UNIP
UNIVERSIDADE PAULISTA

Interativa

Linguagem de Programação de Banco de Dados

Autora: Profa. Vanessa Santos Lessa
Colaboradora: Profa. Larissa Rodrigues Damiani

Professora conteudista: Vanessa Santos Lessa

Doutora em Ciências e Aplicações Geoespaciais pelo Instituto Presbiteriano Mackenzie, mestre em Engenharia Elétrica com ênfase em Inteligência Artificial Aplicada à Automação pelo Centro Universitário da Fundação Educacional Inaciana (FEI). Bacharel em Engenharia da Computação pela Universidade São Judas Tadeu (USJT). Coordenadora do curso de Bacharelado em Ciência da Computação na Universidade Paulista (UNIP) na modalidade Ensino a Distância (EaD). Atua há mais de 18 anos em cargos técnicos e gerenciais na área de computação.

Dados Internacionais de Catalogação na Publicação (CIP)

L638l Lessa, Vanessa Santos.

Linguagem de Programação de Banco de Dados / Vanessa Santos Lessa. – São Paulo: Editora Sol, 2022.

134 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Estrutura. 2. Banco de dados. 3. Gerenciamento. I. Título.

CDU 681.3.07

U516.41 – 22

Sumário

Linguagem de Programação de Banco de Dados

Unidade II

Unidade III

APRESENTAÇÃO

Como cientista da computação, você encontrará desafios que envolvem os dados de uma empresa, como consultas complexas que auxiliam os gestores a tomar importantes decisões. Na análise de dados, é possível integrar aplicativos, sites e softwares utilizando a mesma base de dados ou bases de dados diferentes.

Este livro-texto corresponde a um curso avançado de banco de dados. Assim, o objetivo desta disciplina é familiarizar o aluno com as tecnologias e as metodologias utilizadas para desenvolver aplicações de maior grau de complexidade. Apresentaremos os conceitos de projeto físico, gerenciamento de transações e mecanismos de otimização e controle das atividades em um banco de dados.

Neste livro-texto, você terá a oportunidade de compreender a estrutura da linguagem SQL para acessar banco de dados, ser capaz de decidir pelo melhor conjunto de instruções (algoritmo mais eficiente) a ser utilizado para acesso ao banco de dados e reconhecer a importância do controle de acessos concorrentes, recuperação, segurança e integridade dos dados.

Vale acrescentar que este material é escrito em linguagem simples e direta. Há ainda muitas figuras, que auxiliam no entendimento dos tópicos desenvolvidos. As observações e lembretes são oportunidades para solucionar eventuais dúvidas. Já os saiba mais possibilitam que você amplie seus conhecimentos. Há, ainda, muitos exemplos práticos, que auxiliam a fixação dos assuntos abordados.

Bons estudos.

INTRODUÇÃO

A análise de dados para uma empresa é muito importante, pois possibilita conhecer seu público, desenvolver estratégias mais seguras para lidar com o mercado, prevendo tendências e comportamentos; focar nas metas e melhorar os resultados.

Para análises de dados complexas é essencial compreender a estrutura da linguagem SQL para acessar banco de dados, pensar em algoritmos mais eficientes, decidindo pelo melhor conjunto de instruções a ser utilizado para acesso ao banco de dados e reconhecer a importância do controle de acessos concorrentes, recuperação, segurança e integridade dos dados.

O conteúdo deste livro-texto foi divido em três unidades. Na unidade I, iniciamos com a estrutura de um banco de dados relacional e a arquitetura do sistema. Aprofundaremos o conhecimento da linguagem SQL, o que nos permitirá entender melhor as unidades seguintes. Na unidade II, abordaremos a teoria que lida com consultas em bancos de dados e como melhorar essas consultas. Na unidade III, aprenderemos como gerenciar as transações executadas em um banco de dados para gerenciar as concorrências entre elas.

Espero que você tenha uma boa leitura e se sinta motivado a ler e conhecer mais sobre esta disciplina.

Unidade I

1 ARQUITETURA DO SISTEMA

A arquitetura de um sistema de banco de dados é muito dependente do sistema de computador básico onde ele está sendo executado.

Podemos ter sistemas de banco de dados centralizados, onde o sistema é executado em um único computador, sem interagir com outras máquinas; ou podemos ter sistemas cliente-servidor, no qual uma máquina-servidor executa o trabalho para várias máquinas-cliente. É possível projetar o sistema de banco de dados para aproveitar arquiteturas paralelas de computadores, assim utilizamos várias máquinas distantes geograficamente para ter um sistema de banco de dados distribuído.

1.1 Arquiteturas de sistema de banco de dados

A arquitetura de um banco de dados depende totalmente das definições de hardware e do sistema computacional que será utilizado na estrutura em que se trabalha. É preciso considerar, na análise de um projeto de arquitetura de bancos de dados, o poder de processamento do hardware, a estrutura de rede, a memória e diversos outros componentes.

A partir da história da evolução dos bancos de dados, vamos focar em quatro tipos de arquitetura: centralizada, cliente-servidor, paralela e distribuída.

1.1.1 Arquiteturas centralizadas e cliente-servidor

Arquiteturas centralizadas

Os primeiros bancos de dados foram implementados utilizando arquiteturas centralizadas. Na década de 1970, com a utilização de computadores de grande porte dedicados especialmente para processamento de enormes volumes de dados, conhecidos como mainframes, utilizava-se a implementação centralizada. Os mainframes eram responsáveis por todo o processamento necessário. Os usuários acessavam os dados usando terminais sem capacidade de processamento.

Na arquitetura centralizada, existem duas formas de utilização de sistemas: a multiusuário e a monousuário. Silberschatz, Korth e Sudarshan (2020, p. 542) esclarecem com detalhes essas duas formas.

> Distinguimos duas maneiras como os computadores são usados: como sistemas monousuário e como sistemas multiusuários. Os smartphones e os computadores pessoais caem na primeira categoria. Um sistema

monousuário típico é um sistema usado por uma única pessoa, normalmente com apenas um processador (que pode ter vários núcleos) e um ou dois discos. Um sistema multiusuário típico, por outro lado, possui muitos discos, uma grande quantidade de memória e vários processadores. Esses sistemas atendem a uma grande quantidade de usuários conectados ao sistema remotamente, sendo chamados sistemas servidores.

Arquiteturas cliente-servidor

Com a evolução dos computadores pessoais, as máquinas tornaram-se mais rápidas e com um custo menor, dessa forma o uso da arquitetura centralizada foi diminuindo, pois os terminais sem capacidade de processamento foram substituídos por computadores pessoais, que poderiam desenvolver uma nova funcionalidade. Os computadores pessoais utilizados pelos usuários assumiram o gerenciamento da interface, que antes era responsabilidade do servidor.

Atualmente os sistemas centralizados respondem requisições feitas pelos sistemas-cliente: esse é o conceito de cliente-servidor. A figura 1 representa graficamente esse conceito. As duas máquinas-cliente são conectadas entre si através de uma rede.

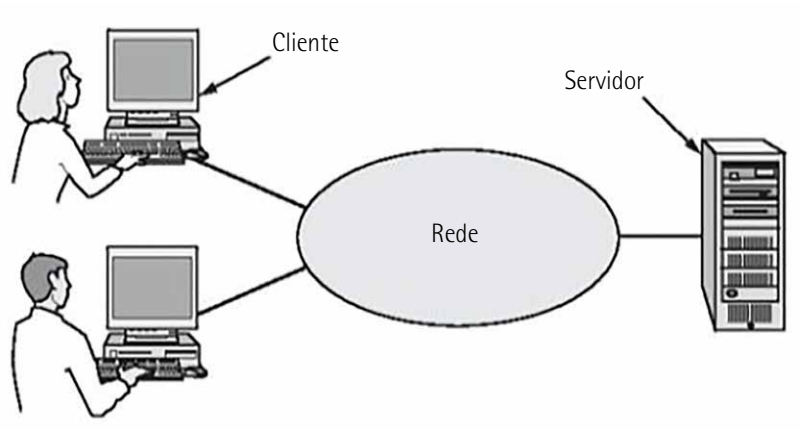

Figura 1 – Uma rede com dois clientes e um servidor

Fonte: Tanenbaum e Wetherall (2011, p. 2).

Saiba mais

O livro *Redes de computadores* apresenta nos seus capítulos iniciais uma introdução sobre redes. Incentivamos você a ler o início desta obra.

TANENBAUM, A. S.; WETHERALL, D. J. *Redes de computadores*. 5. ed. Rio de Janeiro: Pearson Prentice Hall, 2011.

A funcionalidade dos servidores de bancos de dados cliente-servidor pode ser dividida em duas partes:

- **front-end**: faz a interface com o usuário final;
- **back-end**: controla as estruturas de acesso ao dado.

Admitimos a arquitetura das aplicações que usam bancos de dados como back-end, pois fazem o controle de acesso aos dados. A figura 2 apresenta as duas maneiras como uma aplicação de banco de dados normalmente é particionada: duas ou três camadas. As primeiras aplicações de banco de dados utilizavam uma arquitetura de duas camadas, em que a aplicação está na máquina-cliente, que chama a funcionalidade do sistema de banco de dados na máquina-servidora usando instruções da linguagem de consulta.

Os sistemas que possuem uma grande quantidade de usuários utilizam uma arquitetura de três camadas. Silberschatz, Korth e Sudarshan (2020, p. 12) explicam o funcionamento da arquitetura de três camadas.

> As aplicações de banco de dados modernas utilizam uma arquitetura de três camadas, em que a máquina-cliente age meramente como um front-end e não contém quaisquer chamadas de banco de dados diretas; navegadores Web e aplicativos móveis são, atualmente, os clientes de aplicação mais utilizados. O front-end realiza a comunicação com um servidor de aplicação. O servidor de aplicação, por sua vez, se comunica com um sistema de banco de dados para acessar os dados. A lógica de negócios da aplicação, que diz quais ações executar sob quais condições, é incorporada ao servidor de aplicação, em vez de ser distribuída por múltiplos clientes. As aplicações de três camadas oferecem mais segurança, além de melhor desempenho, que as aplicações de duas camadas.

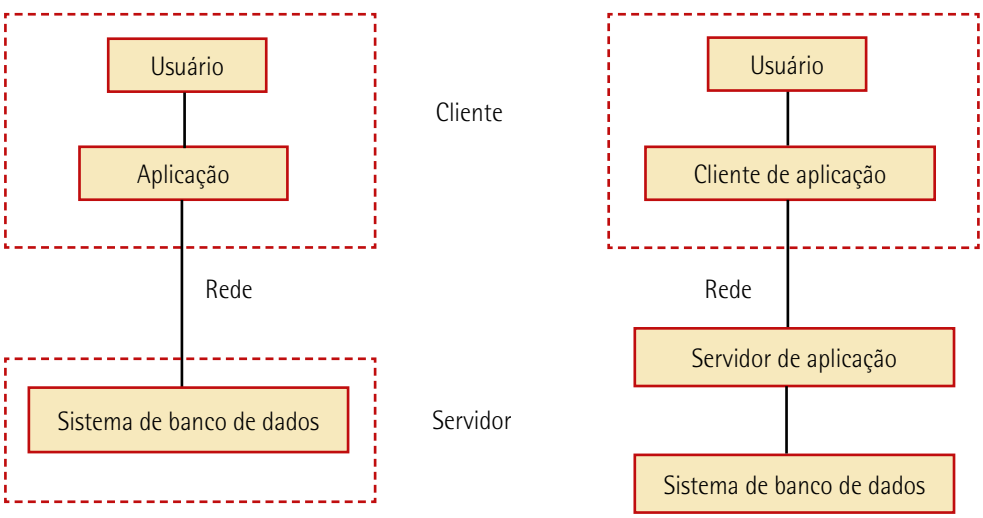

Figura 2 – Arquitetura de duas camadas e três camadas

Fonte: Silberschatz, Korth e Sudarshan (2020, p. 13).

1.1.2 Arquiteturas de sistema servidor

Podemos classificar os sistemas servidores em dois tipos:

- **Sistemas servidores de transação ou servidores de consulta**: disponibilizam uma interface para o cliente, que pode enviar solicitações de ação. O servidor por sua vez executa a ação solicitada pelo cliente e devolve o resultado.

- **Sistemas servidores de dados**: possibilita a interação do cliente com o servidor executando ações como leitura ou atualização de dados, em unidades como arquivos ou páginas.

1.1.3 Sistemas paralelos

O principal objetivo da arquitetura de sistemas paralelos é compartilhar recursos para melhorar o processamento de dados. Os sistemas paralelos utilizam várias CPUs e discos em paralelo para aumentar a velocidade de processamento.

As informações nos cercam em todos os contextos, produzindo uma quantidade enorme de dados em uma velocidade sem precedentes. Os sistemas cliente-servidor não são tão poderosos para lidar com aplicações que necessitam consultar banco de dados extraordinariamente grandes (da ordem de terabytes). Dessa forma, o estudo de sistemas de banco de dados paralelos vem se tornando cada vez mais comum e importante pois as aplicações necessitam consultar esses bancos de dados.

Observação

– um kilobyte (kB) representa 1024 bytes;

– um megabyte (MB) representa 1024 kB;

– um gigabyte (GB) representa 1024 MB;

– um terabyte (TB) representa 1024 GB;

– um petabyte (PB) representa 1024 TB.

No processamento serial, as operações são realizadas de forma sequenciada. De maneira oposta, o processamento paralelo faz diversas operações ao mesmo tempo. Atualmente podemos adquirir no mercado máquinas paralelas com granularidade grossa. Isso significa que a máquina possui uma pequena quantidade de processadores poderosos. Uma máquina paralela com granularidade fina possui milhares de processadores menores. A maioria das máquinas de alto nível atualmente possui um grau de paralelismo com granularidade. As mais comuns trabalham com dois ou quatro processadores.

Podemos analisar o desempenho de um sistema de banco de dados utilizando basicamente duas medidas:

- **throughput**: a quantidade de tarefas que o sistema de banco de dados pode executar completamente em um determinado intervalo de tempo;

- **tempo de resposta**: quanto tempo é necessário para completar uma determinada tarefa a partir do instante em que ela foi submetida.

Considerando um sistema que executa muitas pequenas transações, seu processamento paralelo pode melhorar o throughput. Um sistema que executa grandes transações utilizando o processamento paralelo ganha em tempo de resposta e melhora o throughput, pois executa as subtarefas da transação em paralelo.

1.1.4 Sistemas distribuídos

Os sistemas distribuídos caracterizam-se pela utilização de diferentes máquinas que compartilham recursos conectadas entre si, por meio de uma rede de comunicação, para atingir um objetivo comum ou compartilhado. A diferença entre os sistemas distribuídos e os sistemas paralelos está na localização das máquinas. Nos sistemas paralelos, por exemplo, as máquinas devem estar próximas, já para os sistemas distribuídos, a localização das máquinas é indiferente, podendo estar geograficamente distantes.

Um exemplo mais claro de sistema distribuído é a internet, em que é possível ter uma única tarefa de processamento de dados se estendendo a várias máquinas da rede (DATE, 2003). Silberschatz, Korth e Sudarshan (2020, p. 557) explicam a estrutura de um sistema de banco de dados distribuído:

> Em um sistema de banco de dados distribuído, o banco de dados é armazenado em nós localizados em sítios (sites) separados geograficamente. Os nós em um sistema distribuído se comunicam entre si por meio de diversos meios de comunicação, como redes privadas de alta velocidade ou pela internet. Eles não compartilham memória principal ou discos.

A figura 3 apresenta um exemplo de uma estrutura geral de um sistema distribuído:

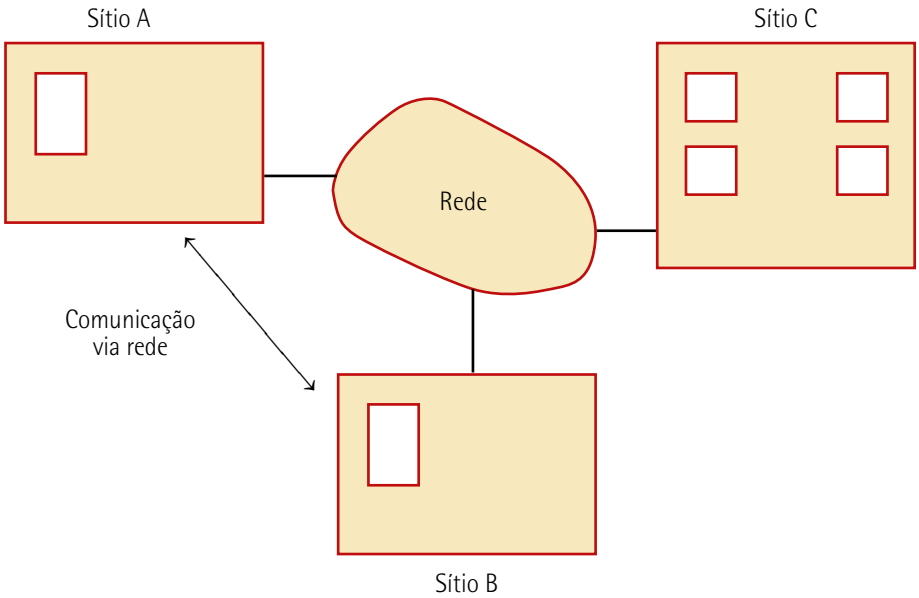

Figura 3 – Sistema distribuído

Fonte: Silberschatz, Korth e Sudarshan (2020, p. 557).

Podemos listar três importantes vantagens de se utilizar um banco de dados distribuídos:

- **compartilhamento de dados**: um usuário em um sítio consegue acessar dados de outro sítio;
- **autonomia**: cada sítio controla os seus dados sendo capaz de administrar a distribuição deles;
- **disponibilidade**: caso os dados sejam replicados entre os sítios, é possível que em caso de falha em um sítio, outro sítio pode assumir a demanda e o sistema continuará disponível.

1.2 Bancos de dados paralelos e dados distribuídos

Na década de 1980, surgiu o sistema de gerência de banco de dados paralelos (SGBDP) com o objetivo de resolver o problema de "gargalo de E/S dos bancos de dados convencionais", devido ao alto custo de acesso aos mecanismos de armazenamento secundário (discos), se comparado com a memória principal. Uma solução foi particionar os dados, utilizando vários discos, que podem ser acessados paralelamente durante uma consulta. Dessa forma, teríamos uma melhora do throughput para cada disco utilizado na estrutura.

Um sistema de gerência de banco de dados distribuído (SGBDD) constitui-se de uma relação de nós (pontos de conexão, como os computadores), onde cada um pode participar da execução de transações que acessam dados em um ou mais nós. Em um SGBDD podemos armazenar os dados em vários computadores que se comunicam entre si, por meio de comunicação como redes sem fio, redes de alta velocidade e memória principal.

As definições de um SGBDP e SGBDD podem se confundir na literatura, pois ambas estão relacionadas com a questão do particionamento. No entanto, podemos observar algumas diferenças entre SGBDP e SGBDD:

Quadro 1 – Diferenças entre SGBDP e SGBDD

SGBDP	SGBDD
Fortemente acoplados (um ponto de controle para fragmentar a consulta e executar em paralelo)	Fracamente acoplados
Distribuição depende da arquitetura dos sistemas	Distribuem os dados conforme sua utilização (princípio da localidade)
Mais adequados para a escalabilidade do sistema	Adequados para aumentar a robustez e a disponibilidade dos dados

Os SGBDD endereçam questões de autonomia, distribuição e heterogeneidade entre os componentes da estrutura, e por sua origem distribuída podem levar ao paralelismo.

1.2.1 Paralelismo de E/S

O paralelismo na entrada e na saída de dados (E/S) tem objetivo de diminuir o tempo de processamento de um programa, particionando as relações sobre múltiplos discos. Podemos listar três tipos de particionamento de dados para conseguirmos o paralelismo de E/S:

- horizontal;
- vertical;
- misto.

Particionamento horizontal

O particionamento horizontal é o tipo mais utilizado. A figura 4 apresenta um exemplo onde cada tupla (uma linha) de uma relação é distribuída entre os três discos, dessa forma cada tupla estará em um disco diferente.

Codigo_cliente	Nome_cliente	Endereço	Dt_nasc	Sexo	Credito
C-001	Monteiro Lobato	Praça Picapau Amarelo	18/04/1882	M	5.000,00
C-002	José de Alencar	Rua Iracema	01/05/1829	M	4.000,00
C-101	Cecília Meireles	Av Espectros	07/11/1901	F	9.000,00
C-102	Carlos Drummond	Av Poemas	31/10/1902	M	7.000,00
C-103	Machado de Assis	Rua Dom Casmurro	21/06/1839	M	4.000,00
C-202	Clarice Lispector	Rua Todos os Contos	10/12/1920	F	8.000,00

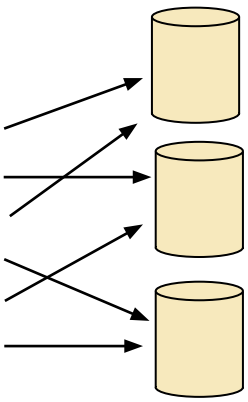

Figura 4 – Particionamento horizontal

Podemos listar três técnicas de particionamento horizontal:

- particionamento Round-Robin (circular);
- particionamento hashing;
- particionamento por Range.

Particionamento Round-Robin (circular)

Distribuindo as tuplas sem distinção entre os dispositivos, cada nova tupla é colocada em um disco diferente. Na figura 5, demonstramos uma distribuição utilizando particionamento Round-Robin, onde a 1ª tupla é colocada no 1º disco, a 2ª tupla é coloca no 2º disco, a 3ª tupla colocada no 3º disco, a 4ª tupla coloca no 1º disco e assim a distribuição continua mantendo um equilíbrio com relação à quantidade de tuplas.

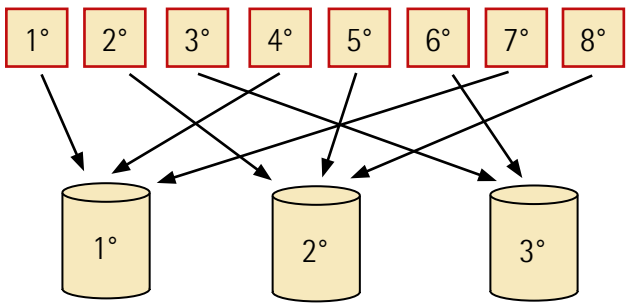

Figura 5 – Particionamento Round-Robin (circular)

Particionamento hashing

Designamos um ou mais atributos do esquema de uma relação para ser utilizado no particionamento. Escolhemos uma função hash (algoritmo transforma dados de comprimento variável para dados de comprimento fixo). Cada tupla da relação original é separada pelo atributo de particionamento, conforme o retorno da função hash utilizada e então a tupla é alocada no disco. Podemos observar na figura 6 que no particionamento hashing os dados ficarão espalhados entre os discos.

Saiba mais

Para saber mais sobre a forma de organizar a entrada de dados aplicando hashing, sugerimos a leitura do subtópico 8.3 do livro:

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de gerenciamento de bancos de dados*. 3. ed. Nova York: McGraw-Hill, 2008.

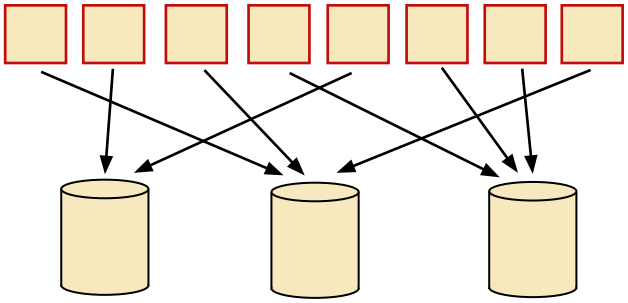

Figura 6 – Particionamento hashing

Particionamento por Range

Para utilizar o particionamento por Range (ou intervalo), selecionamos um atributo para utilizar no particionamento e separamos por faixa. Por exemplo, na figura 7 poderíamos considerar a relação de clientes, e o **nome** do cliente como atributo escolhido. Assim, de A até H seria colocado no 1º disco, de I até P seria colocado no 2º disco e de Q até Z seria colocado no 3º disco.

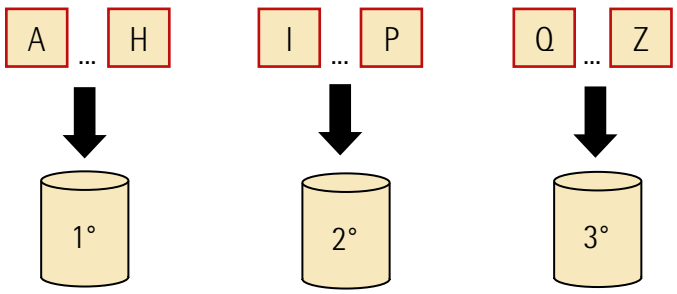

Figura 7 – Particionamento por Range

Particionamento vertical

Na figura 8, exemplificamos o particionamento vertical, no qual os campos de uma relação são separados entre os discos, e cada campo reside em um ou mais discos. Esse tipo de particionamento divide a relação original em um conjunto de relações menores, com o objetivo de diminuir o tempo de execução da aplicação que utiliza esses fragmentos.

A partição vertical é mais difícil de implementar do que a partição horizontal, pois é necessária uma análise estática dos acessos de todos os atributos da relação. Para se reconstruir a relação original, utiliza-se a replicação da chave primária da relação original nas relações menores.

Observação

A chave primária, ou primary key (PK), é o identificador único de uma tupla em uma relação. Podemos usar um atributo para identificar uma tupla (chave simples) ou dois ou mais atributos (chave composta). Por exemplo: para identificar um cliente pessoa física poderíamos utilizar o CPF como chave primária.

Codigo_cliente	Nome_cliente	Endereço	Dt_nasc	Sexo	Credito
C-001	Monteiro Lobato	Praça Picapau Amarelo	18/04/1882	M	5.000,00
C-002	José de Alencar	Rua Iracema	01/05/1829	M	4.000,00
C-101	Cecília Meireles	Av Espectros	07/11/1901	F	9.000,00
C-102	Carlos Drummond	Av Poemas	31/10/1902	M	7.000,00
C-103	Machado de Assis	Rua Dom Casmurro	21/06/1839	M	4.000,00
C-202	Clarice Lispector	Rua Todos os Contos	10/12/1920	F	8.000,00

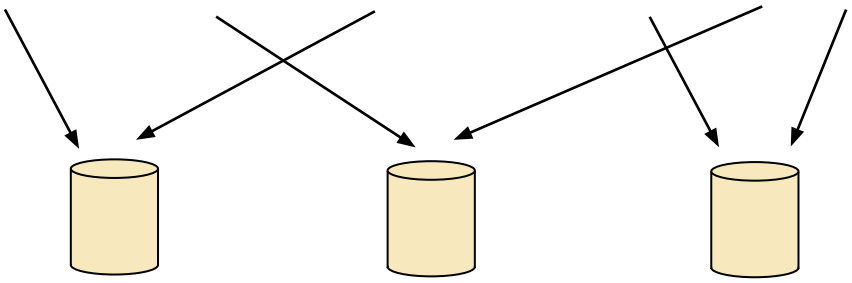

Figura 8 – Particionamento vertical

Particionamento misto

Em algumas situações, a utilização de apenas um tipo de particionamento horizontal ou vertical não satisfaz os acessos da aplicação. Para essas situações existem dois tipos de particionamento misto:

- **HV**: primeiro executamos uma partição horizontal na relação, em seguida, nos dados fragmentos, executamos uma partição vertical.

- **VH**: primeiro executamos uma partição vertical na relação, em seguida, nos dados fragmentos, executamos uma partição horizontal.

Em um banco de dados paralelo, há dois tipos de paralelismo possíveis para o processamento de consultas:

- **Interconsultas**: quando utilizamos esse tipo de consulta, o throughput aumenta, pois podemos executar distintas transações ou consultas em paralelo. Dessa forma, obtemos ganho de escala nos sistemas que processam transações, que pode receber uma quantidade maior de transações por segundo.

- **Intraconsultas**: consultas de longa duração são beneficiadas, pois executamos uma única consulta em paralelo, utilizando diversos discos e processadores, o que melhora o tempo de resposta das consultas.

1.2.2 Banco de dados distribuídos homogêneos e heterogêneos

Diferentemente dos sistemas paralelos, em que os processadores trabalham juntos com um único sistema de banco de dados, um sistema distribuído é composto por vários sites pouco acoplados, no qual cada site possui um grau de independência.

Banco de dados distribuído homogêneo

O banco de dados distribuído homogêneo é mais fácil de desenvolver e de gerenciar. Os sites utilizam os mesmos softwares e cooperam no processamento das requisições do usuário, entregando parte de sua autonomia com relação às modificações do esquema ou do software. Um sistema gerenciador de banco de dados (SGBD) homogêneo apresenta-se como um sistema único para o usuário.

Para o SGBD homogêneo, alguns requisitos precisam ser atendidos como: as estruturas de dados e o aplicativo de banco de dado usados em cada local devem ser iguais ou compatíveis.

Banco de dados distribuído heterogêneo

Diferentemente dos bancos de dados distribuídos homogêneos, nos bancos de dados distribuídos heterogêneos podemos encontrar diferentes esquemas e diferentes softwares distribuídos entre os sites. Essa diferença causa problemas para processar as consultas e as transações, pois os sites podem fornecer poucos recursos para compartilhar o processamento de transações. Além disso, podemos encontrar incompatibilidade de hardware, software e estrutura de dados entre os diferentes nós.

Para utilizar sistemas heterogêneos, precisamos executar traduções para permitir a comunicação entre os diversos sites. Nesse sistema, os usuários devem conseguir fazer requisições em um idioma de banco de dados em seus sites locais (geralmente em Structured Query Language – SQL). O sistema heterogêneo normalmente não é viável técnica ou economicamente.

Os usuários devem utilizar o sistema como um sistema lógico, uma distribuição transparente. Cada transação deve garantir a integridade de todos os bancos de dados, transação transparente. Cada transação precisa ser dividida em subtransações e cada uma afeta um sistema de banco de dados.

Observação

O termo usuários utilizado nessa seção refere-se aos usuários finais ou programadores de aplicações que executam as operações de manipulação de dados. Essas operações devem manter-se logicamente inalteradas, diferentemente das operações de definição de dados, que necessitarão de extensão em um sistema distribuído.

Os usuários em um sistema distribuído devem proceder como se o sistema não fosse distribuído. Date (2003, p. 558) apresenta 12 importantes regras básicas de SGBD distribuído.

- **Autonomia local**: qualquer nó que compõe o sistema distribuído precisa ser independente dos outros nós. Qualquer nó deve fornecer mecanismos de segurança, bloqueio, acesso, integridade e recuperação após falha.

- **Não dependência de um nó central**: um sistema de banco de dados distribuído não precisa de um nó central, pois um único ponto de falha afetaria todos os demais nós. Um nó central poderia ficar sobrecarregado, resultando em perda de desempenho do sistema.

- **Operação contínua**: um sistema de banco de dados distribuído em nenhum momento deve ser desativado. As operações de backup e recuperação precisam acontecer on-line. Essas operações precisam ser rápidas o bastante, com o objetivo de não afetarem o funcionamento do sistema (backup incremental, por exemplo).

- **Transparência/independência de localização**: os usuários do sistema não precisam conhecer o local onde estão armazenados os dados; precisam proceder como se os dados estivessem armazenados localmente.

- **Independência de fragmentação**: as tabelas de um sistema de banco de dados distribuído podem apresentar-se divididas em fragmentos, localizados fisicamente em diversos nós, mas de forma transparente para o usuário.

- **Independência de replicação**: dados podem estar replicados em diversos nós da rede, de forma transparente. As réplicas de dados precisam ser preservadas sincronizadas automaticamente pelo SGBD.

- **Processamento de consultas distribuído**: o desempenho de uma consulta não deve depender do local onde ela é submetida. Um SGBDD é capaz de selecionar o melhor caminho para o acesso a um determinado nó da rede, e otimiza o desempenho de uma consulta distribuída, considerando a localização dos dados, utilização de CPU, I/O e o tráfego na rede.

- **Gerenciamento de transações distribuídas**: um SGBDD precisa suportar transações atômicas. As propriedades ACID (atomicidade, consistência, independência e durabilidade) das transações e a serialização precisam ser suportadas para as transações locais e para as transações distribuídas.

- **Independência de hardware**: um SGBDD precisa funcionar e acessar dados em diferentes plataformas de hardware, independentemente de uma em específico.

- **Independência de sistema operacional**: um SGBDD precisa funcionar em sistemas operacionais diferentes. Não deve depender de um sistema operacional em especial.

- **Independência de rede**: um SGBDD tem de ser projetado para funcionar independentemente do protocolo de comunicação e da topologia de rede usada para interligar os diversos nós que compõem a rede.

- **Independência de SGBD**: um SGBDD precisa ter capacidades de comunicação com outros sistemas de gerenciamento de banco de dados que são executados em nós diferentes, independentemente dos sistemas de bancos de dados (heterogêneos).

1.2.3 Armazenamento de dados distribuídos

Admitindo uma relação que precisa ser salva no banco de dados, podemos salvá-la de duas formas diferentes no banco de dados distribuído:

- **Replicação**: o sistema mantém cópias iguais da relação e salva cada réplica em um site diferente.

- **Fragmentação**: o sistema divide a relação em diversos fragmentos e salva cada fragmento em sites diferentes.

1.2.4 Transações distribuídas

As transações distribuídas são aquelas que precisam de mais de um servidor. As transações podem ser locais, quando executadas apenas em um banco de dados local; ou globais, quando acessam e atualizam dados em diferentes bancos de dados locais. Os servidores que participam de uma transação distribuída devem colaborar entre si quando uma transação for efetivada ou cancelada. Caso aconteça o cancelamento de uma transação, os servidores possuem um software de gerenciamento de recuperação que garante a recuperação quando ocorre uma falha.

Os servidores que participam da transação distribuída precisam avisar no momento que a transação é efetivada. A comunicação dos servidores é organizada pelo coordenador de transação distribuída. Um servidor começa uma transação distribuída mandando uma solicitação de transação para o coordenador de transação distribuída. Por sua vez, o coordenador abre a transação distribuída e devolve seu identificador para o servidor que fez a solicitação. O coordenador que abriu a transação distribuída é responsável por executar ou cancelar a transação.

1.2.5 Protocolos commit

A atomicidade de uma transação obriga que em seu final todas as operações devem ser realizadas em grupo ou que nenhuma operação seja efetivada. O gerenciador de transação administra a execução das transações (ou subtransações) que precisam acessar os dados armazenados em um site local. O protocolo de efetivação de uma fase permite que uma transação seja efetivada ou cancelada quando o coordenador da transação emite uma mensagem para todos os participantes da transação, solicitando a efetivação ou cancelamento. O coordenador fica repetindo o envio da mensagem até que todos os participantes confirmem que concluíram suas operações. Este é um exemplo de protocolo de efetivação de uma fase.

Os protocolos commit (encerra a transação e salva permanentemente todas as alterações) mais usados são:

- 2PC (protocolo commit de duas fases);
- 3PC (protocolo commit de três fases).

2PC (protocolo commit de duas fases)

O protocolo de efetivação de duas fases permite que qualquer servidor participante cancele uma transação. Devido à atomicidade de uma transação, quando uma operação da transação é cancelada, a transação inteira é cancelada. É tudo ou nada. Entretanto, o protocolo de efetivação de duas fases permite que um servidor participante possa cancelar apenas uma parte da transação.

Considere uma transação T que inicia no site Si, o coordenador da transação Ci e o Gi o gerenciador de transação.

- **1ª fase (prepare)**: cada participante vota na transação a ser efetivada ou cancelada.

1. Ci acrescenta o registro <prepare T> ao log;
2. Ci envia uma mensagem prepare T para todos os sites onde T será executada;
3. Gi determina se deseja confirma sua parte T;
   - Caso negativo, acrescenta o registro <no T> e responde aborte T para Ci.
   - Caso positivo, acrescenta o registro <ready T>, força o log para o armazenamento estável e responde ready T para Ci.

- **2ª fase (commit)**: os participantes decidem em conjunto se a transação será efetivada.

1. Ci recebe respostas de todos os sites à mensagem prepare T;
2. Ci determina se T será confirmada ou abortada;
   - Ci confirma a transação T, caso tenha recebido a mensagem ready T de todos os sites participantes de T e acrescenta o registro <commit T> e força o log para armazenamento estável;
   - Ci aborta a transação T, caso não tenha recebido a mensagem ready T de todos os sites participantes de T e acrescenta o registro <abort T> e força o log para armazenamento estável;
3. Ci envia a mensagem de commit T ou abort T para todos os sites participantes;
4. Os sites registram a mensagem de log.

Algumas falhas podem acontecer e o protocolo 2PC responde a elas da seguinte forma:

- **Falha de um site participante**: se Ci identificar a falha do site na 1ª fase, ele considera a mensagem abort T enviada. Se Ci recebeu mensagem ready T e depois aconteceu a falha do site, Ci ignora a falha do site e executa o restante do protocolo commit normalmente.

- **Falha do coordenador**: caso o coordenador falhe durante o protocolo commit, os sites participantes devem decidir o futuro da transação T.

  - **Confirmar T**: se o site receber um registro <commit T>, então T é confirmada;

  - **Abortar T**: se o site receber um registro <abort T>, então T é abortado. Outra situação é se não tiver o registro de <ready T> em seu log, significa que Ci falhou e pode não ter decidido se confirma T, dessa forma a melhor decisão é abortar.

  - **Aguardar o retorno de Ci**: caso o site tenha um registro <ready T> mas não tenha um registro o controle adicional <abort T> ou <commit T> devido a falha de Ci, os sites não podem decidir e devem aguardar a recuperação de Ci.

- **Partição da rede**: caso uma rede seja dividida, podemos encontrar duas situações:

  - se todos os participantes estão na mesma partição a falha não tem impacto sobre o protocolo commit;

  - se os participantes estão distribuídos entre as partições, a perspectiva dos sites de uma partição é que os outros sites que estão em outra partição falharam. Os sites que estão em uma partição diferente da partição do coordenador irão executar protocolos de falha do coordenador.

O protocolo 2PC tem uma desvantagem em relação a possíveis falhas do coordenador quando existe partição da rede. A decisão dos sites de abortar T ou confirmar T até que Ci se recupere pode gerar um bloqueio.

3PC (protocolo commit de três fases)

O protocolo commit de três fase deriva do protocolo de duas fases e impede a situação de bloqueio, isto é, estende o protocolo para permitir que seja possível contornar as falhas do coordenador.

Uma terceira fase é adicionada, na qual vários sites são responsáveis pela decisão de confirmação da transação. Antes de confirmar uma transação, o Ci garante que outros sites fiquem sabendo que ele pretende confirmar T. Caso Ci falhe, os sites ativos escolhem um outro coordenador, que verifica a situação do protocolo e se algum site tiver recebido de Ci a intenção de confirmar T (antes da falha), o novo coordenador garante a intenção e commit T, caso contrário ele aborta.

2 BANCOS DE DADOS RELACIONAIS

Os bancos relacionais são os bancos mais utilizados no mundo. Em 1970 Edgar Frank Codd definiu o modelo relacional, que continua dominante no mercado. Codd (1970) propôs uma nova forma de pensamento, que desconectou a estrutura lógica do banco de dados do método de armazenamento físico, dessa maneira a organização dos dados poderia ser tratada considerando um conjunto de relações.

O propósito desse tópico é apresentar os fundamentos e os conceitos dos bancos relacionais. Vamos começar com o modelo relacional e a estrutura dos bancos de dados relacionais até o projeto de um banco de dados relacional.

2.1 Modelo relacional

O modelo de dados considera diversas ferramentas conceituais para descrever os dados. A relação entre os dados, sua semântica, as restrições e a consistência podem propor uma maneira de descrever um projeto de um banco de dados.

O modelo relacional criado por Codd (1970) é baseado no modelo de dados para aplicações de processamento de dados. Utilizamos um conjunto de tabelas para caracterizar os dados e as relações entre eles, em que cada tabela pode conter diversas colunas e cada coluna receber um nome único. O modelo relacional é baseado na estrutura de registros de formato fixo de vários tipos. As tabelas possuem registros (tipos próprios), e cada registro tem um número fixo de atributos (campos). Os atributos correspondem às colunas das tabelas.

Saiba mais

O artigo de Codd (1970), intitulado "A relational model of data for large shared data banks", apresenta nas suas seções uma discussão sobre os modelos de dados e certas operações sobre relações aplicadas aos problemas de redundância e consistência. Incentivamos você a ler esta obra.

CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, Nova York, v. 13, n. 6, p. 377-38, 1970. Disponível em: https://bit.ly/3D40K0N. Acesso em: 18 out. 2022.

2.1.1 Estrutura dos bancos de dados relacionais

A estrutura básica de um banco de dados relacional é um conjunto de tabelas, no qual cada uma possui um nome único. Uma linha em uma tabela caracteriza a relação entre um conjunto de valores, de modo que uma tabela tem várias relações. Existe uma analogia entre o conceito de tabela e o conceito matemático de relação.

Vamos analisar a tabela 1. Ela apresenta três cabeçalhos de coluna: codigo_cliente, nome_cliente e credito. Considerando a terminologia do modelo relacional, denominamos esses cabeçalhos como atributos, em que cada atributo possui um conjunto de valores possíveis, chamado de domínio do atributo. Por exemplo, para o atributo crédito o domínio é o conjunto dos números positivos. Dessa forma temos:

- **D1**: o conjunto de todos os códigos de clientes.
- **D2**: o conjunto de todos os nomes de clientes.
- **D3**: o conjunto de todos os créditos.

Tabela 1 – A relação cliente

Codigo_cliente	Nome_cliente	Credito
C-001	Monteiro Lobato	5.000,00
C-002	José de Alencar	4.000,00
C-101	Cecília Meireles	9.000,00
C-102	Carlos Drummond	7.000,00
C-103	Machado de Assis	4.500,00
C-202	Clarice Lispector	8.000,00

Qualquer linha da tabela cliente deve consistir em uma tupla de 3 (v_1, v_2 e v_3).

- V_1 é um código do cliente, isto é, está no domínio de D1;
- V_2 é um nome do cliente, isto é, está no domínio de D2;
- V_3 é um crédito, isto é, está no domínio de D3.

Lembrete

Cada linha formada por uma lista ordenada de colunas representa um registro ou tupla. Por exemplo, na tabela 1 temos seis tuplas na relação cliente.

Geralmente o cliente conterá um subconjunto do conjunto de todas as linhas possíveis (D1 × D2 × D3). Uma tabela de n atributos precisará conter um subconjunto (D1 × D2 ×...Dn-1 × Dn).

Precisamos de uma maneira para identificar uma tupla dentro de uma relação. Para isso acontecer, os valores de atributo de uma tupla precisam permitir a identificação unicamente da tupla, isto é, nenhum par de tuplas em uma relação poderá ter o mesmo valor em todos os atributos. Usaremos o nome chave primária para o atributo escolhido pelo projetista do banco de dados como principal forma

de identificar tuplas dentro de uma relação. Considerando a tabela 1, o atributo codigo_cliente poderia ser uma chave primária. Nenhum outro par de tuplas poderia ter o mesmo valor. O uso de chaves representa uma restrição constante, caracterizando uma modelagem na empresa real.

2.1.2 Operações fundamentais da álgebra relacional

Temos dois grupos de operações fundamentais da álgebra relacional. No primeiro, chamado de operações unárias pois operam em uma relação, temos as operações de seleção, projeção e renomeação. No segundo grupo, chamado de operações binárias pois operam em pares de relações, temos as operações de união, diferença de conjunto e produto cartesiano.

Operação de seleção

Utilizamos a operação de seleção quando precisamos apresentar tuplas que satisfaçam um determinado predicado. A letra grega sigma (σ) é usada para denotar seleção. Por exemplo, podemos considerar a tabela 1 para encontrar todas as tuplas em que o crédito do cliente seja maior que 7500, escrevendo:

$$\sigma_{\text{credito} > 7500}(\text{cliente})$$

Para comparações, podemos usar =, ≠, <, ≤, >, ≥ no predicado da seleção. Podemos conjugar diversos predicados em um predicado maior, usando os conectivos e (∧), ou (∨) e não ().

A tabela 2 apresenta o resultado da consulta dos clientes com crédito maior que 7500.

Tabela 2 – Resultado de $\sigma_{\text{credito} > 7500}$ (cliente)

Codigo_cliente	Nome_cliente	Credito
C-101	Cecília Meireles	9.000,00
C-202	Clarice Lispector	8.000,00

Operação de projeção

Utilizamos a operação de projeção quando precisamos selecionar colunas específicas de uma relação. Como exemplo, podemos aplicar a operação de projeção caso seja necessário listar todos os códigos dos clientes e seus valores de crédito. A letra grega pi (π) é utilizada para projeção. Relacionamos todos os atributos que precisamos que apareçam no resultado como um subscrito de π, seguindo a relação de argumento entre parênteses:

$$\pi_{\text{codigo\_cliente, credito}}(\text{cliente})$$

A tabela 3 apresenta a projeção da relação de clientes listando os atributos codigo_cliente e credito.

Tabela 3 – Resultado de $\pi_{codigo\_cliente,\ credito}$ (cliente)

Codigo_cliente	Credito
C-001	5.000,00
C-002	4.000,00
C-101	9.000,00
C-102	7.000,00
C-103	4.500,00
C-202	8.000,00

Operação de renomeação

Os resultados das expressões de álgebra relacional não recebem um nome para serem referenciados. Dessa forma, utilizamos o operador de renomeação indicado pela letra grega minúscula ro (ρ). Podemos utilizar a renomeação para autorrelacionamento. Considere uma expressão de álgebra relacional E.

$\rho_X(E)$

Retorne o resultado da expressão E sob o nome de x. Por exemplo, para renomear a relação cliente para comprador podemos utilizar a expressão a seguir:

$\rho_{comprador}(cliente)$

Operação de união

Considere uma consulta para encontrar os nomes de todos os clientes que fizeram um pedido ou uma cotação. Podemos observar que a relação clientes (tabela 1) não possui essas informações. Para responder essa consulta, precisamos das informações dos pedidos (tabela 4) e das cotações (tabela 5).

Tabela 4 – Relação pedido

Codigo_cliente	Nome_cliente	Valor_pedido
C-001	Monteiro Lobato	1.000,00
C-101	Cecília Meireles	800,00
C-202	Clarice Lispector	2.000,00

Tabela 5 – Relação cotação

Codigo_cliente	Nome_cliente	Valor_cotacao
C-001	Monteiro Lobato	600,00
C-002	José de Alencar	1.500,00
C-101	Cecília Meireles	1.000,00
C-102	Carlos Drummond	700,00

Sabemos como encontrar o nome de todos os clientes que fizeram pedido, usando a operação de projeção:

$\pi_{nome\_cliente}$ (pedido)

Sabemos também como utilizar a projeção para encontrar o nome dos clientes que fizeram cotação:

$\pi_{nome\_cliente}$ (cotação)

Para responder à consulta e encontrar os nomes de todos os clientes que fizeram um pedido ou uma cotação, vamos precisar da união desses dois conjuntos, isto é, precisamos dos nomes de todos os clientes que aparecem em uma ou em ambas as relações. Como na teoria dos conjuntos, utilizando a operação binária união $\cup$, temos:

$\pi_{nome\_cliente}$ (pedido) $\cup$ $\pi_{nome\_cliente}$ (cotação)

A relação resultante está tabela 6. Podemos observar que existem cinco tuplas no resultado, ainda que tenhamos três clientes distintos em pedidos e quatro em cotação. Essa diferença ocorre porque, como as relações são conjuntos, os valores duplicados são eliminados.

Tabela 6 – Relação resultante de $\pi_{nome\_cliente}$ (pedido) $\cup$ $\pi_{nome\_cliente}$ (cotação)

Nome_cliente
Monteiro Lobato
José de Alencar
Cecília Meireles
Carlos Drummond
Clarice Lispector

Operação de diferença de conjunto

A operação de diferença de conjunto (r - s) é utilizada quando queremos encontrar tuplas que estão em uma relação, mas não estão em outra. Por exemplo, podemos encontrar todos os clientes que fizeram cotação, mas não fizeram pedido, escrevemos:

$\pi_{nome\_cliente}$ (cotação) - $\pi_{nome\_cliente}$ (pedido)

O resultado dessa consulta está na tabela 7.

Tabela 7 – Relação resultante de $\pi_{nome\_cliente}$ (cotação) - $\pi_{nome\_cliente}$ (pedido)

Nome_cliente
José de Alencar
Carlos Drummond

Operação de produto cartesiano

Para combinar duas relações, utilizamos a operação produto cartesiano, representada pelo sinal de vezes (×), temos $r_1 \times r_2$. Por exemplo, o esquema da relação de r = cliente × pedido está representado na tabela 8, onde temos:

(cliente.codigo_cliente, cliente.nome_cliente, credito, pedido.codigo_cliente, pedido.nome_cliente, valor_pedido)

Podemos distinguir cliente.cod_cliente de pedido.cod_cliente, assim como podemos distinguir cliente. nome_cliente de pedido.nome_cliente. Para os atributos que não estão nas duas relações, podemos omitir o nome da relação. O resultado do produto cartesiano de duas relações contém todas as combinações possíveis entre os elementos de r_1 e r_2. O número de colunas é igual a soma das quantidades de colunas das duas relações iniciais, e o número de linhas será o produto do número de linhas das relações iniciais. Podemos observar que a relação cliente possui seis linhas e três colunas e a relação pedido possui três linhas e três colunas. No resultado, na tabela 8, temos 18 linhas (6 x 3) e 6 colunas (3 + 3).

Tabela 8 – Relação resultante de cliente × pedido

Cliente.codigo_cliente	Cliente.nome_cliente	Credito	Pedido.codigo_cliente	Pedido.nome_cliente	Valor_pedido
C-001	Monteiro Lobato	5.000,00	C-001	Monteiro Lobato	1.000,00
C-001	Monteiro Lobato	5.000,00	C-101	Cecília Meireles	800,00
C-001	Monteiro Lobato	5.000,00	C-202	Clarice Lispector	2.000,00
C-002	José de Alencar	4.000,00	C-001	Monteiro Lobato	1.000,00
C-002	José de Alencar	4.000,00	C-101	Cecília Meireles	800,00
C-002	José de Alencar	4.000,00	C-202	Clarice Lispector	2.000,00
C-101	Cecília Meireles	9.000,00	C-001	Monteiro Lobato	1.000,00
C-101	Cecília Meireles	9.000,00	C-101	Cecília Meireles	800,00
C-101	Cecília Meireles	9.000,00	C-202	Clarice Lispector	2.000,00
C-102	Carlos Drummond	7.000,00	C-001	Monteiro Lobato	1.000,00
C-102	Carlos Drummond	7.000,00	C-101	Cecilia Meireles	800,00
C-102	Carlos Drummond	7.000,00	C-202	Clarice Lispector	2.000,00
C-103	Machado de Assis	4.500,00	C-001	Monteiro Lobato	1.000,00
C-103	Machado de Assis	4.500,00	C-101	Cecília Meireles	800,00
C-103	Machado de Assis	4.500,00	C-202	Clarice Lispector	2.000,00
C-202	Clarice Lispector	8.000,00	C-001	Monteiro Lobato	1.000,00
C-202	Clarice Lispector	8.000,00	C-101	Cecília Meireles	800,00
C-202	Clarice Lispector	8.000,00	C-202	Clarice Lispector	2.000,00

Suponha que queremos encontrar os nomes e os créditos de todos os clientes que apresentem valor do pedido maior ou igual a 1000,00. Para isso, são necessárias as informações da relação cliente e pedido. Podemos iniciar escrevendo a seleção:

$\sigma_{valor\_pedido >= 1000}$ (cliente x pedido)

Então teremos o resultado apresentado na relação da tabela 9. Temos uma relação de valores de pedido maior ou igual a 1000,00. Entretanto, a coluna cliente.nome_cliente pode conter clientes que não possuem um pedido, pois o produto cartesiano considera todos os pares possíveis de uma tupla de cliente com uma tupla de pedido.

Tabela 9 – Relação resultante de $\sigma_{valor\_pedido >= 1000}$ (cliente × pedido)

Cliente.codigo_cliente	Cliente.nome_cliente	Credito	Pedido.codigo_cliente	Pedido.nome_cliente	Valor_pedido
C-001	Monteiro Lobato	5.000,00	C-001	Monteiro Lobato	1.000,00
C-001	Monteiro Lobato	5.000,00	C-202	Clarice Lispector	2.000,00
C-002	José de Alencar	4.000,00	C-001	Monteiro Lobato	1.000,00
C-002	José de Alencar	4.000,00	C-202	Clarice Lispector	2.000,00
C-101	Cecília Meireles	9.000,00	C-001	Monteiro Lobato	1.000,00
C-101	Cecília Meireles	9.000,00	C-202	Clarice Lispector	2.000,00
C-102	Carlos Drummond	7.000,00	C-001	Monteiro Lobato	1.000,00
C-102	Carlos Drummond	7.000,00	C-202	Clarice Lispector	2.000,00
C-103	Machado de Assis	4.500,00	C-001	Monteiro Lobato	1.000,00
C-103	Machado de Assis	4.500,00	C-202	Clarice Lispector	2.000,00
C-202	Clarice Lispector	8.000,00	C-001	Monteiro Lobato	1.000,00
C-202	Clarice Lispector	8.000,00	C-202	Clarice Lispector	2.000,00

Sabemos que se um cliente tiver um pedido, então existe alguma tupla em cliente × pedido que contenha seu código, e cliente.codigo_cliente = pedido.codigo_cliente. Podemos escrever:

$\sigma_{cliente.codigo\_cliente = pedido.codigo\_cliente}$ ($\sigma_{valor\_pedido >= 1000}$ (cliente × pedido))

Obtemos apenas as tuplas de cliente × pedido que pertencem aos clientes que tenham pedido com valor maior igual a 1000,00, conforme tabela 10.

Tabela 10 – Resultante de $\sigma_{cliente.codigo\_cliente = pedido.codigo\_cliente}$ ($\sigma_{valor\_pedido >= 1000}$ (cliente × pedido))

Cliente.codigo_cliente	Cliente.nome_cliente	Credito	Pedido.codigo_cliente	Pedido.nome_cliente	Valor_pedido
C-001	Monteiro Lobato	5.000,00	C-001	Monteiro Lobato	1.000,00
C-202	Clarice Lispector	8.000,00	C-002	Clarice Lispector	2.000,00

Finalmente, como queremos apenas o nome e o crédito do cliente, podemos utilizar a projeção e escrever:

$\pi_{nome\_cliente,\ credito}$ ($\sigma_{cliente.codigo\_cliente = pedido.codigo\_cliente}$ ($\sigma_{valor\_pedido >= 1000}$ (cliente × pedido))

O resultado dessa expressão, mostrado na tabela 11, é a resposta da consulta.

Tabela 11 – Resultante de $\pi_{nome\_cliente,\ credito}\ (\sigma_{cliente.codigo\_cliente\ =\ pedido.codigo\_cliente}\ (\sigma_{valor\_pedido\ >=\ 1000}\ (cliente \times pedido))$

Cliente.nome_cliente	Credito
Monteiro Lobato	5.000,00
Clarice Lispector	8.000,00

2.1.3 Valores nulos

O valor especial nulo significa valor desconhecido ou inexistente. Os valores nulos também indicam valores não aplicáveis ou a serem adicionados posteriormente. Por exemplo, é possível que o valor de crédito de um cliente não seja conhecido quando o cliente faz seu cadastro, pois poderá ser calculado posteriormente. Geralmente há mais de uma forma possível para tratarmos valores nulos. Recomenda-se, quando possível, evitar operações de comparação utilizando valores nulos.

Considerando as operações aritméticas (+, -, *, /), quando utilizados valores nulos, o resultado deve ser nulo. Considerando as comparações (<, <=, >, >=, ≠) que têm um valor nulo, são consideradas para o valor especial desconhecido; não podemos afirmar que o resultado é verdadeiro ou falso, então o resultado será valor desconhecido.

As expressões envolvendo nulos podem ocorrer em expressões booleanas e precisamos definir como as três operações booleanas devem lidar com isso. A tabela 12 mostra os resultados da utilização do operador E (AND) para duas expressões booleanas.

Tabela 12 – Operador booleano AND com valor desconhecido

Expressão 1	Expressão 2	Resultado
Verdadeiro	Desconhecido	Desconhecido
Falso	Desconhecido	Falso
Desconhecido	Desconhecido	Desconhecido

A tabela 13 mostra os resultados da utilização do operador OU (OR) para duas expressões booleanas considerando valor desconhecido.

Tabela 13 – Operador booleano OR com valor desconhecido

Expressão 1	Expressão 2	Resultado
Verdadeiro	Desconhecido	Verdadeiro
Falso	Desconhecido	Desconhecido
Desconhecido	Desconhecido	Desconhecido

O operador booleano não (NOT) resulta: (não DESCONHECIDO) = DESCONHECIDO.

2.1.4 Modificação do banco de dados

Executamos modificações no banco de dados utilizando operações de atribuição. Vamos estudar como inserir, remover e alterar os dados no banco de dados.

Inserção

Para inserir uma tupla em uma relação, os valores de atributos inseridos devem ser do domínio do atributo. A álgebra relacional representa uma inserção, considerando a relação (r) e uma expressão (E), fazendo E ser uma relação que contém uma tupla. Podemos expressar por:

$r \leftarrow r \cup E$

Por exemplo, precisamos inserir um novo cliente na relação cliente, podemos escrever:

cliente $\leftarrow$ cliente $\cup$ {("C-301", "Mauricio de Sousa",3000)}

Exclusão

Para excluir uma tupla em uma relação, excluímos apenas tuplas inteiras, não excluímos valores em atributos específicos. A expressão na álgebra relacional é:

$r \leftarrow r - E$

Por exemplo, precisamos excluir o cliente com código "C-301", podemos escrever:

cliente $\leftarrow$ cliente – $\sigma_{\text{codigo\_cliente = "C-301"}}$ (cliente)

Atualização

Para atualizar um valor em uma tupla sem modificar todos os outros atributos, podemos usar o operador de projeção e escrever a expressão:

$r \leftarrow \pi_{F1, F2, \ldots Fn}$ (r)

onde Fi (somente constantes e atributos de r, quando teremos o novo valor para o atributo). Cada Fi ou é o i-ésimo atributo de r, se seu valor não é modificado, ou temos uma expressão para o valor do atributo modificado. Para ilustrar melhor, suponha que desejamos aumentar em 10% o valor de crédito de todos os clientes. Podemos escrever:

cliente $\leftarrow \pi_{\text{codigo\_cliente, nome\_cliente, credito * 1,10}}$ (cliente)

2.2 Projeto de banco de dados relacional

Um projeto de banco de dados relacional consiste em uma coleção de esquemas de relação que possibilita a recuperação dos dados sem dificuldade e o armazenamento de dados sem redundância. Quando o projeto respeita uma forma normal adequada, conseguimos alcançar esses objetivos, consequentemente necessitamos de informações da empresa que estamos modelando. Para entender a empresa, podemos utilizar o diagrama entidade-relacionamento e informações adicionais fornecidas pela empresa.

Observação

O diagrama de entidade-relacionamento (DER) é uma ferramenta muito utilizada para modelagem conceitual do banco de dados. Considerado de fácil compreensão, representa graficamente três conceitos:

Entidades: conjuntos de fatos (concretos ou abstratos) do mundo real. Por exemplo: clientes, pedidos, departamentos etc.

Relacionamentos: apresentam as associações entre as entidades.

Atributos: propriedades de uma entidade ou de um relacionamento. Por exemplo: nome de um cliente.

2.2.1 Características de um bom projeto relacional

Para o sucesso na implantação de um banco de dados em uma empresa, é essencial um bom projeto relacional. Quando o projeto é para uma pequena empresa, muitas vezes o responsável ignora algumas etapas importantes e começa a criação do banco de dados físico, criando tabelas, colunas e índices. Quando o projeto de banco de dados é para uma empresa grande, deve-se respeitar todas as etapas do projeto para garantir ao usuário todos os requisitos de informações necessários, com qualidade na disponibilidade desses dados, desempenho e confiabilidade.

Consideramos três fases em um projeto de banco de dados:

- **Modelo conceitual**: usado para representar as regras e os conceitos do negócio e como são associados. Seu maior objetivo é fornecer uma visão geral do negócio, envolvendo os desenvolvedores e os usuários que não precisam ter conhecimentos técnicos. Nessa primeira fase, não existe dependência de tecnologia para a implementação de banco de dados, apenas os conceitos de entidades, relacionamentos e atributos são necessários para a modelagem conceitual do negócio. Nessa fase, geramos dois produtos:

  —diagrama de entidade-relacionamento;

  —lista de regras de restrição de integridade.

- **Modelo lógico**: derivado do modelo conceitual, representa as características das estruturas de dados que serão desenvolvidas levando em conta os limites colocados pelo modelo de dados utilizado para o banco de dados (banco de dados hierárquico, banco de dados de rede, banco de dados relacionais etc.). Nessa fase, geramos o projeto lógico observando algumas características:

  – entidades associativas, e não relacionamento n:m;

  – chaves primárias;

  – chaves estrangeiras;

  – normalização;

  – consideração da nomenclatura da empresa;

  – dicionário de dados (entidades e atributos).

- **Modelo físico**: é a implementação do modelo lógico utilizando algum banco de dados e considerando os requisitos não funcionais de segurança, desempenho e disponibilidade que foram levantados pelo analista de requisitos. Nessa fase criamos o modelo físico a partir do modelo lógico, utilizando a linguagem SQL (structured query language) para definição da estrutura, manipulação e controle dos dados.

2.2.2 Domínios atômicos e primeira forma normal

O principal objetivo da normalização de tabelas é resolver problemas de atualização de banco de dados, minimizando redundâncias. Executamos um processo de desconstrução, removendo tabelas ruins e separando seus atributos em esquemas relacionais mais simples que satisfaçam as propriedades exigidas.

Codd (1970) sugeriu inicialmente o processo de normalização executando diversas verificações para confirmar se está na primeira, segunda e terceira forma normal. Cada etapa ou teste corresponde a um determinado padrão regular, representando uma melhoria progressiva na estrutura das tabelas. Desse modo, uma tabela na terceira forma normal é considerada mais normalizada (digamos, mais enxuta) do que uma apenas na segunda forma normal.

O processo de normalização tem como objetivo organizar e é possível acontecer durante a criação do modelo conceitual, durante a criação do modelo lógico para o relacional, ou após a criação do modelo lógico.

As principais vantagens de uma base de dados normalizada são:

- geração de aplicações mais estáveis;

- crescimento do número de tabelas utilizadas;
- diminuição dos tamanhos médios das tabelas.

Os principais benefícios da normalização são:

- **Estabilidade do modelo lógico**: a capacidade de um modelo permanecer inalterado diante de mudanças que podem ser percebidas ou introduzidas no ambiente modelado.
- **Flexibilidade**: a capacidade de se adaptar a diferentes necessidades, como expandir, reduzir etc.
- **Integridade**: refere-se à qualidade do dado. Um dado identificado em vários locais de modo diferente, com valores criados de modos diferentes, pode ser indício de que não há integridade entre eles.
- **Economia**: no espaço de armazenamento relativo ao custo de processamento de dados (processamento de grandes volumes de dados), custos causados por atrasos na entrega das informações desejadas.
- **Fidelidade ao ambiente observado**: ajuda a identificar alguns elementos que foram considerados durante o processo de modelagem.

Primeira forma normal – 1FN

Uma relação está na 1FN quando todos os domínios de atributos possuem apenas valores atômicos (simples e indivisíveis) e os valores para cada atributo na tupla são um único valor, ou seja, A 1NF não permite um conjunto ou série de valores em atributo de uma tupla, o que quer dizer que relacionamentos dentro de relacionamentos não são permitidos. Portanto, todos os atributos compostos e multivalorados devem ser divididos em atributos atômicos.

Procedimentos para 1FN

1. remova o(s) atributo(s) que viola(m) a primeira forma normal;
2. coloque-o em uma relação em separado, juntamente com a chave primária da relação original;
3. a chave primária da nova relação é composta da chave primária da relação original mais uma chave candidata dos atributos da nova relação.

Observação

Podemos utilizar quantos atributos forem necessários para formar uma nova chave na relação, mesmo que todos os atributos em uma linha possam ser concatenados. Na prática, se houver muitas colunas formando uma chave pode haver problemas, portanto, nesse caso, uma coluna contadora artificial pode ser criada.

A tabela 14 apresenta um exemplo da estrutura original da tabela clientes. Podemos observar que a tabela não está na 1FN, pois dois atributos apresentam problema: o endereço e o telefone.

Tabela 14 – Tabela de clientes estruturada originalmente

Id_cliente	Nome_ cliente	Endereço	Telefone
1	Monteiro Lobato	Praça Picapau Amarelo, 10 Morumbi 12345-678	(11) 1234-5678 (11) 2345-6789
2	José de Alencar	Rua Iracema, 189 Liberdade 23456-789	(11) 3456-7890 (11) 4567-8901
3	Cecília Meireles	Av Espectros, 588 Moema 34567-890	(11) 5678-9012 (11) 6789-0123

Para normalizar a tabela clientes, primeiramente vamos tratar o atributo endereço, que é composto de vários atributos: rua, bairro e CEP. Para tratar esse atributo composto, separamos cada atributo. O resultado está na tabela 15.

Tabela 15 – Resolução do atributo composto (endereço)

Id_cliente	Nome_ cliente	Rua	Bairro	CEP	Telefone
1	Monteiro Lobato	Praça Picapau Amarelo, 10	Morumbi	12345-678	(11) 1234-5678 (11) 2345-6789
2	José de Alencar	Rua Iracema, 189	Liberdade	23456-789	(11) 3456-7890 (11) 4567-8901
3	Cecília Meireles	Av Espectros, 588	Moema	34567-890	(11) 5678-9012 (11) 6789-0123

Observando o atributo telefone, identificamos que existem muitos valores para esse atributo por tupla. Para tratar atributos multivalorados, podemos separar o atributo em uma outra tabela e copiar a chave primária. O resultado está na tabela 16.

Tabela 16 – Resolução do atributo multivalorado (telefone)

Tabela: cliente

Id_cliente	Nome_ cliente	Rua	Bairro	CEP	Telefone
1	Monteiro Lobato	Praça Picapau Amarelo, 10	Morumbi	12345-678	(11) 1234-5678
2	José de Alencar	Rua Iracema, 189	Liberdade	23456-789	(11) 2345-6789
3	Cecília Meireles	Av Espectros, 588	Moema	34567-890	(11) 3456-7890

Tabela: telefone

Id_cliente	Telefone
1	(11) 1234-5678
1	(11) 2345-6789
2	(11) 3456-7890
2	(11) 4567-8901
3	(11) 5678-9012
3	(11) 6789-0123

2.2.3 Decomposição usando dependências funcionais

Existe uma metodologia formal baseada nos conceitos de chaves e dependências funcionais que nos auxilia na análise de um esquema relacional, caso precise ser decomposto.

A redundância é a causa de diversos problemas com esquemas relacionais, por exemplo, armazenamento redundante, anomalias de inserção, de exclusão e de atualização.

Segunda forma normal – 2FN

Uma relação está na 2FN se estiver na primeira forma normal e todos os atributos que não participam da chave primária dependem dela. Portanto, precisamos garantir que todos os atributos sejam dependentes da chave primária e remover quaisquer atributos de grupo independentes da relação, criando uma relação que contém esse atributo como não chave. Dessa maneira, a segunda forma normal evita inconsistências devido a duplicidade.

Procedimentos para 2FN

1. localize os atributos que não são funcionalmente dependentes de toda a chave primária;

2. exclua da tabela os atributos identificados e crie uma tabela com esses atributos.

A tabela 17 apresenta a tabela de pedidos estruturada originalmente. Essa tabela não está na 2FN pois possui atributos que não dependem da chave primária. Os atributos referentes aos produtos não dependem da chave primária da tabela pedidos.

Tabela 17 – Tabela de pedidos estruturada originalmente

N_pedido	Codigo_produto	Nome_produto	Qtd	Vlr_unitário	Subtotal
1	001	Televisão	3	2.000,00	6.000,00
2	002	Computador	2	5.000,00	10.000,00
3	003	Bicicleta	2	1.000,00	2.000,00
4	004	Fogão	1	3.000,00	3.000,00

Para normalizar a tabela pedidos após identificados os atributos não dependentes da chave primária, removemos da entidade os atributos e criamos a tabela. O resultado está na tabela 18.

Tabela 18 – Resolução do atributo não dependente da chave primária (nome_produto)

Tabela: pedidos

N_pedido	Codigo_produto	Qtd	Vlr_unitário	Subtotal
1	001	3	2.000,00	6.000,00
2	002	2	5.000,00	10.000,00
3	003	2	1.000,00	2.000,00
4	004	1	3.000,00	3.000,00

Tabela: produtos

Codigo_produto	Nome_produto
001	Televisão
002	Computador
003	Bicicleta
004	Fogão

Terceira forma normal – 3FN

Uma relação está na 3FN se estiver na 2FN e atributos não chave que se refiram a outros atributos não chave. Precisamos verificar se há um atributo que não depende diretamente da chave e removê-lo, criando uma relação que contenha esse grupo de atributos usando a chave para determinar os atributos dos quais o grupo depende diretamente

Procedimentos para 3FN

1. localize todos os atributos que são funcionalmente dependentes de outros atributos não chave e remova-os;

2. na chave primária da nova tabela os atributos removidos são funcionalmente dependentes.

Podemos observar a tabela pedidos da tabela 18. Essa tabela não está na 3FN pois existe atributo que está relacionando a outro atributo não chave. O atributo subtotal é dependente do cálculo dos atributos Qtd, que multiplica o valor do atributo Vlr_unitário. Podemos remover o atributo subtotal e criar a tabela conforme a tabela 19.

Tabela 19 – Resolução do atributo dependente de atributo não chave (subtotal)

Tabela: pedidos

N_pedido	Codigo_produto	Qtd	Vlr_unitário
1	001	3	2.000,00
2	002	2	5.000,00
3	003	2	1.000,00
4	004	1	3.000,00

3 LINGUAGEM SQL

A SQL (structured query language), ou linguagem de consulta estruturada, é um tipo de linguagem de consulta muito utilizada para executar operações em bancos de dados relacionais.

3.1 Tipos de linguagem SQL

A linguagem SQL é dividia em subconjuntos, que se baseiam nas operações efetuadas sobre um banco de dados, tais como:

- **data definition language (DDL)**: linguagem de definição de dados;
- **data manipulation language (DML)**: linguagem de manipulação de dados;
- **data query language (DQL)**: linguagem de consulta de dados;
- **data control language (DCL)**: linguagem de controle de dados;
- **data transaction language (DTL)**: linguagem de transação de dados.

3.1.1 Data definition language (DDL)

O subconjunto DDL nos permite criar, atualizar e remover tabelas e elementos associados. São poucos os comandos básicos da DDL:

- **CREATE**: cria um objeto (por exemplo uma tabela) dentro do banco de dados;
- **DROP**: remove um objeto do banco de dados;

- **ALTER**: altera um objeto do banco de dados (por exemplo, adicionando uma coluna a uma tabela existente).

3.1.2 Data manipulation language (DML)

O subconjunto DML é utilizado para realizar inclusões, consultas, alterações e exclusões de dados existentes em registros. Podemos executar essas ações em um ou em vários registros de diferentes tabelas ao mesmo tempo. Os comandos básicos que executam as funções são:

- **INSERT**: insere um registro (tupla) em uma tabela do banco de dados;
- **UPDATE**: modifica os valores de uma ou várias linhas em uma tabela do banco de dados;
- **DELETE**: remove uma ou várias linhas em uma tabela do banco de dados.

3.1.3 Data query language (DQL)

O subconjunto DQL permite ao usuário executar uma consulta (query) nas tabelas do banco de dados. O DQL é o mais utilizado, pois nos permite executar consultas das mais simples às mais elaboradas. Tem apenas um comando:

- **SELECT**: consulta dados específicos de uma ou várias tabelas.

3.1.4 Data control language (DCL)

O subconjunto DCL gerencia aspectos de autorização de dados e liberação de usuários para acessos de visualização ou manipulação de dados dentro do banco de dados. Os comandos utilizados são:

- **GRANT**: autoriza ao usuário executar determinada operação no banco;
- **REVOKE**: remove a capacidade de um usuário de executar operações no banco.

3.1.5 Data transaction language (DTL)

O subconjunto DTL controla a execução de transações em um banco de dados. As transações são unidades de execução de programa que atualizam dados. Os comandos utilizados são:

- **BEGIN**: marca o começo de uma transação de banco de dados que pode ser completada ou não (BEGIN TRANSACTION);
- **COMMIT**: finaliza uma transação que esteja sendo executada dentro de um sistema de gerenciamento de banco de dados;

- **ROLLBACK**: faz com que as mudanças atuais nos dados existentes sejam abortadas, restaurando o banco até o estado do último COMMIT ou ROLLBACK.

A figura 9 apresenta um resumo dos subconjuntos da SQL e seus principais comandos. Ela pode ajudar a entender melhor a organização da SQL e facilitar a fixação dos comandos. A seguir aprofundaremos os estudos com exemplos práticos para cada subconjunto.

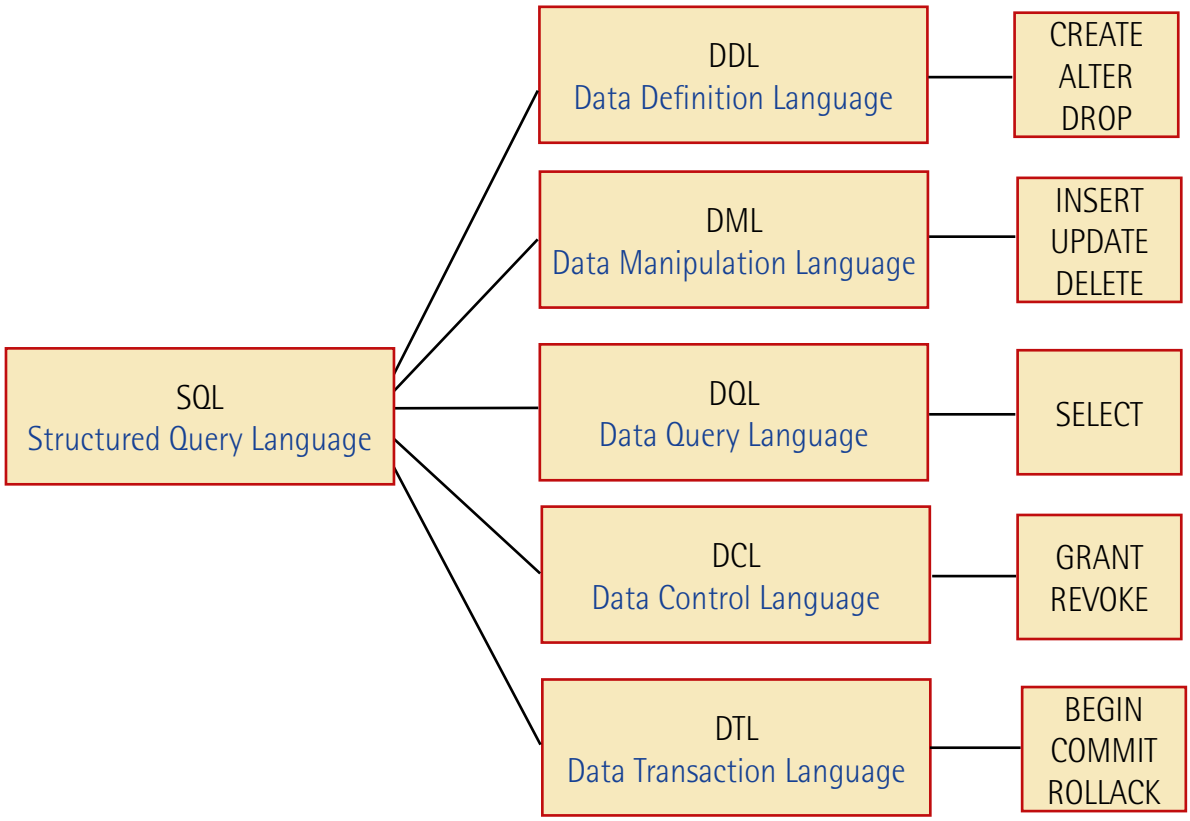

Figura 9 – Resumo dos subconjuntos da SQL e seus principais comandos

3.2 Estruturas básicas do SQL

Embora chamemos a SQL de linguagem de consulta, ela pode fazer muito mais do que simplesmente consultar um banco de dados. Ela pode definir uma estrutura de dados, modificar dados de banco de dados e definir restrições de segurança.

Para o desenvolvimento dos exemplos a seguir, utilizamos a ferramenta SQL Server Management Studio (SSMS), que fornece acesso ao projeto de banco de dados, facilitando seu uso com recursos visuais. O SSMS aceita gerenciamento centralizado, permite que um administrador de banco de dados crie, modifique e copie esquemas de banco de dados do SQL Server e objetos como views, tabelas etc.

3.2.1 Definição de dados

As tabelas em um banco de dados precisam ser criadas para o sistema, utilizando uma linguagem de definição (DDL) que possibilite não apenas a especificação como também informações sobre cada tabela, como:

- o esquema para cada relação;
- o domínio dos valores de cada atributo;
- as restrições de integridade;
- o conjunto dos índices para cada tabela;
- as informações de segurança e autorização de acesso para cada tabela;
- a estrutura de armazenamento físico de cada relação no disco.

Tipos de domínios básicos

O padrão SQL reconhece vários tipos de domínios internos:

Quadro 2 – Tipos de domínio

Tipo	Descrição
char(n)	Uma string composta de caracteres de tamanho fixo (n)
varchar(n)	Uma string composta de caracteres de tamanho variado com no máximo (n)
int ou integer	Um número inteiro
smallint	Um número inteiro pequeno
numeric(p,d)	Um número precisão, onde p é a quantidade de dígitos mais um sinal, e d dos p dígitos está à direita da vírgula decimal
real, double precision	Número de ponto flutuante com precisão dupla que depende da máquina utilizada
float(n)	Um número de ponto flutuante com precisão de no mínimo n dígitos

O dicionário de dados

O dicionário de dados contém a definição de todas as tabelas, todos os campos, de cada objeto, enfim, a definição do próprio banco de dados. Definindo melhor, um dicionário de dados armazena informações sobre todos os objetos de um banco de dados.

Existem vários objetos criados pela DDL, mas a mais importante das construções é a criação de tabelas. Também é possível criar outros objetos como visualizações, índices, usuários, conjuntos, procedimentos armazenados (stored procedures) e gatilhos (triggers).

Criando tabelas

O código a seguir demonstra um exemplo de uso da SQL para criação das tabelas de Funcionario e Departamento. Utilizamos o comando USE para selecionar a base de dados utilizada, em seguida utilizamos o comando GO para executar o comando anterior. O atributo id_Funcionario com domínio do tipo inteiro não pode receber nulo (NOT NULL).

```
USE BD_LOJA
GO

CREATE TABLE Funcionario (
      id_Funcionario integer NOT NULL,
      Nome CHAR(40),
      Logradouro CHAR (40),
      Bairro CHAR(15),
      Salario numeric (11,2)
)
GO

CREATE TABLE Departamento (
      id_Departamento integer NOT NULL,
      Nome VARCHAR(14),
      Localizacao VARCHAR(13))
GO
```

Alterando tabelas

A seguir exemplos de alterações de tabela:

- Adicionando a coluna idade na tabela funcionário:

```
ALTER TABLE Funcionario
ADD idade int
GO
```

- Modificando o tamanho da coluna Localizacao da tabela Departamento de VARCHAR(13) para VARCHAR(20):

```
ALTER TABLE Departamento
ALTER COLUMN Localizacao VARCHAR(20)
GO
```

- Excluindo a tabela Departamento do banco de dados:

```
DROP TABLE Departamento
GO
```

3.2.2 Estrutura básica de consultas SQL

A estrutura básica de consulta de dados em SQL de uma tabela consiste em três cláusulas:

SELECT lista_atributos

FROM tabela

WHERE condição

Comando SELECT

O retorno de uma consulta SQL é uma relação que lista todos os atributos selecionados no resultado de uma consulta. O comando SELECT facilita a projeção das informações, pois possibilita a eliminação de dados duplicados, podemos retornar valores calculados utilizando operadores aritméticos (+, -, *, /).

- Para listar todos os atributos de todos os funcionários utilizaríamos o comando:

```
SELECT * FROM Funcionario
GO
```

- Para listar os nomes e os salários de todos os funcionários utilizaríamos o comando:

```
SELECT Nome, Salario FROM Funcionario
GO
```

- Para forçar a eliminação de nomes duplicados na tabela funcionário utilizaríamos o comando:

```
SELECT distinct NOME FROM Funcionario
GO
```

Cláusula WHERE

Para selecionar valores específicos de algum atributo, utilizamos a cláusula WHERE junto com o SELECT. Usamos os conectores and, or e not juntamente com os operadores de comparação (=, <>, >, >=, <, <=) para comparar strings, expressões aritméticas e tipos específicos como data.

- Para listar os nomes dos funcionários, considerando um filtro no atributo nome, podemos escrever os comandos a seguir:

```
-- Nome do Funcionario igual a Tadeu
SELECT Nome FROM Funcionario WHERE Nome = 'Tadeu'

-- Nome do Funcionario diferente a Tadeu
SELECT Nome FROM Funcionario WHERE Nome <> 'Tadeu'

-- Nome do Funcionario maior a Tadeu
SELECT Nome FROM Funcionario WHERE Nome > 'Tadeu'

-- Nome do Funcionario maior igual a Tadeu
SELECT Nome FROM Funcionario WHERE Nome >= 'Tadeu'

-- Nome do Funcionario menor a Tadeu
SELECT Nome FROM Funcionario WHERE Nome < 'Tadeu'

-- Nome do Funcionario menor igual a Tadeu
SELECT Nome FROM Funcionario WHERE Nome <= 'Tadeu'
```

- Utilizando operador de comparação between para facilitar as cláusulas WHERE que especifiquem um intervalo de valor (com mínimo e máximo).

```
-- Funcionários com nome entre João e Tadeu (inclusivo)
SELECT Nome FROM Funcionario WHERE Nome between 'João' and 'Tadeu'

-- Funcionários com nome entre João e Tadeu (não inclusivo)
SELECT Nome FROM Funcionario WHERE Nome not between 'João' and 'Tadeu'
```

- Para listar registros considerando valores de pertinência dos elementos do conjunto ('João','Tadeu'):

```
-- Funcionários com os nomes iguais João e Tadeu
SELECT Nome FROM Funcionario WHERE Nome in ('João','Tadeu')

-- Funcionários com os nomes diferentes João e Tadeu
SELECT Nome FROM Funcionario WHERE Nome not in ('João','Tadeu')
```

- Para buscar padrões considerando uma cadeia de caracteres em um atributo string:

```
-- Funcionários com nomes iniciados pela letra 'J'
SELECT Nome FROM Funcionario WHERE Nome like 'J%'
```

- Utilizando o conectivo or para listar os funcionários com nome igual a Tadeu ou Vania:

```
-- Funcionários com Nome igual a Tadeu ou Vania
SELECT Nome FROM Funcionario WHERE Nome = 'Tadeu' or Nome = 'Vania'
```

- Utilizando o conectivo and para listar os funcionários com nomes iniciados pela letra 'J' e salário > 1000:

```
-- Funcionários com nomes iniciados pela letra 'J' e Salário maior que 1000
SELECT Nome FROM Funcionario WHERE Nome like 'J%' and Salario > 1000
```

Cláusula FROM

A cláusula FROM define um produto cartesiano das relações na cláusula.

3.2.3 Operações de conjunto

As funções dos conjuntos podem nos lembrar da noção de conjuntos do campo da matemática, em que poderíamos formar conjuntos de números combinando, separando e observando o que cada um tem em comum. No SQL, temos operações que levam esse conceito em consideração e o utilizam para agrupar dados que formam uma espécie de conjuntos de dados, ou seja, funções que agregam conjuntos de dados usando os mesmos princípios operacionais estudados na disciplina de matemática. Essas operações são: UNION, INTERSECT e EXCEPT.

Os operadores de conjunto agrupam os resultados da consulta de duas seleções em um único resultado. As consultas contêm operadores bem estabelecidos, chamados consultas compostas. Com esses operadores, é possível combinar várias consultas e todos os operadores estabelecidos têm a mesma precedência.

Se as consultas retornarem valores do tipo de texto de comprimento igual, então os valores devolvidos têm tipo de texto equivalentes. Se as consultas retornarem valores do tipo de texto e com diferentes comprimentos, o valor de retorno é texto variado e é considerado o comprimento do maior valor.

UNION

A operação UNION elimina os valores duplicados diferente da operação SELECT, isto é, combina os resultados de duas linhas, eliminando as linhas duplicadas, ou seja, todas as linhas que aparecerão no resultado serão distintas.

- Para encontrar o nome de todos os funcionários e o nome de todos os clientes eliminando os nomes duplicados, utilizamos o comando:

```
(select Nome
from Funcionario)
union
(select Nome
from Cliente)
```

- Para encontrar o nome de todos os funcionários e o nome de todos os clientes mantendo os nomes duplicados utilizamos o UNION ALL, utilizamos o comando:

```
(select Nome
from Funcionario)
union all
(select Nome
from Cliente)
```

INTERSECT

A operação INTERSECT encontra os registros que pertencem aos dois SELECTs, isto é, combina os resultados e retorna somente as linhas resultantes por todas as consultas. No entanto ele só retorna as linhas de conjuntos de dados caso um registro exista em uma consulta e nas demais também.

- Para encontrar o nome de todos os funcionários que também possuem nome na tabela clientes eliminando os nomes duplicados, utilizamos o comando:

```
(select Nome
from Funcionario)
intersect
(select Nome
from Cliente)
```

- Para encontrar o nome de todos os funcionários cujo nome também está na tabela clientes sem eliminar os nomes duplicados INTERSECT ALL, utilizamos o comando:

```
(select Nome
from Funcionario)
intersect all
(select Nome
from Cliente)
```

EXCEPT

A operação EXCEPT encontra os registros que pertencem ao primeiro SELECT e não aparece no segundo.

- Para encontrar o nome de todos os funcionários cujo nome não está na tabela clientes eliminando os nomes duplicados, utilizamos o comando:

```
(select Nome
from Funcionario)
except
(select Nome
from Cliente)
```

- Para encontrar o nome de todos os funcionários cujo nome não está na tabela clientes sem eliminar os nomes duplicados EXCEPT ALL, utilizamos o comando:

```
(select Nome
from Funcionario)
except all
(select Nome
from Cliente)
```

3.2.4 Funções agregadas

As funções de agregação são funções da SQL que executam uma operação aritmética nos valores de uma coluna específica ou em todos os registros de uma tabela do banco de dados, por exemplo: o cálculo da soma ou da média de um atributo ou apenas contar o número de registros que atendem determinados critérios. Temos as funções COUNT, MAX / MIN, SUM e AVG.

- COUNT (contador de ocorrências [de um atributo]):

```
-- Apresenta a quantidade de funcionários
SELECT count(id_Funcionario) as QtTotal
FROM Funcionario
```

- MAX / MIN (valores máximos / mínimo de um atributo):

```
-- Apresenta o maior Salário
SELECT max(Salario) as Maior_Salario
FROM Funcionario

-- Apresenta o menor Salário
SELECT min(Salario) as Menor_Salario
FROM Funcionario
```

- SUM (somador de valores de um atributo):

```
-- Apresenta a soma dos Salários
SELECT sum(Salario) as Total
FROM Funcionario
```

- AVG (média de valores de um atributo):

```
-- Apresenta a média de Salário dos Funcionários
SELECT avg(Salario) as Media
FROM Funcionario
```

Observação

A cláusula as pode aparecer tanto na cláusula SELECT para renomear atributos, quanto na cláusula FROM para renomear relações.

3.2.5 Valores nulos

A SQL permite a utilização de valores nulos para registrar a ausência de dado sobre o valor do atributo. Utilizamos a palavra null para localizar esses registros.

- Para localizar os funcionários que não possuem setor, utilizamos o comando:

```
SELECT Nome
FROM Funcionario
WHERE Setor is null
```

- Para localizar os funcionários que possuem setor, utilizamos o comando:

```
SELECT Nome
FROM Funcionario
WHERE Setor is not null
```

3.2.6 Views

Por questões de segurança, não é interessante que todos os usuários vejam as relações reais armazenadas no banco de dados (modelo lógico), por isso pode ser necessário ocultar alguns dados para os usuários (usuários do banco de dados). Por exemplo, um usuário pode precisar ver os dados de uma fatura do cartão de crédito de um cliente, mas não deve visualizar seu código de segurança, nem senha. Para isso, podemos criar uma relação personalizada que corresponda às necessidades do usuário.

Definimos uma VIEW usando o comando CREATE VIEW, atribuímos um nome e a consulta que a VIEW apresentará.

Criar uma view

Criamos uma view com o nome VW_Analise que apresente apenas os campos CPF, nome, vencimento e valor da tabela fatura.

```
CREATE VIEW VW_Analise AS
(SELECT CPF, Nome, Vencimento, Valor
FROM Fatura)
GO
```

Utilizando uma view

- Utilizamos a VIEW no SELECT como uma tabela do banco de dados:

```
SELECT Valor
FROM VW_Analise
WHERE CPF = '11111111111'
GO
```

Removendo uma view

- Para eliminar a VIEW do banco de dados, utilizamos o comando:

```
DROP VIEW VW_Analise
GO
```

3.2.7 Modificação do banco de dados

Além da operação de extração de dados (SELECT), outras operações são muito importantes, como EXCLUSÃO, INSERÇÃO e ATUALIZAÇÃO.

Exclusão

Podemos excluir apenas tuplas inteiras, e não apenas atributos. A SQL expressa a exclusão da seguinte forma:

DELETE FROM tabela

WHERE [condição]

- Podemos utilizar sem a cláusula WHERE, o que irá excluir todas as tuplas da relação funcionário:

```
DELETE FROM Funcionario
```

- Podemos implementar a cláusula WHERE para excluir as tuplas que atendem a condição desejada. No caso, excluir a tupla onde id_Funcionario seja igual a 3.

```
DELETE FROM Funcionario
WHERE id_Funcionario = 3
```

Inserção

Para inserir dados em uma relação, especificamos quais dados serão inseridos ou escrevemos uma consulta cujo resultado sejam as tuplas inseridas na relação. Para inserir registo na tabela funcionários temos:

```
INSERT INTO Funcionario
VALUES (1,'Tadeu', 'Av. Paulista', 'Moema', 1500, 32);
GO
```

Atualização

Podemos precisar modificar o valor de um atributo em uma determinada tupla ou em um grupo específico respeitando uma condição.

- Para modificar o bairro de um funcionário onde o id_funcionario seja igual a 1, podemos escrever:

```
UPDATE Funcionario
SET Bairro = 'Ibira'
WHERE id_Funcionario = 3;
GO
```

- Para aumentar em 100 o salário dos funcionários que ganham menos que 1000 podemos escrever:

```
UPDATE Funcionario
SET Salario = Salario + 100
WHERE Salario <= 1000;
GO
```

4 SQL AVANÇADA

A linguagem SQL cresceu muito desde 1970, evoluindo de recursos básicos para recursos avançados para satisfazer diferentes usuários.

Lembrete

Conforme apontado, embora seja uma linguagem de consulta, a SQL pode fazer mais do que simplesmente consultar um banco de dados, pois pode definir uma estrutura de dados, modificar dados de banco de dados e definir restrições de segurança.

4.1 Tipos de dados

Além dos tipos básicos como integer, real e character, o padrão SQL aceita outros tipos de dados:

Quadro 3 – Tipos de dados

Tipo	Descrição
date	Uma data contendo ano (quatro dígitos), mês e dia Exemplo: '2000-02-25'
time	A hora do dia, em horas, minutos e segundos Exemplo: '10:35:53'
timestamp	Uma combinação de date e time Exemplo: '2000-04-25 11:29:01.35'

4.2 Restrições de integridade

4.2.1 Definindo a integridade referencial

Para assegurar a integridade referencial entre as tabelas do banco de dados, precisamos criar dois objetos e associá-los à tabela: uma chave primária e uma chave estrangeira (referência). Ambos são restrições que garantem a qualidade dos dados no banco de dados.

A chave primária (primary key) assegura que não haverá duplicidade de linhas com valores idênticos para as colunas da chave primária.

A chave estrangeira (foreign key) assegura que uma referência a um dado em outra tabela será sempre válida. A chave estrangeira sempre aponta para uma chave primária de outra tabela com um valor existente nesta outra tabela.

Criar chave primária para uma tabela (primary key)

A tabela filial possui uma chave primária composta de dois campos id_empresa e id_filial. Essa chave assegura que não teremos duplicidades de empresa e filial no cadastro. O código de criação ficaria assim:

```
CREATE TABLE Filial(
      id_filial integer NOT NULL,
      id_empresa integer NOT NULL,
      cod_tipo varchar(20),
      dt_cadastro date,
      CONSTRAINT PK_FILIAL PRIMARY KEY(id_empresa,id_filial))
GO
```

Criar chave estrangeira para uma tabela (foreign key)

A chave estrangeira na tabela filial apresenta a referência das tabelas empresa e filial, isto é, empresa. id_empresa = Filial.id_empresa. O código de criação ficaria assim:

```
CREATE TABLE Empresa(
      id_empresa integer NOT NULL,
      nome varchar(80) NOT NULL,
CONSTRAINT PK_EMPRESA PRIMARY KEY(id_empresa))
GO

CREATE TABLE Filial(
      id_filial int NOT NULL,
      id_empresa int NOT NULL,
      cod_tipo varchar(20),
      dt_cadastro date,
CONSTRAINT PK_FILIAL PRIMARY KEY (id_empresa, id_filial),
CONSTRAINT FK_EMPRESA FOREIGN KEY(id_empresa) REFERENCES Empresa(id_empresa))
GO
```

Restrição CHECK

A restrição CHECK é usada para limitar o intervalo de valores que podem ser colocados em uma coluna. Se definirmos uma restrição CHECK em uma coluna, ela permitirá apenas determinados valores para aquela coluna. Se especificarmos uma restrição CHECK em uma tabela, ela pode limitar os valores de algumas colunas com base nos valores de algumas outras colunas.

A restrição CHECK garante a qualidade dos dados adicionados/atualizados na tabela. Recomenda-se sempre usar a restrição CHECK, sobretudo para colunas que representam dados com um determinado conjunto de valores, por exemplo:

- Criar uma restrição CHECK no comando de criação de uma tabela:

```
CREATE TABLE Funcionario (
      id_Funcionario int NOT NULL,
      Nome CHAR(40),
      Logradouro CHAR (40),
      Bairro CHAR(15),
      Salario numeric (11,2),
      idade int CHECK (idade>=18)
)
GO
```

- Criar uma restrição CHECK para mais de uma coluna:

```
CREATE TABLE Funcionario (
      id_Funcionario int NOT NULL,
      Nome CHAR(40),
      Logradouro CHAR (40),
      Bairro CHAR(15),
      Salario numeric (11,2),
      idade int,
      CONSTRAINT CHK_Funcionario CHECK (idade>=18 AND salario > 10)
)
GO
```

- Adicionar uma restrição CHECK para uma tabela que já existe no banco de dados:

```
ALTER TABLE Funcionario
ADD CHECK (idade>=18);
```

- Adicionar uma restrição CHECK para mais de uma coluna em uma tabela que já existe no banco de dados:

```
ALTER TABLE Funcionario
ADD CONSTRAINT CHK_Funcionario CHECK (idade>=18 AND salario > 10);
```

- Remover uma restrição CHECK de uma tabela:

```
ALTER TABLE Funcionario
DROP CONSTRAINT CHK_Funcionario;
```

Restrição UNIQUE

Uma restrição UNIQUE garante que todos os valores em uma coluna sejam únicos. As restrições UNIQUE e PRIMARY KEY garantem a exclusividade de uma coluna ou conjunto de colunas.

Uma restrição PRIMARY KEY tem automaticamente uma restrição UNIQUE. No entanto, pode haver várias restrições UNIQUE por tabela, mas apenas uma restrição PRIMARY KEY por tabela.

- Criar uma restrição UNIQUE no comando de criação de uma tabela:

```
CREATE TABLE Funcionario (
      id_Funcionario int NOT NULL,
      CPF CHAR(11) UNIQUE,
      Nome CHAR(40),
      Logradouro CHAR (40),
      Bairro CHAR(15),
      Salario numeric (11,2),
      idade int
)
GO
```

- Criar uma restrição UNIQUE para mais de uma coluna:

```
CREATE TABLE Funcionario (
      id_Funcionario int NOT NULL,
      CPF CHAR(11),
      Nome CHAR(40),
      Logradouro CHAR (40),
      Bairro CHAR(15),
      Salario numeric (11,2),
      idade int
      CONSTRAINT UC_Funcionario UNIQUE (id_funcionario, cpf)
)
GO
```

- Adicionar uma restrição UNIQUE a uma tabela já existente no banco de dados:

```
ALTER TABLE Funcionario
ADD UNIQUE (cpf);
```

- Adicionar uma restrição UNIQUE para mais de uma coluna em uma tabela já existente no banco de dados:

```
ALTER TABLE Funcionario
CONSTRAINT UC_Funcionario UNIQUE (id_funcionario, cpf);
```

- Remover uma restrição UNIQUE de uma tabela:

```
ALTER TABLE Funcionario
DROP CONSTRAINT UC_Funcionario;
```

4.3 Autorização

Por questões de segurança, podemos conceder a um usuário várias formas de autorização sobre partes do banco de dados. O padrão SQL controla os privilégios SELECT, INSERT, UPDATE e DELETE. A forma básica dessa instrução é:

GRANT <lista de privilégios>

ON <nome da relação ou nome da view>

TO <lista de usuários ou papéis>

- Autorização para ler dados:

```
-- Concede ao usuário Pedro privilégio SELECT na tabela Funcionário
GRANT SELECT ON Funcionario TO Pedro
```

- Autorização para inserir novos dados:

```
-- Concede ao usuário Pedro privilégio INSERT na tabela Funcionário
GRANT INSERT ON Funcionario TO Pedro
```

- Autorização para atualizar dados:

```
/* Concede ao usuário Pedro privilégio de atualização
 do atributo salário na tabela Funcionário */
GRANT UPDATE (salario) ON Funcionario TO Pedro
```

- Autorização para excluir dados:

```
-- Concede ao usuário Pedro privilégio DELETE na tabela Funcionário
GRANT DELETE ON Funcionario TO Pedro
```

Para anular uma autorização, usamos a instrução **REVOKE**. Ela possui a mesma forma básica da instrução GRANT:

REVOKE <lista de privilégios>

ON <nome da relação ou nome da view>

TO <lista de usuários ou papéis>

4.4 SQL embutida

A linguagem SQL possui uma linguagem de consulta declarativa muito poderosa. Escrever consultas em linguagem de programação é muito mais complexo do que escrever consultas em SQL.

A SQL embutida ocorre sempre que colocamos código SQL em uma linguagem de programação. A linguagem está embutida no programa e é responsável por gerar a SQL e enviá-la ao banco de dados para fazer o trabalho. As instruções são executadas pelo programa de computador, assim como todas as decisões necessárias para a consulta ao banco de dados. O banco de dados receberá da aplicação apenas uma SQL estática, na maioria dos casos, e, por sua vez, a aplicação ganha independência do banco de dados.

Após o programa ser compilado pelo compilador da linguagem, usamos a instrução EXEC SQL para identificar requisições de SQL embutidas para o pré-processamento. A sintaxe do comando é:

EXEC SQL <instruções SQL embutidas> **END-EXEC**

Observação

A sintaxe exata do comando para requisições SQL embutidas depende da linguagem em que a SQL está embutida.

4.5 SQL dinâmica

SQL dinâmica é um recurso muito usado por programadores, no qual é possível construir e submeter comandos SQL em tempo de execução. Dessa forma, o programador pode adequar as necessidades do seu negócio ou do seu usuário ao montar a SQL. Por exemplo: depois de executar o comando enviado ao banco de dados, o banco retorna as informações nas variáveis do programa, e o programador poderá manipular em tempo de execução.

4.6 Funções de construções procedurais

A partir de 1999, a linguagem SQL aceita funções e procedimentos que podem ser escritas na própria SQL ou em uma linguagem de programação externa. As funções são muito úteis com tipos de dado especializados, por exemplo: imagens (funções para verificar semelhança entre imagens) e objetos geométricos (funções para verificar se existe sobreposição de polígonos). Alguns sistemas de banco de dados aceitam funções com valor de tabela e que podem retornar uma outra tabela como resultado. Um grande conjunto de instrução também é aceito: loops, if-then-else e atribuições.

Podemos definir uma função que, dado o ID de um cliente, retorne a contagem do número de pedidos pertencentes a esse cliente.

```
CREATE FUNCTION dbo.conta_pedido (@id_cliente int)
      RETURNS integer
      AS
      BEGIN
            DECLARE @contagem_c integer;
                  SELECT @contagem_c = count(*)
                  FROM PEDIDO
                  WHERE Pedido.id_cliente = @id_cliente
            RETURN @contagem_c;
      END
GO
```

Podemos encontrar os nomes dos clientes e a quantidade de pedidos feitos por cada um.

```
SELECT Nome, dbo.conta_pedido(id_cliente) as 'Qtde_Pedido'
FROM CLIENTE
```

Em 2003, a linguagem SQL acrescentou funções que retornam uma relação como resultado. Por exemplo, podemos encontrar todos os pedidos de um determinado cliente:

```
CREATE FUNCTION pedidos_de (nome_cliente varchar(50))
      RETURN TABLE (
            num_pedido char(10),
            produto varchar(50),
            valor numeric(12,2))
      RETURN TABLE
            (SELECT num_pedido, produto, valor
            FROM PEDIDO
            WHERE exists(
                  SELECT *
                  FROM CLIENTE
```

Para utilizar:

```
SELECT * FROM TABLE (pedidos_de('Machado de Assis'))
```

A função conta_pedido poderia ser escrita como um procedimento:

```
CREATE PROCEDURE dbo.proc_conta_pedido (@id_cliente integer, @contagem_c integer OUTPUT)
      AS
      BEGIN
            SELECT @contagem_c = count(*)
            FROM PEDIDO
            WHERE Pedido.id_cliente = @id_cliente
      END
```

Para chamar um procedimento no SQL ou no SQL embutido, usamos a instrução EXECUTE:

```
DECLARE @contagem int;
EXECUTE dbo.proc_conta_pedido 2, @contagem OUTPUT;
PRINT  @contagem
```

A linguagem SQL permite mais de um procedimento/função com o mesmo nome (overloading) desde que o número de argumentos ou os tipos sejam diferentes.

Podemos utilizar diversas instruções SQL entre a instrução composta **begin ... end**. As variáveis locais podem ser declaradas dentro de uma instrução composta.

- Podemos utilizar loops como as instruções WHILE e REPEAT:

```
DECLARE n integer default 0;
WHILE n < 5 do
     SET n = n + 1;
END WHILE
REPEAT
     SET n = n - 1;
END REPEAT
```

- Podemos utilizar loops como a instrução FOR, que permite a interação sobre todos os resultados de uma consulta. Por exemplo, encontrar o valor de todos os pedidos de um cliente:

```
DECLARE n integer default 0;
     FOR r as
          SELECT valor FROM pedido
          WHERE nome_cliente = 'Machado de Assis'

          SET n = n + r.valor
     END FOR
```

- Instrução condicional também é aceita na SQL. Por exemplo, para encontrar a soma dos pedidos agrupados em três categorias:

```
IF r.valor < 1000
      THEN set cat_1 = cat_1 + r.valor
ELSE IF r.valor < 5000
      THEN set cat_2 = cat_2 + r.valor
ELSE set cat_3 = cat_3 + r.valor
END IF
```

4.7 Consultas recursivas

Considere um banco de dados com dados de funcionários e suponha que tenha a relação gerente(nome_funcionario, nome_gerente). Para identificar quais funcionários são supervisionados, direta ou indiretamente, por um determinado gerente, podemos observar a VIEW abaixo (SILBERSCHATZ; KORTH; SUDARSHAN, 2020, p. 101).

```
with recursive func (nome_funcionário, nome_gerente) as (
      select nome_funcionário, nome_gerente
      from gerente
   union
      select gerente.nome_funcionário, func.nome_gerente
      from gerente, func
      where gerente.nome_gerente = empl.nome_funcionário
   )
select *
from func
```

As VIEWS recursivas possibilitam escrever consultas que não podem ser escritas sem recursão ou repetição.

Resumo

Nesta unidade, você aprendeu que a arquitetura de um sistema de banco de dados é muito dependente do sistema básico do computador onde ele está sendo executado. Vimos que os primeiros bancos de dados foram implementados utilizando arquiteturas centralizadas. Com a evolução dos computadores pessoais, as máquinas tornaram-se mais rápidas e menos custosas, dessa forma o uso da arquitetura centralizada foi diminuindo e deu espaço para a arquitetura cliente-servidor.

Analisamos as diferenças entre os sistemas distribuídos e os sistemas paralelos. Nos sistemas paralelos, as máquinas devem estar próximas, já para os sistemas distribuídos a localização das máquinas é indiferente, podendo estar geograficamente distantes.

Na década de 1980, surgiu o sistema de gerência de banco de dados paralelos, com o objetivo de resolver o problema de gargalo de entrada e saída dos bancos de dados convencionais. Comparamos com o sistema de gerência de banco de dados distribuído, que se constitui de uma relação de pontos de conexão em que cada um pode participar da execução de transações que acessam dados em um ou mais nós.

Os bancos relacionais foram criados em 1970 por Edgar Frank Codd, que definiu o modelo relacional. Esse modelo continua dominante no mercado, sendo o mais utilizado no mundo. Estudamos os fundamentos e os conceitos dos bancos relacionais, como os elementos básicos da estrutura. Por exemplo: conjunto de tabelas (ou relação) e as linhas (ou tuplas). Aprendemos o básico de álgebra relacional para compreender as operações executadas sobre uma ou várias tabelas.

Para o sucesso na implantação de um banco de dados em uma empresa é essencial um bom projeto relacional. Consideramos três fases em um projeto de banco de dados que devem ser tratadas com muita responsabilidade: modelo conceitual, modelo lógico e modelo físico.

A normalização é uma metodologia para projetos de banco de dados relacionais que nos ajuda a organizar as tabelas, evitando problemas de redundância e anomalias de atualização que podem aparecer em uma tabela. Para resolver esse problema, precisamos fazer a decomposição de uma relação, utilizando as regras de normalização.

A linguagem SQL (structured query language), ou linguagem de consulta estruturada, é um tipo de linguagem de consulta muito utilizada para executar operações em bancos de dados relacionais. Possui cinco subconjuntos baseados nas suas operações:

- **Data definition language (DDL)**: linguagem de definição de dados;
- **Data manipulation language (DML)**: linguagem de manipulação de dados;
- **Data query language (DQL)**: linguagem de consulta de dados;
- **Data control language (DCL)**: linguagem de controle de dados;
- **Data transaction language (DTL)**: linguagem de transação de dados.

Embora chamemos o SQL de linguagem de consulta, ela oferece muito mais possibilidades do que simplesmente consultar um banco de dados, pois com ela podemos definir uma estrutura de dados, modificar dados de banco de dados e definir restrições de segurança.

Exercícios

Questão 1. Há dois grupos de operações fundamentais da álgebra relacional: o grupo de operações unárias e o grupo de operações binárias. As operações unárias operam em apenas uma relação, e seu grupo abrange as operações de seleção, de projeção e de renomeação. As operações binárias, por sua vez, operam em pares de relações, e seu grupo abrange as operações de união, de diferença de conjunto e de produto cartesiano. A respeito dessas operações fundamentais, avalie as afirmativas.

I – Utilizamos a operação de projeção quando precisamos apresentar tuplas que satisfaçam a determinado predicado.

II – Utilizamos a operação de renomeação quando precisamos selecionar colunas específicas de uma relação.

III – A operação de diferença de conjunto é utilizada quando queremos encontrar tuplas que estão em uma relação, mas não estão em outra.

É correto o que se afirma em

A) I, apenas.

B) III, apenas.

C) I e II, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa B.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: a operação de projeção é utilizada quando precisamos selecionar colunas específicas de uma relação. Desse modo, a projeção pode ser entendida como uma operação que "filtra" as colunas de uma tabela de dados. A descrição dada na alternativa diz respeito à operação de seleção.

II – Afirmativa incorreta.

Justificativa: a operação de renomeação é utilizada para renomear determinada relação, já que os resultados das expressões da álgebra relacional não recebem um nome pelo qual são referenciados. A descrição dada na alternativa diz respeito à operação de projeção.

III – Afirmativa correta.

Justificativa: a operação de diferença de conjunto é dada como uma subtração entre dois conjuntos de dados. Por exemplo, a operação a – b nos traz como resultado os elementos que pertencem ao conjunto a, mas que não pertencem ao conjunto b. Logo, ela pode ser utilizada quando queremos encontrar tuplas que estão em uma relação, mas não estão em outra.

Questão 2. (Instituto AOCP/2018, adaptada) A linguagem SQL pode ter vários enfoques, e é por meio de comandos SQL que os usuários podem montar consultas poderosas, sem a necessidade da criação de um programa, podendo utilizar ferramentas front-end para a montagem de relatórios.

A respeito da linguagem SQL, avalie as afirmativas a seguir.

I – Uma das vantagens da utilização da linguagem SQL é a sua independência de fabricante, pois é adotada por praticamente todos os SGBDs relacionais existentes no mercado.

II – A linguagem SQL é uma linguagem padronizada (ANSI).

III – Entre seus comandos, o comado ALTER TABLE remove uma tabela do banco de dados especificado.

IV – O comando SELECT é usado para atualizar os dados de uma ou mais tabelas.

É correto o que se afirma em

A) I, apenas.

B) III, apenas.

C) I e II, apenas.

D) II e IV, apenas.

E) I, II, III e IV.

Resposta correta: alternativa C.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: a linguagem SQL é oferecida em quase todos os sistemas de gerenciamento de bancos de dados (SGBDs) atualmente. Isso faz com que ela possa ser utilizada independentemente do fabricante do SGBD, o que é considerado como uma vantagem. Entre os SGBDs atuais que usam o SQL como a linguagem padrão para o acesso às bases de dados, podemos citar o ORACLE, da Oracle Corporation, e o SQL Server, da Microsoft.

II – Afirmativa correta.

Justificativa: o SQL foi originalmente desenvolvido pela IBM, na década de 1970, mas sua expansão ocasionou o surgimento de diversas versões desenvolvidas por outros fabricantes. Essa expansão levou à necessidade de padronização da linguagem. Na década de 1980, essa tarefa foi realizada primeiramente pela American National Standards Institute (ANSI).

III – Afirmativa incorreta.

Justificativa: o comando ALTER TABLE permite a realização de alterações no conteúdo da tabela especificada. A descrição da afirmativa diz respeito ao comado DROP TABLE, que é responsável por remover uma tabela do banco de dados especificado.

IV – Afirmativa incorreta.

Justificativa: o comando SELECT é utilizado para extrair os dados das tabelas de um banco de dados. A descrição da afirmativa diz respeito ao comado UPDATE, que é usado para atualizar os dados de uma ou mais tabelas.