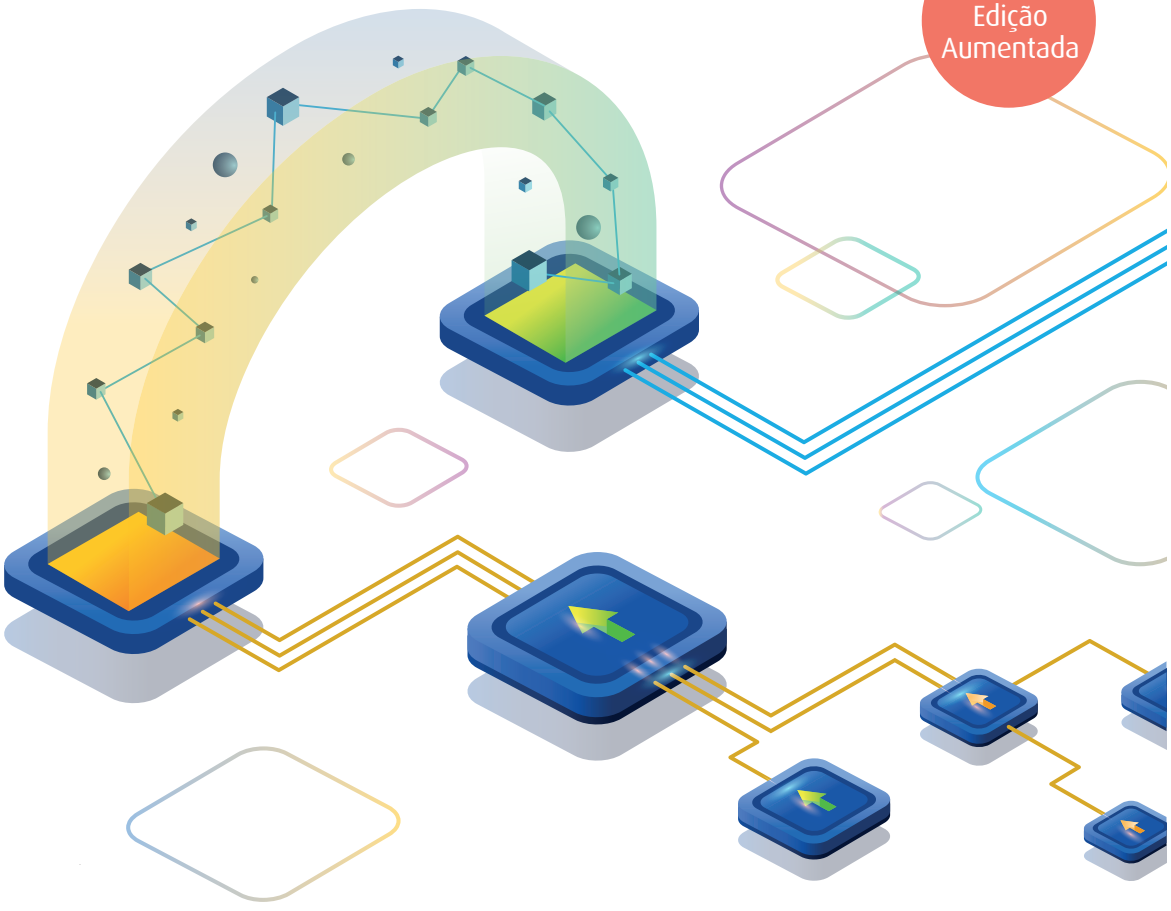
Feliz Gouveia

Bases de Dados

Fundamentos e Aplicações

2.ª Edição Aumentada



FCA

Edição
FCA – Editora de Informática, Lda.
Av. Praia da Vitória, 14 A – 1000-247 Lisboa
Tel: +351 213 511 448
fca@fca.pt
www.fca.pt

Distribuição
Lidel – Edições Técnicas, Lda.
Rua D. Estefânia, 183, R/C Dto. – 1049-057 Lisboa
Tel: +351 213 511 448
lidel@lidel.pt
www.lidel.pt

Livraria
Av. Praia da Vitória, 14 A – 1000-247 Lisboa
Tel: +351 213 511 448
livraria@lidel.pt

ISBN edição impressa: 978-972-722-901-7
1.ª edição impressa: setembro 2014
2.ª edição impressa: fevereiro 2021

Paginação: Carlos Mendes
Impressão e acabamento: Tipografia Lousanense, Lda. - Lousã
Depósito Legal n.º 479533/21
Capa: José Manuel Reis

Todos os nossos livros passam por um rigoroso controlo de qualidade, no entanto aconselhamos a consulta periódica do nosso site (www.fca.pt) para fazer o *download* de eventuais correções.

Não nos responsabilizamos por desatualizações das hiperligações presentes nesta obra, que foram verificadas à data de publicação da mesma.

Os nomes comerciais referenciados neste livro têm patente registada.

ÍNDICE GERAL

1

INTRODUÇÃO

"We find countless examples of programmers who have moved on, leaving programs that are undecipherable for maintenance or modification. Had they been written in COBOL, they would have had more than zero salvage value."

R.W. Bemmer, *A view of the history of COBOL* (1971)

SUMÁRIO

Este capítulo apresenta o tema do livro e as definições iniciais necessárias ao desenvolvimento dos capítulos seguintes. É introduzido o conceito de bases de dados e apresentada a sua arquitetura, bem como as características essenciais da sua utilização, tal como o conceito de transação e de controlo de concorrência. O capítulo inclui ainda um breve histórico do aparecimento e da evolução da tecnologia dos sistemas de gestão de bases de dados. No final, descrevem-se os diferentes capítulos do livro, permitindo ao leitor definir o seu percurso de leitura.

1.1 BASES DE DADOS

Em 2009, festejou-se o 40.º aniversário da definição do Modelo Relacional de Dados, considerado o marco do nascimento das bases de dados modernas (Codd, 1969). Na realidade, o Modelo Relacional é hoje utilizado praticamente em todos os sistemas de gestão de bases de dados existentes, os SGBD, embora existam outros modelos de dados desenvolvidos para aplicações específicas, mas que não tiveram, até ao momento, o sucesso e a divulgação do Modelo Relacional. A indústria baseada no Modelo Relacional de Dados vale atualmente dezenas de milhares de milhões de euros, e a tendência é para continuar a crescer. Como veremos, as bases de dados relacionais não têm o exclusivo do armazenamento da enorme quantidade de dados gerada em todas as aplicações e sistemas, contudo, apresentam uma percentagem elevada de adoção, que é reveladora da adequação da sua arquitetura e funcionalidades às necessidades atuais. Praticamente todas as aplicações informáticas gerem os seus dados com um SGBD, independentemente da quantidade de dados ou do número de pessoas que utiliza a aplicação. Alguns SGBD gerem milhares de acessos simultâneos e enormes quantidades de informação, por exemplo: aplicações de reserva de bilhetes de avião, sistemas de informação empresariais e lojas em linha.

1.2 CENÁRIOS DE UTILIZAÇÃO

Atualmente, as bases de dados estão omnipresentes na vida das pessoas e das organizações, constituindo o núcleo a partir do qual os sistemas de informação são construídos e operam. Dos telemóveis aos cartões bancários, passando por todo um conjunto de serviços que o cidadão tem à sua disposição na *Web*, encontram-se bases de dados de dimensão e complexidade variáveis. A sua utilização passou do ambiente puramente organizacional, onde se armazenavam dados de recursos humanos, produtos, vendas, clientes, para aspetos que têm um impacto direto e importante na vida das pessoas. As bases de dados com informação de saúde, genética e financeira são hoje comuns, aumentando a responsabilidade de todos os que direta ou indiretamente estão envolvidos, e aumentando consideravelmente o impacto da sua utilização incorreta. Com efeito, o aumento exponencial do número e diversidade de serviços disponíveis em linha só foi possível com a entrada em produção de bases de dados com uma capacidade cada vez maior de gerir volumes consideráveis de dados e de gerir corretamente centenas ou milhares de acessos simultâneos.

Uma base de dados pode caracterizar-se pelo número de utilizadores simultâneos que podem ter acesso à mesma, bem como pela quantidade de dados que armazena. Quando instalada num dispositivo portátil, tem características de utilização muito diferentes das da base de dados de um banco, por exemplo. No entanto, em qualquer das situações, espera-se que os dados não desapareçam nem sejam modificados de forma não desejada, que possam resistir a incidentes, físicos e lógicos, que afetem a base de dados, e que não sejam necessários conhecimentos especializados de utilização. Mesmo um utilizador especializado, regra geral, não necessita de otimizar a sua interação com a base de dados, existindo componentes que se encarregam de encontrar a melhor forma de processar os comandos, escondendo assim do utilizador os detalhes técnicos da estruturação e armazenamento dos dados.

1.3 SGBD

Existem várias definições de base de dados, mas nesta fase inicial podemos utilizar a seguinte:

> **Uma base de dados é uma coleção organizada de informação estruturada relacionada entre si que persiste durante um determinado período de tempo. É gerida por um Sistema de Gestão de Bases de Dados (SGBD), que condiciona a forma como se cria, acede, modifica e elimina informação.**

Os aspetos principais desta definição incluem a noção de **informação estruturada**, de **persistência** e de um sistema de gestão responsável por toda a interação dos utilizadores ou de programas com os dados guardados em memória permanente. O sistema de gestão da base de dados, o SGBD, serve de interface com o utilizador e gere toda a interação deste com os dados.

2

O MODELO RELACIONAL

"[…] the database systems of 1970 were costly to use because of the low-level interface between the application program and the DBMS, and because the dynamic nature of user data mandates continued program maintenance. The relational data model, pioneered by E. F. Codd in a series of papers in 1970-72, offered a fundamentally different approach to data storage. Codd suggested that conceptually all data be represented by simple tabular data structures (relations), and that users access data through a high-level, nonprocedural (or declarative) query language."

Avi Silberschatz, Michael Stonebraker, Jeff Ullman (Eds.), *Database Systems: Achievements and Opportunities* (1991)

SUMÁRIO

Neste capítulo, introduzimos o Modelo Relacional de Dados, os seus constituintes e a sua implementação nos SGBD. Apresentamos as características do Modelo Relacional e os conceitos base utilizados. Os requisitos que um sistema relacional deve respeitar são introduzidos sob a forma de regras. Descrevemos a álgebra relacional e os seus operadores para manipulação do Modelo Relacional, e abordamos os cinco operadores básicos da álgebra, bem como um conjunto de extensões que simplificam a sua utilização na escrita de consultas.

2.1 INTRODUÇÃO

No fim dos anos 60, os precursores dos SGBD modernos usavam essencialmente os Modelos CODASYL e Hierárquico. Ambos implicavam programar aplicações para lidar com estruturas de baixo nível, como apontadores e noções de adjacência entre dados, estruturas que eram frequentemente alteradas para responder a novas necessidades das aplicações. Deste modo, as aplicações tinham de ser modificadas, uma vez que dependiam da forma de representação dos dados e do acesso aos mesmos.

O Modelo Relacional de Dados foi proposto por Edgar Codd (1969) para responder às dificuldades colocadas pelos sistemas existentes, que obrigavam os programadores a conhecer a implementação física dos dados para os manipular. Qualquer mudança na estrutura física dos dados obrigava a alterar todos os programas que os utilizavam. Do ponto de vista da produtividade, uma grande parte do esforço dos programadores era despendida em alterações constantes para adaptar os programas às contínuas modificações da representação dos dados.

Codd propôs que qualquer modelo fosse constituído pelos seguintes componentes:

- Um conjunto de tipos de estruturas de dados;
- Um conjunto de operadores para manipular as estruturas do ponto anterior;
- Um conjunto de regras de integridade que definem os estados consistentes da base de dados e as mudanças de estado permitidas.

Segundo esta definição, o Modelo Relacional de Dados foi o primeiro a ser proposto. Antes do seu aparecimento, existiam, como vimos, implementações de bases de dados hierárquicas e em rede (ou CODASYL), mas que não tinham o suporte concetual de um modelo, sendo que as suas implementações não se regiam por regras claras. Contudo, o Modelo Relacional definiu os três componentes acima descritos. Para especificar as estruturas de dados e as regras de integridade, um SGBD fornece uma Linguagem de Definição de Dados (LDD) e para manipular as estruturas de dados, incluindo inserção, remoção e modificação, um SGBD fornece uma Linguagem de Manipulação de Dados (LMD).

2.2 O MODELO RELACIONAL DE DADOS

A relação é o elemento fundamental do Modelo Relacional. Matematicamente, uma relação é um conjunto de tuplos sobre os domínios D_1, D_2, ..., D_n, não necessariamente exclusivos, em que cada elemento do tuplo é retirado de D_1, D_2, ..., e D_n. Um tuplo designa-se também como *n*-tuplo, sendo *n* o número de domínios. Por sua vez, um domínio é um conjunto de valores atómicos, isto é, indivisíveis, de um determinado tipo. Alguns exemplos comuns de domínios são:

- **Nomes**: O conjunto de nomes de pessoas;
- **Moradas**: O conjunto de moradas;
- **Números de aluno**: O conjunto de números de aluno;
- **Telefones**: O conjunto de números de telefone;
- **Distritos**: Aveiro, Beja, Braga, Bragança, etc.

A cada domínio está associado um tipo de dados; por exemplo, **moradas** é do tipo texto, podendo ser especificado eventualmente um número máximo de caracteres. Diferentes domínios podem ter o mesmo tipo de dados. Os SGBD não fazem distinção entre os domínios e os seus tipos, utilizando apenas tipos de dados. No entanto, a maioria dos SGBD permite tipos definidos pelo utilizador, que permitem o equivalente à definição de domínios.

Uma relação define assim a estrutura dos dados a armazenar. Todos os dados de uma relação – tuplos, linhas ou registos – têm a mesma estrutura, isto é, os mesmos atributos. O esquema de um SGBD é o conjunto de bases de dados definidas, e o esquema da base de dados é o conjunto das relações definidas. Deste modo, o esquema de uma relação inclui o seu nome e a lista dos nomes das suas colunas. Um exemplo de esquema da relação DISCIPLINA é o seguinte:

3

PROJETO DE BASES DE DADOS

"The basic objectives of database design are to enable users to obtain the exact data they need to perform their duties within the organization, and to make the data available in a reasonable amount of time."

Toby Teorey, James Fry, *The Logical Record Access Approach to Database Design* (1980)

SUMÁRIO

Neste capítulo, vamos descrever o projeto de uma base de dados relacional, começando pelo modelo concetual mais utilizado, o Modelo Entidade-Relacionamento (E-R). Veremos depois como traduzir esse modelo para o Modelo Relacional, descrevendo as regras de mapeamento mais utilizadas. As relações obtidas são analisadas nesta fase de projeto lógico, pelo que se introduzem as anomalias, e se aprofunda o conceito de dependências e outras regras de integridade. São descritos tipos de dependências e algoritmos para determinar vários conjuntos que são utilizados para determinar diversas propriedades das relações. É descrito o processo de normalização e as formas normais resultantes.

3.1 INTRODUÇÃO

O projeto de uma base de dados começa com o levantamento de requisitos da informação que deverá estar representada, processo que, segundo as aplicações, poderá ser prolongado e envolver várias pessoas, com perspetivas e linguagens diferentes sobre a mesma informação. A utilização de um modelo concetual permite separar as questões técnicas e simplificar a notação da informação, ao mesmo tempo que se consegue partilhar o modelo com vários tipos de interlocutores. A necessidade de se encontrar uma representação da informação que pudesse ser compreendida, analisada e trabalhada por pessoas com um perfil não técnico levou ao aparecimento dos modelos concetuais. Para poder ser interpretado por vários tipos de utilizadores, um modelo concetual deve oferecer alguma flexibilidade de interpretação e ser fácil de modificar, à medida que a informação vai sendo refinada e ganhando consistência. Por isso, os modelos concetuais são geralmente modelos gráficos, de fácil leitura e interpretação.

3.2 MODELOS CONCETUAIS

Um modelo concetual representa a informação de forma abstrata, permitindo recolher e representar a informação que se pretende, para que esta possa ser gerida pelo SGBD. A tarefa de guiar o processo de recolha de informação, envolvendo todos os interlocutores, e a sua representação no modelo concetual está geralmente a cargo de um ou mais analistas. Após todos os intervenientes estarem de acordo, o projeto concetual dá lugar ao projeto lógico da base de dados.

Nas secções seguintes, vamos descrever um modelo concetual simples, o Modelo E-R, que, historicamente, é o precursor de modelos concetuais para o projeto lógico, e veremos, depois, como utilizá-lo para o projeto lógico da base de dados.

3.3 MODELO E-R

O Modelo E-R foi proposto, em 1976, por Peter Chen (Chen, 1976, 1977) como uma alternativa aos conceitos do Modelo Relacional, permitindo modelar problemas de uma forma mais próxima da perspetiva das pessoas e do negócio. O Modelo E-R é constituído por entidades e pelos seus relacionamentos, conceitos que são representados graficamente e podem ser apreendidos com facilidade sem qualquer conhecimento técnico de bases de dados. Pelo contrário, o Modelo Relacional implica o conhecimento das suas bases, e é mais restritivo na representação lógica de informação para um não iniciado, permitindo apenas o uso de relações para representar toda a informação.

O Modelo E-R introduz três conceitos fundamentais:

- Conjuntos de entidades;
- Atributos;
- Relacionamentos.

Os conjuntos de entidades são conjuntos de informação que têm uma existência própria, muitas vezes física, e que fazem parte do problema a modelar. Exemplos de conjuntos de entidades são o aluno, o docente, a disciplina, o curso e a sala. As entidades são os alunos, os docentes, etc., ou seja, as instâncias que pertencem a um determinado conjunto. As entidades são do tipo determinado pelo conjunto a que pertencem. Por sua vez, o tipo de entidade determina as características de todas as entidades; o conjunto de entidades reúne entidades do mesmo tipo[1]. Estas entidades são caracterizadas por um conjunto de atributos, cada um com o seu nome e tipo de dados. Cada conjunto de entidades possui um conjunto de atributos únicos, identificativos, que corresponde à sua chave primária. Se esta não puder ser definida a partir dos atributos existentes, cria-se um identificador único para esse fim. No Modelo E-R um conjunto de entidades é representado graficamente por um retângulo.

[1] Sempre que for evidente no contexto, usamos indiferentemente "tipo de entidade" e "conjunto de entidades".

4

SQL

"(Codd gave a speech and he said) Sure, you can express a question by writing a navigational plan. But if you think about it, the navigational plan isn't really the essence of what you are trying to accomplish. You are trying to find the answer to some question, and you are expressing the question as a plan for navigating. Wouldn't it be nice if you could express the question at a higher level and let the system figure out how to do the navigation?"

Charles Babbage Institute, Entrevista com Donald Chamberlin (2001)

SUMÁRIO

Neste capítulo vamos introduzir a linguagem SQL, focando essencialmente o conjunto de comandos previsto na norma SQL99. Embora os SGBD existentes implementem a norma SQL de formas diferentes, quer fornecendo extensões quer alterando a sintaxe de alguns comandos, o que torna os esforços de compatibilidade e de portabilidade complexos, é possível encontrar um núcleo de comandos comuns à maioria dos SGBD. A inexistência tanto de especificações completas como de baterias de testes de conformidade leva a que os SGBD optem por implementações que lhes sejam mais convenientes. Por outro lado, a norma SQL está em constante evolução e várias das suas funcionalidades não tiveram aceitação generalizada - por exemplo, as extensões objeto-relacional e temporal são implementadas por poucos produtos. Por isso, este capítulo é uma introdução à linguagem SQL, devendo o leitor recorrer a obras especializadas, e aos manuais dos SGBD que utilizar, para tirar pleno partido das potencialidades da linguagem.

4.1 INTRODUÇÃO

A linguagem SQL foi desenvolvida para simplificar a interação de um utilizador com uma base de dados relacional. Foi desenvolvida pela IBM, no âmbito do projeto **System R**, e originalmente chamava-se *Structured English Query Language* (SEQUEL) (Chamberlin e Boyce, 1974). A SEQUEL evitava ter de escrever consultas em álgebra ou cálculo relacional, o que seria um grande obstáculo à adoção dos SGBD baseados no Modelo Relacional. Os comandos de SEQUEL eram essencialmente uma forma de escrever em inglês uma consulta, devendo manter o poder expressivo da álgebra e do cálculo relacional. Devido a um problema com o registo do nome, a SEQUEL passou a ser designada como *Structured Query Language* (SQL).

A proposta original de SQL foi sendo adotada inicialmente por diferentes produtos, cada um interpretando-a por vezes de forma diferente, ou adicionando novos operadores e

funções, não existentes ou não compatíveis com os existentes noutros produtos. Para evitar a proliferação de versões de SQL e limitar os problemas de compatibilidade entre diferentes produtos, o ANSI publicou, em 1986, a primeira normalização da linguagem SQL (conhecida como SQL86), que sofreu uma pequena revisão em 1989, dando origem à versão SQL89.

Em 1992, a *International Standards Organization* (ISO) publica juntamente com a ANSI uma revisão mais profunda da norma existente, dando origem à versão SQL92 (também designada como SQL2), que definia três níveis de compatibilidade (básico, intermédio e completo), em parte devido à complexidade da norma e à dificuldade dos SGBD em respeitarem todas as funcionalidades.

Em 1999, é publicada uma nova revisão da norma, a SQL99 (também designada como SQL3), reestruturando-a a vários níveis e aumentando consideravelmente as funcionalidades e o seu âmbito (ISO/IEC 9075, 1999). A SQL99 introduziu uma especificação em componentes, consistindo num núcleo base de especificações e em várias extensões. Note-se que a conformidade com o núcleo é um requisito para qualquer produto relacional, sendo que as extensões são opcionais.

Regularmente, são adicionadas extensões a SQL, tendo sido publicadas as SQL2003, SQL2006 (extensões XML), SQL2008 (extensões orientadas a objetos) e SQL2011 (extensões temporais). O núcleo de SQL99 tem permanecido estável e em SQL2011 é constituído por cerca de 170 requisitos. Nas secções seguintes, vamos ver as funcionalidades essenciais de SQL, devendo alguns aspetos de sintaxe, tipos de dados e comportamento de alguns operadores ser verificados na documentação do produto específico que se estiver a usar. Damas (2005) compara a sintaxe de vários SGBD, sendo, por isso, uma referência útil nas questões de portabilidade entre SGBD.

As funcionalidades essenciais de SQL podem dividir-se em dois grupos: Linguagem de Definição de Dados (LDD) e Linguagem de Manipulação de Dados (LMD). Nas secções seguintes, serão introduzidos os comandos mais comuns, contudo, não se fará uma apresentação exaustiva da norma SQL, nem das variantes de cada comando. O leitor será remetido para obras na bibliografia e para o manual do SGBD que estiver a utilizar.

4.2 LINGUAGEM DE DEFINIÇÃO DE DADOS

A LDD permite criar e modificar o esquema da base de dados, bem como definir vários tipos de restrições e de índices.

4.2.1 ESQUEMAS

Um esquema de base de dados é um conjunto de tabelas. Por razões de segurança ou de estruturação lógica da informação na base de dados, podem usar-se esquemas diferentes na mesma base de dados e dar autorizações diferentes a diferentes utilizadores. Um esquema define um espaço de nomes, um pouco como os componentes (*package*)

5

ARMAZENAMENTO E INDEXAÇÃO

"[...] The actual value of *w* is usually thought of as being outside the domain of systems analysis (which is correct), and it is hoped that the decision maker will be able to give a reasonable estimate of it (which is, usually, incorrect). The poor decision maker, confronted with questions like 'Would you prefer a system having 1.59 accesses and 400 million bytes to a system having 1.73 accesses and 320 million bytes?', will usually demand a set of optimizing Q values for a set of *w's* with the vague hope that more numbers will make it easier to decide."

Y. Milman, *An Approach to Optimal Design of Storage Parameters in Databases* (1977)

SUMÁRIO

Neste capítulo, vamos descrever as estruturas de armazenamento e de indexação, e apresentar a estrutura de um registo, as alternativas para representar registos no disco, os tipos mais comuns de ficheiros, a forma como estes organizam os registos e as suas características face às operações habituais. São também apresentados o gestor de armazenamento, os modelos mais comuns de organização física dos registos em disco e os métodos mais usuais de indexação, sendo analisado o seu desempenho em diferentes situações. O capítulo termina com uma análise da utilização de índices e da importância da sua escolha no desempenho da base de dados.

5.1 INTRODUÇÃO

O SGBD permite esconder do utilizador a gestão dos dados fornecendo uma interface de gestão de ficheiros, o que, na prática, consiste em coleções de registos. Estes, por sua vez, são guardados em várias páginas ou blocos. Como o SGBD tem de garantir persistência, os ficheiros são geralmente guardados em memória secundária, em discos magnéticos. Note-se que o custo do acesso aos dados em disco constitui uma parcela importante do custo de execução de consultas, e é considerado o fator mais importante no desempenho do SGBD. Por essa razão, as estruturas de armazenamento e de indexação devem ser cuidadosamente pensadas e geridas.

Outra abstração fundamental é o identificador único de um registo ou RID[1], que permite a identificação da página física onde o registo está guardado. Assim, bastará guardar o seu

[1] *Record Identifier* ou *Tuple Identifier* (TID).

RID no local onde for necessário armazenar informação de um registo, devendo o RDI ser independente de operações de reorganização de registos dentro das páginas; caso contrário, qualquer alteração da localização física do registo implicaria tornar obsoleto o seu RID e substituí-lo por um novo.

5.2 GESTOR DE ARMAZENAMENTO

O gestor de armazenamento é o componente do SGBD responsável por gerir as páginas que organizam os registos das relações no disco e executa essencialmente as seguintes funções:

- Lê e escreve páginas para o disco;
- Reserva e liberta páginas no disco.

O gestor de armazenamento recebe pedidos do gestor de memória, que gere as páginas necessárias às consultas em execução. Vamos ver de seguida os formatos de registos e de páginas mais utilizados, e quais as suas implicações na operação do gestor de armazenamento.

5.3 REGISTOS

Os ficheiros estão organizados em registos, sendo que cada registo representa um tuplo. Os registos podem ser de comprimento fixo ou variável. Para os registos de comprimento fixo, o catálogo da base de dados dá toda a informação necessária sobre os campos e o seu tamanho. Para os registos de comprimento variável, é necessário guardar no cabeçalho do registo o tamanho de cada campo, por exemplo, guardando o deslocamento para o início de cada campo.

5.3.1 REGISTOS DE COMPRIMENTO FIXO

Um registo de comprimento fixo reserva sempre o mesmo espaço. O esquema da tabela correspondente guarda o número de campos, o seu tipo e a sua ordem. A partir de um esquema, pode obter-se o tamanho do registo, calcular a sua posição na página, bem como a posição de cada coluna. Esta opção de guardar registos apresenta as seguintes características:

- Inserir ou remover registos é fácil, porque todos têm o mesmo tamanho; quando um registo é removido, pode marcar-se o lugar onde este se encontrava como livre. Se existir uma lista ligada de espaços livres, ou um mapa de *bit*, a escolha de um espaço livre é simplificada;
- Se a página não for um múltiplo do tamanho do registo, este pode ficar armazenado em duas páginas. Tal situação pode, no entanto, ser detetada e evitada, embora

6

CONSULTAS

"Database management systems provide three essential groups of services. First, they maintain both data and associated metadata in order to make databases self-contained and self-explanatory, at least to some extent, and to provide data independence. Second, they support safe data sharing among multiple users as well as prevention and recovery of failures and data loss. Third, they raise the level of abstraction for data manipulation above the primitive access commands provided by file systems with more or less sophisticated matching and inference mechanisms, commonly called the query language or query-processing facility."

G. Graefe, *Query Evaluation Techniques for Large Databases* (1993)

SUMÁRIO

Neste capítulo, introduzimos o processamento de consultas, as suas fases, e descrevemos os principais problemas de otimização. Começamos por apresentar as etapas envolvidas no processamento de consultas, na análise sintática, na geração de planos lógicos, na geração de planos físicos e na sua otimização. São também descritas as heurísticas de transformação de planos e as regras de equivalência utilizadas para a transformação de planos lógicos, apresentados os diferentes operadores físicos correspondentes aos operadores lógicos, e analisados os fatores determinantes do seu desempenho do ponto de vista do custo da execução.

6.1 INTRODUÇÃO

A forma como se processa uma consulta influencia consideravelmente o seu tempo de execução. A necessidade de otimizar o processamento de consultas foi identificada desde o primeiro protótipo do **System R** (Selinger et al., 1979), e muitos dos princípios aí definidos ainda hoje estão em utilização. Em particular, a otimização baseada num algoritmo de programação dinâmica continua a ser, com alterações, a aproximação mais usada nos SGBD.

Vimos no Capítulo 1 que o SGBD fornece uma interface e uma linguagem de alto nível evitando que o programador tenha de decidir qual a melhor forma de execução das suas consultas. Esta independência física dos dados é uma das características principais dos SGBD e do Modelo Relacional de Dados. Assim, compete ao SGBD encontrar a melhor forma de as executar, otimizando-as sem a intervenção do utilizador. Sendo a SQL uma linguagem declarativa, não há qualquer indicação de como as consultas devem ser executadas. Outro aspeto importante tem que ver com a imprevisibilidade das consultas,

pelo que o processador deve considerar que qualquer tipo de consulta poderá acontecer (Chamberlin, Astrahan et al., 1981).

O processamento de consultas parte da consulta especificada em linguagem SQL e produz uma sequência de operadores físicos que manipulam diretamente as estruturas de acesso e os registos. A escolha dos operadores e da sua ordem constituem as principais decisões a tomar. Geralmente os SGBD adotam um modelo de custo e utilizam-no para ordenar alternativas de execução, que se chamam planos da consulta. A geração dos planos e a sua ordenação em função do custo tem de ser feita em tempo real, à medida que as consultas são submetidas, devendo estas tarefas ter idealmente um impacto mínimo no tempo de execução. Embora o modelo baseado em custos seja o mais comum, continuam a ser usados métodos heurísticos ainda que raramente em regime de exclusividade.

Existem diferentes modelos de custo em função dos parâmetros que estes utilizam. Na prática, é impossível determinar um custo exato, da mesma forma que não é possível gerar em tempo útil todas as alternativas possíveis de processamento de uma dada consulta. Os parâmetros de custo mais comuns são: a velocidade de transferência do disco, o tempo de CPU utilizado e os custos de gestão de memória. Noutros modelos, considera-se, ainda, por exemplo, a forma como os tuplos são lidos no disco – sequencial ou aleatoriamente – uma vez que a maioria dos discos atuais apresenta uma diferença substancial entre os dois modos de leitura.

Em todos os modelos, a qualidade do resultado depende do erro com que este é calculado, ou seja, do erro da estimativa do número de linhas do resultado de um operador. Estimar com a maior precisão possível o resultado de um operador, isto é, a sua seletividade, constitui a principal tarefa da fase de otimização do processamento de consultas. Assim, estimar a seletividade implica que o processador de consultas tenha informação estatística sobre o conteúdo da base de dados, ou seja, sobre os tuplos em cada relação e sobre algumas das suas características. Note-se que estas estatísticas são recolhidas em momentos definidos, por vezes de forma periódica, de forma a não interferir com o desempenho da base de dados.

O custo é calculado para cada um dos operadores que constituem o plano de uma consulta, e o custo total desta é a soma de todas as suas parcelas. Repare-se que o objetivo do modelo de custo não é calcular de forma absoluta o tempo de processamento de uma consulta, mas sim obter um valor para ordenar os seus diferentes planos, permitindo escolher o mais eficiente.

A execução de consultas deve respeitar os aspetos de controlo de concorrência na base de dados, ou seja, permitir diferentes consultas, em que uma ou várias modificam o conteúdo da base de dados, não podendo interferir de forma imprevista, fornecendo resultados não esperados. Veremos nos Capítulos 8 e 9 como diferentes transações podem executar concorrencialmente e respeitar as condições ACID definidas na secção 1.4. Neste capítulo, consideramos que, de um modo geral, cada consulta é apenas de leitura e está completamente isolada de outras consultas a executar concorrencialmente. No entanto, os conceitos apresentados são igualmente válidos para as operações de inserção, atualização e remoção de tuplos.

7

OTIMIZAÇÃO DE CONSULTAS

"Imagine yourself standing in front of an exquisite buffet filled with numerous delicacies. Your goal is to try them all out, but you need to decide in what order. What exchange of tastes will maximize the overall pleasure of your palate? Although much less pleasurable and subjective, that is the type of problem that query optimizers are called to solve. Given a query, there are many plans that a database management system (DBMS) can follow to process it and produce its answer. All plans are equivalent in terms of their final output but vary in their cost, i.e., the amount of time that they need to run. What is the plan that needs the least amount of time?"

Y. Ioannidis, *Query Optimization* (1996)

SUMÁRIO

Neste capítulo, vamos introduzir o conceito de otimização de consultas, as suas fases e apresentar os principais algoritmos de otimização. Começamos por descrever as etapas envolvidas no processamento de consultas, na análise sintática, na geração de planos lógicos, na geração de planos físicos e na sua otimização. São descritas as heurísticas de transformação de planos e as estratégias mais comuns de geração de planos físicos, e apresentados os modelos de custo, a informação que é utilizada no cálculo do custo das operações, bem como outras estruturas mais elaboradas, como os histogramas.

7.1 INTRODUÇÃO

O processo de otimização de consultas corresponde à segunda e à terceira etapas do processamento introduzido no Capítulo 6. O processo de otimização procura encontrar um plano de consulta que seja razoavelmente eficiente e não o plano ótimo, tarefa que provavelmente ultrapassaria o tempo disponível. A otimização é feita quando a consulta SQL é submetida para execução, pelo que se deve estabelecer um compromisso entre o tempo necessário para encontrar um "bom" plano e o tempo para executá-lo. Não faria sentido se o tempo gasto na otimização fosse superior ao tempo de execução de uma versão ineficiente da consulta. Assim, a otimização pode fazer toda a diferença no desempenho de um SGBD, sendo uma área muito ativa e importante, como demonstram as centenas de patentes existentes sobre os mais variados aspetos das técnicas envolvidas.

A Figura 7.1 mostra as etapas da otimização no processamento geral de uma consulta.

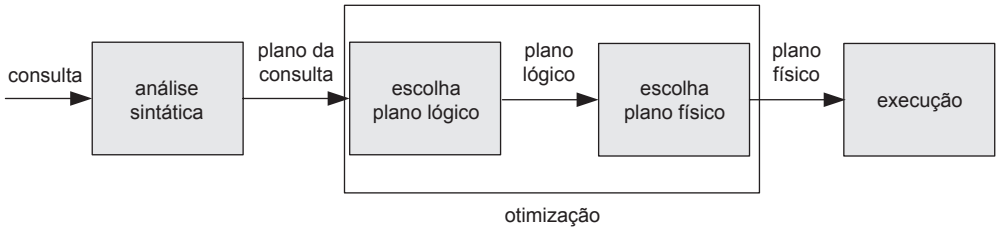


FIGURA 7.1 – ETAPAS DA OTIMIZAÇÃO

O componente de otimização recebe como entrada um plano da consulta, expresso em álgebra relacional ou numa forma equivalente, e deve produzir, no menor espaço de tempo um plano físico, identificando as operações a realizar, bem como a forma e a ordem em que serão realizadas. O processo de otimização de consultas deve ter as seguintes características:

- Ser correto, produzindo um plano que dá os mesmos resultados que a consulta original;
- Ser completo, contemplando todos os aspetos da linguagem de consulta;
- Evitar os piores planos, tentando produzir o plano mais eficiente, que pode ser o mais rápido, o que consome menos recursos, ou o que produz tuplos mais rapidamente. Alguns SGBD permitem definir o critério principal a ser utilizado;
- Ser eficiente, produzindo o melhor plano num espaço de tempo razoável;
- Ser resiliente, permitindo que mesmo face a consultas muito complexas, e que seriam muito morosas de analisar em profundidade, seja produzido o melhor plano possível;
- Ser robusto, face a erros e a situações imprevistas no código.

Na prática, o componente de otimização engana-se frequentemente e pode efetuar escolhas desastrosas em termos de tempo de execução. Uma escolha errada pode acontecer por várias razões:

- As estatísticas utilizadas estão desatualizadas ou não existem, o que provoca erros nos cálculos da seletividade e do custo dos operadores físicos, levando a escolhas erradas;
- As fórmulas de cálculo utilizadas não são adequadas a determinadas situações e provocam erros maiores do que os esperados;
- Os parâmetros utilizados no cálculo do custo estão desajustados da realidade do equipamento (por exemplo, nos rácios entre o custo de acesso aleatório ao disco e o custo de acesso sequencial, ou no custo da CPU em relação a uma leitura no disco);
- A memória disponível durante a execução da consulta não é a que foi assumida nos cálculos;
- O componente de otimização pode ter erros nos seus algoritmos e utilizar heurísticas cujos resultados não são satisfatórios para determinadas consultas.

8

GESTÃO DE TRANSAÇÕES

"The transaction concept derives from contract law. In making a contract, two or more parties negotiate for a while and then make a deal. The deal is made binding by the joint signature of a document or by some other act (as simple as a handshake or not). If the parties are rather suspicious of one another or just want to be safe, they appoint an intermediary (usually called an escrow officer) to coordinate the commitment of the transaction."

J. Gray, *The Transaction Concept: Views and Limitations* (1981)

SUMÁRIO

Neste capítulo, vamos introduzir o conceito de transação, descrever as suas propriedades e apresentar os critérios para definir a correção da execução concorrente de um conjunto de transações. Revisitamos as propriedades ACID e apresentamos as transições de estados permitidas numa transação, o componente do SGBD responsável por garantir uma execução correta de um conjunto de transações e o seu princípio geral de funcionamento. Introduzimos a noção de conflitos num escalonamento, os problemas mais comuns que daí podem decorrer e mostramos como esta noção pode ser usada para garantir a correção dos escalonamentos. Serão igualmente apresentados os critérios que um escalonamento deve respeitar e que permitem a recuperação dos dados em caso de falha. Introduzimos a noção de granularidade dos dados e a sua influência no controlo de concorrência. Por fim, apresentamos as noções de pontos de restauro, e de transações imbricadas e encadeadas.

8.1 INTRODUÇÃO

O controlo de concorrência é um dos componentes fundamentais de um SGBD, sem o qual não seria possível garantir tanto a integridade dos dados, como a utilização dos mesmos por vários utilizadores em simultâneo. Do ponto de vista do utilizador e da aplicação, o controlo de concorrência é responsável por lhes dar a ilusão de que têm a base de dados disponível apenas para si, sem terem de se preocupar com as ações de outros utilizadores ou aplicações. Como vimos, os acessos à base de dados são estruturados em transações, que têm um início e um fim bem delimitados, respeitando as propriedades ACID apresentadas no Capítulo 1. O objetivo do componente de controlo de concorrência é o de maximizar o número de transações que acedem concorrencialmente aos dados, rentabilizando assim os recursos físicos e lógicos disponíveis, e garantindo ao mesmo tempo que as transações não interferem de forma indesejável entre si. Adicionalmente, o gestor de transações envia informação ao gestor de recuperação, que permite, em caso

de falha ou acidente, voltar a colocar a base de dados num estado consistente, ou seja, permite a sua recuperação.

É comum as transações serem executadas dentro de programas escritos em linguagens de alto nível, que podem ser iniciados por ação de um utilizador ou de quaisquer outros programas. Um administrador pode estar a iniciar transações em linha, num cliente em linha de comando ou gráfico, enquanto vários outros utilizadores executam os seus comandos, e programas de manutenção e de administração executam diversos tipos de transações. O que é comum a todas estas situações é que uma transação deve ser uma unidade lógica de processamento, devendo os programadores de transações garantir que as suas fronteiras são bem delimitadas e que a sua execução respeita regras básicas do contexto em que é executada.

Num sistema multiutilizador, a utilização dos dados e dos recursos necessários à mesma, pode ser otimizada, se as transações intercalarem algumas das suas ações entre si, em vez de serem executadas na sua totalidade, em série, por ordem de chegada. Esta última opção poderia colocar transações mais pequenas, eventualmente apenas de consulta de dados, à espera de que transações mais longas terminassem embora estas necessitassem de elementos diferentes da base de dados. Pode também otimizar-se a utilização de recursos se o sistema puder ir processando uma transação enquanto espera por uma operação de entrada/saída de outra transação. Embora a execução em série das transações garanta sempre resultados corretos, sendo portanto a mais segura, não rentabiliza os recursos colocados à disposição do SGBD e, em alguns casos, pode provocar tempos de espera que não são toleráveis numa aplicação real.

Para atingir o objetivo de servir com tempos de resposta aceitáveis os seus utilizadores, o SGBD deve produzir um escalonamento das operações das diferentes transações que seja correto, ou seja, um escalonamento em que os resultados das transações sejam os esperados. A Figura 8.1 mostra o componente do gestor de transações responsável por escalonar transações. O algoritmo de controlo de concorrência é o responsável por ordenar as ações das diferentes transações, que, por sua vez, acedem a dados que se encontram num tampão em memória ou que são lidos da base de dados para essa memória, se necessário.

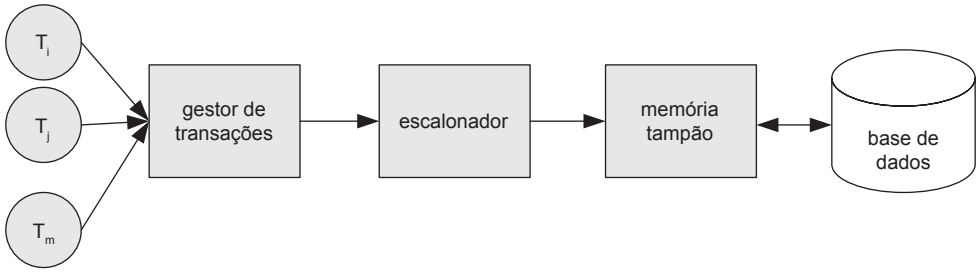


FIGURA 8.1 – EXECUÇÃO CONCORRENTE DE TRANSAÇÕES

Uma das funções importantes do controlo de concorrência é produzir um escalonamento correto das ações das diferentes transações que são iniciadas, função que é assumida pelo **escalonador**. Este recebe as operações à medida que elas vão sendo disponibilizadas

9

CONTROLO DE CONCORRÊNCIA

"We call the steps executed by a transaction that is later aborted wasted work. A transaction may also not be doing useful work because it is waiting. If the total number of concurrent transactions is n, and any pair of transactions conflict with probability p, we define the effective level of concurrency E(n, p) as the expected number of the n transactions that will be doing useful work. The expected remaining n - E(n, p) transactions will either be doing wasted work or else be waiting."

P. Franazesk, J. Robinson, *Limitations of Concurrency in Transaction Processing* (1985)

SUMÁRIO

Este capítulo introduz e descreve as técnicas de controlo de concorrência, o componente do gestor de transações que é responsável por garantir o isolamento das transações e a persistência dos seus efeitos, garantindo que a base de dados pode ser utilizada por vários utilizadores e processos simultaneamente. São apresentados os algoritmos mais utilizados de controlo de concorrência e a sua relação com os níveis de isolamento, bem como as técnicas otimistas e pessimistas de controlo de concorrência e as suas variantes. São introduzidos os níveis de isolamento e analisadas as diferentes variantes e interpretações com base no conceito de anomalia. Por último, são brevemente apresentados, do ponto de vista do controlo de concorrência, os principais SGBD existentes e as técnicas que utilizam.

9.1 INTRODUÇÃO

No Capítulo 8, apresentámos as propriedades que um escalonamento de transações deve respeitar para garantir a consistência da base de dados e assegurar que, em caso de falha, é possível recuperar a base de dados para um estado consistente. Neste capítulo, vamos apresentar as técnicas de controlo de concorrência que um SGBD pode implementar para garantir a serialização e a possibilidade de recuperação de um escalonamento. O controlo de concorrência não pode esperar que um escalonamento esteja completo para testar se este é serializável ou não; o que faz é garantir que as operações das transações que vão sendo submetidas para execução não poderão originar um escalonamento não serializável, proibindo quaisquer interações que o possam provocar ao impor uma ordem na sua execução. As técnicas de controlo de concorrência que vamos estudar neste capítulo têm a função de, ao utilizarem a informação de escalonamentos incompletos, ordenar as ações que vão sendo submetidas pelas diferentes transações de uma forma que nunca viole as

condições de serialização escolhidas pelas transações. Um requisito essencial a qualquer técnica de escalonamento passível de implementação prática é o seu elevado desempenho, tanto em tempo de execução como de utilização de memória, mesmo em presença de cargas transacionais elevadas.

As técnicas de controlo de concorrência mais estudadas, sendo algumas as mais utilizadas na maioria dos SGBD existentes, são:

- Fechos[1];
- Validação na confirmação (também chamada controlo otimista ou certificação);
- Ordenação temporal;
- Multiversão.

Regra geral, o uso de fechos é o mais comum, sendo a técnica de controlo de concorrência que apresenta melhor desempenho na maioria das situações (Carey e Stonebraker, 1984; Agrawal, Carey et al., 1987a). É também a técnica mais estudada e, historicamente, a preferida. As técnicas de validação na confirmação e de ordenação temporal apresentam melhor desempenho nas situações em que os acessos são essencialmente leituras, ou em que existe pouca contenção sobre os elementos da base de dados. No entanto, são pouco ou nada utilizadas, por exemplo: a validação na confirmação é utilizada nos produtos comerciais **GemStone** e **Versant**, que são bases de dados orientadas a objetos, sendo que pelo menos um SGBD desenvolvido por uma universidade, **Pyrrho**, utiliza também técnicas otimistas. A ordenação temporal é, por sua vez, utilizada em conjunto com outras técnicas, mas não sozinha. As técnicas multiversão têm vindo a ser cada vez mais adotadas, estando presentes em quase todos os SGBD mais populares, com a exceção de **DB2**. A experiência parece confirmar que as técnicas multiversão são adequadas a muitas das situações de utilização de bases de dados encontradas atualmente, nomeadamente, situações em que o número de leituras concorrentes é substancialmente superior ao das escritas – como acontece, por exemplo, nas aplicações *Web* ou nas aplicações analíticas de apoio à decisão.

Vamos começar por descrever os algoritmos, deixando para mais tarde as questões da sua implementação prática nos produtos disponíveis.

9.2 ALGORITMOS DE FECHO

Os algoritmos de fecho[2] baseiam-se na aquisição de fechos para os elementos que vão ser acedidos por uma transação, podendo o algoritmo especificar se pretende efetuar uma operação de leitura ou de escrita (Gray, Lorie e Putzolu, 1975; Korth, 1983). Concetualmente, um fecho colocado num elemento reserva a acesso a esse elemento para uma dada transação. A ordenação das operações das transações é feita porque uma transação só pode prosseguir se obtiver o fecho pedido, caso contrário, fica em espera. Assim, um

[1] *Locks* na terminologia anglo-saxónica.
[2] *Locking algorithms* na terminologia anglo-saxónica.

10

NÍVEIS DE ISOLAMENTO

> "The only advantage of lower degrees of consistency is performance. If less is locked then less computation and storage is consumed. Further if less is locked, concurrency is increased since fewer conflicts appear".
>
> J. Gray, R. A. Lorie et al., *Granularity of Locks and Degrees of Consistency in a Shared Data Base* (1976)

SUMÁRIO

Este capítulo introduz os níveis de isolamento, apresentando uma abordagem clássica em termos de anomalias e outra mais teórica que permite compreender a complexidade do tema. São analisadas as classes de escalonamentos permitidas em cada nível de isolamento e as anomalias que permitem evitar. Apresentam-se as limitações da definição SQL dos níveis de isolamento, devido à utilização implícita de fechos, e alternativas para alguns níveis de isolamento. É introduzido o nível *snapshot isolation*, pela sua crescente importância. Do ponto de vista da implementação dos níveis de isolamento pelos SGBD, são comparadas as opções de um grupo representativo de sistemas.

10.1 INTRODUÇÃO

Os algoritmos estudados no capítulo anterior destinam-se a garantir que um dado escalonamento é serializável e que, portanto, está isento de interferências indesejáveis entre as transações que o constituem. Vimos que os diferentes algoritmos podem ser classificados essencialmente em otimistas e pessimistas, dependendo da forma como detetam e gerem os conflitos entre as diferentes transações. É geralmente aceite que os algoritmos pessimistas introduzem mais contenção em cenários de elevada concorrência e que os algoritmos otimistas são mais eficazes quando as transações são essencialmente de leitura e provocam poucos conflitos.

Os algoritmos otimistas deferem as verificações de conflitos para o momento dos pedidos de confirmação das transações, evitando realizar esse trabalho durante a transação e esperando que nenhum problema ocorra no final. Por sua vez, os algoritmos pessimistas testam em cada operação a existência de conflitos e as suas possíveis implicações, prevendo o pior cenário possível. Pelo meio, temos algoritmos que misturam as aproximações otimistas e pessimistas.

Adotando esta divisão entre otimistas e pessimistas, a Figura 10.1 mostra o relacionamento entre os diferentes algoritmos analisados.

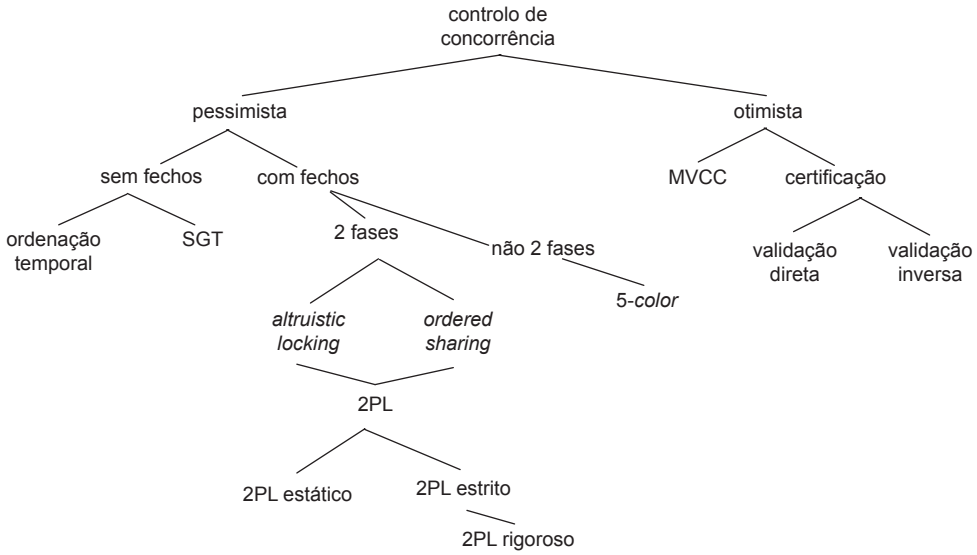


FIGURA 10.1 – ALGORITMOS DE CONTROLO DE CONCORRÊNCIA

Propondo que a consistência pode ser uma questão de compromisso dependendo do ganho de desempenho que daí pudesse resultar, Gray et al. (1976) e Gray (1980) introduziram a noção de granularidade de elementos da base de dados e propuseram os **níveis de consistência** que um escalonamento poderia tolerar, que resultaram posteriormente nos **níveis de isolamento** incluídos na normalização da linguagem SQL feita pela ANSI e pela ISO.

Os níveis de consistência destinam-se a permitir um aumento do desempenho de um SGBD, relaxando a verificação de conflitos e permitindo que certas anomalias, no pior cenário, possam ocorrer. Na prática, os níveis de consistência apareceram porque se verificava que os algoritmos de escalonamento diminuíam desnecessariamente o desempenho do SGBD ao imporem e testarem condições de interferência muito restritivas mas que raramente se verificavam. Ou seja, muitas transações estavam imunes aos problemas de consistência que uma execução serializada procurava evitar, ou então as consequências por obterem dados incorretos não eram graves e poderiam ser toleradas pelas aplicações. Como exemplo, as aplicações de processamento analítico de dados que só efetuam leituras podem dispensar muitas verificações de conflitos, sendo essencialmente estatísticas, com o consequente aumento de desempenho.

10.2 NÍVEIS DE ISOLAMENTO

No seu trabalho percursor, Gray et al. (1980) propuseram quatro níveis de consistência, de 0 a 3, sendo o nível 0 o mais permissivo e o nível 3 o mais restritivo. Na proposta original, os níveis foram definidos, de um ponto de vista operacional, da seguinte forma:

11

RECUPERAÇÃO

"Recovery data can itself be protected from failures by yet further recovery data which allow restoration of the primary recovery data in the event of its corruption. This progression could go on indefinitely. In practice, of course, there must be reliance on some ultimate recovery data (or rather, acceptance that such recovery data cannot be not totally reliable)."

J. Verhofstad, *Recovery Techniques For Database Systems* (1978)

SUMÁRIO

Este capítulo introduz e descreve as técnicas de recuperação, tarefa a cargo do gestor de recuperação, o componente do gestor de transações que é responsável por garantir que seja possível manter a consistência da base de dados quer em caso de interrupção das transações, quer em caso de falhas diversas. Apresentamos os conceitos de gestão de memória e as suas implicações nas estratégias de recuperação, bem como os princípios gerais das técnicas de recuperação, a interação com o gestor de transações, a interação entre o gestor de memória e o gestor de recuperação e as questões da sua implementação. Apresentamos também algumas das questões relacionadas com a gestão da memória disponível e demonstramos detalhadamente o algoritmo de recuperação mais divulgado e utilizado – ARIES (*Algorithm for Recovery and Isolation Exploiting Semantics*) –, assim como algumas das suas variantes.

11.1 INTRODUÇÃO

A recuperação de dados, seja de transações individuais ou da base de dados, é um dos aspetos essenciais da gestão de transações. No Capítulo 10, apresentámos o outro aspeto essencial, o controlo de concorrência. Vimos também que, para preservar as características ACID das transações, o SGBD tem de garantir a correção dos mecanismos de concorrência, ao mesmo tempo que garante a possibilidade de recuperação. Neste capítulo, vamos ver que as tarefas de recuperação estão a cargo de dois componentes que interagem diretamente com o gestor de transações:

- O gestor de recuperação, que regista as transações, gere a sua confirmação ou interrupção, e é igualmente responsável por gerir o arranque do SGBD em caso de falha;
- O gestor de *log*, que fornece um serviço de *log* para o gestor de recuperação e para qualquer outro componente que dele necessite, permitindo assim um registo redundante de informação em armazenamento estável.

A tarefa de recuperação a cargo do gestor de recuperação consiste geralmente numa das seguintes ações:

- Recuperar os efeitos de uma transação interrompida;
- Recuperar de uma falha no sistema;
- Recuperar a base de dados por completo, a partir de uma cópia de arquivo.

Do ponto de vista das propriedades ACID, o objetivo do componente de recuperação é o de garantir a atomicidade e a durabilidade das transações; como vimos, a consistência e o isolamento ficam a cargo do componente de controlo de concorrência. O componente de recuperação consegue cumprir este objetivo através da sua capacidade de, em caso de ocorrência de uma falha:

- Desfazer os efeitos de transações ainda não confirmadas à altura da falha, o que garante o carácter "tudo ou nada" das transações, ou seja, a sua atomicidade;
- Recuperar os resultados das transações confirmadas à altura da falha, o que garante o seu carácter permanente, ou seja, a sua durabilidade;
- Colocar a base de dados num estado consistente, o que resulta da realização dos dois anteriores.

A fiabilidade de qualquer sistema só pode ser garantida com alguma forma de redundância. Regra geral, quanto maior for a redundância, maior a fiabilidade do sistema. A capacidade de recuperação de um SGBD depende então do grau de redundância da informação manipulada que for capaz de manter. Nesta perspetiva, o componente de gestão de *log* é essencial. Para garantir esta capacidade, Gray, McJones et al. (1981) propuseram o paradigma DO-UNDO--REDO no protótipo **System R**. Este protocolo previa guardar informação da operação (DO) que pudesse ser utilizada em caso de falha para refazer os efeitos de transações confirmadas (REDO), bem como para desfazer os efeitos de transações não confirmadas à altura da falha (UNDO). Hadzilacos (1988) chamou resiliência à propriedade de uma base de dados poder ser sempre recuperada a partir de informação guardada em memória estável. Verhofstad (1978) define vários tipos de recuperação, associando-os ao que chamou qualidade da recuperação.

A Figura 11.1 mostra os componentes do SGBD envolvidos na recuperação.

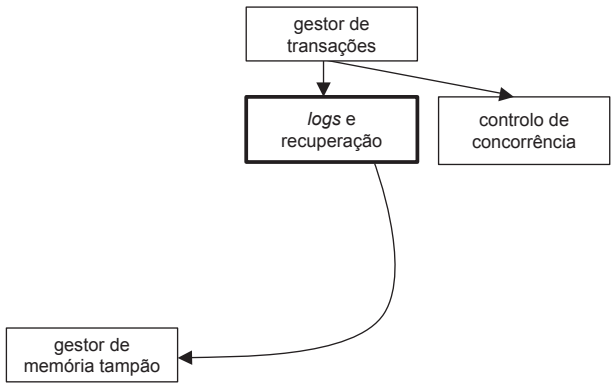


FIGURA 11.1 – GESTOR DE *LOGS* E RECUPERAÇÃO

12

OUTROS MODELOS

"There are a variety of existing and newly-emerging applications that can benefit from data management and processing principles and techniques. At the same time, these applications are very much different from business data processing and from each other there seems to be no obvious way to support them with a single code line. The «one size fits all» theme is unlikely to successfully continue under these circumstances."

M. Stonebraker, U. Çetintemel, "One Size Fits All", in *An Idea Whose Time Has Come and Gone* (2005)

SUMÁRIO

Neste capítulo, descrevemos e analisamos os modelos de bases de dados não relacionais mais importantes e comparamos as suas características principais com as do Modelo Relacional. A maioria dos modelos propostos tem aplicações específicas, embora a sua utilização se comece a generalizar num cenário de grandes volumes de dados cada vez menos estruturados e mais dinâmicos. Algumas das características destes modelos têm sido incorporadas nos produtos relacionais e, sempre que possível, será feita referência a essas propostas. São também discutidos cenários típicos de aplicação para estes modelos, analisando o seu possível desempenho e o seu comportamento. Ao contrário dos produtos relacionais, não existe uma normalização, nem em termos de tecnologia nem em terminologia destes modelos, pelo que tentaremos evitar uma classificação muito rígida dos diferentes produtos e propostas.

12.1 INTRODUÇÃO

Este capítulo descreve e analisa os modelos de bases de dados não relacionais mais utilizados – de uma forma geral, os modelos pertencentes ao movimento que ficou sendo conhecido por NoSQL[1]. Para simplificar, classificamos como pertencendo a esta categoria todos os modelos que não o relacional, independentemente de a linguagem de consulta ser ou não SQL.

As bases de dados não relacionais foram sendo propostas e desenvolvidas para dar resposta a problemas que os produtos baseados no Modelo Relacional apresentavam dificuldade em resolver convenientemente. A necessidade de representar dados com características não adequadas às estruturas relacionais, aliada ao crescimento rápido do volume de

[1] Geralmente associado a *Not Only SQL*.

dados a tratar, motivou o aparecimento de outros modelos, alguns sem as propriedades tradicionais dos SGBD relacionais, como as propriedades ACID ou a linguagem SQL.

Dados com estrutura recursiva, hierárquica ou fortemente conectada não são representados de uma forma natural no Modelo Relacional. Da mesma forma, dados com estrutura variável ao longo do tempo e evolutiva, tais como montagens de produtos complexos, dados hipermédia, dados de sensores e dados geográficos, são cada vez mais comuns e colocam desafios a tal modelo. O tipo de processamento exigido por esses dados também requer operadores que não se encontram em SQL. Atravessar grafos, para operações dedutivas ou outras, por exemplo, são operações comuns que não são facilmente suportadas no Modelo Relacional, principalmente se as quantidades de dados não forem muito grandes.

Existem essencialmente duas características a destacar nos problemas colocados às bases de dados, que condicionam a escolha da tecnologia de armazenamento a utilizar: quando o volume de dados não é muito grande[2] e os dados são estruturados, a tecnologia relacional é geralmente adequada; quando as consultas deixam de ser transacionais e passam a ser analíticas, geralmente aparecem as primeiras limitações de uma solução puramente relacional. Neste caso, os chamados armazéns de dados, assentes nos modelos em colunas, podem permitir um maior desempenho ao agrupar os dados que são necessários numa mesma consulta. De certa forma, temos modelos relacionais com um mecanismo de armazenamento de dados diferente (ver Capítulo 5). O problema do grande volume de dados é também enfrentado com partições dos dados em vários servidores, diminuindo o volume de dados por servidor, mas complicando a junção de dados em servidores diferentes.

Noutras situações, os dados são pouco estruturados, de estrutura variável no tempo ou não são suscetíveis de ser representados em relações. Por exemplo, quando os dados são baseados numa qualquer estrutura de generalização e de especialização, a utilização de herança entre tipos de dados pode ser útil. Como o modelo orientado por objetos fornece esta possibilidade, existem bases de dados orientadas por objetos que são especialmente adequadas para este tipo de modelos e existem também algumas extensões ao Modelo Relacional. Por exemplo, PostgreSQL, DB2 e Oracle oferecem os chamados modelos objeto-relacionais ao permitir relações com herança entre si, métodos e encapsulação.

Em algumas áreas de aplicação, os dados são inerentemente interligados entre si, por exemplo, no caso dos vocabulários para a *Web* semântica e das ontologias. Neste caso, a representação em grafos é natural e a utilização de uma base de dados de grafos pode ser a opção escolhida. Quando a estrutura dos dados é muito complexa e variável, como por exemplo na área da saúde, pode ser benéfico representar diretamente na base de dados essas estruturas sem passar por um processo de normalização[3]. As bases de dados de documentos JSON permitem representar e gerir essas estruturas, dispensando um esquema comum, como no Modelo Relacional. No entanto, na maioria das aplicações continuamos a querer manter a integridade dos dados, os acessos concorrentes, a possibilidade de recuperação e a utilização de uma linguagem simples de acesso aos dados.

[2] A definição do que é um grande volume de dados muda rapidamente. Atualmente, podemos falar de dezenas de *terabyte* de dados.

[3] Por exemplo a norma FHIR, disponível em https://www.hl7.org/fhir/. Qualquer recurso FHIR tem relacionamentos múltiplos com muitos outros recursos, e muitas propriedades, com múltiplos valores, estruturadas.

