

Tecnologia

BANCO DE DADOS

Giulliana Martins de Souza

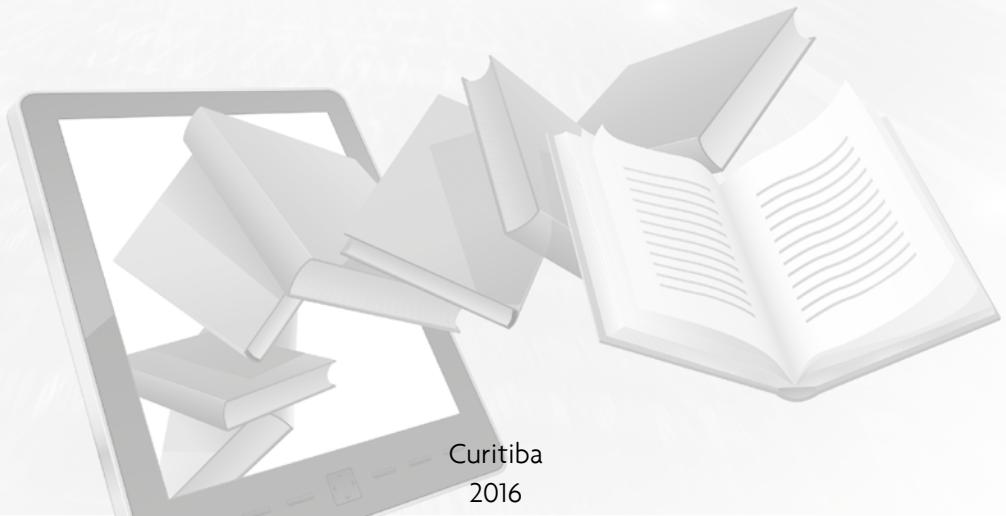
BANCO DE DADOS

Giulliana Martins de Souza



Banco de Dados

Giulliana Martins de Souza



Curitiba
2016

Ficha Catalográfica elaborada pela Fael. Bibliotecária – Cassiana Souza CRB9/1501

S729b Souza, Giuliana Martins de

Banco de dados / Giuliana Martins de. – Curitiba: Fael, 2016.

224 p.; il.

ISBN 978-85-60531-65-3

1. Banco de dados 2. Modelagem de dados 3. Modelo racional
(Informática) 4. SQL (Programa de computador) I. Título

CDD 005.74

Direitos desta edição reservados à Fael.

É proibida a reprodução total ou parcial desta obra sem autorização expressa da Fael.

FAEL

Direção Acadêmica	Francisco Carlos Sardo
Coordenação Editorial	Raquel Andrade Lorenz
Revisão e Diagramação	Editora Coletânea
Projeto Gráfico	Sandro Niemicz
Capa	Vitor Bernardo Backes Lopes
Imagen da Capa	Shutterstock.com/Scanrail1/Frannyanne
Revisão de diagramação	Evelyn Caroline dos Santos Betim

Sumário

DEDICATÓRIA | 5

CARTA AO ALUNO | 7

1. BANCOS DE DADOS | 9
2. MODELAGEM DE DADOS | 25
3. MODELO RELACIONAL E NORMALIZAÇÃO | 43
4. SQL | 59
5. SQL DML | 79
6. SQL AVANÇADO | 103
7. TRANSAÇÕES E TÉCNICAS AVANÇADAS | 127
8. RECURSOS AVANÇADOS | 151
9. BACKUP, RESTORE, SEGURANÇA E OTIMIZAÇÃO | 169
10. TECNOLOGIAS AVANÇADAS | 189

CONCLUSÃO | 205

GABARITO | 207

REFERÊNCIAS | 219

DEDICO ESTE LIVRO à minha família. Agradeço aos meus pais Cleonice e Rubens, pela sua dedicação, por me ensinar que o melhor caminho é o do estudo, e por fazê-lo prioridade na minha educação. Obrigada, tia Neide, por me introduzir ao mundo fascinante da leitura, desde criança, e ainda melhorando a interpretação de texto na hora de contar as estórias dos livros lidos.

Carta ao aluno

PREZADO(A) ALUNO(A),

HOJE É COMUM referir-se à informação como o maior patrimônio de uma organização. Considerando que grande parte dos sistemas de informação e aplicativos em geral manipulam dados e informações que ficam armazenados permanentemente, conhecer sobre banco de dados torna-se imprescindível.

BANCO DE DADOS é uma das grandes áreas da computação, responsável por cuidar das técnicas de coleta, armazenagem, recuperação e distribuição de dados. O termo banco de dados também designa genericamente o *software* utilizado para gerenciar todos os dados de um empreendimento. A área de banco de dados é ampla e fascinante, com um mercado de trabalho atraente, diferentes áreas de atuação e muitas vagas à espera de bons profissionais.

Banco de Dados

Este livro foi elaborado de forma que o aprendizado seja sequencial, fácil e prático. O conteúdo explica a essência de banco de dados, abordando a teoria acadêmica e a prática profissional. Isso permite que, no futuro, você tenha conhecimento suficiente para se preparar para concursos, se especializar em banco de dados ou seguir a carreira acadêmica.

Os capítulos desse livro foram cuidadosamente elaborados, mostrando as características de bancos de dados atuais do mercado. Ele não foi escrito especificamente para o público que deseja trabalhar diretamente com banco de dados, e sim para profissionais de todas as áreas que queiram compreender a criação e funcionamento do banco para programar ou fazer a análise do sistema. Os profissionais que pretendem ser administradores de banco de dados e também gestores, que tomam decisões sobre o futuro da empresa utilizando ferramentas atuais, encontrarão seu espaço no conteúdo deste material.

Espero que esta obra seja útil para estimular o leitor sobre o tema e desta forma despertar o interesse em desbravar as técnicas dos bancos de dados.

Desejo a você muita diversão nesta leitura!

1

Bancos de dados

BANCO DE DADOS é a estrutura utilizada para armazenar dados que precisam ser recuperados posteriormente por algum motivo ou necessidade específica. Por exemplo, quando é necessário armazenar um cadastro de clientes ou até mesmo uma agenda telefônica, pode-se utilizar um caderno ou uma agenda em papel preparada para isso.

Quando se utiliza uma quantidade pequena de dados, é fácil recuperá-los e mesmo alterá-los, mas tudo ainda deve ser feito de modo manual. Com a automatização causada pela utilização de computadores, tornou-se necessária a mudança da forma de trabalhar com o armazenamento dos dados.

Objetivos de aprendizagem:

- ✗ definir bancos de dados;
- ✗ mostrar seus principais conceitos;
- ✗ apresentar seus principais tipos e estruturas.

1.1 Definição

Segundo Date (2000), sistema de banco de dados é basicamente um sistema computadorizado de armazenamento de registros, ou seja, um sistema cujo propósito é armazenar informações e permitir ao usuário buscar e atualizar as informações quando solicitado. Elmasri e Navathe (2013) definem banco de dados como uma coleção de dados relacionados, em que dados indicam fatos conhecidos que podem ser registrados e apresentam significado implícito.

Bancos de dados precisam de um equipamento onde os dados possam ser armazenados e posteriormente recuperados. Esses dados podem ser definidos como qualquer tipo de estrutura que precisa ser armazenada – inicialmente, os dados eram apenas letras e números, por exemplo: nome, endereço, telefone, salário; hoje em dia são armazenadas também voz e imagens.

Inicialmente bancos de dados foram criados para serem utilizados em instituições financeiras e, principalmente, pelas Forças Armadas, que tinham o objetivo de armazenar grande quantidade de dados que depois pudessem ser recuperados. Com a evolução dos computadores de grande porte (*Mainframes*) para os computadores pessoais (*personal computers* – PC) passaram a existir bancos de dados de pequeno e médio porte, que são utilizados em empresas de menor porte.

Bancos de dados são formados pelo *hardware*, que é a parte física em que os dados estão armazenados, e pelo *software*, que gerencia os dados. Atualmente, há equipamentos como celulares e *tablets* com grande capacidade de armazenamento que também utilizam recursos de bancos de dados.

A figura 1.1 mostra a representação de ações que podem ser feitas em um sistema de banco de dados, caracterizando-se por um conjunto de discos, em que se pode executar várias operações, como:

- ✗ recuperar dados;
- ✗ inserir dados;
- ✗ remover dados;
- ✗ alterar dados;
- ✗ transferir dados para outros dispositivos, como *pen drives* ou outros discos;
- ✗ eliminar dados que não são mais necessários;
- ✗ copiar dados;
- ✗ manter a segurança das informações que estão armazenadas;
- ✗ executar a manutenção física e lógica dos dados que estão armazenados.

Figura 1.1 – Banco de dados



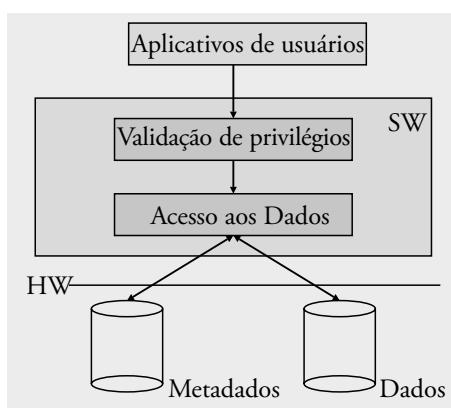
Fonte: Shutterstock.com/Colorlife.

Outro ponto que deve ser observado é a diferença de nomenclatura, por exemplo, banco de dados é qualquer estrutura que armazena os dados, mas também temos os sistemas gerenciadores de banco de dados (SGBD) e os sistemas de banco de dados (SBD). Qual seria a diferença entre eles?

Com o aumento da complexidade dos dados e a necessidade de segurança para esses dados (que eram simplesmente armazenados em um lugar onde qualquer um pudesse acessar a qualquer momento, mesmo sem a permissão necessária), ou até mesmo a necessidade de criação de cópia de segurança dos dados em caso de alguma falha no equipamento, observou-se a necessidade de criar recursos de segurança e melhorias na estrutura do banco de dados. Para isso, foram criados o que chamamos de sistemas de banco de dados ou sistemas gerenciadores de bancos de dados. A diferença é que os sistemas gerenciadores de banco de dados (SGBD), além do armazenamento dos dados, fornecem uma infraestrutura de *software* que protege e otimiza a utilização e o acesso aos dados, ou seja, um SGBD é composto por dados e programas. Atualmente, não existe uma padronização para nomenclatura, então, muitas vezes, um SGBD é referido simplesmente como banco de dados.

Para utilizar um SGBD (figura 1.2), utiliza-se os aplicativos de usuários ou de administradores de SGBD, que, via *software*, acessam o SGBD. Nesse caso, é feita a validação dos privilégios e, em caso deste ser válido, segue para o módulo, que escolhe o caminho e acessa os dados, retornando para quem solicitou o acesso. Até o ponto de acesso aos dados, tudo é gerenciado por *software* (SW), a partir deste ponto, solicita-se o acesso aos dispositivos de *hardware* (HW), onde estão armazenados fisicamente os dados.

Figura 1.2 – Visão macro de um SGBD



Fonte: elaborado pelo autor.

1.2 Conceitos

Os programas que foram incluídos nos sistemas gerenciadores de banco de dados servem para auxiliar e resolver problemas como os apresentados por Silberchatz (1999).

- ✗ Inconsistência e redundância de dados: evita que a informação seja repetida em diversos lugares e de maneira diferente umas das outras. Por exemplo, a mudança de endereço de um cliente poderia ser atualizada em um lugar e não no outro, deixando a dúvida de qual o endereço deve ser o correto.
- ✗ Dificuldade de acesso aos dados e isolamento dos dados: garantir que, ao projetar o banco de dados, sejam atendidas todas as solicitações e ações necessárias às tarefas dos usuários, agregando dados que estejam dispersos em vários arquivos.
- ✗ Problemas de integridade: os valores dos dados atribuídos e armazenados devem satisfazer restrições para garantir a consistência. Por exemplo, todo cliente cadastrado deve ter um nome.
- ✗ Problemas de atomicidade: em caso de falhas em um sistema de banco de dados, deve-se garantir que a transação executada seja completamente finalizada ou totalmente desfeita. Por exemplo, no caso de transferência de saldo entre contas correntes, se após o saque da conta original ocorrer uma falha, o sistema deve garantir, já que não foi possível crédito na conta destino, que o dinheiro seja devolvido à conta origem.
- ✗ Anomalias no acesso recorrente: muitas vezes acontece o acesso simultâneo a determinados dados em um banco, nesse caso, o SGBD deve garantir que o dado seja acessado pelas demais solicitações somente depois de finalizada a ação já iniciada.
- ✗ Problemas de segurança: somente as pessoas autorizadas devem acessar o banco de dados e de acordo com privilégio permitido a cada um dos usuários.

Importante

A sigla ACID resume os itens que são garantidos pelo SGBD: Atomicidade, Consistência, Integridade e Durabilidade.

Em relação à utilização de bancos de dados, pode-se definir vários tipos, conforme segue:

- × analista ou projetista de banco de dados – analisa o negócio e as necessidades para a modelagem e criação do banco;
- × administrador de bancos de dados (DBA) – responsável por criação, gestão e manutenção dos bancos de dados. Podem ser divididos em DBA de desenvolvimento, projeto ou produção;
- × usuários finais – utilizam os bancos de dados por meio de aplicativos que acessam o SGBD;
- × usuários sofisticados – analistas de negócios que muitas vezes utilizam ferramentas de *business intelligence* (BI) para acessar os bancos de dados e extrair relatórios e gráficos.

Todos os usuários são importantes para o projeto e criação do banco de dados, afinal cada uma de suas tarefas vai definir o que precisa ou não ser armazenado.

1.3 Estrutura de banco de dados

Os bancos de dados são divididos e armazenados fisicamente em arquivos. Para o sistema operacional, esses arquivos têm estruturas próprias e geralmente só podem ser abertos corretamente pelo SGBD. Ao tentar acessar os arquivos por outra ferramenta, tentar mover o arquivo ou até mesmo apagá-lo enquanto o SGBD estiver em funcionamento, a ação não será permitida pelo sistema operacional, que enviará a mensagem de erro de arquivo em uso.

Como banco de dados é um consumidor muito grande dos recursos do sistema operacional, principalmente na entrada e saída de dados (*input e output – IO*), utilização da CPU (processador) e de memória, existem funções dentro do SGBD que se comunicam diretamente com o sistema operacional via chamadas de sistema (*system calls*).

Dependendo do SGBD utilizado, ele pode oferecer diferentes estruturas de auxílio no gerenciamento dos recursos. Alguns oferecem somente

o acesso aos dados, com tratamento básico de proteção e privilégios, outros oferecem recursos de programação dentro do banco de dados como *triggers*, *functions* e *stored procedures* – além de um nível profundo de gerenciamento de proteção e segurança, como criptografia. O mais importante é conhecer as estruturas e analisar qual o SGBD mais apropriado para as necessidades de determinado cliente.

Para conversar com o banco de dados, é necessária uma linguagem específica para que o SGBD entenda quais ações devem ser executadas. Atualmente, a linguagem mais utilizada para banco de dados é a SQL (*Structured Query Language*), que permite definir, manipular e gerenciar estruturas de dados de um banco de dados.

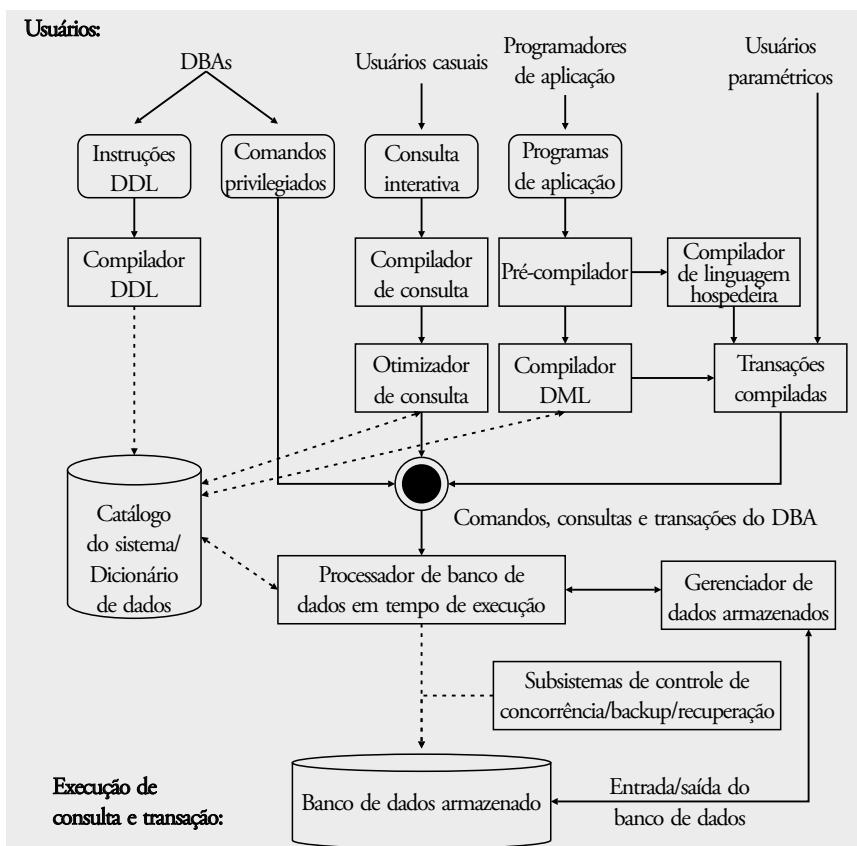
Um item especial da linguagem SQL são as instruções DDL (*Data Definition Language*). Essas instruções de linguagem de definição de dados são utilizadas para criar os SGBDs e as estruturas que o compõem. A figura 1.3 apresenta os DBAs, que podem executar comandos privilegiados e instruções DDL (DBAs têm privilégios de *system administrator*, dentro do banco de dados, ou seja, eles podem executar qualquer tarefa dentro do banco de dados). Após as instruções DDL, vem o compilador DDL, que verifica se a instrução está escrita de maneira correta; após essa etapa, a execução é feita numa área especial de armazenamento do banco de dados, que é conhecida como catálogo do sistema, dicionário de dados ou metadados. Nessa área, também são armazenados os dados estatísticos que direcionam a melhor forma de conduzir a pesquisa dentro do banco de dados.

Além dos DBAs, acessam banco de dados usuários casuais, programas de aplicação e usuários paramétricos. Pode-se ter consultas interativas e consultas já existentes nos programas de aplicação. As consultas podem passar por um compilador de consulta e por um compilador de DML (*Data Manipulation Language*) – outro grupo de instruções SQL utilizado para a manipulação de dados –, para verificar se a instrução SQL está correta. Após a verificação da estrutura, pode-se acessar o otimizador de consulta, para escolher nas estatísticas do catálogo do sistema o melhor caminho a ser percorrido para buscar o resultado necessário para a solicitação.

Banco de Dados

Na parte inferior da figura 1.3, além do catálogo do sistema, existe a área utilizada para o armazenamento dos dados do banco de dados. O subsistema controla a concorrência de acesso aos dados e também é responsável por *backup* (cópia de segurança) e recuperação de dados (em caso de alguma falha física do equipamento ou do usuário). O subsistema gerenciador de dados armazenados verifica em qual arquivo físico do sistema operacional deve armazenar os dados e como a manutenção física deve ser feita.

Figura 1.3 – Módulos componentes de um SGBD



Fonte: Elmasri; Navathe (2013).

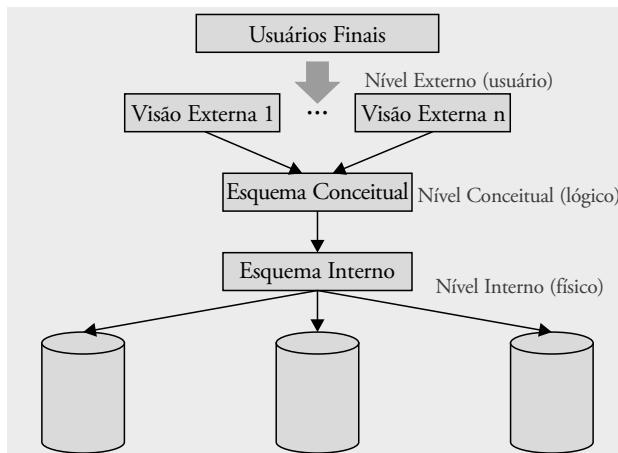
O processador de banco de dados em tempo de execução recebe comandos, consultas e transações, verifica o catálogo do sistema e finalmente faz o acesso aos dados devolvendo o resultado para quem solicitou a execução.

Bancos de dados podem ser classificados de acordo com a sua estrutura. Podem ser de modelo hierárquico, redes, relacionais e orientados a objetos. Atualmente, o modelo mais utilizado é o relacional.

A arquitetura de um SGBD (SILBERSCHATZ, 1999) pode ser definida em três esquemas (figura 1.4), cujo objetivo é separar o banco de dados físico, do usuário comum de acordo com a visibilidade da estrutura que cada um pode ter. Segundo a literatura, eles podem ser divididos em níveis.

- ✗ Externo: mostra a visão que o usuário tem do banco de dados. Para cada usuário ou grupo de usuário é apresentada somente a parte do banco necessária para sua utilização.
- ✗ Conceitual: também conhecido como modelo lógico, é a descrição do banco de dados inteiro, mostra entidades, relacionamentos e restrições.
- ✗ Físico ou interno: apresenta como o banco de dados está armazenado fisicamente, descrevendo detalhes e caminhos de acesso.

Figura 1.4 – Arquitetura de três esquemas



Fonte: Elmasri; Navathe (2013).

Todas as estruturas apresentadas dos SGBDs têm suas tarefas específicas e complexidade, cujo objetivo é facilitar e otimizar as tarefas de acordo com a funcionalidade. Por exemplo, as estatísticas permitem que se escolha o melhor caminho para acessar os dados, otimizando o tempo de resposta, enquanto os compiladores de DDL e DML verificam o comando antes de acessar os dados, garantindo que não executem acessos desnecessários, causados por comandos errados. A arquitetura facilita a visibilidade de cada um dos agentes que utilizam o SGBD, e a sua classificação auxilia na escolha daquele que melhor se adequa a cada situação.

1.4 Utilitários de SGBD

Os SGBDs, além de apresentação de gerenciamento dos dados e da segurança destes, podem oferecer outras ferramentas para auxiliar no gerenciamento de banco de dados.

As ferramentas podem ser as seguintes:

- ✗ *carga* – existe a necessidade de carregar arquivos de dados existentes (txt, csv, etc.); para facilitar, o arquivo a ser lido precisa estar no formato que o SGBD consiga reconhecer, com isso a ferramenta de carga recebe o arquivo e importa para o SGBD. Existe também a opção de importar e exportar dados, que permite a conexão com outros SGBDs para carregar dados ou enviar dados;
- ✗ *backup* – na história da computação, existe uma ferramenta importante que deve ser usada em todos os aplicativos, o *backup*, afinal, erros físicos e lógicos podem acontecer. Um usuário pode apagar registros, tabelas ou até mesmo o banco de dados, intencionalmente ou não; falhas físicas também podem acontecer nos discos, memória, CPU, etc. A melhor forma de garantir que pode se recuperar o banco de dados é fazer uma cópia de segurança, também chamada de *backup*. Essas cópias podem ser feitas em disco ou fitas, devem ser feitas com frequência e guardadas em lugar seguro. Os backups podem ser criptografados e protegidos por senha, para evitar acesso indevido aos dados;

- ✗ reorganização – à medida que os dados vão sendo inseridos, o banco de dados vai expandindo a sua ocupação nos discos físicos, mas à medida que os dados são apagados, os espaços ocupados por esses dados são deixados vazios. Dependendo da frequências das operações e da quantidade de dados que estão sendo apagados, os arquivos de dados e índices do banco de dados vão ficando fragmentados (semelhante a fragmentação de disco do sistema operacional), para melhorar o acesso e desempenho, além de liberar espaço em disco, o SGBD permite a reorganização dos dados e índices;
- ✗ monitoração – o SGBD pode monitorar o ambiente do banco de dados e gerar alertas para determinados erros ou condições enviando *e-mails*, mensagens, etc.;
- ✗ análise de desempenho – essa ferramenta é muito importante, pois consegue analisar o ambiente de memória, CPU, disco, consultas (*Query*) e outras estruturas do SGBD, sugerindo melhorias, criação de índices e aumento de memória.

Dependendo do SGBD, existem outras ferramentas para *Data Mining*, *Dataware house*, *Business Inteligence* e outras mais. À medida que a necessidade dos usuários vai crescendo os SGBDs vão evoluindo e apresentando ferramentas integradas ou não, que podem ser o diferencial na escolha do SGBD pelo cliente.

1.5 Arquiteturas de banco de dados

A arquitetura de banco de dados segue a evolução das arquiteturas computacionais. Existem as seguintes arquiteturas:

- ✗ centralizada;
- ✗ cliente servidor;
- ✗ distribuídos;
- ✗ orientado a objeto;
- ✗ virtualização;
- ✗ nuvem.

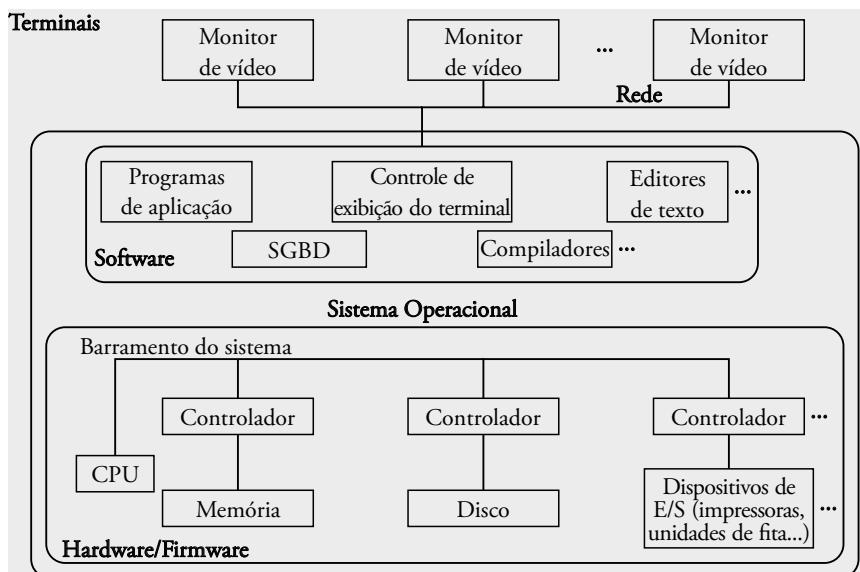
A tendência dos SGBDs é evoluir da mesma forma. Deve-se levar em consideração a necessidade de cada um. Não existe arquitetura melhor e perfeita, o que existe é a melhor arquitetura de acordo com cada necessidade. Atualmente, as mais utilizadas são centralizada e cliente-servidor.

1.6 Arquitetura centralizada

O processamento é executado remotamente em um computador central, e os dados são enviados, via rede, para os computadores locais. Foi muito utilizada no início, depois muitas aplicações foram direcionadas para arquitetura cliente-servidor; com a virtualização, passou a ser muito utilizada novamente.

A figura 1.5 apresenta um modelo de arquitetura centralizada, em que há um computador de grande porte (*Mainframe*) rodando sobre um sistema operacional e vários terminais (geralmente máquinas sem processamento local, ou PCs que emulam terminais).

Figura 1.5 – Arquitetura centralizada



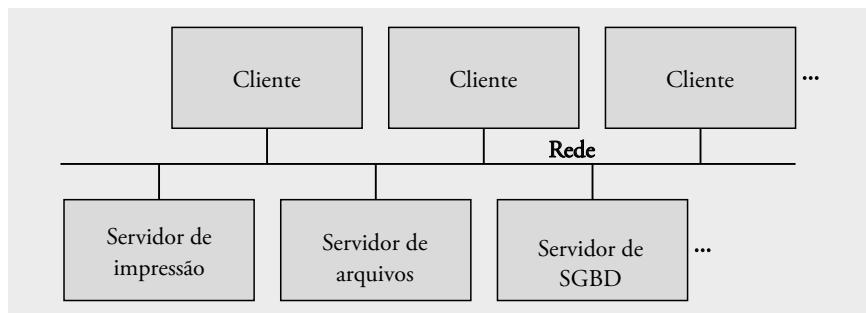
Fonte: Elmasri; Navathe (2013).

O hardware foi se tornando mais barato, e muitos usuários foram adquirindo PCs. Da mesma forma, a internet passou a ser mais acessível, e, lentamente, os sistemas SGBDs foram acompanhando a evolução e utilizando o poder de processamento oferecido pelo usuário, levando à utilização de arquiteturas cliente-servidor.

1.7 Arquitetura cliente-servidor

Na arquitetura cliente-servidor, há vários clientes e pode-se ter um ou mais servidores. Segundo Elmasri e Navathe (2013), a arquitetura cliente-servidor foi desenvolvida para utilizar computadores pessoais, estações de trabalho e servidores de arquivos. A ideia era definir servidores especializados para cada funcionalidade. A figura 1.6 mostra a arquitetura cliente-servidor no nível lógico, e a figura 1.7 mostra arquitetura cliente-servidor em duas camadas. Nessa abordagem, o nível de servidor inclui o *software* do SGBD, responsável por armazenamento, concorrência e recuperação, e o nível do cliente trata de funções de interações do SGBD, como tipos de dados e dicionário de dados.

Figura 1.6 – Arquitetura cliente-servidor

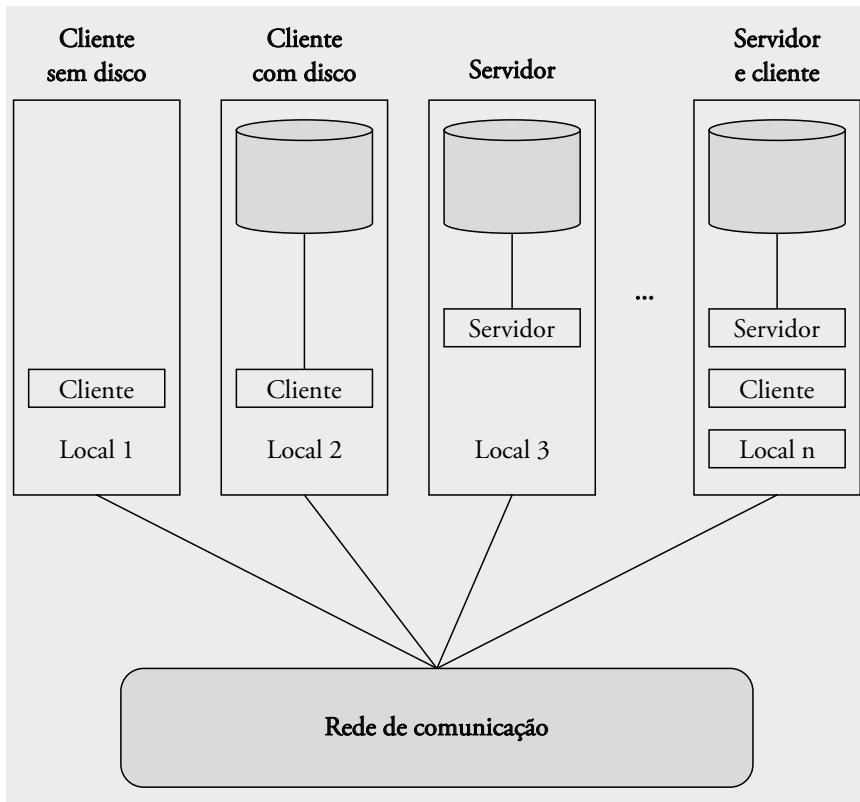


Fonte: Elmasri; Navathe (2013).

Arquitetura em duas camadas (figura 1.8) é formada por componentes de *softwares* distribuídos – uma parte rodando no cliente, a outra rodando no servidor –, cuja vantagem são simplicidade e compatibilidade com os sistemas existentes.

Banco de Dados

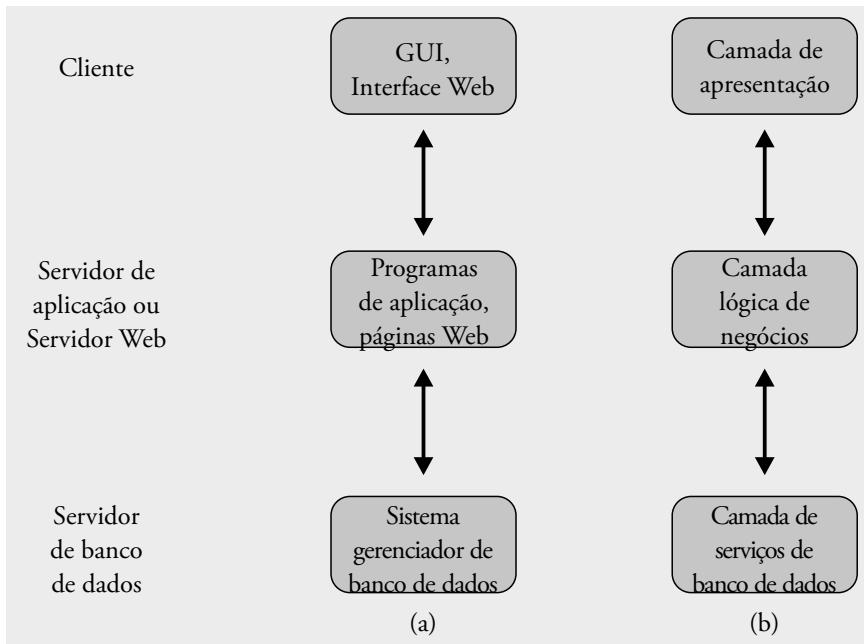
Figura 1.7 – Arquitetura cliente-servidor



Fonte: Elsmari, 2013.

Com a evolução dos *softwares* de aplicação (figura 1.9), passou a existir uma camada intermediária, denominada servidor de aplicação, que pode ser um servidor *web*, por exemplo. Esse serviço executa uma função intermediária entre a execução do programa de aplicação e o armazenamento de regras de negócios, que podem ser procedimentos ou restrições. Nesse caso, o cliente tem a interface gráfica e algumas regras específicas da aplicação, já o servidor de aplicação aceita e processa as solicitações do cliente e envia consultas e comandos do banco de dados para o servidor de banco de dados (ELMASRI; NAVATHE, 2013).

Figura 1.8 – Arquitetura cliente-servidor em três camadas



Fonte: Elmasri; Navathe (2013).

Outra melhoria ocorrida, com os avanços da tecnologia de criptografia, é a possibilidade de trafegar dados confidenciais pela rede, em formato criptografado, do servidor ao cliente e vice-versa. Isso pode ser feito tanto por *hardware*, quanto *software*, outro item importante é compactação de dados, que ajuda a transferir grande quantidade de dados entre cliente e servidor por redes com ou sem fio (Wi-fi).

Síntese

O capítulo trouxe o banco de dados para o mundo conceitual. Inicialmente, tinha-se uma visão abstrata e obscura do funcionamento de banco de dados e de um sistema gerenciador de banco de dados. Apresentou-se suas principais diferenças e a evolução de serviços oferecidos, garantindo integridade, durabilidade,

atomicidade e consistência aos dados armazenados, além das ferramentas que foram criadas para facilitar todas as tarefas necessárias. Graças aos sistemas gerenciadores de bancos de dados, o mundo computacional jamais será o mesmo.

Foram mostradas também as ferramentas que podem ser utilizadas com os SGBDs e a evolução das arquiteturas.

Da teoria para a prática

Pesquise os SGBDs existentes no mercado e elabore um texto justificando qual seria o sistema adequado para cada um dos clientes a seguir.

- ✗ Confeitaria de pequeno porte utilizando sistema operacional Windows 10.
- ✗ Papelaria de médio porte utilizando sistema operacional Linux Ubuntu.
- ✗ Sistema de agendamento de pacotes de viagens, com mais de 300 lojas no Brasil, utilizando sistema Windows 8.1.
- ✗ Sistemas de geoprocessamento para análise da geografia do estado do Amazonas.

2

Modelagem de Dados

BANCO DE DADOS (BD) são utilizados para armazenar dados importantes e que posteriormente possam ser recuperados. Mas deve-se armazenar todos os dados? Tudo é importante? Como diferenciar e compreender o que realmente deve ir para o banco e o que não precisa ser armazenado? Cada negócio e cada empresa têm dados diferentes, por isso cada situação deve ser analisada cuidadosamente. Após essa análise, quais são os próximos passos?

Objetivos de aprendizagem:

- × apreender entidade;
- × conhecer relacionamentos;
- × Modelo Entidade Relacionamento;
- × Modelo Entidade Relacionamento Estendido.

2.1 Modelo Entidade Relacionamento (MER)

O Modelo Entidade Relacionamento foi criado para mapear o negócio que está sendo analisado em estruturas de modelagem de banco de dados.

Importante

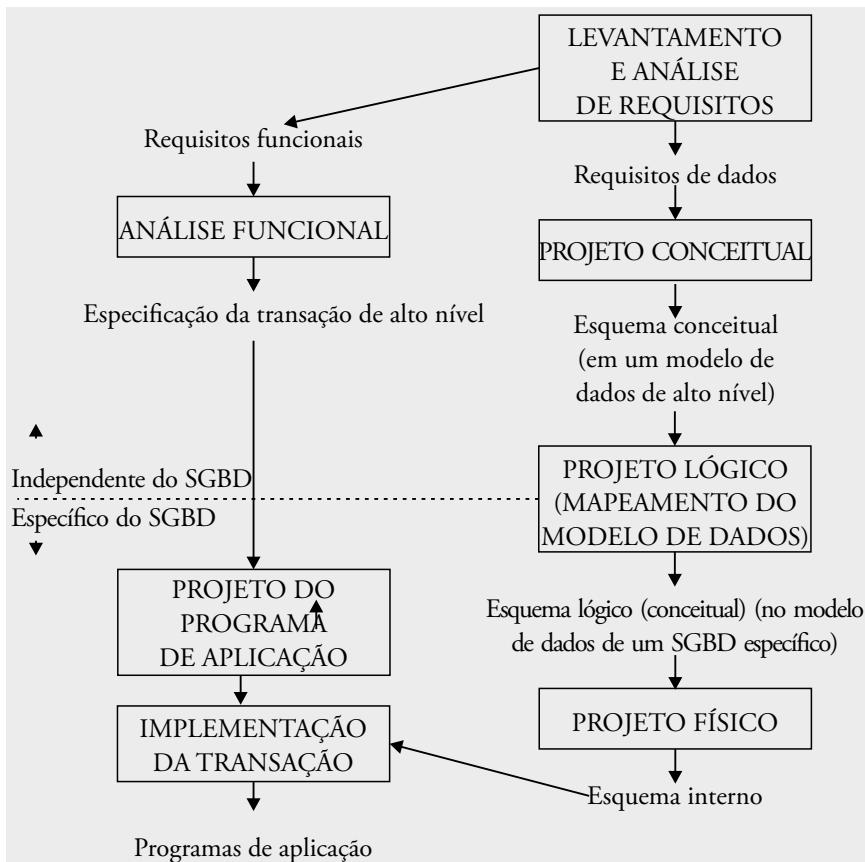
Quando analistas começam a avaliar o negócio para construir uma solução, escolhem uma modelagem para desenvolvimento, que pode ser análise estruturada, orientada a objeto, entre outras. Essas soluções são utilizadas para o desenvolvimento do sistema em si, e não do banco de dados; para a modelagem e o desenvolvimento do banco de dados são utilizadas outras ferramentas.

Durante a criação dos sistemas, acontecem várias reuniões e discussões das tarefas necessárias para os clientes, começando pelo levantamento e pela análise de requisitos (Figura2.1) e quais dados são tramitados em cada etapa das tarefas. Nesse momento, o analista responsável pelo projeto de banco de dados deve preparar o modelo conceitual, evoluir para o projeto lógico, identificando quais dados devem ser armazenados e como, e finalmente chegar ao modelo físico e à implementação.

Importante

Durante o levantamento de dados, nunca “ache”, tenha sempre certeza, pergunte quantas vezes forem necessárias e revise o projeto com o cliente.

Figura 2.1 – Principais fases do projeto de BD



Fonte: Elmasri, Navathe (2013). p. 133.

Para saber quais dados devem ser armazenados, deve-se discutir com o cliente quais informações transitam entre os processos, identificar as características de cada um deles e verificar quais realmente precisam ser armazenados. Essa é uma etapa cansativa e delicada e não deve ser feita com pressa e incertezas.

Nos itens a seguir, começaremos a estudar elementos que constituem a base para os modelos lógicos e físicos.

2.2 Entidade

Entidade é o elemento principal para a criação do modelo de entidade relacionamento. É nessa estrutura que se esclarecem quais são as características do objeto ou elemento que se pretende definir (DATE, 2000).

Uma entidade pode ser uma estrutura do mundo real, por exemplo, um carro, como também pode ser uma estrutura abstrata, como o aluguel de um carro. No caso do carro, tem-se características que separam um carro de uma moto ou de um caminhão: marca, modelo, ano de fabricação, chassi, quantidade de portas, entre outros. Pode-se tocar no carro, ele existe de forma concreta, mas, no caso do aluguel do carro, não existe um elemento concreto que se possa tocar, existem somente características que o definem: placa, data da locação, cliente, entre outros.

Uma entidade é composta por atributos, que são características, propriedades, elementos que a definem como uma estrutura única diferente de outra entidade. Aquilo que diferencia um carro de uma moto, por exemplo.

2.3 Atributos

Os atributos definem a entidade, e cada atributo é um elemento com valor que caracteriza a entidade.

Segundo Silberchatz, Korth e Sudarshan (1999), os tipos de atributos são:

- ✗ **simples ou compostos** – os atributos simples são estruturas atômicas que não podem ser divididas em outras menores, por exemplo, idade. E os atributos compostos são formados por mais de uma estrutura, como data de nascimento, formada por dia, mês, ano.
- ✗ **monovalorados ou multivalorados** – monovalorados são atributos que só podem ter um único valor em determinado momento, por exemplo, a data de nascimento é única para cada pessoa, mas telefones podem ser vários: celular, residencial e comercial; nesse caso, são multivalorados.
- ✗ **nulos** – são atributos que ainda não têm valores, mas em determinado momento receberão valor, por exemplo: existe um registro

da análise e da publicação de um livro; enquanto está em fase de análise, o valor da data de publicação é nulo, pois ainda não foi publicado. Zero e espaço em branco não são valores nulos, e sim valores válidos para os respectivos tipos de dados.

- ✗ **derivados** – são atributos que podem ser calculados a partir de outros atributos ou a partir de fórmulas aplicadas sobre atributos. Caso existam os atributos idade e data de nascimento, pode-se aplicar facilmente a subtração da data atual pela data de nascimento, obtendo-se a idade, portanto, a idade nesse caso é um atributo derivado.

Importante

Siga sempre um padrão de nomenclatura para todas as entidades e atributos, seja para início com letras maiúsculas ou minúsculas, seja com abreviações, entre outros.

Na Figura 2.2, a entidade foi nomeada Cliente e tem os atributos de CPF, RG, Endereço, Dt_Nasc e Telefone. Em cada modelo, nomes de entidades não podem ser repetidos e cada entidade deve ter um nome diferente do outro. A escolha do nome da entidade deve ser feita de maneira que indique claramente o que ela representa, dessa forma, qualquer pessoa que ler a documentação compreenderá facilmente o que ela representa.

Figura 2.2 – Entidade



Fonte: elaborado pela Autora.

Saiba mais

Para nomes de entidades e atributos, é importante utilizar somente letras, números e _ (sublinhado), começando sempre por letra. Isso evita problemas posteriores com importação e exportação de dados e estruturas e facilita a criação de comandos e acesso a outros bancos de dados. É possível utilizar caracteres especiais e acentos, mas não é recomendável.

Sendo a entidade formada por atributos, é importante selecionar de forma crítica e com bastante atenção todos os atributos que representam a entidade de forma completa. Qualquer falha no projeto do banco de dados pode comprometer toda a solução.

2.4 Relacionamento

Entre as entidades, podem existir associações, que são chamadas de relacionamentos.

Os relacionamentos podem ter vários níveis (ELMASRI; NAVATHE, 2013):

- ✗ **binários** – entre duas entidades;
- ✗ **ternário** – entre três entidades;
- ✗ “**n-ário**” – entre mais de três entidades.

Quanto maior for a quantidade de relacionamentos entre as entidades, mais complexo ficará o banco de dados. Relacionamentos também podem ter atributos, e isso acontece quando a associação entre as entidades gera mais dados que precisam ser armazenados. Basicamente, são as entidades que não são concretas, o resultado e alguma ação ou tarefa que precisa ser armazenada, por exemplo, o aluguel de um carro.

Definir os relacionamentos é mais complexo do que se imagina. As entidades têm características que as definem claramente. Os relacionamentos podem existir entre várias tabelas e dependendo da ação do processo que pode causar essa associação nem sempre é fácil visualizar. Algumas vezes é necessário olhar os possíveis valores de atributos para encontrar o relacionamento.

Para facilitar a compreensão, utiliza-se a letra E para representar uma Entidade.

Importante

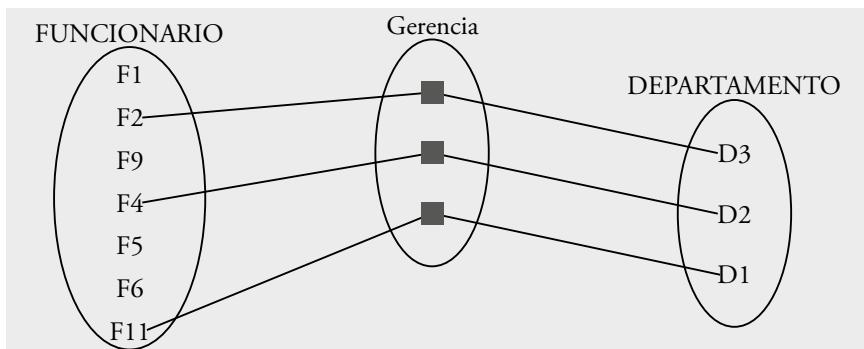
Outro fator a ser considerado é se o relacionamento é obrigatório ou não. Quando for obrigatório (participação total), para cada elemento existente na E1, existirá pelo menos um elemento relacionado na E2. Para relacionamentos não obrigatórios (participação parcial), para cada elemento na E1 pode não existir nenhum elemento relacionado na E2 ou podem existir um ou mais elementos referenciados na E2.

Também temos a cardinalidade (restrições de mapeamento de relacionamentos), que indica o número de relacionamentos dos quais uma entidade pode participar.

A cardinalidade apresenta os seguintes tipos:

- ✗ **1:1** – relacionamentos um para um indicam que, para cada um elemento na entidade E1, existe somente um elemento na entidade E2. A Figura 2.3 apresenta esse relacionamento, no qual um funcionário gerencia somente um departamento e cada departamento é gerenciado por apenas um funcionário

Figura 2.3 – Cardinalidade 1:1

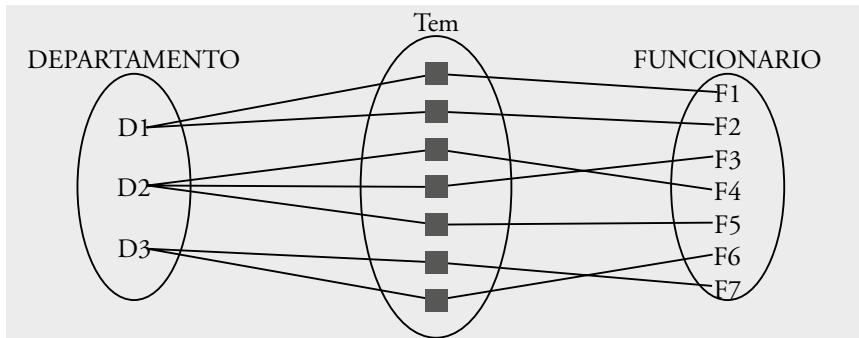


Fonte: elaborado pela autora .

Banco de Dados

- × **1:N** – relacionamentos um para N indicam que, para cada elemento de E1, podem existir N (vários) elementos na E2. A Figura 2.4 apresenta esse relacionamento, no qual um departamento pode ter vários empregado, mas cada empregado pertence a somente um departamento.

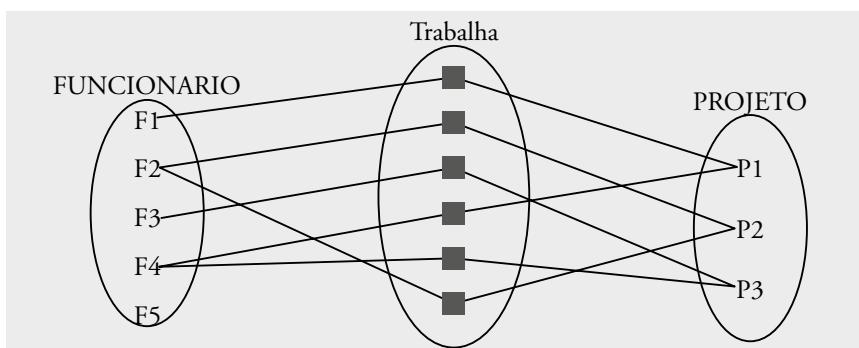
Figura 2.4 – Cardinalidade 1:N



Fonte: elaborado pela autora.

- × **N:M** – relacionamentos N para M indicam que, para cada elemento de E1, podem existir N (vários) elementos na E2, e para cada elemento em E2 podem existir M (vários) elementos em E1. Esse relacionamento é apresentado na Figura 2.5, no qual um funcionário pode trabalhar em vários projetos e um projeto pode ter vários funcionários.

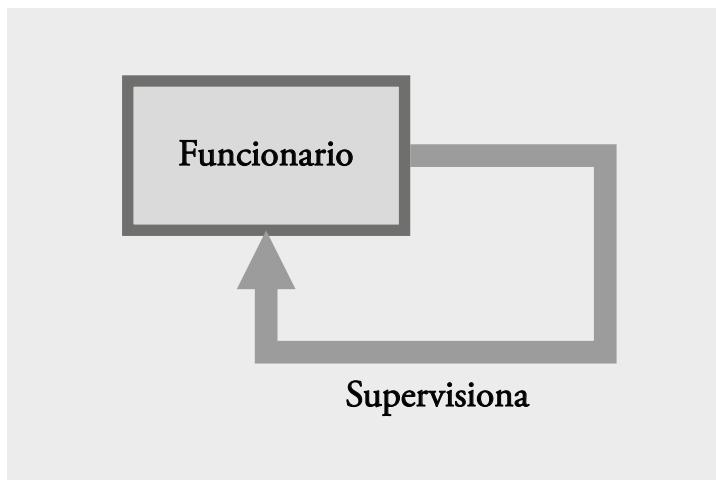
Figura 2.5 – Cardinalidade N:M



Fonte: elaborado pela autora.

Existe também um relacionamento incomum, mas completamente válido, chamado de relacionamento recursivo ou autorrelacionamento, que ocorre quando a entidade se relaciona com ela mesma. A Figura 2.6 mostra esse relacionamento, no qual um funcionário supervisor supervisiona um ou vários funcionários. Nesse caso, não existe a necessidade de criar uma entidade supervisor com os mesmos atributos de funcionários, e sim criar um relacionamento da entidade com ela mesma.

Figura 2.6 – Relacionamento recursivo



Fonte: elaborado pela autora.

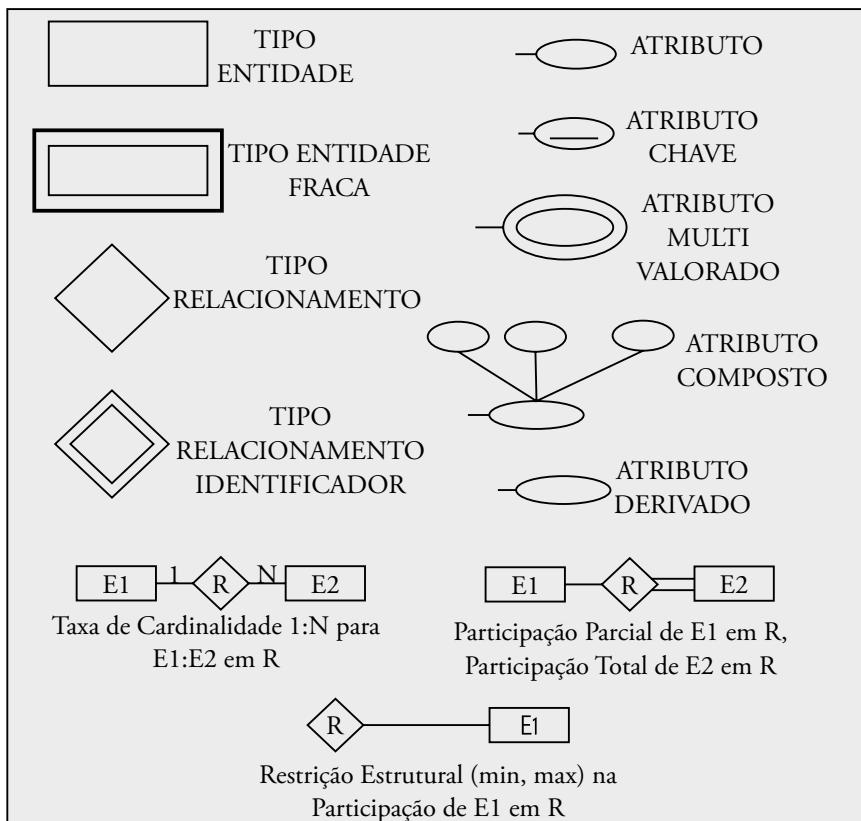
Com a definição de entidade e relacionamento, pode-se apresentar o Diagrama de Entidade Relacionamento, uma das formas de representação gráfica do modelo entidade relacionamento.

2.5 Diagrama de Entidade Relacionamento

O Diagrama de Entidade Relacionamento (DER) é formado por um conjunto de objetos gráficos que representa todos os objetos do modelo Entidade Relacionamento, tais como entidades, atributos, atributos-chaves, relacionamentos, restrições estruturais etc.

O DER fornece uma visão lógica do banco de dados, com um conceito mais generalizado de como estão estruturados os dados de um sistema. Existem inúmeras formas de representação para um DER; uma delas foi criada por Peter Chen, como mostrado na Figura 2.7, na qual pode-se representar entidades, relacionamentos, cardinalidades, tipos de atributos e restrições.

Figura 2.7 – DER

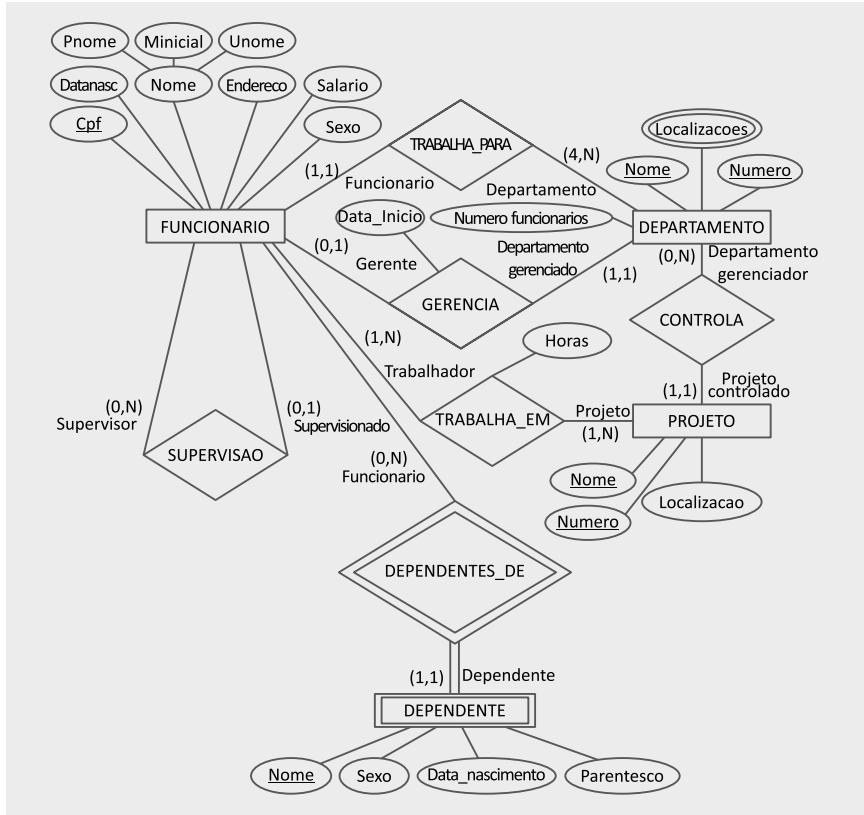


Fonte: adaptado de Chen (1990).

Inicialmente, essa abordagem é muito interessante para o aprendizado, principalmente quando se está atuando no modelo lógico, recuperando e identificando as informações de negócios.

O modelo da Figura 2.8 mostra o DER de uma empresa que tem vários departamentos nos quais trabalham vários funcionários. Os funcionários podem ter nenhum, um ou vários dependentes. Existem vários projetos vinculados a departamento nos quais vários funcionários podem atuar.

Figura 2.8 – DER de uma empresa



Fonte: Elmasri, Navathe (2013) p. 149.

Existem inúmeras maneiras de representar o mesmo modelo. Analisando-se a forma gráfica, é muito fácil compreender quais atributos formam as entidades, quais são chaves primárias, os relacionamentos existentes, a cardinalidade e as entidades fracas.

2.5.1 Chaves

Outras restrições muito importantes em uma entidade são as chaves. A entidade possui um atributo cujos valores são distintos para cada elemento entidade individual. Esse atributo é chamado chave e seus valores podem ser utilizados para identificar cada elemento da entidade de forma única ou aplicar a integridade referencial entre relacionamentos.

As chaves podem ser:

- ✗ **candidatas** – atributos que atendem à condição de chave primária. Por exemplo: CPF, RG.
- ✗ **primárias (*primary key*)** – atributo cujo conteúdo não se repete em nenhum elemento dentro da entidade; dessa forma, esse atributo consegue identificar cada elemento individualmente. Ele deve ser único e não nulo. Por exemplo: CNPJ.
- ✗ **concatenadas ou compostas** – quando a entidade não apresenta nenhum atributo que seja chave candidata, ela não teria chave. Mas ainda existe a opção de juntar um ou mais atributos, e a combinação desses atributos atender às necessidades da chave primária. Por exemplo: no relacionamento Aluguel, que tem os atributos CPF, Dt_hr_locacao, ao combinarmos esses dois, temos uma chave primária, afinal, um cliente pode alugar dois carros no mesmo dia, mas não na mesma hora, minuto e segundo.
- ✗ **estrangeiras (*foreign key*)** – servem para criar a integridade referencial, ou seja, criar dentro do banco o relacionamento existente no modelo.

Saiba mais

Quando existe mais de uma chave candidata, quais critérios devem ser utilizados para escolher a melhor para ser a chave primária?

- ✗ Escolher a de menor tamanho.
- ✗ Entre atributos numéricos e strings, escolher o numérico.
- ✗ Escolher o atributo mais usado pela aplicação.

A escolha correta da chave primária é muito importante para o bom funcionamento do banco de dados.

2.5.2 Entidades fracas

As entidades fracas são aquelas que não têm chave primária, nem mesmo ao tentar a composição entre os atributos (chave primária composta). Isso implica que as combinações dos valores de seus atributos podem ser idênticas. As entidades desse tipo precisam estar relacionadas com uma entidade pertencente ao tipo entidade proprietária. E esse relacionamento é chamado de relacionamento identificador.

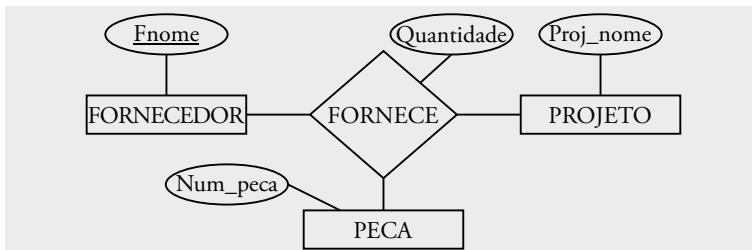
Um tipo de entidade fraca possui uma chave parcial que, juntamente com a chave primária da entidade proprietária, forma uma chave primária composta. Outra forma de se resolver essa condição é criar um campo identificador, geralmente um atributo de autoincremento, chamado código.

2.5.3 Relacionamentos com grau maior que dois

Relacionamentos binários ocorrem entre duas entidades, mas pode haver relacionamentos ternários ou até com graus maiores. A quantidade de entidades relacionadas entre si define o grau do relacionamento.

A Figura 2.9 mostra um exemplo de relacionamento ternário, no qual há um fornecedor enviando peças para um projeto. Nesse caso, temos de dizer exatamente quantas peças foram enviadas pelo fornecedor para o projeto. Temos também mais um exemplo de um relacionamento FORNECE com atributos.

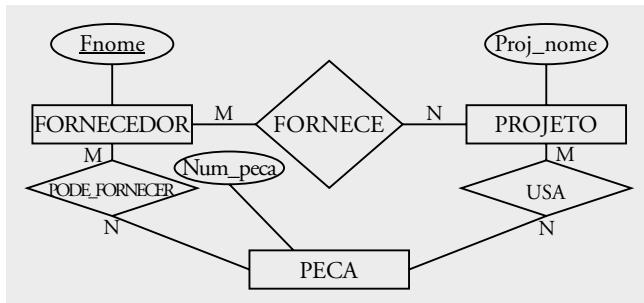
Figura 2.9 – Relacionamento ternário



Fonte: Elmasri, Navathe (2013) p. 151.

Para fazermos uma comparação, veja a Figura 2.10. Temos três relacionamentos binários: um representa quais peças o fornecedor pode fornecer, quais fornecedores atendem aos projetos e quais peças podem ser enviadas para o projeto; essa representação diz o que pode ser feito, o que pode ser fornecido e a quem, bem diferente do exemplo da Figura 2.9, que mostra quem forneceu quais e quantas peças para quais projetos.

Figura 2.10 – Relacionamento binário



Fonte: Elmasri, Navathe (2013) p. 151.

Importante

Modelar corretamente exige treino, testes e experiência. Cada vez que os requisitos são analisados, o Administrador de banco de dados (DBA) aprofunda o conhecimento e melhora o modelo, e somente a prática dirá quais das soluções citadas pode ser a melhor. Em caso de dúvida, deve-se perguntar ao cliente de que ele precisa exatamente, se deve ser de acordo com a Figura 2.8 ou a Figura 2.9. Ambas estão corretas, tudo depende da necessidade do cliente.

2.6 Modelo Entidade Relacionamento Estendido

O modelo Entidade Relacionamento permite modelar a maioria dos bancos de dados, mas, devido a determinadas aplicações, surgiu a necessidade de novas estruturas para a modelagem de informações mais complexas.

O modelo Entidade Relacionamento Estendido (ERE) visa fornecer essas estruturas para permitir a representação de informações complexas. O modelo estendido engloba todos os conceitos do modelo ER mais os conceitos de subclasse, superclasse, generalização e especialização.

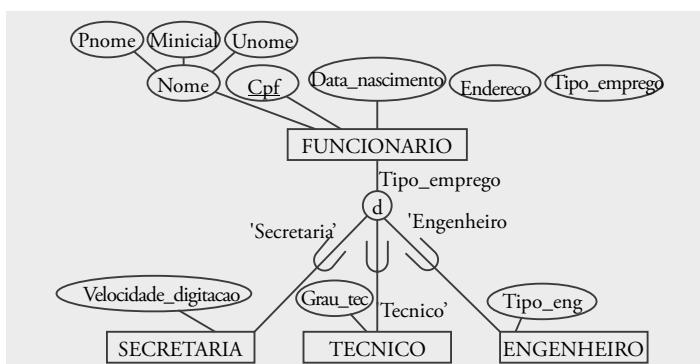
Segundo Elmasri e Navathe (2013), o modelo estendido permite demonstrar estruturas gráficas como apresentado na Figura 2.11. Observe que temos a entidade Funcionário com as características que são comuns às entidades: Secretária, Técnico e Engenheiro. Essas entidades têm atributos independentes: secretária tem velocidade_digitacao, Técnico tem grau_tec e Engenheiro tem tipo_eng.

2.6.1 Superclasse e subclasse (Supertipo e subtipo)

A entidade Funcionário é chamada de superclasse ou supertipo, pois apresenta características comuns às demais entidades, enquanto as demais entidades fazem parte da subclasse ou subtipo.

Além dos atributos comuns e específicos, outro fator importante é identificar os relacionamentos, quais se relacionam com cada entidade. Se o relacionamento entre entidades estiver vinculado somente com Engenheiro, ele deve ser vinculado à entidade engenheiro, e não ao Funcionário. Relacionamentos associados a todas as subclasses devem ser apontados para Funcionário.

Figura 2.11 – Superclasse, subclasse



Fonte: Elmasri, Navathe (2013) p. 166.

Importante

Superclasse e subclasse não se referem ao modelo orientado a objetos, e sim à definição de grupos de relacionamento em banco de dados.

A subclasse nos permite:

- ✗ apresentar os subtipos de uma entidade;
- ✗ mostrar os atributos específicos para cada subtipo;
- ✗ estabelecer os relacionamentos específicos entre subtipos e outros tipos de entidades.

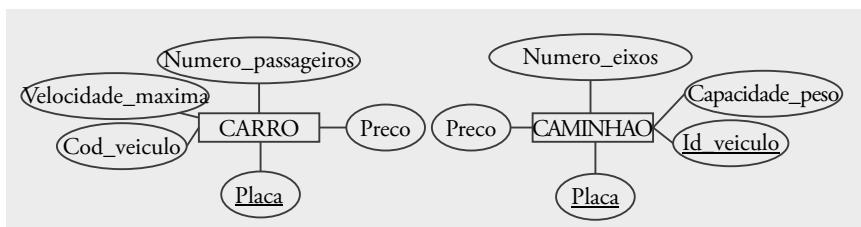
Isso é importante para que se possa decidir qual processo será seguido: generalização ou especialização.

2.6.2 Generalização e especialização

Quando é possível definir as superclasses e subclasses, podemos avaliar qual é a melhor forma de implementar a solução, se será generalizada ou especializada.

Especialização é o processo de definição de um conjunto de classes de um tipo; onde a superclasse será especializada, será implementada cada entidade separadamente. A Figura 2.12 apresenta a especialização da superclasse Veículo, a qual foi separada em carro e caminhão.

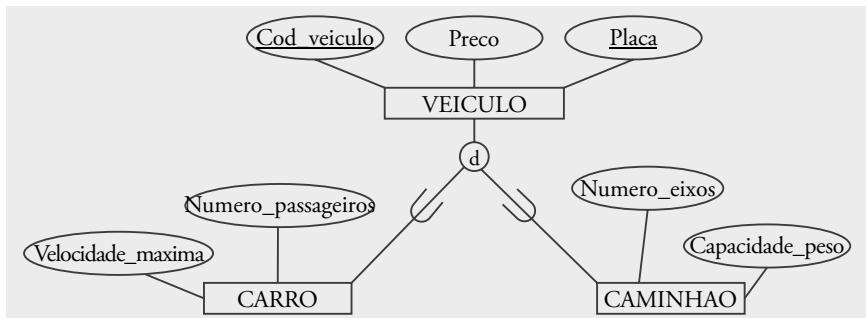
Figura 2.12 – Especialização



Fonte: Elmasri, Navathe (2013) p. 165.

A generalização, mostrada na Figura 2.13, é um processo inverso ao da especialização, no qual são suprimidas as diferenças entre diversos tipos entidades, identificando suas características comuns e generalizando essas entidades em uma superclasse. Nesse caso, criando uma única entidade, na qual será acrescentado o atributo tipo de dado e esse tipo terá o valor que indica se o veículo será um carro ou um caminhão.

Figura 2.13 – Generalização



Fonte: Elmasri, Navathe (2013) p. 165.

Qual é o melhor: generalizar ou especializar? Depende do modelo, da solução. O importante é gerar a superclasse com subclasses e analisar quantos e quais atributos e relacionamentos estão na superclasse ou na subclass, e a partir dessa etapa decidir se o modelo será especializado ou generalizado. Se existirem várias superclasses e subclasses no mesmo modelo, cada uma deve ser analisada individualmente.

Síntese

Este capítulo trouxe o banco de dado do mundo conceitual para o modelo lógico e apresentou as etapas de modelagem de banco de dados. Mostrou-se o que é um modelo de entidade relacionamento, suas estruturas, seus relacionamentos, suas cardinalidades; o que são chaves e quais são seus tipos; e como escolher a chave primária entre as chaves candidatas. Foi apresentado, ainda, como identificar superclasse e subclass para auxiliar na decisão de especializar ou generalizar.

Este capítulo é muito complexo e com muitas informações e somente a prática aprimorará seu conhecimento. Leia a bibliografia e crie vários modelos apresentados nos exercícios.

Da teoria para a prática

Crie um modelo Entidade Relacionamento que apresente os pedidos de produtos de cliente. Um cliente pode comprar vários produtos no mesmo pedido e os produtos podem ser vendidos para vários clientes. Deve-se saber o nome, o tipo e o valor do produto. Do cliente, precisa-se no mínimo do nome, da cidade e do pedido, além da quantidade comprada. Precisa-se, também, armazenar o valor unitário e o valor total.

- ✗ Entidades
- ✗ Relacionamento
- ✗ Chaves primárias
- ✗ Tipos de dados

3

Modelo Relacional e Normalização

O MODELO ENTIDADE relacionamento (MER) e o modelo entidade relacionamento estendido (ERE) são partes do modelo conceitual, por isso é necessário modificar o modelo para definir suas estruturas de modo que fique mais próximo da implementação física.

PARA ISSO, TEMOS a transformação do modelo lógico em modelo relacional, logo após aplicar as regras de normalização.

Objetivos de aprendizagem:

- × compreender o modelo relacional;
- × mapear o modelo entidade relacionamento para o modelo relacional;
- × mostrar normalização;
- × normalizar 1.a, 2.a e 3.a forma normal.

3.1 Modelo relacional

O modelo relacional (MR) define uma entidade como relação, na qual cada conjunto de valores dos atributos pode ser considerado um registro, pois seus valores estão interligados representando um elemento.

Na terminologia do modelo relacional, temos as seguintes nomenclaturas:

- × entidade – tabela ou relação;
- × cada linha de uma tabela – linha, tupla ou registro;
- × cada coluna – atributo ou coluna;
- × cada tipo de dado de cada – domínio.

Importante

Guarde as nomenclaturas, pois elas são usadas em diferentes literaturas, concursos e no Exame Nacional de Desempenho de Estudantes (Enade).

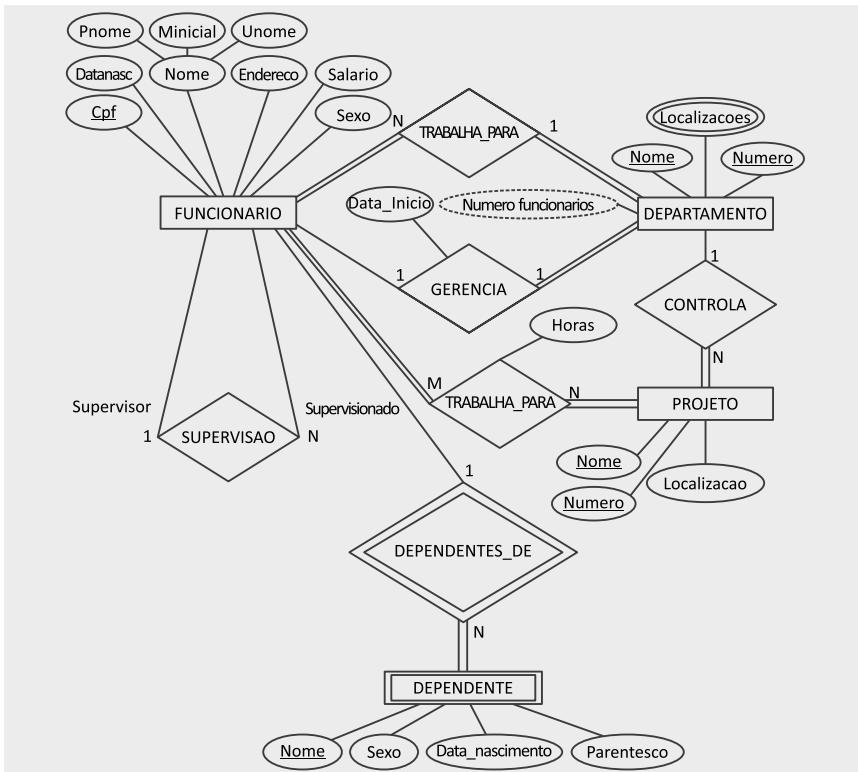
Veremos a aplicação desse modelo no mapeamento do modelo relacional, no qual os conceitos serão explicados passo a passo.

3.2 Mapeamento de modelo entidade relacionamento para modelo relacional

Existem regras para se realizar o correto mapeamento do MER e modelo ERE. Elmasri e Navathe (2013) criaram um algoritmo com um conjuro de

passos para cada situação que precise ser mapeada; para exemplificar, será utilizado o modelo da Figura 3.1.

Figura 3.1 – Modelo entidade relacionamento



Fonte: Elmasri, Navathe (2013). p. 190.

- ✗ **Mapeamento de tipos de entidade regular:** para cada entidade forte E, no MER, crie uma relação R (tabela) e inclua todos os atributos simples e todos os componentes simples de atributos compostos de E. Entre as chaves candidatas, escolha a chave primária (que pode ser simples ou composta). Exemplo: utilizando a Figura 3.1, temos Funcionário, Projeto e Departamento, que são entidades fortes, então devem ser criadas tabelas para cada uma e escolhida a chave primária, como mostrado na Figura 3.2.

Banco de Dados

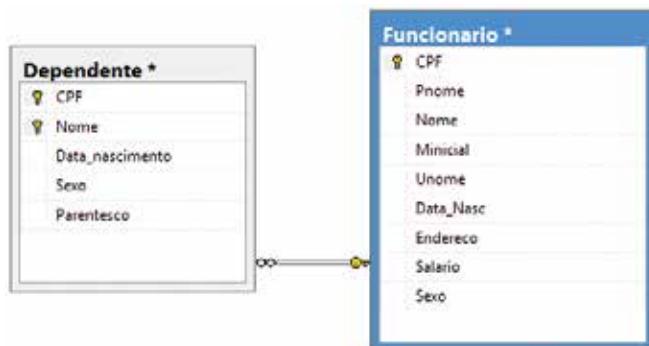
Figura 3.2 – Passo 1: Mapeamento



Fonte: elaborado pelo Autor.

- × **Mapeamento de tipos de entidades fracas:** para cada entidade fraca F, no MER, com a entidade proprietária (Entidade pai), crie uma relação R e inclua todos os atributos simples e todos os componentes simples de atributos compostos de F. Inclua como atributos de chave estrangeira de R as chaves primárias das relações correspondentes às Entidades proprietárias. A chave primária de F é a combinação das chaves estrangeiras e a chave parcial da entidade F, caso exista. Exemplo: temos a entidade fraca Dependente, após aplicar a regra (Figura 3.3).

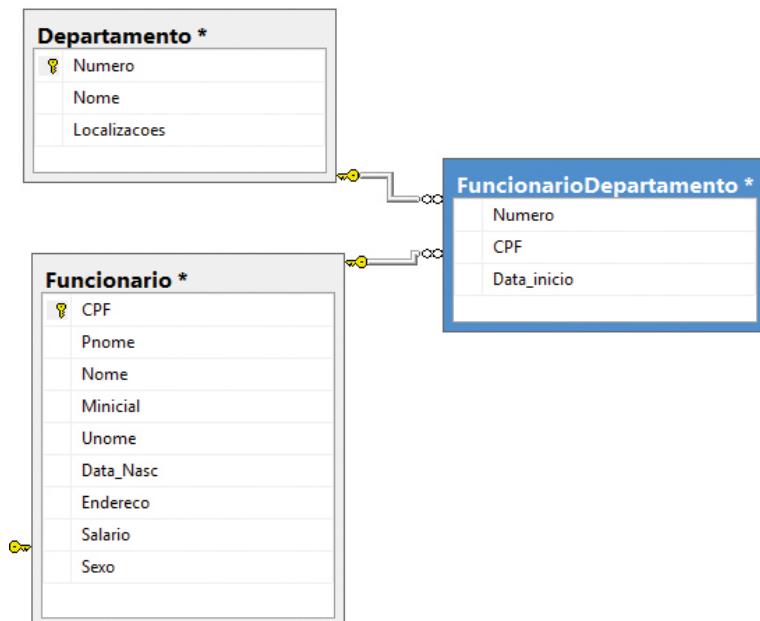
Figura 3.3 – Passo 2



Fonte: elaborado pelo autor.

- ✗ **Mapeamento de relacionamentos binários 1:1:** para esse mapeamento, existem três técnicas possíveis.
- ✗ **Chave estrangeira:** escolha uma das relações S e inclua como chave estrangeira de S a chave primária de T. Inclua todos os atributos simples ou componentes simples dos atributos compostos.
- ✗ **Relação mesclada:** mescle os dois tipos de entidades e o relacionamento em uma única relação R; isso acontece quando ambas as participações são totais.
- ✗ **Relação de referência cruzada:** crie uma terceira relação R com as chaves primárias de T e S. Essa técnica é obrigatória nos relacionamentos N:M e é chamada de tabela associativa, tabela de pesquisa ou relação de relacionamento. A Figura 3.4 mostra esse passo, criando a tabela FuncionarioDepartamento com o relacionamento 1:1.

Figura 3.4 – Passo 3

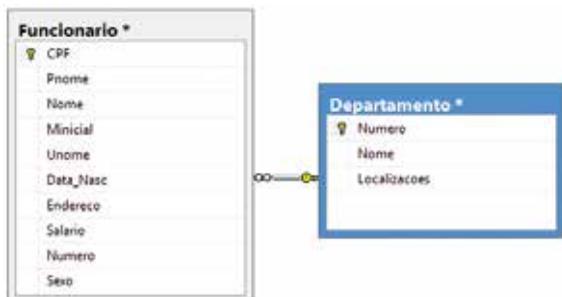


Fonte: elaborado pelo autor.

Banco de Dados

- × **Mapeamento de tipo de relacionamento binário 1:N:** para cada relacionamento 1:N do MER, identifique a relação que fica do lado N do relacionamento e inclua como chave estrangeira em S a chave primária de T. Inclua em S todos os atributos simples e todos os componentes simples de atributos compostos. A Figura 3.5 apresenta o exemplo sendo aplicado em Funcionário e Departamento, onde um departamento tem vários funcionários.

Figura 3.5 – Passo 4



Fonte: Elaborado pelo Autor (2016).

- × **Mapeamento de relacionamento binário M:N:** nesse caso, para cada tipo de relacionamento M:N, crie uma nova relação S e inclua como atributos as chaves primárias das relações das entidades participantes. Inclua em S todos os atributos simples e todos os componentes simples de atributos compostos, caso existam.

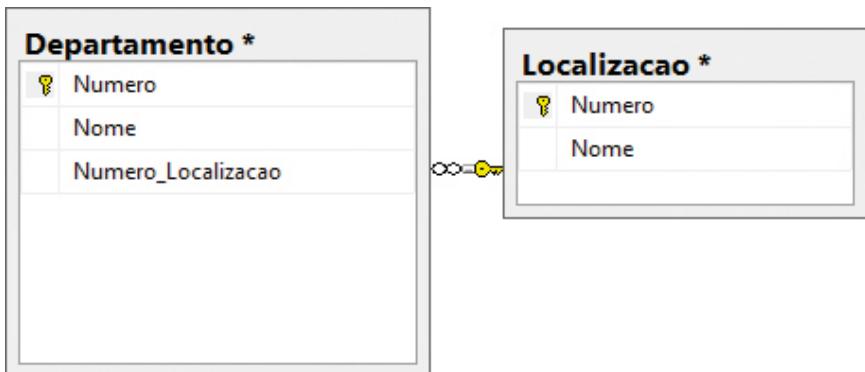
Figura 3.6 – Passo 5



Fonte: elaborado pelo autor.

- × **Mapeamento de atributos multivalorados:** para cada atributo multivalorado A, crie uma Relação R, com o atributo correspondente e a chave primária da entidade proprietária como chave estrangeira de R. Caso existam atributos compostos, os componentes simples serão incluídos. Exemplo: na entidade Departamento existia o atributo multivalorado localizações, o passo foi aplicado, gerando a relação Localização, removendo assim os atributos monovalorados (Figura 3.7).

Figura 3.7 – Passo 6

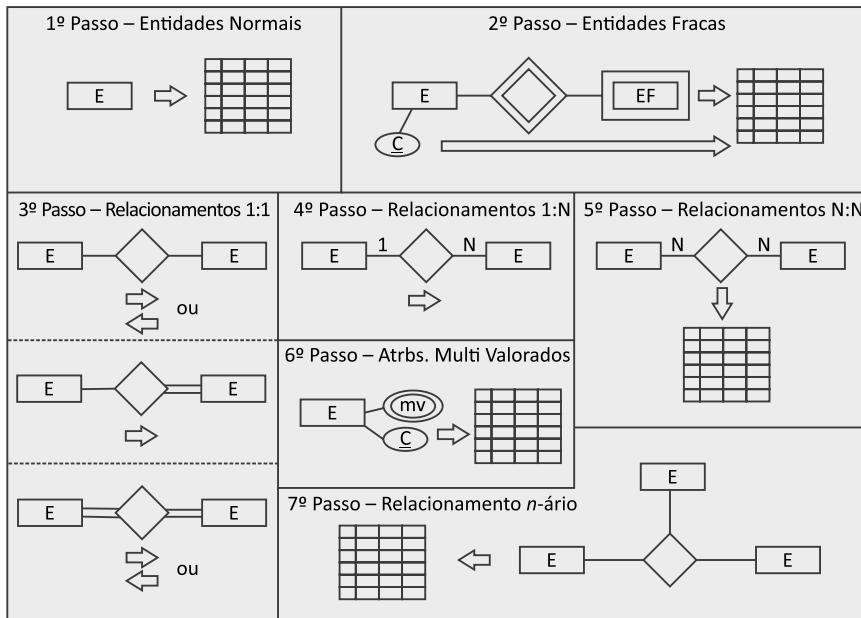


Fonte: elaborado pelo autor.

- × **Mapeamentos de tipo de relacionamento N-ário:** crie uma relação S para representar R e inclua como chave estrangeira em S as chaves primárias das entidades relacionadas. Inclua em S todos os atributos simples e todos os componentes simples de atributos compostos. No modelo da Figura 3.1, não temos como aplicar o passo, mas imagine que existisse o seguinte relacionamento: projeto, funcionário, departamentos, no qual o relacionamento fosse ternário. Nesse caso, seria criada uma tabela associativa para relacionar as três tabelas.

A Figura 3.8 apresenta de forma gráfica os passos apresentados: 1 a 7 equivalem de a) a g). Assim é possível visualizar de forma rápida e resumida cada passo.

Figura 3.8 – Mapeamento para o modelo relacional



Fonte: Meira (1997).

Os passos para o mapeamento do modelo são claros e fáceis de serem executados. O problema surge quando temos modelos muitos grandes e complexos nos quais é necessário observar detalhadamente cada entidade e seus relacionamentos no momento de mapear.

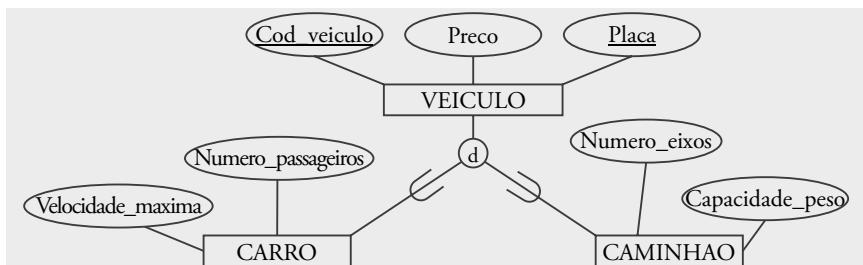
3.3 Mapeamento de modelo entidade relacionamento estendido para modelo relacional

Das formas de mapeamento de generalização e especialização, as mais aplicadas são generalizar completamente ou especializar. Existem outras formas na literatura, mas na prática se tornam inviáveis por questões de performance.

Elmasri e Navathe (2013) apresentam as seguintes soluções:

- ✗ **múltiplas relações** – apenas relações de subclasses (especialização); nesse caso, deve-se criar uma nova relação para cada subclasse. Utilizando como base a Figura 3.9, seriam criadas duas relações: uma para carro e outra para caminhão.
- ✗ **relação única com atributo de tipo** – crie uma única relação com os atributos da superclasse e de todas as subclasses, crie um atributo (chamado de atributo de tipo ou atributos discriminador) que indique a qual subclasse pertence o elemento. Utilizando a Figura 3.9 como exemplo, seria criada apenas uma única relação Veículo com todos os atributos de veículo, carro e caminhão, e mais um atributo tipo que indicaria se esse veículo é um carro ou um caminhão.

Figura 3.9 – Especialização/generalização



Fonte Elmasri, Navathe (2016). p. 165.

O item de especialização/generalização pode ser considerado um oitavo passo no mapeamento do MER para o MR.

3.4 Normalização

Segundo Silberchatz, Korth e Sudarshan (1999), existem determinadas condições que podem interferir na qualidade de um projeto de banco de dados. Essas propriedades indesejáveis podem ser:

- ✗ informações repetidas;
- ✗ inconsistências de dados;
- ✗ incapacidade de representação de algumas informações.

Repetições de dados ocasionam desperdício de espaço e dificuldades de atualização, pois, como os dados são redundantes, todos devem ser atualizados e nem sempre isso ocorre, causando inconsistências dos dados.

A normalização é o processo para resolver essas repetições e inconsistências, decompondo as relações por meio da separação de seus atributos em relações menos complexas (DATE, 2000).

Existem cinco formas normais, mas na prática só normalizamos até a terceira, pois a normalização resolve muitos problemas de inconsistência, dependência e redundância. Em compensação, quanto mais decomposições de relações são feitas, mais junções entre as tabelas são necessárias para a recuperação dos dados, dessa forma, a performance fica reduzida.

Saiba mais

Para maiores informações sobre anomalias 4a e 5a formas normais, consulte a bibliografia do capítulo.



3.5 1.a forma normal – 1FN

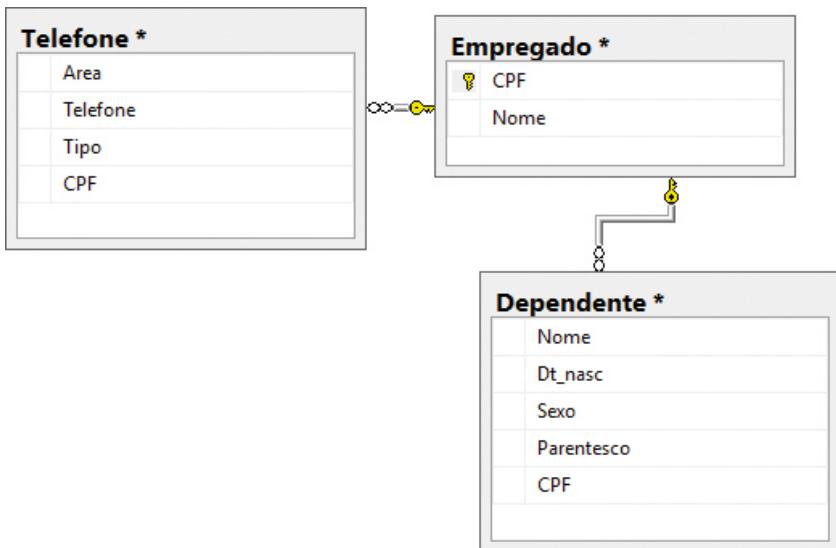
Segundo Date (2000), uma variável de relação (tabela) está na 1FN se e somente se todo valor válido dessa variável de relação contiver exatamente um valor para cada atributo. Isso quer dizer que os atributos devem ser atômicos, não permitindo atributos multivvalorados ou atributos compostos.

Exemplo: apresenta-se a Relação empregado com os atributos CPF, Nome, Dependente e Telefone, onde dependentes e telefone são multivvalorados (representados por chaves), afinal o empregado pode ter vários dependentes, tais como filhos e cônjuge, e vários telefones, como celular, residencial e comercial.

Empregado: CPF, Nome, {Dependente}, {Telefone}

Essa relação não está na primeira forma normal, então deve-se decompor a relação empregado em relações menores, ou seja, criar novas relações para atender dependentes e telefones, conforme a Figura 3.10.

Figura 3.10 – 1FN



Fonte: elaborado pelo autor.

Deve-se proceder dessa maneira em todo o modelo. Depois, deve-se verificar se, além da primeira forma normal, está na segunda forma normal.

3.6 2.a forma normal – 2FN

Uma variável de relação está na 2FN se e somente se está na 1FN e todo o atributo não chave é completamente dependente da chave primária. Essa condição é chamada de dependência total (DATE, 2000). Já a dependência parcial acontece quando uma coluna depende apenas de parte de uma chave primária composta.

Nesse caso, tem-se uma chave primária composta e os atributos devem ter vínculo em todos os atributos da chave primária. Veja o exemplo da Figura 3.11: a chave primária é composta, formada por CodProjeto e Empregado, existe o atributo horas e nomeProjeto. O atributo horas indica quais foram as horas que o empregado trabalhou no projeto, portanto, precisa-se saber qual é o projeto e qual é o empregado; a dependência do atributo Horas é total.

Banco de Dados

O atributo nomeProjeto indica qual é o nome do projeto; independentemente de qual empregado trabalha no projeto; o nome do projeto mudará somente de acordo com o CodProjeto, portanto, nomeProjeto só depende do atributo CodProjeto; dessa forma, a dependência é parcial e não atende à segunda forma normal.

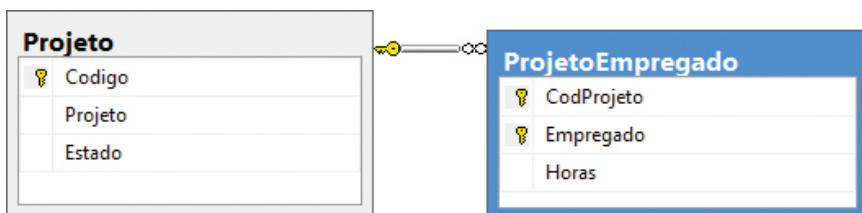
Figura 3.11 – 2FN



Fonte: elaborado pelo autor.

Para corrigir essa condição, deve-se separar a relação inicial em relações menores, criando uma nova relação para cada chave parcial e incluindo os atributos que dependem dessa chave (Figura 3.12).

Figura 3.12 – 2FN



Fonte: elaborado pelo autor.

Observe a Figura 3.12. Para resolver o problema, foi criada uma tabela projeto e adicionado o nome do projeto (atributo projeto). Agora o modelo está de acordo com a 2FN.

3.7 3.a forma normal – 3FN

A 3a Forma Normal apresenta o conceito de dependência transitiva. Uma variável da relação está na 3FN se e somente se estiver na 2FN e os atributos não forem chaves (DATE, 2000):

- ✗ **Mutuamente independentes** – Quando os atributos não participam da chave primária da variável de relação considerada. Isto é, se nenhum deles é funcionalmente dependente de qualquer combinação dos outros. Para que isso aconteça, cada um desses atributos pode ser atualizado independentemente dos demais.
- ✗ **Completamente dependentes da chave primária.**

A Figura 3.13 mostra um exemplo de dependência transitiva. Foi criada uma tabela chamada dependenciaTransitiva para a explicação, na qual temos claramente as informações do empregado e do departamento e a seguinte condição:

- ✗ **CPF { Nome_Empregado }** – Indica a dependência do nome do empregado com o CPF.
- ✗ **CPF { CPF_Gerente_Depto }** – Indica a dependência do CPF do empregado com o gerente do departamento.
- ✗ **CodDept { Nome_Depto, CPF_Gerente_Depto }** – O nome do departamento e do gerente do departamento são dependentes do código do departamento.

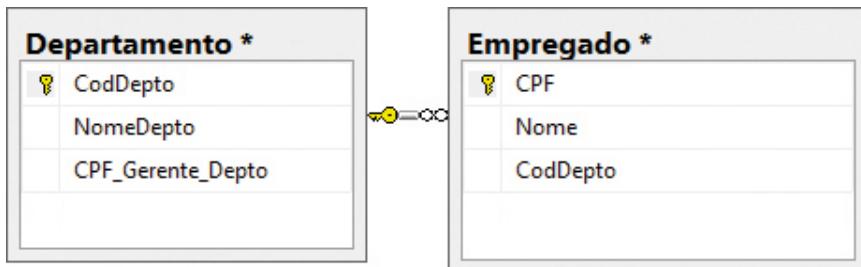
Figura 3.13 – Dependência transitiva

DependenciaTransitiva	
CPF	
NomeEmpregado	
CodDept	
NomeDept	
CPF_Gerente_Depto	

Fonte: elaborado pelo autor

Normalizando a Figura 3.13 para ficar na 3FN, temos o resultado na Figura 3.14.

Figura 3.14 – 3FN



Fonte: elaborado pelo autor.

A relação foi decomposta em duas, Empregado e Departamento, as dependências transitivas foram separadas e as relações atendem à 3FN.

Quando se normaliza, não é necessário fazer passo a passo, com a prática, basta analisar cada tabela e verificar se atendem aos requisitos de cada uma das formas normais.

Saiba mais

Existem inúmeros exemplos na bibliografia; analise e tente normalizar vários deles. A experiência deixará esses conceitos muito mais consolidados e fáceis de reconhecer.

3.8 Dicionário de dados

Para finalizar a documentação, além dos modelos criados, é preciso documentar cada tabela e cada atributo de tabela e definir o domínio (tipo de dado e precisão). Para isso, é necessário criar um documento que contenha:

- ✗ todas as tabelas com a descrição da finalidade de cada uma;

- ✗ todos os atributos com nome, descrição, tipo de dado, precisão.

Muitos sistemas de gerenciamento de banco de dados (SGBDs) já oferecem essa opção nos metadados, no momento da criação das tabelas e dos atributos. Pode-se inserir a descrição de cada um deles e posteriormente gerar relatórios para obter essas informações.

Saiba mais

Precisão indica até que faixa de valor do tipo de dados será utilizada, por exemplo, no MS SQL server temos int, smallint e bigint, todos são inteiros com tamanhos diferentes para suportar faixas diferentes de valores.

Síntese

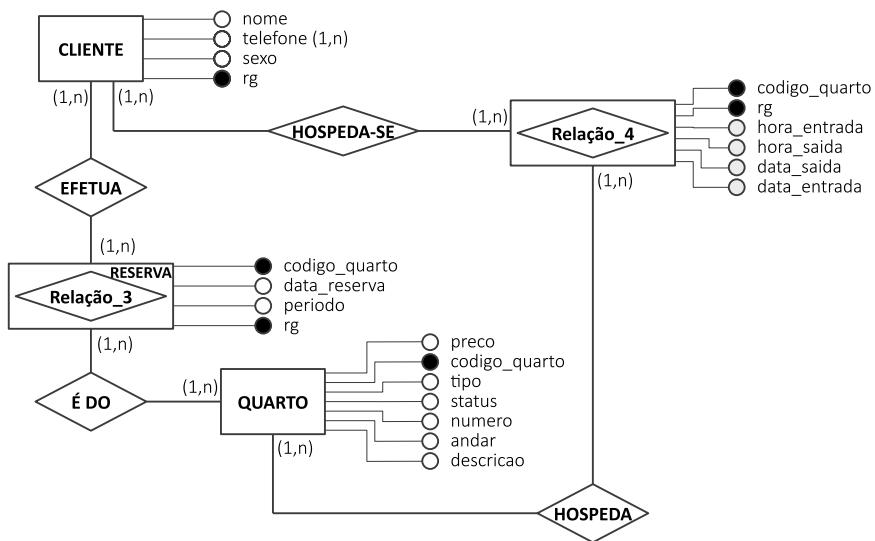
Com o conhecimento adquirido neste capítulo, você deve estar pronto para preparar todo o material necessário e implementar o banco de dados físico. Depois de converter para o modelo relacional, normalizar até a terceira forma normal e preparar a documentação. Agora falta somente escolher o SGBD que será utilizado e fazer a última análise e os ajustes dos tipos de dados no dicionário de dados, afinal, cada SGBD oferece vários tipos diferentes, e então construir o banco de dados.

Da teoria para a prática

Dado o modelo lógico a seguir, crie o modelo relacional que atenda aos critérios:

- ✗ Converta para o modelo relacional.
- ✗ Normalize até 1FN.
- ✗ Normalize até 2FN.
- ✗ Normalize até 3FN.

Banco de Dados



4

SQL

O MODELO ENTIDADE Relacionamento (MER) está pronto, o modelo físico também. A análise do negócio foi feita e a documentação está pronta, os próximos passos são: criar o banco de dados e as estruturas dentro do banco de dados.

NESTE LIVRO, os exemplos apresentados serão demonstrados pelo SGBD Microsoft SQL server, a linguagem será Transact-Sql, que é a extensão da linguagem SQL para este SGBD.

Importante

Os SGBDs têm características detalhadas sobre os serviços e pacotes que estão incluídos em cada versão. Observe cuidadosamente o que é realmente necessário para esta solução, qual a versão do SGBD, se ela é adequada ao sistema operacional utilizado nos servidores e nas máquinas do cliente.

Objetivos de aprendizagem:

- ✗ apresentar a estrutura física de um SGBD;
- ✗ criar banco dados;
- ✗ utilizar *Data Definition Language* (DDL).

4.1 Structured Query Language (SQL)

Existem diversas versões de SQL. Segundo Siberchatz (1999), o SQL foi criado na década de 1970 pela IBM, implementada como parte do projeto do sistema R, então, a linguagem foi evoluindo e hoje inúmeros produtos dão suporte para a linguagem SQL. Em 1986, o *American National Standard Institute* (ANSI) e a *International Standard Organization* (ISO) publicaram os padrões para SQL-86. Uma extensão do padrão SQL, chamado de SQL-89 foi publicada em 1989, logo depois veio a SQL-92. Existem ainda as versões 2003 e 2011.

Saiba mais

Qualquer SGBD que utilize o padrão SQL ANSI pode ser facilmente portável entre diferentes arquiteturas de bancos de dados.

Os SGBDs criam extensões do padrão SQL, no caso do Oracle PL/SQL e da Microsoft SQL Server existe o Transact SQL. Essas extensões oferecem muitos serviços e funções que podem ser utilizadas dentro do comando SQL.

4.2 SQL DDL

DDL (*Data Definition Language*) é parte da linguagem SQL utilizada para definições de dados, utilizando esses comandos pode-se criar dados e suas estruturas.

Um banco de dados é dividido em um ou vários arquivos físicos que são gerenciados pelo sistema operacional, no caso do SQL. Tem-se um ou vários arquivos para dados, como a extensão MDF (*master data file*), para o primeiro arquivo de dados e a extensão NDF (*Secondary Data File*) para os demais arquivos de dados. Existe também um ou vários arquivos de LOG (*Log Data File*), para o armazenamento das transações. Estas extensões de arquivos são meramente informativas. É possível colocar a extensão que se desejar, mas como manter o padrão é extremamente importante para qualquer solução, deve-se utilizar o padrão existente.

Cada SGBD tem sua estrutura própria para arquivos de dados, log, índices, entre outros. O DB2 (banco relacional da IBM) ou o Oracle, tem formatos, variações, nomenclaturas e arquivos físicos bem diferentes entre si.

4.2.1 Create Database

O primeiro passo é criar um banco de dados, lembre-se do padrão de nomenclatura de banco de dados, que pode ser formado por números, letras e caracteres especiais. Mas para evitar problemas de acessibilidade e portabilidade deve-se utilizar letras, números e sublinhado. O nome do banco de dados deve ser o mais próximo a sua finalidade.

No quadro 4.1, tem-se o exemplo do comando básico de *Create database*. O comando completo é bem mais complexo, mas o objetivo deste capítulo é mostrar as principais propriedades e características de criação de banco de dado. Observe que o nome do banco é inserido neste comando. Este é o nome pelo qual o banco de dados será reconhecido dentro do SQL Server. Na sequência, aparece o Name, onde é inserido um nome lógico do arquivo de dados, e outro para o nome lógico do arquivo de log. Estes nomes são referências internas no catálogo do sistema para fazer referência entre o caminho e o nome físico que foi dado para ambos os arquivos.

Banco de Dados

Quadro 4.1 – Create database

```
CREATE DATABASE [NomeDobanco]

ON PRIMARY

( NAME = 'NomeLogicodeArquivodeData',

FILENAME = 'Path\NomeFisico.mdf' , SIZE =
5120KB , FILEGROWTH = 1024KB )

LOG ON

( NAME = 'NomeLogicodeArquivodeLog,

FILENAME = 'Path\NomeFisico_log.ldf' ,
SIZE = 1024KB , FILEGROWTH = 10%)
```

Fonte: Elaborado pela autora.

A demonstração deste comando na prática é apresentada na figura 4.2. O nome do banco é DbFael, seguido do nome lógico dbFael para o arquivo físico DbFael.mdf. Este arquivo vai ter o tamanho inicial de 5120 KB e o crescimento automático de 1024KB. O nome lógico dbFael_lof.ldf faz referência ao DbFael_log.ldf. O tamanho inicial deste arquivo é de 1024KB, com crescimento automático de 10%. O caminho físico no disco do computador para cada um desses arquivos é C:\Program Files\Microsoft SQL Server\MSSQL12.FAEL\MSSQL\DATA\.

Quadro 4.2 – Create Database. Exemplo prático

```

CREATE DATABASE [DbFael]
ON PRIMARY
(
    NAME = N'DbFael', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.FAEL\MSSQL\DATA\DbFael.mdf' , SIZE = 5120KB
, FILEGROWTH = 1024KB )
LOG ON
(
    NAME = N'DbFael_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.FAEL\MSSQL\DATA\DbFael_log.ldf' , SIZE =
1024KB , FILEGROWTH = 10%)
GO

```

Fonte: elaborado pela autora.

É possível ter inúmeros bancos de dados dentro de um SGBD, depende da solução que foi escolhida para isto, geralmente são criadas instâncias por aplicação, desta forma a SGBD fica mais organizador. Os arquivos físicos de cada um destes bancos podem ficar em vários drives e discos diferentes, desde que estejam disponíveis para o gerenciamento do sistema operacional. Em aplicações de grande porte deve-se utilizar *storage* no lugar de discos locais, além da redundância (recuperação de dados em caso de falha de disco), eles são muito mais rápidos.

Figura 4.1 – Arquivos físicos de banco de dados

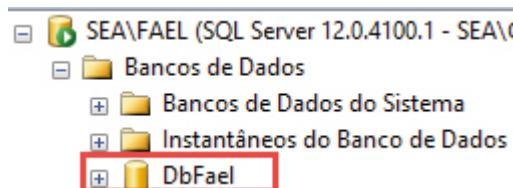
Nome	Data de modificação	Tipo	Tamanho
DbFael.mdf	24/07/16 17:37	SQL Server Database	5.120 KB
DbFael_log.ldf	24/07/16 17:37	SQL Server Database	1.024 KB

Fonte: elaborado pela autora (2016).

Banco de Dados

Os arquivo físicos se encontram no caminho indicado no comando `create database`, a `DbFael.mdf` para o arquivo de dados e `DbFael_log.ldf` para os arquivos de log (figura 4.1). E na figura 4.2, tem-se o banco de dados `DbFael` aparecendo na lista de bancos de dados da ferramenta.

Figura 4.2 – Representação no SSMS



Fonte: elaborado pela autora (2016)

O banco de dados foi criado e o próximo passo é criar as estruturas dentro do banco de dados, que podem ser tabelas, visões, procedimentos armazenados, funções e outros. Estas estruturas variam de acordo com o SGBS usado.

Para eliminar o banco de dados com suas estruturas, utiliza-se o comando `Drop Database`.

Sintaxe:

`Drop database nomeDoBanco`

Este comando apaga inclusive os arquivos físicos do banco.

Para criar cada uma das tabelas, deve-se obedecer o modelo físico gerado, com os mesmos nomes, tipos de dados e precisão documentados.

4.2.2 Tipos de dados

Como representar a tabela apresentada no modelo de dados dentro do banco de dados? Para isso, utiliza-se o comando `create table`. Neste comando é importante saber o tipo de dado para cada atributo, sua precisão e quanto de armazenamento em disco vai ocupar cada um.

Dos principais tipos de dados, tem-se

Tipos de dados numéricos (quadro 4.3) podem armazenar inteiros ou números decimais. Existem também os tipos numeric e decimal, onde pode-se indicar quantos números o tipo de dados vai ter e quantos destes números vão compor a parte inteira e a parte decimal. Declaração: decimal[(p[,s])] e numeric[(p[,s])], onde P é a quantidade total de caracteres e S indica a quanto do total vai ser a parte decimal. Exemplo: Numeric (5,2) são 3 inteiros e 2 casas decimais.

Quadro 4.3 – Tipos de dados numéricos

Tipo de dado	Faixa	Armazenamento
bigint	-2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)	8 Bytes
int	-2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)	4 Bytes
smallint	-2^15 (-32,768) to 2^15-1 (32,767)	2 Bytes
tinyint	0 to 255	1 Byte
Float	- 1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308	Depends on the value of n
Real	- 3.40E + 38 to -1.18E – 38, 0 and 1.18E – 38 to 3.40E + 38	4 Bytes
Money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
smallmoney	- 214,748.3648 to 214,748.3647	4 bytes

Fonte: elaborado pela autora.

4.2.3 Tipos de dado data e hora

Servem para armazenar data, hora e data e hora. Existem tipos timestamp, datetime e smalldate time. Os principais tipos para data e hora estão apresentados no quadro 4.4. Dependendo do tipo de dado, a precisão vai até milissegundos. Portanto, deve-se cuidar com a precisão no momento da pesquisa pela data e hora completa.

Quadro 4.4 – Tipos de dados data e hora

Tipo de dados	Faixa	Armazenamento
Date	0001-01-01 até 9999-12-31 D.C.	3 bytes
Datetime	Janeiro 1, 1753, até dezembro 31, 9999 00:00:00 até 23:59:59.997	8 bytes
Time	00:00:00.0000000 até 23:59:59.9999999	5 bytes

Fonte: elaborado pela autora.

Estes tipos de dados têm tratamento especial, por serem complexos o SGBD oferece funções prontas para trabalhar com data, como:

- ✗ adicionar ou subtrair dias, meses ou anos;
- ✗ subtrair duas datas;
- ✗ recuperar somente o dia, mês ou ano.

4.2.4 Tipos de dados caracteres (strings)

São tipos que armazenam qualquer tipo de caractere, inclusive acentos, espaço em branco e caracteres especiais. Os principais são:

- ✗ char [(n)]: Caracteres não Unicode¹ de comprimento fixo. Onde n define a quantidade de caracteres e deve ser um valor de 1 a 8.000;
- ✗ varchar [(n | max)]: Caracteres não Unicode de tamanho variável. n define a quantidade de caracteres e pode ser um valor de 1 a 8.000. max indica o tamanho máximo de armazenamento, que é $2^{31}-1$ bytes (2 GB).

Em caso de acesso de outra plataforma ou posterior migração do banco de dados para outra arquitetura, pode ser necessária a utilização do padrão Unicode, nestes tipos de dados, basta utilizar nchar e nvarchar na declaração da variável.

¹ Para maiores informações sobre unicode acess os sites <https://tools.ietf.org/html/rfc3629> ou <http://unicode-table.com/en/>

As recomendações da Microsoft para utilização de char e varchar são:

- ✗ char quando os tamanhos dados da coluna forem consistentes.
- ✗ use varchar quando os dados da coluna variarem consideravelmente.
- ✗ Use varchar(max) quando os tamanhos das entradas de dados da coluna variarem consideravelmente e o tamanho puder exceder 8.000 bytes.

Saiba mais

Existem outros tipos de dados e estes variam tanto na precisão quanto na declaração nos diferentes SGBDs, consulte sempre as informações de cada fabricante para ter certeza da utilização do tipo de dado correto, isso também pode variar de acordo com o sistema operacional no qual o banco foi instalado, existem diferenças entre tamanho ocupado por tipo de dado numa arquitetura 32 ou 64 bits.



4.2.5 Create table

Para criar a tabela precisa-se criar os atributos com seus respectivos tipos de dados e também suas restrições.

As restrições ou *constraints* servem para proteger os dados e meta dados de valores, estruturas ou condições inválidas, as principais restrições são:

- ✗ chave primária: *primary key*, indica cada registro como único dentro de uma tabela, o campo que forma a chave primária, não pode ser nulo nem se repetir dentre os registros da tabela.
- ✗ Sintaxe: Create Table nomeTabela
 - (Atributo Tipo de dado Constraint pk_nomeDaConstraint Primary key,
 - Atributo2 Tipo
 -);

Exemplo:

```
Create table Cliente  
    ( CPF Numeric(11) Constraint pk_cliente  
primary key,  
        nome varchar(70),  
        Idade smallint  
    )
```

- ✗ chave estrangeira: *foreign key*, responsável por manter a integridade referencial (apresentada no capítulo de modelagem), é importante para manter a relação entre as tabelas de forma correta (efetiva o relacionamento dentro do banco de dados) onde a chave primária da tabela pai vai para a tabela filho como chave estrangeira.

Create Table nomeTabela

(Atributo tipodedado constraint foreign key fk_attributo references TabelaPai(Chave, primaria)

Exemplo:

```
Create table Dependentes  
    Nome varchar(70),  
    Responsavel numeric(11) Constraining FK_  
    AlunoDependente foreign key (Responsavel)  
    references Aluno(CPF)
```

- ✗ *Null/not null*: Indica se o campo poderá ter valor nulo ou não.

Create Table nomeTabela

(Atributo tipodedado not null)

Exemplo:

```
Create table Cliente
  ( CPF Numeric(11) Constraint pk_cliente
primary key,
  nome varchar(70) Not Null,
  Idade smallint Null)
```

- ✗ *check*: verifica se o valor atribuído ao campo é válido a partir de determinadas regras definidas pelo negócio.

Create Table nomeTabela

(Atributo tipodedado constraint ck_attributo Attibuto Regra-deValidacao)

Exemplo:

```
Create table Cliente
  ( CPF Numeric(11) Constraint pk_cliente
primary key,
  nome varchar(70),
  Idade smallint,
  Sexo char (1) Constraint ck_sexo Check
(sexo in ('M', 'F'))
)
```

- ✗ *default*: caso não seja inserido um valor para o campo no momento da inserção do registro, o banco de dados insere um valor pré-definido para o atributo, como valor padrão.

Create Table nomeTabela

(Atributo tipodedado default valor)

Exemplo:

```
Create table Cliente  
    ( CPF Numeric(11) Constraint pk_cliente  
primary key,  
        nome varchar(70),  
        Idade smallint,  
        Estado char(2) default 'PI')
```

- ✗ *unique*: garante que o valor deste atributo, dentre os vários registros, não pode ser repetido.

Sintaxe: Create Table nomeTabela

(Atributo tipodedado Constraint UK_Attributo Unique)

Exemplo:

```
Create table Cliente  
    ( CPF Numeric(11) Constraint pk_cliente  
primary key  
        nome varchar(70),  
        RG varchar (15) Constraint UK_RG Unique,  
        Idade smallint)
```

Figura 4.3 – Modelo para representar a criação das tabelas



Fonte: elaborado pela autora.

Como podem existir inúmeros bancos de dados em uma instância de banco de dados, deve-se sempre indicar em qual banco está sendo manuseado no momento. Para isso utiliza-se o comando Use Nomedobanco.

Quadro 4.5 – Aluno

```

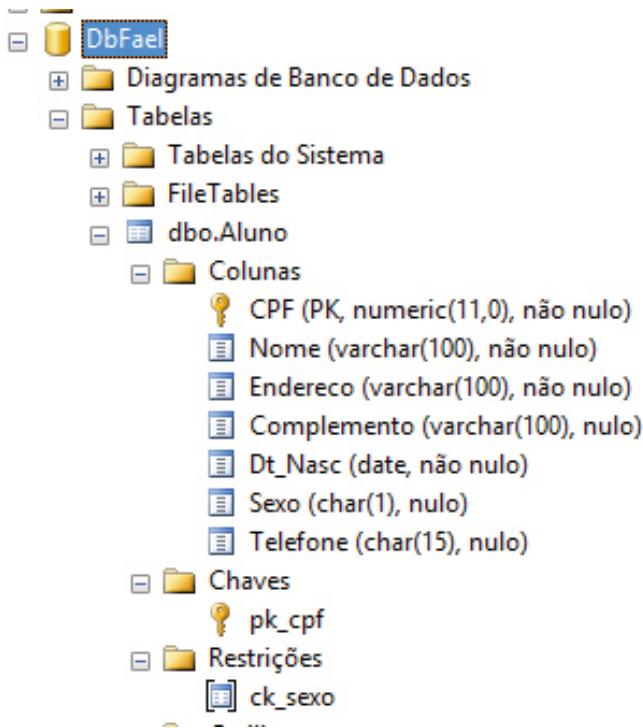
Use DbFael
Go
Create table Aluno
  (CPF Numeric (11) not null constraint pk_cpf primary key,
  Nome varchar(100) not null,
  Endereco varchar(100) not null,
  Complemento varchar(100),
  Dt_Nasc date not null,
  Sexo char(1) constraint ck_sexo check
  (sexo in ('F', 'M')) ,
  Telefone char (15)
  )
  
```

Fonte: elaborado pela autora.

Banco de Dados

O quadro 4.5 mostra a criação da tabela Aluno. Inicialmente se indica qual banco de dados deve ser utilizado, neste caso DbFael. Dentro do comando create table indica-se o nome da tabela e, depois, cada um dos atributos com suas respectivas restrições. Para as constraints de primary key, foreign key, check e unique recomenda-se fortemente utilizar padrão de nomenclatura, isso facilita na manutenção futura.

Figura 4.4 – Após a execução do comando



Fonte: elaborado pela autora.

Ao executar o comando exibido no quadro 4.5, o banco de dados DbFael terá sua primeira tabela como demonstrado na figura 4.4. A interface gráfica do SQL Server permite que seja visualizado as chaves e restrições existentes em cada tabela. Estas informações se encontram no catálogo do sistema de cada SGBD.

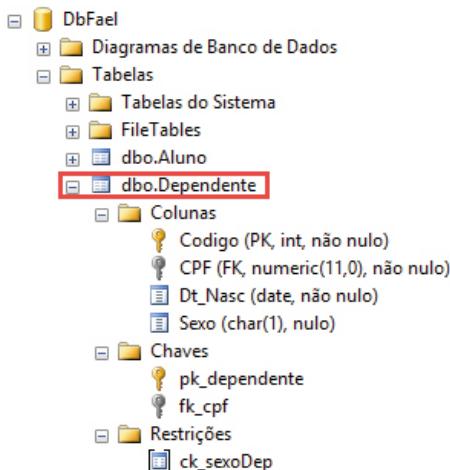
Quadro 4.6 – Create table dependente

```
Create table Dependente
(Codigo int identity (1,1),
CPF Numeric (11) not null constraint fk_cpf
foreign key references Aluno(CPF),
Dt_Nasc date not null,
Sexo char(1) constraint ck_sexoDep check
(sexo in ('F', 'M')) ,)
```

Fonte: elaborado pela autora (2016).

Ao executar o comando do quadro 4.6, cria-se a tabela dependente. Observe que, como é uma entidade fraca, foi criada o atributo código do tipo auto-incremento, chamado de *identity*, com valor inicial igual a 1 e incrementando de 1 em 1. Nesta tabela, o CPF é chave estrangeira, que faz referência a tabela pai Aluno, o nome do campo não precisa ser o mesmo, mas o tipo de dados e precisão devem ser exatamente o mesmo entre chaves primárias e estrangeiras.

Figura 4.5 – Tabela Dependente criada no banco de dados



Fonte: elaborado pela autora.

No banco de dados DbFael existem duas tabelas (figura 4.5), a tabela Aluno e a tabela Dependente.

Procede-se da mesma forma para criar as demais tabelas do modelo, utilizando o padrão de nomenclatura para as restrições. As tabelas pai devem ser criadas antes das tabelas filhos, para que o banco de dados consiga reconhecer a integridade referencial. Outra possibilidade é criar todas as tabelas primeiro e usando o comando *Alter table*, pode-se adicionar ou remover atributos e restrições.

Para eliminar uma tabela com todo o seu conteúdo, utiliza-se o comando Drop Table. Sintaxe: Drop table nome da Tabela.

Após executar este comando com sucesso, tanto os dados são removidos do banco quanto as informações da tabela no catálogo do sistema. A tabela será removida se o usuário tiver privilégio suficiente para executar o comando e nenhuma regra ou integridade referencial seja afetada.

Saiba mais

Os nomes dos atributos devem ser únicos por tabela e o nome das restrições devem ser únicos no banco de dados.

4.3 Álgebra relacional

A álgebra relacional é um conjunto de operações tendo uma ou duas tabelas (relações) e produzindo como resultado uma diferente relação (Silberschatz, 1999). Estas operações são utilizadas para selecionar linhas e colunas de tabelas para especificar uma consulta em um determinado banco de dados. O resultado de cada operação é uma nova tabela que pode ser utilizada para novas operações.

As operações fundamentais, também conhecidas como primárias, atuam sobre uma única relação, são elas:

- ✗ select
- ✗ project
- ✗ rename

Os SGBDs aplicam muitos comandos de álgebra relacional nas suas consultas, mas os operadores de SQL DML oferecem muitos recursos que os existentes na álgebra relacional

4.3.1 Select

A operação Select é utilizada para selecionar um conjunto de linhas (tuplas) de uma relação, sendo que estas tuplas devem satisfazer uma condição de seleção. A sintaxe uma operação select é:

$$\sigma_{<\text{condição de seleção}>} (<\text{nome da relação}>)$$

- ✗ A letra grega minúscula sigma – é utilizada para representar a operação de seleção;
- ✗ Condição de seleção. É uma expressão lógica (booleana) aplicada sobre os atributos da relação;
- ✗ Nome da relação. É o nome da tabela sobre a qual será aplicada a operação select.

Exemplo: este exemplo recupera todas as colunas da relação (tabela) Aluno desde que o Sexo seja feminino, e armazena o resultado na Consulta.

Consulta = – Sexo = ‘F’ (Aluno)

Os operadores que podem ser aplicados na operação Select são:

- ✗ relacionais: <, >, <>, <=, >=, =;
- ✗ lógicos: and, or, not.

4.3.2 Project

A operação Project seleciona um conjunto específico de colunas de uma relação. A sintaxe da operação project é:

$\Pi <\text{lista de atributos}> (<\text{name da relação}>)$

- ✗ A letra grega Π representa a operação project;
- ✗ Lista de atributos indica quais os atributos que o usuário deseja selecionar;

Banco de Dados

- × Nome da relação. É a tabela sobre a qual a operação project será aplicada.

Exemplo: este exemplo mostra as colunas nome e data de nascimento da relação dependentes. Todos os registros são mostrados.

Consulta = $\prod \text{Nome, Dt._Nasc}^{(\text{DEPENDENTES})}$

4.3.3 Operação sequencializada

As operações project e select podem ser utilizadas de forma combinada, permitindo que apenas determinadas colunas de determinadas tuplas possam ser selecionadas.

A sintaxe de uma operação sequencializada é

$\prod <\text{lista de atributos}> (\sigma <\text{condição de seleção}> (<\text{name da relação}>))$

Exemplo: neste caso será apresentado o nome e a data de nascimento, da tabela Aluno, cujo sexo seja masculino.

Consulta = $\prod \text{nome, dt_nasc} (\sigma \text{ sexo} = 'M'^{(\text{Aluno})})$

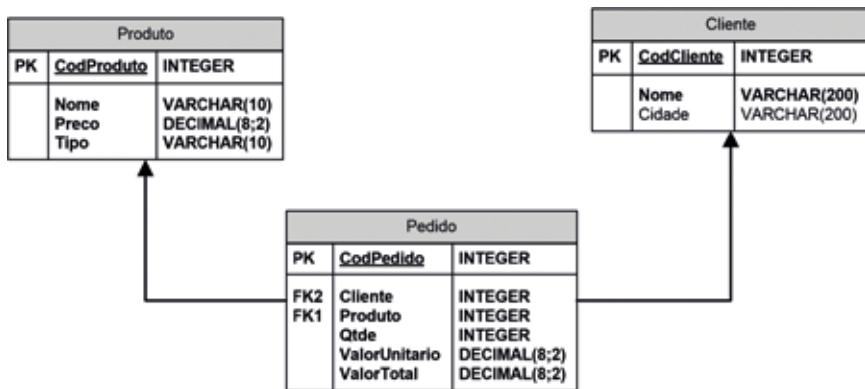
A álgebra relacional é muito importante para a criação de consultas em SQL. Os bancos de dados relacionais utilizam muitos conceitos da matemática em sua criação e execução.

Síntese

Este capítulo apresentou como funciona a estrutura de armazenamento de um bando de dados com seus arquivos físicos e suas referências lógicas que são armazenados nos metadados do banco. Como criar o banco de dados a partir de um modelo físico de dados, criando tabelas e restrições que existem para proteger os dados e a integridade.

Da teoria para a prática

A partir do modelo apresentado:



- × crie o Banco de Dados;
- × crie a tabela Produtos com suas restrições;
- × crie a tabela Cliente com suas restrições;
- × crie a tabela Pedidos com suas restrições.

5

SQL DML

DML (LINGUAGEM DE manipulação de dados) é uma linguagem utilizada para recuperar e trabalhar com dados em SQL. Use estas instruções para inserir, atualizar, alterar, consultar ou apagar dados de um banco de dados. Neste capítulo, apresentaremos os recursos de SQL DML para especificar consultas simples.

Importante

A tarefas de DML podem ser definidas pelos acrônimos CRUD (Create, read, update, delete) ou CERA (criar, excluir, recuperar e apagar)

Objetivos de aprendizagem:

- ✗ apresentar DML;
- ✗ mostrar a utilização dos comandos básicos de DML;
- ✗ mostrar exemplos práticos de DML.

5.1 Select

Um banco de dados é formado por uma coleção de tabelas (tuplas ou relação) cada uma designada por um único nome. Cada tabela possui várias colunas (atributos). Para acessar o conteúdo destes atributos (registro ou linha), utilizamos a linguagem de manipulação de dados, especificamente o comando Select. Esta instrução é utilizada para recuperar os dados do banco, ela não é a mesma operação Select da álgebra relacional, existem muitas opções para o comando Select que não são apresentados na álgebra relacional.

A estrutura básica de uma expressão Select em SQL consiste em três cláusulas (Siberschatz, 1999):

- ✗ select – é operação de projeção (Π) na álgebra relacional. É usada para mostrar os atributos desejados no resultado de uma consulta.
- ✗ from – é operação do produto cartesiano (X). Associa as tabelas que serão pesquisadas durante a evolução de uma expressão.
- ✗ where – é a operação de seleção (σ). Consiste em um predicado envolvendo atributos da tabela que aparece no from.

Uma consulta simples em SQL, figura 5.1, onde C são as colunas referenciadas, T são as tabelas e P é o predicado.

Figura 5.1 – Select

```

        Select C1, C2, ..., Cn
        From T1, T2, ..., Tn
        Where P
    
```

Fonte: elaborado pelo autor.

Em álgebra relacional:

$$\Pi C1, C2, \dots, Cn (\sigma_P (T1 \times T2 \times \dots \times Tn))$$

Ao ser executado, o SQL forma um produto cartesiano (combinação de todas as linhas de todas as tabelas envolvidas) entre as tabelas existentes no From, executa uma seleção usando o predicado do Where e projeta os resultados sobre os atributos da cláusula Select. Obtendo como resultado da consulta outra tabela, chamada de *result set* ou conjunto resultado.

Exemplo: Encontre os nomes de todos os alunos da tabela Aluno (figura 5.2).

Figura 5.2 – Select

The screenshot shows the Microsoft SQL Server Management Studio interface. In the top-left pane, there is a query editor window containing the following SQL code:

```

        Select nome
        From Aluno
    
```

A red box highlights the word "Query" in the title bar of the query editor. Below the editor, the status bar shows "100 %". At the bottom of the screen, there is a toolbar with two buttons: "Resultados" and "Mensagens".

The main area displays the results of the query in a table titled "Resultados". The table has one column labeled "nome" and five rows of data:

	nome
1	José da Silva
2	Maria da Silva
3	Joana Souza
4	João Pereira
5	Marco Antônio Alvarenga

A red box highlights the word "Result Set" in a callout bubble pointing to the table above. The table itself is also enclosed in a red rounded rectangle.

Fonte: elaborado pelo autor (2016)

Neste exemplo (figura 5.2), a coluna é denominada nome e a tabela aluno.

Saiba mais

Se na query (consulta), não existir cláusula Where, o predicado P é verdadeiro, ou seja, não haverá nenhuma restrição; então todos os registros serão apresentados.

O padrão do SQL é apresentar duplicidade dos registros nas tabelas, pois a eliminação da duplicidade demanda muito tempo. Isso não significa quebra de chave primária, pois no exemplo da figura 5.2, apresenta-se somente o campo nome, e podem existir homônimos. Para forçar a eliminação de duplicidade utiliza-se a cláusula **distinct** logo após o Select (figura 5.3), neste caso, se existir repetição do registro (utiliza-se todos os campos apresentados no Select para esta avaliação), esta será suprimida, e somente será apresentado um único registro para cada repetição, independente, da quantidade de repetições existentes. Select nome from Aluno e Select ALL nome from aluno, trazem o mesmo resultado, pois o default é ALL, e esta cláusula pode ser omitida sem nenhum prejuízo.

Figura 5.3 – Distinct

```
:Select distinct  
nome  
From Aluno
```

Fonte: elaborado pelo autor (2016)

Quando é necessário trazer todos os atributos, deve-se utilizar o asterisco “*” (figura 5.4) para indicar “todos os atributos”. Esta opção só deve ser utilizada se todos os atributos forem realmente utilizados, pois trazer dados que não são necessários tornam a consulta mais pesada, e geralmente temos vários usuários acessando o banco ao mesmo tempo, quanto mais dados não

necessários utilizarem os recursos do SGBD, E/S e rede, mas tempo e recursos são desperdiçados, quando poderiam estar sendo utilizados para trazer os resultados importantes para o usuário.

Figura 5.4 – Asterisco

```

Select *
From Aluno
Ou
Select Aluno.*
From Aluno

```

The screenshot shows a SQL interface with two queries. The first query selects all columns from the 'Aluno' table. The second query selects the table itself. Below the queries is a results grid showing five rows of student data:

CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Idade	Estado	Salário
11	José da Silva	Rua dos Guarapes, 37, Bairro Centro	NULL	1963-12-07	M	NULL	RJ	RJ	1200,00
22	Maria da Silva	Rua do Chapéu, 455	Ap 12	1989-02-02	F	NULL	Manaus	AM	2600,00
33	Joana Souza	Rua das Memórias, 137	Ap 43	1997-12-05	F	NULL	Salvador	BA	NULL
44	João Pereira	Rua Rui Barbosa, 24	NULL	1990-10-15	M	NULL	Porto Alegre	RS	1590,00
55	Marco Antônio Alves	Av XV de Novembro, 1733	NULL	1974-11-11	M	NULL	Brasília	DF	3700,00

Fonte: elaborado pelo autor.

É permitido aplicar expressões aritméticas e fórmulas entre números e colunas no Select. Pode-se utilizar as expressões aritméticas (+, -, * e /).

Por exemplo (figura 5.5), caso se deseje um relatório para calcular um aumento de 5% no salário de cada aluno, o Select e o resultado para a query acima é demonstrada na figura 5.6, observe que no quadro azul aparece a fórmula para o cálculo, que segue as mesmas regras matemáticas de ordem para aplicação dos operadores. Outro detalhe importante é que quando a coluna está na cláusula Select, o nome do atributo é apresentado no *result set*, mas quando se aplica uma fórmula ou operador matemático sobre a coluna, o nome do atributo não aparece mais no cabeçalho do resultado. Select não altera nenhum valor do atributo, as fórmulas aplicadas afetam somente o resultado apresentado, e não o conteúdo da tabela original.

Figura 5.5 – Operadores matemáticos

The screenshot shows a SQL query window titled "SQLQuery4.sql - SEA...SEA\Giulliana (52)*". The query is:

```
SELECT Nome, (Salario * 5 / 100) + Salario
FROM Aluno
```

The results grid has two columns: "Nome" and "Salario". The header "Nome" is highlighted with a red box. The first row shows "José da Silva" and "1260,00". The second row shows "Maria da Silva" and "2730,00". The third row shows "Joana Souza" and "NULL". The fourth row shows "João Pereira" and "1669,50". The fifth row shows "Marco Antônio Alvarenga" and "3885,00".

Existe uma forma de melhorar a apresentação e alterar o conteúdo mostrado no cabeçalho, são chamados Aliases e sua demonstração será apresentada mais adiante.

Fonte: elaborado pelo autor (2016)

5.2 Where

Da mesma maneira que muitas vezes recuperar todos os atributos não é necessário, também pode-se aplicar uma condição (predicado) sobre as colunas, mas neste caso para restringir a quantidade de registros no resultado de acordo com o conteúdo da coluna.

O where é utilizado para aplicar a condição lógica e retornar os registros caso estejam de acordo com a condição. Para selecionar o nome de todos os alunos do sexo feminino, a query é demonstrada na figura 5.6.

Figura 5.6 – Where

The screenshot shows a SQL query window with a red callout pointing to the "Where" clause. The query is:

```
SELECT Nome, Salario
FROM Aluno
WHERE Sexo = 'F'
```

The results grid has two columns: "Nome" and "Salário". The first row shows "Maria da Silva" and "2600,00". The second row shows "Joana Souza" and "NULL".

Fonte: elaborado pelo autor.

Veja que a cláusula Where não interfere na cláusula Select, neste exemplo foi utilizado a projeção para recuperar o nome e o salário, em conjunto com o predicado Sexo = 'F' no Where.

O SQL usa os operadores lógicos AND, OR e NOT (figura 5.7), e os operadores de comparação <, <=, >, >=, = e <> (figura 5.8) na cláusula Where.

Figura 5.7 – Exemplo de OR

```
Select Nome, Salario
From Aluno
Where Estado = 'AM' or Estado = 'PR'
```

Nome	Salario
Maria da Silva	2600.00

Figura 5.8 – Exemplo de And, >=

```
Select Nome, Sexo
From Aluno
Where Estado = 'AM' and Salario >= 500
```

Nome	Sexo
Maria da Silva	F

Fonte: elaborado pelo autor.

Não trazer nenhum resultado, figura 5.9, também conhecido como resultar um conjunto vazio, não significa erro ou falha, apenas mostra que não existe nenhum registro que atenda a condição estipulada. Um *result set* pode ser vazio, unitário ou com vários elementos.

Figura 5.9 – Resultado é um conjunto vazio

```
Select Nome, Salario
From Aluno
Where Estado = 'MA' or Estado = 'PR'
```

Nome	Salario
------	---------

Fonte: elaborado pelo autor.

O operador de comparação Between, figura 5.10, pode ser usado para simplificar a cláusula Where quando especificar um valor que seja maior que ou igual a valor e menor que ou igual a outro valor.

Figura 5.10 – Between

The screenshot shows a SQL query window with the following code:

```
Select Nome, Salario
From Aluno
Where Salario between 500 and 2000
```

Below the code is a results grid:

	Nome	Salario
1	José da Silva	1200,00
2	João Pereira	1590,00

Figura 5.11 – Not Between

The screenshot shows a SQL query window with the following code:

```
Select Nome, Salario
From Aluno
Where Salario not between 500 and 2000
```

Below the code is a results grid:

	Nome	Salario
1	Maria da Silva	2600,00
2	Marco Antônio Alvarenga	3700,00

Fonte: elaboradas pelo autor.

Da mesma maneira, o not between, figura 5.11, especifica os valores que estão fora da faixa apresentada, neste caso são apresentados os valores que sejam menores e maiores à faixa.

Importante

Quando o resultado não for aquele esperado, verifique a cláusula Where, os exemplos citados são bem simples, mas na prática um Where pode ser muito complexo.

5.3 Aliases

O *result set* tem um cabeçalho onde geralmente fica o nome de cada coluna, ou sem valor no caso de aplicar fórmulas ou operadores matemáticos. Em determinadas situações, existe a necessidade de nomear ou renomear o nome de uma tabela ou coluna em uma *query*:

- × seja para eliminar ambiguidades (quando existem referências a atributos com o mesmo nome, em tabelas diferentes ou no caso de acesso recursivo a tabela);

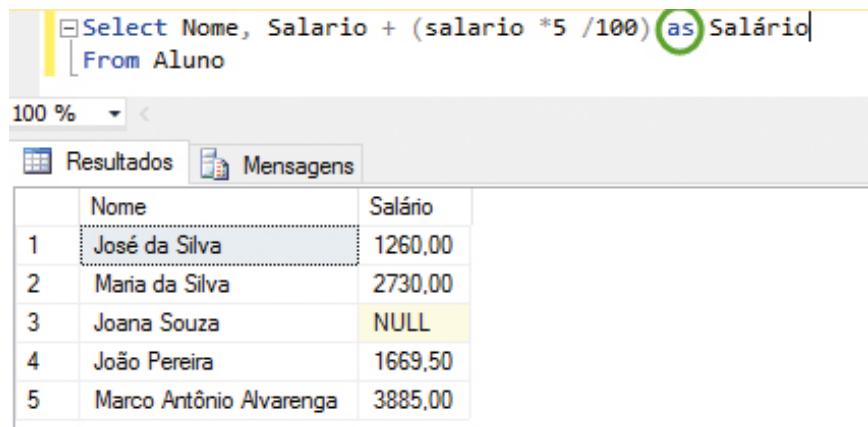
- ✗ para deixar o cabeçalho padronizado;
- ✗ para se utilizar esta consulta como “tabela” de entrada para outra consulta.

Pode-se renomear tanto atributos quanto tabelas. Para renomear tabelas ou atributos utilizamos a cláusula AS, que pode aparecer tanto no Select quanto no From. Pode ser representado de duas formas, com o AS entre o nome antigo e o novo ou com espaço em branco substituindo o AS:

- ✗ nome_antigo **as** nome_novo (figura 12)
- ✗ nome_antigo nome_novo (figura 13)

Exemplo:

Figura 5.12 – Aliases



```
SELECT Nome, Salario + (salario * 5 /100) as Salário
FROM Aluno
```

	Nome	Salário
1	José da Silva	1260,00
2	Maria da Silva	2730,00
3	Joana Souza	NULL
4	João Pereira	1669,50
5	Marco Antônio Alvarenga	3885,00

Fonte: elaborado pelo autor.

O alias está sendo utilizado para mudar o campo que estaria sem cabeçalho na aplicação de fórmula, para o valor de Salário (observe que está acentuado). O resultado é mesmo utilizando AS, como na figura 12 ou utilizando espaço como na figura 13. A utilização de um ou outro depende de dois fatores, se existir um padrão, utilize-o, senão, a escolha fica a critério do DBA, que utiliza o que for mais natural, mas após a escolha de usar ou não o AS, todas as *queries* devem seguir o mesmo padrão.

Banco de Dados

Figura 5.13 – Alias com espaço

A screenshot of the SQL Server Management Studio interface. The query window contains the following code:

```
Select Nome, Salario + (salario *5 /100) Salário
From Aluno
```

The result set shows five rows of data:

	Nome	Salário
1	José da Silva	1260,00
2	Maria da Silva	2730,00
3	Joana Souza	NULL
4	João Pereira	1669,50
5	Marco Antônio Alvarenga	3885,00

A red arrow points from the text "Alias com espaço" to the alias "Salário" in the query.

Fonte: elaborado pelo autor.

O alias pode ser usado para nomear tabelas. Em SQL avançado, será muito útil, principalmente para evitar ambiguidades. A figura 5.14 mostra que além de nomear a tabela, o alias pode ser utilizado para especificar qual o campo se refere a cada tabela.

Figura 5.14 – Alias em tabela

Two screenshots of the SQL Server Management Studio interface, side-by-side, demonstrating the use of aliases in a table.

The left screenshot shows the following query:

```
Select Nome, A.Dt_Nasc
From Aluno as A
```

The right screenshot shows the same query with a different alias usage:

```
Select Nome, A.Dt_Nasc
From Aluno A
```

Both screenshots show the same result set:

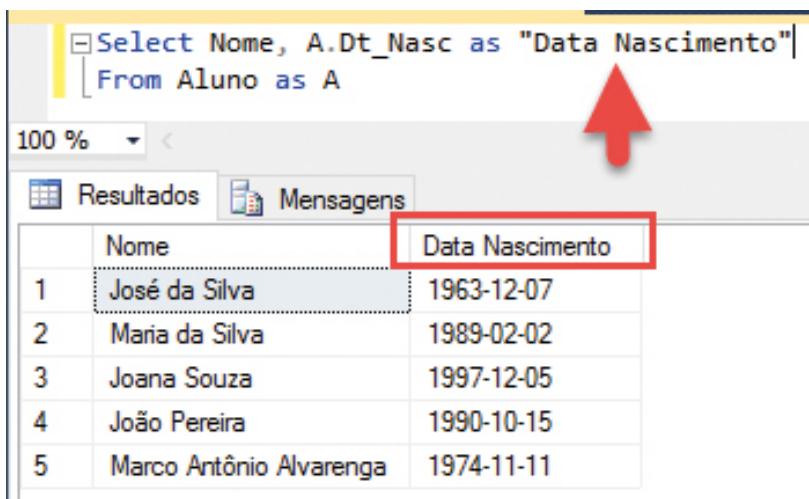
	Nome	Dt_Nasc
1	José da Silva	1963-12-07
2	Maria da Silva	1989-02-02
3	Joana Souza	1997-12-05
4	João Pereira	1990-10-15
5	Marco Antônio Alvarenga	1974-11-11

A red arrow points from the text "Alias em tabela" to the alias "A.Dt_Nasc" in the first query.

Fonte: elaboradas pelo autor.

Também pode-se utilizar alias no atributo e na tabela como na figura 5.15, e quando for necessário um alias formado por mais de um nome, como Data Nascimento, ele deve ficar entre aspas.

Figura 5.15 – Alias em atributo e tabela



```
Select Nome, A.Dt_Nasc as "Data Nascimento"
From Aluno as A
```

	Nome	Data Nascimento
1	José da Silva	1963-12-07
2	Maria da Silva	1989-02-02
3	Joana Souza	1997-12-05
4	João Pereira	1990-10-15
5	Marco Antônio Alvarenga	1974-11-11

Fonte: elaborado pelo autor.

O alias não altera qualquer valor na tabela original, ele só muda o *result set*. Nas recomendações de nome de atributos e tabelas não se recomenda caracteres especiais, caso seja necessária acentuação ou semelhante deve-se utilizar os aliases.

5.4 Ordenação

No mundo dos negócios, muitas vezes se faz necessária a utilizada de dados ordenados, para que sejam exibidos em relatórios. O SQL oferece a cláusula *order by*, que permite que os registros de um resultado de uma consulta apareçam em uma determinada ordem.

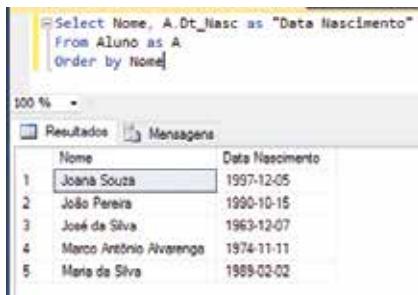
A sintaxe da ordenação é

Order by C1, C2, ... CN

Banco de Dados

Para cada atributo do `order by`, pode-se definir se a ordenação vai ser ascendente [asc] figura 15, que é *default* e pode ser omitida na *query*, ou se vai ser descendente [desc], que deve ser declarada explicitamente, figura 17.

Figura 5.16 – Order by



A screenshot of a database query results window. The query is:

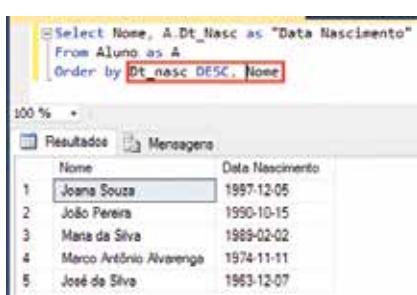
```
Select Nome, A.Dt_Nasc as "Data Nascimento"
From Aluno as A
Order by Nome
```

The results show five rows of data:

	Nome	Data Nascimento
1	Joana Souza	1997-12-05
2	João Pereira	1990-10-15
3	José da Silva	1963-12-07
4	Marco Antônio Alverenga	1974-11-11
5	Maria da Silva	1989-02-02

Fonte: elaborado pelo autor.

Figura 5.17 – Order by desc



A screenshot of a database query results window. The query is:

```
Select Nome, A.Dt_Nasc as "Data Nascimento"
From Aluno as A
Order by Dt_nasc DESC, Nome
```

The results show five rows of data:

	Nome	Data Nascimento
1	Joana Souza	1997-12-05
2	João Pereira	1990-10-15
3	Maria da Silva	1989-02-02
4	Marco Antônio Alverenga	1974-11-11
5	José da Silva	1963-12-07

Fonte: elaborado pelo autor.

A ordenação varia de acordo com o tipo de dado, e segue a ordem do primeiro campo, se existir duplicidade e ordena o registro pelo segundo campo e assim sucessivamente.

Saiba mais

Utilize ordenação com cuidado, muitos recursos são utilizados, principalmente em banco de dados com tabelas muitos grandes.

A ordenação não altera a posição dos registros no banco de dados, ele ordena somente o *result set*.

5.5 Valores nulos

É importante relembrar que valores nulos, indicam que a coluna não teve nenhum valor atribuído, serve para indicar a ausência de informação.

Pode-se utilizar a palavra `is null`, figura 5.18, para testar a existência de valores nulos, enquanto se utiliza `is not null`, figura 19, para verificar a ausência de valores nulos.

Figura 5.18 – Is Null

```
Select Nome, Salario
From Aluno
Where Salario is Null
```

The screenshot shows a SQL query in the top pane and its results in the bottom pane. The results table has columns 'Nome' and 'Salario'. A row for 'Joana Souza' has 'NULL' in the 'Salario' column, which is highlighted with a yellow background.

	Nome	Salario
1	Joana Souza	NULL

Figura 5.19 – Is not null

```
Select Nome, Salario
From Aluno
Where Salario is not Null
```

The screenshot shows a SQL query in the top pane and its results in the bottom pane. The results table has columns 'Nome' and 'Salario'. All rows have non-null values in the 'Salario' column.

	Nome	Salario
1	José da Silva	1200,00
2	Maria da Silva	2600,00
3	João Pereira	1590,00
4	Marco Antônio Alvarenga	3700,00

Fonte: elaboradas pelo autor.

Se qualquer um dos valores de entrada for nulo, o resultado de uma expressão aritmética (+, -, * e /) é nula, representado pela figura 5.20.

Figura 5.20 – Cálculo com valores nulos

```
Select Nome, Salario + 500 as Salário
From Aluno
```

The screenshot shows a SQL query in the top pane and its results in the bottom pane. The results table has columns 'Nome' and 'Salário'. The row for 'Joana Souza' has 'NULL' in the 'Salário' column, indicated by a red arrow pointing to it.

	Nome	Salário
1	José da Silva	1700,00
2	Maria da Silva	3100,00
3	Joana Souza	NULL
4	João Pereira	2090,00
5	Marco Antônio Alvarenga	4200,00

Fonte: Elaborado pelo autor.

Saiba mais

Nulo quer dizer ausência de valor, ou seja, espaço em branco, não é nulo, é caractere; zero não é nulo é um valor numérico válido.

5.6 Insert

Uma das funções importantes do SQL é o Insert, este comando valida a restrições de valores de cada atributo no momento em que os dados estão sendo inseridos dentro do banco de dados. As restrições (*constraint*) de Check, Primary Key, Foreign Key, Unique, Not Null, Default são validada neste momento.

A sintaxe do Insert é

Insert into tabela

Values

(V1, V2, ..., Vn)

Neste caso, os valores V devem ter o mesmo tipo de dado especificado no Create Table e também os valores a serem inseridos devem estar na mesma ordem em que as colunas foram criadas no momento em que a tabela foi criada.

Ou

Insert into tabela

(C1, C2, ..., Cn)

Values

(V1, V2, ..., Vn)

Neste caso, os valores V não precisam estar na mesma ordem de criação das colunas no Create Table, e sim, na ordem que foram definidas as colunas C na cláusula insert. Esta forma é obrigatória quando:

- ✗ existir campo auto incremento na tabela;

- ✗ quando não é necessário inserir todos os atributos, desde que sejam Not Null;
- ✗ quando se deseja utilizar o valor default que foi atribuído na criação do atributo na forma de restrição; observe a integridade referencial, ao inserir dados em uma tabela “filho” tenha certeza que existe chave estrangeira correspondente a chave primária da tabela “pai”, senão o registro não será inserido.

De acordo com o tipo de dados, no Insert as seguintes regras devem ser observadas na atribuição de valores

- ✗ ao inserir um valor nulo, utilize NULL, maiúsculo ou minúsculo, sem aspas;
- ✗ ao inserir datas, caracteres e *strings* utilize sempre aspas simples ('');
- ✗ para valores numéricos, utilize o ponto (.) para casa decimal;
- ✗ em datas, verifique o formato. No Brasil é DD/MM/AAAA, mas o formato americano é MM-DD-AAAA. Verifique sempre o formato para evitar erros como: 03/07/2016 que pode ser armazenado como 07/03/2016. Neste caso, a data que se deseja é 3 de julho e pode ser armazenado como 7 de março.

Utilizando a Tabela Aluno para exemplificar, os erros que são gerados pelas constraints, são erros totalmente válidos para garantir as regras de um SGBD, dados inválidos não devem ser inseridos no banco de dados.

Na figura 5.21, temos um exemplo de violação de PK (primary key). Este erro demonstra que já existe um CPF do aluno com valor 11.

Figura 5.21 – Falha de PK

```

=INSERT INTO Aluno
VALUES
(11, 'Teste de Insert', 'Este é o Endereço', NULL, '23-JUN-1995', 'F',
null, 'Belo Horizonte', 'MG', 0.0)
GO
% +
Mensagens
Mensagem 2627, Nível 14, Estado 1, Linha 2
Violacion de restriccion PRIMARY KEY 'pk_cpf'. No es posible insertar la clave duplicada en el objeto 'dbo.Aluno'.
El valor de la clave duplicada es {11}.
La instruccion fue finalizada.

```

Fonte: elaborado pelo autor

Banco de Dados

Na figura 5.22, existe a tentativa de inserção de valor nulo no atributo endereço que não pode ser nulo, tendo a restrição Not Null.

Figura 5.22- Erro de Not Null

The screenshot shows an SQL query window with the following code:

```
INSERT INTO Aluno
VALUES
(66, 'Teste de Insert', NULL, NULL, '23-JUN-1995', 'F',
Null, 'Belo Horizonte', 'MG', 0.0)
GO
```

Below the query window is a message window titled "Mensagens" containing the following error message:

Mensagem 515, Nível 16, Estado 2, Linha 2
Não é possível inserir o valor NULL na coluna 'Endereco', tabela 'DbFael.dbo.Aluno';
a coluna não permite nulos. Falha em INSERT.
A instrução foi finalizada.

Fonte: elaborado pelo autor.

Na figura 5.23, existe a tentativa de se inserir o valor A para sexo que só permite M ou F.

Figura 5.23 – Erro de Check

The screenshot shows an SQL query window with the following code:

```
INSERT INTO Aluno
VALUES
(66, 'Teste de Insert', 'Este é o endereco', NULL, '23-JUN-1995', 'A',
Null, 'Belo Horizonte', 'MG', 0.0)
GO
```

Below the query window is a message window titled "Mensagens" containing the following error message:

Mensagem 547, Nível 16, Estado 0, Linha 2
A instrução INSERT conflictou com a restrição do CHECK "ck_sexo". O conflito ocorreu no banco de dados
| "DbFael", tabela "dbo.Aluno", column "Sexo".
A instrução foi finalizada.

Fonte: elaborado pelo autor.

E, finalmente, depois de todas as correções feitas (figura 24), o comando Insert foi executado com sucesso.

Figura 5.24 – Insert completo

```

INSERT INTO Aluno
VALUES
(66, 'Teste de Insert', 'Este é o endereço', NULL, '23-JUN-1995', 'F',
Null, 'Belo Horizonte', 'MG', 0.0)
GO
  
```

100 % < Mensagens

(1 linha(s) afetadas)

Fonte: elaborado pelo autor.

Na figura 5.25, temos o exemplo de Insert utilizando a segunda forma, onde as colunas são apresentadas, indicando a posição de cada valor para cada atributo. Facilita bastante em tabelas muito grandes, onde não é necessário inserir todos os campos, que podem ser preenchidos depois.

Figura 5.25 – Segunda opção de insert com declaração de colunas

```

INSERT INTO Aluno
(CPF, Nome, Endereco, Dt_Nasc, Sexo, Cidade, Estado)
VALUES
(77, 'Teste de Insert 2', 'Rua das Flores', '15-MAI-1981', 'M',
'Maceió', 'AL')
  
```

100 % < Mensagens

(1 linha(s) afetadas)

Fonte: elaborado pelo autor.

Sempre são inseridos registros inteiros de cada vez, mesmo que nem todos os atributos recebam valor.

5.7 Delete

Assim como precisamos inserir dados no banco, existem momentos que apagar os dados é muito necessário. Este comando remove tuplas (registros)

de uma tabela. Ele pode remover vários registros por vez ou somente um ou grupos de registros que atendem a uma cláusula Where.

Outro fator importante é a integridade referencial, caso a intenção seja deletar dados que tem correspondentes em outras tabelas isso não será permitido para que não existam registros órfãos.

Importante

Sem cláusula where, o delete vai apagar todos os registros da tabela, às vezes é necessário, mas tome cuidado se é realmente isso que se quer fazer, o mesmo vale para o Where que pode não ser tão restritiva quanto se deseja.

Sintaxe do Delete

Delete Tabela

Onde todos os registros serão apagados.

Ou

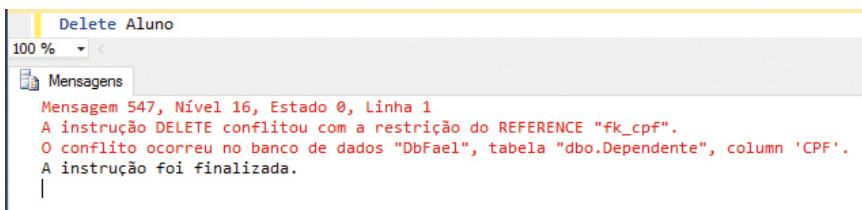
Delete Tabela

Where Condição

Onde somente os registros que atenderem o predicado Where serão apagados, se nenhum registro atender a condição, nada será apagado.

A figura 5.26 apresenta um erro de integridade referencial, na tentativa de deletar todos os alunos, inclusive aqueles que tem dependente. Neste caso, nada foi apagado.

Figura 5.26 – Delete – Integridade Referencial



Fonte: elaborado pelo autor.

O delete apresentado na figura 27, mostra que não ocorreu nenhum erro de restrição, portanto, o registro cujo o CPF (PK) que era 33 foi apagado.

Figura 5.27 – Delete

```
Delete Aluno
Where CPF = 33
```

100 % Mensagens

(1 linha(s) afetadas)

Fonte: elaborado pelo autor.

Existe outra opção para apagar todos os registros de uma tabela, conhecido como comando truncate. Ele elimina todos os registros de uma tabela, mas precisa de privilégios especiais para ser executado, além de permissão de leitura e escrita. O truncate é mais rápido que o delete, pois não registra as exclusões das linhas individuais na Log do banco de dados.

A sintaxe é

Truncate Table nome da tabela

A figura 5.28 apresenta um exemplo da eliminação de todas a linhas de uma tabela utilizando o comando Truncate.

Figura 5.28 – Truncate Table

```
Truncate table Dependente
```

100 % Mensagens

Comando(s) concluído(s) com êxito.

Fonte: elaborado pelo autor.

Após a efetivação da deleção dos registros, estes podem ser recuperados somente com a restauração de um backup.

5.8 Update

Com as atualizações da vida real, como mudança de endereço, telefone, aumento de preços, de salários, promoções, entre outros, passa a ser necessária a atualização do banco de dados para que ele corresponda com a realidade.

Para isso, o SQL tem o comando Update. este comando é utilizado para modificar valores de um ou mais atributos de um ou mais registros selecionados.

A sintaxe para o Update é

Update Tabela

Set C1 = V1,

C2= V2,

....

CN = VN

Where Condição

O Where pode ser omitido caso se deseje alterar todas as linhas da tabela.

Os valores a serem atribuídos para as colunas podem ser valores fixos, fórmulas e até mesmo cálculos de outras colunas.

A figura 5.29 apresenta a execução de um Update com Where, que atuaiza dois específicos registros.

Figura 5.29 – Update

The screenshot shows a MySQL command-line interface window. The command entered is:

```
Update Aluno  
Set Salario = 1500  
Where CPF = 66 or CPF = 77
```

(2 linha(s) afetadas)

Fonte: elaborado pelo autor.

A figura 5.30 apresenta um cálculo sobre o atributo salário, o qual recebe um aumento de R\$ 500,00, e o atributo complemento também é atualizado para um registro específico.

Figura 5.30 – Update com cálculo de atributos

```
Update Aluno
Set Salario = Salario + 500,
    Complemento = 'Ap 37'
Where CPF = 66
```

100 % <

Mensagens

(1 linha(s) afetadas)

Fonte: elaborado pelo autor.

Pode-se alterar qualquer campo desde que não quebre nenhuma restrição. Alguns SGBDs não permitem a atualização da chave primária, só os demais campos. Se no Update houver alguma tentativa de quebra de restrições, uma mensagem de erros será apresentada e o Update não será executado.

5.9 Operações com strings

Strings são conjuntos de caracteres que podem conter letras, números e caracteres especiais. Como é um tipo de dado complexo, pode-se utilizar e tratar a pesquisa de diferentes maneiras.

Umas das opções mais usadas são as verificações de coincidências, usando o operador like.

O like pode ser combinado com alguns caracteres especiais para melhorar as opções de pesquisa, eles podem ser:

- ✗ porcentagem (%) – compara qualquer *substring*;
- ✗ sublinhado (_) – compara qualquer caractere;
- ✗ barra (/) – utilizando como caractere de escape para comparar caracteres especiais.

Observe que:

- ✗ “La%” – strings que comecem com “La”;
- ✗ “%aro%” – strings que possua uma substring “aro”;
- ✗ “___” – strings com somente 3 caracteres;
- ✗ “__%” – strings com pelo menos 3 caracteres.

Dependendo da configuração de linguagem, podemos ter algumas diferenças na pesquisa, No SQL server existe um item chamado *Collation*, que define alguns parâmetros para consultas com strings. Seguem alguns deles:

- ✗ AS – *Accent Sensitive*, palavras acentuadas são diferentes de palavras não acentuadas, ou seja, leva em consideração a acentuação, a é diferente de A;
- ✗ AI – *Accent Insensitive*, desconsidera a acentuação, por exemplo, a = á = à;
- ✗ CS – *Case Sensitive*: Maiúsculas não são iguais a minúsculas.
- ✗ CI – *Case Insensitive*: Maiúsculas e minúsculas são iguais

Comparando o resultado da querie nas figuras 5.31 e 5.32, pode-se observar o funcionamento do parâmetro *AS (Accent Sensitive)*, na figura 31 ele trouxe um registro, mas na figura 5.32, o result set foi vazio, pois á é diferente de ao.

Figura 5.31 – Case Insensitive

The screenshot shows a SQL query window with the following code:

```
Select nome
From Aluno
Where Nome like '%ão%'
```

The results window below shows one row:

nome
João Pereira

Figura 5.32 – Case Sensitive

The screenshot shows a SQL query window with the following code:

```
Select nome
From Aluno
Where Nome like '%ao%'
```

The results window below is empty:

nome

Fonte: elaboradas pelo autor (2016)

Existem inúmeras formas de tratar *strings*, cada linguagem oferece forma diferentes de pesquisa e resultado. A vantagem que isso oferece uma flexibilidade imensa permitindo a utilização do SGBD sem se restringir a linguagem utilizada.

Saiba mais

Acesse a internet e procure por SQL Server Transact-SQL DML Reference – Microsoft, você vai ver o manual completo com todas as opções que cada comando apresentado aqui oferecem.

Síntese

Neste capítulo foi apresentada uma introdução ao SQL DML, começando com o Select e sua combinação com álgebra relacional, cada uma das cláusulas do Select foi apresentado, o From e o Where e quais operadores lógicos e de comparação podem ser usados.

Depois, foi apresentada a ordenação de registros para o *result set*, e a forma de tratamento de valores nulos.

O comando Insert veio a seguir, com exemplos práticos e diferentes formas de adicionar dados da tabela, seguido pelo Delete para apagar registros e o Update atualizar dados no banco.

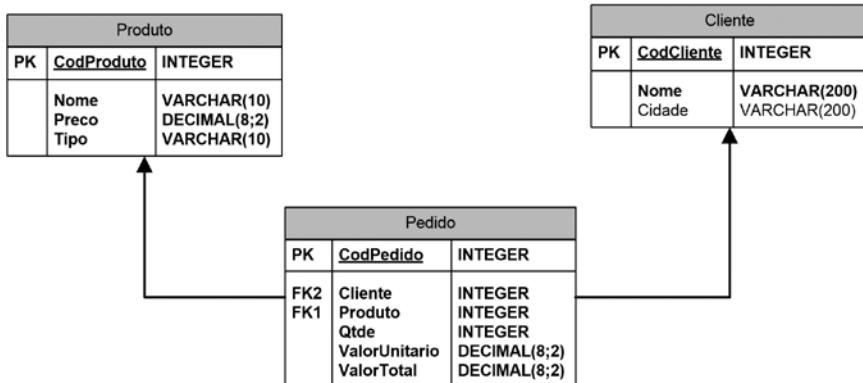
Finalizamos o capítulo com formas de comparação de string e detalhes.

Saiba mais

Acesse a internet e procure mais no site da Microsoft sobre Collation. Isso vai melhorar bastante o entendimento sobre as diversas linguagens como o chinês, árabe, russo, entre outras são tratados pelo SGBD.

Atividades

Utilizando o modelo a seguir, faça o que se pede.



- ✗ Selecione o nome e o preço de todos os produtos.
- ✗ Selecione o nome de todos os clientes que moram em Brasília.
- ✗ Selecione o nome e o preço dos produtos que custam menos de R\$ 100, ordenado pelo preço, começando pelo menor valor.
- ✗ Selecione todos os clientes que tenha a substring ANA no nome.

6

SQL Avançado

Os **COMANDOS ESSENCIAIS** do SQL já foram apresentados, mas ainda existem outros que auxiliam na recuperação de dados. Esses comandos facilitam muito o resultado para relatórios e pesquisas mais complexas, como agregação e recuperação de dados de várias tabelas.

Objetivos de aprendizagem:

- × mostrar funções agregadas;
- × apresentar junções;
- × subconsultas aninhadas;
- × uniões;
- × visões.

6.1 Funções agregadas

Funções agregadas são utilizadas quando se precisa aplicar cálculos sobre grupos de registros. Estas funções recebem os valores como entrada e retornam um valor simples.

As funções agregadas do SQL são:

- × contador (*count*) – count – os dados podem ser não numéricos;
- × média (*average*) – avg – somente com números;
- × mínimo (*minimum*) – min – os dados podem ser não numéricos;
- × máximo (*maximum*) – max – os dados podem ser não numéricos;
- × total (*total*) – sum – somente números.

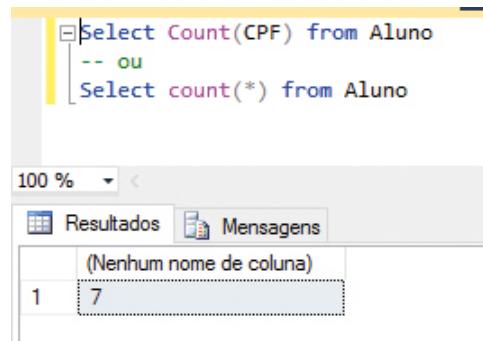
Sintaxe:

```
Select Count(atributo) from Tabela  
Select Min(atributo) from Tabela  
Select Avg(atributo) from Tabela  
Select Max(atributo) from Tabela  
Select Sum(atributo) from Tabela
```

O contador é usado quando se precisa calcular quantos registros atendem determinada condição, representado na figura 6.1. Pode-se utilizar tanto

o nome do atributo quando o *, neste caso tem-se a quantidade total de todos os registros existentes na tabela Aluno.

Figura 6.1 – Count



```

Select Count(CPF) from Aluno
-- ou
Select count(*) from Aluno

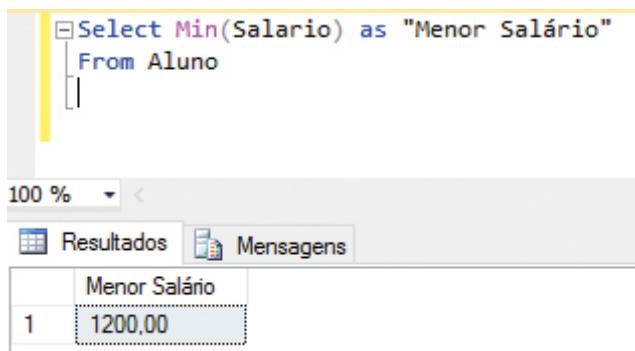
```

Resultados	
(Nenhum nome de coluna)	
1	7

Fonte: elaborada pelo autor.

Mínimo e máximo são usados quando se precisa saber quais os valores mínimos (figura 6.2) e máximo (figura 6.3), respectivamente, de um atributo.

Figura 6.2 – Min



```

Select Min(Salario) as "Menor Salário"
From Aluno

```

Resultados	
	Menor Salário
1	1200,00

Fonte: elaborada pelo autor.

Pode-se utilizar *aliases* e outros comando no SQL. Veja que as figuras 6.2 e 6.3 mostram o Menor Salário e o Maior Salário como nome da coluna no *result set*.

Banco de Dados

Figura 6.3 – Max

The screenshot shows a SQL query window in SSMS. The query is:

```
Select Max(Salario) as "Maior Salário"
From Aluno
```

The results pane shows one row with the value 3700,00.

Maior Salário
1 3700,00

Fonte: elaborada pelo autor.

A soma totaliza os valores de uma coluna que atenda a condição. No exemplo da figura 6.4, é totalizado o salário dos alunos, mas somente aqueles que são do sexo feminino.

Figura 6.4 – Sum

The screenshot shows a SQL query window in SSMS. The query is:

```
Select Sum(Salario) as "Total"
From Aluno
Where Sexo = 'F'
```

The results pane shows one row with the value 4600,00.

Total
1 4600,00

Fonte: elaborada pelo autor.

A média calcula a média aritmética dos valores de um determinado atributo. Na figura 6.5, temos a média salarial dos alunos do sexo masculino.

Figura 6.5 – Avg

```

Select Avg(Salario) as "Média"
From Aluno
Where Sexo = 'M'

```

The screenshot shows a SQL query in the top pane. The results pane below shows a single row with one column labeled 'Média' containing the value '1997,50'. The 'Resultados' tab is selected.

	Média
1	1997,50

Fonte: elaborada pelo autor.

Pode-se utilizar as funções agregadas em uma *query*. Tudo que é apresentado na linguagem SQL pode ser combinado para se obter consultas melhores. Na figura 6.6, tem-se a utilização da média, máximo e contador na mesma consulta.

Figura 6.6 – Funções agregadas

```

Select Avg(Salario) as 'Média',
       Max(salario) as 'Maior',
       Count(*) as Quantidade
From Aluno

```

The screenshot shows a SQL query in the top pane. The results pane below shows a single row with three columns: 'Média' (2098,3333), 'Maior' (3700,00), and 'Quantidade' (7). The 'Resultados' tab is selected.

	Média	Maior	Quantidade
1	2098,3333	3700,00	7

Fonte: elaborada pelo autor.

Importante

As repetições são importantes no cálculo da média, mas se precisa eliminar as repetições, pode-se utilizar o *distinct* na expressão agregada.

SQL não permite o uso do *distinct* com *count (*)*, somente *distinct count* (atributo).

6.2 Group By e Having

A cláusula *Group By* é usada quando existe a necessidade de se agrupar registros ou aplicar uma função agregada a um grupo de conjunto de registros.

Sintaxe (dados entre colchetes são opcionais):

```
Select Atributos ou funções agregadas  
From Tabela  
[Where condição]  
Group By Atributos a serem agrupados  
[Having condição]
```

O(s) atributo(s) em uma cláusula *Group Gy* é(são) utilizado(s) para formar grupos, ou seja, os registros com os mesmos valores de todos os atributos da cláusula *Group By* são colocados em um grupo. O *Group By* executa uma ordenação dos dados para formar os grupos.

O exemplo apresentado na figura 6.7 atende a seguinte solicitação: selecione todos os alunos por sexo. Isto significa agrupar os dados por sexo e contar quantos alunos existem para cada sexo. São 3 mulheres e 4 homens.

Figura 6.7 – Group By

The screenshot shows a SQL query window titled "SQLQuery1.sql - SEA...SEA\Giulliana (52)*". The query is:

```
SQLQuery1.sql - SEA...SEA\Giulliana (52)*
Select Sexo, count(*) Quantidade
From Aluno
Group by Sexo
```

The results tab is selected, displaying the following data:

	Sexo	Quantidade
1	F	3
2	M	4

Fonte: elaborada pelo autor.

A cláusula *having* é utilizada quando se necessita aplicar condições sobre os dados agrupados. O *having* é como uma cláusula *where*, a diferença é que *having* se aplica somente a grupos (ou seja, as linhas grupos), enquanto a cláusula *where* se aplica a linhas individuais. Se uma cláusula *where* e *having* aparecem na mesma consulta, a condição que aparece no *where* é aplicada primeiro.

A figura 6.8 representa a utilização da cláusula *having*. Neste caso, a solicitação é: traga a média salarial por sexo, mas somente daqueles que tiverem a média salarial maior que R\$ 2.000,00. Assim, o agrupamento é feito por sexo, depois a média salarial é calculada, com o *having* verificando quais são as condições do grupo maiores que R\$ 2.000,00 e, finalmente, esses dados são apresentados. A figura 6.9 comprova que a média salarial masculina é abaixo de R\$ 2.000,00, exatamente por isso o grupo não aparece no resultado final.

Figura 6.8 – Having

The screenshot shows a SQL query window titled "SQLQuery1.sql - SEA...SEA\Giulliana (52)*". The query is:

```
SQLQuery1.sql - SEA...SEA\Giulliana (52)*
Select sexo, avg(salario) 'Media salarial'
From Aluno
Group by Sexo
Having avg (salario) > 2000
```

The results tab is selected, displaying the following data:

	sexo	Media salarial
1	F	2300,00

Fonte: elaborada pelo autor.

Figura 6.9 – Média por sexo

```
100 % < >
Resultados Mensagens
Sexo (Nenhum nome de coluna)
1 F 2300,00
2 M 1997,50
```

Fonte: elaborada pelo autor.

Group By, *Having* e funções de agregação são muito importantes na geração de relatórios, uma vez que já devolvem o *result set* com o cálculo e o agrupamento prontos, sem a necessidade de trafegar todos os dados pela rede para calcular dentro do programa.

6.3 Junções

Junções são utilizadas quando precisamos acessar dados de outras tabelas que tem relacionamento entre si. As junções podem ser:

- ✗ *inner join* (interna);
- ✗ *left outer join* (externa à esquerda);
- ✗ *right outer join* (externa à direita);
- ✗ *outer Full join* (externa à direita e à esquerda combinadas);
- ✗ *cross join* (cruzada).

6.3.1 Junção interna – *Inner Join*

A Junção Interna (*Inner Join*) conecta duas ou mais tabelas e retorna apenas as linhas que satisfazem a condição de junção. Esta condição de junção

é feita pelos mesmos atributos que compõe a integridade referencial, ou seja, chave primária comparada com chave estrangeira.

Sintaxe:

```
Select atributos
From tabela1
Inner Join tabela2
On Condição de Junção
[ Inner Join tabelaN
On Condição de Junção]
```

Neste caso, a condição é *primary key* da tabela1 = *foreign key* da tabela2.

No exemplo da figura 6.10, tem-se a junção da tabela Aluno e da tabela Dependente. A integridade referencial é formada pelo atributo CPF, chave primária em Aluno e chave estrangeira em Dependente. Como os atributos CPF e Nome tem o mesmo nome, é necessária a criação de *alias* para indicar de qual tabela vem cada atributo.

Figura 6.10 – *Inner Join*

The screenshot shows the SQL query window with the following code:

```
Select A.Nome, D.nome
From Aluno A
Inner Join
Dependente D
On A.CPF = D.CPF
Order by D.nome
```

A red callout bubble points to the alias 'A' in the query, containing the text: "A é o alias para Aluno e D o alias para Dependente".

Below the query window, there are two tabs: "Resultados" and "Mensagens". The "Resultados" tab is selected, displaying the following table:

	Nome	nome
1	José da Silva	André
2	Diego Oliveira	Eduarda
3	Monica Santos	Maria Luiza

Fonte: elaborada pelo autor.

Caso exista a necessidade de fazer junção com mais tabelas, é só adicionar mais uma cláusula *inner join* com a condição de junção.

6.3.2 Junção externa à esquerda – *Left Outer Join*

Uma junção externa (*Left Outer Join*) mostra todas as linhas da tabela 1 e as combinações existentes na tabela 2.

Sintaxe:

```
Select atributos  
From tabela1  
Left [Outer] Join tabela2  
On Condição de Junção
```

Figura 6.11 – *Left Join*

The screenshot shows a SQL query window with the following code:

```
Select A.Nome, D.Nome  
From Aluno A  
Left Join  
Dependente D  
On A.CPF = D.CPF  
Order by A.nome
```

The word "Left Join" is highlighted with a red rectangle. Below the query window is a results grid titled "Resultados". The data is as follows:

	Nome	Nome
1	Diego Oliveira	Eduarda
2	Joana Souza	NULL
3	João Pereira	NULL
4	José da Silva	André
5	Marco Antônio Alvarenga	NULL
6	Maria da Silva	NULL
7	Monica Santos	Maria Luiza

Fonte: elaborada pelo autor.

6.3.3 Junção externa à direita – Right Outer Join

Uma junção externa (*Right Outer Join*) mostra todas as linhas da tabela 2 e as combinações existentes na tabela 1.

Sintaxe:

```
Select atributos
From tabela1
Right [Outer] Join tabela2
On Condição de Junção
```

No exemplo da figura 6.12, tem-se como resultado todos os registros da tabela dependente (que no caso são somente 3) e todas as combinações destes 3 registros com a tabela aluno.

Figura 6.12 – *Right Join*

The screenshot shows the Microsoft SQL Server Management Studio interface. In the top-left pane, there is a tree view of database objects. Below it, the main area contains a query window with the following T-SQL code:

```
Select A.Nome, D.Nome
From Aluno A
Right Join Dependente D
On A.CPF = D.CPF
Order by A.nome
```

A red box highlights the word "Right Join". The bottom part of the interface shows a results grid titled "Resultados". The data is as follows:

	Nome	Nome
1	Diego Oliveira	Eduarda
2	José da Silva	André
3	Monica Santos	Maria Luiza

Fonte: elaborada pelo autor.

Um fator importante é que a utilização do *left* e *right* podem mudar os resultados dependendo da ordem das tabelas na junção. Veja que a Figura 6.13 foi a primeira apresentada na tabela dependente e o *Right Join* foi colocada na tabela Aluno.

Figura 6.13 – *Right Join*

The screenshot shows a Microsoft SQL Server Management Studio interface. In the top-left pane, there is a query editor window containing the following T-SQL code:

```
Select A.Nome, D.Nome  
From Dependente D  
Right Join  
Aluno A  
On D.CPF = A.CPF  
Order by A.nome
```

The word "Right Join" is highlighted with a red rectangular box. Below the query editor is a results grid titled "Resultados". The grid has two columns: "Nome" and "Nome". The data is as follows:

	Nome	Nome
1	Diego Oliveira	Eduarda
2	Joana Souza	NULL
3	João Pereira	NULL
4	José da Silva	André
5	Marco Antônio Alvarenga	NULL
6	Maria da Silva	NULL
7	Monica Santos	Maria Luiza

Fonte: elaborada pelo autor.

Se você observar, o resultado é exatamente o mesmo da figura 6.11, onde se usou o *Left Join*. Portanto, tenha muito cuidado ao usar o *Left* e o *Right Join*. Tenha certeza que o resultado apresentado é exatamente aquele que você espera atingir.

6.3.4 Junção cruzada e completa – Cross Join e Full Join

A junção cruzada (*Cross Join*) de tabelas, também denominada junção irrestrita, faz um produto cartesiano tabela1 X tabela2, ou seja, mostra um resultado formado pela combinação de todas as linhas da primeira tabela por todas as linhas da segunda. Não existe uma condição de junção.

O *Full Join* faz uma combinação, trazendo todas as linhas da primeira tabela com seus respectivos registros relacionados na tabela 2, e depois traz todos os registros da tabela 2 e seus registros relacionados na tabela 1.

Na figura 6.14 temos o exemplo de *Cross Join*, e na figura 6.15, o exemplo de *Full Join*. A junção cruzada trouxe todas as combinações possíveis entre linhas das tabelas, enquanto a Junção completa apresentou o resultado de um *Left Join* e *Right Join* nas mesmas tabelas.

Figura 6.14 – *Cross Join*

The screenshot shows a SQL query window and a results grid. The query is:

```

Select A.Nome, D.Nome
From Dependente D
Cross Join
Aluno A
Order by A.nome
  
```

The results grid has two columns: 'Nome' and 'Nome'. It contains 21 rows, each with a number from 1 to 21 in the first column and two names in the second column. The names are repeated across the rows, showing all combinations of students and dependents.

	Nome	Nome
1	Diego Oliveira	André
2	Diego Oliveira	Maria Luiza
3	Diego Oliveira	Eduarda
4	Joana Souza	André
5	Joana Souza	Maria Luiza
6	Joana Souza	Eduarda
7	João Pereira	Eduarda
8	João Pereira	Maria Luiza
9	João Pereira	André
10	José da Silva	Maria Luiza
11	José da Silva	André
12	José da Silva	Eduarda
13	Marco Antônio Alvarenga	André
14	Marco Antônio Alvarenga	Maria Luiza
15	Marco Antônio Alvarenga	Eduarda
16	Maria da Silva	André
17	Maria da Silva	Maria Luiza
18	Maria da Silva	Eduarda
19	Monica Santos	André
20	Monica Santos	Eduarda
21	Monica Santos	Maria Luiza

Fonte: elaborada pelo autor.

Figura 6.15 – Full Join

The screenshot shows a SQL query window with the following code:

```
Select A.Nome, D.Nome
From Dependente D
Full Join
Aluno A
On D.CPF = A.CPF
Order by A.nome
```

The results pane shows the output of the query:

	Nome	Nome
1	Diego Oliveira	Eduarda
2	Joana Souza	NULL
3	João Pereira	NULL
4	José da Silva	André
5	Marco Antônio Alvarenga	NULL
6	Maria da Silva	NULL
7	Monica Santos	Maria Luiza

Fonte: elaborada pelo autor.

Não é recomendável a utilização de nenhuma das junções apresentadas (cruzada e completa). Estas junções têm um grande custo e a utilização de recursos para um resultado que não é aplicado com frequência na prática.

6.4 Subconsultas aninhadas

Subconsultas aninhadas são utilizadas quando existe a necessidade de colocar uma expressão *Select From Where* inserida dentro de outra consulta. Esta subconsulta gera um conjunto de valores e serão validadas de acordo com a condição da cláusula *where*.

Sintaxe:

```

Select atributos
From Tabela
Where Atributo IN
  (Select atributo from Tabela2 [Where condição])
Ou
Select atributos
From Tabela1
Where Atributo NOT IN
  (Select atributo from Tabela2 [Where condição])

```

O conectivo *in* testa se os valores resultantes da consulta aninhada atendem a condição do *Where* para tabela 1, enquanto o conectivo *not in* testa a ausência dos valores de um conjunto.

No exemplo da figura 16, temos na cláusula *Where* e o CPF *in* (*Select CPF From Dependente*). Neste caso, o SGBD vai na tabela dependente e recupera todos os CPFs, gerando assim um *result set* R1 com uma única coluna. Depois, ele compara o CPF do aluno com cada CPF do *result set*, trazendo os registros da tabela Aluno que atendem a condição.

Esta comparação é feita uma a uma, como se fosse um *Where CPF = V1 or CPF = V2 or ... CPF = VN*, sendo V1 a Vn todos os valores trazidos no *result set* R1.

Nesta *query* (figura 6.16), ele está trazendo o nome do todos os alunos que tem dependentes. Neste momento ele não precisa saber quais são os dependentes.

Banco de Dados

Figura 6.16 – Consulta aninhada *In*

The screenshot shows a database interface with a query editor and a results viewer. The query in the editor is:

```
SELECT Nome
  FROM Aluno
 WHERE CPF IN (SELECT CPF FROM Dependente)
```

The results viewer shows a table titled "Resultados" with one column "Nome". The data is:

	Nome
1	José da Silva
2	Monica Santos
3	Diego Oliveira

Fonte: elaborada pelo autor.

A figura 6.17 faz a mesma execução da figura 6.16, com a diferença que traz somente os Alunos que não têm dependentes. Ou seja, enquanto no *IN* a comparação é feita como uma igualdade, no *NOT IN* a comparação é feita de maneira que não seja igual a cada elemento do *result set*.

Figura 6.17 – Consulta aninhada *Not In*

The screenshot shows a database interface with a query editor and a results viewer. The query in the editor is:

```
SELECT Nome
  FROM Aluno
 WHERE CPF NOT IN (SELECT CPF FROM Dependente)
```

The results viewer shows a table titled "Resultados" with one column "Nome". The data is:

	Nome
1	Maria da Silva
2	Joana Souza
3	João Pereira
4	Marco Antônio Alvarenga

Fonte: elaborada pelo autor.

Importante

Tome cuidado ao usar as subconsultas aninhadas em tabelas muitos grandes, pois a comparação feita uma a uma vai consumir muito tempo e muitos recursos.

6.5 Uniões

As uniões (*union*) combinam os resultados de duas ou mais *queries* em um único *result set*. Todas as linhas de todas as consultas dentro da união vão sendo adicionadas.

As uniões exigem que as seguintes regras sejam obedecidas:

- ✗ os tipos de dados não precisam ser iguais, mas devem ser compatíveis para a conversão implícita;
- ✗ a quantidade e a ordem das colunas devem ser as mesmas em todas as consultas.

O exemplo da figura 6.18 apresenta a utilização de *union*. Observe que o *select* é feito em duas tabelas Aluno e Funcionário. Os atributos não têm o mesmo nome, mas são de tipos compatíveis, então o comando funciona corretamente.

Figura 6.18 – Union

```

Select Nome from Aluno
Union
Select NomeFuncionario from Funcionario
  
```

	Nome
1	Carla Brunni
2	Diego Oliveira
3	Joana Souza
4	João da Silva
5	João Pereira
6	José da Silva
7	Marco Antônio Alvarenga
8	Maria da Silva
9	Maria de Souza
10	Monica Santos

Fonte: elaborada pelo autor.

A figura 6.19 apresenta uma *union* das mesmas tabelas, mas além do Nome, também é solicitado o sexo. A tabela Aluno tem este atributo, mas a tabela funcionário não tem. Neste caso, houve a necessidade de se colocar NULL para indicar que nas linhas que não tivesse o valor para sexo será preenchido com ele.

Figura 6.19 – Union

The screenshot shows a SQL query window and a results grid. The query is:

```
Select Nome, Sexo From Aluno
Union
Select NomeFuncionario, NULL From Funcionario
```

The results grid has two columns: 'Nome' and 'Sexo'. The data is as follows:

	Nome	Sexo
1	Carla Brunni	NULL
2	Diego Oliveira	M
3	Joana Souza	F
4	João da Silva	NULL
5	João Pereira	M
6	José da Silva	M
7	Marco Antônio Alvarenga	M
8	Maria da Silva	F
9	Maria de Souza	NULL
10	Monica Santos	F

Fonte: elaborada pelo autor.

6.6 Visões – Views

Um SGBD oferece várias estruturas ou objetos (não confundir com orientação a objeto). Primeiro, foi apresentado o banco de dados. Dentro do banco já foram criadas tabelas, relacionamentos e restrições. As *views* são mais uma das estruturas do banco de dados que auxiliam ao usuário.

View (visão) é uma tabela virtual formada por linhas e colunas de dados vindos de outras tabelas ou views criadas por uma *query*.

A importância de uma *view* é que, ao criá-la, pode-se filtrar os nomes de colunas, conteúdo de uma tabela ou mais tabelas a serem exibidas, agrupando e protegendo determinadas colunas e simplificando o código para o programador.

As *views* não ocupam espaço no banco de dados. Suas linhas e colunas são geradas dinamicamente no momento em que é feita uma referência a ela.

Sintaxe:

```
Create view NomeDaVisao
AS
Query
```

A figura 6.20 apresenta a criação de uma *view* para a tabela aluno, apresentando somente os atributos Nome, Sexo e Salário. Veja que foi utilizado um *alias* no nome e no salário para ocultar os verdadeiros nomes de colunas.

Figura 6.20 – Criação de *view*

```
>Create view vwAluno
as
Select Nome as Aluno, Sexo, Salario as Salário
From Aluno
```

100 % <

Mensagens

Comando(s) concluído(s) com êxito.

Fonte: elaborada pelo autor.

A figura 6.21 nos mostra que podemos criar *views* com *union*. A flexibilidade de utilização das visões é bem grande, praticamente pode-se utilizar qualquer *query* para criar uma *view*.

Banco de Dados

Figura 6.21 – Criação de *view*

```
Create view vwVisaoNomeSexo
as
Select Nome, Sexo from Aluno
Union
Select NomeFuncionario, NULL from Funcionario
```

Mensagens
Comando(s) concluído(s) com êxito.

Fonte: elaborada pelo autor.

Para utilizar uma *view*, você aplica um *select*, como se a *view* fosse uma tabela como as demais. Pode-se aplicar também *Order By*, *Group By*, *Where* etc (figuras 6.22 e 6.23).

Figura 6.22 – Utilizando *view*

```
Select *
From vwAluno
Order by Aluno
```

Resultados

	Aluno	Sexo	Salário
1	Diego Oliveira	M	1500,00
2	Joana Souza	F	NULL
3	João Pereira	M	1590,00
4	José da Silva	M	1200,00
5	Marco Antônio Alvarenga	M	3700,00
6	Maria da Silva	F	2600,00
7	Monica Santos	F	2000,00

Fonte: elaborada pelo autor (2016).

Figura 6.23 – Usando View com Where

The screenshot shows a Microsoft SQL Server Management Studio interface. In the top-left pane, there is a code editor window containing the following T-SQL query:

```
Select *
From vwAluno
Where Sexo = 'M'
Order by Aluno
```

Below the code editor is a results grid titled "Resultados". The grid displays four rows of data from the view, showing columns for "Aluno", "Sexo", and "Salário". The data is as follows:

	Aluno	Sexo	Salário
1	Diego Oliveira	M	1500,00
2	João Pereira	M	1590,00
3	José da Silva	M	1200,00
4	Marco Antônio Alvarenga	M	3700,00

Fonte elaborado pelo autor (2016).

Em uma *view*, podem ser feitas as operações *select*, *insert*, *update* e *delete*, desde que não interfira em nenhuma restrição. Portanto, seja cuidadoso na criação de *views* somente para leitura.

Para alterar o conteúdo de uma visão, utilize a seguinte sintaxe.

```
Alter view NomeDaVisao
```

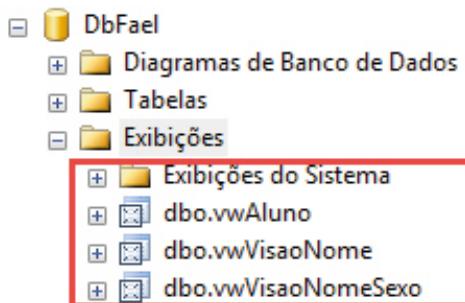
```
AS
```

```
Query
```

Para visualizar as visões existentes em um banco de dados, basta selecionar o nome do banco na interface gráfica do MS SQL SERVER e clicar em **exibições do sistema** (Figura 6.24).

Banco de Dados

Figura 6.24 – Exibição de *view*



Fonte: elaborada pelo autor.

Para eliminar uma visão, utilize a seguinte sintaxe.

```
Drop view NomeDaVisao
```

Após executar o seguinte comando Drop View vwVisaoNomeSexo, a *view* será eliminada do SGBD.

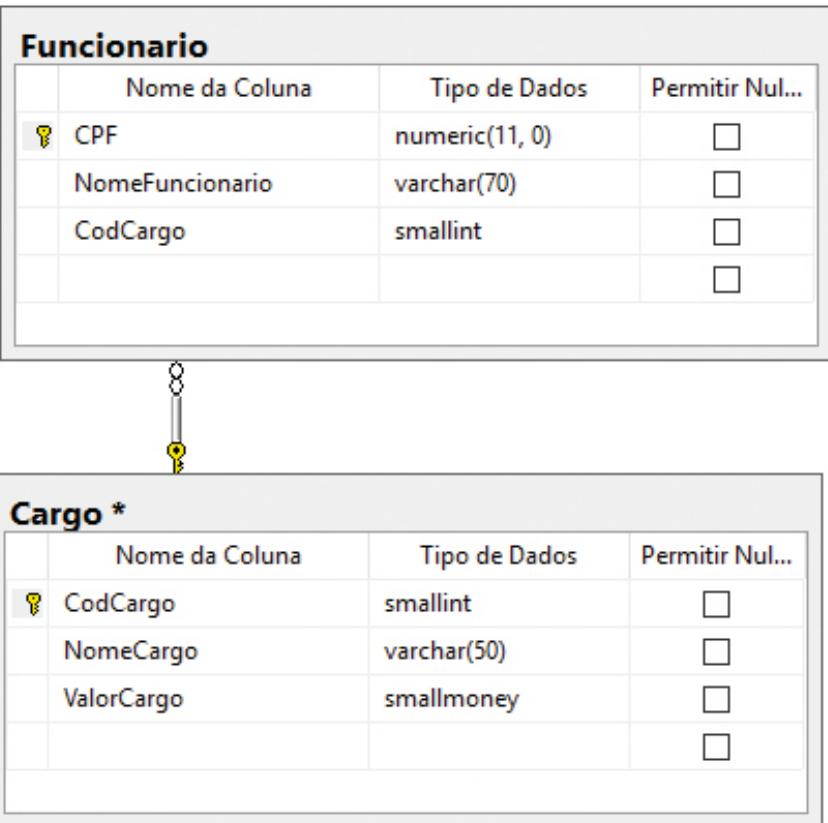
Síntese

Este capítulo apresentou várias funções oferecidas pela SQL para que o SGBD seja utilizado da melhor maneira. Foram vistas funções agregadas, que já fazem vários cálculos sobre os atributos, e que podem ser utilizadas funções de agrupamento e aplicar condições sobre os grupos utilizando a cláusula *having*.

As junções mostraram que se pode recuperar dados entre várias tabelas seguindo os relacionamentos entre elas. Já a consultas aninhadas podem utilizar um *result set* como entrada para uma cláusula *where*.

Atividades

Dado o modelo a seguir, faça as seguintes tarefas:



- ✗ Crie um banco de dados, dentro dele crie as tabelas e insira 3 registros em cada.
- ✗ Crie uma consulta que selecione o nome do funcionário e nome do cargo.
- ✗ Crie uma visão para o item B.
- ✗ Utilize a *view*, ordenando por nome cargo, descendente.

7

Transações e Técnicas Avançadas

Os SISTEMA DE gerenciamento de banco de dados (SGBDs), além de fornecerem funções básicas de *Structured Query Language* (SQL – Linguagem de Consulta Estruturada, em português), oferecem o controle de transações, concorrências e outros recursos que permitem a programação avançada dentro do banco de dados. Os servidores de banco de dados são equipamentos muito potentes com excelente configuração de hardware e sistema operacional, o que permite que os programas oferecidos dentro do SGBD sejam executados com muita rapidez e eficiência, evitando que os dados trafeguem na rede sem necessidade.

Objetivos de aprendizagem:

- × explicar o conceito de transação;
- × apresentar a extensão da linguagem SQL Transact SQL;
- × compreender Stored Procedures.

7.1 Transação

Existem determinadas ações dentro de um banco de dados que atualizam vários registros. Às vezes, durante essa atualização, ocorrem falhas que podem ser falhas de sistemas, como falta de energia, ou falhas de hardware, como um disco defeituoso ou uma fonte com problemas. O SGBD deve garantir que, caso essas falhas ocorram, o conteúdo do banco de dados continue sendo confiável. Para resolver esse problema, foi criado o conceito de transação.

Segundo Elmasri e Navathe (2013), uma transação é uma unidade atômica de trabalho que deve ser completamente concluída ou completamente desfeita. As transações possuem várias propriedades, conhecidas como ACID, sigla formada por:

- × **atomicidade** – uma transação deve ser totalmente realizada ou deverá ser desfeita;
- × **consistência** – uma transação deve ser executada do começo ao fim sem a interferência de outras transações, garantindo a consistência dos dados;
- × **isolamento** – mesmo existindo várias transações sendo executadas ao mesmo tempo, a transação deve ser executada isoladamente, sem interferir em outras transações;
- × **durabilidade** – as mudanças aplicadas pela transação no banco de dados devem ser armazenadas no banco e não podem ser perdida por causa de falhas.

Para garantir a recuperação dos dados e manter a integridade do banco, o sistema precisa registrar quando cada transação começa, termina, confirma ou cancela. Para atender às condições da transação, foram criadas as seguintes operações (ELMASRI; NAVATHE, 2013):

- ✗ *begin transaction* – marcando o início da transação.
- ✗ *read/write* – leitura ou gravação nos registros do banco, que podem ser insert, update, delete e select.
- ✗ *end transaction* – indica que a operação de leitura e escrita terminou.
- ✗ *commit transaction* – indica que as operações foram bem-sucedidas e podem ser efetivadas no banco.
- ✗ *rollback (abort)* – indica que as operações não obtiveram sucesso e não podem ser efetivadas no banco.

Log file é outra estrutura utilizada para auxiliar no gerenciamento da transação, é um arquivo físico no qual são inseridas as transações que estão sendo efetuadas. O SGBD tem um *buffer de log*; quando esse *buffer* fica cheio, os dados são armazenados no arquivo de log, cada transação é inserida nesse arquivo e quando a transação é finalizada com sucesso o Commit é executado e os dados armazenados no log são efetivados no arquivo de dados. Caso o Rollback seja executado, os dados são eliminados do log file sem serem efetivados no banco.

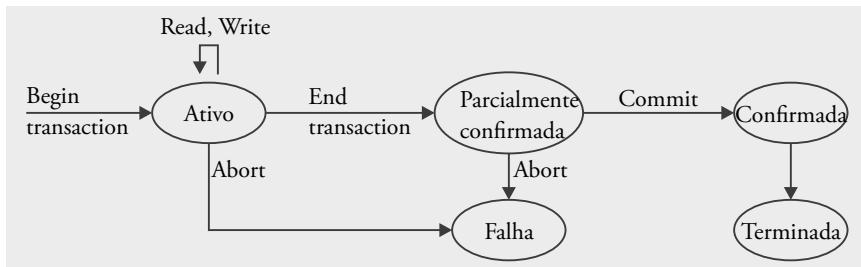
A execução das operações de transação ocorrem conforme a Figura 7.1: primeiro indica-se que a partir de determinado ponto começará a transação, a partir daí todos os comandos que fazem parte dessa transação são armazenados no log, até que seja:

- ✗ abortada ou ocorra uma falha – executa o rollback;
- ✗ ocorra a confirmação da transação efetuada com sucesso – executa o Commit.

Dessa forma o SGBD garante a consistência do banco de dados.

Banco de Dados

Figura 7.1 – Transação



Fonte: Elmasri, Navathe (2013, p. 506).

Para exemplificar a utilização da transação, utilizaremos a tabela Aluno, com o conteúdo mostrado na Figura 7.2.

Figura 7.2 – Conteúdo da tabela

A captura de tela mostra uma interface de usuário de um gerenciador de banco de dados. No topo, há uma barra com o texto "Select * from Aluno" e uma barra de escala de 100%. Abaixo, uma tabela intitulada "Resultados" é exibida. A tabela tem 10 colunas: CPF, Nome, Endereço, Complemento, Dt_Nasc, Sexo, Telefone, Cidade, Estado e Salário. Existem 7 registros na tabela:

	CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Cidade	Estado	Salário
1	11	José da Silva	Rua dos Guarapes, 37, Bairro: Centro	NULL	1983-12-07	M	NULL	RJ	RJ	1200,00
2	22	Maria da Silva	Rua do Chapéu, 455	Ap 12	1989-02-02	F	NULL	Manaus	AM	2600,00
3	33	Joana Souza	Rua das Mendes, 127	Ap 43	1997-12-05	F	NULL	Salvador	BA	NULL
4	44	João Pereira	Rua Rui Barbosa, 24	NULL	1990-10-15	M	NULL	Ponto Alegre	RS	1500,00
5	55	Marco Antônio Alves	Av XV de Novembro, 1733	NULL	1974-11-11	M	NULL	Brasília	DF	3700,00
6	66	Monica Santos	Este é o endereço	Ap 37	1995-06-23	F	NULL	Belo Horizonte	MG	2000,00
7	77	Diego Oliveira	Rua das Flores	NULL	1981-05-15	M	NULL	Maceió	AL	1500,00

Fonte: elaborado pelo autor (2016)

Agora vejamos um exemplo prático de transação no MS SQL Server.

Sintaxe: observe que você pode utilizar a palavra transaction ou tran e o resultado será o mesmo, e o Commit pode até ser omitido sem problemas.

```
Begin tran[saction]  
Commit [Tran]  
-- Ou  
Rollback [Tran]
```

Na Figura 7.3, temos o início da transação no código SQL pelo comando Begin Transaction, depois o comando Delete, que apaga todos os dados da tabela Aluno, dos alunos que não têm dependentes. Quatro linhas foram apagadas, mas a seguir temos o Rollback desfazendo todas as alterações feitas pelo Delete.

Figura 7.3 – Executando transações

```

-- Começo da transação
-- Comandos
Delete from Aluno
where cpf not in (select CPF from Dependente)
Rollback -- Desfazendo as alterações


```

100 % <

Mensagens

(4 linha(s) afetadas)

Fonte: elaborado pelo autor.

Ao fazer o select novamente na tabela Aluno, todos os registros continuam lá, comprovando que, quando o Rollback foi executado, desfez todos os deletes.

Figura 7.4 – Rollback executado

	CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Idade	Estado	Salário
1	11	Joel de Silva	Rua dos Guararapes, 37, Bairro: Centro	NULL	1963-12-07	M	NULL	RJ	RJ	1200,00
2	22	Maria da Silva	Rua do Chapéu, 455	Ap 12	1989-02-02	F	NULL	Manaus	AM	2600,00
3	33	Joana Souza	Rua das Memórias, 137	Ap 43	1997-12-05	F	NULL	Salvador	BA	NULL
4	44	João Pereira	Rua Rui Barbosa, 24	NULL	1990-10-15	M	NULL	Porto Alegre	RS	1550,00
5	55	Marco Antônio Alverenga	Av XV de Novembro, 1733	NULL	1974-11-11	M	NULL	Brasília	DF	3700,00
6	66	Monica Santos	Este é o endereço	Ap 37	1995-06-23	F	NULL	Belo Horizonte	MG	2000,00
7	77	Diego Oliveira	Rua das Flores	NULL	1981-05-15	M	NULL	Macedônia	AL	1500,00

Fonte: elaborado pelo autor.

Banco de Dados

Na Figura 7.5, a transação é inicializada com o Begin Tran, depois é feito um Update que aumenta o salário de todos os Alunos em 10%. Em seguida, é realizado um Commit, que vai efetivar as atualizações no banco de dados.

Pode-se colocar inúmeros comandos dentro de uma transação, que podem ser de select, insert, update e delete, sendo que no momento que for feito o Commit ou o Rollback serão efetivados ou desfeitos todos os comandos pendentes no log de transação, independentemente de seu tipo. Após as operações serem efetivadas (commit), não há mais como desfazer, a não ser restaurando backup.

Figura 7.5 – Execução com Commit

```
-- Begin Transaction -- Começo da transação
-- Comandos
Update Aluno
Set Salario = Salario + salario * 0.1
Commit -- Efetivando as alterações
(7 linha(s) afetadas)
```

Fonte: elaborado pelo autor.

Após a execução do Commit, a Figura 7.6 mostra um select comprovando que o salário de todos os alunos sofreu aumento de 10%.

Figura 7.6 – Após o Commit

	CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Cidade	Estado	Salario
1	11	José da Silva	Rua dos Guarapes, 37, Bairro: Centro	NULL	1963-12-07	M	NULL	RJ	RJ	1320,00
2	22	Maria da Silva	Rua do Chapéu, 455	Apt 12	1989-02-02	F	NULL	Manaus	AM	2880,00
3	33	Joá Souza	Rua das Memórias, 137	Apt 43	1997-12-05	F	NULL	Salvador	BA	NULL
4	44	João Pereira	Rua Rui Barbosa, 24	NULL	1990-12-15	M	NULL	Porto Alegre	RS	1745,00
5	55	Marco Antônio Alvesnega	Ay XIV de Novembro, 1733	NULL	1974-11-11	M	NULL	Brasília	DF	4070,00
6	66	Monica Sartori	Este é o endereço	Apt 37	1995-06-23	F	NULL	Belo Horizonte	MG	2200,00
7	77	Diego Oliveira	Rua das Flores	NULL	1981-05-15	M	NULL	Macedônia	AL	1650,00

Fonte: elaborado pelo autor.

Um fator importante quando as transações são armazenadas no arquivo de log é o momento em que ocorre uma falha, o serviço de SQL cai ou até mesmo o servidor é reiniciado. Quando a instância do banco de dados sobe, o SGBD verifica todas as transações pendentes no log e desfaz todas elas.

7.2 Transact SQL (T-SQL)

Muitos SGBDs oferecem extensão do SQL padrão: o Oracle tem o PL SQL, o SQL Server tem o Transact SQL, e assim para os demais SGBDs. Esse complemento da linguagem SQL é muito útil, pois oferece novas funcionalidades a serem utilizadas dentro do banco.

Importante

Aprenda mais sobre Transact SQL executando o tutorial da Microsoft sobre T-SQL: <[https://msdn.microsoft.com/pt-br/library/ms365303\(v=sql.120\).aspx](https://msdn.microsoft.com/pt-br/library/ms365303(v=sql.120).aspx)>.

Vejamos algumas funções da Transact SQL:

Quadro 7.1 – Transact SQL

Função	Sintaxe
Declaração de variáveis	DECLARE @V1, @V2, ... @Vn Tipo de dados
Indicadores de blocos	BEGIN END
Atribuição de valores	SELECT @V1 = valor1, @V2 = valor2 Ou SET @V1 = valor1
While	WHILE condição

Função	Sintaxe
Case	CASE expressão WHEN Condição THEN resultado [ELSE resultado do Else] END
Convert	CONVERT (tipo de dado des- tino [(tamanho)], expressão)
Cast	CAST (expressão AS tipo de dado [(tamanho)])
Data do sistema	GETDATE()
Retorna parte da data	DATEPART (Parte que se deseja, data) Parte que se deseja é DD para, dia MM, para mês, YYYY para ano
Adiciona/subtrai valores a data	DATEADD (Parte que se deseja, número, data) DATEDIFF (Parte que se deseja, data1, data2)
Imprimir	Print

Fonte: elaborado pelo autor.

No exemplo da Figura 7.7, no início tem-se a declaração das variáveis, basta separar por vírgula e é possível declarar quantas variáveis forem necessárias. Variáveis locais, que são vistas apenas na sessão, são indicadas por @; já nas variáveis globais e de sistemas são utilizadas @@. Logo depois vem a atribuição, utilizando o comando set, que só aceita uma atribuição por vez. O comando Print imprime o conteúdo da variável @nome; Print somente mostra strings, e qualquer outro tipo de dado deve ser convertido para usar o comando.

Depois tem-se o comando While, que segue o mesmo padrão de linguagens de programação: se for somente uma linha de execução, não é preciso colocar os blocos de início e fim (begin e end), mas se for mais de um comando dentro do while é obrigatório o uso de definição de blocos.

Figura 7.7 – Exemplos de TSQL

The screenshot shows a SQL Server Management Studio window. On the left, there's a tree view of the script structure. The main pane contains the following T-SQL code:

```

    Declare @nome varchar(10),
            @num smallint

    Set @nome = 'Fernanda'
    Set @num = 1

    Print @Nome

    While @num < 11
    Begin

        Print 'O número é ' + Convert(char(5), @num)
        Set @num = @num + 1

    End
  
```

Below the code, the status bar shows "100 %". To the right, under the "Mensagens" tab, the output is displayed as:

```

  Mensagens
  Fernanda
  O número é 1
  O número é 2
  O número é 3
  O número é 4
  O número é 5
  O número é 6
  O número é 7
  O número é 8
  O número é 9
  O número é 10
  
```

Fonte: elaborado pelo autor.

Dentro do while temos a impressão incremento da variável @num, a cada volta do loop. Observe que foi necessária a conversão para char de tamanho 5, e poderia ter sido usado o comando Cast no lugar do convert.

A Figura 7.8 mostra a declaração de variáveis do tipo datetime e inteiro. Para se trabalhar com datas, é preciso funções especiais, afinal data é do tipo composto formado por dia, mês e ano e datetime é formado por dia, mês, ano, hora, minuto, segundo e milissegundo. Outra informação importante em relação a tipos de data são os formatos: no Brasil, temos DD/MM/AAAA, mas em outras regiões os formatos são diferentes, como em países de idioma inglês, nos quais o formato é MM-DD-YYYY.

Figura 7.8 – Exemplos TSQL

The screenshot shows a T-SQL script in the SSMS query editor and its execution results in the 'Mensagens' (Messages) window.

```
Declare @dt_ini Datetime,  
        @dt_fim Datetime,  
        @num int  
  
Select @dt_fim = GETDATE (), @dt_ini = '01/01/1972'  
  
Print 'Diferença das datas em anos é ' +  
      Convert(char(5), Datediff(yy, @dt_ini, @dt_fim))  
  
Print 'Diferença das datas em meses é ' +  
      Convert(char(5), Datediff(MM, @dt_ini, @dt_fim))  
  
Print 'Diferença das datas em dias é ' +  
      Convert(char(5), Datediff(DD, @dt_ini, @dt_fim))
```

100 % <

Mensagens

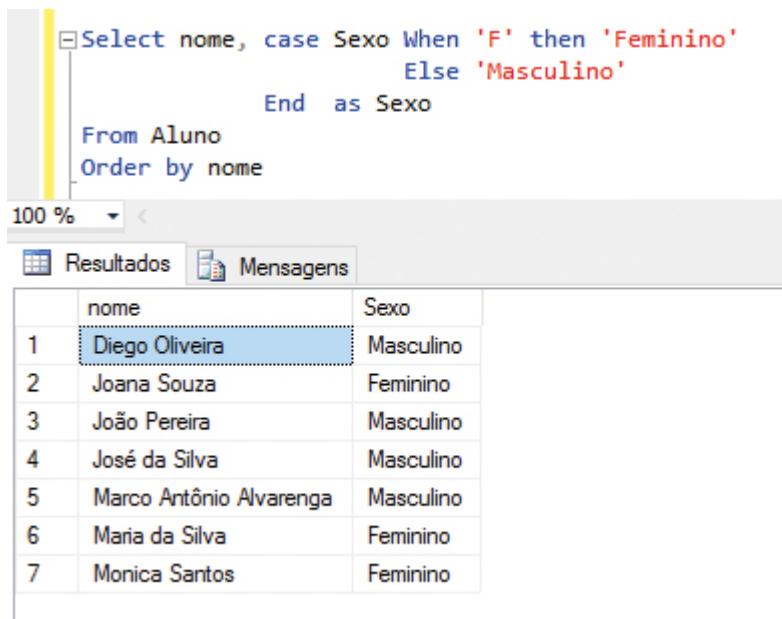
```
Diferença das datas em anos é 44  
Diferença das datas em meses é 535  
Diferença das datas em dias é 16311
```

Fonte: elaborado pelo autor.

Tem-se a atribuição de GETDATE(), que traz a data e a hora do servidor e a dt_fim atribuída manualmente. Depois vem a utilização da Função DateDiff, trazendo a diferença das datas em anos, meses e dias.

Na Figura 7.9 tem-se a demonstração da utilização do CASE, nesse caso dentro do Select. O case executa determinadas funções de acordo com o valor da variável de entrada. O Select traz o nome e o sexo, mas, ao invés de mostrar F e M, foi criado um case para sexo, para mudar o conteúdo a ser apresentado de acordo com o valor da coluna.

Figura 7.9 – Case



```

Select nome, case Sexo When 'F' then 'Feminino'
                      Else 'Masculino'
                 End   as Sexo
From Aluno
Order by nome
  
```

The screenshot shows a SQL query in the query editor of SSMS. The query uses a CASE statement to map gender values from the 'Sexo' column to 'Feminino' or 'Masculino'. The results are displayed in a table titled 'Resultados' with columns 'nome' and 'Sexo'. The data shows seven rows of student names and their corresponding genders.

	nome	Sexo
1	Diego Oliveira	Masculino
2	Joana Souza	Feminino
3	João Pereira	Masculino
4	José da Silva	Masculino
5	Marco Antônio Alvarenga	Masculino
6	Maria da Silva	Feminino
7	Monica Santos	Feminino

Fonte: elaborado pelo autor.

Existem muitas outras funções e comandos oferecidos pelo T-SQL; quanto mais comandos você aprender, melhor irá usá-los.

7.3 Stored Procedures (SP)

Uma Stored Procedure, também conhecida como um procedimento armazenado, funciona de forma semelhante aos procedimentos de linguagem de programação. São códigos previamente escritos e armazenados no servidor

e, quando necessário, solicita-se sua execução. No SQL Server é um grupo de uma ou mais instruções T-SQL. Os procedimentos podem:

- × aceitar parâmetros de entrada e retornar vários valores de saída para o programa de chamada;
- × ter instruções de programação que executam comandos no banco de dados. Pode-se também ter chamada a outros procedimentos;
- × devolver um valor de retorno que mostra o status para quem chamou o procedimento, indicando sucesso ou erro.

As vantagens do uso de procedimentos armazenados, segundo as recomendações da Microsoft são (cada stored procedure segue a extensão de linguagem SQL específica do SGBDs, portanto escolheu-se o padrão de T-SQL para o estudo neste capítulo):

- × **tráfego de rede de servidor/cliente reduzido** – a execução de uma em um único lote de código pode reduzir significativamente o tráfego da rede entre cliente e servidor porque a única chamada para executar o procedimento é enviada pela rede e somente o resultado é devolvido pela rede.
- × **segurança mais forte** – o procedimento controla quais processos e comandos são executados e protege o banco, pois nem o usuário nem o programador tem acesso direto ao código. Ajuda protegendo contra os ataques de injeção sql e os procedimentos podem ser criptografados.
- × **reutilização de código** – em sps, o código é um só, acessado e executado por qualquer usuário ou programa.
- × **manutenção facilitada** – quando existe a necessidade de alterar o código de uma sp, não é necessária nenhuma alteração na aplicação (a menos que os parâmetros mudem), os privilégios e acessos continuam os mesmos e outra vantagem é que, quando o procedimento é atualizado, todos as chamadas a essa sp acessarão a nova versão automaticamente.
- × **melhor desempenho** – como o procedimento é compilado na primeira vez em que é executado e cria um plano de execução em

cada execução, caso não haja nenhuma mudança significativa, o mesmo plano é utilizado cada vez que a SP é executada, economizando tempo.

Vejamos, a seguir, exemplos de criação de Stored Procedures.

7.3.1 Criando SPs

A sintaxe para criar uma Stored Procedure é:

```
CREATE PROC [ EDURE ] procedure_name
[ {@parametro tipo de dados }
[ VARYING ] [ = default ] [ OUTPUT ] ] [
,...n ]
WITH
{ RECOMPILE | ENCRYPTION | RECOMPILE ,
ENCRYPTION } ]
[ FOR REPLICATION ]
AS Comandos_sql [ ...n ]
```

Todas as SPs devem ter um nome único em cada banco de dados. Coloca-se o nome da SP o mais próximo de sua utilização, os parâmetros são facultativos e podem ser de INPUT quando vêm com entrada para a SP ou OUTPUT quando são retornados pela SP. RECOMPILE e ENCRYPTION são as opções para recompilar e criptografar as SPs, e FOR REPLICATION indica que a SP é usada para replicação. Logo depois do AS, vêm os comandos SQL. Todos os que foram vistos até agora podem ser utilizados dentro de uma SP, e muitos outros.

O exemplo da Figura 7.10 mostra a criação de uma Stored Procedure, nesse caso chamada pLeAalunos, que faz o select de todas as colunas da tabela aluno e ordena por nome.

Banco de Dados

Figura 7.10 – Criando SPs

The screenshot shows a code editor window with the following T-SQL script:

```
Create Procedure pLeAlunos
as
Select * from Aluno
Order by Nome
```

Below the code editor, a message window displays:

100 % <

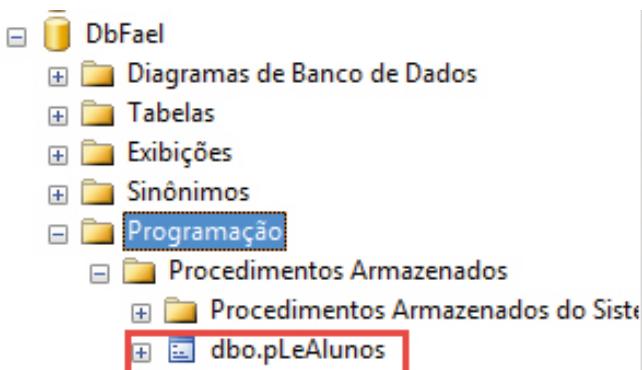
Mensagens

Comando(s) concluído(s) com êxito.

Fonte: elaborado pelo autor.

A Stored Procedure `pLeAluno`, no momento da criação, foi armazenada dentro do banco de dados, que é possível visualizar na Figura 7.11. Dentro do item programação, todas as Stored Procedures criadas são armazenadas dentro do banco de dados e ficam até que sejam eliminadas. Dessa forma, qualquer usuário ou aplicativo que tenha privilégio pode executá-las.

Figura 7.11 – Mostrando SPs no Catalogo



Fonte: elaborado pelo autor.

Existem duas formas de se executar uma Stored Procedure, como mostra a Figura 7.12, utilizando o comando Exec seguido pelo nome da Stored Procedure ou utilizando Execute e o nome da SP. Em alguns casos, é permitido somente chamar o nome da SP, principalmente no ambiente gráfico do SQL Server e utilizando SPs sistemas.

Figura 17.12 – Executando SP

The screenshot shows the SSMS interface with the following details:

- Top pane (Query):** Shows the T-SQL command:


```
Exec spAlunos
-- ou
Execute spAlunos
```
- Bottom pane (Results):** Shows the output of the query. The table has the following columns: CPF, Nome, Endereço, Complemento, Dt_Nasc, Sexo, Telefone, Cidade, Estado, and Salario. The data is as follows:

CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Cidade	Estado	Salario
1 77	Diego Oliveira	Rua das Rosas	NULL	1981-05-15	M	NULL	Maceió	AL	1650,00
2 33	Joane Souza	Rua das Memórias, 137	Ap 43	1997-12-05	F	NULL	Salvador	BA	NULL
3 44	João Pereira	Rua Rui Barbosa, 24	NULL	1990-10-15	M	NULL	Porto Alegre	RS	1749,00
4 11	José da Silva	Rua das Guarapari, 37, Bairro Centro	NULL	1963-12-07	M	NULL	RU	RU	1320,00
5 55	Marcos Antônio Alves	Az XV de Novembro, 1733	NULL	1974-11-11	M	NULL	Brasília	DF	4070,00
6 22	Maria da Silva	Rua do Chapéu, 455	Ap 12	1989-02-02	F	NULL	Manaus	AM	2880,00
7 66	Monica Santos	Este é o endereço	Ap 37	1995-06-23	F	NULL	Belo Horizonte	MG	2200,00

Fonte: elaborado pelo autor.

É muito fácil a criação e a execução de Stored Procedures, e sua utilização é fortemente recomendada, principalmente em aplicações de duas e três camadas.

7.3.2 Criando SPs com parâmetros de entrada

Vimos a criação de uma Stored Procedure simples, sem parâmetros de entrada, agora conhceremos como funciona a passagem de parâmetros nas SPs.

Para criar uma SP com passagem de parâmetros, logo depois da declaração do nome da Procedure são declarados os parâmetros seguindo o mesmo padrão da declaração de variáveis, @ o nome da variável e depois o tipo de dado. Se necessário, pode-se declarar um valor default, caso não seja enviado nenhum valor para o parâmetro na chamada da SP. No exemplo da Figura 7.13 tem-se o parâmetro @nome com o valor default % (ou seja, qualquer string). Esse exemplo recupera o nome e o sexo do Aluno cujos nomes sejam equivalentes ao parâmetro ordenado por nome.

Banco de Dados

Figura 7.13 – SP com parâmetro de Input

The screenshot shows a code editor window with a yellow vertical bar on the left. Inside, there is a script for creating a stored procedure:

```
Create Procedure pLeAlunosNome
    @Nome Varchar(70) = '%'
    as
        Select Nome, Sexo from Aluno
        Where Nome like @Nome
        Order by Nome
```

Below the code editor is a message window titled "Mensagens" (Messages) with the following content:

10 % < Mensagens
Comando(s) concluído(s) com êxito.

Fonte: elaborado pelo autor.

Existem diversas formas de se chamar uma SP com passagem de parâmetros, e vemos alguns deles na Figura 7.14. Como existe valor default no primeiro item, não é passado nada, o parâmetro assumirá o % e trará todos os registros. No segundo, trará todos os alunos que começam com a letra J. No terceiro, todos os Alunos que começem com a letra A, mas mostrando que existe a declaração explícita do parâmetro. No quarto exemplo, trará os Alunos cujo nome comece com as letras C ou K, tenha substring ars no meio seguido de O ou E e termine com a letra n. E, no último exemplo, recuperará o Aluno que se chame somente João.

Figura 7.14 – Execução de SP com parâmetro de Input

The screenshot shows a code editor window with a yellow vertical bar on the left. It contains several EXECUTE statements for the stored procedure pLeAlunosNome:

```
EXECUTE pLeAlunosNome
-- Ou
EXECUTE pLeAlunosNome 'J%'
-- Ou
EXECUTE pLeAlunosNome @Nome = 'A%'
-- Ou
EXECUTE pLeAlunosNome '[CK]ars[OE]n'
-- Ou
EXECUTE pLeAlunosNome 'João'
```

Fonte: elaborado pelo autor.

Os resultados da execução da SP, de acordo com o parâmetro, são representados no Quadro 7.2. No primeiro caso, apresenta todos os alunos, pois, como não foi passado parâmetro, assumiu o %. No segundo caso, todos que começaram com a letra J; e nos últimos três não foi encontrado resultado que atenda às condições enviadas no parâmetro.

Quadro 7.2 – Execução de SP com parâmetro de Input

Num	Forma de Chamar SP	Resultado																								
1	EXECUTE pLe-AlunosNome	<table border="1"> <thead> <tr> <th></th> <th>Nome</th> <th>Sexo</th> </tr> </thead> <tbody> <tr><td>1</td><td>Diego Oliveira</td><td>M</td></tr> <tr><td>2</td><td>Joana Souza</td><td>F</td></tr> <tr><td>3</td><td>João Pereira</td><td>M</td></tr> <tr><td>4</td><td>José da Silva</td><td>M</td></tr> <tr><td>5</td><td>Marco Antônio Alvarenga</td><td>M</td></tr> <tr><td>6</td><td>Maria da Silva</td><td>F</td></tr> <tr><td>7</td><td>Monica Santos</td><td>F</td></tr> </tbody> </table>		Nome	Sexo	1	Diego Oliveira	M	2	Joana Souza	F	3	João Pereira	M	4	José da Silva	M	5	Marco Antônio Alvarenga	M	6	Maria da Silva	F	7	Monica Santos	F
	Nome	Sexo																								
1	Diego Oliveira	M																								
2	Joana Souza	F																								
3	João Pereira	M																								
4	José da Silva	M																								
5	Marco Antônio Alvarenga	M																								
6	Maria da Silva	F																								
7	Monica Santos	F																								
2	EXECUTE pLeA-lunosNome 'J%'	<table border="1"> <thead> <tr> <th></th> <th>Nome</th> <th>Sexo</th> </tr> </thead> <tbody> <tr><td>1</td><td>Joana Souza</td><td>F</td></tr> <tr><td>2</td><td>João Pereira</td><td>M</td></tr> <tr><td>3</td><td>José da Silva</td><td>M</td></tr> </tbody> </table>		Nome	Sexo	1	Joana Souza	F	2	João Pereira	M	3	José da Silva	M												
	Nome	Sexo																								
1	Joana Souza	F																								
2	João Pereira	M																								
3	José da Silva	M																								
3	EXECUTE pLe-AlunosNome @Nome = 'A%'	<table border="1"> <thead> <tr> <th>Nome</th> <th>Sexo</th> </tr> </thead> </table>	Nome	Sexo																						
Nome	Sexo																									
4	EXECUTE pLe-AlunosNome '[CK]ars[OE]n'	<table border="1"> <thead> <tr> <th>Nome</th> <th>Sexo</th> </tr> </thead> </table>	Nome	Sexo																						
Nome	Sexo																									
5	EXECUTE pLe-AlunosNome 'João'	<table border="1"> <thead> <tr> <th>Nome</th> <th>Sexo</th> </tr> </thead> </table>	Nome	Sexo																						
Nome	Sexo																									

Fonte: elaborado pelo autor

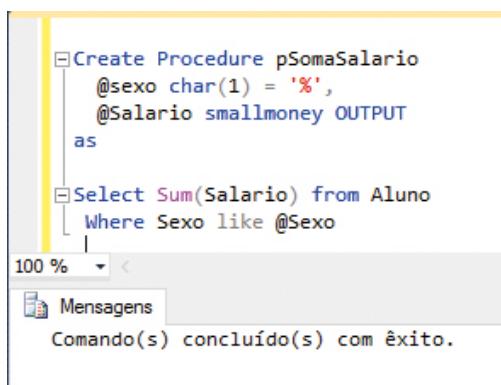
A passagem de parâmetro flexibiliza bastante o uso das Stored Procedures, permitindo que sejam atendidas outras condições. Pode-se passar quantos parâmetros forem necessários; existe uma limitação, mas ela é bem extensa, em torno de 2,1 mil.

7.3.3 Criando SPs com parâmetros de saída

Outra facilidade fornecida são os parâmetros de OUTPUT, que permitem a saída de resultados da SP. São permitidos vários parâmetros de saída em uma SP.

A Figura 7.15 mostra o exemplo da criação de uma SP com parâmetro de OUTPUT. A SP pSomaSalario recebe sexo como parâmetro de input e retorna salário como OUTPUT. Essa Procedure executa a soma dos salários da tabela Aluno de acordo com o valor do sexo que foi enviado como parâmetro de entrada.

Figura 7.15 – Criando SP com parâmetro de Output



```
Create Procedure pSomaSalario
    @sexo char(1) = '%',
    @Salario smallmoney OUTPUT
as
Select Sum(Salario) from Aluno
Where Sexo like @Sexo
```

Fonte: elaborado pelo autor.

Para executar uma SP com parâmetro de OUTPUT é necessário declarar a variável que receberá o parâmetro de OUTPUT, deve-se declarar quantas variáveis forem necessárias para o número de parâmetros de OUTPUT existentes. No caso da Figura 7.16, foi declarada uma variável @total do tipo smallmoney.

Observe que, na Figura 7.16, não é passado o parâmetro sexo, a SP assumirá o valor default % e trará a soma do salário de todos os alunos. Na segunda execução é passado o parâmetro F (feminino), então a SP trará a soma dos salários dos alunos do sexo feminino. E na terceira chamada é importante salientar que, quando se declara o nome do parâmetro na chamada, não é necessário manter a ordem de chamada da SP. O parâmetro de OUPUT @salario foi colocado antes de Sexo; funcionando corretamente, será analisado o nome declarado na chamada, e não a posição.

Figura 7.16 – Executando SP com parâmetro de Output



```

-- Declare @Total smallmoney

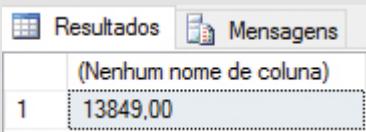
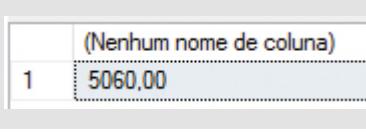
EXECUTE pSomaSalario @Salario = @Total OUTPUT
-- Ou
EXECUTE pSomaSalario 'F', @Salario = @Total OUTPUT
-- Ou
EXECUTE pSomaSalario @Salario = @Total OUTPUT, @Sexo = 'M'

```

Fonte: elaborado pelo autor.

O resultado da execução de cada chamada da Figura 7.16 se encontra no Quadro 7.3.

Quadro 7.3 – Resultado SP com parâmetro de Output

Num	Chamada	Resultado		
1	EXECUTE pSomaSalario @Salario = @Total OUTPUT	 <p>(Nenhum nome de coluna)</p> <table border="1"> <tr> <td>1</td> <td>13849,00</td> </tr> </table>	1	13849,00
1	13849,00			
2	EXECUTE pSomaSalario 'F', @Salario = @Total OUTPUT	 <p>(Nenhum nome de coluna)</p> <table border="1"> <tr> <td>1</td> <td>5060,00</td> </tr> </table>	1	5060,00
1	5060,00			

Num	Chamada	Resultado		
3	<code>EXECUTE pSomaSalario @Salario = @Total OUTPUT, @Sexo = 'M'</code>	<table border="1"><tr><td>(Nenhum nome de coluna)</td></tr><tr><td>1 8789,00</td></tr></table>	(Nenhum nome de coluna)	1 8789,00
(Nenhum nome de coluna)				
1 8789,00				

Fonte: elaborado pelo autor.

Com parâmetros de OUTPUT, a flexibilidade da utilização das SPs fica maior ainda. Quanto mais opções o SGBD oferece, melhor é o trabalho de programadores e usuários que utilizam esses recursos.

7.3.4 Tipos de procedimentos armazenados

Existem três tipos de procedimentos armazenados no MS SQL Server:

- ✗ **usuário** – são procedimentos criados pelo usuário e podem ser criados nos bancos de dados existentes. Para criar qualquer objeto no banco de dados de sistema (catálogo), o usuário deve ter permissão.
- ✗ **temporário** – procedimentos temporários são procedimentos definidos pelo usuário. Geralmente as sps são armazenadas de forma permanente, exceto os procedimentos temporários que são armazenados no banco de dados especial, tempdb. Assim como as variáveis, existem dois tipos de procedimentos temporários: local e global. Os procedimentos temporários locais são representados por uma única # como primeiro caractere no nome, visíveis somente na sessão atual do usuário e excluídas quando a conexão é fechada. Os procedimentos temporários globais são representados por ## como os dois primeiros caracteres de seus nomes, que ficam visíveis para qualquer usuário depois de criados e são excluídos no fim da última sessão do procedimento.
- ✗ **sistema** – os procedimentos do sistema são fornecidos com o MS SQL Server. Como os procedimentos do sistema começam com o prefixo sp_, recomenda-se que utilize um padrão diferente ao criar as SPs de usuário.



Saiba mais



Para obter uma lista completa dos procedimentos do sistema, procure na internet por SQL Server Procedimentos armazenados do sistema.

7.3.5 Alterar, eliminar e verificar uma SP

As Stored Procedures ficam armazenadas de forma permanente no banco, enquanto o banco existir ou até que se elimine explicitamente, utilizando o comando Drop Procedure.

Sintaxe:

```
Drop Proc[edure] nome da procedure;
```

Caso se deseje alterar o conteúdo da SP, basta utiliza o comando alter, definindo os novos comandos da SP.

Sintaxe:

```
Alter PROC [ EDURE ] procedure_name
[ {@parametro tipo de dados }
  [ VARYING ] [ = default ] [ OUTPUT ] ] [
,...n ]
WITH
  { RECOMPILE | ENCRYPTION | RECOMPILE ,
ENCRYPTION } ]
[ FOR REPLICATION ]
AS Comandos_sql [ ...n ]
```

E quando se necessita verificar o conteúdo da SP, basta utilizar uma procedure de sistema chamada `sp_helptext`, colocando em seguida o nome da SP que se deseja visualizar o conteúdo (figura 7.17).

Síntese

Este capítulo é muito importante para a utilização dos recursos avançados de banco de dados. Por isso, foi apresentado o conceito de transação que garante as regras ACID, mantendo o banco consistente quando uma ou várias operações são efetuadas no banco de dados e quando várias transações são geradas simultaneamente.

Também foi apresentada a função do arquivo de log de transação, que auxilia a gerenciar as transações.

Foram exploradas funções de extensão da linguagem SQL (Transact-SQL), permitindo que sejam criados programas dentro do banco de dados e também foi visto que esses programas podem ser armazenados permanentemente no banco de dados com as Stored Procedures.

Da teoria para a prática

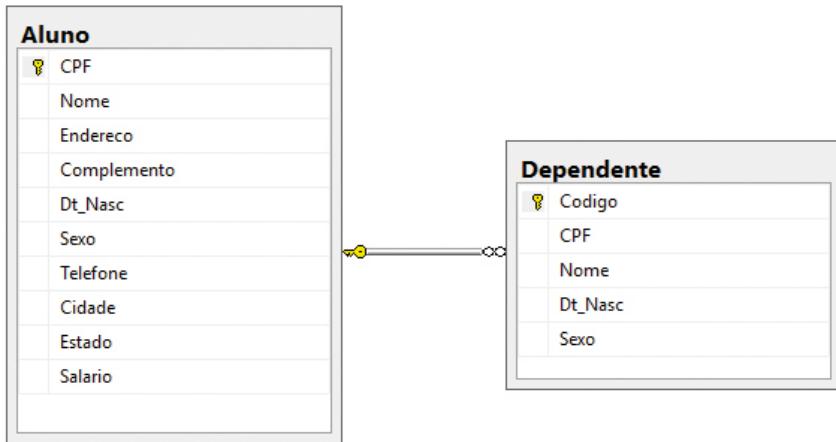
Dado o modelo a seguir, crie uma SP que:

- ✗ receba o CPF do aluno como parâmetro de entrada;
- ✗ delete o aluno que não tenha dependente;
- ✗ delete somente se o salário do aluno for maior ou igual a R\$ 10.000,00;
- ✗ utilize o conceito de transação.

Figura 7.17 – Verificando o conteúdo de uma SP

```
sp_helptext pSomaSalario
100 % <
Resultados Mensagens
Text
1
2 Create Procedure pSomaSalario
3   @sexo char(1) = '%',
4   @Salario smallmoney OUTPUT
5   as
6
7   Select Sum(Salario) from Aluno
8   Where Sexo like @Sexo
```

Fonte: elaborado pelo autor.



8

Recursos Avançados

NESTE CAPÍTULO VAMOS ver outros recursos oferecidos pelo MS SQL Server. A grande maioria desses recursos existe em outros SGBDs. A sintaxe varia, mas são muito importantes para utilização correta e otimizada de banco de dados.

Objetivos de aprendizagem:

- × apresentar as variáveis de ambiente;
- × cursores;
- × tabelas temporárias;
- × triggers.

8.1 Variáveis de Ambiente

Cada SGBD oferece um conjunto de variáveis do ambiente para que se possa utilizar e verificar situações do sistema. No caso do MS SQL Server, basta utilizar o comando SET ou SELECT, antes da variável, para obter o resultado. Relembrando que as variáveis locais são precedidas por @ e as variáveis globais são precedidas por @@.

Eis algumas das principais variáveis de ambiente (MSDN, 2016):

- × @@DATEFIRST – define o primeiro dia da semana, para os EUA. Seu valor é 7, considerado o primeiro dia;
- × @@SERVERNAME – indica qual o nome do servidor/instância que está rodando o SQL;
- × @@LANGUAGE – indica qual a linguagem está sendo executada atualmente no SQL;
- × @@LOCK_TIMEOUT – mostra configuração de tempo limite do bloqueio atual em milissegundos para a sessão corrente;
- × @@SPID – mostra o ID de sessão do processo de usuário atual;
- × @@VERSION – é uma função muito importante, pois retorna muitas variáveis de ambientes de configuração do sistema, apresentado na figura 8.1:
- × versão do SQL Server;
- × arquitetura do processador;
- × data de compilação do SQL Server;
- × instrução de direitos autorais;

- ✗ edição do SQL Server;
- ✗ versão do sistema operacional.

Figura 8.1 – @@Version

```
Microsoft SQL Server 2014 - 12.0.4213.0 (X64)
Jun 9 2015 12:06:16
Copyright (c) Microsoft Corporation
Enterprise Edition (64-bit) on Windows NT 6.3 <X64> (Build 10586: ) (Hypervisor)

(1 linha(s) afetadas)
```

Fonte: elaborado pelo autor.

Importante

Existe um comando que é utilizado para matar (suspending a execução) processos que estão processando ou travados e por algum motivo não há mais necessidade de continuar sua execução, este é o comando KILL. Esta instrução poderá demorar algum tempo para ser concluída, principalmente quando envolver um rollback de uma transação longa.

O comando KILL pode ser usado para terminar uma conexão normal, o que finaliza internamente as transações associadas à ID (pode ser identificado usando @@spid, ou a SP de sistema sp_who2) da sessão especificada.

8.2 Cursosres

Existe uma estrutura especial que permite retornar um result set e trabalhar com esse conjunto, linha a linha, para fazer várias ações dentro de um T-SQL ou de uma SP.

Sintaxe:

```
DECLARE nome_cursor [ INSENSITIVE ] [  
SCROLL ] CURSOR  
FOR Comando_select  
[ FOR { READ ONLY | UPDATE [ OF nome_da  
coluna [ ,...n ] ] } ]  
[ ; ]
```

Explicando a sintaxe, tem-se (MSDN, 2016):

- ✗ nome_Cursor – é o nome do cursor que vai ser utilizado, deve seguir o mesmo padrão de nomenclatura de variáveis.
- ✗ INSENSITIVE – esse cursor faz uma cópia temporária no tempdb dos dados a serem usados por ele e as modificações feitas posteriormente na tabela não são refletidas nos dados desse cursor.
- ✗ SCROLL – o cursor tem um ponteiro que faz a movimentação entre as linhas. As opções podem ser: FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE. Se SCROLL não for especificado, NEXT será a única opção de busca com suporte.
- ✗ SELECT_STATEMENT – é uma instrução SELECT padrão que define o conjunto de resultados de um cursor. Podem-se usar as funções de agregação e criar SQL com cláusula *where* que faça comparações com variáveis.
- ✗ READ ONLY – previne atualizações feitas por esse cursor. Essa opção anula a funcionalidade padrão de um cursor para ser atualizado.
- ✗ UPDATE [OF column_name [...n]] – define colunas atualizáveis em um cursor. Se OF column_name [...] for especificado, somente as colunas listadas permitirão modificações. Se a UPDATE for especificada sem uma lista de colunas, todas as colunas poderão ser atualizadas.

Agora que foi apresentada a sintaxe do cursor, tem-se mais uma ferramenta para auxiliar no seu trabalho:

- ✗ open cursor – utilizado para executar o *select* que foi definido dentro do cursor.

Sintaxe:

```
Open Cursor nomeCursor
```

- ✗ fetch – utilizado para navegar entre as linhas do cursor e recuperar valores do *result set* para variáveis.

Sintaxe:

```

FETCH
[ [ NEXT | PRIOR | FIRST | LAST
| ABSOLUTE { n | @nvar }
| RELATIVE { n | @nvar }
]
FROM
]
{ { cursor_name } | @cursor_variable_name
}
[ INTO @variable_name [ ,...n ] ]

```

- ✗ NEXT – retorna a linha de resultado imediatamente seguinte à atual e adiciona a linha atual à retornada. Se FETCH NEXT for a primeira busca de um cursor, a primeira linha do conjunto de resultados será retornada. NEXT é a opção padrão de busca de cursor.
- ✗ PRIOR – retorna a linha de resultado imediatamente anterior à atual e decremente a linha atual da retornada. Se FETCH PRIOR for a primeira busca de um cursor, nenhuma linha será retornada e o cursor será deixado posicionado antes da primeira.

- ✗ FIRST – retorna a primeira linha no cursor e a torna a linha atual.
- ✗ LAST – retorna a última linha no cursor e a torna a linha atual.
- ✗ ABSOLUTE { n| @nvar} – se n ou @nvar for positivo, a linha que está n linhas a partir do início do cursor será retornada e a linha retornada se tornará a nova linha atual. Se n ou @nvar for negativo, a linha que está n linhas antes do final do cursor será retornada e a linha retornada se tornará a nova linha atual. Se n ou @nvar for 0, nenhuma linha será retornada. N deve ser uma constante de número inteiro e @nvar deve ser smallint, tinyint ou int.
- ✗ RELATIVE { n| @nvar} – se n ou @nvar for positivo, a linha que está n linhas após a linha atual será retornada e a linha retornada se tornará a nova linha atual. Se n ou @nvar for negativo, a linha que está n linhas antes da linha atual será retornada e a linha retornada se tornará a nova linha atual. Se n ou @nvar for 0, a linha atual será retornada. Se FETCH RELATIVE for especificado com n ou @nvar definido como números negativos ou 0 na primeira busca feita em um cursor, nenhuma linha será retornada. N deve ser uma constante de número inteiro e @nvar deve ser smallint, tinyint ou int.
- ✗ cursor_name – é o nome do cursor aberto a partir do qual a busca deve ser feita. Se um cursor global e um cursor local existirem com cursor_name como seu nome, cursor_name será o cursor global, se GLOBAL for especificado, e será o cursor local, se GLOBAL não for especificado.
- ✗ @cursor_variable_name – é o nome de uma variável de cursor que faz referência ao cursor aberto a partir do qual a busca deve ser feita.
- ✗ INTO @variable_name[,...n] – permite que os dados das colunas de uma busca sejam colocados em variáveis locais. Cada variável na lista, da esquerda para a direita, está associada à coluna correspondente no conjunto de resultados do cursor. O tipo de dados de cada variável deve corresponder ou ser uma conversão implícita com suporte do tipo de dados da coluna do conjunto de resultados correspondente. O número de variáveis deve corresponder ao número de colunas na lista de seleção do cursor (MSDN, 2016).

Título do box: Tabelas temporárias no TempDb

Existe uma variável para validar a ação do Fetch, @@Fetch_Status. Ela indica se o fetch foi executado com sucesso ou não. Os possíveis valores para @@Fetch_Status são:

- × 0: O FETCH foi realizado com sucesso;
- × -1: O FETCH falhou;
- × -2: O registro trazido foi perdido.
- × close – utilizado para fechar um cursor.

Sintaxe:

```
Close nomeCursor
```

- × deallocate – utilizado para liberar os recursos alocados pelo cursor.

Sintaxe:

```
Deallocate: nomeCursor
```

O exemplo da figura 8.2 mostra a utilização de cada um dos itens apresentados.

Figura 8.2 – Cursor

```
CreateProcedure pLeAluno
@Nome varchar(70)='%'
as
Declare @idade smallint,
@Data date,
@DataAtual datetime
Set @dataAtual =Getdate()
-- Declaração do Cursor
Declare cTeste cursorfor
```

Banco de Dados

```
Select Nome, Dt_Nasc
From Aluno
Where Nome like @nome
Orderby Nome

-- Abrir o cursor, executar o select
Open cTeste;
-- Verifica error na abertura do cursor
If @@error!= 0
Begin
Print' Erro na abertura do cursor!'
return
End

Fetchnextfrom cTeste into @nome, @data

While @@FETCH_STATUS= 0
Begin

Print @nome +' Idade
'+Convert(char(5), Datediff(yy, @Data, @DataAtual))
Fetchnextfrom cTeste into @nome, @data

End

Close cTeste
Deallocate cTeste
```

Fonte: elaborado pelo autor.

O objetivo desta SP é mostrar o uso do Cursor. A SP recebe o nome de um Aluno como parâmetro de entrada e caso não seja enviado nada assume o % por default. É criado um cursor que faz um select utilizando o parâmetro

como condição na cláusula where; logo depois é feito o Open Cursor que executa o comando que foi declarado dentro do cursor. É neste momento que o select é realmente executado. A variável @@Error é validada, para garantir que a execução do cursor ocorreu corretamente, quando aberto. O ponteiro aponta para o primeiro registro do result set, na sequência é feito um fetch que recupera os dados do result set da linha apontada naquele momento e atribui o valor de cada coluna em cada variável, seguindo a ordem do select, ou seja, de acordo com a posição das colunas devem estar as variáveis que vão recebê-las. Após associar as variáveis com o conteúdo da coluna o fetch NEXT, posiciona o ponteiro do cursor para o próximo registro do result set.

No while é verificado o valor de @@Fetch_status. Enquanto for = 0, os registros estão sendo recuperados com sucesso do result set. Dentro do while tem-se o Print do nome do aluno, seguido da idade (que é calculada a partir da subtração da data atual, recuperada pelo getdate), pela data de nascimento. Um novo Fetch é feito dentro do while para ir recuperando as demais linhas do result set. Quando chegar ao último registro, na próxima leitura o @@fetch_status passa a ser -1, ocasionando a saída do while. Depois o cursor é fechado, e todos os recursos são liberados com o Deallocate.

A figura 8.3 mostra a execução da SP, sem passar parâmetro e traz o resultado da execução.

Resultado:

Figura 8.3 – Cursor

The screenshot shows the results of executing the stored procedure pLeAluno. The results are displayed in a grid format with columns for Name and Age. The data is as follows:

Name	Age
Diego Oliveira	35
Joana Souza	19
João Pereira	26
José da Silva	53
Marco Antônio Alvarenga	42
Maria da Silva	27
Monica Santos	21

Fonte: elaborado pelo autor.

Você pode utilizar um cursor, com conceitos de transação, com selects mais complexos, dentro e fora de SPs. As aplicações são inúmeras, acrescentando muitas funções, que podem ser executadas pelo banco de dados, para atender às necessidades das aplicações. O exemplo apresentado utilizou somente NEXT, para pesquisar linha a linha do cursor, mas você pode utilizar o PRIOR, LAST, FIRST, conceito de posição: absoluto ou relativo de acordo com a posição do cursor, para acessar os dados.

8.3 Tabelas temporárias

A criação de tabelas temporárias é muito útil em situações em que se precisa utilizar uma tabela por um determinado tempo e depois ela pode ser dropada (eliminada).

Tabelas temporárias são criadas em um banco especial chamado tempdb, que armazena os dados temporários de uma instância SQL.

Para criar uma tabela com visualização local, utilize o # no início do nome da tabela. Para criar a tabela com visualização global, utilize ##, em vez de #.

Sintaxe:

```
Create Table #nomeTabela
  (Atributo Tipo de dado Constraint pk_nome-
  DaConstraint Primary key,
   Atributo2 Tipo
  );
-- OU
Create Table ##nomeTabela
  (Atributo Tipo de dado Constraint pk_nome-
  DaConstraint Primary key,
   Atributo2 Tipo
  );
```

A cada exemplo mostramos conceitos mais complexos para ampliar o conhecimento. Na figura 8.4 temos primeiramente a introdução ao conceito de select into. Este comando permite que seja inserido o resultado de um select em uma tabela comum, se for uma tabela temporária não precisa ser previamente criada; se for uma tabela temporária não precisa ser previamente criada (neste caso específico). Logo após é feito um select na tabela temporária para mostrar que o conteúdo da tabela Aluno realmente foi inserido na tabela #tempTable. E no fim do T-SQL, a tabela foi explicitamente eliminada.

Figura 8.4 – Tabela temporária

```

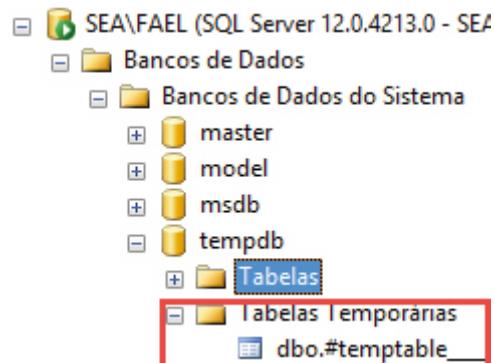
Select * into #temptable from Aluno
Select * from #temptable
Drop table #temptable
  
```

The screenshot shows a SQL query window with three statements. The first statement creates a temporary table #temptable by selecting all columns from the Aluno table. A callout bubble points to this statement with the text "Inserindo os valores da tabela aluno, diretamente na tabela temporária". The second statement selects all data from the newly created #temptable table. The third statement drops the #temptable table. Below the query window, the results pane displays a table with 7 rows of student data. The columns are: CPF, Nome, Endereço, Complemento, Dt_Nasc, Sexo, Telefone, Cidade, Estado, and Salário.

	CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Cidade	Estado	Salário
1	11	José da Silva	Rua dos Guarapeas, 37, Bairro: Centro	NULL	1963-12-07	M	NULL	RJ	RJ	1320,00
2	22	Maria da Silva	Rua do Chapéu, 455	Ap 12	1989-02-02	F	NULL	Manaus	AM	2860,00
3	33	Joana Souza	Rua das Memórias, 137	Ap 43	1997-12-05	F	NULL	Salvador	BA	NULL
4	44	João Pereira	Rua Rui Barbosa, 24	NULL	1980-10-15	M	NULL	Ponto Alegre	RS	1749,00
5	55	Marco Antônio Alverenga	Av XV de Novembro, 1793	NULL	1974-11-11	M	NULL	Brasília	DF	4070,00
6	66	Monica Santos	Este é o endereço	Ap 37	1995-06-23	F	NULL	Belo Horizonte	MG	2200,00
7	77	Diego Oliveira	Rua das Flores	NULL	1981-05-15	M	NULL	Macedô	AL	1550,00

Fonte: elaborado pelo autor.

Figura 8.5 - Tabelas temporárias no TempDb



Observe que a tabela temporária aparece no banco de dados TEMPDB, até que seja dropada, ou a sessão eliminada.

Fonte: elaborado pelo autor.

Podem-se criar quantas tabelas temporárias foram necessárias. Você pode criar explicitamente, utilizando o comando `create table`, da mesma forma que é criada uma tabela normal.

8.4 Triggers

Uma trigger, também conhecida como gatilho, é um tipo especial de procedimento armazenado, que é disparado automaticamente quando uma determinada ação ocorre no servidor de banco de dados.

Triggers são estruturas simples para criação, mas pode conter códigos complexos. Portanto, deve-se analisar bem a solução criada.

Existem dois tipos de triggers:

- ✗ Data Manipulation Language (DML);
- ✗ Data Definition Language (DDL).

Triggers DML são executadas quando um usuário tenta modificar dados por meio de comandos de `INSERT`, `UPDATE` ou `DELETE` em uma tabela ou view. Elas são disparadas quando qualquer evento válido é acionado, e não depende de quais linhas da tabela serão afetadas ou não.

Sintaxe:

```
CREATE TRIGGER nome_trigger
  ON { tabela | view }
  [ WITH <dml_trigger> [ ,...n ] ]
  { FOR | AFTER | INSTEAD OF }
  { [ INSERT ] [ , ] [ UPDATE ] [ , ] [
    DELETE ] }
  [ NOT FOR REPLICATION ]
AS {Comando SQL [ ; ] [ ,...n ]}
```

Triggers DDL são executadas em resposta a diversos comandos de CREATE, ALTER e DROP. Existem também triggers de logon executadas em resposta ao evento LOGON que é gerado quando as sessões de um usuário estão sendo estabelecidas.

Sintaxe:

```
CREATE TRIGGER nome_trigger
    ON { ALL SERVER | DATABASE }
        [ WITH <ddl_trigger> [ ,...n ] ]
        { FOR | AFTER } { tipo do evento | grupo
de eventos } [ ,...n ]
    AS { sql_statement[ ; ] [ ,...n ] }
```

Explicação da sintaxe (MSDN, 2016):

- ✗ nome_trigger – é o nome da trigger que vai ser utilizada, deve seguir o mesmo padrão de nomenclatura de variáveis;
- ✗ tabela ou view – é a tabela ou view onde DML é executada. Uma view só pode ser referenciada por uma trigger INSTEAD OF. Triggers DML não podem ser definidas para tabelas temporárias locais ou globais;
- ✗ database – aplica o escopo de uma trigger DDL ao banco de dados atual. Se for especificada, a trigger será acionada sempre que event_type ou event_group ocorrer no banco de dados atual;
- ✗ FOR ou AFTER – AFTER especifica que a trigger DML é disparada (iniciada) apenas quando todas as operações especificadas na instrução SQL de trigger são executadas com êxito. Todas as verificações de restrição devem finalizar com sucesso para que esta trigger seja disparada. AFTER é o padrão quando FOR é a única palavra-chave especificada. Triggers AFTER não podem ser definidas em views;

- ✗ instead of – especifica que a trigger DML será executada em vez da instrução SQL de trigger, substituindo as ações das instruções de trigger. INSTEAD OF não pode ser especificado para triggers DDL ou de logon. No máximo, uma trigger INSTEAD OF pode existir por instrução INSERT, UPDATE ou DELETE;
- ✗ { [DELETE] [,] [INSERT] [,] [UPDATE] } – especifica quais são as instruções de modificação de dados que, disparam a trigger DML. É necessário especificar pelo menos uma opção. É permitida qualquer combinação dessas opções em qualquer ordem na definição da trigger.

Para exemplificar (figura 8.6) o uso da trigger, criou-se um exemplo que grava o histórico das ações na tabela Aluno, para uma tabela AlunoHist, que além de todos os campos da tabela Aluno tem também a ação que foi feita (insert, delete ou update) e a data e hora em que foi modificada.

Logo no início temos o nome da trigger como traudit, aplicada na tabela aluno para insert, delete ou update. Observe que o comando está dentro do quadro vermelho **exists**. Este comando serve para verificar se há a existência de algum registro em determinada tabela. Ele pode ser utilizado em vários lugares de T-SQL, não somente entre triggers. O MS SQL Server trata a inserção, deleção e atualização de uma forma diferenciada. Ele cria tabelas temporárias importantes. A tabela inserted vai ter o conteúdo exato dos dados que vão ser inseridos na nova tabela. Caso esses dados não quebrem nenhuma regra de integridade referencial ou de restrições, o SQL pega os dados da tabela inserted e efetiva, neste caso, na tabela aluno. Temos a tabela deleted que faz exatamente a mesma coisa, mas, quando os dados estão sendo deletados de uma tabela. Então, primeiramente o SQL SERVER grava esses dados numa tabela temporária, que tem a mesma estrutura da tabela que está sendo modificada e se a deleção não ferir nenhuma regra de integridade ou restrição, o delete é efetivado.

Já para o update, o tratamento é diferente, pois o SQL server pega os dados correntes da tabela aluno, insere na tabela deleted, e os novos dados são inseridos na tabela inserted. Caso o update não quebre nenhuma regra de integridade referencial ou restrição, é feito um delete na tabela daquele registro e o insert do novo registro.

Figura 8.6 – Trigger

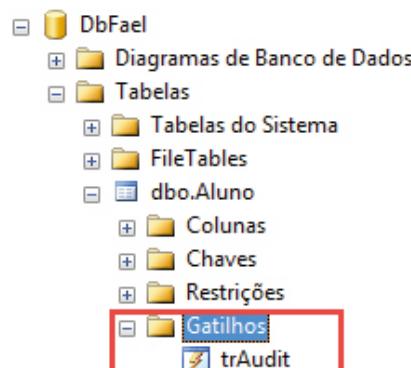
```

Create trigger trAudit on Aluno
for insert, delete, update
as
if EXISTS (SELECT * FROM INSERTED) AND NOT EXISTS (SELECT * FROM DELETED)
begin
    Insert into AlunoHist
    Select *, I as 'acao', getdate() as DataAteracao
    from Inserted
    Print 'Insert'
end
Else
Begin
    IF NOT EXISTS (SELECT * FROM INSERTED) AND EXISTS (SELECT * FROM DELETED)
begin
    Insert into AlunoHist
    Select *, D as 'acao', getdate() as DataAteracao
    from Deleted
    Print 'Delete'
End
Else
Begin
    Insert into AlunoHist
    Select *, 'U' as 'acao', getdate() as DataAteracao
    from Deleted
    Print 'Update'
end
End

```

Fonte: elaborado pelo autor.

Figura 8.7 – Mostrar Trigger



Ao criar a trigger com sucesso, ela aparece na interface gráfica, na opção triggers ou gatilhos, depende da linguagem com a qual o SQL Server foi instalado.

Fonte: elaborado pelo autor.

Banco de Dados

A figura 8.8 mostra a execução do delete e o disparo da trigger.

Figura 8.8 – Execução da trigger

```
Delete
From Aluno
Where CPF = 22|
```

100 % < Mensagens

(1 linha(s) afetadas)

Delete

(1 linha(s) afetadas)

Fonte: elaborado pelo autor.

Observe que, na figura 8.9, o registro com CPF 22 foi deletado.

Figura 8.9 – Registro deletado

	CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Cidade	Estado	Salário
1	11	José da Silva	Rua dos Guarapezes, 37, Bairro: Centro	NULL	1963-12-07	M	NULL	RI	RI	1325,00
2	33	Joana Souza	Rua das Memórias, 137	Ap 43	1997-12-05	F	NULL	Salvador	BA	NULL
3	44	João Peixoto	Rua Rui Barbosa, 24	NULL	1990-10-15	M	NULL	Ponta Alegre	RS	1749,00
4	55	Marco Antônio Alves	Av XV de Novembro, 1733	NULL	1974-11-11	M	NULL	Brasília	DF	4070,00
5	66	Monica Santos	Este é o endereço	Ap 37	1995-06-23	F	NULL	Belo Horizonte	MG	2200,00
6	77	Diego Oliveira	Rua das Flores	NULL	1981-05-15	M	NULL	Maceió	AL	1650,00

Fonte: elaborado pelo autor.

Enquanto na figura 8.10, ele foi inserido na tabela AlunoHist, com Ação D e a data de alteração.

Figura 8.10 – Registro inserido

	CPF	Nome	Endereço	Complemento	Dt_Nasc	Sexo	Telefone	Cidade	Estado	Salário	Ação	DataAlteração
1	22	Maria da Silva	Rua do Chapéu, 455	Ap 12	1969-02-02	F	NULL	Manaus	AM	2860,00	D	2016-08-30 21:50:01,173

Fonte: elaborado pelo autor.

Para alterar uma trigger, usa-se o comando a seguir.

Sintaxe:

```
ALTER TRIGGER nome_trigger
  ON { tabela | view }
  [ WITH <dml_trigger> [ ,...n ] ]
  { FOR | AFTER | INSTEAD OF }
  { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
  [ NOT FOR REPLICATION ]
  AS {Comando SQL [ ; ] [ ,...n ]}
```

Para eliminar uma trigger, utiliza-se o comando a seguir.

Sintaxe:

```
Drop Trigger Nome da trigger.
```

Triggers podem parecer complexas e difíceis de trabalhar, mas com a prática e muitos exercícios, você consegue dominar essa técnica e elaborar grandes soluções.

Síntese

Este capítulo apresenta vários recursos avançados, como utilização de funções de sistema e outras técnicas que permitem a criação de programas mais complexos como cursor que permitem o acesso e manipulação linha a linha de cada registro e atributo. Também a utilização de tabelas temporárias que são muito úteis em situação que se precisa de armazenamento volátil dos dados para auxiliar nos cálculos e ações. E finalizando com a utilização de triggers que permite geração de relatórios, proteção de dados, armazenamento de histórico para futuras auditorias.

Da teoria para a prática

Com a estrutura da tabela *Aluno* a seguir, faça o que se pede.

			dbo.Aluno
			Colunas
			CPF (PK, numeric(11,0), não nulo)
			Nome (varchar(100), não nulo)
			Endereco (varchar(100), não nulo)
			Complemento (varchar(100), nulo)
			Dt_Nasc (date, não nulo)
			Sexo (char(1), não nulo)
			Telefone (char(15), nulo)
			Cidade (varchar(50), não nulo)
			Estado (char(2), não nulo)
			Salario (smallmoney, nulo)

- × Crie a tabela adicionando os campos Ação (I, D, U) – Insert, Delete, Update e da Data e hora da modificação, conforme a tabela a seguir.

			dbo.AlunoHist
			Colunas
			CPF (numeric(11,0), não nulo)
			Nome (varchar(100), não nulo)
			Endereco (varchar(100), não nulo)
			Complemento (varchar(100), nulo)
			Dt_Nasc (date, não nulo)
			Sexo (char(1), não nulo)
			Telefone (char(15), nulo)
			Cidade (varchar(50), não nulo)
			Estado (char(2), não nulo)
			Salario (smallmoney, nulo)
			Acao (char(1), nulo)
			DataAlteracao (datetime, nulo)

- × Crie uma ou mais triggers que, para cada ação, gravem o registro do aluno, a ação que está sendo executada e a data e hora da modificação:
- × Crie para Insert, delete e update;
- × Apresente uma alternativa de solução diferente da exposta como exemplo no livro; teste a trigger para cada ação.

9

Backup, Restore, Segurança e Otimização

ENTRE TODAS AS ações, tarefas e planejamento que devem ser feitos, a mais importante de todas é o backup (cópia de segurança). Por que é tão importante assim? A importância de um backup bem feito e seguro tem a mesma importância que o seu banco de dados e todas as informações que ele contém, afinal, caso algum problema ocorra, seja ele físico ou lógico, existe a possibilidade de recuperação. Tão importante quanto o backup é a capacidade de restaurá-lo, pois de nada serve um backup se você não consegue recuperar as informações.

Outra tarefa importante é a criação de logins e usuários. É preciso dar privilégios necessários a cada um deles, de forma adequada.

E, finalizando este capítulo, veremos uma introdução à otimização de *queries*.

Objetivos de aprendizagem:

- ✗ apresentar backup e restore;
- ✗ mostrar criação, manutenção de ids e privilégios;
- ✗ otimização de comandos SQL.

9.1 Backup

O que aconteceria se, por algum motivo, o banco de dados fosse perdido? Qual o impacto que ele teria para a empresa?

Falhas podem acontecer e deve-se estar preparado para recuperar os dados, caso esses problemas aconteçam. Os principais tipos de falha são:

- ✗ lógicas – erros de usuário, sejam propositais ou não, que removem ou alteram dados ou estruturas;
- ✗ sistema operacional ou instância do banco corrompida;
- ✗ físicas – problemas no hardware, em alguma peça do servidor ou *storage* (armazenamento), ou até mesmo *crash* (dano completo) de um servidor;
- ✗ desastres naturais ou artificiais – incêndios, alagamentos.

O backup oferece uma proteção fundamental, para registros críticos dos bancos de dados. Para minimizar o risco de perda, deve ser feito backup dos bancos, para manter salvas as ações frequentes nos dados, cuja melhor forma de recuperação, em caso de perda, é uma estratégia de backup e restauração bem planejada.

Um backup é a geração de um ou mais arquivos físicos que podem ser armazenados em discos, fitas, e outros dispositivos. Esse arquivo contém os registros do banco de dados, de forma que possam ser recuperados posteriormente.

Pela disponibilidade pode-se classificar o banco de dados como:

- ✗ *online* – é executado com o Banco de dados online, com pequeno impacto aos usuários que estão utilizando o banco no momento;
- ✗ *offline* – neste caso é feita uma cópia física, de todos os arquivos que compõem o banco de dados. Para isso o serviço do banco tem que ser parado, de forma que o sistema operacional permita que a cópia seja feita.

Os tipos de backup são:

- ✗ completo (*full*) – todos os dados existentes no bando de dados no momento do backup, incluindo todas as entradas do log de transações, são gravados no arquivo de backup;
- ✗ diferencial (*incremental*) – todos os dados diferentes, ou seja, que sofreram alterações desde o último backup completo, são gravados no arquivo de backup;
- ✗ log de transação – os dados da log de transação são gravados no arquivo de backup.

Importante

O início de qualquer estratégia dever ser sempre com o backup completo. Para usar o incremental ou o backup de log, deve existir um backup completo.

Para montar uma estratégia de backup precisa-se calcular:

- ✗ **qual período do dia os aplicativos precisam acessar o banco de dados.**

Essa informação é importante para escolher o melhor horário para agendar o backup do banco de dados, porque, dependendo do tamanho do banco, isso pode durar horas e, às vezes, dias, para terminar o backup completo. Deve-se agendar um horário em que a utilização do banco seja muito pequena, para que o impacto seja o menor possível na concorrência entre o backup e os usuários pelo uso do banco, e para que termine mais rápido.

- ✗ **com que frequência as alterações e atualizações no banco deverão ocorrer.**

É importante verificar, em caso de recuperação por backup, qual o tempo de dados perdidos a empresa pode suportar. Por exemplo, caso as alterações sejam raras, pode-se agendar um backup de log a cada hora, ou, se forem frequentes, geralmente a agenda fica para fazer o backup de log a cada 5 minutos, já que, em caso de necessidade de recuperar dados do backup, somente os últimos cinco minutos de dados são perdidos. Também, pode-se intercalar com backup incremental.

- ✗ **qual o tamanho de um backup completo de banco de dados.**

É importante saber qual a capacidade de armazenamento dos arquivos de backup, se eles irão para disco ou fita, e qual a retenção destes arquivos (dias, semanas, meses).

Com essas informações, já se deve criar a estratégia de backup, por exemplo: um backup completo aos domingos, ao meio-dia; backups incrementais: terça e quinta, à meia noite, e backups de log a cada quinze minutos; backup completo diariamente à meia noite, backup de log a cada 10 minutos.

Saiba mais

Existem bancos de dados que devem estar online 24 horas por dia, para isso, além do backup, existem soluções de alta disponibilidade.



O backup completo segue a seguinte sintaxe:

```
BACKUP DATABASE nome_bancodedados  
TO backup_device [ ,...n ]  
[ WITH com_opções[ ,...o ] ] ;
```

O backup incremental segue a seguinte sintaxe:

```
BACKUP DATABASE nome_bancodedados  
TO <backup_device>WITH DIFFERENTIAL;
```

O backup incremental segue a seguinte sintaxe:

```
BACKUP Lognome_bancodedados  
TO backup_device[ ,...n ]  
[ WITH com_opções[ ,...o ] ] ;
```

Recomendam-se extensões para cada um dos tipos de backup, porém não são obrigatórias, e sim sugestões de padrão:

- ✗ completo – BAK;
- ✗ incremental – INC;
- ✗ LOG – TRN.

Outro ponto importante é identificar os backups com data e hora, assim tem-se o tempo exato em que foi criado.

O SGBD precisa ser informado quais são os tipos de backup que vão ser feitos. Isso é indicado no atributo do banco de dados, chamado de modelo de recuperação (*recovery model*). As operações de backup e restauração ocorrem dentro desse contexto que controla a forma de gerenciamento do log de transações. O modelo de recuperação de um banco de dados indica para quais tipos de backups e cenários de restauração o banco de dados oferece suporte. É utilizado o modelo de recuperação simples para backups completos ou o modelo de recuperação completa para backups de log de transação.

Importante

O modelo de recuperação deve ser selecionado no banco de dados antes de começar qualquer estratégia de backup e restore, pois só funcionarão corretamente depois da solução escolhida.

Banco de Dados

Sintaxe para backup somente completos:

```
USE master ;  
ALTER DATABASE NomeBanco SET RECOVERY SIMPLE ;
```

Sintaxe para backup completo e de log:

```
USE master ;  
ALTER DATABASE NomeBanco SET RECOVERY FULL ;
```

A figura 9.1 apresenta o exemplo de um backup completo do banco de dados dbFael. Esse backup está armazenado no <c:\Fael\DbFael.bak>, as opções de backup apresentadas estão logo depois do WITH, Compression é uma opção para compactar o backup usando o sistema de compactação do SQL Server e, STATS indica o percentual de conclusão. Neste caso, mostrar a mensagem a cada cinco por certo completo.

Figura 9.1 – Backup completo

The screenshot shows a SQL Server Management Studio (SSMS) interface. A query window is open with the following T-SQL command:

```
backup database [DbFael] to disk = 'c:\Fael\DbFael.bak' with compression, Stats = 5
```

The status bar at the bottom indicates the command is executing, with progress messages:

100 % ->

Mensagens

6 por cento processado(s).
10 por cento processado(s).
15 por cento processado(s).
21 por cento processado(s).
25 por cento processado(s).
30 por cento processado(s).
36 por cento processado(s).
40 por cento processado(s).
45 por cento processado(s).
51 por cento processado(s).
55 por cento processado(s).
60 por cento processado(s).
66 por cento processado(s).
70 por cento processado(s).
75 por cento processado(s).
81 por cento processado(s).
85 por cento processado(s).
90 por cento processado(s).
96 por cento processado(s).
Processadas 368 páginas para o banco de dados 'DbFael', arquivo 'DbFael' no arquivo 1.
100 por cento processado(s).
Processadas 6 páginas para o banco de dados 'DbFael', arquivo 'DbFael_log' no arquivo 1.
BACKUP DATABASE processou com êxito 374 páginas em 0.558 segundos (5.225 MB/s).

Fonte: elaborado pelo autor.

Observe que o arquivo foi gerado no caminho (path) especificado (figura 9.2).

Figura 9.2 – Arquivo de backup

	Nome	Data de modificação	Tipo	Tamanho
	DbFael.bak	01/09/16 01:33	Arquivo BAK	497 KB

Fonte: elaborado pelo autor.

Importante

Banco de dados e os backups devem estar em dispositivos separados, pois se o dispositivo que contém o banco de dados falhar, seus backups ficarão indisponíveis.

A figura 9.3 demonstra como apagar alguns registros.

Figura 9.3 – Alterando o conteúdo do banco

```

Delete from Aluno
Where CPF not in (Select CPF from Dependente)

```

(3 linha(s) afetadas)

Delete

(3 linha(s) afetadas)

Fonte: elaborado pelo autor.

Agora fazendo backup de log (figura 9.4), o Stats foi alterado para 20.

Banco de Dados

Figura 9.4 - Backup de log de transação

```
backup log [DbFael] to disk = 'c:\Fael\DbFael.trn' with compression, Stats = 20
```

100 % <

Mensagens

27 por cento processado(s).
40 por cento processado(s).
67 por cento processado(s).
81 por cento processado(s).
100 por cento processado(s).
Processadas 59 páginas para o banco de dados 'DbFael', arquivo 'DbFael_log' no arquivo 1.
BACKUP LOG processou com êxito 59 páginas em 0.129 segundos (3.569 MB/s).

Fonte: elaborado pelo autor.

Os dois arquivos de backup se encontram no diretório indicado no comando, como mostra a figura 9.5.

Figura 9.5 – Arquivos de backup

	Nome	Data de modificação	Tipo	Tamanho
WS-BT	DbFael.bak	01/09/16 01:33	Arquivo BAK	497 KB
rs-W5	DbFael.trn	01/09/16 01:48	SQL Server Transa...	121 KB

Fonte: elaborado pelo autor.

Com os backups prontos, agora deve-se compreender como funciona o processo de restauração.

9.2 Restore (recuperação do backup)

O restore é um processo com várias fases que copia todos os dados e páginas de log de um backup do SQL Server, para um banco de dados especificado e, logo após, executa rollforward de todas as transações registradas no backup de log, aplicando as alterações registradas (TECHNET, 2016).

Assim como no backup, tudo se inicia no backup completo, o restore funciona da mesma maneira.

Quando ocorrer um problema que precise restaurar o backup qual a forma correta de fazer?

Primeiro verifique qual o backup completo que existe antes de o problema acontecer. Esse é o backup que vai ser restaurado.

Depois restaure os backups incrementais, se existirem, do mais antigo para o mais recente.

Por último restaure todos os backups de log, do mais antigo ao mais novo, até o horário anterior ao horário em que ocorreu o problema.

Saiba mais

Durante o processo de restore, o banco não fica disponível para utilização pelo usuário. Somente quando o SBDB executar completamente todas as fases do restore, o banco ficará disponível para utilização.

Os SGBDs controlam a ordem em que o backup foi feito, para garantir que o restore siga essa ordem e não haja falhas. O SQL Server usa o conceito de LSN (número de sequência de log) para definir o ponto de recuperação para uma operação de restore.

Os LSNs são usados internamente durante uma sequência RESTORE para localizar o *point-in-time* para o qual os dados foram restaurados. Quando um backup é restaurado, os dados são restaurados ao LSN que corresponde ao *point-in-time* em que o backup foi realizado. O backup diferencial e o backup de log avançam o banco de dados restaurado para uma hora posterior que corresponde a um LSN mais alto. Cada registro do log de transações é identificado de forma exclusiva por um LSN (número da sequência de log). Os LSNs são ordenados de tal modo que se LSN2 for maior do que LSN1, a alteração descrita pelo registro de log mencionado por LSN2 ocorreu depois da alteração descrita pelo registro de log LSN (MSDN, 2016b).

Se houver quebra de LSN, algum arquivo de backup faltando ou corrompido, o restore não poderá prosseguir para os próximos backup files, portanto seja cuidadoso em relação ao ambiente de armazenamento dos files de backup.

Sintaxe:

```
RESTORE DATABASE NomeDatabase
FROM Disk| Tape = '\\Caminho\\nomebackup
[WITH MOVE 'nome logico' TO '\\\\Caminho\\
nomefile',
REPLACE,
RECOVERY | NORECOVERY,
STATS]
```

Pontos importantes em relação à sintaxe:

- ✗ se o banco de dados já existir, utilize a opção REPLACE para sobrescrever;
- ✗ se precisar mudar os arquivos do banco de lugar, use MOVE para cada arquivo, colocando o nome lógico do file e apontando para onde vai ficar o arquivo físico.
- ✗ se precisar aplicar outros backups (log por exemplo) utilize a opção NORECOVERY. Nesse caso, o banco fica esperando próximos arquivos a serem restaurados. Caso não seja necessário aplicar mais nenhum backup, utilize o RECOVERY.

A figura 9.6 demonstra o restore do backup que foi feito no passo anterior. Como o banco já existia, foi usado REPLACE para sobrescrever e NORECOVERY para deixar o banco à espera de novos restores.

Figura 9.6 – Restore

The screenshot shows a SQL query window with the following content:

```
USE [master]
RESTORE DATABASE [DbFael] FROM
DISK = N'c:\\Fael\\DbFael.bak' WITH NORECOVERY, REPLACE, STATS = 20
GO
```

Below the query window, a message window titled "Mensagens" displays the following output:

```
21 por cento processado(s).
40 por cento processado(s).
60 por cento processado(s).
81 por cento processado(s).
100 por cento processado(s).
Processadas 368 páginas para o banco de dados 'DbFael', arquivo 'DbFael' no arquivo 1.
Processadas 6 páginas para o banco de dados 'DbFael', arquivo 'DbFael_log' no arquivo 1.
RESTORE DATABASE processou com êxito 374 páginas em 0.300 segundos (9.720 MB/s).
```

Fonte: elaborado pelo autor.

Enquanto espera outros restores, o banco fica indisponível e no status de restaurando, conforme a figura 9.7.

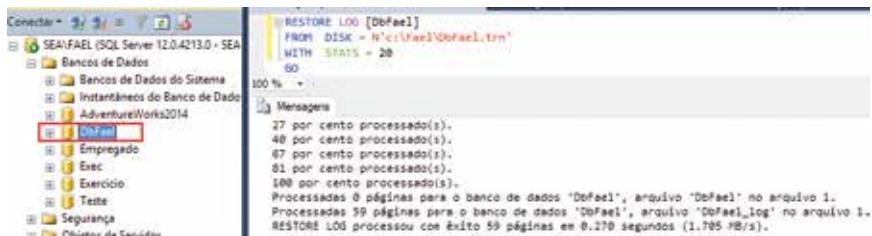
Figura 9.7 – Restaurando



Fonte: elaborado pelo autor.

Após o restore do log com NORECOVERY (que foi omitido, pois é a opção default), observe, figura 9.8, que o banco agora está disponível para utilização.

Figura 9.8 – Restore de log



Fonte: elaborado pelo autor.

O planejamento da estratégia de backup e restore é crucial para a recuperação em caso de problemas. Depois de criar o plano, teste várias vezes, veja o tempo necessário para restaurar tudo e confira se está de acordo com a necessidade do cliente.

9.3 Otimização

Existem inúmeras técnicas de otimização que podem ser utilizadas em cada SGBD, e cada uma delas depende das especificidades com a qual o SGBD executa suas queries e planos de execução. Neste item será discutida a importância do índice sua otimização.

Os índices são estruturas utilizadas para recuperar informações em uma tabela com o menor número possível de operações de leituras, deixando a pesquisa mais rápida e eficiente. Os índices podem ser:

- ✗ **clusterizados** – conhecidos como índices físicos, só pode existir um por tabela. Geralmente utiliza-se a chave primária para criar esses índices;
- ✗ **não clusterizados** – chamados de índices lógicos, podem existir vários por tabelas.

Índices devem ser criados com o mínimo de colunas possível. Utilize-os para melhorar o desempenho da consulta em tabelas com baixos requisitos de atualização (insert, delete e update), mas com grandes volumes de dados.

Segundo MSDN (2016a), existem algumas recomendações para criar índices:

- ✗ índices em views podem melhorar bastante o desempenho quando tiver: agregações, junções de tabela ou uma combinação de agregações e junções;
- ✗ índices não clusterizados podem ser usados nas colunas frequentemente usadas em predicados e condições de junção em consultas;
- ✗ mantenha o comprimento da chave de índice curto para os índices clusterizados;
- ✗ as colunas que forem do tipo de dados ntext, text, image, varchar(max), nvarchar(max), e varbinary(max) não podem ser especificadas como colunas de chave de índice.

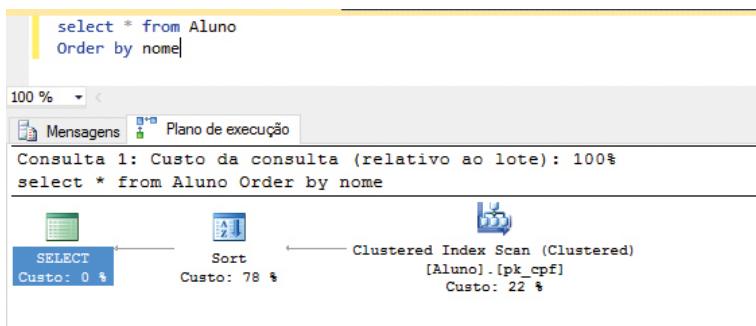
A ordem das colunas que são usadas na cláusula WHERE é importante. Em um critério de consulta igual a (=), maior que (>), menor que (<) ou BETWEEN, ou que participa em uma junção, deve ser posicionada primeiro. Colunas adicionais devem ser ordenadas com base em seu nível de distinção, ou seja, do mais distinto ao menos distinto.

A sintaxe para criação de um índice é:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX NomeIndice  
    ON tabela ( coluna [ ASC | DESC ] [ ,...n  
] )  
    [ INCLUDE ( column_name [ ,...n ] ) ]
```

Observe o plano de execução apresentado na figura 9.9. Ele mostra como o SQL Server faz para pesquisar os dados nas tabelas, compara as estatísticas e escolhe um caminho a seguir. Neste caso, ele faz uma pesquisa pelo índice clusterizado e depois executa o Sort, para fazer o orderby.

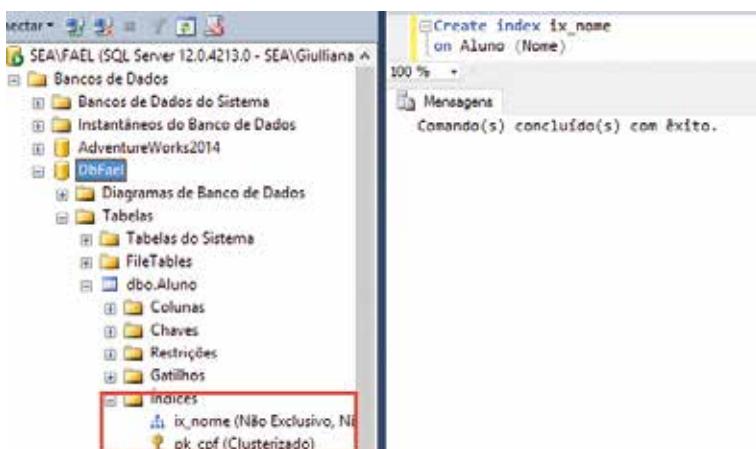
Figura 9.9 – Plano de execução



Fonte: elaborado pelo autor.

O exemplo de criação do índice está na figura 9.10. Foi criado um índice `ix_nome` na tabela `Aluno`, utilizando a coluna `Nome`, como indexador.

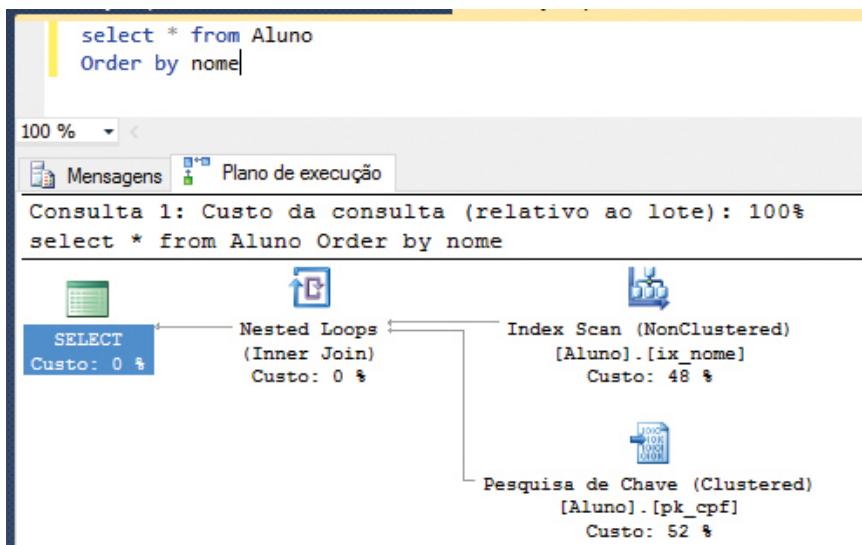
Figura 9.10 – Criação Index



Fonte: elaborado pelo autor.

A figura 9.11 apresenta o resultado da mesma pesquisa após a criação do índice. Nem sempre a criação do índice garante que o SGBD vai utilizá-lo, por isso deve-se analisar para ter certeza de que o índice está sendo realmente útil.

Figura 9.11 – Após criação do índice



Fonte: elaborado pelo autor.

Sintaxe para excluir um índice:

```
DROP INDEX <nomeTabela>.<Nomeindice>
```

Grandes números de índices em uma tabela afetam o desempenho das instruções INSERT, UPDATE e DELETE, porque todos os índices precisam ser ajustados adequadamente à medida que os dados são alterados em uma tabela. Sempre valide a real necessidade de criação de índice.

9.4 Segurança

Para acessar os bancos de dados, é necessário que existam logins e usuários. Os logins são utilizados para conexão com o banco de dados e

os usuários são vinculados aos logins e têm os privilégios relacionados ao banco de dados.

O SQL Server utiliza dois tipos de logins:

- × Windows – são criados no Active Directory (AD) do Windows e utilizados no SQL server;
- × SQL – são criados dentro do próprio SQL, precisam de definição de senha, e se necessário, data de expiração, mudança da senha e checar a política de segurança para verificar a qualidade da senha.

Sintaxe para criar login do Windows (Este login já deve existir no AD):

```
CREATE LOGIN [<Dominio>\<login>] FROM  
WINDOWS;  
  
GO
```

Sintaxe para criar login do SQL:

```
USE [master]  
  
GO  
  
CREATELOGIN [LogindoSQL] WITHPASSWORD=N'Password' [MUST_CHANGE, ] DEFAULT_DATABASE=[DefaultDatabase], CHECK_EXPIRATION=ON | OFF, CHECK_POLICY=ON | OFF  
  
GO
```

Para nomear logins, utilize as mesmas regras para definição de nomes de banco e tabelas. Comece por uma letra, utilize letras, números e sublinhado. Não utilize caracteres especiais.

A figura 9.12 mostra um exemplo de criação de login SQL, até o momento em que não foi dado nenhum privilégio ainda.

Banco de Dados

Figura 9.12 – Criação de logins

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer tree shows the database structure under the 'Bancos de Dados' node. A new login, 'LogindoSQL', has been created and is highlighted with a red box. On the right, the Results pane displays the T-SQL command used to create the login:

```
USE [master]
GO
CREATE LOGIN [LogindoSQL] WITH PASSWORD=N'P@$$word'
    ,INST_CHANGE, DEFAULT_DATABASE=[master], CHECK_EXPIRATION=ON, CHECK_POLICY=ON
GO
```

Below the command, a message in the 'Mensagens' tab states: 'Comando(s) concluído(s) com êxito.'

Fonte: elaborado pelo autor.

O SQL Server já vem com ROLES (regras de privilégios) preparadas, demonstradas na figura 9.13, para leitura, escrita, proprietário do banco, entre outros.

Figura 9.13 – Roles

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer tree shows the database structure under the 'Bancos de Dados' node. A new user, 'LogindoSQL', has been created and is highlighted with a red box. On the right, the Results pane displays the T-SQL commands used to create the user and add it to specific roles:

```
USE [DbFael]
GO
CREATE USER [LogindoSQL] FOR LOGIN [LogindoSQL]
GO
USE [DbFael]
GO
ALTER ROLE [db_datareader] ADD MEMBER [LogindoSQL]
GO
USE [DbFael]
GO
ALTER ROLE [db_datawriter] ADD MEMBER [LogindoSQL]
GO
```

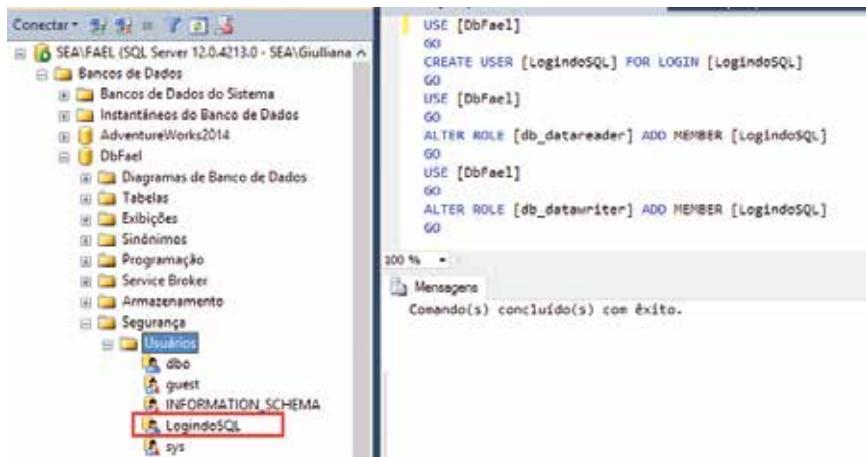
Below the command, a message in the 'Mensagens' tab states: 'Comando(s) concluído(s) com êxito.'

Fonte: elaborado pelo autor.

Pode-se dar o privilégio direto utilizando o comando GRANT, quanto der o privilégio pela role, se o usuário não existir no banco, ele vai ser criado

e vinculado ao login. Se já existir, o SQL só faz o vínculo e adiciona como membro da role, assim um usuário pode acessar o banco e seus objetos de acordo com os privilégios (figura 9.14).

Figura 9.14 – Adiciona role



The screenshot shows the Object Explorer on the left and a query window on the right. In the Object Explorer, under the 'Segurança' node, the 'LogindoSQL' user is selected and highlighted with a red box. The query window contains the following T-SQL code:

```

USE [DbFael]
GO
CREATE USER [LogindoSQL] FOR LOGIN [LogindoSQL]
GO
USE [DbFael]
GO
ALTER ROLE [db_datareader] ADD MEMBER [LogindoSQL]
GO
USE [DbFael]
GO
ALTER ROLE [db_datawriter] ADD MEMBER [LogindoSQL]
GO

```

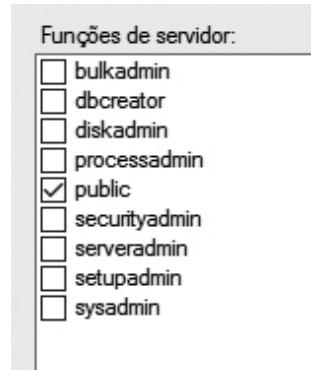
A message box at the bottom right of the query window says: "Comando(s) concluído(s) com êxito."

Fonte: elaborado pelo autor.

Figura 9.15 – Funções do sistema

Para remover os privilégios, pode-se utilizar o REVOKE ou remover o login da role.

Além das roles, temos as funções do servidor que podem ser dadas aos usuários, mas essas funções são recomendadas somente para DBAs e demais administradores.



Fonte: elaborado pelo autor.

É muito importante fazer a verificação e dar permissões corretamente, pois muitos desses sistemas participam de auditorias que podem desclassificar e/ou até cancelar privilégios das empresas por falta de segurança.

Síntese

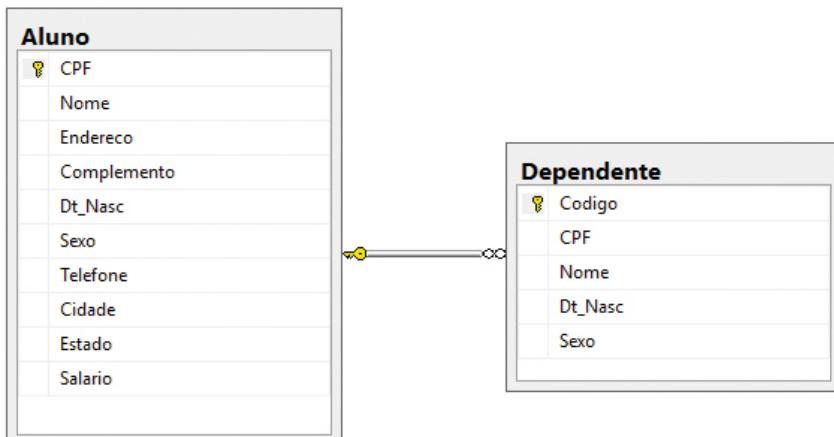
Esse capítulo é um dos mais importantes, porque garante a manutenção do banco de dados com as estratégias de backup e restore, que devem ser muito bem planejadas. Geralmente os backups ficam fisicamente em outro centro de dados, separado do local de armazenamento do banco.

Em relação à otimização, um ponto a ser bem analisado é a criação de índices e verificar sua utilização, o que pode ser feito vendo o plano de execução e observando qual caminho está sendo seguido.

E por último, a estrutura de logins, usuários e privilégios. Lembre-se, se o seu sistema não está corretamente protegido, você pode ter dados maquiados, alterados, apagados e inconsistentes, portanto planeje muito bem quem pode acessar o banco de dados e o que pode ser feito.

Da teoria para a prática

Dado o modelo lógico a seguir, crie as tarefas solicitadas.



- ✗ Faça um backup completo do banco de dados.
- ✗ Faça um backup de log do banco de dados.
- ✗ *Restore* os backups.
- ✗ Crie um login chamado IdTest e dê privilegio de leitura no banco de dadosdbAluno.

10

Tecnologias Avançadas

A TOMADA DE decisão nas empresas é, atualmente, uma das mais complexas e importantes tarefas dos administradores. O mundo está mudando rapidamente, novas estruturas de comunicação e comércio aparecem a cada ano, existem redes sociais, smartphones, tablets e outros equipamentos que mudaram a forma de o consumidor escolher produtos. Existem muitos dados sendo atualizados o tempo todo.

Desta forma, alguém precisa analisá-los e selecionar o que de fato é informação importante para a tomada de decisão correta. A capacidade de armazenamento e velocidade de processamento em banco de dados acompanhou de perto essa evolução, pois hoje se tem Terabytes, Petabytes de dados para analisar. Isso seria humanamente impossível, pois quando se conseguissem analisar todos esses dados, eles já estariam obsoletos. Para ajudar a tomada de decisão, foram criadas inúmeras tecnologias, algumas já embutidas em banco de dados, outras não, mas agora o administrador tem ferramentas que auxiliam a analisar e até projetar o futuro.

Objetivos de aprendizagem:

- ✗ introdução ao *Business Intelligence*;
- ✗ apresentação do *Data Warehouse*;
- ✗ mostra do *Data Mining*.

10.1 Business Intellingence (BI)

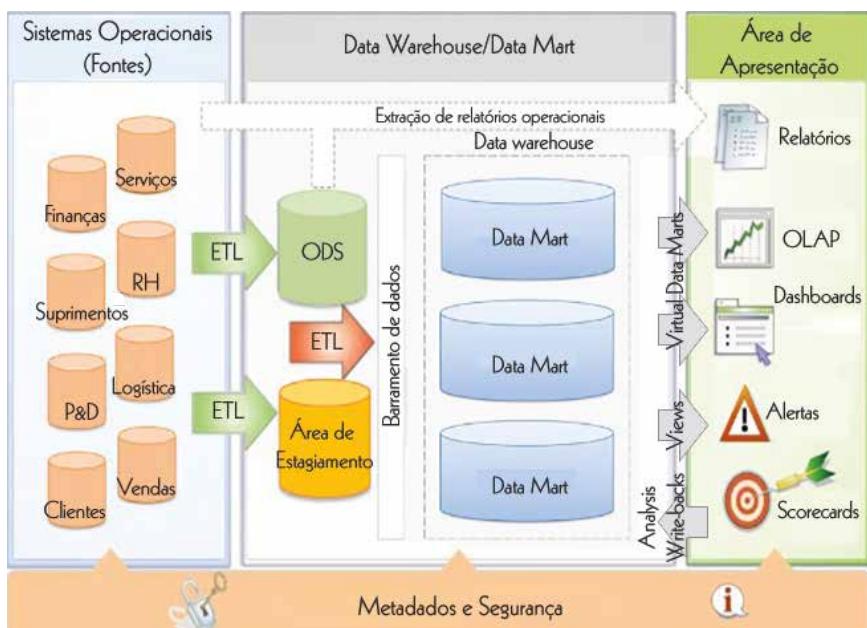
Existem inúmeras definições para BI. Côrtes (2002) define como um conjunto de conceitos e metodologias com o objetivo de apoio à tomada de decisões nos negócios a partir da transformação do dado em informação e da informação em conhecimento, fazendo a análise de dados, e visando à descoberta de novas oportunidades.

À medida que os sistemas de informação evoluíram, as empresas receberam novas ferramentas para auxiliar na tomada de decisão e hoje sua utilização correta é fundamental para o futuro das empresas.

A figura 10.1 apresenta a arquitetura básica de BI. Temos como fonte de dados os sistemas operacionais, (embora tendo o mesmo nome, esses sistemas não são os operacionais de computadores, e sim os sistemas aplicativos utilizados para manter o operacional das empresas, as transações). Observe que temos vários sistemas por departamentos com seus respectivos armazémenotos. São utilizadas ferramentas de ETL (*Extract, Transformand Load*) para extrair os dados da fonte, transformá-los (limpar e padronizar) e finalmente carregá-los em novas bases que podem ser *data warehouse* ou *data mart*. A

partir deste ponto temos dados prontos para serem analisados, relatórios, *data mining*, OLAP (*Online Analytical Processing*), e outros recursos.

Figura 10.1 – Arquitetura BI



Fonte: Silva (2011, p. 34).

As ferramentas de BI podem auxiliar em vários departamentos das empresas, permitindo a análise de:

- ✗ tendências do mercado;
- ✗ mudanças no comportamento de clientes;
- ✗ padrões de consumo.

Ferramentas de BI conseguem analisar as bases de dados e fornecer informações de forma que as empresas possam direcionar, oferecendo novos serviços ou produtos, de forma diferenciada, de acordo com os recursos e perfil de mercado. Mas essas informações ainda precisam ser analisadas por administradores para a tomada de decisão.

10.2 Data Warehouse (DW)

Geralmente, em uma empresa, temos os dados distribuídos em sistemas setoriais, como: financeiro, contábil, recursos humanos, entre outros. Esses sistemas não estão integrados e cada uma tem sua própria arquitetura com formas de armazenamento diferentes e dados replicados.

O *data warehouse* funciona como um armazém de dados. Inmon (1993, p.31) definiu um DW como “um conjunto de dados orientados por assunto, integrado, variável com tempo e não volátil, que fornece suporte ao processo de tomada de decisão do negócio”. A seguir algumas características:

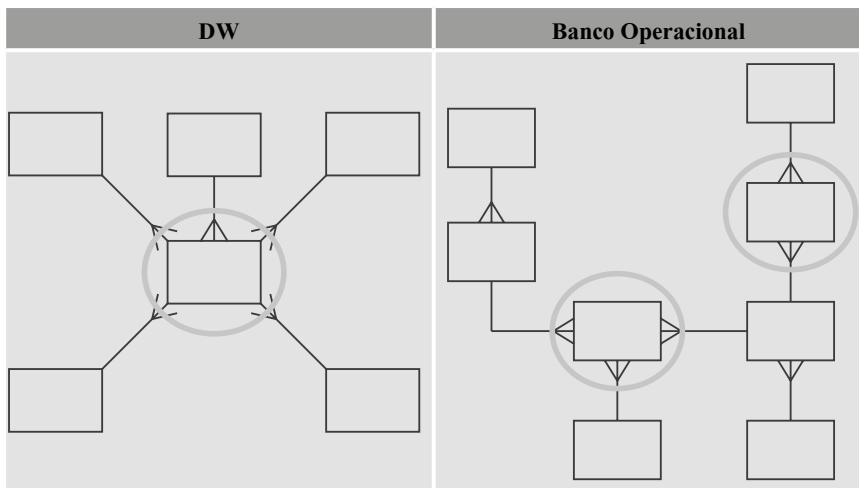
- ✗ orientado por assunto – especifica suas tabelas em temas de negócio;
- ✗ integrado – contém informações de vários setores e sistemas da empresa;
- ✗ variável – com o passar do tempo vai recebendo mais informações;
- ✗ não volátil – o armazenamento dos dados é permanente.

Um DW dá suporte para a análise e tomada de decisões do negócio, criando e mantendo um banco de dados integrado, com informações históricas, consistentes, orientadas para o assunto, com benefícios de custo, economia de tempo e aumento de produtividade. As decisões podem ser tomadas mais rapidamente com a certeza de que os dados são atuais e precisos.

Para melhor compreensão da utilização de um DW ou de Banco de Dados Operacional (Transacional), veja o comparativo na figura 10.2.

Figura 10.2 – DW x Banco Transacional

DW	Banco Operacional
Orientado ao assunto	Orientado à transação
Dados Históricos	Dados atuais
Grande quantidade de dados	Média e grande quantidade de dados
Carga de acordo com a granularidade	Atualizações diárias
Desnormalizado	Normalizado



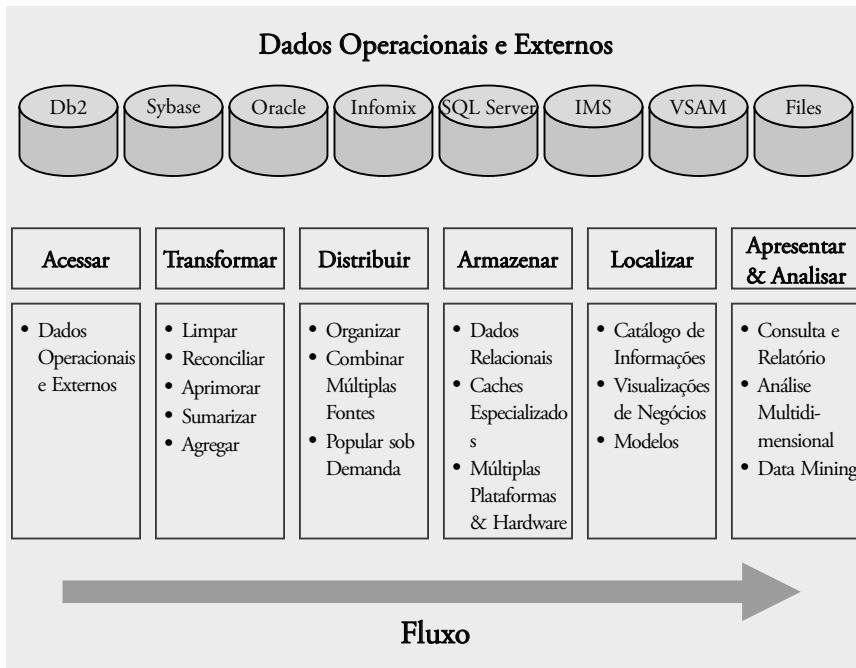
Fonte: elaborado pelo autor.

O ciclo de vida do DW é formado por (SINGH, 2001):

- ✗ análise;
- ✗ design;
- ✗ importação dos dados;
- ✗ instalação das ferramentas;
- ✗ teste e implementação.

A figura 10.3 apresenta os componentes do DW, onde se apresentam dados operacionais externos que podem vir de inúmeros SGBDs, e até arquivos (texto, planilhas, entre outros). Inicialmente o DW deve ter a estrutura de acessar os dados, depois transformá-los, pois podem ter diferentes representações, tipo de dado, formato e precisão, e uma conversão explícita não funcionária, por exemplo, em um sistema Sexo estaria como 'M' ou 'F', em outro poderia ser 0 para masculino e 1 para feminino, em outro sistema poderia vir como 'H' ou 'M', para homem e mulher. Nesse caso é necessário primeiro escolher qual padrão vai ser utilizado no DW e, em seguida, converter todas as entradas diferentes. Depois montar a solução de como serão distribuídos e modelados os dados, seguido do armazenamento, a localização e finalmente utilização de ferramentas para análise.

Figura 10.3 – Componentes do DW



Fonte: Singh (2001).

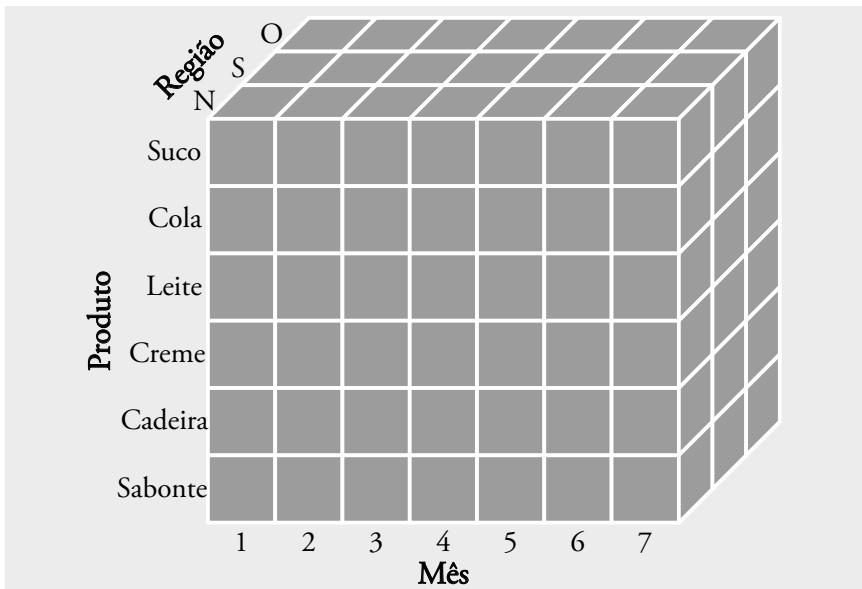
Cada uma das etapas deve ser discutida e revisada antes da sua implementação.

10.3 Modelo multidimensional

O modelo multidimensional apresenta informações do ponto de vista de tempo, diferente do banco transacional que se observa por transações. Essa técnica é uma modelagem conceitual de negócios e é formada por tabelas, dimensões e fatos.

O modelo multidimensional relaciona tabelas de fatos com tabelas de dimensões em um banco de dados. A figura 10.4 mostra esse modelo, que é conhecido como Cubo, onde se tem o produto, por região e mês.

Figura 10.4 – Modelo dimensional



Fonte: MSDN (2007).

Para fazer essa modelagem, deve-se levar em consideração o que é necessário do negócio para ela, quais são os fatos que precisam ser medidos (produto x mês x região é o fato) e cada um desses itens individualmente é a dimensão. Esta etapa é extremamente importante, pois deve considerar somente os dados necessários. Qual é a granularidade, ou seja, qual o nível de detalhamento (sumarização), porque quanto menor a granularidade, mais detalhada a informação. Isso afeta o tamanho do banco de dados, pois quanto mais detalhe, mais espaço será ocupado. No exemplo da figura 10.3 foi escolhida a granularidade por mês, mas poderia ser por dia ou semana. Isso vai variar de acordo com a necessidade da informação solicitada pelo cliente. E finalmente, quais os atributos serão necessários para cada dimensão.

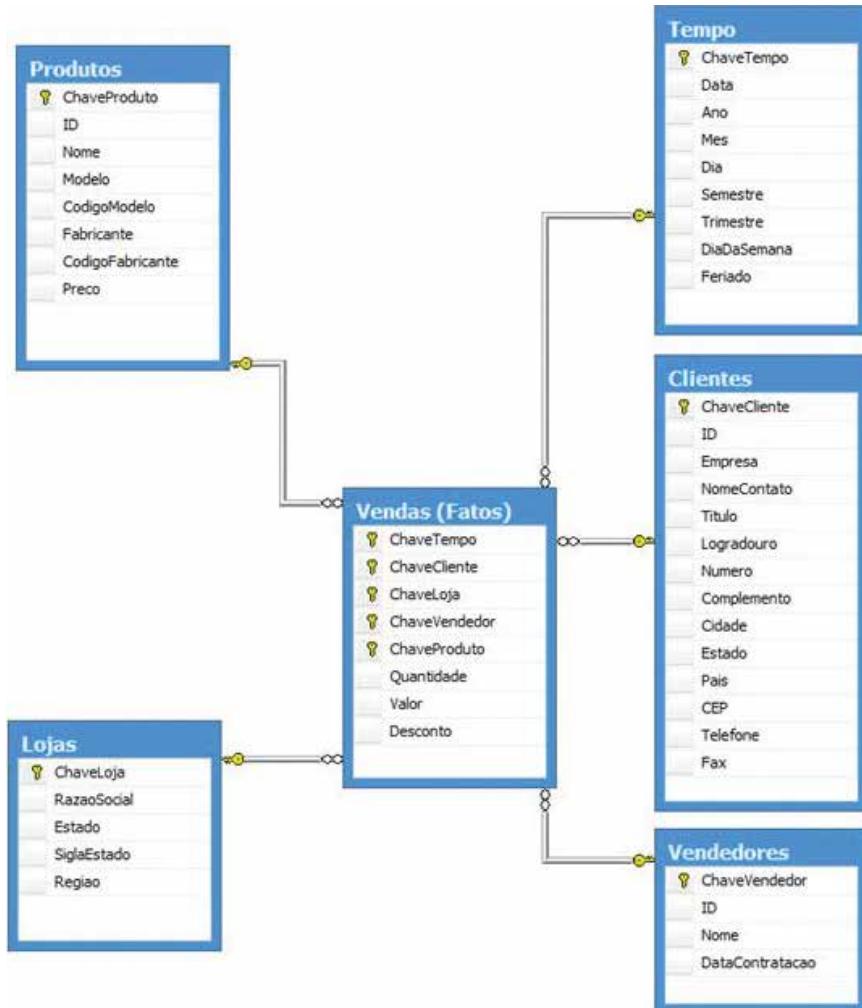
Basicamente são utilizados dois tipos para a modelagem de um DW, são:

- ✗ estrela (*star*);
- ✗ floco de neve (*snowflake*).

Banco de Dados

O esquema estrela é apresentado na figura 10.5. Basicamente tem-se a tabela, fatos e suas dimensões, sem outros níveis de relacionamento. O banco de dados é desnormalizado para ser mais rápido na geração das consultas e relatórios.

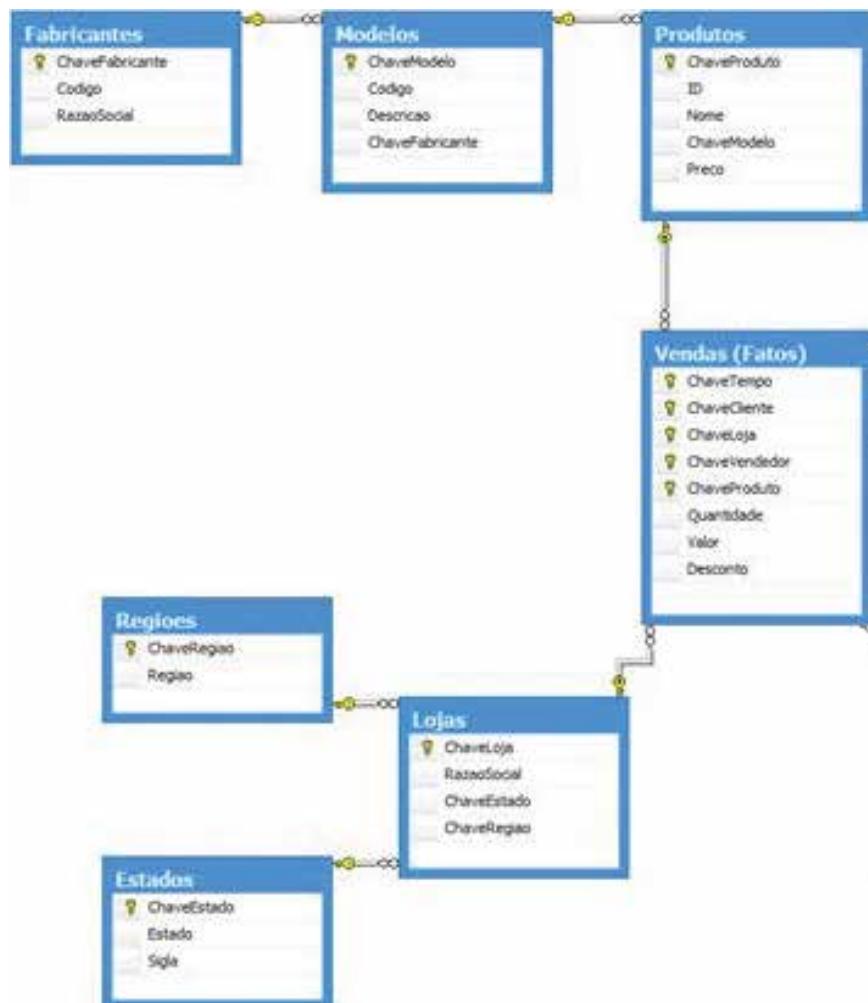
Figura 10.5 – Star



Fonte: MSDN (2007).

O modelo floco de neve permite uma ligeira normalização, figura 10.6, para melhor estruturar os atributos entre as tabelas. Mas é preciso ter cuidado, pois quanto mais normalizado, mais lento vai ficar.

Figura 10.6 – *Snowflake*



Fonte: MSDN (2007).

Depois de definir o modelo, deve-se começar a inserção dos dados. Um DW é somente de leitura, não se alteram dados, somente se inserem os dados. O período para a transformação e carga dos dados deve ser escondido, por exemplo, se os dados vão ser carregados semanalmente ou mensalmente. Geralmente são criados Jobs (tarefas agendadas) para executar essa função.

Um DW tende a tornar-se um banco de dados muito grande, portanto deve-se analisar a escalabilidade de hardware e software para garantir que irá suportar o crescimento.

Outro fator importante é que, para criar um DW completo, é muito pesado e complexo, então são criadas partes do DW chamadas de *data marts*. A diferença está no escopo, pois enquanto o DW é feito para atender a uma empresa como um todo, o *data mart* é criado para atender a um ou vários setores da empresa.

Saiba mais

Faça o tutorial de Modelagem dimensional para SQL Server em <[https://msdn.microsoft.com/pt-br/library/hh231691\(v=sql.120\).aspx](https://msdn.microsoft.com/pt-br/library/hh231691(v=sql.120).aspx)>.

10.4 Data Mining (DM)

O Data mining, também chamado de mineração de dados, é o processo de extração de conhecimento de grandes bases de dados, utilizando técnicas de IA que procuram relações que existam entre os dados num banco.

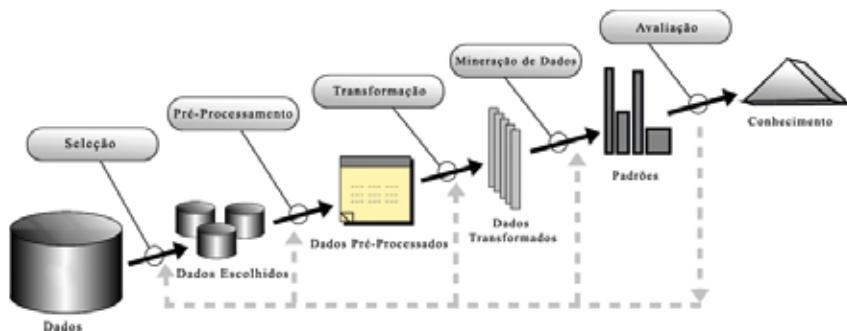
A mineração de dados é o processo de descoberta de informações potencialmente úteis, buscando relações e padrões globais em grandes conjuntos de dados. DM usa análise matemática para derivar padrões e tendências que existem nos dados. Geralmente, esses padrões não podem ser descobertos com a exploração de dados tradicional pelo fato de as relações serem muito complexas ou por haver muitos dados (MSDN, 2016).

O conhecimento descoberto durante a aplicação de técnicas de DM ajuda na tomada de decisões e/ou avaliação de resultados. A utilização de DM é capaz de:

- × criar parâmetros para entender o comportamento do consumidor – o que é mais consumido em períodos do mês ou estações do ano, quais compras têm produtos associados, ou seja, prever hábitos de compra.
- × identificar as relações das escolhas de produtos e serviços – quais grupos procuram determinados produtos e serviços, quando são procurados.
- × analisar comportamentos habituais para detectar fraudes – muito utilizado em bancos e companhias de cartões de crédito, usando o comportamento usual do cliente para identificar possíveis fraudes.

O data mining é, na realidade, uma fase no processo de descoberta de conhecimento em base de dados (KDD- *Knowledge Discovery in Databases*).

Figura 10.7 – KDD



Fonte: Usama (1996).

Definição de metas: definição do problema a ser resolvido pelo processo KDD (FAYAD, 1996).

- × **Seleção:** seleciona os dados apropriados para análise de acordo com a necessidade do cliente.

- ✗ **Processamento:** é a fase de limpeza dos dados, onde certos dados são removidos se forem indicados como desnecessários.
- ✗ **Transformação:** os dados são padronizados, para assegurar um formato consistente afinal, vindos de ambientes diferentes, os dados podem não ter o mesmo padrão ou até não estarem preenchidos. Neste caso precisa-se resolver se vão ser descartados ou receberem valores padrão.
- ✗ **Mineração:** nesta fase são aplicadas as técnicas de DM.
- ✗ **Avaliação:** os resultados das técnicas são avaliados.

DM tem várias técnicas de mineração, porém não existe uma técnica específica para resolver todos os problemas, assim, métodos devem ser utilizados para propósitos diferentes e cada um deles oferece vantagens e desvantagens. As principais técnicas são:

- ✗ **regras de associação** – estabelecem uma relação estatística entre certos itens de dados em um conjunto. São utilizadas na tarefa de associação.
- ✗ **árvores de decisão** – é a técnica indicada no não terminal. Representa um teste ou decisão sobre o item de dado considerado, cujo objetivo é separar as classes. Listas de classes diferentes tendem a ser alocadas em subconjuntos diferentes. Árvores de decisão, em geral, são apropriadas para classificação e regressão.
- ✗ **algoritmos genéticos (AG)** – são métodos de busca e otimização que simulam os processos naturais de evolução. Os AGs utilizam operadores de seleção, cruzamento e mutação para desenvolver gerações de solução. Essa técnica é apropriada para tarefas de classificação e segmentação.
- ✗ **redes neurais (RN)** – são uma classe especial de sistemas que seguem analogia com funcionamento do cérebro humano, já que são formados neurônios artificiais similares aos neurônios do ser humano. Geralmente são apropriadas para as tarefas de classificação e segmentação.

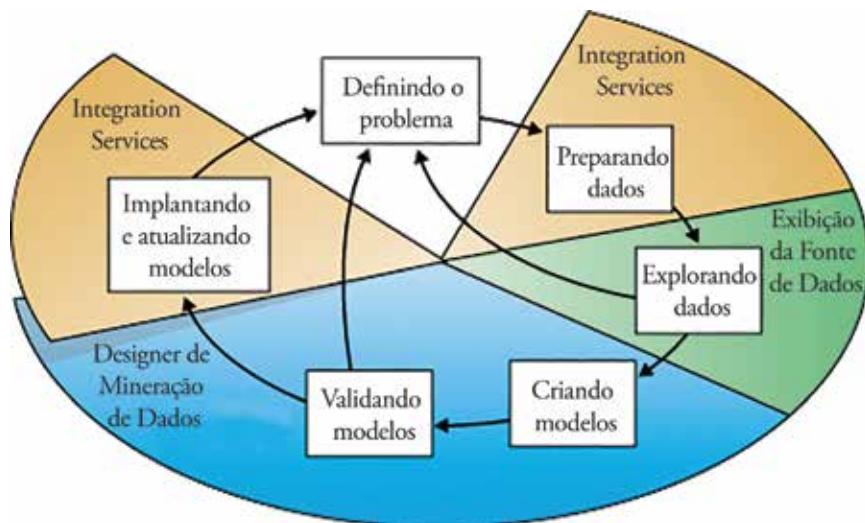
Técnicas de mineração de dados podem ser aplicadas a várias tarefas. Segundo Fayad (1996), estas tarefas são:

- × **classificação** – a análise das características de um conjunto de dados de treinamento, construindo um modelo para cada classe baseado nas características dos dados.
- × **associação** – consiste em identificar fatos que possam ocorrer numa mesma transação, pode ser usada para avaliar alguma relação temporal entre os itens de um banco de dados.
- × **agregação** – também conhecida como *clustering*, é utilizada para buscar similaridades entre os dados como agrupamento de classes ou categorias. Ajuda a identificar grupos significativos que decompõem o sistema de grande escala em componentes menores.
- × **modelagem de dependência** – procura encontrar um modelo que mostre as dependências entre as variáveis.
- × **regressão** – procura por uma função que represente o comportamento apresentado pelo fenômeno de um estudo. Dá para prever valores futuros a partir de valores existentes.
- × **Sumarização:** envolve métodos para encontrar uma descrição compacta de um subconjunto de dados.

Segundo MSDN (2016), o modelo de mineração representa apenas uma parte de um processo que inclui desde perguntas sobre dados e criação de um modelo até respostas para as perguntas feitas e implantação do modelo em um ambiente de trabalho. Esse processo é apresentado utilizando as seis etapas a seguir:

- × definindo o problema;
- × preparando dados;
- × explorando dados;
- × criando modelos;
- × explorando e validando modelos;

Figura 10.8 – Implantando e atualizando modelos



Fonte: MSDN, 2007

O diagrama indica um processo cíclico, ou seja, criar um modelo de mineração de dados é um processo dinâmico e repetitivo. Depois de explorar os dados, você pode descobrir que eles são insuficientes para criar modelos de mineração apropriados e que terá, portanto, que obter mais dados. Ou você pode criar vários modelos e, depois, perceber que os modelos não respondem adequadamente o problema definido e que você deverá redefini-lo. Talvez seja necessário atualizar os modelos depois de eles serem implantados, pois haverá mais dados disponíveis. Cada etapa do processo pode precisar ser repetida muitas vezes para criar um bom modelo (MSDN, 2016).

Os modelos de mineração podem ser aplicados a cenários específicos, como:

- ✗ **previsão** – estimando vendas, prevendo cargas de servidor ou tempo de inatividade de servidor.
- ✗ **risco e probabilidade** – escolhendo os melhores clientes para malas diretas, determinando o ponto equilibrado provável para cenários de risco, atribuindo probabilidades a diagnósticos ou outros resultados.

- ✗ **recomendações** – determinando quais produtos são mais prováveis de serem vendidos juntos, gerando recomendações.
- ✗ **localizando sequências** – analisando seleções de cliente em um carrinho de compras, prevendo os próximos eventos prováveis.
- ✗ **agrupamento** – separando clientes ou eventos em cluster de itens relacionados, analisando e prevendo afinidades (MSDN, 2016).

Saiba mais

Faça o tutorial de Data mining do SQL Server em <[https://msdn.microsoft.com/pt-br/library/ms167167\(v=sql.120\).aspx](https://msdn.microsoft.com/pt-br/library/ms167167(v=sql.120).aspx)>.

Síntese

Neste capítulo foi apresentada uma introdução às técnicas avançadas que utilizam banco de dados. Muitas dessas técnicas já são utilizadas em empresas de grande e médio porte, mas podem ser aplicadas em empresas pequenas.

Foi apresentado BI, com suas etapas e as tecnologias aplicadas, entre elas temos o OLAP para criação e demonstração das informações obtidas nas aplicações de *data warehouse* e *data mining*.

Alguns SGBDs oferecem essas ferramentas embutidas na compra da ferramenta, outras oferecem em pacotes separados. Os tutoriais sugeridos para o MS SQL Server auxiliam na melhor compreensão do funcionamento prático.

Da teoria à prática

- ✗ Pesquise e explique qual a diferença entre OLAP x OLTP
- ✗ Explique por que BI é importante para as empresas.
- ✗ Qual a importância da utilização de *data warehouse*?
- ✗ Qual a importância da utilização de *data mining*?

Conclusão

A DISCIPLINA DE banco de dados está inserida no currículo de diversos cursos, em áreas de conhecimento diferentes, cada um abordando-a de acordo com seus interesses. Por exemplo: um bibliotecário preocupa-se em catalogar e disponibilizar dados de forma eficiente, enquanto um biólogo se interessa em armazenar e consultar dados sobre suas pesquisas com proteínas e suas funções. Para o pessoal da área da computação, o interesse é diferente, pois são eles os responsáveis pela análise, projeto, implementação e manutenção dos dados.

Banco de Dados

Quem trabalha na área de banco de dados atua em uma das três frentes principais: construção de aplicativos para clientes utilizando um gerenciador de dados; administração do gerenciador, que é quem autoriza o uso dos diversos conjuntos de dados armazenados para os diversos usuários (setores e/ou funcionários da empresa) e se responsabiliza pela manutenção do sistema em operação adequada; e na construção dos gerenciadores de dados.

Por ser uma pessoa que trabalha há muito tempo com SGBDs e por ser totalmente apaixonada por essa área, preparei o livro de forma que, com o conhecimento adquirido neste estudo, você esteja preparado para começar a trabalhar na área de banco de dados, caso tenha interesse. De qualquer forma, o livro atende ao público em geral, mesmo para quem vai seguir outra área e precise do conhecimento básico em bancos de dados.

Com o material apresentado, você terá o conhecimento necessário para fazer análise e modelagem do banco de dados, criação do banco no SGBD, programação em SQL e planejamento de backups para garantir a segurança dos dados em caso de falha.

Aproveite para executar os tutoriais apresentados nas dicas e não esqueça das recomendações importantes, como seguir padrão e rever todas as etapas com o cliente.

Este é apenas o começo da viagem, a área de banco de dados é muito ampla e com muitas oportunidades de mercado.

Obrigada e bom estudo.

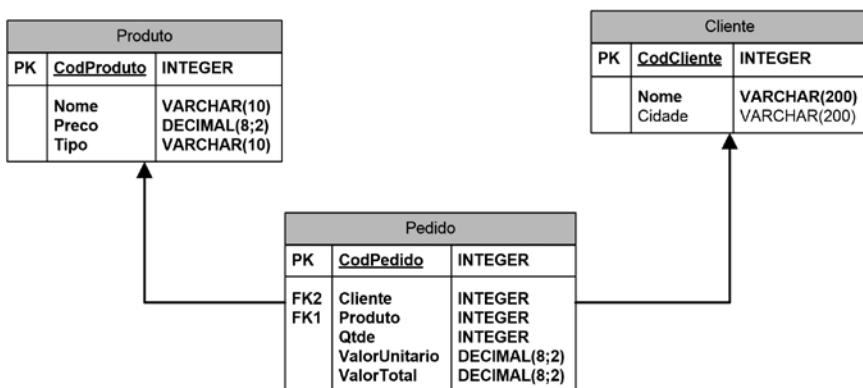
Gabarito

1. Banco de dados

- ✗ Confeitaria de pequeno porte utilizando sistema operacional Windows 10.
Resposta: SQL Server Express ou My SQL.
- ✗ Papelaria de médio porte utilizando sistema operacional Linux Ubuntu.
Resposta: My SQL.
- ✗ Sistema de agendamento de pacotes de viagens, com mais de 300 lojas no Brasil, utilizando sistema Windows 8.1.
Resposta: SQL Server.
- ✗ Sistemas de geoprocessamento para análise da geografia do estado do Amazonas.
Resposta: Oracle.

2. Modelagem de Dados

Este modelo é somente para referência, não é obrigatório inserir os tipos de dados nem as chaves estrangeiras.



4. SQL

- ✗ crie o Banco de dados

```
CREATE DATABASE [Exercicio]
ON PRIMARY
( NAME = N'Exercicio', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.FAEL\MSSQL\DATA\ Exercicio.mdf' , SIZE =
5120KB , FILEGROWTH = 1024KB )
LOG ON
( NAME = N'Exercicio_log', FILENAME =
N'C:\Program Files\Microsoft SQL Server\MSSQL12.FAEL\MSSQL\DATA\ Exercicio_log.ldf',
, SIZE = 1024KB , FILEGROWTH = 10%)
```

- ✗ crie a tabela Produto com suas restrições

```
Create table Produto  
(  
    CodProduto integer Constraint PK_produto  
        primary key,  
    Nome Varchar(10) not null,  
    Preco decimal(8,2) not null,  
    Tipo varchar(10)  
)
```

- ✗ crie a tabela Cliente com suas restrições

```
Create table Cliente  
(  
    CodCliente integer Constraint PK_cliente  
        primary key,  
    Nome varchar(200),  
    Cidade varchar (200)  
)
```

- ✗ crie a tabela Pedidos com suas restrições

```
Create table pedido
(
    Pedido integer Constraint PK_pedido primary key,
    Cliente integer Constraint fk_pedido-Cliente foreign key (Cliente) references Cliente(codCliente),
    Produto integer Constraint fk_pedidoProduto foreign key (Produto) references Produto (codProduto),
    QTDE Integer not null,
    ValorUnitario Decimal(8,2) not null,
    ValorTotal Decimal(8,2) not null
)
```

5. SQL DML

- × Selecione o nome e o preço de todos os produtos.
R – Select Nome, Preço From Produto.
- × Selecione o nome de todos os clientes que moram em Brasília.
R – Select Nome From Cliente Where Cidade like ‘Brasilia’
- × Selecione o nome e o preço dos produtos que custam menos de R\$ 100, ordenado pelo preço, começando pelo menor valor.
R – Select Nome, Preco from Produto Where preco < 100 Order by Preco
- × Selecione todos os clientes que tenha a substring ANA no nome.
R – Select * From Cliente Where Nome like ‘%ana%’

6. SQL Avançado

a)

```
Create database Teste;
Go
Use Teste
Go
CREATE TABLE Cargo
(
    CodCargo smallint NOT NULL,
    NomeCargo varchar(50) NULL,
    ValorCargo smallmoney NULL,
    PRIMARY KEY(CodCargo)
)
go
CREATE TABLE Funcionario
(
    CPF numeric (11),
    NomeFuncionario varchar(70) NOT NULL,
    CodCargo Smallint Not NULL,
    PRIMARY KEY(CPF),
    FOREIGN KEY (CodCargo) REFERENCES CARGO (Cod-
Cargo)
)
GO
INSERT CARGO (CodCargo, NomeCargo, ValorCargo)
VALUES (1, 'Repcionista', 1400.00)
INSERT CARGO (CodCargo, NomeCargo, ValorCargo)
VALUES (2, 'Vendedor', 1200.00)
INSERT CARGO (CodCargo, NomeCargo, ValorCargo)
VALUES (3, 'Faxineiro' , 1050.00)
GO
-- Povoando a tabela FUNCIONARIO
INSERT FUNCIONARIO (CPF, NomeFuncionario, Cod-
Cargo) VALUES (11, 'João da Silva' , 1)
INSERT FUNCIONARIO (CPF, NomeFuncionario, Cod-
Cargo) VALUES (12, 'Maria de Souza' , 2)
INSERT FUNCIONARIO (CPF, NomeFuncionario, Cod-
Cargo) VALUES (13, 'Carla Brunni' , 3)
GO
```

b)

```
Select NomeFuncionario, nomeCargo  
From Funcionario F  
Inner join Cargo C  
On F.CodCargo = C.CodCargo
```

c)

```
Create view vwExercicio  
as  
Select NomeFuncionario, nomeCargo  
From Funcionario F  
Inner join Cargo C  
On F.CodCargo = C.CodCargo
```

d)

```
Select *  
From vwExercicio  
Order by nomeCargo desc
```

7. Transações e Técnicas Avançadas

```
Create procedure pExercicio
@CPF Numeric(11)
as
Declare @Salario Smallmoney
Set @Salario = 0
Select @Salario = Salario
From Aluno
Where CPF = @CPF
Begin Tran --Transação
Delete Aluno
Where CPF = @CPF
IF @Salario >= 10000
Commit
Else
Rollback
```

8. Recursos Avançados

```
Create trigger TrExercicio on Aluno
as
    SELECT *, 'I', Getdate()
    FROM INSERTED
    WHERE NOT EXISTS
        (SELECT * FROM DELETED WHERE INSERTED.CPF
        = DELETED. CPF )
    SELECT *, 'U', Getdate()
    FROM INSERTED
    WHERE EXISTS
        (SELECT * FROM DELETED WHERE INSERTED.
        CPF= DELETED. CPF )
    SELECT *, 'D', Getdate()
    FROM DELETED
    WHERE NOT EXISTS
        (SELECT * FROM INSERTED WHERE DELETED.
        CPF= INSERTED. CPF )
```

9. Backup, Restore, Segurança e Otimização

Resposta a)

```
Backup database dbAluno to disk = 'd:\\
banco\dbAluno.bak' with compresion
```

Banco de Dados

Resposta b)

```
Backup log dbAluno to disk = 'd:\banco\
dbAluno.trn' with compresion
```

Resposta c)

```
Restore database dbAluno from disk = 'd:\\
banco\dbAluno.bak' with norecovery
```

```
Go
```

```
Restore log dbAluno from disk = 'd:\\
banco\dbAluno.trn' with recovery
```

```
Go
```

Resposta d)

```
USE [master]
GO
CREATE LOGIN [IdTest] WITH PASSWORD=N'P@$$w0rd', 
MUST_CHANGE, DEFAULT_DATABASE=[dbAluno],
CHECK_EXPIRATION=ON, CHECK_POLICY=ON
GO
USE [DbAluno]
GO
CREATE USER [IdTest] FOR LOGIN [IdTest]
GO
USE [DbAluno]
GO
ALTER ROLE [IdTest] ADD MEMBER [IdTest]
GO
```

10. Tecnologias Avançadas

Resposta a)

OLAP: *online Analytical Processing* – sistemas que permitem a análise da informação.

OLTP: *online Transaction Processing* – sistemas de processamento de transações, ou seja, sistemas transacionais, onde são inseridas as operações das empresas.

OLTP é voltado para sistema de transações, regras de negócio que são aplicadas no sistema, por exemplo, um sistema de supermercado. São as compras e vendas de produtos e serviços das empresas.

OLAP é voltado para análise das informações, ou seja, cálculos e consultas mais complexas, relatórios e gráficos sobre os dados operacionais após o processamento e transformação.

Resposta b)

Com a concorrência das empresas e vários fatores modificando o mundo dos negócios, é extremamente importante para a sobrevivência das empresas se manterem atualizadas e conseguir rapidamente informações sobre o seu negócio e o comportamento dos clientes e do mercado, para tomar decisões que irão direcionar o caminho que a empresa vai percorrer para crescer melhor.

Resposta c)

Permite tomar decisões embasadas em um histórico dos dados integrado. Identifica tendências, de forma a posicionar a empresa, melhorando a competitividade, ampliando lucros e minimizando os erros. Como é um banco à parte do banco operacional não gera concorrência de utilização com o sistema transacional da empresa.

Resposta d)

Os benefícios do DM variam desde ajudar a tomar decisões corretas e mais rápidas, apontar falhas nos sistemas e trazer mais lucros da empresa, melhorando o Marketing. Suas funções são: descobrir as principais características dos consumidores, melhorar o processo produtivo, avaliar o risco, detectar e prevenir fraudes, acompanhar e antecipar uma mudança no comportamento do mercado.

Referências

CHEN, P. **Modelagem de dados:** a abordagem entidade relacionamento para projeto lógico. São Paulo: Makron Books, 1990.

DATE, C. J. **Introdução a sistemas de bancos de dados.** 7. ed. Rio de Janeiro: Campus, 2000.

ELSMARI, R.; NAVATHE, S. B. **Sistemas de bancos de dados.** 6. ed, São Paulo: Pearson, 2013.

FAYYAD, U. M. et al. **Advanced in knowledge discovery and data mining.** Cambridge: AAAI press, 1996.

GOEBEL, M; GRUENWALD, E. A survey of data mining and knowledge. **SIGKDD Explorations**, v. 1, n. 1, 1999.

INMON, W. H. **Building the data warehouse.** Indianapolis: Wiley, 1998.

KIMBALL, R. **The data warehouse toolkit.** New York: John Wiley & Sons, 2000.

MEIRA, F. **Sistemas de banco de dados.** São Paulo: UFRGS, 1997. Disponível em: <https://chasqueweb.ufrgs.br/~paul.fisher/apostilas/basdad/unimar/index.htm>. Acesso em: 20 ago. 2016.

MSDN. **Conceitos de mineração de dados.** 2016. Disponível em: <<https://msdn.microsoft.com/pt-br/library/ms174949.aspx>>. Acesso em: 30 ago. 2016.

_____. **Fundamentos e Modelagem de Bancos de Dados Multidimensionais.** 2007. Disponível em: <<https://msdn.microsoft.com/pt-br/library/cc518031.aspx>>. Acesso em: 30 ago. 2016.

_____. **Guia de criação de índice do SQL Server.** Disponível em: <[https://msdn.microsoft.com/pt-br/library/jj835095\(v=sql.120\).aspx](https://msdn.microsoft.com/pt-br/library/jj835095(v=sql.120).aspx)>. Acesso em: 29 ago. 2016a.

_____. **Microsoft.** Disponível em: <<https://msdn.microsoft.com/pt-br/library/ms180169.aspx>>. Acesso em: 28 ago. 2016.

_____. **Recuperar para um número de sequência de log (SQL Server).** Disponível em: <<https://msdn.microsoft.com/pt-br/library/ms191459.aspx>>. Acesso em: 29 ago. 2016b.

PROCEDIMENTO armazenados (Mecanismo de Banco de Dados). **Microsoft**, 2016. Disponível em: <<https://msdn.microsoft.com/pt-br/library/ms190782.aspx>>. Acesso em: 20 ago. 2016.

SELL, D. **Uma arquitetura para business intelligence baseada em tecnologias semânticas para suporte a aplicações analíticas**. 2006. Florianópolis: Universidade Federal de Santa Catarina, programa de pós-graduação em Engenharia de Produção, 2006.

SIBERCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de banco de dados**. 3. ed. São Paulo: Makron Books, 1999.

SILVA, D. C. da. **Uma arquitetura de business intelligence para processamento analítico baseado em tecnologias semânticas e em linguagem natural**. Florianópolis: Universidade Federal de Santa Catarina, 2011.

TECHNET. **Estratégias de backup**. Disponível em: <<https://technet.microsoft.com/pt-br/library/cc974133.aspx>>. Acesso em: 29 ago. 2016.

O livro apresenta os conceitos de bancos de dados, desde sua modelagem a partir das regras de negócio até a utilização de suas ferramentas e técnicas avançadas.

O livro apresenta em seu primeiro capítulo a definição e os conceitos básicos de banco de dados e sua utilização.

No segundo e terceiro capítulo, temos a demonstração de como transformar um objeto de negócio que precisa ser modelado para que seja transformado em estruturas representativas de banco de dados. Estas estruturas começam com entidades e relacionamentos que precisam ser aprimorados e modelados para as estruturas físicas, com as regras de normalização e dicionário de dados.

No quarto capítulo temos a atuação direta nas ferramentas dos Sistemas Gerenciadores de Banco de Dados (SGBD), iniciando com a utilização da *Structured Query Language* (SQL) juntamente com a *Data Definition Language* (DDL), que apresenta a forma de se criar as estruturas (tabelas) do modelo relacional dentro do SGBD. Veremos ainda as restrições e integridade referencial.

No quinto e sexto capítulo são apresentados a Data Manipulation Language (DML), para fazer inserções, deleções e atualizações de registros, e a DML avançada, com funções de agregação, ordenação, junções, queries aninhadas e uniões.

No sétimo capítulo, temos um dos conceitos mais importantes para garantir a consistência do banco de dados, a transação, programação T-SQL e *Stored procedures*.

No oitavo capítulo são apresentados cursores, tabelas temporárias e triggers, que são estruturas avançadas que auxiliam numa melhor utilização do banco de dados, oferecendo recursos que melhoraram o desempenho e a programação.

O nono capítulo mostra as ferramentas do SGBD que melhoraram a segurança dos dados para o acesso, ou em caso de falhas humanas ou de hardware.

São apresentados backup e restore, criação, manutenção de ids e privilégios e otimização de comandos SQL.

E, finalmente, veremos no último capítulo as tecnologias avançadas para utilização do banco de dados como ferramentas de *Data Warehouse* e *Data mining*.

Recomenda-se que você leia o livro na sequência apresentada, fazendo os exercícios e solidificando o conhecimento.



EDITORIA
FAEL

ISBN 978-85-60531-65-3

9 788560 531653