

BANCO DE DADOS

ME. EDSON YANAGI

ME. WILLIAM ROBERTO PELISSARI

ESP. CARLOS DANILÓ LUZ

ESP. JEFERSON KAISER

ESP. VÍCTOR DE MARQUÍ PEDROSO

ORGANIZADORA - HELOISA ROSA

EXPEDIENTE

Coordenador(a) de Conteúdo

Flavia Lumi Matuzawa

Projeto Gráfico e Capa

Arthur Cantareli Silva

Editoração

Lucas Pinna Silveira Lima

Design Educacional

Aguinaldo José Lorca Ventura Júnior

Revisão Textual

Keli Francieli Cattoni

Gabriel Augusto Lenzi

Ilustração

Geison Odlevati Ferreira

Wellington Vainer

Fotos

Freepik

Shutterstock

FICHA CATALOGRÁFICA

C397 **Centro Universitário Leonardo da Vinci.**

Núcleo de Educação a Distância. **YANAGA**, Edson; **PELISSARI**, William Roberto; **LUZ**, Carlos Danilo; **KAIER**, Jeferson; **PEDROSO**, Victor de Marqui.

Banco de Dados / Edson Yanaga, William Roberto Pelissari, Carlos Danilo Luz, Jeferson Kaiser, Victor de Marqui Pedroso, Heloisa Rosa (Org.). - Indaial, SC: Arqué, 2023.

240 p.

ISBN papel 978-65-5466-000-6

ISBN digital 978-65-5466-001-3

"Graduação - EaD".

1. Banco de Dados 2. SQL 3. Dados. 4. EaD. I. Título.

CDD - 005.74

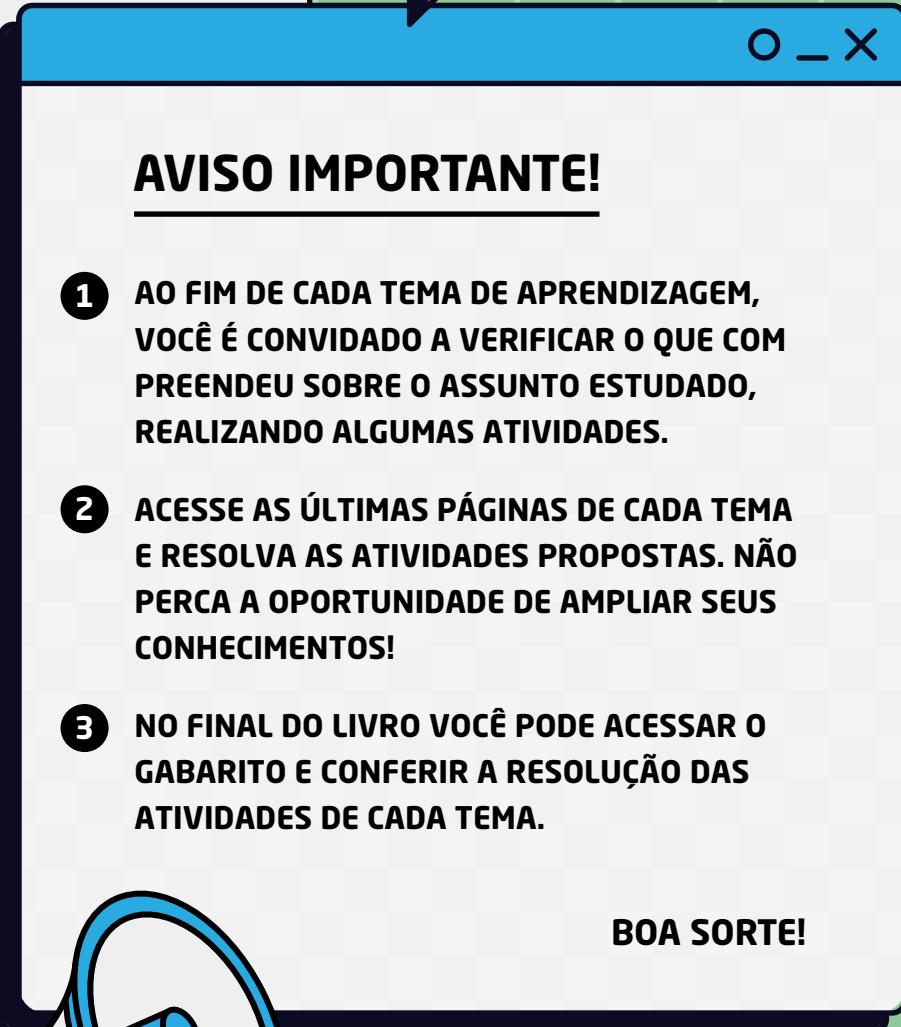
Bibliotecária: Leila Regina do Nascimento - CRB- 9/1722.

Ficha catalográfica elaborada de acordo com os dados fornecidos pelo(a) autor(a).

Impresso por:



ACADÊMICO!

An illustration of a computer monitor with a blue header bar containing a circular icon and an 'X'. The main screen displays the word 'AVISO IMPORTANTE!' in bold capital letters.

AVISO IMPORTANTE!

- 1 AO FIM DE CADA TEMA DE APRENDIZAGEM, VOCÊ É CONVIDADO A VERIFICAR O QUE COM PREENDEU SOBRE O ASSUNTO ESTUDADO, REALIZANDO ALGUMAS ATIVIDADES.**
- 2 ACESSE AS ÚLTIMAS PÁGINAS DE CADA TEMA E RESOLVA AS ATIVIDADES PROPOSTAS. NÃO PERCA A OPORTUNIDADE DE AMPLIAR SEUS CONHECIMENTOS!**
- 3 NO FINAL DO LIVRO VOCÊ PODE ACESSAR O GABARITO E CONFERIR A RESOLUÇÃO DAS ATIVIDADES DE CADA TEMA.**

BOA SORTE!



RECURSOS DE IMERSÃO

APROFUNDANDO

Utilizado para temas, assuntos ou conceitos avançados, levando ao aprofundamento do que está sendo trabalhado naquele momento do texto.

PENSANDO JUNTOS

Este item corresponde a uma proposta de reflexão que pode ser apresentada por meio de uma frase, um trecho breve ou uma pergunta.

ZOOM NO CONHECIMENTO

Utilizado para desmistificar pontos que possam gerar confusão sobre o tema. Após o texto trazer a explicação, essa interlocução pode trazer pontos adicionais que contribuam para que o estudante não fique com dúvidas sobre o tema.

EU INDICO

Utilizado para agregar um conteúdo externo. Utilizando o QR-code você poderá acessar links de vídeos, artigos, sites, etc. Acrescentando muito aprendizado em toda a sua trajetória.



PLAY NO CONHECIMENTO

Professores especialistas e convidados, ampliando as discussões sobre os temas por meio de fantásticos podcasts.



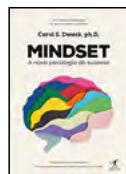
INDICAÇÃO DE FILME

Uma dose extra de conhecimento é sempre bem-vinda. Aqui você terá indicações de filmes que se conectam com o tema do conteúdo.



INDICAÇÃO DE LIVRO

Uma dose extra de conhecimento é sempre bem-vinda. Aqui você terá indicações de livros que agregarão muito na sua vida profissional.



CAMINHOS DE APRENDIZAGEM

9

UNIDADE 1

CONCEITO DE BANCO DE DADOS	10
MINHAS METAS	10
O QUE É UM BANCO DE DADOS	12
PRINCIPAIS VANTAGENS DE UM BANCO DE DADOS	13
SISTEMAS DE GERENCIAMENTO DE BANCO DE DADOS (SGBD)	14
SGBD NO MERCADO	17
MODELO DE DADOS HIERÁRQUICO	23
MODELO DE DADOS REDE	24
MODELO DE DADOS RELACIONAL	25
MODELO DE DADOS ORIENTADO A OBJETOS	26
MODELO RELACIONAL	28
DIFERENÇAS ENTRE O MYSQL E O ORACLE	29
CARACTERÍSTICAS DE SISTEMAS DE BANCOS DE DADOS	42
TRANSAÇÕES	44
VANTAGENS DE SE UTILIZAR UM SGBD	46
 MODELAGEM DE DADOS	50
MINHAS METAS	50
MODELAGEM DE DADOS	52
MODELO ENTIDADE RELACIONAMENTO (MER)	55
RELACIONAMENTOS	59
NORMALIZAÇÃO	66

CAMINHOS DE APRENDIZAGEM

MODELO RELACIONAL	74
MINHAS METAS	74
O MODELO RELACIONAL	76
INTRODUÇÃO À MODELAGEM	77
ATRIBUTOS	79
TIPOS DE ATRIBUTOS	80
DOMÍNIO	84
CHAVE ESTRANGEIRA (FOREIGN KEY)	85
RELACIONAMENTOS	86
CARDINALIDADE	88
 95 UNIDADE 2	
SQL BÁSICO	96
DEFINIÇÕES DE DADOS E TIPOS EM SQL	98
TIPOS DE DADOS	101
RESTRIÇÕES	102
Consultas Básicas em SQL	104
COMANDOS DE MODIFICAÇÃO DE DADOS EM SQL	111
MAIS SQL	118
CONSULTAS ENVOLVENDO NULL	119
CONSULTAS ANINHADAS (SUBQUERIES)	121
CONSULTAS UTILIZANDO JOINS	125
ESTUDO DE CASO	136

CAMINHOS DE APRENDIZAGEM

DESCREVENDO O ESTUDO DE CASO	137
criando as entidades e os relacionamentos (DER)	138
trabalhando com SQL	146



UNIDADE 3

criando um banco de dados	166
SCHEMA	167
TIPOS DE DADOS	171
CRIAÇÃO E ALTERAÇÃO DE TABELAS	173
CHAVE PRIMÁRIA	174
POPULANDO AS TABELAS CRIADAS	178
COMANDO DESCRIBE	181
a linguagem sql	186
HISTÓRIA DA SQL	187
DQL - Linguagem de consulta de dados	190
DML - LINGUAGEM DE MANIPULAÇÃO DE DADOS	194
DDL - LINGUAGEM DE DEFINIÇÃO DE DADOS	196
DCL - LINGUAGEM DE TRANSAÇÃO DE DADOS	201
DTL - LINGUAGEM DE TRANSAÇÃO DE DADOS	203
manipulação de dados	206
EXTRAIR DADOS DE UMA TABELA	207
AGRUPANDO A EXIBIÇÃO DOS DADOS	212
ORDENAÇÃO NA EXIBIÇÃO DOS DADOS	213

CAMINHOS DE APRENDIZAGEM

CRIANDO CONSULTAS COM FILTROS ESPECÍFICOS	214
VALORES NULOS	215
RENAME	215
CONSULTA UTILIZANDO MAIS DE UMA TABELA	216
CONSULTAS COM SUBQUERIES	218
TESTE DE RELAÇÕES VAZIAS	219
ALTERANDO OS DADOS COM UPDATE	219
REMOVENDO OS DADOS COM DELETE	222
ROLLBACK E COMMIT	223
TRUNCATE E DROP	224



unidade





TEMA DE APRENDIZAGEM 1

CONCEITO DE BANCO DE DADOS

ME. EDSON YANAGI
ESP. CARLOS DANILÓ LUZ

MINHAS METAS

- Compreender o que é um banco de dados.
- Características de sistema de banco de dados.
- Definindo o conceito de transação.
- Pontos positivos sobre o uso do sistema de gerenciamento de banco de dados.

INICIE SUA JORNADA

Seja bem-vindo(a) aluno(a)! Neste tema, compreenderemos o que é um banco de dados e entenderemos que a ideia de banco de dados não é uma coisa da atualidade, mas um assunto tratado há muitos anos. Trata-se de um tema em constante evolução em conjunto com a Tecnologia da Informação (TI). Você, aluno(a), pode manipular informações em um banco de dados sem perceber.

A ideia de banco de dados está em evolução. Por volta do início de 1980, surgiram os primeiros Sistemas de Gerenciamento de Banco de Dados (SGBD), criados inicialmente para resolver problemas de incompatibilidade e comunicação. Sabemos que há várias linguagens de programação, cada uma com sua própria forma de manipular um banco de dados, trazendo consigo várias melhorias em manipulação de dados. Conheceremos também alguns dos SGBDs mais utilizados no mercado, os quais contam com suas características individuais.

Os SGBDs mais conhecidos do mercado têm sua estrutura baseada no modelo de dados relacional, mas conheceremos os modelos que o antecederam, como o hierárquico e de rede, além de um conceito mais atual, o de orientado a objetos.

Para que você, aluno(a), compreenda a estrutura de um banco de dados, entenderemos melhor o modelo relacional, que tem a sua base fundada na matemática, na teoria de conjuntos e na lógica de predicados de primeira ordem, em uma visão geral de tabelas e colunas, na qual os dados estão contidos em linhas.

No modelo relacional, podemos representar as informações por meio de tuplas, atributos e relação. Por fim, veremos uma breve comparação entre os bancos de dados, MySQL e Oracle, uns dos mais conhecidos por empresas e desenvolvedores. Ainda, aprenderemos passo a passo como instalar e configurar o MySQL Server e MySQL Workbench, além do Oracle XE.

Espero que este tema enriqueça seus conhecimentos, vamos lá! Ótimo estudo!

VAMOS RECORDAR?

Sabemos que a ideia de banco de informações/dados veio antes da era dos computadores. Desde os primórdios, eles já tinham seu modo de armazenamento de informações por meio de livros e registros, por exemplo. Atualmente, com o advento tecnológico, ainda que não percebamos, manipulamos informações em um banco de dados por meio de um sistema. Com o crescimento da tecnologia e das diversas linguagens de programação, houve a necessidade de padronizar o acesso ao banco de dados. Eis que surgem os SGBDs, com o intuito de resolver problemas de acesso e confiabilidade de informações, atribuindo também o padrão de linguagem SQL.

Ao conhecermos o SGBD, entenderemos que sua base estrutural segue o modelo de dados relacional, que é utilizado pela grande maioria dos sistemas de gerenciamento. O modelo relacional tem sua base na matemática, com a representação das informações em tabelas e colunas. Ao compararmos o SGBDs MySQL e Oracle, por exemplo, considerados os bancos mais utilizados no mercado, é preciso refletir sobre os pontos positivos e negativos desses sistemas, para que, assim, consigamos chegar a uma consideração relevante.

DESENVOLVA SEU POTENCIAL

O QUE É UM BANCO DE DADOS

Caro(a) aluno(a), você já ouviu falar em banco de dados? Sabe o quanto o utilizamos em nosso dia a dia? Alguns devem ter pensado “eu não o uso em meu cotidiano”, mas estão enganados. Se pensarmos em qualquer empresa, assimilaremos a gestão a um sistema – e todo e qualquer sistema tem um banco de dados vinculado. Um usuário que utiliza algum desses sistemas está manipulando as informações em um banco de dados.

A ideia de banco de dados não surgiu na era dos computadores, já que desde os primórdios há formas de armazenamento de informações. Uma das mais tradicionais que conhecemos são os livros, que armazenavam informações de pessoas e de empresas, além de fichários, livros-ponto, cadernetas, arquivo morto, dentre outras. Alguns dos maiores problemas eram o de armazenagem, busca e

manutenção de informações, já que as informações eram escritas em folhas de papel e, muitas vezes, guardadas em inúmeras caixas.

Para Heuser (2009, p. 22), banco de dados é um “[...] conjunto de dados integrados que tem por objetivo atender a uma comunidade de usuários”. O que são dados, afinal? São as informações de um usuário, por exemplo: para uma fundação bancária os “dados do usuário” nada mais são que suas informações pessoais cadastradas junto à instituição.



Banco de dados é basicamente um sistema computadorizado de manutenção de registros; em outras palavras, é um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando solicitar. (DATE, 2003, p. 06)

Podemos compreender que o banco de dados evoluiu em conjunto com a tecnologia, com o crescimento e a popularização das Tecnologias de Informação e Comunicação (TICs). Ao utilizar qualquer programa de gestão, as informações são transmitidas e armazenadas em um banco de dados.



PENSANDO JUNTOS

Será que ainda existem empresas que trabalham com um banco de dados como apenas um livro de registro?

PRINCIPAIS VANTAGENS DE UM BANCO DE DADOS

Ao compararmos o sistema tradicional de banco de dados, arquivos em papel, com o sistema digital, podemos destacar:

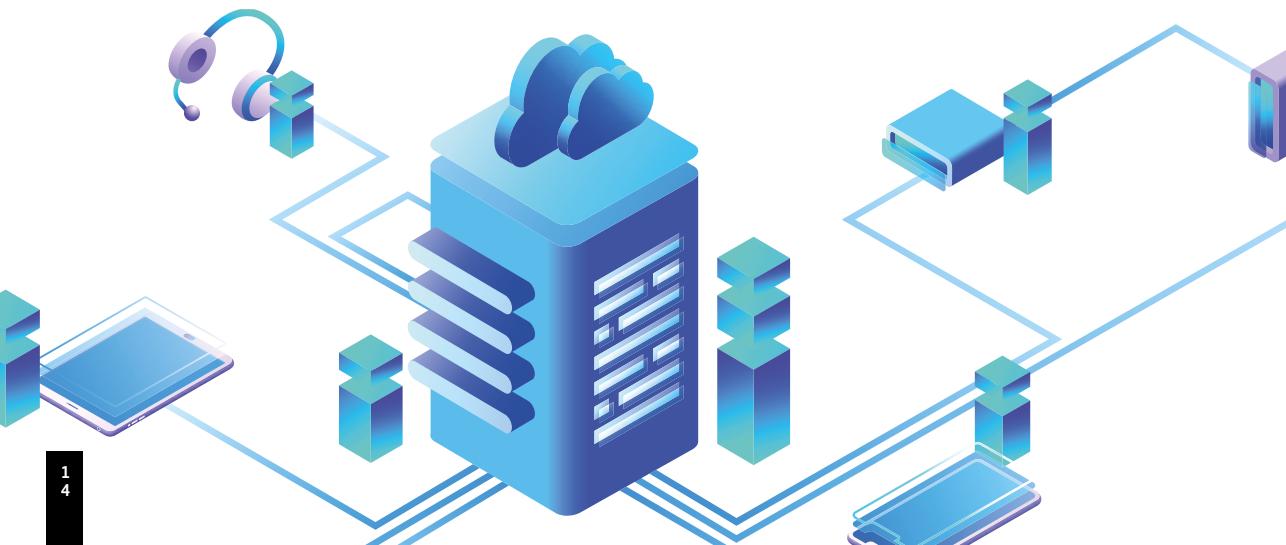
- **Volume** – como os dados são armazenados de forma digital, não há necessidade de arquivos de papel.
- **Agilidade nas informações** – os computadores têm maior capacidade de processamento. Ao requisitarmos informações ou atualizações, isso é efetuado em segundos, ao contrário do ser humano comum.

- **Menor trabalho** – com a eliminação do papel, a mão de obra é reduzida em função da organização de arquivos. As tarefas são sempre feitas pelo usuário final, por meio de um sistema.
- **Confiabilidade** – as informações são mais precisas e atualizadas, pois estão disponíveis a qualquer momento.
- **Proteção** – os dados não ficam expostos para qualquer pessoa acessar. Eles ficam bem protegidos, só com pessoas autorizadas tendo acesso.
- **Compartilhamento de informações** – pelo fato de os dados ficarem disponíveis a qualquer momento, vários departamentos têm acesso às informações simultaneamente.

SISTEMAS DE GERENCIAMENTO DE BANCO DE DADOS (SGBD)

Você já deve saber que temos vários tipos de aplicações e linguagens de desenvolvimento, tais como: Java, Delphi, PHP, C# e outras. Podemos considerar como linguagens primordiais: COBOL, Basic, C etc.

No início da década de 1980, surgiram os primeiros padrões de acesso a dados. Identificou-se que eles continham muitas funcionalidades em comum, por exemplo, a interface gráfica para interagir com o usuário. Porém, nem toda parte do código é igual, já que cada linguagem e programa contêm rotinas e funcionalidades diferentes para se manipular as informações.



Pensando em uma forma de comunicação padronizada de acesso para as linguagens, surgiram os **Sistemas de Gerenciamento de Banco de Dados (SGBDs)**, justamente para solucionar a incompatibilidade de acesso pelas linguagens.

Para Heuser (2009, p. 22), SGBD é um “[...] software que incorpora as funções de definição, recuperação e alteração de dados em um Banco de Dados”. Assim, fica a cargo dele o armazenamento e o controle das informações.

Segundo Amadeu (2015, p. 26), o “SGDB é parcialmente responsável por garantir que todo estado do Banco de Dados seja um estado válido, ou seja, um estado que satisfaça a estrutura e as restrições especificadas no esquema”.

Encontramos no mercado vários tipos de SGBD, sendo que alguns deles serão apresentados mais à frente. O modelo de dados mais utilizado é o relacional. O padrão adotado para a comunicação e o acesso é o *Standard Query Language* (SQL). Dessa forma, foi possível efetuar a separação entre o sistema e os processos de manipulação de um banco de dados, sem termos funcionalidades exclusivas para cada tipo de linguagem.

Observamos a Figura 1, que mostra a representação simples de como é a interação do SGBD com os componentes: dados, hardware, software e usuários.

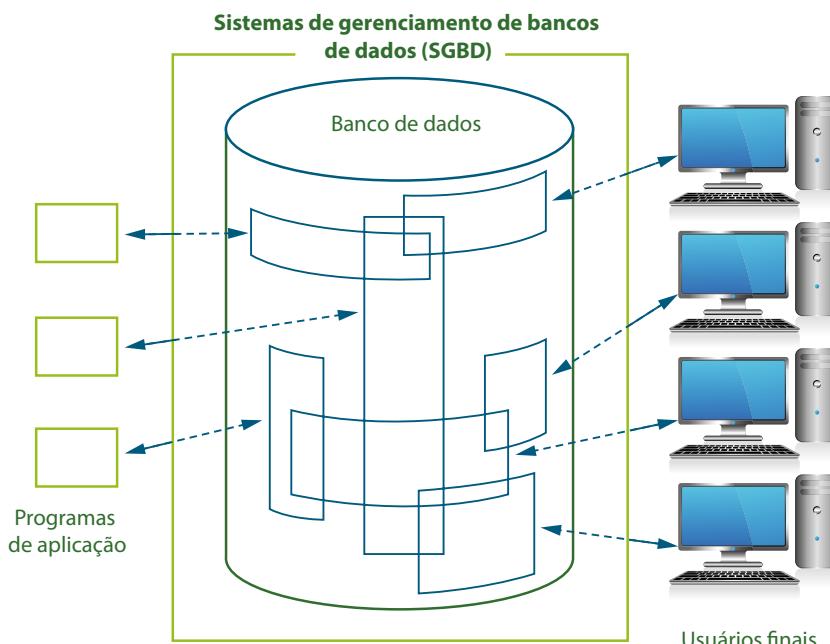


Figura 1: Representação de um SGBD / Fonte: Adaptada de Date (2003, p. 5).

Características de SGBD

Além da padronização da linguagem e da separação entre aplicação e banco de dados, um SGBD nos traz outras vantagens.

Gerenciar redundância de dados

Vamos utilizar como exemplo uma universidade e seus departamentos: secretaria acadêmica e financeiro. No sistema convencional de arquivos, em cada setor, teríamos os mesmos dados de cada aluno(a), gerando, assim, uma redundância de informações. Toda vez que fosse necessário a modificação de dados em um grupo, todos os demais deveriam efetuar a modificação para manter os dados por igual, podendo haver inconsistência ou falha da informação. Ao utilizar o SGBD, a informação será única no sistema; quando houver alguma alteração, todos os departamentos estarão alinhados.

Gerenciamento de acesso

Devemos nos lembrar que, em um grande sistema, temos vários usuários, e a interação com um banco de dados é igual. Ao utilizar o SGBD, é possível criar usuários ou grupos com acessos privilegiados, nos quais alguns grupos não podem ter acesso a outros. Continuando com o exemplo da universidade, os dados financeiros do(a) acadêmico(a) podem ou não ser visualizados pelos demais grupos, ou apenas alguns setores e usuários podem efetuar modificações dos dados.

Consultas eficazes

Pelo fato de as informações contidas em um banco de dados estarem armazenadas em disco, o SGBD fornece funcionalidades ágeis para efetuar consultas e atualizações de forma mais eficiente. São utilizados objetos chamados *indexs* (indexados), eles têm como base os dados em árvores ou *hash*. Ao efetuar a consulta, os dados são transferidos para a memória do computador de modo temporário para agilizar a execução, sem ter que acessar o banco de dados inteiro a cada nova consulta. Os

módulos responsáveis pela escolha da consulta mais ágil, baseando-se nos dados já armazenados, são os de processamento de consulta e otimização.

Backup e restauração

Já sabemos que podem ocorrer falhas de hardware ou software. Por exemplo, imagine que estamos trabalhando normalmente e temos uma falha de software: e agora, o que fazer? Nesse momento, o subsistema de backup e recuperação entra em ação. Ele é responsável por alocar os dados no sistema do mesmo modo que estavam antes da execução. O subsistema tenta restaurar os dados até o ponto do problema. É também por meio do SGBD que temos ferramentas de fácil manipulação, podendo ser utilizadas para efetuar backup, restauração e transferência entre servidores ou banco de dados, sendo em formato digital (texto ou binário).

SGBD NO MERCADO

Prezado(a) aluno(a), agora que já vimos as principais funcionalidades e benefícios de se utilizar um SGBD, conheceremos os mais utilizados por empresas e desenvolvedores.

MySQL

O MySQL foi criado pela empresa MySQL AB e teve os primeiros manuais desenvolvidos por David Axmark, Allan Larsson e Michael (Monty) Widenius. O grande sucesso do MySQL se deve à grande facilidade de interagir com a programação PHP (*Hypertext Preprocessor*), sendo considerado um dos principais no desenvolvimento de websites. Em virtude de ser suportado por vários sistemas operacionais, ele sempre está disponível em pacotes de hospedagem de sites na internet.

Ele ganha destaque pelo seu desempenho, ótima estabilidade e por não exigir grandes recursos de hardware para seu uso, compatível com várias linguagens além do PHP, como Java, Delphi, Python e ASP (MySQL AB, 2001, s.p.).

Em 16 de janeiro de 2008, a empresa desenvolvedora do MySQL foi comprada pela Sun Microsystems. O MySQL possui duas formas de distribuição: *open source* com base na GPL e com licença comercial.

 ZOOM NO CONHECIMENTO

O que seria uma licença General Public License (GNU)? Em uma tradução para o português, seria licença pública geral. Também conhecida como software livre, ela tem como base quatro principais termos:

- Todos podem executar o programa para qualquer propósito.
- Todos podem estudar como o software funciona e ajustá-lo para as suas necessidades. O acesso ao código-fonte é uma característica dessa liberdade.
- A liberdade de distribuir cópias com o propósito de ajudar ao próximo.
- Manipular e aperfeiçoar o programa, distribuir as alterações de modo que toda a comunidade faça proveito e se beneficie.

Para mais informações sobre a GPL, acesse o QR Code.

Fonte: adaptado de GNU ([2016], s.p.).

Oracle

Foi desenvolvido pela empresa Oracle Corporation por meio de seus fundadores Larry Ellison, Bob Miner e Ed Oates. Larry Ellison viu uma oportunidade de desenvolver um projeto que atende a uma falha dos demais concorrentes. Em 1977, os sistemas eram enormes e com software ineficientes, somente técnicos bem treinados conseguiam manipular e gerir a entrada e a saída de dados. Em 1978, entrava no mercado o Oracle Versão 1, que, por sua vez, revolucionou a computação empresarial (ORACLE, [2016], s.p.). Algumas das principais características do Oracle são:

- Ótima performance para trabalhar com grande volume de dados.
- Destaque sobre o volume de armazenamento e capacidade de expansão.
- Segurança e confiabilidade dos dados armazenados.
- Carregamento de vários tipos binários: imagens, sons e vídeos.
- Sistema multiusuário, por meio de consultas simultâneas, atualizações e edições dos dados armazenados.

O banco de dados Oracle contempla diversas versões e licenças do produto, sendo que a empresa Oracle Corporation subdivide seu produto em edições por questões de controle de licenças. Veremos agora algumas dessas edições.

Standard Edition (SE)

Contempla funcionalidades básicas. Essa versão tem como base da licença a quantidade de usuários ou de *sockets*, tendo um limite para servidores com até quatro *Computer Processors* (CPUs). Caso a empresa deseje aumentar a capacidade de CPUs, deve migrar para a licença *Enterprise*. Se apresentar problemas como limite de memória, pode utilizar o Oracle RAC sem custo adicional para efetuar uma clusterização.

Standard Edition One

A versão está lançada com algumas restrições de funcionalidades, sendo destinada para sistemas que possuem um ou dois CPUs, não tendo limitações de memória.

Enterprise Edition (EE)

Conta com as funcionalidades da *Standard Edition* e algumas adicionais, dando destaque às áreas de performance e segurança. Essa licença tem como base o número de usuários ou de CPUs. Normalmente, é destinada para servidores com quatro ou mais CPUs. Esta versão não tem limite de memória e pode utilizar clusterização por meio do Oracle RAC.

Express Edition (Oracle Database XE)

Surgida em 2005, a licença GPL (*General Public License*) e livre para distribuição contempla os sistemas operacionais Windows e Linux. O tamanho desta versão para distribuição é em torno de 150 MB e se restringe ao uso de apenas um CPU, tendo também um limite de, no máximo, 4 GB de dados por usuário e 1 GB (Gigabyte) de memória. O suporte ela é efetuado por meio de comunidades on-line. No final deste tema, será demonstrado como efetuar a instalação da versão Oracle Database XE.

SQL Server

SGBD é desenvolvido e distribuído pela empresa Microsoft. Ele teve seu início de desenvolvimento em parceria com a Sybase, com foco nas plataformas da IBM (*International Business Machines*). Após o surgimento do Windows NT, tornou-se exclusivo para o sistema operacional Windows. Vendo sua perda de espaço no mercado para os concorrentes, a empresa anunciou em 2016 a versão do SQL Server para o Linux, tentando, assim, atingir a meta de SGBD mais utilizado no mundo corporativo. Algumas de suas principais funções são:

- Replicação de dados entre servidores.
- Manipulação de dados OLAP (*On-line Analytical Processing*).
- Migração das informações do banco de dados Oracle para SQL Server.
- Armazenamento de dados OLTP (*On-line Transaction Processing*).
- *Functions*.
- *Triggers*.

EDIÇÃO / VERSÃO	CARACTERÍSTICAS
<i>Enterprise</i>	O SQL Server Enterprise oferece recursos abrangentes de <i>datacenter</i> de ponta para requisitos exigentes de bancos de dados e <i>business intelligence</i> .
<i>Standard</i>	O SQL Server Standard oferece as principais funcionalidades de gerenciamento de dados e <i>business intelligence</i> para <i>workloads</i> não críticas com recursos mínimos de TI.
<i>Developer</i>	O SQL Server Developer Edition agora é uma edição gratuita que permite aos desenvolvedores aproveitarem todos os recursos avançados no SQL Server. Essa edição destina-se apenas a ambientes de desenvolvimento e testes, não a ambientes de produção, nem ao uso com dados de produção.

EDIÇÃO / VERSÃO	CARACTERÍSTICAS
<i>Business intelligence</i>	O SQL Server Business Intelligence capacita organizações a criarem e implantarem soluções corporativas de BI self-service seguras, escaláveis e gerenciáveis.
<i>Express</i>	O SQL Server Express é uma edição gratuita do SQL Server ideal para desenvolvimento e capacitação de aplicativos para a área de trabalho, web e pequenos servidores.

Tabela 1: Edições disponíveis para SQL Server / Fonte: Microsoft ([2016], s.p.).

 **EU INDICO**

Se você, aluno(a), também ficou como nós, empolgados, com a notícia sobre a versão do SQL Server para o Linux, acesse o link e obtenha mais informações.

PostgreSQL

PostgreSQL, originalmente chamado de Postgres, foi criado na Universidade de Berkeley Califórnia pelo professor de ciência da computação Michael Stonebraker. O Postgres começou em 1986 com base no projeto Ingres entre 1986-1994. O projeto teve o papel de abrir e explorar novos conceitos de banco de dados.

Durante o período, Postgres passou por várias mudanças em seu código, sendo inseridas regras, procedimentos, tipos extensíveis com índices e conceitos objeto-relacional. Em 1996, surge o PostgreSQL, elaborado por Marc Fournier, Bruce Momjian e Vadim B. Mikheev, que incluiu o interpretador SQL ao projeto e distribuiu a base na GPL. Essa etapa contribuiu com a popularização do banco pelo mercado e transformou radicalmente Postgres, trazendo também a visão de um novo banco de dados, que ganhou uma reputação de uma rocha sólida e estável.

O PostgreSQL começou na versão 6.0 com a ajuda de centenas de programadores pelo mundo. O sistema foi modificado e melhorado em quase todas

as áreas. Ao longo dos anos, foi tendo grandes melhorias em recursos e SQL, além de novos tipos de dados nativos (novos tipos de datas e hora e tipos geométricos).

Vamos ver algumas das principais características:

- Controle de Concorrência de Multiversão (MVCC).
- *Commit* e *rollback*.
- *Triggers* e *constraints*.
- Sem restrições em tamanho de registro.
- Aceita os tipos de índices B-Tree, rTree e *hash*.

Por ser um projeto *open source*, ele conta com milhares de programadores empenhados na melhoria de seus códigos e se mantém por meio de patrocínios e doações. Atualmente, a versão mais estável é a 9.4 e traz consigo a ferramenta PG Admin III, destinada a uma maior facilidade na manipulação dos dados.

Os SGBDs apresentados têm como base o modelo de dados relacional. SILBERSCHATZ; KORTH; SUDARSHAN *et al.* (2006, p. 15) o definem como sendo “[...] um conjunto de tabelas para representar tanto os dados quanto as relações entre eles”.

Pelo fato de as informações serem representadas de forma simples, sua implementação é utilizada pela maioria dos SGBDs do mercado e temos que entender também, prezado(a) aluno(a), qual o conceito de modelo de dados.



Modelo de Dados: uma coleção de ferramentas conceituais para descrever dados, relações de dados, semântica de dados e restrições de consistência. Um modelo de dados oferece uma maneira de descrever o projeto de um Banco de Dados no nível físico, lógico e de view. (SILBERSCHATZ; KORTH; SUDARSHAN *et al.*, 2006, p. 5)

O modelo de dados pode ser considerado também um conjunto de operações para recuperar e alterar a base de dados. Como não temos apenas um tipo de modelo de dados, vamos conhecer um pouco dos demais modelos.

MODELO DE DADOS HIERÁRQUICO

O modelo hierárquico **tem sua representação de dados em uma estrutura de árvore ou hierarquia**. Cada nível da árvore contém um registro de dados, sendo que cada registro armazena uma coleção de informações de modo individual.

Outra ideia de representação seria a de registro principal, secundário, terciário etc., sendo que o principal vai sempre estar no topo e tendo a leitura de dados abaixo de seu nó da esquerda para direita.

Esse modelo de dados pode ser apresentado por meio de um diagrama, no qual as caixas representam os dados registrados e as linhas fazem a correlação entre as informações. Veja o exemplo na Figura 2.

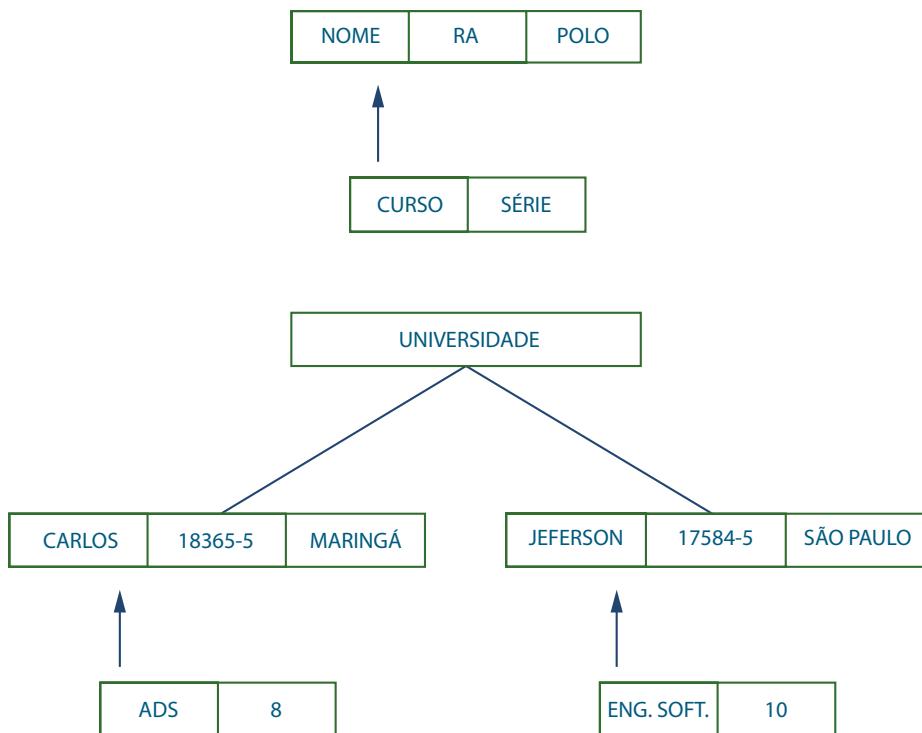


Figura 2: Exemplo de diagrama de modelo de dados hierárquico
Fonte: adaptada de Silberschatz, Korth e Sudarshan (2006).

O modelo hierárquico conta com alguns problemas, sendo os três principais:

- Dificuldade na representação dos diagramas de árvore com grandes volumes de informações.
- Muita complexidade de efetuar consultas, tendo que ler vários nós até encontrar a informação correta.
- Inconsistência das informações quando tivermos atualizações dos dados.

MODELO DE DADOS REDE

O modelo de rede foi criado como uma extensão do modelo hierárquico, sendo **seu principal diferencial permitir que um mesmo tipo de registro esteja agrupado em um único setor**. Com base na Figura 2 do modelo hierárquico, temos as informações dos alunos separadas em linha e as informações da instituição em um nível abaixo. O modelo de rede agrupa todos os dados dos alunos em um único lugar e os da instituição em outro.

O sistema de *Data Base Task Group* (DBTG) normatiza esse modelo de dados, sendo seu foco garantir a recuperação dos dados para as aplicações do mesmo modo que estão armazenadas.



A representação desse modelo por meio de um diagrama é muito similar ao do hierárquico, tendo apenas como diferencial o agrupamento das informações, conforme mostra a Figura 3.

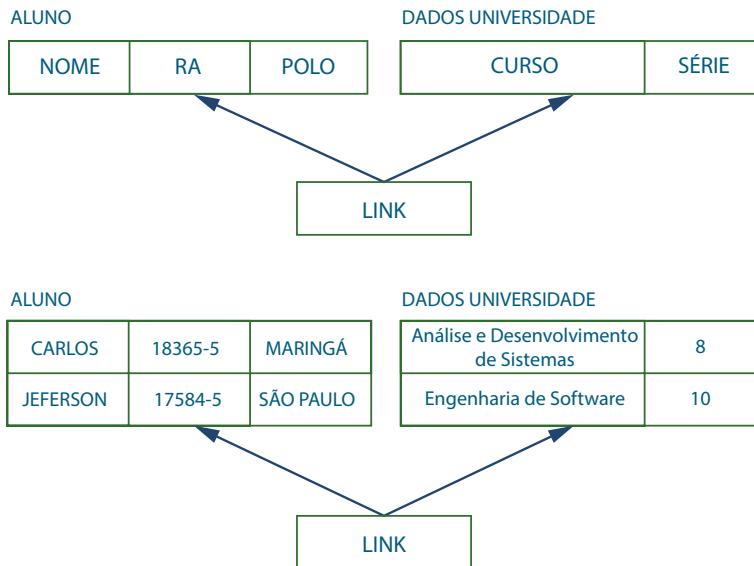


Figura 3: Exemplo de diagrama modelo de dados de rede
Fonte: adaptada de Silberschatz, Korth e Sudarshan (2006).

MODELO DE DADOS RELACIONAL

O modelo de dados relacional teve sua origem pensando na necessidade de aumentar a independência dos SGBDs. Seu princípio é focado em armazenamento e recuperação de dados. Hoje, esse modelo é utilizado por grande maioria dos SGBDs, sendo o principal fato a organização de informações.

Sua estrutura básica é formada por tabelas e colunas. Ao contrário dos modelos anteriores, o acesso às informações não tem caminhos a serem percorridos, já que as informações estão sempre no mesmo nível. Alguns cuidados devem ser tomados com base nesse modelo para evitarmos alguns problemas, como as repetições de dados ou a perda de informações. O indicado é trabalharmos com índices e chaves para evitá-los. Com base nas informações da Figura 4, modelo de dados de rede, veremos um diagrama do modelo relacional.

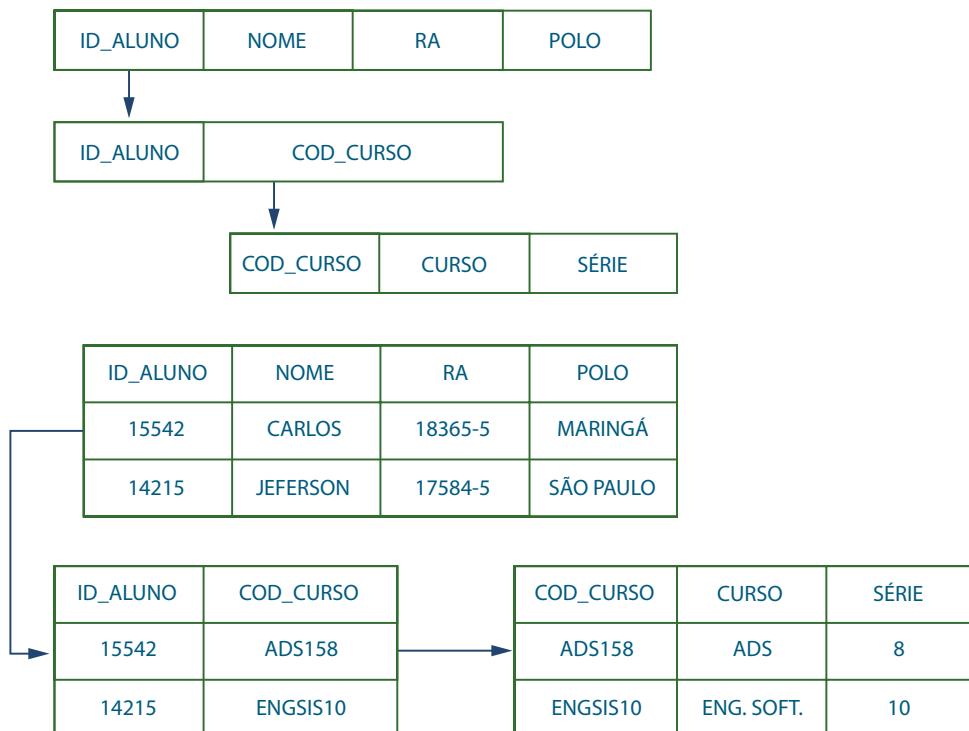


Figura 4: Exemplo de diagrama modelo de dados relacional
Fonte: adaptado de Silberschatz, Korth e Sudarshan (2006).

MODELO DE DADOS ORIENTADO A OBJETOS

Esse modelo surgiu em função de limite de armazenagem de dados e a forma de representar as informações apresentadas no modelo relacional. Uma das características da programação orientada a objetos seria a **habilidade de manipular e criar tipos de dados conforme a necessidade**, por meio de bibliotecas de classes, mas essas estruturas de dados utilizam o armazenamento permanente.

O grupo *Object Database Management Groups* (ODMG) desenvolveu a estrutura padrão para banco de dados orientado a objetos, grupo esse representado pelos principais SGBDs disponíveis no mercado.

Alguns dos possíveis problemas sobre o modelo relacional foram solucionados. Acredita-se que esse tipo de banco deve ganhar mais mercado nos próximos 10 anos. Acredita-se também que esses tipos de bancos de dados orientados a objetos sejam destinados às aplicações especializadas e preparadas para eles, mas

o modelo relacional ainda toma grande parte do mercado por sustentar os sistemas de negócios tradicionais.

Conforme utilizado nos modelos anteriores, vamos verificar o diagrama do modelo de dados orientado a objetos. Para representá-lo, é utilizada a Linguagem de Modelagem Unificada – *Unified Modeling Language* (UML).

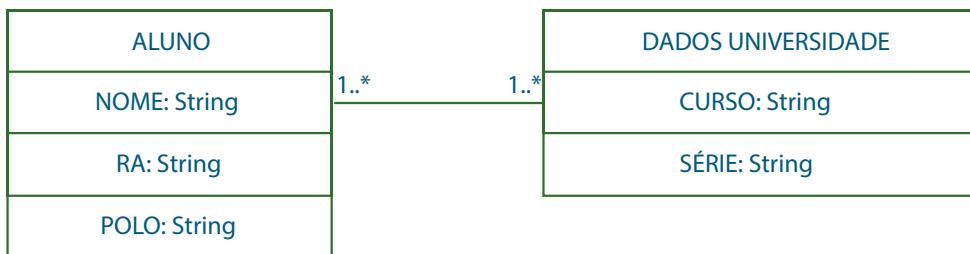


Figura 5: Exemplo de diagrama UML, modelo de dados orientado a objetos
Fonte: adaptado de Silberschatz, Korth e Sudarshan (2006).



MODELO RELACIONAL

Ao pensarmos em um banco de dados, devemos entender a sua forma de estrutura básica. Inúmeros SGBDs têm como base o modelo relacional e, por esse motivo, você, aluno(a), deve compreender esse modelo para estruturar seu banco de dados de modo adequado.

O modelo relacional surgiu em 1970 por Ted Codd, da IBM Research, tendo como conceito a relação matemática, tabelas e valores. Sua teoria está vinculada à teoria de conjuntos e à lógica de predicados de primeira ordem. O uso do modelo relacional em sistemas comerciais se deu no início dos anos 1980, com o surgimento do SGBD Oracle e SQL/DS para o sistema operacional da IBM. Desde então, esse modelo vem sendo seguido em inúmeros SGBDs do mercado.



O modelo relacional representa o Banco de Dados como uma coleção e relações. Informalmente, cada relação se parece com uma tabela de valores ou, em alguma extensão, com um arquivo de registros “plano”. (ELMASRI; NAVATHE, 2005, p. 90)

Ao pensarmos em uma tabela, as informações contidas em cada linha representam uma coleção de dados relacionados, sendo que cada valor representa um fato ou descrição. A sua representação por meio do modelo relacional contém a tupla que representa uma linha, atributos seriam as colunas do cabeçalho e a relação, o nome da tabela.

Veremos, a seguir, um exemplo utilizando o modelo relacional, contendo seus atributos, tuplas e relação.

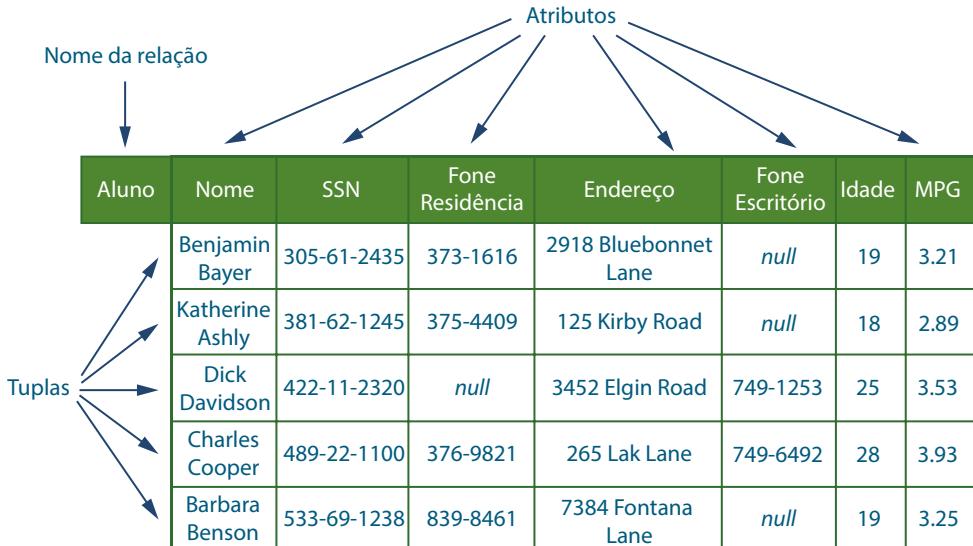


Figura 6: Representação de informações no modelo relacional

Fonte: Adaptada de Elmasri e Navathe (2005, p. 91).

DIFERENÇAS ENTRE O MYSQL E O ORACLE

Desenvolver um software que realize o acesso às informações de diferentes fabricantes de bancos de dados pode se tornar complexo pelas particularidades nas instruções SQL e em cada SGBD. Por isso, torna-se necessário demonstrar as principais diferenças entre as instruções SQL utilizadas no SGBD MySQL e no Oracle.

Outra consideração importante sobre as diferenças entre os SGBDs está na evolução do universo que abrange o mundo dos negócios. Além de muitas empresas criarem a necessidade de softwares com acesso às informações dos mais variados bancos de dados e de diferentes fabricantes, ainda há a necessidade de se adaptar a novas oportunidades do mercado que exigem o conhecimento da linguagem SQL e suas variações para trabalhar com os SGBDs que serão utilizados.

Ao abordar as principais diferenças nas instruções SQL utilizadas pelos SGBDs MySQL e Oracle, serão apresentadas as particularidades de cada um deles. Como a evolução dos recursos da TI exige a constante procura das empresas por novas tecnologias para um diferencial nesse competitivo mercado, as inovações acabam sendo uma questão de necessidade, como é o caso de organizações que sofrem uma evolução rápida em seus negócios.

Essa evolução pode envolver diversas tecnologias e produtos de diferentes fabricantes. Um grande exemplo disso é a necessidade de adquirir ou desenvolver um novo sistema para melhorar o fluxo de informações interno ou até externo.

Outra situação típica da utilização de SGBDs de diferentes fabricantes é a fusão de empresas, na qual as informações não podem ser descartadas e necessitam de acesso e integração às bases de dados antes individualizadas.

Com base em situações como essas, é possível compreender que a própria linguagem SQL tem uma base padrão na qual as instruções são utilizadas em praticamente todos os SGBDs. Contudo, existem diferenças ou pequenos ajustes que devem ser observados. A seguir, estão apresentadas algumas dessas diferenças.

A primeira diferença fundamental é na estrutura criada para o MySQL, em que são criadas várias bases diferentes para projetos diferentes em um mesmo servidor, enquanto no Oracle, um servidor significa ter apenas uma base de dados. Cada usuário criado enxerga apenas seu *schema*, ou seja, o conjunto de suas tabelas, dados, *triggers*, *procedures* etc., sendo que cada usuário tem as suas visualizações.

No MySQL, são criados *databases* para cada projeto; no Oracle, são criados usuários para cada projeto ou até *tablespaces*, que são uma ou mais unidades de armazenamento lógicas que armazenam coletivamente todos os dados do banco de dados.

Para iniciar um projeto no Oracle, é necessário logar com o usuário SYS e criar um novo usuário para o projeto. Não é indicado utilizar o usuário SYS nos projetos. Um detalhe importante é que a Oracle possui o Oracle Database 11g Express Edition (Oracle Database XE), ou simplesmente Oracle xe, que é uma versão gratuita do banco de dados relacional com determinadas limitações comerciais.

Sua instalação é simples e fácil de gerenciar e de desenvolver. A interface é intuitiva, acessada por meio de um navegador para administração do banco de dados, criação de tabelas, exibições e outros objetos de banco de dados, importação, exportação e dados de exibição de tabela, executar consultas e *scripts* SQL e gerar relatórios.

Configurando o MYSQL

Partindo do ponto que você, aluno(a), já deve ter lido nesta unidade sobre as principais características do MySQL, vamos para a instalação e a configuração dele. Primeiramente, vamos acessar o site oficial do MySQL1.



Figura 7: Local onde devemos encontrar o link para download / Fonte: o autor.

Após o acesso à página, clique na aba “Downloads” para ter acesso a uma outra página, como mostra a Figura 8, que deverá seguir para a página “Community”:

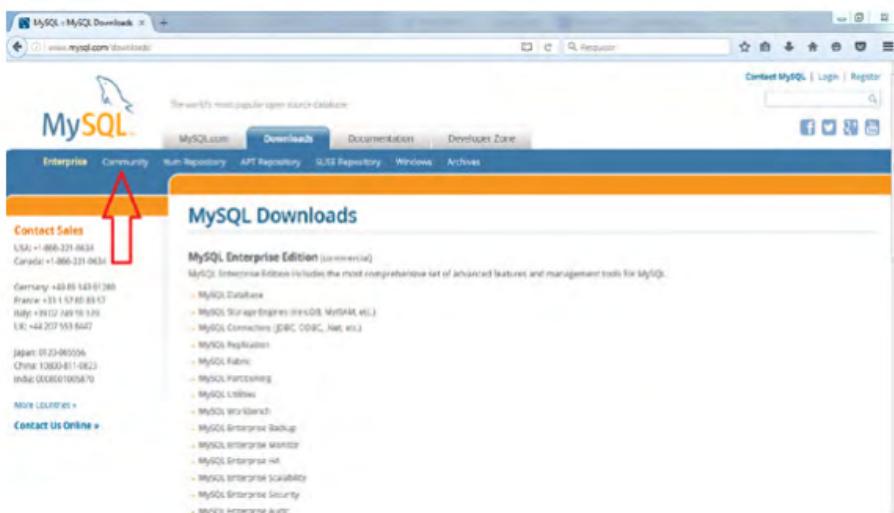


Figura 8: Informação de qual o tipo de download / Fonte: o autor.

Agora, vá até o final da página e clique em “Community (GLP) Download”:



Figura 9: Informação de qual versão deve efetuar o download / Fonte: o autor.

Chegou o momento de selecionarmos o tipo de download a ser efetuado, MySQL Windows (*installer* e *tools*), devendo-se sempre selecionar o maior arquivo para download.

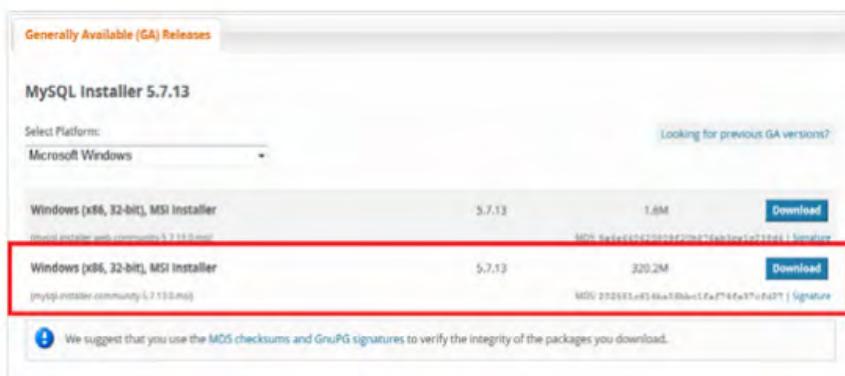


Figura 10: Qual a opção correta para *download* / Fonte: o autor.

Para realmente efetuar o download, é necessário se cadastrar, já ser cadastrado ou ignorar o login e clicar em:

Begin Your Download - mysql-installer-community-5.7.13.0.msi

[Login Now or Sign Up for a free account.](#)

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system
- Comment in the MySQL Documentation

[Login »](#)

using my Oracle Web account

[Sign Up »](#)

for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can sign up for a free account by clicking the Sign Up link and following the instructions.

[No thanks, just start my download.](#)

Figura 11: Informação de como proceder para efetuar o download / Fonte: o autor.

Agora com o download completo, basta darmos dois cliques sobre ele para iniciar a execução e a instalação do nosso MySQL. Após o início da instalação, será necessário concordar com a licença de uso, conforme a imagem a seguir:

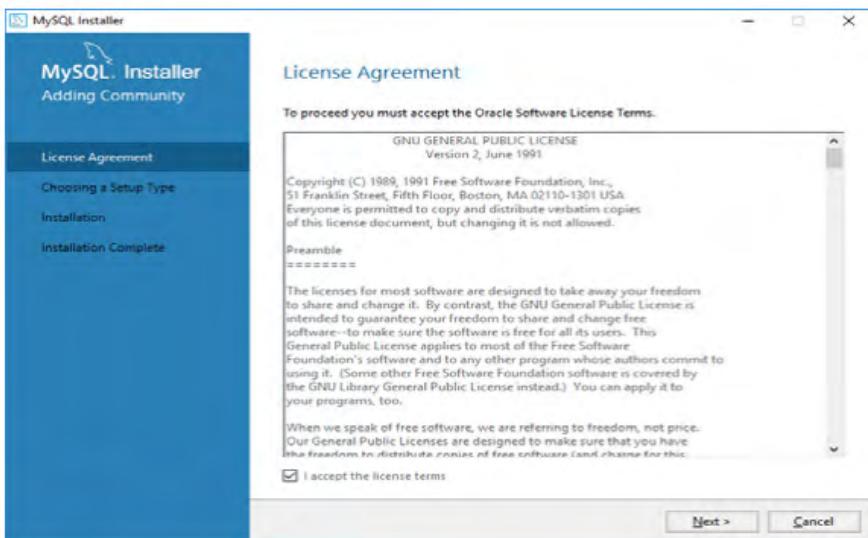


Figura 12: Primeira tela de execução da instalação / Fonte: o autor.

Para prosseguirmos com a instalação, vamos escolher a opção personalizado (*custom*), selecionar as opções destacadas a seguir e, por fim, clicar em próximo (*next*):

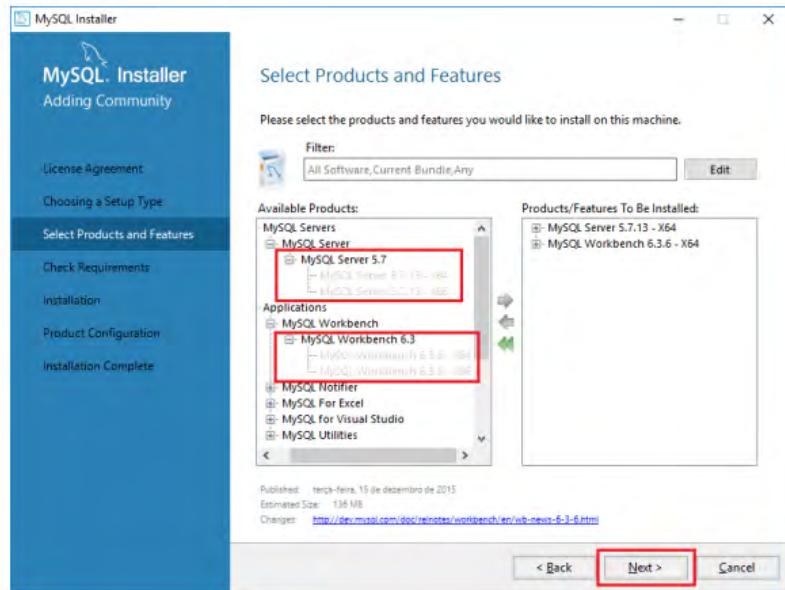


Figura 13: Opções a serem escolhidas para a instalação / Fonte: o autor.

Para irmos ao passo seguinte, é necessária a instalação do Microsoft Visual C++. Basta selecionar conforme tela a seguir e clicar em “execute”:

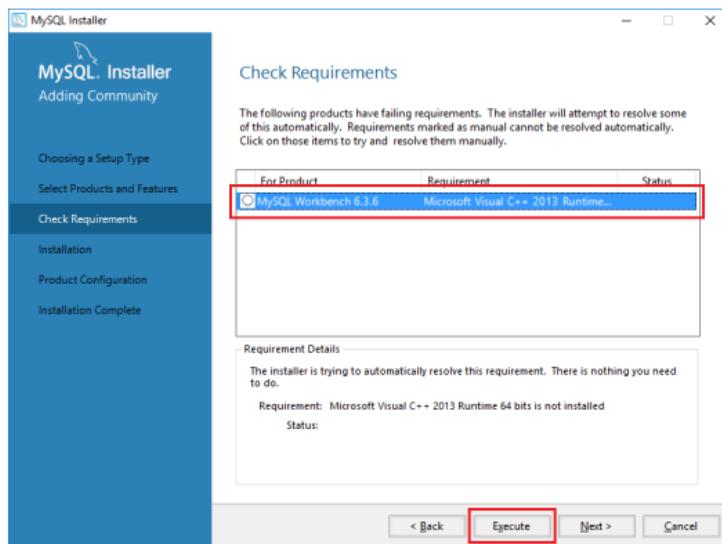


Figura 14: Opções a serem escolhidas para a instalação / Fonte: o autor.

Clicando em “executar”, basta apenas concordarmos com a licença do Visual C++ e clicar em “*Install*”:

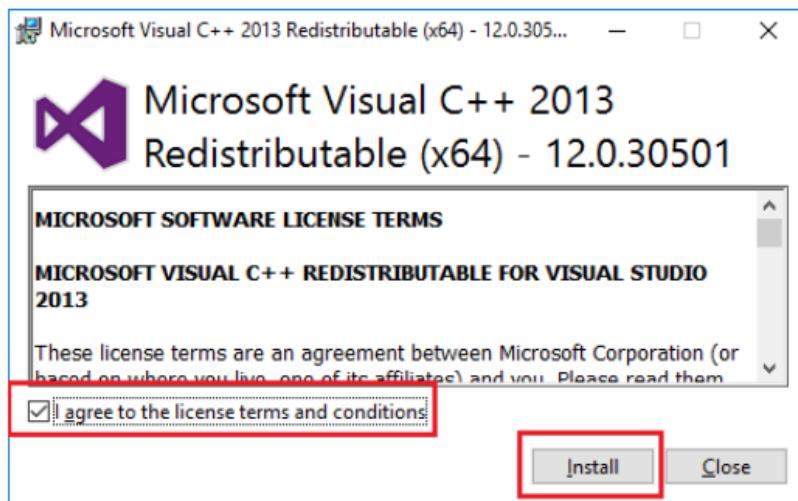


Figura 15: Instalação de dependências, para executar o MySQL / Fonte: o autor.

Agora com o Visual C++ já instalado, a tela da instalação do nosso MySQL ficará marcada, indicando que já possuímos o Visual C++:

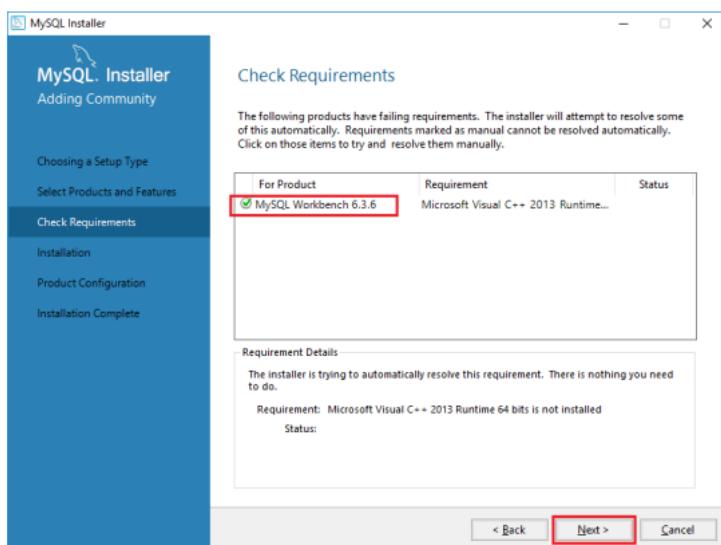


Figura 16: Opções a serem escolhidas para a instalação / Fonte: o autor.

Agora basta clicar em “próximo” (*next*) para prosseguirmos com a instalação. Quando chegar à tela seguinte, apenas clique em “executar”:

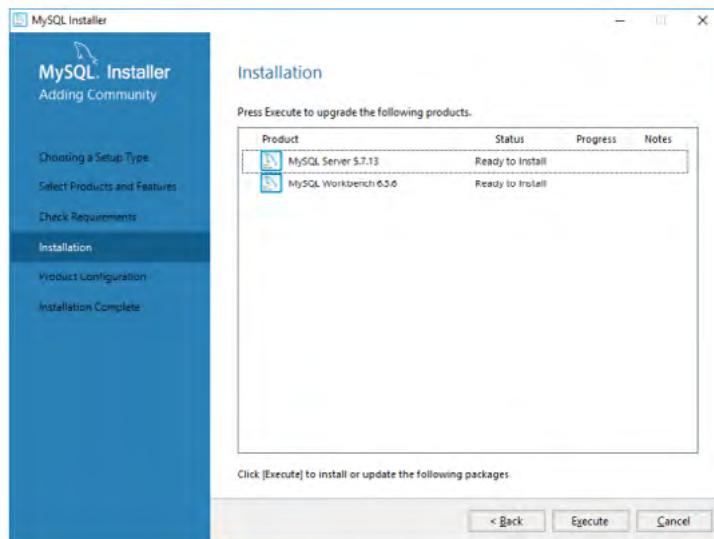


Figura 17: Opções a serem escolhidas para a instalação / Fonte: o autor.

Com a instalação finalizada, porém ainda com o banco de dados não configurado, a tela ficará da seguinte maneira:

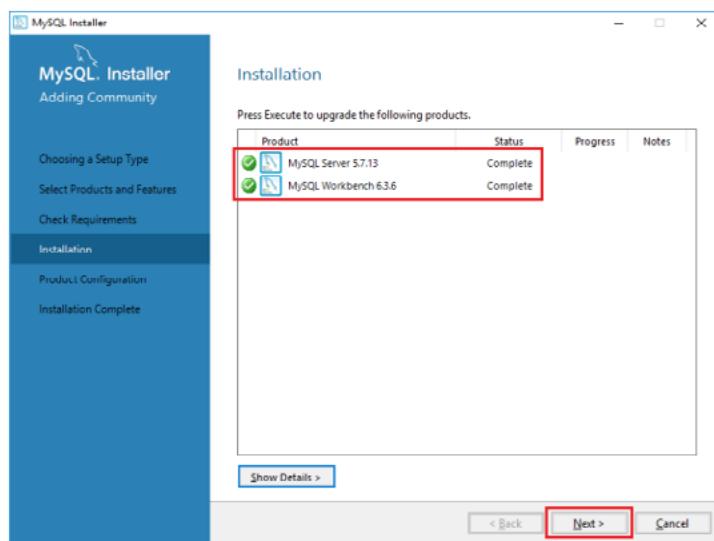


Figura 18: Opções a serem escolhidas para instalação / Fonte: o autor.

Para finalizar a instalação, basta clicar sequencialmente em “Next” até se deparar com uma tela solicitando a inclusão da senha para o banco de dados. Seja cauteloso com a senha, pois será necessário utilizá-la posteriormente. Segue a tela com a solicitação de senha:

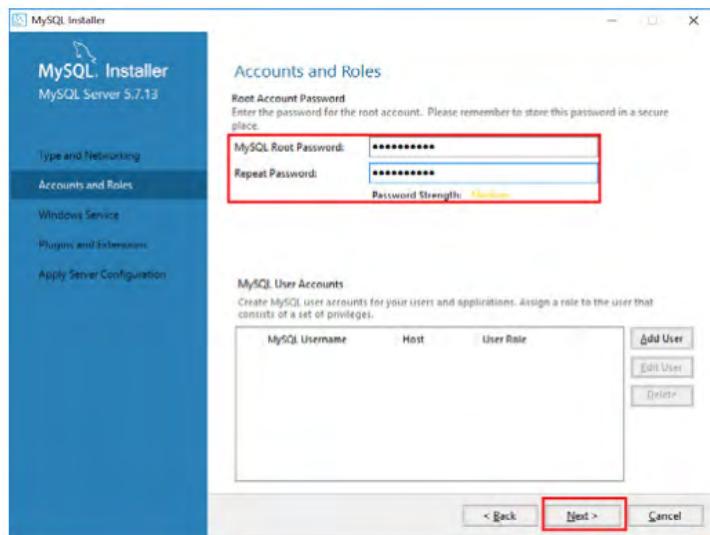


Figura 19: Opção de inserção, login e senha para acessar o MySQL / Fonte: o autor.

Para irmos ao próximo passo importante, devemos clicar em “Next” até chegar à seguinte tela:

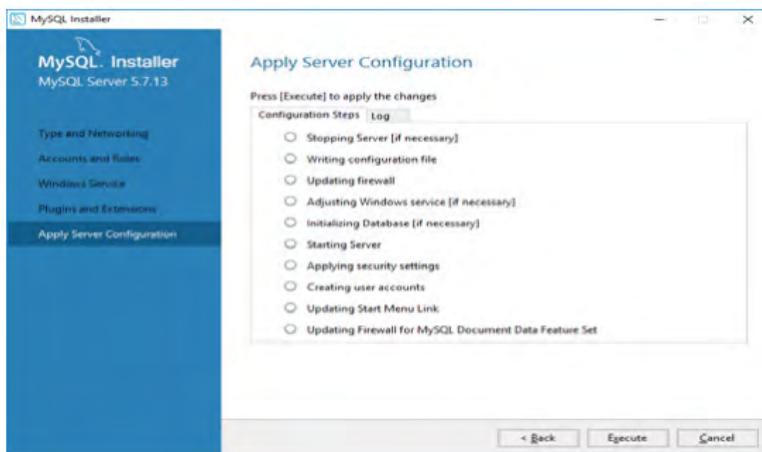


Figura 20: Opções a serem escolhidas para a instalação / Fonte: o autor.

Clicando em “Execute”, finalizamos o último passo da instalação do MySQL. Com a finalização da instalação, o banco de dados está pronto para ser acessado por meio do utilitário MySQL Workbench conforme a seguinte tela:

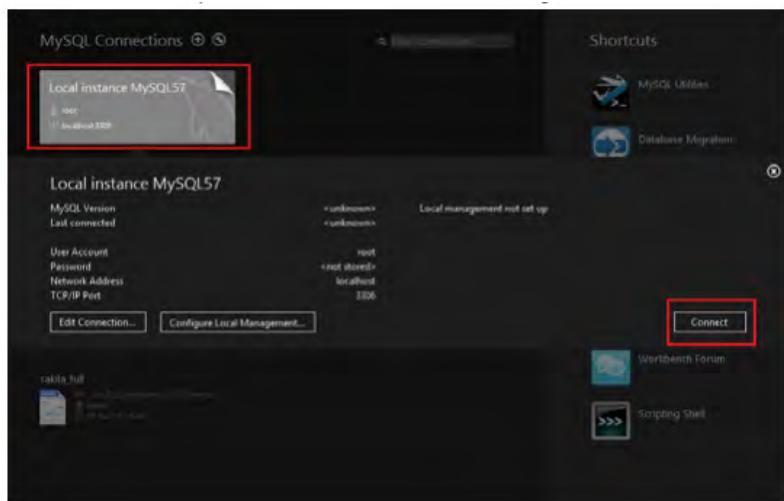


Figura 21: Primeira tela do programa instalado e sendo executado / Fonte: o autor.

Para conectar ao banco de dados instalado por meio dos passos acima, basta clicar sobre a instância criada, conforme destacado na imagem acima. Posteriormente, clique em “Connect”, informando a senha da instalação conforme a imagem a seguir:

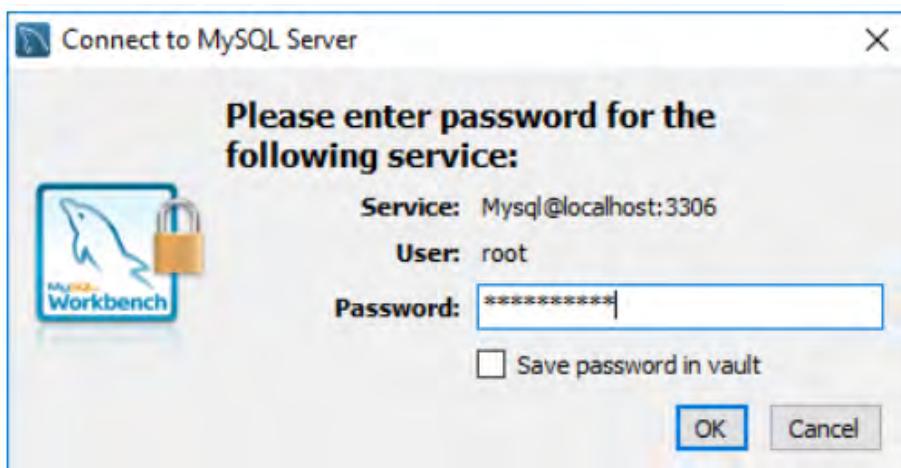


Figura 22: Inserindo login e senha para acesso ao banco de dados / Fonte: o autor.

Configurando o Oracle XE

O Oracle XE tem uma interface baseada em browser de usuário e é utilizado para administrar o banco de dados, executar scripts e consultas, a construção de aplicativos baseados na web, dentre outros.

Outra interface também utilizada para outras versões do Oracle, o SQL*PLUS, é uma interface pela qual é possível entrar e executar comandos SQL. Há vários comandos de SQL*PLUS, os quais podem facilitar processos e formatar resultados de comandos de SQL, podendo editar e até gravar.

Ao conectar o Oracle XE, é obrigatório informar “XE” como nome da base de dados. Em outra versões, apenas informar o endereço do servidor é o bastante, já que existe apenas uma base por servidor.

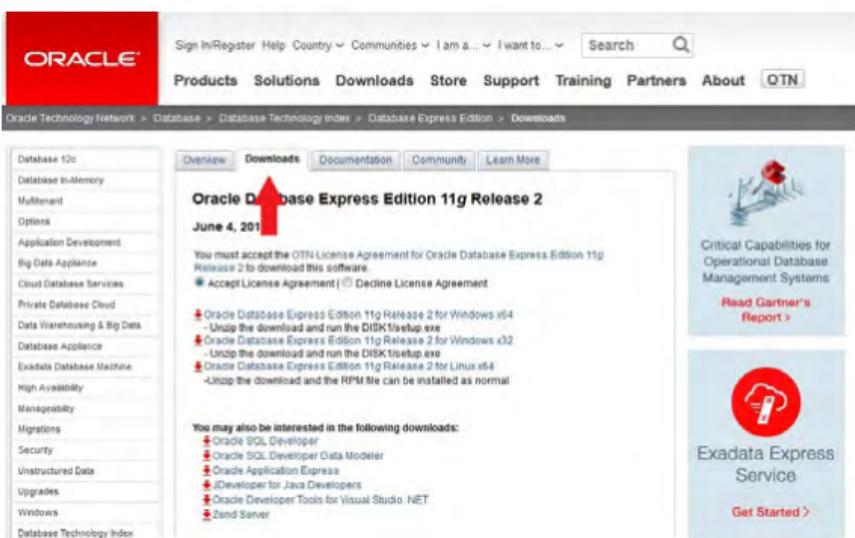


Figura 23: Opção de download do Oracle XE / Fonte: o autor.

- Após ter sido feito o download do instalador do Oracle, deve-se executá-lo e seguir os passos.
- Registrar o cadastro de uma senha para o usuário *system*.

DICA

Geralmente, o próprio instalador configura como serviço do Windows, mas como exige-se muitos recursos, em “Serviços” é possível configurar o serviço OracleXETNSListener e OracleServiceXE para iniciar manualmente. Assim, quando necessitar do banco de dados, é só iniciá-lo no “Menu Iniciar”, opção “Start Database”.

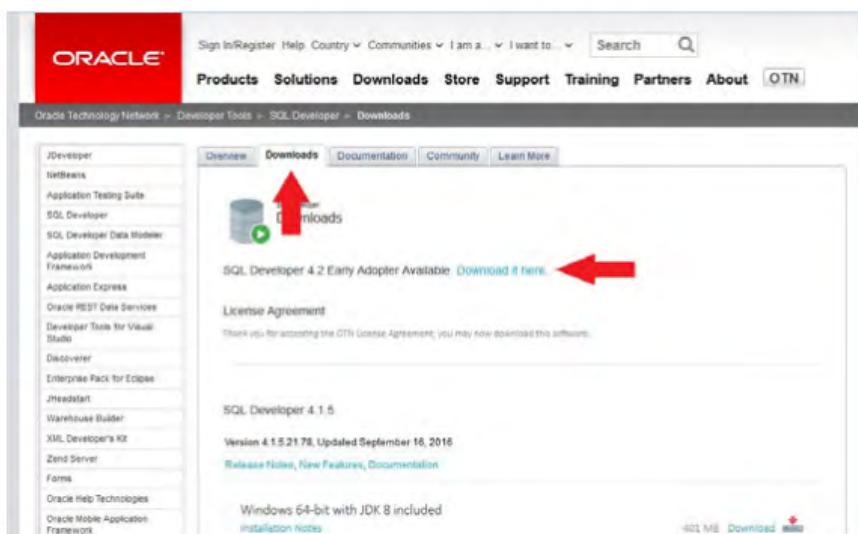


Figura 24: Opção para download do Oracle SQL Developer / Fonte: o autor.

Esse é um *client* gráfico, muito utilizado e gratuito.

- Baixando o SQL*Plus.

O SQL*Plus é muito utilizado pelos administradores de bancos de dados e desenvolvedores na interação com as bases de dados. Com ele, é possível executar todas as instruções SQL e programas PL/SQL, formatar as consultas e administrar a base de dados.

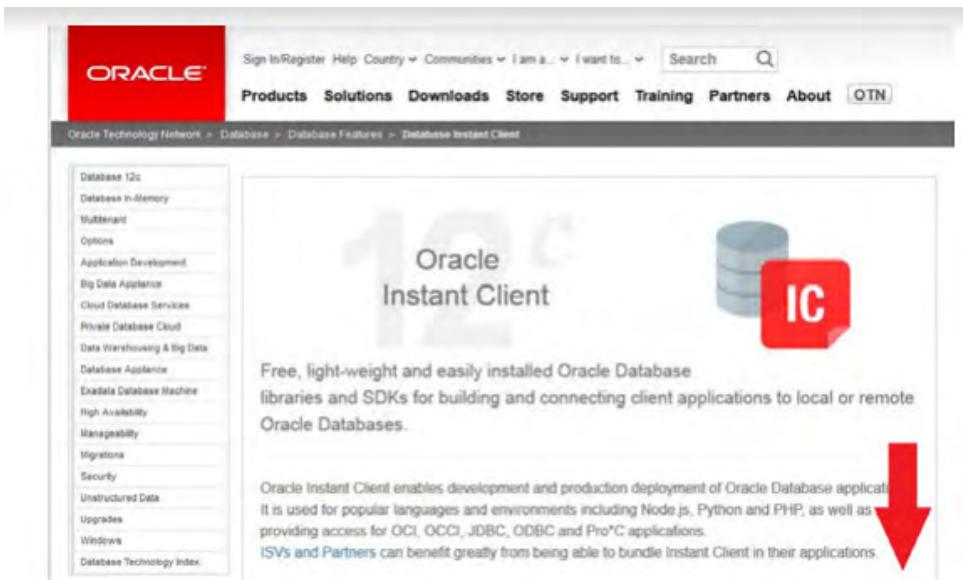


Figura 25: Parte superior da página de download / Fonte: o autor.

Instant Client Downloads

Please note that Instant Client is provided under a separate OTN Development and Distribution License for Instant Client that allows most licensees to download, redistribute, and deploy in production environments, without charge. Please consult the license and your legal department for clarification, if necessary.

- [Instant Client for Microsoft Windows \(x64\)](#)
- [Instant Client for Microsoft Windows \(32-bit\)](#)
- [Instant Client for Mac OS X \(Intel x86\) \(32-bit and 64-bit\)](#)
- [Instant Client for Mac OS X \(PPC\)](#)
- [Instant Client for Linux x86-64](#)
- [Instant Client for Linux x86](#)
- [Instant Client for Linux AMD64 \(32-bit and 64-bit\)](#)
- [Instant Client for Linux on Power Big Endian \(32-bit\)](#)
- [Instant Client for Linux on Power Big Endian \(64-bit\)](#)
- [Instant Client for Linux on Power Little Endian \(64-bit\)](#)
- [Instant Client for z/Linux \(31-bit and 64-bit\)](#)
- [Instant Client for Solaris Operating System \(SPARC\) \(64-bit\)](#)
- [Instant Client for Solaris Operating System \(SPARC\) \(32-bit\)](#)
- [Instant Client for Solaris x86](#)
- [Instant Client for Solaris x86-64](#)

Figura 26: Opções para download / Fonte: o autor.



CARACTERÍSTICAS DE SISTEMAS DE BANCOS DE DADOS

Se utilizar um sistema de banco de dados parece uma solução natural, qual seria a solução alternativa? **Pense em algumas aplicações que você utiliza e que não fazem uso de SGBDs.** Processadores de texto, planilhas, ferramentas de desenho etc. são alguns exemplos dessas aplicações. O que todas têm em comum? A necessidade de se armazenar a informação manipulada por meio de arquivos.

Em qualquer aplicação que necessite do armazenamento de dados, faz-se necessário dispor de algum mecanismo que permita que esses sejam gravados de modo persistente. A abordagem de arquivos tem suas vantagens, por exemplo, a portabilidade dos dados. Você pode carregá-los eletrônica ou fisicamente para locais diferentes de modo bastante simples, mas dentre as desvantagens dessa abordagem, há todo o trabalho necessário para se criar um formato e processar a sua gravação e recuperação – e, acredite, não é pouco trabalho!

Um SGBD, por outro lado, já dispõe de uma série de funcionalidades prontas para serem utilizadas pelo desenvolvedor da aplicação. Desse modo, uma série de preocupações passa a ser delegada a um software de terceiros (o SGBD). A seguir, apresentaremos uma série de características que diferenciam a abordagem de sistemas de banco de dados da manipulação manual das informações (como em arquivos, por exemplo).

Natureza autodescritiva: uma característica fundamental que distingue os sistemas de bancos de dados de outras abordagens é o fato de que, nos SGBDs, o banco de dados e as metainformações sobre o banco de dados são armazenados conjuntamente. Essas metainformações armazenadas contêm informações como o tipo, o tamanho e as restrições do banco de dados. Em termos técnicos, as metainformações são chamadas de esquema (ou *schema*, em seu termo original em inglês).

Isolamento entre programa e dados: numa aplicação que utiliza arquivos para o armazenamento de dados, quaisquer alterações na estrutura do arquivo também implicarão alterações no programa. Nesse caso, dizemos que o programa é altamente acoplado à sua estrutura de armazenamento de dados. Em contraste, SBGDs permitem que o programa somente informe quais dados são armazenados, sem se importar em como esses dados serão manipulados internamente. Essa característica aumenta bastante o nível de manutenibilidade do sistema, quando bem aplicada.

Múltiplas visões dos dados: essas não são uma característica fundamental, mas muitos SGBDs fornecem a possibilidade de que diferentes usuários com diferentes permissões possam acessar diferentes “visões” dos dados. Essas visões (*views*) correspondem a estruturas virtuais criadas a partir dos dados armazenados e podem conter, além dos próprios dados, informações derivadas (calculadas) a partir desses dados.



INDICAÇÃO DE LIVRO

Sistema de banco de dados

Autores: Abraham Silberschatz, Henry F. Korth e S. Sudarshan

Editora: Elsevier – Campus

Sinopse: este é um livro-texto clássico sobre treinamento em banco de dados. A obra apresenta os conceitos fundamentais do gerenciamento de banco de dados de uma maneira intuitiva e motivadora para os alunos, inclusive com um novo exemplo de banco de dados de uma universidade, que ilustra escolhas mais complexas de projeto. O texto enfatiza questões práticas, aplicações e implementação, junto com uma cobertura dos principais conceitos teóricos. Nesta sexta edição, todos os capítulos foram revisados para refletir os avanços mais recentes na tecnologia. Os autores fizeram ainda uma cobertura revisada e expandida dos seguintes temas: variantes da SQL em sistemas reais; projeto relacional; aplicações web e segurança; indexação e otimização de consulta; gerenciamento de transação; bancos de dados paralelos e distribuídos.



A criação de diferentes usuários com diferentes permissões e visões específicas é uma abordagem muito utilizada em sistemas cliente/servidor ou na integração de aplicações mediante banco de dados. O auge do uso dessas abordagens deu-se no final da década de 1990, embora ainda hoje seja possível testemunhar aplicações sendo executadas sob esse modelo. Recomenda-se fortemente que, no desenvolvimento de novas aplicações, a abordagem de múltiplas visões e de integração mediante banco de dados seja substituída por uma abordagem orientada a serviços como SOA (*Service Oriented Architecture*) ou como REST (*REpresentational State Transfer*).

Visões não são uma má prática. Trata-se de um recurso bastante útil, mas não imprescindível. Como toda ferramenta, quando bem utilizada e de modo adequada, é um recurso valioso.

Acesso concorrente de múltiplos usuários: um SGBD multiusuário, como o próprio nome já define, deve permitir o acesso de múltiplos usuários. Além disso, o acesso deve ser concorrente, permitindo que todos os usuários conectados executem operações “ao mesmo tempo”.

Vale a pena refletir sobre dois termos muitas vezes utilizados de forma errônea na área de tecnologia da informação: “paralelo” e “concorrente”. Paralelismo puro é algo raro em computação, embora seja perfeitamente possível. Ao lidarmos com sistemas de banco de dados, utilizamos o termo “concorrente”, pois vários usuários têm a impressão de que estão executando instruções ao mesmo tempo – quando, na verdade, por se tratar de informações acessadas em disco ou com um único barramento de acesso, torna-se necessário algum mecanismo de contenção que serialize (coloque em fila) cada uma dessas instruções. Como idealmente a execução dessas instruções é bastante curta, tem-se a impressão do pseudoparalelismo.

Importante lembrar que um conceito fundamental para que o acesso desses múltiplos usuários mantenha o banco de dados em um estado consistente é o mecanismo de transações.

TRANSAÇÕES

O conceito de transação é fundamental em muitas áreas da computação e particularmente fundamental em sistemas de banco de dados. Consideraremos como transação uma determinada “unidade de trabalho”, que é realizada em qualquer

sistema computacional de um modo coerente e independente de outras transações. Essas transações devem permitir que o sistema esteja em um estado coerente antes e depois de sua execução, independentemente de falhas ou outros problemas que possam ocorrer. Devem permitir também que vários clientes diferentes acessem concorrentemente o sistema sem que isso possa corromper ou levar a estados que não sejam considerados coerentes.

Uma definição clássica do conceito de transações envolve o acrônimo ACID, oriundo das propriedades de atomicidade, consistência, isolamento e durabilidade.

Atomicidade: a propriedade atomicidade de banco de dados advém do conceito de átomo da física – o qual até recentemente supunha-se indivisível. Essa indivisibilidade pressupõe que as operações realizadas numa transação sejam todas realizadas por completo ou que nenhuma seja realizada. Popularmente, seria o conceito do “tudo ou nada”. Isso permite que durante a nossa interação com um banco de dados, possamos agrupar vários comandos relacionados com a garantia de que todos sejam executados, de modo que as informações armazenadas permaneçam num estado consistente após a execução da transação.

Consistência: a propriedade de consistência assegura que a execução de qualquer transação trará o banco de dados de um estado consistente para outro estado também consistente. No caso, a “consistência” implica que todos os dados de um banco de dados devem ser válidos de acordo com um conjunto de regras que podem incluir restrições de tipo, valor, referências entre informações etc.

Isolamento: a propriedade de isolamento determina que o resultado da execução concorrente de um conjunto de transações terá o mesmo resultado de sua execução em série (uma após a outra). O isolamento transacional é o que garante e permite o acesso concorrente de múltiplos usuários ao mesmo SGBD.

Durabilidade: a propriedade de durabilidade garante que uma vez que uma transação tenha sido finalizada com sucesso, os dados terão a garantia de terem sido armazenados corretamente – independentemente da eventualidade de falhas, falta de energia, erros de aplicação etc.

É justamente a propriedade de durabilidade que faz com que os bancos de dados sejam posicionados como “ferramentas sagradas” em muitas empresas. Novamente, não há menosprezo algum em dizer que o mais importante é o código que atende aos processos de negócios. Durabilidade é essencial: imagine qualquer empresa perdendo todos os seus dados. A continuidade do próprio negócio está em risco, mas mais importante do que os dados em si é o uso que se faz deles.

VANTAGENS DE SE UTILIZAR UM SGBD

A seguir, enumeraremos as vantagens de um modo mais detalhado, de forma a justificar seu uso em uma diversidade de situações.

- 1. Diminuir a redundância e fornecer consistência:** imagine uma situação bastante comum em que você resolve elaborar um documento e necessita da colaboração de várias pessoas para fazê-lo. Você, então, cria o esboço desse documento e o envia por e-mail a todos os interessados. Cada pessoa realiza as suas modificações em suas próprias cópias dos documentos e, depois, repassa novamente por e-mail. Quem tem a última versão do documento? Quais são os dados corretos? Essas são perguntas difíceis de serem respondidas nessa abordagem e provavelmente exigirá muito trabalho manual para se chegar à versão final do documento. Um SGBD centraliza todas essas informações, fazendo com que todos os usuários acessem os mesmos dados. Desse modo, diminui-se a redundância: há somente uma cópia dos dados a serem manipulados. Isso permite também que o banco de dados sempre permaneça em um estado consistente, pois todos os usuários terão sempre a “última versão” dos dados. Não há a possibilidade de alguém permanecer com um “pedaço” dos dados antigos e outro “pedaço” com a informação atual.
- 2. Controle de acesso:** muitas informações armazenadas em sistemas são confidenciais. Ao mesmo tempo, é necessário que essas informações sejam compartilhadas com as pessoas para que sejam trabalhadas. Ao utilizar arquivos, é necessário que uma cópia seja enviada aos interessados. Por múltiplos motivos, essas cópias podem acabar sendo enviadas por e-mail a pessoas cujo acesso é indevido ou deixadas em um dispositivo de armazenamento removível esquecido em alguma mesa de reunião. No mínimo, um SGBD oferece uma combinação de login/senha para acesso a um determinado banco de dados. Outras restrições relativas a qual usuário pode acessar quais dados também costumam ser

implementadas pela maioria dos SBGDs. Como o acesso é centralizado, também tem-se uma única cópia para proteção.

3. **Consultas eficientes:** como são aplicações de software de propósito específico, os SGBDs são especialmente projetados para armazenar eficientemente os dados a eles delegados e para permitir formas de consulta eficientes aos mesmos dados. Cada SGBD possui sua estratégia interna para transformar essas informações em bytes gravados no dispositivo de armazenamento, mas, de um modo geral, não há uma grande diferença de desempenho entre diferentes produtos em uma quantidade razoável de aplicações. Em casos de usos típicos, é muito mais importante a eficiência em consultas do que a eficiência em armazenamento de informações. Assim, os SBGDs utilizam dispositivos como índices (que são estruturas criadas para otimizar consultas baseadas em certos critérios) e cachês (*caches*) para armazenar em memória os dados mais frequentemente acessados. Esses dispositivos permitem que as consultas possam ser executadas de modo mais rápido e, em muitos SGBDs, adequar esses dispositivos de modo otimizado chega a quase ser uma arte, tamanha a quantidade de opções disponíveis.
4. **Backup e restore:** para garantir a continuidade dos negócios, é essencial executar periodicamente o backup das informações armazenadas no servidor. Em vez de cópias físicas dos arquivos do SGBD, é comum os próprios SGBDs fornecerem ferramentas que permitem a exportação dos dados para um formato intermediário (texto ou binário) para backup. Essas mesmas ferramentas suportam a restauração desses dados em caso de necessidade. As rotinas de backup/*restore* também são uma ferramenta bastante útil na migração ou cópia de servidores em que o mesmo SGBD esteja instalado. Situações de migração costumam ocorrer em caso de falhas ou upgrade de equipamento. Cópias costumam ser utilizadas para permitir o teste de aplicações em desenvolvimento.

NOVOS DESAFIOS

Caro(a) aluno(a), conhecemos o conceito de banco de dados e aprendemos que a ideia de banco de informações veio antes da era dos computadores. Desde os primórdios, já havia um modo de armazenamento de informações, por meio de livros e registros em papel. Destacamos algumas vantagens de utilizar o banco de dados. Identificamos que, ainda que não percebamos, manipulamos informações em um banco de dados por meio de um sistema.

Com o crescimento da tecnologia e das diversas linguagens de programação, houve a necessidade de padronizar o acesso ao banco de dados. Eis que surgem os SGBDs, com o intuito de resolver problemas de acesso e confiabilidade de informações, atribuindo também o padrão de linguagem SQL.

Também foi apresentado a você, aluno(a), alguns dos mais conceituados tipos de SGBDs, sendo o MySQL, Oracle, SQL Server, PostgreSQL, temos disponíveis vários outros não mencionados aqui. Ao conhecermos o SGBD, vimos que sua base estrutural segue o modelo de dados relacional, mas para compreendermos o porquê dessa escolha, estudamos os outros tipos de modelos de dados, hierárquico, rede, relacional e orientado a objetos.

O modelo de dados relacional é utilizado pela grande maioria dos SGBDs, por isso que estudamos esse modelo com mais detalhes, que tem sua base na matemática, com a representação das informações em tabelas e colunas.

Podemos considerar que fizemos uma breve comparação entre o SGBDs MySQL e Oracle, considerados os bancos mais utilizados no mercado, além de efetuarmos passo a passo a instalação deles. Esses programas serão utilizados no decorrer de nossa disciplina. Esperamos que tenha aproveitado o conteúdo apresentado para enriquecer seus conhecimentos, até a próxima!

AGORA É COM VOCÊ

1. Conforme vimos neste tema, a ideia de um Banco de Dados veio antes da era dos computadores, o conceito de banco de informações não se limita apenas em arquivos digitais. Assinale Verdadeiro (V) ou Falso (F) no que podemos considerar como Banco de Dados:
 Planilhas Eletrônicas.
 Papel de Carta.
 Livro de Registros.
 Arquivo Morto.
 Arquivo.
2. O Modelo de Dados utilizados pelos SGBDs é o relacional, tendo como conceito a relação matemática, tabelas e valores. Sua teoria está vinculada a teoria de conjuntos e na lógica de predicados de primeira ordem. A representação do modelo relacional conta com alguns elementos. Esses elementos são:
I - Identificação.
II - Tuplas.
III - Atributos.
IV - Correlações.
V - Relação.
a) Apenas I, II e V.
b) Apenas II e III.
c) Apenas II, III e IV.
d) Apenas II, III e V.
e) Todas as afirmativas.
3. Ao enumerar as vantagens de se utilizar um SBGD, fizemos uma comparação com a utilização de arquivos para armazenamento dos dados. Tendenciosamente, o SGBD apareceu como o vencedor nas comparações. Quais seriam as situações em que os arquivos seriam uma solução mais adequada? Você consegue exemplificar alguma outra situação em que uma terceira alternativa seria mais viável?



MODELAGEM DE DADOS

ESP. CARLOS DANILO LUZ

MINHAS METAS

- Conhecer os conceitos básicos sobre os tipos de modelagem.
- Entender o que é Modelo Entidade Relacionamento (MER).
- Apresentar os elementos que compõem um Diagrama Entidade Relacionamento (DER).
- Conceituar a normalização de dados.

INICIE SUA JORNADA

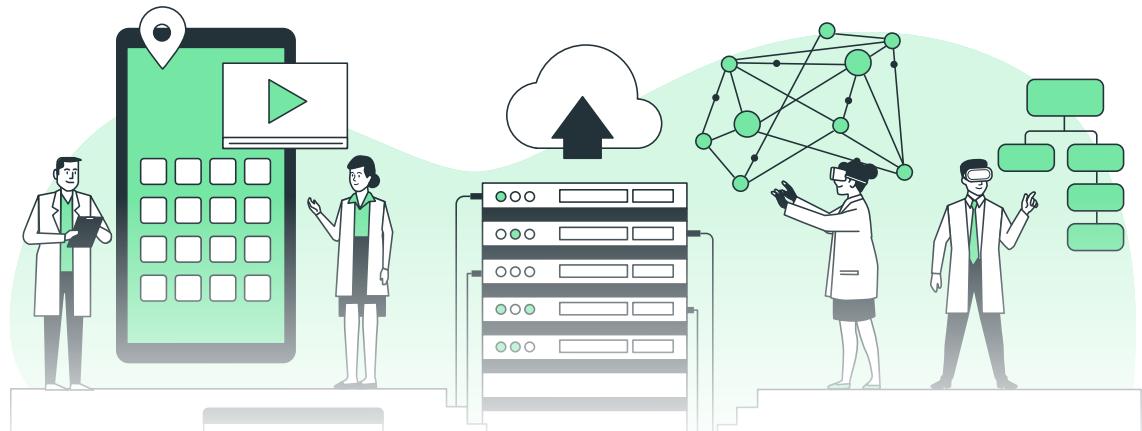
Vamos conhecer como estruturar o nosso banco de dados? Aqui apresentaremos as primeiras etapas na estruturação de um banco de dados. Antes de inserir todas as informações em um SGBD, devemos colocar as ideias “no papel”, ela se inicia com o processo de modelagem de dados. Utilizando os processos da engenharia de requisitos, vamos colher as primeiras informações de nosso sistema.

Compreenderemos os três tipos de modelagem de dados, sendo o conceitual, lógico e físico. Com base no modelo conceitual, vamos utilizar o Modelo Entidade Relacionamento (MER) e o Diagrama Entidade Relacionamento (DER). A utilização de um DER na elaboração da estrutura do banco de dados tem como função prevenir possíveis falhas ou um banco mal estruturado. Ele nos dá uma visão total do banco, sendo que a representação do DER é efetuada por meio de elementos gráficos e textuais.

Para efetuar a composição do DER, utiliza-se entidades, as quais representam as tabelas de um banco de dados, e atributos que são os dados da tabela, que podemos considerar como as colunas da tabela. As entidades de um banco de dados têm associações. Para representá-las, utilizamos o relacionamento, item que caracteriza essa associação.

Devemos pensar também na regra de negócio proposta no início do projeto, em que teremos alguns registros e obrigações em alguns relacionamentos. Para isso, vamos utilizar a cardinalidade de relacionamentos, sendo indicado o número máximo e mínimo de associações permitidos para os relacionamentos entre as entidades.

Em alguns casos, vamos ter a nossa disposição informações exportadas de um Sistema Gerenciador de Banco de Dados (SGBD) ou disponibilizadas pelo cliente. Para que possamos entender como as informações devem estar armazenadas, utilizaremos a normalização para esse procedimento, no qual estudaremos a 1FN, 2FN e 3FN.



MODELAGEM DE DADOS

Podemos considerar a modelagem de dados como uma das mais importantes etapas no desenvolvimento de um banco de dados. Ela tem como base os processos da engenharia de requisitos. Um dos métodos mais utilizados nessa etapa é a entrevista, que se baseia em uma conversa entre o cliente e o analista. Nela, o cliente expõe suas ideias sobre o que deseja em seu sistema. Para Puga, França e Goya (2013, p. 77):



a modelagem de dados é um método de análise que, a partir de fatos relevantes a um contexto de negócio, determina a perspectiva dos dados, permitindo organizá-los em estruturas bem definidas e estabelecer regras de dependência entre eles, além de produzir um modelo expresso por uma representação descritiva e gráfica.

Alguns dos objetivos dessa etapa são:

- Entender melhor a regra de negócio do cliente.
- Colher as informações que compõem o sistema.
- Elaborar o banco de dados.
- Contribuir para a organização das informações.

Podemos classificar as etapas da modelagem de dados em alto nível (modelo conceitual), representacional (modelo lógico) e baixo nível (modelo físico).

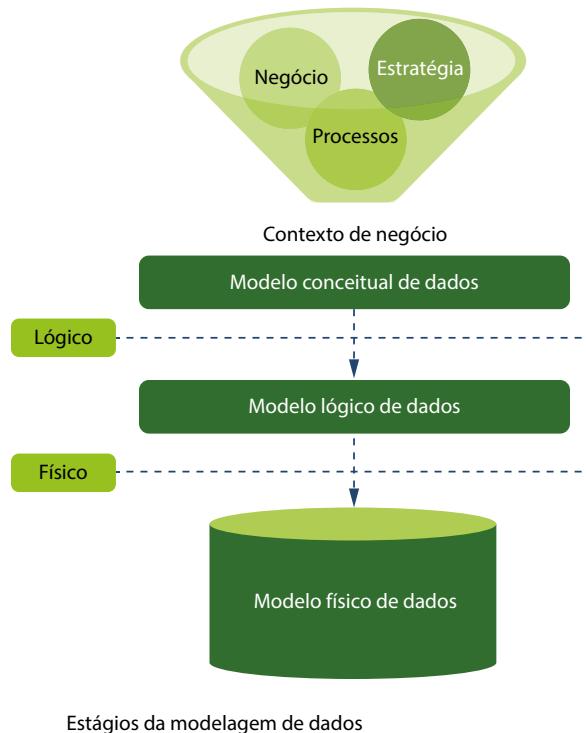


Figura 1: Estágios da modelagem de dados / Fonte: Adaptada de Puga, França e Goya (2005).

ZOOM NO CONHECIMENTO

Os processos atribuídos à engenharia de requisitos não se limitam apenas ao levantamento de informações, mas também refletem a questões sobre qualidade de software, desenvolvimento, manutenção, custos, prazos, qualidade do produto, dentre outros. Para mais informações sobre engenharia de requisitos, veja a indicação de livros no material complementar, ao final desta unidade.

Fonte: os autores.

Modelo Conceitual

O modelo conceitual é a representação do banco de dados sem a identificação do SGBD ou da forma de implementação. Ele organiza as informações da forma que vão estar presentes no banco de dados, mas não a forma de armazenamento com base em um SGBD.

Para Heuser (2009), o modelo conceitual é definido como um modelo de dados abstratos que descreve a estrutura de um banco de dados de forma independente de um SGBD particular. Assim, ele pode ser representado pelo conceito de entidade-relacionamento (E-R), visualmente por um DER, diagrama esse que veremos mais à frente.

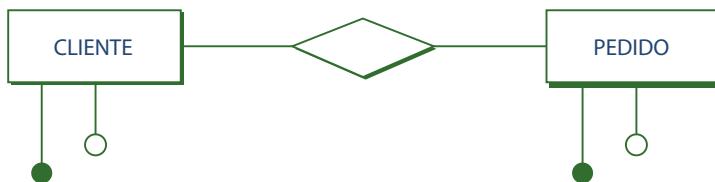


Figura 2: Exemplo de modelo conceitual / Fonte: adaptado de Heuser (2009).

Modelo Lógico

O modelo lógico se utiliza dos dados informados no modelo anterior e efetua a descrição dos itens, de forma que seja interpretada, posteriormente, por um SGBD ou pelo seu usuário. Nesse ponto, o SGBD se torna relevante, pois temos vários tipos de modelos de dados (hierárquico, rede, relacional e orientado a objetos). Como o modelo relacional é o mais utilizado nos SGBDs no mercado atual, nossos exemplos seguirão com base nesse modelo.

CLIENTE			
CodCliente	Nome	telefone	Tipo Pessoa
1	Carlos Danilo	39025-7514	Física
2	Roberta Rampani	38598-7596	Física
3	ARJ Alimentos	58458-8652	Jurídica
4	Bela Arte Aviamentos	79851-9671	Jurídica

PEDIDO	CodCliente	Valor Pedido	Forma Pagamento
14521	1	150.00	Cartão

14598	3	1521.28	Boleto
15428	4	14548.50	Boleto
15428	1	258.59	Cartão

Figura 3: Exemplo de modelo lógico com base no modelo relacional
Fonte: adaptada de Heuser (2009).

Modelo Físico

Este modelo é utilizado para efetuar a sintonia de banco de dados com o intuito de otimizar o desempenho. Puga, França e Goya (2013, p. 80) afirmam que “[...] o modelo físico de dados representa a estrutura para armazenamento físico dos dados, expressando a forma como as informações serão armazenadas fisicamente, em termos computacionais”.

Conforme mencionado no modelo lógico, a implementação do SGBD pode influenciar as estruturas físicas que são criadas nessa etapa de acordo com o SGBD escolhido. Alterações efetuadas no modelo físico não interferem nas aplicações que estão utilizando o banco de dados. Essas alterações podem acontecer após a implementação e funcionamento do SGBD.



PENSANDO JUNTOS

É possível estruturarmos um banco de dados sem efetuar os passos de modelagem de dados?

MODELO ENTIDADE RELACIONAMENTO (MER)

Ao pensarmos em criar um banco de dados ou planejar a estrutura de um sistema, devemos pensar primeiro em como as informações serão distribuídas, armazenadas e relacionadas de forma mais simplificada e segura. Com base nessa ideia, podemos utilizar métodos para efetuar a representação visual do banco de dados.

Em 1976, Peter Chen criou o Modelo Entidade Relacionamento (MER) e o Diagrama Entidade Relacionamento (DER), considerados modelos padrões da

modelagem conceitual. Atualmente, o MER ou DER são utilizados como um dos primeiros passos na estruturação de um banco de dados.

Cardoso e Cardoso (2012) afirmam que “[...] projetos que excluem esse processo apresentam muitos erros e falhas, que, ao longo de seu desenvolvimento e aplicação, causam inúmeros problemas”, **um projeto mal elaborado ou com falhas pode prejudicar o bom funcionamento do sistema.**

Diagrama Entidade Relacionamento (DER)

O Diagrama Entidade Relacionamento (DER) é considerado um padrão da modelagem conceitual. A representação de um DER é efetuada por meio de esquemas gráficos e textuais, conforme Figura 4.

Conceito	Símbolo
Entidade	Retângulo
Relacionamento	Diamond
Atributo	—○ ou Oval

Figura 4: Símbolos usados no desenvolvimento do DER / Fonte: adaptada de Cardoso e Cardoso (2012).

Entidades

Uma entidade representa uma tabela do Banco de Dados no DER. Podemos considerar a tabela como um conjunto ou objeto de informações. Vamos considerar uma entidade com o nome de alunos. Na tabela alunos, será armazenado um conjunto de informação sobre a entidade. Para Heuser (2009), entidade é um conjunto de objetos da realidade modelada sobre os quais deseja-se manter informações no banco de dados. Dessa forma, o banco de dados é separado por entidades (tabelas).

Conforme informado anteriormente, uma entidade é representada em um DER por meio de um retângulo com o nome da tabela ao centro dele. Alguns exemplos de entidades são mostrados na Figura 5.



Figura 5: Exemplos de representação de entidades / Fonte: os autores.

Podemos classificar as entidades como concretas e abstratas:

- **Concretas** – são objetos que tenham um único sentido e de fácil distinção. Exemplo: carro. Quando falamos de um carro, estamos claramente falando de um veículo.
- **Abstratas** – não temos uma informação clara do objeto. Exemplo: aluguel, que pode ser de um carro, casa, livro, dentre outras situações. Percebemos que não temos uma definição do que se trata a entidade aluguel.

A representação de ambas as entidades se faz da mesma forma, por meio de um retângulo com o nome da tabela no centro. Alguns exemplos de entidades concretas e abstratas são encontrados na Figura 6:

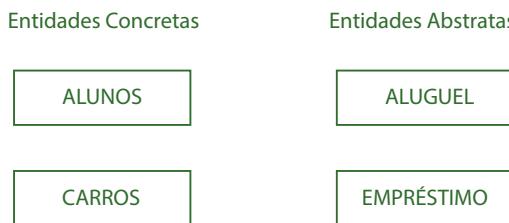


Figura 6: Exemplos de entidades concretas e abstratas / Fonte: os autores.

Atributos

Os atributos são informações ou descrições das entidades. Podemos considerar também que são as colunas de nossa tabela, pois cada atributo deverá armazenar uma informação específica. A representação de um atributo no DER pode ser de forma textual ou gráfica, conforme Figura 7.



Figura 7: Exemplo de representação de atributos / Fonte: os autores.

Vamos utilizar a entidade alunos como exemplo. Os atributos dela são: matrícula, nome, CPF etc. Perceba que os atributos são os dados que esperamos no cadastro de um aluno. Veremos, na Figura 8, como é a composição do DER com entidades e atributos.

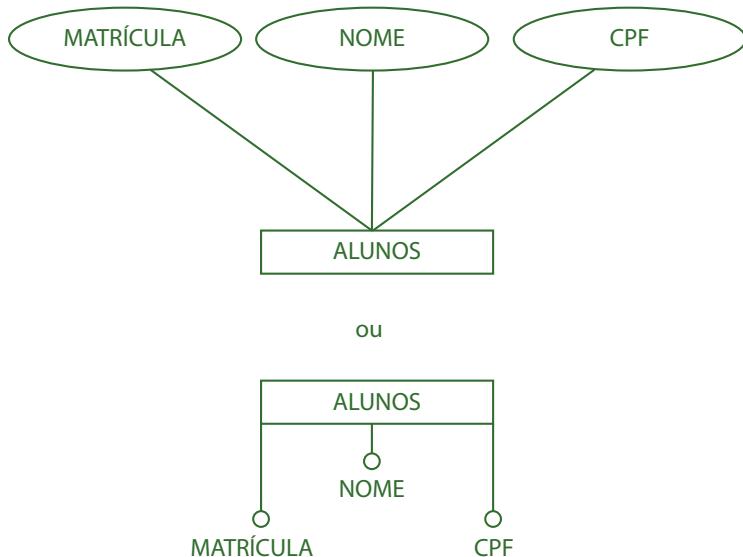


Figura 8: Exemplo de um DER com entidades e atributos / Fonte: os autores.

Tipos de Atributos

Os atributos podem ser classificados como simples, multivalorados e compostos. Neste livro, somente vamos abordar o uso do simples e multivalorado, por serem os tipos mais usuais em um DER.

- **Simples:** ele suporta apenas um único registro. Ao utilizarmos o exemplo da entidade alunos, cada registro terá o nome de um aluno. A informação contida nesse atributo pode ser repetida na existência de alunos com o mesmo nome. A representação no DER é igual à já apresentada, uma elipse com a informação no centro.



Figura 9: Representação de atributo simples / Fonte: os autores.

- **Multivalorado:** ele suporta múltiplos registros. Novamente, vamos utilizar o exemplo da entidade alunos. No cadastro de um aluno, temos que registrar os telefones para contato. Se utilizamos um atributo simples, vamos ter telefone_residencial, telefone_comercial, telefone_recado. Podemos simplificar somente com um atributo multivalorado como telefones, o qual pode conter o registro de todos os números. A sua representação continua sendo uma elipse, agora com duas bordas e a informação no centro.



Figura 10: Representação de atributo multivalorado / Fonte: os autores.

Chaves

Dentre os atributos, devemos informar qual será o de identificação, sendo único em toda a tabela e nunca nulo (preenchimento vazio). Trata-se de atributo que podemos chamar de atributo-chave. Com base nos exemplos apresentados anteriormente, a informação da matrícula será nosso atributo-chave. Sua representação pode ser com o texto sublinhado ou com a borda destacada em negrito.



Figura 11: Representação de atributo-chave / Fonte: os autores.

RELACIONAMENTOS

Ao analisarmos o modelo relacional, as entidades não ficam isoladas. Em um banco de dados, as tabelas são interligadas de forma que ocorra associação entre objetos. No MER, utiliza-se o relacionamento para representar essa associação. Heuser (2009) define relacionamento como um conjunto de associações entre ocorrências de entidades.

No MER, as entidades não podem ser interligadas diretamente. Elas utilizam um intermediário, o relacionamento. Utiliza-se no DER um losango sendo o relacionamento e linhas retas para representar a ligação entre elas. Ao efetuar

o DER, o relacionamento deve ser identificado por um verbo, como vemos no exemplo da Figura 12:



Figura 12: Representação do relacionamento em um DER

Fonte: adaptada de Cardoso e Cardoso (2012).

Tipos de Relacionamentos

Podemos encontrar casos variados de relacionamentos. A classificação deles se dá pela quantidade de entidades associadas. São eles:

- Autorrelacionamento – refere-se a um relacionamento composto de apenas uma entidade. Podemos citar como exemplo um casamento no qual marido e esposa (cada um deles) corresponde a uma ocorrência.

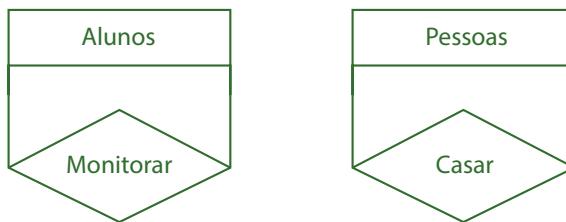


Figura 13: Representação de um autorrelacionamento / Fonte: adaptada de Cardoso e Cardoso (2012).

- Binário ou grau dois – neste tipo de relacionamento, temos duas entidades sendo interligadas. Podemos usar como exemplo nossa disciplina de Banco de Dados, que será ministrada por um professor.



Figura 14: Representação do relacionamento binário / Fonte: adaptada de Cardoso e Cardoso (2012).

- Ternário – relacionamento em que temos três entidades sendo interligadas. Com base no relacionamento binário, vamos inserir no DER a entidade curso.



Figura 15: Representação do relacionamento ternário / Fonte: adaptada de Cardoso e Cardoso (2012).

Além dos tipos de relacionamentos apresentados anteriormente, podemos encontrar algumas situações nas quais temos mais de um relacionamento atribuído para as mesmas entidades. Vamos dizer que temos um projeto na instituição, o qual está sendo gerenciado pelo professor Roberto. Junto a esse projeto, temos algumas disciplinas que o professor Roberto também vai ministrar. Com base nessas informações, vejamos o DER na Figura 16.

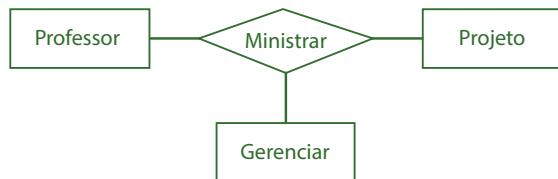


Figura 16: Exemplo de mais de um relacionamento sendo associado a entidades
Fonte: adaptada de Cardoso e Cardoso (2012).

Cardinalidade de Relacionamentos

Prezado(a) aluno(a), agora que você já conhece os elementos básicos para montar um DER, é necessário inserir algumas restrições e informações para que ele não fique generalizado, mas o mais próximo da regra de negócio apresentada. Por meio do MER, podemos utilizar a cardinalidade de relacionamentos.



A cardinalidade permite expressar o número de ocorrências com que uma entidade pode tomar parte em um relacionamento. Permite também expressar as possibilidades e restrições de associações entre uma entidade e outra. (CARDOSO; CARDOSO, 2012, p. 35)

A cardinalidade rege as regras para a implementação das restrições em um DER. Ela pode ser máxima e mínima.

- **Cardinalidade máxima:** trata-se da quantidade máxima de ocorrências que pode haver entre as entidades, sendo representada por 1:1, 1:N, N:1 e N:N.
- **Cardinalidade mínima:** trata-se da quantidade mínima de ocorrências que pode haver entre as entidades, sendo representada por 0:1 ou 0:N.

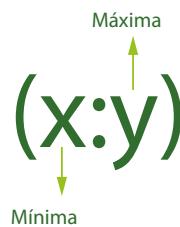


Figura 17: Representação da cardinalidade mínima e máxima / Fonte: os autores.

Cardinalidade 1:1

Acontece quando temos apenas uma ocorrência a ser registrada para cada entidade em ambos os lados, em que a cardinalidade atribuída é 1:1. Podemos utilizar como exemplo o relacionamento professor e disciplina sendo: um professor pode ministrar uma disciplina.

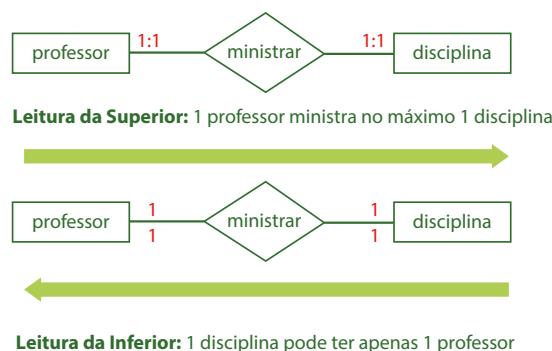


Figura 18: Representação da cardinalidade 1:1 / Fonte: adaptada de Cardoso e Cardoso (2012).

Cardinalidade 1:N

Acontece quando temos uma ocorrência mínima para muitos, então a cardinalidade atribuída é 1:N. Com base no exemplo anterior, vamos mudar um pouco o cenário: um professor pode ministrar várias disciplinas no curso.

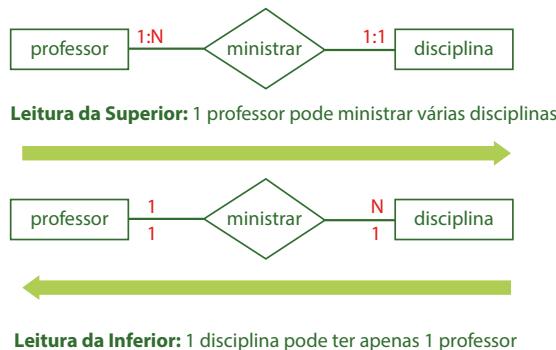


Figura 19: Representação da cardinalidade 1:N / Fonte: adaptada de Cardoso e Cardoso (2012).

Cardinalidade N:1

Podemos nos deparar com o contrário, quando temos múltiplas ocorrências para apenas 1 e a cardinalidade atribuída é N:1. Com base no exemplo anterior, vamos mudar novamente o cenário: uma disciplina pode ser ministrada por vários professores.

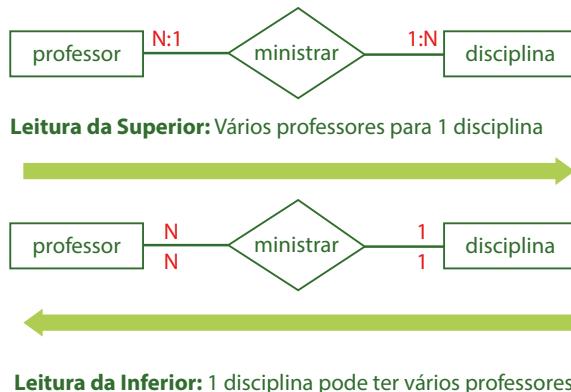


Figura 20 - Representação da cardinalidade N:1 / Fonte: adaptada de Cardoso e Cardoso (2012).

Cardinalidade N:N

Podemos também encontrar a situação de muitas ocorrências para muitas, a cardinalidade atribuída é N:N. Continuando com os exemplos anteriores: a disciplina pode ser ministrada por muitos professores e muitos professores podem ministrar várias disciplinas.

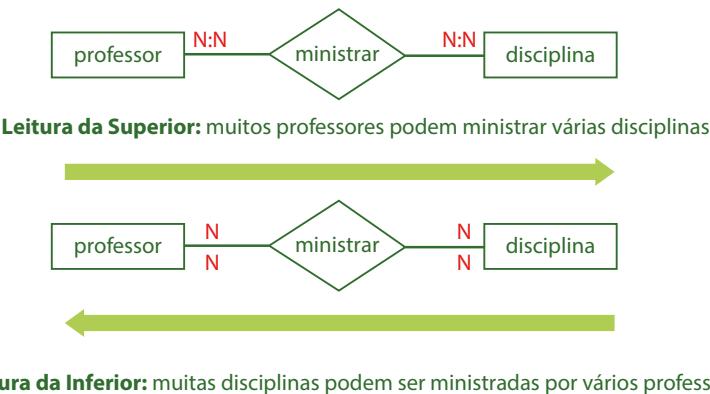


Figura 21: Representação da cardinalidade N:N / Fonte: adaptada de Cardoso e Cardoso (2012).

Cardinalidade 0:1

O número 0 sendo atribuído como cardinalidade mínima representa um item opcional para a existência da ocorrência. Podemos utilizar como exemplo uma empresa em que temos o setor de desenvolvimento. Contamos com uma mesa disponível no setor, mas para que ela esteja no setor, não é obrigatório ter um empregado vinculado a ela.

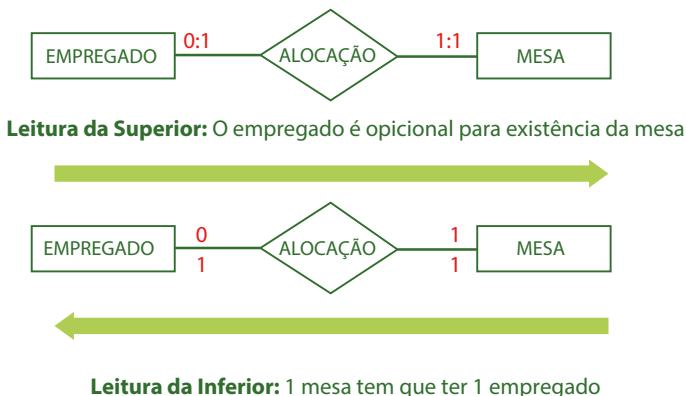


Figura 22: Representação da cardinalidade 0:1 / Fonte: adaptada de Heuser (2009).

Cardinalidade 0:N

Acontece quando temos muitas ocorrências sem termos algumas restrições mínimas. Continuando com o exemplo da empresa: contamos com muitas mesas disponíveis no setor, mas para que elas estejam no setor, não é obrigatório ter um empregado vinculado a elas.

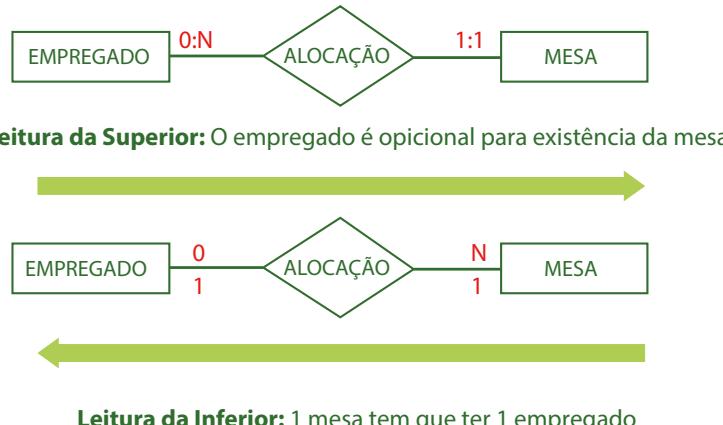


Figura 23: Representação da cardinalidade 0:N / Fonte: adaptada de Heuser (2009).

Cardinalidade Ternária

Conforme mencionado anteriormente, podemos nos deparar com a associação de três entidades, caracterizando um relacionamento ternário. A cardinalidade também deve ser atribuída nesses tipos, mas com algumas particularidades. Vamos ter como exemplo um centro de distribuição, em que ela pode acontecer da seguinte forma:

- Um distribuidor pode atender a muitas cidades com muitos produtos.
- Muitas cidades podem ter muitos produtos e apenas um distribuidor.
- Muitos produtos têm apenas um distribuidor. Esses produtos podem ser distribuídos para muitas cidades.

Podemos entender melhor observando o DER da Figura 24:

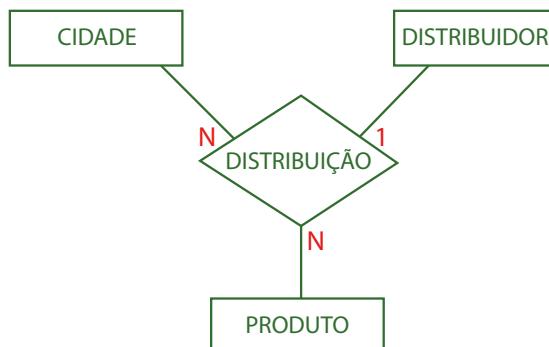


Figura 24: Representação da cardinalidade ternária / Fonte: adaptada de Heuser (2009).

NORMALIZAÇÃO

A proposta da normalização é obter um modelo relacional a partir de qualquer tipo de representação de dados. Por meio dela, podemos identificar as entidades e os atributos das informações do arquivo em questão. Para Heuser (2009), o objetivo da normalização é:

- Reagrupar informações de forma a eliminar redundâncias de dados que possam existir nos arquivos.
- Reagrupar informações de uma forma que permita a obtenção de um modelo ER.



A normalização utiliza-se de cinco etapas, chamadas de formas normais (1FN, -2FN, 3FN, 4FN, 5FN), mas aqui utilizaremos apenas as três iniciais, pois são as mais relevantes.

Vamos utilizar para exemplificar as etapas das formas normais o documento apresentado na forma de tabela não normalizada, conforme a Figura 25.

CodCurso	Tipo	Nome	Universidade					
			CodAluno	Nome	Módulo	Série	Data Início	Total Mod
ADSI01	Graduação	Análise e Des. Sistemas	2146	João	51	4	10/10/15	24
			3145	Silvio	52	4	07/08/15	24
			6128	José	54	9	05/02/16	18
			1214	Carlos	52	4	15/02/16	18
			8191	Mario	51	4	25/03/16	12
EADT01	Graduação	Engenharia de Software	8191	Mario	51	4	27/09/14	12
			4212	João	52	4	18/04/13	24
			6126	José	54	9	15/08/15	12

Figura 25: Tabela não normalizada / Fonte: adaptada de Heuser (2009).

Primeira Forma Normal (1FN)

A primeira etapa da primeira forma normal (1FN) é a transformação dos dados da tabela não normalizada na modelagem de dados lógica. Podemos considerar que uma tabela está na 1FN quando não temos tabelas aninhadas. Podemos utilizar duas formas de transformar uma tabela não normalizada para a 1FN:

- Elaborar uma única tabela, mas ela vai contar com redundância de dados. Nessa tabela, as informações vão aparecer repetidas em cada linha.
- Outra alternativa é criar outra tabela para cada tabela alinhada. Dessa forma, os dados são separados em duas tabelas para melhorar a organização.

Vamos utilizar a segunda opção para a melhor compreensão da normalização. Vejamos como a tabela não normalizada fica após a 1FN. Na Figura 26, temos que observar que CodCurso da primeira tabela alinha com uma chave, e que CodCurso e CodAluno são as chaves da segunda tabela.

CURSO		
CodCliente	Tipo	Nome
ADSI01	Graduação	Análise e Des. Sistemas
EADT01	Graduação	Engenharia de Software

CURSO UNIVERSIDADE						
CodCurso	CodAluno	Nome	Módulo	Série	Data Início	Módulo
ADSI01	2146	João	51	4	10/10/15	24
ADSI01	3145	Silvio	52	4	07/08/15	24
ADSI01	6126	José	54	9	05/02/16	18
ADSI01	1214	Carlos	52	4	15/02/16	18
ADSI01	8191	Mario	51	4	25/03/16	12
EADT01	8191	Mario	51	4	27/09/14	12
EADT01	4212	João	52	4	18/04/13	24
EADT01	6126	José	54	9	15/08/15	12

Figura 26: Tabela representada na primeira forma normal / Fonte: adaptada de Heuser (2009).

Segunda Forma Normal (2FN)

A etapa da segunda forma normal (2FN) tem como função a eliminação de redundância de dados. Para que seja possível efetuar a 2FN da tabela, deve-se já estar na 1FN. Vamos dividir novamente a segunda tabela de forma que a nossa chave é CodAluno, conforme a Figura 27.

CURSO		
CodCurso	Tipo	Nome
ADSI01	Graduação	Análise e Des. Sistemas
EADT01	Graduação	Engenharia de Software

CURSO UNIVERSIDADE			
CodCurso	CodAluno	Data Início	Total Mód
ADSI01	2146	10/10/15	24
ADSI01	3145	07/08/15	24

ADSI01	6126	05/02/16	18
ADSI01	1214	15/02/16	18
ADSI01	8191	25/03/16	12
EADT01	8191	27/09/14	12
EADT01	4212	18/04/13	24
EADT01	6126	15/08/15	12

ALUNOS			
CodAluno	Nome	Módulo	Série
2146	João	51	4
3145	Silvio	52	4
6126	José	54	9
1214	Carlos	52	4
8191	Mario	51	4
8191	Mario	51	4
4212	João	52	4
6126	José	54	9

Figura 27: Tabela representada na segunda forma normal / Fonte: adaptada de Heuser (2009).

Terceira Forma Normal (3FN)

Nesta etapa, já conseguimos extrair os dados para o modelo relacional. A cada forma normal que a tabela passa, os dados são separados com o intuito de um melhor entendimento e cada vez evitar a redundância de dados. A terceira forma normal (3FN) é uma continuação direta das tabelas da 2FN. Nesta etapa, a nova tabela não tem dependências de chaves externas de outras tabelas.

A partir da 3FN, já podemos desenvolver um DER caso for necessário. Acompanhe na Figura 28 as tabelas na 3FN.

CURSO		
CodCurso	Tipo	Nome
ADSI01	Graduação	Análise e Des. Sistemas
EADT01	Graduação	Engenharia de Software

CURSO UNIVERSIDADE				ALUNOS			
CodCurso	CodAluno	Data Início	Total Mód	CodAluno	Nome	Módulo	Série
ADSI01	2146	10/10/15	24	2146	João	51	4
ADSI01	3145	07/08/15	24	3145	Silvio	52	4
ADSI01	6126	05/02/16	18	6126	José	54	9
ADSI01	1214	15/02/16	18	1214	Carlos	52	4
ADSI01	8191	25/03/16	12	8191	Mario	51	4
EADT01	8191	27/09/14	12	8191	Mario	51	4
EADT01	4212	18/04/13	24	4212	João	52	4
EADT01	6126	15/08/15	12	6126	José	54	9

MÓDULO	SÉRIE
51	4
52	4
54	9

Figura 28: Tabela representada na terceira forma normal / Fonte: adaptada de Heuser (2009).

 INDICAÇÃO DE LIVRO

Engenharia de Requisitos

Autores: Carlos Eduardo Vazquez e Guilherme Siqueira Simões

Editora: Brasport

Sinopse: este livro apresenta a engenharia de requisitos de um ponto de vista prático com diversos exercícios e estudo de caso, sendo, principalmente, voltado à comunicação com o cliente.



NOVOS DESAFIOS

Prezado(a) aluno(a)! Aqui compreendemos que a estruturação de um banco de dados se inicia na modelagem de dados. Esta etapa é de suma importância na construção do banco de dados, a fim de evitarmos falhas no decorrer do projeto. Ainda, compreendemos que a modelagem pode ser classificada como conceitual, lógica e física.

Aprendemos os conceitos do MER e sobre o DER, elementos que são considerados padrões do modelo conceitual para a modelagem de dados. Ao conhecermos os itens que compõem um DER, sendo as entidades e os atributos, temos uma visão mais ampla sobre o nosso banco de dados, de forma simples, mas muito eficiente. Compreendemos também que as entidades não podem ficar soltas no DER, pois igualmente como ocorre no banco de dados, as tabelas podem ser interligadas. Por isso, foi lhe apresentado o relacionamento, forma essa de demonstrar a associação entre entidades.

Para que nosso DER represente de forma mais fiel a regra de negócio estabelecida no início da nossa estrutura, temos que aplicar a cardinalidade de relacionamentos que nos informa quais as obrigações e as restrições no relacionamento entre entidades, podendo ser elas de autorrelacionamento, binário ou ternário.

Por fim, vimos a normalização, item que é muito utilizado na engenharia reversa de dados quando temos algum documento e gostaríamos de gerar um DER. Os dados desse documento se encontram em uma tabela não normalizada, passando pelas 1FN, 2FN e 3FN, para que possamos extrair as informações em tabelas com base no modelo relacional.

Pois bem, caro(a) aluno(a), esperamos que tenha aproveitado o conhecimento aqui exposto. Até mais!

AGORA É COM VOCÊ

1. Para a estruturação de um Banco de Dados, utilizamos os conceitos de modelagem de dados, a fim de entender melhor a regra de negócio, colher as informações do sistema, organização das informações etc. Essa coleta de dados pode acontecer por meio de uma entrevista, conversa entre cliente e desenvolvedor, esse processo faz parte de qual conceito ?
 - a) Coleta de Dados.
 - b) Estruturação do Banco de Dados.
 - c) Engenharia de Requisitos.
 - d) Análise de Requisitos.
 - e) DER.
2. Com base no que estudamos neste tema, defina de forma simples o que são entidade e atributos no modelo relacional.
3. Analise as frases a seguir e identifique suas entidades:
 - a) Um pedido pode ter vários produtos, mas apenas um cliente.
 - b) Professor Antônio ministra disciplinas no curso de Matemática.
 - c) Em uma biblioteca, os livros estão separados por seções.
 - d) Na Clínica CRD, médicos prescrevem os tratamentos para pacientes da rede pública.

MEU ESPAÇO



MODELO RELACIONAL

ESP. VICTOR DE MARQUIS PEDROSO

MINHAS METAS

- Desenvolver entendimento sobre os conceitos básicos na criação de um banco de dados.
- Apresentar a melhor maneira de aplicação dos conceitos básicos.

INICIE SUA JORNADA

A partir desta unidade, abordaremos o modelo de dados relacional, que é o modelo utilizado nos bancos de dados relacionais. O advento do modelo relacional é atribuído a Ted Codd, da IBM, em 1970. Imediatamente, popularizou-se devido à sua simplicidade e sólidos fundamentos matemáticos: é baseado na teoria geral dos conjuntos e em lógica matemática.

Os primeiros SGDBs relacionais apareceram na década de 1980 como uma novidade boa no meio da computação, tanto que esses SGDBs acabaram sucedendo os bancos de dados hierárquicos em rede predominantes por *mainframes*.

Por serem bancos de dados com características fortes, como capacidade de inserir grande quantidade de dados, rapidez na identificação e tratamento de erros, realização de pesquisa de dados de maneira rápida e garantir a integridade dos dados, sua expansão foi enorme e, aos poucos, acabou crescendo tanto que tornou-se sinônimo de “banco de dados”. Alguns bons exemplos de bancos de dados relacionais são DB2 (IBM), Oracle e MySQL (Oracle), SQL-Server (Microsoft), Firebird (software livre), Interbase (Borland) e PostgreSQL (software livre).

Agora que já sabemos quais são os bancos de dados relacionais existentes, vale lembrar que devemos sempre trabalhar para uma boa operacionalização deles e, para que isso ocorra, precisamos nos basear em normas concisas no momento da criação de um projeto de bancos de dados. Uma vez que nos aprofundemos nos estudos da teoria relacional, estaremos capacitados na criação de projetos que sejam corretos, consistentes e bem estruturados, evitando, assim, um futuro problema ou retrabalho.

Dada a relevância do assunto, estudaremos o modelo relacional introduzindo alguns conceitos fundamentais e importantes desse modelo, sempre mesclando a teoria com a prática e utilizando também exemplos concretos e que estejam participando mais proximamente da nossa rotina de trabalho e do nosso cotidiano.

DESENVOLVA SEU POTENCIAL

O MODELO RELACIONAL

Quando nos propomos a criar um banco de dados, temos que saber da importância do modelo relacional, pois é nele que nos basearemos para uma implementação inicial. O modelo relacional é um modelo da segunda geração que surgiu depois dos modelos pré-relacionais, hierárquicos e de rede. Os modelos que hoje tentam substituir são os de terceira geração, os pós-relacionais, os modelos orientados a objetos, objeto relacional, temporal e geográfico. O modelo relacional tem uma sólida base formal, sendo construído sob a teoria dos conjuntos. Seu nome é devido à relação matemática da teoria dos conjuntos, e não aos relacionamentos, como muitos pensam. Trata-se de um modelo com estruturas de tabelas e alguns conceitos.

O modelo relacional permite a representação da estrutura lógica do projeto com uma visão genérica. Sua estrutura é feita de forma clara e simples, possibilitando representar os dados do mundo real como objetos denominados entidades ou conjunto de entidade.

Aqui, utilizaremos a notação de Peter Chen (1990), notação essa que fora criada, em 1976, pelo Dr. Peter Pin-Shan Chen, que é um cientista da computação americano e professor de Ciência da Computação na Louisiana State University, conhecido por ser o criador do modelo entidade-relacionamento.

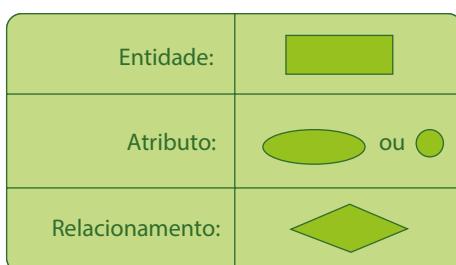


Figura 1: Componentes DER (diagrama entidade e relacionamento) / Fonte: os autores.

Entidades (Tabelas)

A entidade ou tabela é uma representação gráfica de um conjunto cuja representação física ou gráfica padrão é feita por meio de um retângulo com o nome da entidade dentro dele. Para identificarmos uma entidade, devemos considerar os objetos, coisas ou algo que seja relevante no levantamento dos dados. A seguir, estão alguns exemplos (Figura 2 e Figura 3):

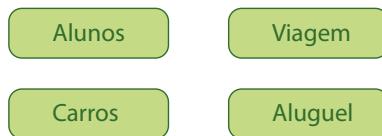


Figura 2: Exemplo de entidades / Fonte: os autores.

No exemplo dado na Figura 2, temos uma entidade “Alunos”, em que é representado um conjunto de alunos. Da mesma forma, podemos dizer que a entidade “Carros” representa um conjunto de carros e assim sucessivamente.

Com base no conceito de entidade, podemos classificar as entidades em concretas e abstratas, sendo que as entidades concretas são entendidas como objetos do mundo real que podem ser separados e distinguíveis de outro objeto. Já as entidades abstratas são aquelas que não temos de maneira tangível (intangível). Com isso, vale lembrar que, em ambos os tipos de entidades, a representação é a mesma. A seguir (Figura 3), classificaremos algumas entidades:



Figura 3: Exemplo de entidades concretas e entidades abstratas / Fonte: os autores.

INTRODUÇÃO À MODELAGEM

Agora que já sabemos o que é uma entidade, é importante entendermos o motivo pelo qual ela será criada. Para isso, podemos utilizar a descrição textual narrativa, sendo essa o levantamento de requisitos junto ao cliente, ou seja, uma entrevista

em que iremos retirar as informações devidas para a implementação do nosso sistema. Nesse momento, é importante anotarmos todas as necessidades do nosso cliente e, a partir dessas anotações, analisar os substantivos das frases. Caso esse substantivo seja relevante ao sistema, poderemos transformá-lo em uma entidade, por exemplo, na frase “a bibliotecária empresta um livro”, podemos retirar duas possíveis entidades, sendo elas:



Figura 4: Exemplo de entidades / Fonte: os autores.

Já na frase “o carro percorre vários itinerários”, podemos retirar duas possíveis entidades, sendo elas:



Figura 5: Exemplo de entidades / Fonte: os autores.

Assim, iremos popular o nosso sistema com as entidades necessárias para a implementação. Vale lembrar que essa fase é importantíssima e deve ser feita com muita atenção. Você pode se perguntar o motivo de tanta importância e vou exemplificar de maneira prática. A comparação dessa fase inicial é a mesma de uma obra de uma casa, pois, uma vez que for mal dimensionada a estrutura, podemos ter sérios e irreversíveis problemas no futuro. A seguir, seguem dois exemplos de uma análise a partir de uma descrição textual narrativa.

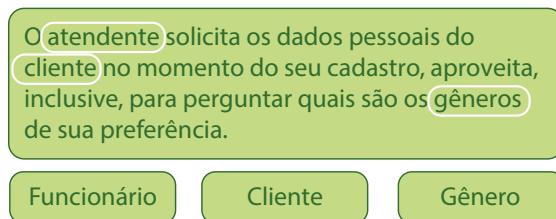


Figura 6: Exemplo 1 de descrição textual narrativa / Fonte: os autores.

Analizando o Exemplo 1 (Figura 6), temos como substantivos atendente, que podemos nomear como funcionário. Temos também as palavras “cliente” e “gêneros”. Com isso, para a implementação do que fora relatado e do que é relevante, teremos a necessidade de criação de três entidades, sendo elas: funcionário, cliente e gênero.

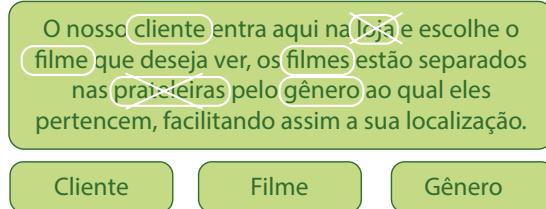


Figura 7: Exemplo 2 de descrição textual narrativa / Fonte: os autores.

Analizando o Exemplo 2 (Figura 7), temos como substantivos as palavras “cliente”, “loja”, “filme”, “filmes”, “prateleiras” e “gênero”. Veja que riscamos a palavra prateleiras, pois não temos a necessidade de cadastrar as prateleiras e a localização pode ser cadastrada diretamente na entidade “filme”. Outra palavra riscada foi “loja”, sendo que essa seria relevante e poderia ser aproveitada apenas se tivéssemos mais de uma loja, ou seja, tudo varia conforme a regra do negócio que está sendo analisado. Com isso, concluímos que as entidades que serão implementadas serão: cliente, filme e gênero.

ZOOM NO CONHECIMENTO

Dicas para uma boa modelagem:

- Ter em mente o cenário a ser modelado.
- Detectar os substantivos no momento da análise do sistema.
- Nomear apropriadamente as entidades detectadas.
- Padronizar os nomes (plural, singular, abreviações).
- Fazer o diagrama em um rascunho de próprio punho em papel.
- Definir o tipo de organização mais adequado.
- Realizar um bom levantamento do método manual e do procedimento manual junto ao principal usuário.

Fonte: o autor.

ATRIBUTOS

Os atributos são propriedades utilizadas para descrever uma entidade. Podemos afirmar que os atributos são as características contidas nas entidades, por exemplo, em uma entidade cliente, podemos relacionar os atributos CPF, nome, idade, endereço, bairro, cidade etc. Vamos ao exemplo passo a passo!

1º) Vamos imaginar uma entidade produto.

Ela será simbolizada da seguinte maneira:



2º) Agora, iremos relacionar alguns Atributos a essa Entidade:

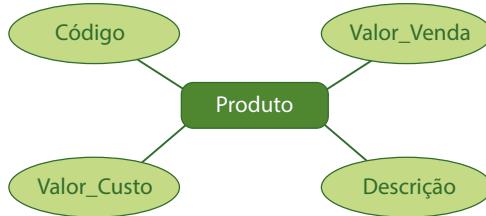


Figura 8: Exemplo de atributos em uma entidade / Fonte: os autores.

3º) Outra maneira de demonstrar uma entidade e atributos é em forma de planilha:

Observando a Figura 9, verificamos que a tabela Produtos contém atributos que são os nomes das colunas e um conceito novo chamado de tupla. Definindo de modo formal, uma linha é denominada de tupla.

PRODUTOS			
Código	Descrição	Valor_Custo	Valor_Venda
01	Lápis	1,00	2,00
02	Borracha	1,50	3,00
03	Mouse	10,00	20,00

→ Entidade
 → Atributos
 → Tupla

Figura 9: Exemplo de uma entidade em formato de colunas / Fonte: os autores.

TIPOS DE ATRIBUTOS

- **Atributo simples:** o atributo simples contém um único valor para cada elemento da entidade. Nesse caso, pode ocorrer uma informação re-

petida, ou seja, para uma entidade cliente, temos um nome para cada cliente, podendo acontecer de dois clientes terem o mesmo nome, mas informando um dado para cada cliente. A seguir, temos a representação do atributo simples:

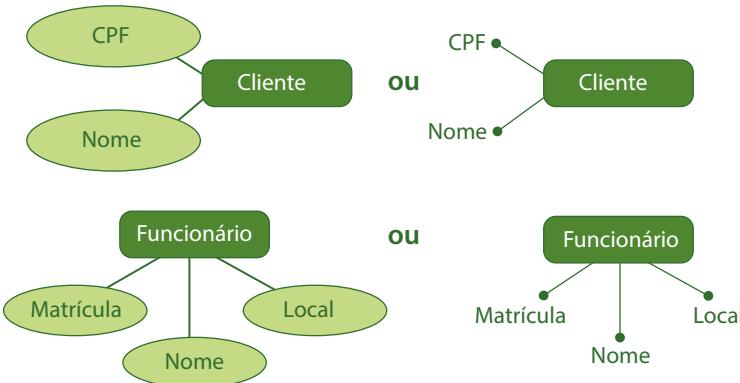


Figura 10: Exemplo de tipos de atributos / Fonte: os autores.

- **Atributo multivalorado:** o atributo multivalorado permite conter informações com diversos valores. É a solução do problema quando, por exemplo, você tem vários telefones para um funcionário. A seguir, temos a representação do atributo multivalorado:

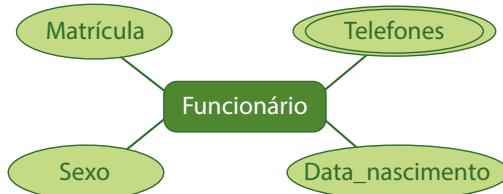


Figura 11: Exemplo de um atributo do tipo multivalorado / Fonte: os autores.

- **Atributo composto:** o atributo composto nos permite indicar um atributo que pode ser dividido em outros. Um exemplo pode ser o endereço, que podemos dividir em rua, cidade, estado e CEP. A seguir, temos a representação do atributo composto:

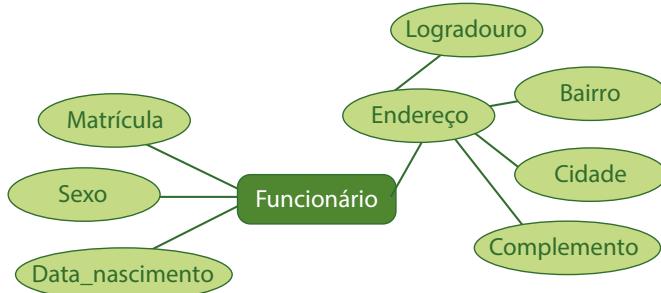


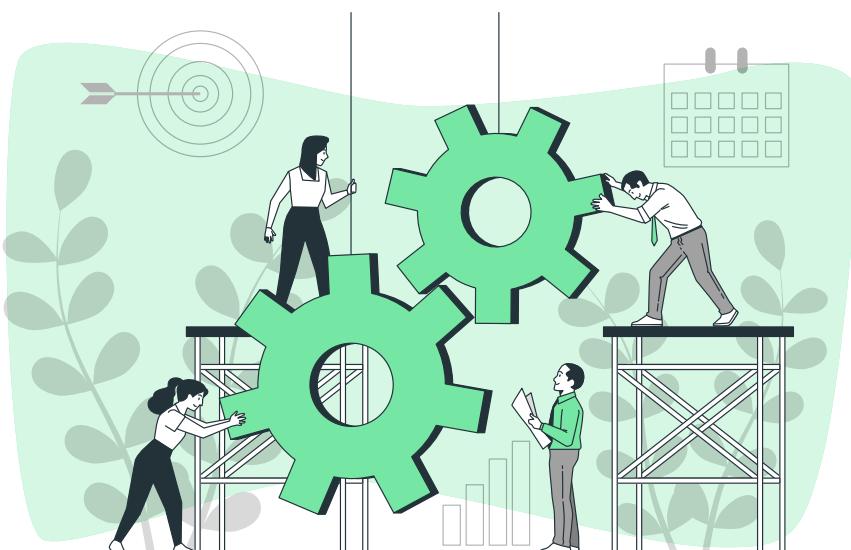
Figura 12: Exemplo de um atributo do tipo composto / Fonte: os autores.

- **Atributo-chave:** quando temos os atributos de uma entidade, é importante sempre indicarmos um identificador, que podemos chamar também de atributo-chave. Esse atributo irá identificar o item da entidade de maneira única (sem repetição) no conjunto de elementos.

O atributo-chave deve ser íntegro, ou seja, sem repetições, e não pode ser nulo (valores vazios). A sua representação pode ser demonstrada de maneira sublinhada ou com o círculo destacado na borda em negrito, conforme a Figura 13:



Figura 13: Exemplo de um atributo do tipo chave / Fonte: os autores.



 ZOOM NO CONHECIMENTO

10 curiosidades sobre o Google

O Google está na mídia o tempo todo, mas existem dados sobre a companhia que algumas pessoas ainda desconhecem. O Business Insider reuniu 10 fatos curiosos sobre a gigante da web que merecem atenção. Confira a seguir.

1. O Google.com, que abriga as mais importantes empresas do mundo, contém 23 erros em seu código.
2. A empresa já fotografou mais de 8 milhões de quilômetros para o Street View.
3. Originalmente, a companhia se chamaria "Googol", mas os investidores escreveram "Google" no primeiro cheque de contribuição e o nome permaneceu.
4. O banco de dados de buscas do Google tem mais 100 milhões de gigabytes. Seria necessário 100 mil HDs externos de 1 terabyte para armazenar todos esses dados.
5. O mundo assiste a mais de 450 mil anos de vídeos no YouTube por mês. Isso é mais que o dobro de anos de existência dos humanos modernos.
6. O Google usa o captcha para ensinar computadores a ler textos digitalizados de livros. São 200 milhões de captchas resolvidos por dia.
7. A página do Google tem um layout simples porque Sergey Brin e Larry Page não sabiam HTML. A dupla decidiu deixar o site da mesma forma para reforçar a identidade.
8. A gigante da web deve ser a única companhia que tem como objetivo explícito reduzir o tempo que as pessoas passam em seu site.
9. Na média, a companhia adquiriu mais de uma empresa por semana desde 2010.
10. Em 2011, 96% dos US\$ 37,6 bilhões em receita do Google vieram apenas de anúncios.

Fonte: Nuvens (2018, s.p.).

Exemplificando, para a entidade “Funcionários”, temos os seguintes atributos:

Endereço – analisando esse atributo, sabemos que pode haver mais de um funcionário morando no mesmo endereço, logo, ele não poderia ser classificado como atributo identificador.

Nome – esse atributo pode confundir um pouco, pois cada funcionário tem seu nome, porém pode haver funcionários com o mesmo nome. Logo, podemos perceber que o nome ser utilizado como um identificador pode nos trazer problemas.

Matrícula – para cada funcionário, é gerado um número de matrícula que o identificará na empresa e que não pode se repetir. Esse atributo pode ser classi-

ficado como atributo-chave, devendo ser destacado dentre os demais. A seguir, temos a representação do exemplo atributo-chave em uma entidade (Figura 14):

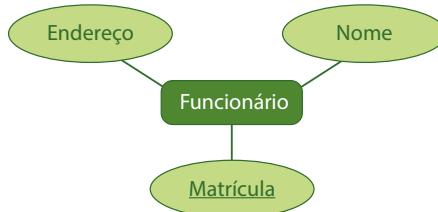


Figura 1.4: Exemplo de um atributo-chave em uma entidade / Fonte: os autores.

DOMÍNIO

As restrições de domínio especificam que o valor de cada atributo deve ser um valor atômico e, para cada atributo criado, devemos associar um tipo a ele. A essa associação, damos o nome de domínio. A seguir, demonstraremos alguns tipos:

- Numéricos – inteiros e reais.
- Caracteres.
- Booleanos.
- Cadeias de caracteres de tamanho fixo e tamanho variável.
- Data.
- Hora.

A seguir, criaremos uma tabela chamada “Funcionário”, contendo nome do campo, tipos e tamanho para exemplificar a aplicação dos tipos. Vejamos as propriedades dos campos da tabela “Funcionário”:

NOME DO CAMPOS	TIPO DO CAMPO	TAMANHO
Matrícula	Inteiro	3
Nome	Caractere	40
Idade	Inteiro	3
Data Admissão	Data	8

Fonte: o autor.

CHAVE ESTRANGEIRA (*FOREIGN KEY*)

Em nossos estudos, não podemos deixar de estudar a chave estrangeira ou, em inglês, *foreign key*, a qual trata-se de um campo que aponta para a chave primária de outra tabela. Nessa relação entre linhas (tuplas) de duas entidades, o objetivo da chave estrangeira é garantir a integridade dos dados referenciais, pois, nesses casos, serão permitidos valores que irão aparecer na base de dados. Vale lembrar que, após estabelecer uma chave estrangeira, o atributo marcado não permitirá a exclusão, inserção ou modificação de dados em tabelas que estejam dependentes umas das outras (“*foreign key*”), tendo que ter uma maior atenção dos administradores do banco de dados.

A seguir, exemplificaremos uma chave estrangeira entre duas tabelas (entidades) em que o relacionamento é 1:N:

Tabela: Departamento

Dept_Cod	Dept_Descr
1	Marketing
2	Compras
3	Almoxarifado

Tabela: Funcionário

Func_Matrícula	Func_Nome	Func_Dpto	Func_cpf
100200	João da Silva	1	041.304.279-33
100201	Maria Luisa da Rocha	3	041.304.279-34
100202	Bruna Pereira	2	045.304.279-35

Figura 15: Exemplo da aplicação de uma chave estrangeira entre duas tabelas
Fonte: os autores.

Outro exemplo:

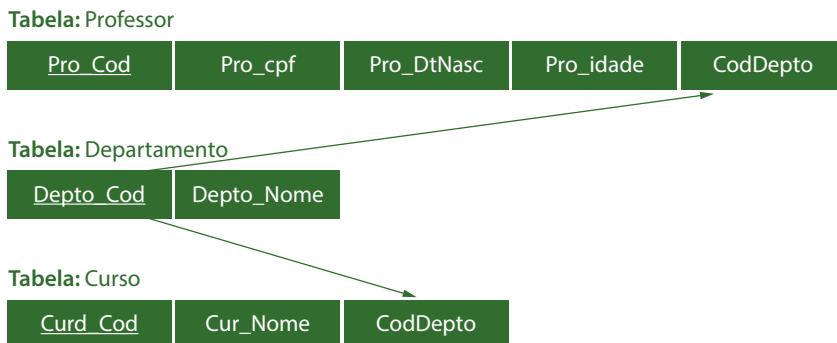


Figura 16: Exemplo da aplicação de uma chave estrangeira entre duas tabelas / Fonte: os autores.

RELACIONAMENTOS

Analisando o modelo relacional, as entidades não podem ficar isoladas, uma vez que as informações estarão organizadas futuramente para o acesso de forma integrada. Para essa organização sem perda de conteúdo, as entidades devem estar associadas, ligadas entre si. No Modelo Entidade Relacionamento (MER), não é permitido ligar uma entidade diretamente a outra. Quando há uma associação, ela é representada por um relacionamento e o relacionamento é apresentado na forma de um losango. Para a associação entre entidades, deve-se seguir a notação básica, que são entidades ligadas ao relacionamento por linhas retas, conforme a Figura 17. Sempre que um relacionamento for indicado, é necessário verificar a validade nos dois sentidos. Atenção! As setas não fazem parte do diagrama e são apenas para ilustrar os sentidos.



Figura 17: Exemplo de notação básica de relacionamentos / Fonte: os autores.

Para definir um relacionamento entre duas entidades, devemos verificar se há correlação entre elas e podemos fazer isso colocando um verbo para tentar associá-las. É importante averiguar se a associação entre as entidades é verdadeira em ambos os sentidos. Para entender melhor, podemos, na situação a seguir (Figura 18), descrever dizendo que “o funcionário trabalha no setor”.



Figura 18: Exemplo de notação básica de relacionamentos / Fonte: os autores.

Tipos de relacionamentos

A classificação dos relacionamentos é baseada no número de entidades que participam em um conjunto de relacionamentos, o que determina também o grau desse conjunto.

- **Autorrelacionamento ou relacionamento recursivo:** nesse caso, são enquadrados relacionamentos com apenas uma entidade. Exemplo:



Figura 19: Exemplo de relacionamento recursivo ou autorrelacionamento / Fonte: os autores.

- **Relacionamento binário:** o relacionamento binário é de grau dois, pois temos duas entidades. Exemplo:



Figura 20: Exemplo de relacionamento binário / Fonte: os autores.

- **Relacionamento ternário:** o relacionamento ternário é de grau três, pois temos três entidades associadas no relacionamento. Exemplo:



Figura 21: Exemplo de relacionamento ternário / Fonte: Cardoso e Cardoso (2012, p. 62).

Vale lembrar que, entre duas entidades, também pode haver relacionamento, ou seja, uma entidade pode estar associada a outra por mais de um relacionamento, conforme o exemplo a seguir:

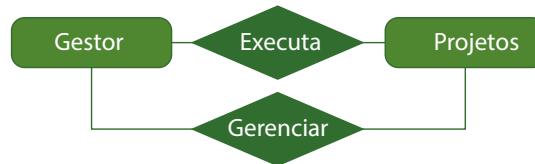


Figura 22: Exemplo de mais de um relacionamento entre duas entidades
Fonte: Cardoso e Cardoso (2012, p. 62).

Outra particularidade de um relacionamento é que os relacionamentos podem conter atributos. Eles não fazem parte de maneira obrigatória das propriedades das entidades, porém, quando inserimos um atributo associado a um relacionamento, ele deve ser comum às duas entidades. A seguir, vamos mostrar um exemplo relativo a esse tipo de relacionamento:



Figura 23: Exemplo de relacionamento contendo um atributo / Fonte: Cardoso e Cardoso (2012, p. 63).

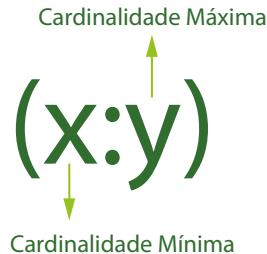
Conforme o exemplo citado (Figura 23), podemos dizer que o atributo horário é parte comum das entidades associadas no relacionamento, pois esse informa em que horário o palestrante ministra o referido tema.

CARDINALIDADE

A cardinalidade permite expressar o número de ocorrências com que uma entidade pode tomar parte em um relacionamento. Permite também expressar as possibilidades e as restrições de associações entre uma entidade e outra. Podemos também

definir como a “regra de negócio” entre as entidades envolvidas no relacionamento ou até como a frequência com que essas funcionalidades podem ocorrer.

Notação:



Cardinalidade Máxima

Trata-se do limite máximo de ocorrências de uma entidade em relação a outra:

- Um para um (1:1).
- Um para muitos (1:N).
- Muitos para muitos (N:N) ou N:M.

Cardinalidade um para um (1:1):

Acontece quando a ocorrência de uma entidade se relaciona com, no máximo, uma ocorrência de outra e vice-versa.

Exemplo:



Figura 24: Exemplo de cardinalidade um para um / Fonte: Cardoso e Cardoso (2012, p. 64).

Cardinalidade um para muitos (1:N):

A ocorrência de uma entidade se relaciona com, no máximo, muitas ocorrências de outra, porém a ocorrência de outra entidade se relaciona com, no máximo, uma ocorrência da primeira.



Figura 25: Exemplo de cardinalidade um para N / Fonte: Cardoso e Cardoso (2012, p. 65).

Cardinalidade muitos para muitos (N:N) ou (N:M):

Acontece quando a ocorrência de uma entidade se relaciona com, no máximo, muitas ocorrências de outra e vice-versa. Exemplo:



Figura 26: Exemplo de cardinalidade N para N / Fonte: Cardoso e Cardoso (2012, p. 65).

Quando a leitura é feita de (1:N) em ambos os sentidos das entidades, teremos, assim, um resultado (N:N), conforme a seguir (Figura 27):



Figura 27: Exemplo de cardinalidade N para N / Fonte: Cardoso e Cardoso (2012, p. 66).

Cardinalidade mínima

Trata-se do mínimo de ocorrências de uma entidade em relação a outra:

- **Opcional (0)** – é quando uma ocorrência se relaciona com, no mínimo, nenhuma de outra entidade. Abaixo, temos a representação:
- **(0:1)** – nesse caso, a representação textual seria “no mínimo, nenhuma ocorrência em uma entidade para, no máximo, uma ocorrência na outra entidade”.

- **(0:N)** – nesse caso, a representação textual seria “no mínimo, nenhuma ocorrência em uma entidade para, no máximo, muitas ocorrências na outra entidade”.
- **Obrigatória (1)** – uma ocorrência se relaciona com, no mínimo, uma de outra entidade.

No exemplo a seguir (Figura 28), devemos analisar a regra de negócio que, de acordo com a cardinalidade mínima ser marcada como 0 (zero), significa que o cliente não é obrigatório no momento da venda do produto e que o produto é comprado por, no máximo, um cliente.



Figura 28: Exemplo de cardinalidade N para N / Fonte: Cardoso e Cardoso (2012, p. 67).

NOVOS DESAFIOS

Procuramos demonstrar os assuntos de maneira que você possa entender melhor as questões que permeiam os conceitos básicos do modelo relacional de banco de dados. Vale enfatizar que tivemos a oportunidade de estudar a respeito de conceitos básicos do modelo relacional, como o Diagrama Entidade e Relacionamento (DER), tuplas, entidades, atributos, relacionamentos, cardinalidades etc. Esses conceitos são importantes para quem deseja aprender, de maneira correta, a iniciar a criação de um projeto de banco de dados, tendo como principal objetivo dar a você, estudante, uma melhor aplicabilidade no momento da iniciação de um projeto.

Espero que esses conceitos tenham sido explanados de maneira clara e objetiva, sempre buscando exemplificar para que seja direto ao ponto e, assim, possa facilitar o seu estudo, entendimento e aprendizado. Aproveito o momento para deixar aqui o meu incentivo à fixação desse conteúdo, pois esses conceitos são imprescindíveis para quem deseja iniciar o desenvolvimento de um banco de dados.

AGORA É COM VOCÊ

1. A partir do estudado neste tema, defina Entidades Concretas e Entidades Abstratas.

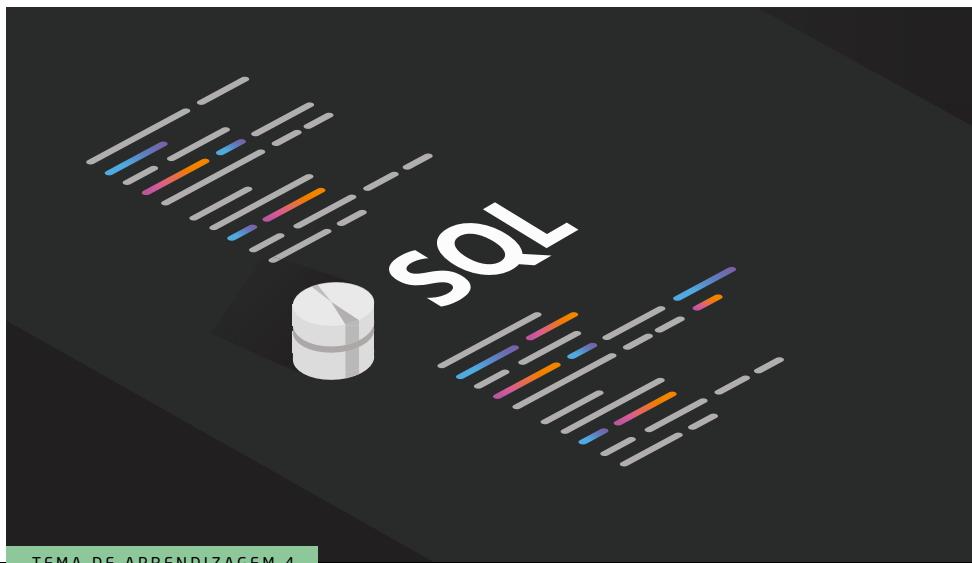
2. Crie uma Entidade Produtos com os seguintes atributos:
 - a) Código do Produto
 - b) Descrição do Produto
 - c) Unidade do Produto
 - d) Valor do Produto
 - e) Classificação do Produto
 - f) Valor Custo do Produto

3. Analise as frases abaixo e crie as possíveis entidades:
 - a) "... o atendente matricula o aluno no curso de Administração..."
 - b) "... a secretária agenda pacientes para atendimento médico..."
 - c) "...é necessário cadastrar os produtos para realizar as vendas aos clientes..."

MEU ESPAÇO

MEU ESPAÇO





TEMA DE APRENDIZAGEM 4

SQL BÁSICO

ME. EDSON YANAGI

MINHAS METAS

- Definir o que é SQL.
- Apresentar os comandos de definição e uso da SQL.

INICIE SUA JORNADA

A primeira ideia que vem à cabeça de um desenvolvedor experiente, quando se fala de banco de dados relacionais, é SQL. Talvez uma das boas razões pelas quais os Sistemas Gerenciadores de Banco de Dados (SGDBs) relacionais são tão difundidos deve-se ao fato de que a SQL é uma ferramenta bastante madura, bem elaborada e projetada.

Em português, pronuncia-se SQL como uma sigla, com o som de cada uma das letras separadas, “esse-quê-ele”. Porém, nas conversas em projetos internacionais que você fará ou em conferências das quais você participará em sua vida profissional, perceberá que, em inglês, SQL é pronunciado como “síquel”. O motivo é curioso e não tem a ver com a sigla SQL, mas com o nome original dessa linguagem. Atualmente, a sigla SQL significa *Structured Query Language* (Linguagem de Consulta Estruturada), mas originalmente seu nome era *Structured English QUERy Language* (SEQUEL – Linguagem de Consulta em Inglês Estruturado), por isso o motivo da pronúncia como “síquel”.

SQL é uma linguagem diferente das linguagens de programação que você provavelmente aprendeu até agora. Em qualquer curso de programação, costuma-se ensinar inicialmente linguagens de programação imperativas (como C, Pascal, Java ou Python), em que você é responsável por escrever os comandos na ordem de execução esperada. Nesse tipo de linguagem, preocupamo-nos em instruir o computador no modo como ele deve executar as tarefas. O resultado de seu processamento é uma consequência daquilo que comandamos. Já a SQL é uma linguagem declarativa, pois nela define-se o que deve ser retornado como resultado do processamento, sem especificar como isso será feito.

Permita-nos realizar uma reflexão sobre a natureza declarativa da SQL. Na SQL, ao definirmos somente o que esperamos de resultado em vez do como, permitimos que o SGBD decida como é que ele deve executar as instruções. Há 10 anos, esse seria um fator determinante para decidir entre um produto e outro. A evolução dos produtos comerciais e livres fez com que essa diferença diminuisse de modo significativo – embora, dependendo dos casos de uso de sua aplicação, a diferença ainda possa ser relevante. Em alguns casos, centenas de parâmetros de configuração do produto permitem alterar a forma com que o SGBD executa o “como”: uma tarefa que, muitas vezes, chega a ser minuciosa – tudo isso para conseguir aumentar o desempenho do seu SGBD.

Para tentar diminuir as diferenças entre as diversas implementações e variantes da SQL utilizadas em produtos distintos, a American National Standards Institute (ANSI) e a International Standards Organization (ISO) uniram-se para criar um padrão para a SQL. A versão mais popular desse padrão é a SQL-92, embora haja uma versão mais recente: a SQL:1999. Infelizmente, os fabricantes de SGBDs não seguem 100% o padrão, o que torna a tarefa de migração de um produto para outro um pouco mais difícil. Comercialmente, é uma estratégia interessante para os fabricantes, pois se baseia no aprisionamento do cliente: uma vez comprometido com um produto, o custo para migração (em tempo e esforço) torna-se elevado o suficiente para que o cliente desista da ideia. Já para os usuários, esse é um fato infeliz. De positivo do padrão SQL-92, temos que, apesar das sutis diferenças entre as implementações da SQL em diferentes produtos, as semelhanças se sobressaem e permitem que os desenvolvedores de software possam aprender facilmente a lidar com produtos concorrentes.

A SQL possui comandos tanto para a criação de definições de dados (criação de *schemas*) quanto para a execução de comandos de manipulação de banco de dados (consultas e atualizações). É uma linguagem bastante abrangente. É por esse motivo que trataremos da SQL agora.

DESENVOLVENDO SEU POTENCIAL

DEFINIÇÕES DE DADOS E TIPOS EM SQL

Em SQL, os termos relação, tupla e atributo do modelo relacional são denominados respectivamente de tabela, linha e coluna. Boa parte dos comandos SQL relacionados à criação e definição de dados utiliza o comando “create”.

O Conceito de Schema

Em sua versão inicial, a SQL não possuía um mecanismo para agrupar tabelas relacionadas. Como consequência, todas as tabelas no SGBD coexistiam dentro de um mesmo “ambiente”. A partir do SQL-92, criou-se o conceito de *schema*, que é simplesmente um conjunto de tabelas relacionadas. Do mesmo modo que

em UML um pacote é um conjunto de classes relacionadas, um *schema* é um conjunto de tabelas relacionadas. Por exemplo, o seguinte comando cria um *schema* denominado de “agenda”. Todos os comandos em SQL são finalizados por um ponto e vírgula:

```
create schema agenda;
```

Além de agrupar logicamente as tabelas, um *schema* também é convenientemente utilizado para autorizar/restrinir o acesso pelos usuários. Você pode autorizar determinados usuários a acessar um *schema* e restringir o acesso a outros.

O comando CREATE TABLE

O comando CREATE TABLE é utilizado para criar uma tabela. O primeiro parâmetro do comando é o nome da tabela a ser criada, seguido dos atributos e de seus respectivos tipos e eventuais restrições do atributo. Restrições de integridade referencial podem ser definidas ao final do comando.

contato

id	nome	sobrenome	nascimento	peso

email

id	email	contato_fk

telefone

id	telefone	contato_fk

grupo

id	nome

afiliação

grupo_fk	contato_fk

Figura 1: Exemplo da estrutura de tabelas e seus respectivos atributos / Fonte: os autores.

Os seguintes comandos criam as tabelas definidas na Figura 1:

```
CREATE TABLE contato (
    id INT PRIMARY KEY,
    nome VARCHAR(30) NOT NULL,
    sobrenome VARCHAR(30) NOT NULL,
    nascimento DATE,
    peso DECIMAL(10,2)
);
```

```
CREATE TABLE email (
    id INT PRIMARY KEY,
    email VARCHAR(60) NOT NULL,
    contato_fk INT,
    FOREIGN KEY (contato_fk) REFERENCES contato(id)
);
```

```
CREATE TABLE telefone (
    id INT PRIMARY KEY,
    telefone VARCHAR(20) NOT NULL,
    contato_fk INT,
    FOREIGN KEY (contato_fk) REFERENCES contato(id)
);
```

```
CREATE TABLE grupo (
    id INT PRIMARY KEY,
    nome VARCHAR(30) NOT NULL
);
```

```
CREATE TABLE afiliacao (
    grupo_fk INT NOT NULL,
    contato_fk INT NOT NULL,
    PRIMARY KEY (grupo_fk, contato_fk),
    FOREIGN KEY (grupo_fk) REFERENCES grupo(id),
    FOREIGN KEY (contato_fk) REFERENCES contato(id)
);
```

Em algumas situações, não é possível definir as restrições de integridade referencial e chave estrangeira no próprio CREATE TABLE, pois as tabelas referenciadas ainda não existem.

TIPOS DE DADOS

A SQL define um conjunto de tipos de dados básicos para os seus atributos. Diferentes produtos adicionam tipos de dados diferentes ao conjunto suportado, mas todos os produtos disponíveis no mercado suportam ao menos estes tipos básicos:

TIPOS NUMÉRICOS	Tipos de dados numéricos suportam dados inteiros (INT e SMALLINT) e de ponto flutuante (FLOAT e DOUBLE). Números com precisão decimal (normalmente utilizados em cálculos de moeda, por exemplo) são DECIMAL ou NUMERIC, declarados como DECIMAL(a,b) ou NUMERIC(a,b), em que “a” é o número de dígitos inteiros e “b” é o número de dígitos decimais.
TIPOS DE CARACTERE	Tipos de caractere podem ser de tamanho fixo ou variável. Os atributos de tamanho fixo podem ser declarados como CHAR(n), em que “n” é o número máximo de caracteres suportado pelo atributo. Para especificar um atributo de tamanho variável, utiliza-se o tipo VARCHAR(n). Para se entender o critério de uso entre um e outro, é necessário entender como é a alocação de espaço desses tipos no arquivo físico. Se o tamanho for fixo (CHAR), o SGBD já aloca esse tamanho predefinido no arquivo: as buscas podem ser mais rápidas, pois o SGBD já sabe o tamanho do campo ao “pular bytes” na busca sequencial. Entretanto, se o conteúdo dos atributos não preencher todo o tamanho definido, há o desperdício de espaço de armazenamento. Por outro lado, utilizando-se VARCHAR, o SGBD aloca um ponteiro para determinar qual o tamanho do atributo: as buscas são mais lentas, mas não há desperdício de espaço.

TIPOS BOOLEANOS	Assim como em linguagens de programação, tipos booleanos podem assumir os valores TRUE ou FALSE. Muitos SGBDs mapeiam esses valores em "1" e "0", respectivamente.
TIPOS TEMPORAIS	O tipo DATE suporta dados temporais no formato AAAA-MM-DD (ano, mês, dia), enquanto o tipo TIME utiliza o formato HH:MM:SS (hora, minuto e segundo). A própria SQL assegura representações temporais válidas.

RESTRIÇÕES

Abordaremos agora as restrições que podem ser declaradas em SQL no comando “create table”.

Valores Null e Valores Padrão

A linguagem SQL permite que todos os atributos (com exceção daqueles que compõem a chave primária) sejam nulos. Se o seu modelo de negócios não permite que um atributo seja nulo, é necessário especificar uma restrição de “*not null*” na declaração do atributo.

Outra consideração (pequena talvez) sobre o “*not null*” é que, no mínimo, o SGBD terá que gravar um bit (ou um byte) a mais em cada atributo em casos de campos “*null*”. Se um atributo permitir nulos, então o SGBD terá que, primeiramente, saber se o campo é nulo ou não e, depois, armazenar o próprio conteúdo.

Além de valores nulos, também há possibilidade de se definir um valor padrão para os atributos utilizando-se a cláusula “DEFAULT <valor>”. Caso esse atributo seja omitido durante a inserção de uma linha da tabela, assume-se o valor padrão.

Por padrão na SQL, caso nenhuma cláusula seja declarada, os atributos permitirão valores nulos e o valor padrão também será nulo. Como exemplo, poderíamos definir “Silva” como o sobrenome padrão dos nossos contatos no comando CREATE TABLE:

```
CREATE TABLE contato (
    id INT PRIMARY KEY,
    nome VARCHAR(30) NOT NULL,
    sobrenome VARCHAR(30) NOT NULL DEFAULT 'Silva',
    nascimento DATE,
    peso DECIMAL(10,2)
);
```

Chaves e Integridade Referencial

Para especificar uma chave primária, utiliza-se a cláusula “PRIMARY KEY”. Caso a chave primária possua um único atributo, ela pode ser declarada no próprio atributo (como no exemplo da tabela contato):

```
id INT PRIMARY KEY
```

Havendo mais de um atributo na chave primária, é necessário declará-la ao final, como no exemplo a seguir:

```
PRIMARY KEY (grupo_fk, contato_fk)
```

A cláusula “UNIQUE” especifica chaves únicas (não primárias) em uma tabela. Poderíamos alterar a definição da tabela e-mail para garantir que não haja e-mails duplicados em nossa aplicação:

```
CREATE TABLE email (
    id INT PRIMARY KEY,
    email VARCHAR(60) NOT NULL UNIQUE,
    contato_fk INT,
    FOREIGN KEY (contato_fk) REFERENCES contato(id)
);
```

Já a integridade referencial e as chaves estrangeiras são definidas por meio da cláusula “FOREIGN KEY”, como já utilizada no nosso exemplo nas tabelas e-mail e telefone:

```
CREATE TABLE email (
    id INT PRIMARY KEY,
    email VARCHAR(60) NOT NULL,
    contato_fk INT,
    FOREIGN KEY (contato_fk) REFERENCES contato(id)
);
```

```
CREATE TABLE telefone (
    id INT PRIMARY KEY,
    telefone VARCHAR(20) NOT NULL,
    contato_fk INT,
    FOREIGN KEY (contato_fk) REFERENCES contato(id)
);
```

CONSULTAS BÁSICAS EM SQL

O comando em SQL para a execução de operações de consulta é o SELECT, e as consultas que podem ser elaboradas com esse comando variam das mais simples até

as bem complicadas. Em sua forma fundamental, um comando de consulta SELECT assume a forma SELECT <atributos> FROM <tabelas> WHERE <condições>.

De acordo com Silberschatz, Korth e Sudarshan (1999), a expressão básica de consulta em SQL consiste em três cláusulas: SELECT, FROM e WHERE:

1. A cláusula de SELECT é utilizada para listar os atributos desejados no resultado da consulta.
2. A cláusula FROM lista as relações (tabelas) que devem ser examinadas na avaliação da expressão SQL.
3. A cláusula WHERE corresponde ao predicado envolvendo os atributos das relações que aparecem na cláusula FROM.

As condições do comando SQL podem utilizar comparadores lógicos similares aos de outras linguagens de programação já conhecidas, tais como = (igual), < (menor), <= (menor ou igual), > (maior), >= (maior ou igual) e <> (diferente).

Provavelmente, uma das melhores formas de se aprender é por meio de exemplos. Em vez de nos atermos aos detalhes sintáticos e conceituais de cada tipo de consulta, apresentaremos exemplos de consultas a seguir.

Todos os exemplos de consulta serão realizados tendo-se como base a definição de esquema proposto na figura a seguir.



Figura 2: Exemplo de entidades / Fonte: os autores.

Exemplo 1: selecione o nome e o telefone de todos os contatos cujo sobrenome seja “Silva”.

```

SELECT nome, telefone FROM contato, telefone WHERE contato.id =
telefone.contato_fk AND contato.sobrenome = 'Silva';

```

Nesse exemplo, a condição contato.sobrenome = ‘Silva’ é um predicado que seleciona somente as linhas especificadas na tabela contato. A condição contato.id = telefone.contato_fk é denominada de condição de *join*, pois combina duas tabelas diferentes baseadas, nesse caso, em uma chave estrangeira de um relacionamento entre ambas.

Exemplo 2: selecione o nome, o telefone e o e-mail de todos os contatos cujo nome seja “Edson”.

```
SELECT nome, telefone, email  
FROM contato, telefone, email  
WHERE contato.id = telefone.contato_fk AND contato.id = email.contato_  
fk AND contato.nome = ‘Edson’;
```

Nesse exemplo, pode-se notar que, em uma consulta, é permitido realizar o *join* entre várias tabelas diferentes e, novamente nesse caso, todas estão relacionadas por meio de uma condição de combinação baseada em chaves estrangeiras.

Exemplo 3: selecione o id do telefone de todos os contatos cujo peso seja maior que 70.

```
SELECT telefone.id  
FROM contato, telefone  
WHERE telefone.contato_fk = contato.id  
AND contato.peso >70
```

O Exemplo 3 mostra como devemos definir os nomes dos atributos nas cláusulas quando há a possibilidade de ambiguidade na definição dos nomes. Tanto a tabela contato quanto a tabela telefone possuem um atributo denominado de “id”.

Nesse caso, devemos informar à SQL qual é o atributo “id” que desejamos obter, prefixando o atributo com o nome de sua respectiva tabela. No caso do exemplo, eliminamos a ambiguidade descrevendo o atributo como “telefone.id”.

Exemplo 4: selecione o nome do grupo e o nome do contato de todos os contatos cujo nome seja “Joaquim”.

```
SELECT contato.nome as n, grupo.nome as g
FROM contato, grupo, afiliacao
WHERE contato.id = afiliacao.contato_fk AND grupo.id = afiliacao.grupo_
fk AND n = 'Joaquim';
```

Uma facilidade na construção de consultas que têm termos repetidos sendo referenciados é a criação de um *alias*. Um *alias* é um apelido definido para um determinado termo da consulta SQL. No Exemplo 4, definimos que o atributo contato.nome possui um *alias* “n” e que o atributo grupo.nome possui um *alias* “g”. Desse modo, no restante dessa consulta, não precisamos mais nos referenciar a esses atributos pela referência completa, torna-se possível, então, simplesmente escrevermos a condição final da consulta como sendo n = ‘Joaquim’.

Exemplo 5: selecione o peso de todos os contatos com peso < 100.

```
SELECT c.peso
FROM contato c
WHERE c.peso < 100;
```

No Exemplo 5, demonstramos que a definição de um *alias* não está restrita aos atributos. Um *alias* pode também ser definido como um apelido para uma tabela na consulta SQL.

Exemplo 6: selecione a data de nascimento de todos os contatos.

```
SELECT nascimento  
FROM contato;
```

A cláusula WHERE de uma consulta SQL é opcional. Embora, em muitos casos, você, como programador(a), deva se questionar se isso é oportuno ou não, já que, potencialmente, a quantidade de registros em uma tabela pode chegar a milhões ou bilhões. Quando a cláusula WHERE é omitida, a consulta irá processar todos os registros das tabelas referenciadas. No Exemplo 6, demonstramos como obter todas as datas de nascimento por meio do processamento de todas as linhas da tabela contato.

Exemplo 7: selecione todos os atributos de contatos cujo peso = 75.

```
SELECT *  
FROM contato  
WHERE peso = 75;
```

Quando não se deseja limitar quais atributos devem ser retornados na consulta, pode-se utilizar um asterisco (“*”) para determinar ao SQL que processe todos os atributos de todas as tabelas da cláusula FROM no resultado da consulta SQL.

Exemplo 8: selecione todos os sobrenomes distintos de todos os contatos.

```
SELECT DISTINCT sobrenome  
FROM contato;
```

Embora o modelo relacional seja baseado na teoria geral dos conjuntos e, matematicamente, em conjuntos não haja elementos repetidos, permite-se elementos repetidos em tabelas e, consequentemente, nos resultados das consultas, esses elementos repetidos são exibidos. Nas situações em que se deseja eliminar as tuplas repetidas nos resultados das consultas, pode-se utilizar a cláusula “DISTINCT”, como no Exemplo 8.

Exemplo 9: selecione todos os telefones cujo número comece com “44”.

```
SELECT *
FROM telefone
WHERE telefone LIKE '44%';
```

No Exemplo 9, utilizamos o comparador “LIKE” para definir uma busca por padrões em *strings*. O caractere “%” é utilizado em condições LIKE para definir zero ou mais caracteres. Nesse exemplo, o “44%” determina que a *string* deve iniciar com “44” e pode possuir zero ou mais caracteres posteriores.

Exemplo 10: selecione todos os contatos que nasceram na década de 1980.

```
SELECT *
FROM contato
WHERE nascimento LIKE '198_ - __ - __';
```

Outro caractere especial que pode ser utilizado em condições “LIKE” é o “_”. Ele representa um único caractere arbitrário utilizado na busca. Como as datas em SQL podem ser representadas como uma *string* “AAAA-MM-DD”, utilizamos o “_” para preencher os campos da nossa busca.

Exemplo 11: selecione todos os contatos cujo peso esteja entre 90 e 100.

```
SELECT *
FROM contato
WHERE peso BETWEEN 90 AND 100;
```

A condição “BETWEEN” da SQL pode ser utilizada para determinar intervalos de valor em comparações.

Exemplo 12: selecione o nome de todos os contatos por ordem alfabética crescente.

```
SELECT nome
FROM contato
ORDER BY nome ASC;
```

A cláusula “ORDER BY” da SQL permite que o resultado da busca seja ordenado de acordo com os parâmetros informados. Uma cláusula “ORDER BY” pode ordenar os resultados de modo ascendente ou descendente. Para tanto, basta adicionar respectivamente o modificador “ASC” ou “DESC” na cláusula. O valor “ASC” é o padrão e o valor assumido caso o modificador seja omitido.

Exemplo 13: selecione o nome e o sobrenome de todos os contatos cujo sobrenome inicie com “A” e ordene por sobrenome em ordem decrescente e por nome em ordem crescente.

```
SELECT nome, sobrenome  
FROM contato  
WHERE sobrenome LIKE 'A%'  
ORDER BY sobrenome DESC, nome ASC;
```

No Exemplo 13, demonstramos como ordenar o resultado de uma consulta a partir de dois critérios diferentes. Os critérios são avaliados pela ordem em que são declarados.

COMANDOS DE MODIFICAÇÃO DE DADOS EM SQL

Até agora, pudemos definir quais são os comandos básicos da SQL para a execução de consultas básicas em nossos bancos de dados. A partir de agora, abordaremos os comandos da SQL que permitem a adição, a atualização e a remoção de tuplas (linhas), que, respectivamente, correspondem ao *insert*, *update* e *delete*.

O comando INSERT

O comando INSERT é utilizado para inserir linhas em uma determinada tabela. Devido à definição formal do *schema* da tabela, precisamos informar os valores de inserção na tabela dentro de uma ordem específica. Essa ordem pode ser a própria ordem determinada pela definição do *schema* ou pode ser a ordem em que definimos os nomes das colunas da cláusula de INSERT. O comando INSERT, em sua forma mais simples, pode ser exemplificado do seguinte modo:

```
INSERT INTO contato  
VALUES (10, 'Edson', 'Yanaga', '1978-04-12', 95);
```

Nesse exemplo, determinamos que os valores da cláusula “VALUES” seguem a mesma ordem da definição do *schema*, como definido no início desta unidade. Note que, durante a inserção, os valores informados devem satisfazer as condições de restrições de domínio, de integridade nula e de integridade referencial.

```
INSERT INTO contato (nome, sobrenome, peso, nascimento, id)
VALUES ('Edson', 'Yanaga', 95, '1978-04-12', 10);
```

Nessa segunda forma do comando INSERT, nós especificamos explicitamente qual a ordem desejada de inserção de cada um dos atributos da tabela contato. Uma questão que surge para os desenvolvedores é: qual seria a forma mais adequada? Certamente, não há uma resposta que possa ser considerada melhor ou pior, mas um argumento a favor da segunda forma estabelece que, quando a ordem dos argumentos é especificada no próprio comando INSERT, evita-se que este torne-se inválido quando o *schema* da tabela for modificado para adição ou alteração de ordem de alguma coluna. O fato de o comando tornar-se inválido provavelmente provocaria um erro da aplicação em tempo de execução, já que esse tipo de bug não pode ser identificado pelo compilador.

Uma terceira forma do comando INSERT permite que os valores informados para inserção sejam determinados por uma cláusula SELECT, em vez de serem argumentos literais na própria cláusula. Podemos criar uma tabela adicional no nosso *schema* para armazenar esses dados provenientes do comando SELECT:

```
CREATE TABLE lista_de_nomes (
    nome varchar(30),
    sobrenome varchar(30)
);
```

Baseando-se nos dados já existentes na tabela contato, a recém-criada tabela lista_de_nomes poderia ser populada com o seguinte comando INSERT:

```
INSERT INTO lista_de_nomes (nome, sobrenome)
SELECT nome, sobrenome
FROM contato
WHERE nascimento > '1980-01-01';
```

O comando UPDATE

O comando UPDATE modifica os valores de uma ou mais tuplas (linhas) das tabelas selecionadas. Nesse comando, a cláusula WHERE determina quais são as linhas da tabela selecionadas para modificação. Em sua forma fundamental, um comando de modificação UPDATE assume a forma UPDATE <tabela> SET <atributos e valores> WHERE <condições>. Note que, diferentemente do comando SELECT, o comando UPDATE só pode ser aplicado em uma única tabela. Caso seja necessário modificar os valores de atributos de mais de uma tabela, vários comandos UPDATE terão que ser executados – todos possivelmente agrupados dentro de uma única transação.

```
UPDATE contato
SET peso = 99, nascimento = '1982-04-25'
WHERE nome = 'Carlos';
```

Nesse exemplo do comando UPDATE, estamos modificando o valor de dois atributos das tuplas cujo nome = ‘Carlos’. É uma prática bastante comum executarmos comandos UPDATE no banco de dados somente identificando a chave primária na cláusula WHERE. Assim, temos a garantia de que um único registro será modificado de cada vez, já que cada chave primária é única dentro de uma mesma tabela, se assim for o caso de uso desejado.

```
UPDATE contato
SET peso = peso * 1.1;
```

Assim como no comando SELECT, a cláusula WHERE é opcional também no comando UPDATE. Nesse caso, todas as linhas da tabela informada serão selecionadas para a execução das modificações solicitadas. No exemplo anterior, demonstramos que podemos executar operações aritméticas com os valores dos atributos das tabelas. Nesse exemplo, atualizamos para 10% acima o peso de todos os contatos, no caso de uma hipotética epidemia de obesidade.

```
UPDATE contato  
SET nascimento = NULL;
```

O valor nulo também pode ser utilizado como valor de atribuição em comandos UPDATE, desde que as restrições do *schema* do banco de dados assim o permitam, assim como o comando INSERT. Vale lembrar que todas as restrições do *schema* que se aplicam ao comando INSERT também são válidas para o comando UPDATE.

O comando DELETE

O comando DELETE na SQL remove linhas de uma determinada tabela. Assim como os comandos INSERT e UPDATE, ele possui uma cláusula WHERE para limitar as linhas que serão processadas pelo comando. Novamente, assim como nos comandos INSERT e UPDATE, a ausência da cláusula WHERE implica que todas as linhas de uma determinada tabela serão processadas, o que, no caso do comando DELETE implica que o resultado será uma tabela vazia.

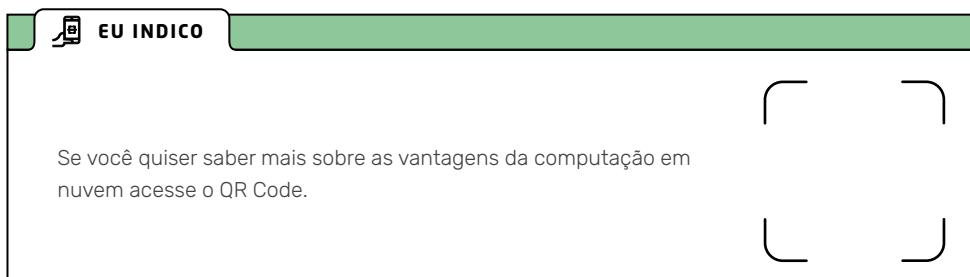
Em sua forma fundamental, um comando de modificação DELETE assume a forma DELETE FROM <tabela> WHERE <condições>.

```
DELETE FROM telefone  
WHERE id = 5;
```

O exemplo anterior é a forma mais comum de execução do comando DELETE. A exemplo do comando UPDATE, o comando DELETE só pode ser aplicado em uma única tabela de cada vez.

```
DELETE FROM contato;
```

Em um caso extremo, esse exemplo resultaria em uma tabela contato vazia. Entretanto, como nas operações de DELETE as restrições de integridade referencial são verificadas, esse comando falharia com um erro caso alguma outra tabela (telefone, por exemplo) tivesse alguma chave estrangeira apontando para uma linha da tabela contato.



NOVOS DESAFIOS

Temos a convicção de que você, como profissional comprometido(a) e fluente em inglês (sim, na área de Tecnologia da Informação, inglês é obrigatório e deveria ser o idioma principal), já abordará seus(suas) colegas, alunos(as) e profissionais, falando “síquel” em vez do famigerado “esse-quê-ele”, quando se referir à linguagem SQL.

Como toda tecnologia e assunto novo, SQL exige prática para o domínio. Acreditamos piamente na educação por meio de exemplos como a melhor forma de se formar profissionais que consigam utilizar os conhecimentos assimilados na execução prática das tarefas. Durante este tema, pudemos estudar a formação dos comandos INSERT, UPDATE e DELETE e suas respectivas sintaxes e cláusulas individuais. Em seguida, por meio de exemplos, praticamos uma série de diferentes definições de comandos e explicamos o que se esperava de cada um deles, bem como sua motivação.

Crie seus próprios *schemas* baseado(a) nas abstrações reais do mundo que o cerca, exercite-se e execute consultas e comandos de modificação SQL nesses seus *schemas*! Com a prática cotidiana, você perceberá que SQL também é bastante simples.

Até agora, fomos capazes de abordar as estruturas básicas da linguagem SQL para que nos nossos próximos passos possamos nos dedicar a alguns casos mais elaborados de uso dessa popular linguagem.

AGORA É COM VOCÊ

Todas as atividades deste tema baseiam-se na figura abaixo.

aluno				
id	nome	sobrenome	ra	email

professor			
id	nome	sobrenome	titulação

curso		
id	nome	ano

matrícula	
curso_fk	aluno_fk

disciplina			
id	nome	curso_fk	professor_fk

1. Considere o exemplo de schema da figura apresentada. Crie os comandos SQL para definição e criação das tabelas.

2. Elabore consultas SQL para selecionar:
 - a) O nome de todos os alunos matriculados no curso com nome = 'Banco de Dados'.
 - b) A titulação do professor da disciplina com nome = 'SQL'.
 - c) O Nome, sobrenome e RA de todos os alunos matriculados nas disciplinas lecionadas pelo professor com nome 'Edson'.
 - d) Todos os atributos de todos os cursos com ano > 1990.

3. Elabore comandos de modificação de dados para incluir, modificar e remover linhas das diferentes tabelas desse schema.



MINHAS METAS

- Definir consultas complexas em SQL.
 - Apresentar os comandos de alteração de definições em SQL.

INICIE SUA JORNADA

Você provavelmente já aprendeu a sintaxe básica dos comandos SQL, então podemos nos aventurar em consultas um pouco mais complexas. Em praticamente todos os sistemas de bancos de dados disponíveis no mercado, sejam eles relacionais ou NoSQL, as funções de consulta e manipulação básicas se equivalem. Isso significa que ainda não foi possível perceber um dos diferenciais competitivos dos bancos de dados relacionais, que é justamente a capacidade de se executar essas consultas um pouco mais complexas e sofisticadas.

Quando citamos essas consultas “um pouco mais complexas” do SQL, não o fazemos com o propósito de intimidá-lo(a). Muito pelo contrário. Aprendemos com a teoria geral dos sistemas que sempre que há um problema “complexo”, é possível dividi-lo em problemas menores até que esses sejam de fácil resolução. É com essa definição em mente que apresentaremos nesta unidade uma grande variedade de exemplos que podem ser solucionados com essas consultas mais elaboradas da SQL.

Abordaremos consultas envolvendo valores nulos, *subqueries*, consultas com *joins* e consultas com funções de agregação, além dos comandos de alteração de *schema*.

DESENVOLVENDO SEU POTENCIAL

CONSULTAS ENVOLVENDO NULL

O valor NULL representa um valor ausente, mas que pode ter diferentes interpretações. Algumas possibilidades de uso para o valor NULL são:

- **O valor é desconhecido.** Pense que um telefone, por exemplo, pode ser NULL se você não sabe o valor do telefone do contato.
- **O valor não está disponível.** No caso do telefone, você conhece o número do telefone do contato, mas não gostaria que ele fosse exibido ou armazenado, setando o valor NULL para representá-lo.
- **O valor não é aplicável.** Caso algum contato não tenha telefone, certamente não faz sentido querer armazenar essa informação.

A avaliação de comandos de consulta SQL com valores NULL merece uma atenção especial. Todos os fundamentos de computação baseiam-se na lógica booleana, o que implica que as expressões possuem sempre somente dois valores possíveis: verdadeiro (TRUE) ou falso (FALSE).

Como em SQL os atributos podem ter valor nulo, agora as expressões que envolvem os atributos podem resultar em valores verdadeiros (TRUE), falso (FALSE) ou em um terceiro valor, que é representado por NULL. Esse terceiro valor pode ser checado de um modo especial com os operadores SQL definidos, como IS e IS NOT. Ademais, todas as condições da cláusula WHERE de comandos SQL filtram as linhas baseando-se no valor verdadeiro (TRUE) das expressões. Nesse caso, linhas que sejam avaliadas pelas expressões da cláusula WHERE como falso (FALSE) ou como o valor representado por NULL simplesmente são descartadas (com exceção da operação de OUTER JOIN).

Comecemos com alguns exemplos ainda baseados no *schema* apresentado no tema anterior.

Exemplo 1: selecione todos os contatos que não possuem data de nascimento definida.

```
SELECT *
FROM contato
WHERE nascimento IS NULL;
```

Exemplo 2: situação oposta à do Exemplo 1. Selecionaremos todos os contatos que possuem uma data de nascimento definida.

```
SELECT *
FROM contato
WHERE nascimento IS NOT NULL;
```

CONSULTAS ANINHADAS (SUBQUERIES)

Algumas consultas em SQL são mais facilmente construídas se pudermos buscar primeiramente alguns valores das tabelas e utilizá-los posteriormente em nossa consulta. Essas consultas diferenciadas podem ser formuladas com certa conveniência por meio de consultas aninhadas (uma consulta dentro de outra) ou, como popularmente denominadas, de *subqueries*.

Exemplo 3: selecione todos os telefones de contatos com sobrenome = 'Machado'.

```
SELECT telefone
FROM telefone, contato
WHERE contato.id = telefone.contato_fk and sobrenome = 'Machado';
```

A consulta anteriormente apresentada foi criada utilizando-se um *join* normal. Agora, no exemplo a seguir, a reescreveremos utilizando uma *subquery*.

```
SELECT telefone
FROM telefone
WHERE contato_fk IN
(
    SELECT id
    FROM contato
    WHERE sobrenome = 'Machado'
);
```

Uma dúvida comum a muitos desenvolvedores está relacionada à frequência de uso de cada opção. Matematicamente, de acordo com o modelo relacional, não

há diferença entre as duas consultas: são equivalentes. Toda consulta que utiliza um *join* pode ser reescrita na forma de uma consulta aninhada (com *subqueries*).

Decidir entre uma forma e outra passa a ser uma questão de gosto, conveniência e legibilidade de código. Há alguns anos, poderia ser argumentado que uma forma seria mais rápida que outra ou vice-versa. Entretanto, devido à evolução dos interpretadores de SQL nos Sistemas Gerenciadores de Banco de Dados (SGDBS) modernos, essa diferença hoje é praticamente nula: todos os SGBDs modificam internamente as consultas fornecidas e automaticamente já escolhem o melhor plano de execução. Isso faz que boa parte das supostas “otimizações” que muitos DBAs realizam em consultas SQL tornem-se inócuas, pois o SGBD, na maioria das vezes, reescreverá as consultas para a melhor forma possível.

No exemplo anterior, note o uso da cláusula IN (<subquery>). A cláusula IN espera um conjunto de valores sendo retornado pela *subquery* dentro dos parênteses, que deve ser compatível com o atributo sendo comparado pela cláusula IN. Na hipótese de você ter certeza da sua *subquery* retornar um único valor em vez de retornar um conjunto de valores, pode substituir o IN pelo operador de igual ('='). Mas até nas situações de um único valor sendo retornado, o operador IN continua equivalente. É por esse motivo que muitos programadores acabam adotando a convenção de se utilizar o operador IN em todas as consultas que envolvem *subqueries*.

Considere nos exemplos o seguinte *schema* representado pela Figura 1:

funcionário			
id	nome	sobrenome	cargo

subordinado			
id	nome	sobrenome	superior_fk

Figura 1: Exemplo das tabelas “Funcionário” e “Subordinado” / Fonte: o autor.

O *schema* da Figura 1 pode ser construído da seguinte forma:

```
CREATE TABLE funcionario (
    id int primary key,
    nome varchar(30),
    sobrenome varchar(30),
    cargo varchar(30)
);
```

```
CREATE TABLE subordinado (
    id int primary key,
    nome varchar(30),
    sobrenome varchar(30),
    superior_fk int,
    FOREIGN KEY (superior_fk) REFERENCES funcionario(id)
);
```

Exemplo 4: selecione o nome e o sobrenome de todos os funcionários que possuem subordinados com o mesmo nome.

```
SELECT f.nome, f.sobrenome
FROM funcionario AS f, subordinado AS s
WHERE f.id = s.superior_fk AND f.nome = s.nome;
```

Reescrevendo o exemplo acima com uma *subquery*:

```
SELECT f.nome, f.sobrenome  
FROM funcionario AS f  
WHERE id IN(  
    SELECT superior_fk  
    FROM subordinado AS s  
    WHERE f.nome = s.nome  
) ;
```

Esse exemplo reescrito com uma *subquery* é um caso especial de consulta aninhada em SQL, pois, como você pode notar, a *subquery* utiliza atributos da consulta externa em sua cláusula WHERE. Chamamos esse caso especial de consultas **aninhadas correlacionadas**.

```
SELECT f.nome, f.sobrenome  
FROM funcionario AS f  
WHERE EXISTS (  
    SELECT *  
    FROM subordinado AS s  
    WHERE f.nome = s.nome  
) ;
```

Apresentamos a mesma consulta do exemplo reescrito anteriormente com um novo operador denominado de EXISTS. Esse operador retorna um resultado verdadeiro (TRUE), se a sua *subquery* retornar ao menos uma linha de resultado, e falso (FALSE), se o resultado for vazio.

Exemplo 5: selecione o nome e o sobrenome de todos os funcionários que não possuem subordinados.

```
SELECT f.nome, f.sobrenome  
FROM funcionario AS f  
WHERE NOT EXISTS (  
    SELECT *  
    FROM subordinado AS s  
    WHERE s.superior_fk = f.id  
) ;
```

A exemplo do operador EXISTS, o operador NOT EXISTS retorna o oposto do operador EXISTS, sendo falso (FALSE), se o resultado possuir ao menos uma linha, e verdadeiro (TRUE) se o resultado for vazio.

CONSULTAS UTILIZANDO JOINS

No tema anterior, nós vimos que o conceito de *join* permite que façamos consultas que utilizam duas ou mais tabelas, unidas por meio de uma ou mais condições que unem os elementos das duas ou mais tabelas. Em alguns casos, é mais fácil compreender as consultas se elas forem escritas na forma com *join* em vez de misturar as condições de *join* na cláusula WHERE. Voltemos a utilizar o *schema* definido no tema anterior em nossos exemplos a seguir.

Exemplo 6: selecione o nome de todos os contatos cujo telefone inicie com “44”.

```
SELECT nome  
FROM contato, telefone  
WHERE contato.id = telefone.contato_fk and telefone.telefone LIKE  
'44%' ;
```

Agora vejamos essa mesma consulta reescrita com um JOIN:

```
SELECT nome
FROM (contato JOIN telefone ON contato.id = telefone.contato_fk)
WHERE telefone.telefone LIKE '44%';
```

Nesse caso do exemplo reescrito com JOIN, a cláusula FROM possui uma *joined table* que contém todos os atributos de ambas as tabelas unidas pelo JOIN e pela condição do JOIN, que é o predicado após o ON. Na SQL, o tipo de JOIN padrão, quando simplesmente declarado pela cláusula JOIN, é o *inner join*, que descarta todas as tuplas que não possuam um valor correspondente na segunda tabela do JOIN. Os outros tipos de JOIN disponíveis são descritos na tabela a seguir:

TIPO DE JOIN	SEMÂNTICA
INNER JOIN	É o tipo de JOIN padrão. Somente tuplas que satisfaçam a condição do JOIN são selecionadas.
LEFT OUTER JOIN ou LEFT JOIN	Todas as tuplas da tabela do lado esquerdo do ON são selecionadas. Caso não haja uma tupla correspondente na tabela do lado direito do JOIN, os valores são preenchidos com NULL.
RIGHT OUTER JOIN ou RIGHT JOIN	É a condição inversa do LEFT JOIN. Todas as tuplas da tabela do lado direito do ON são selecionadas. Caso não haja uma tupla correspondente na tabela do lado esquerdo do JOIN, os valores são preenchidos com NULL.
FULL OUTER JOIN ou FULL JOIN	Todas as tuplas dos dois lados do JOIN são selecionadas. Caso não haja correspondência na condição do JOIN, o lado vazio é preenchido com NULL.

Tabela 1: Tipos de *join* / Fonte: o autor.

Exemplo 7: selecione todos os nomes de contatos que iniciem com a letra “A” e seus respectivos telefones. Se o contato não tiver um telefone, mostre somente o nome e NULL como o valor do telefone.

```
SELECT nome, telefone  
FROM contato LEFT JOIN telefone ON contato.id = telefone.contato_fk  
WHERE contato.nome LIKE 'A%';
```

É um caso típico de LEFT JOIN, em que você deseja listar todos os contatos, tendo eles telefone ou não.

Consultas com Funções de Agregação

Uma das grandes vantagens da SQL e dos bancos de dados relacionais, se comparados com outras alternativas não relacionais, são as suas funções de agregação. Essas funções permitem uma análise resumida das informações armazenadas nas tabelas. Funções de agregação populares da SQL incluem COUNT, SUM, MAX, MIN e AVG que executam as funções matemáticas respectivas de contagem, soma, valor máximo, valor mínimo e média aritmética.

Exemplo 8: selecione o peso mínimo e máximo de todos os contatos.

```
SELECT MAX(peso), MIN(peso)  
FROM contato;
```

As funções de agregação em SQL recebem como parâmetro o nome do atributo em que se deseja aplicá-las. Nesse exemplo, é o caso do atributo peso.

Exemplo 9: selecione o número total de contatos cujo peso > 80.

```
SELECT COUNT (*)
FROM contato
WHERE peso > 80;
```

Nesse uso da função COUNT, o asterisco (“*”) representa o número de linhas do resultado da consulta e é bastante utilizado para se determinar a quantidade de resultados retornados.

Exemplo 10: selecione a quantidade de pesos distintos de todos os contatos.

```
SELECT COUNT(DISTINCT peso)
FROM contato;
```

Nesse exemplo, contamos a quantidade de pesos distintos de nossos contatos. Caso a cláusula DISTINCT não fosse aplicada, contaríamos somente a quantidade de pesos dos contatos.

Funções de Agrupamento

Em muitas situações, desejamos aplicar as funções de agregação não em todos os itens das tuplas selecionadas, mas em determinados grupos de tuplas – separados dentro da tabela, baseados em um determinado valor. Para conseguir esse objetivo em SQL, utilizamos a cláusula GROUP BY. Ao utilizarmos o GROUP BY, separaremos as tuplas em grupos distintos em que todas as tuplas dentro de um determinado grupo possuem o mesmo valor avaliado pelas condições do GROUP BY.

Exemplo 11: selecione o sobrenome e a quantidade de contatos que possuem o mesmo sobrenome.

```
SELECT sobrenome, COUNT(*)  
FROM contato  
GROUP by sobrenome;
```

Na avaliação dessa consulta, todas as tuplas da tabela contato são divididas em grupos cujo sobrenome seja igual. Ao aplicarmos a função COUNT(*), em vez de contar todas as tuplas da tabela, ela conta somente as tuplas de cada grupo. O resultado é uma lista que contém os sobrenomes e as quantidades de contatos com cada sobrenome.

Exemplo 12: selecione o sobrenome e a quantidade de contatos que possuem o mesmo sobrenome, desde que haja pelo menos dois contatos com o mesmo sobrenome.

```
SELECT sobrenome, COUNT(*)  
FROM contato  
GROUP by sobrenome  
HAVING COUNT(*) > 1;
```

Em algumas situações, desejamos agrupar as tuplas em grupos, mas queremos selecionar apenas alguns desses grupos no resultado – e não todos. A cláusula que permite filtrar quais grupos serão exibidos no resultado é a HAVING. Somente grupos que satisfaçam a condição imposta pelo HAVING são selecionados no resultado.

É importante salientar a diferença entre as cláusulas WHERE e HAVING, pois aparentemente ambas filtram os resultados da consulta. A cláusula WHERE filtra as tuplas avaliadas primeiramente. Portanto, a cláusula WHERE é avaliada antes de qualquer função de agregação ou qualquer agrupamento ser avaliado. Já a cláusula HAVING é avaliada somente depois que os grupos já foram formados e serve, então, para filtrar esses grupos do resultado final.

COMANDOS DE ALTERAÇÃO DE SCHEMA

Nós definimos como *schema evolution* o processo de alterações da estrutura do *schema*. Normalmente, essas alterações de estrutura não são frequentes e são motivadas por alterações dos requisitos do negócio e, consequentemente, também da aplicação.

Os comandos em SQL que podem alterar as definições de *schema* são os comandos para adicionar, modificar e remover *schemas*, tabelas, atributos e restrições. Vamos abordá-los nos exemplos seguintes.

Exemplo 13: remova o *schema* agenda do banco de dados.

```
DROP SCHEMA agenda;
```

Esse comando remove o *schema* e, por consequência, todas as tabelas dentro do *schema*. Há certa controvérsia nesse comando. Alguns SGBDs removem todas as tabelas do *schema* ao remover o próprio *schema*. Outros só permitem a remoção do *schema* se ele não contiver nenhuma tabela, sendo necessário remover todas as tabelas antecipadamente.

Exemplo 14: remova a tabela telefone do *schema*.

```
DROP TABLE telefone;
```

Nesse exemplo, a tabela telefone seria removida do *schema*. É importante notar que a tabela sendo removida do *schema* pode conter dependências de integridade referencial e chave estrangeira em outras tabelas. Nesse caso, se alguma outra tabela depender da tabela removida, esse comando irá falhar devido à

restrição de integridade referencial do banco, sendo necessário primeiro remover a integridade referencial.

Exemplo 15: adicione uma coluna apelido na tabela contato contendo 15 caracteres.

```
ALTER TABLE contato ADD COLUMN apelido VARCHAR(15);
```

Esse exemplo adiciona uma coluna denominada “apelido” no final da definição da tabela contato com o tipo definido de VARCHAR(15). Quando um valor DEFAULT não é especificado no ALTER TABLE, todas as tuplas existentes na tabela recebem um valor NULL na coluna adicionada.

Exemplo 16: adicione uma coluna apelido na tabela contato contendo 15 caracteres e com valor padrão de “Senhor”:

```
ALTER TABLE contato ADD COLUMN apelido VARCHAR(15) DEFAULT 'Senhor';
```

O Exemplo 16 é a mesma situação do Exemplo 15, apenas definindo um valor padrão para a coluna que está sendo adicionada.

Exemplo 17: altere o tamanho da coluna apelido para 25 caracteres.

```
ALTER TABLE contato ALTER COLUMN apelido VARCHAR(25);
```

A definição da coluna apelido da tabela contato foi modificada e o seu tipo de dados agora define a capacidade de 25 caracteres.

Exemplo 18: remova a coluna apelido da tabela contato.

```
ALTER TABLE contato DROP COLUMN apelido;
```

A coluna apelido é removida da tabela contato desde que não haja nenhuma restrição de integridade referencial nessa coluna.

Exemplo 19: supondo que ainda não houvesse uma integridade referencial entre a tabela telefone e a tabela contato, adicione-a.

```
ALTER TABLE telefone ADD FOREIGN KEY (contato_fk) REFERENCES contato(id);
```

Esse comando adiciona a restrição de integridade referencial entre a tabela telefone e a tabela contato. Também é possível remover as restrições de integridade referencial por meio do comando ALTER TABLE <tabela> DROP FOREIGN KEY <nome>, mas, para tanto, é necessário saber o nome da restrição no banco de dados.



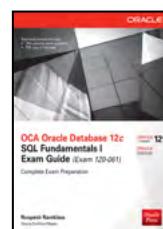
INDICAÇÃO DE LIVRO

Fundamentos I SQL (2008).

Autor: John Watson e Roopesh Ramklass

Editora: Alta Books

Sinopse: este livro é um guia de treinamento oficial da Oracle para o exame 1Z0-052, com mais exercícios, dicas e perguntas de teste. O leitor poderá aprender a criar um banco de dados Oracle, configurar uma rede Oracle, administrar a segurança do usuário, fazer backup e recuperação e outros.

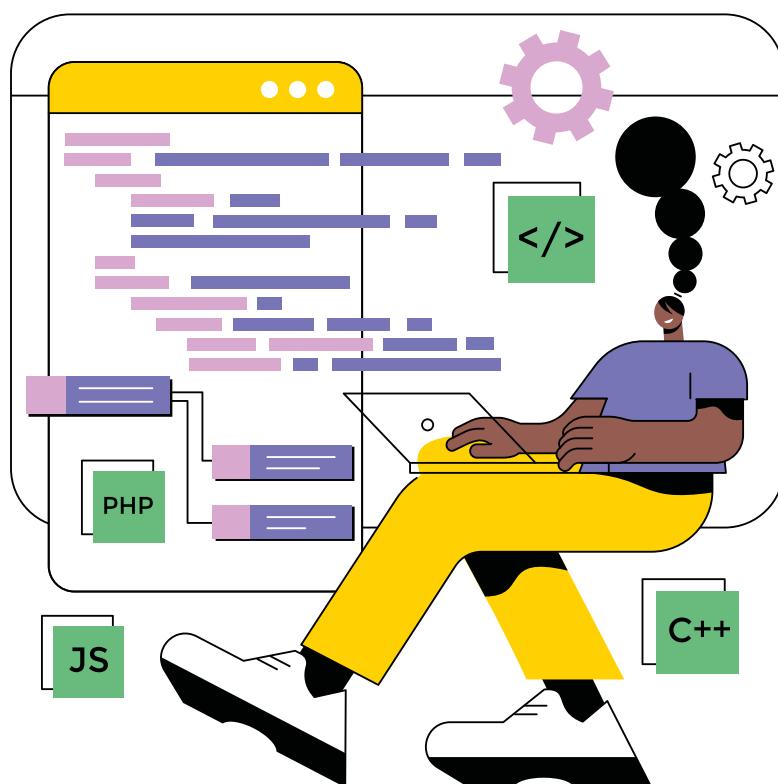


NOVOS DESAFIOS

Finalmente, chegamos ao fim do nosso tema. Nesse ponto, você provavelmente já terá assimilado conteúdo suficiente para poder desenvolver algumas aplicações utilizando sistemas de bancos de dados. Como descrevemos lá no início, é bastante provável que você se depare, em sua vida profissional, com consultas um tanto quanto complexas.

Lembre-se de que todo problema pode ser decomposto em partes menores de fácil solução. Utilize essa técnica e aplique os conceitos assimilados com os exemplos apresentados neste tema para resolvê-los. A teoria é importantíssima, mas a prática é uma atividade fundamental para que você possa converter toda essa teoria aprendida em resultados – tanto pessoais quanto profissionais.

A prática das atividades de autoestudo pode auxiliá-lo(a) na trabalhosa tarefa de assimilação dos conceitos apresentados nesta unidade. Analise-as com carinho e tenha um bom proveito.



AGORA É COM VOCÊ

Todas as atividades deste tema baseiam-se na figura abaixo.

beneficiário

id	nome	sobrenome	altura	plano_fk

dependente

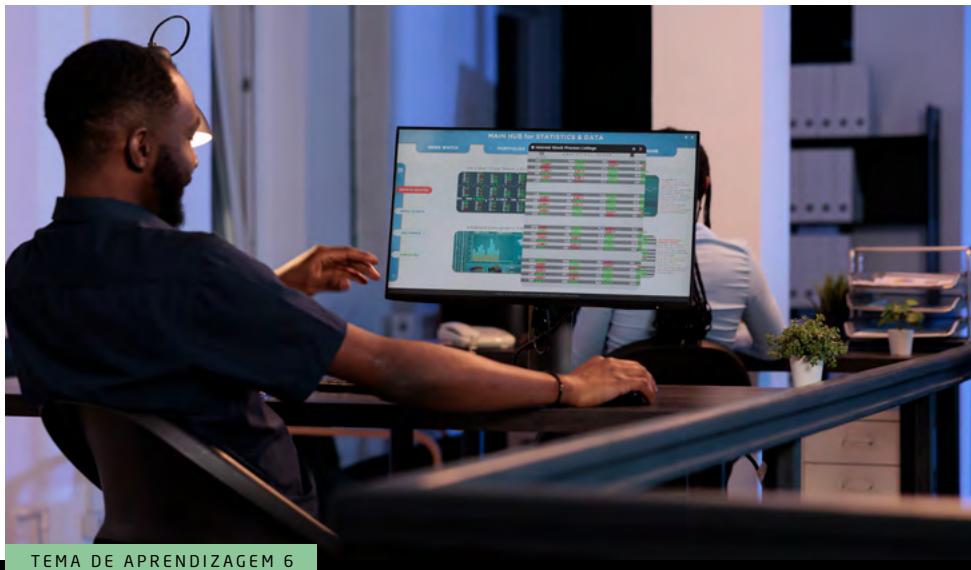
id	nome	sobrenome	beneficiário_fk

plano

id	nome	valor

1. Considere o exemplo de schema da figura apresentada. Crie os comandos SQL para definição e criação das tabelas.
2. Elabore consultas SQL para:
 - a) Selecione todos os beneficiários que possuem dependentes com o mesmo nome utilizando uma condição de JOIN.
 - b) Execute a mesma consulta anterior utilizando uma subquery.
 - c) Execute a mesma consulta anterior utilizando uma cláusula EXISTS.
 - d) Selecione o beneficiário que contém dependente que possui a maior altura entre todos.
3. Utilizando a cláusula de GROUP BY, elabore consultas para:
 - a) Agrupe os beneficiários por nome e selecione o sobrenome e a altura de cada um deles.
 - b) Selecione todos os beneficiários que contêm mais de um dependente com o mesmo sobrenome.
 - c) Selecione todos os planos que contêm mais de um beneficiário com altura > 1.75.

MEU ESPAÇO



TEMA DE APRENDIZAGEM 6

ESTUDO DE CASO

ESP. VICTOR DE MARQUIS PEDROSO

MINHAS METAS

- Demonstrar ao aluno a prática na criação de um banco de dados.
- Etapas de produção a seguir para o desenvolvimento de um banco de dados.

INICIE SUA JORNADA

Levando em consideração que já sabemos os conceitos básicos de banco de dados e englobamos alguns assuntos como Sistemas Gerenciadores de Banco de Dados (SGDBS), em que vimos os tipos existentes e como funcionam. Vimos também a teoria relacional, fazendo uma análise de como iniciar a criação de um banco de dados e seus respectivos relacionamentos, e aprendemos a respeito de *Structured Query Language* (SQL), tanto os tipos *Data Manipulation Language* (DML) quanto os *Data Definition Language* (DDL), sendo esses conceitos imprescindíveis para a implementação prática de bancos de dados relacionais.

Saber a respeito desses outros conceitos que englobam o banco de dados é muito importante, porém, para uma melhor aplicabilidade, devemos fazer com que essa teoria estudada seja concretizada e a melhor maneira de fazer isso é realizando as atividades de forma prática.

Neste tema, teremos como objetivo principal o estudo e a demonstração prática da produção de um banco de dados. Espero que você considere este tema como um auxílio ao processo de criação de um projeto de banco de dados, podendo realizar consultas constantes.

DESENVOLVENDO SEU POTENCIAL

DESCREVENDO O ESTUDO DE CASO

A empresa Top Uniformes deseja um sistema que gerencie as vendas da empresa. Nessas vendas, seria importante termos o controle dos clientes que compraram, um controle de produtos com seus respectivos preços e um controle de vendedores para que seja possível a distribuição da comissão.

Restrições e Premissas:

- Uma venda pode ter apenas um cliente.
- Uma venda pode ter apenas um vendedor.

- Uma venda pode ter vários itens, porém esses itens podem ter apenas um produto cada.
- Não teremos cadastros de empresas, pois não temos filiais.

Vamos analisar o estudo de caso.

O passo inicial que podemos tomar no estudo de caso é a análise dos substantivos. Vamos à análise:

A empresa TOP UNIFORMES deseja um sistema que gerencie as vendas da empresa, nestas vendas seria importante termos o controle dos clientes que compraram, um controle de produtos com seus respectivos preços e também um controle de vendedores para que seja possível a distribuição da comissão.

Após a análise, retiramos as possíveis entidades:

- Empresa.
- Vendas.
- Produtos.
- Clientes.
- Vendedores.

Importante: a entidade “Empresa” existe no texto do estudo de caso, porém, conforme a restrição nos determina, não será criada por não haver filiais.

CRIANDO AS ENTIDADES E OS RELACIONAMENTOS (DER)

Para facilitar o mapeamento do DER, é importante que ele seja feito em etapas, para não perdermos nenhuma informação e restrição imposta pelo projeto. De acordo com as restrições passadas no estudo de caso, abordaremos cada um dos casos.

1º Relacionamento:

- Uma venda pode ter apenas um cliente.



Figura 1: Exemplo de relacionamento em que uma venda pode ter apenas um cliente
Fonte: o autor.

Tabela: CLIENTES

CLI_CHAVE	CLI_NOMEFANTASIA	CLI_CIDADE	CLI_ESTADO
1	REDE CIDADE SUPERMERCADO	SÃO PAULO	SP
2	JOÃO DA SILVA	MARINGÁ	PR

Tabela: VENDA

VDA_CHAVE	VDA_DATAMOVTO	VDA_VRECEBIDO	VDA_VDESCONTO	VDA_VTOTAL	VDA_CODCLI_CLI
1	23/01/2015	100,00	0,00	100,00	2
2	15/02/2015	150,00	25,00	125,00	1

Figura 2: Exemplo de relacionamento em que uma venda pode ter apenas um vendedor demonstrado em formato de tabela / Fonte: o autor.

No exemplo anterior, temos a representação em forma de tabela do relacionamento entre as entidades “CLIENTE” e “VENDA”, em que a representação simboliza que a tabela “VENDA” receberá o código do cliente.

2º Relacionamento:

- Uma venda pode ter apenas um vendedor.



Figura 3: Exemplo de relacionamento em que uma venda pode ter apenas um vendedor.
Fonte: o autor.

Tabela: VENDEDOR

VEN_COD	VEN_NOME	VEN_FONE	VEN_CIDADE	VEN_UF
1	PEDRO DA SILVA	9999-3939	CAMPINAS	SP
2	PAULO GOMES	9993-2929	LONDRINA	PR

Tabela: VENDA

VDA_CHAVE	VDA_DATAMOVTO	VDA_VRECEBIDO	VDA_VDESCONTO	VDA_VTOTAL	VDA_CODVEN_VEN
1	23/01/2015	100,00	0,00	100,00	2
2	15/02/2015	150,00	25,00	125,00	1

Figura 4: Exemplo de relacionamento em que uma venda pode ter apenas um vendedor demonstrado em formato de tabelas / Fonte: o autor.

No exemplo anterior, temos a representação em forma de tabela do relacionamento entre as entidades “VENDEDOR” e “VENDA”, em que a representação simboliza que a tabela “VENDA” receberá o código do vendedor.

3º Relacionamento:

- Uma venda pode ter vários itens.



Figura 5: Exemplo de relacionamento em que uma venda pode ter apenas vários itens / Fonte: o autor.

No exemplo a seguir, temos a representação em forma de tabela do relacionamento entre as entidades “VENDA” e “VENDA_ITENS”, em que a representação simboliza que a tabela “VENDA_ITENS” receberá o código da venda e não há problemas em haver repetições, pois vale lembrar que a chave nessa tabela é composta (VDI_CHAVE_VDA e VDI_NSEQUEN) e o campo VDI_NSEQUEN deverá ser preenchido de maneira sequencial e, assim, não teremos o risco de haver tuplas (linhas) duplicadas.

Tabela: VENDA

VDA_CHAVE	VDA_DATAMOVTO	VDA_VRECEBIDO	VDA_DESCONTO	VDA_TOTAL
1	23/01/2015	1000,00	0,00	1000,00
2	15/02/2015	5000,00	200,00	4800,00

Tabela: VENDA_ITENS

VDI_CHAVE_VDA	VDI_NSEQUEN	VDI_QQUANTI	VDI_PREUNI	VDA_VTOTAL
1	1	5	100,00	500,00
1	2	10	10,00	100,00
1	3	400	10,00	400,00
2	1	5	1000,00	5000,00

Figura 6: Exemplo de relacionamento em que uma venda pode ter apenas vários itens demonstrado em formato de tabelas / Fonte: o autor.

4º Relacionamento:

- Um produto pode estar em vários itens.



Figura 7: Exemplo de relacionamento em que um produto pode estar em vários itens
Fonte: o autor.



Tabela: PRODUTOS

PROD_CHAVE	PROD_FAMILI	PROD_DESCRI	PROD_UNIDAD	PROD_VPRECO
1	ESCRITORIO	GRAMPEADOR	UN	3,00
2	LIMPEZA	SABÃO EM PÓ	UN	9,00

Tabela: VENDA_ITENS

VDI_CHAVE_VDA	VDI_NSEQUEN	VDI_CODPRO_PROD	VDI_QQUANTI	VDI_PREUNI	VDA_VTOTAL
1	1	1	5	3,00	15,00
2	1	2	5	9,00	45,00

Figura 8: Exemplo de relacionamento em que um produto pode estar em vários itens demonstrado em formato de tabelas / ,Fonte: o autor.

No exemplo anterior, temos a representação em forma de tabela do relacionamento entre as entidades “PRODUTOS” e “VENDA_ITENS”, em que a representação simboliza que a tabela “VENDA_ITENS” receberá o código do produto.



DER (Diagrama de Entidade e Relacionamento) - Completo

Para uma melhor visualização dos relacionamentos das entidades, utilizamos o Diagrama de Entidade e Relacionamento (DER). Veja, na sequência, a representação completa das entidades relativas ao nosso estudo de caso:

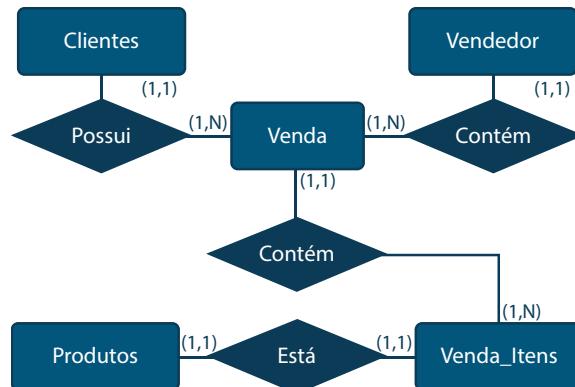


Figura 9: Diagrama de Entidade e Relacionamento (DER) / Fonte: o autor.

As Entidades e Seus Atributos

A seguir, do lado esquerdo, teremos a descrição das entidades e seus atributos de maneira individual, conforme o modelo lógico, e, do lado direito, temos a representação da entidade e seus atributos no modelo conceitual, conforme a notação de Peter Chen (1976).

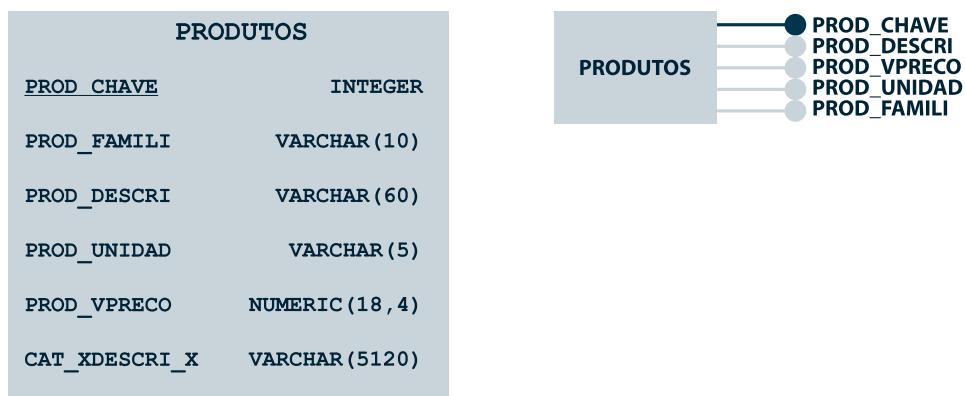


Figura 10: Exemplo da tabela “Produtos” e seus atributos / Fonte: o autor.

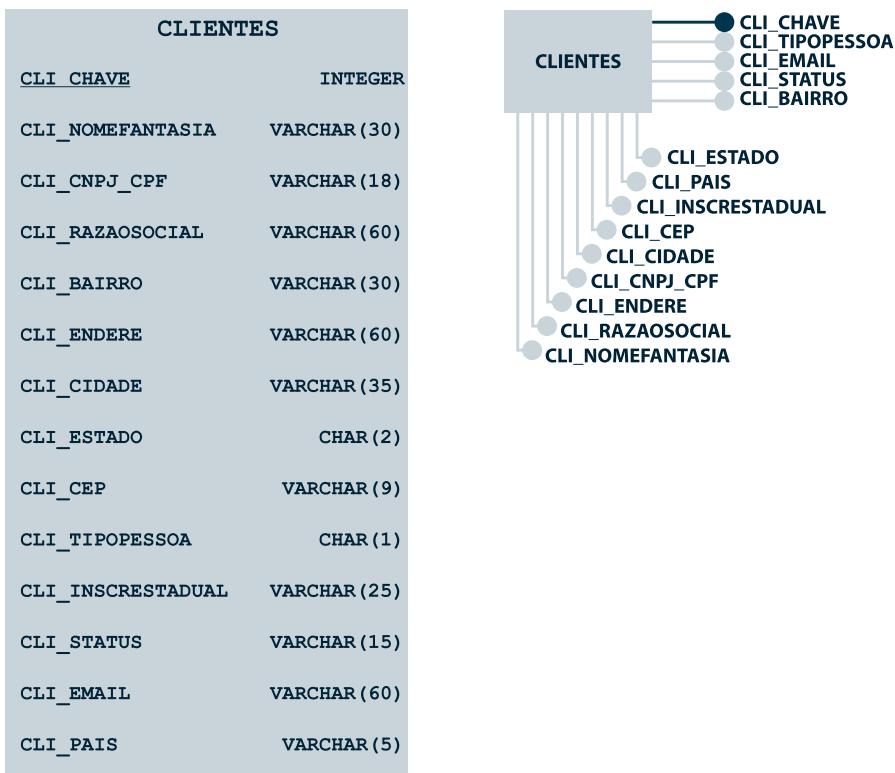


Figura 11: Exemplo da tabela “Clientes” e seus atributos / Fonte: o autor.

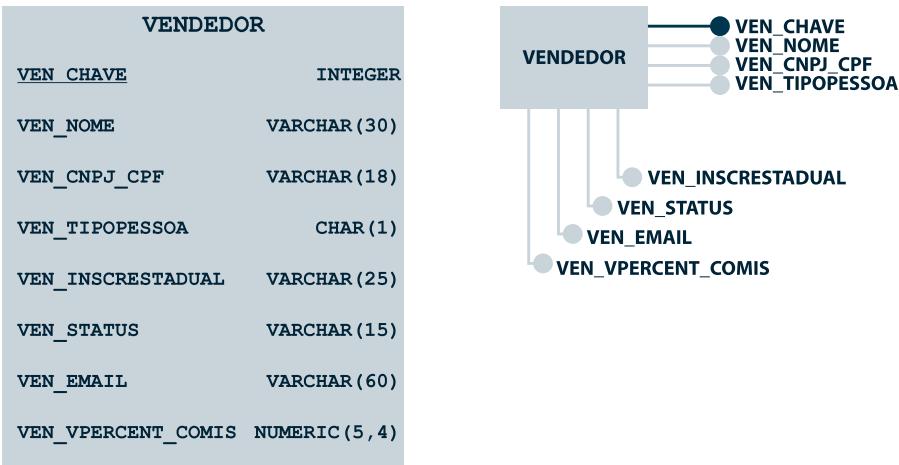


Figura 12: Exemplo da tabela “Vendedor” e seus atributos / Fonte: o autor.

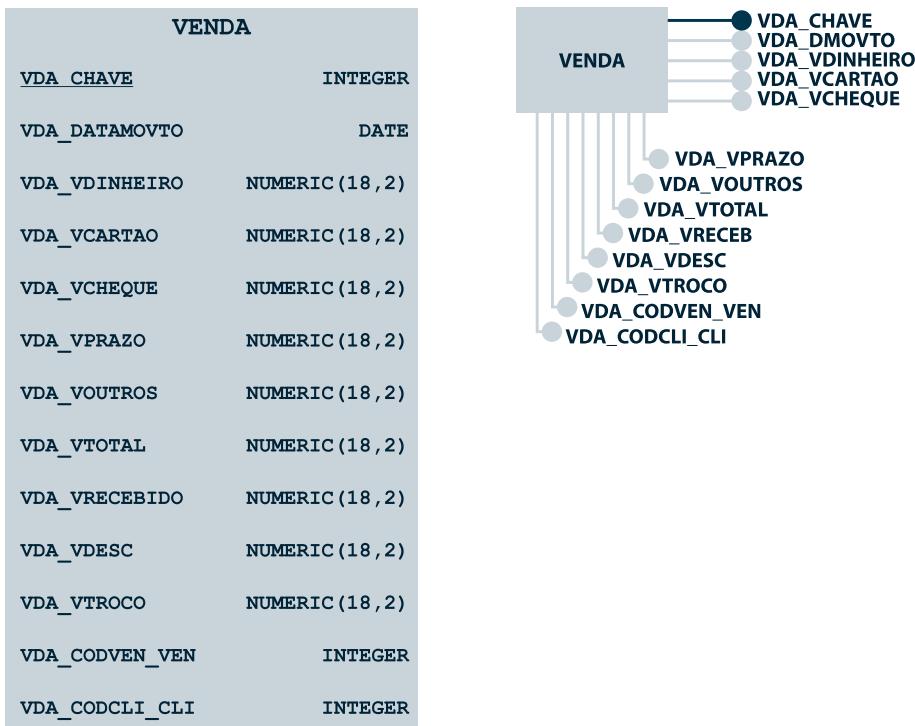


Figura 13: Exemplo da tabela “Venda” e seus atributos / Fonte: o autor.

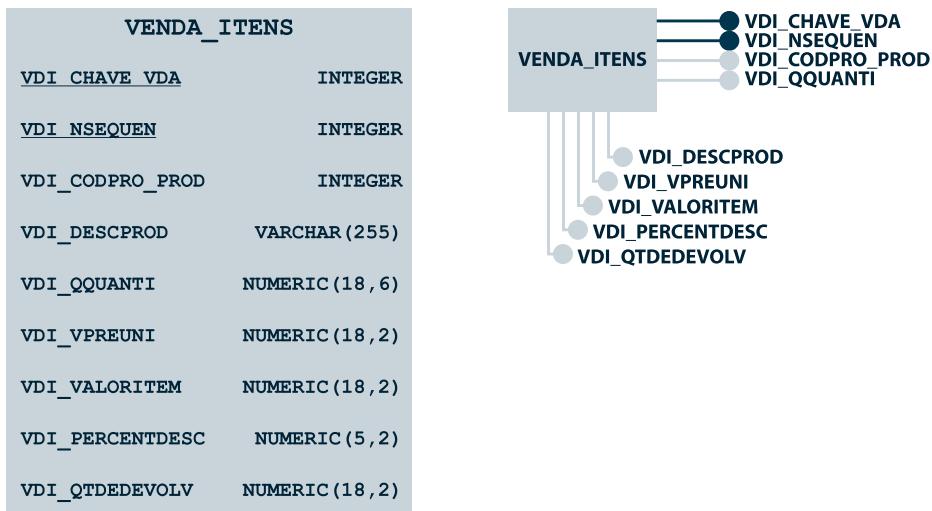


Figura 14: Exemplo da tabela “Venda_Itens” e seus atributos / Fonte: o autor.

TRABALHANDO COM SQL

Aqui, vamos trabalhar com a implementação das entidades criadas no estudo de caso, abordando tanto as metodologias do modelo *Data Definition Language* (DDL) e *Data Manipulation Language* (DML).

Trabalhando com *Data Definition Language* (DDL)

- Criando as tabelas

Para criar tabelas, utilizamos o comando CREATE TABLE. A seguir, demonstraremos cada tabela:

- PRODUTOS

```
CREATE TABLE PRODUTOS (
    PROD_CHAVE      INTEGER PRIMARY KEY,
    PROD_DESCRI     VARCHAR(60),
    PROD_VPRECO     NUMERIC(18,4),
    PROD_UNIDAD     VARCHAR(5),
    PROD_FAMILI     VARCHAR(10)
);
```

■ CLIENTES

```
CREATE TABLE CLIENTES (
    CLI_CHAVE           INTEGER PRIMARY KEY,
    CLI_TIPOPESSOA      CHAR(1),
    CLI_STATUS          VARCHAR(15),
    CLI_BAIRRO          VARCHAR(30),
    CLI_ESTADO          CHAR(2),
    CLI_PAIS            VARCHAR(6),
    CLI_INSCRESTADUAL  VARCHAR(25),
    CLI_CEP              VARCHAR(9),
    CLI_CIDADE          VARCHAR(35),
    CLI_CNPJ_CPF         VARCHAR(18),
    CLI_ENDERE          VARCHAR(60),
    CLI_RAZAOSOCIAL     VARCHAR(60),
    CLI_NOMEFANTASIA   VARCHAR(30),
    CLI_EMAIL            VARCHAR(60)
);
```

■ VENDEDOR

```
CREATE TABLE VENDEDOR (
    VEN_CHAVE           INTEGER PRIMARY KEY,
    VEN_NOME             VARCHAR(30),
    VEN_CNPJ_CPF         VARCHAR(18),
    VEN_TIPOPESSOA       CHAR(1),
    VEN_INSCRESTADUAL   VARCHAR(25),
    VEN_STATUS           VARCHAR(15),
    VEN_EMAIL            VARCHAR(60),
    VEN_VPERCENT_COMIS  NUMERIC(5,4)
);
```

■ VENDA

```
CREATE TABLE VENDA (
    VDA_CHAVE      INTEGER PRIMARY KEY,
    VDA_DMOVTO     DATE,
    VDA_VDINHEIRO  NUMERIC(18,2),
    VDA_VCARTAO    NUMERIC(18,2),
    VDA_VCHEQUE    NUMERIC(18,2),
    VDA_VPRAZO     NUMERIC(18,2),
    VDA_VOUTROS    NUMERIC(18,2),
    VDA_VTOTAL     NUMERIC(18,2),
    VDA_VRECEB     NUMERIC(18,2),
    VDA_VTROCO     NUMERIC(18,2),
    VDA_CODCLI_CLI INTEGER,
    VDA_CODVEN_VEN INTEGER,
    VDA_VDESC      NUMERIC(18,2),
    FOREIGN KEY (VDA_CODCLI_CLI) REFERENCES CLIENTES(CLIENTE),
    FOREIGN KEY (VDA_CODVEN_VEN) REFERENCES VENDEDOR(VENDEDOR));

```

■ VENDA_ITENS

```
CREATE TABLE VENDA_ITENS (
    VDI_CHAVE_VDA      INTEGER,
    VDI_NSEQUEN        INTEGER,
    VDI_CODPRO_PROD    INTEGER,
    VDI_QQUANTI        NUMERIC(18,2),
    VDI_VPREUNI        NUMERIC(18,2),
    VDI_VALORITEM      NUMERIC(18,2),
    VDI_PERCENTDESC   NUMERIC(5,2),
    VDI_QTDEDEVOLV    NUMERIC(18,2),
    VDI_DESCPROD       VARCHAR(255),
    PRIMARY KEY(VDI_CHAVE_VDA, VDI_NSEQUEN),
    FOREIGN KEY (VDI_CHAVE_VDA) REFERENCES VENDA (VDA_CHAVE),
    FOREIGN KEY (VDI_CODPRO_PROD) REFERENCES PRODUTOS (PROD_CHAVE));

```

Inserindo Atributos nas Tabelas

Para inserir atributos nas tabelas, utilizamos o comando ALTER TABLE ADD. A seguir, incluiremos alguns atributos para cada tabela:

- PRODUTOS

```
ALTER TABLE PRODUTOS ADD PROD_DTVENCIMENTO DATE;  
ALTER TABLE PRODUTOS ADD PROD_DESCRCOMPLETA VARCHAR(255);  
ALTER TABLE PRODUTOS ADD PROD_PRECOVENDA NUMERIC(18,4);
```

- CLIENTES

```
ALTER TABLE CLIENTES ADD CLI_FONECONTATO VARCHAR(14);  
ALTER TABLE CLIENTES ADD CLI_CONTATO VARCHAR(30);  
ALTER TABLE CLIENTES ADD CLI_COMPLEMENTOEND VARCHAR(30);
```

- VENDEDOR

```
ALTER TABLE VENDEDOR ADD VEN_FONECONTATO VARCHAR(14);  
ALTER TABLE VENDEDOR ADD VEN_ENDERECHO VARCHAR(60);  
ALTER TABLE VENDEDOR ADD VEN_ESTADO VARCHAR(2);
```

- VENDA

```
ALTER TABLE VENDA ADD VDA_STATUS CHAR(1);
ALTER TABLE VENDA ADD VDA_CFOP VARCHAR(7);
ALTER TABLE VENDA ADD VDA_DTENVIO DATE;
```

- VENDA_ITENS

```
ALTER TABLE VENDA_ITENS ADD VDI_PERCENTICMS NUMERIC(5,4);
ALTER TABLE VENDA_ITENS ADD VDI_TOTALICMS NUMERIC(18,4);
ALTER TABLE VENDA_ITENS ADD VDI_VALORIPI NUMERIC(18,4);
```

Deletando os Atributos nas Tabelas

Para deletar atributos nas tabelas, utilizamos o comando ALTER TABLE DROP. A seguir, excluiremos alguns atributos para cada tabela:

- PRODUTOS

```
ALTER TABLE PRODUTOS DROP PROD_FAMILY;
ALTER TABLE PRODUTOS DROP PROD_UNIDAD;
ALTER TABLE PRODUTOS DROP PROD_VPRECO;
```

- CLIENTES

```
ALTER TABLE CLIENTES DROP CLI_PAIS;  
ALTER TABLE CLIENTES DROP CLI_BAIRRO;  
ALTER TABLE CLIENTES DROP CLI_STATUS;
```

- VENDEDOR

```
ALTER TABLE VENDEDOR DROP VEN_EMAIL;  
ALTER TABLE VENDEDOR DROP VEN_INSCRESTADUAL;  
ALTER TABLE VENDEDOR DROP VEN_STATUS;
```

- VENDA

```
ALTER TABLE VENDA DROP VDA_VTROCO;  
ALTER TABLE VENDA DROP VDA_VPRAZO;  
ALTER TABLE VENDA DROP VDA_VOUTROS;
```

- VENDA_ITENS:

```
ALTER TABLE VENDA_ITENS DROP VDI_QQUANTI;  
ALTER TABLE VENDA_ITENS DROP VDI_VPREUNI;  
ALTER TABLE VENDA_ITENS DROP VDI_VALORITEM;
```

Deletando as Tabelas

Para deletar as tabelas, utilizamos o comando `DROP TABLE`. A seguir, excluiremos as tabelas do sistema:

```
DROP TABLE PRODUTOS;
DROP TABLE CLIENTES;
DROP TABLE VENDEDOR;
DROP TABLE VENDA_ITENS;
DROP TABLE VENDA;
```

Trabalhando com *Data Manipulation Language (DML)*

Agora, trabalharemos com os comandos de manipulação de dados. Esses comandos nos ajudam a implementar e a trabalhar com o conteúdo das tabelas e, para isso, utilizaremos os comandos de inserção, alteração, seleção e deleção.

Inserindo novos registros nas tabelas

Para inserir registros em uma tabela, utilizamos o comando `INSERT`. A seguir, demonstraremos em cada tabela como devemos proceder:

- **PRODUTOS**

```
/* Inserindo Registros na tabela Produtos*/

INSERT INTO PRODUTOS (PROD_CHAVE, PROD_DESCRI, PROD_VPRECO, PROD_
UNIDAD, PROD_FAMILI) VALUES (1, 'NOBREAK', 200, 'UN', 'ESCRITÓRIO');

INSERT INTO PRODUTOS (PROD_CHAVE, PROD_DESCRI, PROD_VPRECO, PROD_
UNIDAD, PROD_FAMILI) VALUES (2, 'NOTEBOOK', 2500, 'UN', 'ESCRITÓRIO');

INSERT INTO PRODUTOS (PROD_CHAVE, PROD_DESCRI, PROD_VPRECO, PROD_
UNIDAD, PROD_FAMILI) VALUES (3, 'SABÃO EM PÓ', 9.00, 'UN', 'LIMPEZA');
```

■ CLIENTES

```
/* Inserindo Registros na tabela Clientes*/

INSERT INTO CLIENTES (CLI_CHAVE, CLI_TIPOPESSOA, CLI_STATUS, CLI_BAIRRO, CLI_ESTADO, CLI_PAIS, CLI_INSCRESTADUAL, CLI_CEP, CLI_CIDADE, CLI_CNPJ_CPF, CLI_ENDERE, CLI_RAZAOSOCIAL, CLI_NOMEFANTASIA, CLI_EMAIL)
VALUES (1, 'F', 'ATIVO', 'JARDIM COLINA', 'SP', 'BRASIL', 'ISENTO', '67009-490', 'CAMPINAS', '090.345.729-49', 'RUA JOÃO DE BARRO 946', 'RONALDO DA SILVA', 'RONALDO DA SILVA', 'ronaldoteste@hotmail.com');

INSERT INTO CLIENTES (CLI_CHAVE, CLI_TIPOPESSOA, CLI_STATUS, CLI_BAIRRO, CLI_ESTADO, CLI_PAIS, CLI_INSCRESTADUAL, CLI_CEP, CLI_CIDADE, CLI_CNPJ_CPF, CLI_ENDERE, CLI_RAZAOSOCIAL, CLI_NOMEFANTASIA, CLI_EMAIL)
VALUES (2, 'J', 'ATIVO', 'ZONA 1', 'PR', 'BRASIL', '123.123.234', '87009-678', 'MARINGÁ', '73.486.451/0001-54', 'AV. BRASIL 1001', 'MERCADO REDE CIDADE LTDA', 'MERCADO REDE CIDADE', 'adm@redecidade.com.br');
```

■ VENDEDOR

```
/* Inserindo Registros na tabela Vendedor*/

INSERT INTO VENDEDOR (VEN_CHAVE, VEN_NOME, VEN_CNPJ_CPF, VEN_TIPOPESSOA, VEN_INSCRESTADUAL, VEN_STATUS, VEN_EMAIL, VEN_VPERCENT_COMIS) VALUES (1, 'MARIA DAS GRAÇAS', '302.027.591-13', 'F', 'ISENTO', 'ATIVO', 'vendas@hotmail.com', 1.5);

INSERT INTO VENDEDOR (VEN_CHAVE, VEN_NOME, VEN_CNPJ_CPF, VEN_TIPOPESSOA, VEN_INSCRESTADUAL, VEN_STATUS, VEN_EMAIL, VEN_VPERCENT_COMIS) VALUES (2, 'MALU REPRESENTAÇÃO', '48.825.337/0001-64', 'J', 'ISENTO', 'ATIVO', 'comercial@malurepresenta.com.br', 2.0);
```

■ VENDA

```
/* Inserindo Registros na tabela Venda*/
INSERT INTO VENDA (VDA_CHAVE, VDA_DMOVTO, VDA_VDINHEIRO, VDA_VCARTAO,
VDA_VCHEQUE, VDA_VPRAZO, VDA_VOUTROS, VDA_VTOTAL, VDA_VRECEB, VDA_VTROCO,
VDA_CODCLI_CLI, VDA_CODVEN_VEN, VDA_VDESC) VALUES (1, '23/01/2015', 0,
0, 0, 0, 10900, 10900, 0, 1, 1, 0);
INSERT INTO VENDA (VDA_CHAVE, VDA_DMOVTO, VDA_VDINHEIRO, VDA_VCARTAO,
VDA_VCHEQUE, VDA_VPRAZO, VDA_VOUTROS, VDA_VTOTAL, VDA_VRECEB, VDA_VTROCO,
VDA_CODCLI_CLI, VDA_CODVEN_VEN, VDA_VDESC) VALUES (2, '15/02/2015', 0,
0, 0, 0, 4800, 5000, 0, 2, 2, 200);
```

■ VENDA_ITENS

```
/* Inserindo Registros na tabela Venda_Itens*/
INSERT INTO VENDA_ITENS (VDI_CHAVE_VDA, VDI_NSEQUEN, VDI_CODPRO_PROD,
VDI_QQUANTI, VDI_VPREUNI, VDI_VALORITEM, VDI_PERCENTDESC, VDI_QTDEDEVOLV,
VDI_DESCPROD) VALUES (1, 1, 1, 5, 200, 1000, 0, 0, 'NOBREAK');

INSERT INTO VENDA_ITENS (VDI_CHAVE_VDA, VDI_NSEQUEN, VDI_CODPRO_PROD,
VDI_QQUANTI, VDI_VPREUNI, VDI_VALORITEM, VDI_PERCENTDESC, VDI_QTDEDEVOLV,
VDI_DESCPROD) VALUES (1, 2, 1, 9, 1000, 9000, 0, 0, 'NOTEBOOK');

INSERT INTO VENDA_ITENS (VDI_CHAVE_VDA, VDI_NSEQUEN, VDI_CODPRO_PROD,
VDI_QQUANTI, VDI_VPREUNI, VDI_VALORITEM, VDI_PERCENTDESC, VDI_QTDEDEVOLV,
VDI_DESCPROD) VALUES (1, 3, 1, 100, 9.00, 900, 0, 0, 'SABAO EM PÓ');

INSERT INTO VENDA_ITENS (VDI_CHAVE_VDA, VDI_NSEQUEN, VDI_CODPRO_PROD,
VDI_QQUANTI, VDI_VPREUNI, VDI_VALORITEM, VDI_PERCENTDESC, VDI_QTDEDEVOLV,
VDI_DESCPROD) VALUES (2, 1, 1, 5, 1000, 5000, 0, 0, 'NOTEBOOK');
```

Alterando Registros das Tabelas

Para realizarmos alterações nos registros em uma tabela, utilizamos o comando UPDATE. Vale lembrar que devemos ter uma certa atenção com esse comando, pois ele normalmente vem acompanhado da cláusula WHERE, o que causa uma restrição/limite nas alterações. Por isso, caso você omita o comando WHERE dentro do comando UPDATE, ele será executado em toda a tabela e, caso essa não seja a pretensão, o dano pode ser grande, portanto, atenção ao executar esse comando. A seguir, demonstrarei, em algumas tabelas, como devemos proceder para alterarmos algum registro:

- PRODUTOS

```
/* Neste comando iremos modificar a descrição do produto cujo código
chave é igual a 1*/
UPDATE PRODUTOS
SET PROD_DESCRI = 'NOBREAK 700VA'
WHERE PROD_CHAVE = 1;

/* Neste comando iremos modificar a família do produto cujo código
chave é igual a 1*/
UPDATE PRODUTOS
SET PROD_FAMIL = 'CONSUMO'
WHERE PROD_CHAVE = 1;
```

■ CLIENTES

```
/* Neste comando iremos modificar o Status de TODOS os Clientes para INATIVO*/  
UPDATE CLIENTES  
SET CLI_STATUS = 'INATIVO';  
  
/* Neste comando iremos modificar o Bairro do cliente 1 para Jardim Paulista III*/  
UPDATE CLIENTES  
SET CLI_BAIRRO = 'JARDIM PAULISTA III'  
WHERE CLI_CHAVE = 1;
```

■ VENDEDOR

```
/* Neste comando iremos modificar o nome do vendedor 1 para Maria das Graças da Silva*/  
UPDATE VENDEDOR  
SET VEN_NOME = 'MARIA DAS GRAÇAS DA SILVA'  
WHERE VEN_CHAVE = 1;  
  
/* Neste comando iremos modificar o percentual de comissão do vendedor 1 para 2.0*/  
UPDATE VENDEDOR  
SET VEN_VPERCENT_COMIS = 2.0  
WHERE VEN_CHAVE = 1;
```

Deletando Registros das Tabelas

Em um banco de dados, podemos excluir os registros em uma tabela utilizando o comando DELETE. É importante lembrar que a exclusão só é possível se não há nenhuma dependência desse registro em outra tabela. Vamos a alguns exemplos:

■ PRODUTOS

```
/* Com o comando abaixo, iremos apagar todos os registros da Tabela Produtos*/
DELETE FROM PRODUTOS;

/*Neste comando será deletado o produto cujo código chave é igual a 1 */
DELETE FROM PRODUTOS
WHERE PROD_CHAVE = 1;
```

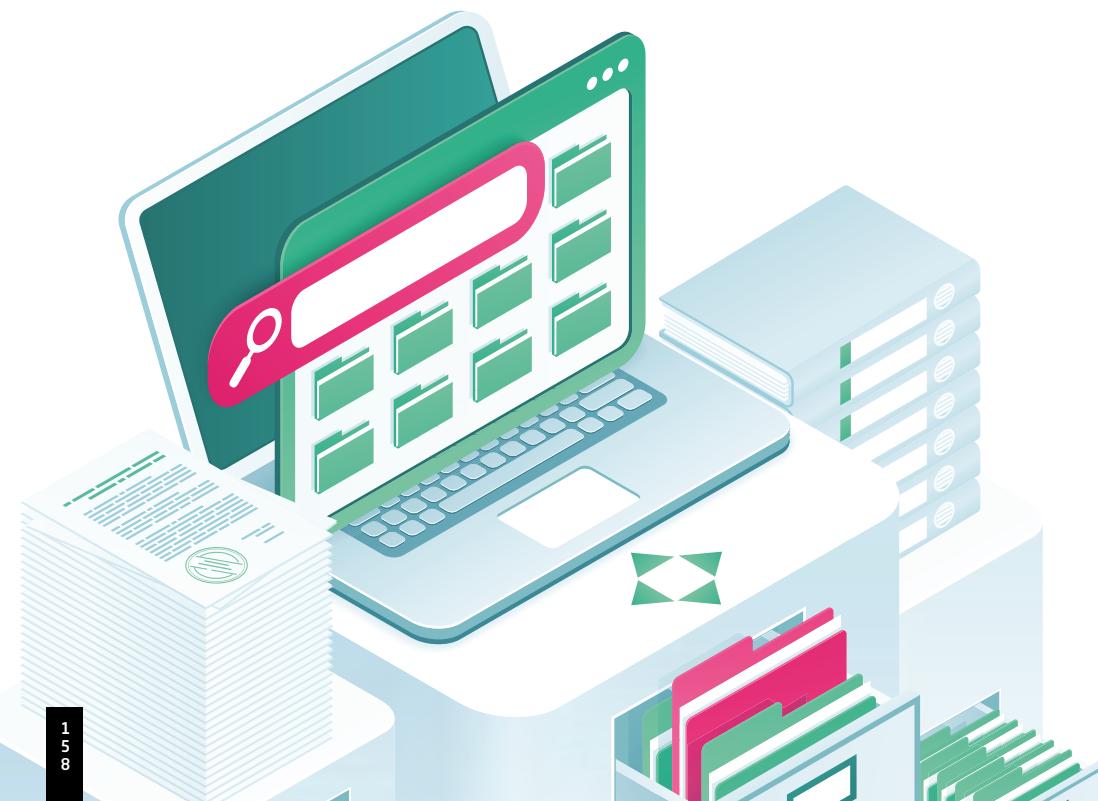
■ CLIENTES

```
/*Neste comando será deletado o cliente cujo código chave deste cliente é igual a 1*/
DELETE FROM CLIENTES
WHERE CLI_CHAVE = 1;
```

- VENDEDOR

```
/*Neste comando será deletado o Vendedor cujo código chave deste  
Vendedor é igual a 1*/  
  
DELETE FROM VENDEDOR  
  
WHERE VEN_CHAVE = 1;
```

É importante saber que os comandos UPDATE e DELETE devem ser utilizados de maneira cuidadosa, pois alguns dados, depois de inseridos no sistema, não podem ser modificados e/ou excluídos. Um exemplo claro do nosso dia a dia é a Nota Fiscal Eletrônica (NF-e), em que, após o envio dessa nota à Secretaria da Fazenda (SEFAZ), ela não pode ser modificada. Caso haja modificação e a empresa sofra uma auditoria, ela pode ser acusada de fraude e responder gravemente por essa modificação. Portanto, tenha sempre em mente que é muito grande a nossa responsabilidade com os dados inseridos em um banco de dados.



Selecionando os Registros de Tabelas

Demonstraremos agora o comando SELECT, comando esse que nos ajuda no momento da busca de dados no banco de dados de maneira organizada. Seguem alguns exemplos:

■ CLIENTES

```
/*Este comando seleciona todos os registros da tabela juntamente
com todos os seus atributos */

SELECT * FROM CLIENTES;

/*Seleciona o Cliente cuja chave é igual a 1, juntamente com todos
os seus campos por conta do * (asterisco) */

SELECT *
FROM CLIENTES
WHERE CLI_CHAVE = 1;

/*Seleciona o Cliente cujo Nome fantasia inicia com M, trazendo
os respectivos atributos específicos conforme descrito abaixo*/

SELECT
CLI_CHAVE,
CLI_TIPOPESSOA,
CLI_STATUS,
CLI_NOMEFANTASIA,
CLI_ESTADO,
CLI_PAIS
FROM CLIENTES
WHERE CLI_NOMEFANTASIA LIKE 'M%';
```

■ VENDAS

```

/* Selecionando todas as vendas cuja Data da Venda seja maior que
01/01/2015 */

SELECT *
FROM VENDA
WHERE VDA_DMOVTO > '01/01/2015';

/* Selecionando a data da Venda cujo valor total seja maior que
6000.00 */

SELECT VDA_DMOVTO
FROM VENDA
WHERE VDA_VTOTAL > 6000.00;

/* Selecionando o valor da maior venda */

SELECT MAX(VDA_VTOTAL)
FROM VENDA;

```

■ VENDA E VENDAS_ITENS

```

/* Na consulta a seguir, temos a seleção de campos da tabela
Venda e Venda_itens. neste caso utilizamos o JOIN para que independente se temos itens correspondentes ou não a aquela venda a mesma
seja retornada. */

SELECT
VENDA.VDA_CHAVE AS "Cód. Venda",
VENDA.VDA_DMOVTO AS "Dt. Movimento",
VENDA.VDA_VTOTAL AS "Total da Venda",
VENDA_ITENS.VDI_CODPRO_PROD AS "Código do Produto",
VENDA_ITENS.VDI_QQUANTI AS "Quantidade",
VENDA_ITENS.VDI_VPREUNI AS "Preço Unitário",
VENDA_ITENS.VDI_VALORITEM AS "Valor Item"
FROM VENDA INNER JOIN VENDA_ITENS ON VENDA.VDA_CHAVE = VENDA_ITENS.VDI_CHAVE_VDA;

```



INDICAÇÃO DE LIVRO

Sistemas de Banco de Dados.

Editora: Pearson

Autor: Elmasri e Navathe

Sinopse: disponibilizado na Biblioteca Pearson, este livro reúne teoria e exemplos práticos com as mais modernas tecnologias. Ele introduz os conceitos fundamentais necessários para projetar, usar e implementar os sistemas de banco de dados e suas aplicações. De fácil compreensão, o texto aborda com profundidade a modelagem e o projeto de banco de dados, suas linguagens e as funcionalidades dos sistemas de gerenciamento de banco de dados e as técnicas de implementação desses sistemas, além de data mining e sistemas de banco de dados. Destina-se a estudantes de graduação, pós-graduação ou a usuários familiarizados com programação e conceitos de estruturação de dados e organização básica de computadores.



NOVOS DESAFIOS

No estudo de banco de dados, nada melhor que a prática aliada à teoria. Neste tema, tivemos como objetivo demonstrar um passo a passo de como é a criação de um banco de dados, partindo desde o momento em que as características são passadas pelo cliente (análise dos requisitos) até o momento da criação prática utilizando a linguagem *Structured Query Language* (SQL) ou linguagem de consulta estruturada.

Ainda aqui, demonstramos alguns comandos que nos ajudam tanto na criação como na manutenção de um banco de dados. Tenha este tema como base de consulta em seus estudos diários, assim poderá não apenas saber os comandos ensinados, mas também aplicá-los de maneira correta sempre que necessário.

Contudo, vale enfatizar que o aprendizado de banco de dados e SQL demanda muita leitura e muita prática/treino. Nossa sugestão é que você tente sempre fazer os exercícios e, em seguida, possa revisá-los para que a fixação do conteúdo seja ainda melhor.

AGORA É COM VOCÊ

- O comando DROP TABLE é utilizado para excluir uma tabela. Partindo desta afirmação analise a instrução SQL abaixo e responda assinalando a alternativa correta:

```
DROP TABLE PRODUTOS WHERE COD_PRODUTO = 1;
```

- a) O comando está correto pois o mesmo irá apagar a tabela PRODUTOS.
- b) O comando está incorreto pois o comando utilizado para excluir tabelas é o comando NOT IN.
- c) O comando está incorreto pois o WHERE não pode ser utilizado junto com o comando DROP.
- d) Este comando está correto pois o mesmo irá apagar apenas o produto cujo Código é igual a 1.

YANAGA, Edson; PEDROSO, Victor de Marqui. **Banco de dados**, Maringá-Pr.: UniCesumar, 2016. Unidade V p 143-149.

- O comando UPDATE modifica os valores de uma ou mais tuplas (linhas) das tabelas selecionadas. Nesse contexto, analise a instrução SQL a seguir e assinale a alternativa correta:

```
UPDATE FUNCIONARIOS  
SET FUN_DT ADMISSAO = '20/10/2015'  
WHERE FUN_MATRICULA = 100;
```

- a) O resultado deste comando será a alteração do Nome do Funcionário cuja matrícula é igual a 100.
- b) O resultado deste comando será a alteração da data de admissão de todos funcionários para 20/10/2015.
- c) O resultado deste comando será a alteração da matrícula para 100 dos funcionários que foram admitidos no dia 20/10/2015.
- d) O resultado deste comando será a alteração da Data de Admissão para 20/10/2015 do funcionário cuja matricula é igual a 100.

YANAGA, Edson; PEDROSO, Victor de Marqui. **Banco de dados**, Maringá-Pr.: UniCesumar, 2016. Unidade V p 149-155.

AGORA É COM VOCÊ

3. Para nos ajudar a entender o motivo pelo qual uma entidade será criada, podemos utilizar do recurso da Descrição Textual Narrativa. Baseando-se nesse conceito, leia a frase a seguir: "O gerente do departamento de compras recebe e aprova vários pedidos ao dia".

Ao analisar a frase podemos identificar quais entidades?

- a) Funcionário, Pedidos e Dia.
- b) Gestor, Aprovador e Pedidos.
- c) Funcionário, Recebedor e Compras.
- d) Funcionário, Departamento e Pedidos.

YANAGI, Edson; PEDROSO, Victor de Marqui. **Banco de dados**, Maringá-Pr.: UniCesumar, 2016. Unidade V p 135-143.

MEU ESPAÇO



unidade





TEMA DE APRENDIZAGEM 7

CRIANDO UM BANCO DE DADOS

ESP. JEFERSON KAISER

MINHAS METAS

- Conhecer como se cria um banco de dados.
- Compreender a criação e a alteração de tabelas na prática.
- Verificar quais tipos de dados utilizar nos campos das tabelas.
- Conhecer chave primária na prática.
- Inserir dados nas tabelas.

INICIE SUA JORNADA

Olá, caro(a) aluno(a)! Neste tema, abordaremos a criação de um banco de dados no MySQL, ilustrando o passo a passo para a criação de um banco de dados de fato, além da criação, alteração e remoção das tabelas.

Iremos também popular as tabelas criadas. Nossa intenção é fazer com que você, acadêmico(a), verifique na prática a manipulação, indo desde a criação de um *schema* até popular dados nas tabelas. Alertamos que, nesse momento, não é necessária a preocupação com a consulta de dados. Vamos praticar, pois com a prática, você entenderá melhor todos os conceitos.

Logo quando falamos em banco de dados, a primeira coisa que nos vêm à cabeça é a linguagem *Structured Query Language* (SQL). Essa linguagem é muito madura e instável no mercado de banco de dados. Fique muito atento aos detalhes dos exemplos de comandos trabalhados aqui, pois um pequeno detalhe fará com que seu comando não seja executado corretamente na ferramenta que utilizaremos. Assim, você será impossibilitado de concluir todos os comandos.

Nossa ideia para esse momento é que se você executar todos os passos, será criado um cadastro de pessoa completo, contendo dados para que seja possível você colocar em prática todos os assuntos estudados.

Ótimo estudo!

DESENVOLVA SEU POTENCIAL

SCHEMA

Antes do SQL-92 não existia o conceito de *schema*, em que todas as tabelas e demais arquivos do banco de dados eram criados dentro de um mesmo ambiente, não existindo, assim, um agrupamento de tabelas. Um *schema* é representado por uma coleção de vários objetos de um ou mais usuários de banco de dados, por exemplo: tabelas, sequências, índices etc. São associados a um banco de dados em razão de vários esquemas para um banco de dados, facilitando a administração dos objetos e dos dados.

O comando a seguir é responsável pela criação de um *schema*:

```
CREATE SCHEMA UniCesumar;
```

Assim, universidade é o nome do *schema* a ser criado pelo comando. Uma das principais vantagens da utilização de *schemas* em banco de dados são as permissões que se pode atribuir e revogar ao usuário, pois podemos, dessa maneira, ter vários *schemas* e vários usuários em um banco de dados. Entretanto, determinados usuários têm acesso a somente um *schema* no banco de dados.

Banco de Dados MySQL

O MySQL é um Sistema de Gerenciamento de Banco de Dados (SGBD) que utiliza a linguagem SQL como interface com o usuário. É um dos bancos de dados mais populares para a web. É rápido, confiável e fácil de utilizar.

Em todo o nosso desenvolvimento, vamos utilizar o MySQL Workbench, que é a ferramenta oficial do MySQL. É um ambiente completo que permite, além de realizar consultas, criar diagramas e trabalhar com engenharia reversa.

Com o Workbench devidamente instalado, chegou a hora de nos conectarmos à base de dados e criamos um *schema* para que possamos, a partir dele, criar as nossas tabelas de uma forma mais organizada. Para que seja possível a criação, deveremos efetuar o login na base de dados e, após estarmos dentro do Workbench, clicar no botão “create new schema”, conforme a Figura 1:

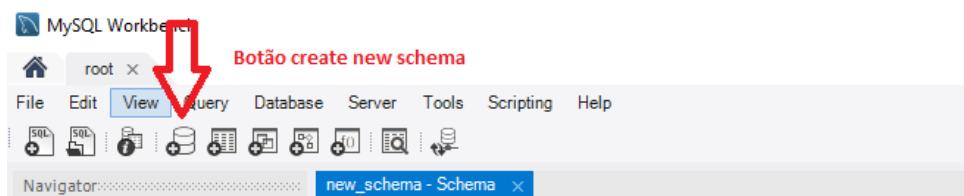


Figura 1: Workbench / Fonte: o autor.

Após clicar neste botão, será possível darmos um nome ao nosso novo *schema* de dados, conforme a Figura 2:

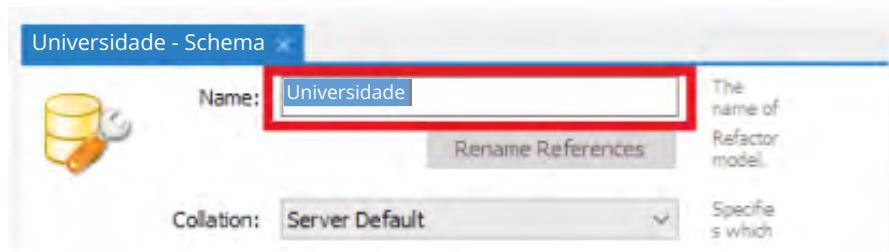


Figura 2: Nome do *schema* / Fonte: o autor.

Conforme vemos, o nome do nosso novo *schema* será “Universidade”. Vale lembrar que essa notação será utilizada em todo o nosso desenvolvimento. Para que seja criado de fato esse novo *schema*, será necessário clicar em “Apply”, em que será mostrado uma tela com o comando SQL a ser aplicado no banco de dados para a criação.



Figura 3: Tela com o comando SQL / Fonte: o autor.

Dessa maneira, basta clicar em “Apply” e, depois, em “Finish”, para que o SGBD aplique o comando no banco de dados e crie o nosso *schema*, que pode ser consultado por meio dos *schemas* presentes do lado esquerdo inferior no Workbench, conforme a Figura 4:

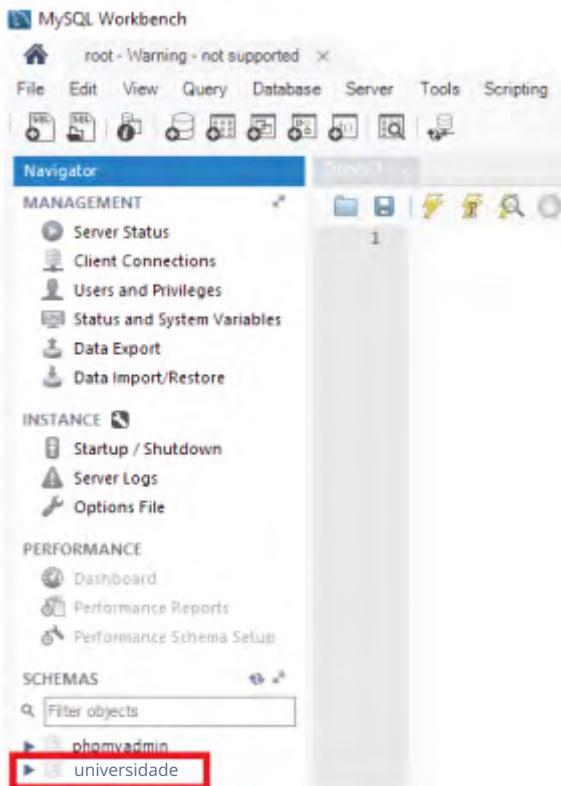


Figura 4: Workbench / Fonte: o autor.

A consulta aos *schemas* criados dentro do banco de dados pode acontecer de várias formas. Uma forma bem simples de verificar a criação do *schema* seria o seguinte comando:

Show schemas

Esse comando deverá ser executado com script SQL no Workbench. Para que seja aberto, devemos clicar no botão, logo abaixo de “File”. Após o editor de SQL aberto, basta digitar o comando e clicar no botão para que o Workbench execute os comandos. Após a execução, os dados serão apresentados logo a seguir, conforme vemos na Figura 5:

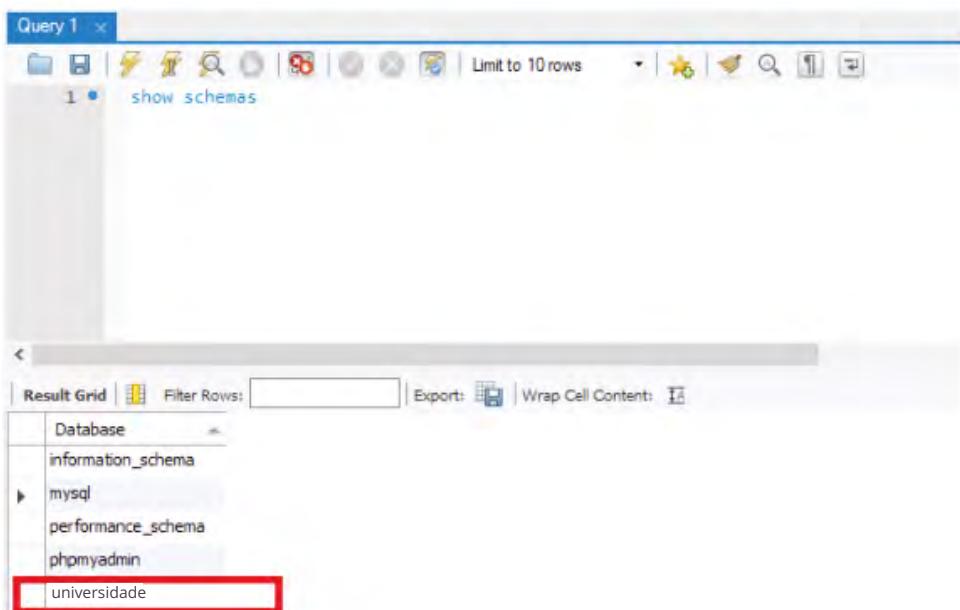


Figura 5: Exemplo da execução / Fonte: o autor.

Dessa forma, caro(a) aluno(a), você pode se perguntar: do que se trata a ferramenta Workbench? Ela é uma interface gráfica que traduz comandos feitos pelo usuário em linguagem SQL, para que, assim, seja possível sua aplicação no banco de dados. Todas as ações feitas dentro do Workbench podem ser substituídas por comandos via terminal no *prompt* do DOS, por exemplo.

TIPOS DE DADOS

Para que possamos efetuar a criação de uma tabela, primeiro necessitamos saber mais um pouco sobre os tipos de dados. Em seguida, identificaremos os principais tipos de dados do MySQL, que será a ferramenta adotada para o desenvolvimento de todo o nosso conteúdo. Conforme Passos (2010, s.p.):



Alguns campos numéricos possuem a opção UNSIGNED. Isso quer dizer que o número não pode ser negativo. Por exemplo: o campo TINYINT com a opção UNSIGNED vai de 0 até 255, sem a opção UNSIGNED vai de -128 até 127. A opção ZEROFILL completa o

campo com zeros. Por exemplo: se você cadastrar '65' em um campo INT(5) – ou seja, inteiro com 5 dígitos – com a opção ZEROFILL habilitada, o MySQL irá cadastrar '00065'. Sem a opção ZEROFILL habilitada, o MySQL irá cadastrar '65'.

Figura 6: Tabela de tipos numéricos / Fonte: adaptado de Passos (2010).

CRIAÇÃO E ALTERAÇÃO DE TABELAS

O comando para a criação de tabelas é um comando *Data Definition Language* (DDL). Dessa forma, trabalharemos com as tabelas de dados formando um cadastro de pessoa, assim podemos representar graficamente, conforme a Figura 7:

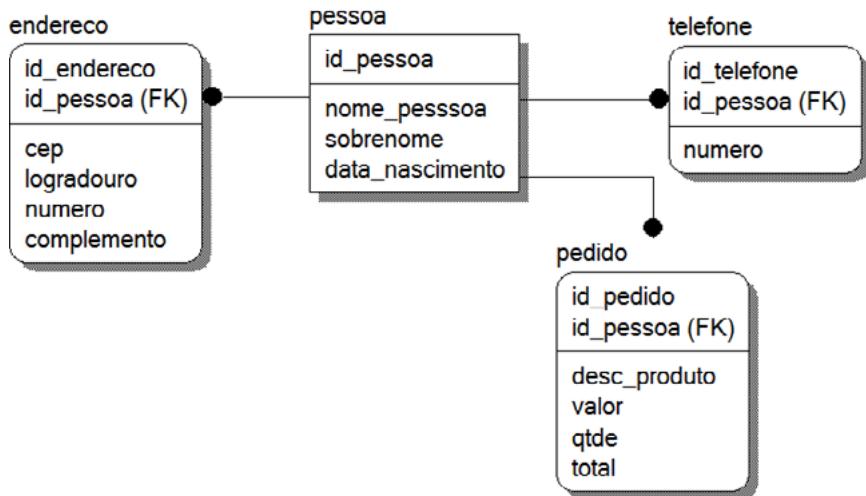


Figura 7: Comandos para criação de tabelas / Fonte: o autor.

De acordo com a figura, vamos ter em nossa base de dados um “cadastro” de pessoas que poderão ter vários endereços e telefones.

Sugiro que você, aluno(a), siga todos os passos deste livro utilizando o MySQL, pois, ao final, estaremos todos com os mesmos dados no banco de dados. Nosso próximo passo agora é a criação e a alteração das tabelas em nosso banco de dados.

A criação de tabelas se dá em um comando DDL. Para evidenciar melhor na prática como é de fato a criação e a alteração de tabelas em um banco de dados, os comandos a seguir podem ser executados de fato em seu banco de dados, a fim de que nossa disciplina fique mais prática. A sequência de comandos a seguir criará em nossa base de dados uma tabela chamada “pessoa”, a qual estará ligada a outras duas tabelas, endereços e telefones. Seguindo o raciocínio, nossa intenção é ter uma pessoa cadastrada em nossa base que possa conter vários endereços e telefones:

Criação da tabela pessoa:

```
CREATE TABLE 'unicesumar'.'pessoa' ('id_pessoa' INT NOT NULL,  
          'nome' VARCHAR(200) NOT NULL,  
          'sobrenome' VARCHAR(200) NOT NULL,  
          'data_nascimento' DATE NULL  
        );
```

Criação da tabela endereço:

```
CREATE TABLE 'unicesumar'.'endereco' ('id_endereco' INT NOT NULL,  
          'cep' INT(8) NOT NULL,  
          'logradouro' VARCHAR(200) NULL,  
          'numero' VARCHAR(20) NULL,  
          'complemento' VARCHAR(200) NULL  
        );
```

Criação da tabela telefone:

```
CREATE TABLE 'unicesumar'.'telefone' ('id_telefone' INT  
NOT NULL,  
          'numero' BIGINT(12) NOT NULL  
        );
```

CHAVE PRIMÁRIA

As criações de nossas tabelas foram feitas sem a identificação de uma chave primária. A função da chave primária em uma tabela no banco de dados é a de que nunca haverá a repetição de um mesmo valor para esse campo.

Visto tal conceito, alteraremos as nossas tabelas modificando os campos que iniciam com “id_” para que sejam nossas chaves primárias.

Alteração da tabela pessoa:

```
ALTER TABLE 'unicesumar'.'pessoa'  
ADD PRIMARY KEY ('id_pessoa');
```

Alteração da tabela endereço:

```
ALTER TABLE 'unicesumar'.'endereco'  
ADD PRIMARY KEY ('id_endereco');
```

Alteração da tabela telefone:

```
ALTER TABLE 'unicesumar'.'telefone'  
ADD PRIMARY KEY ('id_telefone');
```

Agora, com os comandos executados em seu banco de dados, temos as três tabelas devidamente criadas, mas sem dependência entre elas. Ou seja, todas estão distintas no banco de dados. Para solucionar esses problemas e trabalharmos melhor com a integridade dos dados, criaremos mais uma coluna (id_pessoa) nas tabelas de endereço e telefone. As tabelas mencionadas farão parte da criação de nossa *foreign key*, sendo referenciadas pelo campo identificador da tabela de pessoa.

Alteração da tabela de endereço:

```
ALTER TABLE 'unicesumar'.'endereco'  
ADD COLUMN 'id_pessoa' INT NOT NULL;
```

Alteração da tabela de telefone:

```
ALTER TABLE 'unicesumar'.'telefone'  
ADD COLUMN 'id_pessoa' INT NOT NULL;
```

Agora com as tabelas prontas, podemos criar referências entre as tabelas endereço e telefone com a tabela de pessoa. Para isso, devemos efetuar a criação da *foreign key* na tabela de endereço e telefone referenciando a tabela de pessoa.

Criação da foreign key na tabela de endereço:

```
ALTER TABLE 'unicesumar'.'endereco'  
ADD CONSTRAINT 'pessoa_fk_endereco'  
FOREIGN KEY ('id_pessoa')  
REFERENCES 'unicesumar'.'pessoa' ('id_pessoa');
```

Criação de foreign key na tabela de telefone:

```
ALTER TABLE 'unicesumar'.'telefone'  
ADD CONSTRAINT 'pessoa_fk_telefone'  
FOREIGN KEY ('id_pessoa')  
REFERENCES 'unicesumar'.'pessoa' ('id_pessoa');
```

Dessa maneira, garantiremos a integridade de nosso banco de dados, a fim de que todas as ligações tenham efeito. Antes de todas as alterações nas tabelas, tínhamos as tabelas distribuídas em nosso banco de dados, conforme a Figura 8:

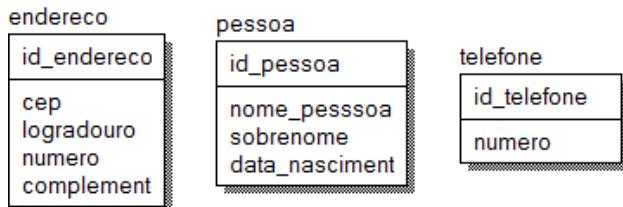


Figura 8: Exemplo / Fonte: o autor.

Após todas as alterações aplicadas em nossa base de dados, criando, assim, a integridade em nossos dados, elas, que antes não tinham nenhum tipo de ligação, agora formam o Modelo Entidade Relacionamento (MER) da Figura 9:

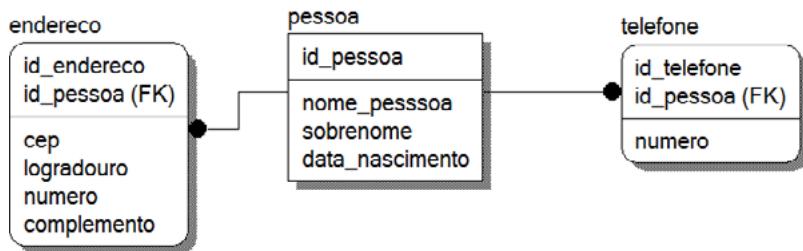


Figura 9: MER / Fonte: o autor.





POPULANDO AS TABELAS CRIADAS

O comando “**INSERT**” faz parte dos comandos de *Data Manipulation Language* (DML). Nesta etapa, iremos popular nossas tabelas de acordo com a criação e respeitando a integridade dos dados. Sempre que gravamos dados em tabelas, devemos obedecer à ordem hierárquica delas. Veremos a seguir o que seria essa ordem.

Para inserir os dados nas tabelas, vamos iniciar pela tabela de pessoa, em que iremos inserir um registro somente:

```
INSERT INTO 'unicesumar'.'pessoa'  
    ('id_pessoa',  
     'nome',  
     'sobrenome',  
     'data_nascimento')  
VALUES  
    (1, /*campo id_pessoa*/  
     'João', /*campo nome */  
     'Pereira', /*campo sobrenome */  
     '1983-09-15'); /* campo data_nascimento
```

Com os dados inseridos em nossa base, na tabela de pessoa, agora adicionaremos dois endereços referenciando essa pessoa cadastrada. Com a integridade entre as tabelas criadas adequadamente, podemos ter vários endereços para uma única pessoa e um endereço poderá estar ligado a apenas uma pessoa. Temos, assim, uma cardinalidade de N:1 partindo da tabela de pessoa. Atenção: no código a seguir, temos comentários sobre os campos do código que estão entre /* */

Inserindo o endereço 1:

```
INSERT INTO 'unicesumar'.'endereco'  
    ('id_endereco',  
     'cep',  
     'logradouro',  
     'numero',  
     'complemento',  
     'id_pessoa')  
VALUES  
    (1, /*campo id_endereco*/  
     86890880, /*campo cep*/  
     'Av. Paraná', /*campo logradouro*/  
     '27356', /*campo numero*/  
     'Casa', /*campo complemento*/  
     1); /*campo id_pessoa FK com a tabela de pessoa*/
```

Inserindo o endereço 2:

```
INSERT INTO 'unicesumar'.'endereco'  
    ('id_endereco',  
     'cep',  
     'logradouro',  
     'numero',  
     'complemento',  
     'id_pessoa')  
VALUES  
    (2, /*campo id_endereco*/  
     86890880, /*campo cep*/  
     'Av. Brasil', /*campo logradouro*/  
     '412 sala 2', /*campo numero*/  
     'Comerical', /*campo complemento*/  
     1); /*campo id_pessoa FK com a tabela de pessoa*/
```

Agora, nós já temos uma pessoa em nossa base que possui dois endereços. Por final, iremos inserir os telefones pertencentes a essa pessoa.

Inserindo o telefone 1:

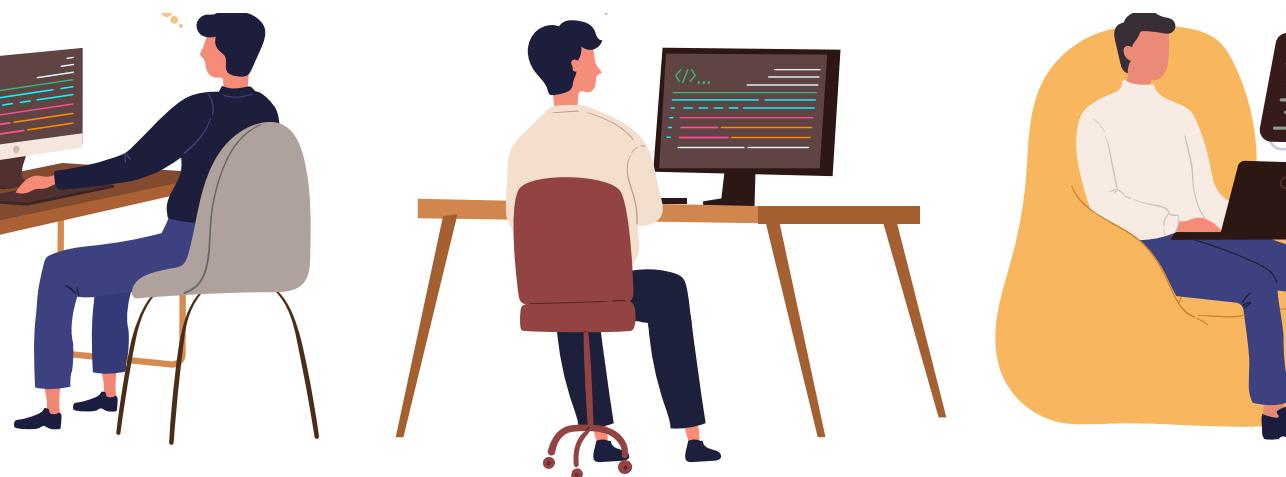
```
INSERT INTO 'unicesumar'.'telefone'  
    ('id_telefone',  
     'numero',  
     'id_pessoa')  
VALUES  
    (1, /* campo id_telefone */  
     4499786543, /* campo numero*/  
     1); /*campo id_pessoa FK com a tabela de pessoa*/
```

Inserindo o telefone 2:

```
INSERT INTO 'unicesumar'.'telefone'  
    ('id_telefone',  
     'numero',  
     'id_pessoa')  
VALUES  
    (2, /* campo id_telefone */  
     4499786523, /* campo numero*/  
     1); /*campo id_pessoa FK com a tabela de pessoa*/
```

A partir dos dados inseridos nas tabelas, podemos tirar algumas conclusões. A pessoa com nome de João possui dois endereços, um na Av. Paraná e outro na Av. Brasil, sendo o primeiro o endereço de sua casa, e o outro de sua empresa. Além disso, ele possui dois números de telefone.

Um forte elemento para a aprendizagem da linguagem SQL é a prática. Desse modo, sugiro a você que efetue mais inserções no banco de dados seguindo os comandos exemplificados anteriormente, sempre lembrado de que não é possível ter um valor de uma chave primária repetida na tabela.



COMANDO DESCRIBE

É de extrema importância consultar a estrutura das tabelas já criadas em um banco de dados. Essa consulta é efetuada por meio do comando “describe”. Com ele, é possível visualizar as colunas e os tipos de colunas de uma tabela em específico.

Para o uso desse comando, é necessário efetuar a interpretação dos dados retornados, pois será a partir desses dados que você saberá tudo sobre a tabela. Desses dados retornados, podemos evidenciar: colunas de índices, NOT NULL, NULL, PRIMARY KEY e quais os campos que compõem a PRIMARY KEY (chave primária). Também veremos qual é o valor *default* (padrão) para determinada coluna e se a PRIMARY KEY é autoincrementada pelo próprio banco de dados. Vale lembrar que o comando DESC é um sinônimo do comando DESCRIBE.

Para utilizar esse comando, devemos obedecer à sintaxe a seguir:

```
DESCRIBE nome_da_tabela;
```

Vamos exemplificar esse comando com a tabela pessoa, a qual já criamos nesta mesma unidade. O comando a ser executado deverá ser:

```
describe pessoa;
```

O retorno dos dados para esse comando deverá ser uma linha para cada coluna da tabela conforme Figura 10:

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
id_pessoa	int(11)	NO	PRI	NULL	
nome	varchar(200)	NO		NULL	
sobrenome	varchar(200)	NO		NULL	
data_nascimento	date	YES		NULL	

Figura 10: Modelo de uma linha para cada coluna / Fonte: o autor.

As informações mais importantes sobre o retorno desse comando são:

- **Field** – essa informação representa o nome das colunas presentes na tabela do banco de dados.
- **Type** – indica o tipo de dado de cada coluna da tabela.
- **Null** – o conceito é bem simples, indica a obrigatoriedade da coluna, ou seja, se ela aceita valores nulos, representados pelo retorno YES e NO. Como no exemplo anterior, apenas o campo data_nascimento poderá ser nulo, os demais são campos obrigatórios.
- **KEY** – esse indicador poderá retornar apenas três valores: PRI, UNI, MUL.
- **PRI** – indica que a coluna é uma chave primária da tabela ou faz parte da composição juntamente com outros campos. Caso seja uma chave primária composta, o valor PRI estará presente em mais campos da tabela, referenciadas na coluna KEY.
- **UNI** – é um campo UNIQUE + NOT NULL. Esse tipo de campo é muito parecido com a chave primária da tabela, pois toda chave primária da tabela não pode ser nula e deve ser única. Esse conceito de UNI é utilizado para que não existam repetições de valores nesse campo presente na tabela.

- **MUL** – é a definição de uma coluna que é ou é parte de um índice não único que aceita valores múltiplos e NULL.
- **DEFAULT** – é utilizado para que seja possível o banco de dados inserir um valor padrão caso esse valor não seja informado no momento do *insert*. Apesar de um comando muito simples, o DESCRIBE é um dos recursos disponíveis mais importantes do MySQL. Por isso, conhecê-lo e saber interpretar os seus resultados é de extrema importância. No dia a dia dos programadores de banco de dados, esse é um dos comandos mais utilizados.

 INDICAÇÃO DE LIVRO

Projeto de Banco de Dados - vol. 4 (2008).

Editora: Bookman

Autor: Carlos Alberto Heuser

Sinopse: em sua sexta edição e adotado por faculdades de todo o Brasil, o livro aborda as duas primeiras etapas do ciclo de vida de um banco de dados: modelagem conceitual e projeto lógico.



NOVOS DESAFIOS

Prezado(a) aluno(a)! Neste tema, procuramos demonstrar na prática a utilidade dos conceitos mais importantes. Trabalhamos na prática os comandos DDL e DML. Espero que esta prática tenha evidenciado a você como é prazeroso trabalhar com banco de dados, compreendendo os princípios teóricos que abrangem este grande mercado de trabalho. Dessa maneira, procuramos exemplos da prática. Sugiro que caso você não tenha seguidos os passos, execute como exercício cada comando deste tema, pois o passo a passo criará sua base de dados, as tabelas e irá populá-las.

Todos os conceitos abordados fazem parte do dia a dia de um programador e é essencial para o desenvolvimento de qualquer programa que trabalhe em conjunto com um banco de dados.

AGORA É COM VOCÊ

1. O comando alter table se encontra em qual definição?

- a) DML
- b) DTL
- c) DQL
- d) DDL
- e) DCL

2. O comando insert se encontra em qual definição?

- a) DML
- b) DTL
- c) DQL
- d) DDL
- e) DCL

3. Sobre os tipos de dados assinale, verdadeiro ou falso para os itens relacionados abaixo:

- () DATETIME: combinação de data e hora no formato
- () YEAR: ano com 4 dígitos
- () LONGBLOB: string com até 4Gb
- () DECIMAL: não armazena números inteiros

- a) V, V, V, F
- b) V, F, V, F
- c) V, V, F, F
- d) F, V, F, V
- e) F, F, V, V

MEU ESPAÇO



A LINGUAGEM SQL

ESP. JEFERSON KAISER

MINHAS METAS

- Compreender cada divisão da linguagem SQL.
- Importância e ligações entre as divisões da linguagem SQL.

INICIE SUA JORNADA

Olá, caro(a) aluno(a)! Neste tema, serão descritas as linguagens formais que proporcionarão uma notação concreta para o desenvolvimento de consultas, alterações e remoções em um banco de dados. Contudo, os bancos de dados comerciais necessariamente precisam de uma linguagem fácil para o usuário. Ainda neste tema, traremos para você os principais conceitos da linguagem SQL. A sigla SQL vem do inglês *Structured Query Language*, em português: Linguagem de Consulta Estruturada.

Embora estejamos falando da linguagem SQL como uma linguagem de consulta de dados, ela tem várias outras funções. Ela também é utilizada para a definição da estrutura de dados, modificações do banco de dados, especificações de segurança, dentre outras.

Aqui, não temos a intenção de passar detalhadamente tudo sobre SQL para você. Vamos apenas trabalhar os construtores e os principais conceitos da linguagem SQL, pois essa linguagem certamente é a linguagem padrão dos bancos de dados relacionais, devido à sua simplicidade e facilidade de uso. A SQL é a linguagem de mais alto nível para a manipulação de dados dentro do modelo relacional (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

Bom estudo!

DESENVOLVA SEU POTENCIAL

HISTÓRIA DA SQL

Tudo começou no início dos anos 1970, por meio de uma pesquisa da empresa International Business Machine (IBM), com o Dr. E. F. Codd, a qual levou ao desenvolvimento de um produto chamado SEQUEL (ou Linguagem de Consulta em Inglês Estruturado, em português). Posteriormente, a SEQUEL veio a se transformar na SQL ou linguagem de consulta estruturada.

A IBM e os demais fornecedores de banco de dados relacionais querem padronizar a forma de acessar e manipular seus dados em um banco de dados

relacional. Embora a IBM tenha sido a primeira a desenvolver esse tipo de tecnologia, foi a Oracle quem lançou comercialmente o banco de dados relacional.

SQL Como uma Linguagem Padrão

Um grande padrão de banco de dados é a linguagem SQL decorrente de sua simplicidade e facilidade de uso. Diferencia-se das demais linguagens de consulta de banco de dados no sentido em que se é especificada a forma do resultado, sem ter nenhum tipo de preocupação com o percurso percorrido para se chegar ao resultado. A SQL tem o seu ciclo de aprendizado menor que as demais linguagens de programação, pois ela é uma linguagem declarativa, o que se opõe às demais linguagens de programação procedurais.

Embora a SQL tenha sido criada pela IBM, ocorreram várias derivações dessa linguagem criadas por outros autores. Visto o crescimento da linguagem, foi necessária a criação de uma padronização para a linguagem. Essa tarefa ficou a cargo do American National Standards Institute (ANSI) no ano de 1986 e pela International Organization for Standards (ISO) em 1987. A SQL foi revista em 1992 e essa versão recebeu o nome de SQL-92, sendo revista novamente em 1999, levando o nome de SQL-99 ou de SQL-3. Embora existam essas padronizações ANSI e ISO, ela ainda possui variações e extensões produzidas pelos diferentes fabricantes de Sistema de Gerenciamento de Banco de Dados (SGBD). A linguagem SQL pode ser definida em várias partes:

- ***Data Manipulation Language – DML (Linguagem de Manutenção de Dados)*** – esses comandos são utilizados para realizar inclusões, exclusões e alterações de dados, os quais são utilizados a partir dos comandos INSERT, UPDATE e DELETE (WATSON; RAMKLASS, 2012).
- ***Data Definition Language – DDL (Linguagem de Definição de Dados)*** – conjunto de comandos dentro da SQL que permitem ao desenvolvedor utilizá-las para a definição das estruturas de dados, contendo instruções as quais permitem a criação, modificação e remoção de tabelas, bem como a criação, a alteração e a remoção de elementos associados às tabelas (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

- **Data Control Language – DCL (Linguagem de Controle de Dados)**
– são os comandos que permitem ao administrador do banco de dados gerenciar as autorizações dos dados e licenças dos usuários para controlar o acesso de quem pode ver ou manipular dados dentro do banco de dados. Os comandos mais utilizados são: GRANT, REVOKE, SET e LOCK.
- **Data Transaction Language – DTL (Linguagem de Transação de Dados)** – são os comandos utilizados para gerenciar as mudanças feitas nos dados do banco de dados por meio de um comando DML. Ele permite que as instruções sejam agrupadas por transações. Os principais comandos são: ROLLBACK, COMMIT e SAVEPOINT.
- **Doctrine Query Language – DQL (Linguagem de Consulta de Dados)** – embora com apenas um comando, se não o mais importante de todos os comandos utilizados na SQL, o SELECT permite ao usuário efetuar uma consulta no banco de dados. Esse comando é composto de várias cláusulas e opções, possibilitando das consultas mais simples às consultas mais complexas (CARVALHO, s.p., 2016).



DQL - LINGUAGEM DE CONSULTA DE DADOS

A estrutura de um *select* simples consiste em três cláusulas: *select*, *from* e *where*.

- **Cláusula SELECT** – é responsável por relacionar os atributos desejados no resultado de uma consulta, ou seja, ele permite recuperar os dados de uma tabela do banco de dados. Esse comando corresponde à operação da álgebra relacional.
- **Cláusula FROM** – é obrigatória em toda instrução *select*, pois é ela a responsável por associar as relações que serão pesquisadas durante a evolução da expressão. É ela a responsável por ligar o campo que iremos recuperar no *select* com a tabela no banco de dados. Essa cláusula corresponde à operação de produto cartesiano da álgebra relacional.
- **Cláusula WHERE** – é responsável por estabelecer uma condição de pesquisa, ou seja, tem a função de filtrar os dados a serem recuperados do banco de dados. A cláusula *where* também é utilizada nos comandos de UPDATE e DELETE. Ela corresponde à seleção do predicado da álgebra relacional.

Cláusula Select

A declaração SELECT é um mecanismo extremamente elegante, fácil de ser trabalhado e altamente extensível. O mecanismo foi criado com o intuito de recuperar os dados existentes no banco de dados. Afinal, não teria lógica ter dados gravados em um banco de dados se não fosse possível fazer a leitura deles. As cláusulas do SELECT vêm do inglês e, se pensarmos a partir da tradução, ficará mais simples ainda a compreensão. As cláusulas são:

SELECT – em português, “selecionar”.

FROM – em português, “a partir de”.

WHERE – em português, “onde”.

Os dados retornados por uma consulta SQL são, naturalmente, uma relação, pensando em uma consulta simples. Segue o exemplo de um SELECT simples:

```
SELECT nome_pessoa  
      FROM pessoa
```

A partir dessa consulta, encontraremos o nome de todas as pessoas da tabela “pessoa” do banco de dados. Esse resultado é uma relação de um campo simples, titulada de nome_pessoa na tabela “pessoa”. A linguagem SQL, em sua maioria, permite duplicidade em suas relações, podendo trazer dados duplicados no resultado de sua consulta. No caso, para forçarmos a eliminação dos resultados duplicados, utilizamos a instrução *distinct* depois da cláusula *select*. Se reescrevermos a consulta anterior, ficará da seguinte maneira:

```
SELECT DISTINCT nome_pessoa  
      FROM pessoa
```

No mesmo formato anterior, também é possível a utilização do asterisco (*). Ele é usado para denotar que o *select* deverá trazer todos os campos disponíveis para consulta. A consulta com o uso do asterisco ficará da seguinte maneira:

```
SELECT DISTINCT *  
      FROM pessoa
```

Dessa maneira, solicitamos ao banco de dados que sejam apresentados todos os campos disponíveis na tabela “pessoa” e que não possua dados repetidos.

WHERE

O WHERE é utilizado para delimitar os dados a serem retornados pela nossa consulta. Os filtros a serem aplicados se restringem às linhas utilizando operadores de comparação em um conjunto de campos e valores literários. Os filtros trabalham com os operadores booleanos, que fornecem um mecanismo para

especificar condições múltiplas para filtrar as linhas a serem retornadas. Segue o exemplo de como ficaria a implementação do *where* na consulta simples:

```
SELECT *
FROM     pessoa
WHERE   codigo_pessoa =1
```

```
SELECT *
FROM     pessoa
WHERE   nome_pessoa = 'JOAO'
```

O primeiro exemplo trata de uma consulta para trazer os dados da tabela “pessoa”, desde que exista uma pessoa com o código da pessoa que contenha o número um. No segundo exemplo, o resultado depende extremamente de o nome da pessoa ser JOAO.

Condições Baseadas em Números

As condições devem ser propriamente utilizadas de acordo com o tipo de dado a ser filtrado. Vale lembrar que, nas condições numéricas, não é necessário colocar a restrição entre aspas simples. Segue um quadro com os possíveis operadores para as condições numéricas:

OPERADOR	DESCRIÇÃO
<	Menor que
>	Maior que
<=	Menor que ou igual a
>=	Maior que ou igual a
>=	Não igual
!=	Não igual
=	Igual a

Quadro 1: Possíveis operadores para as condições numéricas / Fonte: o autor.

A partir desse quadro, podemos elaborar os filtros que se aplicam a nossa necessidade.

Condições Baseadas em Caracteres

As condições determinantes para as linhas selecionadas, baseadas em caracteres, se dão pelo fato de delimitar os caracteres dentro de aspas únicas, em que a não utilização das aspas gera um erro em sua execução. Lembre-se de que existe a distinção entre maiúsculos e minúsculas. Exemplo: where nome = ‘joão’ é diferente de where nome = ‘João’.

Condições Baseadas em Data

Os campos de tipo data são muito úteis para o armazenamento de datas e horas. As datas, quando utilizadas nos filtros, devem seguir o mesmo padrão utilizados para os caracteres sendo delimitadas por aspas única.

FROM

Finalmente, vamos falar da cláusula FROM. Ela é responsável por fazer a ligação com a tabela que vamos efetuar a consulta. Em resumo, ela especifica as tabelas que possuem as colunas listadas na cláusula SELECT. Por exemplo, para trazermos em uma consulta todos os dados da tabela “pessoa”, teríamos que especificar a tabela “pessoa” na cláusula FROM do SELECT, conforme exemplo a seguir:

```
SELECT *
FROM     pessoa
```

Nesse exemplo, trazemos todos os dados da tabela pessoa.

```

        }
        return trim(str);
    }

    if (!defined('T_ML_COMMENT'))
        define('T_ML_COMMENT', '');
    else
        define('T_DOC_COMMENT', '');

    function strip_comments($source)
    {
        $tokens = token_get_all($source);
        foreach ($tokens as $token) {
            if (is_string($token)) {
                $ret .= $token;
            } else if (is_array($token)) {
                switch ($token[0]) {
                    case T_COMMENT:
                    case T_DOC_COMMENT:
                        break;
                    default:
                        $ret .= $token[1];
                }
            }
        }
        return $ret;
    }
}

```

DML - LINGUAGEM DE MANIPULAÇÃO DE DADOS

DML é a linguagem de manipulação de dados e são os comandos dentro da linguagem SQL utilizados para a inclusão, a remoção e a alteração de dados em um banco de dados. Lembrando que os principais comandos são: INSERT, UPDATE e DELETE.

Segundo John Watson e Roopesh Ramklass (2012), a maioria dos profissionais não inclui o *select* como sendo um comando DML, pois ele é considerado uma linguagem separada em seu próprio direito.

INSERT

O comando INSERT é simples, digamos que ele é um pedido de inclusão de uma linha em uma tabela. Vale lembrar que os valores a serem inseridos no banco de dados devem pertencer ao domínio do campo. Contudo, as ordens dos campos devem seguir a ordem de criação dos campos na tabela. As linhas podem ser preenchidas de diversas maneiras, mas a maneira mais utilizada e simples é o INSERT. As versões mais básicas da instrução inserem apenas uma linha em uma tabela, mas as variações mais complexas podem chegar a inserir várias linhas em várias tabelas.

Por meio do comando INSERT, podemos inserir uma ou mais linhas no banco de dados. Segue um exemplo da instrução INSERT em sua variação mais simples.

```

INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3, ....
    colunaN)

VALUES (valor1, valor2, valor3, ..., valorN);

```

UPDATE

O comando UPDATE é utilizado para efetuar alterações nas linhas já existentes no banco de dados, que, possivelmente, foram gravadas por meio do comando INSERT. Da mesma maneira que o INSERT, o UPDATE pode afetar uma linha somente ou um conjunto de linhas existentes na tabela. Para que se possa delimitar a linha ou quais linhas receberão a alteração, é utilizada a cláusula WHERE, que recebe os mesmos tratamentos do SELECT.

Um grande diferencial do UPDATE é que não é possível atualizar campos de mais de uma tabela de uma única vez. Ao atualizar uma linha ou várias linhas, o UPDATE determina quais colunas atualizar e, assim, não é necessário atualizar cada coluna da linha. Poderá ser alterada apenas uma coluna da linha. Caso a coluna já estiver preenchida com algum valor, o valor antigo será substituído pelo novo valor passado pelo comando UPDATE. Caso a coluna estiver vazia, ela será preenchida depois do UPDATE com o novo valor. A sintaxe básica é a seguinte:

```
UPDATE nome_da_tabela SET coluna = 'novo_valor'
```

DELETE

As linhas inseridas e alteradas com os exemplos anteriores agora poderão ser removidas por meio do comando DELETE. Esse comando pode remover uma linha ou um conjunto de linhas da tabela. A quantidade de linhas a ser removida dependerá da cláusula WHERE. Caso a cláusula não for utilizada, o comando removerá todas as linhas pertencentes à tabela, o que pode ser um problema se você se esquecer de colocar a cláusula accidentalmente. Uma exclusão é tudo ou nada, pois não se pode definir colunas para a remoção, apenas a linha inteira. A sintaxe mais simples para a remoção de uma linha é a seguinte:

```
DELETE nome_da_tabela;
```

Por meio desse comando, removeremos todos os dados da tabela informada. O efeito de uma instrução DML não é permanente até você confirmar a transação que irá o incluir. A transação é uma sequência de instruções SQL, podendo ser uma única instrução DML. Até que uma transação seja confirmada, ela pode ser revertida (desfeita).

DDL - LINGUAGEM DE DEFINIÇÃO DE DADOS

Como já tratamos no início desta unidade, DDL (em inglês, *Data Definition Language*) é um conjunto dos comandos da linguagem SQL utilizado para a definição das estruturas dos dados, fornecendo as instruções para a criação, a modificação e a remoção das tabelas do banco de dados.

Valores NOT NULL

Na linguagem SQL, podemos definir que uma tabela poderá conter campos com valores nulos. O único campo que não poderá ficar nulo é a chave primária. Caso seja obrigatório o preenchimento, deve-se criar esse campo com o atributo NOT NULL, pois, assim, reforçamos a necessidade de se informar um valor para um determinado campo na tabela. Dessa maneira, o campo especificado não poderá deixar de ser informado, pois não pode ficar em branco, ou seja, não poderá conter NULL em seu valor.

Chaves e Integridade

Chaves primárias

Para que seja possível especificar uma chave primária, utiliza-se a cláusula PRIMARY KEYS (em inglês, *Primary Keys* ou “PK”). Uma chave primária tem a função de se tornar um registro em uma tabela única. Assim, nunca haverá a repetição de um mesmo valor para este campo. A chave primária pode ser atribuída a um ou mais campos, considerando, dessa maneira, que nunca haverá a repetição da combinação desses dois valores na mesma tabela no banco de da-

dos. Segue a sintaxe para a utilização da PRIMARY KEYS, lembrando que esses comandos vão ao final da sintaxe de criação das tabelas:

```
CREATE TABLE nome_da_tabela (
    nome_do_campo INT NOT NULL
    PRIMARY KEY (nome_do_campo));
```

A chave estrangeira acontece quando um campo de uma tabela for chave primária em outra tabela. Podemos pensar da seguinte maneira: sempre que houver o relacionamento 1:N entre duas tabelas, a tabela 1 receberá a chave primária e a tabela N receberá a chave estrangeira. A seguir, a sintaxe para a criação:

```
CREATE TABLE nome_da_tabela(
    'id_tabela' INT NOT NULL,
    'campo1' VARCHAR(45) NOT NULL,
    'campo2' INT NOT NULL,
    PRIMARY KEY ('id_tabela'),
    CONSTRAINT 'campo2'
        FOREIGN KEY ('campo2')
            REFERENCES 'tabela_referenciada' ('campo_referenciado')
            ON DELETE NO ACTION
            ONUPDATE NO ACTION);

```

Criando uma Tabela Simples

Há várias maneiras de se armazenar uma tabela no banco de dados, porém a mais simples e utilizada é a tabela empilhada. Essa pilha são as linhas. O seu comprimento pode ser variado de forma aleatória, podendo haver uma correlação entre as linhas inseridas com a ordenação de armazenamento. Para a criação das tabelas, é utilizado o comando CREATE TABLE, em que o primeiro parâmetro

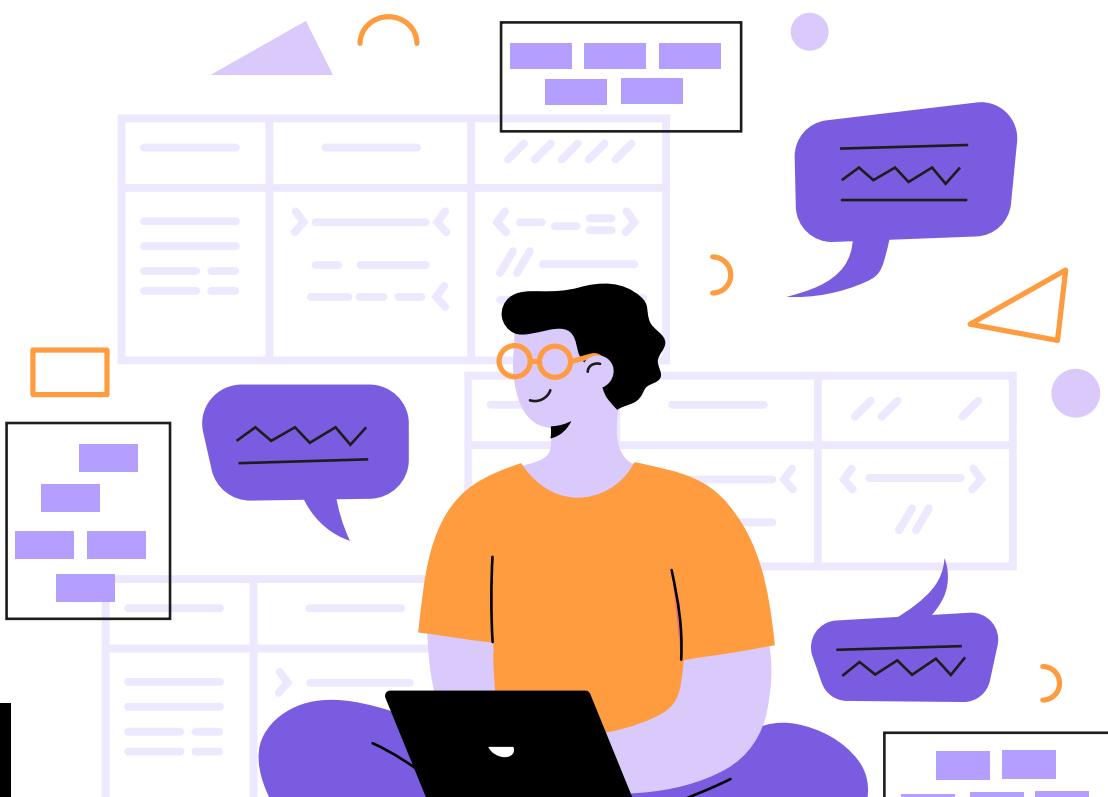
desse comando é o nome da tabela, seguido dos campos, seus respectivos tipos e suas devidas restrições. Por exemplo:

```
CREATE TABLE schema.nome_da_tabela (nome_da_coluna tipo_da_coluna);
```

Obrigatoriamente, deve-se especificar o nome da tabela a ser criada. Essa tabela deverá conter no mínimo um campo, para que seja possível sua criação. Segue o exemplo para a criação da tabela “pessoa” e “telefone” da Figura 1:



Figura 1: Exemplo para a criação da tabela / Fonte: o autor.



Os comandos a seguir são responsáveis por criar as tabelas definidas na figura:

```
CREATE TABLE 'pessoa' (
    'id_pessoa' INT NOT NULL,
    'nome_pessoa' VARCHAR(100) NOT NULL,
    'peso' DECIMAL(10,2) NULL,
    PRIMARY KEY ('id_pessoa'));
```

```
CREATE TABLE 't telefone' (
    'id_telefone' INT NOT NULL,
    'numero' VARCHAR(45) NOT NULL,
    'id_pessoa' INT NOT NULL,
    PRIMARY KEY ('id_telefone'),
    CONSTRAINT 'id_pessoa'
        FOREIGN KEY ('id_pessoa')
        REFERENCES 'pessoa' ('id_pessoa')
        ON DELETE NO ACTION
        ON UPDATE NO ACTION);
```

Alterando as Definições de uma Tabela já Existente

Há muitas alterações que podem ser efetuadas em uma tabela após a sua criação e muitas alterações em seu meio físico. Essas são de responsabilidade do administrador do banco de dados, mas muitas outras são puramente lógicas e poderão ser feitas por meio dos desenvolvedores SQL. Seguem exemplos das alterações possíveis:

Adicionar uma coluna

```
alter table nome_da_tabela add (nome_do_campo tipo_do_campo);
```

Modificando uma coluna

```
alter table nome_da_tabela modify (nome_do_campo novo_tipo_campo);
```

Deletando uma coluna

```
alter table nome_da_tabela drop column nome_do_campo;
```

Adicionando uma restrição a coluna

```
ALTER TABLE nome_da_tabela ADD FOREIGN KEY (tabela1_fk_tabela2) REFERENCES tabela_referenciada;
```

Para adicionar a restrição de não nulo

```
ALTER TABLE nome_da_tabela ALTER COLUMN nome_do_campo SET NOT NULL;
```

Removendo restrição

```
ALTER TABLE produtos DROP CONSTRAINT nome_da_restricao;
```

Removendo restrição *not null*

```
ALTER TABLE produtos ALTER COLUMN cod_prod DROP NOT NULL;
```

Deletando uma Tabela

O comando `DROP TABLE` permite a remoção de uma tabela do banco de dados. Essa operação remove linhas, estrutura e índices de acesso associados à tabela. Esse comando é bem simples em relação aos demais. Segue a sintaxe:

```
drop table nome_da_tabela;
```

DCL - LINGUAGEM DE TRANSAÇÃO DE DADOS

A DCL, conhecida também como linguagem de controle de dados, permite o controle de acesso e manipulação a dados dentro do banco de dados. Normalmente, esses comandos são utilizados para controlar a distribuição de privilégios entre um usuário e outro.

APROFUNDANDO

O que é computação nas nuvens (*cloud computing*)?

O termo “computação nas nuvens” vem do inglês “*cloud computing*”, que é a atual tendência para softwares. Alguns anos atrás, para você poder acessar um programa, como o Word, por exemplo, era necessário instalá-lo no seu computador, pagando uma taxa (em geral alta). Quando falamos de computação nas nuvens, partimos do princípio de que você não precisa instalar o Word no seu computador e poderá acessá-lo pela internet (pagando pelo uso ou um valor fixo mensal – bem abaixo do que quando precisava comprá-lo). Mas isso é possível? Sim, o Word pode ser acessado apenas pela internet. É fácil perceber que os dados desse aplicativo não estão no seu computador, mas em uma rede com acesso à internet. Uma vez conectado, você poderá desfrutar de todas as suas ferramentas on-line e acessar seu trabalho de qualquer lugar.

Fonte: Computação... (2015, s.p.).

Concedendo Permissões

O comando GRANT é o meio pelo qual é possível conceder privilégios para acessar objetos dentro do banco de dados, papéis e públicos. As concessões também permitem conceder privilégios a nível de sistema do banco de dados para usuário e papéis.

As permissões em tabelas podem conceder diferentes tipos de níveis de acesso dentro do banco de dados. Esse acesso poderá ser muito específico, por exemplo, você pode conceder acesso às colunas específicas de uma tabela. As permissões a nível de sistema permitem conceder diferentes funcionalidades a determinados usuários dentro do banco de dados, como a capacidade de criar uma tabela ou alterar as configurações de parâmetros de uma sessão de um usuário. Uma vez que um privilégio é atribuído a um usuário, ele entrará em vigor imediatamente.

Segue a sintaxe do GRANT:

```
GRANT ALL ON *.* TO 'usuario'@'seuhost';
```

O comando REVOKE revoga direitos de acesso do usuário ou privilégios para os objetos de banco de dados. Podemos dizer que esse comando remove os privilégios concedidos pelo comando GRANT.

Um determinado usuário pode remover somente os privilégios que foram concedidos diretamente por esse usuário. Se, por exemplo, o usuário José concedeu um privilégio com opção de concessão para o usuário Maria e o usuário Maria, por sua vez, concedeu o privilégio para o usuário João, então o usuário José não poderá revogar diretamente o privilégio de João. Em vez disso, o usuário José poderá revogar a opção de concessão do usuário Maria usando a opção CASCADE, para que o privilégio seja, por sua vez, revogado do usuário João. Outro exemplo é o caso em que tanto José quanto Maria concederam o mesmo privilégio a João. Nesse caso, José poderá revogar sua própria concessão, mas não poderá revogar a concessão feita por Maria e, portanto, João continuará com o privilégio ainda que José revogue o privilégio.

Segue a sintaxe do REVOKE:

```
REVOKE ALL ON *.* TO 'usuario'@'seuhost';
```

Dessa maneira, aprendemos como conceder e remover os privilégios de um usuário, a nível de um objeto ou até a nível de banco de dados. Esse tipo de controle é muito utilizado a fim de garantir a segurança e a integridade das informações no banco de dados.

DTL - LINGUAGEM DE TRANSAÇÃO DE DADOS

O conceito por trás de uma transação faz parte de um paradigma do banco de dados relacional. Uma transação completa consiste em um ou mais comandos DML seguidos por um comando COMMIT ou ROLLBACK. Uma sessão começa uma transação a partir do momento que qualquer instrução INSERT, UPDATE ou DELETE é emitida ao banco de dados. Essa transação continuará por inúmeros comandos DML até que a sessão emita uma instrução COMMIT ou ROLLBACK. Somente após esses comandos, a alteração acontecerá de fato no banco de dados e se tornará visível aos demais usuários do banco de dados.

- **COMMIT** – ele tem a função de confirmar os comandos DML aplicados ao banco de dados.
- **ROLLBACK** - faz com que as mudanças nos dados feitos pelos comandos DML existentes, desde o último COMMIT ou ROLLBACK, sejam descartadas.

Digamos que você pediu para excluir dados dentro do banco de dados:

```
DELETE FROM pessoa;
```

Quando uma exclusão é efetuada, podemos confirmar essa exclusão com a utilização do comando *commit*. Tome muito cuidado com essa utilização, pois uma confirmação efetuada (*commit*) se torna irreversível. Esse comando é muito útil e muito utilizado quando temos cenários com vários usuários conectados à mesma base de dados, pois ele trabalha especificamente em cada seção dos usuários. Após o seu uso, as modificações se tornam visíveis a todos os demais usuários (CARVALHO, 2016, s.p.).

Mas caso não queira que os dados sejam gravados de fato no banco de dados, podemos utilizar o ROLLBACK. Dessa maneira, os dados anteriores são “copiados” novamente para o banco de dados, retornando, assim, à sua versão original.



PENSANDO JUNTOS

Tenha em mente que somente dados corretos proporcionarão informações confiáveis para criar estratégias e soluções válidas para a empresa. Afinal, não adianta investir em ferramentas, tecnologias e aplicativos variados se os dados que estão armazenados não têm qualidade e credibilidade.

Fonte: o autor.

NOVOS DESAFIOS

Caro(a) aluno(a), com a finalização do estudo sobre este conteúdo, podemos concluir que a prática da SQL é de extrema importância para o amplo entendimento e conhecimento especializado. Durante este tema, estudamos os principais comandos de um banco de dados para a criação, a alteração e a remoção das tabelas, além de comandos para a inserção, a alteração e a remoção de dados em uma tabela. Também estudamos as permissões dos usuários com a concessão e revogação de privilégios, a fim de garantir a segurança e a integridade do nosso banco de dados.

Vimos que a cada comando DML executado no banco de dados é controlado por uma transação, ou seja, por meio dos comandos DTL. Todos esses dados foram trabalhados de uma forma genérica para todos os bancos de dados que usam como linguagem a SQL.

AGORA É COM VOCÊ

1. Quais são os comandos DML?
 - a) Insert, update, select, delete
 - b) Drop, delete, create
 - c) Truncate, delete
 - d) Insert, update, delete
 - e) Rollback, commit.
2. O comando é responsável por relacionar os atributos desejados no resultado de uma consulta, ou seja, ele permite recuperar os dados de uma tabela do Banco de Dados, esse comando corresponde à operação da álgebra relacional. Estamos falando de?
 - a) Select
 - b) From
 - c) Where
 - d) And
 - e) In
3. O _____ é utilizado para delimitar os dados a serem retornados pela nossa consulta, os filtros a serem aplicados se restringem a linhas utilizando operadores de comparação em um conjunto de campos e valores literários.
 - a) Select
 - b) From
 - c) Where
 - d) Width
 - e) View



MANIPULAÇÃO DE DADOS

ESP. JEFERSON KAISER

MINHAS METAS

- Identificar na prática a utilidade das teorias aplicadas com relação à linguagem SQL.
- Compreender as atividades básicas do dia a dia de um programador de Banco de Dados.

INICIE SUA JORNADA

Caro(a) acadêmico(a)! Neste tema, vamos nos aprofundar mais sobre os comandos da Structured Query Language (SQL), scripts que podemos utilizar por meio de um Sistema de Gerenciamento de Banco de Dados (SGBD). Dessa forma, iremos trabalhar com retorno de dados, consultas e comandos avançados, sendo utilizados para um melhor aproveitamento de nosso Banco de Dados.

Muito provavelmente, para você ter chego a este tema, você deve ter um conhecimento anterior e relativo à linguagem SQL e suas classificações, como: a Data Query Language (DQL), em português, Linguagem de Consulta de Dados; a Data Manipulation Language (DML), em português Linguagem de Manipulação de Dados; a Data Definition Language (DDL), em português, Linguagem de Definição de Dados; a Data Control Language (DCL), em português, Linguagem de Controle de Dados; e a Data Transaction Language (DTL), em português, Linguagem de Transação de Dados.

Em Banco de Dados MySQL e para qualquer distribuição de Banco de Dados, é extremamente necessário o conhecimento dos conceitos abordados, pois, para a aplicabilidade, é necessário concretizar as teorias. Aqui, nosso objetivo é demonstrar na prática os conceitos abordados sobre DQL, parte dos comandos DML, além dos comandos DTL.

Esperamos que o conteúdo e os assuntos abordados neste tema sejam de muito proveito em sua vida profissional, pois o mundo gira em torno da teoria e da prática, ambos, necessariamente, têm que trabalhar lado a lado, a fim de se obter uma plena excelência sobre o produto desenvolvido. Os melhores sempre são aqueles que têm pleno conhecimento naquilo que fazem. Bom estudo!

DESENVOLVA SEU POTENCIAL

EXTRAIR DADOS DE UMA TABELA

Caro(a) acadêmico(a), compreendemos que as informações ficam armazenadas junto ao Banco de Dados. Devemos saber como efetuar as consultas destas informações para, assim, manipulá-las. Para extrair os dados de uma tabela, ela deve

ser consultada pelo SGBD e deverá conter dados. Dessa forma, iremos trabalhar aqui com as informações utilizadas nos temas 7 e 8. Primeiramente, vamos lembrar a sintaxe do comando:

```
Select <lista de colunas>
From <tabela>
```

Nossa primeira consulta será na tabela de pessoa, da qual iremos trazer todas as colunas. Na linguagem SQL, existe um comando específico para retornar todos os dados da coluna. Esse comando é dado por meio do “*”, que é uma forma abreviada de dizer “todas as colunas”.

```
Select *
From pessoa;
```

Na Figura 1, podemos observar como a saída de nossa consulta deverá ser:

	id_pessoa	nome	sobrenome	data_nascimento
▶	1	João	Pereira	1983-09-15

Figura 1 - Saída da consulta / Fonte: o autor

Para que seja possível trabalharmos com os próximos comandos da linguagem SQL, será necessário mais de um registro presente na tabela de pessoa. Dessa forma, sugerimos que utilize o comando *insert* para efetuar a inserção de mais dados nessa tabela. Dando continuidade em nosso conteúdo, sugerimos que faça a criação das tabelas que faltam em nossa base de dados seguindo o diagrama:

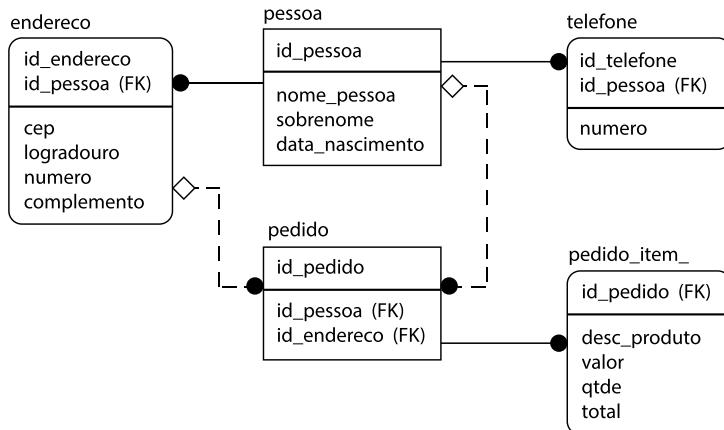


Figura 2 - Tabelas a serem criadas / Fonte: o autor

Com as tabelas já criadas sugerimos que insiram 5 produtos diferentes na tabela de pedido_item_. Lembrando que os 5 registros deverão sempre conter o mesmo id_pessoa. Agora, vamos trabalhar com funções no *select*.

Segundo Silberschatz, Korth e Sudarshan (1999), Funções agregadas são funções que tomam uma coleção (um conjunto ou subconjuntos) de valores de entrada, retornando apenas um valor simples. A linguagem SQL oferece 5 funções agregadas:

- Média (average): avg
- Mínimo (minimum): min
- Máximo (maximum): max
- Total (*sum*): sum
- Contagem (count): count

Para a entrada das funções *sum* e *avg*, os dados necessariamente precisam ser numéricos. Entretanto, as outras funções podem ser operadas com dados do tipo não numéricos, como as strings e seus semelhantes.

Dessa maneira vamos exemplificar a função *sum* com o campo total da tabela pedido_item_:

```

select sum(total)
from universidade.pedido_item

```

Nosso resultado deverá apresentar apenas um registro com a soma dos 5 registros desse campo na tabela.

The screenshot shows a 'Result Grid' window with one row. The first column is empty, and the second column contains the value 'sum(total)'. Below it, another row shows '230.00' with a blue background, indicating it is the selected result.

	sum(total)
▶	230.00

Figura 3 - Soma dos 5 registros / Fonte: o autor

Funciona da mesma maneira para a função avg:

```
select avg(total)
from universidade.pedido_item
```

Nesse caso, nosso resultado será a média entre os 5 registros da tabela. Quando desejamos encontrar o número de registros que um *select* irá nos retornar utilizamos a função count. A notação para esta função sem SQL é count(*), como no exemplo a seguir:

```
select count(*)
from universidade.pedido_item
```

O resultado a ser apresentado deverá ser apenas um registro contendo a quantidade dos registros:

The screenshot shows a 'Result Grid' window with one row. The first column is empty, and the second column contains the value 'count(*)'. Below it, another row shows '5' with a blue background, indicating it is the selected result.

	count(*)
▶	5

Figura 4 - Quantidade de registros / Fonte: o autor

Em nosso exemplo, o resultado foi 5, pois só temos 5 registros inseridos na tabela.

A função min retorna o menor valor de uma coluna em um grupo de linhas. Podemos utilizá-la para colunas do tipo data ou alfanuméricas. Para saber o preço de venda mais alto do pedido, execute o comando a seguir:

```
select avg(total)  
from universidade.pedido_item
```

Já a função max retorna o maior valor de uma coluna em um grupo de linhas. Igualmente ao min, pode-se utilizá-la para colunas do tipo data ou alfanuméricas. Para saber qual é o produto mais caro do pedido, execute o seguinte comando:

```
select max(total)  
from unicesumar.pedido_item
```

Assim, encerramos as principais funções agregadas da linguagem SQL. Essas funções são de uso diário na vida de um programador, pois, além de simples, elas são muito úteis.

APROFUNDANDO

Todo Banco de Dados possui alguma linguagem na qual os comandos devem ser enviados por meio de alguma ferramenta console. Geralmente, essa linguagem conta com elementos padrão do SQL e também com outros elementos adicionais que são específicos do Banco de Dados. De qualquer forma, sempre que um comando não consegue ser executado ou algum outro tipo de problema ocorre, é de responsabilidade do SGBD retornar uma mensagem de erro junto com o código.

Geralmente, essa mensagem de erro está em inglês, apesar de existirem alguns trabalhos de tradução para o português. Além disso, todas as mensagens de erro devem estar devidamente descritas e apresentadas na documentação oficial do Banco de Dados. Para ler o artigo na íntegra, acesse o QR Code.

Fonte: PICHILIANI, M. **Quais são os erros mais comuns em bancos de dados?** 2013. Disponível em: <http://imasters.com.br/banco-de-dados/quais-sao-os-erros-mais-comuns-em-bancos-de-dados/?trace=-1519021197&source=single>. Acesso em: 6 jan. 2023.

AGRUPANDO A EXIBIÇÃO DOS DADOS

Com a utilização do *group by*, é possível efetuar o agrupamento de diversos registros baseados em uma ou mais colunas de uma tabela. Por exemplo, os produtos da tabela pedido_item podem ser agrupados pelo valor (maior valor, menor valor), pelo qtde etc. É importante frisar que a cláusula *group by* é comumente em conjunto com as funções agregadas.

O *group by* é responsável por determinar em quais grupos devem ser colocadas as linhas de saída. Caso nosso *select* tenha funções agregadas, a cláusula *group by* realiza um cálculo a fim de chegar ao valor *sumário* para cada um dos grupos.

Para colocarmos essa cláusula, devemos estar diante de uma das seguintes situações: ou a expressão *group by* deve ser correspondente à expressão da lista de seleção ou cada uma das colunas presentes em uma expressão não agregada na lista de seleção deve ser adicionada à lista de *group by*. Observe o exemplo a seguir:

```
select id_pedido,
       sum(valor)
  from universidade.pedido_item
 group by id_pedido
```

O *select* nos traz a soma da coluna valor agrupador pelo id_pedido, conforme explicação anterior. Para exemplificar esta consulta, foi inserido mais um pedido na tabela de pedido e mais alguns itens relacionados com o segundo pedido.

The screenshot shows a software interface for viewing database results. At the top, there are buttons for 'Result Grid' and 'Filter Rows'. Below is a table with two rows of data:

	id_pedido	sum(valor)
▶	1	46.00
	2	3.00

Figura 5 - Novo pedido / Fonte: o autor

ORDENAÇÃO NA EXIBIÇÃO DOS DADOS

Segundo Silberschatz, Korth e Sudarshan (1999), a linguagem SQL oferece ao usuário um controle sobre a ordem que os dados são retornados para o usuário. A Cláusula `order by` faz com que os registros do resultado de uma consulta apareçam na ordem classificada. Para listar por ordem alfabética todos os produtos da tabela `produto_item`, devemos escrever da seguinte maneira:

```
Select desc_produto, id_pedido  
From pedido_item  
Order by desc_produto
```

Por padrão, a cláusula `order by` lista os resultados em forma crescente. Para que seja possível especificar a ordem de retorno, podemos especificar `desc`, para decrescente, ou `asc`, para a ordem crescente dos dados. Além disso, a ordenação pode ser feita por mais de 1 campo da consulta. Vamos supor que precisamos ordenar uma consulta por mais de um campo:

```
Select *  
From pedido_item  
Order by desc_pedido, valor
```

Dessa forma, primeiro, nossa consulta irá ser ordenada pela descrição do produto e, posteriormente, irá ordenar o segundo campo, sempre respeitando a ordenação anterior. Vale lembrar que, como a classificação de um grande número de registros pode ser demorada, esse tipo deverá ser feito somente quando necessário.

CRIANDO CONSULTAS COM FILTROS ESPECÍFICOS

Vamos ilustrar o uso da cláusula *where* na SQL. Consideremos, então, a seguinte consulta, “encontre todos os registros no pedido os quais o valor é maior que 20”:

```
Select *  
From pedido_item  
Where valor > 20
```

Na linguagem SQL, utilizamos *and*, *or* e *not*, ao invés de utilizarmos notações matemáticas e \neg na cláusula *where*. Os operandos dos conectivos lógicos podem ser expressões envolvendo os seguintes operadores de comparação $<$, \leq , $>$, \geq , $=$ e \neq . Essa linguagem permite operadores que comparam strings, expressões aritméticas além de comparativos entre datas.

A linguagem SQL possui um operador de comparação *between*, que tem o intuito de simplificar a cláusula *where*. Seu intuito é encontrar um valor que seja menor ou igual a algum valor e maior ou igual a um outro valor. No caso, se quisermos encontrar os produtos que estão entre 10 e 20, devemos escrever a consulta da forma apresentada a seguir:

```
Select *  
From pedido_item  
Where valor between 10 and 20
```

Ao invés de:

```
Select *  
From pedido_item  
Where valor >= 10 and valor <= 20
```

Da mesma forma, também é possível utilizar o comando de comparação *not between*.

VALORES NULOS

Podemos realizar o uso de valores nulos para denotar a ausência de informações nos registros. Na SQL, utiliza-se a palavra *null* para testar a presença de um valor nulo. Por exemplo, para buscar os registros que não possuem valor, devemos escrever a seguinte consulta:

```
Select *
From pedido_item
Where valor is null
```

Também podemos utilizar o predicado *is not null*, que testa a ausência de um valor nulo no campo.

RENAME

Um grande mecanismo da linguagem SQL, muito utilizado por todos, é a função de renomear um campo ou tabela da consulta, empregando a utilização da cláusula *AS* da seguinte forma:

```
Nome_do_campo_ou_tabela as nome_novo
```

A utilização dessa cláusula se dá tanto no *select*, quanto no *from* das consultas. Consideremos a seguinte consulta:

```
Select pedido_item.desc_produto,  
      Pedido_item.valor,  
      pedido_item.qtde  
From  pedido_item
```

Agora utilizando a função de rename.

```
Select pd.desc_produto,  
      pd.valor,  
      pd.qtde  
From  pedido_item as pd
```

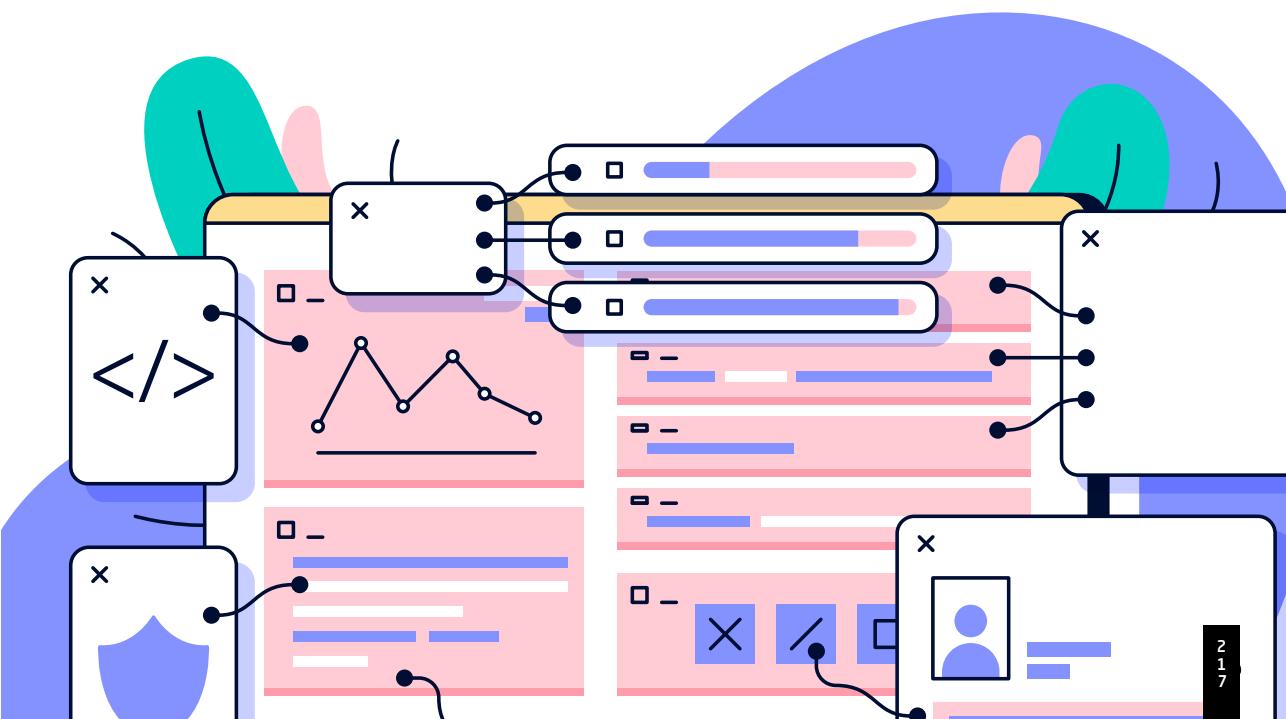
Dessa maneira, rebatizam o nome da tabela, para que a consulta fique mais simples. A utilização na cláusula *from* é logo depois do nome da tabela a qual se deseja a associação, com a palavra *as* entre eles, como no exemplo anterior. Vale lembrar que a utilização do *as* é opcional, pois o próprio SGBD subentende essa utilização. Assim como em muitos livros, aqui iremos tratar esse *rename* como a criação de um *alias* para as tabelas ou colunas.

CONSULTA UTILIZANDO MAIS DE UMA TABELA

Para que seja possível efetuar uma consulta utilizando mais de uma tabela, é necessário trabalharmos com a cláusula *from* de nosso *select*. Veja no exemplo a seguir, em que estamos acrescentando uma tabela no *from*, “Tabela2”, lembrando que para essa prática é necessário “ligarmos” as tabelas com a cláusula *where* em suas respectivas chaves primárias e estrangeiras.

```
SELECT  
    Tabela1.coluna1,  
    Tabela1.coluna2,  
    Tabela2.coluna1,  
    Tabela2.coluna2  
FROM Tabela1, Tabela2  
WHERE Tabela1.chave_primaria = Tabela2.chave_estrangeira
```

Vale lembrar que a utilização de *alias* é opcional antes do nome das colunas. Contudo, essa é uma prática muito utilizada, facilitando assim o entendimento e uma possível manutenção no código. A utilização do *alias* antes do nome do campo se torna obrigatória quando temos o mesmo campo em mais de uma tabela de nossa consulta. Geralmente, isso acontece muito com as chaves primárias e estrangeiras, sendo assim, nesse caso, é obrigatório que se indique de qual tabela deverá retornar o dado. Considere que informar em qual tabela está a coluna facilita o trabalho do Banco de Dados. Essa prática leva a maior agilidade na recuperação da informação. A união regular se dá pela união na cláusula *where* da chave primária com a chave estrangeira das tabelas.



CONSULTAS COM SUBQUERIES

Segundo Silberschatz, Korth e Sudarshan (1999), a SQL fornece um mecanismo para aninhar subconsultas. Uma subconsulta é uma expressão *select - from - where* que é utilizada dentro de outra consulta. Esse uso é muito comum para realizar teste de participação de conjuntos, fazer comparações e determinar a cardinalidade dos conjuntos.

O conectivo *IN* testa um conjunto de valores produzidos pelo *select* na cláusula *where*, ou, ainda, podemos trabalhar com o *NOT IN*, que testa a ausência de um conjunto de valores.

Para exemplificar essa utilização, vamos pensar em um *select*, que retorna todos os pedidos dos clientes cadastrados em nossa base. Primeiro, vamos encontrar os identificadores da tabela de pessoa:

```
Select id_pessoa  
From pessoa
```

Depois de encontrar os dados da pessoa, vamos aninhar essa consulta com a dos pedidos. Vale lembrar que a subconsulta sempre deve estar entre parênteses.

```
Select *  
From pedido  
Where id_pessoa in (Select id_pessoa  
                      From pessoa)
```

Selects que incluem uma subconsulta normalmente têm um destes formatos:

- *WHERE expressão [NOT] IN (subconsulta)*
- *WHERE expressão operador de comparação [ANY | ALL] (subconsulta)*
- *WHERE [NOT] EXISTS (subconsulta)*

TESTE DE RELAÇÕES VAZIAS

A função *Exists* e *Not Exists* da SQL é usada para verificar se o resultado de uma consulta aninhada correlacionada é vazio (não contém nenhum registro) ou não. A construção do *exists* retorna um valor *true* se a subconsulta não é vazia. Pensando dessa maneira, vamos escrever um *select* que nos traga somente os dados das pessoas que possuem pedido.

```
Select pes.*  
From pessoa pes  
Where exists (select ped.*  
               From pedido ped  
               Where ped.id_pessoa = pes.id_pessoa)
```

Da mesma maneira, podemos testar a existência dos registros utilizando o *not exists*. De acordo com o exemplo, se utilizamos o *not exists*, iremos trazer os dados de todas as pessoas que não possuem nenhum pedido.

```
Select pes.*  
From pessoa pes  
Where not exists (select ped.*  
                   From pedido ped  
                   Where ped.id_pessoa = pes.id_pessoa)
```

ALTERANDO OS DADOS COM *UPDATE*

Em muitas situações, temos que alterar um dado de uma tabela, sem alterar todos os seus valores. A instrução *UPDATE* deve ser utilizada nesse caso. Bem parecida com a instrução de *insert* e *delete*, podem ser selecionados os campos que devem ser atualizados. Supondo que desejamos alterar o valor dos produtos em 10%, deveremos escrever o seguinte comando:

```
Update pedido_item  
Set valor = valor * 1.1
```

O comando anterior é aplicado uma vez em cada registro da tabela de pedido_item. Caso desejarmos, por exemplo, aumentar em 10% o valor dos produtos que custam mais de R\$10, deveríamos escrever o seguinte comando:

```
Update pedido_item  
Set valor = valor * 1.1  
Where valor >=10
```

A cláusula *where* da instrução *update* é muito semelhante à instrução *where* do *select*, incluindo *select* aninhado. Como no *insert* e *delete*, um *select* pode referenciar uma relação a ser atualizada do *update*. Pensando dessa forma, vamos escrever um *update* que atualiza o valor dos produtos que estão acima da média em 10%.

```
Update pedido_item  
Set valor = valor * 1.1  
Where valor > (select avg(valor)  
From pedido_item)
```

O *update* fornece uma instrução chamada *case*, que nos dá a opção de fazer duas atualizações com uma única instrução *update*. Em uma forma geral, a instrução *case* é a seguinte:

```
Case  
    When predicado1 then  
        Resultado1  
    When predicado12 then  
        Resultado2  
    Else  
        resultado0  
End
```

Para atualizarmos o valor dos produtos menores que R\$10,00 em 10% e os produtos maiores que R\$10,00 em 5%, devemos escrever o seguinte comando:

```
Update pedido_item  
Set valor = case  
    When valor <=10 then  
        Valor = valor * 1,1  
    Else  
        Valor = valor * 1,05  
End
```



REMOVENDO OS DADOS COM **DELETE**

Uma remoção de dados de uma tabela, às vezes, é expressa do mesmo modo que uma consulta, podendo serem removidos apenas registros completos, não podendo, assim, remover o conteúdo de um campo. Podemos expressar uma remoção da seguinte maneira:

```
Delete from nome_da_tabela  
Where condição
```

Vale lembrar que o comando *delete* trabalha apenas com uma relação. Caso seja necessário remover os dados de diversas tabelas, é necessário utilizar um comando de *delete* para cada uma. A cláusula *where* pode ser tão complexa quanto a cláusula *where* de um *select*, porém, pensando em simplicidade, podemos também executar o comando *delete* sem o uso da cláusula *where* e, assim, todos os registros da tabela em questão serão apagados. Exemplo:

```
Delete from pedido_item
```

No exemplo anterior, estamos apagando todos os itens registrados na tabela de *pedido_item*.

Apresentamos aqui uma série de exemplos de remoção utilizando a SQL: Deleta os registros, desde que o valor seja maior que 10:

```
Delete from pedido_item where valor > 10
```

Apaga os registros que tenham o valor de produto acima da média:

```
Delete from pedido_item where valor > (select avg(valor) from pedido_item)
```

ROLLBACK E COMMIT

Segundo Silberschatz, Korth e Sudarshan (1999), uma transação consiste em uma sequência de instruções de consulta ou de atualização. O padrão SQL especifica que uma transação inicia implicitamente quando uma instrução SQL é executada, em que uma das seguintes instruções precisa finalizar a transação:

- **Commit:** confirma a transação atual, ou seja, torna as atualizações realizadas pela transação permanentes no Banco de Dados. Após a confirmação da transação, uma nova transação é iniciada.



- **Rollback:** faz com que a transação atual seja revertida, ou seja, ele desfaz todas as atualizações realizadas pela instrução SQL na transação. Portanto, o estado do Banco de Dados é restaurado para como era antes da primeira instrução da transação ser executada.

TRUNCATE E DROP

Para que seja possível remover uma tabela de um Banco de Dados, utilizamos o comando *drop table*. O comando *drop table* exclui do Banco de Dados a tabela e todas as informações nela contida. Esse comando é muito simples. Segue a sintaxe:

```
Drop table nome_da_tabela
```

O comando *TRUNCATE* é responsável por limpar os registros de uma tabela e fará isso de uma forma mais rápida que o comando *DELETE*. Esse comando é mais rápido que o comando *delete*, pois ele não faz uma cópia dos dados. Assim, não se tem o comando de *rollback*. O comando *TRUNCATE* é um comando DDL, enquanto o *DELETE* é um comando DML. Observe as principais diferenças entre o *DELETE* e o *TRUNCATE*:

1. *TRUNCATE* é um comando de Linguagem de Definição de Dados, enquanto *DELETE* é de Manipulação de Dados.
2. O Comando *TRUNCATE* não conta com a função *RollBack*, ao contrário do *DELETE*.
3. Uma *TRIGGER* não é disparada quando utilizamos o *TRUNCATE*. Com o comando *DELETE*, se existir ela será disparada.
4. O comando *TRUNCATE* apaga todos os dados da tabela, enquanto podemos associar o *DELETE* a condições (cláusula *WHERE*) (LEITÃO, 2015).



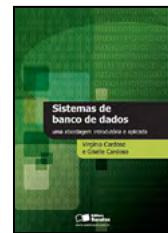
INDICAÇÃO DE LIVRO

Sistemas de Banco de Dados (2012).

Editora: Saraiva

Autor: Virgínia M. Cardoso e Giselle Cristina Cardoso

Sinopse: em texto introdutório, as autoras apresentam tópicos que levam o aluno a conhecer o universo do Banco de Dados, apresentando como projetar, construir e popular um Banco de Dados por meio de exemplos didáticos e práticos. De forma a facilitar o aprendizado, este texto proporciona ferramentas para que o aluno seja capaz de desenvolver de forma independente um Banco de Dados com armazenamento seguro e que ofereça uma pesquisa rápida e eficaz.



Pensando dessa forma utilizamos o comando *drop*, para apagar as tabelas do Banco de Dados juntamente com os seus respectivos conteúdos. Já o comando *truncate*, por sua vez, para limpar os dados de uma tabela.

NOVOS DESAFIOS

Prezado(a) acadêmico(a), após conhecermos os comandos básicos da linguagem SQL, sendo os de Linguagem de Manipulação de Dados (DML), Linguagem de Definição de Dados (DDL) e Linguagem de Controle de Dados (DCL), entre outros apresentados na unidade anterior, apresentamos por meio deste tema alguns conceitos e comandos mais avançados sobre o SQL.

Foram apresentados comandos e formas mais avançadas de seleção de dados em uma tabela. Alguns que facilitam a manipulação da informação e ajudam a refinar nossas buscas, tornando a consulta mais eficaz. Por meio dessas consultas, a manipulação de dados fica mais clara. Podemos citar como exemplo os comandos de contagem, soma, ordenação de apresentação dos dados (crescente ou decrescente) e a cláusula *Where*, que é fundamental em várias consultas ao Banco de Dados.

Ao falarmos de seleção de dados, nos deparamos em casos que precisamos manipular mais do que uma tabela. Trabalhamos, também, neste tema, essa questão, que pode nos ajudar em nosso processo de desenvolvimento de software.

Nesse sentido, aprendemos duas formas de efetuar por meio de uma consulta em várias tabelas ou *subqueries*.

Entendemos que, em um Banco de Dados, as informações podem ser modificadas a qualquer momento. Pensando nisso, conhecemos algumas formas de efetuar a alteração dos dados por meio do comando *UPDATE*, além de entendermos a função dos comandos *ROLLBACK* e *COMMIT*, esses sendo úteis em casos de situação efetuadas de modo errado no Banco de Dados.

Por fim, conhecemos os comandos *TRUNCATE*, *DELETE* e *DROP*. Todos têm o mesmo princípio, limpar os dados da tabela, cada um com sua particularidade. São comandos que podem ser perigosos se não forem manipulados de modo adequado.

Chegamos ao fim deste conteúdo, esperamos que esse tema tenha contribuído de forma significativa em sua aprendizagem. Prezado acadêmico(a), um forte abraço e até a próxima!

AGORA É COM VOCÊ

1. Com a utilização do group by, é possível efetuar o agrupamento de diversos registros baseados em uma ou mais colunas de uma tabela. Dessa maneira dê o exemplo de um select utilizando o group by e o porquê de sua utilização.
2. Sabemos que o truncate é responsável por limpar os registros de uma tabela, e que ele é mais rápido que o comando delete. Esse comando é mais rápido por não se tratar de um comando DML. Dessa maneira, cite as principais diferenças entre os comandos DELETE e TRUNCATE.
3. Uma subconsulta é uma expressão select - from - where que é utilizada dentro de outra consulta. Esse uso é muito comum para se realizar teste de participação de conjuntos, fazer comparações e determinar a cardinalidade dos conjuntos. O conectivo IN testa um conjunto de valores produzidos pelo select na cláusula where, ou ainda podemos trabalhar com o NOT IN que testa a ausência de um conjunto de valores. Exemplifique a utilização de subqueries de acordo com o que estudamos.

REFERÊNCIAS

UNIDADE 1

Tema 1

AMADEU, C. V. **Banco de dados**. São Paulo: Person Education Brasil, 2015.

DATE, C. J. **Introdução a sistemas de banco de dados**. 8. ed. Rio de Janeiro: Elsevier – Campus, 2003.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados**. São Paulo: Pearson Education, 2005.

HEUSER, C. A. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

ORACLE CORPORATION. **Profit**: the executive's guide to oracle applications, [S.I.], p. 26-33, 2007. Disponível em: <http://www.oracle.com/us/corporate/profit/p27anniv-timeline-151918.pdf>. Acesso em: 31 nov. 2016.

PIMENTEL, D.; FONTENELLE, R. Licenças. **O sistema operacional GNU**, 12 abr. 2022. Disponível em: <http://www.gnu.org/licenses/licenses.html>. Acesso em: 31 nov. 2016.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de banco de dados**. 5. ed. Rio de Janeiro: Elsevier-Campus, 2006.

Disponível em: http://www.feg.unesp.br/~saad/mysql/manual_pt.pdf . Acesso em: 31 nov. 2016.

Disponível em: <https://www.microsoft.com/pt-br/server-cloud/products/sql-server-editions/overview.aspx> . Acesso em: 31 nov. 2016.

Disponível em: <https://www.portaleducacao.com.br/informatica/artigos/62872/ransomware-o-sequestrador-de-dados> . Acesso em: 01 dez. 2016.

Tema 2

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**: implementação em SQL, PL/SQL e Oracle 11g. São Paulo: Pearson Education do Brasil, 2013.

CARDOSO, V.; CARDOSO, G. **Sistemas de banco de dados**: uma abordagem introdutória e aplicada. São Paulo: Saraiva, 2012

HEUSER, C. A. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

YANAGI, E. **Banco de dados**. Maringá: UniCesumar, 2012.

REFERÊNCIAS

Tema 3

NUVENS, E. Google: história, curiosidades e tudo que você precisa saber sobre a empresa. **Olhar digital**, 18 dez. 2018. Disponível em: <https://olhardigital.com.br/2018/12/18/tira-duvidas/google-historia-curiosidades-e-tudo-que-voce-precisa-saber-sobre-o-buscador/>. Acesso em: 03 jan 2023.

CARDOSO, V.; CARDOSO, G. **Sistema de banco de dados**: uma abordagem introdutória e aplicada. São Paulo: Saraiva, 2012.

CHEN, P. **Gerenciando banco de dados**: a abordagem entidade-relacionamento para projeto lógico. São Paulo: McGraw-Hill, 1990.3

UNIDADE 2

Tema 4

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de banco de dados**. São Paulo: Makron Books do Brasil, 1999.

CEZAR Taurion, da IBM, fala das vantagens da computação em nuvem. [S. l.: s. n.], 2010. 1 vídeo (7 min.). Publicado pelo canal CDTV. Disponível em: <https://www.youtube.com/watch?v=HJre77TPpSw>. Acesso em: 03 jan. 2023.

Tema 5

WATSON, J.; RAMKASS, R. **OCA Oracle Database 11g**: fundamentos I ao SQL. Rio de Janeiro: Alta Books, 2008.

Tema 6

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. São Paulo: Pearson Addison Wesley, 2011.

REFERÊNCIAS

UNIDADE 3

Tema 7

HEUSER, C. A. **Projeto de banco de dados**: vol. 4. 6. ed. Porto Alegre: Bookman, 2008.

PASSOS, T. Realizando duas ou mais consultas com UNION e UNION ALL no MySQL. **Tiago Passos**, 20 jun. 2010. Disponível em: <http://blog.tiagopassos.com/author/admin/page/37>. Acesso em: 18. nov. 2016.

Tema 8

CARVALHO, P. F. Linguagem SQL. **DocPlayer**, 2016. Disponível em: <http://docplayer.com.br/16602595-Linguagem-sql-dml-linguagem-de-manipulacao-de-dados.html>. Acesso em: 04 jan. 2023.

COMPUTAÇÃO nas nuvens: o que é isto. **Get.commerce**, 29 jul. 2015. Disponível em: <https://rswa.com.br/2015/07/29/o-que-e-computacao-nas-nuvens/>. Acesso em: 04 jan. 2023.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de banco de dados**. 3. ed. São Paulo: Makron Books do Brasil, 1999.

WATSON, J.; RAMKLASS, R. **Fundamentos I SQL**: OCA Oracle Database 11g. Rio de Janeiro: Alta Books, 2012.

Tema 9

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 3 ed. São Paulo: Makron Books do Brasil, 1999

LEITAO, S. M. S. **Diferença entre os comandos TRUNCATE, DELETE e DROP**. 2015. Disponível em: <https://www.profissionaloracle.com.br/2015/08/11/diferenca-entre-os-comandos-truncate-delete-e-drop/?tmpl=compo-nent&type=raw>. Acesso em: 6 jan. 2022.

PICHILIANI, M. **Quais são os erros mais comuns em bancos de dados?** 2013. Disponível em: <http://imasters.com.br/banco-de-dados/quais-sao-os-erros-mais-comuns-em-bancos-de-dados/?trace=-1519021197&source=single>. Acesso em: 6 jan. 2023.

CONFIRA SUAS RESPOSTAS

UNIDADE 1

Tema 1

1. V, F, V, V, F.
2. D.
3. Quando há a necessidade do armazenamento de apenas um tipo de dado, um arquivo comum de texto pode resolver o problema, sem a necessidade de implementação de um sistema de gerenciamento destes dados, ou em casos onde os dados armazenados sejam extremamente simples como na configuração e um maquinário.

Uma combinação destes tipos de sistemas de armazenamento de dados em um sistema pode ocorrer devido a diferentes formas de tratamento de dados, sendo por exemplo possível a existência de arquivos de texto contendo entradas de dados vindos de um equipamento ser um SGBD que serve para alimentar um SGBD mais complexo como pode ocorrer em sistemas de monitoramento de maquinário em uma indústria, por exemplo.

Tema 2

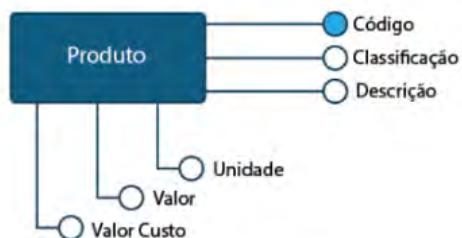
1. C
2. Entidades: podemos considerar sendo as Tabelas do Banco de Dados
Atributos: são os dados da tabela ou chamado de Colunas da Tabela.
3.
 - a) Pedido, Produtos e Cliente.
 - b) Professor, Disciplina e Curso.
 - c) Livros e Seções.
 - d) Médicos, Tratamentos e Pacientes.

Tema 3

1. Entidades Concretas são entendidas como objetos do mundo real que podem ser separadas e distinguíveis de outro objeto. Já as Entidades Abstratas são aquelas que não temos de maneira tangível (intangível).

CONFIRA SUAS RESPOSTAS

2.



3.

a)

Atendente

Matrícula

Aluno

Curso

b)

Secretária

Agenda ou
Atendimento

Pacientes

Médico

c)

Produtos

Vendas

Clientes

CONFIRA SUAS RESPOSTAS

UNIDADE 2

Tema 4

1.

```
CREATE TABLE aluno (
    id INT PRIMARY KEY,
    nome VARCHAR(30),
    sobrenome VARCHAR(30),
    ra DECIMAL(8),
    email VARCHAR(30)
);
CREATE TABLE professor (
    id INT PRIMARY KEY,
    nome VARCHAR(30),
    sobrenome VARCHAR(30),
    titulacao VARCHAR(30)
);
CREATE TABLE curso (
    id INT PRIMARY KEY,
    nome VARCHAR(30),
    ano DECIMAL(4)
);
CREATE TABLE matricula (
    curso_fk INT NOT NULL,
    aluno_fk INT NOT NULL,
    PRIMARY KEY (curso_fk, aluno_fk),
    FOREIGN KEY (curso_fk) REFERENCES curso(id),
    FOREIGN KEY (aluno_fk) REFERENCES aluno(id)
);
CREATE TABLE disciplina (
    id INT PRIMARY KEY,
    nome VARCHAR(30),
    curso_fk INT NOT NULL,
    professor_fk INT NOT NULL,
    FOREIGN KEY (curso_fk) REFERENCES curso(id),
    FOREIGN KEY (professor_fk) REFERENCES professor(id)
);
```

2.

- a) SELECT aluno.nome FROM aluno, curso, matricula WHERE curso.nome="Banco de Dados" AND curso.id=matricula.curso_fk AND matricula.aluno_fk=aluno.id;

CONFIRA SUAS RESPOSTAS

- b) SELECT professor.titulacao FROM professor, disciplina WHERE disciplina.nome="SQL" AND disciplina.professor_fk=professor.id;
- c) Select aluno.nome, aluno.sobrenome, aluno.ra from aluno, matricula, curso, disciplina, professor where matricula.aluno_fk=aluno.id AND matricula.curso_fk=curso.id AND disciplina.curso_fk=curso.id AND disciplina.professor_fk=professor.id AND professor.nome='Edson';
- d) SELECT id, nome, ano FROM curso WHERE ano>1990;
3. Exemplos de comandos para esta atividade, lembrando que podemos gerar inúmeros comandos, INSERT para ter maior quantidade de dados para manipular com os comandos UPDATE e DELETE:
INSERT INTO aluno (id, nome, sobrenome, ra, email) VALUES ("1", "João", "Silva", "100000", "jao@email.com");
UPDATE curso SET ano="2016" WHERE nome="Banco de Dados"
DELETE FROM professor WHERE id="5"

Tema 5

```
1.      CREATE TABLE plano (
          id INT PRIMARY KEY,
          nome VARCHAR(30),
          valor DECIMAL(7,2)
        );
        CREATE TABLE beneficiario (
          id INT PRIMARY KEY,
          nome VARCHAR(30),
          sobrenome VARCHAR(30),
          altura DECIMAL(3,2),
          plano_fk INT,
          FOREIGN KEY (plano_fk) REFERENCES plano(id)
        );
        CREATE TABLE dependente (
          id INT PRIMARY KEY,
          nome VARCHAR(30),
          sobrenome VARCHAR(30),
          beneficiario_fk INT NOT NULL,
          FOREIGN KEY (beneficiario_fk) REFERENCES beneficiario(id)
        );
```

CONFIRA SUAS RESPOSTAS

2.

- a) SELECT beneficiario.nome FROM beneficiario INNER JOIN dependente ON beneficiario.nome = dependente.nome;
- b) SELECT beneficiario.nome FROM beneficiario WHERE nome IN (SELECT nome FROM dependente WHERE beneficiario.nome = dependente.nome);
- c) SELECT beneficiario.nome FROM beneficiario WHERE EXISTS (SELECT * FROM dependente WHERE beneficiario.nome = dependente.nome);
- d) SELECT nome FROM beneficiario WHERE (SELECT MAX(altura) from dependente);

3.

- a) SELECT sobrenome, altura FROM beneficiario GROUP BY nome;
- b) SELECT beneficiario.nome FROM beneficiario WHERE (SELECT COUNT(*) FROM dependente WHERE dependente.beneficiario_fk=beneficiario.id and dependente.sobrenome=beneficiario.sobrenome GROUP BY beneficiario.nome) > 1
- c) SELECT plano.nome FROM plano WHERE (SELECT COUNT(*) FROM beneficiario WHERE beneficiario.altura > 1.75 AND beneficiario.plano_fk = plano.id);

Tema 6

- 1. C.
- 2. D.
- 3. D.

UNIDADE 3

Tema 7

- 1. A.
- 2. A.
- 3. A.

CONFIRA SUAS RESPOSTAS

Tema 8

1. D.
2. A.
3. C.

Tema 9

1. Utilizado para efetuar o agrupamento pelo id_produto da tabela de pedido:

```
select id_pedido,
       sum(valor)
  from unicesumar.pedido_item
```

2. As principais diferenças são:

TRUNCATE é um comando DDL enquanto DELETE é um comando DML.

TRUNCATE é muito mais rápido do que o DELETE.

Não existe Rollback para o comando TRUNCATE, mas para o DELETE sim. O comando TRUNCATE remove o registro permanente.

Em caso de TRUNCATE, a TRIGGER não é disparada, mas no caso do comando DELETE, existindo TRIGGER para deleção, ela é disparada.

Você não pode usar condições (cláusula WHERE) com o comando TRUNCATE. Mas com o comando DELETE, você pode escrever usando condições (cláusula WHERE).

- 3.

```
Select *
  From pessoa
 Where id_pessoa in (Select id_pessoa
                        From pedido)
```

Nesse select estamos retornando somente as pessoas que possuem pedido no Banco de Dados.

CONFIRA SUAS RESPOSTAS

CONFIRA SUAS RESPOSTAS

CONFIRA SUAS RESPOSTAS

CONFIRA SUAS RESPOSTAS
