo Time), los registros correspondientes a un mismo día quedarán ordenados por horas, minutos y segundos.

la clasificación descendente entrega los registros de forma tal que aparecen primero los de ocurrencia más reciente, desplegándose luego hacia atrás en el tiempo.

Capítulo 21

Operadores y Funciones

A partir de este capítulo se expondrán posibilidades más sofisticadas del SQL. Se supone la comprensión de lo expuesto en los capítulos previos, para lo cual es muy conveniente haber practicado los ejercicios propuestos usando las tablas ejemplo.

Las capacidades que se exponen consisten fundamentalmente en los operadores y las funciones que brinda el SQL de IDEAFIX. En relación con las funciones se verán las que permiten trabajar con el valor de un campo, y son las llamadas funciones individuales.

EL Operador like y los Metacaracteres

A veces ocurre que queremos condicionar un **select** por el contenido de un cierto campo, tal como un apellido, y no estamos seguros de su grafía exacta. Un par de símbolos denominados metacaracteres, en conjunción con el operador **like**, nos permiten hacer una comparación aproximada según las reglas que veremos a continuación.

El operador **like** sustituye al signo igual en una cláusula **where**, en la cual el primer operado puede ser un campo, y el segundo una “seudo constante”, que es un string de caracteres con partes fijas y partes variables. El signo de interrogación “?” significa un carácter arbitrario y se puede utilizar cuantas veces y en los lugares que sea necesarios. Así CA??N puede equivaler a : CAZON,CASON,CASAN,CATAN, etc. Pero no a CARMEN o CAMION, pues sólo hay dos letras incógnitas intercaladas y no tres. Veamos un ejemplo:

```
select apellido from obreros
where apellido like 'GOM??';
```

en una hipotética tabla llamada “obreros” en la cual exista un campo denominado “apellido”, puede producir el resultado siguiente:

GOMAR

GOMES

GOMEZ

GOMMI

Como es obvio, para que el query sirva a algún propósito se hubieran debido listar además otros campos de los obreros en cuestión, pero lo principal es comprender el mecanismo que el SQL aplica para hacer la selección.

El restante metacaracter es el asterisco, que simboliza un strintg arbitrario de “n” caracteres, donde **n** puede valer desde cero (NULL string) hasta 65535, que es la máxima longitud admisible para una cadena de caracteres. Probemos ahora un ejemplo concreto:

```
select * from cargos
where descrip like '*ente';
```

¡A no confundirse! El primer asterisco **no es** un metacaracter: es el símbolo adoptado por el SQL para representar todos los campos de la tabla sin tener que detallar sus nombres. La segunda ocurrencia **sí** es un metacaracter, y lo que hace es pedir la selección de todos los cargos que terminen en 'ente'. El resultado deberá ser:

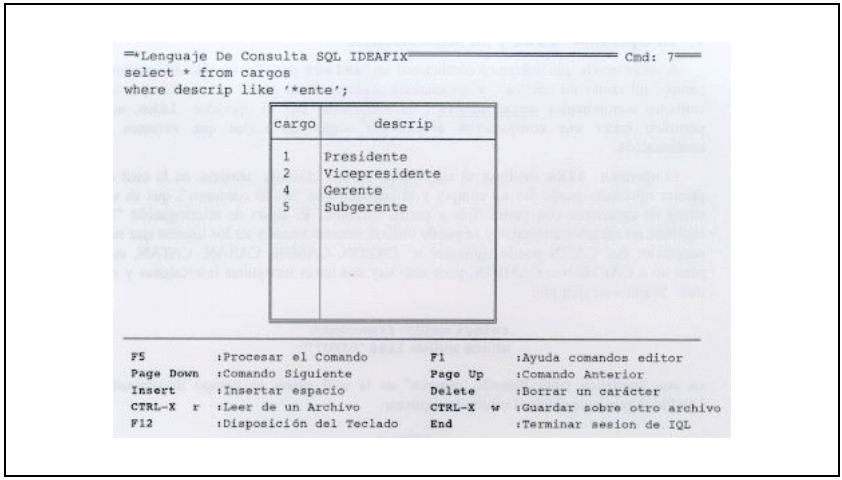


Figura 21.1

Volviendo a nuestra anterior hipotética tabla de obreros, el query:

```
select apellido from obreros
where apellido like 'G*ez';
```

puede brindar una lista en la que figuren: Gálvez, Gómez, Grau Martinez, González, ... y también Gez, si tal apellido existiera.

Por último, se puede anteponer el operador lógico de negación **not** a la especificación **like** como por ejemplo:

```
select apellido from obreros
where apellido not like 'A*';
```

listará todos los apellidos que no comienzan con A.

Operadores in y between

Permiten efectuar comparaciones extendidas. En el primer caso, contra una lista de valores operador IN de modo que la condición se cumple si el contenido de la columna figura en la lista. Recordando que lista va encerrada entre paréntesis, con sus elementos separados por comas, probemos el query:

```
select * from cargos
where cargo in (1,3,4,7);
```

La salida obtenida es:

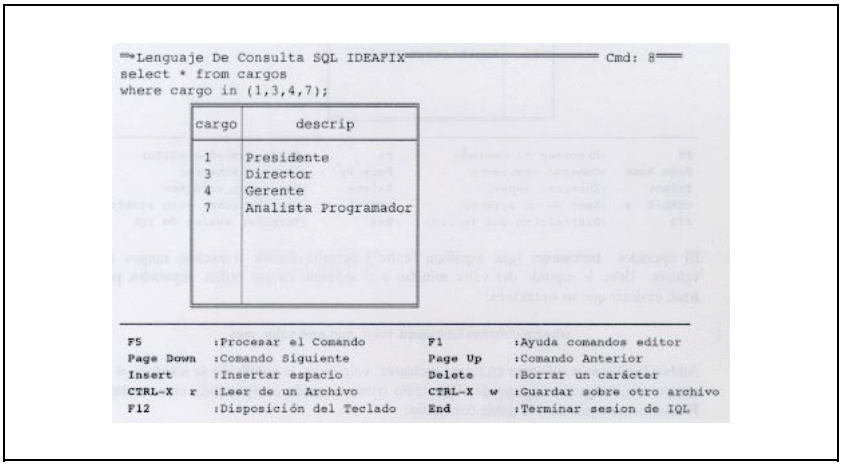


Figura 21.2

Obtendremos, como era de esperar, la información correspondiente a los cargos 1,3,4,7. Como en el caso de **like** es posible prefijar la negación **not** al operador **in** para excluir los casos que figuren en la lista, incluyendo todos los demás. Por consiguiente:

```
select * from cargo
where cargo not in (2,5,6);
```

debe producir exactamente el mismo resultado anterior, la salida a esta línea de comando es:

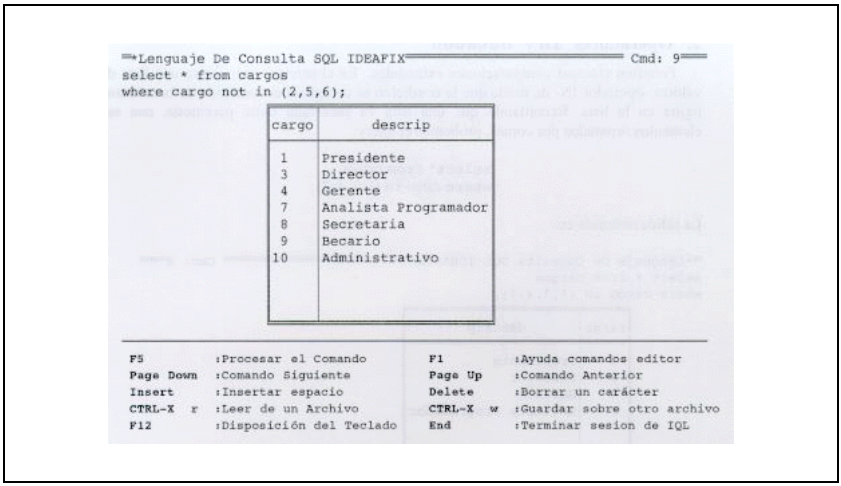


Figura 21.3

El operador **between** (que significa “entre”) permite incluir o excluir rangos de valores. Debe ir seguido del valor mínimo y el máximo en ese orden, separados por **and**, es decir que su sintaxis es:

```
where columna between valor_min and valor_max
```

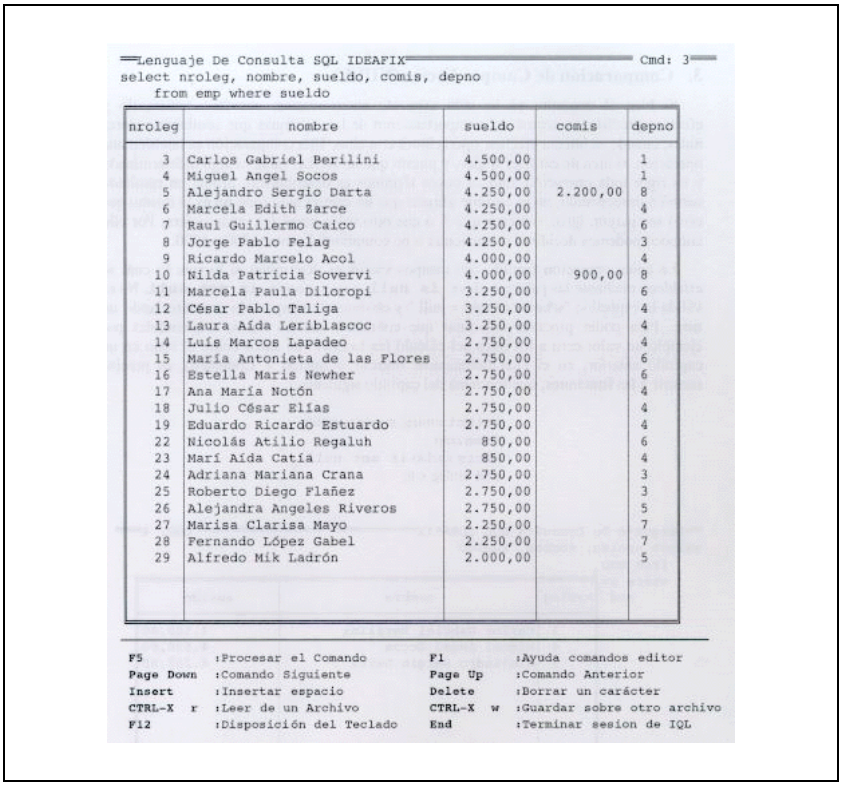
Ambos límites se consideran en forma inclusiva; vale decir, la condición se satisface si el campo resulta igual a alguno de ellos, tanto como si queda comprendido entre ambos. Formalmente equivale a la doble condición:

```
where columna >= valor_min and columna <= valor_max
```

Como ejercicio, probemos el query:

```
select nroleg, nombre, sueldo, comis, depno
from emp where sueldo not between 1300 and 1900;
```

cuya salida es:



Lenguaje De Consulta SQL IDEAFIX — Cmd: 3

```
select nroleg, nombre, sueldo, comis, depno
    from emp where sueldo
```

nroleg	nombre	sueldo	comis	depno
3	Carlos Gabriel Berilini	4.500,00		1
4	Miguel Angel Socos	4.500,00		1
5	Alejandro Sergio Darta	4.250,00	2.200,00	8
6	Marcela Edith Zarce	4.250,00		4
7	Raul Guillermo Caico	4.250,00		6
8	Jorge Pablo Felag	4.250,00		3
9	Ricardo Marcelo Acol	4.000,00		4
10	Nilda Patricia Sovervi	4.000,00	900,00	8
11	Marcela Paula Diloropi	3.250,00		6
12	César Pablo Taliga	3.250,00		4
13	Laura Aída Leriblascoc	3.250,00		4
14	Luís Marcos Lapadeo	2.750,00		4
15	María Antonieta de las Flores	2.750,00		6
16	Estella Maris Newher	2.750,00		6
17	Ana María Notón	2.750,00		4
18	Julio César Elías	2.750,00		4
19	Eduardo Ricardo Estuardo	2.750,00		4
22	Nicolás Atilio Regaluh	850,00		6
23	Marí Aída Catía	850,00		4
24	Adriana Mariana Crana	2.750,00		3
25	Roberto Diego Flañez	2.750,00		3
26	Alejandra Angeles Riveros	2.750,00		5
27	Marisa Clarisa Mayo	2.250,00		7
28	Fernando López Gabel	2.250,00		7
29	Alfredo Mik Ladrón	2.000,00		5

```
F5          :Procesar el Comando        F1          :Ayuda comandos editor
Page Down   :Comando Siguiente          Page Up     :Comando Anterior
Insert      :Insertar espacio           Delete      :Borrar un carácter
CTRL-X  r   :Leer de un Archivo         CTRL-X  w   :Guardar sobre otro archivo
F12         :Disposición del Teclado    End         :Terminar sesion de IQL
```

Figura 21.4

¿por qué no aparece los legajos nro. 1, que **no tienen** un sueldo comprendido entre los valores dados? ¿Por qué no figuran los empleados con sueldo igual a 1900? Si no logra responder a ésto último, relea este apartado. Si no da con la respuesta a la primera cuestión, repase el apartado "Los Valores NULL" del Capítulo 20 y prosiga con la siguiente.

Comparación de Campos Vacíos (NULL)

Si bien el concepto ya ha sido expuesto anteriormente, conviene remarcarlo y efectuar precisiones acerca del comportamiento de las columnas que contienen valores nulos, cuando se intenta efectuar operaciones con ellas. Una comparación es también una operación, si bien de carácter lógico, y puesto que un NULL value es algo indeterminado y en rigor toda operación, uno de cuyos términos es desconocido, brinda un resultado también desconocido, no es factible afirmar que un campo nulo (que **no es** lo mismo que cero) sea mayor, igual, o menor que 5, o que otro valor numérico cualesquiera. Por ello tampoco podemos decidir si se

encuentra o no comprendido entre 1300 y 1900.

La única operación factible con campos vacíos es determinar si lo son, lo cuál se establece mediante las palabras clave **is null**, o su negación **is not null**. No es válida la expresión "**where** campo = null" y obviamente tampoco lo es intercalando un **not**. Para poder procesar columnas que contienen valores nulos, asignándoles por ejemplo un valor cero a los fines del cálculo (es lo obvio en casos como el visto en un capítulo anterior, en el cual definíamos: Ingreso = Sueldo + Comisión), es preciso recurrir a las **funciones**, que son tema del capítulo siguiente.

```
select nroleg, nombre, sueldo
from emp
where sueldo is not null
and nroleg < 6;
```

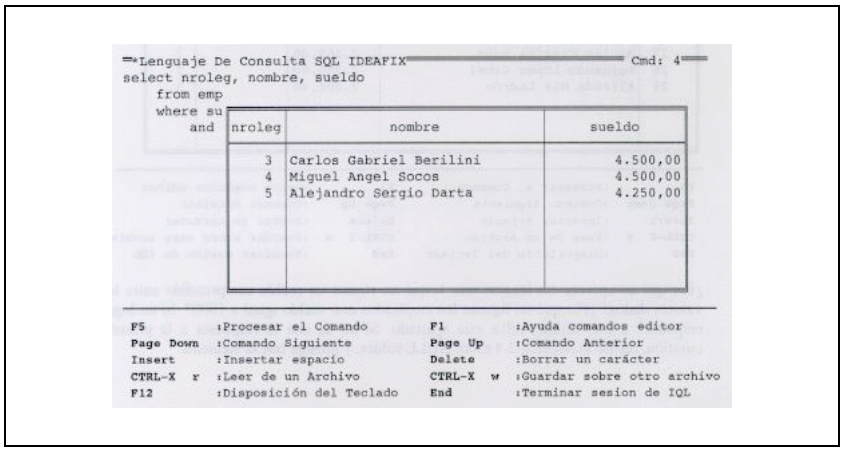


Figura 21.5

deberá listar los datos de los empleados 2, 3, 4, y 5. ¿Qué pasa si subtituimos el **and** por un **or**? Pues que ahora se incluyen todos los empleados. Los legajos Nro. 1 y 2 entran por ser menor que 6 aunque tengan vacío el campo de sueldo, ya que el **or** sólo requiere que se cumpla **una** de las condiciones propuestas. Los restantes con la primera condición de tener sueldo asignado.

Precedencia de Operadores

Cuando el usuario define una expresión mediante los operadores algebraicos de suma, resta, multiplicación, división y potenciación, para crear una nueva columna en la grilla o establecer un elemento de comparación en la cláusula **where**, está en realidad creando una función. Desde el punto de vista matemático, una función no es más que una relación entre varias

variables, tal que el valor de una de ellas la llamada variable “dependiente” queda definido si se conoce el valor de una o más que son las restantes (llamadas “independientes”). La variable dependiente se identifica usualmente con la función: Así en el apartado "Expresiones"del Capítulo 18 definimos la función “Ingreso” como la suma de las variables 'sueldo' y 'comisión'.

El citado ejemplo constituye la más simple de las funciones: la lineal. Cuando una expresión se hace más compleja, es importante tener en cuenta lo que se denomina **precedencia** de los operadores algebraicos. Las operaciones inversas entre sí (suma/resta, multiplicación/división, potenciación/radicación) tienen igual prioridad. La de las tres parejas entre ellas se acorde a su complejidad: la potenciación es más complicada que la multiplicación, y esta lo es más que la suma, de modo que la máxima precendencia (las operaciones que se realizan primero) la tienen potenciación y radicación, la de nivel intermedio es la de multiplicación y división, y la menor prioridad corresponde a suma y resta.

En la expresión:

se procede a computar cada término por separado, que son: A; el producto de B y C, y un tercer término que a su vez implica dos etapas: elevar D al cuadrado, y luego dividir el resultado por E. Una vez hecho todo esto, se sumará el primer término al segundo y se le restará el tercero si bien el orden de las operaciones es irrelevante, ya que lo mismo es restar el tercer término del segundo y después sumarle el primero.

Para alterar la precedencia natural de los operadores se usan los paréntesis. Dando los valores A=3; B =5; C =2; D =8, y E =4, la evaluación de la expresión nos conduce al resultado -3. Proponemos al lector hacer lo mismo con:

y luego con:

Si los resultados son F =0 y G = 9, usted ha hecho bien los deberes.

Las Funciones del Sistema

Llamaremos funciones individuales a las que se refieren a una fila o registro de la tabla, y permiten obtener un valor a partir de operaciones que se realizan sobre otros valores que pueden ser variables (el contenido de una columna) o constantes (un número explícito).

Existen ciertas operaciones frecuentes que se brindan al usuario “listas para usar”, como el truncado de decimales o el redondeo. La primera, que se denomina **trunc** (por 'truncate'), toma la parte entera de un valor numérico seguida por la cantidad deseada de decimales. Así

trunc (SALDO, 0), cuando saldo valga 158.82, devolverá158. En cambio

trunc (SALDO,1) produce el resultado 158,3.

Los usos más frecuentes de trunc son para reducir un número a su parte entera, o bien a dos decimales (centavos) cuando se trata de montos de dinero.

La función **round** es similar, pero sólo cuando la fracción decimal es inferior a . Cuando es

igual o superior, se toma el dígito siguiente. Así **round** (PRECIO * CANT, 2), siendo PRECIO=38,25/kg. y CANT=0.37(370 gramos) produce un importe de 14,15. El resultado exacto tienen cuatro decimales: 14,1525. En cambio una cantidad de 430 gr. o sea 0,30 da lugar a que **round** devuelva el valor 16,45, si bien la cifra exacta es 16,4475.

Como se ha visto en los casos anteriores, una función puede recibir varios **argumentos**. Se llama Así a los diversos valores ya sean variables, constantes o expresiones, dados en forma de lista ordenada que la función espera recibir para efectuar un cierto conjunto de operaciones sobre ellos y producir un resultado. Las funciones **trunc** y **round** reciben dos valores: en primer término aquello que se desea truncar a redondear, y luego la cantidad de decimales que debe tener el resultado.

La función **abs** (SALDO) devuelve el valor absoluto de la variable: “saldo”, que en el caso de un cuenta bancaria puede se positiva o negativa el valor absoluto de un número positivo es igual a sí mismo, mientras que el de un número negativo se obtienen cambiándole el signo. Como es evidente el valor absoluto de cero es cero, y para todos los demás casos debe ser siempre positivo.

El **iql** brinda al usuario las funciones aritméticas que se detallan a continuación con los operadores correspondientes:

Función	Operador	Significado
ADD	+	Suma de dos Números (adición)
DIF	-	Diferencia de dos números
MULT	*	Producto de dos Números (multiplicación
DIV	/	Cociente de dos números (división)
POW	^	Potenciación: el pimer número elevado al exponenete cuyo valor es el segundo número.
REM	%	Residuo de dividir el primer número por el 2¼.

La forma de expresar “D elevado al cuadrado”, que en el ejemplo se indicó con la notación algebraica convencional:

es en realidad en **iql** D^2. Asimismo (A%5) es el residuo de dividir A por 5; si A=19, devuelve el valor 4.

Las funciones indicadas en la primera columna del cuadro anterior no se hallan implementadas en iql por ser innecesario: los operadores actúan directamente de la manera que convenga al usuario si se adoptan los recaudos mínimos que se detallarán al tocar el tema de **precisión** de los resultados. Se las ha incluido al solo efecto de referencia comparativa, por

ser sus nombres usuales en otros lenguajes.

La Función Expresión Condicional (operadores ?:)

La implementación **IDEAFIX** del SQL introduce una función que permite elegir entre dos expresiones, según que una expresión resulte verdadera o falsa. El formato de la función es:

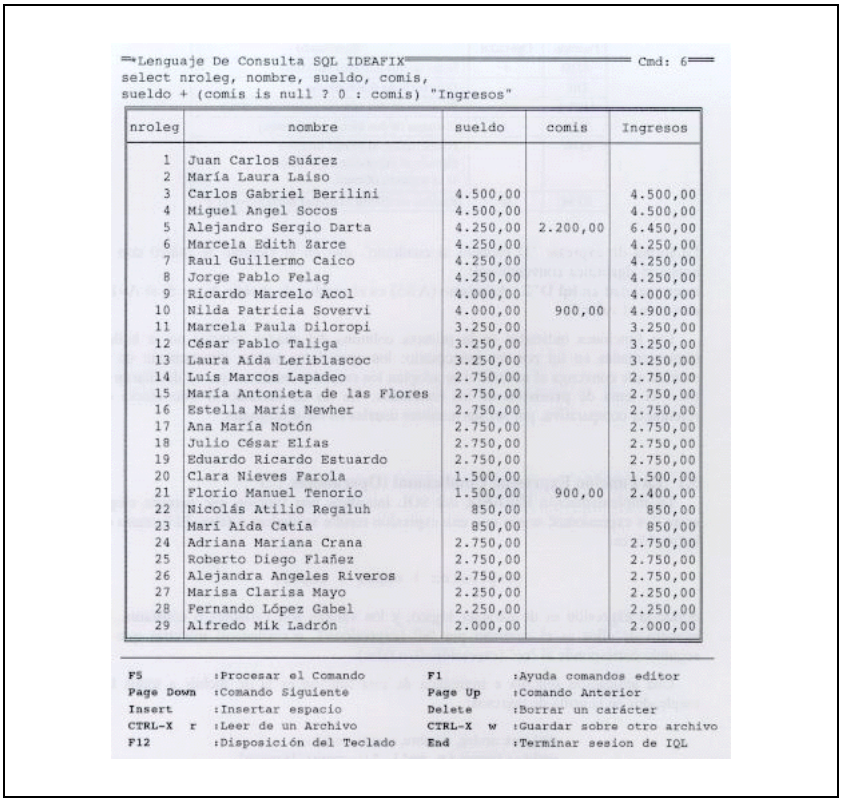
```
(expresión? expr1 : expr2)
```

donde la expresión es de carácter lógico, y los valores son variables o constante. El primero de ellos es el devuelto por “sí” (expresión = 1, o verdadera) mientras que el segundo corresponde al “no” (expresión =0, o falsa).

Una aplicación sencilla e inmediata de esta función es la de incluir a todos los empleados en la grilla de ingresos:

```
select nroleg, nombre, sueldo, comis,
sueldo + (comis is null ? 0 : comis) "ingresos"
from emp;
```

Ahora, antes de indicarle al SQL que sume el sueldo la comisión, averiguamos si esta última consiste en una columna vacía. Si la respuesta es afirmativa la función devuelve cero (primer valor), pero en caso contrario devuelve el valor de la comisión como se muestra en la salida ejemplo.



Lenguaje De Consulta SQL IDEAFIX — Cmd: 6

```
select nroleg, nombre, sueldo, comis,
sueldo + (comis is null ? 0 : comis) "Ingresos"
```

nroleg	nombre	sueldo	comis	Ingresos
1	Juan Carlos Suárez			
2	María Laura Laiso			
3	Carlos Gabriel Berilini	4.500,00		4.500,00
4	Miguel Angel Socos	4.500,00		4.500,00
5	Alejandro Sergio Darta	4.250,00	2.200,00	6.450,00
6	Marcela Edith Zarce	4.250,00		4.250,00
7	Raul Guillermo Caico	4.250,00		4.250,00
8	Jorge Pablo Felag	4.250,00		4.250,00
9	Ricardo Marcelo Acol	4.000,00		4.000,00
10	Nilda Patricia Sovervi	4.000,00	900,00	4.900,00
11	Marcela Paula Diloropi	3.250,00		3.250,00
12	César Pablo Taliga	3.250,00		3.250,00
13	Laura Aída Leriblascoc	3.250,00		3.250,00
14	Luís Marcos Lapadeo	2.750,00		2.750,00
15	María Antonieta de las Flores	2.750,00		2.750,00
16	Estella Maris Newher	2.750,00		2.750,00
17	Ana María Notón	2.750,00		2.750,00
18	Julio César Elías	2.750,00		2.750,00
19	Eduardo Ricardo Estuardo	2.750,00		2.750,00
20	Clara Nieves Farola	1.500,00		1.500,00
21	Florio Manuel Tenorio	1.500,00	900,00	2.400,00
22	Nicolás Atilio Regaluh	850,00		850,00
23	Marí Aída Catía	850,00		850,00
24	Adriana Mariana Crana	2.750,00		2.750,00
25	Roberto Diego Flañez	2.750,00		2.750,00
26	Alejandra Angeles Riveros	2.750,00		2.750,00
27	Marisa Clarisa Mayo	2.250,00		2.250,00
28	Fernando López Gabel	2.250,00		2.250,00
29	Alfredo Mik Ladrón	2.000,00		2.000,00

```
F5          :Procesar el Comando        F1          :Ayuda comandos editor
Page Down   :Comando Siguiente          Page Up     :Comando Anterior
Insert      :Insertar espacio           Delete      :Borrar un carácter
CTRL-X  r   :Leer de un Archivo         CTRL-X  w   :Guardar sobre otro archivo
F12         :Disposición del Teclado    End         :Terminar sesion de IQL
```

Figura 21.6

Veamos otro ejemplo. Supongamos que nuestra tabla de personal tuviera una columna llamada “faltas”, en la cual se registra la cantidad de ausencias en el mes. Se admiten hasta tres, más allá de lo cual se deberá descontar el importe correspondiente a los días no trabajados. Esto se logra con:

```
select ..., sueldo,
sueldo - (faltas > 3 ? sueldo * faltas / 30:0)
...
```

Independiente de otras columnas que puedan introducirse, las que interesan en este caso son el sueldo mensual y los haberes efectivamente percibidos. El primer valor de la mientras que si ello no ha ocurrido, no hay descuento y ambas columnas (sueldo y haberes) serían iguales.

Vemos que un valor a seleccionar en este caso el descuento puede ser a su vez el resultado de una expresión, pero ahora de carácter numérico, como es el cálculo por prorrateo de los días

no trabajados.

Funciones para Campos de Caracteres

En las secciones anteriores hemos visto funciones y operadores aplicables a valores numéricos, que son los más importantes en líneas generales. No obstante, los “strings” también tienen su corazoncito, y es justo que nos ocupemos un poco de ellos.

para los alcances de este manual, el único operador importante que trabaja sobre cadenas de caracteres se denomina “concatenación”, que no es otra cosa que unir una cadena con otra: con -catenar (catena significa cadena en latín). Si por ejemplo el campo “letra” tiene el contenido “alpha” y el campo “ciudad” el valor “ville”, la concatenación:

```
letra | | ciudad
```

da como resultado: *Alphaville.*

El operador de concatenación se simboliza por dos barras verticales consecutivas; en algunos lenguajes, una sola de ellas equivale al operador lógico OR.

Un atributo importante de un campo en caracteres es un longitud. Tal valor es el que devuelve la función **length**, que obviamente debe ser un número entero. Una característica de los campos vacíos o nulos (NULL) es que la función **length** aplicada a ellos devuelve el valor cero: la cantidad de caracteres que contienen es efectivamente cero. Insistimos una vez más: esto en modo alguno significa que el contenido del campo sea asimilable a cero, sino que lo es su longitud.

Es inmediato Así mismo que **length**(A||B) = **length**(A) + **length**(B): la longitud de un campo obtenido por concatenación de otros dos es la suma de las longitudes individuales.

Las funciones **lower** y **upper** convierten un campo respectivamente a minúsculas (“lower case”) o mayúsculas (“upper case”). “Case” significa “tipo de imprenta”, y hasta recordar que “low” es”bajo”y “upper” significa “superior”. Un par de ejemplos:

```
upper (ciudad) devuelve VILLE
upper ('ciudad') es... CIUDAD
```

¡Atención al uso de las comillas!

Queda por analizar el operador “substring”. Conviene empezar por definir el concepto de “subcadena”: es simplemente una porción del campo total, seleccionada arbitrariamente a partir de una cierta posición y una determinada longitud. En **iql**, 3sto se expresa con una notación consistente en indicar a continuación del nombre del campo, y entre paréntesis, dos argumentos separados por una coma. En principio, y conforme a la definición clásica de la función “substring”, el primer valor será el desplazamiento a partir del origen del campo, y el segundo la longitud.

Retornado el ejemplo original de este apartado:

```
letra (0,3) devuelve 'Alp'
```

es decir, los tres primeros caracteres de “Alpha”. El desplazamiento es cero o sea que comenzamos por el principio, y la longitud es tres. En cambio.

```
letra (3,2)
```

devuelve los dos último caracteres: “ha”. El primer argumento puede visualizarse como “la cantidad de caracteres que se incorporan”. Es inmediato que si L=**length**(campo), entonces la subcadena campo(0,L) no es otra cosa que el propio **campo** completo. Asimismo, constituye un error que la suma de los dos argumentos exceda la longitud del campo: se estaría intentando incorporar caracteres inexistentes, es decir, ubicados más allá del fin del campo.

En caso de no especificar el segundo parámetro, se tomará la subcadena desde la posición indicada en el primer argumento hasta el final de la cadena originaria.

Pero existe una segunda forma de usar esta función, aplicando el concepto de separador o delimitador. Si dentro de un string existen varios datos separados por un carácter especial -el delimitador- podemos obtener uno de estos datos parciales en la siguiente forma: ahora el primer argumento es el carácter delimitador, y el segundo -siempre numérico entero en ambas implementaciones- es el número de orden de la ocurrencia que nos interesa. Un ejemplo clarificará las cosas.

Supongamos que el campo “fecha” contiene “21/08/89”. La expresión **fecha** (“/”,1) nos devuelve “80”, por ser la primera ocurrencia de un campo después del delimitador, que es el mes. Se considera que los subcampos están comprendidos entre el comienzo y el primer delimitador, o entre dos de ellos, o bien entre el último delimitador y el fin del campo.

Para visualizar la manera de operar en esta modalidad, imaginemos barrer el campo de izquierda a derecha la cantidad de ocurrencias del delimitador (primer argumento); cuando se haya alcanzado el valor indicado por el segundo operado, se comienza a tomar los sucesivo caracteres que integren el campo, hasta la próxima aparición del delimitador o del campo. Cuando la cantidad de ocurrencias indicada es cero, significa tomar desde el comienzo del campo hasta la primera aparición del delimitador. En un campo “peynom”, que contenga el o los apellidos seguido de los nombres, separados uno y otro tipo de datos por coma:

Gonzales, Carlos Alberto

Rodriquez Larreta, Enrique

Bullrich de Moreno Ocampo, Josefina Inés

podemos descomponer esta información mediante las expresiones:

```
apeynom(',',1) equivale al nombre
apeynom(',',0) equivale al apellido
```

Nótese que al estar encerrada entre apóstrofos o comillas, la primera coma se toma como un literal (en este caso el delimitador), mientras que la segunda constituye la indicación de que detrás viene otro argumento.

Funciones con Fechas

Cuando es necesario obtener un elemento de una fecha (día, mes o año) en cierto formato se pueden aplicar funciones sobre campos de tipo fecha.

Las funciones que veremos a continuación aceptan como argumento un campo o expresión de tipo fecha y retornan un valor numérico. La función **year** retorna un valor numérico correspondiente al año de la fecha dada. La función **month** retorna un número entre 1 y 12 correspondiente al mes, y la función **day** el número del día.

Veamos algunos ejemplos:

```
month ("10/12/88") es 12
day ("10/12/88") es 10
year("10/12/88") es 1988
```

La función **dayname** nos da resultado una cadena de caracteres con el nombre del día de la semana correspondiente a la fecha que se le pasa como argumento. Por ejemplo:

```
dayname ("15/12/89") es VIERNES
```

La última función de este grupo es **monthname**, que recibe un argumento numérico y da como resultado una cadena de caracteres con el nombre del mes. Por ejemplo:

```
monthname (12) es DICIEMBRE
```

En algunos casos puede ser necesario combinar estas funciones. Supongamos que deseamos conocer el nombre del mes en el que ha ingresado cada empleado. Para ello consultamos la tabla **emp**, de la que nos interesa la columna **fingi**r. Pero como deseamos nada más que el nombre del mes asaríamos la expresión:

```
monthname(month (fingir))
```

Esta expresión debe entenderse del siguiente modo: primero se ejecuta la función **month** la cual recibe como argumento la fecha y dará como resultado un valor numérico; este valor será pasado como argumento para la función **monthname** la cual nos dará la cadena de caracteres con el nombre del mes que es lo que deseamos mostrar en la salida.

Funciones de Conversión

Las funciones de Conversión son muy útiles cuando se desea convertir una expresión a otro tipo. La función **char** toma un argumento de cualquier tipo y lo convierte en una cadena de caracteres. Es muy útil cuando se desea concatenar valores de distintos tipos. Supongamos que se deseará imprimir una fecha en el siguiente formato:

```
Viernes 15 de Diciembre
```

Para lograr esta cadena de caracteres a partir de una fecha (seguiremos usando la fecha de

ingreso de la tabla **emp**) usaríamos la expresión:

```
dayname(fingir) | | " " | | char(day(fingr)) | | " de " | |
monthname(month (fingr))
```

La función **day** retorna un valor numérico, por ello hemos utilizado la función **char** para que convierta el valor numérico en una cadena y pueda ser concatenado con los otros componentes de la cadena con el espacio en blanco es para separar el número del nombre del día.

La otra función de Conversión es llamada **num** y convierte una expresión cualquiera en un valor numérico. El resultado dependerá del tipo de expresión que se reciba como argumento, siendo estos:

- Cadena de caracteres: La cadena de caracteres se convertirá al número que representa. Se toma desde el comienzo de la cadena hasta que se encuentre un carácter que no es un dígito, o el fin de la misma. Por ejemplo **num (“123ª”)** dará como valor 123.
- Fecha: Retornará un número que es el número de días transcurridos desde el 01/01/84.
- Hora: Retornará la cantidad de segundos transcurridos desde la hora cero.

Otras Funciones

Restan mencionar algunas funciones. Existe un par de funciones que permiten obtener el mayor o menor de los parámetros que se le suministran y se denominan **major** y **minor** respectivamente. Por ejemplo si deseamos imprimir el monto mayor entre el sueldo y la comisión podemos usar:

```
major (sueldo, comis)
```

Lógicamente todas las expresiones que se pasan como argumento estas funciones deben ser del mismo tipo.

La función **descr** se aplica sobre un campo que sólo puede tomar un valor de un conjunto, es decir, que tiene asociada lo que se denomina una validación **in**. Como cada valor tiene asociada al valor presente en la columna. Por ejemplo se puede usar sobre el campo **tipo** de la tabla **fam**:

```
select nombre, descr (tipo) from fam;
```

En lugar de obtener simplemente el código de tipo de familiar, imprimiremos la descripción asociada al código.

Función count

Recordaremos que un query tal como:

```
select count (comis) from emp;
```

contaba la cantidad de filas de la tabla **emp** -aquí el grupo es la tabla completa por no haberse especificado condicionamientos- para las cuales existía un valor (Así fuera cero) del campo comisión, pero no se contaban los NULL values. Esto es válido en general para todas las funciones: los casos nulos no se toman en cuenta. Veremos que Esto tiene importancia en casos como la función promedio, que será descripta más adelante.

Para hacerlo actuar realmente como una función de grupo, podríamos agregar al query anterior la cláusula:

```
where depno = 4
```

Ahora sí el grupo resulta constituido por los empleados pertenecientes al departamento 4. Habíamos visto también que para contar todos los casos del grupo sin discriminación se utilizaba el formato **count** (*); es decir:

```
select count(*) from emp where depno = 4;
```

nos da la cantidad de personal del departamento en cuestión, ya sea que puedan recibir comisión o no.

Los Valores de Fecha y Hora

Existen dos tipos especiales de campos numéricos, por el significado particular de la información contenida en ellos: los de **fecha** (DATE) y **hora** (TIME). Cada uno de ellos debe manejarse con una aritmética particular, y existe una serie de funciones que facilitan su manipulación.

Desentendiéndonos por ahora del formato interno utilizado por el sistema para almacenar estos campos -lo cual excede los alcances de este manual- analizaremos las posibilidades de manejo de la información y las reglas de juego que al respecto se aplican.

Los Campos de Fecha

Deben ser descriptos con el tipo “date” al momento de definir la tabla de la cual forman parte; esto es, en los DDS (Data Definition Statements) que se verán en el Capítulo 25. El **iql** despliega las fecha en el formato usual en nuestro país, denominado “europeo” por los americanos del norte: tres campos de dos dígitos, separados por barras, que suelen simbolizarse “dd/mm/aa”. Esto significa que se muestra primero el día, luego el mes y finalmente los dos últimos dígitos del año. Así quince de febrero de 1990 aparece como 15/02/90.

Cuando se ingreso por primera vez un campo de fecha, o se modifica una ya existente, el sistema verifica su validez, teniendo en cuenta las diversas duraciones de los meses, los años bisiestos, etc.

Operaciones con Fechas

Para hallar el lapso en días transcurrido entre una fecha y otra, basta restar la menor de la mayor. Supongamos que comienzo=23/12/87 y final=05/03/88. El período abarcado (final-comienzo) resulta igual a 73 días. ¿Qué son más que 72? No, estimado lector: 29/02/88 es una fecha válida.

La sustracción es la única operación aritmética válida entre dos fechas, pero se puede sumar o restar a una fecha un valor en días, para obtener otra fecha. Resulta obvio que Navidad + 7 = Año Nuevo, y que Pascua - 3 = Jueves Santo, cualquiera sea el año que se elija para dichas festividades. Las operaciones lógicas de comparación < , > e = son válidas entre fechas, resultando menor la fecha más antigua y mayor la más reciente. Por consecuencia, es l'cito incluirlas en expresiones que involucren el uso de **in** o **between.**

La condición:

```
where fingr between "01/01/89"and today
```

nos devuelve todos los empleados ingresados con posterioridad al primero de enero de 1989.

Funciones de Fecha

Para obtener la fecha del día (en realidad, la fecha del sistema; veremos en seguida que hay una sutil diferencia), basta recurrir a la función **today**. Ella puede servir para asignar valores en una tabla, usarla como elemento de cálculo, etc.- Cuando un equipo se desconecta por la noche y vuelve a ponerse en funcionamiento a la mañana siguiente, el operador debe efectuar un procedimiento llamado "boot" para volver a cargar en memoria el sistema operativo. En muchas máquinas existe un área alimentada por batería que mantiene en funcionamiento el "timer" (reloj interno), con lo cual se conservan los valores de fecha y hora en su transcurso normal. No obstante, existe la opción de que el operador corrija uno u otro valor -las baterías alguna vez se descargan-, cosa que de opción se convierte en necesidad si el equipo carece del dispositivo para mantener en funcionamiento el timer aún desconectado.

Como consecuencia de lo explicado en el párrafo anterior, al fecha del sistema será la del día siempre y cuando el operador haya hecho bien las cosas.

A partir de un campo fecha, es posible obtener por separado sus componentes mediante varias funciones cuyos nombres son auto explicativos, según se verá con los siguientes ejemplos. Suponiendo **fechaingr** = 25/11/89 (una hipotética fecha de ingreso de un empleado) se tendrá:

```
day (fechingr) = 25
month (fechingr) = 11
year (fechingr) = 1989
dayname (fechingr) = 'sabado'
monthname (fechingr) = 'noviembre'
```

Hemos puesto deliberadamente una fecha poco probable -un sábado- para mostrar la utilidad que pueden tener estas funciones: probablemente al asignar datos en una tabla, tal como el

campo **fingr** en **emp**, pongamos la condición:

```
   dayname (fingr) not in ('sabado', 'domingo')
```

Es posible aplicar las funciones grupales **max** y **min** a un conjunto de fechas, con lo cual obtendremos la más reciente (max) y la más antigua(min). Es decir, **max** (fingr) nos da al más novato y **min** (fingr) al más antiguo de los integrantes.

Los Campos de Hora

Los distingue la característica de haber sido definidos con el tipo “time” al momento de especificar la tabla que los contiene. Ya se ha mencionado la existencia de un reloj interno, incorporado al “hardware”, que se denomina “timer”. La función **hour** nos da la hora del momento de la consulta , y puede utilizarse para distinguir y ordenar eventos ocurridos en la misma fecha.

El formato con el cual se despliega un campo de hora puede simbolizarse como “hh:mm:ss”, es decir tres campos numéricos de dos dígitos cada uno, separados por el carácter “:” (dos puntos). La hora puede variar de 00 a 23, los minutos de 00 a 58 y los segundos de 00 a 59.

Internamente, el iql lleva el tiempo transcurrido del día en segundos, de modo que la diferencia de dos campos de hora es un valor numérico que expresa el intervalo en segundos. Al igual que con las fechas, la resta es la única operación aritmética válida para este tipo de campos.

Puede también sumarse o restarse un valor numérico en segundos a un campo horario, pero debe tenerse en cuenta que si el resultado es negativo o excede de 86399, hemos caído en el día anterior o el siguiente, respectivamente. Esto es consecuencia de que un día tiene 24 * 60 * 60 = 86400 segundos.

Capítulo 22

Consultas Avanzadas

Completaremos en este capítulo los conceptos necesarios para resolver consultas más avanzadas. Se verá como agrupar por distintos criterios los resultados de una consulta, y como obtener totales sobre estos grupos. Esto se consigue mediante las cláusulas **group by** y **having**, y las totalizaciones y otras operaciones sobre los grupos con las llamadas funciones **grupales**.

Se explicará también el concepto de **subquery**, que permite realizar consultas que requieren valores calculados como resultado de una consulta, para ser utilizados por otra.

Las Cláusulas group by y having

Una operación útil -y a veces imprescindible- consiste en agrupar las filas de una tabla por una o varias columnas, entendiéndose que pertenecen al mismo grupo aquellas filas que contienen valores iguales en el campo o campos especificados. Esto se logra mediante la cláusula **group by** la cual debe ir después de la **where** si la hay, y en caso contrario a continuación del **from**. La utilidad de agrupar se comprende mejor con relación a las llamadas "funciones de grupo", que devuelven un valor único característico de él, como pueden ser la suma de los valores de una cierta columna -obviamente numérica- o la cuenta de registros pertenecientes al grupo.

El concepto de grupo no está restringido al uso de la cláusula **group by**, sino que puede resultar de un condicionamiento a través del **where** pero en este caso el grupo resultante es uno solo y parte de la tabla; **group by** maneja en cambio la totalidad de la tabla trabajando y produciendo resultados con tantos grupos como resulten de acuerdo a las especificaciones usadas. Para ejemplificar usaremos la función **count** (cuenta) que da la cantidad de ocurrencias de valores no nulos en una columna del grupo, o bien la cantidad de filas que lo componen.

Si ejecutamos el query:

```
select depno, count (*) from emp group by depno;
```

Obtendremos la cantidad de empleados en cada departamento, conforme a una salida como esta:

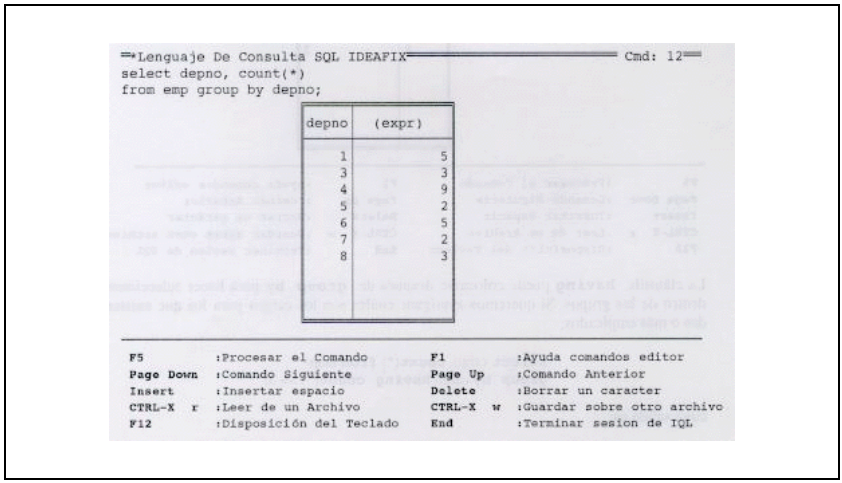


Figura 22.1

La expresión “(*)” le indica a la función que se trata de cuenta simple es decir, que no interviene ningún valor de columna.

Muy distinto es el resultado de:

```
select depno, count(comis) from emp group by depno;
```

Aquí sólo toman en cuenta (ello vale en sentido literal!) las filas de la tabla cuya columna “comis” no sea nula; es decir, le estamos pidiendo a la función que cuente, para cada departamento, cuántos empleados pueden llegar a recibir comisiones (aunque de momento el importe puede ser cero).

La salida del query es:

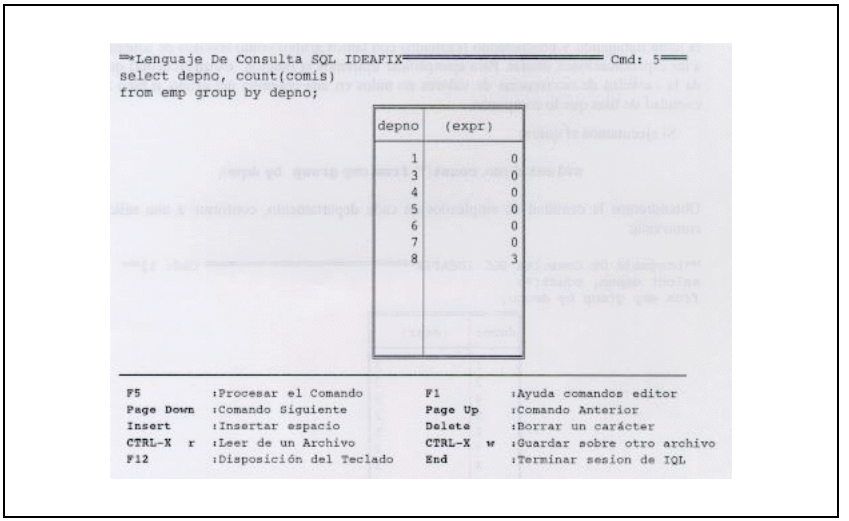


Figura 22.2

La cláusula **having** puede colocarse después de **group by** para hacer selecciones dentro de los grupos, Si queremos averiguar cuáles son los cargos para los que existen dos o más empleados:

```
select cargo, count(*) from emp
group by cargo having count(*) >= 2;
```

cuya salida es:

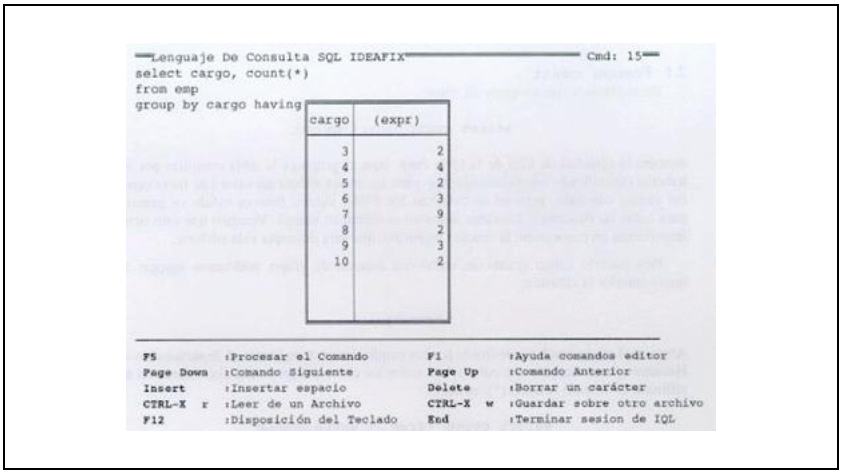


Figura 22.3

Véase que el query puede casi leerse como si tratara de inglés corriente; su expresión en castellano sería: “Seleccionar cargos y **cantidades** de la tabla de **empleados** agrupados por cargo para aquellos grupos que tengan dos o más empleados”.

Una especificación tal como:

```
group by depno,cargo
```

permite obtener información agrupada y ordenada por departamento y cargo ya sea obtenida por medio de **count** u otra de las funciones de grupo. Ahora el grupo está constituido por aquellas filas de la tabla **emp** que tienen valores respectivamente coincidentes en las columnas **depno** y **cargo**.

Las Funciones Grupales

Las funciones grupales son aplicables a un conjunto de valores pertenecientes a una determinada columna; dicho conjunto resulta de considerar ciertas filas de la tabla, que son las que constituyen el grupo, en virtud de cumplir con alguna condición estipulada. naturalmente, el grupo puede consistir de la totalidad de los registros de la tabla.

Ya hemos visto una función grupal sencilla: la **count**. Ahora veremos otras, en principio las que se aplican a campos numéricos. No obstante haremos una revisión de **count**.

Función count

Recordaremos que un query tal como:

```
select count (comis) from emp;
```

contaba la cantidad de filas de la tabla **emp** -Aquí el grupo es la tabla completa por no haberse especificado condicionamientos- para las cuales existía un valor (así fuera cero) del campo comisión, pero no se contaban los NULL values. Esto es válido en general para todas las funciones: los casos nulos no se toman en cuenta. Veremos que esto tiene importancia en casos como la función promedio, que será descripta más adelante.

Para hacerlo actuar realmente como una función de grupo, podríamos agregar al query anterior la cláusula:

```
where depno = 4
```

Ahora sí el grupo resulta constituido por los empleados pertenecientes al departamento 4. habíamos visto también que para contar todos los casos del grupo sin discriminación se utilizaba el formato **count**(*); es decir:

```
select count(*) from emp where depno = 4;
```

nos da la cantidad de personal del departamento en cuestión, ya sea que puedan recibir comisión o no.

Las Funciones Extremas max y min

Es algo que suele darse con frecuencia la necesidad de conocer, dentro de un grupo de valores, cuál es el más grande, o bien el más pequeño. En términos matemáticos hablamos de máximo y mínimo respectivamente. Para campos numéricos el significado es obvio, siempre que recordemos que 0 es **mayor** que cualquier número negativo, y entre dos de estos el de menor valor absoluto: -3 es mayor que -4.

En forma genérica, podemos definir el máximo M de un conjunto de valores:

a los que designamos globalmente como:

como el valor que cumple las dos siguientes condiciones:

para cualquier valor de i;

al menos para un valor de i.

La primer condición implica que el máximo es mayor o igual que cualquier elemento del conjunto y la segunda, que pertenece a el, es decir, se identifica por lo menos con uno de sus componentes. Por consiguiente, **max** sueldo nos dará en principio la remuneración más allá dentro de **emp** ... salvo por el hecho de la existencia de campos NULL. En buen romance, esto significa que no sabemos cuánto gana el capo, pero todos estamos convencidos de que si se supiera, la función **max** apuntará a el.

Como es evidente, todo lo dicho se aplica a la función min (mínimo) sin más que cambiar el operador de comparación a <= en la condición a. Es posible buscar la diferencia de ambas funciones pata hallar lo que se denomina rango de variación:

```
select max (sueldo) - min (sueldo) "Rango", depno
from emp group by depno
having count(*)>1;
```

Este query es un poquito más complejo de lo que estamos acostumbrados a ver, pero no hay que asustarse, vallamos por partes.

El primer campo seleccionado -0 sea lo que aparece a continuación del **select** hasta la coma-, es una expresión formada por la diferencia de las dos funciones que hemos visto en este apartado, a la cual adjudicamos un título; el segundo campo es simplemente una columna de la tabla. Agrupamos por departamento ya que eso es lo que indica la cláusula **group by** , y por último hacemos una selección de grupo de grupos mediante la cláusula **having** (no es posible usar **where** porque se aplica solo a filas individuales, no a grupos), estipulando que se considerarán solamente aquello departamentos con varias personas. El resultado es:

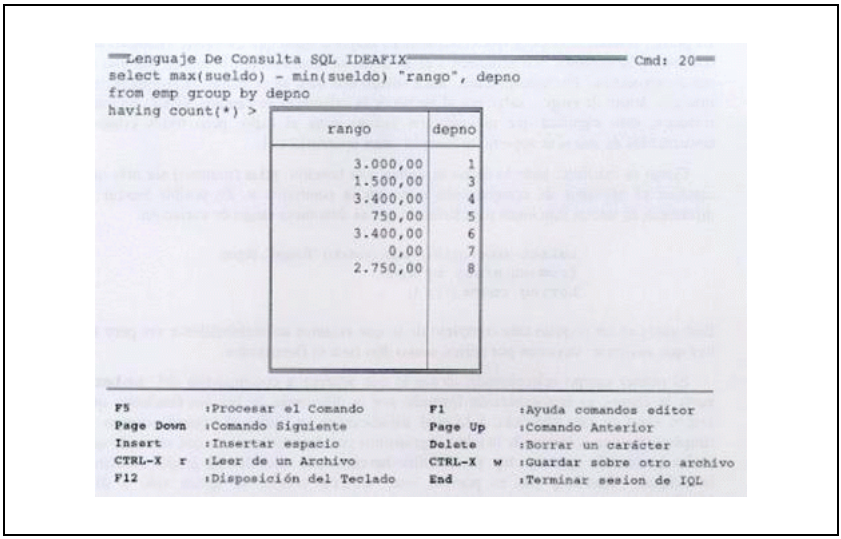


Figura 22.4

Es obvio que en los departamentos con un solo integrante, **max - min** arroja un resultado cero por ser ambas funciones iguales al único valor existente; no obstante, ello ocurrirá también con los departamentos donde los empleados ganen los mismo.

Es factible aplicar las funciones de máximo y mínimo a columnas de caracteres en cuyo caso se obtendrá como “menor” a aquel campo que comience con el carácter más bajo en la secuencia alfabético; a igualdad del primer carácter decide el segundo, y así sucesivamente. Esto significa que aplicadas a la Guía Telefónica, **min** (apellido) nos da el primer abonado de toda la lista y **max** (apellido) el último.

Estas funciones se pueden aplicar también sobre campos de tipo fecha y hora, y lógicamente

el resultado que arrojan es la máxima o mínima fecha u hora del grupo.

Concepto de Sumatoria: Función sum

Como de costumbre, cuando hablamos de la suma de los valores de una columna de la tabla, estamos usando el término en sentido algebraico; vale decir, teniendo en cuenta el signo de cada campo. Así, la suma de 3, -5 y 2 da por resultado cero. La función **sum** procede en esta forma para cada grupo que se defina en el query, y devuelve los valores correspondientes. Por ejemplo:

```
select depno, sum(sueldo), sum (comis)
from emp group by depno;
```

nos dará el importe total de sueldos y comisiones abonado por cada departamento:

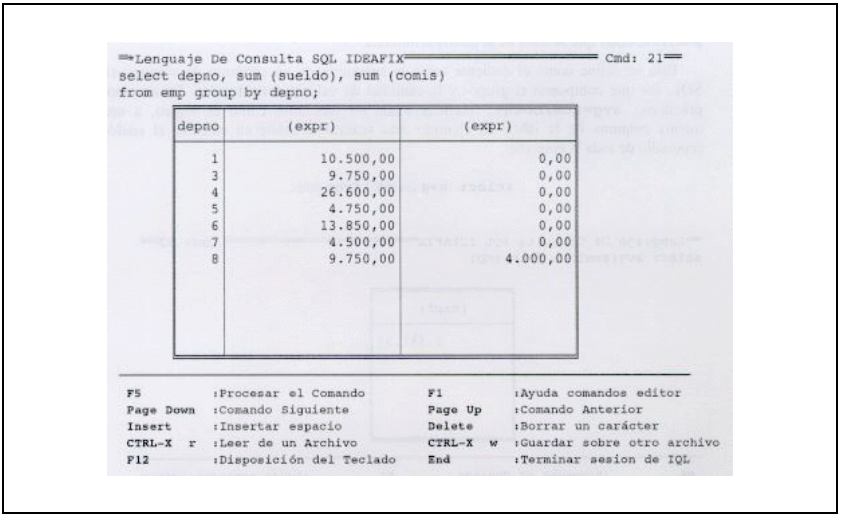


Figura 22.5

Suprimiendo la cláusula **group by**, toda la tabla se convierte en un único grupo, y tendremos los importes totales de sueldos y comisiones pagados por la empresa. ¿Esto es realmente así? No: el query produciría un mensaje de error debido a que se le está requiriendo en la cláusula **select** que muestre el departamento, y este valor ha dejado de pertenecer al grupo. Para que la cosa funcione correctamente, es preciso suprimir **depno** del **select** también.

El ejemplo de error introducido en el párrafo anterior debe hacemos tener presente que no es lícito mezclar en la selección columnas o expresiones correspondientes a niveles de agrupamiento distintos de aquellos que se han especificado. Esto resulta claro si pretendiéramos pedir el nombre o el número de legajo al tiempo que utilizamos una función

de grupo: el contenido de tales columnas es individual y propio de cada fila de la tabla, y entonces el sistema no tiene manera de establecer qué legajo debiera mostrar en correspondencia a un grupo de empleados que ostentan, por ejemplo, el mismo cargo.

Las restricciones que surgen de lo expuesto anteriormente pueden en buena medida superarse a través de un “sub-query”, concepto que será analizado más tarde en este capítulo.

Promedio: La Función avg

El nombre de esta función deriva de la palabra “average”, que significa promedio. En la terminología estadística se denomina escuetamente “media”, dándose por sobre entendido que se trata de la media aritmética.

Esta se define como el cociente entre la sumatoria de un conjunto de valores -en SQL, los que componen el grupo- y la cantidad de valores considerados. En términos prácticos, **avg=sum/count;** referidas todas las funciones como es lógico, a una misma columna de la tabla. El ejemplo más sencillo consiste en calcular el sueldo promedio de toda la empresa:

```
select avg (sueldo) from emp;
```

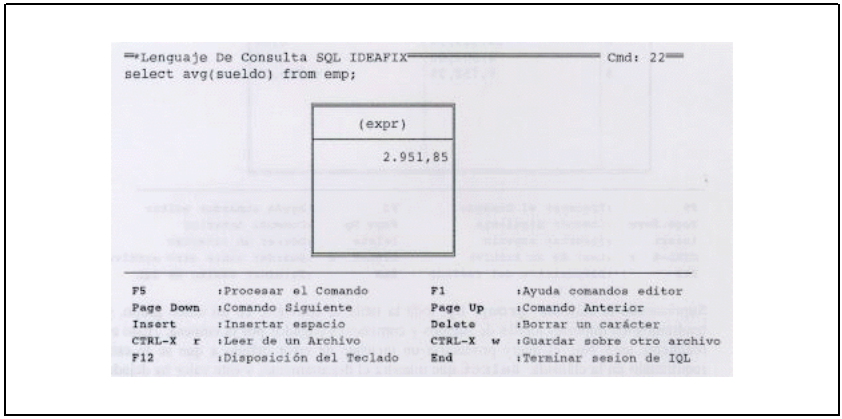


Figura 22.6

El ejemplo devuelve el valor 2.951,85. Dado que hay veintinueve personas registradas en la tabla **emp**, significa esto que el monto total de sueldos es 2.951,85 * 29? No, en realidad: solamente veintisiete empleados han intervenido en el cálculo pues hay dos filas con el campo vacío. De tal modo el total es 2.951,85 * 27 = 79.699,95

Una vez más, es importante tener en cuenta que los valores nulos no serían considerados, lo que en muchas ocasiones puede introducir una distorsión en los resultados, o más bien en la manera de interpretarlos.

Si queremos equiparar los casos nulos a ceros, recordemos que una función puede aplicarse a una expresión, la que en este caso conviene sea una expresión condicional:

```
select avg (sueldo) is null ? 0 : sueldo)
"Sueldo\n Promedio"
from emp;
```

La función **avg** se puede aplicar también sobre campos de tipo fecha u hora. En el primer caso el resultado es una fecha calculada como el promedio de todas las fechas del grupo. Si tenemos por ejemplo los valores:

```
12/01/8 01/07/89 14/03/90
```

nos dará como promedio 29/07/89.

En el caso de los campos con valores horarios la operación es similar.

Funciones Acumulativas

El concepto visto en las secciones anteriores de sumatorias, máximos y mínimos y promedio se aplicaba sobre grupos. Existen funciones que permiten realizar estas operaciones pero sobre las columnas de la tabla de salida. Dichas funciones se denominar: **runsum, runavg, runmin, runmax y runcount.**

Se pueden indicar en una consulta de la misma forma que cualquiera de las funciones vistas en el Capítulo 21 y el resultado que arrojarán será la sumatoria acumulada para cada fila (en el caso de la función **runsum**).

El Concepto de Subquery

Se denomina así un query que es utilizado por una cláusula de otro comando SQL. Generalmente, dicho comando principal es también un query, el cual se vale del comando subordinado -de allí el nombre de subquery- para establecer cierta premisa o determinar un juego de valores seleccionados que será luego objeto de la operación principal. En este apartado veremos únicamente los casos de subqueries usados en la cláusula **where** del query principal.

Para comprender el concepto de subquery es fundamental tener en cuenta que la operación **select** llevada a cabo sobre una tabla tiene como resultado otra tabla, la **grilla** en la terminología de **iql**, que está constituida por una, varias o todas las columnas de la tabla original, y otro tanto ocurre con las filas. Sabemos que **select *** toma todas las columnas, y que una condición tal como **where** nroleg > 0 en la tabla emp incluirá todas las filas. En realidad, puede ocurrir que la cláusula where no se cumpla nunca (v.gr. **where** nroleg **is null**), lo cual producirá una grilla vacía para el subquery y obviamente otro tanto para el query principal. Para ser precisos, entonces, la grilla del subquery puede contener todas, algunas, una o ninguna de las filas de la tabla principal.

Ejemplos de Subquery

La implementación actual del **iql** maneja subqueries que devuelve al query principal -o genéricamente, al de nivel superior inmediato que lo llamó- lo que matemáticamente se denomina un **escala**r; esto es, un valor único para cada fila procesada. En términos prácticos, esto significa que un subquery en **iql** solamente pasará a proceso invocaste valores correspondientes a una columna.

Comencemos por un ejemplo sencillo. Queremos saber quiénes tienen el mismo cargo que un cierto empleado; digamos Tenorio, cuyo número de legajo es 21. Se podrían hacer dos quieres: uno para averiguar el cargo y otro para seleccionar a quienes lo comparten. Es decir:

```
select cargo from emp where nroleg = 21;
```

que nos devolverá el valor 8, luego haríamos:

```
select nroleg, nombre
from emp where cargo = 8;
```

lo que nos daría el resultado final buscado.

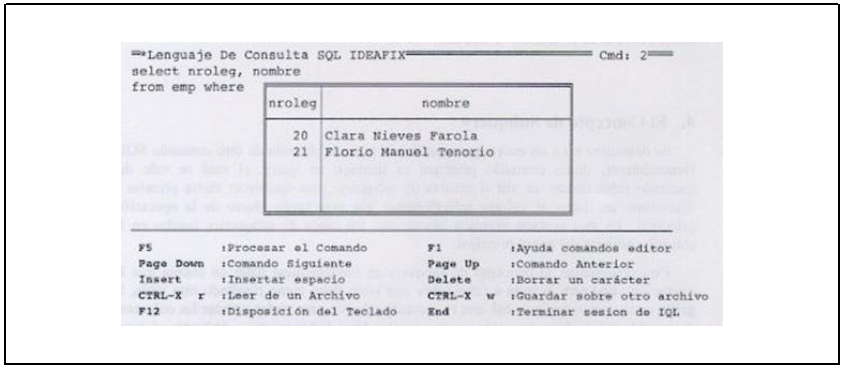


Figura 22.7

El uso del subquery permite obtener dicho resultado en forma directa, es decir en un solo paso, de la siguiente forma:

```
select nroleg, nombre,cargo
from emp
where cargo = (select cargo from emp where nroleg = 21);
```

La salida de este ejemplo es:

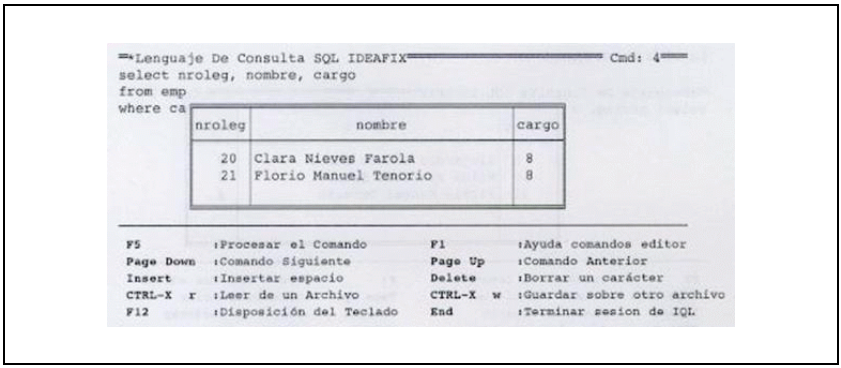


Figura 22.8

Adviértase que el subquery está encerrado entre paréntesis, y no es otra cosa que el primero de los pasos ejecutados según el método anterior. Podría haber sido escrito de corrido en un solo renglón de la pantalla, ya que en este caso sobra lugar para ello, pero se ha querido mostrar la técnica llamada “indentado”, ampliamente usada en los procesos estructurados y principalmente en todos los lenguajes de programación, que tienen por objeto hacer patente el **anidamento** (“nesting”) de procesos. Esto se logra Aquí encolumnado el **where** con su correspondiente **select**.

Tal como se ha hecho con el cargo, podríamos igualmente requerir aquellos empleados que trabajen en el mismo departamento que una cierta persona, o que tengan su mismo jefe. Invitamos al lector a proponerse y practicar ejercicios de este tipo. Puede ocurrir que algún caso no sea resoluble por el sistema: empleados cuya comisión sea mayor que la percibida por un cierto número de legajo! ¡correspondiente a alguien que no recibe comisiones! (Qué lata son estos null values ¿verdad?)

En tales circunstancias se emitirá una advertencia similar a la que se producirá si se requiera un número de legajo inexistente.

Los subqueries pueden ser múltiples, conectados por los operadores lógicos **and** y or. Podemos listar, por ejemplo, los empleados cuya comisión sea mayor o igual que la del empleado número 10, y que pertenezcan a un mismo departamento:

```
select * from emp
where depno = (select depno from emp where nroleg = 21)
and comis = (select comis from emp where nroleg = 21);
```

La salida de este ejemplo es:

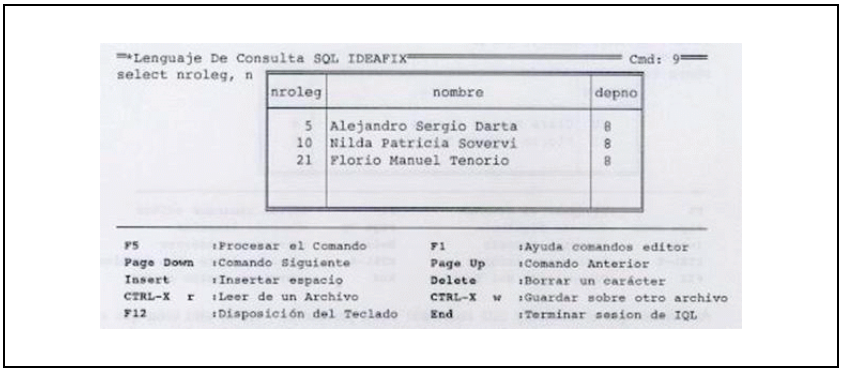


Figura 22.9

Por último, veamos el caso de un subquery de anidado dentro de otro, que involucre además el uso de más de una tabla. Si queremos saber qué empleados ganan más que el promedio de sueldos del departamento de desarrollo, y listar todos sus datos:

```
select * from emp
where sueldo > (select avg(sueldo) from emp
where depno = (select depno from depno where nombre =
'Desarrollo'));
```

Es importante notar dos cosas. Por un lado, el doble cierre de paréntesis al final, debido al anidamiento de subqueries. Por el otro, que la ejecución se realiza “de dentro hacia fuera”; esto es, primero el más interno de los subqueries averigua que el Depto. de Desarrollo es el número 4, luego el de nivel intermedio calcula el promedio de sueldos de dicho departamento, y por último el query principal lista los empleados cuyo sueldo supera dicho valor.

La salida de este query es:

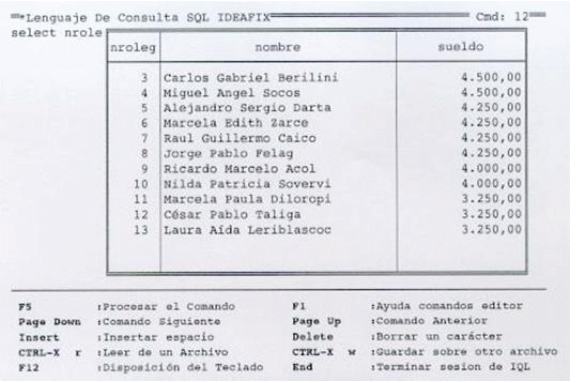

=>Lenguaje De Consulta SQL IDEAFIX Cmd: 12
select nrole
nroleg nombre sueldo
3 Carlos Gabriel Berilini 4.500,00
4 Miguel Angel Socos 4.500,00
5 Alejandro Sergio Darta 4.250,00
6 Marcela Edith Zarce 4.250,00
7 Raul Guillermo Caico 4.250,00
8 Jorge Pablo Felag 4.250,00
9 Ricardo Marcelo Acol 4.000,00
10 Nilda Patricia Sovervi 4.000,00
11 Marcela Paula Diloropi 3.250,00
12 César Pablo Taliga 3.250,00
13 Laura Aida Leriblascoc 3.250,00
F5 :Procesar el Comando
F1 :Ayuda comandos editor
Page Down :Comando Siguiente
Page Up :Comando Anterior
Insert :Insertar espacio
Delete :Borrar un carácter
CTRL-X r :Leer de un Archivo
CTRL-X w :Guardar sobre otro archivo
F12 :Disposición del Teclado
End :Terminar sesion de IQL

Figura 22.10

Capítulo 23

Sentencias de Control

En este capítulo se tratarán las sentencias de control y la forma de ejecutar consultas pre-programadas en modo batch.

Las sentencias de control abarcan una serie de aspectos varios, y el tema de la seguridad de los datos, es decir, quienes tienen acceso a la información contenida en las tablas.

Hasta aquí hemos supuesto que la consulta a nuestras bases de datos se hacía de modo interactivo. Esta modalidad responde generalmente a la característica de “ocasional” o no planeada; principalmente en cuanto se refiere al momento en que se realiza, y también a los detalles específicos que hacen al resultado: condicionamientos, qué cosas se toman o no en cuenta (v.gr. values nulos), ordenamiento de la información etc.

Veremos en este capítulo un mosaico de posibilidades que hacen a variados aspectos de los “queries”, o consultas. Por una parte, la contraposición entre los modos de proceso “batch/interactivo” y los puentes que el **iql** permite tender entre ellos.

Se verán también los aspectos de la **presentación** de los datos de salida.

La Sentencia exec

Esta sentencia permite ejecutar una consulta almacenada en un archivo sin necesidad de traer este sobre la pantalla. La forma de utilizarla es la siguiente:

```
exec archivo;
```

El nombre del archivo también puede ir encerrado entre comillas dobles. Al procesar esta sentencia se ejecutarán las instrucciones SQL almacenadas en el archivo.

La Sentencia close

Cuando se cambia de esquema corriente mediante un **use** o un **create**, ello no significa que el esquema anteriormente corriente deje de estar activo. Si se quiere suprimir un esquema de la

lista de **activos**, se usa una sentencia tal como:

```
close ALFA;
```

Pero si este esquema era precisamente corriente, ¿quién pasa a serlo ahora? Queda indefinido, el usuario debe indicarlo mediante un **use**.

La Sentencia define

Esta sentencia permite definir sinónimos para toda una cadena de caracteres. Es muy útil para acortar palabras muy utilizadas, o darle otro nombre que nos parezca más apropiado que el dado por el SQL. Su sintaxis es la siguiente:

```
define sinonimo as "cadena de caracteres";
```

Luego de ejecutar esta sentencia, toda vez que escribamos la palabra sinónimo será reemplazada por cadena de caracteres.

Veamos por ejemplo como podemos “castellanizar” el lenguaje SQL mediante la definición de sinónimos. Para ello ejecutamos las siguientes sentencias:

```
define listar as "select";
define de as "from";
define donde as "where";
define todo as "*";
```

Habiendo ejecutado estas sentencias, la consulta:

```
select * from emp where empno > 4;
```

puede escribirse:

```
listar todo de emp donde empno > 4;
```

Definición de Variables: set

El SQL de **IDEAFIX** permite definir al usuario variables, a las que puede dar un nombre y valor. Esto se realiza mediante el comando **set**. Existen algunas variables que están pre-definidas en el sistema. El comando **set** emitido sin otra especificación desplegará el estado de todas las variables con los valores que tengan asignados en ese momento. Para modificar uno de ellos, la sintaxis a utilizar es:

```
set variables = valor
```

mientras que:

```
unset variable
```

elimina la variable. Como se verá más adelante, existen algunas variables pre-definidas por

iql que controlan aspectos como el presentación impresa de la consultas.

El Archivo .iqlrc

En muchos casos las sentencias **define** y **set** nos permiten definir nuestro propio modo o perfil de trabajar con SQL. Pero es bastante molesto tener que ingresar dichas sentencias cada vez que iniciamos una sesión con **iql**. para evitar esta molestia existe la posibilidad de crear un archivo llamado ".iqlrc( )" que en principio puede contener cualquier tipo de sentencias válidas. Estas sentencias serán ejecutadas al comienzo de la sesión de **iql** en forma automática.

Usualmente se lo utiliza para definir sinónimos y variables de modo que existan automáticamente al comenzar la sesión. También se puede agregar una sentencia use si es que trabajamos siempre con un esquema.

El archivo puede estar ubicado en:

- UNIX: en directorio de login del usuario, es decir el indicado por la variable de ambiente HOME. En este directorio es donde se busca en primera instancia. Si nos e lo encuentra allí se explora el directorio corriente en el que se está posicionado al momento de invocar a **iql**.
- MS-DOS: en el directorio raíz de la unidad corriente que es donde se busca en primera instancia, o bien en el directorio corriente en el que se está posicionado al momento de invocar a **iql**.

El Proceso Batch

Es preciso hacer un poco de historia (de la computación y del idioma, que resulta ser siempre el inglés). Seguramente pocos saben que "batch" es el sustantivo derivada del verbo "to bake" (hornear, en particular tortas, pan y alimentos análogos; de ahí "baker", "bakery": panadero, panadería). Según el mismo mecanismo de declinación se tienen "to make" a "match" (otro término usado en computación) y "to wake" a"watch".

Por consiguiente, en sentido literal "batch" significa "horneada", y por extensión, conjunto o lote de cosas que se producen juntos. De allí que se haya traducido "batch processing" como "proceso por lotes". No obstante, el término inglés ya ha adquirido carta de ciudadanía de hecho, al igual que bit, byte, hardware, software y muchos otros, por lo que en lo sucesivo suprimiremos las comillas.

En los comienzos de la computación -la época heroica de la tarjeta perforada-, no existía otro proceso que el batch; más aún, cada programa se ejecutaba separadamente y luego la máquina se detenía, salvo que se alimentará un **lote** de programas apropiadamente diseñados para que cada uno de ellos transfiera el control de ejecución al siguiente, salvo el último, que simplemente dejaba al sistema en "Stop". Alguien tuvo entonces la genialidad de concebir un

programa cuya función fuera la de **mandar ejecutar** otros programas -los del usuario- y a este "superprograma" se lo denominó, Según los casos y las épocas, monitor, supervisor o sistema operativo (esta última designación ya no alude a un programa sino a todo un conjunto de ellos). Esto está ligado íntimamente al concepto de "non-ending program" o "programa infinito", el cual una vez ejecutada una tarea permanece a la espera de que se le pida la siguiente.

El primero en tener interactivo con el sistema fue el operador, y a través de terminales que se parecían demasiado a una máquina de escribir. Luego se puso esa misma facilidad a disposición de un pequeño grupo de usuarios, y finalmente con la aparición y difusión de las pantallas CRT ("Cathod Ray Tubes") el modo interactivo -que así también se lo llama- se popularizó en tal medida que hoy en día es preciso explicar a los más jóvenes que hubo épocas en las que tal procesamiento no existía y todo era batchÉ tal como lo ha procurado hacer esta apretada síntesis.

Ejecución Batch del IQL (Consultas pre-programadas)

Ante el prompt del Shell, que usualmente es el símbolo "pesos"($), podemos requerir un proceso batch indicando a continuación del comando que invoca al iql el nombre de un archivo:

```
$ iql comis09
```

Este file, que presumiblemente contendrá una consulta relativa a las comisiones del mes de septiembre, debe llamarse "comisn09.sql", por requerimiento del sistema, pero tanto puede invocarse especificando su nombre completo o bien prescindiendo del sufijo o extensión ".sql", manera esta usada en el ejemplo.

El contenido del archivo en cuestión deberá ser una serie de sentencias de SQL, que podrán especificar la ejecución de uno o varios quieres, y también de otros comandos relacionados.

Otra forma de ejecutar un proceso de iql en modo batch es por medio de la opción "-c". Debemos acotar aquí que el Shell, o procesador de comandos del UNIX, reconoce para cada programa cuya ejecución se le requiere, la especificación de opciones, las cuales se codifican inmediatamente a continuación del nombre del programa, precediendo cada especificación por un guión y separadas por blancos. En este caso particular, "c" significa "character string input"; es decir, que las sentencias de **sql** van a ser especificadas a continuación. Así:

```
$ iql -c "use personal; select * from emp where depno = 3;
```

listará todos los datos, tomados de la tabla **emp** del esquema **personal**, de los empleados pertenecientes al departamento **3.**

A esta altura estamos en condiciones de plantearnos la pregunta: ¿de dónde lee su input el iql? Resulta claro que mando no se especifica la entrada en forma explícita:

```
$ iql <Enter>
```

supone que le va a ser proporcionada desde el teclado, por ser la entrada estándar del sistema. Si en cambio se le indica el nombre de un archivo:

```
$ iql filename <Enter>
```

va buscar su input allí. En el primer caso queda establecido naturalmente que el modo es interactivo, y en el segundo es batch. La tercera posibilidad que hemos visto, que es usar la opción “-c”, permite ejecutar en modo batch pese a que la entrada se efectúa desde el teclado.

Si olvidásemos colocar el guión identifica torio de la opción como tal, el **iql** intentaría ejecutar el contenido de un archivo llamado **c.sql**, probablemente inexistente, aparte de considerar extemporánea la presencia de la cadena de caracteres.

Queda un último aspecto: la salida. Nuevamente, en ausencia de indicación explícita se supone sea la salida normal, que habitualmente está asignada a la pantalla. Esto puede modificarse mediante la cláusula **output to**, que analizaremos en el partado "Formatos de salida".

Consultas Parametrizadas

Un comando del **iql** puede recibir parámetros para su ejecución, lo cual flexibiliza las posibilidades de las consultas pre-programadas. Estos parámetros se referencia en la especificación de la consulta con el signo $ (pesos) seguido por un número de referencia que indicará el orden dentro de la lista ingresada.

Los parámetros se indican en la línea de ejecución a continuación del nombre del archivo que contiene la consulta, como se muestra a continuación:

```
$ iql -b file 1 44
```

En el cuerpo del comando hará referencia al número **1** mediante la especificación $1, y al **44** con $2.

En caso de que el parámetro a informar sea una cadena de caracteres, se debe especificar desde la línea de comandos de la siguiente manera:

```
$ iql -b nombre "*Sergio*"
```

En este caso, como ya sabemos, debe existir un archivo nombre.iql. Dicho archivo contiene las siguientes especificaciones:

```
use personal;
select nroleg, nombre from emp where nombre like %"1";
```

Este comando de iql seleccionará el número de legajo y el nombre de aquellos empleados cuyo nombre contenga la palabra “Sergio”; como se ve, en la especificación se hace referencia a una cadena encerrando entre comillas dobles el número de parámetro; y especificando de la misma forma la cadena desde la línea de comandos.

La salida de esta especificación es:

Interfaz con el Comando execform

El ingreso de parámetro a través de la línea de ejecución puede no ser apropiado para usuarios finales que utilizan las aplicaciones a través de menúes. En estos casos se desea que el ingreso de los parámetros de una consulta se realice mediante algún procesos interactivo, como por ejemplo la carga de un formulario.

Para establecer este tipo de interfaz amigable para el usuario de **iql** a través de las facilidades de formularios, se puede recurrir al utilitario **execform**. Este comando permite utilizar un formulario **IDEAFIX** para valores que se ingresen en los campos se pararán como parámetros en el momento del llamado a dicho proceso.

Por ejemplo, si se desea tener una consulta para listar una tabla con los nombres de empleado para un rango dado, se puede definir un formulario de la siguiente manera:

```
Número de empleado desde :_________.
Número de empleado hasta :_________.
%form
use perosnal;
window label "Consulta de empleados",
border standard;
%fields
desde:defualt 1, on help in emp: (nombre);
hasta:default 9999;
```

Se debe compilar el formulario con el utilitario **fgen**. Si el formulario se denominó consemp.fm, entonces se hará:

```
$ fgen consemp
```

Con esto ya se está en condiciones de correr el programa. Los dos valores que se cargaron en los campos del formulario, se pararán como argumentos al programa que se le indique a execform:

```
$ execform -w consemp "iql -b consemp"
```

Dentro de **execform**, cuando se oprima PROCESAR se invocará a la consulta con los valores cargados en los campos del formulario como parámetros. Por ejemplo si el usuario ingresó 3 en el primer campo y 8 en el segundo, se ejecutará:

```
iql -b consemp 3 8
```

Dentro de la consulta, los parámetros se referencia mediante el signo “$”(pesos) y un número correlativo. Por lo tanto, los argumentos serán:

```
$1 Primer argumento
$2 Segundo argumento
$3 Tercer argumento
É
```

Entonces, el archivo consemp.sql que contiene la consulta definida anteriormente se debe escribir de la siguiente manera:

```
use personal;
select nombre from emp
where nroleg between $1 and $2
output to terminal;
```

Formatos de Salida

Existen varias formas de mostrar los resultados de una consulta. En ausencia de indicación explícita -es decir, por defecto o "default"-, la salida es directamente el contenido de la grilla volcado a un dispositivo que depende de cuál sea la modalidad de ejecución. Asimismo es posible direccional el resultado en la forma que el usuario considere conveniente, Según ya se hecho en el último ejemplo y conforme veremos a continuación.

La Cláusula output to

En modo interactivo, la salida normal del sql es la pantalla, mientras que en modo batch es lo que el sistema operativo (es decir UNIX) tenga definido como salida estándar. Ella es usualmente la terminal.

Cuando se desea alterar el destino de la consulta, podemos recurrir a la cláusula **output to** destino, que debe ser la última del query, y puede tener los siguientes operados:

terminal: Permite dirigir la salida de un proceso batch a la pantalla para ver el resultado mediante una grilla.

printer: Direccional la salida a la impresora (ver variable de ambiente printer).

"filaname": Graba la salida en el archivo "filename".

report "reptname": Esta última opción permite lo que se denomina "formatear" la salida (asignarle un formato), recurriendo a las facilidades que brinda el lenguaje de diseño de formularios(RDL) de **IDEAFIX.**

Variables de Impresión

Cuando se utiliza **output to printer**, o la impresora resulta ser el dispositivo estándar de salida, puede ser necesario tener en cuenta -y eventualmente modificar- ciertos parámetros que hacen a la manera en que se imprimirá el listado. Estos parámetros están representados por variables predefinidas en IQL. Los nombres de variables **iql** de dichos parámetros, que hacen a las características de impresión, junto con su significado y valores por efecto, se brindan en el cuadro siguiente.

Variable	“default”	Descripción
topmarg	0	Margen superior libre en cada página, entre el primer renglón posible y el encabezamiento.
botmarg	0	Margen inferior libre entre el pie de página y la última línea de impresión.
heading	$empresa\t\t#D	Tipo “string”. Define el contenido del encabezamiento de página. Puede tener hasta 3 partes.
footing	\t-#P-	String de caracteres que se imprime al pie de página, con hasta tres secciones como el anterior.
flength	66	Longitud de formulario, expresada en cantidad total de líneas impresas.
fwidth	80	Ancho del formulario, expresado en columnas efectivas de impresión.

Las dos primeras variables, al igual que las dos últimas, son simplemente valores numéricos y no presentan problema. Si queremos modificar -por ejemplo- la longitud del formulario, acortándolo un poco, bastará emitir el comando:

```
set flength=60
```

Las variables de encabezado (heading) y pie de página (footing), en cambio, requieren una explicación. En primer lugar digamos que cada uno de ellos puede constar de varias líneas separadas por “\n”. Cada línea consta de tres secciones que se separan mediante el carácter “TAB” (Tabulado) que se suele simbolizar con una barra inversa (“\”) llamada “backslash”, seguida de la letra “t”- La primera sección va ajustada a la izquierda; la segunda centrada, y la última se ajusta a la derecha.

En el caso del encabezado predefinido, vemos que la primera parte es una **variable de ambiente**, por ir precedida por el signo “"dólar”, que se supone definida como el nombre de la empresa. Es decir, lo que se va a imprimir no es la palabra “empresa”, sino el **contenido** de la variable así llamada. Si esta no ha sido definida (o “ seteada”, Según un barbarismo de amplia difusión), quiere decir que equivale a un valor nulo, luego nada se imprimirá en el lado izquierdo del encabezado.

La presencia de dos tabulados consecutivos indica que el defecto para la sección central es efectivamente un “null value”, y finalmente encontramos en la parte final otra expresión simbólica, esta vez precedida por el símbolo “#” (numeral). El sistema reconoce tres valores de este tipo, que son:

- #P Número de Página (Page)
- #D Fecha del sistema (Date)
- #T Hora del sistema (Time)

Vemos entonces que el pie de página consiste en su número encerrado entre guiones, ubicado en el centro. Esto se debe a que la especificación comienzan con un tabulado, lo convierte a la parte de la izquierda en valor nulo. Como es obvio, al no haber un segundo TAB la sección derecha también es nula.

Salida con Reportes

Cuando se necesita disponer de una salida impresa con ciertos requisitos de presentación y formato, se utiliza la cláusula:

```
output to report reptname
```

donde “reptname” (report name) representa el nombre asignado a un archivo generado por el utilitario **rgen**.

La idea detrás de Esto es usar la grilla creada por una consulta como fuente de información, pero teniendo la posibilidad de formatearla con las facilidades que brinda el lenguaje de reportes de **IDEAFIX** (RDL).

El generador de reportes tiene sus propias normas y convenciones de lenguaje que no vamos a analizar aquí (referirse para ello al manual del programador de **IDEAFIX**, Parte III - Sección 2 Capítulo 17); pero sí daremos una somera idea del procedimiento para utilizarlo.

Los pasos a seguir son:

- Crear las especificaciones para el **rgen** con un editor. El archivo deber‡ tener el sufijo .rp para ser reconocido como especificaciones de reporte.
- Compilar estas especificaciones con el utilitario **rgen** (Report generation). La salida compilada quedará en otro archivo cuya identificación será establecida por el sistema en lo que hace a los sufijos. En otras palabras, esto es transparente para el usuario.

Extensión al IQL para Salida Gráfica

Para obtener Gráfica se deben definir las series numéricos cuya salida se desea volcar en dicho formato. Es factible definir hasta seis (6) series con un máximo de veinte (20) valores en cada una. Las series provienen de las columnas de una tabla; recordemos que un conjunto de columnas o campos es lo que se individualiza en primer término en una sentencia select.

Cada serie puede ser rotulada con un nombre así como también es posible asociar nombres a

los distintos valores. A través de las variables del **iql** se pueden definir títulos, subtítulos y rótulos para los ejes.

Sintaxis

Toda sentencia SELECT se puede utilizar con la extensión:

```
...OUTPUT TO GRAPH [tipo] [INVERTED] [TO PRINTER
[WIDTH[,LENGTH]]]
```

Seguidamente se indican los distintos valores que puede tomar cada uno de los parámetros.

tipo - Indica el formato específico del gráfico según las alternativas:

BAR Produce un gráfico de barras verticales, en el cual cada serie se identifica mediante un relleno o coloración distinta. En este caso se aceptan entre una y seis series a representar. Cuando existe más de una, aparecen para cada rótulo (v.gr. Enero, Febrero, etc.) los grupos de barras adyacentes.

STACKED Es similar al anterior, pero exige al menos dos series; el valor de cada una se apila verticalmente sobre el anterior, de modo que la altura total de la barra corresponde a la sumatoria de los valores de las series.

PIE Acepta una sola serie valores, los cuales indica como partes porcentuales de un total. La representación no es circular sino rectangular. Además de la expresión Gráfica, indica los porcentajes en forma numérica.

XY admite solamente dos series de valores, la primera de ellas tomada como conjunto de abscisas (coordenada X0, y la segunda como conjunto de ordenadas (coordenada Y), representando los puntos correspondientes en un gráfico cartesiano.

width = número La cantidad de columnas de impresión a utilizar. Nótese que este parámetro sólo se utiliza en caso de haber especificado output to printer.

length = número La cantidad de renglones por página. Vale la misma observación anterior referente a la salida impresa.

Variables del IQL involucradas

Con motivo de la implementación de esta nueva facilidad, se han incorporado ciertas variables al sistema, las cuales se distinguen por comenzar con la letra “g”, excepto aquéllas que se refieren a la impresión de formularios, que empiezan con “f”.

gtitle : Título del Gráfico (string)

gstitle : Sub-Título del Gráfico (string)

glabelx : Rótulo de eje X (string)

glabely : Rótulo del eje Y (string)

flength : Largo del formulario de impresión (número de líneas)

fwidth : Ancho del formulario de impresión (número de columnas)

Tipo de Gráficos

Se ilustran dos formatos del mismo caso: utilización de los sistemas de runtime e iql en una instalación, durante los tres primeros meses del año.

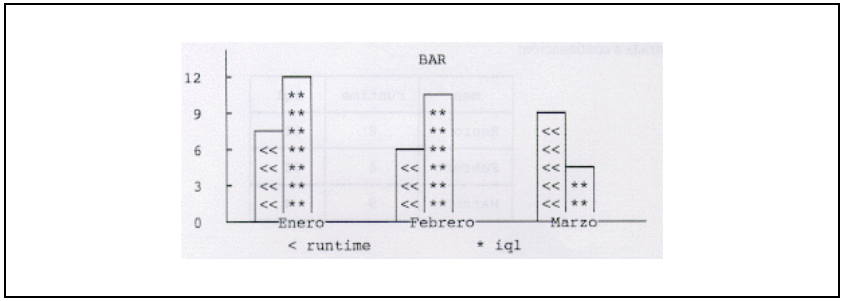


Figura 23.1

El modo “BAR” permite hacer una comparación de valores de las distintas series (en este caso son solamente dos).

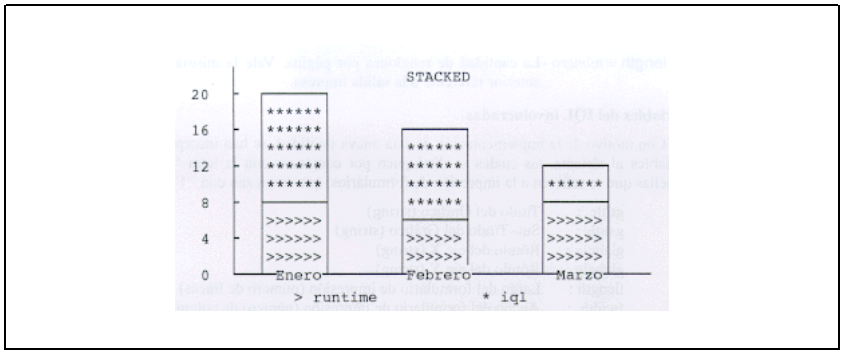


Figura 23.2

Superponiendo los valores con “STACKED” tenemos una idea del valor total de las series para cada caso -que antes no era aparente- manteniendo en alguna medida el carácter comparativo entre series. Nótese la diferencia de escalas elegidas por el sistema.

Los juegos de valores resultantes del select corresponden a una grilla tal como la ilustrada a

continuación:

mes	runtime	iql
Enero	8	12
Febrero	6	10
Marzo	9	4

Definición de las Series a Partir de un Select

Cuando no se utiliza la opción INVERTED la definición de series se efectúa de la siguiente forma:

- Las Series se corresponden con columnas de tipo numérico del resultado del select.
- La primera columna no numérica asigna el nombre a cada valor de las series. El nombre de cada Serie es el nombre de la columna que la define.

Cuando se utiliza la opción INVERTED:

- Las Series corresponden a filas del resultado del select, del cual se consideran exclusivamente los valores numéricos.
- Los nombres de columna definen los nombres de los juegos de valores, mientras que el primer valor con tipo char de cada fila le da nombre a esa serie.

Impresión

Es posible imprimir gráficos que ocupen varios formularios. Para ello basta con definir las dimensiones totales que se quiere que tenga el gráfico, el cual será dividido en tantos formularios como sea necesario para imprimirlo, de forma que uniéndolos se obtenga el gráfico con las dimensiones solicitadas.

Notas y Comentarios

- Si no se indica el tipo de gráfico, se asume BAR.

- Si no se especifica el tamaño del gráfico para la impresión, se supone que es el de un formulario.

- Si el espacio no resultara suficiente para trazar un determinado gráfico, se genera un mensaje de error.

Notas de Programación

Los archivos modificados y el motivo de la modificación son los siguientes:

graf.c Rutinas de graficación

graf.h Headers, etc.

rpoutput.c Caso GRAPH

grinter.c Contiene funciones de manejo de Output que utilizan gráficos. Si bien tales funciones debieran figurar en **rpoutput.c**, se dejaron en otro archivo para evitar el uso de funciones Gráficas y por lo tanto de la librería matemática.

dsquery.c Caso RP_IO_GRAPH para creación de gráficos.

iql.c Variables del iql.

iql.h Idem anterior.

iqlconf.c Tokens para el parser.

iqlexpr.c Evaluación entre distintos tipos de decimales.

iqlparse.y Parser de la opción to graph...

iqlsel.c Modificación por REMOTE y manejo de errores generados por un gráfico.

rp.h Constantes y funciones.

rpdefs.h Cambios en estructura output.

makefile Makefile en uso.

iql Ejecutable sin control de activación (ATENCION!!)

resp Proceso de backup y traslado.

resp.sh Idem en modo shell.

output.doc Documentación de las funciones incorporadas para manejar un OUTPUT a GRAFICO.

iql.msg Mensajes de error de los gráficos.

Datos Compartidos y Seguridad

Con el transcurso del tiempo, las empresas usuarias de equipos de procesamiento de datos han ido tomando conciencia de la imperiosa necesidad de proteger su información. La obtención ilícita de datos reservados, que es lo llamado comúnmente “espionaje industrial”, ya no pasa por los microfilms, los biblioratos o los legajos archivados en gavetas metálicas. Es más práctico y más “limpio” robar la información de las bases de datos del sistema de computación.

En consecuencia, los sistemas han ido dotados de medidas de seguridad para evitar este tipo de actividades. Ello implica que no cualquiera puede tener acceso a una tabla; puede ocurrir también que, estando autorizado a leerla -por ejemplo, haciendo un query- no pueda en cambio introducirle modificaciones. El objeto de este apartado será entonces analizar quién y cómo puede tener acceso a los esquemas y tablas, y con qué alcances.

Concesión de Privilegios

Denominamos privilegios a las autorizaciones que recibe un usuario para efectuar determinadas actividades sobre una tabla o esquema.

Las sentencias GRANT y REVOKE permiten manejar los privilegios de acceso de los distintos usuarios a la Base de Datos. El concepto de “usuario” coincide con el implementado por el Sistema operativo.

Existen dos tipos de usuario respecto de un esquema: el dueño, y los demás. El dueño del esquema tiene todos los privilegios sobre el mismo, y la capacidad de decidir qué privilegios tendrán los restantes usuarios. El dueño de un esquema es aquel que lo crea o quien ha sido designado para serlo (mediante la sentencia CHOWN), y todos los archivos sobre el sistema operativo creados con relación al esquema (el directorio del mismo, tablas, índices, etc) pertenecerán a él.

CHOWN

Este comando permite cambiar el dueño de un esquema existente (**ch**ange **own**er). La sintaxis es análoga a la del comando chown del Sistema Operativo UNIX:

```
CHOWN usuario lista_esquemas
```

donde usuario es el nombre de algún usuario del sistema y lista_esquema es la lista de esquemas, separados por comas, a los cuales es les alterará el dueño. Por ejemplo:

```
chown pedro esq1, esq2,É, esqn
```

GRANT

La sentencia GRANT permite al dueño de un esquema definir privilegios de acceso a uno o varios esquemas y/o a una o varias tablas de un esquema. Por lo tanto este comando puede especificarse de dos formas distintas:

```
GRANT privilegio ON SCHEMA lista_esquemas
TO lista_usuarios
```

Mediante esta alternativa se definen los privilegios de acceso del (los) usuario(s) especificado(s) en lsita_usuarios, al esquema o esquemas indicados en lista_esquemas.

Los privilegios pueden ser:

Permiso	Autoriza a É
USE	Use
TEMP	select É order by É
MANIP	insert - update – delete
ALTER	Modificar estructura del esquema. [create] schema [create] table, etc.

La lista de usuarios puede especificarse Según las siguientes alternativas:

PUBLIC	Todos los usuarios gozan de los privilegios asignados
GROUP nombre_grupo	El grupo de usuarios denominado nombre_grupo goza exclusivamente de los privilegios asignados.
nombre_usuario	El usuario nombre_usuario es el único que goza de los privilegios indicados.

Como se ha mencionado, también se pueden definir los privilegios de acceso de un usuario o de un grupo de usuarios respecto de las tablas pertenecientes a un esquema. Sólo el dueño puede definir todos los privilegios de otros usuarios respecto del esquema en sí. El resto de los usuarios podrá asignar permisos en la medida que estén autorizados a hacerlo. Su sintaxis es:

```
GRANT privilegio-tab ON nombre-tabla TO
{PUBLIC | lista-usuarios}
[WITH GRANT OPTION]
```

Explicación de la Sintaxis

privilegio-tab: Es uno o más de los siguientes permisos (si se especifica más de uno, se debe separar con comas):

INSERT: Agregar nuevas filas (registros) a la tabla (se hereda)

ADD: Idéntico al anterior.

DELETE: Borrar filas de la tabla (se hereda).

UPDATE: Modificar datos de filas existentes (se hereda).

SELECT: Leer datos de filas existentes.

ALL: [PRIVILEGES] - Es sinónimo de todos los anteriores. Puede indicarse simplemente

como ALL, o bien como ALL PRIVILEGES.

nombres-tab: Es el nombre de la tabla para la cual se están otorgando privilegios de acceso. debe pertenecer al esquema corriente; o bien a cualquier esquema activo si se indica "esquema.tabla". la especificación "esquema.*" se refiere a todas las tablas del esquema. Cuando se indica más de un nombre se deben separar con comas.

PUBLIC: Se utiliza cuando los permisos se entregan a todos los usuarios del sistema.

lista-usuarios: Es una lista de nombres de usuarios del sistema separados por comas, a los cuales se les entregan los permisos indicados.

GRANT OPTION: Indica que cualquiera de los usuarios que haya recibido el permiso tendrá la capacidad de pasárselo a otros usuarios. Si así lo desean, pueden a su vez entregar el permiso con GRANT OPTION, para que el receptor pueda igualmente volver a transmitirlo.

Notas

- La sentencia GRANT sobre tablas sólo puede ser ejecutada por el dueño del esquema, o por un usuario que ha recibido un permiso con GRANT OPTION. En este último caso sólo puede entregar los permisos que le fueron dados con dicha opción.

Los puede entregar otro a su vez con el GRANT OPTION.

- Entregar permisos que un usuario ya posee, no tiene ningún efecto.
- La sentencia GRANT no es registrada por el LOG de transacciones.
- Para realizar una modificación de permisos (sentencia GRANT y REVOKE), se intenta bloquear el esquema. Esto es posible sólo si ningún usuario lo está utilizando.
- Realizar un GRANT para el dueño de un esquema o el super-usuario no tiene ningún efecto. El super-usuario posee por naturaleza todos los permisos sobre todas las tablas, o llegado el caso la posibilidad de forzar la concesión de un permiso cualquiera para sí o para otro usuario.

Ejemplos

```
GRANT ALTER ON SHCEMA personal TO pedro
```

El usuario pedro podrá acceder, modificar y eliminar datos del esquema, e incluso cambiar su estructura.

```
GRANT ALL ON emp TO gloria;
```

gloria tiene todas las autorizaciones posibles sobre la tabla emp, pero no puede transferirlas a otros.

```
GRANT SELECT ON emp TO pedro WITH GRANT OPTION;
```

Pedro puede consultar todos los campos de emp, y permitir a otros hacerlo.

Visualizando Permisos

En los primeros capítulos se mencionó y ejemplificó la sentencia **show**. Estamos en condiciones de entender la información completa proporcionada por esta sentencia, ya que comprendemos el significado del término dueño de un esquema y los distintos privilegios de acceso.

En el primer caso la sentencia show nos mostraba la lista de esquema activos, de la siguiente forma:

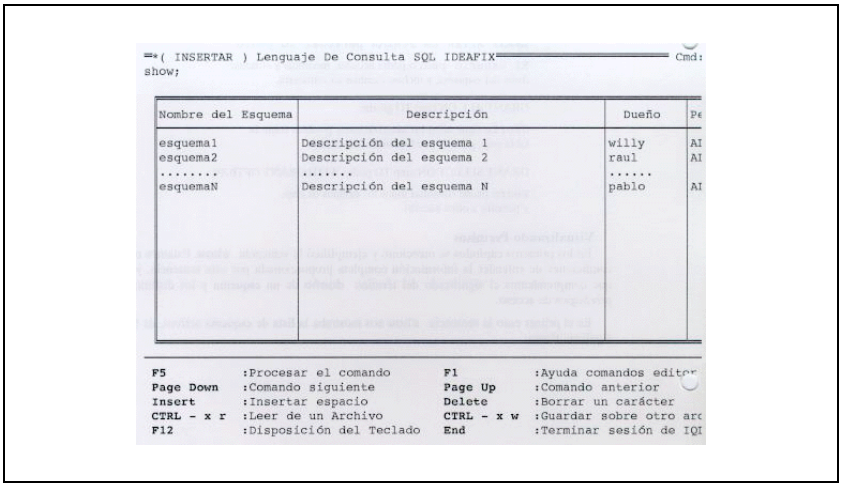


Figura 23.3

La tercer columna nos indica quién es el actual dueño del esquema y la cuarta los permisos que tiene el usuario que ejecuta el comando sobre el esquema.

El segundo caso de la sentencia show nos permitía consultar las tablas de un determinado esquema:

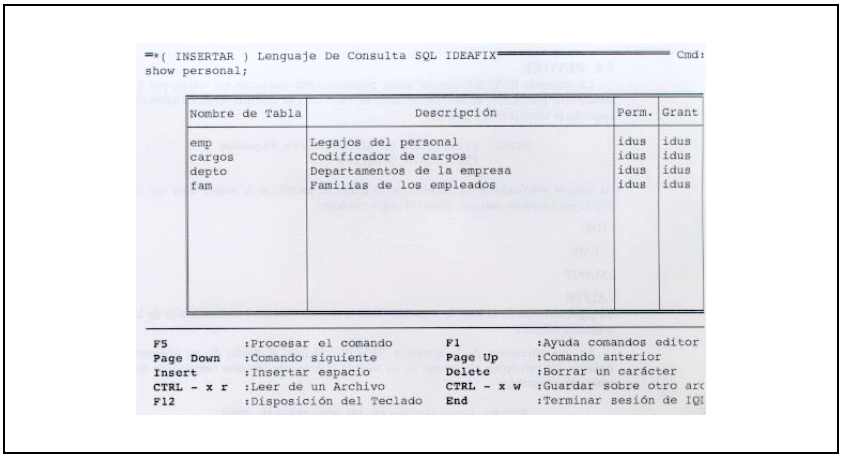


Figura 23.4

La tercera columna indica los permisos que tiene el usuario sobre las tablas y la cuarta los que puede asignar a otros usuarios. La simbología utilizada se interpreta de la siguiente forma:

		Posiciones	
1	2	3	4
i (insert)	d (delete)	u (update)	s (select)

Veamos cómo aparecen en pantalla distintas alternativas de permisos:

Permiso	Descripción
i ---	Permiso de insert
id --	Permiso de insert y delete
idu -	Permiso de insert, delete y update.
idus	Permiso de insert, delete, update y select.

Cualquier combinación de las distintas posiciones es válida. Se utiliza el mismo criterio para la columna titulada grant (permisos que el usuario puede asignar a otros).

REVOKE

La sentencia REVOKE permite quitar permisos sobre esquemas y/o tablas, por lo tanto puede presentarse de dos formas distintas. En el caso de eliminar permisos sobre un esquema la

sintaxis es la siguiente:

```
REVOKE privilegio ON SCHEMA lista_esquemas
FROM lista_usuarios
```

Al quitarle privilegios a un usuario o a un grupo de usuario se le asigna a su vez el privilegio inmediato anterior, siendo el orden creciente:

-USE

-TEMP

-MANIP

-ALTER

La especificación de la lista de esquemas y de la lista usuarios es idéntica a la de la sentencia GRANT.

La otra alternativa de la sentencia REVOKE permite al dueño de un esquema cancelar los privilegios de acceso de un usuario o grupo de usuarios respecto de las tablas. Su sintaxis es:

```
REVOKE privilegio-tab ON nombres-tab FROM
{PUBLIC | lista-usuarios}
```

Explicación de la Sintaxis

privilegio-tab: Es uno o más de los permisos detallados en el apartado anterior ("GRANT"); como antes, si se especifican varios privilegios deben ir separados por comas.

nombre-tab: Es el nombre de la tabla para la cual se están eliminando privilegios de acceso. Debe pertenecer al esquema corriente si sólo se indica su nombre. Cuando integre cualquier otro esquema activo, se indicará “esquema.tabla”.

PUBLIC: Se utiliza cuando el /los permisos se suprimen para todos los usuarios del sistema.

lista-usuarios: Es una lista de nombres de usuarios del sistema separados por comas, a los cuales se les quitan los privilegios indicados.

Notas

- La sentencia REVOKE sólo puede ser aplicada por el dueño del esquema.
- El dueño de un esquema y el super-usuario no pueden quitarse permisos a sí mismos. La sentencia REVOKE en ese caso no realizará cambio alguno.
- Intentar realizar REVOKE de permisos que un usuario no posee resultará en un error de ejecución de la sentencia.
- La sentencia REVOKE no es registrada por el LOG de transacciones.

- Para realizar una modificación de permisos, se intenta bloquear el esquema. Esto sólo es posible si ningún usuario lo está utilizando.

Ejemplos

```
//
// Entregar a Gloria permiso para manejar
// la tabla emp, salvo para selección y
// actualización de registros.
//
GRANT ALL ON emp TO gloria;
REVOKE SELECT, UPDATE ON EMP TO gloria;
// Pedro puede consultar todos los campos,
// y asimismo permitir a otros hacerlo.
GRANT SELECT ON emp TO pedro WITH GRANT OPTION;
// Ahora Pedro puede continuar consultando, pero ya
// no puede otorgar el privilegio a más usuarios.
REVOKE SELECT ON emp FROM pedro;
GRANT SELECT ON emp TO pedro;
```

Nótese que la opción de GRANT no es revocable por sí misma: es necesario revocar el privilegio concedido con ella, y luego volver a otorgarlo **sin GRANT OPTION.**

Otras Sentencias

Veremos ahora cuatro sentencias que son utilizadas para evitar salir del **iql** cuando necesitamos: cambiar de directorio; ejecutar comandos del Sistema Operativo, o correr procesos usuarios del Window Manager.

Por ejemplo, si necesitamos conocer el contenido del directorio actual podemos usar:

```
shell "ls | more";
```

En este caso la sentencia **shell** blanquea la pantalla, ejecuta el comando y vuelve inmediatamente al **iql**.

Es posible ejecutar este mismo comando pero volcando su salida sobre una ventana y esperando el ingreso de una tecla para retorna al lenguaje de consulta. Para ello podemos emplear la sentencia **pipe** de la siguiente manera:

```
pipe "ls | more";
```

También podemos cambiar el directorio corriente. La sentencia para esta operación es **CD**, y su sintaxis es la siguiente:

```
cd/user3/stock/src; — bien:
cd/usr2/acct/mandolo;
```

Por último, veamos la ejecución de un proceso usuario del Window Manager. Usando la sentencia **wcmd**, ejecutamos mediante el comando **doform** el formulario clientes:

```
   wcmd "doform clientes";
```

Capítulo 24

Actualizando la Información

En este capítulo se verán los comandos que permiten modificar los datos que contiene una tabla. El manejo de tal información involucra tres aspectos bien definidos:

- Agregado de nuevos datos a la tabla. Esto implica la incorporación de una o varias filas. Si se trata de la tabla **emp**, por ejemplo, cada nuevo empleado que se incorpore a la empresa dará lugar al agregado de un registro. En otro tipo de tablas, tal como una lista de materiales, la incorporación de un producto requiere la inserción de múltiples registros que corresponden a sus partes o piezas componentes.
- Modificación de los datos existentes. La información de cualquier tabla es cambiante por naturaleza. Los empleados cambian de departamento, reciben aumentos de sueldo, ganan comisiones o pasan a depender de un nuevo jefe. Esto implica la necesidad de alterar el contenido de cualquier columna de un registro determinado.
- Supresión de las filas innecesarias. Es una realidad de la vida que los empleados se jubilan, renuncian, son despedidos ... o se mueren. En otros tipos de tabla, el transcurso del tiempo da lugar también a que ciertos registros pasen a ser la perfecta imagen del burócrata: ocupan lugar pero no sirven para nada. Volviendo a la lista de materiales, es el caso de aquellos productos cuya producción se ha descontinuado.

Analizaremos entonces, en las secciones que siguen, la manera de llevar a cabo cada una de estas tres actividades.

La Sentencia insert

El agregado de una fila o registro a una tabla recibe el nombre específico de **inserción**, de allí el nombre de este comando: **insert** (insertar). Su sintaxis es:

```
insert into tablename (lista_de_columnas)
```

```
values valores;
```

La tabla cuyo nombre hemos simbolizado como “tablaname” debe pertenecer al esquema corriente; si no fuera así, podemos optar entre dos caminos: anteponer el nombre del esquema al de la tabla (el esquema debe estar activo) o bien emitir previamente un comando **use**.

La lista de columnas es optativa, pudiendo indicar un subconjunto del total en cuyo caso las omitidas quedarán con valor nulo -siempre que las especificaciones lo permitan-.

Por último nos queda la especificación de los valores a incluir. La forma más simple consiste en la palabra clave **values** seguida de otra lista con los datos correspondiente.

Así por ejemplo:

```
insert into emp (nroleg, nombre, cargo, fingr, depno)
values (015, "M—nica G—mez", 7,'23/11/89',01);
```

Los campos jefe, fecha de nacimiento, sueldo y comisiones quedan sin valor. De esta manera se incluyen nuevas filas de a una por vez.

En el caso de que no se especifiquen las columnas los valores indicados en valúes deben corresponderse uno a uno con los campos definición de la tabla. Existe otro medio más poderoso -pero que por lo mismo debe utilizarse con cuidado- que es suministrar los valores por medio de un query. Si recordamos que le resultado de una consulta es lo que llamamos **grilla**, que en definitiva no es otra cosa que una tabla, esto no debe sorprender. Como es obvio, el **select** deberá elegir las mismas columnas que hemos especificado en la lista del **insert**, tomadas de una tabla diferente de aquella en la cual se hace la inserción. Esto último significa que no es lícito copiar una tabla sobre s’ misma.

La cláusula **where** permitirá copiar filas en forma selectiva, o bien tomar la totalidad de la tabla si se la omite. esto nos da la posibilidad de hacer lo que se llama un “backup” (copia de respaldo) según el procedimiento que sigue:

```
create table tabkup(... lista de columnas=tabla original-...);
insert into tabkup
select * from tabla orig;
```

Resulta evidente que la tabla de backup (tabkup) es una copia fiel (todas las filas, todas las columnas) de la tabla original (tablaorig).

Actualizando Valores: update

para ubicar un item o dato específico dentro de una tabla, sabemos que es necesario fijar dos referencias o “coordinadas”: la fila y la columna. La primera debe ubicarse a través de un dato que la identifique en forma unívoca -el cual suele ser por ellos la **clave** del registro- como es el número de legajo en la tabla **emp**. La columna sabemos que se identifica por su nombre.

Por otra parte, es posible que la Modificación de un campo quiera hacerse extensiva a un grupo de registros, o aún a toda la tabla. Sería el caso en que se aumentará el sueldo en un porcentaje dado a un departamento, o bien a toda la empresa.

hechas estas consideraciones previas, veamos la sintaxis del comando **update**:

```
update tablename set lista_de_asignaciones
[where condición_de_bœsqueda];
```

La cláusula **where** es optativa: si se utiliza, permite seleccionar uno o varios registros que cumplan con la condición impuesta; de lo contrario, se actualizarán todas las filas de la tabla. Es convención generalmente adoptada el encerrar entre corchetes los ítems optativos; en ningún caso los corchetes forman parte de la sentencia tal como debe ser le'da por el sistema.

La cláusula **set** trabaja por definición con un lista de asignaciones, por lo que no es necesario utilizar paréntesis para delimitarla. El fin de la lista viene dado por lo aparición de la cláusula **where** o bien del punto y coma que señala el final del comando, lo que fuere que ocurra primero.

Las asignaciones deben ir separadas por comas, y cada una de ellas debe observar el siguiente formato:

```
columna=expresión
```

donde la expresión puede ser un valor constante o bien el resultado de operaciones efectuadas con campos pertenecientes a esa misma fila. Esto incluye el valor de la propia columna que se quiere actualizar, como en el caso del incremento de sueldos ya mencionado:

```
update emp set sueldo = sueldo * 1.25
where depno = 2;
```

que otorga un aumento del 25% a todos los empleados del Departamento de Desarrollo. Si suprimimos la cláusula **where**, para lo que deberíamos colocar el ";" luego del factor 1.25, el incremento se hace extensivo a toda la empresa.

En ciertos casos en que el valor a reemplazar puede depender de alguna condición, es útil recordar la posibilidad del uso de la expresión condicional

```
(condición ? trueval : faseval)
```

que se evalúa como "trueval" (valor por verdadero) si la condición "condition" se cumple, mientras que en caso contrario toma el valor "falseval" (valor por falso).

Para actualizar un campo que contuviera el porcentaje de descuento por Obra Social cuando dependía de la existencia o no de cargas de familia, y suponiendo una variable booleana cargfam que lo determinara, la expresión

```
(cargfam = 1 7 3.0 : 2.0)
```

permite asignar el 3% a quienes tienen cargas y el 2% a los que carecen de ellas.

La Sentencia delete

Elimina filas completas de una tabla, conforme a la sintaxis:

```
delete from tablename where condición_de_baja;
```

donde debe tenerse mucho cuidado si se omite la sentencia **where** (que aquí también es optativa) puesto que el comando barrerá con toda la información contenida en la tabla.

Cuando se ejecuta un **delete**, se suprime toda la información pero la tabla en sí como estructura lógica continúa existiendo. La situación es entonces similar a lo que ocurre inmediatamente después de un **create** (cfr. primer párrafo de este capítulo).

Si se ha borrado una tabla completa por error, queda el recurso de restaurar la información apelando a un backup, el cual se recomienda tener disponible hasta tanto se verifique que los datos no sean realmente necesarios. Existe otra posibilidad: ejecutar el comando rollback, del cual hablaremos brevemente en el apartado siguiente.

El Comando rollback

Su efecto consiste en volver atrás lo hecho dejando la tabla en su estado original. Para comprender cómo es posible esto, haremos algunas simples consideraciones.

La Modificación de datos almacenados en un sistema de computación tiene dos etapas bien definidas: la primera de ellas es la registración en memoria de los cambios efectuados -altas, bajas, modificaciones-; la segunda es la grabación de tales cambios en disco para hacerlos permanentes. El estado anterior de la tabla solamente se pierde -abstracción hecha de la existencia de backups- cuando se hace efectiva la segunda operación.

Por ende, si se emite un comando **rollback** antes de producirse la alternación física de la tabla, haremos salvado la dificultad. Como es natural, lo dicho vale tanto para **delete** como para **update**.

El Comando commit work

Este comando finaliza todas las transacciones pendientes. Es decir efectiviza las modificaciones, bajas, altas, etc. de los registros afectados durante la sesión del **iql**. Esto implica que hasta tanto se ejecute este comando, los registros afectados quedarán bloqueados para el resto de los usuarios.

Una vez cumplido el comando, sólo se podrá recuperar con **rollback** aquellas operaciones que se realicen con posterioridad. Esto significa que el **rollback** recuperará el estado anterior de la tabla a partir del último **commit work** que se haya ejecutado.

Si nunca se ejecutara en forma explícita el comando **commit work**, se efectivizarán las transacciones cuando se abandone el utilitario **iql**. Esto equivale a decir que dicha salida implica un “commit work” implícito.

Algunas Consideraciones Adicionales

Siempre se prudente obtener una copia de respaldo de una tabla antes de hacerle modificaciones, particularmente si van a consistir en actualizaciones y bajas.

Antes de dar por buena una alteración, conviene asegurarse cuidadosamente de que efectivamente las cosas se hicieron bien.

Algunas veces -sólo la experiencia puede llegar a indicar cuáles- resulta más conveniente y seguro reemplazar un **update** de condicionamientos complejos y valores múltiples por el mecanismo de reemplazo **delete / insert**. Es decir, se generan en primer lugar registros con el formato apropiado en una tabla auxiliar, cuyo contenido sea exactamente lo que queremos obtener como resultado final. Sí este proceso no produce el resultado querido, puede rehacerse hasta obtenerlo. Luego se suprimen todos los registros correspondientes de la tabla principal, y finalmente se insertan todas las filas tomadas de la tabla auxiliar.

Recalcamos que este es un procedimiento de excepción, y que no debe tomarse como modelo normal de operación.

Capítulo 25

Sentencias de Definición de Datos

Hasta aquí trabajado y propuesto ejemplos de ejercitación sobre tablas ya existentes, provistas junto con el sistema precisamente con ese propósito. En este capítulo veremos la manera de crear y modificar esquemas y tablas, al igual que algunos otros aspectos que hacen al manejo apropiado de ambos. Esto se logra por medio de los Data Definition Statemets (DDS).

Creación de Esquemas y Tablas

Si bien es posible definir, crear y modificar esquemas y tablas de manera interactivo, es recomendable generar archivos que contengan las sentencias DDS y ejecutarlas en modo batch. Esto brinda mayor confiabilidad: un procedimiento muy práctico y que minimiza errores consiste en copiar un file que contenga una definición similar a lo que se desea hacer y editar la copia introduciendo las modificaciones (tales como cambios de nombre, longitud, etc.) que correspondan según el caso.

Cómo Crear un Esquema

Para la creación de un esquema existe la sentencia **create schema**. Así por ejemplo:

```
create schema personal;
```

dará origen a un esquema llamado “personal” y además lo convertirá en el esquema corriente. Veamos que la creación de esquema implícito un use, en tanto fija un nuevo esquema corriente.

Es importante tener presente las consecuencias de lo antedicho: toda operación sobre tablas ejecutada a continuación se referirá a una esquema corriente mientras no se especifique lo contrario.

Supongamos que el esquema corriente sea uno denominado ALFA. Para poder hacer

referencia a una tabla perteneciente a otro esquema, deberá prefijarse el nombre de este (v.gr. BETA.TABXX); de lo contrario obtendremos un mensaje de error dando a TABXX como inexistente. Por supuesto, si ya no precisamos aludir a las tablas del esquema LAFA, será más práctico emitir un comando.

```
use BETA;
```

y luego continuar con nuestro trabajo sobre las tablas de ese esquema.

Definición de Tablas

La sentencia **create table** se utiliza para definir las características de una tabla, para ello se especifica el nombre de la misma, y luego debe seguir una lista encerrada entre paréntesis, con la descripción de las columnas separadas por comas. Antes de entrar en detalle sobre la sintaxis de este comando, es conveniente decir unas palabras sobre la manera de designar esquemas y tablas.

Es útil saber que el concepto de esquema consiste en un conjunto de tablas. Como es obvio, no puede haber dos esquema con el mismo nombre, ni dos nombres de tabla iguales dentro del mismo esquema. El sistema sí acepta que haya dos tablas de nombres iguales pertenecientes a distintos esquemas. Los nombres en sí pueden tener hasta 11 caracteres de longitud en UNIX y 8 en DOS, alfabéticos o numéricos; se acepta como alfabético el carácter subrayado "_". El primero de los caracteres de un nombre de esquema o tabla debe ser alfabético. Son nombres válidos:

PERSON_EXT (Nómina de personal asignado en el exterior)

ESTAD89 (Estadísticas del año 1989)

En cambio no son válidos:

*SALDOS (Primer carácter no alfabético)

SALDOS/88 (Contiene un carácter no alfabético ni numérico)

35_ACUM (Primer carácter no alfabético; es válido en cambio ACUM_35)

No se hace diferencia entre mayúsculas y minúsculas.

La Sentencia create table

Veamos como ejemplo la manera de crear la tabla emp que nos ha servido para ensayar los ejercicios de los capítulos anteriores.

Si bien la mayor parte de los elementos definitorios de una tabla ha surgido implícitamente con el uso, es conveniente establecerlos en forma completa y detallada.

Tal definición consiste en una lista de datos referentes a las columnas que componen la tabla; para cada columna existe un cierto grupo de características obligatorias, mientras que algunas

otras son optativas. Por su parte, la tabla en conjunto tiene como obligatoria la definición de las columnas, pero otras especificaciones son optativas.

Pasemos a detallar entonces la definición de una columna:

Nombre : Debe responder a las normas ya indicadas sin restricciones en la cantidad de caracteres.

Tipo de dato :

- CHAR, cuando contiene caracteres.
- NUM, cuando se trata de valores numéricos.
- FLOAT, para registrar cifras en la modalidad de “punto flotante”. Se utiliza cuando es preciso manejar valores muy grandes o muy pequeños. No lleva cantidad de posiciones.
- DATE, para almacenar fechas en formato interno.
- TIME, para almacenar horas en formato interno.

Longitud : Si indica entre paréntesis para CHAR y NUM la cantidad de caracteres o dígitos, respectivamente. No corresponde ponerlo para los campos de fecha, hora o float, ya que tienen un formato predefinido. En un campo NUM puede especificarse una cantidad de posiciones decimales: (8,2) indica que la longitud es ocho dígitos, pero los dos últimos son centésimos. Si el valor es entero, la cantidad de decimales es cero, v.gr.:(5,0), pero es más cómodo expresarlo simplemente como (5).

Validez : Esta especificación es optativa, y se refiere a la posibilidad de que el iql controle el contenido de las columnas para admitir o rechazar los valores que se intenten incorporar. Los criterios de validez permiten especificar operadores relacionales como <,>, etc. y la verificación de que un valor se encuentre en una tabla.

Otros : En la definición de una columna se pueden especificar una descripción del significado del valor almacenado en la misma y un valor por defecto a utilizar en ciertas operaciones.

Ahora estamos en condiciones de comprender bien la siguiente sentencia:

```
create table emp { // Nómina de empleados
nroleg num(3) not null, // Número de empleado ( legajo)
Nombre char(20) not null,// Nombre y Apellido
cargo num(1), // Tarea que desempeña
jefe num(3), // Nro. de legajo del gerente
fnac date, // Fecha de nacimiento
fingr date, // Fecha de ingreso
sueldo num(7,2), // Remuneración mensual
comis num(7,2), // Comisiones del mes
depno num(2) // Número de departamento
} primary key (nroleg);
```

Las dos primeras columnas tienen una especificación de validez “not null”, que no permite dejar sin un valor estos datos; las restantes sí pueden carecer de valor. Es importante notar la

apertura de paréntesis antes del Nro. de legajo, que indica el comienzo de la lista, y el cierre de la misma luego de la especificación de Nro. de Departamento; la presencia aquí de dos cierres consecutivos de paréntesis se debe a que el primero corresponde a la cantidad de dígitos del departamento: (2), mientras que el siguiente es el que da por terminada la lista de columnas, este último par podría ser también reemplazo por un par de llaves ({}).

Encontramos además una especificación adicional relativa a la tabla, y es que la **clave primaria** (primary key) es el número de empleado. Esto da lugar a que el sistema genere un índice que facilita las tareas de búsqueda. Será en este caso el índice principal o primario, el cual es obligatorio definir al crear la tabla; ello no obsta a la generación de otros índices denominados secundarios, lo cual se analizará en detalle en el apartado "La Indexación y Cómo Lograrla" de este capítulo.

Validación de Campos

Hemos visto que la especificación de una columna puede optativamente llevar lo que se denomina “cláusula de integridad” o validación de contenido del campo, cuyo formato más simple es **not null**.

Existen otras formas de validar la información, que son las cláusulas **in**, **in table**, **mark** y las validaciones con operadores relaciónales. Las dos primeras responden a un concepto similar al de la cláusula **in** de la sentencia **where**: proporcionar una serie de valores elegibles. La diferencia está en que en este último caso se trata de los valores que dan lugar a que un registro se seleccione para el query, en tanto que en una cláusula de integridad se establece que son los únicos aceptables para su incorporación a la tabla; los demás son rechazados.

La cláusula de integridad **in** seguida de una lista de valores que como ya se ha dicho debe ir encerrada entre paréntesis y con sus elementos separados por comas, permite especificar optativamente una descripción, separándola del valor del campo por dos puntos. Así por ejemplo:

```
create table pasises {
ctrycode char (3)
in ("ARG": "Argentina",
"ESP": "España",
"MEX": "MŽxico",
"JAP": "Japón",
"USA": "Estados Unidos"),
.....}
```

La tabla de países contiene, además de otros campos posibles que se señalan con puntos suspensivos, un código de país (country code) de tres caracteres, que no puede ser otro que alguno de los cinco valores indicados.

Ahora bien: ¿ qué pasa si queremos agregar un valor a la lista, tal como “URS”: “Unión Soviética”? Tendríamos que redefinir la tabla, es decir, volver a ejecutar un **create table** con la nueva lista, pero teniendo que repetir todas las especificaciones restantes. Esto no sería tan trágico si hemos tenido la precaución de guardar un archivo con la creación de la tabla, pero aún el proceso insume una cierta cantidad de tiempo y trabajo que puede obviarse si

cambiamos la cláusula de integridad en esta forma:

```
ctrycode char (3) in codpais: descr,
```

Ahora la lista de valores está contenida en una tabla llamada “codpais”, que no es otra cosa que una tabla cada una de cuya filas proporciona uno de los elementos de la lista. De tal manera, al no estar los valores incorporados explícitamente al **create**, la cláusula **in table** permite que la modificación se logre simplemente corrigiendo la tabla auxiliar de validación sin necesidad de redefinir la tabla principal.

La Cláusula mask

Hemos dejado para el final y en sección aparte esta cláusula de integridad, por su mayor grado de complejidad, aparte de ser una implementación propia del **iql**.

Comenzaremos definiendo el concepto de **máscara** (tal el significado de la palabra “mask”) como se utiliza ampliamente en diversas implementaciones propias de la computación, particularmente en la casi totalidad de los lenguajes. Una máscara se define como una cadena de caracteres que responde a una cierta codificación, según la cual se puede especificar:

- Que la posiciones dentro de un campo se toman tal como vienen, o bien surgen de una sustitución.
- Que se acepta cualquier valor o sólo aquellos de un determinado tipo.
- Que una cierta posición pueda faltar (estar en blanco) o no.

A semejanza de la cláusula **in**, la máscara puede estar directamente especificada en la cláusula -teniendo presente que ya no se trata de un lista, sino de una cadena de caracteres-, o bien aludir a una referencia externa que en este caso es una variable ambiental. La misma se especifica precediendo su nombre por el signo “$”, como por ejemplo:

```
nroserie char(10) mask $serialnr
```

donde la variable ambiental “serialnr” contiene la cadena de caracteres que conforma la máscara para validar un número de serie que debe, por ejemplo, contener caracteres alfabéticos en ciertas posiciones y numéricos en otras.

Los distintos tipos de datos que pueden verificarse mediante la máscara se especifican con los códigos:

a, A : Alfabético (en sentido estricto: no se aceptan blancos)

n, N : Numérico (dígitos de 0 a 9; único aceptable para **num**)

x, X : Alfanumérico.

h, H : Numérico Hexadecimal.

: Oculta el carácter.

La diferencia entre el uso de minúscula y mayúscula es que la primera permite la ausencia de un carácter en esa posición -vale decir, la aparición de un blanco-, mientras que en el segundo caso es obligatoria la presencia de un carácter que cumpla con la especificación.

La máscara se corresponde en su expresión elemental con la longitud del campo; vale decir, cada posición de aquella controla la posición correlativa de esta última. No obstante, esto deja de ser aplicable cuando se utiliza alguno de los caracteres prefijables > (convertir a mayúsculas) o< (convertir a minúsculas, que n son válidos para el código “n” o “N”; o bien si se emplea un factor de repetición (3a en lugar de “aaa”).

Si por ejemplo queremos que un campo de cinco posiciones sólo acepte tres caracteres alfabéticos obligatorios, los cuales deben convertirse a mayúsculas, seguidos de dos caracteres numéricos de los cuales sólo el primero es obligatorio, la máscara será:

```
3>ANn.
```

Fuera de los caracteres con significado especial, todos los demás se tomarán literalmente. Es decir, los campos **date** tienen implícita una máscara nN/NN/NN. Esto se menciona a título ilustrativo, ya que en realidad el sistema no permite indicar máscara para campos **date, time** o **float**.

Como ejemplo final, daremos las máscaras para números telefónicos en nuestro país y en EE.UU. de Norteamérica. En el primer caso será:

```
nNN-4N
```

Tres dígitos, dos de ellos obligatorios, separados por un guión de otros cuatro obligatorios.

Para EE.UU. será en cambio:

```
(3N) 3>X-4>X
```

Hay un código de área de tres dígitos que se encierra entre paréntesis, y luego una característica alfanumérica en la cual las letras deben convertirse a mayúsculas, separada por un guión del número de abonado. Todas las posiciones -diez en total- son obligatorias.

Validaciones con Operadores Relacionales

Los campos podrán tener asociados atributos de validación especificados mediante operadores relaciones. Estos atributos de chequeo, impedirán el ingreso de valores que no concuerden con lo indicado en la definición del campo, centralizando de esta forma, la consistencia de los datos. Su sintaxis genérica es:

```
oprel <valor>
```

donde oprel puede ser:

Operador Relacional	Descripción	Ejemplo

<	Menor	< 86
>	Mayor	> "ABC"
=	Igual	= today
<=	Menor o Igual	<= "28/03/90"
>=	Mayor o Igual	>= "09:30"
!= ó <>	Distinto	!= 120
[not] between ... and ...	Rango de valores	between 1 and 100

El operador between permite definir un rango de valores donde los extremos forman parte del mismo. Es posible negar el rango, anteponiéndole not al atributo.

El Atributo default

El significado de la palabra **default** es: valor por defecto; es decir, el valor que asignará el sistema ante la ausencia de una especificación concreta. Incluyendo este atributo en la definición de una columna que se agrega a una tabla, por ejemplo:

```
vigencia date not null default today,
precio num(7,2) not null default 0,
```

para incorporar a una lista de materiales el precio del proveedor y la fecha de vigencia del mismo, logramos evitar el rechazo.

El default para un campo cuando no se lo indica es siempre el valor nulo correspondiente a ese tipo de campo.

El atributo **default** es independiente de la cláusula de validación, si bien debe ser coherente con ella y con el tipo de campo. No es lícito un default “XXX” para un campo numérico (las tres equis son un valor alfabético), ni tampoco para el código de país de la tabla usada como ejemplo anteriormente. Si bien “XXX” responde a la especificación **char(3)**, no figurada en la lista de valores permitidos en la cláusula **in**.

Modificación y Supresión de Tablas

Una vez creada una tabla, cargados sus datos y habiendo comenzado a utilizarla, con el transcurso del tiempo suele ocurrir que se le deben introducir modificaciones. No nos referimos al hecho de agregar, quitar o alterar filas, sino a la necesidad de añadir o suprimir una columna, o bien modificar alguna de sus características.

Esto implica una modificación del diseño lógico de la tabla y como consecuencia también del esquema al cual ella pertenece. Para facilitar las alteraciones a un esquema, es conveniente haber almacenado, inmediatamente después de crearlo y de definir sus tablas, el conjunto de sentencias usadas para hacerlo.

El Comando store

Para poder utilizar esta facilidad, es necesario crear un archivo que contenga las sentencias **use, create**, etc. destinadas a generar el esquema y las tablas deseadas. Luego de ejecutar dicho archivo en modo batch, y suponiendo que el esquema creado fuera el "epsilon", el comando:

```
store schema epsilon;
```

generará un file llamado "epsilon.sc" que contendrá las especificaciones correspondiente volcadas desde el formato interno que utiliza el sistema.

Cuando surja la necesidad de alterar una tabla perteneciente a **epsilon**, se usará un editor -que puede ser el mismo de **iql**, ver Cap. 26- para introducir las modificaciones en la definición de las columnas, en dicho archivo, y se repetirá el procedimiento. Esto es, se ejecutarán los comandos contenidos en el file.

En otras implementaciones del SQL, se emplea un comando **alter** -inexistente en iql- para introducir modificaciones en tablas o clusters, debiendo el programador introducir las alteraciones una a una. Aquí en cambio, es siempre el comando **create** el que se encarga tanto de generar los esquemas y tablas por primera vez como de modificarlos. Es decir, se hace una revisión completa de todas las especificaciones, dejando inalteradas aquellas a las cuales no se ha modificado, y reestructurando lo que ha sido objeto de alteración. En otras palabras, se **re-crea** el esquema.

En secciones siguientes veremos algunos requisitos que deben observarse al proceder a la modificación de esquemas y tablas.

Usando creete para Modificar

En primer lugar remarcaremos que se entiende por modificación de un tabla el hecho de alterar su estructura lógica; esto es, agregar o suprimir una o varias columnas, o bien cambiar su longitud (incluyendo la cantidad de posiciones decimales de un campo numérico, aún sin modificar el total de dígitos) o las reglas de validación.

Cuando se ejecuta un comando **create table** sobre una ya existente, el sistema emite una advertencia ("warning") en tal sentido, para evitar que por una coincidencia accidental de nombres quien quiere en realidad crear una nueva tabla, modifique otra preexistente. Además, el mensaje tiene otra función que es poner sobre aviso al usuario de la posible necesidad de modificar la variable "modify".

Las Variables modify y force

Estas variables controlan la posibilidad de modificar las estructuras de datos. La variable **modify** da el primer nivel de posibilidad de modificación. Su existencia (es decir si está definida, no importando cual es su valor) permite la alteración del esquema en sí, es decir, el

agregado o la supresión de una tabla, En el nivel de tablas permite el agregado de nuevas columnas y la modificación de características de las mismas, salvo el cambio del tipo de dato o cualquier cambio de precisión que implique pérdida de información. No permite en cambio la eliminación de columnas de una tabla.

Para realizar operaciones tales como: suprimir o modificar el tipo de dato de una columna y cambiar la precisión de un campo con posible pérdida de información, existe la variable **force**. Lógicamente **force** también permite todos los cambios que permite la variable **modify**.

Para cambiar el estado de estas variable se utiliza el comando **set**:

```
set modify;
unset modify;
```

Alteración de Columnas y cláusula de Integridad

Cuando se incremente la capacidad de una columna, ello no trae problemas con los datos ya existentes. Sí los hay en cambio cuando se la achica: los campos de caracteres pueden poder algunos en el extremo derecho, mientras que los numéricos pueden ver cercenados dígitos de orden superior, o sea extremo izquierdo. Es responsabilidad del usuario tener en cuanta esta posibilidad, pero normalmente no trae consecuencias en lo que hace a la integridad como norma de validación (obviamente sí en cuando a la integridad entendida en sentido total).

La supresión de una columna acarrea evidentemente la pérdida total de los datos contenidos en ella, pero desde el momento que se la suprime es porque ya no se la considera útil. Esto tampoco incide sobre la cláusula de integridad.

Distinto es lo que ocurre cuando se agrega una nueva columna, o se modifica la propia cláusula de integridad. Esto puede generar incompatibilidades, particularmente cuando la indicación es **not null**, ya que toda columna agregada a una tabla preexistente es llenada por el sistema con valores nulos. Si la columna ya existía, pero tenía valores nulos en algunas filas, la modificación de la cláusula de integridad a **not null** planeta la misma incompatibilidad.

Para resolver este problema en la definición de la columna se puede agregar el atributo **default** de manera que al modificar o agregar la nueva columna esta no quede con valor nulo sino con el indicado en dicha cláusula.

Supresión de Esquemas y Tablas

La antítesis de la sentencia **create** es **drop** (literalmente descartar, dejar caer), que puede aplicarse tanto a esquemas como a tablas. Su formato es:

```
drop schema nombresq;
drop table nombretab;
```

según el caso.

Resulta casi ocioso puntualizar que el **drop** implica la pérdida total de la información contenida en el ente afectado, y además la desaparición de su definición como estructura lógica para el sistema. Si imaginamos tener un esquema llamado heleno, dentro del cual se hallan definidas las tablas ALPHA, BETA y GAMMA, consideremos la sentencia:

```
drop table beta;
```

El objeto que puede tener este comando es liberar espacio en disco cuando la tabla BETA ha dejado de tener utilidad, sea porque se la deja de usar o bien porque ha sido sustituida por otra, ya sea incorporando sus datos a una tabla más amplia o eventualmente llevando la tabla completa a otro esquema. Recordamos que "heleno" debe ser el esquema corriente para la ejecución del comando.

La sentencia:

```
drop schema heleno
```

tiene un efecto más drástico: elimina directamente el esquema como entidad y junto con él a todas las tablas incluidas. Una consecuencia inmediata es que nos quedamos sin esquema activo, por lo cual, si queremos continuar trabajando en **iql**, será conveniente emitir de inmediato un use. Por ello podemos generalizar que, salvo el caso de ser lo último por hacer, todo drop schema debiera ir seguido de un **use**.

Un Breve Resumen

La modificación o supresión de esquemas y tablas es un tema delicado, que debe por consiguiente manejarse con máxima prudencia. Así por ejemplo, es factible editar directamente el file "esquema.sc" en lugar de hacerlo con el archivo correspondiente y luego ejecutarlo, pero no es recomendable, salvo que se tenga ya bastante experiencia en el tema, caso en el cual muy probablemente Ud. no estaría leyendo estas líneas.

Volviendo al mecanismo según el cual el sistema maneja las modificaciones, la idea fundamental es la de reprocesar el file "esquema.sc" conforme a las nuevas especificaciones encontradas en las sentencias **create** o **drop table**, dejando como está lo no afectado y corrigiendo aquello que ha sido objeto de alteración.

Al modificar tablas hay que tener en mente las restricciones relativas a la variable "modify", y la coherencia necesaria entre los valores por default y la cláusula de integridad.

Los Comandos lock/unlock

Una consideración final: ¿que sucede con los demás usuarios mientras alguien (generalmente el dueño del esquema) está introduciendo modificaciones?

Distinguimos dos situaciones: Se están modificando estructuras de datos en el esquema, o bien se están insertando, borrando o modificando filas en alguna tabla.

En el primer caso nadie más que quien está realizando las modificaciones puede tener acceso al esquema, por ello cuando se intenta realizar una modificación, se verifica que ningún otro usuario está usando el esquema. Si esto es así, se procede a bloquear el esquema y realizar la modificación. Si hay otros usuarios accediendo se informará de tal situación con un mensaje que informa de la imposibilidad de bloquear el esquema para acceso exclusivo.

En cuanto a la modificación de datos, desde ya el sistema adopta ciertos recaudos mínimos para impedir que dos usuarios puedan tratar de actualizar la misma tabla al mismo tiempo, pero no inhibe la posibilidad de leerla. Esto significa que un usuario puede en determinado momento acceder a la **versión vieja**, justo antes de que sea actualizada.

Para impedir esta situación, existe el comando **lock** que inhibe el acceso a los demás usuarios, ya sea a una tabla o a un esquema. Así pues:

```
lock schema heleno;
```

negará acceso a la totalidad del esquema nombrado, en tanto que

```
lock table alpha;
```

hace lo propio exclusivamente con esa tabla. Para liberarlos y facilitar el acceso a los demás usuarios, lo cual corresponde hacer al incluir las modificaciones, se usa el comando **unlock** con idéntica sintaxis.

La Indexación y Cómo Lograrla

El índice de una tabla, y más generalmente de cualquier archivo o base de datos, cumple las mismas funciones que las de un libro o manual. Su objeto principal es el de ubicar la información con mayor rapidez. Cuando disponemos de un índice alfabético, es fácil buscar en él una palabra o frase, tal como “expresiones” o “formatos de salida”, y el índice nos da las páginas donde se hace referencia al tema que nos interesa.

Si no dispusiéramos del índice, tendríamos que hacer una revisión a todo lo largo de la publicación hasta encontrar tales datos. De la misma manera, un índice facilita y hace más rápida la individualización de filas que contengan columnas con determinado contenido. Careciendo de él, el sistema necesita hacer un barrido completo de la tabla.

Es inmediato que la presencia de índices abrevia el tiempo requerido para la ejecución de consultas, en tanta mayor medida cuanto mayor sea el tamaño de la tabla.

Creación de Indices

A esta altura del partido, estoy convencido de que no producirá la menor reacción de asombro el hecho de que la cosa se logre mediante un comando:

```
create index indxname on tablname (column_list);
```

El nombre del índice, que hemos simbolizado como “indxname”, debe responder a las mismas normas ya establecidas para los nombres de esquemas y tablas. A los fines prácticos, es conveniente adoptar alguna convención que permita indexan, v.gr. **empx1**, **empx2**, etc. podrían ser los índices de la tabla **emp**. Pero en lo que hace al sistema la denominación es completamente libre.

La lista de columnas permite que la clave, es decir el campo búsqueda, está formado por varios elementos distintos de información. En principio, tales componentes están constituidos por el valor completo de la columna. Sin embargo, cuando se trata de campos de caracteres, es decir definidos como **char**, es posible usar la función que define una subcadena:

campo(10) : para tomar los diez primeros caracteres solamente

campo(3,8) : para ignorar los tres primeros y tomar los ocho siguientes

El Atributo unique

Puesto que podemos definir cuantos índices queramos sobre una tabla, y elegir como clave cualquier campo dentro de ella, es natural que aparezcan repeticiones. Esta situación, que se denomina “clave duplicada” (duplicate key), implica que dos o más filas contienen idéntico valor para una cierta columna -o combinación de ellas- que ha sido definida como clave.

Tal circunstancia puede no ser aceptable cuando se trata de columnas que por su naturaleza deben identificar unívocamente a una fila , tal como ocurre con un número de empleado, de cliente, o código de parte en una lista de materiales. Para asegurarnos de que el sistema rechace la inclusión de claves duplicadas disponemos de dos medios:

- Definir la columna -cuando es única componente de la clave- como “primary key”. En el apartado "Modificación y Supresión de Tablas" de este capítulo se indicó que es obligatorio el definir una clave primaria para cada tabla. Ahora ampliaremos este concepto señalando que la misma lleva implícito el atributo **unique**.
- Emplear el atributo de unicidad de clave en la sentencia **create**:

```
create unique index indxname on tablname (keyname);
```

De esta manera cada fila deberá tener un valor diferente en la columna “keyname”.

Otros Atributos de las Claves

Hasta aquí hemos dado por supuesto que la indexación de una tabla se hacía por el orden ascendente de la o las columnas que integraban la clave. Esto es un típico default pero puede modificarse especificando a continuación del nombre de una columna dentro de la lista, el atributo **descending**, que se abrevia **desc**. Asimismo, podemos excluir los campos nulos de la formación del índice agregando el atributo **not null**.

Como ejemplo, supongamos que se quiere un índice secundario sobre la tabla **emplead** (similar a la **emp**) por el campo de fecha de promoción, en orden cronológico inverso, o sea con las fechas más recientes primero. Tendríamos:

```
create index promind on emplead
(feprom desc not null);
```

Los empleados que no han tenido ninguna promoción tendrán el campo "feprom" en null value y no serán incluidos en el índice creado **promind**.

Especificación de Indices en create table

Ya vimos que después de las especificaciones de columnas, se podía indicar una clave primaria al momento de crear una table. El formato general de tal opción es:

```
primary key (column_list)
```

donde la lista de columnas puede incluir las especificaciones y atributos adicionales que se han explicado en los apartados anteriores de este capítulo.

Pero también se puede incluir al final de la sentencia **create table** la cláusula:

```
index indxname (column_list)
```

además de la cláusula **primary key.**

Supresión de Indices

Se efectúa mediante el comando

```
drop index indxname on tabname;
```

Es la menos traumática de las opciones del **drop**, ya que el índice es una estructura separada de la tabla, aunque vinculada a ella, y su eliminación no produce ningún efecto físico sobre la información propia de la tabla.

Párrafo adicional: ¿Qué pasa cuando se suprime una columna que se en todo o en parte un campo de clave? Debería producirse un **drop** autogenerado ya que ese índice no puede seguir existiendo. Y si se trata del campo que es primary key, ¿impide el sistema la supresión de la columna?

Consideraciones Finales

No siempre es provechoso indexar una tabla, pero seguramente lo será si contiene al menos varios centenares de registros. Crear índices para una tabla con menos de cien registros es una pérdida de espacio y de tiempo: en la mayoría de los casos hará que la respuesta empeore en lugar de mejorar.

No suele ser conveniente tampoco crear demasiados índices para una misma tabla: dos o tres debieran ser suficientes si están adecuadamente elegidos. Hay que tener en cuanta que, a cambio de acelerar los quieres, una gran cantidad de índices hace más lentos los procesos de insert, update y delete, debido a la necesidad de actualizar los índices afectados por esas columnas aparte de la modificación sufrida por la tabla en sí. Esto puede afectar seriamente los procesos batch que involucran agregado, modificación y supresión de registros.

El truco consiste en elegir como clave a las columnas más frecuentemente usadas en las cláusulas **where**. Es así como los índices van a rendir su mejor fruto. No tiene objeto indexar columnas usadas solamente para desplegar valores indicativos. Asimismo, resulta útil crear índices teniendo en cuenta posibles frecuentes operaciones de **join** entre tablas. Construir el índice sobre una u otra de las tablas puede ser indistinto en algunos casos, pero otros es preciso analizar el mecanismo según el cual el sistema busca para cada fila de una de las tablas, las posibles compañeras en la otra. Usando como clave la columna común -o varias, si lo fueran- el índice deberá generarse sobre esta segunda tabla.

Capítulo 26

El Editor de IQL

En los primeros capítulos se han mencionado en forma rápida las características más importantes del editor de iql, es decir de las facilidades de edición en pantalla de las consultas (de aquí en más llamaremos consultas en forma genérica a cualquier sentencia de SQL).

En este capítulo se describirán en forma detallada las posibilidades de esta interfaz de edición. Los usuarios del ambiente de desarrollo de IDEAFIX encontrarán que es totalmente idéntica al editor de ie, por lo que podrán obviar la lectura de este capítulo.

La Línea de Estado

Cuando se inicia la sesión con iql se presenta la pantalla en blanco, con el cursor en el borde superior izquierdo. A medida que se digita el texto en el teclado este aparece en la pantalla. El texto puede insertarse o sobrescribir el existente, según estemos en modo “inserción” o “reemplazo”. Este modo se indica en la línea de estado que está en la parte superior de la pantalla. Esta línea de estado informa lo siguiente:

- Modo inserción o sobrescritura con las indicaciones INSERTAR O REEMPLAZO respectivamente.
- Número de consulta que estamos editando, con la indicación Cmd: NN a la derecha de la línea de estado.
- El comando ha sido modificado, lo cual se indica con un asterisco (*) en la parte izquierda de la línea de estado.

Memorización de las Consultas

Cada consulta que se ejecuta, queda memorizada para poder ser ejecutada nuevamente, o ser corregida o modificada. El Número de consulta que vemos en la línea de estado es la consulta que podemos procesar.

Cuando hemos completado una especificación SQL y deseamos procesarla, se oprime la tecla PROCESAR. En este punto pueden ocurrir dos cosas:

- El comando es correcto y vemos su resultado.
- El comando produce algún tipo de error.

Si el comando es correcto, la indicación Procesando aparecerá en la línea de estado. mientras esta indicación -que usualmente aparece parpadeando- está presente, el sistema está realizando operaciones y debemos esperar que finalice. Cuando la indicación desaparezca podremos ver el resultado del comando. Si este implicaba el despliegue de varias páginas, luego de visualizar cada una debemos esperar que se complete la siguiente. Cuando demos por terminada la visualización del resultado, la pantalla se limpiará y el Número de consulta en la línea de estado avanzará en uno. La consulta anterior ha quedado memorizada. la forma de retroceder o avanzar en las consultas que se realizaron durante la sesión se mediante la teclas PAG SIG para avanzar en uno el Número de consulta, y PAG ANT para retroceder en uno.

Si en cambio se produjo algún error, se verá el mensaje en la zona inferior de la pantalla y la consulta permanecerá en pantalla para su corrección. El mensaje de error indicará en que línea se produjo el mismo. Para posicionarnos con el cursor en una línea conociendo su Número se digita el comando META NN g donde NN es el Número de línea en el cual deseamos posicionar el cursor.

Se debe tener la precaución de no avanzar ni retroceder si la consulta tiene un error, ya que perderemos lo que hemos topeado. En otras palabras, si obtenemos un error, oprimimos PAG ANT y luego oprimimos PAG SIG la consulta aparecerá vacía, ya que las consultas erróneas no son memorizadas.

Comandos

los comandos del editor se realizan mediante teclas de función de IDEAFIX. Algunos de ellos son directamente a través de una tecla de función, y otros usan como prefijo una de las siguientes teclas:

- META
- CTRL-X

por ejemplo cuando deseamos traer el contenido de un archivo sobre la consulta que estamos editando, ingresamos la secuencia CTRL-X r. En este caso en la parte inferior de la pantalla se requerirá el nombre del archivo que deseamos leer. Todos los comandos que requieran alguna información la solicitarán de esta manera. Cuando se desea cancelar un comando que nos está solicitando información, se hace mediante la tecla IGNORAR. Usualmente esta tecla es F8.

En las secciones que siguen a continuación se indican los comandos del editor. Cuando se indica [nombre], es el nombre de una tecla de función Ideafix (la tecla física cambia con la terminal usada, pero se puede consultar la correspondencia entre una y otra, generalmente mediante Ctrl-K o F12). nro: Significa que el usuario digita un Número.

Comandos de Borrado

[RETROCESO] : Borra carácter previo.

[CTRL-X]-b : Borra palabra previa.

[ELIMINAR] : Borra carácter siguiente.

[CTRL-X]-f : Borra palabra siguiente.

[META]-d : Borra línea.

[META]-e : Borra hasta fin de línea.

[CTRL-X]-o : Borra las líneas en blanco alrededor del cursor.

Comandos de Inserción

[INSERTAR] : Inserta espacio.

[META][INSERTAR] : Abre una línea.

[CTRL-X][INSERTAR] : Pasa de modo overwite a insert y viceversa.

Movimiento del Cursor

Los siguiente comandos permitir desplazar el cursor dentro de la consulta de acuerdo a distintos criterios.

[CUR-DER] : Carácter siguiente.

[CUR-IZQ] : Carácter previo.

[CUR-ARR] : Línea previa.

[CUR-ABA] : Línea siguiente.

[META]-[PAG_ANT] : Comienzo de la consulta

[META]-[PAG_SIG] : Fin de la consulta

[CTRL-X]-[CUR-DER] : Palabra siguiente.

[CTRL-X]-[CUR-IZQ] : Palabra anterior.

[META]-[CUR-DER] : Principio de línea.

[META]-[CUR-IZQ] : Fin de línea.

[META] : Nro g ir a la línea indicada.

Búsqueda y Sustitución

Cuando se desea buscar la ocurrencia de determinado patrón de caracteres dentro del texto de la consulta, se utilizan los siguientes comandos:

[META]-f : Busca un cadena hacia adelante desde la posición del cursor.

[META]-b : Igual a [META]-f pero la búsqueda es hacia atrás.

[META]-r : Reemplaza todas las instancias del primer string tecleado por el segundo.

[CTRL-X]-Q : Reemplazo con consulta. Posibles respuestas:

[IGNORAR] : cancele el comando

! : reemplace el resto

? : mostrar lista de opciones

Y : reemplace y continúe

N : no reemplace y continúe

Buffers y Regiones

El buffer temporario es un área interna del editor donde se almacena el texto que se ve afectado por cierto tipo de comandos: borrado de líneas, borrado de regiones, etc. Puede insertarse este buffer en cualquier lugar del texto con el comando: META y.

Las Regiones

una REGION se define como el área entre la marca y la posición actual del cursor. La marca es una posición del texto, que se indica ubicando el cursor en el comienzo de la zona a marcar, y oprimiendo META <ESPACIO> (barra espaciadora).

El procedimiento general para borrar, copiar o mover texto es el siguiente:

- Marcar el comienzo de la región.
- Ir al final de la misma.
- Emitir el comando de borrado o copiado al buffer temporal.
- Eventualmente, ir a la posición deseada y traer el texto desde el buffer.

Los comandos que permiten trabajar con la marca y buffer temporal son:

[META]-<ESAPCIO> : Coloca la marca en la posición actual.

[CTRL-X]-M : Intercambia la posición actual con la marca. Es útil para encontrar en qué posición está la marca.

[META]-p : Copia la región en el buffer temporario.

[META]-w : Borra la región y la almacena esta en el buffer temporario.

Manejo de Archivos

[CTRL-X]-r : Lee un archivo en la consulta actual.

[CTRL-X]-s : Salva la consulta sobre un archivo cuyo nombre será el Número de la consulta con extensión ".sql".

[CTRL-X]-w : Escribe en disco la consulta, preguntando el nombre que se le desea dar.

[META]-n : Pasa al siguiente archivo (next), en procesamiento múltiple.

Otros Comandos

[META]-U : Pasar a mayúsculas la palabra.

[META]-L : Pasar a minúsculas la palabra.

[CTRL-X]-C : Poner en mayúscula la primer letra de una palabra.

[CTRL-X]-L : Pasar a minúsculas la región.

[CTRL-X]-U : Pasar a mayúsculas la región.

[CTRL-X]-t : Transponer caracteres.

[AYUDA] : Muestra cómo están configuradas las funciones y sus teclas.

Capítulo 27

Apéndice A

Se detallan aquí los distintos tipos de mensajes susceptibles de ser emitidos por el sistema. Se clasifican en:

- Mensajes de Error.

  - Errores en tiempo de Compilación.
  - Errores en tiempo de Ejecución.

- Mensajes de Advertencia.

Mensajes de Error

Estos mensajes se pueden dividir en dos grandes grupos: aquellos que son detectados durante la compilación, y los que se presentan al momento de la ejecución.

Mensajes de Error de Compilación

El rango de números dato para estos errores es del 1 al 500. Los casos que pueden detectarse son los siguientes:

5. Error de Sintaxis.

Este mensaje indica que ha habido un especificación incorrecta en una sentencia.

Por ejemplo: falta el punto y coma (“;”) al final de la sentencia; no se intercaló una coma como separador de campos; falta alguna especificación en la cláusula (p.ej. “from”); palabra clave mal escrita (v.gr. were por where), etc.

6. Carácter octal inválido%o.

Existe un carácter que es ilegal en esa posición, o bien es un carácter de control.

7. Cadena de caracteres sin cerrar.

Este mensaje aparecerá cuando se especifique un “string” de caracteres al cual le falta la indicación de cierre, ya sea “ ' ” or “ " ”.

Por ejemplo: el nombre de una columna se especifica como [Nombre del Empleado] -faltan las comillas dobles señalando el fin de la cadena.

8. Archivo no encontrado.

Se intenta leer un archivo inexistente con la sentencia **exec**, o bien no existe el archivo especificado en la línea de comandos. Recomendación: verificar que el nombre este correctamente deletreado. “clientes” no es lo mismo que “clientes”, ni “Deptos” igual a “dptos”.

9. Tabla de marcos excedida.

Se han ejecutado demasiadas sentencias **define**.

10. Sentencia errónea.

El error debiera ser lo suficientemente grueso como para ser detectado a simple vista.

11. Falta la especificación de clave primaria (PRIMARY KEY).

no se ha establecido el campo de clave para acceso básico a la tabla.

12. Error de definición de Marco.

13. Redefinición de la CLAVE PRIMARIA.

El atributo “primary key” ha sido definido para más de un campo, o bien se ha agregado una especificación de índice primario inconsistente.

14. Combinación no soportada de condiciones múltiples de verificación.

Con anterioridad a la Versión 4.1 de **IDEAFIX**, no era válido especificar más de una validación como atributo de campo, salvo que se tratara de dos condiciones muy elementales. Ahora es posible hacerlo, incluyendo la utilización de expresiones condicionales complejas.

15. Valor %s incompatibles con el tipo de campo.

Se especifica un valor que no corresponde al tipo de campo en cuestión. V.gr.: una fecha que contiene caracteres no numérico.

16. longitud (%d) excede valor de longitud del campo (%d).

Se intenta asignar un valor por defecto cuya longitud es mayor que la del campo.

17. Valor con demasiadas (%d) posiciones enteras. Máx.: %d.

Se especifica un valor con mayor cantidad de posiciones enteras que las definidas.

18. Valor con demasiadas (%d) posiciones decimales. Más.: %d.

Se especifica un valor con mayor cantidad de posiciones decimales que las definidas.

19. Fecha (DATE) inválida: %s.

Se ha indicado una hora no válida (por ejemplo:29/02/90).

20. Hora (TIME) inválida: %s.

Se ha indicado una hora no válida (por ejemplo 15:69:42).

21. Tabla %s no encontrada.

Se ha referencia en una operación a una tabla no definida.

22. Tabla %s no declarada.

Se ha mencionado una tabla (por ejemplo, en una validación **in table**) que no pertenece al esquema corriente.

23. Campo no válido.

Se hace referencia un campo inexistente. Como en el caso anterior y otros análogos, verificar que el nombre este correctamente deletreado.

24. Cantidad errónea de valores.

25. Campo '%s'no definido.

26. Tabla '%s'an '%s'no definida.

27. Uso incorrecto del Alias '%s'en '%s'.

28. Redefinición del Alias '%s'.

Se ha intentado redefinir un alias en la sentencia from.

29. Variable ambiental '%s'indefinida.

Se ha hecho referencia a una variable de ambiente que no está definida en el momento de la compilación.

30. Número de dígitos no debe exceder de 15.

Se ha definido un campo numérico con más de 15 dígitos (se debe computar la totalidad: cantidad de enteros más posiciones decimales).

31. Cantidad de decimales debe ser < cantidad de dígitos.

Se ha definido un campo cuya cantidad total de dígitos es menor que la parte decimal.

32. Tabla de variables excedida.

Se han definido demasiadas variables con la sentencia set.

33. Función de Grupo '%s' no utilizable en esta cláusula.

Mensajes de Error en tiempo de Ejecución

La numeración de este grupo de mensajes comienza con el Nro. 501, y ellos son:

501 Sentencia no válida el modo corriente.

Se intenta ejecutar una sentencia de manipulación de datos en el utilitario dgen.

502 Esquema '%s'no encontrado.

Se ha hecho referencia en una operación a un esquema no activo (o incorrectamente deletreado).

503 Tabla '%s'no encontrada.

Una operación ha hecho referencia a una tabla no definida, o no perteneciente a ninguno de los esquemas activos.

504 Campo '%s'no encontrado.

Se ha hecho referencia en una operación a un campo no definido.

505 Indice '%s'no encontrado.

Se ha hecho referencia en una operación a un índice no definido.

506 No se pudo abrir el esquema '%s'.

Este error se puede producir si el usuario:

- Hace referencia a un esquema inexistente.
- No tiene permisos de acceso al esquema.

- Alude a un esquema que sí existe pero está corrupto.
- Intenta usar un esquema bloqueado para uso exclusivo de otro usuario.

507 Falló operación de grabación de definición del esquema '%s'.

En la sentencia store schema, no se pudo abrir el archivo de salida donde se debía grabar la definición.

508 No se pudo abrir archivo HEADER para esquema '%s'.

Mismo caso del mensaje anterior para sentencia store schema header.

509 No se pudo crear la Tabla '%s'.

510 No se pudo crear el Indice '%s'.

511 No se pudo crear el Esquema '%s'.

512 No puede agregarse el Campo '%s'.

Los errores anteriores son susceptibles de producirse al intentar crear o modificar un esquema.

513 Cantidad errónea de campos clave en la cláusula 'in'.

Se ha definido la clave de acceso en un atributo “in” con mayor cantidad de campo que la correspondiente.

514 Demasiadas columnas de salida.

Este error se produce cuando se intenta incluir un Número excesivo de campos en la grilla de salida, lo que lleva a superar la capacidad de manejo del **iql**.

515

Este Número de mensaje no ha sido usado.

516 No puede crearse el esquema '%s': ya existe.

Se intenta crear un esquema dándole el nombre de otro ya generado con anterioridad.

517 No puede crearse la tabla '%s': ya existe.

Similar al mensaje anterior.

518 No es posible convertir tabla '%s'carece de equivalente.

Al intentar modificar una tabla, la nueva especificación omite un campo previamente existente.

519 No es posible convertir tabla '%s' : campo '%s' tiene diferentes tipos.

En la modificación de una tabla, un campo ya existente está siendo especificado de distinto tipo.

520 No puede convertirse tabla '%s' : campo '%s' tiene menor longitud.

Auto-explicativo. Ello implicaría pérdida de información.

521 No puede convertirse tabla '%s' : campo '%s' tiene menos decimales.

Similar al anterior. La menor cantidad de decimales implica pérdida de precisión.

522 No puede convertirse tabla '%s' : '%s' tiene menos ocurrencias.

El campo es cuestión es repetitivo (vector), y la nueva cantidad de elementos es menor que la original.

523 No puede convertirse tabla '%s' : '%s' tiene diferente máscara.

Al modificar una tabla, se intenta asignar a un campo una máscara diferente a la que ya tenía.

524

Número de mensaje no utilizado.

525 No puede borrarse (drop) '%s'.

526 No pudo crearse una tabla temporaria.

527 Redefinición del Alias : %s.

Se ha intentado redefinir un alias en la sentencia **from**.

528 Demasiados Alias.

529 No se pudo crear un Alias.

530 Fecha (DATE) no válida : %s.

Se ha indicado una fecha errónea (por ejemplo: 31/06/90).

531 Hora (TIME) no válida : %s.

Se ha indicado una hora no válida (por ejemplo: 25:32:57).

532 La Función 'resto' (Remainder) no puede aplicarse a números no enteros.

Se ha aplicado la Función rem a un campo con decimales.

533 La operación '%s' no es válida para el tipo '%s'.

534 Operación '%s' : Tipos de dato incompatibles '%s' y '%s'.

Se está realizando una operación entre tipo de dato no compatibles, como por ejemplo dividiendo una fecha por Número entero (sí es lícito **sumar o restar** un valor entero a una fecha, que será interpretada como una cantidad de días de corrimiento). También puede ocurrir cuando no se colocan comillas para encerrar la fecha.

535 No puede cambiarse el tipo de variable.

536 El valor de la longitud (%d) excede la del campo (%d).

537 Inserción cancelada. Registro ya existente.

Se intentaba dar de alta un registro cuya clave ya figura en el índice. Probablemente se trate de una modificación; o bien se debe corregir alguno de los valores que componen la clave, para diferenciarla de la existente.

538 No puede bloquearse el esquema '%s'.

El sistema no está en condiciones de bloquear “LOCK”) el esquema indicado, pues está siendo utilizado por otro proceso.

539 No pudo abrirse el esquema '%s' : está bloqueado por otro proceso.

Para poder usar el esquema, es necesario esperar a que el proceso que lo tiene bloqueado lo libere; mientras tanto, no puede ser abierto.

540 Campo %s no cumple la condición NOT_NULL (no nulo).

El valor del campo es **nulo** pese a existir una especificación de que tal condición no es válida. Debe asignarse un valor, así fuere cero o blanco.

541 Campo: %s, Valor: %s no cumple condición IN.

El contenido del campo no figura dentro del conjunto de valores definidos en la cláusula **IN**. Si se requiere incluir nuevos posibilidades, es preciso redefinir la lista de códigos o datos admisibles.

542 Campo %s, Valor: $s no cumple condición NOT IN (no en).

El contenido del campo está incluido en el conjunto de valores prohibidos que define la cláusula NOT IN.

543 Campo %s, Valore: %s no cumple condición IN TABLE (en tabla).

El valor del campo no está incluido en el conjunto de datos contenidos en la tabla referida en la cláusula **IN TABLE**. Se el valor fuera efectivamente correcto, debe actualizarse previamente la tabla en cuestión para que sea aceptado.

544 Campo %s, Valor: %s no cumple condición NOT IN TABLE (no en tabla).

Caso simétrico al anterior. la tabla contiene ahora los valores **prohibidos**.

545 Campo %s, Valor: %s no cumple la condición BETWEEN (entre).

El valor del campo no está comprendido en el rango de valores aceptables definidos por el atributo BETWEEN.

546 Campo %s, Valor: %s no cumple la condición NOT BETWEEN.

Similar al caso anterior, sólo que ahora el rango no es de validez sino de **exclusión**.

547 Campo %s, Valor: %s no verifica la condición EQUAL (igual).

El valor del campo no corresponde a lo definido en la condición EQAUL.

548 Campo %s, Valor: %s no cumple condición NOT EQUAL (no igual).

El valor del campo es igual al definido en la condición NO IGUAL.

549 Campo %s, Valor: %s no cumple condición GREATER THAN (mayor que).

Auto-explicativa.

550 Campo%s, Valor: %s no cumple condición LOWER THAN (menor que).

Auto-explicativa.

551 Campo %s, Value: %s no cumple condición GREATER or EQUAL mayor o igual a).

Auto-explicativa.

552 Campo %s, Valor: no cumple condición LOWER or EQUAL (menor o igual a).

Auto-explicativa.

553 %s : La Tabla %s no existe.

554 %s : Los nombres de Tabla deben diferir.

No pueden existir definiciones de tablas con el mismo nombre.

555 %s : La Tabla %s ya existe.

556 Las nuevas tablas (como '%s') deben agregarse al final del esquema.

Si se modifica un esquema por adición de tablas, estas últimas deben agregarse al final del esquema en caso de modificarlo.

557 Tabla '%s' fuera de secuencia.

558 No hay esquema corriente.

559 %s : Los nombres de campo deben diferir.

560 %s : El campo %s ya existe.

561 No puede borrarse el Directorio '%s'.

562 Falló operación de borrado (Drop) para el esquema '%s'.

563 La cláusula having requiere especificar la cláusula group by.

Siempre que quiera usarse la cláusula “having”, deberá especificarse conjuntamente “group by”.

564 Cláusula 'Order by' inválida para 'one-row-select' (fila única).

565 Expresión Número %d tiene diferente nivel que la precedente.

566 Expresión de campo en 'one-row-select' (fila única).

567

Este Número de mensaje de error no fue utilizado.

568 Nivel erróneo de expresión para cláusula 'having'.

569 Nivel erróneo de expresión en cláusula 'order by'.

570 Expresión Número %d no incluida en cláusula 'group by'.

571 Demasiados niveles de funciones agrupadas en expresión %d.

572 Expresión con subíndice para campo único '%s'.

573 Subíndice fuera de rango para Campo '%s' Valor %d.

574 Descripción inválida de operando para campo '%s'.

575 Imposible convertir tabla '%s': campo '%s' no nulo y carece de default.

En una conversión de tabla, se está asignando el atributo “not null” a un campo que no lo tenía con anterioridad. Esto es válido solamente si se especifica un valor por defecto (default value) para sustituir a los posibles valores nulos preexistente.

576 La Table '%s' no puede ser bloqueada.

El sistema no está en condiciones de bloquear (“LOCK”) la tabla en cuestión, pues la está utilizando otro proceso.

577 Esquema '%s' bloqueado.

Para poder usar el esquema, es necesario esperar a que el proceso que lo tiene bloqueado lo libere; mientras tanto, no puede ser usado.

578 Esquema '%s' creado por una CPU incompatible.

Existen diferencias insalvables de hardware entre el equipo que generó la Base de Datos y el que ahora pretende procesarla.

Mensajes de Advertencia (Warnings)

Comentarios sin cerrar al fin de la entrada.

El final de la sentencia involucra el cierre de cualquier ítem que aún estuviera abierto, lo que obviamente incluye los comentarios. Pero cualquier otro tipo de cláusula no cerrada dará lugar a error, y no a warning.

Cláusula 'in table' ignorada.

Si la tabla aludida no existe, la pretensión de usarla como verificación no da lugar a error (se crea igualmente el esquema), pero se omite la cláusula de validación **'in table'.**

Expresión de campo '%s' no incluida en la cláusula 'group by'.

Para efectuar agrupamientos, es necesario indicar qué papel cumple cada uno de los campos seleccionados (incluidos en la sentencia select). En caso contrario, el **SQL** no sabe qué hacer con ellos: no se acumulan, y tampoco cortan control.