-PARTE I-

Tema 1. EL LENGUAJE SQL Y LOS SISTEMAS DE GESTIÓN DE BASES DE DATOS.

Autor: Fernando Montero

El lenguaje S.Q.L.

S.Q.L. significa lenguaje estructurado de consulta (*Structured Query Language*). Es un lenguaje estándar de cuarta generación que se utiliza para definir, gestionar y manipular la información contenida en una Base de Datos Relacional.

Se trata de un lenguaje definido por el estándar ISO/ANSI SQL que utilizan los principales fabricantes de Sistemas de Gestión de Bases de Datos Relacionales.

En los lenguajes procedimentales de tercera generación se deben especificar todos los pasos que hay que dar para conseguir el resultado. Sin embargo en SQL tan solo deberemos indicar al SGDB qué es lo que queremos obtener, y el sistema decidirá cómo obtenerlo.

Es un lenguaje sencillo y potente que se emplea para la gestión de la base de datos a distintos niveles de utilización: usuarios, programadores y administradores de la base de datos.

¿Qué es una Base de Datos?

Una base de datos está constituida por un conjunto de información relevante para una empresa o entidad y los procedimientos para almacenar, controlar, gestionar y administrar esa información.

Además, la información contenida en una base de datos cumple una serie de requisitos o características:

- Los datos están interrelacionados, sin redundancias innecesarias.
- Los datos son independientes de los programas que los usan.
- Se emplean métodos determinados para incluir datos nuevos y para borrar, modificar o recuperar los datos almacenados.

¿Qué es un Sistema de Gestión de Bases de Datos?

Un Sistema de Gestión de Bases de Datos (SGBD) es una aplicación comercial que permite construir y gestionar bases de datos, proporcionando al usuario de la Base de Datos las herramientas necesarias para realizar, al menos, las siguientes tareas:

- Definir las estructuras de los datos.
- Manipular los datos. Es decir, insertar nuevos datos, así como modificar, borrar y consultar los datos existentes.
- Mantener la integridad de la información.
- Proporcionar control de la privacidad y seguridad de los datos en la Base de Datos, permitiendo sólo el acceso a los mismos a los usuarios autorizados.

Nota.- La herramienta más difundida para realizar todas estas tareas es el lenguaje SQL.

Algunos de los productos comerciales más difundidos son:

- **ORACLE** de Oracle Corporation.
- **DB2** de I.B.M. Corporation
- **SYBASE** de Sybase Inc.
- **Informix** de Informix Software Inc.
- **SQL Server** de Microsoft Corporation.

Tipos de Bases de Datos.

Existen básicamente tres tipos de bases de datos:

- Bases de Datos Jerárquicas.
- Bases de Datos en Red.
- **Bases de Datos Relacionales.**

Éstas últimas son, con diferencia, las más difundidas y utilizadas en la actualidad debido a su potencia, versatilidad y facilidad de utilización. Se basan en el Modelo Relacional cuyas principales características veremos a continuación. Para gestionarlas se utiliza el lenguaje SQL.

El Modelo de Datos Relacional. Componentes.

El Modelo Relacional fue enunciado por *E.F. Codd.* Sus principales componentes son:

Entidad.

Es un **objeto acerca del cual se recoge información** relevante.

Ejemplo de entidades: `EMPLEADO, CLIENTE, PRODUCTO.`

Atributo.

Es una propiedad o característica de la entidad. Por ejemplo pueden ser atributos de la entidad `PERSONA` los siguientes: `DNI, NOMBRE, EDAD, ...`

Tabla.

Son los objetos de la Base de Datos donde se almacenan los datos.

Ejemplo de tabla de ***empleados***:

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISION	DEP_NO
7499	ALONSO	VENDEDOR	7698	20/02/81	140000	40000	30
7521	LOPEZ	EMPLEADO	7782	08/05/81	135000		10
7654	MARTIN	VENDEDOR	7698	28/09/81	150000	160000	30
7698	GARRIDO	DIRECTOR	7839	01/05/81	385000		30
7782	MARTINEZ	DIRECTOR	7839	09/06/81	245000		10
7839	REY	PRESIDENTE		17/11/81	600000		10
7844	CALVO	VENDEDOR	7698	08/09/81	180000	0	30
7876	GIL	ANALISTA	7782	06/05/82	335000		20
7900	JIMENEZ	EMPLEADO	7782	24/03/83	140000		20

Normalmente **una tabla representa una entidad** aunque también puede representar una asociación de entidades.

Las tablas están formadas por filas y columnas:

- **Cada fila** representa una ocurrencia de la entidad:

Ejemplo: Un empleado si es una tabla de empleados, un departamento si es una tabla de departamentos, un cliente si se trata de una tabla de clientes, o un producto si es una tabla de productos.

- **Cada columna:** Representa un atributo o característica de la entidad. Tiene un nombre y puede tomar por un conjunto de valores.

Ejemplo: La tabla de empleados puede tener como columnas o atributos: numero de empleado, nombre, fecha de alta, salario,...

Ejemplo de tabla de ***departamentos:***

	DEP_NO	DNOMBRE	LOCALIDAD
Fila 1 ->	10	CONTABILIDAD	BARCELONA
Fila 2 ->	20	INVESTIGACION	VALENCIA
Fila 3 ->	30	VENTAS	MADRID
Fila 4 ->	40	PRODUCCION	SEVILLA
	Columna 1	*Columna 2*	*Columna 3*

A lo largo de este curso utilizaremos, además de las tablas de empleados y departamentos, las tablas de clientes, productos y pedidos cuyo contenido es el siguiente:

TABLA DE CLIENTES:

CLIENTE_NO	NOMBRE	LOCALIDAD	VENDEDOR_NO	DEBE	HABER	LIMITE_CREDITO
101	DISTRIBUCIONES GOMEZ	MADRID	7499	0	0	500000
102	LOGITRONICA S.L	BARCELONA	7654	0	0	500000
103	INDUSTRIAS LACTEAS S.A.	LAS ROZAS	7844	0	0	1000000
104	TALLERES ESTESO S.A.	SEVILLA	7654	0	0	500000
105	EDICIONES SANZ	BARCELONA	7499	0	0	500000
106	SIGNOLOGIC S.A.	MADRID	7654	0	0	500000
107	MARTIN Y ASOCIADOS S.L.	ARAVACA	7844	0	0	1000000
108	MANUFACTURAS ALI S.A.	SEVILLA	7654	0	0	500000

TABLA DE PRODUCTOS

PRODUCTO_NO	DESCRIPCION	PRECIO_ACTUAL	STOCK_DISPONIBLE
10	MESA DESPACHO MOD. GAVIOTA	55000	50
20	SILLA DIRECTOR MOD. BUFALO	67000	25
30	ARMARIO NOGAL DOS PUERTAS	46000	20
40	MESA MODELO UNIÓN	34000	15
50	ARCHIVADOR CEREZO	105000	20
60	CAJA SEGURIDAD MOD B222	28000	15
70	DESTRUCTORA DE PAPEL A3	45000	25
80	MODULO ORDENADOR MOD. ERGOS	55000	25

TABLA DE PEDIDOS

PEDIDO_NO	PRODUCTO_NO	CLIENTE_NO	UNIDADES	FECHA_PE
1000	20	103	3	06/10/99
1001	50	106	2	06/10/99
1002	10	101	4	07/10/99
1003	20	105	4	16/10/99

1004	40	106	8 20/10/99
1005	30	105	2 20/10/99
1006	70	103	3 03/11/99
1007	50	101	2 06/11/99
1008	10	106	6 16/11/99
1009	20	105	2 26/11/99
1010	40	102	3 08/12/99
1011	30	106	2 15/12/99
1012	10	105	3 06/12/99
1013	30	106	2 06/12/99
1014	20	101	4 07/01/00
1015	70	105	4 16/01/00
1016	30	106	7 18/01/00
1017	20	105	6 20/01/00

Relación.

Conexión que puede haber entre dos entidades.

Por ejemplo: Cliente-> **compra**-> Producto
Empleado-> **pertenece a** -> Departamento

En nuestras tablas podemos observar las siguientes relaciones:

- **La tabla EMPLEADOS está relacionada con la tabla DEPARTAMENTOS através de la columna DEP_NO** (numero de departamento) que se encuentra en ambas tablas. De esta forma podemos saber, por ejemplo que el empleado GIL pertenece al departamento 20. Y si vamos a la tabla departamentos comprobaremos que el departamento 20 es INVESTIGACION y se encuentra en VALENCIA. Por tanto, el empleado GIL pertenece al departamento de INVESTIGACION que está en VALENCIA.

- **La tabla EMPLEADOS también se relaciona consigo misma mediante las columnas EMP_NO y DIRECTOR**. Cada empleado tiene un número de empleado (EMP_NO) y suele tener también un DIRECTOR. Esta última columna contiene un número de empleado que, suponemos, es el director del empleado en cuestión.
  Así podemos saber que REY es el director de GARRIDO y de MARTINEZ; y que el director de JIMENEZ es MARTINEZ, etcétera.

- **La tabla PEDIDOS se relaciona con PRODUCTOS mediante la columna PRODUCTO_NO y con CLIENTES mediante la columna CLIENTE_NO**. De esta forma sabemos que el pedido número 1000 lo ha realizado el cliente INDUSTRIAS LACTEAS S.A. y que el producto solicitado es SILLA DIRECTOR MOD. BUFALO a un precio de 67000, etcétera.

- **La tabla CLIENTES se relaciona con EMPLEADOS por medio de la columna VENDEDOR_NO de la primera que hace referencia a la columna EMPLEADO_NO de la segunda**. Así cada cliente tendrá asignado un vendedor.

El SGBD velará porque todas las operaciones que se realicen respeten estas restricciones manteniendo así la integridad de la información.

¿ Qué podemos hacer con SQL?

Todos los principales SGBDR incorporan un motor SQL en el Servidor de Base Datos, así como herramientas de cliente que permiten enviar comandos SQL para que sean procesadas por el motor del servidor. De esta forma, todas las tareas de gestión de la Base de Datos (BD) pueden realizarse utilizando sentencias SQL.

- Consultar datos de la Base de Datos.
- Insertar, modificar y borrar datos.
- Crear, modificar y borrar objetos de la Base de Datos.
- Controlar el acceso a la información.
- Garantizar la consistencia de los datos.

Tipos de sentencias SQL.

Entre los trabajos que se pueden realizar en una base de datos podemos distinguir dos tipos: definición y manipulación de datos. Por ello se distinguen dos tipos de sentencias SQL:

- Sentencias de manipulación de datos. (Lenguaje de Manipulación de Datos **DML**).

Se utilizan para:

- Recuperar información. (**SELECT**)
- Actualizar la información:
  - Añadir filas (**INSERT**)
  - Eliminar filas (**DELETE**)
  - Modificar filas (**UPDATE**)

- Sentencias de definición de datos. (Lenguaje de Definición de Datos **DDL**). Se utilizan para:

- Crear objetos de base de datos (**CREATE)**
- Eliminar objetos de base de datos (**DROP)**
- Modificar objetos de base de datos (**ALTER)**

SQL EN ACCES.

El motor de la base de datos ACCES se llama **Microsoft Jet**, permite administrar la base de datos, recuperar y almacenar datos en bases de datos del sistema y de los usuarios. Sentencias SQL que podemos manejar en ACCESS:

DML
DDL

SELECT
INSERT
DELETE
UPDATE
CREATE
DROP
ALTER

Tema 2. ELEMENTOS DEL LENGUAJE

Autor: Fernando Montero

Introducción.

Hay dos cuestiones a tener en cuenta a la hora de abordar esta unidad:

1º Se trata de una guía para que sirva de referencia o de consulta cuando se necesite a lo largo del curso.

2º En esta unidad, al igual que en la última, se abordan cuestiones que, aunque están definidas por el estándar ANSI/ISO SQL, no están asumidas al 100% por todos los fabricantes. Por tanto, pueden existir ligeras diferencias de algunos productos con algunas de las especificaciones que se aquí se exponen.

Tipos de datos.

Las columnas de la base de datos almacenan valores que pueden ser de diversos tipos: numérico, carácter, fecha, etcétera. A continuación se indican algunos de los tipos más utilizados.

- CHAR (*longitud*) se utiliza para guardar cadenas de caracteres de longitud fija especificada entre paréntesis. El espacio no utilizado se rellena con blancos.

- VARCHAR (*longitud*) almacena cadenas de caracteres de longitud variable cuyo límite máximo es especifica como *longitud*.

- NUMBER(*escala, precisión*) se utiliza para guardar datos numéricos. La escala indica el número total de dígitos y la precisión el número de posiciones decimales.

En Access no se indica ni escala ni precisión. Por defecto crea un tipo Numérico Doble. Podremos indicar INTEGER, REAL, DOUBLE, BYTE.

- DATE puede almacenar fechas. En algunos SGDBR también se puede almacena la hora en este tipo de datos.

La mayoría de los productos incluyen tipos de datos extendidos e incluso algunos productos ofrecen la posibilidad de que el usuario defina sus propios tipos. Todos estos tipos y posibilidades aparecen documentados en las especificaciones del producto, y escapan del objetivo de este curso

Identificadores.

Son nombres que sirven para identificar objetos de la base de datos: usuarios, tablas, columnas. El estándar define que pueden tener hasta 18 caracteres empezando con un carácter alfabético y continuando con caracteres numéricos y alfabéticos.

En la práctica, algunos productos no permiten nombres de usuario de más de ocho caracteres, pudiendo incluir hasta 30 ó más en los nombres de tablas y columnas.

Los ejemplos que aparecen en este curso se corresponden a la notación utilizada por ORACLE y se ajustan a las especificaciones del estándar ANSI/ISO SQL.

Operadores y expresiones.

Las sentencias SQL pueden incluir expresiones constituidas por nombres de columnas, constantes, funciones y operadores.

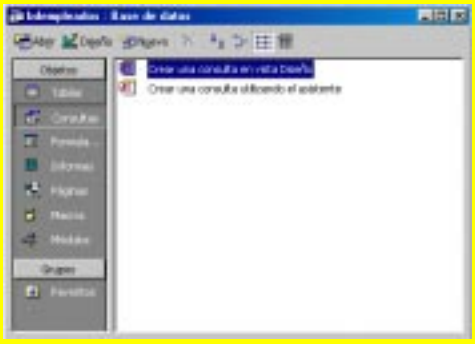
Por ejemplo la siguiente sentencia visualizará el apellido, la fecha de alta, el salario y la suma del salario con un complemento o gratificación de 100000 Ptas. de todos los empleados.

```
SQL> SELECT apellido, fecha_alta, salario, salario + 100000 FROM empleados;

APELLIDO FECHA_AL    SALARIO SALARIO+100000
-------- -------- ---------- --------------
REY      17/11/81     600000         700000
GARRIDO  01/05/81     385000         485000
MARTINEZ 09/06/81     245000         345000
ALONSO   20/02/81     140000         240000
LOPEZ    08/05/81     135000         235000
MARTIN   28/09/81     150000         250000
CALVO    08/09/81     180000         280000
GIL      06/05/82     335000         435000
JIMENEZ  24/03/83     140000         240000

9 filas seleccionadas.
```

Para probar las consultas en Access seguiremos los siguientes pasos:

Abrir la base de datos haciendo doble clic sobre ella. Aparece la ventana de la Base de datos, a continuación elegimos el Objeto *Consultas* y doble clic en *Crear una consulta en vista Diseño.* Ver Figura1.

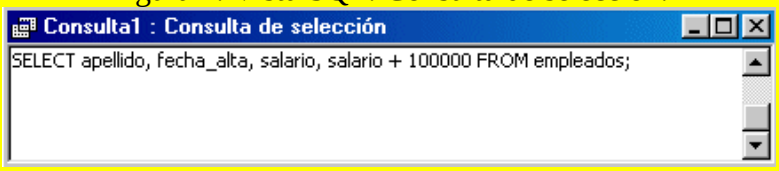
Figura1. Vista de la base de datos. Crear una consulta.



Cerramos la ventana *Mostrar Tabla*, a continuación abrimos la *vista SQL* pulsando al botón 

de la barra de herramientas. Aparece la ventana para consultas SQL, en esa ventana escribimos la orden SELECT. Ver Figura 2:

Figura 2. Vista SQL. Consulta de selección.

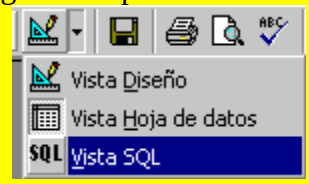


Para ver el resultado de la consulta pulsamos el botón *Ejecutar consulta*  de la barra de herramientas, o al botón *Vista hoja de datos*  , aparece el resultado de la SELECT, en la vista de hoja de datos. Ver Figura 3:

Figura 3. Vista de hoja de datos. Resultado de la consulta.

Para volver a la vista SQL desplegamos el botón *Vista* de la barra de herramientas y elegimos *Vista SQL*. Ver figura 4.

Figura 4. Opciones de vistas.



Si pulsamos el botón *Vista diseño*  , sin desplegar la lista, aparece la vista de diseño de la consulta, ver Figura 5. Esta es otra forma de realizar consultas sobre una tabla, sólo basta con seleccionar el campo y arrastrarlo a las columnas inferiores.

Figura 5. Vista diseño de consultas.

En este curso todas las consultas las realizaremos en modo *vista SQL*.

Bueno pues si hemos entrado en la *vista de diseño* y luego volvemos a la *vista SQL* observamos que la SELECT ha cambiado automáticamente y aparece:

```
SELECT empleados.APELLIDO, empleados.FECHA_ALTA,
empleados.SALARIO, salario+100000 AS Expr1
FROM empleados;
```

Vemos que cada columna va acompañada del nombre de la tabla y Access además asigna un alias a las expresiones (un nombre asociado para identificarlas). Estos cambios los hace de forma automática, solo si cambiamos de la *Vista diseño* a la *Vista SQL*, o viceversa.

Si pulsamos al botón *Guardar*  Access guarda la consulta para una posterior utilización. Al guardarla la damos un nombre.

Constantes.

En SQL podemos utilizar los siguientes tipos de constantes:

Constantes numéricas.

Construidas mediante una cadena de dígitos que puede llevar un punto decimal, y que pueden ir precedidos por un signo + ó -. (Ej.: -2454.67)
También se pueden expresar constantes numéricas empleado el formato de coma flotante. (Ej.: 34.345E-8).

Constantes de cadena.

Consisten en una cadena de caracteres encerrada entre comillas simples. (Ej.: 'Hola Mundo').

En Access las constantes de cadena se pueden definir indistintamente utilizando la comilla simple o la doble. .

Constantes de fecha.

En realidad las constantes de fecha, en Oracle y otros productos que soportan este tipo, se escriben como constantes de cadena sobre las cuales se aplicarán las correspondientes funciones de conversión (ver TO_DATE en el epígrafe *de funciones de conversión* de este mismo capítulo) o bien, el gestor de la base de datos realizará una conversión automática de tipo. (Ej.: '27-SEP-1997').

En Access las constantes de fecha se definen entre el carácter #fecha#. Por ejemplo #08-SEP-81#.

Existe una gran cantidad de formatos aplicables a estas constantes (americano, europeo, japonés, etcétera) . Algunos productos como Oracle pueden trabajar también con FECHA Y HORA en distintos formatos.

Operadores aritméticos.

Se emplean para realizar cálculos. Son los ya conocidos: (+ , - , * , /).
Devuelven un valor numérico como resultado de realizar los cálculos indicados.

Algunos de ellos se pueden utilizar **también con fechas**:

f1 - f2 Devuelve el número de días que hay entre las fechas *f1* y *f2*.

f + n Devuelve una fecha que es el resultado de sumar *n* días a la fecha *f*.

f – n Devuelve una fecha que es el resultado de restar *n* días a la fecha *f*.

Operadores de concatenación:

Para unir dos o más cadenas se utiliza el operador de concatenación ||

Ej.: 'buenos' || 'días' daría como resultado 'buenosdias'

En Access para concatenar cadenas se utiliza el signo +, no reconoce ||

Operadores de comparación:

Igual	=
Distinto	!= En Access se utiliza <>
Menor que	<
Mayor que	>
Menor o igual	<=
Mayor o igual	>=
Otros operadores	IS NULL, BETWEN, LIKE, IN, etcétera

Las expresiones formadas con operadores de comparación dan como resultado un valor de tipo *verdadero*/*falso* (*true/false).*

Ejemplos:

La expresión: APELLIDO = 'JIMENEZ' será verdadera (*true*) en el caso de que el valor de la columna APELLIDO (suponemos que se trata de una columna) sea 'JIMENEZ' y falsa (*false*) en caso contrario.

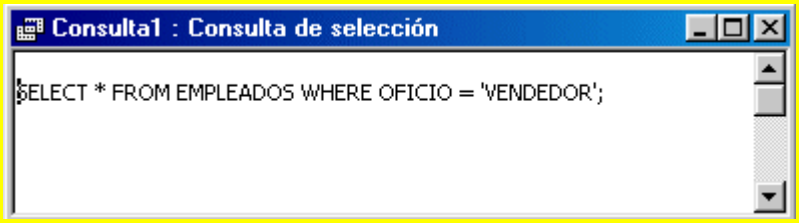
La expresión: SALARIO > 300000 será verdadera (*true*) en el caso de que SALARIO tenga un valor superior a 300000 y falsa (*false*) en caso contrario.

Estos operadores de comparación se utilizan fundamentalmente para construir condiciones de búsqueda en la base de datos. De esta forma se seleccionarán aquellas filas que cumplan la condición especificada (aquellas filas para las que el valor de la expresión sea *true*). Por ejemplo, el siguiente comando seleccionará todas las filas de la tabla empleados que en la columna OFICIO aparezca el valor 'VENDEDOR'.

```
SQL> SELECT * FROM EMPLEADOS WHERE OFICIO = 'VENDEDOR';

    EMP_NO APELLIDO OFICIO       DIRECTOR FECHA_AL    SALARIO   COMISION     DEP_NO
---------- -------- ---------- ---------- -------- ---------- ---------- ----------
      7499 ALONSO   VENDEDOR         7698 20/02/81     140000      40000         30
      7654 MARTIN   VENDEDOR         7698 28/09/81     150000     160000         30
      7844 CALVO    VENDEDOR         7698 08/09/81     180000          0         30
```

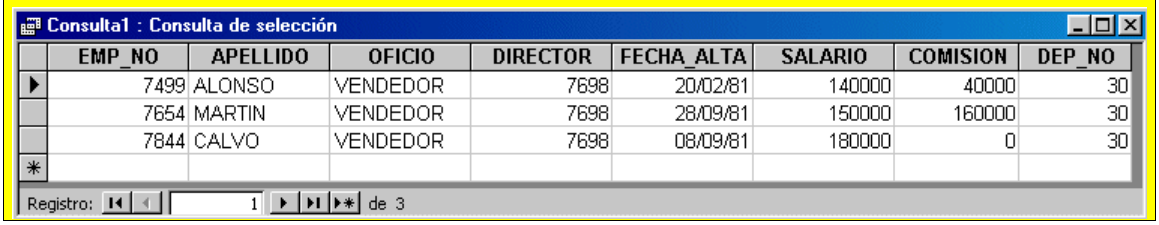
Para probarlo en Access, desde la Vista SQL, borrar la sentencia anterior y copiar la nueva sentencia SELECT, ver figura 6:

Figura 6. Vista SQL. Nueva consulta.



Pulsar el botón *Ejecutar consulta*  o al botón *Vista hoja de datos*  para ver el resultado. Ver Figura 7.

Figura 7. Vista hoja de datos. Nueva consulta.



Nota sobre la utilización de valores nulos

En SQL la ausencia de valor se expresa como valor nulo (NULL). Esta ausencia de valor o valor nulo **no equivale** en modo alguno al valor 0.

Cualquier expresión aritmética que contenga algún valor nulo retornará un valor nulo.

Así, por ejemplo, si intentamos visualizar la expresión formada por las columnas SALARIO + COMISION de la tabla empleados la salida será similar a la siguiente:

```
SQL>  SELECT APELLIDO, SALARIO, COMISION, SALARIO + COMISION
       FROM EMPLEADOS;

APELLIDO    SALARIO   COMISION SALARIO+COMISION
-------- ---------- ---------- ----------------
REY          600000
GARRIDO      385000
MARTINEZ     245000
```

```
ALONSO       140000      40000           180000
LOPEZ        135000
MARTIN       150000     160000           310000
CALVO        180000          0           180000
GIL          335000
JIMENEZ      140000

9 filas seleccionadas.
```

Para probarlo en Access hacer lo mismo que en el caso anterior. Desde la Vista SQL, borrar la sentencia anterior y copiar la nueva sentencia SELECT.

En el ejemplo anterior observamos que la expresión SALARIO + COMISION retornará un valor nulo siempre que alguno de los valores sea nulo incluso aunque el otro no lo sea. También podemos observar que el valor 0 en la comisión retorna el valor calculado de la expresión.

En SQL un valor nulo ni siquiera es igual a otro valor nulo tal como podemos apreciar en el siguiente ejemplo:

```
SQL> SELECT * FROM EMPLEADOS WHERE COMISION = NULL;

ninguna fila seleccionada
```

Probar la SELECT en Access desde la vista SQL.

La explicación es que un valor nulo es indeterminado, y por tanto, no es igual ni distinto de otro valor nulo.

Cuando queremos comprobar si un valor es nulo emplearemos el operador **IS NULL** (o IS NOT NULL para comprobar que es distinto de nulo):

```
SQL> SELECT * FROM EMPLEADOS WHERE COMISION IS NULL;

    EMP_NO APELLIDO OFICIO       DIRECTOR FECHA_AL    SALARIO   COMISION     DEP_NO
---------- -------- ---------- ---------- -------- ---------- ---------- ----------
      7839 REY      PRESIDENTE            17/11/81     600000                    10
      7698 GARRIDO  DIRECTOR         7839 01/05/81     385000                    30
      7782 MARTINEZ DIRECTOR         7839 09/06/81     245000                    10
      7521 LOPEZ    EMPLEADO         7782 08/05/81     135000                    10
      7876 GIL      ANALISTA         7782 06/05/82     335000                    20
      7900 JIMENEZ  EMPLEADO         7782 24/03/83     140000                    20

6 filas seleccionadas.
```

Probar la SELECT en Access desde la vista SQL.

Como acabamos de ver los valores nulos en muchas ocasiones pueden representar un problema, especialmente en columnas que contienen valores numéricos. Para evitar estos problemas se suele utilizar:

- La restricción NOT NULL (es una orden de definición de datos) que impide que se incluyan valores nulos en una columna.
- La función NVL (que veremos en detalle más adelante) que se utiliza para devolver un valor determinado en el caso de que el valor del argumento sea nulo. Por ejemplo NVL(COMISION, 0) retornará 0 cuando el valor de comisión sea nulo. (En Access se llama NZ)

Operadores logicos: AND, OR y NOT.

Ya hemos indicado que los operadores de comparación devuelven un valor lógicos de tipo verdadero/falso (*true/false*). En ocasiones se necesita trabajar con varias expresiones de comparación (por ejemplo cuando queremos formar una condición búsqueda que cumpla dos condiciones, etcétera) en estos casos debemos recurrir a los operadores lógicos AND, OR y NOT .

Supongamos que queremos consultar los empleados cuyo OFICIO = 'VENDEDOR' **y** que además su SALARIO > 150000. En este caso emplearemos el operador lógico **AND**. Este operador **devolverá el valor *true* cuando los dos operandos o expresiones son verdaderas**. Simplificando podemos decir que se utiliza cuando queremos que se cumplan las dos condiciones.

Ejemplo:

```
SQL> SELECT * FROM EMPLEADOS WHERE OFICIO = 'VENDEDOR' AND SALARIO > 150000;

    EMP_NO APELLIDO OFICIO       DIRECTOR FECHA_AL    SALARIO   COMISION     DEP_NO
---------- -------- ---------- ---------- -------- ---------- ---------- ----------
      7844 CALVO    VENDEDOR         7698 08/09/81     180000          0         30
```

Probar la SELECT en Access desde la vista SQL.

Cuando lo que queremos es buscar filas que cumplan **alguna** de las condiciones que se indican emplearemos el operador **OR**. Este operador **devolverá el valor *true* cuando alguno de los dos operandos o expresiones es verdadero** (cuando se cumple la primera condición, o la segunda o ambas).

Ejemplo:

```
SQL> SELECT * FROM EMPLEADOS WHERE OFICIO = 'VENDEDOR' OR SALARIO > 150000;

    EMP_NO APELLIDO OFICIO       DIRECTOR FECHA_AL    SALARIO   COMISION     DEP_NO
---------- -------- ---------- ---------- -------- ---------- ---------- ----------
      7839 REY      PRESIDENTE            17/11/81     600000                    10
      7698 GARRIDO  DIRECTOR         7839 01/05/81     385000                    30
      7782 MARTINEZ DIRECTOR         7839 09/06/81     245000                    10
      7499 ALONSO   VENDEDOR         7698 20/02/81     140000      40000         30
      7654 MARTIN   VENDEDOR         7698 28/09/81     150000     160000         30
      7844 CALVO    VENDEDOR         7698 08/09/81     180000          0         30
      7876 GIL      ANALISTA         7782 06/05/82     335000                    20
```

Probar la SELECT en Access desde la vista SQL.

El operador NOT se utiliza para cambiar el valor devuelto por una expresión lógica o de comparación, tal como se ilustra en el siguiente ejemplo:

```
SQL> SELECT * FROM EMPLEADOS WHERE NOT OFICIO = 'VENDEDOR';

    EMP_NO APELLIDO OFICIO       DIRECTOR FECHA_AL    SALARIO   COMISION     DEP_NO
---------- -------- ---------- ---------- -------- ---------- ---------- ----------
      7839 REY      PRESIDENTE            17/11/81     600000                    10
      7698 GARRIDO  DIRECTOR         7839 01/05/81     385000                    30
      7782 MARTINEZ DIRECTOR         7839 09/06/81     245000                    10
      7521 LOPEZ    EMPLEADO         7782 08/05/81     135000                    10
      7876 GIL      ANALISTA         7782 06/05/82     335000                    20
      7900 JIMENEZ  EMPLEADO         7782 24/03/83     140000                    20
```

Probar la SELECT en Access desde la vista SQL.

Observamos en el ejemplo anterior que han sido seleccionadas aquellas filas en las que **no** se cumple la condición de que el oficio sea vendedor.

Podemos formar expresiones lógicas en las que intervengan varios operadores lógicos de manera similar a como se haría con expresiones aritméticas en las que intervienen varios operadores aritméticos.

Ejemplos:

```
SQL> SELECT * FROM EMPLEADOS WHERE NOT OFICIO = 'VENDEDOR' AND SALARIO >
150000;

    EMP_NO APELLIDO OFICIO       DIRECTOR FECHA_AL    SALARIO   COMISION     DEP_NO
---------- -------- ---------- ---------- -------- ---------- ---------- ----------
      7839 REY      PRESIDENTE            17/11/81     600000                    10
      7698 GARRIDO  DIRECTOR         7839 01/05/81     385000                    30
      7782 MARTINEZ DIRECTOR         7839 09/06/81     245000                    10
      7876 GIL      ANALISTA         7782 06/05/82     335000                    20


SQL>  SELECT * FROM EMPLEADOS WHERE OFICIO = 'VENDEDOR' AND SALARIO > 150000
OR DEP_NO = 20;

    EMP_NO APELLIDO OFICIO       DIRECTOR FECHA_AL    SALARIO   COMISION     DEP_NO
---------- -------- ---------- ---------- -------- ---------- ---------- ----------
      7844 CALVO    VENDEDOR         7698 08/09/81     180000          0         30
      7876 GIL      ANALISTA         7782 06/05/82     335000                    20
      7900 JIMENEZ  EMPLEADO         7782 24/03/83     140000                    20
```

Probar la SELECT en Access desde la vista SQL.

En todo caso deberemos tener en cuanta la **prioridad o precedencia del operador** ya que puede afectar al resultado de la operación.

A continuación se detallan las tablas de valores de los operadores lógicos NOT, AND y OR, teniendo en cuenta todos los posibles valores incluida la ausencia de valor (NULL).

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Podemos establecer:

- El operador AND devolverá *true* cuando los dos operandos sean verdaderos, y *false* cuando algún operando sea falso; en el caso de que ambos operandos tengan valor Null devolverá Null y cuando ningún operando es False y algún operando es Null también devolverá Null.
- El operador OR devolverá *true* cuando alguno de los operandos sea verdadero (con independencia de que el otro sea verdadero, falso o nulo); *false* cuando los dos operandos sean falsos; y *null* en los demás casos (cuando ningún operando sea verdadero y alguno sea nulo).
- El operador NOT devuelve *true* cuando el operando es falso, y *false* cuando el operando es *true*; cuando el operando es nulo devuelve *null*.

Precedencia o prioridad en los operadores.

El orden de precedencia o prioridad de los operadores determina el orden de evaluación de los operandos de una expresión.

Por ejemplo, la siguiente expresión: `12 + 24 / 6`

Dará como resultado `16`

Ya que la división se realizará primero.

La tabla siguiente muestra los operadores disponibles agrupados y ordenados de mayor a menor por su orden de precedencia.

Prioridad	Operador	Operacion
1º	**, NOT	exponenciación, negación
2º	*, /	multiplicación, división
3º	+, -, ||	suma, resta, concatenación. Para concatenar Access se utiliza el signo +.
4º	= , != , < , > , <= , >= , IS NULL , LIKE , BETWEEN , IN	Comparación. En Access la comparación != es <>
5º	AND	conjunción
6º	OR	inclusión

Esta es la prioridad establecida por defecto. Se puede cambiar utilizando paréntesis.

En la expresión anterior, si queremos que la suma se realice antes que la división, lo indicaremos:

```
(12 + 24) / 6
```

En este caso el resultado será: `6`

Los operadores que se encuentran en el mismo grupo tienen la misma precedencia. En estos casos no se garantiza el orden de evaluación. Si queremos que se evalúen en algún orden concreto deberemos utilizar paréntesis.

NOTA: un error relativamente frecuente consiste en utilizar expresiones del tipo:

100000 >= SALARIO <= 200000

Este tipo de expresiones es ilegal y provocará un error ya que al evaluar la primera parte de la expresión se sustituirá por un valor lógico de tipo *true/false* y este resultado no puede compararse con un valor numérico. La expresión correcta sería:

SALARIO BETWEEN 100000 AND 200000

O bien:

SALARIO >= 100000 AND SALARIO <= 200000

Ejemplos de expresiones:

`SALARIO + COMISION ->` Devuelve un valor numérico como resultado de sumar al salario del empleado la comisión correspondiente.

`EMPLEADOS.DPT_NO = DEPARTAMENTOS.DPT_NO ->` Devuelve verdadero o falso dependiendo de que el número de departamento del empleado seleccionado coincida con el número de departamento del departamento seleccionado.

`FECHA_AL BETWEEN '01/01/80' AND '01/10/82'->` Devuelve verdadero si la fecha de alta del empleado seleccionado se encuentra entre las dos fechas especificadas.

`COMISION IS NULL ->` dará como resultado verdadero si la comisión del empleado seleccionado no tiene ningún valor.

Funciones predefinidas.

En SQL disponemos de funciones predefinidas que devuelven un valor en función de un argumento que se pasa en la llamada. Algunas de estas funciones no se pueden utilizar en Access o se llaman de otra manera.

Ejemplos:

Llamada a la función	**Valor devuelto**	**Explicación.**
ABS(-5)	5	el valor absoluto del número.
ROUND(8.66, 0)	9	el número redondeando a cero decimales
TRUC(8.66, 0). En Acces esta función no se encuentra.	8	el número truncando los decimales.
SIGN(8.88). En Acces se llama SGN()	1	ya que el número es mayor de 0
INITCAP('hola'). En Acces esta función no se encuentra.	'Hola'	cadena con el primer carácter en mayúsculas.
UPPER('hola'). En Acces se llama UCASE()	'HOLA'	cadena en mayúsculas
SUBSTR('hola', 2, 2). En Acces se llama MID()	'ol'	subcadena tomando dos posiciones desde la posición 2.
SYSDATE. En Acces se llama DATE()	01/02/00	la fecha del sistema
USER. En Acces se llama CURRENTUSER()	CURSOSQL	el nombre del usuario en nuestro caso el que se indica.

Cabe subrayar que la función no modifica el valor del argumento sino que devuelve un valor creado a partir del argumento que se le pasa en la llamada.

A continuación se indican las funciones predefinidas más utilizadas:

Funciones numéricas o aritméticas:

Función	Valor que devuelve.
ABS(*num*)	Valor absoluto de *num*.
MOD(*num1, num2*). **En Access se pone: *num1 MOD num2***	Resto de la división entera.
POWER(*num1, num2*). **En Access se pone: *num1* ^ *num2*.**	Devuelve *num1* elevado a *num2*.
ROUND(*num1, num2*)	Devuelve *num1* redondeado a *num2* decimales.
SIGN(*num*). En Access se llama SGN().	Si *num* < 0 devuelve -1, si *num* = 0 devuelve 0, si *num* > 0 devuelve 1.
SQRT(*num*). En Access se llama SQR.	Raíz cuadrada de *num*
TRUNC(*num1, num2*). **No se encuentra en Access**	Devuelve *num1* truncado a *num2* decimales. Si se omite *num2*, a 0 decimales.

Funciones de caracteres:

Función	Valor que devuelve.
ASCII(*c1*). **En Access se llama ASC.**	Codigo ASCII del carácter *c1*.
CHR(*num*)	Carácter correspondiente al código indicado por *num* del juego de caracteres utilizado.
CONCAT(*c1,c2*). **Para concatenar en Acess utilizamos + o &.**	Concatena *c1* con *c2*. Es equivalente al operador ||. En Access. Pondremos c1&c2 o también c1+c2.
INITCAP(*c1*). **No se encuentra en Access**	Devuelve *c1* poniendo en mayúscula la primera letra de cada palabra de la cadena y el resto en minúsculas.

LENGTH(*c1*) **En Access se llama LEN().**	Longitud de *c1*.
LOWER(*c1*) **En Access se llama LCASE.**	La cadena *c1* en minúsculas.
UPPER(*c1*) **En Access se llama UCASE.**	La cadena *c1* en mayúsculas.
LPAD(*c1, n, c2*) **No se encuentra en Access**	c1 se visualiza con longitud n y justificado a la derecha. Si c1 < n, c2 es la cadena con la que se rellena por la izda.
RPAD(*c1, n, c2*) **No se encuentra en Access**	Igual que LPAD pero por la derecha.
LTRIM(*c1*)	Suprime blancos a la izquierda de *c1*.
RTRIM(*c1*)	Suprime blancos a la derecha de *c1*.
SUBSTR(*c1, n, m*) **En Access se llama MID()**	Devuelve una subcadena a partir de *c1* comenzando en la posición *n* tomando *m* caracteres.
TRANSLATE(*c1, c2, c3*) **No se encuentra en Access**	Devuelve la cadena *c1* con cada ocurrencia de *c2* que contenga reemplazada por *c3*.

Funciones de fecha:

Función	Valor que devuelve.
ADD_MONTHS(*f, n*) **En Access se utiliza:** **DateAdd("y",n, f). Suma n años** **DateAdd("m",n,f). Suma n meses** **DateAdd("d",n,f). Suma n días**	Incrementa *n* meses a la fecha *f*.. En el caso de Access podemos sumar años, días o meses a una fecha dada.
LAST_DAY(*f*) **No se encuentra en Access**	Último día del mes de la fecha.
MONTHS_BETWEEN (f1, f2) **En Access se utiliza:** **DateDiff("m",f1,f2). Los meses.** **DateDiff("d",f1,f2). Los días.** **DateDiff("yyyy",f1,f2). Los años**	Número de meses entre las dos fechas. El resultado puede contener decimales correspondientes a fracciones de mes. En el caso de Access podemos saber el número de días, de años y de meses.

SYSDATE. **En Access se llama DATE().**	Fecha actual. También la hora si se especifica en el formato. En Access es **NOW()** la función que visualiza la hora y la fecha.

Funciones de conversión:

Se utilizan para pasar datos de un tipo a otro: carácter a número, fecha a carácter, etcétera.

<table>
<tr><th>Función</th><th>Valor que devuelve.</th></tr>
<tr><td>TO_NUMBER(c1)
En Access se llama VAL().</td><td>Convierte una cadena a tipo numérico.</td></tr>
<tr><td>TO_CHAR(n, formato)</td><td>Devuelve un número en formato char según las siguientes especificaciones de formato:

<table>
<tr><th>Cod.</th><th>Ejemplo</th><th>Descripción</th></tr>
<tr><td>9</td><td>999</td><td>Cada 9 indica un dígito</td></tr>
<tr><td>0</td><td>099</td><td>Visualiza ceros a la izquierda</td></tr>
<tr><td>$</td><td>$99</td><td>Antepone el símbolo de $</td></tr>
<tr><td>MI</td><td>99MI</td><td>Visualiza un '-' después de un número negativo</td></tr>
<tr><td>B</td><td>B99</td><td>En lugar de 0 visualiza blancos</td></tr>
<tr><td>,</td><td>999,999</td><td>Visualiza el separador de los miles</td></tr>
<tr><td>.</td><td>999,999.99</td><td>Visualiza el punto de separación de decimales</td></tr>
</table></td></tr>
<tr><td>TO_CHAR(fecha, formato)</td><td>Devuelve cadena a partir de una fecha según las siguientes especificaciones de formato:

<table>
<tr><th>Formato</th><th>Descripción</th></tr>
<tr><td>YYYY</td><td>Cuatro dígitos en el año.</td></tr>
<tr><td>YY</td><td>Últimos dos dígitos del año (también con tres y uno)</td></tr>
<tr><td>YEAR</td><td>Año deletreado.</td></tr>
<tr><td>MM</td><td>Mes en número.</td></tr>
<tr><td>MONTH</td><td>Mes en letra.</td></tr>
<tr><td>MON</td><td>Abreviatura tres letras del mes</td></tr>
<tr><td>DD</td><td>Día del mes</td></tr>
<tr><td>DDD</td><td>Día del año</td></tr>
<tr><td>D</td><td>Día de la semana</td></tr>
<tr><td>DAY</td><td>Nombre del día de la semana</td></tr>
<tr><td>DY</td><td>Nombre del día con tres letras</td></tr>
<tr><td>WW</td><td>Semana del año</td></tr>
<tr><td>W</td><td>Semana del mes</td></tr>
<tr><td>Q</td><td>Trimestre del año</td></tr>
<tr><td>AM/PM</td><td>Formato 12 horas indicando AM o PM</td></tr>
<tr><td>HH24</td><td>Formato 24 horas</td></tr>
<tr><td>MI</td><td>Minutos</td></tr>
<tr><td>SS</td><td>Segundos</td></tr>
<tr><td>SSSS</td><td>Hora en segundos desde las 0 horas del día.</td></tr>
</table></td></tr>
</table>

Para convertir una fecha o un número a cadena Access utiliza la función **FORMAT**.

Format(fecha, formato)
Format(número, formato)

Si no ponemos formato Access devuelve la cadena correspondiente a la fecha o al número. En algunos casos utiliza los mismos formatos que ORACLE. También tiene otros formatos predefinidos como *"Long Time"*. Para representar dígitos Access utiliza el carácter #.

Ejemplos:

Format(Now(),"Long Time") Devuelve la hora actual. Hora larga. “09:00:33 p.m. “
Format(Date(), "Long Date") Devuelve la fecha actual completa: “Martes, 17 de Mayo de 2001”
Format(#21:6:51#, "h:m:s"). Devuelve “21:6:51”
Format(#21:6:51#, "hh:mm:ss AMPM"). Devuelve “09:06:51 p.m.”
Format(#12/11/99#, "dddd, d mmm yyyy"). Devuelve: “ Sábado, 11 Dic 1999”
Format(33459.46, "##,##0.000"). Devuelve: " 33,459.460".
Format(334.9, "###0.00"). Devuelve: "334,90".
Format(67, "0.00%"). Devuelve: "6700,00%".
Format("MINÚSCULAS", "<"). Devuelve "minúsculas".
Format("Ejemplito”, ">") . Devuelve "EJEMPLITO".

TO_DATE(fecha, formato) **En Acces se llama CDATE(fecha), y la convierte al tipodd/mm/aa.**	**Devuelve fecha a partir de una cadena según las especificaciones de formato indicadas antes.** Ej.: TO_DATE('27-OCT-95', 'DD-MON-YY') Ej en Access: CDate("12 mayo 96"), devuelve 12/5/96

Otras funciones:

Función	Valor que devuelve.
GREATEST(*lista de valores*) **No se encuentra en Access**	Valor más grande de una lista.
LEAST(*lista de valores*) **No se encuentra en Access**	Valor más pequeño de una lista.
NVL(*exp1, exp2*) **En Access se llama NZ(*exp1, exp2*)**	Si *exp1* es nulo devuelve *exp2*. Sino devuelve exp1.

USER **En Acces se llama CURRENTUSER()**	Devuelve el nombre del usuario actual.
UID **No se encuentra en Access**	Devuelve el número de identificación del usuario actual. Esta función, igual que la anterior, no tiene argumentos.

En los próximos temas se estudiarán todas estas funciones con diversos ejemplos y ejercicios.

Consideraciones sobre la sintaxis utilizada.

Para especificar la sintaxis el estándar SQL ANSI/ISO utiliza una notación muy precisa y completa pero presenta serios problemas didácticos especialmente para personas que se inician en este lenguaje. Por esta razón hemos utilizado para especificar los formatos una notación más sencilla y asequible primando el aspecto didáctico. A este respecto procede realizar las siguientes puntualizaciones:

- Las palabras reservadas de SQL aparecen en mayúsculas.
- Los nombres de objetos (tablas, columnas, etcétera) suelen aparecer en minúsculas.
- La notación *lista_de_elementos* especifica una lista de elementos separados por comas.
- La barra vertical (|) indica la elección entre dos elementos.
- Las llaves ({ }) indican la elección obligatoria entre varios elementos.
- Los corchetes ([ ]) encierran un elemento opcional.
- El punto y coma (;) que aparece al final de cada comando, en realidad no forma parte de la sintaxis del lenguaje SQL pero suele ser un elemento requerido por las herramientas de cliente para determinar el final del comando SQL y enviar la órden (sin el ;) al servidor.

Tema 3. CONSULTAS SENCILLAS.

Autora: Maria Teresa Miñana

Consulta de los datos.

Realizar una consulta en SQL consiste en recuperar u obtener aquellos datos que, almacenados en filas y columnas de una o varias tablas de una base de datos, cumplen unas determinadas especificaciones. Para realizar cualquier consulta se utiliza la sentencia **SELECT**.

Las primeras consultas van a ser escritas con un formato inicial de la sentencia SELECT, que se irá completando en temas siguientes.

Formato inicial de la sentencia SELECT

```
SELECT ???ALL  ???????*?????????????????????????>
         ?DISTINCT?  ?lista_de_elementos?

>??FROM lista_de_tablas ??????????????????????????>

>????????????????????????????????????????????????>
   ?WHERE condición_de_selección?

>????????????????????????????????????????????????>;
   ?ORDER BY especificaciones_para_ordenar?
```

Tablas utilizadas:

TABLA DE EMPLEADOS

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISIÓN	DEP_NO
7499	ALONSO	VENDEDOR	7698	20/02/81	140000	40000	30
7521	LOPEZ	EMPLEADO	7782	08/05/81	135000		10
7654	MARTIN	VENDEDOR	7698	28/09/81	150000	160000	30
7698	GARRIDO	DIRECTOR	7839	01/05/81	385000		30
7782	MARTINEZ	DIRECTOR	7839	09/06/81	245000		10
7839	REY	PRESIDENTE		17/11/81	600000		10
7844	CALVO	VENDEDOR	7698	08/09/81	180000	0	30
7876	GIL	ANALISTA	7782	06/05/82	335000		20
7900	JIMENEZ	EMPLEADO	7782	24/03/83	140000		20

TABLA DE DEPARTAMENTOS

DEP_NO	DNOMBRE	LOCALIDAD
10	CONTABILIDAD	BARCELONA
20	INVESTIGACION	VALENCIA

```
30 VENTAS         MADRID
40 PRODUCCION     SEVILLA
```

Consultas sencillas.

La consulta más sencilla consiste en recuperar una o varias columnas de una tabla.

```
SELECT ???ALL  ???????*?????????????????????????>
         ?DISTINCT?  ?lista_de_elementos?

>??FROM tabla ????????????????????????????????????>;
```

- ***lista_de_elementos*:** nombres de columnas o expresiones obtenidas a partir de ellas, y separadas por comas, que son seleccionadas de cada fila para conocer sus valores.

- ***:** selecciona todas las columnas de la tabla.

- **ALL:** obtiene los valores de todos los elementos seleccionados en todas las filas, aunque sean repetidos. Es la opción por defecto.

En Acces la opción ALL va seguida de * : `SELECT ALL * FROM EMPLEADOS;`

- **DISTINCT:** obtiene los valores no repetidos de todos los elementos.

- **FROM** *tabla*: indica el nombre de la tabla en la que se realiza la consulta. Si la tabla no es de la propiedad del usuario, aunque tenga permiso de acceso, deberá usarse con ***nombre_propietario.tabla***.

Alias de tabla. SQL permite asignar más de un nombre a la misma tabla, dentro de la misma consulta. Usar alias para una tabla puede es opcional cuando su finalidad consiste en simplificar su nombre original, y obligatorio en consultas cuya sintaxis lo requiera.

```
SELECT .....
  FROM empleados e                  alias de empleados → e
  ..............................
```

Recuerda: para probar las sentencias SQL en Access, desde la vista de la base de datos, elegir el objeto *Consultas* y doble clic en *Crear una consulta en vista Diseño*. Cerrar la ventana *Mostrar Tabla*, y a continuación abrir la *vista SQL* pulsando al botón  de la barra de herramientas. En Access podremos guardar todas las consultas que hagamos si elegimos la opción *Guardar Como* del menú *Archivo*. Tendremos que dar un nombre a cada consulta. Para ver el resuiltado de la consulta pulsaremos al botón *Ejecutar consulta*  o al botón *Vista hoja de datos*  .

Ejemplos.

1. Obtener todos los empleados de la tabla *empleados* con todos sus datos.

```
SQL> SELECT *   FROM empleados;          o

SQL> SELECT ALL   FROM empleados;
```

```
EMP_NO APELLIDO OFICIO      DIRECTOR FECHA_AL   SALARIO  COMISION    DEP_NO
------ -------- ---------- -------- --------- -------- --------- ---------
  7499 ALONSO   VENDEDOR       7698 20/02/81    140000     40000        30
  7521 LOPEZ    EMPLEADO       7782 08/05/81    135000                  10
  7654 MARTIN   VENDEDOR       7698 28/09/81    150000    160000        30
  7698 GARRIDO  DIRECTOR       7839 01/05/81    385000                  30
  7782 MARTINEZ DIRECTOR       7839 09/06/81    245000                  10
  7839 REY      PRESIDENTE          17/11/81    600000                  10
  7844 CALVO    VENDEDOR       7698 08/09/81    180000         0        30
  7876 GIL      ANALISTA       7782 06/05/82    335000                  20
  7900 JIMENEZ  EMPLEADO       7782 24/03/83    140000                  20
```

```
En Acces: SELECT  ALL *  FROM EMPLEADOS;
          SELECT  *  FROM EMPLEADOS;
```

2. Obtener los números de empleados, los apellidos y el número de departamento de todos los empleados de la tabla *empleados*.

```
SQL> SELECT emp_no, apellido,dep_no FROM empleados;
```

```
EMP_NO APELLIDO    DEP_NO
------ -------- ---------
  7499 ALONSO          30
  7521 LOPEZ           10
  7654 MARTIN          30
  7698 GARRIDO         30
  7782 MARTINEZ        10
  7839 REY             10
  7844 CALVO           30
  7876 GIL             20
  7900 JIMENEZ         20
```

Alias de columna. Los títulos o cabeceras que muestra la salida de una consulta para las columnas seleccionadas, se corresponden con los nombres de las columnas de las tablas. Para mejorar su legibilidad y estética se utilizan los **alias de columna**. El alias se escribe detrás de la columna, separado de ella al menos por un espacio. Si el alias comprende más de una palabra deberá ir dentro de dobles comillas.

```
SQL> SELECT emp_no "Nº Empleado", apellido, dep_no Departamento
       FROM empleados;
```

En Acces para poner un alias a una columna utilizamos la palabra *AS* seguida del nombre. El nombre no debe contener espacios:

```
SELECT DISTINCT DEP_NO AS NUM_DEPART  FROM EMPLEADOS;
SELECT EMP_NO AS NºEMPLEADO, APELLIDO, DEP_NO AS DEPARTAMENTO
FROM EMPLEADOS;
```

3. Obtener el total a cobrar por cada empleado, suponiendo que se trata de sumar a su salario la correspondiente comisión, si la tuviera.

```
SQL> SELECT apellido,salario+comision  "Importe Total"
       FROM empleados;
```

En Acces:

```
SELECT apellido,salario+comision AS  Importe_Total FROM empleados;
```

```
APELLIDO Importe Total
-------- -------------
ALONSO          180000
LOPEZ
MARTIN          310000
GARRIDO
MARTINEZ
REY
CALVO           180000
GIL
JIMENEZ
```

En esta salida, algunos empleados no llevan importe. El motivo es tener `NULL` (sin información) en su comisión, y por lo tanto no se realiza la operación suma. Para solucionarlo hemos de recurrir al uso de funciones.

Sugerencia.- **Ver `FUNCIONES (En MATERIALES)`**.

El ejemplo anterior debería resolverse con la función `NVL` que transforma la ausencia de información al valor que se le especifique, tal como sigue:

```
SQL> SELECT apellido,salario+NVL(comision,0)  "Importe Total"
       FROM empleados;
```

La nueva salida sería:

```
APELLIDO Importe Total
-------- -------------
ALONSO          180000
LOPEZ           135000
MARTIN          310000
```

```
GARRIDO         385000
MARTINEZ        245000
REY             600000
CALVO           180000
GIL             335000
JIMENEZ         140000
```

En Acces:

```
SELECT apellido,salario+NZ(comision,0)AS Importe_Total FROM empleados;
```

4. Concatenar cada empleado con su oficio mediante “es”.

```
SQL> SELECT CONCAT(CONCAT(apellido, ‘ es ‘),oficio)
       “Empleado y su oficio”
        FROM empleados;
```

```
Empleado y su oficio
---------------------
ALONSO es VENDEDOR
LOPEZ es EMPLEADO
MARTIN es VENDEDOR
GARRIDO es DIRECTOR
MARTINEZ es DIRECTOR
REY es PRESIDENTE
CALVO es VENDEDOR
GIL es ANALISTA
JIMENEZ es EMPLEADO
```

En Acces:

```
SELECT APELLIDO + '  ES ' +  OFICIO  AS  EMPLEADO_Y_SU_OFICIO
FROM EMPLEADOS;
```

5. Obtener la fecha de alta de cada empleado con el nombre del mes completo y en castellano.

```
SQL> SELECT TO_CHAR(FECHA_ALTA,’ddmonthyy’,
       ’NLS_DATE_LANGUAGE=Spanish’) “Fecha de alta”
        FROM empleados;
```

```
Fecha de alta
--------------
20febrero   81
08mayo      81
28septiembre81
01mayo      81
09junio     81
17noviembre 81
08septiembre81
06mayo      82
24marzo     83
```

En Acces:

```
SELECT Format(FECHA_ALTA,'DDMMMMYY') AS Fecha_de_alta
FROM Empleados;
```

Tabla DUAL.

En SQL, toda petición de datos se escribe mediante una consulta, y cualquier consulta debe realizarse sobre una tabla. Cuando la consulta consiste en obtener los valores de una determinada función aplicada a una constante, no a una columna de una tabla, o en acceder a la fecha del sistema, la tabla que se utiliza en la cláusula FROM es la **tabla DUAL**, disponible a todo usuario.

Ejemplo. Calcular la quinta potencia de 5.

```
SQL> SELECT POWER(5,5) " 5 Elevado a 5"
       FROM DUAL;

5 Elevado a 5
--------------
         3125
```

En Access no existe la tabla DUAL, sin embargo se pueden realizar operaciones para probar funciones utilizando una tabla.

```
SELECT DISTINCT(5^5) AS 5_Elevado_a_5 FROM EMPLEADOS;
```

Condiciones de selección.

Para seleccionar las filas de la tabla sobre las que realizar una consulta, la cláusula **WHERE** permite incorporar una ***condición de selección*** a la sentencia SELECT.

Formato de consulta con condición de selección

```
SELECT ???ALL  ???????*?????????????????????????>
         ?DISTINCT?  ?lista_de_elementos?

>??FROM lista_de_tablas ????????????????????????>

>??????????????????????????????????????????????>;
   ?WHERE condición_de_selección?
```

Condición_de_selección: expresión formada por columnas de la tabla, constantes, funciones, operadores de comparación y operadores lógicos, que deberá ser cierta para que una fila de la tabla sea seleccionada como parte de la salida obtenida por la consulta.

. Operadores de comparación.
- aritméticos: `=, >, <`
- de caracteres: `LIKE`, máscaras (`%, _`)
- lógicos: `IN, BETWEEN`

. Operadores lógicos-booleanos: `AND,OR,NOT` . Permiten construir condiciones de selección compuestas. El uso de paréntesis ayuda a escribir correctamente, a mejorar la legibilidad de las condiciones compuestas y a establecer prioridades de evaluación para los operadores.

Ejemplos.

1. Obtener la lista de los empleados vendedores, con su nombre, salario y comisión**.**

```
SQL> SELECT apellido,salario,comision
       FROM empleados
        WHERE UPPER(oficio)='VENDEDOR';

APELLIDO   SALARIO  COMISION
-------- --------- ---------
ALONSO      140000     40000
MARTIN      150000    160000
CALVO       180000         0
```

UPPER*(expresión_alfabética)* obtiene la *expresión_alfabética* en mayúsculas.

En Acces:
```
SELECT APELLIDO,SALARIO,COMISION FROM EMPLEADOS
WHERE UCASE(OFICIO) = 'VENDEDOR';
```

2. Seleccionar aquellos empleados cuyo apellido empiece por "M" y su salario esté comprendido entre 100.000 y 200.000 pesetas. Visualizar su número de empleado, apellido y departamento.

```
SQL> SELECT emp_no "Nº Empleado" ,apellido,dep_no Departamento
       FROM empleados
       WHERE (apellido LIKE 'M%') AND
       (salario >=100000 AND salario<= 200000);

Nº Empleado APELLIDO DEPARTAMENTO
----------- -------- ------------
       7654 MARTIN             30
```

En Acces:

```
SELECT EMP_NO AS NºEMPLEADO ,APELLIDO, DEP_NO AS DEPARTAMENTO
FROM EMPLEADOS WHERE (APELLIDO LIKE 'M*') AND (SALARIO >=100000 AND
SALARIO<= 200000);
```

El operador **LIKE** usado con **%** indica que la comparación del apellido se realiza sólo en el primer carácter, que debe ser "**M**".El % sustituye al resto de los caracteres.

En Access NO se utiliza el carácter %, se utilizan los siguientes caracteres comodín:

- El signo de **interrogación** (?) para sustituir un carácter por cualquiera en esa posición. Por ejemplo ?a busca aquellos valores que empiecen por cualquier letra y la segunda sea una "a".
- El **asterisco (*)** para representar cualquier número de caracteres situados en la misma posición que el asterisco.

Por ejemplo

Criterio

Resultado

`LIKE 'B*'`

Departamentos cuyo nombre de localidad empieza por B

`LIKE 'M????D'`

Departamentos cuya localidad empieza por M, seguido de 4 letras cualquiera y terminan en D. (MADRID, por ejemplo)

`LIKE '?A*'`

Departamentos cuya localidad empieza por cualquier letra, le sigue una A u luego cualquier número de caracteres.

`LIKE 'B*O'`

Departamentos cuya localidad empieza por B y termina en O.

`LIKE '*O'`

Departamentos cuya localidad termina en O.

`LIKE '*A*'`

Todos los departamento que contengan una A en el nombre de la localidad.

El operador **BETWEEN** comprueba si una expresión toma valores dentro del intervalo que le acompaña.

El mismo ejemplo podría haberse escrito:

```
SQL> SELECT emp_no "Nº Empleado" ,apellido,dep_no Departamento
```

```
        FROM empleados
        WHERE (apellido LIKE 'M%') AND
        salario BETWEEN 100000 AND 200000;
```

En Acces:

```
SELECT EMP_NO AS NºEMPLEADO ,APELLIDO, DEP_NO AS DEPARTAMENTO
FROM EMPLEADOS WHERE (APELLIDO LIKE 'M*') AND
Salario BETWEEN 100000 AND 200000;;
```

3. Seleccionar aquellos empleados cuyo apellido incluya una "Z" en el segundo carácter.

```
SQL> SELECT emp_no "Nº Empleado" ,apellido,dep_no Departamento
       FROM empleados
       WHERE (apellido LIKE '_Z%') ;

ninguna fila seleccionada
```

El operador **LIKE** usado con '_' indica que ocupa la posición de un carácter.

En Acces:

```
SELECT EMP_NO AS NºEMPLEADO ,APELLIDO, DEP_NO AS DEPARTAMENTO
FROM EMPLEADOS WHERE (APELLIDO LIKE '?Z*');
```

4. Seleccionar los empleados existentes en los departamentos 10 y 30.

```
SQL>SELECT emp_no "Nº Empleado",apellido,dep_no Departamento
      FROM empleados
      WHERE dep_no=10 OR dep_no=30;
```

o

```
SQL>SELECT emp_no "Nº Empleado",apellido,dep_no Departamento
       FROM empleados
       WHERE dep_no IN(10,30);

Nº Empleado APELLIDO DEPARTAMENTO
----------- -------- ------------
       7499 ALONSO             30
       7521 LOPEZ              10
       7654 MARTIN             30
       7698 GARRIDO            30
       7782 MARTINEZ           10
       7839 REY                10
       7844 CALVO              30
```

El operador **IN** comprueba si una determinada expresión toma alguno de los valores indicados entre paréntesis.

En Acces:

```
SELECT EMP_NO AS NºEMPLEADO ,APELLIDO, DEP_NO AS DEPARTAMENTO
FROM EMPLEADOS WHERE Dep_no=10 OR Dep_no=30;

SELECT EMP_NO AS NºEMPLEADO ,APELLIDO, DEP_NO AS DEPARTAMENTO
FROM EMPLEADOS WHERE Dep_no IN(10,30);
```

Ordenación.

Para obtener la salida de una consulta clasificada por algún criterio o especificación, la sentencia `SELECT` dispone de la cláusula **`ORDER BY`** para ordenar.

Formato de consulta con ordenación

```
SELECT ???ALL  ???????*?????????????????????????>
         ?DISTINCT?  ?lista_de_elementos?

>??FROM lista_de_tablas ?????????????????????????>

>??????????????????????????????????????????????????>
   ?WHERE condición_de_selección?

>??????????????????????????????????????????????????>;
   ?ORDER BY especificación_para_ordenar?
```

***Especificación_para_ordenar* :** lista de columnas o expresiones obtenidas a partir de ellas, separadas por comas, y cada una de ellas con indicación del tipo de ordenación.

Tipo de ordenación: **`ASC`** (ascendente) o **`DESC`** (descendente). Por omisión es `ASC`.

Los nombres de columnas de la *especificación_para_ordenar* pueden ser sustituidos por el número de orden que ocupan en la tabla.

Si la *especificación_para_ordenar* contiene más de una columna o expresión, el orden en que se realizan las clasificaciones es de izquierda a derecha.

Ejemplos.

1. Obtener relación alfabética de todos los empleados con todos sus datos.

```
SQL>SELECT * FROM empleados
      ORDER BY apellido;
```

2. Obtener clasificación alfabética de empleados por departamentos.

```
SQL>SELECT * FROM empleados
      ORDER BY dep_no, apellido;
```

o

```
SQL>SELECT * FROM empleados
      ORDER BY 8,2;
```

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISION	DEP_NO
7499	ALONSO	VENDEDOR	7698	20/02/81	140000	40000	30
7844	CALVO	VENDEDOR	7698	08/09/81	180000	0	30
7698	GARRIDO	DIRECTOR	7839	01/05/81	385000		30
7876	GIL	ANALISTA	7782	06/05/82	335000		20
7900	JIMENEZ	EMPLEADO	7782	24/03/83	140000		20
7521	LOPEZ	EMPLEADO	7782	08/05/81	135000		10
7654	MARTIN	VENDEDOR	7698	28/09/81	150000	160000	30

3. Obtener los datos de los empleados clasificados por oficios y en orden descendente de salarios.

```
SQL>SELECT * FROM empleados
      ORDER BY oficio,salario DESC;
```

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISION	DEP_NO
7876	GIL	ANALISTA	7782	06/05/82	335000		20
7698	GARRIDO	DIRECTOR	7839	01/05/81	385000		30
7782	MARTINEZ	DIRECTOR	7839	09/06/81	245000		10
7900	JIMENEZ	EMPLEADO	7782	24/03/83	140000		20
7521	LOPEZ	EMPLEADO	7782	08/05/81	135000		10
7839	REY	PRESIDENTE		17/11/81	600000		10
7844	CALVO	VENDEDOR	7698	08/09/81	180000	0	30
7654	MARTIN	VENDEDOR	7698	28/09/81	150000	160000	30
7499	ALONSO	VENDEDOR	7698	20/02/81	140000	40000	30

Tema 4. AGRUPAMIENTO Y FUNCIONES DE COLUMNA.

Autora: Maria Teresa Miñana

SQL permite agrupar las filas de una tabla, seleccionadas en una consulta, y obtener salidas, calculadas a partir de los grupos formados.

El **criterio para agrupar** suele ser una de las columnas de la tabla. Si no se especifica ninguno, las filas de la tabla, seleccionadas en la consulta, forman un grupo único.

Las salidas obtenidas son los resultados de aplicar **funciones de columna** a los grupos de filas.

Funciones de columna

FUNCIONES	DESCRIPCIÓN DE LA FUNCIÓN
SUM(*expresión/DISTINCT nombre de columna*)	Calcula la suma de valores de la expresión o de la columna indicada dentro del paréntesis, teniendo en cuenta que la cláusula DISTINCT omite valores repetidos.
AVG(*expresión/DISTINCT nombre_ de_columna*)	Calcula el valor medio de la expresión que se indique dentro del paréntesis, teniendo en cuenta que los valores NULL no son incluidos.
MIN(*expresión*)	Devuelve el valor mínimo de la expresión que le acompaña.
MAX(*expresión*)	Devuelve el valor máximo de la expresión que le acompaña.
COUNT(*nombre_ columna/DISTINCT nombre_de_ columna)*	Cuenta el número de valores de datos que hay en una columna, sin incluir los valores NULL
COUNT(**)*	Cuenta todas las filas de la tabla, sin considerar que en algunas columnas existan valores NULL.
STDDEV(*expresión*) **En Access se llama STDEV(*expresión*)**	Calcula la desviación típica para los valores de la expresión.
VARIANCE(*expresión*) **En Access se llama VAR(*expresión*)**	Calcula la varianza para los valores de la expresión.

En Access la *expresión* puede incluir un nombre de campo de una tabla o una función definida pero NO puede incluir ninguna de las funciones de columna.
Para resolver las consultas que utilizan funciones anidadas realizaremos varias consultas como se verá más adelante.

Los valores NULL no intervienen en las funciones de conjunto o de columna.

Formato de consulta con funciones de columna (sin criterio de agrupación)

```
SELECT  funciones_de_columna
  FROM tabla
  [WHERE  condición];
```

Ejemplos.

1. Obtener la masa salarial mensual de todos los empleados.

```
SQL>SELECT SUM(salario)
      FROM empleados;

SUM(SALARIO)
------------
     2310000
```

2. Obtener los salarios máximo, mínimo y la diferencia existente entre ambos.

```
SQL>SELECT MAX(salario),MIN(salario),MAX(salario)-MIN(salario)
      FROM empleados;

MAX(SALARIO) MIN(SALARIO) MAX(SALARIO)-MIN(SALARIO)
------------ ------------ -------------------------
      600000       135000                    465000
```

3. Obtener la fecha de alta más reciente.

```
SQL>SELECT MAX(fecha_alta)"Fecha alta" FROM empleados;

Fecha Al
--------
24/03/83
```

```
En Acces el alias no se puede llamar como el nombre de la columna:

      SELECT MAX(Fecha_alta) as FECH_ALTA FROM EMPLEADOS;
```

4. Calcular el salario medio de los empleados.

```
SQL>SELECT AVG(salario) "Salario medio" FROM empleados;

Salario medio
-------------
    256666,67
```

```
En Access:
      SELECT AVG(Salario) As Salario_Medio  FROM Empleados;
```

A veces no da el mismo resultado que:

```
SQL>SELECT SUM(salario)/COUNT(*) FROM empleados;
```

porque `COUNT` cuenta el número de valores de datos que hay en una columna, sin incluir los valores `NULL`, y por el contrario, `COUNT(*)` cuenta todas las filas de la tabla, sin considerar que en algunas columnas existan valores `NULL`.

Por lo general, las funciones de columna se utilizan sobre más de un grupo de filas. La cláusula **`GROUP BY`** establece el criterio de agrupación.

Formato de consulta con funciones de columna (con criterio de agrupación)

```
SELECT funciones_de_columna ??????????????????>

>??FROM lista_de_tablas ??????????????????????>

>??????????????????????????????????????????????>
   ?WHERE condición_de_selección?


>??????????????????????????????????????????????>
   ?GROUP BY lista_de_columnas_para_agrupar?

>??????????????????????????????????????????????>;
   ?ORDER BY especificaciones_para_ordenar?
```

Ejemplos.

1. Obtener los salarios medios por departamento.

```
SQL>SELECT dep_no Departamento, AVG(salario) "Salario Medio"
     FROM empleados
     GROUP BY dep_no;

DEPARTAMENTO Salario Medio
------------ -------------
          10     326666,67
          20        237500
          30        213750
```

```
En Acess:
      SELECT dep_no as Departamento, AVG(salario) as  Salario_Medio
      FROM empleados GROUP BY dep_no;
```

2. Obtener cuántos empleados hay en cada departamento.

```
SQL>SELECT dep_no Departamento, COUNT(*) "Nº de Empleados"
     FROM empleados
     GROUP BY dep_no;
```

```
DEPARTAMENTO Nº de Empleados
------------ ---------------
          10               3
          20               2
          30               4
```

```
En Acess:
      SELECT Dep_no As Departamento, COUNT(*) as  NUM_Empleados
      FROM empleados GROUP BY dep_no;
```

3. Obtener el empleado que mayor salario tiene dentro de cada oficio, excluyendo al presidente.

```
SQL>SELECT MAX(salario), oficio FROM empleados
      WHERE UPPER(oficio) <>'PRESIDENTE'
      GROUP BY oficio;

MAX(SALARIO) OFICIO
------------ ---------
      335000 ANALISTA
      385000 DIRECTOR
      140000 EMPLEADO
      180000 VENDEDOR
```

```
En Acess:
           SELECT MAX(salario), oficio FROM empleados
           WHERE UCASE(oficio) <>'PRESIDENTE'
           GROUP BY oficio;
```

Selección de grupos

Del mismo modo que la cláusula `WHERE` permite la selección de filas en una sentencia `SELECT`, la cláusula **`HAVING`** permite realizar una selección sobre los grupos obtenidos por la cláusula **`GROUP BY`**.

Formato de consulta con selección de grupos

```
SELECT funciones_de_columna ?????????????????????>

>??FROM lista_de_tablas ?????????????????????????>

>????????????????????????????????????????????????>
   ?WHERE condición_de_selección?

>????????????????????????????????????????????????>
   ?GROUP BY lista_de_columnas_para_agrupar?

>????????????????????????????????????????????????>
   ?HAVING condición_de_selección?

>????????????????????????????????????????????????>;
   ?ORDER BY especificaciones_para_ordenar?
```

La **condición de selección** podrá estar formada por constantes, columnas de agrupación y funciones de columna.

Ejemplos.

1. Seleccionar los oficios que tengan dos o más empleados:

```
SQL> SELECT oficio, COUNT (*) FROM empleados
       GROUP BY oficio HAVING COUNT (*) >= 2;

OFICIO       COUNT(*)
---------- ----------
DIRECTOR            2
EMPLEADO            2
VENDEDOR            3
```

La cláusula **`HAVING` actúa como un filtro sobre filas agrupadas**, a diferencia de la cláusula `WHERE` que actúa sobre las filas antes de la agrupación.

2. Seleccionar los oficios que tengan dos o más empleados, cuyo salario supere las 140000 ptas.

```
SQL> SELECT oficio, COUNT (*)
       FROM empleados
       WHERE salario > 140000
       GROUP BY oficio
       HAVING COUNT (*) >= 2;

OFICIO       COUNT(*)
---------- ----------
DIRECTOR            2
VENDEDOR            2
```

`SQL` realiza la selección de grupos en el proceso siguiente:

- A partir de la tabla sobre la que se realiza la consulta, la cláusula `WHERE` actúa como un primer filtro que da como resultado una tabla interna cuyas filas cumplen la condición especificada en el `WHERE`.

- La cláusula `GROUP BY` produce la agrupación de las filas de la segunda tabla, dando como resultado una tercera tabla.

- La cláusula `HAVING` actúa filtrando las filas de la tercera tabla, según la condición de selección especificada, dando como resultado la salida de la consulta.