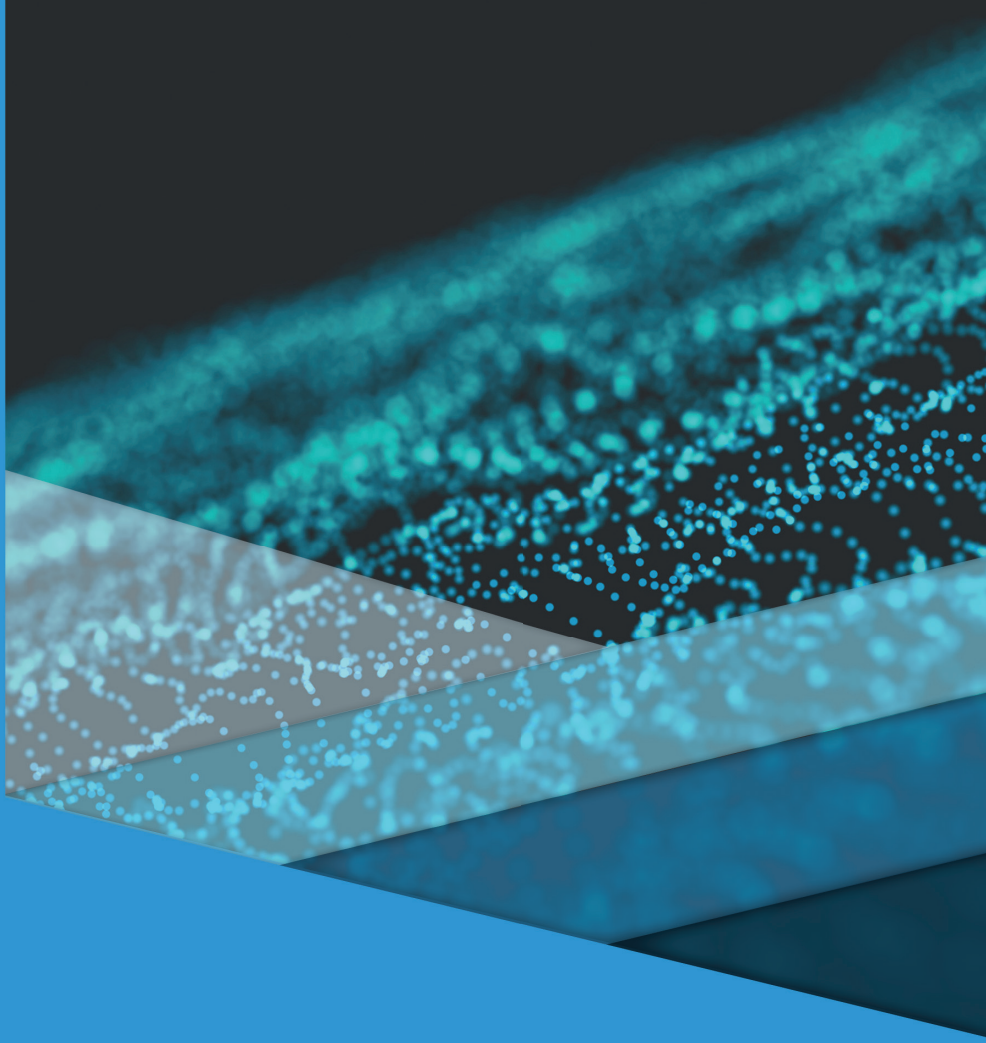


Programação em Banco de Dados

Programação em Banco de Dados

Sergio Eduardo Nunes
Ricardo Alexandre Plati Moura

Dados Internacionais de Catalogação na Publicação (CIP)

Nunes, Sergio Eduardo
N972p Programação em banco de dados / Sergio Eduardo
Nunes, Ricardo Alexandre Plati Moura. – Londrina : Editora e Distribuidora Educacional S.A., 2018.
224 p.

ISBN 978-85-522-1165-5

1. Programação. 2. Banco de dados. 3. SQL. I. Nunes, Sergio Eduardo. II. Moura, Ricardo Alexandre Plati. III. Título.

CDD 005

Thamiris Mantovani CRB-8/9491

2018
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

Palavras do autor

Atualmente, a tecnologia faz parte de nosso cotidiano e já não vivemos sem ela em vários setores, tais como comunicação, entretenimento, lazer, esportes, educação, trabalho, saúde e segurança. A tecnologia aplicada à todas estas áreas gera dados e a correlação desses dados gera informação. A quantidade de informação que temos criado, propagado e repassado mundialmente nos leva a números extraordinários, que nos permitem afirmar que qualquer organização precisa estar bem informada para tomar decisões, conhecendo o mundo ao seu redor. Mas para que tenhamos informação a nossa disposição, devemos nos lembrar que ela é gerada por meio de dados, portanto, esses dados devem ser armazenados de maneira organizada, acessível, confiável e segura. Em Programação em Bancos de Dados, vamos lidar com uma das principais competências dos profissionais do mercado de tecnologia da informação e comunicação. Entre esses profissionais, podemos citar administradores de banco de dados, administradores de dados, programadores, cientistas de dados, etc.

Você entenderá o que envolve o repositório de dados, a começar por comandos de consulta e pelas instruções SQL, que o auxiliarão a entender como um banco de dados foi estruturado, e, após isso, começará a planejar sua estrutura para o projeto Guia Turístico, criando o seu banco de dados e estrutura de tabelas. Na outra unidade, você começará a manipulação de dados, o que envolverá inserção, alteração e exclusão de dados e tabelas. Com esses conhecimentos adquiridos, você aprenderá instruções que facilitarão a criação de consultas avançadas, utilizando junções, agregação e sub consultas. Para finalizar, você utilizará recursos avançados e automação de alguns processos, com a construção de visões, controles transacionais e criação de rotinas. Você vivenciará os planejamentos necessários e colocará em prática a execução de ações para que, ao final, de cada unidade ocorra sempre a apresentação de resultados, em uma evolução crescente.

Se falhar, não desanime: isso faz parte do processo do seu aprendizado. Um passo de cada vez, sem perder o foco. Para isso será necessário tempo, dedicação e muita energia. Entre todas as

competências que você estará adquirindo, identifique aquela que você mais gosta e aprofunde-se nela, ou seja: estude sempre.

Lembre-se que o sucesso de um profissional está diretamente ligado ao prazer de praticar o seu conhecimento.

Repositório de dados

Convite ao estudo

A palavra **repositório** significa "local em que algumas coisas são guardadas, arquivadas ou colecionadas" (DICIO, 2018). Se você tiver acesso a um repositório institucional de uma determinada organização, isso significa que você estará acessando informações da instituição que foram armazenadas com o intuito de serem preservadas, arquivadas e colecionadas. Com isso em mente, quando falarmos sobre um repositório de dados, estaremos falando sobre armazenar dados. Nesta unidade, você começará a entender como criar um repositório de dados em um Sistema Gerenciador de Banco de Dados (SGBD). Para isso, será necessário abordarmos a linguagem de consulta estruturada (SQL) e as instruções necessárias para a criação do banco de dados, ou seja, seu repositório de dados. Ao final da unidade, você estará apto a criar um roteiro de instruções, na linguagem SQL (script), que criará suas tabelas e definirá sua estrutura de banco de dados.

Muitas pessoas têm as viagens como sua principal forma de diversão. Seus interesses são por grandes metrópoles, lugares exóticos, montanhas, praias, florestas e parques, em busca de conhecer novas culturas, pessoas, linguagens, comidas típicas, etc. Tendo em foco esta situação, uma famosa empresa multinacional de serviços online e softwares necessita desenvolver um novo serviço a seus seguidores: um Guia Turístico. Você, como recém-contratado dessa empresa, terá a oportunidade de desenvolver tal serviço. Para isso, terá que obter os conhecimentos sobre o Oracle® MySQL e seus editores SQL e, assim, conseguirá entender as instruções que te darão os detalhes de estruturas de banco de dados, com suas

tabelas, campos, índices e relacionamentos. Após isso, você terá condições de planejar a estrutura de banco de dados do projeto Guia Turístico e criar suas tabelas, campos e chaves primárias.

Há a necessidade de criar um Diagrama Entidade-Relacionamento (DER)? Será que a utilização de acentuação e outros caracteres especiais pode ocasionar algum problema? Todas a informações para definição da estrutura do banco de dados do projeto Guia Turístico estão disponíveis? Após apoderar-se de todas as informações, quais serão as tabelas e seus respectivos campos?

Você, agora, terá acesso ao MySQL, utilizando instruções SQL, para entender a estrutura de um banco e realizar algumas consultas. Posteriormente, entenderá os conceitos que devem ser analisados para um bom planejamento e aprenderá como criar um banco de dados com padrão internacional. Para finalizar, compreenderá como utilizar instruções para a criação e definição das características de uma tabela.

Seção 1.1

Linguagem de consulta estruturada

Diálogo aberto

A programação em banco de dados depende do conhecimento da linguagem SQL. Por isso, você precisará, inicialmente, conhecer a ferramenta MySQL e, em especial, sua versão gráfica: o MySQL Workbench. Essa poderosa ferramenta gráfica foi desenvolvida para o ambiente Windows e realiza de maneira gráfica todas as tarefas possíveis de serem realizadas na versão de linha de comando (no inglês, *Command Line Client*). Com as instruções SQL, você poderá investigar como é a definição da estrutura de qualquer banco de dados.

Você foi recém-contratado por uma empresa multinacional para desenvolver um repositório de dados para o projeto de um Guia Turístico. Como ele será desenvolvido utilizando o SGBD MySQL, nesta primeira etapa, você precisa demonstrar aos seus superiores que conhece os editores do MySQL e seu ambiente de trabalho, para poder utilizar instruções SQL e para investigar a estrutura e as consultas em um banco de dados já existente. Na instalação básica do MySQL já temos alguns bancos de dados prontos. No banco de dados **world**, você deve utilizar as instruções SQL para entender sua estrutura de tabelas, identificar suas chaves primárias e estrangeiras e determinar quais os relacionamentos entre as tabelas, podendo, então, estabelecer instruções de consultas. Como é a utilização dos editores SQL do MySQL? Como acessar um banco de dados específico? Quais instruções me auxiliarão nessas consultas à estrutura? Como determinar um relacionamento no banco de dados? É possível extrair informações desses relacionamentos?

Na presente seção, você vai conhecer as características de buscas em bancos de dados, ter uma introdução à linguagem de consulta estruturada SQL, conhecer as características dessa linguagem e vai conhecer as extensões da linguagem de consulta estruturada SQL.

Pronto para começar? Bons estudos!

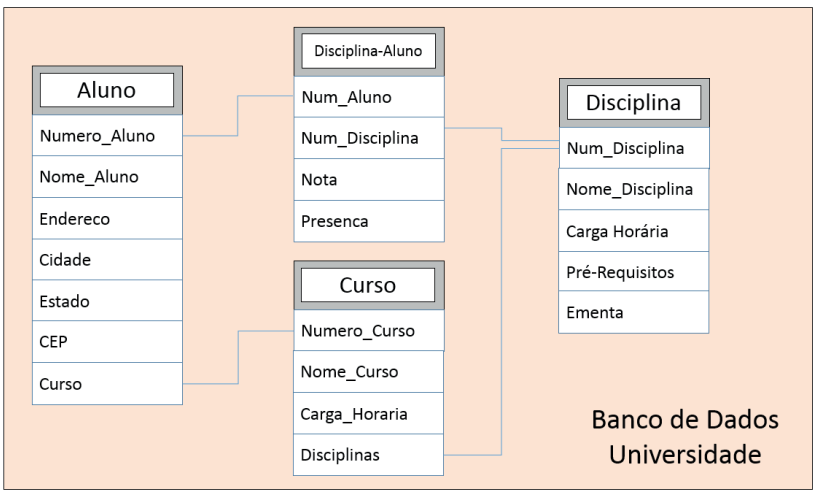
Não pode faltar

Quando falamos em Programação em Bancos de Dados, estamos tocando em dois assuntos igualmente importantes:

- **Programação** – Conjunto de técnicas para criar procedimentos estruturados que permitem que os computadores realizem tarefas desejadas (e que é o assunto principal desse livro)
- **Bancos de Dados Relacionais** – Conjunto de dados estruturados e correlacionados/correlacionáveis, organizados de forma a permitir eficiência em sua manipulação por meio de uma linguagem formal (no caso, o SQL).

Antes de entrarmos na programação propriamente dita, é importante lembrarmos alguns dos pontos mais importantes acerca dessas estruturas, os Bancos de Dados. A Figura 1.1, a seguir, ilustra um banco de dados e nos permite analisar vários de seus principais conceitos.

Figura 1.1 | Representação de um Banco de Dados



Fonte: elaborada pelo autor.

Temos, na Figura 1.1, um banco de dados chamado "Universidade", com quatro tabelas:

- **Aluno** – Tabela com uma chave primária para o número do aluno (pode ser, por exemplo, seu R.A.). Contém informações gerais sobre o aluno, inclusive o nome do curso em que ele está matriculado na instituição. Esse curso será detalhado

em outra tabela, e podemos assumir que cada aluno faz apenas um curso, estabelecendo uma relação 1:1 com a tabela Curso (a ser vista a seguir).

- **Curso** – Tabela que contém informações acerca dos vários cursos oferecidos pela universidade (exemplos: Medicina, Engenharia Elétrica, Arquitetura, etc.). Cada curso tem informações de carga horária e um conjunto de disciplinas. Cada curso terá várias disciplinas componentes e cada disciplina pode ser oferecida em mais de um curso, criando uma relação N:N entre essas tabelas.
- **Disciplina_Aluno** – conecta duas tabelas (**Disciplina** e **Aluno**), contém informações sobre as disciplinas que um aluno cursa em determinado semestre.

O banco de dados ilustrado na Figura 1.1 é um exemplo típico (ainda que simplificado) dos bancos de dados vistos em ambientes profissionais: grupos de tabelas que contém colunas nomeadas, cada uma contendo um tipo de dado. As linhas são os registros, e as tabelas podem ter informações arbitrárias, coletadas para formarem outras tabelas.

A Linguagem SQL, objeto desta disciplina, é a forma de tratarmos os vários aspectos de um banco de dados, desde sua criação e de suas tabelas componentes, inserção dos dados e, principalmente, manipulação desses dados para que sua correlação nos gere informações úteis para uso em nossos projetos.

Você poderá ver um exemplo de relações 1-1, 1-n, n-n acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u1/s1/codigo.pdf> ou pelo QR Code.

A linguagem SQL se estabeleceu como a linguagem padrão dos SGBD. Essa linguagem de consulta estruturada (SQL, do inglês *Structured* Query *Language*) tem um nome que não representa toda sua abrangência (MANZANO, 2011).

Além de instruções de consultas ao banco de dados, podemos citar instruções para:

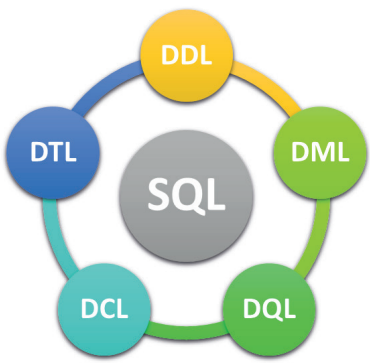
- Definir esquemas de relacionamento, excluir relações e modificar estruturas.

- Criar restrições em relacionamentos, garantindo condições específicas de integridade e proibindo qualquer violação.
- Criar visões específicas sobre determinados dados.
- Realizar consultas interativas, baseadas em álgebra relacional, podendo, até mesmo, incluir, atualizar e excluir dados.
- Determinar a segurança do ambiente com todo o controle de acesso ao banco de dados, tabelas ou campos específicos.
- Determinar todo o controle de transações, garantindo a persistência e integridade dos dados.
- Permitir utilização autônoma ou como parte de outras aplicações.

Nos seus estudos, você utilizará o SGBD MySQL. Ele tem seu código aberto (em inglês *Open Source*), é distribuído e suportado pela Oracle Corporation e trabalha com várias plataformas, tendo sido escrito em C e C++ (MANZANO, 2011). Com o MySQL, você pode definir instruções SQL incorporadas de maneira embutida ou dinâmica na maioria das linguagens de programação, como Node.js, PHP, C#, C++, Java, Android, Swift, para citar apenas algumas delas (ORACLE, 2018).

Na linguagem SQL, destacam-se cinco subconjuntos de instruções que estão representadas conforme a Figura 1.2.

Figura 1.2 | Subconjuntos de instruções da linguagem SQL



Fonte: elaborada pelo autor.

No MySQL, as instruções SQL foram implementadas também em subconjuntos, sempre seguindo os requisitos de confiabilidade, performance, integridade e disponibilidade. Você verá a seguir cada um dos subconjuntos e algumas instruções da linguagem SQL (MANZANO, 2011).

Linguagem de Definição de Dados (DDL, do inglês *Data Definition Language*)

Conjunto de instruções SQL para definição dos dados e sua estrutura.

`CREATE` – cria banco de dados, tabelas, colunas.

`DROP` – exclui banco de dados, tabelas, colunas.

`ALTER` – altera banco de dados, tabelas, colunas.

`TRUNCATE` – esvazia toda a tabela.

Linguagem de Manipulação dos Dados (DML, do inglês *Data Manipulation Language*)

Conjunto de instruções SQL para inserção e manutenção dos dados.

`INSERT` – insere dados em uma tabela.

`UPDATE` – atualiza os dados existentes em uma tabela.

`DELETE` – exclui registros de uma tabela.

Linguagem de Consulta a Dados (DQL, do inglês *Data Query Language*)

Conjunto de instruções SQL para consulta de todos os dados armazenados e suas relações, e ajuda para comandos de sintaxe.

SELECT – principal instrução de consulta do SQL.

SHOW – exibe todas as informações além dos dados (metadata).

HELP – exibe informações do manual de referência do MySQL.

Linguagem de Controle de Dados (DCL, do inglês *Data Control Language*)

Conjunto de instruções SQL para controle de autorizações de acesso e seus níveis de segurança.

`GRANT` – essa instrução concede privilégios às contas de usuário.

`REVOKE` – essa instrução permite revogar os privilégios da conta de usuário.

Linguagem de Transação de Dados (DTL, do inglês *Data Transaction Language*)

Conjunto de instruções para o controle de transações lógicas

que são agrupadas e executadas pela DML.

`START TRANSACTION` – inicia uma nova transação.

`SAVEPOINT` – identifica um determinado ponto em uma transação.

`COMMIT` – é uma instrução de entrega ao SGBD, fazendo com que todas as alterações sejam permanentes.

`ROLLBACK [TO SAVEPOINT]` – é uma instrução ao SGBD para reverter toda a transação, cancelando todas as alterações ou até determinado ponto da transação.

`RELEASE SAVEPOINT` – instrução para remoção de um `SAVEPOINT`.

Estrutura básica das Consultas SQL

Em um repositório de dados, apenas para relembrá-lo, temos a definição de tabelas de dados baseada nas seguintes premissas (MACHADO, 2014):

- Cada tabela é uma relação.
- O registro, ou seja, o conjunto de uma linha e colunas, é chamado de tupla.
- Cada campo ou coluna dessa tabela tem um nome único representando um domínio distinto.
- A ordem dos registros é irrelevante.
- Não pode ocorrer dois registros iguais.
- A ordem dos campos é irrelevante.
- Cada tabela deve ser identificada de maneira única, distinta de qualquer outra tabela do banco de dados, utilizando-se um nome próprio.

Em um SGBD relacional, teremos repositórios de dados armazenados, mas com um conjunto de tabelas que podem ser identificadas unicamente (CARDOSO, 2013).

As instruções SQL permitem que, em tabelas, os campos tenham valores nulos, para indicar que seu valor é inexistente ou desconhecido. Permitem também a instrução explícita dos campos que não poderão conter valores nulos (HEUSER, 2009).

Já vimos que na DQL há uma instrução básica, composta por três cláusulas:

- `SELECT` – identificação dos campos desejados em uma consulta.

- `FROM` – lista as tabelas que deverão ser lidas.
- `WHERE` – consiste em expressões lógicas envolvendo os campos das tabelas da cláusula `FROM`.

Por isso, você deve entender que consultas são resultados de um produto cartesiano das tabelas especificadas na cláusula `FROM`, realizadas por uma ou mais condições da cláusula `WHERE`, aplicando os resultados nos campos da cláusula `SELECT` (MACHADO, 2014).

Produto cartesiano são conjuntos distintos multiplicados por seus pares ordenados. Isso significa que, em SQL, uma consulta será sempre o resultado da multiplicação da quantidade de registros das tabelas mencionadas em sua instrução.

Por essa razão, devemos sempre estabelecer condições precisas para nossas consultas, pois podemos apresentar um produto cartesiano, ou seja, um resultado, enorme e impreciso, que determina um alto custo de processamento

Exemplo de consulta precisa:

```
SELECT nome, nascimento, cpf FROM clientes WHERE
    cpf = '12345678901';
```

Exemplo de consulta imprecisa:

```
SELECT * FROM clientes;
```

Exemplificando

Antes de qualquer consulta ou acesso no editor SQL, tenha certeza de que está acessando o banco de dados desejado. Supondo que você deseja acessar o banco chamado *"mundo"*, para isso utilize primeiramente a instrução:

```
USE mundo;
```

Essa instrução define ou altera o banco em que você está trabalhando.

Para visualizar todas as definições de uma determinada tabela, como a tabela *"cidade"*, utilize essa instrução:

```
SHOW COLUMNS FROM cidade;
```

Será exibida uma tabela contendo os nomes dos campos e suas propriedades.

A cláusula `SELECT`

O resultado de uma consulta SQL é uma tabela. Para entendermos essas consultas, vamos considerar o banco de dados "mundo".

- cidade (Id, Nome, CodigoPais, Estado, Populacao)
- pais (Codigo, Nome, Continente, Regiao, Area, Populacao)
- linguapais (CodigoPais, Lingua, Oficial, Porcentagem)

A sintaxe das instruções é explicada no manual de referências do MySQL (ORACLE, 2018). Vamos estabelecer uma consulta simples em nosso exemplo de banco de dados: "Encontre os nomes de todas as cidades na tabela cidade".

```
SELECT Nome
FROM cidade;
```

O resultado desta sintaxe será uma tabela contendo apenas um campo com o cabeçalho *Nome*. Você deve ficar atento, pois o SQL permite resultados duplicados, isso significa que, se houver na tabela *cidade* alguns registros com os mesmos conteúdos, eles serão apresentados, mas nós podemos explicitar a eliminação de qualquer duplicidade, utilizando a palavra-chave DISTINCT:

```
SELECT DISTINCT Nome
FROM cidade;
```

Você pode também fazer o contrário, especificando que você deseja que as duplicidades não sejam removidas. Sempre que for necessário, utilize o DISTINCT para remover duplicidades em suas consultas.

```
SELECT ALL Nome
FROM cidade;
```

Outra opção é utilizar "*" para indicar "*todos os campos*". Um exemplo está na sintaxe:

```
SELECT *
FROM cidade;
```

Na cláusula SELECT, você pode também incluir expressões aritméticas utilizando os operadores +, -, * e /. Como, por exemplo, na sintaxe:

```
SELECT Nome, Populacao / 2
FROM cidade;
```

Esse comando exibirá uma tabela com os campos "Nome" e "Populacao", porém o valor de "Populacao" será dividido por 2.

Além disso, existem operações com datas e outros tipos especiais, e outras funções aritméticas.

A cláusula `WHERE`

Para entendermos um pouco mais sobre a cláusula WHERE, vamos supor a seguinte consulta: "Encontre todos os nomes de cidades que tenham uma população inferior a 100000". Para isso execute a sintaxe:

```
SELECT Nome, Populacao
FROM cidade
WHERE Populacao < 100000;
```

Os conectivos `AND`, `OR` e `NOT` podem também ser utilizados como subqualificadores para o qualificador WHERE. Também pode-se utilizar operadores de comparação <, <= , >, >=, = e <>. Eles podem ser utilizados com strings e expressões aritméticas além de tipos especiais e tipos de data.

Para simplificar algumas operações utilizado o WHERE, existe o `BETWEEN`:

```
SELECT Nome, Populacao
FROM cidade
WHERE Populacao BETWEEN 90000 AND 100000;
```

Em vez de:

```
SELECT Nome, Populacao
FROM cidade
WHERE Populacao >= 90000 AND Populacao <= 100000;
```

Podemos, ainda, usar o operador de comparação `NOT BETWEEN`:

```
SELECT Nome, Populacao
FROM cidade
WHERE Populacao NOT BETWEEN 60000 AND 70000;
```

Exemplificando

Supondo que uma tabela "*cidade*" e uma tabela "*pais*" contenham 1000 registros cada. Em uma consulta SQL, o resultado de uma consulta sem condições lógicas para delimitação envolvendo

estas duas tabelas provocará a exibição de 1.000.000 de registros como resultado:

```
SELECT * FROM cidade, pais;
```

A cláusula FROM

Como já dissemos, a cláusula FROM fornece um produto cartesiano das tabelas especificadas na cláusula. Portanto, para a consulta "Encontre todas os nomes e população das cidades e línguas do seu país", temos a sintaxe:

```
SELECT cidade.Nome, cidade.Populacao,
linguapais.Linguagem
FROM cidade, pais, linguapais
WHERE cidade.CodigoPais = pais.Codigo AND
pais.Codigo = linguapais.CodigoPais;
```

Note que estamos utilizando na instrução SQL a notação nome-tabela.nome-campo. Isso deve ser feito para evitar ambiguidades, pois temos alguns campos com nomes iguais em tabelas distintas, por exemplo o campo "CodigoPais" ou "Nome".

Podemos ainda verificar a mesma consulta para as cidades que falem português, com a consulta "Encontre todos os nomes e população das cidades e dos países que falem Português", utilizando a sintaxe:

```
SELECT cidade.Nome, cidade.Populacao,
linguapais.Linguagem
FROM cidade, pais, linguapais
WHERE cidade.CodigoPais = pais.Codigo AND
pais.Codigo = linguapais.CodigoPais AND
linguaPais.Linguagem = "Português";
```

Assimile

Se pudéssemos criar uma classificação das instruções SQL mais utilizadas, as instruções de consulta (DQL) com certeza estarão em primeiro lugar. Por isso, estudar a fundo as sintaxes, as cláusulas e as finalidades de utilização destas instruções levarão você a explorar toda a potencialidade de um SGBD e seus bancos de dados relacionais.

A operação de renomeação e variáveis do registro

Os campos podem ser renomeados e, para isso, usa-se a cláusula AS, da seguinte forma:

nome-antigo **AS** nome-novo

Essa cláusula poderá ser utilizada na cláusula SELECT e FROM.

Utilizando alguns exemplos anteriores, o efeito seria:

```
SELECT Nome, Populacao as PopulacaoDaCidade
FROM cidade;
```

Teremos como resultado uma tabela com o campo *Populacao* sendo renomeado para *PopulacaoDaCidade*.

Também podemos utilizar a cláusula AS para definição de variáveis de registro, que deverão estar associadas a uma determinada tabela. Elas são definidas na cláusula FROM. Por exemplo, a consulta "Encontre todos os nomes e população das cidades e línguas do seu país" pode ser realizada com a sintaxe:

```
SELECT C.Nome, C.Populacao, L.Linguagem
FROM cidade as C, pais as P, linguapais as L
WHERE C.CodigoPais = P.Codigo AND
P.Codigo = L.CodigoPais;
```

Operações de *String* e Ordenação

O operador LIKE determina a correspondência de padrões, que são descritos usando caracteres especiais:

- Porcentagem (%): corresponde a qualquer *substring*.
- Sublinhado (_): corresponde a qualquer caractere.

Para ilustrar, considere os seguintes exemplos:

- 'Sor%' localizará qualquer *string* iniciando com "Sor".
- '%or%' localizará qualquer *string* iniciando contendo "or".
- '___' localizará qualquer *string* com exatamente três caracteres.
- '___%' localizará qualquer *string* com pelo menos três caracteres.

Um exemplo do que acabamos de falar, é a realização da consulta "Encontre os nomes de todas as cidades na tabela cidade com nomes iniciados por 'Sor'", com a sintaxe:

```
SELECT Nome
```

```
FROM cidade
WHERE Nome like 'Sor%';
```

Outra cláusula importante é a `ORDER BY`, que controla os registros em relação à sua ordenação pelo campo especificado, como na consulta "Encontre os nomes de todas as cidades na tabela cidade, ordenadas por nome":

```
SELECT Nome
FROM cidade
ORDER BY Nome;
```

Essa cláusula ordena de forma crescente, por padrão, mas você pode explicitar essa ordem utilizando `ASC` ou, ainda, ordenar de maneira decrescente, com `DESC`. Por exemplo, a sintaxe abaixo realiza a busca "Encontre os nomes de todas as cidades na tabela cidade ordenados por nome em ordem decrescente":

```
SELECT Nome
FROM cidade
ORDER BY Nome DESC;
```

Pesquise mais

Para conhecer outras instruções e aprofundar seus conhecimentos nas instruções SQL, acesse o manual do MySQL 5.7.

ORACLE CORPORATION. Chapter 13 SQL Statement Syntax. In: Oracle Corporation. **MySQL 5.7 Reference Manual**. [S.p.]. [S.l.], 12 jul. 2018. Disponível em: <https://dev.mysql.com/doc/connectors/en/connector-odbc-reference.html>. Acesso em: 12 jul. 2018.

Sem medo de errar

Você sempre deverá se lembrar de que todas as instruções SQL podem ser utilizadas em todos os ambientes do MySQL (Windows, Linux, MAC). Para iniciar, você deveria investigar como a estrutura do banco de dados "world" foi criada.

No editor SQL, primeiro você deve identificar ao MySQL qual a base padrão de trabalho:

```
USE world;
```

Após isso todas as instruções SQL serão direcionadas para o banco de dados identificado. Então já podemos investigar a estrutura desta base e, para isso, iremos executar a instrução:

```
SHOW TABLES;
```

Nesse momento, você poderá visualizar todas as tabelas que compõem o banco de dados "world". Você deve ter visto três tabelas listadas: "city", "country", "countrylanguage".

Sabendo o nome de cada tabela, você poderá olhar os campos de cada uma delas por meio da instrução:

```
SHOW COLUMNS FROM nome-tabela;
```

Vamos investigar a estrutura da tabela "city":

```
SHOW COLUMNS FROM city;
```

Você terá como resultado uma tabela com 6 colunas, cada uma exibindo uma propriedade do campo. São elas: "Field" (campo), "Type" (tipo), "Null" (pode conter valor nulo), "Key" (chave), "Default" (valor padrão), "Extra" (características especiais).

Os campos são representados pelas linhas dessa tabela. Isso significa que a tabela "city" é composta por cinco campos: *ID, Name, CountryCode, District, Population*.

Essas são as características principais que podemos extrair de uma estrutura de tabelas, e você, após examiná-las com muita atenção, conseguirá determinar a estrutura de cada tabela e como elas se relacionam. Pronto! O rascunho que dará origem ao DER já poderá ser desenhado.

Após estabelecer os relacionamentos existentes, verifique se de fato eles estão consistentes, usando a DQL com a instrução `SELECT`.

Essas análises são vitais quando um banco de dados não foi desenvolvido por você. Isso acontece muito nos ambientes profissionais, nos quais é preciso verificar qual foi o modelo conceitual que uma outra equipe elaborou para determinado banco de dados. Além disso, é necessário testar se há consistência nos relacionamentos, criando consultas simples e, gradualmente, ao passo que verifica as tabelas, com seus campos e valores armazenados, você pode certificar-se de que os relacionamentos estão de fato estabelecidos.

Como exemplo, podemos consultar:

- "Encontre os nomes de todas as cidades na tabela cidade com nomes iniciados por 'Sor'":

```
SELECT Name
FROM city
WHERE Name LIKE 'Sor%';
```

- "Encontre os nomes e a população de todas as cidades com nomes iniciados por 'Sor'":

```
SELECT Name, Population
FROM city
WHERE Name LIKE 'Sor%';
```

- "Encontre os nomes, sua população e os países em que se encontram, para todas as cidades com nomes iniciados por 'Sor'":

```
SELECT city.Name, city.Population,
country.
Name
FROM city, country
WHERE city.Name LIKE 'Sor%' AND city.
CountryCode = country.Code;
```

Quanto mais consultas com relacionamentos forem criadas, mais consistente seu banco de dados se demonstrará. Pronto! Você conseguiu demonstrar que tem o conhecimento necessário para a próxima etapa.

Avançando na prática

Análise do banco de dados **sakila**

Descrição da situação-problema

Um famoso escritório de projetos, que presta serviços às maiores empresas do estado, está passando por uma situação crítica. Muitos documentos relacionados aos projetos do último semestre, por uma falha de hardware, não estão mais disponíveis. Atualmente, há cinco bancos de dados criados no servidor de banco de dados MySQL, utilizado para desenvolvimento, com todas as respectivas estruturas. Um desses bancos é chamado de **sakila**. Você é responsável por determinar quais são as finalidades principais desse repositório de dados. Nessa análise, precisamos identificar em qual prestação de

serviço esse banco de dados pode ser utilizado e quais os módulos que, em uma suposta aplicação, você indicaria para o desenvolvimento.

Resolução da situação-problema

No editor SQL, primeiro você deve indicar ao MySQL qual a base padrão de trabalho:

```
USE sakila;
```

Após isso, todas as instruções SQL serão direcionadas para o banco de dados identificado. Em seguida, podemos investigar a estrutura desta base, executando a instrução:

```
SHOW TABLES;
```

Neste momento, você poderá visualizar todas as tabelas que compõem o banco de dados **sakila**. Sabendo o nome das tabelas, você poderá olhar os campos de cada uma delas com a instrução:

```
SHOW COLUMNS FROM nome-tabela;
```

Após essa investigação, determine os relacionamentos entre as tabelas. Sobre os módulos que você indicaria para o desenvolvimento, preste atenção especial às tabelas" store", "staff", "rental", "inventory", "customer".

Faça valer a pena

1. Em uma empresa que extremamente dependente de bancos de dados relacionais, existe um banco de dados voltado para o controle de Notas Fiscais Eletrônicas. Esse banco detém toda a estrutura para atender às necessidades e requisitos estabelecidos pela Secretaria da Fazenda do seu estado. Há uma demanda para que essa base seja adequada para a Secretaria da Fazenda de outros estados e, para isso, deverá ocorrer uma adequação: a inserção de mais 34 tabelas no banco de dados e a exclusão de 22 tabelas.

Baseado na adequação acima, pode-se afirmar que as mudanças baseadas na linguagem SQL estarão baseadas no subconjunto que contém as instruções:

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

2. O setor de Marketing de uma Organização Não Governamental (ONG) está em uma nova campanha regional para auxílio aos imigrantes da Venezuela. Ela detém um banco de dados com todas as informações daqueles que passaram legalmente pelas nossas fronteiras. Será necessário determinar quais as cidades e estados que estão recebendo todas estas pessoas, para que possam receber uma autorização de residência em nosso país, assim como restabelecer as famílias que possam estar separadas.

Para você determinar as respostas acima, incluindo Cidades, Estados e famílias, você necessitará utilizar o subconjunto com as instruções:

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

3. Atualmente, muitos livros são descartados por pessoas que não querem mais guardá-los após os lerem. Você, como empreendedor, logo viu nesse cenário um novo negócio: o empréstimo destes livros para pessoas que gostariam de lê-los. O processo de coleta dos livros começará e a divulgação abrirá a consulta à este acervo para que muitas pessoas possam retirar o livro de sua necessidade. Você necessitará organizar tudo isso com urgência, desde a recepção do livro até a consulta pelas pessoas em um website. Você já definiu toda a estrutura de seu banco de dados, com suas tabelas e relacionamentos, e recebeu uma lista de pessoas de sua cidade com várias informações

Você deverá ter uma lista de pessoas com renda inferior a R$ 500,00. A instrução SQL que determinará este resultado é:

a) `SELECT NOME, ENDERECO FROM RENDA = 500;.`
b) `SELECT NOME, IDADE FROM RENDA < 500;.`
c) `SELECT NOME, CIDADE FROM CIDADE;.`
d) `SELECT * FROM PESSOA WHERE RENDA < 500;.`
e) `SELECT * FROM PESSOA WHERE NOME LIKE "SOR%";.`

Seção 1.2

Criação de banco de dados

Diálogo aberto

Você sabia que, há muito tempo, diversas organizações esportivas demandam que as federações de cada país coletem dados acerca de todos os aspectos de uma partida de determinados esportes, e que esses dados coletados estão à disposição de cada time daquele país? Trata-se de um enorme manancial de dados. Um dado é uma quantificação de um aspecto qualquer. No futebol, por exemplo, chutes a gol ou escanteios concedidos são dados relevantes. Um goleiro que sabe que um determinado jogador cobrou o último pênalti do lado esquerdo do gol, por exemplo, tem um dado em suas mãos. Tem utilidade? Tem, mas é uma utilidade limitada, não é mesmo? E se ele juntar vários dados e descobrir que nos últimos 30 pênaltis cobrados o jogador à sua frente mandou a bola à meia altura, do lado esquerdo em 18 cobranças? Bem, aí ele já está correlacionando dados, e acaba de obter uma informação que pode ser bem mais útil que o dado isolado: se pular à meia altura para o lado esquerdo, talvez tenha mais chances de defender o pênalti. O poder dos bancos de dados é este: permitir que façamos correlações com os dados e deles extraiamos informações úteis.

Você trabalha em uma empresa que está desenvolvendo um aplicativo para o serviço de um Guia Turístico. Quando pensamos em um aplicativo como esse, entendemos que ele deve seguir a mesma organização que temos em nossa vida real, ou seja, os elementos turísticos estão localizados em cidades, e todas as cidades compõem um estado, que, por sua vez, compõem um país. O Guia Turístico a ser desenvolvido poderá apresentar informações nacionais e internacionais. Você deverá propor uma abstração dessa realidade por meio de um Diagrama Entidade-Relacionamento. Esse será o documento que, no decorrer de seu projeto, terá que ser mantido atualizado para referência de seu banco de dados. Isso significa que será um documento que deve ser alterado constantemente e, por isso, deve-se ter controle sobre ele, sendo possível identificar as suas alterações. Agora, você já

deverá preocupar-se com um fator técnico, que caracteriza nossa linguagem: o banco de dados deve utilizar caracteres acentuados, além de estar preparado para a internacionalização, e, para isso, você terá que especificar na criação do banco estas definições. Após observar esses requisitos e o entendimento de como utilizar as instruções SQL de forma correta, você estará dando os primeiros passos para começar a criar um banco de dados. Nessa fase, você ainda terá muito planejamento e rascunhos, pois é o trabalho de definição das características de seu banco de dados, mas tudo isso dará a você o conhecimento e certeza de prevenção de muitos problemas.

Para essa criação de banco de dados, você deverá apreender questões de internacionalização e verificará quais elementos na sua criação poderão limitar a utilização de seu banco de dados. Lembre-se: as propriedades de um banco de dados são de vital importância para determinar como os dados serão armazenados e devem ser estudadas por qualquer profissional de tecnologia.

Bons estudos!

Não pode faltar

Quando vamos construir uma casa, não saímos assentando alicerces e levantando paredes. O procedimento correto exige que, antes da execução, haja planejamento. Antes de levantarmos uma casa, precisamos de um projeto arquitetônico, de um documento de cálculo estrutural, de um projeto hidráulico e de um projeto elétrico. Isso tudo tem que ser muito bem pensado e esquadrinhado antes de fazermos sequer um furo no solo, ou de assentarmos um tijolo que seja.

Com bancos de dados, ocorre o mesmo: é fundamental realizarmos um planejamento sobre como deverá ser o banco de dados, quais serão suas tabelas e como serão relacionadas antes que qualquer comando de criação seja executado. Esse será nosso ponto inicial.

Planejamento de um banco de dados

O planejamento de um banco de dados não tratará somente da forma como os dados serão armazenados, mas também de como você vai definir sua estrutura, e chamamos isso de metadados. Podemos afirmar que metadados são os dados da estrutura de um banco de dados (CARDOSO; CARDOSO, 2013).

Para planejar um banco de dados, entendemos que há cinco passos principais (MACHADO, 2014):

1. Coletar informações.
2. Identificar suas principais estruturas.
3. Modelar a estrutura.
4. Identificar nas estruturas os tipos de dados.
5. Identificar quais são seus relacionamentos.

Vamos brevemente retomar cada um desses passos.

1) **Coletar informações** – Deve-se ter um entendimento do trabalho a que o seu banco de dados se destina. Ele deverá fornecer todas as informações necessárias para alcançar seu objetivo. É importante entender qual a abrangência de seu uso e o público-alvo, pois muitas informações deverão ser úteis àqueles que no futuro utilizarão seu banco de dados. Também é necessário identificar limitações e problemas, para que, na definição de sua estrutura, você possa ultrapassá-las.
2) **Identificar suas principais estruturas** – Identificar suas principais entidades, ou seja, suas tabelas, pois elas serão gerenciadas pelo banco de dados. Essas tabelas representarão algo tangível ou intangível, e isso quer dizer que podemos representar pessoas, locais, clientes, cidades, bem como uma venda, uma reclamação, uma autorização de compra, um período de pagamentos de impostos, entre outras coisas.
3) **Modelar a estrutura** – A parte da modelagem de estrutura pressupõe que as principais tabelas estejam identificadas, então faz-se necessário gerar uma documentação, que é o Diagrama Entidade-Relacionamento (DER) (HEUSER, 2009). Neste momento, você pode utilizar alguns softwares para esse desenho ou simplesmente utilizar papel e lápis. Mas lembre-se sempre que, por se tratar de um registro intelectual de sua criação, esse documento servirá para que outras pessoas entendam o que você idealizou como modelo de abstração para a resolução de um problema, e por isso ele sempre deverá estar atualizado.
4) **Identificar nas estruturas os tipos de dados** – Neste momento, você identificará detalhadamente os tipos de dados que deverão ser armazenados, como números inteiros

ou decimais, cadeia de caracteres, datas, etc. Mas podemos ter uma classificação interessante sobre essas informações e classificá-las quanto ao seu conteúdo como:

- **Dados brutos** – exemplo de colunas que armazenarão nomes, endereços, telefones, datas de nascimento, etc.
- **Dados de categorização** – como verdadeiro/falso, certo/errado, casado/solteiro, rua/avenida, gerente/colaborador, etc.
- **Dados de identificação** – esses campos terão a finalidade de criar um identificador único de um registro em suas tabelas, ou seja, ele não pode ser nulo ou repetir-se; na maioria das vezes, terá o sufixo "id" na definição do nome, como "cliente_id", "produto_id", etc.
- **Dados de relação ou referência** – são os campos responsáveis por armazenar a relação entre duas tabelas, e é através dela que o banco de dados poderá gerenciar integridades, consultas e relacionamentos.

5) **Identificar quais são seus relacionamentos** – Neste passo, você demonstrará a razão de um banco de dados relacional existir: associar dados e relacionar informações sobre várias tabelas de seu banco de dados. Você poderá fazer as mais variadas combinações, pois os mecanismos do banco de dados são desenvolvidos para desenvolver relações com definições lógicas de agregação ou desagregação de dados.

Até aqui, você deve ter notado como é importante o planejamento de um banco de dados. Antes de utilizá-lo, há a necessidade de esclarecer e executar cada um dos passos descritos. Ao final, você terá um Diagrama Entidade-Relacionamento (DER), que vai direcionar suas atividades para implementação, além de poder compartilhar conhecimentos com outras pessoas de sua equipe.

Internacionalização de um banco de dados

Quando estamos desenvolvendo um banco de dados, devemos prestar atenção na sua abrangência de utilização, pois os dados que serão armazenados deverão respeitar as regras de escrita e gramática ou representação de cada país. Os sistemas gerenciadores de bancos de dados têm algumas cláusulas especiais para lidar com

estas questões, e o MySQL não é diferente. As cláusulas são `CHARSET` e `COLLATION`. Primeiro, vamos entender a funcionalidade de cada uma delas e após isso veremos como utilizá-las.

Reflita

Você pode já ter acessado algum aplicativo ou site na internet que exibia algumas palavras com a grafia incorreta. Talvez não seja um erro de conteúdo, mas uma falha de apresentação deste conteúdo, por isso a necessidade de entendimento da utilização do `CHARSET` e `COLLATION`.

Conjunto de caracteres (`CHARSET`) e agrupamentos (`COLLATION`)

A cláusula `CHARSET`, como o próprio nome indica, designa um conjunto de símbolos e codificações e como eles são representados binariamente. Já a cláusula `COLLATION` é o conjunto de regras para comparação de caracteres em um conjunto de caracteres.

Imagine a seguinte situação: vamos supor que nosso alfabeto é composto apenas das letras A, B, a, b. Imagine as letras como símbolos e para cada símbolo vamos designar um número, conforme a Figura 1.3.

Figura 1.3 | Exemplo de um alfabeto com apenas quatro letras

Fonte: elaborada pelo autor.

Esse é o nosso `CHARSET`. Mas, suponha que você tenha que comparar dois valores de caracteres, A e B. Podemos ver que o número associado ao caractere A é o número zero (0), e ao B é um (1). Aqui podemos criar uma primeira regra: "Compare as codificações". Isso significa que o número 0 é menor que 1, e podemos dizer que A é menor que B. A primeira regra de nosso `COLLATION` (codificação de caracteres em SQL) foi criada. Imagine que queremos designar uma segunda regra: "Não distinguir diferença entre maiúsculas e minúsculas", isso significa que o código 0 é equivalente ao código 2, e consequentemente o código 1 é equivalente ao código 3. Esta

regra fará com que, em uma busca pela letra "a", possamos ter como resultados as letras "a" e "A". Em nossa vida real, sabemos que temos muitos alfabetos inteiros, às vezes múltiplos alfabetos ou sistemas de escrita, como os orientais, com milhares de caracteres, juntamente com muitos símbolos especiais e sinais de pontuação.

Pesquise mais

Joel Spolsky é um dos cofundadores do Stack Overflow, e se você ainda não o conhece, com certeza o conhecerá em sua vida profissional, pois o Stack Overflow (https://stackoverflow.com/) é um site de perguntas e respostas sobre desenvolvimento de software. Mas, além deste site de informação, Joel Spolsky fez uma profunda pesquisa sobre estas questões de internacionalização, publicando um artigo muito interessante:

SPOLSKY, Joel. **O Mínimo Absoluto Que Todo Desenvolvedor De Software Absolutamente, Positivamente Precisa Saber Sobre Unicode E Conjunto de Caracteres (Sem Desculpas!)**. 8 out. 2003. Tradução: Paulo André. Disponível em: <http://local.joelonsoftware.com/wiki/O_M%C3%ADnimo_Absoluto_Que_Todo_Desenvolvedor_de_Software_Absolutamente%2C_Positivamente_Precisa_Saber_Sobre_Unicode_e_Conjuntos_de_Caracteres_(Sem_Desculpas!)>. Acesso em: 24 abr. 2018.

O MySQL suporta inúmeros conjuntos de caracteres e seus respectivos agrupamentos de regras de comparação. Para verificar quais conjuntos de caracteres estão instalados no seu MySQL, utilize a instrução:

```
SHOW CHARACTER SET;
```

Essa instrução exibirá todos os conjuntos de caracteres, mas vamos analisar especificamente dois deles, o "latin1" e o UTF-8.

Utilize a instrução da seguinte maneira:

```
SHOW CHARACTER SET WHERE charset LIKE 'latin1';
```

Ele tem em sua descrição (`DESCRIPTION`) "cp 1252 West European" e, por padrão, associa-se o conjunto de regras (*`DEFAULT COLLATION`*) com o conteúdo "latin1_swedish_ci". Agora utilize a instrução da seguinte maneira:

```
SHOW CHARACTER SET WHERE charset LIKE 'utf8';
```

O banco de dados tem em sua descrição (DESCRIPTION) "UTF-8 Unicode" e, por padrão, associa-se o conjunto de regras (DEFAULT COLLATION) com o conteúdo "utf8_general_ci".

A diferença entre eles é bem simples: imagine várias línguas e todos os caracteres e símbolos que elas detêm. Agora separe apenas as línguas com base latina: nesse subconjunto se encontra a língua portuguesa e ela representa um conjunto de caracteres muito menor (ORACLE, 2018). O UTF-8 representa muitas línguas, e o "latin1" uma parte destas línguas. Por essa razão, aplicações que apresentam uso com abrangência internacional deverão utilizar o padrão UTF-8.

Você também pode notar que, no MySQL, toda a cadeia de caracteres (CHARSET) já apresenta um conjunto de regras associado (COLLATION), mas para você exibir todas as instruções instaladas, utilize a seguinte instrução:

```
SHOW COLLATION;
```

Para você visualizar o conjunto de regras do UTF-8, utilize a instrução:

```
SHOW COLLATION WHERE collation LIKE 'utf8_general_ci';
```

Você deve ter notado que alguns conjuntos de regras apresentam o sufixo "_ci", ou "_cs". As letras "CI" (*case insensitive)* demonstram que, dentre todas as suas regras, não há distinção entre maiúsculas e minúsculas. Pesquisas feitas utilizando a palavra "moura", retornarão "Moura", "MOURA" ou "moura". Já as letras "CS" (*case sensitive*) demonstram que, dentre todas as suas regras, há distinção entre maiúsculas e minúsculas. Pesquisas feitas utilizando a palavra "moura" retornarão apenas "moura".

Reflita

Todo esse estudo sobre internacionalização teve seu embasamento em uma tabela chamada ASCII, que representa apenas 128 caracteres de impressão e controle. Mas, qual a relação dessa tabela limitada com as questões de internacionalização até aqui estudadas? Qual língua ela representa e por quê?

Linguagem de Definição de Dados (DDL – *Data Definition Language*)

Para criarmos um banco de dados, deveremos utilizar as instruções da classe da linguagem de definição de dados (DDL). A instrução a seguir tem essa finalidade:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS]
db_name
    [create_specification] ...
    create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
```

Ela cria um banco de dados com o nome especificado. `CREATE SCHEMA` é apenas um sinônimo para `CREATE DATABASE`. Ocorrerá um erro se o banco de dados existir e você não especificar `IF NOT EXISTS`. As opções *create_specification* identificam características do banco de dados, seu conjunto de caracteres (`CHARACTER SET`) e o conjunto de regras de comparação (`COLLATE`). As características do banco de dados são armazenadas no arquivo db.opt no diretório do banco de dados.

Para criarmos um banco de dados chamado "**mundo**", com o `CHARSET` UTF-8 e `COLLATION` "**utf8_general_ci**", a instrução será a seguinte:

```
CREATE DATABASE mundo
   DEFAULT CHARSET = utf8
   DEFAULT COLLATE = utf8_general_ci;
```

Seu banco de dados foi criado e, utilizando a instrução `SHOW DATABASES`, você poderá vê-lo.

Dica

Se você utilizar a instrução de criação de banco de dados e o mesmo já existir, visualizará uma mensagem de erro. Para isso não ocorrer, utilizamos a instrução da seguinte maneira:

```
CREATE DATABASE IF NOT EXISTS mundo
   DEFAULT CHARSET = utf8
   DEFAULT COLLATE = utf8_general_ci;
```

Agora você não terá um erro, mas um alerta de que o banco já existe.

Supondo que você deseja alterar o CHARSET ou COLLATION de seu banco de dados, você deve usar a instrução:

```
ALTER {DATABASE | SCHEMA} [db_name]
```

```
    alter_specification ...
    alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

Para alterar o CHARSET do banco de dados **mundo** para "**latin1**":

```
ALTER DATABASE mundo CHARSET = latin1;
```

É necessário muito cuidado com a próxima instrução, pois ela apagará seu banco de dados:

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

A execução deste comando apagará o banco de dados especificado em db_name (nome do banco de dados)

Exemplificando

A cláusula IF EXISTS previne que não seja gerado um erro se a base não existir. A instrução com a cláusula no banco de dados **world** ficaria assim:

```
DROP DATABASE IF EXISTS mundo;
```

Desta forma, você não terá um erro se o banco de dados não existir, mas apenas um alerta.

Todas as instruções SQL poderão estar em um arquivo texto que chamamos de *Script* (roteiros de instruções SQL), o qual é bastante utilizado para criação de bancos de dados. Em um *Script*, você pode adicionar quaisquer comandos SQL, porém eles devem estar em uma ordem lógica de execução.

Portanto, no decorrer de seu aprendizado, você poderá criar o seu *Script* de estudo com os comandos que você está testando.

Assimile

Manter uma documentação atualizada sobre todas as definições e alterações em um repositório de banco de dados é vital para a manutenção de *Scripts* na sustentação de uma área de desenvolvimento. No progresso de um projeto, é um requisito vital o compartilhamento de informações.

Sem medo de errar

Você trabalha em uma empresa que está desenvolvendo um aplicativo para o serviço de um Guia Turístico. Chegou a hora de darmos continuidade ao projeto. Para determinar a abrangência da utilização do seu banco de dados, podemos afirmar que ele será internacional. Isso não significa que ele já começará a ser utilizado no mundo inteiro, mas que você pode deixá-lo preparado para isso. Porém, antes de criar o seu banco de dados, você já deve ter iniciado as discussões sobre suas tabelas básicas.

Esta seria uma estrutura básica:

- Países – armazenar os dados sobre nome, continente, área, ano de independência, população, expectativa de vida, forma de governo, capital, moeda.
- Estados – armazenar os dados sobre nome, sigla, região, capital.
- Cidades – nome, população, data de criação.
- Pontos de interesses turístico – podemos entender como pontos de interesse público:
    - Os atrativos turísticos existentes na região (naturais, culturais, atividades econômicas, realizações técnicas, científicas e artísticas e eventos programados).
    - Os serviços e equipamentos turísticos (de hospedagem, alimentação, agenciamento, transporte, eventos, lazer e entretenimento, etc.).
    - A infraestrutura de apoio ao turismo (meios de acesso ao município, sistema educacional, médico-hospitalar, de comunicações, de transportes, de segurança, entre outros.).
    - As instituições e organizações responsáveis pelo desenvolvimento da atividade turística (órgãos estaduais de turismo, fóruns e conselhos estaduais regionais e municipais de turismo, etc.).

Para a criação do banco de dados, temos o UTF-8 (*encoding*) como conjunto de caracteres para aplicações internacionais e, portanto, a criação do banco de dados deverá ser elaborada com a seguinte instrução:

```
CREATE DATABASE IF NOT EXISTS gt
```

```
DEFAULT CHARSET = utf8
DEFAULT COLLATE = utf8_general_ci;
```

Na execução dessa instrução, o MySQL utiliza esta base com a cadeia de caracteres UTF-8, e isso significa que há uma abrangência de caracteres internacionais. No padrão de pesquisa para esse banco, estão sendo utilizadas regras comuns de comparação do "utf8_general_ci", logo não há distinção de caracteres maiúsculos e minúsculos.

Avançando na prática

Alterar a forma de pesquisa em um banco de dados

Descrição da situação-problema

Como DBA de uma grande empresa, você acaba de receber (vindo de uma empresa recém-adquirida pela sua) um banco de dados chamado "vendas", que já está criado e populado. Porém, necessidades recentes dos usuários – que inserem dados usando o alfabeto em uso no Brasil – requerem uma mudança: a alteração do padrão de pesquisa (`COLLATION`).

O padrão de pesquisa pedido é que não haja distinção de maiúsculas e minúsculas, nem de caracteres acentuados e cedilha. Isso significa que, se houver uma busca pela palavra "***intúicão***", caso a palavra "**Intuição**" esteja cadastrada, será exibida. Atualmente, o `COLLATION` está como "latin1_general_cs". Qual a característica de pesquisa do "latin1_general_cs"? Qual será a instrução de alteração e `COLLATION` para este procedimento?

Resolução da situação-problema

Notamos que temos duas questões importantes a serem respondidas. Vamos à primeira: o padrão de regras de pesquisa para o `COLLATION` "latin1_general_cs", pela sua identificação nas últimas duas letras, *"cs"* (*case sensitive*), demonstra que há distinção de letras maiúsculas e minúsculas. Nesse caso, se buscarmos a palavra "estudo", opções como "Estudo" e "ESTUDO" não serão retornados. Outra informação é que esse *collate* abrange os caracteres de alguns países, identificados por "Multilingual (Western European)", ou seja, Multilíngue (Europa Ocidental). Uma pesquisa mais aprofundada nos mostra que este `COLLATION` está preparado para os idiomas

da Europa Ocidental, como francês, espanhol, catalão, basco, português, italiano, albanês, holandês, alemão, dinamarquês, sueco, norueguês, finlandês, faroense, islandês, irlandês, escocês e inglês.

No MySQL, usando a instrução "`SHOW COLLATION WHERE Charset = 'latin1';`", você obterá todos os padrões de pesquisa instalados para o `CHARSET` "latin1". Após sua pesquisa por esses padrões, você deve ter identificado um em especial, o "***latin1_swedish_ci***". A característica deste `COLLATION` é justamente a especificada. A instrução para esta alteração deverá ser:

`ALTER DATABASE` mundo `CHARSET = latin1 COLLATE = latin1_swedish_ci;`

Faça valer a pena

1. Você está recebendo um banco de dados no MySQL chamado "**projesp**". Essa empresa de projetos especiais teve um banco de dados criado, mas não sabe suas características. Ao receber este banco, você primeiramente executa a instrução:

```
SHOW CREATE DATABASE projesp;
```

Você identifica que o CHARSET padrão deste banco é o "latin1".

Com as afirmativas acima, podemos ter certeza que o banco de dados:

a) Está preparado para ser usado na maioria dos países, ou seja, é internacional.
b) Não aceita cedilhas.
c) Será utilizado apenas no Brasil, Chile, Argentina, Venezuela, Peru, Bolívia e demais países da América Latina.
d) Pode ser utilizado por vários países, mas não a sua maioria.
e) Não pode ser utilizado por países de língua inglesa.

2. No banco de dados "**projesp**", já criado no MySQL, foi pedida uma alteração de padrão de internacionalização, pois muitos projetos deverão ser administrados, inclusive fora do Brasil. Além disso, para as pesquisas nesse banco, não deverá haver distinção de maiúsculas e minúsculas.

Com os requisitos acima a instrução SQL deverá ser:

a) `CREATE DATABASE projesp;.`
b) `ALTER DATABASE projesp CHARSET = utf8 COLLATE = latin1_swedish_ci;.`

c) `ALTER DATABASE projesp CHARSET = utf8 COLLATE = utf8_general_cs;.`
d) `ALTER DATABASE projesp CHARSET = utf8 COLLATE = utf8_general_ci;.`
e) `CREATE DATABASE projesp CHARSET = utf8 COLLATE = utf8_general_ci;.`

3. Você é o administrador de um servidor MySQL e, como tal, a responsabilidade qualquer criação ou alteração de estrutura de banco de dados é sua. Chegou até você uma tarefa de exclusão do banco de dados "**projesp**". Antes de você executar a tarefa, surgem algumas questões.

O diretor da área pede que você identifique qual a classe SQL e instrução específica para esta tarefa. Você, sem sombra de dúvidas, responde:

a) A classe é a DDL (*Data definition language*), e a instrução "`CREATE DATABASE projesp;`".
b) A classe é a DDL (*Data definition language*), e a instrução "`DROP DATABASE IF EXISTS projesp;`".
c) A classe é a DML (*Data manipulation language*), e a instrução "`DROP DATABASE IF EXISTS projesp;`".
d) A classe é a DTL (*Data transaction language*), e a instrução "`DROP DATABASE IF EXISTS projesp;`".
e) A classe é a DDL (*Data definition language*), e a instrução "`ALTER DATABASE projesp;`".

Seção 1.3

Criação de tabelas

Diálogo aberto

Você deve ter notado que, em sites de pesquisa, há vários critérios para podermos executar uma consulta. Na verdade, esses critérios são parâmetros de pesquisa que tentarão levá-lo a estabelecer um método para te conduzir a melhores resultados. Por exemplo, quando vamos a uma livraria para comprar um bom livro de detetives, procuramos na sessão de ficção; lá chegando, vamos à parte de histórias de detetive. Percebe que na vida real já buscamos limitar nossas buscas a fim de facilitar nossa procura? Essa busca é simplificada porque os organizadores das livrarias se preocupam em criar estantes com materiais afins, o que facilita a nossa atividade como consumidores de livros.

Nesta seção você entenderá as instruções envolvidas na criação de tabelas. Um banco de dados contém várias tabelas, e cada tabela representa, de forma matricial, com suas linhas e colunas, dados que poderão, em suas relações, representar informações.

Você trabalha em uma empresa que está desenvolvendo um Guia Turístico e é responsável por desenvolver o repositório de dados que armazenará informações do aplicativo. Você já criou um banco de dados e utilizou instruções para consultas nele. Também já planejou a estrutura das tabelas que comporão esse banco de dados. Agora, sua responsabilidade no projeto é a de implantar, sob forma de tabelas, a estrutura do repositório de dados do projeto Guia Turístico e, para isso, você terá que ter seu Diagrama Entidade-Relacionamento finalizado. Ainda há tempo de fazer as definições dos principais campos em cada tabela e seus respectivos tipos de dados, os quais serão armazenados. Os pontos de interesse turístico deverão estar organizados de forma que possam ser classificados como atrativos, serviços, equipamentos, infraestrutura de apoio e instituições ou organizações. Para cada elemento turístico haverá uma coordenada, que deverá armazenar sua latitude e longitude, pois algumas funcionalidades poderão ser utilizadas com um GPS. Por enquanto, crie uma tabela adicional para armazenar essas informações, relacionando-a ao restante da estrutura. O elemento turístico deve ter um campo para mostrar se está publicado ou não. Seu valor padrão é falso.

Para auxiliá-lo na resolução deste problema, nesta seção você verá os conceitos dos elementos envolvidos na criação de tabelas, quais são a estrutura e os identificadores utilizados na criação de tabelas e como isso afeta suas propriedades. Com esses conceitos, você dominará as instruções para criar suas tabelas e implementar a definição da estrutura de seu repositório de dados.

Não pode faltar

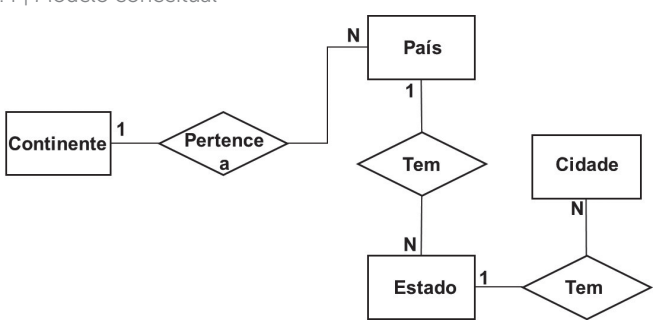
Modelo de dados

O Diagrama Entidade-Relacionamento (DER) é um dos diagramas mais comuns para representação de modelos de dados para sistemas gerenciadores de bancos de dados. O objetivo deste modelo é a certeza de que todos os dados existentes em um determinado contexto estão completamente representados, e com precisão (MACHADO, 2014).

Um administrador de banco de dados (DBA, do inglês *data base administrator*) utilizará esses modelos para gerar uma cópia fidedigna na construção física do banco de dados. O ato de tentar entender, modelar e classificar uma realidade é a abstração de um modelo. Abstrair é observar, em um contexto, as várias características e propriedades sobre fatos ou objetos que são relevantes ou essenciais em uma observação ou análise.

Segundo Heuser (2009), quando observamos um mapa-múndi, temos a representação do planeta Terra, na qual vemos alguns continentes, mas excluímos os detalhes dos países, seus limites, as etnias, suas línguas, extensões territoriais e outros detalhes. Mas, quando desta abstração você retrata a visão clara de regras de negócio e o entendimento da finalidade com que foi criada, teremos um **modelo conceitual**, conforme a Figura 1.4.

Figura 1.4 | Modelo conceitual

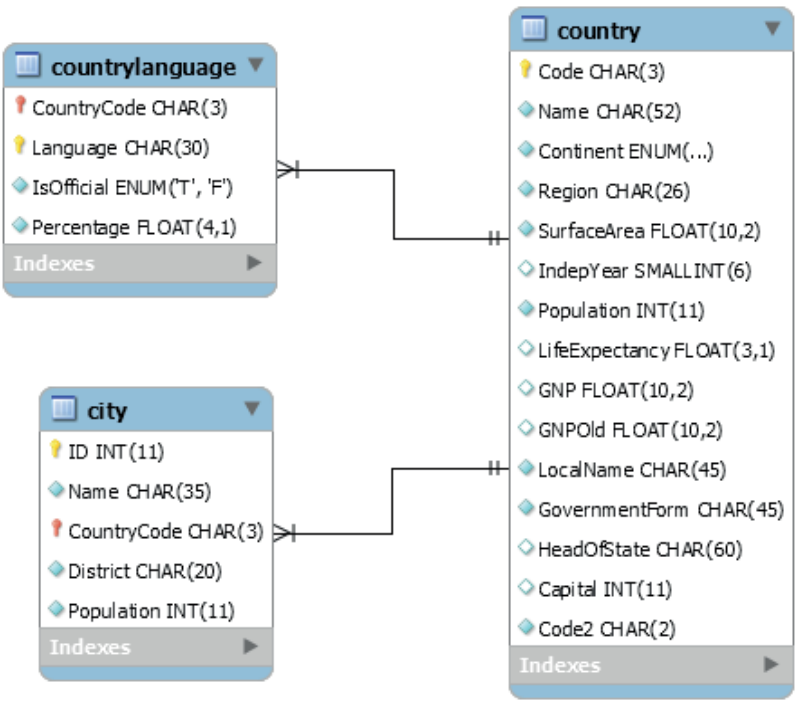


Fonte: elaborada pelo autor.

No modelo conceitual, a compreensão é simples e fácil pelo usuário final, pois nesse modelo sabemos quais dados devemos armazenar no banco de dados (MACHADO, 2014).

Um Diagrama Entidade-Relacionamento é o **modelo lógico** de uma estrutura de um banco de dados, em que não haverá nenhuma especificidade do Sistema Gerenciado de Banco de Dados. O modelo lógico descreve a estrutura que estará no banco de dados, conforme a Figura 1.5.

Figura 1.5 | Modelo lógico



Fonte: captura de tela do MySQL Workbench, elaborada pelo autor.

Reflita

Será que há a necessidade de planejamento para um simples repositório de dados?

Todos os modelos de repositórios servirão para determinar a real e correta implementação de um banco de dados. É o modelo lógico

que determinará as estruturas físicas de armazenamento de dados, descrevendo seus campos com tipos e tamanhos, índices, alguns tipos de preenchimento de campos, nomenclaturas, relacionamentos, etc.

A instrução para a criação de tabelas e sua estrutura é a `CREATE TABLE`. Em sua sintaxe, há vários parâmetros, mas aqui veremos as principais (ORACLE, 2018):

```
CREATE TABLE [IF NOT EXISTS] nome_tabela (
   Lista_campos
);
```

Cria a tabela com o nome especificado. Se esta já existir e a cláusula `IF NOT EXISTS` for utilizada, não ocasionará um erro, apenas um alerta.

Na lista de campos, a sintaxe é:

```
nome_campo tipo_campo[tamanho] [NOT NULL|NULL] [DEFAULT valor] [AUTO_INCREMENT] [PRIMARY KEY]
```

Nessa sintaxe, a cláusula NOT NULL|NULL indica se o campo aceita valores nulos ou não, `DEFAULT` especifica o valor padrão do campo e `AUTO_INCREMENT` identifica que o valor do campo é incrementado automaticamente quando um novo registro é inserido na tabela. Cada tabela poderá ter apenas um campo `AUTO_INCREMENT`.

Abaixo temos os tipos de dados que o MySQL pode armazenar, classificados em 3 diferentes tipos: numéricos, data e hora e texto.

1) Tipos Numéricos

- `SMALLINT [(M)] [UNSIGNED] [ZEROFILL]` - Um número inteiro no intervalo de -32768 a 32767. O intervalo sem sinal é de 0 a 65535.
- `MEDIUMINT [(M)] [UNSIGNED] [ZEROFILL]` - Um número inteiro no intervalo de -8388608 a 8388607. O intervalo sem sinal é de 0 a 16777215.
- `INT [(M)] [UNSIGNED] [ZEROFILL]` - Um número inteiro no intervalo de -2147483648 a 2147483647. O intervalo sem sinal é de 0 a 4294967295. INTEGER é um sinônimo de INT.
- `BIGINT [(M)] [UNSIGNED] [ZEROFILL]` - Um número inteiro no intervalo de -9223372036854775808

a 9223372036854775807. O intervalo sem sinal é de 0 a 18446744073709551615.

- `FLOAT [(M, D)] [UNSIGNED] [ZEROFILL]` - Um número de ponto flutuante, de precisão simples. Os valores admissíveis são -3,402823466E+38 a -1,175494351E-38, 0 e 1,175494351E-38 a 3,402823466E+38.
- `DOUBLE [(M, D)] [UNSIGNED] [ZEROFILL]` - Um número de ponto flutuante de precisão dupla. Os valores admissíveis são -1,7976931348623157E+308 a -2,2250738585072014E-308,0 e 2,2250738585072014E-308 a 1,7976931348623157E+308. No lugar de `DOUBLE`, você também pode utilizar o sinônimo `DOUBLE PRECISION`.

2) Tipo Data e hora

- `DATE` - O intervalo suportado é '1000-01-01' a '9999-12-31'. O MySQL exibe valores DATE no formato 'YYYY-MM-DD', mas permite a atribuição de valores a colunas DATE usando *strings* ou números.
- `DATETIME [(fsp)]` - Uma combinação de data e hora. O intervalo suportado é '1000-01-01 00: 00: 00.000000' a '9999-12-31 23: 59: 59.999999'. O MySQL exibe valores de DATETIME no formato 'AAAA-MM-DD HH: MM: SS [fração]', mas permite a atribuição de valores a colunas DATETIME usando *strings* ou números. Um valor '`fsp`' opcional no intervalo de 0 a 6 pode ser dado para especificar a precisão dos segundos fracionários. Um valor de 0 significa que não há parte fracionária. Se o qualificador '`fsp`' for omitido, a precisão padrão é 0.
- `TIMESTAMP [(fsp)]` - O intervalo é '1970-01-01 00: 00: 01.000000' UTC para '2038-01-19 03: 14: 07.999999' UTC. Os valores de TIMESTAMP são armazenados como o número de segundos desde a época ('1970-01-01 00:00:00' UTC).
- `TIME [(fsp)]` - O intervalo é '-838: 59: 59.000000' para '838: 59: 59.000000'. O MySQL exibe valores "TIME" no formato 'HH: MM: SS [fração]', mas permite a atribuição de valores a colunas "TIME" usando *strings* ou números.
- `YEAR [(4)]` - Um ano no formato de quatro dígitos. O MySQL exibe valores YEAR (ano) no formato YYYY, mas permite

a atribuição de valores a colunas YEAR usando *strings* ou números. Os valores são exibidos de 1901 a 2155 e de 0000.

3) Tipo Texto

- `CHAR [(M)] [CHARACTER SET charset_name] [COLLATE collation_name]` - Uma cadeia de comprimento fixo que é sempre preenchida à direita com espaços para o comprimento especificado quando armazenada. 'M' representa o comprimento da coluna em caracteres. O intervalo de M é de 0 a 255. Se M for omitido, o comprimento é 1.
- `VARCHAR (M) [CONJUNTO DE CARACTERES charset_name] [COLLATE collation_name]` - Uma cadeia de comprimento variável. M representa o comprimento máximo da coluna em caracteres. O intervalo de M é de 0 a 65.535. O comprimento máximo efetivo de um `VARCHAR` está sujeito ao tamanho máximo da linha (65.535 bytes, que é compartilhado entre todas as colunas) e ao conjunto de caracteres utilizado.
- `BINARY [(M)]` - O tipo BINARY é semelhante ao tipo CHAR, mas armazena cadeias de bytes binários em vez de cadeias de caracteres não binários. Um comprimento opcional M representa o comprimento da coluna em bytes. Se omitido, M é padronizado como 1.
- `VARBINARY (M)` - O tipo `VARBINARY` é semelhante ao tipo `VARCHAR`, mas armazena cadeias de bytes binários em vez de cadeias de caracteres não binários. M representa o comprimento máximo da coluna em bytes.
- `TINYBLOB` - Uma coluna "BLOB" com um comprimento máximo de 255 (28 - 1) bytes. Cada valor `TINYBLOB` é armazenado usando um prefixo de comprimento de 1 byte, que indica o número de bytes no valor.
- `TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]` - Uma coluna TEXT com um comprimento máximo de 255 (28 - 1) caracteres. O comprimento máximo efetivo é menor, se o valor contiver caracteres multibyte. Cada valor TINYTEXT é armazenado usando um prefixo de comprimento de 1 byte, que indica o número de bytes no valor.
- `BLOB [(M)]` - Uma coluna "BLOB" com um comprimento máximo de 65.535 (216 - 1) bytes. Cada valor BLOB é

armazenado usando um prefixo de comprimento de 2 bytes, que indica o número de bytes no valor.

- `TEXT [(M)][CHARACTER SET charset_name] [COLLATE collation_name]` - Uma coluna TEXT (texto) com um comprimento máximo de 65.535 (216 - 1) caracteres. O comprimento máximo efetivo é menor se o valor contiver caracteres multibyte. Cada valor "TEXT" é armazenado usando um prefixo de comprimento de 2 bytes, que indica o número de bytes no valor.
- `MEDIUMBLOB` - Uma coluna "BLOB" com um comprimento máximo de 16.777.215 (224 - 1) bytes. Cada valor "MEDIUMBLOB" é armazenado usando um prefixo de comprimento de 3 bytes, que indica o número de bytes no valor.
- `MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]` - Uma coluna TEXT com um comprimento máximo de 16.777.215 (224 - 1) caracteres. O comprimento máximo efetivo é menor se o valor contiver caracteres multibyte. Cada valor MEDIUMTEXT é armazenado usando um prefixo de comprimento de 3 bytes, que indica o número de bytes no valor.
- `LONGBLOB` - Uma coluna "BLOB" com um comprimento máximo de 4.294.967.295 ou bytes de 4 GB (232 - 1). O comprimento máximo efetivo das colunas "LONGBLOB" depende do tamanho máximo de pacote configurado no protocolo cliente / servidor e da memória disponível.
- `LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]` - Uma coluna TEXT com um tamanho máximo de 4.294.967.295 ou 4 GB (232 - 1) caracteres.
- `ENUM ('valor1', 'valor2', ...) [CHARACTER SET charset_name] [COLLATE collation_name]` - Um objeto de *string* que pode ter apenas um valor, escolhido na lista de valores 'valor1', 'valor2', [...], NULO, ou o valor de erro especial ''. Valores ENUM são representados internamente como números inteiros.
- `SET ('valor1', 'valor2', ...) [CHARACTER SET charset_name] [COLLATE collation_name]` - Um objeto de *string* que pode ter zero ou mais valores, e cada um dos quais deve ser escolhido na lista de valores 'valor1', 'valor2', [...]. Valores SET são representados internamente como números inteiros.

Pesquise mais

A instrução `CREATE TABLE` é a que contém mais parâmetros e exige um bom estudo e aprofundamento em suas pesquisas, pois, como ela determina a estrutura do banco de dados, seu comportamento será ditado por esta instrução. Para mais detalhes, acesse o link a seguir: ORACLE CORPORATION. CREATE TABLE Syntax. In: Oracle Corporation. **MySQL 5.7 Reference Manual**. 12 jul. 2018. Disponível em: <https://dev.mysql.com/doc/refman/5.7/en/create-table.html>. Acesso em: 16 jul. 2018.

O MySQL também pode armazenar dados espaciais, ou seja, que contenham valores geométricos, e dados do tipo JSON (notação de objetos JavaScript) (ORACLE, 2018).

Exemplificando

Para a criação de um campo tipo JSON, a instrução SQL poderia ser:

```
CREATE TABLE cinema
      (id INT AUTO_INCREMENT PRIMARY KEY,
      filme JSON
);
```

O campo **filme** poderá armazenar conteúdos JSON como:

```
[
   {
      "titulo": "JSON Zorro",
      "resumo": "a história da lenda do velho oeste",
      "ano": 2014,
      "genero": ["ação", "ficção"]
   }
]
```

Para exemplificar a criação de uma tabela, vamos criar uma chamada de "convidado". Sua estrutura será composta por Identificação, Nome, Sobrenome, E-mail, Data de registro e Data de nascimento. Vamos exemplificar a instrução e seus detalhes de sintaxe:

```
CREATE TABLE convidados (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   nome VARCHAR(30) NOT NULL,
   sobrenome VARCHAR(30) NOT NULL,
   email VARCHAR(50),
   data_reg DATETIME,
   nascimento DATE
);
```

O campo id tem seu comprimento em até 6 dígitos, e a cláusula `AUTO_INCREMENT` designa que este campo seja automaticamente preenchido pelo SGBD e incrementado a cada inserção de registro. Além disso, ele é a chave primária da tabela, por utilizar a cláusula `PRIMARY KEY`.

Os campos "nome" e "sobrenome" são do tipo `VARCHAR` e podem armazenar até 30 caracteres. Já os dois primeiros campos não podem armazenar valores nulos, assim como o campo "email", que também é do tipo `VARCHAR`, porém pode armazenar até 50 caracteres, assim como um valor nulo.

O campo "data_reg" pode armazenar data e hora, e o campo "nascimento" pode armazenar somente uma data.

Assimile

A criação de todas as tabelas necessárias para a definição de seu banco de dados determinará a sua estrutura, índices e relacionamentos. Quando tudo isso estiver finalizado, você terá criado o modelo de dados ou **metadados**.

Sem medo de errar

Atuando como responsável pelo banco de dados em um aplicativo de um Guia Turistico que está sendo desenvolvido, você agora deve desenvolver o repositório de dados que armazenará informações sobre

esse roteiro de turismo. Nessa etapa do projeto, você vai se concentrar na criação de tabelas, desenvolvendo um repositório de dados. Após suas pesquisas, você deve ter um DER com algumas tabelas básicas e para que elas sejam criadas.

A definição da estrutura de um banco de dados consiste em definir as tabelas para organização das informações e a sistematização de como estes dados se relacionam. Para a criação das tabelas que poderão definir a estrutura de dados, as instruções podem ser as descritas no Quadro 1.1.

Quadro 1.1 – Criação das tabelas do banco de dados

```
CREATE TABLE IF NOT EXISTS pais (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL DEFAULT '',
    continente ENUM('Ásia', 'Europa', 'América', 'África',
    'Oceania', 'Antártida')
    NOT NULL DEFAULT 'América',
    codigo CHAR(3) NOT NULL DEFAULT '',
);
CREATE TABLE IF NOT EXISTS estado (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL DEFAULT '',
    sigla CHAR(2) NOT NULL DEFAULT '',
);
CREATE TABLE IF NOT EXISTS cidade (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL DEFAULT '',
    populacao INT(11) NOT NULL DEFAULT '0',
);
CREATE TABLE IF NOT EXISTS ponto_tur (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL DEFAULT '',
    populacao INT(11) NOT NULL DEFAULT '0',
    tipo ENUM('Atrativo', 'Serviço', 'Equipamento',
    'Infraestrutura', Instituição', 'Organização'),
    publicado ENUM('Não', 'Sim') NOT NULL DEFAULT 'Não'
);
```

Fonte: elaborado pelo autor

Para armazenar as coordenadas de GPS:

```
CREATE TABLE IF NOT EXISTS coordenada (
    latitude FLOAT(10,6),
    longitude FLOAT(10,6)
);
```

Ao final destas etapas, você terá desenvolvido o repositório de dados.

Avançando na prática

Relacionando a cadeia de caracteres aos padrões de pesquisa

Descrição da situação-problema

Como administrador de banco de dados em uma empresa multinacional, você é responsável por manter a coerência entre os dados a serem armazenados e os padrões de pesquisa que serão utilizados com os bancos de dados em questão. Em uma dessas situações, você se depara com um erro legado de um administrador anterior, que já não está mais com a empresa. No banco de dados principal da empresa, há uma tabela com um campo específico, que foi definido com um padrão de caracteres e pesquisa diferente do padrão adotado em todo o banco de dados.

Nesse caso, o banco de dados foi criado com o charset UTF-8, mas um dos campos deverá ter o padrão Latin 1, para manter a compatibilidade com um sistema externo.

Essa tabela deverá ter um campo de identificação, inteiro; um campo nome, de caracteres, e um campo de descrição, que será consultado pelo sistema legado.

Sua responsabilidade, agora, é recriar a tabela para que ela contenha esses três campos. Como proceder?

Resolução da situação-problema

Para resolver o problema, primeiramente você precisa definir os campos da tabela:

- id, inteiro com até 11 dígitos, chave primária, auto incremento.
- nome, varchar com 50 caracteres.

- descricao, varchar com 50 caracteres, padrão de caracteres Latin1.

O campo 'nome' terá uma cadeia de caracteres e regras de pesquisa padrão do banco de dados e o campo 'descricao' deverá ter uma cadeia de caracteres Latin1. Sendo assim, as regras de pesquisa serão diferentes dos padrões do campo 'nome'.

No MySQL, é possível tipificar colunas específicas com cadeias de caracteres e padrões de pesquisa diferenciados do padrão do banco de dados. Nesse caso, teremos comportamentos diferentes quando consultas forem utilizadas em uma mesma tabela, no momento em que estas especificidades ocorrerem.

Para o exemplo dado, a instrução SQL seria:

```
CREATE TABLE IF NOT EXISTS tabela1 (
  id INT(11) AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(50),
  descricao VARCHAR(5)
    CHARACTER SET latin1
    COLLATE latin1_german1_ci
);
```

No exemplo apresentado, os padrões de pesquisa terão comportamentos diferentes quando executados.

Faça valer a pena

1. Em um determinado banco de dados há uma tabela de clientes. Nela estão cadastrados todos os nomes desses clientes e seus respectivos números de CPF. Na tabela 'cliente', foi especificado o campo 'nome' como tipo VARCHAR(50), e o campo 'cpf' como tipo CHAR(11).

Nestes dois campos temos os tipos VARCHAR e CHAR, então podemos afirmar que:

a) São idênticos, pois CHAR é sinônimo de VARCHAR.
b) VARCHAR não pode armazenar CPF.
c) CHAR armazena apenas letras do alfabeto de A à Z.
d) CHAR armazena dados de tamanho fixo.
e) Armazenar dados com VARCHAR desperdiça muito espaço em disco.

2. Determinada empresa financeira tem como principal produto comercializado os empréstimos para aposentados. Em sua tabela de clientes, há um campo chamado 'nascimento' e o tipo desse campo é TIMESTAMP. Nessa base de clientes, 90% deles são aposentados.

Você como DBA (administrador de banco de dados), verifica esta tabela e pode afirmar que:

a) O campo nascimento está definido corretamente e qualquer data será armazenada.
b) O campo nascimento está definido incorretamente, pois armazenará apenas horas.
c) O campo nascimento está definido incorretamente, pois armazenará datas entre 1970 e 2038.
d) Os tipos TIMESTAMP e DATETIME são sinônimos.
e) O campo nascimento não poderia ser do tipo DATETIME..

3. Você é o DBA (administrador de banco de dados) de uma empresa, e recebe um script (roteiro de instruções) SQL para ser executado. Ao abrir o script você verifica em seu conteúdo o seguinte:

```
CREATE TABLE X2 (
   id INT PRIMARY KEY AUTO_INCREMENT,
   nascimento TIMESTAMP);
```

Após verificar seu conteúdo, você pode afirmar que:

a) Sua execução não funcionará, pois uma tabela não pode se chamar X2.
b) O campo nascimento pode armazenar data com limites entre 1970 e 2038.
c) "id" é um nome de campo inválido.
d) Esta tabela não tem chave primária.
e) O script será executado e a tabela será criada como especificado.

Referências

CARDOSO, V.; CARDOSO, G. **Linguagem SQL** - Fundamentos e Práticas. 1. ed. São Paulo: Saraiva, 2013. 193p.

DICIO. **Dicionário Online de Português.** 2018. Disponível em: <https://www.dicio.com.br>. Acesso em: 25/03//2018.

HEUSER, C. A. **PROJETO DE BANCO DE DADOS**. 6. ed. Porto Alegre: Amgh, 2009. 282p.

MACHADO, F. N. R. **Banco de Dados** - Projeto e Implementação. 3. ed. São Paulo: Érica, 2014. 400p.

MANZANO, J. A. N. G. **MySQL 5.5 interativo**: guia essencial de orientação e desenvolvimento. 1. ed. São Paulo: Érica, 2011. 240p.

ORACLE CORPORATION. **MySQL 5.7 Reference Manual**. 7 jul. 2018. Disponível em: <https://dev.mysql.com/doc/refman/5.7/en/>. Acesso em: 16 jul. 2018.

Manipulação de dados e estruturas

Convite ao estudo

Em diversas situações do nosso cotidiano, utilizamos aplicativos com consulta/criação de banco de dados que precisam de alterações, condições especiais de preenchimento ou, ainda, ter alguns itens temporários que precisam ser excluídos depois de um determinado tempo. Pense, por exemplo, em uma casa de câmbio. Caso o sistema que eles utilizam mantenha apenas os valores oficiais das moedas para cada dia do ano, não seria interessante incluir nesse banco de dados os valores de compra e venda? Outro exemplo é o cadastro em algum site específico para maiores de 13 anos, você já pensou em como criar um campo que apenas permita a quem tem esta idade se cadastrar? Se, no meio de um processo, fosse necessário criar uma consulta apenas para gerar relatórios ou agregar dados, depois que essa informação não for mais necessária, ela pode ser excluída.

Nesta unidade, você terá acesso às instruções de manipulação de dados e também de alteração de estruturas, pois é comum em um projeto ocorrerem alterações de estrutura, seja por erros ou por novos requisitos do cliente. Por isso, você irá compreender o que está envolvido nas instruções para criação e manipulação de dados e estrutura de um banco de dados. Na prática, isso envolve aplicar e desenvolver os conhecimentos necessários para a elaboração de um script SQL com comandos que permitam a manipulação de dados e estruturas.

Lembre-se que você trabalha em uma empresa multinacional, em um projeto de desenvolvimento de um Guia Turístico. Na fase anterior do projeto, você participou

da criação do banco de dados, assim como toda a estrutura de tabelas, campos, chaves e relacionamentos. Nessa nova fase, você iniciará a manipulação de dados na estrutura criada. Nesse momento você terá que fazer algumas alterações em tabelas e rever seus relacionamentos. Após a manipulação de dados, você terá restrições a serem criadas para garantir a integridade de dados, alterar algumas estruturas excluindo ou adicionando colunas, tendo acesso às instruções de alteração de tabelas e restrições. Caso haja, ao final, alguma estrutura que deva ser refeita, você poderá excluir todas as tabelas desnecessárias. Portanto, agora, você executará toda inclusão e manutenção de dados. Isso resultará num repositório de dados consistente e acessível. Ao trabalhar nesse banco de dados, você terá que se fazer alguns questionamentos: será que todas as chaves primárias foram designadas? Posso criar um padrão para alguns campos após os mesmos já terem sido criados? Nos relacionamentos, será que devo criar restrições? Qual a finalidade de uma restrição?

Na primeira seção, você aprenderá sobre os principais comandos aplicados em banco de dados (inserção, atualização, exclusão e recuperação de dados, e suas clausulas). Em seguida, você aprenderá sobre arquitetura das tabelas e suas características, como alterá-las e como alterar *constraints*. Na última seção, você aprofundará seu conhecimento para manipulação de tabelas no que diz respeito a exclusão desse item em um projeto. Bons estudos!

Seção 2.1

Comandos utilizados na manipulação de bancos de dados

Diálogo aberto

Quando você utiliza algum aplicativo e faz alguma pesquisa, seja qual for, no celular, sites de músicas, filmes, etc., você estará acessando um banco de dados. Esse banco de dados foi manipulado para inserir, atualizar ou deletar informações e, após uma recuperação de dados, estes são exibidos a você.

Trabalhando em uma multinacional, no desenvolvimento de um Guia Turístico, você está na etapa de manipulação de banco de dados. Até aqui, você já deve ter criado seu banco de dados e uma estrutura básica, com algumas tabelas e suas respectivas propriedades, ou seja, a definição dos campos e tipos de dados que serão armazenados. Após a criação, você deve testar sua estrutura e, para isso, será necessário conhecer instruções para manipulação de banco de dados. Você realiza esses testes inserindo dados nas tabelas e, em seguida, verificando seu conteúdo para se certificar de que os dados necessários para o guia turístico estão sendo devidamente armazenados, nos formatos necessários e nas posições necessárias. Deverá, como parte do teste, utilizar o comando de atualização para modificar dados inseridos, verificando se as ações estão surtindo os efeitos desejados. Os dados a serem inseridos, são:

- Países (com respectivos continentes): Brasil, Índia, China e Japão;
- Estados (com respectivas siglas): Maranhão, São Paulo, Santa Catarina, Rio de Janeiro;
- Cidades (com respectivas populações aproximadas): Sorocaba, Déli, Xangaim Tóquio;
- Pontos turísticos (com sua especificação: Quinzinho de Barros (Instituição), Parque Estadual do Jalapão (Atrativo),Torre Eiffel (Atrativo), Fogo de Chão (Restaurante).

Os dados a serem alterados são:

- Alterar para "Atrativo" o primeiro ponto turístico da lista
- Alterar o segundo país (Índia) para ter o código "IND"

Por fim, você deverá deletar a primeira cidade da lista.

Em todos os casos, verifique – por meio do comando de seleção – se as inserções e alterações foram feitas corretamente. O teste servirá para avaliar a sua estrutura, bem como as informações ali armazenadas. Você manipulará seus dados e a estrutura de suas tabelas.

Nesta seção, você conhecerá o subconjunto das instruções SQL para a linguagem de manipulação de dados (DML, do inglês *Data Manipulation Language*), e isso inclui as instruções de inserção (`INSERT`). Após um dado armazenado, você poderá ter duas ações, sendo a primeira de atualizar os dados (`UPDATE`) ou exclusão de dados (`DELETE`). Com as informações contidas, veremos a recuperação de alguns dados (`SELECT`) e seus relacionamentos poderão ser revisados.

Não pode faltar

A partir de agora, você começará a utilizar o principal subconjunto de instruções SQL, a linguagem de manipulação de dados (DML). Esta manipulação são as instruções que irão inserir, atualizar ou modificar dados, assim como excluir dados em tabelas (CARDOSO; CARDOSO, 2009).

Reflita

É comum projetos precisarem sofrer ajustes? Em quais situações? Esses ajustes podem ser apenas pequenos ou também podem ser reestruturadores? Qual a importância dos comandos de manipulação de tabelas nessas alterações?

O subconjunto de instruções de DML é a principal instrução e a mais utilizada, por isso seu estudo deve ser levado a sério, pois desde um profissional especializado na área de banco de dados até o programador, obrigatoriamente, utilizará essas instruções em sua vida profissional. Ou seja, uma hora ou outra você terá que se lembrar dessas instruções e de seu comportamento

Nas instruções de manipulação de dados, existem as cláusulas identificadas como condicionais, e podemos utilizar mais de uma condição numa mesma cláusula por meio de **operadores lógicos**. Podemos obter um resultado único com a utilização desses operadores.

Na linguagem SQL, temos três operadores: `AND`, `OR` e `NOT` (E, OU e NÃO). A utilização desses operadores segue o mesmo conceito de uma tabela verdade, conforme você pode observar no Quadro 2.1.

Quadro 2.1 | Tabela verdade com operadores lógicos do SQL

X	Y	X e Y	X ou Y	Não-X
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro	Falso
Verdadeiro	Falso	Falso	Verdadeiro	Falso
Verdadeiro	Nulo	Nulo	Verdadeiro	Falso
Falso	Falso	Falso	Falso	Verdadeiro
Falso	Nulo	Falso	Nulo	Verdadeiro
Nulo	Nulo	Nulo	Nulo	Nulo

Fonte: elaborado pelo autor.

Como esses operadores são cumulativos, você pode trocar as ordens dos operadores lógicos sem alterar o resultado da ação desejada. Veremos agora as instruções para a manipulação de dados `INSERT`, `UPDATE` *e* `DELETE` (ORACLE, 2018).

A cláusula `SELECT`

O resultado de uma consulta SQL é uma tabela. As instruções SQL estão demonstradas conforme sintaxe no manual de referências do MySQL (ORACLE, 2018).

Na sintaxe da instrução `SELECT`, temos a cláusula `WHERE`, em que os conectivos `AND`, `OR` e `NOT` também podem ser utilizados. Também pode-se utilizar operadores de comparação (`<`, `<=`, `>`, `>=`, `=` e `<>`), com *strings* e expressões aritméticas, além de tipos especiais e tipos de data. A cláusula `FROM` especifica uma ou mais tabelas de sua consulta (MACHADO, 2014). Para nosso exemplo da tabela "convidado", podemos ter alguns exemplos de consulta como:

```
SELECT * FROM convidado;
```

Toda a tabela será exibida.

```
SELECT * FROM convidado WHERE nome LIKE 'A%';
```

Exibirá todos os convidados com o nome iniciando com 'A'.

Em nossa Seção 2.1 temos vários exemplos de consultas com

a utilização de várias cláusulas, que servirão de referência para auxiliá-lo nas consultas.

Instrução de Inserção (**INSERT**)

A instrução INSERT permite adicionar novas linhas ou registros numa tabela existente. Sua sintaxe é:

```
INSERT
   [INTO] nome_tabela
   [(nome_coluna [, nome_coluna] ...)]
   {VALUES | VALUE} (lista_valores) [, (lista_
valores)] ...
```

A declaração VALUES especificará os valores explícitos para serem inseridos. Caso a instrução seja utilizada sem parâmetros:

```
INSERT INTO nome_tabela () VALUES();
```

Se a lista de colunas e a lista VALUES estiverem vazias, a instrução determinará que o MySQL crie uma linha com cada coluna configurada com seus valores padrões.

Uma expressão pode se referir a qualquer coluna que tenha sido definida anteriormente em uma lista de valores. Por exemplo, você pode utilizar esse comando porque o valor para col2 refere-se a col1, atribuído anteriormente:

```
INSERT INTO nome_tabela (col1, col2) VALUES(15,
col1*2);
```

Como essa instrução está correta, ela será interpretada pelo MySQL. Porém, se você atribuir que o valor para col1 refere-se a col2 e colocá-lo após col1, como demonstrado abaixo, a sintaxe estará errada e não irá funcionar.

```
INSERT INTO nome_tabela (col1, col2)
VALUES(col2*2, 15);
```

Você também pode utilizar a declaração VALUES para inserir múltiplas linhas. Para fazer isso, inclua várias listas de valores de coluna separados por vírgula, com listas entre parênteses e também separadas por vírgulas, como descrito a seguir:

```
INSERT INTO nome_tabela (a,b,c)
   VALUES(1,2,3),(4,5,6),(7,8,9);
```

Cada lista de valores deve conter exatamente a quantidade de valores inserida por linha, como visto no exemplo acima, 3 valores por linha. Se você tentar inserir o número total de valores (9), em vez de três listas de três valores cada, você terá uma sintaxe que não irá funcionar, como representada a seguir.

```
INSERT INTO tbl_name (a,b,c)
  VALUES(1,2,3,4,5,6,7,8,9);
```

Para exemplificar algumas instruções, imagine uma tabela criada com a seguinte instrução:

```
CREATE TABLE IF NOT EXISTS convidado (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL DEFAULT '',
    nascimento DATE,
    estudante ENUM('Não', 'Sim') NOT NULL DEFAULT 'Não'
);
```

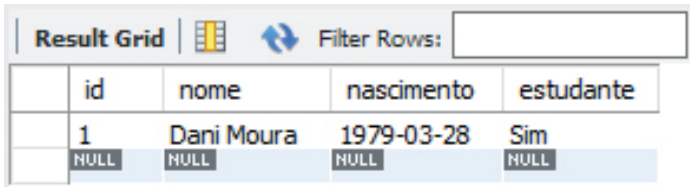
Ao utilizar a instrução abaixo, você estará inserindo uma linha à tabela "convidado", incluindo nas colunas "nome", "nascimento" e "estudante" os respectivos valores da cláusula `VALUES`. Não é necessário utilizar a ordem de criação das colunas na instrução.

```
INSERT INTO CONVIDADO (nome, nascimento, estudante)
  VALUES ('Dani Moura', '1979-03-28', 'Sim');
```

Para verificar o resultado, utilize a sintaxe a seguir. O resultado será visualizado conforme Figura 2.1.

```
SELECT * FROM convidado;
```

Figura 2.1 | Visualização de consulta da tabela convidado



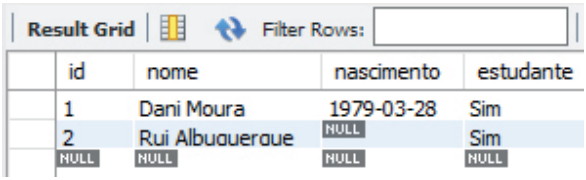
Fonte: captura de tela do MySQL Workbench, elaborada pelo autor.

Quando a coluna não for especificada, nada será atribuído e, ao consultar esta coluna, será exibida como NULL (nula). Para atribuir explicitamente um valor nulo, a instrução deverá ser:

```
INSERT INTO CONVIDADO (nome, nascimento, estudante)
        VALUES ('Rui Albuquerque', null , 'Sim');
```

Após a instrução, o resultado exibido ao consultar a tabela será como o descrito na Figura 2.2:

Figura 2.2 | Visualização de consulta da tabela convidado



Fonte: captura de tela do MySQL Workbench, elaborada pelo autor.

Fique atento, pois caso a coluna "nascimento" seja criada com a opção *NOT NULL*, a instrução não será executada, pois esta é uma restrição que exige um valor para a coluna.

Instrução de Atualização (UPDATE)

UPDATE é uma instrução DML que altera ou atualiza linhas em uma tabela. Quando se deseja definir expressões de tabelas comuns acessíveis no UPDATE, essa instrução pode começar com uma cláusula WITH. Um exemplo desta sintaxe está escrito a seguir:

```
UPDATE tabela_referência
     SET lista_atribuição
     [WHERE condição]
     [ORDER BY ...]
     [LIMIT quantidade_linhas]
value:
     {expr | DEFAULT}
assignment:
     nome_coluna = valor
lista_atribuições:
     atribuição [, atribuição] ...
```

A instrução `UPDATE` atualiza colunas de linhas existentes na tabela nomeada com novos valores. Já a cláusula `SET` indica quais colunas modificar e os valores que devem ser fornecidos. Cada valor pode ser fornecido como uma expressão ou a palavra-chave `DEFAULT` para definir uma coluna explicitamente como seu valor padrão. Um exemplo do comando `UPDATE` é:

```
UPDATE convidado
    SET estudante = 'Sim'
    WHERE nome = 'Lebrencio Grulher'
    AND nascimento = '08-Jul-1990';
```

O comando acima atualiza como sendo estudantes todos os convidados de nome Lebrencio Grulher que tenham nascido na data de 08 de Julho de 1990. Homônimos de Lebrencio Grulher que tenham nascido em datas diferentes ou outros convidados que não se chamam Lebrencio Grulher mas que tenham nascido em 08 de Julho de 1990 não serão atualizados para a condição de estudante.

Se fornecida, a cláusula `WHERE` especifica as condições que identificam quais linhas devem ser atualizadas. Fique atento, pois se não for apresentada nenhuma cláusula ***`WHERE`***, todas as linhas serão atualizadas. Por exemplo, para determinar que na tabela "convidado" a coluna "estudante" seja atualizada para "Não", como na sintaxe a seguir, a atualização ocorrerá em todas as linhas da tabela pela falta da cláusula `WHERE`.

```
UPDATE convidado SET estudante = 'Não';
```

O comando acima de fato será executado quando emitido via **MySQL Command Line Client**. Já na ferramenta gráfica **MySQL Workbench,** essa ação gera um erro, em que se pede que a instrução seja executada utilizando a cláusula ***`WHERE`***, especificando a chave do registro a ser atualizado, uma precaução para que uma atualização indevida seja executada. Isso mostra a atenção que você deve ter sobre determinadas ações e, por bom hábito, sempre que você utilizar instruções de **atualização** ou **exclusão**, utilize uma condição (`WHERE`) para limitar a ação da instrução.

Assimile

Tanto a instrução `UPDATE` quanto a opção `SET` atuam sobre atributos já existentes em tabelas.

Se a cláusula `ORDER BY` for especificada, as linhas serão atualizadas na ordem especificada. A cláusula `LIMIT` impõe um limite no número de linhas que podem ser atualizadas. A instrução a seguir exemplifica a cláusula `ORDER BY`:

```
UPDATE convidado
    SET estudante = 'Sim'
    WHERE nascimento < '08-Jul-1990'
    ORDER BY nome;
```

Esse comando vai atualizar todos os cadastrados nascidos antes de 08 de Julho de 1990 como sendo estudantes.

Quando adicionamos um limite por meio da cláusula `LIMIT` , o que teremos é uma limitação nas atualizações. No caso do comando acima, se quisermos limitar o número de atualização aos primeiros 10 nomes (em ordem alfabética), teremos:

```
UPDATE convidado
    SET estudante = 'Sim'
    WHERE nascimento < '08-Jul-1990'
    LIMIT 10
    ORDER BY nome;
```

Se você acessar uma coluna da tabela para ser atualizada em uma expressão, `UPDATE` usará o valor atual da coluna. Por exemplo, a instrução a seguir define "col1" com o valor atual incrementado em um:

```
UPDATE tabela_ref SET col1 = col1 + 1;
```

Se você definir uma coluna para o valor que ela já possui atualmente, o MySQL percebe isso e não o atualiza (ORACLE, 2018).

Exemplificando

Se uma tabela contiver 1 e 2 na coluna ID (*primary key* – chave primária) e 1 for atualizado para 2 antes de 2 ser atualizado para 3, ocorrerá um erro. Para evitar esse problema, inclua uma cláusula `ORDER BY` para fazer com que as linhas com valores de ID maiores sejam atualizadas antes daquelas com valores menores:

```
UPDATE tabela SET ID = ID + 1 ORDER BY ID
   DESC;
```

Você também pode executar operações UPDATE abrangendo várias tabelas. No entanto, você não pode usar ORDER BY ou LIMIT com um UPDATE de várias tabelas. Aqui está um exemplo juntando duas tabelas:

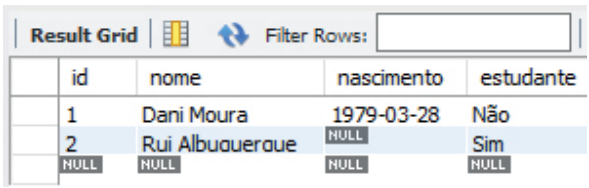
```
UPDATE lista, produto SET lista.preco = produto.preco
WHERE lista.id = produto.id;
```

Em nossa tabela "conv¬idado", vamos executar a seguinte instrução:

```
UPDATE convidado SET estudante = 'Não' WHERE id = 1;
```

Após esta instrução teremos como resultado a Figura 2.3:

Figura 2.3 | Visualização de consulta da tabela convidado



Fonte: captura de tela do MySQL Workbench, elaborada pelo autor.

Instrução de Exclusão (`DELETE`)

`DELETE` é uma instrução DML que exclui linhas de uma tabela. Sua sintaxe geral é dada por:

```
DELETE FROM nome_tabela
    [WHERE condição]
    [ORDER BY ...]
    [LIMIT quantidade_linhas]
```

Assim como vimos antes para a instrução `UPDATE`, as condições na cláusula `WHERE` irão identificar quais linhas serão excluídas. E sem a utilização da cláusula WHERE, todas as linhas são excluídas. Quando declarada, a condição é avaliada a cada linha, e se verdadeira ela é excluída. Exemplo:

```
DELETE FROM convidados
    WHERE estudante = 'Sim'
    ORDER BY nome
    LIMIT 10;
```

O comando acima deleta os dez primeiros estudantes (por ordem alfabética) classificados como estudantes.

Se a cláusula `ORDER BY` estiver presente, as linhas serão excluídas na ordem especificada.

A cláusula `LIMIT` estabelece um limite para o número de linhas que podem ser excluídas. Mesmo quando você excluir a linha que contém o valor máximo de uma coluna "AUTO_INCREMENT", o valor não será reutilizado.

Se a instrução `DELETE` incluir uma cláusula `ORDER BY`, as linhas serão excluídas na ordem especificada pela cláusula. Quando utilizadas em conjunto, as cláusulas `ORDER BY` e `LIMIT` são bastante uteis. Veja a sintaxe a seguir:

```
DELETE FROM log_usuario WHERE usuario = 'rm'
ORDER BY datahora_acao LIMIT 1;
```

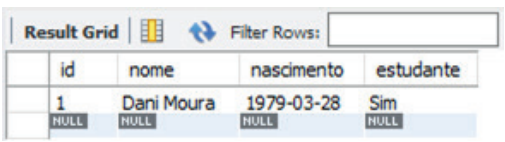
A instrução acima localiza linhas correspondentes ao que foi especificado na cláusula ***WHERE***, classificando-as pela coluna "datahora_acao" do tipo ***DATETIME*** e excluindo a primeira, ou seja, a mais antiga.

Aplicando essa instrução ao nosso banco de dados texto, mais especificamente em nossa tabela convidado, teremos:

```
DELETE FROM convidado WHERE id = 2;
```

Após a instrução, o resultado da consulta a tabela "convidado" será conforme a Figura 2.4, com o convidado 2 deletado:

Figura 2.4 | Visualização de consulta da tabela convidado



Fonte: captura de tela do MySQL Workbench, elaborada pelo autor.

Pesquise mais

Existem mais detalhes da sintaxe das instruções e outras instruções de manipulação de dados (DML) que não veremos aqui. Para conhecê-las acesse o material de referência a seguir: ORACLE CORPORATION. Data Manipulation Statements. In: Oracle Corporation. **MySQL 8.0 Reference Manual**. 24 jul. 2018. Disponível em: <https://dev.mysql.

com/doc/refman/8.0/en/sql-syntax-data-manipulation.html>. Acesso em: 24 jul. 2018.

Trabalhando em uma empresa que está desenvolvendo um Guia Turístico, você está em uma etapa mais avançada do projeto. Até aqui, você já deve ter criado um repositório de banco de dados com características de cadeia de caracteres com padrões internacionais, assim como ter definido algumas tabelas. Em cada tabela houve a definição de algumas colunas e de suas características.

Mas um repositório é criado vazio, gerando a necessidade de começar a testar seus tipos e conteúdo, valores padrões, comportamentos das chaves, além de outras características, ou seja, começar a testar e ver o comportamento. Para isso, devemos começar a manipulação de dados e visualizar os resultados com consultas. Você poderia executar as instruções para inserção de dados similares às apresentadas no Quadro 2.2.

Quadro 2.2 | Inserções e alterações no banco de dados "guia turístico"

```
1   INSERT INTO pais (nome, continente, codigo)
2         VALUES('Brasil', 'América', 'BRA'),
3             ('Índia', 'Ásia', 'IDN'),
4             ('China', 'Ásia', 'CHI'),
5             ('Japão', 'Ásia', 'JPN');
6   SELECT * FROM pais;
7   INSERT INTO estado (nome, sigla)
8         VALUES('Maranhão', 'MA'),
9             ('São Paulo', 'SP'),
10            ('Santa Catarina', 'SC'),
11            ('Rio de Janeiro', 'RJ');
12  SELECT * FROM estado;
13  INSERT INTO cidade (nome, populacao)
14        values('Sorocaba', 700000),
15            ('Déli', 26000000),
16            ('Xangai', 22000000),
17            ('Tóquio', 38000000);
18  SELECT * FROM cidade;
19  INSERT INTO ponto_tur (nome, tipo)
20        VALUES('Quinzinho de Barros', 'Instituição'),
```

```
21              ('Parque Estadual do Jalapão',
22                  'Atrativo'),
23              ('Torre Eiffel', 'Atrativo'),
24              ('Fogo de Chão', 'Restaurante');
25      SELECT * FROM ponto _ tur;
```

Fonte: elaborado pelo autor.

Após isso, você deve testar algumas alterações, como colocar como "Atrativo" o primeiro ponto turístico da lista e alterar o segundo país (Índia) para ter o código "IND". Você pode realizar estas tarefas da seguinte forma:

```
UPDATE ponto_tur SET tipo = 'Atrativo' WHERE id = 1;
SELECT * FROM ponto_tur;

UPDATE pais SET codigo = 'IND' WHERE id = 2;
SELECT * FROM pais;
```

Você pode finalizar os testes com instruções de exclusão, objetivando deletar a primeira cidade da lista:

```
DELETE FROM cidade WHERE id = 1;
SELECT * FROM cidade;
```

Outros testes, como excluir um dos pontos turísticos da planilha "ponto_tur" de alguma posição específica, como a linha 4, podem ser realizadas:

```
DELETE FROM ponto_tur WHERE id = 4;
SELECT * FROM ponto_tur;
```

Pronto, você conseguir demonstrar que o banco de dados tem a estrutura adequada até o momento, e responde corretamente a alterações.

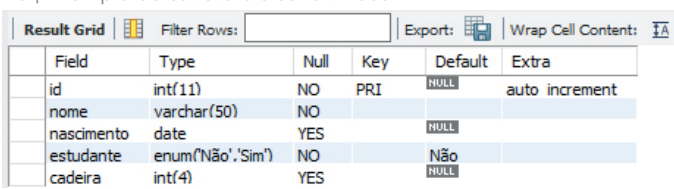
Avançando na prática

Reorganização de numeração das cadeiras para um evento.

Descrição da situação-problema

Você está trabalhando em uma grande empresa organizadora de eventos. Em um determinado momento, você recebeu um banco de dados com uma tabela de convidados para um evento de integração de uma empresa. Durante a organização do evento, essa base foi muito acessada e, por essa razão, houve um grande número de alterações e exclusões em seus dados. A empresa que contratou a organização da festa solicitou que a distribuição do salão fosse alterada, sendo que o número de cadeiras deveria estar distribuído por ordem alfabética. Isso significa que a numeração das cadeiras começará com o primeiro nome em ordem alfabética para cadeira 1 e, assim consecutivamente. Você deverá realizar essa alteração no banco de dados para gerar a ordem correta de nomes, e essa tarefa deve ser feita com rapidez, pois os nomes deverão ainda ser impressos e colocados no salão de eventos. A estrutura do banco de dados utilizada está descrita na Figura 2.5.

Figura 2.5 | Exemplo de estrutura a ser utilizada



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
nome	varchar(50)	NO			
nascimento	date	YES		NULL	
estudante	enum('Não','Sim')	NO		Não	
cadeira	int(4)	YES		NULL	

Fonte: captura de tela do MySQL Workbench, elaborada pelo autor.

Resolução da situação-problema

Para a resolução desse problema, nota-se a necessidade de atualizar a coluna cadeira, conforme a ordenação alfabética dos nomes dos convidados. Para isso, você deve armazenar o número da cadeira numa variável e incrementar esse número. A criação da variável pode ser feita no MySQL com a seguinte instrução:

```
SET @numero = 0;
```

Essa instrução cria a variável e armazena o valor 0 nela.

Após isso, poderemos utilizar um comando `SELECT` para verificar o valor dessa variável com a seguinte instrução:

```
SELECT @numero;
```

Será exibido o valor 0.

Agora podermos executar a instrução de atualização dos números das cadeiras:

```
UPDATE convidado SET cadeira = @numero := @numero + 1 ORDER
    BY nome;
```

Essa instrução irá ordenar a tabela por nome, em ordem alfabética, e passará a atribuir à coluna cadeira o número armazenado na variável "@numero".

É importante observar que esse resultado é obtido somente quando os comandos forem executados no MySQL Command Line Client. Quando utilizamos o MySQL Workbench, a interface não executará a ação por questão de segurança.

Faça valer a pena

1. Você está liderando um projeto e, em sua equipe, há um estagiário com dúvidas na execução de uma tarefa. Ele executou a seguinte instrução e obteve sucesso:

```
CREATE TABLE IF NOT EXISTS usuario (
    id INT(11) NOT NULL KEY,
    nome VARCHAR(50) NOT NULL DEFAULT '',
    nascimento date,
    estudante ENUM('Não', 'Sim') NOT NULL DEFAULT 'Não',
    cadeira int(4)
);
```

Após essa ação, ele tenta executar a seguinte instrução, obtendo apenas uma mensagem de erro:

```
INSERT INTO usuario (nome, nascimento, estudante, cadeira)
    VALUES ('Matheus Ferrari', '2000-02-01', 'Sim', 27);
```

Assinale a alternativa que especifica corretamente porque ocorreu a mensagem de erro:

a) O formato da data deveria ser '2000/02/01'.
b) O formato da data deveria ser '01-02-2000'.
c) A coluna cadeira não pode receber valor.
d) A coluna estudante só pode receber o valor DEFAULT 'Não'.
e) A coluna id não está especificada.

2. Você está liderando um projeto e, em sua equipe, há um estagiário com dúvidas na execução de uma tarefa que você atribuiu a ele. Ele executou a seguinte instrução e obteve sucesso:

```
CREATE TABLE IF NOT EXISTS usuario (
   id INT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
   nome VARCHAR(50) NOT NULL DEFAULT '',
   nascimento date,
   estudante ENUM('Não', 'Sim') NOT NULL DEFAULT
'Não',
   cadeira int(4)
);
```

Após isso, executou a seguinte instrução:

```
INSERT INTO usuario (id) values (2);
```

Ao executar a instrução `SELECT * FROM usuario WHERE id = 2;`, ele comprovará que:

a) A coluna id conterá o valor 2 e, a coluna cadeira conterá o valor 1.
b) A coluna id conterá o valor 2 e, a coluna estudante conterá o valor 'Sim'.
c) A coluna id conterá o valor 2 e, a coluna estudante conterá o valor 'Não'.
d) O MySQL retornará uma mensagem de erro, pois o nome não pode ser nulo.
e) A coluna id conterá o valor 2 e, todas as colunas conterão o valor nulo.

3. Você está liderando um projeto e, em sua equipe, há um estagiário com dúvidas na execução de uma tarefa que você atribuiu a ele. Ele executou a seguinte instrução e obteve sucesso:

```
CREATE TABLE IF NOT EXISTS sala (
   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
   nome VARCHAR(50) NOT NULL DEFAULT ' ',
   nascimento date,
   estudante ENUM('Não', 'Sim') NOT NULL DEFAULT
'Não',
   cadeira int(4)
);
```

Em seguida, executou a seguinte instrução:

```
INSERT INTO sala () VALUES ();
```

Ao executar a instrução `SELECT * FROM sala;`, ele comprovará que:

a) A coluna id conterá o valor 1 e, a coluna nome conterá um espaço em branco.
b) A coluna id conterá o valor 1 e, a coluna estudante conterá o valor 'Sim'.
c) A coluna id conterá o valor 1 e, a coluna cadeira conterá o valor 1.
d) Obterá uma mensagem de erro, pois o nome não pode ser nulo.
e) Essa instrução está incorreta.

Seção 2.2

Alteração de tabelas e *constraints*

Diálogo aberto

Olá! Vamos começar mais uma seção de exploração e aprendizado da Programação em Banco de Dados. Você notará em sua vida profissional que todo projeto necessitará de ajustes após o seu início. Quando um projeto está em fase embrionária, isso não é um problema, pois estamos querendo realmente colocar à prova nossa estrutura, por isso devemos sempre ter o cuidado de atender a todos os requisitos necessários em nossa especificação. Porém, nem sempre o projeto está em seu início. Novos pacotes de software, por exemplo, estão sujeitos a novas necessidades com base no feedback de clientes e novos objetivos dos criadores. Em muitos desses casos, a estrutura do banco de dados projetada para a solução pode sofrer alterações. Agora, teremos que fazer algumas alterações em tabelas e rever seus relacionamentos. Após isso, deveremos criar restrições para a integridade de dados. É sobre essas ações que falaremos nessa seção.

Trabalhando em seu projeto de construção do Guia Turístico, neste momento, seu repositório de dados está todo testado e temos vários dados e relacionamentos criados, mas houve um requisito muito importante que foi esquecido até esta etapa. Qual é a língua ou línguas nativas que o país pode ter? Há outras alterações que devem ser feitas na estrutura do baco de dados, nas tabelas e no Diagrama Entidade-Relacionamentos (DER)? Num mesmo país, podemos ter mais de uma língua, e o guia deve levar isso em consideração. Não se esqueça que esse tipo de ação pode demandar alterações no DER. Após essa ação, você deve criar uma tabela que contemple este relacionamento com um país, de acordo com o que estipulou no DER. Além desta mudança na definição de seu banco de dados (DDL), você deve novamente, por meio das instruções de consulta de dados (DQL), identificar quais campos estão vazios ou quais relacionamentos estão quebrados por meio das instruções de manipulação de dados (DML). Em seguida,

corrija estas situações, garantindo que não haja campos vazios ou relacionamentos quebrados. Certifique-se que todas as alterações até o momento surtiram o efeito desejado.

É possível fazer estas alterações sem a necessidade de reconstruir toda a base de dados e todas as tabelas? E o que acontece com os dados já existentes?

Nesta seção, você vai entender como são feitas as alterações nas tabelas e também como criar restrições para as mesmas.

O trabalho é interessante e desafiador. Vamos em frente!

Não pode faltar

Nesta seção, você irá conhecer as instruções SQL para estabelecer alterações em suas tabelas, e isso significa que você poderá acrescentar, alterar ou excluir colunas. Além disso, você pode também acrescentar ou excluir restrições (*constraints*) às suas colunas (MACHADO, 2014).

Alteração de tabelas (`ALTER TABLE`)

`ALTER TABLE` é um comando que altera a estrutura de uma tabela. Por exemplo, você pode adicionar ou excluir colunas, criar ou destruir índices, alterar os tipos de coluna existentes ou renomear colunas, ou a própria tabela. Você também pode alterar características, como o mecanismo de armazenamento usado para a tabela ou o comentário da tabela.

A sintaxe da instrução é:

```
ALTER TABLE nome_tabela
    [especificação_alteração
        [,especificação_alteração] ...]
```

No Quadro 2.3, você tem os principais qualificadores de especificação_alteração que podem ser utilizados.

Quadro 2.3 | Qualificadores do comando `ALTER TABLE`.

```
1  opções _ tabela
2  | ADD [COLUMN] (nome _ coluna column _ definition,...)
3  | ADD [COLUMN] nome _ coluna column _ definition
4        [FIRST | AFTER nome _ coluna]
5  | ADD {INDEX|KEY} [índice _ nome]
```

```
6         [índice_tipo] (índice_nome_coluna,...) [índice_
7   opção] ...
8   | ALTER [COLUMN] nome_coluna {SET DEFAULT literal | DROP
9   DEFAULT}
10  | ALTER INDEX índice_nome {VISIBLE | INVISIBLE}
11  | CHANGE [COLUMN] old_nome_coluna new_nome_coluna
12  column_definition
13        [FIRST|AFTER nome_coluna]
14  | [DEFAULT] CHARACTER SET [=] charset_nome [COLLATE [=]
15  collation_nome]
16  | CONVERT TO CHARACTER SET charset_nome [COLLATE
17  collation_nome]
18  | {DISABLE|ENABLE} KEYS
19  | {DISCARD|IMPORT} TABLESPACE
20  | DROP [COLUMN] nome_coluna
21  | DROP {INDEX|KEY} índice_nome
22  | DROP PRIMARY KEY
23  | MODIFY [COLUMN] nome_coluna column_definition
24        [FIRST | AFTER nome_coluna]
25  | ORDER BY nome_coluna [, nome_coluna] ...
26  | RENAME COLUMN old_nome_coluna TO new_nome_coluna
27  | RENAME {INDEX|KEY} old_índice_nome TO new_índice_
28  nome
29  | RENAME [TO|AS] new_tbl_nome
```

Fonte: elaborado pelo autor.

A sintaxe para muitas das alterações permitidas é semelhante às cláusulas da instrução `CREATE TABLE`. A palavra `COLUMN` é opcional e pode ser omitida, exceto `RENAME COLUMN` (para distinguir uma operação de renomeação de coluna da operação de renomeação de tabela `RENAME`).

Assimile

Quando temos a necessidade de provocar alterações na estrutura de uma tabela, temos que nos certificar de executar essa instrução com o banco de dados sem nenhum acesso.

Múltiplas cláusulas `ADD`, `ALTER`, `DROP` e `CHANGE` são permitidas em uma única instrução `ALTER TABLE`, separadas por vírgulas. Essa é

uma extensão do MySQL para o SQL padrão, que permite apenas uma cláusula por instrução `ALTER TABLE`. Por exemplo, para descartar várias colunas em uma única instrução, execute o seguinte comando:

```
ALTER TABLE cliente DROP COLUMN parentesco,
DROP COLUMN telefones;
```

Na instrução acima, a tabela "cliente" terá as colunas "parentesco" e "telefones" excluídas.

Para você alterar o valor do campo auto incremento de uma tabela, você utilizará a instrução:

```
ALTER TABLE cliente AUTO_INCREMENT = 13;
```

Como sabemos, uma tabela pode ter apenas uma coluna identificada como auto incremento e, no exemplo da instrução acima, na tabela "cliente", temos o campo auto incremento sendo alterado explicitamente para o valor 13. A próxima inserção de registro nessa tabela terá neste campo automaticamente preenchido para o valor 13, e assim consecutivamente.

Exemplificando

No MySQL temos uma característica muito interessante. Podemos ter no nosso banco de dados, conforme você já estudou, por questões de internacionalização, uma cadeia de caracteres (`CHARSET`) específica. No entanto o MySQL permite que uma tabela tenha em sua especificidade uma cadeia de caracteres diferenciada. Vamos supor que seu banco de dados esteja especificado com o padrão **UTF-8**. Todas as tabelas que você criar estarão utilizando este padrão e, por uma questão de algum requisito específico, você necessita alterar uma tabela com outra cadeia de caracteres, neste caso o **LATIN-1**. Para isso, a instrução seria:

```
ALTER TABLE pessoas CHARACTER SET = latin1;
```

Após a execução dessa instrução, todas as tabelas criadas estarão utilizando o *charset* **UTF-8**, com exceção da tabela "pessoas".

Adicionando e excluindo colunas (`ADD` e `DROP`)

Use o comando `ADD` para adicionar novas colunas a uma tabela e o comando `DROP` para remover colunas existentes. `DROP` nome_coluna é uma extensão do MySQL para o SQL padrão.

Para adicionar uma coluna a uma posição específica dentro de uma linha da tabela, deve-se usar `FIRST` ou `AFTER` nome_coluna. O padrão é adicionar a coluna à última posição.

Você deve ficar atento a algumas condições quando for fazer exclusão de colunas:

- Se uma tabela contiver apenas uma coluna, a coluna não poderá ser eliminada.
- Se o que você pretende é remover a tabela, use a instrução `DROP TABLE`.
- Se as colunas forem eliminadas de uma tabela, as colunas também serão removidas de qualquer índice do qual façam parte.
- Se todas as colunas que compõem um índice forem descartadas, o índice também será eliminado.
- Se você usa `CHANGE` ou `MODIFY` para encurtar uma coluna para a qual existe um índice na coluna, e o tamanho da coluna resultante é menor que o tamanho do índice, o MySQL encurta o índice automaticamente.

Já a instrução para adicionar um campo, por exemplo o campo nome como o tipo especificado na tabela "pessoas", será realizada por:

```
ALTER TABLE pessoas ADD nome VARCHAR(50);
```

Para excluir o campo "sobrenome" da mesma tabela a instrução seria:

```
ALTER TABLE pessoas DROP COLUMN sobrenome;
```

Após executar essa instrução, ao consultar a tabela "pessoas", o campo "sobrenome" já não será mais exibido.

Renomeando, redefinindo e reordenando colunas (`CHANGE`, `MODIFY`, `RENAME COLUMN` e `ALTER`)

As cláusulas `CHANGE`, `MODIFY`, `RENAME COLUMN` e `ALTER` permitem que os nomes e definições de colunas existentes sejam alterados. Eles têm essas características comparativas:

- `CHANGE`: pode renomear uma coluna e alterar sua definição ou ambos. Esse comando tem mais capacidade que `MODIFY` ou `RENAME COLUMN`, mas às custas de conveniência para algumas operações. O comando CHANGE requer a nomeação da coluna duas vezes, se ela já não tiver sido renomeada, e requer que você especifique novamente a definição da

coluna, apenas renomeando-a. Juntamente com `FIRST` ou `AFTER`, esse comando pode reordenar colunas.

Para alterar uma coluna a fim de alterar seu nome e sua definição, use `CHANGE`, especificando os nomes antigos e novos e a nova definição. Observe a seguinte instrução

```
ALTER TABLE pessoas CHANGE antigo novo BIGINT NOT NULL;
```

Nesse exemplo de alteração, uma coluna está sendo renomeada e especificada como `NOT NULL` de "antigo" para "novo", e tendo sua definição alterada para usar o tipo de dados `BIGINT`, mantendo o atributo `NOT NULL`.

O comando `ALTER TABLE` também pode alterar uma definição de coluna, mas não seu nome, sendo mais conveniente do que CHANGE.

Além disso, usando os qualificadores `FIRST` ou `AFTER`, você pode reordenar colunas. Observe a instrução abaixo

```
ALTER TABLE pessoas MODIFY novo INT NOT NULL;
```

`MODIFY` é mais conveniente para alterar a definição sem alterar o nome por que requer o nome da coluna apenas uma vez.

- `RENAME COLUMN`: pode alterar o nome de uma coluna, mas não sua definição, sendo um comando mais conveniente do que `CHANGE` para renomear uma coluna sem alterar sua definição pois exige apenas os nomes antigos e novos. Observe a seguinte instrução que está renomeando, na tabela "pessoas", o campo chamado "novo" para "antigo"

```
ALTER TABLE pessoas RENAME COLUMN novo TO antigo;
```

Reflita

Você deve ter notado que muitas das cláusulas utilizadas na instrução `ALTER TABLE` são idênticas da instrução `CREATE TABLE`. Por que isso ocorre? Na sua opinião, por que as restrições geralmente são criadas quando esta definição está pronta?

Usando restrições (*constraints*)

Se você adicionar os qualificadores `UNIQUE INDEX` ou `PRIMARY KEY` a uma tabela, o MySQL a armazenará antes de qualquer índice

não exclusivo para permitir a detecção de chaves duplicadas o mais cedo possível.

Você pode identificar, numa cláusula da instrução `CREATE TABLE` ou `ALTER TABLE`, um campo como `PRIMARY KEY`. Por exemplo, veja a instrução que está especificando o campo "id" como chave primária e sem que ele possa conter valores nulos:

```
CREATE TABLE pessoa(
   id int NOT NULL PRIMARY KEY,
   nome varchar(255) NOT NULL,
   sobrenome varchar(255),
   idade int
);
```

Agora, vamos supor que você deseja alterar sua chave primária, ela conterá duas colunas e não apenas uma. Primeiro você deverá retirar a chave primária declarada:

```
ALTER TABLE pessoa
   DROP PRIMARY KEY;
```

Após isso, será necessário a nomeação de uma restrição `PRIMARY KEY`, e para defini-la em várias colunas, use a seguinte sintaxe SQL:

```
ALTER TABLE pessoa
   ADD CONSTRAINT PK_pessoa PRIMARY KEY (id, sobrenome);
```

Nesse momento, você está declarando em sua tabela "pessoa" uma chave primária composta, com o nome de PK_pessoa e os dois campos que a compõe.

O MySQL suporta chaves estrangeiras, que permitem a referência cruzada de dados relacionados entre tabelas e restrições de chaves estrangeiras, o que ajuda a manter consistentes esses dados dispersos. A sintaxe essencial para uma definição de restrição de chave estrangeira em uma instrução `CREATE TABLE` ou `ALTER TABLE` é semelhante à apresentada no Quadro 2.4.

Quadro 2.4 | Comando ALTER TABLE com referências.

```
1  ALTER TABLE tbl_name
2        ADD [CONSTRAINT [símbolo]] FOREIGN KEY
3        [index_nome] (index_col_nome, ...)
4        REFERENCES tbl_nome (index_col_nome,...)
```

```
5         [ON DELETE referências]
6         [ON UPDATE referências]
7
8   referências:
9         RESTRICT | CASCADE | SET NULL | NO ACTION
```

Fonte: elaborado pelo autor.

O MySQL cria implicitamente um índice de chave estrangeira que é nomeado de acordo com as seguintes regras:

- Se definido, o valor do símbolo `CONSTRAINT` é usado. Caso contrário, o valor do index_nome `FOREIGN KEY` é usado.
- Se nenhum símbolo `CONSTRAINT` ou `FOREIGN KEY` index_nome estiverem definidos, o nome do índice de chave estrangeira será gerado usando o nome da coluna de chave estrangeira de referência.

Os relacionamentos de chave estrangeira envolvem uma tabela pai (que contém os valores de dados centrais) e uma tabela filha (com valores idênticos apontando para seu pai). A cláusula `FOREIGN KEY` é especificada na tabela filha. Fique atento, pois as tabelas pai e filha devem usar o mesmo mecanismo de armazenamento. Colunas correspondentes na chave estrangeira e a chave referenciada devem ter tipos de dados semelhantes. O tamanho e o sinal dos tipos inteiros devem ser os mesmos. O comprimento dos tipos de *string* não precisa ser o mesmo. O MySQL requer índices em chaves estrangeiras e chaves referenciadas para que as checagens de chaves estrangeiras possam ser rápidas e não requererem uma varredura de toda a tabela. Na tabela de referência, deve haver um índice em que as colunas de chave estrangeira sejam listadas como as primeiras colunas na mesma ordem. Esse índice é criado automaticamente na tabela de referência, se ela não existir (ORACLE, 2018). Veja as instruções no Quadro 2.5.

Quadro 2.5 | Criação de tabelas pai e filha.

```
1   CREATE TABLE pai (
2         id INT NOT NULL,
3         nome VARCHAR(50),
4         PRIMARY KEY (id)
5   );
6   CREATE TABLE filha (
7         id INT PRIMARY KEY,
8         parente_id INT,
9         nome VARCHAR(50)
10  );
```

Fonte: elaborado pelo autor.

Com esses qualificadores, as duas tabelas serão criadas, e podemos notar um campo de relacionamento entre elas, demonstrado pelo campo parente_id na tabela filha. Por exemplo, vamos criar uma integridade referencial entre as tabelas, utilizando esse relacionamento por meio da seguinte instrução:

```
ALTER TABLE filha
  ADD CONSTRAINT FK_parente
  FOREIGN KEY (parente_id) REFERENCES pai(id);
```

Após esta instrução, o MySQL cria uma *integridade referencial*, uma *chave estrangeira* referenciando a tabela filha com a tabela pai, por meio dos campos *parente_id* e *id*, respectivamente. Com esses comandos, embora não esteja explícito, o MySQL criou um índice para a coluna *parente_id* automaticamente.

Para garantir a integridade referencial, o MySQL rejeita qualquer operação `INSERT` ou `UPDATE` que tente criar um valor de chave estrangeira em uma tabela filha, se não houver um valor de chave candidato correspondente na tabela pai (CARDOSO; CARDOSO, 2013). Quando uma operação `UPDATE` ou `DELETE` afeta um valor de chave na tabela pai que possui linhas correspondentes na tabela filha, o resultado depende da ação referencial especificada usando as subcláusulas `ON UPDATE` e `ON DELETE` da cláusula `FOREIGN KEY`. O MySQL suporta opções sobre a ação a ser tomada, listadas aqui:

- `CASCADE`: esse qualificador exclui ou atualiza a linha da tabela pai e exclui ou atualiza automaticamente as linhas correspondentes na tabela filha. Tanto o qualificador `ON DELETE CASCADE` como o `ON UPDATE CASCADE` são suportados. Entre duas tabelas, não devem ser definidas várias cláusulas `ON UPDATE CASCADE` para atuarem na mesma coluna na tabela pai ou na tabela filha.
- `SET NULL`: esse qualificador exclui ou atualiza a linha da tabela pai e define como `NULL` a coluna ou colunas de chave estrangeira na tabela filha. Ambas as cláusulas `ON DELETE SET NULL` e `ON UPDATE SET NULL` são suportadas. Você deve ter o cuidado de certificar-se de que, ao especificar uma ação `SET NULL`, não tenha declarado as colunas na tabela filha como `NOT NULL`.

- `RESTRICT`: rejeita a operação de exclusão ou atualização da tabela pai. Especificar `RESTRICT` (ou `NO ACTION`) é o mesmo que omitir a cláusula `ON DELETE` ou `ON UPDATE`.
- `NO ACTION`: essa é uma palavra-chave do SQL padrão. No MySQL, a equivalente é a `RESTRICT`. O servidor MySQL rejeita a operação de exclusão ou atualização para a tabela pai, se houver um valor de chave estrangeira relacionado na tabela referenciada. Alguns sistemas de banco de dados possuem cheques diferidos, e `NO ACTION` é um cheque adiado. No MySQL, as restrições de chave estrangeira são verificadas imediatamente, portanto, `NO ACTION` é o mesmo que `RESTRICT`.

Para excluir uma restrição, execute a instrução abaixo, em que a chave criada é mantida após esta execução, assim somente a restrição será excluída.

```
ALTER TABLE filha DROP FOREIGN KEY FK_parente;
```

Pesquise mais

Nada melhor para aprimorar o conhecimento em restrições do que praticá-lo, alterando as restrições para poder notar seus comportamentos. Abaixo há uma lista de alguns sites de referência para estudos

- COMO adicionar uma foreign key em uma tabela já criada. 29 jan. 2016. Disponível em: <https://pt.stackoverflow.com/questions/110806/como-adicionar-uma-foreign-key-em-uma-tabela-j%C3%A1-criada>. Acesso em: 1 maio 2018.
- MYSQL foreign key constraints, cascade delete. 26 maio 2010. Disponível em: <https://stackoverflow.com/questions/2914936/mysql-foreign-key-constraints-cascade-delete>. Acesso em: 1 maio 2018.
- QUAL é a finalidade das opções RESTRICT, CASCADE, SET NULL e NO ACTION? 9 jun. 2017. Disponível em: <https://pt.stackoverflow.com/questions/211737/qual-%C3%A9-a-finalidade-das-op%C3%A7%C3%B5es-restrict-cascade-set-null-e-no-action/211751>. Acesso em: 1 maio 2018.

Sem medo de errar

Retomando nossa tarefa de construção do Guia Turístico, nessa etapa você deve provocar algumas mudanças no trabalho realizado até agora. Para isso, você deve revisar seu DER e em seguida readequá-lo para que contemple as novas tabelas e relacionamentos a serem criadas. Não se esqueça que este tipo de ação pode demandar alterações no DER. Após essa ação, você deve criar uma tabela que contemple este relacionamento com um país, de acordo com estipulado no DER. Além dessa mudança na definição de seu banco de dados (DDL), você deve novamente, por meio das instruções de consulta de dados (DQL), identificar quais campos estão vazios ou quais relacionamentos estão quebrados, com as instruções de manipulação de dados (DML). Em seguida, corrija essas situações, garantindo que não haja campos vazios ou relacionamentos quebrados. Certifique-se que todas as alterações até o momento surtiram o efeito desejado.

Você deve ter notado que, para a criação de relacionamentos entre tabelas, será necessária uma tabela pai e uma filha. Na tabela filha, além da estrutura definida para sua finalidade, também deverá existir o campo que servirá de referência para a tabela pai. Notamos um novo requisito, que é o de determinados países terem mais de um idioma, embora um seja determinado como o idioma oficial. Nessa relação, temos a cardinalidade de 1 para N, ou seja, teremos um país que pode ter um idioma ou mais.

Uma das soluções poderá ser uma tabela:

```
CREATE TABLE IF NOT EXISTS linguagemPais (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    codigoPais INT(11),
    linguagem VARCHAR(30) NOT NULL DEFAULT '',
    oficial ENUM('Sim', 'Não') NOT NULL DEFAULT
'Não'
);
```

A coluna "codigoPais" será a referência para sua tabela pai, nomeada "pais".

Porém, não está explícita a restrição que determinará a integridade referencial. Para isso, deve-se executar a instrução abaixo:

```
ALTER TABLE linguagemPais
   ADD CONSTRAINT FK_linguagemPais
   FOREIGN KEY (codigoPais) REFERENCES pais(id);
```

Após essa instrução, a chave estrangeira está referenciada e somente poderão ser adicionadas linguagens a um país existente.

Com isso em mente, todas as tabelas definidas deverão ter os campos que serão utilizados para criar restrições, além de integridades referenciais. Para um relacionamento obter êxito, os campos das tabelas filhas não podem conter valores nulos. Você deverá, então, novamente, revisar o DER de seu projeto e verificar todas as tabelas e relacionamentos. Após atualizado seu projeto lógico, fisicamente você deverá executar todas as alterações e inclusões de campos e restrições, que determinarão a integridade referencial, conforme o DER.

Avançando na prática

Controlar atividades de um professor

Descrição da situação-problema

Você foi procurado pela direção de uma faculdade para uma consultoria. A faculdade, uma das mais populares e procuradas da região, o procurou sabendo que você é um profissional competente de bancos de dados, pois necessita controlar e correlacionar:

- Os dados cadastrais dos alunos: Nome; Registro Acadêmico (RA); Endereço; Curso; Semestre; e Coeficiente de rendimento (CR).
- Os cursos oferecidos: Direito; Administração; Enfermagem.
- As disciplinas de cada curso: Dir_Disciplina1 até Direit_DisciplinaN; Adm_Disciplina1 até Adm_DisciplinaN; e Enf_Disciplina1 até Enf_DisciplinaN.
- As características de cada disciplina: Carga Horária; Semestres; e Conteúdo (grande campo de texto)

A solicitação envolve criar um banco de dados com estrutura suficiente para conter essas tabelas e, com elas, estabelecer adequadamente os relacionamentos. Para isso você terá que usar o MySQL. Porém, existem algumas restrições. Você deve determinar que, se um aluno for excluído,

consequentemente, suas informações também deverão ser excluídas. A outra restrição é que um curso só poderá ser excluído se não houver nenhum aluno matriculado. A coluna CR não poderá armazenar valores nulos a menos que o aluno esteja cursando o primeiro semestre. É possível realizar a solicitação da universidade? Como fazê-lo de maneira a atender a todas as solicitações?

Resolução da situação-problema

Podemos começar a tarefa pela criação da base de dados que vai conter as tabelas necessárias:

```
CREATE DATABASE faculdade;
USE faculdade;
CREATE TABLE aluno (
    ra INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome CHAR(50) NOT NULL
-- Adicione nas próximas linhas os demais elementos solicitados para a tabela Aluno
);
```

Em seguida, deve ser criada a tabela que vai conter os cursos:

```
CREATE TABLE curso (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome CHAR(50) NOT NULL
);
```

Para finalizar, deve ser criada a tabela de disciplinas:

```
CREATE TABLE disciplina (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nome CHAR(50) NOT NULL
-- inserir as características de cada disciplina (Carga Horária, Semestre em
-- que é minstrada, e conteúdo, que deve ser um campo de texto de 500 caracteres
```

```
);
```

E pronto! A diretoria da universidade agora pode inserir os dados e realizar buscas nas tabelas, bem como operações relacionais com os dados que forem inseridos nas tabelas.

Faça valer a pena

1. Você está recebendo a responsabilidade de criar uma integridade referencial entre duas tabelas. A tabela que contém notas fiscais tem em sua estrutura uma coluna chamada cliente_id. Essa coluna tem uma restrição determinando sua integridade referencial com a tabela de clientes. Porém, ao excluir um cliente, a restrição não inibe essa ação, mesmo que exista uma nota fiscal com o mesmo cliente que foi excluído.

Para determinar na restrição criada que a mesma iniba a ação de exclusão do cliente, ela deve conter a cláusula:

a) `ON DELETE STRUCTURE.`
b) `ON DELETE RESTRICT.`
c) `ON DELETE CASCADE.`
d) `ON DELETE DEFAULT.`
e) `ON DELETE NO ACTION.`

2. Você está recebendo a responsabilidade de criar uma integridade referencial entre duas tabelas. A tabela que contém notas fiscais tem em sua estrutura uma coluna chamada cliente_id. Essa coluna tem uma restrição determinando sua integridade referencial com a tabela de clientes. Porém, ao excluir um cliente, a restrição deveria excluir todas as notas fiscais desse mesmo cliente. Para determinar na restrição criada que a mesma exclua todas as notas relacionadas, ela deveria conter uma cláusula específica.

Assinale a alternativa que contém a cláusula mencionada no texto acima:

a) `ON DELETE STRUCTURE.`
b) `ON DELETE RESTRICT.`
c) `ON DELETE CASCADE.`
d) `ON DELETE DEFAULT.`
e) `ON DELETE NO ACTION.`

3. Você recebeu a tarefa de analisar um banco de dados e consistir em toda sua estrutura. Ele contém tabelas com índices, relacionamentos e restrições já criadas, mas, ao verificar a tabela de Clientes, você nota que não há um campo para armazenar o documento de cadastro de pessoas físicas, e isso é obrigatório. Para corrigir essa tabela, uma instrução deve ser executada.

Assinale a alternativa que contém a representação correta da instrução a ser executada:

a) `ALTER TABLE cliente ADD COLUMN cpf int NOT NULL;.`
b) `ALTER TABLE cliente ADD COLUMN cpf VARCHAR(11) NOT SPACE;.`
c) `ALTER TABLE cliente DROP COLUMN cpf INT(11) NOT NULL;.`
d) `ALTER TABLE cliente ADD COLUMN cpf VARCHAR(11);.`
e) `ALTER TABLE cliente ADD COLUMN cpf VARCHAR(11) NOT NULL;.`

Seção 2.3

Exclusão de tabelas em banco de dados

Diálogo aberto

Olá aluno! Você já viu como se realiza a inclusão e alteração de restrições nas tabelas, agora é hora de continuar seu aprendizado, alterando e removendo tabelas, entre outras ações. Os comandos de deleção devem ser cuidadosamente compostos, sob pena de gerarem problemas enormes para o banco de dados. Em 2005, a empresa britânica Data Captain perdeu vários dados de seus clientes e de suas transações quando um programador digitou o comando `delete [table] where [condition]`, quando queria digitar `delete from [table] where [condition]`. No primeiro caso, a cláusula de condição WHERE é ignorada, e a tabela é deletada por completo, já no segundo, apenas as linhas que atendem à CONDIÇÃO são apagadas. Como a empresa tinha um backup de alguns dias de idade, perdeu dados e transações equivalentes a centenas de milhares de libras esterlinas por conta da falta da condicional WHERE. Você entende a necessidade de cuidados para comandos de deleção? Pois é, ela é enorme.

Lembre-se que você está trabalhando em uma empresa multinacional em um projeto bastante relevante: o desenvolvimento de um guia turístico. Você já tem um banco de dados com uma estrutura pronta e os relacionamentos criados. Com a definição de chaves primárias e estrangeiras, você pôde criar restrições em seus relacionamentos, determinando a integridade desses campos e regras para sua manutenção de dados. Porém, em alguns momentos, após as definições ou novos requisitos, há a necessidade de excluir estruturas. Após o início de um projeto isso deve ser revisado, mas usualmente sempre ocorrerá. Um GPS, por exemplo, tem como função básica determinar localizações utilizando o modelo de latitude e longitude. Essas informações são de números flutuantes com no mínimo seis casas decimais. Será de vital importância poder consultar as informações de um ponto turístico e sua exata localização. Nós deixamos essa informação

na fase inicial do projeto, numa tabela específica, porém, isso não será necessário. Altere a sua tabela de Elementos Turísticos, adicionando esses dois campos. Será necessário, também, alterar a tabela "Países", adicionando uma nota de 0 a 10 com o nível de interesse para o turista (quanto mais pontos turísticos famosos, maior o interesse), e a tabela "Cidades", incluindo uma lista com os três melhores restaurantes. Após isso, não teremos a necessidade de manter essa tabela e você poderá excluí-la. Novamente, você deverá manipular os dados para testar nossa estrutura atual e, em equipe, você deverá discutir suas mudanças e atualizar o diagrama de entidades e relacionamentos, de modo que reflita exatamente a estrutura atual de seu banco de dados.

Para realizar as alterações de estrutura quando tabelas devem ser excluídas, você precisa aprender que há a necessidade da verificação das limitações e restrições que foram criadas no relacionamento entre as tabelas envolvidas. Você também deverá aprender em quais as etapas que pode exclui-las, assim como as instruções SQL e a definição de novas estruturas aplicadas com estas alterações.

Vamos em frente com nosso aprendizado!

Não pode faltar

Após seu banco de dados ter sua estrutura definida, com todos os seus campos, tipos de dados, chaves primárias e estrangeiras, ele está apto a ter sua manutenção de dados, ou seja, o ato de incluir, alterar e excluir dados.

Contudo, ao executar essas ações, você deve ter criado restrições, justamente para previnir que os relacionamentos entre as tabelas estejam íntegros e protegidos. Imagine que você terá que excluir uma tabela, alterando sua estrutura. Após essa certificação, caso você tenha que alterar seu banco de dados, você deverá garantir que suas mudanças poderão ser executadas e todas as restrições respeitadas.

Como você já estudou, toda tabela deve ter uma chave primária identificando um registro. Já uma chave estrangeira é a responsável por unir duas tabelas. Essa chave é o campo que se refere a uma chave primária em outra tabela. É importante observar que a tabela que contém a chave estrangeira é chamada de tabela filha e a tabela que contém a chave candidata (seja ela primária ou não) é chamada de tabela pai ou referenciada (CARDOSO; CARDOSO, 2009).

A restrição `FOREIGN KEY` é usada para impedir ações que destruam links entre tabelas, além de impedir que dados inválidos sejam inseridos na coluna de chave estrangeira, já que ela deve ser um dos valores contidos na tabela para a qual ela aponta. No Quadro 2.6 temos essa estrutura criada com algumas instruções.

Quadro 2.6 | Criação de Tabelas para atribuição e cálculo de notas

```
1   CREATE TABLE aluno (
2         id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
3         nome CHAR(50) NOT NULL
4   );

5   CREATE TABLE curso (
6         id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
7         nome CHAR(50) NOT NULL
8   );

9   CREATE TABLE nota (
10        aluno_id INT NOT NULL,
11        curso_id INT NOT NULL,
12        dataavaliacao DATE NOT NULL,
13        nota DOUBLE NOT NULL,
14        PRIMARY KEY(aluno_id, curso_id, dataavaliacao),
15        INDEX i2 (curso_id),
16        FOREIGN KEY (aluno_id) REFERENCES aluno(id) ON DELETE
17                    CASCADE,
18        FOREIGN KEY (curso_id) REFERENCES curso(id) ON DELETE
19                    RESTRICT
20  );
```

Fonte: elaborado pelo autor.

Como você pode notar nessa estrutura, a tabela nota detém duas restrições, tendo como referência as tabelas:

- **aluno** – (linhas 1 a 4) contém as informações acerca dos alunos da instituição e pode ser alterada para conter ainda mais dados de registro desse aluno, caso necessário; e
- **curso** – (linhas 5 a 8) contém informações acerca do curso em que os alunos poderão estar matriculados.

Essas duas tabelas estão cedendo suas chaves primárias para se relacionar com a tabela "nota". As referências contidas no exemplo poderão ser confirmadas por meio da instrução abaixo, que mostra o comando `CREATE TABLE`, que cria a tabela nomeada. Você deve ficar atento que para usar essa declaração, pois deve ter privilégios adequados para essa tabela (ORACLE, 2018).

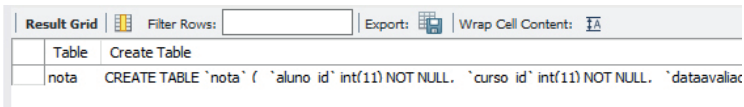
```
SHOW CREATE TABLE nome_tabela;
```

Ao ser executada a instrução abaixo, será criada uma tabela de nome "nota", com exibição dos resultados:

```
SHOW CREATE TABLE nota;
```

Ao executar esta instrução, como resultado, teremos a mensagem conforme mostra a Figura 2.6.

Figura 2.6 | Resultado da instrução após execução



Fonte: captura de tela do MySQL Workbench, elaborada pelo autor.

Na tela referenciada na Figura 2.6, acima, ao clicar com o botão direito do mouse sobre a linha acima, na opção `COPY FIELD UNQUOTED` teremos como resultado a instrução no Quadro 2.7, apresentado a seguir:

Quadro 2.7 | Criando uma tabela com restrições (*constraints*)

```
1   CREATE TABLE `nota` (
2         `aluno_id` int(11) NOT NULL,
3         `curso_id` int(11) NOT NULL,
4         `dataavaliacao` date NOT NULL,
5         `nota` double NOT NULL,
6     PRIMARY KEY (`aluno_id`,`curso_id`,`dataavaliacao`),
7     KEY `i2` (`curso_id`),
8     CONSTRAINT `nota_ibfk_1` FOREIGN KEY (`aluno_id`)
9   REFERENCES
10        `aluno` (`id`) ON DELETE CASCADE,
11    CONSTRAINT `nota_ibfk_2` FOREIGN KEY (`curso_id`)
12  REFERENCES
13        `curso` (`id`)
14  ) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Fonte: elaborado pelo autor.

O conjunto de instruções entre as linhas 6 (inclusive) e 13 (inclusive) deve sempre ser utilizado para determinar as restrições utilizadas nas tabelas que poderão ser alteradas ou excluídas. Como já mencionamos, analisando essa instrução podemos nos certificar que a tabela "nota" é a filha da tabela "aluno" e "curso". A tentativa de exclusão da tabela aluno ou curso está restringida, ou seja, não será possível.

Exemplificando

As duas restrições criadas na tabela "nota" foram nomeadas automaticamente pelo MySQL. A primeira restrição, `nota_ibfk_1`, fazendo referência à tabela "aluno", e a segunda restrição como `nota_ibfk_2`. Isso ocorre por que, na utilização da instrução para sua criação na sintaxe utilizada, não há um nome explícito.

Quando você deseja remover uma ou mais tabelas, a instrução `DROP TABLE` deve ser utilizada. Atenção: essa instrução remove a definição da tabela e todos os dados da tabela (MANZANO, 2011). Sua sintaxe envolve mencionar as tabelas que devem sofrer a ação como mostrado abaixo.

```
DROP TABLE [IF EXISTS] nome_tabela [, nome_tabela] ...
```

Se quaisquer tabelas nomeadas na lista de argumentos não existirem, a instrução falhará, com um erro indicando por nome quais tabelas não existentes não foram possíveis descartar, não sendo realizada nenhuma alteração. Existe uma maneira de evitar esse erro: se você utilizar a cláusula `IF EXISTS` em vez de um erro, um alerta é gerado para cada tabela inexistente. Isso também pode ser útil para eliminar tabelas em circunstâncias incomuns, sob as quais há uma entrada no dicionário de dados. Porém, essa cláusula não deleta nenhuma tabela gerenciada pelo mecanismo de armazenamento. Para utilizar a cláusula (`IF EXISTS`) em nosso exemplo, averiguando a existência da tabela "aluno" antes de executar o comando DROP, poderemos utilizar a instrução a seguir:

```
DROP TABLE IF EXISTS aluno;
```

Como resultado, aparecerá uma mensagem de alerta identificando que há uma restrição de chave estrangeira que impede a exclusão. Mas, vamos supor que essa ação de exclusão da tabela "aluno" fosse inevitável e, após sua análise, a tabela não fosse mais necessária em seu banco de dados. Para você poder exclui-la, você terá que antes excluir a restrição. Para instruir o MySQL para excluir a restrição da tabela "nota", que faz referência à tabela "aluno" (pai), você deve executar:

```
ALTER TABLE nota DROP FOREIGN KEY nota_ibfk_1;
```

Após essa instrução, você poderá executar a exclusão com sucesso.

Esse é o processo que deve ser feito ao se alterar uma estrutura: certificar-se de todas as restrições que estejam declaradas em seu banco de dados para, então, determinar estas alterações (MACHADO, 2014).

Aqui torna-se fundamental reforçar a importância da inclusão de restrições adequadas para evitar erros graves quando da execução do comando `DROP TABLE`. A exclusão de tabelas implica na exclusão de todos os dados, e, em um sistema em que os backups não são instantâneos (leia-se: quase todos os sistemas em funcionamento), a remoção inadvertida de uma tabela implicará em perda de dados. Caso se trate de um sistema de faturamento, por exemplo, um comando de remoção de tabelas erroneamente aplicado pode implicar em prejuízos para a organização. Essa é a razão principal de termos estudado as restrições, seu funcionamento e sua sintaxe antes de estudarmos a exclusão de tabelas.

A instrução `DROP TABLE` não deve ser confundida com a instrução que apagará todo o conteúdo de uma tabela, ou seja, esvaziar sua tabela. A instrução para esvaziar uma tabela completamente é a `TRUNCATE TABLE`, e sua sintaxe é:

```
TRUNCATE [TABLE] nome_tabela;
```

Logicamente, `TRUNCATE TABLE` é semelhante a uma instrução `DELETE` (que exclui todas as linhas) ou a uma sequência de instruções `DROP TABLE` e `CREATE TABLE`.

Reflita

Embora o comando `TRUNCATE TABLE` seja semelhante ao comando `DELETE`, ele é classificado como uma instrução **DDL** (Linguagem de definição de dados) em vez de uma instrução **DML** (Linguagem de manipulação de dados). Por que o comando `TRUNCATE` difere do comando `DELETE`? Haveria coerência em classificarmos o comando `DELETE` como parte da **DML**? Por quê?

A instrução `TRUNCATE TABLE` falha para uma tabela, se houver alguma restrição FOREIGN KEY de outras tabelas que a referenciam, já restrições de chaves estrangeiras entre colunas da mesma tabela são permitidas. Contando que a definição da tabela seja válida, a tabela pode ser recriada como uma tabela vazia com `TRUNCATE TABLE`, mesmo que os dados ou arquivos de índice tenham sido corrompidos.

O campo `AUTO_INCREMENT` é o campo que vai automaticamente sendo incrementado em cada registro, funcionando naturalmente como chave primária, caso o usuário deseje. Qualquer valor `AUTO_INCREMENT` é redefinido para seu valor inicial.

Existe também um recurso que deve ser utilizado com cuidado: podemos instruir o MySQL a, em determinadas circunstâncias especiais, mesmo que existam restrições em seu banco de dados, ignorá-las. Observe a instrução abaixo (ORACLE, 2018):

```
SET FOREIGN_KEY_CHECKS = 1;
```

Ao executar essa instrução, o MySQL irá verificar antes da exclusão quais são as restrições impostas na estrutura do banco de dados. Se definido como 1 (padrão), as restrições de chave estrangeira para tabelas são verificadas. Se definido como 0, as restrições de chave estrangeira serão ignoradas, com algumas exceções. Ao recriar uma tabela que foi descartada, um erro será retornado, caso a definição da tabela não esteja de acordo com as restrições de chave estrangeira que fazem referência à tabela. Da mesma forma, uma operação `ALTER TABLE` retornará um erro se uma definição de chave estrangeira for formada incorretamente.

Assimile

Definir *`foreign_key_checks`* como 1 não aciona uma varredura dos dados da tabela existente. Portanto, as linhas adicionadas à tabela enquanto a condição *`foreign_key_checks = 0`* for verdadeira não serão verificadas quanto à consistência.

Normalmente, essa configuração permanece ativada durante a operação normal para impor a integridade referencial. Contudo, é importante observar que desabilitar a verificação de chave

estrangeira pode ser útil para recarregar as tabelas em uma ordem diferente da exigida pelos seus relacionamentos.

Em nosso exemplo, uma eventual tentativa de excluir a tabela "curso" não seria bem-sucedida, pois existe uma restrição na tabela "nota" fazendo sua referência e, assim, impedindo sua exclusão. Mas, ao executar a instrução `SET FOREIGN_KEY_CHECKS = 0;`, o MySQL está sendo instruído a ignorar quaisquer restrições que estejam criadas. Em seguida, executando a instrução `DROP TABLE IF EXISTS curso;`, a exclusão da tabela "curso" ocorrerá com sucesso. Então, após isso, você pode fazer com que o MySQL volte ao seu estado padrão, utilizando todas as restrições existentes para mudanças de estrutura, ao executar a instrução `SET FOREIGN_KEY_CHECKS = 1;`.

Pesquise mais

No MySQL existem as chamadas **variáveis de ambiente**, ou **variáveis de sistema**. Essas variáveis podem ser utilizadas para alterar o comportamento do SGBD. Para conhecer todas elas e seus comportamentos, pesquise o link a seguir:

ORACLE CORPORATION. InnoDB Startup Options and System Variables. In: Oracle Corporation. **MySQL 8.0 Reference Manual**. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html>. Acesso em: 25 jul. 2018.

Sem medo de errar

Trabalhando no desenvolvimento do Guia Turístico, houve, no início do projeto, um requisito de criar uma tabela que pudesse armazenar informações de GPS (latitude e longitude). Essa tabela estaria relacionada com uma de pontos turísticos justamente para referenciá-los. Porém, agora notou-se que não há a necessidade de manter essas informações em uma tabela separada. Para resolver esse problema, iremos alterar a estrutura da tabela de pontos turísticos. Sua estrutura era assim:

```
CREATE TABLE IF NOT EXISTS ponto_tur (
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(50) NOT NULL DEFAULT '',
```

```
        populacao INT(11) NOT NULL DEFAULT '0',
        tipo ENUM('Atrativo','Serviço','Equipamento',
'Infraestrutura', 'Instituição', 'Organização'),
        publicado ENUM('Não', 'Sim') NOT NULL DEFAULT
'Não'
    );
```

Para alterá-la, devemos utilizar a seguinte instrução:

```
ALTER TABLE ponto_tur ADD coordenada GEOMETRY;
```

Note que não adicionamos dois campos de latitude e longitude, e isso ocorre por que, no MySQL, já é possível criar campos do tipo "*Geometry*".

Nesse campo, podemos trabalhar com coordenadas, mapas geográficos e, especificamente, uma coordenada pode ser entendida como um ponto com duas referências, com número contendo várias casas decimais, por exemplo: **POINT(-23.5111264 -47.4461944)**. A tabela criada anteriormente para esta função poderá ser excluída, bastando utilizar a instrução:

```
DROP TABLE IF EXISTS coordenada;
```

Agora dê continuidade às solicitações, seguindo o modelo acima e os comandos aprendidos nessa seção, para alterar a tabela "Países", incluindo uma nota de 0 a 10, denotando o interesse turístico do país em questão. Por fim, use o mesmo mecanismo para alterar a tabela "Cidades", incluindo uma lista que conterá os três melhores restaurantes da cidade em questão.

Avançando na prática

Importação de arquivo CSV em banco de dados

Descrição da situação-problema

Você acaba de receber um arquivo CSV com mais de 20.000 linhas, contendo as seguintes colunas: Idprojeto; Responsável; Endereço; GPS e Data.

Você tem um banco de dados em que estão armazenadas todas as informações de uma construtora, e sua área de projetos acaba de receber uma lista de projetos de uma aquisição feita do seu concorrente. Os projetos estão relacionados por Empresa, Região e Pesquisa Social. Em todos estes relacionamentos, foram criadas restrições para atualização de dados as referenciando. As pesquisas detêm informações relacionadas à projeção de clientes, renda per capita e outras informações relacionadas ao projeto. Você deverá importar essas informações e, em sua tabela de projetos, os cinco campos também existem. A tabela "projetos" é uma tabela filha. Porém, no arquivo, você não terá as chaves estrangeiras que determinam os relacionamentos dessa tabela com outras. Como você deve importar este arquivo?

Resolução da situação-problema

A instrução para esse tipo de importação é bem simples:

```
USE construtora;
LOAD DATA INFILE projeto.csv' INTO TABLE projeto
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

Essa instrução utiliza o banco de dados "construtora" e, em sua tabela "projetos", ela irá inserir todas as linhas do arquivo projetos.csv. Os campos são separados por vírgula e identificados por aspas. O cabeçalho do arquivo na primeira linha é ignorado.

Porém, há um problema aí, pois foi determinado que essa tabela é filha e tem várias restrições criadas, conforme a instrução abaixo:

```
CREATE TABLE projeto (
     idProjeto INT NOT NULL,
     idRegiao INT NOT NULL,
     idPesquisa INT NOT NULL,
     responsável VARCHAR(50),
     endereco VARCHAR(100),
```

```
        gps GEOMETRY,
        data DATE NOT NULL,
        PRIMARY KEY(idProjeto),
        FOREIGN KEY (idRegiao) REFERENCES
regiao(idRegiao) ON DELETE
            CASCADE,
        FOREIGN KEY (idPesquisa) REFERENCES
pesquisa(idPesquisa) ON
            DELETE RESTRICT
    );
```

Portanto, para que isso funcione, deverá seguir estas instruções:

```
USE construtora;
SET FOREIGN_KEY_CHECKS = 0;
LOAD DATA INFILE projeto.csv' INTO TABLE projeto
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
SET FOREIGN_KEY_CHECKS = 1;
```

Isso determinará que o MySQL ignore as restrições, importe do arquivo e depois volte ao seu padrão.

Faça valer a pena

1. Num banco de dados, algumas informações de determinadas tabelas deverão ser adicionadas provenientes duma estrutura idêntica, porém, os dados poderão ter algumas chaves primárias com dados que poderão se repetir. As restrições criadas previamente poderão interferir na atualização dos dados.

Para inserir esses dados sem a restrição de chaves estrangeiras, primeiro deverá ser executada a instrução:

a) `SET FOREIGN_KEY_CHECKS = 0.`
b) `SET FOREIGN_KEY_CHECKS = 10.`

c) `SET FOREIGN_KEY_CHECKS = 012.`
d) `SET PRIMARY_KEY_CHECKS = 0.`
e) `SET STRANGER_KEY_CHECKS = 0.`

2. Num determinado banco de dados, você tem armazenadas todas as preferências de seus clientes. Isso foi feito para armazenar dado, que possam direcionar determinadas ofertas de um comércio eletrônico, relacionando as preferências às promoções da semana. Porém, um erro foi detectado, pois uma das opções de preferências é "outros...". Muitas pessoas estão com essa opção relacionada a sua preferência e, por isso, as campanhas promocionais não estão adequadas.

A determinação de toda a diretoria comercial foi que essa informação não terá mais validade e, portanto, essa tabela deverá ser esvaziada. Ao utilizar a instrução adequada para tal esvaziamento, você está utilizando uma instrução do subconjunto:

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

3. Você está recebendo em sua empresa um software administrativo contábil. Esse software tem o MySQL como seu banco de dados e já era utilizado em outra empresa. Porém, as naturezas de operação das empresas não correspondem e todo o plano contábil deverá ser apagado, sendo completamente refeito.

Para determinar as restrições de uma tabela, você utilizará a instrução:

a) `USE.`
b) `TRUNCATE.`
c) `SHOW TABLES.`
d) `SHOW CREATE TABLE.`
e) `SHOW DATABASES.`

Referências

CARDOSO, V.; CARDOSO, G. **Linguagem SQL** - Fundamentos e Práticas. 1. ed. São Paulo: Saraiva, 2013. 193p.

MACHADO, F. N. R. **Banco de Dados** - Projeto e Implementação. 3. ed. São Paulo: Érica, 2014. 400p.

MANZANO, J. A. N. G. **MySQL 5.5 interativo**: guia essencial de orientação e desenvolvimento. São Paulo: Érica, 2011. 240p.

ORACLE CORPORATION. **MySQL 8.0 Reference Manual**. 24 jul. 2018. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/>. Acesso em: 24 jul. 2018.

Consultas avançadas

Convite ao estudo

Caro aluno, você já parou para pensar em como são organizados os bancos de dados de grandes companhias, tais como uma instituição financeira? Comumente, para a maioria dos sistemas, bancos de dados possuem diversas tabelas e seus respectivos relacionamentos, que permitem realizar consultas cruzando os dados entre as tabelas. Quando utilizamos um caixa eletrônico para fazer operações como saque, depósito ou empréstimo, o sistema necessita fazer consultas em diversas tabelas ao mesmo tempo. Aliás, a maioria das consultas que fazemos nos sistemas, é feita cruzando dados de muitas tabelas. É lógico que essas operações não são perceptíveis ao usuário. Agora, imagine como é complexo trabalhar com toda esta volumetria de tabelas e correlações. Uma instituição financeira tem centenas de clientes, cada um deles fazendo transações diversas e simultaneamente. Isso faz com os selects também se tornem cada vez mais complexos. Para isso, existem algumas técnicas que podem ser aplicadas para auxiliar a manipulação de dados.

As técnicas que serão discutidas nesta unidade são recursos que possibilitam a você, futuro desenvolvedor, conhecer e compreender a criação e manipulação de tabelas para funções avançadas. Assim, com esses conhecimentos você será capaz de elaborar scripts SQL para consultas avançadas em tabelas.

Você foi contratado por uma empresa para atuar em um projeto que busca soluções para todos os problemas relacionados ao banco de dados de um cliente, que é uma loja de games. Essa empresa está estabelecida em uma loja física, e ao longo de vários anos sempre se manteve atualizada com as novidades relacionadas aos lançamentos de jogos e consoles. Porém, o gerenciamento da loja sempre foi feito de forma manual, o que acarretou algumas insatisfações

dos clientes, devido ao elevado tempo de atendimento em certos momentos. Para dar mais mobilidade no ambiente da loja e realizar um pré-atendimento, o que poderia agilizar e modernizar a forma de atendimento, o gerente da loja solicitou que fossem desenvolvidos tokens para serem espalhados pela loja. Sua responsabilidade para que o cliente fique satisfeito com o desenvolvimento da aplicação é muito grande, e a sua equipe conta com você. Quais relações devem ser desenvolvidas entre as tabelas do banco de dados para que o sistema seja funcional após a automatização? Quais comandos avançados devem ser utilizados para agilizar a comunicação entre as tabelas? Quais são os desafios ao se lidar com a automatização de um banco de dados cujas tabelas já estão povoadas? Há necessidade de revisitar o DER para este processo de automatização?

Para resolver essa situação, na primeira seção desta unidade de ensino, você irá estudar como realizar junções horizontais e verticais, aprendendo a estrutura e os comandos para estas junções. Na segunda seção, você irá ser apresentado às funções de agregação, seus comandos, estrutura e exemplos. E, para finalizar, na Seção 3, você irá estudar as subconsultas em banco de dados, os comandos, a estrutura desse tipo de consulta e verá exemplos.

Dessa forma, você vai conhecer e compreender a criação e manipulação de tabelas para funções avançadas.

Bons estudos!

Seção 3.1

Junção horizontal e vertical de dados

Diálogo aberto

Caro aluno, imaginemos um hospital que tem sua estrutura organizada em um banco de dados que contém a relação de todos os pacientes. Além da tabela com os dados do paciente, em que consta qual é o seu plano de saúde, há também uma tabela das condições do plano de saúde, por exemplo, se prevê quarto individual ou enfermaria, ou, ainda, a cobertura de exames. Quando um paciente dá entrada, é necessário relacionar horizontalmente essas duas tabelas (cadastro de clientes e planos de saúde) para saber onde o paciente será internado e por quais exames ele deverá pagar, além do plano que possui. Este é apenas um relacionamento entre tabelas, porém, isso pode ocorrer entre diversas tabelas e também de forma vertical.

A empresa em que você trabalha foi contratada por uma loja que vende jogos de videogame e consoles, que deseja automatizar o atendimento aos seus clientes com *tokens*. Assim, por meio de alguns toques na tela (sistema *touchscreen*, usado em *tablets* e *smartphones*), o cliente conseguirá localizar os títulos, valores de jogos, e demais informações. O primeiro problema a ser endereçado por este projeto é a questão da mobilidade dos clientes na localização dos jogos dentro da loja. Inicialmente, foi desenvolvida a sintaxe do banco de dados, descrita no Quadro 3.1.

Quadro 3.1 | Script de criação de BD e tabelas

```
1  create database SuperGames;
2  use SuperGames;
3
4  create table localizacao(
5     Id int(3) primary key auto_increment,
6     Secao varchar(50) not null,
7     Prateleira int(3) not null
8  );
9
```

```
10  create table jogo(
11     Cod int(3) primary key auto_increment,
12     Nome varchar(50) not null,
13     Valor decimal(6, 2) not null,
14     Localizacao_Id int(3) not null,
15     foreign key (Localizacao_Id) references
16  Localizacao(Id)
    );
```

Fonte: elaborado pelo autor.

Sua tarefa inicial será desenvolver as sintaxes para que a aplicação seja capaz de:

- Identificar o nome do jogo e a prateleira, dando o nome de uma seção;
- Identificar todas as seções e os respectivos nomes dos jogos, ordenando as seleções em ordem crescente pelo nome dos jogos.
- Identificar o nome dos jogos da seção de jogos de guerra, por serem os mais procurados.

Para entregar uma versão prévia para aprovação do cliente, ele sugeriu que você comece trabalhando com as seções de Guerra (prateleiras 1 e 2), Aventura (prateleiras 100 e 101) e RPG (prateleiras 150 e 151). Como exemplos de jogos, utilize: COD 3 (R$ 125.00, Guerra), BF (R$ 150.00, Guerra), GOW 4 (R$ 200.00, Aventura), SLY (R$ 99.00, Aventura) e FF XV (R$ 205.00, RPG.

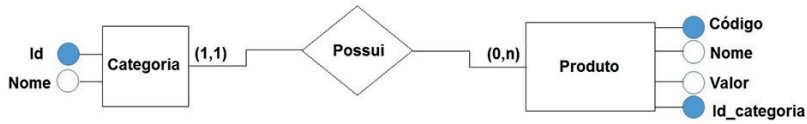
Para desenvolver essa tarefa, você precisa conhecer a estrutura de junções horizontais e verticais, assim como os comandos aplicados para elas. Aplique todas as técnicas que serão discutidas nesta seção para que você possa se tornar cada vez melhor na programação de banco de dados e desenvolver as suas habilidades. Bons estudos!

Não pode faltar

Caro aluno, você estudou formas de seleções de tuplas em uma única tabela, mas essas técnicas não permitem realizar consultas em múltiplas tabelas, o que faz com que as discussões acerca de banco de dados do tipo relacional, que é o nosso objeto de estudo, sejam colocadas em prática no uso dos *selects*.

Para isso, vamos tomar de exemplo o Diagrama de Entidade Relacionamento representado na Figura 3.1, para as discussões a seguir.

Figura 3.1 | DER de exemplo das Junções



Fonte: elaborada pelo autor.

Observe que, na tabela "Categoria", a coluna "Id" é a chave primária. Já na tabela "Produto", a chave primária é a coluna "Codigo", e a chave estrangeira que relaciona as duas tabelas é o campo "Id_Categoria". Dessa forma, o script em SQL para essa representação pode ser observado no Quadro 3.2:

Quadro 3.2 | Script de exemplo

```
1   create database Loja;
2   use Loja;
3
4   create table Categoria(
5      Id int(3) primary key auto_increment,
6      Nome varchar(50) not null
7   );
8
9   create table Produto(
10     Codigo int(3) primary key auto_increment,
11     Nome varchar(50) not null,
12     Valor decimal(6, 2) not null,
13     Id_Categoria int(3) not null,
14     foreign key (Id_Categoria) references Categoria(Id)
15  );
16
```

Fonte: elaborado pelo autor.

Assim, para que possamos efetuar as consultas relacionais que serão discutidas adiante, é necessário inserir alguns registros em ambas as tabelas, conforme se observa no Quadro 3.3:

Quadro 3.3 | *Inserts* de exemplo

```
1  INSERT Categoria VALUES (0, "DVD"),
2     (0, "Livro"),
3     (0, "Informática");
4
5  INSERT Produto VALUES (0, "Código da Vinci", "39.99", 2),
6     (0, "Hancock", "89.99", 1),
7     (0, "Dario de um Mago", "19.99", 2),
8     (0, "Eu sou a lenda", "39.99", 1);
```

Fonte: elaborado pelo autor.

Dessa forma, ao fazer uma simples seleção nas duas tabelas, teremos os resultados representados nas Figuras 3.2a e 3.2b:

Figura 3.2 | Select: a) na tabela "Categoria"; e b) na tabela "Produto".

a)

```
mysql> SELECT * FROM Categoria;
+----+--------------+
| Id | Nome         |
+----+--------------+
|  1 | DVD          |
|  2 | Livro        |
|  3 | Informática  |
+----+--------------+
3 rows in set (0.00 sec)
```

b)

```
mysql> SELECT * FROM Produto;
+--------+------------------+-------+--------------+
| Codigo | Nome             | Valor | Id_Categoria |
+--------+------------------+-------+--------------+
|      1 | Codigo da Vinci  | 39.99 |            2 |
|      2 | Hancock          | 89.99 |            1 |
|      3 | Dario de um Mago | 19.99 |            2 |
|      4 | Eu sou a lenda   | 39.99 |            1 |
+--------+------------------+-------+--------------+
4 rows in set (0.08 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Segundo Silberschatz (2010), as condições para se efetuar uma junção dependem diretamente do tipo de junção e de uma condição de junção, dessa forma, com o SQL, será possível retornar relações como resultados.

Para isso, temos que:

- Tipo de Junção: define/trata as tuplas em cada uma das relações que não corresponda a alguma das tuplas da outra relação, sendo dividido em relação interna, com o comando

INNER JOIN, e relações externas, LEFT JOIN, RIGHT JOIN e FULL JOIN.

- Condição de Junção: define se as tuplas nas duas relações são correspondentes, garantindo que os atributos utilizados em ambas as tabelas estejam presentes tanto na sintaxe SQL como nos seus resultados.

Para se realizar tais junções nas consultas às tabelas, faz-se necessário a utilização de um dos comandos mais importantes na estrutura SQL, o JOIN e suas variações.

Parâmetro JOIN

Segundo Milani (2007), com a utilização do comando JOIN (Junção) é possível, por meio do SELECT, unir duas ou mais tabelas, ao se apontar os campos correspondentes entre elas. Uma premissa para se garantir a eficiência em sua utilização é a necessidade de que as tabelas existentes no banco de dados estejam normalizadas.

Reflita

Quando um banco de dados é projetado, entre as diversas atividades envolvidas está o planejamento de otimização, que garante que, ao se efetuar as consultas, o tempo de resposta seja o menor possível. A normalização é uma das técnicas que pode auxiliar nessa tarefa. Por qual motivo o comando JOIN exige que o banco de dados esteja normalizado para não comprometer o tempo de resposta nas consultas? Embora um banco de dados não esteja normalizado, ainda assim as consultas retornarão uma resposta?

A sintaxe utilizada para se efetuar junções nas consultas em SQL é definida como:

```
SELECT[campo] FROM [tabela_1>JOIN<tabela_2] ON
     [tabela_1].[chave_primária] =
[tabela_2].[chave_estrangeira]
     WHERE [condição];
```

Caro aluno, repare que, ao demonstrar a sintaxe, a palavra `JOIN` foi destacada em negrito, pois os mecanismos de junção disponíveis a partir da versão SQL-92 permitem três tipos de JOIN: `INNER, LEFT e RIGHT JOIN.`

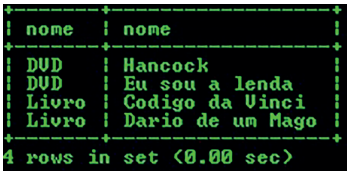
INNER JOIN

No exemplo utilizado ao longo desta seção, em que temos a relação entre categorias e produtos, se quisermos efetuar uma consulta que nos retorne o nome da categoria e seus respectivos nomes dos produtos, deve ser utilizado o seguinte comando:

```
SELECT categoria.nome, produto.nome FROM Categoria INNER JOIN
        Produto
    ON Categoria.Id = Produto.Id_Categoria;
```

Dessa forma, vamos ter a saída demonstrada na Figura 3.3:

Figura 3.3 | Select **JOIN**.



Fonte: captura de tela do MySQL, elaborada pelo autor.

É importante que você compreenda alguns detalhes desse comando:

- "`SELECT categoria.nome, produto.nome`": repare que o campo "nome" está tanto na tabela "Categoria" como na tabela "Produto", assim, para que o SQL possa distingui-las, devemos utilizar a seguinte sintaxe: <tabela>.<coluna>.
- "`FROM Categoria` **`INNER JOIN`** `Produto`": o comando INNER JOIN promove a junção das tabelas "Categoria" e "Produto", dessa forma é possível fazer o produto cartesiano entre ambas.

- "ON Categoria.Id = Produto.Id_Categoria;": a palavra "ON" tem a função de fazer o apontamento da chave primaria da tabela "Categoria" para a chave estrangeira da tabela "Produto".

Ainda, se assim quisermos, é possível utilizar as condições nas consultas, quando for necessário fazer as junções entre as tabelas. Para exemplificarmos tal técnica, tomamos como exemplo uma consulta na qual se deseja exibir o nome da categoria como "Tipo", o nome do produto como "Produto" e o valor dos produtos, em uma condição que o valor dos produtos seja menor que R$ 50,00. O comando SQL é descrito a seguir:

```
SELECT categoria.nome as "Tipo", produto.nome as "Produto",
produto.valor FROM Categoria INNER JOIN Produto
    ON Categoria.Id = Produto.Id_Categoria
    Whereproduto.valor< 50.00;
```

Sendo gerada a saída mostrada na Figura 3.4:

Figura 3.4 | Select JOIN com condição.

```
+-------+------------------+-------+
| Tipo  | Produto          | valor |
+-------+------------------+-------+
| DVD   | Eu sou a lenda   | 39.99 |
| Livro | Codigo da Vinci  | 39.99 |
| Livro | Dario de um Mago | 19.99 |
+-------+------------------+-------+
3 rows in set (0.00 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Exemplificando

O comando JOIN é utilizado com maior frequência do que você possa imaginar. Vejamos alguns exemplos, utilizados em aplicações web:

- Sites de hospedagem: são necessárias relações entre tabelas diferentes para gerar o resultado da pesquisa. Assim é possível determinar data de entrada/saída, categoria, distância do centro da cidade, e opcionais (piscina, WiFi, café da manhã etc.);
- Sites de compra: assim como no exemplo anterior, é possível relacionar o local do vendedor, a amplitude de valor, a

relevância do anúncio, o tipo de pagamento e a forma de envio;

- Buscadores Web: são aplicações que efetuam buscas na web por meio de palavras-chave digitadas pelo usuário. Os mais conhecidos são Google, Bing e Yahoo. Neles, é possível utilizar filtros por data, tipo de conteúdo, idioma, país de origem etc.

As consultas realizadas até o foram internas, mas também é possível realizar consultas externas.

JUNÇÃO EXTERNA

Segundo Silberschatz (2010), quando o operador de junção externa for utilizado no SQL, é gerado o resultado da junção mais as linhas não combinadas. É possível efetuar junções externas em ambos os lados, ou seja, da esquerda para a direita, e da direita para a esquerda, dessa forma, a junção externa, independentemente do lado escolhido, gera uma nova tabela, que é a junção das linhas combinadas e não combinadas.

LEFT JOIN

Conforme afirma Silberschatz (2010), no comando `LEFT JOIN`, as linhas da tabela da esquerda são projetadas na seleção juntamente com as linhas não combinadas da tabela da direita. Ou seja, como resultado dessa seleção, algumas linhas em que não haja relacionamento entre as tabelas da esquerda para a direita retornarão o valor nulo (`NULL`). Para isso, o SQL utiliza a seguinte sintaxe:

```
SELECT [campo] FROM [tabela_1] LEFTJOIN [tabela_2] ON
          [tabela_1].[chave_primária]           =
[tabela_2].[chave_estrangeira]

          WHERE [condição];
```

Caro aluno, para sua melhor compreensão vamos tomar o seguinte exemplo:

```
SELECT categoria.nome as "Tipo", produto.nome as
"Produto",
      produto.valor FROM Categoria LEFT JOIN Produto
       ON Categoria.Id = Produto.Id_Categoria;
```

Com isso, é gerada a saída apresentada na Figura 3.5:

Figura 3.5 | Select LEFT JOIN.

```
+-------------+-----------------+--------+
| Tipo        | Produto         | valor  |
+-------------+-----------------+--------+
| Livro       | Codigo da Vinci | 39.99  |
| DVD         | Hancock         | 89.99  |
| Livro       | Dario de um Mago| 19.99  |
| DVD         | Eu sou a lenda  | 39.99  |
| Informática | NULL            |  NULL  |
+-------------+-----------------+--------+
5 rows in set (0.00 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

No exemplo demonstrado na Figura 3.5, com o trecho do código "FROM Categoria LEFT JOIN Produto" são projetados todos os registros da tabela da esquerda e, na última seleção, a linha não combinada, pois não existe nenhum produto associado à categoria "Informática", consequentemente as colunas "Produto" e "Valor" receberam nulo (NULL) como entrada.

Assimile

Os métodos de normalização de banco de dados são divididos em: primeira forma normal (1FN), em que não podem ocorrer atributos com mais de um valor; segunda forma normal (2FN), em que todos os atributos não chaves devem depender unicamente da chave primária da tabela; e terceira forma normal (3FN), em que os atributos devem ser mutuamente independentes, eliminando assim as funções transitivas.

Ao colocar as tabelas nas três formas normais, podemos garantir menor taxa de processamento, principalmente nas consultas em que são utilizadas junções.

RIGHT JOIN

Conforme afirma Silberschatz (2010), similar ao comando LEFT JOIN, com o comando RIGHT JOIN as linhas da tabela da direita são projetadas na seleção juntamente com as linhas não combinadas da tabela da esquerda. Para isso, o SQL utiliza a seguinte sintaxe:

```
SELECT [campo] FROM [tabela_1] RIGHTJOIN [tabela_2]
    ON     [tabela_1].[chave_primária]           =
[tabela_2].[chave_estrangeira]
    WHERE [condição];
```

Vejamos um exemplo:

```
SELECT categoria.nome as "Tipo", produto.nome as
"Produto",
produto.valor FROM Categoria RIGHT JOIN Produto
ON Categoria.Id = Produto.Id_Categoria;
```

Assim, é gerada a saída demonstrada na Figura 3.6:

Figura 3.6 | Select `RIGHT JOIN`

```
+-------+-----------------+-------+
| Tipo  | Produto         | valor |
+-------+-----------------+-------+
| DVD   | Hancock         | 89.99 |
| DVD   | Eu sou a lenda  | 39.99 |
| Livro | Codigo da Vinci | 39.99 |
| Livro | Dario de um Mago| 19.99 |
+-------+-----------------+-------+
4 rows in set (0.00 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

No exemplo demonstrado na Figura 3.6, o trecho do código "`FROM Categoria RIGHT JOIN Produto`" projeta todos os registros da tabela da direita. Como não existe nenhum produto associado à categoria "Informática", consequentemente nenhum registro do tipo nulo (`NULL`) é demonstrado na seleção dos registros.

Pesquise mais

Os bancos de dados não são somente do tipo relacional. Com a crescente utilização de *big data,* os bancos de dados não relacionais ganharam destaque por sua eficiente aplicação em grandes volumes de dados. O vídeo do BeabáTec demonstra os conceitos e aplicações no sistema de gerenciamento de banco de dados, MongoDB.

BEABÁTEC. **Bancos NoSQL –** Um Exemplo Com o MongoDB. 20 dez. 2016. Disponível em: <https://www.youtube.com/watch?v=ghDRQwSGXak>. Acesso em: 20 jun. 2018.

Sem medo de errar

Você está trabalhando em uma empresa contratada para desenvolver tokens interativos para localização de jogos de vídeo game e consoles em uma loja. Foi atribuído a você o desenvolvimento de três *scripts* SQL para identificação de jogos, prateleiras, categorias e preços. Para sua primeira prévia do projeto, você deve utilizar as seções de Guerra (prateleiras 1 e 2), Aventura (prateleiras 100 e 101) e RPG (prateleiras 150 e 151). Como exemplos de jogos, utilize: COD 3 (R$ 125.00, Guerra), BF (R$ 150.00, Guerra),GOW 4 (R$ 200.00, Aventura), SLY (R$ 99.00, Aventura) e FF XV (R$ 205.00, RPG).

Antes de construir os comandos solicitados, é necessário efetuar algumas inserções de dados na tabela da estrutura, que já foi apresentada (Quadro 3.1). Podem ser inseridos os registros solicitados por meio do script SQL representado no Quadro 3.4.

Quadro 3.4 – Inserts SuperGames

```
INSERT localizacao VALUES (0, "Guerra", "001"),
   (0, "Guerra", "002"),
   (0, "Aventura", "100"),
   (0, "Aventura", "101"),
   (0, "RPG", "150"),
   (0, "RPG", "151");

INSERT jogo VALUES (0, "COD 3", 125.00, 1),
   (0, "BF 1", 150.00, 2),
   (0, "GOW 4", 200.00, 3),
   (0, "SLY", 99.00, 4),
   (0, "FF XV", 205.00, 5);
```

Fonte: elaborado pelo autor.

Note que ao inserirmos dados na tabela "Values" os códigos, por ser chave primária, eles serão auto incrementados.

Com isso, é possível desenvolver os *scripts* dos três filtros e testá-los por meio de suas respectivas saídas:

1. Identificar o nome do jogo e a prateleira, fornecendo o nome de uma seção;

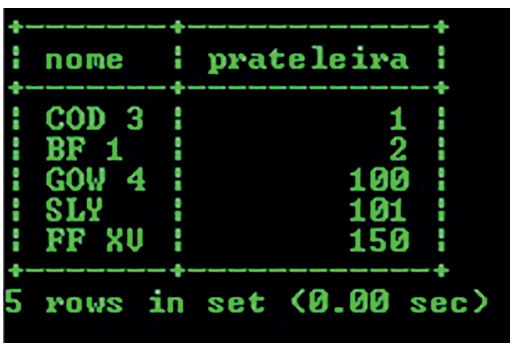
Para isso, foi desenvolvido:

```
SELECT jogo.nome, localizacao.prateleira FROM jogo INNER JOIN

          localizacao ON localizacao.Id = jogo.localizacao_Id;
```

Assim, foi gerada a saída demonstrada na Figura 3.7:

Figura 3.7 | Saída Filtro 1



Fonte: captura de tela do MySQL, elaborada pelo autor.

2. Identificar o nome dos jogos da seção de jogos de guerra.

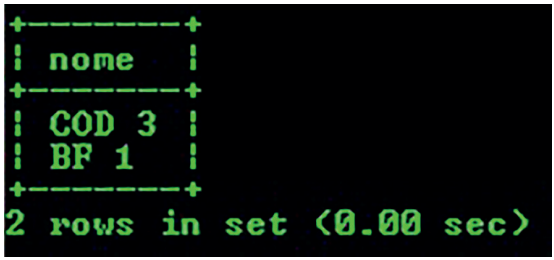
Para isso, foi desenvolvido:

```
SELECT jogo.nome FROM jogo INNER JOIN localização

ON localizacao.Id = jogo.localizacao_Idwheresecao = "Guerra";
```

O que gerou a saída demonstrada na Figura 3.8:

Figura 3.8 | Saída Filtro 2



Fonte: captura de tela do MySQL, elaborada pelo autor.

3. Identificar todas as seções e os respectivos nomes dos jogos, ordenando as seleções em ordem crescente pelo nome dos jogos.

Para isso foi desenvolvido:

```
SELECT localizacao.secao, jogo.nome FROM localizacao LEFT JOIN jogo
    ON localizacao.Id =
jogo.localizacao_Idorderbyjogo.nomeasc;
```

Que gerou a saída demonstrada na Figura 3.9:

Figura 3.9 | Saída Filtro 3

Fonte: captura de tela do software MySQL, elaborada pelo autor.

Feito isso, o desenvolvedor da aplicação poderá utilizar tais *scripts* para que os *tokens* auxiliem os usuários dentro da loja, aumentando assim o nível de satisfação dos clientes.

Avançando na prática

Filtros para site de hospedagem de pets

Descrição da situação-problema

Os donos de pets enfrentam dificuldades com seus animais quando necessitam sair para trabalhar ou fazer uma viagem. Para isso, um site propõe cadastrar cuidadores e donos de animais para que, por meio de um filtro, possa ocorrer o encontro entre as partes, possibilitando a hospedagem dos pets. Sua empresa foi contratada para este trabalho. inicialmente, foi desenvolvido o banco de dados por meio da linguagem SQL, representado no Quadro 3.5:

Quadro 3.5 | *Script pets*

```
1   create table Pet(
2      Id int(3) primary key auto_increment,
3      Nome varchar(50) not null,
4      Raca varchar(50) not null
5   );
6
7   create table Dono(
8      Cod int(3) primary key auto_increment,
9      Nome varchar(50) not null,
10     Pet_Id int(3) not null,
11     foreign key (Pet_Id) references Pet(Id)
12  );
13
14  create table Cuidador(
15     Codigo int(3) primary key auto_increment,
16     Nome varchar(50) not null,
17     Telefone int(10) not null
18  );
19
20  create table Hospedagem(
21     Num int(3) primary key auto_increment,
22     Pet_Id int(3) not null,
23     Cuidador_Codigo int(3) not null,
24     foreign key (Pet_Id) references Pet(Id),
25     foreign key (Cuidador_Codigo) references
```

```
26  Cuidador(Codigo)
    );
```

Fonte: elaborado pelo autor.

O cliente pediu, então, que, para auxiliar no gerenciamento das hospedagens, seja criado um filtro que retorne o número da hospedagem (como "Registro"), o nome do animal (como "Pet"), o nome do respectivo dono (como "Proprietário") e o nome do cuidador responsável pela hospedagem ("Cuidador"). A seleção de dados deve ser exibida em ordem crescente, a partir do nome do pet. Você precisa, então, desenvolver este procedimento.

Resolução da situação-problema

Soluções para aplicações devem concatenar algumas junções, como no exemplo do banco de dados para hospedagem de pets, em que se tem quatro tabelas e três delas estão relacionadas: "Hospedagem" com "Pets", "Hospedagem" com "Cuidador"; já a tabela "Dono" apenas se relaciona com a tabela "Pets".

Para fazer o filtro temos que:

```
Select hospedagem.num as "Registro", Pet.Nome as "Pet", Dono.Nome

as "Proprietário", Cuidador.Nome as "Cuidador"

from Hospedagem inner join Pet

on Hospedagem.Pet_Id = Pet.Id

inner join Dono

on Dono.Pet_id = pet.id

innerjoin cuidador

on Hospedagem.Cuidador_codigo = Cuidador.Codigo

order by Pet.Nome;
```

Quando você simular os registros, será gerada a saída representada na Figura 3.10:

Figura 3.10 | Select pets

Registro	Pet	Proprietário	Cuidador
3	Joe	Manuel	Ana
2	Lulu	Maria	Mariana
1	Totó	Joao	Juliana

3 rows in set (0.00 sec)

Fonte: captura de tela do MySQL, elaborada pelo autor.

Faça valer a pena

1. Uma empresa mantém o cadastro dos seus colaboradores em uma base de dados e seus respectivos dependentes em outra tabela. Isso foi feito para fins de convênio médico, salário família, entre outros benefícios concedidos aos colaboradores. A estrutura do banco de dados foi desenvolvida da seguinte forma:

Tbl_Funcionarios (Id_Funcionario, Nome_Funcionario)
Tbl_Dependentes (Codigo, Nome_Dependente, Parentesco, Id_Funcionario)

Onde as colunas com simples sublinhado representam as chaves primárias, e as com duplo sublinhado a chave estrangeira.

Foi solicitada a elaboração de uma consulta utilizando a linguagem de programação de banco de dados SQL, na qual se deseja retornar o nome de cada funcionário e o nome do seu respectivo dependente (ou nulo, quando o colaborador não possuir dependente).
Assinale a alternativa que descreva corretamente o filtro solicitado (SELECT, da linguagem SQL):

a)
```
SELECT Funcionarios.Nome, Dependentes.Nome
       FROM Funcionarios, Dependentes
       WHERE Funcionarios.Id_Funcionarios =
Dependentes.Id_Funcionario;.
```

b)
```
SELECT Funcionarios.Nome, Dependentes.Nome
       FROM Funcionarios, Dependentes
       WHERE
Dependentes.Id_Funcionario          =Funcionarios.Id_
Funcionarios;.
```

c)
```
SELECT Funcionarios.Nome, Dependentes.Nome
FROM Funcionarios INNER JOIN Dependentes
ON    Funcionarios.Id_Funcionarios    =    Dependentes.Id_
Funcionario;.
```

d) `SELECT Funcionarios.Nome, Dependentes.Nome`
`FROM Funcionarios RIGHT JOIN Dependentes`
`ON Funcionarios.Id_Funcionarios = Dependentes.Id_Funcionario;.`

e)`SELECT Funcionarios.Nome, Dependentes.Nome`
`FROM Funcionarios RIGHT JOIN Funcionarios`
`ON Funcionarios.Id_Funcionarios = Dependentes.Id_Funcionario;.`

2. Os bancos de dados do tipo relacional estão presentes na maioria das infraestruturas de TI, sendo muito comuns em sistemas de gestão em diversos seguimentos. São vantajosos quando se necessita dos dados organizados, seguindo criteriosamente as regras de normalização, para se obter maior performance nas consultas em que haja maior demanda de processamento. Embora as junções aumentem a taxa de utilização do servidor em suas consultas, é imprescindível que um banco de dados, por meio da linguagem de programação de banco de dados, como o SQL, permita efetuar tais consultas.

As técnicas SQL denominadas JOIN possibilitam efetuar consulta em duas tabelas ou mais, dependendo da necessidade da busca. Nesse contexto, observe as afirmativas a seguir:

I. Uma relação `INNER JOIN` pode exibir todos os campos das tabelas da relação.
II. Uma relação `LEFT JOIN` exibe todos os registros da direita e somente os registros que coincidem da tabela da esquerda.
III. Uma relação `RIGHT JOIN` exibe todos os registros da direita e somente os registros que coincidem da tabela da esquerda.

Assinale a alternativa Correta:

a) Somente as alternativas I e II estão corretas.
b) Somente as alternativas I e III estão corretas.
c) Somente as alternativas II e III estão corretas.
d) Somente a alternativa I está correta.
e) As alternativas I, II e III estão corretas.

3. Observe o script a seguir, em que está representado um banco de dados para uma seguradora de veículos e as tabelas "proprietário" e "veículo" são relacionadas por meio da tabela "apólice"

Quadro | Relação entre as tabelas "proprietário" e "veículo"

```
1   create table Proprietario(
2   Id int(3) primary key auto_increment,
3   Nome varchar(50) not null,
4   CPF int(11) not null);
5
6   create table Veiculo(
7   RENAVAN int(15) not null,
8   Placa varchar(7) not null,
9   Marca varchar(30) not null,
10  Modelo varchar(30) not null);
11
12  Create table Apolice(
13  Numeroint(3) primary key auto_increment,
14  Veiculo_RENAVANint(15) not null,
15  Proprietario_Id not null,
16  foreignkey (Veiculo_RENAVAN) references Veiculo (RENAVAN),
17  foreign key (Proprietario_Id) references Proprietario (Id));
```

Fonte: elaborado pelo autor.

O administrador do banco de dados escreveu um comando, porém, parte foi perdida devido a uma falha no sistema operacional, conforme pode ser observado a seguir:

```
selectProprietario.nome as "Segurado", Veiculo.placa as
"Placa",
Apolice.Numero
from Apolice_____________________ Proprietario
onApolice.Proprietario_Id = Proprietario.Id
inner join _____________________
onApolice.Veiculo_RENAVAN = Veiculo.RENAVAN
WHERE ______________________________ = "Jhonny"
order by Apolice.Numeroasc;
```

Assinale a alternativa que completa as lacunas corretamente, para que o comando possa selecionar o nome do segurado Jhonny, as respectivas placas de veículos registradas em seu nome e o número da apólice de seguro, ordenando tal pesquisa pelo número da apólice, em ordem crescente.

a) `innerjoin - Proprietario - Veiculo.RENAVAN.`
b) `rightjoin - Veiculo - Proprietario.Nome.`
c) `leftjoin - Proprietario - Proprietario.Nome.`
d) `innerjoin - Veiculo - Proprietario.Nome.`
e) `left join - Proprietario - Veiculo.RENAVAN.`

Seção 3.2

Funções de agregação em banco de dados

Diálogo aberto

Caro aluno, é possível que você já tenha utilizado algum site de compras on-line. Quando efetuamos uma busca por um produto específico ou por uma classe de produtos utilizando uma palavra chave, obtemos certa quantidade de produtos encontrados. Nessas mesmas buscas, é possível utilizar filtros para descobrir o menor valor ofertado, ou qual o produto mais comprado. Para que esses facilitadores de compra funcionem, os desenvolvedores dos sites comumente utilizam um recurso conhecido por funções de agregação. Esse é o assunto que trataremos na presente seção.

Você trabalha em uma empresa desenvolvendo todas as demandas relacionadas ao banco de dados do projeto para a loja de jogos para videogames e computadores. Na primeira fase, foram desenvolvidas as sintaxes para fazer algumas seleções de dados em mais de uma tabela, e isso possibilitou que o sistema embarcado nos *tokens* de autoatendimento espalhados pela loja pudesse ser utilizado pelos clientes. Apenas para que você se recorde, o banco de dados deste projeto está estruturado como demonstrado nas Figuras 3.11 e 3.12.

Figura 3.11 | *Describe* na tabela Jogos

```
mysql> describe jogo;
+---------------+---------------+------+-----+---------+----------------+
| Field         | Type          | Null | Key | Default | Extra          |
+---------------+---------------+------+-----+---------+----------------+
| Cod           | int(3)        | NO   | PRI | NULL    | auto_increment |
| Nome          | varchar(50)   | NO   |     | NULL    |                |
| Valor         | decimal(6,2)  | NO   |     | NULL    |                |
| Localizacao_Id | int(3)       | NO   | MUL | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
4 rows in set (0.54 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Figura 3.12 | *Describe* na tabela Localização

Fonte: captura de tela do MySQL, elaborada pelo autor.

A segunda parte do projeto prevê o desenvolvimento de algumas funções operacionais/administrativas, pois tais funções eram feitas manualmente pelo gerente da loja. Com isso, o gerente de projetos solicitou que você desenvolva os seguintes scripts para que tais funções sejam agregadas ao sistema de gerenciamento, entre eles, desenvolver funções de agregação que retornem: a quantidade de registros na tabela jogo; o valor do jogo de maior preço (valor); o valor do jogo de menor preço (valor); o valor médio dos jogos de guerra; e que o valor total em estoque na loja.

Nesta seção, para auxiliar o desenvolvimento desse aplicativo, você irá aprender sobre as funções de agregação em programação de bancos de dados, seus comandos, estruturas e alguns exemplos. Bons estudos!

Não pode faltar

Os bancos de dados possibilitam agregar diversas funcionalidades ao desenvolvimento de sistemas. Tais funções devem ser exploradas a fim de permitir que as aplicações ofereçam recursos de consultas avançadas à base de dados. Anteriormente, você pôde compreender as técnicas SQL que permitem efetuar junções nas consultas de duas ou mais tabelas em um banco de dados.

Nesse momento, o seu objeto de estudo são as funções de agregação em SQL, que permitirão desenvolver algumas consultas utilizando os valores das colunas como parâmetro de pesquisa (`SELECT`). Para exemplificar os conceitos e aplicações que veremos, vamos considerar o banco de dados para guardar as informações, conforme demonstrado na Figura 3.13:

Figura 3.13 | DESCRIBE na tabela Veículos

```
mysql> describe Veiculos;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| Id     | int(3)       | NO   | PRI | NULL    | auto_increment |
| Marca  | varchar(30)  | NO   |     | NULL    |                |
| Modelo | varchar(30)  | NO   |     | NULL    |                |
| Valor  | decimal(10,2)| YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

Fonte: captura de tela do software MySQL, elaborada pelo autor.

Considere também que os registros necessários foram inseridos conforme demonstrado na Figura 3.14:

Figura 3.14 | SELECT na tabela Veículos

```
mysql> select * from Veiculos;
+----+--------------+----------+------------+
| Id | Marca        | Modelo   | Valor      |
+----+--------------+----------+------------+
|  1 | BMW          | 320i     | 160000.00  |
|  2 | Mercedes_Benz| C180     | 140000.00  |
|  3 | Hyundai      | Azera    | 120000.00  |
|  4 | Mercedes_Benz| CLA 200  | 140000.00  |
|  5 | BMW          | 328i     | 210000.00  |
|  6 | Volkswagem   | Passat   | 140000.00  |
|  7 | BMW          | 316i     | 115000.00  |
|  8 | Mercedes_Benz| Classe E | 248000.00  |
|  9 | Mercedes_Benz| C 250    | 180000.00  |
| 10 | Jaguar       | XF       | 220000.00  |
| 11 | BMW          | 535i     | 500000.00  |
| 12 | Jaguar       | VZ       |       NULL |
+----+--------------+----------+------------+
12 rows in set (0.00 sec)
```

Fonte: captura de tela do software MySQL, elaborada pelo autor.

Antes de falarmos das funções de agregação propriamente ditas, vale a pena ressaltar os conceitos relacionados à organização de dados em grupo. Date (2000) diz que os resultados das seleções podem ser organizados em grupos, baseados no conteúdo existente em uma ou mais colunas. Para isso, deve ser utilizada a palavra GROUP BY na seguinte sintaxe SQL:

```
SELECT [coluna]
    FROM [tabela]
    GROUP BY [coluna];
```

É possível, ainda, utilizar a clausula WHERE com o agrupamento, com a seguinte sintaxe SQL:

```
SELECT [coluna]
```

```
FROM <tabela]

WHERE [condição]

GROUP BY [coluna];
```

Segundo Silberschatz (2010), as funções agregadas são aquelas que utilizam um multiconjunto de valores como entrada, porém, seu retorno é um único valor. O SQL oferece cinco funções de agregação nativas, entre elas:

- `AVERAGE:` retorna a média de uma consulta.
- `MINIMUM:` retorna o menor valor de uma consulta.
- `MAXIMUM:` retorna o maior valor de uma consulta.
- `TOTAL:` retorna o somatório de uma determinada consulta.
- `COUNT:` retorna a contagem de uma determinada consulta.

COUNT

Segundo Silberschatz (2010), a função agregada `COUNT` permite que se possa contar o número de registros de uma relação. A sintaxe SQL utilizada para tal função é descrita como:

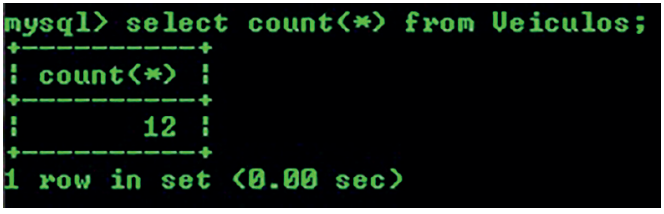
```
SELECT COUNT(*) FROM <tabela>;
```

No exemplo utilizado nessa seção, temos que:

```
SELECT COUNT(*) FROM Veiculos;
```

Esta sintaxe gerará o retorno demonstrado na Figura 3.15:

Figura 3.15 | `COUNT` na tabela Veículos



Fonte: captura de tela do software MySQL, elaborada pelo autor.

Nessa figura, você pode observar que são demonstrados os `INSERTS` da tabela "Veiculos", e podemos perceber que na coluna "Id"

o auto incremento registrou 12 entradas. Com a função `COUNT`, o SQL efetuou a contagem de todas as colunas, utilizando a opção "(*)".

Assimile

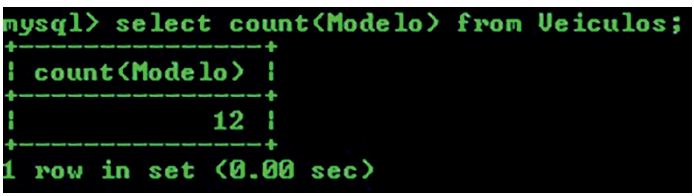
Quando desenvolvemos alguma aplicação que necessite de uma conexão com banco de dados e, por algum motivo, deseja-se efetuar a contagem de registro de uma tabela, devemos atentar às colunas que possuem valores nulos. Se o COUNT em colunas com valor nulo for utilizado em uma função em que algo é calculado, o resultado pode ser totalmente alterado.

É possível, também, em vez de utilizarmos "*", que propicia a contagem de todas as colunas, fazermos o apontamento para uma coluna em especifico. Por exemplo:

```
SELECT COUNT(Modelo) FROM Veiculos;
```

Gerando o retorno demonstrado na Figura 3.16:

Figura 3.16 | `COUNT` em uma coluna específica



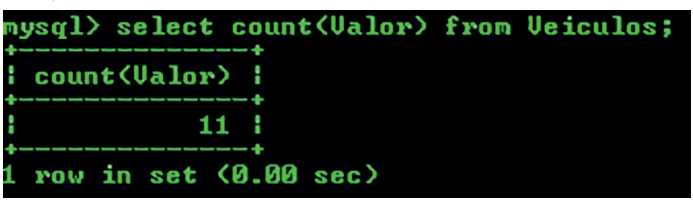
Fonte: captura de tela do software MySQL, elaborada pelo autor.

Porém, temos que tomar cuidado ao utilizar a função `COUNT`, pois o contador ignora os registros em que haja valor nulo (`NULL`), por exemplo:

```
SELECT COUNT(Valor) FROM Veiculos;
```

Gerando o retorno demonstrado na Figura 3.17:

Figura 3.17 | `COUNT` na coluna Valor



Fonte: captura de tela do software MySQL, elaborada pelo autor.

Se tomássemos a coluna "Valor" para determinar a quantidade de veículos registrados no banco de dados, seriamos induzidos ao erro. Portanto, a função *COUNT* não efetua a contagem de tuplas com valor nulo, embora se permita inserir registro em algumas colunas com vazio.

Outro problema encontrado está na utilização da função `COUNT` para contar a quantidade de marcas de carro cadastrados. Certamente o resultado retornado seria doze, ou seja, a quantidade de veículos registrados na tabela. Para isso, o SQL possui a função `DISTINCT`, que é utilizada juntamente com o `COUNT`, e, no exemplo acima, teríamos a seguinte sintaxe:

```
SELECT COUNT( DISTINCT MARCA) FROM Veiculos;
```

Essa sintaxe gera o retorno demonstrado na Figura 3.18:

Figura 3.18 | `DISTINCT` na coluna Marca

Fonte: captura de tela do software MySQL, elaborada pelo autor.

Exemplificando

Imagine que você possua uma tabela com dez mil clientes cadastrados, com informações como Nome, Telefone, Estado e e-mail, e, por algum motivo, queira selecionar os estados com clientes cadastrados. Se não for utilizada a clausula `DISTINCT`, serão retornados dez mil estados (com vários registros repetidos). Se agregarmos a função DISTINCT, o SQL vai selecionar os registros distintamente, evitando informações redundantes.

MINIMUM (MIN)

Segundo Silberschatz (2010), a função agregada `MIN` permite que se possa determinar o menor valor de registro em uma coluna. A sintaxe SQL utilizada para tal função é descrita como:

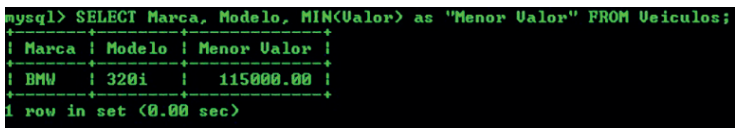
```
SELECT MIN(<coluna>) FROM <tabela>;
```

No exemplo utilizado nessa seção, vamos selecionar a marca, o modelo e o veículo de menor valor registrado na tabela, com o seguinte comando SQL:

```
SELECT Marca, Modelo, MIN(Valor) as "Menor Valor"
FROM Veiculos;
```

Gerando o retorno demonstrado na Figura 3.19:

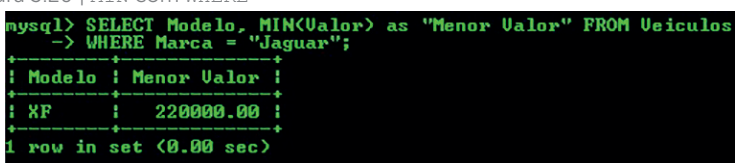
Figura 3.19 | MIN na tabela Veículos



Fonte: captura de tela do software MySQL, elaborada pelo autor.

Também é possível fazer a combinação com WHERE, conforme exemplo demonstrado na Figura 3.20:

Figura 3.20 | MIN com WHERE



Fonte: captura de tela do software MySQL, elaborada pelo autor.

Note que nesse caso, foi selecionado o modelo de menor valor da marca "Jaguar".

MAXIMUM (MAX)

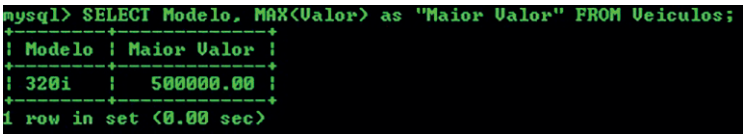
Segundo Silberschatz (2010), analogamente à função MIN, o recurso MAX permite que se possa determinar o maior valor de registro em uma coluna. A sintaxe SQL utilizada para tal função é descrita como:

```
SELECT MAX(<coluna>) FROM <tabela>;
```

No exemplo utilizado nesta seção, vamos selecionar o modelo do veículo de maior valor registrado na tabela, com o comando SQL abaixo, que irá gerar o retorno demonstrado na Figura 3.11.

SELECT Marca, Modelo, MAX(Valor) as "Maior Valor" FROM Veiculos;

Figura 3.21 | MAX na tabela Veículos



Fonte: captura de tela do software MySQL, elaborada pelo autor.

Dessa forma, o SQL encontrou o veículo com o valor maior registrado na tabela.

AVERAGE (AVG)

Segundo Silberschatz (2010), a função AVG (abreviação do termo em inglês *average*, que quer dizer média) retorna a média dos valores em uma determinada coluna. Para isso, o SQL faz a somatória dos valores (obrigatoriamente numéricos) e divide o resultado pelo número de registros diferentes de nulo (NULL).

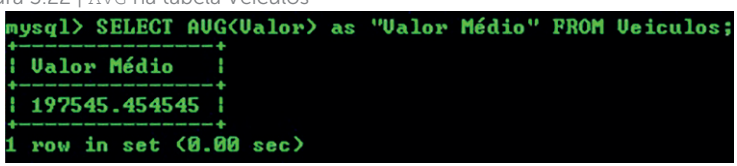
A sintaxe SQL utilizada para tal função é descrita como:

```
SELECT AVG(<coluna>) FROM <tabela>;
```

No exemplo utilizado nessa seção, vamos selecionar o valor médio dos veículos registrados na tabela, com o comando SQL abaixo, que gera o retorno como mostra a Figura 3.22.

```
SELECT AVG(Valor) as "Valor Médio" FROM Veiculos;
```

Figura 3.22 | AVG na tabela Veículos



Fonte: captura de tela do software MySQL, elaborada pelo autor.

Com essa sintaxe sendo aplicada, embora haja 12 veículos registrados (sendo um com o valor nulo), o SQL efetuou o somatório somente dos veículos com valor diferente de nulo, e fez a divisão pelo número de registros com valor também diferente de nulo.

A função de agregação AVG permite que o qualificador GROUP BY seja utilizado em conjunto. Para isso, vamos tomar um exemplo em que se deseja selecionar as marcas e o valor médio dos veículos conforme

as marcas, agrupando as informações, com o comando SQL descrito a seguir:

```
SELECT Marca, AVG(Valor) as "Valor Médio"
    FROM Veiculos
    GROUP BY Marca;
```

Gerando o retorno demonstrado na Figura 3.23.

Figura 3.23 | AVG na tabela Veículos

```
mysql> SELECT Marca, AVG(Valor) as "Valor Médio" FROM Veiculos GROUP BY Marca;
+--------------+--------------+
| Marca        | Valor Médio  |
+--------------+--------------+
| BMW          | 246250.000000 |
| Hyundai      | 120000.000000 |
| Jaguar       | 220000.000000 |
| Mercedes_Benz | 177000.000000 |
| Volkswagen   | 140000.000000 |
+--------------+--------------+
5 rows in set (0.00 sec)
```

Fonte: captura de tela do software MySQL, elaborada pelo autor.

Observe que, com essa função de agregação, foi possível agrupar o resultado das médias segundo as marcas registradas no banco de dados.

Reflita

Anteriormente, você pôde compreender como consultar o banco de dados em tabelas relacionadas. Seria possível utilizar as funções agregadas, como SUM, AVG, MIN, MAX e COUNT, juntamente com os SELECTS em múltiplas tabelas?

TOTAL (SUM)

Segundo Silberschatz (2010), a função SUM retorna o somatório dos valores em uma determinada coluna. Para isso, o SQL faz o somatório dos valores (obrigatoriamente numéricos). A sintaxe SQL utilizada para tal função é descrita como:

```
SELECT SUM(<coluna>) FROM <tabela>;
```

No exemplo utilizado nessa seção, vamos fazer o somatório do valor dos veículos registrados na tabela com o comando SQL a seguir:

```
SELECT SUM(Valor) as "Total" FROM Veiculos;
```

Gerando o retorno demonstrado na Figura 3.24.

Figura 3.24 | SUM na tabela Veículos

```
mysql> SELECT SUM(Valor) as "Total" FROM Veiculos;
+-------------+
| Total       |
+-------------+
| 2173000.00 |
+-------------+
1 row in set (0.00 sec)
```

Fonte: captura de tela do software MySQL, elaborada pelo autor.

Pesquise mais

As linguagens de programação mais utilizadas, como o Java, PHP e C#, utilizam as sintaxes SQL embutidas em suas estruturas. A linguagem PHP é um exemplo de uma ferramenta para desenvolvimento Web que utiliza amigavelmente a conexão com banco de dados.

O vídeo a seguir demonstra a simplicidade em se utilizar a linguagem de programação SQL com a linguagem PHP.

SUPERCODIGO. **3. Curso PHP + mYsql -** Mostrar datos (SELECT). 23 jul. 2015. Disponível em: <https://www.youtube.com/watch?v=ynP2RI15nGE>. Acesso em: 30 maio 2018.

Sem medo de errar

Trabalhando em um projeto de desenvolvimento de *tokens* de uma loja física de jogos de vídeo game e consoles, você é responsável pelas demandas relacionadas ao banco de dados. Após desenvolver as sintaxes para fazer algumas seleções de dados em mais de uma tabela, agora você deve desenvolver algumas funções operacionais/administrativas voltadas ao gerenciamento da loja. O gerente de projetos solicitou que você desenvolvesse algumas funções de agregação que retornassem: a quantidade de registros na tabela jogo; o valor do jogo de maior preço (valor); o valor do jogo de menor preço (valor); o valor médio dos jogos de guerra; e o valor total em estoque na loja.

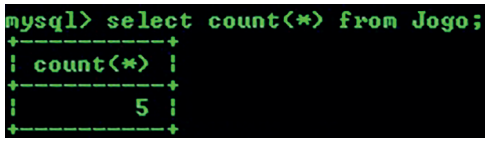
A fim de resolver esta questão, temos as seguintes atividades a realizar:

- Desenvolver uma função de agregação que retorne a quantidade de registros na tabela jogo. Você deve utilizar o comando SQL:

```
SELECT count(*) FROM Jogo;
```

Esse comando terá a saída representada na Figura 3.25.

Figura 3.25 | Primeiro comando SQL na segunda parte do projeto



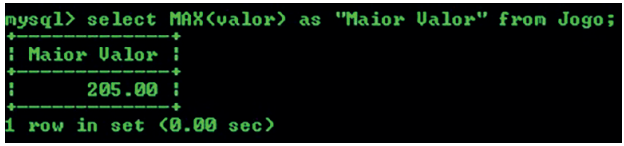
Fonte: captura de tela do software MySQL, elaborada pelo autor.

- Desenvolver uma função de agregação que retorne o valor do jogo de maior preço (valor). Você deverá utilizar o comando SQL:

```
SELECT MAX(valor) AS "Maior Valor" FROM Jogo;
```

Esse comando terá a saída representada na Figura 3.26.

Figura 3.26 | Segundo comando SQL na segunda parte do projeto



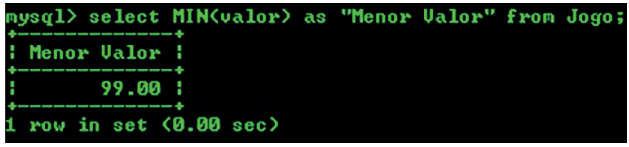
Fonte: captura de tela do software MySQL, elaborada pelo autor.

- Desenvolver uma função de agregação que retorne o valor do jogo de menor preço (valor). O comando a ser utilizado deverá conter o agregador `MIN`:

```
SELECT MIN(valor) AS "Menor Valor" FROM Jogo;
```

Esse comando terá a saída representada na Figura 3.27.

Figura 3.27 | Terceiro comando SQL na segunda parte do projeto



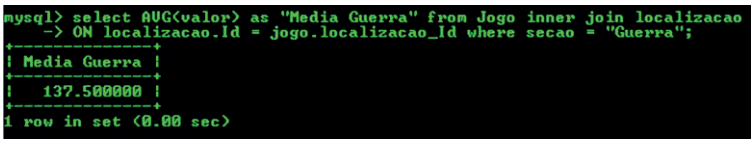
Fonte: captura de tela do software MySQL, elaborada pelo autor.

- Desenvolver uma função de agregação que retorne o valor médio dos jogos de guerra. Aqui você deverá utilizar o agregador AVG como no comando SQL:

```
SELECT AVG(valor) AS "Media Guerra" FROM Jogo
    INNER JOIN localização
    ON localizacao.Id = jogo.localizacao_Id
    WHERE secao = "Guerra";
```

Este comando terá a saída representada na Figura 3.28.

Figura 3.28 | Quarto comando SQL na segunda parte do projeto



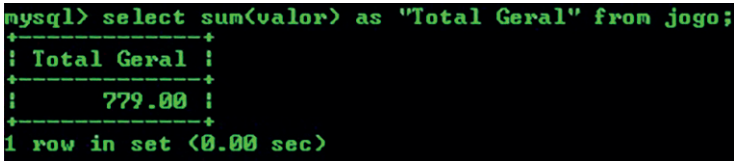
Fonte: captura de tela do software MySQL, elaborada pelo autor.

- Desenvolver uma função de agregação que retorne o valor total em estoque na loja. Aqui você deverá utilizar o agregador `SUM`:

```
SELECT sum(valor) AS "Total Geral" FROM jogo;
```

Com a saída representada na Figura 3.29.

Figura 3.29 | Quinto comando SQL na segunda parte do projeto



Fonte: captura de tela do software MySQL, elaborada pelo autor.

Com esses comandos implementados, você terá realizado sua demanda, um banco de dados bastante útil para as funções administrativas da loja.

Avançando na prática

Agenciador de Atletas

Descrição da situação-problema

Você trabalha em um site que faz análise de dados relacionados a esportes. Para esse site, foi desenvolvido um banco de dados com

a estrutura SQL e os registros em sua base de dados conforme descrito no Quadro 3.6:

Quadro 3.6 | Script Criação de BD e Tabelas

```
CREATE DATABASE Banco de Atletas;
USE Banco de Atletas;
CREATE table Atletas (
    RA int(3) primary key not null auto_increment,
    Nome varchar(20) not null,
    Clube varchar(20) not null,
    Qtd_Titulos int(3) not null
);

insert Atletas values (0, "Romario", "Vasco", 10),
    (0, "Gustavo Borges", "Pinheiros", 25),
    (0, "Maguila", "Aacademia de Boxe", 5),
    (0, "Ayrton Senna", "Willians", 3),
    (0, "Bob Burnquest", "semclube", 15),
    (0, "Ronaldo", "Real Madrid", 6),
    (0, "Ueda", "sem clube", 11),
    (0, "Gabriel Jesus", "Palmeiras", 4),
    (0, "Jaqueline", "SESI", 14),
    (0, "Serena Williams", "Sem Clube", 7);

```

Fonte: elaborado pelo autor.

O intuito desse site é permitir conhecer algumas curiosidades a respeito da carreira de alguns atletas. Para isso, foi solicitado que você desenvolva os filtros (*selects* com funções agregadas) que forneçam as seguintes informações: o atleta com mais títulos; o atleta com menos títulos; a média de títulos dos atletas cadastrados; a quantidade de atletas cadastrados; e o total de títulos de todos os atletas cadastrados.

Resolução da situação-problema

Você trabalha em um site que disponibiliza algumas curiosidades relacionadas aos esportes. Esse projeto prevê, inicialmente, cinco seleções de dados. Para realizar essa tarefa e tornar possível efetuar as consultas descritas, você deve gerar os scripts demonstrados a seguir:

1. O atleta com mais títulos

```
Select MAX(Qtd_Titulos) as "Mais Títulos" from
Atletas;
```

2. O atleta com menos títulos.

```
Select MIN(Qtd_Titulos) as "Menos Títulos" from
Atletas;
```

3. A média de títulos dos atletas cadastrados.

```
Select AVG(Qtd_Titulos) as "Média de Títulos"
from Atletas;
```

4. A quantidade de atletas cadastrados.

```
Select COUNT(*) as "Número de Registros" from
Atletas
```

5. O total de títulos de todos os atletas cadastrados

```
Select SUM(Qtd_Titulos) as "Total de Títulos"
from Atletas;
```

Faça valer a pena

1. Os bancos de dados, em sua grande maioria, possuem muitos registros inseridos em sua base. As funções de agregação são recursos que permite ao desenvolvedor efetuar algumas "contagens", com a utilização da linguagem de banco de dados SQL, independente da linguagem de programação escolhida para se desenvolver a aplicação.
Segundo Silberschatz (2010), as funções agregadas são recursos utilizados em conjunto com `SELECT`, que permite ao administrador de banco de dados efetuar consultas em um determinado conjunto de valores numéricos.
Observe o quadro abaixo, que contém a função agregada em banco de dados e a respectiva função em SQL:

Quadro | Agregadores, SQL e funções agregadoras

Agregador em SQL	Função agregadora
A- `SUM`	(I) Retorna a média em uma determinada coluna numérica.
B- `COUNT`	(II) Retorna a quantidade de registro em uma tabela.
C- `MAX`	(III) Retorna a somatória de uma coluna numérica.
D- `AVG`	(IV) Retorna o menor valor numérico em uma coluna.
E- `MIN`	(V) Retorna o maior valor numérico em uma coluna.

Fonte: elaborada pelo autor.

Assinale a alternativa que correlaciona as colunas corretamente.

a) A-II, B-V, C-I, D-III, E-IV.
b) A-V, B-IV, C-III, D-II, E-I.
c) A-III, B-II, C-V, D-I, E-IV.
d) A-I, B-III, C-V, D-IV, E-II.
e) A-IV, B-III, C-I, D-II, E-V.

2. Observe a estrutura da tabela a seguir:

```
create table y(
   a int(3),
   b int(3),
   c int(3),
   d varchar(3)
);
```

Em que foram inseridos os seguintes valores:

```
insert y values (1, 2, 3, "p"),
   (1, 2, 3, "p"),
   (2, 3, 4, "q"),
   (3, 4, 5, "r"),
   (4, 5, 6, "s"),
   (5, 6, 7, "t"),
   (6, 7, 8, "u"),
   (7, 8, 9, "v"),
   (8, 9, 10, "x"),
   (9, 10, 11, "z");
```

Dada a tabela desenvolvida no banco de dados e suas respectivas inserções, analise as sintaxes SQL:
I. select MAX(b) from y where b < 8;
II. select SUM(c) from y where a > 5;
III. select COUNT(d) from y;

Assinale a alternativa que descreve corretamente os respectivos valores de saída:

a) 8 – 45 – 9.5.
b) 9 – 60 – 0.
c) 9 – 55 – 0.
d) 7 – 55 – 10.
e) 7 – 38 – 10.

3. Segundo Mizrahi (2008), as sintaxes são um conjunto composto por regras acerca de uma linguagem, em que são estabelecidas a suas palavras e o seu respectivo significado ou função a ser desempenhada. Nas linguagens de programação em sistemas computacionais, as sintaxes podem ser divididas em:

- Léxicas: moldam as características da linguagem de programação quanto a palavras reservadas, identificadores, operadores e demais objetos da linguagem.
- Sintáticas: nesse quesito estão as instruções válidas para determinadas funções, como os comandos, expressões, rotinas e etc.

Desse modo, podemos compreender como demonstrar um recurso por meio de uma estrutura de sintaxe pode ser uma poderosa ferramenta para proporcionar o entendimento acerca de uma linguagem computacional.

Observe as sintaxes das funções agregadas a seguir e assinale (V) para verdadeiro e (F) para falso:

```
(  )select sum([coluna]) from [tabela];
(  )select AVG([coluna]) from [tabela];
(  )select MAX() from [coluna];
(  )select COUNT(*) from [tabela];
(  )select MIN([tabela]) from [coluna];
```

Assinale a alternativa com a sequência correta.

a) V – F – V – F – V.
b) V – V – F – V – F.
c) F – V – F –V – F.
d) F – F – V – F – V.
e) F – F – F – V – V.

Seção 3.3

Subconsultas em banco de dados

Diálogo aberto

Caro aluno, certamente você deve conhecer os sites de hospedagem em resorts, hotéis e pousadas. Ao efetuar uma pesquisa por um destino, esses sites disponibilizam diferentes filtros, como os de níveis de estrelas, valor da diária, estacionamento, piscina, refeições, entre outros. Ou seja, em uma consulta em um banco de dados, é possível fazer subconsultas selecionando filtros. Quantos sites de diferentes serviços ou vendas você conhece que também apresentam essas possibilidades? Consultas e subconsultas em banco de dados são realmente muito uteis.

Você trabalha na empresa que está prestando serviço para a loja de games especializada em jogos para os consoles e computadores de última geração. Devido a seu sucesso no mercado, ela agora está inovando tanto sua parte gerencial como o atendimento aos clientes. Para esse último, foram instalados *tokens* de autoatendimento na loja para a consulta de localização dos jogos e os seus respectivos valores. Com o sucesso de vendas na loja, devido à tecnologia disponibilizada, novos títulos foram colocados nas prateleiras, entre eles: jogo de corrida com carros esportivos europeus, chamado "Super Drive", por R$ 250,00; jogo online de guerra futurista, chamado "Neo", vendido a R$100,00; jogo de aventura com um herói desajeitado, chamado "Max Joe", por R$ 120,00; jogo de RPG de mundo aberto, jogado do modo online e colaborativo, chamado "N. New", vendido a R$199,00. Com isso, os jogos mais antigos como os "BF 1" e "COD 3" tiveram um reajuste de preço de 50% de desconto.

Tudo isso afeta o banco de dados utilizado nos *tokens*. Como você é o responsável por ele, seu gerente de projetos solicitou que você desenvolvesse as seguintes tarefas:

- Inserir os novos títulos no banco de dados, para que os clientes possam consultá-los.
- Alterar o valor dos jogos em promoção.

- Criar uma tabela chamada promoção, com um número identificador da promoção e o código do jogo (Chave estrangeira da tabela jogo).
- Inserir os jogos em promoção na tabela criada.
- Uma forma de selecionar o nome do jogo, o valor e o nome da seção dos títulos em promoção.
- Uma forma de selecionar o nome dos títulos e seus respectivos valores, que não estejam em promoção, fazendo com que retorne somente os mais novos disponíveis na loja.

Para realizar tais tarefas, será necessário que você planeje a implementação da tabela "promoção", e a forma como ela vai estar relacionada com a tabela "Jogo". Já para desenvolver os filtros de consultas, devem ser utilizadas as técnicas de subconsultas por meio da linguagem de programação de banco de dados SQL.

Para realizar essa tarefa com sucesso e os clientes poderem aproveitar o sucesso da promoção de venda dos jogos, você irá aprender nesta seção como desenvolver subconsultas em SQL. Bons estudos!

Não pode faltar

Utilizar bancos de dados para guardar os dados é uma eficiente forma de gerenciar e manter organizados os objetos acerca de um domínio. Porém, somente mantê-los arquivados não possibilitará gerar informações. Para isso, quanto aos recursos disponíveis em banco de dados, as consultas realizadas por meio das *queries*, que são comandos do SQL utilizados para fazer consultas, incluir ou alterar dados nas tabelas de um banco de dados. Anteriormente, você pôde compreender como as consultas a duas ou mais tabelas são efetuadas, porém, a linguagem de programação de banco de dados SQL permite desenvolver algumas consultas aninhadas.

Para exemplificar os conceitos e aplicações envolvidos nessa seção de estudos, foi desenvolvido um banco de dados para guardar as informações de uma biblioteca, conforme demonstradas no esquema representado a seguir:

- Aluno (RA, nome, telefone)

- Funcionário (matrícula, nome, cargo)
- Livro (isbn, nome, seção)
- Empréstimo (número, retirada, devolução, aluno_RA, funcionario_matricula, livro_isbn)
- Restrição (Id, aluno_RA, livro_isbn)

Assimile

Você deve sempre lembrar que, a partir da Unidade 3, os atributos com sublinhado simples, são as chaves primária das tabelas. Já as colunas com duplo sublinhado, são as chaves estrangeiras da tabela.

Para acompanharmos o desenvolvimento desta seção, considere que os registros foram inseridos nas tabelas como demonstrado na Figura 3.30.

Figura 3.30 | *Select* na tabelas da Biblioteca

```
mysql> select * from aluno;
+--------+-----------------+------------+
| RA     | nome            | telefone   |
+--------+-----------------+------------+
| 11223  | Serj Tankian    | 987658899  |
| 12345  | Joye Ramone     | 991213344  |
| 54321  | Lars Ulrich     | 977889966  |
| 56789  | Corey Taylor    | 901238525  |
| 98765  | Vicky Psarakis  | 922556688  |
+--------+-----------------+------------+
5 rows in set (0.08 sec)

mysql> select * from funcionario;
+-----------+---------------------+--------------------+
| matricula | nome                | cargo              |
+-----------+---------------------+--------------------+
|         1 | Melvil Dewey        | Bibliotecario 1    |
|         2 | Manuel Bastos Tigre | Bibliotecario 2    |
+-----------+---------------------+--------------------+
2 rows in set (0.07 sec)

mysql> select * from livro;
+-------+-------------------+-------------+
| isbn  | nome              | secao       |
+-------+-------------------+-------------+
| 11111 | Vida Punk         | musica      |
| 22222 | Mestres da Bateria | musica      |
| 33333 | Sexta-feira 13    | terror      |
| 44444 | Mulheres do Rock  | musica      |
| 55555 | O exorcista       | terror      |
| 66666 | O chamado         | terror      |
| 77777 | Mascaras          | musica      |
| 88888 | ToxiCity          | musica      |
| 99999 | Diario de um Mago | esoterismo  |
+-------+-------------------+-------------+
9 rows in set (0.00 sec)

mysql> select * from emprestimo;
+---------+------------+------------+-----------+---------------------+-------------+
| numero  | retirada   | devolucao  | aluno_RA  | funcionario_matricula | livro_isbn |
+---------+------------+------------+-----------+---------------------+-------------+
|       8 | 2018-01-02 | 2018-01-17 |     12345 |                   1 |       11111 |
|       9 | 2018-01-15 | 2018-02-01 |     11223 |                   2 |       88888 |
|      10 | 2018-04-05 | 2018-04-20 |     56789 |                   2 |       77777 |
|      11 | 2018-03-15 | 2018-03-30 |     98765 |                   1 |       44444 |
|      12 | 2018-06-06 | 2018-06-21 |     56789 |                   1 |       55555 |
|      13 | 2018-08-01 | 2018-08-16 |     12345 |                   2 |       22222 |
|      14 | 2018-10-10 | 2018-10-25 |     11223 |                   1 |       66666 |
+---------+------------+------------+-----------+---------------------+-------------+
7 rows in set (0.03 sec)

mysql> select * from restricao;
+----+-----------+------------+
| Id | aluno_RA  | livro_isbn |
+----+-----------+------------+
|  1 |     12345 |      22222 |
+----+-----------+------------+
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Segundo Silberschatz (2010), é considerada uma subconsulta uma expressão em SQL, composta por `SELECT-FROM-WHERE`, aninhada dentro de outra consulta, permitindo fazer comparações entre os conjuntos de dados.

A sintaxe presente no SQL permite que sejam feitas consultas usando diversas relações entre inúmeras tabelas presentes nos bancos de dados, bastando a utilização dos conectivos. Para isso, temos o conectivo `IN`, que efetua o teste no conjunto de dados, que é fruto de uma coleção de valores produzidos por meio de um `SELECT`, e o conectivo `NOT IN`, que permite efetuar a ausência em um conjunto de valores.

A sintaxe utilizada com o conectivo `IN` pode ser definida como:

```
SELECT [campo]

FROM [tabela]

WHERE [campo] IN (SELECT [campo] FROM [tabela]);
```

Dica

Você pode notar que a linguagem de programação de banco de dados SQL permite várias maneiras de se escrever uma consulta, deixando a linguagem mais próxima ao natural e flexível.

Para compreender essa técnica, vamos tomar como exemplo uma consulta em que desejamos encontrar o nome de todos os alunos que efetuaram empréstimos, mas que estão com restrição para locar novos livros. Assim, teremos:

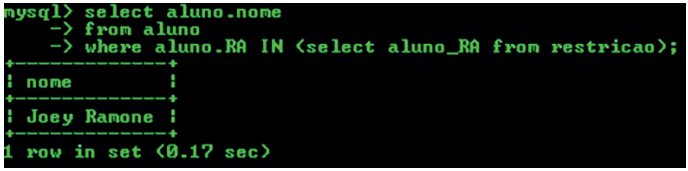
```
SELECT aluno.nome

FROM aluno

WHERE aluno.RA IN (SELECT aluno_RA FROM restrição);
```

Esta sintaxe diz que deve SELECIONAR o nome do aluno, NA tabela aluno, QUANDO o RA do aluno estiver na seleção do RA do aluno, na tabela restrição. A saída gerada pela subconsultas é demonstrada na Figura 3.31.

Figura 3.31 | Exemplo de subconsulta 1



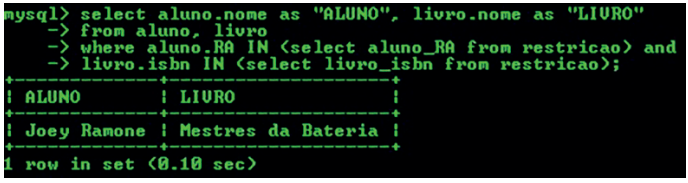
Fonte: captura de tela do MySQL, elaborada pelo autor.

É possível, também, consultar o nome do livro que o aluno "Joey Ramone" não entregou, que o deixa com restrição para locar um novo livro. Para isso, utilize a sintaxe:

```
SELECT aluno.nome as "ALUNO", livro.nome as "LIVRO"
FROM aluno, livro
WHERE aluno.RA IN(SELECT aluno_RA FROM restrição)
AND Livro.isbn IN (selectlivro_isbnfrom restrição);
```

Essa consulta irá gerar saída demonstrada na Figura 3.32.

Figura 3.32 | Exemplo de Subconsulta 2



Fonte: captura de tela do MySQL, elaborada pelo autor.

Nesses exemplos, é possível compreender como a subconsulta pode ser encadeada à consulta, ou seja, um `SELECT` dentro de outro `SELECT` pode testar a relação de um atributo no relacionamento entre as tabelas no banco de dados.

O conectivo `NOT IN` é utilizado de maneira semelhante, conforme pode ser observado na sintaxe a seguir:

```
SELECT [campo]
FROM [tabela]
WHERE [campo] NOT IN (SELECT [campo] FROM [tabela]);
```

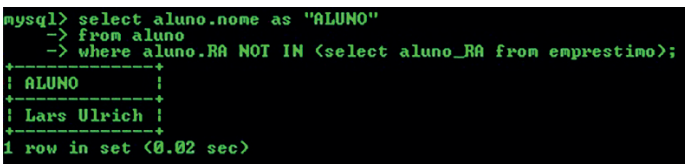
Segundo Silberschatz (2010), embora os conectivos possuam sintaxes idênticas, a sua aplicação é completamente diferente, pois

o `NOT IN` permite que a seleção seja negada. Para visualizar essa diferença, vamos selecionar o nome dos alunos que nunca tomaram um livro emprestado. Para isso, foi desenvolvido o seguinte comando:

```
SELECT aluno.nome as "ALUNO"
FROM aluno
WHERE aluno.RA NOT IN (SELECT aluno_RA FROM emprestimo);
```

Essa sintaxe irá SELECIONAR o nome do aluno, NA tabela aluno, QUANDO o RA do aluno NÃO estiver nela (seleção do RA do aluno na tabela empréstimo). A saída gerada pela subconsulta é demonstrada na Figura 3.33.

Figura 3.33 | Exemplo de Subconsulta 3



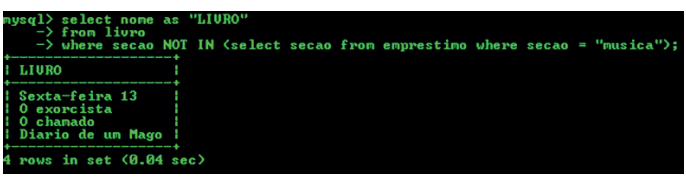
Fonte: captura de tela do MySQL, elaborada pelo autor.

Não necessariamente as subconsultas necessitam ser feitas em duas tabelas relacionadas, sendo possível utilizar esse recurso em apenas uma tabela. Por exemplo, uma seleção do nome dos livros em que sejam ignorados os livros da seção "música":

```
SELECT nome as "LIVRO"
FROM livro
WHERE seção NOT IN (SELECT seção FROM emprestimo where seção = "música");
```

Com isso, será possível observar a saída descrita na Figura 3.34.

Figura 3.34 | Exemplo de Subconsulta 4



Fonte: captura de tela do MySQL, elaborada pelo autor.

Exemplificando

Ao permitir comparar conjuntos de seleções de dados, o SQL proporciona aos desenvolvedores incrementar significativamente as possibilidades de busca de informações. Imagine que você possui um banco de dados com o nome de todas as seleções que já participaram na copa do mundo, mas você deseja exibir apenas se o número de campeãs europeias for maior do que as seleções campeãs da África. Certamente, os nomes seriam exibidos, pois o número de seleções da Europa é maior do que o do continente africano.

Comparação de Conjuntos

Segundo Date (2004), a sintaxe SQL permite o desenvolvimento de subconsultas aninhadas, em que é possível fazer a comparação entre conjuntos de dados, utilizando condições (`WHERE`). Porém, para que sejam efetuadas as comparações, deve ser inserida a palavra "some" na sintaxe, nos operadores comparativos representados no Quadro 3.6.

Quadro 3.6 | Operadores de Comparação em Subconsultas SQL.

Operador Matemático	SELECT com WHERE SQL	Subconsulta SQL
=	`WHERE` campo = condição	`WHERE` campo = some (`SELECT`)
≠	`WHERE` campo <> condição	`WHERE` campo <> some (`SELECT`)
>	`WHERE` campo > condição	`WHERE` campo > some (`SELECT`)
≥	`WHERE` campo >= condição	`WHERE` campo >= some (`SELECT`)
<	`WHERE` campo < condição	`WHERE` campo < some (`SELECT`)
≤	`WHERE` campo <= condição	`WHERE` campo <= some (`SELECT`)

Fonte: adaptado de Silberschatz (2010, p. 64).

Assimile

O conceito de comparação entre conjuntos de dados está diretamente relacionado às sistematizações utilizadas na lógica matemática, em que se espera resultados como verdadeiro ou falso.

Vamos realizar uma consulta dos nomes dos livros e da seção, se a quantidade de livros da seção "esoterismo" for menor do que a quantidade de livros. Para isso, utilize a sintaxe SQL a seguir:

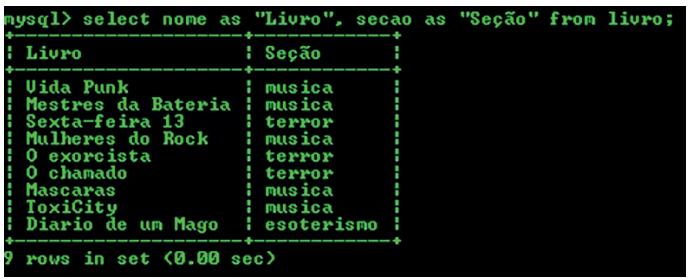
```
select nome as "Livro", secao as "Seção"

from livro

where nome> some (select nome from livro where secao = "esoterismo");
```

A consulta "`select nome as "Livro", secao as "Seção" FROM livro`" irá gerar a saída com nove registros, conforme demonstrado na Figura 3.35.

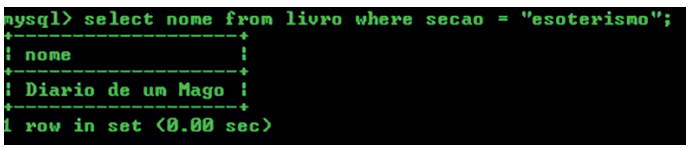
Figura 3.35 | Consulta na comparação de conjuntos



Fonte: captura de tela do MySQL, elaborada pelo autor.

A subconsulta (`select nome from livro wheresecao = "esoterismo"`) gera apenas uma saída, como mostra a Figura 3.36.

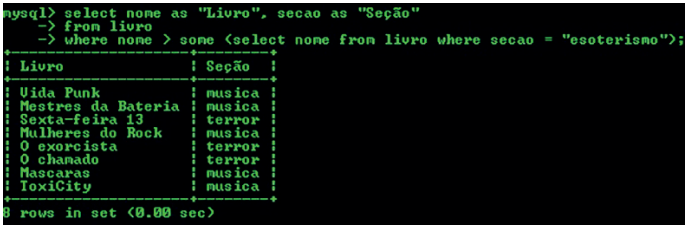
Figura 3.36 | Subconsulta na comparação de conjuntos 1



Fonte: captura de tela do MySQL, elaborada pelo autor.

Com isso, o trecho "`where nome> some`" da sintaxe SQL faz a comparação nome de livros 9 é maior que 1 registro de livro da seção esoterismo. Como essa afirmativa é verdadeira, pois 9 é maior do que 1, então será gerada a saída representada na Figura 3.37.

Figura 3.37 | Subconsulta Verdadeira



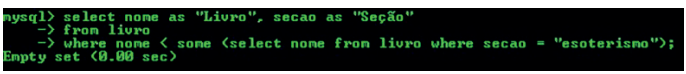
Fonte: captura de tela do MySQL, elaborada pelo autor.

Reflita

Se utilizarmos um conectivo NOT IN a seleção vai ser negada. Quando utilizado a expressão "`WHERE campo <> some (select...)`", estaremos negando uma seleção. Embora as sintaxes para as duas técnicas sejam diferentes, os resultados gerados pela seleção dos dados serão iguais?

Se for invertida a condição, trocando o sinal de maior (>) por menor, nenhum livro será exibido, conforme representado na Figura 3.38.

Figura 3.38 | Subconsulta falsa



Fonte: captura de tela do MySQL, elaborada pelo autor.

Isso ocorre porque a condição "`WHERE nome > some`" é falsa. Analisando matematicamente, temos que 9 não é maior do que 1.

Pesquise mais

A forma como as técnicas são utilizadas dependem da arquitetura do sistema de gerenciamento de banco de dados, como no exemplo, nos do tipo relacional e não relacional. O vídeo a seguir, produzido pela DevMedia, é uma entrevista que visa elucidar as diferenças entre as duas formas de banco de dados.

DEVMEDIA. **Qual a diferença entre Bancos de dados relacionais e não relacionais.** 3 maio 2018. Disponível em: <https://www.youtube.com/watch?v=76rdRd83Ox0>. Acesso em: 10 jun. 2018.

Sem medo de errar

Com o sucesso da implementação de *tokens* na loja de *games*, foi necessária a inserção no banco de dados de novos títulos colocados à venda (um jogo de corrida com carros esportivos europeus, chamado "Super Drive", por R$ 250,00; um jogo de guerra futurista no modo online, chamado "Neo", vendido a R$100,00; um jogo de aventura com um herói desajeitado, chamado "Max Joe", por R$ 120,00; e um jogo de RPG de mundo aberto, jogado do modo online e colaborativo, chamado "N. New", vendido a R$199,00; além do reajuste de preços de jogos mais antigos, como o "BF 1" e "COD 3", que tiveram 50% de desconto). Com isso, serão necessárias atualizações no banco de dados dos *tokens*: inserção dos novos títulos no banco de dados, para que os clientes possam consultá-los; alteração do valor dos jogos em promoção; desenvolvimento de uma tabela chamada promoção, com um número identificador da promoção e o código do jogo (chave estrangeira da tabela jogo); inserção dos jogos em promoção na tabela criada; uma forma de selecionar o nome do jogo, o valor e o nome da seção dos títulos em promoção e uma forma de selecionar o nome dos títulos que não estejam em promoção e seu respectivo valor.

Para atender a essas demandas, você deve desenvolver comandos semelhantes às sintaxes abaixo.

1) Para inserir os novos títulos:

```
INSERT jogo VALUES (0, "Super Driver", 205.00, 2),
   (0, "Neo", 100.00, 2),
   (0, "Max Joe", 120.00, 3),
   (0, "N. New", 199.00, 4);
```

2) Para alterar o valor dos jogos em promoção:

```
   UPDATE jogo SET valor = valor * 0.5 where nome =
"BF 1";

   UPDATE jogo SET valor = valor * 0.5 where nome =
"COD 3";
```

3) Para desenvolver uma tabela para inserir os jogos em promoção:

```
create table promocao(
Promo int(3) primary key auto_increment,
Cod_Jogoint(3) not null,
foreign key (Cod_Jogo) references Jogo(Cod)
);
```

4) Para inserir os jogos na tabela promoção:

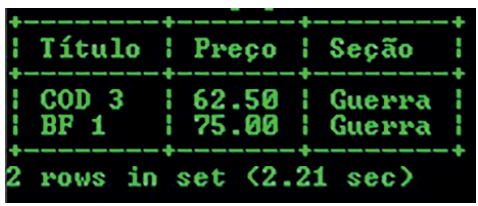
```
INSERT promocao VALUES (0, 1),
(0, 2);
```

5) Para o primeiro filtro, a sintaxe SQL desenvolvida foi:

```
selectjogo.nome as "Título", jogo.valor as "Preço", localizacao.secao as "Seção"
from jogo inner join localizacao ON localizacao.Id = jogo.localizacao_Id
where jogo.COD IN (select Cod_Jogo from promocao);
```

Essa última sintaxe irá gerar uma saída como demonstrado na Figura 3.39.

Figura 3.39 | Primeiro filtro



Fonte: captura de tela do MySQL, elaborada pelo autor.

6) Para o segundo filtro a sintaxe SQL desenvolvida foi:

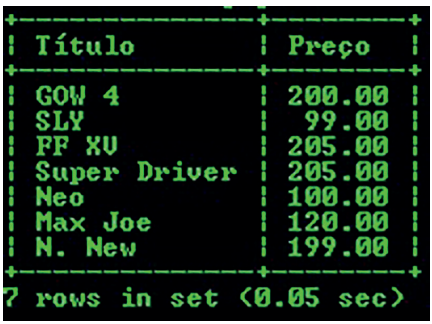
```
selectjogo.nome as "Título", jogo.valor as "Preço"

from jogo

where jogo.COD NOT IN (select Cod_Jogo from promocao);
```

Que irá gerar a saída demonstrada na Figura 3.40:

Figura 3.40 | Segundo filtro



Fonte: captura de tela do MySQL, elaborada pelo autor.

Com o desenvolvimento desses comandos, você atenderá à demanda do seu gestor, otimizando a utilização do *tokens*.

Avançando na prática

Seleções campeãs do mundo

Descrição da situação-problema

Uma grande marca mundial está preparando uma promoção para ser utilizada durante o campeonato mundial de futebol. Para isso, ela contratou a empresa em que você trabalha para atuar no desenvolvimento do banco de dados necessário. Uma das tabelas do banco de dados deve conter o histórico dos campeões dos últimos 10 eventos, que ocorre de 4 em 4 anos. O Quadro 3.8 mostra estes dados.

Quadro 3.8 | Copa do Mundo

Ano	Campeã	Continente do Campeão	País Sede	Continente Sede
1978	Argentina	América	Argentina	América
1982	Itália	Europa	Espanha	Europa
1986	Argentina	América	México	América
1990	Alemanha	Europa	Itália	Europa
1994	Brasil	América	Estados Unidos	América
1998	França	Europa	França	Europa
2002	Brasil	América	Japão	Ásia
2006	Itália	Europa	Alemanha	Europa
2010	Espanha	Europa	África do Sul	África
2014	Alemanha	Europa	Brasil	América

Fonte: adaptado de <https://goo.gl/suTFrV>.

O cliente deseja que, para este Quadro do banco de dados, sejam criados os seguintes filtros:

- Que retorne o nome das seleções campeãs, o ano da respectiva copa e o país sede, somente dos países do continente europeu;
- Que retorne o nome das seleções campeãs, o ano da respectiva copa e o país sede, somente dos países do continente americano.

Será sua responsabilidade desenvolver os comandos para atender a essa demanda.

Resolução da situação-problema

Para realizar essa tarefa, deverá ser desenvolvido o banco conforme o esquema relacional a seguir:

Copa (ano, campeã, contCampea, paisSede, contSede);

Em seguida, os registros contidos no Quadro 3.8 devem ser inseridos nesse banco de dados.

Para desenvolver o filtro que selecionará o nome das seleções campeãs, o ano da respectiva copa e o país sede somente dos países do continente americano, deverá ser desenvolvido o script:

```
selectcampea as "Seleção Campeã", ano as "Ano",
sede as "País Sede"

from CampeasSede

where campea IN (select campea from CampeasSede
where contCampea = "America");
```

Que deverá retornar os registros descritos na Figura 3.41:

Figura 3.41 | *Selects* seleções da Europa

Seleção Campeã	Ano	País Sede
Argentina	1978	Argentina
Argentina	1986	Mexico
Brasil	1994	Estados Unidos
Brasil	2002	Japao

Fonte: captura de tela do MySQL, elaborada pelo autor.

Já para desenvolver o filtro que permita selecionar o nome das seleções campeãs, o ano da respectiva copa e o país sede somente dos países do continente europeu, será desenvolvido o script:

```
selectcampea as "Seleção Campeã", ano as "Ano",
sede as "País Sede"

from CampeasSede

where campea IN (select campea from CampeasSede
where contCampea = "Europa");
```

Que deverá retornar os registros descritos na Figura 3.42:

Figura 3.42 | *Selects* Seleções das Américas

Seleção Campeã	Ano	País Sede
Italia	1982	Espanha
Alemanha	1990	Italia
Franca	1998	Franca
Italia	2006	Alemanha
Espanha	2010	Africa do Sul
Alemanha	2014	Brasil

Fonte: captura de tela do MySQL, elaborada pelo autor.

Faça valer a pena

1. Quando os bancos de dados do tipo relacional são projetados, o DBA deve se preocupar em efetuar a conexão entre as tabelas. Para isso, a linguagem de banco de dados SQL possui mecanismos de identificação das chaves primária e estrangeira, conforme a representação sintática a seguir:

```
Create table NomeDaTabela(
   Campo1 tipo(tamanho)primary key,
   CampoChaveEstrangeira tipo(tamanho)not null,
   Foreign key(CampoChaveEstrangeira) references Tabela
ChaveEstrangeira (Campo ChaveEstrangeira));
```

Com isso, é possível desenvolver subconsultas por meio do SQL para extrair um maior número de informações, pois os dados são acessados em mais de uma tabela.

Segundo Silberschatz (2010), é considerada uma subconsulta uma expressão SQL composta por **SELECT-FROM-WHERE**, que é aninhada dentro de outra consulta, permitindo fazer comparações entre os conjuntos de dados.

Assinale a alternativa que representa a sintaxe SQL correta, utilizando o conectivo `IN`.

a) SELECT <campo>
FROM <tabela>
WHERE IN <campo> (SELECT <campo> FROM <tabela>);.
b) SELECT <campo>
FROM <tabela> IN
WHERE <campo> (SELECT <campo> FROM <tabela>);.
c) SELECT <campo>
FROM <tabela>
WHERE <campo> IN (SELECT <campo> FROM <tabela>);.
d) SELECT <campo>
FROM IN<tabela>
WHERE <campo> (SELECT <campo> FROM <tabela>);.
e) SELECT <campo>
FROM <tabela>IN
WHERE <campo> IN (SELECT <campo> FROM <tabela>);.

2. Um pesquisador desenvolveu um banco de dados, porém, ao acessar o arquivo para executá-lo no SGBD, o script de criação das tabelas foi perdido, restando somente os dados que seriam inseridos:

```
insert abc values
```

```
(10, “Z”, “sim”),
(20, "W", "sim"),
(30, "X", "nao"),
(40, "Y", "sim"),
(50, "H", "nao"),
(60, "P", "nao"),
(70, "Q", "nao"),
(80, "M", "sim"),
(90, "N", "sim"),
(99, "S", "nao"),
(22, "T", "nao");
```

Junto com o script SQL, foram encontrados os comandos a seguir:

/*Primeiro Script*/

```
SELECT a, b, c
FROM abc
WHERE c NOT IN (SELECT c FROM abc where c <> "SIM");
```

/*Segundo Script*/

```
SELECT a, b, c
FROM abc
WHERE c NOT IN (SELECT c FROM abc where c <> "Nao");
```

/*Terceiro Script*/

```
SELECT a, b, c
FROM abc
WHERE c NOT IN (SELECT c FROM abc where c <> "SIM" OR
"NAO");
```

Com base nessas informações, assinale a alternativa que representa as saídas corretamente.

a) Primeiro Script: (30, X, Não), (50, H, Não); Segundo Script: (30, X, Não), (50, H, Não); Terceiro Script: (10, Z, Sim), (20, W, Sim).
b) Primeiro Script: (10, Z, Sim), (20, W, Sim); Segundo Script: (30, X, Não), (50, H, Não); Terceiro Script: (10, Z, Sim), (20, W, Sim).
c) Primeiro Script: (10, Z, Sim), (20, W, Sim); Segundo Script: (30, X, Não), (50, H, Não); Terceiro Script: (30, X, Não), (50, H, Não).
d) Primeiro Script: (10, Z, Sim), (20, W, Sim); Segundo Script: (30, X, Não), (50, H, Não); Terceiro Script: Empty set, 4 warnings.
e) Primeiro Script: (10, Z, Sim), (20, W, Sim); Segundo Script: Empty set, 2 warnings; Terceiro Script: Empty set, 2 warnings.

3. Na matemática, quando temos, por exemplo, três conjuntos A, B e C quaisquer, temos algumas propriedades básicas, como:
(A ⊂ B e B ⊂ A), portanto A = B(A ⊂ B e B ⊂ A), portanto A = B
(A ⊂ B e B ⊂ C), portanto A ⊂C(A ⊂ B e B ⊂ C), portanto A ⊂ C
Se A possui n elementos, então o conjunto das partes representado por P(A), possui A^n subconjuntos.
Com isso, é possível estabelecer algumas operações de conjuntos, como:

- União de conjuntos;
- Intersecção de conjuntos;
- Diferença de conjuntos;
- Complementar de conjuntos.

Porém, as operações com conjuntos não são uma exclusividade do campo da matemática, estando presentes nas ciências biológicas, engenharias, sociologia, computação, entre outras áreas do conhecimento.

Observe a afirmativa a seguir:
A técnica de _____ de conjuntos em bancos de dados possibilita que um grupo de dados seja criado, permitindo, posteriormente, trabalhar a relação sobre eles. Com isso, o desenvolvedor pode efetuar subconsultas aninhadas e comparar os conjuntos de dados, utilizando-se condições (_____). Para que isso ocorra à sintaxe disponível no SQL, necessita que a _____ em conjunto com os operadores matemáticos de comparação.

Assinale a alternativa que complete as lacunas corretamente.

a) intersecção – `WHERE` – `SOME`.
b) intersecção– `FROM` – `WHERE`.

c) comparação – `FROM` – `WHERE`.
d) comparação – `WHERE` – `SOME`.
e) comparação – `SOME` – `ELSE`.

Referências

DATE, C. J. **Introdução a Sistemas de Banco de Dados.** 8. ed. São Paulo: Elsevier, 2012. 921 p.

MILANI, A. **MySQL**: Guia do Programador. São Paulo: Editora Novatec. 2007. 400 p.

MIZRAHI, V. V. **Treinamento em Linguagem C**. 2. ed. São Paulo: Pearson Prentice Hall, 2008. 405 p.

SILBERSCHATZ, A.; KORTH, H.; SUNDARSHAN, S. **Sistema de Banco de Dados**. Rio de Janeiro: Editora Elsiever, 2010. 904 p.

Recursos avançados e automação de processos

Convite ao estudo

Caro aluno, você já deve ter efetuado algum tipo de compra nos mais diversos sites disponíveis na internet. No momento em que estamos buscando o produto, os mecanismos que visam tornar as buscas mais rápidas fazem com que o tempo de resposta seja mais eficiente. Posteriormente quando os produtos são inseridos no carrinho de compras, alguns cálculos são feitos automaticamente, como um desconto ao escolher determinada forma de pagamento, a taxa de entrega de acordo com a localização, um desconto ao inserir um *voucher*, entre outros.

As técnicas que serão discutidas nesta unidade são recursos que possibilitam a você automatizar alguns recursos dentro do banco de dados, agregando qualidade às aplicações devido à diminuição no tempo de respostas nas consultas. Assim, com esses conhecimentos, você será capaz de elaborar *scripts* SQL para automação de tarefas em tabelas.

Inicialmente, serão estudados os índices e as visões, permitindo que você possa pré-configurar algumas rotinas nos bancos de dados. Após isso, o objeto de estudo será o controle transacional, em que será possível automatizar algumas execuções, abrindo algumas possibilidades de deixar o banco mais funcional. Finalmente, a discussão será entorno dos procedimentos e funções possíveis de serem realizadas com a linguagem de programação de banco de dados SQL. Assim, será possível desenvolver algumas rotinas, que podem possibilitar que algumas funções possam ser executadas automaticamente.

Ao longo das seções de aprendizagem, será possível conhecer e compreender a criação e manipulação de tabelas para funções avançadas, o que permitirá buscar soluções para as aplicações por meio das sintaxes e recursos disponíveis na linguagem de programação de banco de dados, o SQL.

A fim fixar as técnicas discutidas ao longo dessa unidade, vamos utilizar um cenário em que você trabalha na prefeitura de uma cidade litorânea como responsável pelo setor de TI. Você foi designado a desenvolver um banco de dados para efetuar o controle e gerenciamento das escunas que levam os turistas aos passeios nas ilhas próximas ao continente. A sua responsabilidade frente a esse projeto é muito grande. Inicialmente, deverá ser desenvolvida uma visão no site, a fim de se diminuir o tempo de consulta e a carga de processamento. Em seguida, para garantir a integridade dos dados, devem ser criados pontos de restauração para serem usados durante o desenvolvimento de algumas tarefas. Para finalizar o projeto, você deve desenvolver um procedimento que conceda desconto aos turistas no momento de pagar os passeios de escuna.

Para resolver esse problema, estude as técnicas abordadas nesta unidade. Tais aplicações discutidas ao longo desses estudos vão permitir que você conheça e compreenda a automação de processos em banco de dados, bem como sua importância frente às aplicações desenvolvidas.

Seção 4.1

Visões e índices

Diálogo aberto

Quando efetuamos uma pesquisa nos buscadores de internet, o volume de dados pesquisado é muito grande. Imagine que você esteja buscando informações a respeito de certificações em banco de dados. Para agilizar tais buscas, os desenvolvedores web utilizam recursos que servem como referência, por meio de palavras-chave.

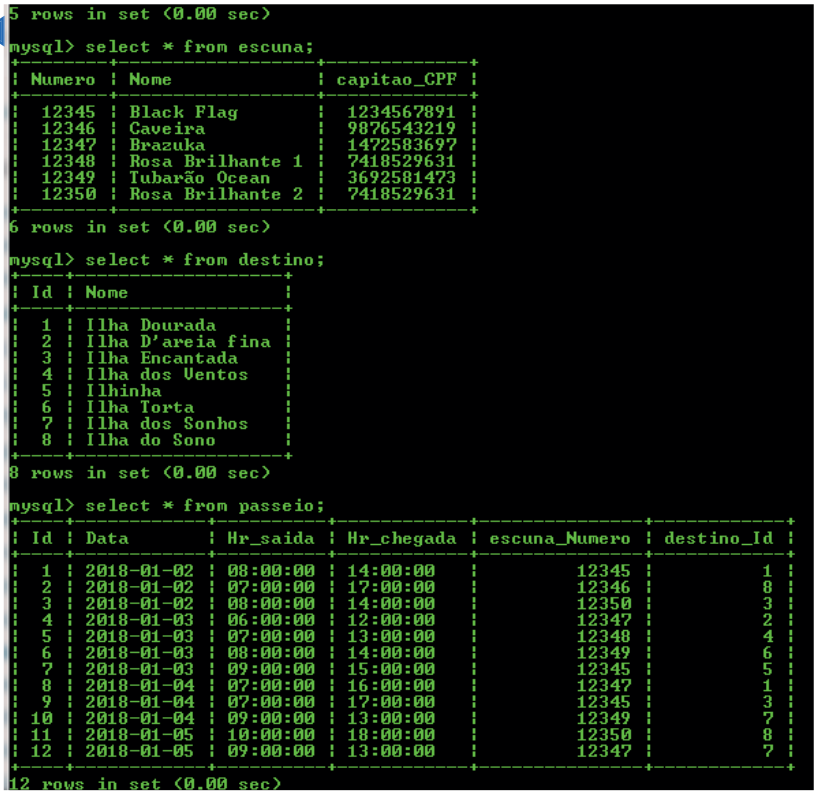
Assim como os buscadores, o banco de dados possui recursos que permitem o desenvolvimento de índices, buscas textuais e visões, que possibilitam agregar qualidade às buscas de informações nas diversas tabelas.

O município para o qual você trabalha está situado no litoral, e as suas principais atividades econômicas estão relacionadas ao turismo. Isso porque existem diversas ilhas próximas à costa, o que atrai muitas pessoas. O movimento no entorno do *pier* é sempre grande, pois lá estão os hotéis, bares, restaurantes e as agências de passeio de escuna para as ilhas.

A fim de garantir a segurança dos turistas, o órgão fiscalizador notificou a prefeitura para que ela disponibilizasse um banco de dados para gerenciamento e controle dos passeios, escunas e barqueiros. Como você trabalha na prefeitura e é o responsável pelos bancos de dados utilizados na administração municipal, desenvolveu a estrutura demonstrada Figura 4.1.

Figura 4.1 | Banco de dados implementado

```
mysql> select * from capitao;
+------------+---------+---------------+---------+------------+
| CPF        | Nome    | Endereco      | Numero  | Celular    |
+------------+---------+---------------+---------+------------+
| 1234567891 | Jack    | Rua Robalo    |     100 | 998976554  |
| 1472583697 | Miguel  | Rua do Mar    |     250 | 999226655  |
| 3692581473 | Adriano | Rua das Ondas |    1200 | 994495885  |
| 7418529631 | Paula   | Rua Marinha   |      89 | 977885512  |
| 9876543219 | Ian     | Rua Robalo    |      55 | 988772200  |
+------------+---------+---------------+---------+------------+
```



```
5 rows in set (0.00 sec)

mysql> select * from escuna;
+---------+--------------------+--------------+
| Numero  | Nome               | capitao_CPF  |
+---------+--------------------+--------------+
|  12345  | Black Flag         |  1234567891  |
|  12346  | Caveira            |  9876543219  |
|  12347  | Brazuka            |  1472583697  |
|  12348  | Rosa Brilhante 1   |  7418529631  |
|  12349  | Tubarão Ocean      |  3692581473  |
|  12350  | Rosa Brilhante 2   |  7418529631  |
+---------+--------------------+--------------+
6 rows in set (0.00 sec)

mysql> select * from destino;
+----+--------------------+
| Id | Nome               |
+----+--------------------+
|  1 | Ilha Dourada       |
|  2 | Ilha D'areia fina  |
|  3 | Ilha Encantada     |
|  4 | Ilha dos Ventos    |
|  5 | Ilhinha            |
|  6 | Ilha Torta         |
|  7 | Ilha dos Sonhos    |
|  8 | Ilha do Sono       |
+----+--------------------+
8 rows in set (0.00 sec)

mysql> select * from passeio;
+----+------------+-----------+-------------+----------------+------------+
| Id | Data       | Hr_saida  | Hr_chegada  | escuna_Numero  | destino_Id |
+----+------------+-----------+-------------+----------------+------------+
|  1 | 2018-01-02 | 08:00:00  | 14:00:00    |          12345 |          1 |
|  2 | 2018-01-02 | 07:00:00  | 17:00:00    |          12346 |          8 |
|  3 | 2018-01-02 | 08:00:00  | 14:00:00    |          12350 |          3 |
|  4 | 2018-01-03 | 06:00:00  | 12:00:00    |          12347 |          2 |
|  5 | 2018-01-03 | 07:00:00  | 13:00:00    |          12348 |          4 |
|  6 | 2018-01-03 | 08:00:00  | 14:00:00    |          12349 |          6 |
|  7 | 2018-01-03 | 09:00:00  | 15:00:00    |          12345 |          5 |
|  8 | 2018-01-04 | 07:00:00  | 16:00:00    |          12347 |          1 |
|  9 | 2018-01-04 | 07:00:00  | 17:00:00    |          12345 |          3 |
| 10 | 2018-01-04 | 09:00:00  | 13:00:00    |          12349 |          7 |
| 11 | 2018-01-05 | 10:00:00  | 18:00:00    |          12350 |          8 |
| 12 | 2018-01-05 | 09:00:00  | 13:00:00    |          12347 |          7 |
+----+------------+-----------+-------------+----------------+------------+
12 rows in set (0.00 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Você poderá verificar o código de implementação desse banco de dados por meio do link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u4/s1/codigo_insercao.pdf> ou do QR Code.

Ao utilizar o banco de dados, os funcionários do órgão regulador relataram muita lentidão ao gerarem as consultas para o relatório dos passeios, para o qual são necessárias informações como: nome da escuna, destino, horário de saída e chegada e data do passeio. Essa notificação chegou ao seu conhecimento, e foi solicitado que você desenvolvesse uma solução para o problema relatado. Para isso, nesta seção, você aprenderá a implementar visões e índices nos bancos de dados.

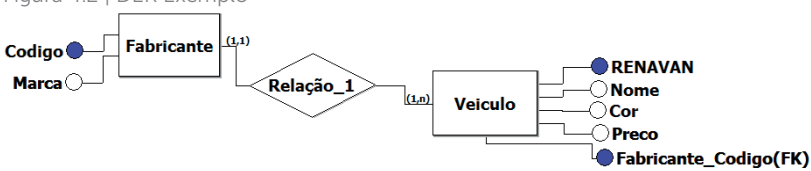
Os barqueiros contam com a sua qualificação técnica para que o problema relacionado ao banco de dados seja resolvido, não comprometendo o funcionamento dos passeios, para que o número de turistas não caia, pois poderia ocorrer um impacto econômico para o município Pronto para mais esse desafio?

Não pode faltar

Caro aluno, anteriormente, você estudou algumas técnicas de consultas avançadas utilizando a linguagem de consulta estruturada (SQL, do inglês *Structured Query Language*). Porém, conforme as consultas se tornam mais complexas, existe um comprometimento na carga de processamento. Dessa forma, são necessárias técnicas que proporcionem um maior aproveitamento dos recursos disponíveis.

Para contextualizarmos as aplicações dos conceitos discutidos nessa seção de aprendizagem, vamos tomar o exemplo de banco de dados representado no diagrama de entidade relacionamento (DER) da Figura 4.2.

Figura 4.2 | DER Exemplo



Fonte: elaborada pelo autor.

O código para implementação deste banco de dados está descrito no Quadro 4.1.

Quadro 4.1 | Scripts para exemplos

```
1  create database Car;
2  use Car;
3
4  create table Fabricante (
5      Codigo int(3) primary key auto_increment,
6      Marca char (20) not null
```

```
    );

create table Veiculo (
    RENAVAN int(8) primary key,
    Nome varchar (30) not null,
    Cor varchar (20) not null,
    Preco decimal (10,2) not null,
    fabricante_Codigo int(3) not null,
    foreign key (fabricante_Codigo) references Fabricante
(Codigo)
);
```

Fonte: elaborado pelo autor.

Depois de o banco de dados representado no DER da Figura 4.2 ter sido desenvolvido no sistema gerenciador de banco de dados (SGBD) MySQL, os registros foram inseridos conforme ilustrado na Figura 4.3.

Figura 4.3 | Inserções no BD de Exemplo

```
mysql> select * from Fabricante;
+--------+------------+
| Codigo | Marca      |
+--------+------------+
|      1 | Volk       |
|      2 | Fait       |
|      3 | Chevvroles |
|      4 | Fordys     |
|      5 | Maudi      |
|      6 | Junday     |
+--------+------------+
6 rows in set (0.00 sec)

mysql> select * from Veiculo;
+----------+-------------+----------+----------+--------------------+
| RENAVAN  | Nome        | Cor      | Preco    | Codigo_fabricante  |
+----------+-------------+----------+----------+--------------------+
|  1234567 | Cersas      | azul     | 15000.00 |                  3 |
|  1444558 | Já          | verde    | 49000.00 |                  4 |
|  2582582 | Montanha    | lilas    | 62000.00 |                  3 |
|  2589967 | Hideas      | prata    | 44000.00 |                  2 |
|  4445566 | AAR5        | azul     | 80000.00 |                  5 |
| 10102020 | Cheveiro    | preto    | 22000.00 |                  1 |
| 11111111 | EspacialFex | amarelo  | 39000.00 |                  1 |
| 11122255 | 10S         | preto    | 33000.00 |                  3 |
| 12312312 | Cersas      | rosa     | 18000.00 |                  3 |
| 12345678 | AAR3        | prata    | 44000.00 |                  5 |
| 14714714 | Jatus       | prata    | 45000.00 |                  1 |
| 22222222 | Seniel      | preto    | 18000.00 |                  2 |
| 30303030 | Estradus    | preto    | 27000.00 |                  2 |
| 33333333 | Pins        | preto    | 40000.00 |                  3 |
| 36544477 | Linearrr    | prata    | 35000.00 |                  2 |
| 44444444 | Pins        | prata    | 38000.00 |                  3 |
| 45645645 | Hideas      | branco   | 42000.00 |                  2 |
| 55220044 | Festinnn    | branco   | 25000.00 |                  4 |
| 65465465 | AAR3        | verde    | 54000.00 |                  5 |
| 66666666 | Já          | preto    | 19000.00 |                  4 |
| 74174174 | 10S         | azul     | 23000.00 |                  3 |
| 77889966 | Montanha    | preto    | 32000.00 |                  3 |
| 78889994 | Jatus       | prata    | 55000.00 |                  1 |
```

```
| 78978998 | Golos       | dourado | 82000.00 |                 1 |
| 85285285 | Linearrr    | amarelo | 55000.00 |                 2 |
| 87654321 | Golos       | azul    | 32000.00 |                 1 |
| 95195195 | Golos       | preto   | 18000.00 |                 1 |
| 96396396 | Festinnn    | marrom  | 25000.00 |                 4 |
| 98798798 | AAR5        | blindado| 40000.00 |                 5 |
+----------+-------------+---------+----------+-------------------+
29 rows in set (0.00 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Você poderá verificar o código de inserção de registros que resulta na Figura 4. 3 por meio do link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u4/s1/gerenciamento_passeios.pdf > ou do QR Code.

Agora que o banco de dados está criado e com registros inseridos, podemos explorar os conceitos de visões (*VIEW*) em banco de dados. Segundo Ferrari (2007), o recurso SQL para gerar visões é uma alternativa para visualizar os dados de uma ou mais tabelas de um BD. Uma visão, pode ser considerada uma "tabela virtual", ou, ainda, uma consulta pré-armazenada por meio de scripts. Geralmente, a técnica de `VIEW` encapsula uma seleção de dados (`SELECT`), na qual esses dados da tabela virtual são armazenados no *cache* do SGBD. Lembrando que *cache* é uma memória com capacidade de armazenamento temporário que facilita a recuperação dos dados. Um exemplo é o cache de memória dos navegadores de internet.

Uma das técnicas que podem auxiliar na diminuição na carga de processamento é a `VIEW`. Vamos entender o porquê? Utilizar uma `VIEW` para efetuar seleção de dados, torna as consultas mais rápidas e exige uma carga de processamento menor. Isso ocorre por que uma `VIEW` não necessita fazer o retrabalho ao executar um `SELECT`, pois a seleção já está pré-armazenada. Mas, você deve estar se perguntando: e quando a tabela receber alterações, inserções ou exclusões? Para isso, o sistema de gerenciamento de banco de dados se encarrega de atualizar a `VIEW` sempre que algo for alterado nas tabelas. A sintaxe utilizada para desenvolver uma `VIEW` está descrita a seguir:

```
CREATE VIEW [nome_da_VIEW] AS
SELECT [coluna]
FROM [tabela]
WHERE [condições];
```

Exemplificando

Partindo do cenário desenvolvido, vamos desenvolver uma VIEW que selecione o fabricante, o nome, a cor e o preço de um veículo, quando o valore for menor que R$ 50.000,00. Para isso, foi desenvolvida a sintaxe:

```
CREATE VIEW v_select1 AS
      SELECT veiculo.nome as "Veiculo", fabricante.
      marca as "Marca", veiculo.cor as "Cor", veiculo.
      preco as "Valor"
      FROM veiculo INNER JOIN fabricante
      WHERE veiculo.fabricante_Codigo = fabricante.
      Codigo AND veiculo.preco<= 50000;
```

Você se lembra que as VIEWs são consideradas "tabelas virtuais"? Pois bem, para visualizá-las, basta exibirmos as tabelas inseridas no BD, por meio da sintaxe SHOW TABLES. Observe na Figura 4.4.

Figura 4.4 | *Inserts* no BD de Exemplo

Fonte: captura de tela do MySQL, elaborada pelo autor.

Assimile

Repare que os nomes utilizados para os BDs, tabelas e colunas seguem as regras da maioria das linguagens de programação, não sendo permitido iniciar com números ou caracteres, palavras reservadas e etc. Alguns recursos utilizados no SQL devem ser identificados por meio de um prefixo, para que possamos diferenciá-los dos demais.

Por exemplo, é uma boa prática nomear as VIEWS com o prefixo v_ nomeDaView. Dessa forma, ao consultarmos as VIEWS desenvolvidas no BD, poderemos diferenciá-las das tabelas.

Para utilizarmos uma `VIEW` para exibir uma consulta, devemos utilizar a sintaxe a seguir:

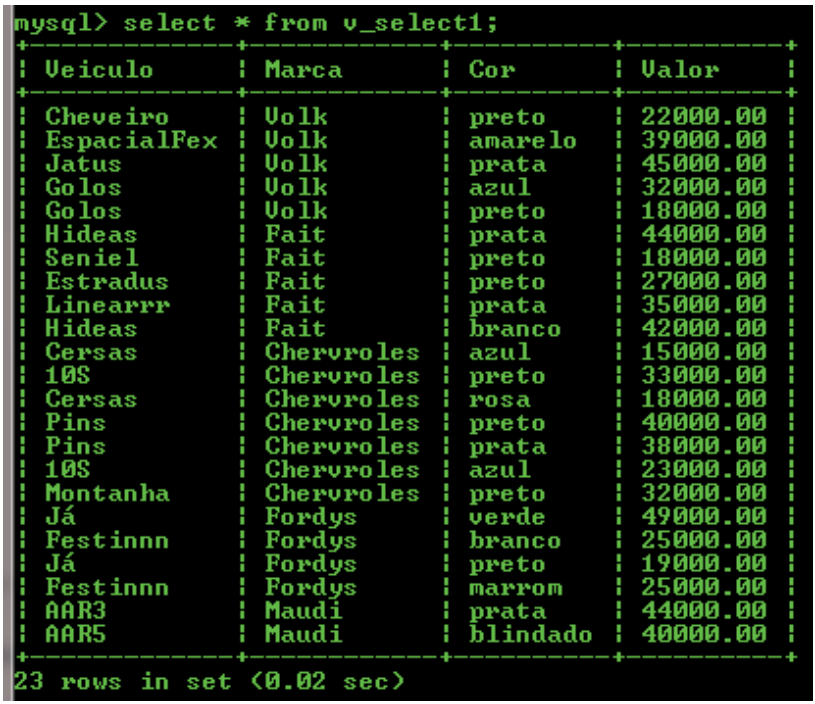
```
SELECT * FROM [nome_da_VIEW];
```

Em nosso exemplo desenvolvido anteriormente, teríamos:

```
SELECT * FROM v_select1;
```

Isso geraria a saída representada na Figura 4.5.

Figura 4.5 | `VIEW` no BD de exemplo



```
mysql> select * from v_select1;
+-------------+-----------+----------+-----------+
| Veiculo     | Marca     | Cor      | Valor     |
+-------------+-----------+----------+-----------+
| Cheveiro    | Volk      | preto    | 22000.00  |
| EspacialFex | Volk      | amarelo  | 39000.00  |
| Jatus       | Volk      | prata    | 45000.00  |
| Golos       | Volk      | azul     | 32000.00  |
| Golos       | Volk      | preto    | 18000.00  |
| Hideas      | Fait      | prata    | 44000.00  |
| Seniel      | Fait      | preto    | 18000.00  |
| Estradus    | Fait      | preto    | 27000.00  |
| Linearrr    | Fait      | prata    | 35000.00  |
| Hideas      | Fait      | branco   | 42000.00  |
| Cersas      | Chevroles | azul     | 15000.00  |
| 10S         | Chevroles | preto    | 33000.00  |
| Cersas      | Chevroles | rosa     | 18000.00  |
| Pins        | Chevroles | preto    | 40000.00  |
| Pins        | Chevroles | prata    | 38000.00  |
| 10S         | Chevroles | azul     | 23000.00  |
| Montanha    | Chevroles | preto    | 32000.00  |
| Já          | Fordys    | verde    | 49000.00  |
| Festinnn    | Fordys    | branco   | 25000.00  |
| Já          | Fordys    | preto    | 19000.00  |
| Festinnn    | Fordys    | marrom   | 25000.00  |
| AAR3        | Maudi     | prata    | 44000.00  |
| AAR5        | Maudi     | blindado | 40000.00  |
+-------------+-----------+----------+-----------+
23 rows in set (0.02 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Para excluir uma `VIEW`, a sintaxe deve ser:

```
DROP VIEW [nome_da_VIEW];
```

Segundo Date (2004), as vantagens de utilizar `VIEWs` são:

- Economia de tempo: em vista do desenvolvimento, ocorre uma diminuição na carga de criação de comandos

SELECT com finalidades similares, pois é possível utilizar os dados já armazenados na tabela virtual.

- Velocidade de acesso: devido às VIEWs estarem pré-armazenadas, a carga de processamento é menor, consequentemente, a resposta para o usuário será mais rápida.
- Ocultação da complexidade: do ponto de vista do usuário, ele não necessita saber quantos e quais campos existem em um BD, nem a forma como são selecionados.

Assimile

Como podemos diferenciar a vantagem de processamento de um SELECT e de uma VIEW? Para isso, é necessário ter uma grande base de dados. Imagine a base de dados existentes em um supermercado, sites de vendas, entre diversos outros. A quantidade de verificações que o SQL necessita fazer é bem maior, desta forma, a utilização de uma ou outra técnica para selecionar os dados pode ser determinante no desempenho das consultas.

Índices em banco de dados relacional

Antes de discutirmos diretamente o índice, vamos resgatar o conceito do mecanismo utilizado para fazer consultas nos BDs. Quando é executada uma seleção de dados, o SGBD faz a checagem de similaridade em um ou mais campos, operação chamada de *Table Scan*. Como já estudado, nesse processo em tabelas com milhares de registros, o tempo de verificação tende a ser muito grande, comprometendo a qualidade do serviço (QoS, do inglês *Quality of Service*).

Segundo Silberschatz (2010), a utilização dos índices é opcional para a seleção de dados, pois os índices são considerados estruturas redundantes. O SGBD pode decidir quais índices devem ser criados, porém, nem sempre essa escolha automatizada pode trazer algum benefício no processamento.

Reflita

Conforme visto, a utilização dos índices não é uma obrigatoriedade nas estruturas das tabelas em um banco de dados. Porém, o fato de ser opcional não quer dizer que esse recurso não tenha serventia no SQL. Então, quando um desenvolvedor/administrador de banco de dados deve utilizar um índice em uma tabela?

Uma analogia para auxiliar na compreensão desse recurso são os índices utilizados nos livros ou revistas. Um leitor, ao buscar por um assunto específico, pode consultar a página de índices, que nada mais é que uma lista ordenada, que fornece as referências cruzadas de página e conteúdo. Assim, é mais fácil e rápido para o leitor conseguir localizar as informações no meio de tantas páginas, dados, informações, etc. Esse processo parece bem familiar ao recurso de índice em banco de dados, não?

Segundo Silberschatz (2010), o recurso de índice (`INDEX`, no MySQL) não era admitido até a versão SQL:1999. Após isso, os engenheiros buscaram um recurso para diminuir a taxa de processamento nas buscas nas tabelas e para imposição das restrições de integridade. Para isso, por meio da palavra reservada `INDEX`, devemos utilizar as seguintes sintaxes:

Declarar um índice, quando no desenvolvimento da tabela:

```
CREATE TABLE [nomeDaTabela] (
     Campo1 tipo(tamanho),
     Campo2 tipo(tamanho),
     INDEX(Campo1)
     );
```

Declarar um índice, em uma tabela existente no BD:

```
CREATE TABLE [nomeDoIndice] ON
[nomeDaTabela](Campo);
```

Para auxiliar a compreensão da sintaxe SQL para índices, vamos criar um índice na chave primária RENAVAN (Registro Nacional de Veículos Automotores) da tabela veículo, no exemplo utilizado nessa seção de estudos. Com isso é utilizada a sintaxe SQL apresentada a seguir:

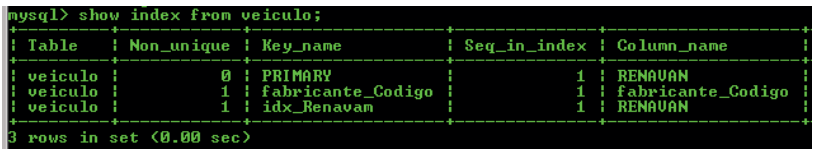
```
create index idx_Renavam ON
    veiculo(RENAVAN);
```

Embora o MySQL retorne a mensagem "*Query OK, 0 rows affected*", para nos certificarmos que os índices foram criados, é necessário utilizar a sintaxe a seguir:

```
SHOW INDEX FROM [nomeDaTabela];
```

Ao efetuar a consulta no exemplo desenvolvido, pode-se observar o resultado apresentado na Figura 4.6.

Figura 4.6 | Exemplo *Show* index



Fonte: captura de tela do MySQL, elaborada pelo autor.

Você deve ter reparado que, na terceira linha, existe uma chave nomeada como "*idx_Renavam*", pois, assim como a `VIEW`, os nomes dos índices necessitam de um prefixo, para fins de identificação. Veja que o uso do prefixo nos nomes não é obrigatório, porém, no dia a dia, é uma boa prática, pois evita que ocorra confusão entre os recursos desenvolvidos. Para utilizar um índice, a sintaxe necessária pode ser observada a seguir:

```
SELECT [coluna] FROM [nomeDaTabela]
    USE INDEX (nomeDoIndice)
    WHERE [condições];
```

No exemplo desenvolvido,

```
SELECT nome AS "Veiculo", cor AS "Cor",
    Preço AS "Valor" FROM veiculo
    USE INDEX(idx_Renavam)
    WHEREpreco<= 50000;
```

Em que houve o retorno gerado na Figura 4.7.

Figura 4.7 | Exemplo utilização do index

Cersas	azul	15000.00
Já	verde	49000.00
Hideas	prata	44000.00
Cheveiro	preto	22000.00
EspacialFex	amarelo	39000.00
10S	preto	33000.00
Cersas	rosa	18000.00
AAR3	prata	44000.00
Jatus	prata	45000.00
Seniel	preto	18000.00
Estradus	preto	27000.00
Pins	preto	40000.00
Linearrr	prata	35000.00
Pins	prata	38000.00
Hideas	branco	42000.00
Festinnn	branco	25000.00
Já	preto	19000.00
10S	azul	23000.00
Montanha	preto	32000.00
Golos	azul	32000.00
Golos	preto	18000.00
Festinnn	marrom	25000.00
AAR5	blindado	40000.00

23 rows in set (0.04 sec)

Fonte: captura de tela do MySQL, elaborada pelo autor.

Os índices, quando utilizados dentro de um `SELECT`, podem conter recursos como: junções, funções agregadas, subconsultas, e quaisquer outras técnicas do SQL que se mostrem necessárias. Isso por que a intenção em se desenvolver um índice é ganhar eficiência nas consultas, dessa forma, podemos utilizar diversos recursos para a consulta.

Para excluir um índice, a sintaxe utilizada pode ser observada a seguir:

```
DROP INDEX (nomeDoIndice);
```

`FULLTEXT` em banco de dados relacional

Outro recurso que tem uma função muito parecida com o `INDEX` é o `FULLTEXT`. Silberschatz (2010) afirma que esse recurso tem

a capacidade de buscar um trecho dentro de várias *strings*, assim como a função "localizar" existente nos navegadores de internet, editores de texto, etc. Para isso, é utilizada a sintaxe:

```
ALTER TABLE [nome_tabela] ADD FULLTEXT (nome_
da_coluna);
```

Nesse comando, ao especificar uma determinada coluna como `FULLTEXT`, a mesma passa a ter as *strings* no interior de um texto, monitoradas.

Para utilizar esse recurso, deve-se utilizar a sintaxe descrita a seguir:

```
SELECT [coluna] from nome_da_tabela
    WHERE    MATCH(coluna)    AGAINST('palavra_
    desejada');
```

Isso permite buscar palavras dentro de longos textos, em que:

- `MATCH(coluna):` tem a função de informar ao sistema de gerenciamento de banco de dados a coluna na qual deve ser usada na consulta do `FULLTEXT`;
- `AGAINST('palavra_desejada')`: tem como objetivo informar ao sistema de gerenciamento de banco de dados a palavra chave que deve ser buscada no `FULLTEXT.`

Pesquise mais

Não é somente o banco de dados que possui o recurso de "leitura" de *strings*. O *Optical Character Recognition* (OCR) é uma tecnologia que permite reconhecer caracteres a partir de diversas fontes. Conheça mais sobre o assunto no artigo a seguir:

MORIMOTO, C. E. **OCR**. 26 jun. 2005. Disponível em <https://www.hardware.com.br/termos/ocr> Acesso em: 17 set. 2018.

Sem medo de errar

Você trabalha no setor de informática da prefeitura de uma cidade litorânea e é o responsável por todos os desenvolvimentos relacionados aos bancos de dados. Os fiscais responsáveis pelo

controle dos passeios de escuna abriram uma reclamação de lentidão no sistema. Essa reclamação chegou até seu setor, e você deve desenvolver uma solução para a lentidão. Ao explorar os scripts do sistema, você verificou que, atualmente, a consulta é feita por meio do comando:

```
SELECT escuna.nome AS "Escuna", destino.nome AS
        "Ilha", Hr_saida AS "Saida", Hr_chegada
        AS "Chegada", Data
        FROM passeio INNER JOIN escuna, destino
        WHERE passeio.escuna_Numero = escuna.
        numero AND passeio.destino_Id = destino.
        Id ORDER BY passeio.data;
```

Nele, foi gerada a saída representada na Figura 4.8.

Figura 4.8 | Consulta Pier

```
+-----------------+-----------------+----------+----------+------------+
| Escuna          | Ilha            | Saida    | Chegada  | Data       |
+-----------------+-----------------+----------+----------+------------+
| Black Flag      | Ilha Dourada    | 08:00:00 | 14:00:00 | 2018-01-02 |
| Caveira         | Ilha do Sono    | 07:00:00 | 17:00:00 | 2018-01-02 |
| Rosa Brilhante 2| Ilha Encantada  | 08:00:00 | 14:00:00 | 2018-01-02 |
| Rosa Brilhante 1| Ilha dos Ventos | 07:00:00 | 13:00:00 | 2018-01-03 |
| Black Flag      | Ilhinha         | 09:00:00 | 15:00:00 | 2018-01-03 |
| Tubarão Ocean   | Ilha Torta      | 08:00:00 | 14:00:00 | 2018-01-03 |
| Brazuka         | Ilha D'areia fina| 06:00:00 | 12:00:00 | 2018-01-03 |
| Tubarão Ocean   | Ilha dos Sonhos | 09:00:00 | 13:00:00 | 2018-01-04 |
| Brazuka         | Ilha Dourada    | 07:00:00 | 16:00:00 | 2018-01-04 |
| Black Flag      | Ilha Encantada  | 07:00:00 | 17:00:00 | 2018-01-04 |
| Brazuka         | Ilha dos Sonhos | 09:00:00 | 13:00:00 | 2018-01-05 |
| Rosa Brilhante 2| Ilha do Sono    | 10:00:00 | 18:00:00 | 2018-01-05 |
+-----------------+-----------------+----------+----------+------------+
12 rows in set (0.58 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Para solucionar o problema de lentidão, você pode optar pelo uso de `VIEWS`, criando uma visão (`VIEW`) na tabela de controle dos passeios que saem do píer da cidade.

Para solucionar tal apontamento feito pelos fiscais, que pode comprometer o embarque dos turistas no píer, você deverá desenvolver uma sintaxe SQL, como a representada a seguir:

```
CREATE VIEWv_consulta AS
     SELECT escuna.nomeAS "Escuna", destino.nomeAS
     "Ilha", Hr_saidaAS "Saida", Hr_chegadaAS
     "Chegada", Data
          FROM passeio INNER JOIN escuna, destino
```

```
WHERE passeio.escuna_Numero = escuna.
numeroAND passeio.destino_Id = destino.
IdORDER BYpasseio.data;
```

Desta forma, é possível utilizar a `VIEW` desenvolvida por meio do comando SQL representado a seguir:

```
SELECT * FROM v_consulta;
```

Essa sintaxe irá gerar a saída representada na Figura 4.9.

Figura 4.9 | Consulta Píer por `VIEW`

```
+-----------------+-------------------+----------+-----------+------------+
| Escuna          | Ilha              | Saida    | Chegada   | Data       |
+-----------------+-------------------+----------+-----------+------------+
| Black Flag      | Ilha Dourada      | 08:00:00 | 14:00:00  | 2018-01-02 |
| Caveira         | Ilha do Sono      | 07:00:00 | 17:00:00  | 2018-01-02 |
| Rosa Brilhante 2| Ilha Encantada    | 08:00:00 | 14:00:00  | 2018-01-02 |
| Rosa Brilhante 1| Ilha dos Ventos   | 07:00:00 | 13:00:00  | 2018-01-03 |
| Black Flag      | Ilhinha           | 09:00:00 | 15:00:00  | 2018-01-03 |
| Tubarão Ocean   | Ilha Torta        | 08:00:00 | 14:00:00  | 2018-01-03 |
| Brazuka         | Ilha D'areia fina | 06:00:00 | 12:00:00  | 2018-01-03 |
| Tubarão Ocean   | Ilha dos Sonhos   | 09:00:00 | 13:00:00  | 2018-01-04 |
| Brazuka         | Ilha Dourada      | 07:00:00 | 16:00:00  | 2018-01-04 |
| Black Flag      | Ilha Encantada    | 07:00:00 | 17:00:00  | 2018-01-04 |
| Brazuka         | Ilha dos Sonhos   | 09:00:00 | 13:00:00  | 2018-01-05 |
| Rosa Brilhante 2| Ilha do Sono      | 10:00:00 | 18:00:00  | 2018-01-05 |
+-----------------+-------------------+----------+-----------+------------+
12 rows in set (0.10 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Note que, ao utilizar o `SELECT`, a resposta foi de 0,58 segundos. Já com a utilização da `VIEW`, o tempo de resposta caiu para 0,10 segundos. Dessa forma, o objetivo de diminuir o tempo de resposta e a taxa de processamento na geração do relatório, conforme reclamação do órgão regulamentador notificou, foi solucionado.

Assim, os barqueiros podem continuar efetuando os passeios com os turistas, não comprometendo financeiramente o município.

Avançando na prática

Análise de obras literárias

Descrição da situação-problema

Um grupo de pesquisa universitária na área de literatura está solicitando uma consultoria para desenvolver um banco de dados para análise de diversas obras. Inicialmente, eles pediram um

teste com obras de Machado de Assis. A fim de extrair algumas informações de suas obras, foi desenvolvido um banco de dados com alguns trechos, conforme pode ser observado no Quadro 4.2.

Quadro 4.2 | Script BD passagens de obras de Machado de Assis*

```
CREATE TABLE MachadoDeAssis(
Livro varchar(30) NOT NULL,
Ano int(4) NOT NULL,
Texto varchar(200) NOT NULL
);
insert MachadoDeAssis values ("A cartomante", 1884, "Realmente, era graciosa e viva nos gestos, olhos cálidos, boca fina e interrogativa."),
("A causa secreta", 1885, "Insensivelmente estendeu a mão e apertou o pulso ao marido, risonha e agradecida, como se acabasse de descobrir-lhe o coração."),
("Falenas", 1870, "Rosa branca do céu, perfume, alento, vida. Palpita o coração já crente, já desperto; Povoa-se num dia o que era agro deserto; "),
("Iaia Garcia", 1878, "A meu marido. Iaiá beijou com ardor a singela dedicatória, como beijaria a madrasta se ela lhe aparecesse naquele instante."),
("Mariana", 1871, "Mariana foi com ele até à porta, interrogando baixo e procurando-lhe no rosto a verdade que a boca não queria dizer."),
("Quincas Borbas", 1891, "Era daquela casta de mulheres que o tempo, como um escultor vagaroso, não acaba logo, e vai polindo ao passar dos longos dias.")
;
```

Fonte: adaptado de Assis ([s.d.; s.p.]).

Em conversa com os pesquisadores do grupo, foi requerido que você desenvolva uma seleção de dados para que sejam respondidos os seguintes questionamentos:

1. Quantas vezes a palavra "marido" aparece nos trechos cadastrados no BD.

2. Qual o nome da obra e o ano que a palavra "coração" aparece nos trechos.

Resolução da situação-problema

Ao desenvolver um banco de dados com diversos trechos das obras de Machado de Assis para responder aos dois questionamentos, o pesquisador nos obriga a efetuar seleções de *strings*. Para isso, deve-se desenvolver um `FULLTEXT`, em que será possível fazer as consultas desejadas. Para isso, você deverá desenvolver a sintaxe:

```
ALTER TABLE MachadoDeAssis ADD FULLTEXT(texto);
```

Para responder ao primeiro questionamento, deverá ser utilizado o `FULLTEXT` desenvolvido por meio da sintaxe a seguir:

```
SELECT COUNT(*) FROM MachadoDeAssis
WHERE MATCH(texto) AGAINST("marido");
```

Com isso, será possível obter os resultados demonstrados na Figura 4.10.

Figura 4.10 | Questionamento 1

```
mysql> SELECT COUNT(*) FROM MachadoDeAssis
    -> WHERE MATCH(texto) AGAINST("marido");
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.04 sec)
```

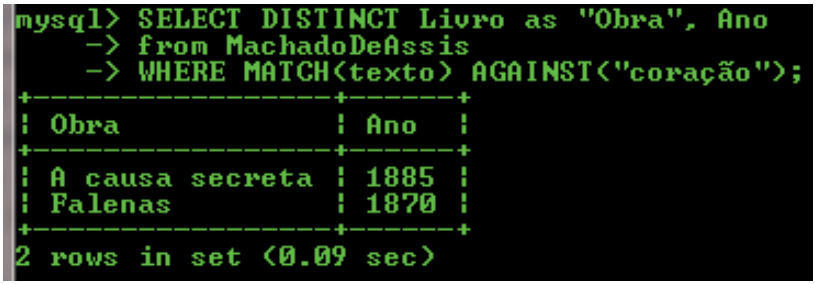
Fonte: captura de tela do MySQL, elaborada pelo autor.

Já para responder ao segundo questionamento, deverá ser utilizada a sintaxe a seguir:

```
SELECT DISTINCT Livro as "Obra", Ano
from MachadoDeAssis
WHERE MATCH(texto) AGAINST("coração");
```

Com isso, poderá gerar os resultados demonstrados na Figura 4.11.

Figura 4.11 | Questionamento 2



Fonte: captura de tela do MySQL, elaborada pelo autor.

Faça valer a pena

1. Quando os bancos de dados, as aplicações, as redes de computadores e demais recursos tecnológicos são desenvolvidos, uma grande preocupação é com a eficiência do desenvolvimento e o melhor aproveitamentos dos recursos disponíveis.
Ao desenvolver as seleções nos bancos de dados, assim como em outras áreas da tecnologia da informação, a rapidez e eficiência das *Querys* também são uma preocupação. Para isso, a linguagem de programação de banco de dados SQL possui alguns recursos que visam fornecer ferramentas para que o desenvolvedor possa garantir a QoS (*Qualityof Service* – Qualidade de Serviço).

Observe o texto a seguir:
O recurso SQL denominado _____ tem a capacidade de encapsular um `SELECT`, em que é criada uma _____ não física no *cache* do sistema de gerenciamento de banco de dados. Quando utilizada, a sua execução é mais rápida em comparação a um `SELECT` não encapsulado, pois as seleções já estão _____.

Assinale a alternativa que completa as lacunas corretamente.

a) *INDEX* – tabela – pré-armazenadas.
b) *VIEW* – atributo – disponibilizadas.
c) *VIEW* – tabela – pré-armazenadas.
d) *INDEX* – atributo – armazenadas.
e) *VIEW* – visão – armazenadas.

2. Buscar palavras dentro de textos é uma forma de se obter informações acerca de um determinado assunto, e esse recurso pode ser utilizado na tomada de decisão. Por exemplo, na posse do presidente eleito dos Estados Unidos da América, Donald Trump, em 2017, foi utilizada a técnica de análise em nuvem das palavras, conforme pode ser observado na Figura abaixo.

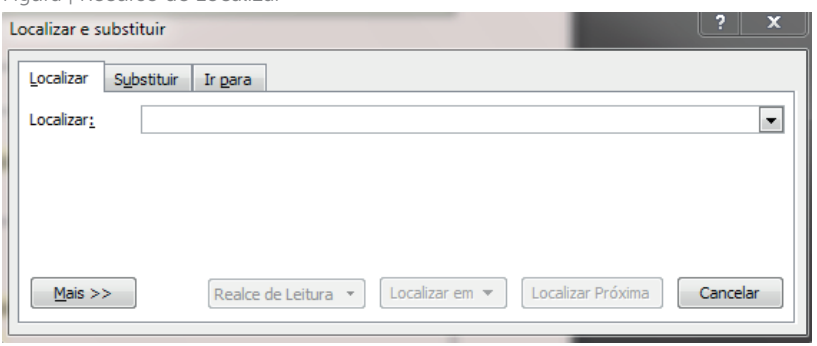
Figura | Discurso Trump

Fonte: <https://www.publico.pt/2017/01/20/mundo/noticia/as-palavras-que-mais-ocuparam-os-discursos-dos-ultimos-tres-presidentes-dos-eua-1759146>. Acesso em: 22 jun. 2018.

Esses recursos disponíveis nos softwares podem ser utilizados como uma poderosa ferramenta na busca de informações. Nesse exemplo, as palavras de maior tamanho são as que foram repetidas mais vezes, porem todas as palavras da figura foram ditas pelo menos uma vez.

Agora, observe a Figura a seguir.

Figura | Recurso de Localizar



Fonte: captura de tela do Microsoft Word, elaborada pelo autor.

O recurso demonstrado na Figura acima tem como intuito fazer a seleção de uma palavra específica em um texto longo. Analogamente, assinale o recurso SQL que possa ser utilizado da mesma forma em um banco de dados:

a) VIEW.
b) FULLTEXT.
c) PROCEDURES.
d) FUNCTIONS.
e) INDEX.

3. Os índices são recursos importantes, que permitem tanto ao administrador de banco de dados quanto ao desenvolvedor agilizarem a forma de realizar as consultas, por meio de uma sintaxe simples, possibilitada pela linguagem SQL.
O mecanismo de funcionamento do `INDEX` no SQL nos permite fazer uma analogia com os índices disponíveis nos livros, revistas, etc. Nesses casos, a busca por determinado assunto é feita de forma muito mais rápida, pois existe uma correlação entre os assuntos e os seus respectivos números de páginas. O esforço do usuário é menor para localizar alguma informação, da mesma forma que a taxa de processamento do servidor será menor ao serem utilizados os índices nas consultas.
Quando esse recurso é utilizado nas consultas em base de dados com poucos registros, o tempo de resposta não é perceptível, assim como não há diferença na taxa de processamento do servidor. Porém, quando o índice é utilizado nas consultas em tabelas com muitos registros, o aumento da performance e a qualidade do serviço ficam evidentes ao usuário.

Observe as afirmativas a seguir:
I. O `INDEX`, no MySQL, tem o intuito de impor as restrições de disponibilidade nos bancos de dados.
II. Um índice não pode ser declarado na criação de uma tabela.
III. No SQL, a sintaxe utilizada para fazer a exclusão de um índice deve ser:
`DROP INDEX (nomeDoIndice);`

Assinale a alternativa correta:

a) Somente a alternativa I está correta.
b) Somente as alternativas I e II estão corretas.
c)Somente a alternativa III está correta.
d) Somente as alternativas II e III estão corretas.
e) As alternativas I, II e III estão corretas.

Seção 4.2

Controle transacional

Diálogo aberto

Caro aluno, certamente você já deve ter tido a desagradável experiência de perder os seus dados. Seja no *smartphone*, ao apagar uma foto por acidente, ou no computador, quando um documento não foi salvo, sem contar os textos perdidos. Esse tipo de "acidente", nas empresas, pode ter consequências desastrosas. Por isso, os sistemas gerenciadores de banco de dados (SGBD) e as linguagens de programação de banco de dados devem ter mecanismos que possam garantir a integridade das informações arquivadas nas tabelas.

A cidade litorânea em que você trabalha para a prefeitura tem a população vivendo basicamente de turismo. Por ser contratado como responsável pelos bancos de dados desenvolvidos nos projetos, você está ligado a um novo projeto para gerenciamento e controle dos passeios de escunas e os barqueiros, a fim de se garantir a segurança dos turistas. O sistema de consulta passou por uma mudança significativa em sua última atualização, quando você implementou uma visão (`VIEW`), fazendo com que o tempo de consulta diminuísse. Agora, um colaborador da prefeitura necessita fazer uma alteração no nome de um dos destinos, porém, por uma falha de operação, todos os nomes foram alterados de forma errada, como pode ser observado na Figura 4.12.

Figura 4.12 | Consulta a tabela destino

```
mysql> select * from destino;
+----+----------------------+
| Id | Nome                 |
+----+----------------------+
|  1 | Pequena ilha do Mar  |
|  2 | Pequena ilha do Mar  |
|  3 | Pequena ilha do Mar  |
|  4 | Pequena ilha do Mar  |
|  5 | Pequena ilha do Mar  |
|  6 | Pequena ilha do Mar  |
|  7 | Pequena ilha do Mar  |
|  8 | Pequena ilha do Mar  |
+----+----------------------+
8 rows in set (0.00 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Tal falha acarretou uma indisponibilidade do sistema por algumas horas, gerando insatisfação dos turistas e donos de escunas. Para isso, a fim de se prevenir que o mesmo problema ocorra futuramente, foi solicitado que você desenvolva as seguintes atividades:

- A alteração do COMMIT para que as alterações não sejam gravadas automaticamente;
- A criação de um ponto de restauração no banco;
- Um teste para gerar o mesmo erro que o colaborador da prefeitura cometeu, ou seja, nomear todos os registros da com o mesmo nome;
- Uma utilização teste do ponto de restauração criado;
- Gravar as alterações feitas;
- Criar um novo ponto de restauração, pois o anterior será apagado pela gravação das alterações.

Tais ferramentas aplicadas na situação problema irão permitir a garantia da integridade dos bancos de dados por você desenvolvidos. Além disso, os comerciantes locais os turistas contam com a sua competência técnica em banco de dados para que problemas como os relatados anteriormente não venham causar prejuízos e insatisfação.

Dessa forma, nesta seção de aprendizagem você vai poder aprofundar o seu conhecimento em controle transacional em banco de dados, utilizando as sintaxes `SAVEPOINT`, `COMMIT` e `ROLLBACK`, presentes no SQL. Você não pode deixar de aplicar tais recursos de controle de transação em banco de dados! Pronto para mais esse desafio?

Não pode faltar

Nesta seção de estudo, você vai ver os recursos da linguagem SQL que efetuam o controle transacional no banco de dados. Por meio das sintaxes `COMMIT`, `ROLLBACK`, e `SAVEPOINT`, será possível compreender como esses recursos podem auxiliar nas tarefas para garantir a integridade dos bancos de dados. Essas discussões são muito importantes no dia a dia do administrador de banco de dados

(BD), pois permitem criar pontos de salvamento e retornar para um ponto anterior ou, ainda, cancelar uma alteração.

Para que possamos demonstrar as técnicas discutidas nesta seção de aprendizagem, vamos tomar como exemplo o script demonstrado no Quadro 4.3.

Quadro 4.3 | *Script* Criação de BD e Tabelas

```
1  CREATE TABLE Enfermeiro(
2  Coren INT PRIMARY KEY,
3      Nome varchar(50) not null
4  );
5
6  CREATE TABLE Paciente(
7  Numint PRIMARY KEY,
8      Nome VARCHAR(50) NOT NULL
9  );
10
11 CREATE TABLE Remedio(
12 Cod INT PRIMARY KEY,
13     Nome VARCHAR(50) NOT NULL
14 );
15
16 CREATE TABLE Medicacao(
17 Id INT PRIMARY KEY AUTO_INCREMENT,
18 Data DATE NOT NULL,
19 Hora TIME NOT NULL,
20 PacienteNum INT NOT NULL,
21 RemedioCod INT NOT NULL,
22 EnfermeiroCoren INT NOT NULL,
23 FOREIGN KEY(PacienteNum) REFERENCES Paciente(Num),
24 FOREIGN KEY(RemedioCod) REFERENCES Remedio(Cod),
25 FOREIGN KEY (EnfermeiroCoren) REFERENCES Enfermeiro(Coren)
26 );
27
```

Fonte: elaborado pelo autor.

Para que possamos compreender as operações realizadas de controle transacional, foram inseridos os registros das tabelas, conforme demonstrado no Quadro 4.4.

Quadro 4.4 | Script Inserção de dados nas tabelas

```
INSERT Enfermeiro VALUES (11111, “Enfermeiro 1”),
(22222, “Enfermeiro 2”),
(33333, “Enfermeiro 3”);

INSERT Paciente VALUES (1000, “Paciente A”),
(1001, “Paciente B”),
(1002, “Paciente C”),
(1003, “Paciente D”),
(1004, “Paciente E”),
(1005, “Paciente F”),
(1006, “Paciente G”),
(1007, “Paciente H”),
(1008, “Paciente I”);

INSERT Remedio VALUES (100, “Controle de pressao”),
(101, “Problemas no estomago”),
(102, “Soro”),
(103, “Calmante”),
(104, “Analgesico”),
(105, “Rins”);

INSERT Medicacao VALUES
(0, current_date, “05:00:00”, 1003, 104, 11111),
(0, current_date, “08:00:00”, 1001, 100, 11111),
(0, current_date, “08:20:00”, 1007, 102, 11111),
(0, current_date, “08:30:00”, 1007, 105, 11111),
(0, current_date, “09:00:00”, 1004, 104, 22222),
(0, current_date, “09:30:00”, 1005, 105, 33333),
(0, current_date, “10:20:00”, 1001, 103, 11111),
(0, current_date, “12:00:00”, 1008, 102, 22222),
(0, current_date, “12:20:00”, 1002, 105, 22222),
(0, current_date, “13:30:00”, 1001, 100, 11111),
(0, current_date, “15:00:00”, 1003, 104, 22222),
(0, current_date, “16:00:00”, 1001, 103, 11111),
(0, current_date, “20:30:00”, 1008, 100, 22222),
(0, current_date, “21:00:00”, 1002, 105, 11111),
(0, current_date, “21:10:00”, 1006, 102, 11111),
```

```
38 (0, current_date, "23:00:00", 1003, 104, 33333);
39
```

Fonte: elaborado pelo autor.

Nesse exemplo, vamos tomar informações como data, hora, paciente, remédio e o enfermeiro que foi responsável pela administração do medicamento, considerando a saída, conforme demonstrado na Figura 4.13.

Figura 4.13 | Consulta ao Banco de Dados

Registro	Data	Hora	Paciente	Medicacao	Enfermeiro
1	2018-07-01	05:00:00	Paciente D	Analgesico	Enfermeiro 1
2	2018-07-01	08:00:00	Paciente B	Controle de pressao	Enfermeiro 1
3	2018-07-01	08:20:00	Paciente H	Soro	Enfermeiro 1
4	2018-07-01	08:30:00	Paciente H	Rins	Enfermeiro 1
5	2018-07-01	09:00:00	Paciente E	Analgesico	Enfermeiro 2
6	2018-07-01	09:30:00	Paciente F	Rins	Enfermeiro 3
7	2018-07-01	10:20:00	Paciente B	Calmante	Enfermeiro 1
8	2018-07-01	12:00:00	Paciente I	Soro	Enfermeiro 2
9	2018-07-01	12:20:00	Paciente C	Rins	Enfermeiro 2
10	2018-07-01	13:30:00	Paciente B	Controle de pressao	Enfermeiro 1
11	2018-07-01	15:00:00	Paciente D	Analgesico	Enfermeiro 2
12	2018-07-01	16:00:00	Paciente B	Calmante	Enfermeiro 1
13	2018-07-01	20:30:00	Paciente I	Controle de pressao	Enfermeiro 2
14	2018-07-01	21:00:00	Paciente C	Rins	Enfermeiro 1
15	2018-07-01	21:10:00	Paciente G	Soro	Enfermeiro 1
16	2018-07-01	23:00:00	Paciente D	Analgesico	Enfermeiro 3

Fonte: captura de tela do MySQL, elaborada pelo autor.

Você poderá verificar o código que gera a Figura 4.13 acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u4/s2/selecao_de_atendimento.pdf> ou por meio do QR Code.

Controle de Transação

Segundo Silberschatz (2010), uma transação pode ser considerada um conjunto de operações com uma única unidade lógica de trabalho, em que uma instrução pode acessar, alterar ou excluir vários dados em uma ou mais tabelas. Esse mecanismo pode ser iniciado pela aplicação por meio de uma linguagem de programação de alto nível ou, ainda, por uma linguagem de programação de banco de dados, como o SQL.

O controle das transações ocorridas em um banco de dados deve garantir a integridade dos dados contidos nas tabelas, para isso, deve-se manter:

- Atomicidade: é a garantia de que todas as operações serão corretamente refletidas em todo o banco de dados. Caso isso não seja possível, nenhuma das operações deve ser concluída, evitando que ocorra somente uma execução parcial de uma transação.

- Consistência:é a garantia de se manter os dados íntegros durante e mesmo com a finalização de uma transação efetuada no banco de dados.

- Isolamento: embora várias transações ocorram simultaneamente, na visão dos mecanismos envolvidos nos bancos de dados, tais execuções devem se comportar de forma isolada, de forma que, uma transação não fique "sabendo" o que está ocorrendo nas demais transações em execução.

- Durabilidade: após as transações serem finalizadas com sucesso, deve-se garantir que as alterações persistam nas tabelas do banco de dados.

Essas propriedades das transações nos bancos de dados são conhecidas como **ACID**. Elas devem estar em um dos seguintes estados:

- Ativa: o estado inicial e a transação permanecem nesse estado no momento em que está sendo executada uma transação.

- Parcialmente confirmada: parte da execução não pode ser finalizada, por um erro de alguma natureza.

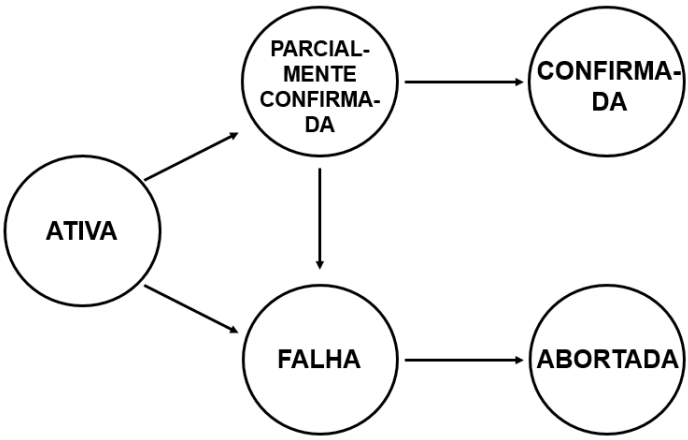
- Falha: não consegue realizar as execuções pertinentes.

- Abortada: depois que uma transação foi feita no banco de dados, poderá ser revertida em um estado anterior.

- Confirmada: quando o término de uma transação é feito com sucesso.

A Figura 4.14, por meio do diagrama, demonstra as possíveis situações dos estados das transações, conforme pode ser observado.

Figura 4.14 | Diagrama de estado transacional



Fonte: adaptada de Silberschatz (2010).

Controle transacional com o comando `COMMIT`

Para Date (2012), quando uma transação se completa, é considerada CONFIRMADA (*committed*), com isso, automaticamente é criado um novo estado, que deve garantir a persistência, mesmo em caso de falhas.

Exemplificando

Algumas vezes, por falha do SGBD, a garantia da durabilidade não é efetivada, comprometendo assim a integridade do BD. Imagine que você efetue um depósito em sua conta corrente e, posteriormente, o sistema confirme a entrada de um novo valor (`COMMIT`), mas, por uma falha de falta de energia elétrica, quando o sistema se restabelecer, o valor depositado não esteja mais na sua conta.

Nesse caso, a durabilidade não foi garantida e, consequentemente, a integridade da transação não foi efetivada.

Por padrão, o MySQL, que é o sistema de gerenciamento de banco de dados utilizado nos exemplos, vem com o recurso `COMMIT` em modo automático, ou seja, `AUTOCOMMIT`. Nesse caso, para qualquer alteração feita no banco de dados, o SGBD armazena as atualizações no disco, sem que usuário necessite realizar um comando para fazê-

lo. Porém, é possível que as confirmações sejam determinadas pelo administrador do banco de dados. Para isso, o seguinte comando dever ser digitado no SGBD: `SET AUTOCOMMIT=0;`

Ao alterar o valor do `AUTOCOMMIT` para zero, estamos concordando que o `COMMIT` seja feito manualmente, e não mais de forma automática. No exemplo do banco de dados utilizado nesta seção, será efetuada a alteração do enfermeiro que fez o primeiro atendimento, de "Enfermeiro 1" para "Enfermeiro 2", conforme pode ser observado na Figura 4.15.

Figura 4.15 | Alteração sem `COMMIT`

```
mysql> update Medicacao
    -> set EnfermeiroCoren = 22222
    -> where Id = 1;
Query OK, 1 row affected (0.01 sec)
Linhas que combinaram: 1 - Alteradas: 1 - Avisos: 0

mysql> select * from Medicacao where Id = 1;
+----+------------+----------+-------------+------------+-----------------+
| Id | Data       | Hora     | PacienteNum | RemedioCod | EnfermeiroCoren |
+----+------------+----------+-------------+------------+-----------------+
|  1 | 2018-07-01 | 05:00:00 |        1003 |        104 |           22222 |
+----+------------+----------+-------------+------------+-----------------+
1 row in set (0.00 sec)
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Porém, não será utilizado o `COMMIT` para registrar a alteração. Ao fazer *logout* e novamente se conectar ao banco para realizar a consulta, pode ser observado na Figura 4.16 que a alteração do registro não foi efetuada.

Figura 4.16 | `SELECT` sem `COMMIT`

```
mysql> select * from Medicacao where Id = 1;
+----+------------+----------+-------------+------------+-----------------+
| Id | Data       | Hora     | PacienteNum | RemedioCod | EnfermeiroCoren |
+----+------------+----------+-------------+------------+-----------------+
|  1 | 2018-07-01 | 05:00:00 |        1003 |        104 |           11111 |
+----+------------+----------+-------------+------------+-----------------+
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Para que o `UPDATE` feito na tabela fique gravado (persistência), após a transação ser confirmada, é necessário utilizar o `COMMIT`. O comando é simples e direto:

```
COMMIT;
```

Desta forma, a integridade dos dados na tabela está garantida, fazendo com que a transação seja confirmada, conforme pode ser observado na Figura 4.17

Figura 4.17 | SELECT com a utilização do COMMIT

Id	Data	Hora	PacienteNum	RemedioCod	EnfermeiroCoren
1	2018-07-01	05:00:00	1003	104	22222

Fonte: captura de tela do MySQL, elaborada pelo autor.

Pesquise mais

É muito importante o administrador de banco de dados se assegurar de possíveis falhas ocorridas. O vídeo a seguir mostra uma forma de danificar um banco por meio da técnica SQL Injection. Você como administrador de um banco deve criar mecanismos para proteger o sistema desses ataques.

CONHECIMENTO HACKER. **Como Invadir o Banco de Dados de um Site** – SQLinjection. 29 maio 2017. Disponível em: <https://www.youtube.com/watch?v=Qwzf5riMG14>. Acesso em: 18 set. 2018.

Controle transacional com o comando ROLLBACK

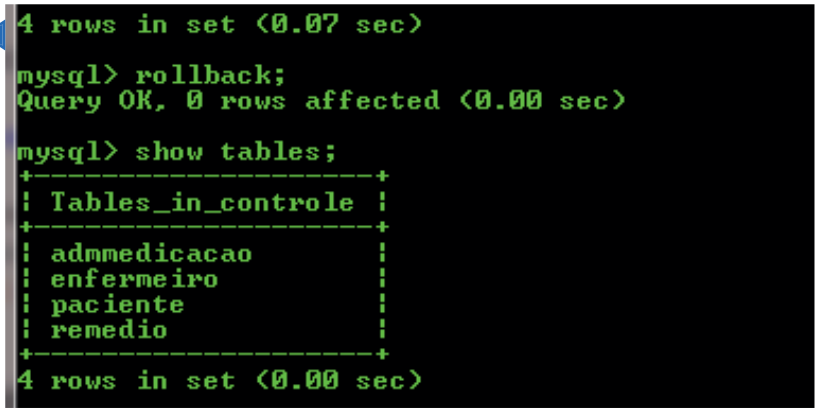
Segundo Date (2012), para que uma transação feita no banco de dados possa ser revertida, é possível utilizar o recurso de ROLLBACK para retornar ao estado anterior da transação. Porém, as instruções na linguagem de definição de dados (DDL, do inglês, *data definition language*) de criação e exclusão de banco de dados, ou ainda, as alterações, exclusões e criação de tabelas não admitem o uso do ROLLBACK, conforme pode ser observado na Figura 4.18:

Figura 4.18 | Mensagem de proibição do ROLLBACK

```
mysql> RENAME TABLE Medicacao TO AdmMedicacao;
Query OK, 0 rows affected (0.12 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+--------------------+
| Tables_in_controle |
+--------------------+
| admmedicacao       |
| enfermeiro         |
| paciente           |
| remedio            |
+--------------------+
```



Fonte: captura de tela do MySQL, elaborada pelo autor.

Embora, ao efetuar o ROLLBACK, o SGBD tenha confirmado que o comando foi executado (Query OK), nenhum registro foi alterado (0 rows affected).

Assimile

Para utilizar o comando ROLLBACK, o profissional de banco de dados deve saber que não é possível fazê-lo no grupo de comandos denominados DDL, ou seja, aqueles comandos SQL utilizados para definição dos dados.

Para que possamos retornar a determinado ponto com o ROLLBACK, o SQL permite a criação de pontos de restauração, por meio da sintaxe:

```
SAVEPOINT [nomeDoPonto];
```

Posteriormente, para utilizar esse ponto, deve ser usada a sintaxe:

```
ROLLBACK TO SAVEPOINT [nomeDoPonto];
```

Vale ressaltar que, para os controles transacionais SAVEPOINT e ROLLBACK funcionarem, deve-se alterar o valor do AUTOCOMMIT para zero, pois a grande maioria dos sistemas de gerenciamento de banco de dados vem como valor padrão do AUTOCOMMIT igual a um, isso significa que, quando ocorre uma exclusão de dados, alteração de dados, inserção de dados ou criação de SAVEPOINT, o COMMIT grava as alterações no banco e apaga o ponto criado

(SAVEPOINT). Para alterar o valor do AUTOCOMMIT, deve ser utilizada a sintaxe SQL: `SET AUTOCOMMIT=0;`

No exemplo desta seção, a tabela enfermeiro possui três registros: "Enfermeiro 1", "Enfermeiro 2" e "Enfermeiro 3". Com isso, vamos criar um ponto de salvação, por meio do comando:

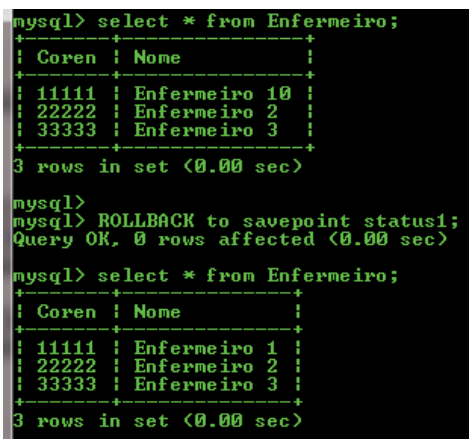
```
SET AUTOCOMMIT=0;
  SAVEPOINT status1;
```

Posteriormente, o nome do "Enfermeiro 1" foi alterado para "Enfermeiro 10". Para retornar o nome do enfermeiro, o ponto de salvação deve ser retornado por meio do comando:

```
ROLLBACK to savepoint status1;
```

Tais mecanismos podem ser observados na Figura 4.19.

Figura 4.19 | Mensagem de Proibição do ROLLBACK



Fonte: captura de tela do MySQL, elaborada pelo autor.

Desta forma, podemos garantir que as transações sejam revertidas para um estado anterior.

Reflita

O recurso ROLLBACK permite que, ao efetuarmos um UPDATE em uma tabela, os dados sejam retornados para um estado anterior. Podemos utilizar o mesmo recurso quando utilizamos o INSERT ou DELETE?

Sem medo de errar

Trabalhando na prefeitura, você é o responsável por todo o desenvolvimento dos bancos de dados de uma cidade litorânea para o gerenciamento dos passeios de escuna. Na fase anterior, foi desenvolvido um BD, porém, após um colaborador acidentalmente renomear todos os registros com o mesmo nome, impedindo que os passeios fossem realizados, as modificações preventivas foram realizadas:

- Para fazer a alteração do `COMMIT` para que as alterações não sejam gravadas automaticamente, você deverá utilizar a sintaxe:

  ```
  SET AUTOCOMMIT=0;
  ```

- Já, para criar um ponto de restauração no banco, deverá usar a sintaxe:

  ```
  SAVEPOINT point1;
  ```

- Para fins de teste, a sintaxe a seguir visará gerar o mesmo erro cometido pelo colaborador:

  ```
  UPDATE destino
  SET Nome = "Pequena ilha do Mar";
  ```

As alterações propostas podem ser observadas na saída representada na Figura 4.20.

Figura 4.20 | Erro para testar o ponto de salvação

```
mysql> select * from destino;
+----+--------------------+
| Id | Nome               |
+----+--------------------+
|  1 | Pequena ilha do Mar |
|  2 | Pequena ilha do Mar |
|  3 | Pequena ilha do Mar |
|  4 | Pequena ilha do Mar |
|  5 | Pequena ilha do Mar |
|  6 | Pequena ilha do Mar |
|  7 | Pequena ilha do Mar |
|  8 | Pequena ilha do Mar |
+----+--------------------+
8 rows in set (0.00 sec)
```

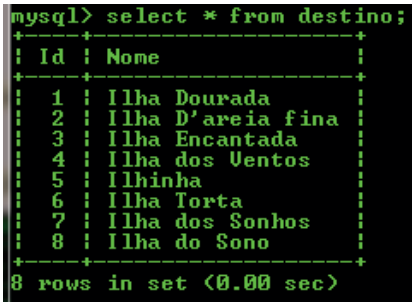
Fonte: captura de tela do MySQL, elaborada pelo autor.

- Para utilizar o ponto de restauração criado e testá-lo, deverá ser utilizada a sintaxe SQL:

  ```
  ROLLBACK TO SAVEPOINT point1;
  ```

Para verificar se os nomes foram restaurados, os registros poderão ser observados na saída representada na Figura 4.21.

Figura 4.21 | Restauração dos registros na tabela destino



Fonte: captura de tela do MySQL, elaborada pelo autor.

- Para gravar as alterações feitas, deve ser utilizado o comando:

  ```
  commit;
  ```

- Crie um novo ponto de restauração, pois os mesmos serão apagados pela gravação das alterações:

  ```
  SAVEPOINT point2;
  ```

Pronto! A partir de agora, mudanças inesperadas (e indesejadas) não mais provocarão erros e perdas de dados.

Avançando na prática

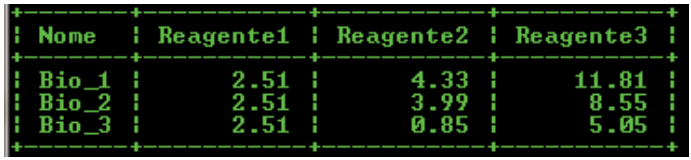
Análise de eficácia na mistura de agentes biológicos

Descrição da situação-problema

Você trabalha na área de TI de uma empresa agroquímica e foi alocado para auxiliar um pesquisador que irá realizar diversos testes envolvendo reações para obtenção de defensivos agrícolas.

O pesquisador está trabalhando com três reagentes, para aplicá-los como defensivo agrícola. Durante os trabalhos, ele solicitou que você desenvolvesse um banco de dados para registro da quantidade em gramas dos reagentes químicos em cada uma das misturas, conforme demonstrado na Figura 4.22.

Figura 4.22 | Valores químicos dos reagentes



Nome	Reagente1	Reagente2	Reagente3
Bio_1	2.51	4.33	11.81
Bio_2	2.51	3.99	8.55
Bio_3	2.51	0.85	5.05

Fonte: captura de tela do MySQL, elaborada pelo autor.

O pesquisador quer testar a mistura do Reagente1 com o Reagente2 (matematicamente, os dois reagentes devem ser multiplicados), porém, a nova mistura nunca pode ser superior ao Reagente3, senão o agente biológico não será eficaz. Para isso, você deverá desenvolver a função e o `SELECT` para comprovação dos testes, como demonstrados no Quadro 4.5.

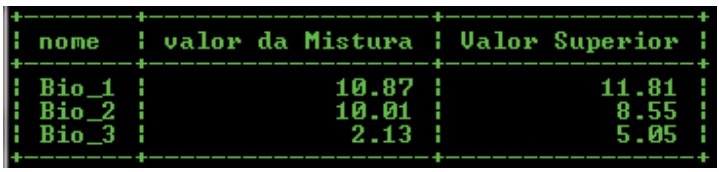
Quadro 4.5 | Script para análise

```
1  CREATE FUNCTION fn_eng (x DECIMAL (4,2), y DECIMAL (4,2))
2  RETURNS DECIMAL (4,2
3  RETURN x * y;
4
5  SELECT nome, fn_eng (Reagente1, Reagente2) AS "valor da
   Mistura", Reagente3 AS "Valor Superior"
6  FROM Analise;
```

Fonte: elaborado pelo autor.

Isso irá gerar a saída apresentada na Figura 4.23.

Figura 4.23 | `SELECT` dos experimentos 1



nome	valor da Mistura	Valor Superior
Bio_1	10.87	11.81
Bio_2	10.01	8.55
Bio_3	2.13	5.05

Fonte: captura de tela do MySQL, elaborada pelo autor.

O problema é que o segundo experimento, "Bio_2", tem o valor da mistura maior que o valor superior. Dessa forma, o pesquisador solicitou a troca do Reagente1 para 1.10, com a condição que posteriormente seja retornado no mesmo ponto de quando o banco foi criado. Como proceder para fazer esta alteração?

Resolução da situação-problema

Para isso, você deverá realizar as seguintes tarefas:

1º passo: zerar o `AUTCOMMIT`:

```
SET AUTOCOMMIT=0;
```

2º passo: criar um ponto de restauração:

```
savepoint pequisa1;
```

3º passo: alterar o valor do Reagente1:

```
update Analise set Reagente1 = 1.10 where
Reagente1;
```

4º passo: Gerar os resultados dos experimentos por meio do `SELECT` fornecido no Quadro 4.3 (linhas 5 e 6), gerando os resultados presentes na Figura 4.24.

Figura 4.24 | `SELECT` dos experimentos 2

nome	valor da Mistura	Valor Superior
Bio_1	10.87	11.81
Bio_2	10.01	8.55
Bio_3	2.13	5.05

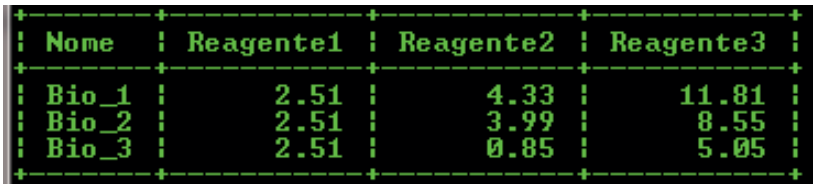
Fonte: captura de tela do MySQL, elaborada pelo autor.

5º Passo: retornar o banco ao ponto criado no primeiro passo:

```
ROLLBACK to savepoint pequisa1;
```

Assim, os valores do Reagente1 voltarão ao original, conforme pode ser observado na Figura 4.25.

Figura 4.25 | Valores químicos dos reagentes original



Nome	Reagente1	Reagente2	Reagente3
Bio_1	2.51	4.33	11.81
Bio_2	2.51	3.99	8.55
Bio_3	2.51	0.85	5.05

Fonte: captura de tela do MySQL, elaborada pelo autor.

Pronto! Agora as reações são realizadas dentro dos parâmetros solicitados pelo pesquisador.

Faça valer a pena

1. Segundo o dicionário Houaiss, integridade é:
1. Estado ou característica daquilo que está inteiro, que não sofreu qualquer diminuição;
2. Característica ou estado daquilo que se apresenta ileso, intato, que não foi atingido ou agredido.

(HOUAISS, A.; VILLAR, M. S. **Dicionário Houaiss da língua portuguesa.** 1. ed. Rio de Janeiro: Objetiva, 2009)

Do ponto de vista do banco de dados, o administrador deve garantir que os dados contidos nas tabelas sejam confiáveis. Esse recurso pode ser utilizado tanto pela aplicação, por meio de uma linguagem de programação de alto nível, como por uma linguagem de programação de banco de dados, como o SQL.
Com isso, algumas situações em que se efetue uma transação como `UPDATE`, `INSERT` ou `DELETE`, são realmente efetivadas após a sua execução em um banco de dados, de que se deseja posteriormente extrair algumas informações.

A fim de se garantir a integridade nos bancos de dados, são utilizadas as propriedades das transações nos bancos de dados, conhecidas como ACID, que corresponde a:
I. Atomicidade;
II. Controle;
III. Isolamento;
IV. Dinamismo.
Assinale a alternativa correta:

a) Somente acrônimos I e II estão corretos.
b) Somente acrônimos I e III estão corretos.

c) Somente acrônimos I, II e III estão corretos.
d) Somente acrônimos I, II e IV estão corretos.
e) Somente acrônimos III e IV estão corretos.

2. A linguagem SQL possui muitos recursos, que permitem ao administrador de banco de dados efetuar diversas operações, divididas nas categorias/comandos:

- DDL (Data Definition Language): `CREATE`, `DROP`, `ALTER`, `TRUNCATE`, `COMMENT` e `RENAME`.
- DML (Data Manipulation Language): `SELECT`, `INSERT`, `UPDATE` e `DELETE`.
- TCL (Transaction Control Language): `COMMIT`, `ROLLBACK` e `SAVEPOINT`.
- DCL (Data ControlLanguage): `GRANT` e `REVOKE`.

Para que possamos utilizar os comandos de forma incremental nas aplicações, é necessário que as sintaxes SQL de cada uma das técnicas sejam implementadas corretamente. Caso uma transação não consiga executar os comandos por completo, os recursos disponíveis na categoria TCL irão fornecer subsídios para que as ações parcialmente executadas no banco de dados sejam revertidas em um momento anterior.

Observe o texto a seguir:
Ao ser completada, uma transação é confirmada por meio do recurso SQL denominado _____ e, com isso, é criado um novo estado. No modo default do MySQL, o `COMMIT` vem configurado no modo automático, ou seja _____. É possível alterar esse estado por meio do comando `SET AUTOCOMMIT=0` e, posteriormente, voltar ao padrão, com a sintaxe _____.

Assinale a alternativa que completa as lacunas corretamente.

a) ROLLBACK – FASTOCOMMIT - SET AUTOCOMMIT=0.
b) COMMIT – INTEROCOMMIT - SET AUTOCOMMIT=ON.
c) ROLLBACK – AUTOCOMMIT - SET AUTOCOMMIT=1.
d) COMMIT – AUTOCOMMIT - SET AUTOCOMMIT=1.
e) COMMIT – INTEROCOMMIT - SET AUTOCOMMIT=ON.

3. Segundo Silberschatz (2010), os bancos de dados surgiram para informatização de dados comerciais, por volta de 1960. Tal ferramenta computacional pode ser utilizada de duas formas:

- Web: o banco de dados é instalado em um servidor de internet e, por meio de uma aplicação, é possível efetuar as operações disponíveis no sistema.
- LocalHost: o banco de dados é instalado em um servidor dentro de uma rede local, onde é possível efetuar as operações disponíveis ou autorizadas pelo administrador.

As linguagens mais comuns utilizadas pelos desenvolvedores estão representadas na Figura abaixo.

Figura | Linguagens de Programação

Fonte:<https://pixabay.com/pt/programa%C3%A7%C3%A3o-idiomas-%C3%ADcone-cole%C3%A7%C3%A3o-898961/>. Acesso em: 18 set. 2018.

A maioria das linguagens web também podem ser utilizadas em LocalHost, possibilitando diversos recursos. Para tanto, a linguagem SQL deve permitir funções agregadas, seleções encadeadas, funções, procedimentos, controle de transação, entre diversas outras operações em banco de dados.

Observe os comandos disponíveis no Quadro abaixo.

Quadro | Script

```
1  SET AUTOCOMMIT=0;
2  createtable q(
```

```
3          Col_1 int,
4          Col_2 varchar(3),
5          Col_3 decimal(3,1)
6          );
7   insert q values(1, “ABC”, 41.5),
8   (10, “DEF”, 15.4),
9   (100, “GHI”, 54.1);
10  savepoint q1;
11  delete from q where Col_1 = 100;
12  commit;
```

Fonte: elaborado pelo autor.

Assinale a alternativa correta para a utilização dos comandos a seguir:

```
ROLLBACK to savepoint q1;
select * from q;
```

a) O primeiro comando retorna do `savepoint` a mensagem “Query OK”; o segundo comando retorna apenas três registros.
b) O primeiro comando retorna que o `savepoint` não existe; o segundo comando retorna apenas três registros.
c) O primeiro comando retorna o `savepoint` “Query OK”; o segundo comando retorna apenas dois registros.
d) O primeiro comando gera um erro de sintaxe; o segundo comando retorna um erro de sintaxe.
e) O primeiro comando retorna que o `savepoint` não existe; o segundo comando retorna apenas dois registros.

Seção 4.3

Procedimentos e funções

Diálogo aberto

Caro aluno, você já deve ter percebido que em alguns sites existe um campo para inserir códigos promocionais (*vouchers*), normalmente disponíveis no carrinho de compras. Ao digitar um código válido, o banco de dados executa algumas funções, podendo ou não estar dentro de um procedimento armazenado, e recalcula o valor da sua compra. Essas rotinas só são possíveis por que existem recursos da linguagem de programação SQL que permitem automatizar algumas execuções dentro das tabelas de um banco de dados.

Você trabalha na prefeitura de uma cidade litorânea que tem no setor turístico uma importante renda, sendo você o colaborador responsável pelos bancos de dados. Seu projeto atual envolve o controle de passeios de barcos para as ilhas próximas à cidade. Recentemente, devido a um melhor controle dos passeios para levar segurança aos usuários, foi instalado no píer um ponto de venda de passagens para os passeios. Com isso, foi necessário desenvolver uma tabela para registrar as vendas com a estrutura apresentada na Figura 4.26:

Figura 4.26 | Descrição da tabela Vendas

```
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| Numero    | int(11) | NO   | PRI | NULL    | auto_increment |
| DestinoId | int(11) | NO   | MUL | NULL    |                |
| Embarque  | date    | NO   |     | NULL    |                |
| Qtd       | int(11) | NO   |     | NULL    |                |
+-----------+---------+------+-----+---------+----------------+
```

Fonte: captura de tela do MySQL, elaborada pelo autor.

Você poderá verificar o código que gera a Figura 4.26 acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u4/s3/bancodedadosexemplo_1.pdf> ou pelo QR Code.

Em seguida, foi adicionado o campo "Valor" na tabela "Passeio", por meio do comando:

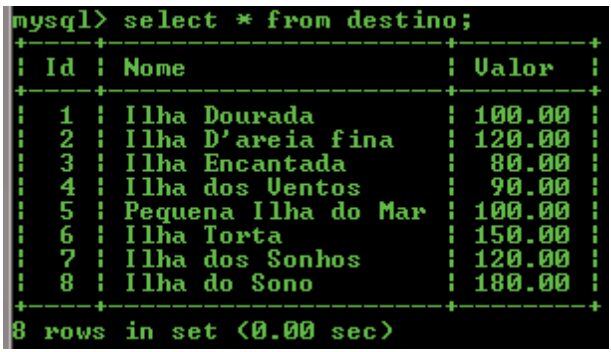
```
alter table destino add column Valor decimal(5,2);
```

Isso exigiu que todos os registros que não continham o valor do passeio fossem deletados. Para tanto, foi utilizado o comando:

```
truncate destino;
```

Finalmente, foi possível inserir os passeios e os seus respectivos valores, conforme pode ser observado na Figura 4.27:

Figura 4.27 | Seleção de dados na tabela Passeio



Fonte: captura de tela do MySQL, elaborada pelo autor.

Você poderá verificar o código que gera a figura 4.30 acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u4/s3/bancodedadosexemplo_2.pdf> ou pelo QR Code.

Como a falta de turistas na baixa temporada impacta diretamente o emprego e a renda dos moradores locais, pensando em tornar a cidade mais atrativa, foi decidido que os passeios nessa época do ano deveriam receber um desconto de 30%. O gerente de T.I. da prefeitura pediu para que você desenvolvesse alguma solução no banco de dados para que o operador possa digitar o número da venda já efetuada e seja retornado o valor que deverá ser pago, devido à promoção dos passeios de escuna na baixa temporada, que oferece 30% de desconto.

Ao resolver o problema do banco de dados para o caixa de passeios no píer, as técnicas como `FUNCTIONS` e `PROCEDURES` terão que ser aplicadas, proporcionando a compreensão da aplicabilidade de uma automatização de execuções nas tabelas de um banco de dados por meio das sintaxes SQL.

Nesta seção, iremos discutir as técnicas `FUNCTIONS` e `PROCEDURES`, que irão permitir criar rotinas de automatização dentro do banco de dados. Todos estão contando com o seu empenho e a sua qualidade técnica para uma solução inteligente no banco de dados dos passeios de escuna no píer da cidade. Pronto para esse último desafio? Então, mãos à obra!

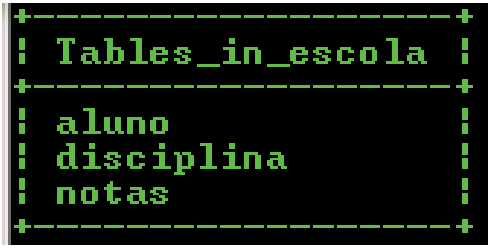
Não pode faltar

Caro aluno, no final do semestre, quando saem as médias finais, você já deve ter reparado que o sistema mostra algumas informações, como a média do primeiro bimestre e a média do segundo bimestre, que foram inseridas pelo professor. Nas últimas colunas, são exibidos a média final e o status, ou seja, se você está aprovado ou se necessita fazer o exame para ser aprovado. A coluna em que ocorre o cálculo da média final requer uma função (`FUNCTION`) ou, ainda, um procedimento armazenado (`PROCEDURE`) para que seja possível efetuar os cálculos. Logicamente, no momento de conferência dos resultados, essas execuções não são perceptíveis ao aluno.

A linguagem de programação de banco de dados SQL possui recursos específicos, que podem automatizar algumas execuções, permitindo que o tempo de algumas consultas, o resultado dos cálculos e outros processos possíveis dentro de um banco de dados sejam efetuados em menor tempo e com menor consumo de memória/processamento do servidor em que o sistema de gerenciamento de banco de dados (SGBD) está instalado.

Para exemplificar os conceitos e aplicações envolvidos nessa seção de estudos, foi desenvolvido um banco de dados para guardar as informações, e a Figura 4.28 demonstra as tabelas desenvolvidas para esse fim:

Figura 4.28 | Tabelas do banco de dados do exemplo



Fonte: captura de tela do MySQL, elaborada pelo autor.

Você poderá verificar o código que gera a figura 4.28 acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u4/s3/passeio.pdf> ou pelo QR Code.

Posteriormente, os registros demonstrados na Figura 4.29 foram inseridos nas tabelas.

Figura 4.29 | *Inserts* nas Tabelas do banco de dados do exemplo

Fonte: captura de tela do MySQL, elaborada pelo autor.

Você poderá verificar o código que gera a figura 4.29 acessando o link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-em-banco-de-dados/u4/s3/vendas.pdf> ou pelo QR Code.

FUNÇÕES (`FUNCTION`)

Segundo Silberschatz (2010), as funções são definidas na linguagem SQL a partir da versão SQL:2003. Essa técnica possibilita realizar cálculos aritméticos complexos utilizando os valores das colunas existentes em um banco de dados. Basicamente, o intuito ao se utilizar uma `FUNCTION` é retornar tabelas como resultado, conhecidas como funções de tabela.

Exemplificando

Diversos sistemas de banco de dados aceitam tanto as funções, quanto os procedimentos armazenados. Veja os Exemplos das Figuras 4.30 e 4.31:

Figura 4.30 | Função no PostgreSQL:

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

Fonte: PostgreSQL (2016, [s.p.])

Figura 4.31 | Procedimento armazenado em SQL Server (Microsoft):

```
-- Transact-SQL Syntax for Stored Procedures in SQL Server and Azure SQL Database

CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
```

Fonte: Microsoft (2017, [s.p.])

A sintaxe SQL para criar uma função é definida como `FUNCTION` (DATE, 2012), e para ser estruturada em um banco de dados, utiliza-se a estrutura SQL representada a seguir:

```
CREATE FUNCTION nome_da_funcao (x tipo, y
tipo)
RETURNS tipo
RETURN (função);
```

Em que:

- `nome_da_função:` pode ser escolhido pelo desenvolvedor. Uma boa prática é nomeá-las com o prefixo `fn`, por exemplo: `fn_teste`.
- `(x tipo, y tipo)`: são declaradas duas variáveis (`x` e `y`), e os seus respectivos tipos. Por exemplo: `(w int, z decimal(6,2)).`
- `RETURNS tipo`: determina que tipo de dado será retornado após a execução da função. Por exemplo: `RETURNS decimal(6,2)`.
- `RETURN (função)`: é o trecho da expressão em que são definidas as expressões aritméticas, determinadas em `(x tipo, y tipo)`. Por exemplo: `RETURN (x + 2) – (y + 1)`.

Para a compreensão da técnica, vamos desenvolver uma função no banco de dados do exemplo para o cálculo da média final, em que: Média Final = (NotaP1 * 0,4) + (NotaP2 * 0,6). Para isso foi desenvolvia a sintaxe SQL:

```
CREATE FUNCTION fn_media(x DECIMAL(3,1), y
DECIMAL(3,1))
RETURNS DECIMAL(3,1)
RETURN (x * 0.4) + (y * 0.6);
```

A sintaxe SQL para utilizar uma função desenvolvida em uma tabela deve ser estruturada como demonstrado a seguir:

```
SELECT nome_da_funcao (parâmetro x, parâmetro
y)
FROM nome_da_tabela
WHERE nome_da_coluna (condição);
```

Em que:

- `(parâmetro x, parâmetro y)`: devem ser colocadas as colunas utilizadas na função.

Dessa forma, para utilizarmos a `FUNCTION` desenvolvida no exemplo (`fn_media`), com que se deseja selecionar o nome do aluno, a disciplina, e a média final dos alunos de exame (com notas entre 4,1 e 6,9 inclusive), deve ser utilizada a seguinte sintaxe SQL:

```
SELECT Aluno.Nome, disciplina.Nome AS "Disiciplina",
   fn_media(NotaP1, NotaP2) AS "Média Final"
FROM Notas INNER JOIN Aluno
ON Notas.AlunoRA = Aluno.RA
INNER JOIN Disciplina
ON Notas.DisciplinaId = Disciplina.Id
WHERE fn_media(NotaP1, NotaP2) >= 4.0
   AND fn_media(NotaP1, NotaP2) <= 6.9;
```

Que gera a saída representada na Figura 4.32.

Figura 4.32 | Exemplo de saída utilizando `FUNCTION`

Nome	Disiciplina	Média Final
Aluno_A	Banco de dados	6.1
Aluno_B	Banco de dados	6.1
Aluno_D	Banco de dados	4.1
Aluno_F	Banco de dados	6.9
Aluno_A	Programação estruturada	6.9
Aluno_C	Programação estruturada	4.7
Aluno_E	Programação estruturada	6.9
Aluno_F	Programação estruturada	5.7
Aluno_A	Redes decomputadores	6.7
Aluno_B	Redes decomputadores	5.9
Aluno_C	Redes decomputadores	4.9
Aluno_D	Redes decomputadores	5.0
Aluno_E	Redes decomputadores	5.5
Aluno_A	LFA	5.9
Aluno_C	LFA	6.7
Aluno_D	LFA	6.6

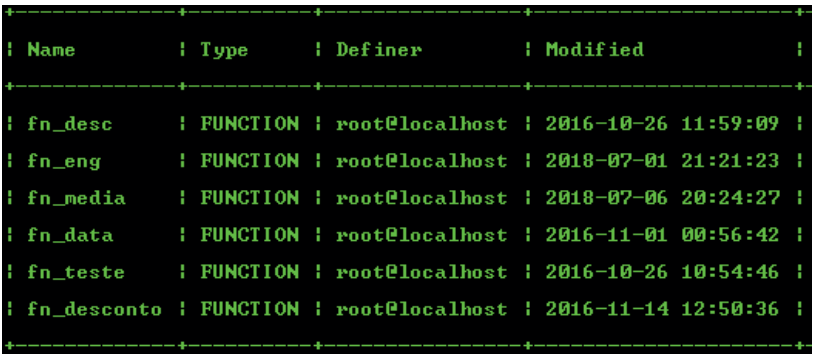
Fonte: captura de tela do MySQL, elaborada pelo autor.

Para exibir todas as funções desenvolvidas, deve ser utilizada a seguinte sintaxe SQL:

```
SHOW FUNCTION STATUS;
```

O retorno desse comando demonstra todas as funções criadas dentro do sistema de gerenciamento do banco de dados, conforme pode ser observado na Figura 4.33.

Figura 4.33 | Exemplo dos status das `FUNCTION`



Name	Type	Definer	Modified
fn_desc	FUNCTION	root@localhost	2016-10-26 11:59:09
fn_eng	FUNCTION	root@localhost	2018-07-01 21:21:23
fn_media	FUNCTION	root@localhost	2018-07-06 20:24:27
fn_data	FUNCTION	root@localhost	2016-11-01 00:56:42
fn_teste	FUNCTION	root@localhost	2016-10-26 10:54:46
fn_desconto	FUNCTION	root@localhost	2016-11-14 12:50:36

Fonte: captura de tela do MySQL, elaborada pelo autor.

O terceiro registro demonstra a `FUNCTION` desenvolvida no exemplo. Para exibir a estrutura de uma função, deve ser utilizada a seguinte sintaxe SQL:

```
SHOW CREATE FUNCTION nome_da_funcao;
```

Para demonstração, no exemplo foi desenvolvido o comando SQL a seguir:

```
SHOW CREATE FUNCTION fn_Media;
```

Que gera a saída demonstrada na Figura 4.34.

Figura 4.34 | Exemplo de estrutura da `FUNCTION`

fn_media	NO_ENGINE_SUBSTITUTION
return (x * 0.4) + (y * 0.6)	cp850

Fonte: captura de tela do MySQL, elaborada pelo autor.

Para excluir uma função, deve ser utilizada a seguinte sintaxe SQL:

```
DROP FUNCTION nome_da_funcao;
```

Esse comando não será demonstrado, pois a função será utilizada nos exemplos de procedimentos armazenados, discutidos a seguir.

PROCEDIMENTO ARMAZENADO (PROCEDURE)

Segundo Silberschatz (2010), os procedimentos armazenados foram definidos na versão SQL:1999, para que fosse obtida capacidade procedural nos bancos de dados, porém, não com a intenção de substituir técnicas que também estão disponíveis nas linguagens de programação, como Java, C++ ou C#.

Esse recurso deve permitir armazenar procedimentos como seleção de dados, exclusão de registros, alteração de dados, entre outras funções disponíveis na linguagem de programação de banco de dados SQL.

Assimile

No SGBD MySQL, os procedimentos armazenados (`PROCEDURES`) são inseridos em uma tabela chamada ROUTINES, no banco de dados chamado INFORMATION_SCHEMA, disponível no dicionário de dados.

A sintaxe SQL para se criar um procedimento armazenado é definida por meio da palavra-chave `PROCEDURE` (DATE, 2012). Para ser estruturada em um banco de dados, deve ser utilizado o comando SQL representado a seguir:

```
CREATE PROCEDURE nome_do_procedure (var_nome
tipo)
   Declarações.
```

Em que:

- `NOME_DO_PROCEDURE`: identifica o nome do procedimento. Como boa prática utilize o prefixo `proc`. Por exemplo: `proc_teste`.
- `(var_nome tipo)`: deve ser criada uma variável. Uma boa prática é utilizar o prefixo `var` e, em seguida, o tipo dessa variável. Por exemplo: `var_teste int`.
- `Declarações`: podem ser utilizadas as seleções de dados.

Para a compreensão da técnica, vamos desenvolver um procedimento armazenado para calcular a média geral de todos os alunos que estão de exame nas disciplinas:

1- Banco de Dados

2- Programação Estruturada

3- Redes de computadores

4- LFA

Para isso, foi desenvolvida a sintaxe SQL:

```
CREATE PROCEDURE proc_MediaExame (var_
DisciplinaId int)
SELECT AVG(fn_media(NotaP1, NotaP2)) AS
"Média Exame"
FROM Notas
WHERE DisciplinaId = var_DisciplinaId
  AND (fn_media(NotaP1, NotaP2) >= 4.0
  AND fn_media(NotaP1, NotaP2) <= 6.9);
```

Em que a parte de sintaxe SQL do procedimento do exemplo:

- `(var_DisciplinaIdint)`: cria uma variável que faz referência à coluna Id da tabela Disciplina, e é do tipo inteiro. Essa variável é utilizada na chamada do procedimento armazenado.
- `SELECT AVG(fn_media(NotaP1, NotaP2))`: efetua a seleção da média geral da função criada com as médias de cada um dos alunos.
- `WHERE DisiciplinId = var_DisciplinaId`: impõe uma condição que o valor disponível no campo "`DisciplinaId`" na tabela "Notas" deve ser igual ao encontrado na variável "`var_DisciplinaId`".
- `(fn_media(NotaP1, NotaP2) >= 4.0 AND fn_media(NotaP1, NotaP2) <= 6.9)=` é a condição de seleção dos alunos que estão de exame, com notas entre 4,1 e 6,9 (inclusive).

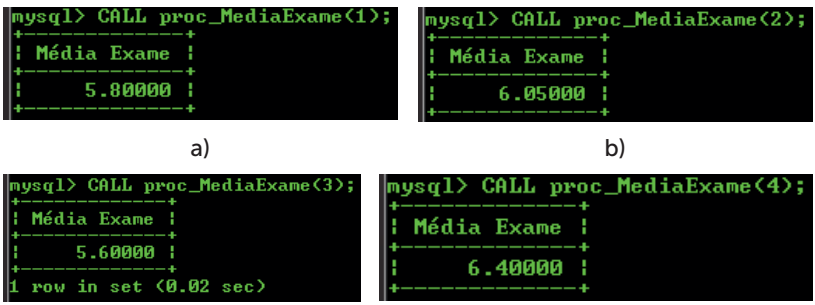
Para utilizar um procedimento armazenado, a seguinte sintaxe SQL deve ser utilizada:

```
CALL nome_do_procedure (var_nome);
```

Observe que, na sintaxe, o trecho do comando `(var_nome)` faz referência ao exemplo criado, em que se é definida a variável `(var_DisciplinaIdint)`. Ao definirmos o valor dessa variável, o banco vai restringir a busca, comparando com o valor encontrado na condição, como em `WHERE DisiciplinId = var_DisciplinaId`.

Para compreender como os procedimentos são utilizados, observe a Figura 4.35.

Figura 4.35 | Média Geral dos alunos de exame em: a) banco de dados; b) programação estruturada; c) rede de computadores; e d) linguagem formal e autômatos (LFA)



Fonte: capturas de tela do MySQL, elaboradas pelo autor.

Observe que, ao chamar o `PROCEDURE`, apenas o valor da variável `VAR_DISCIPLINAID` foi alterado em todos os exemplos.

Reflita

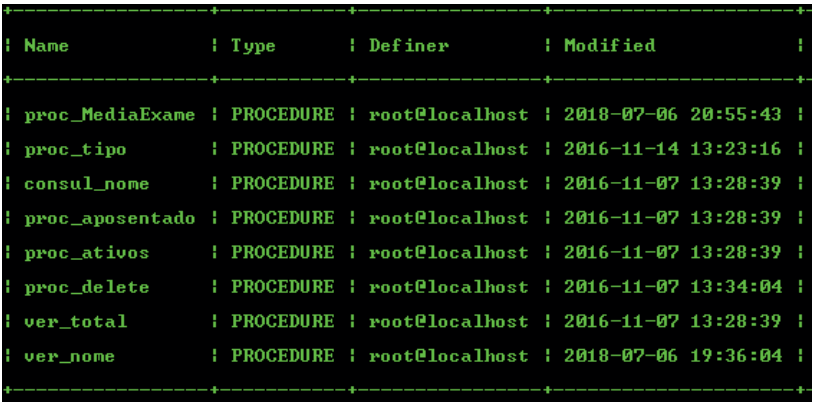
Os procedimentos armazenados são ferramentas que permitem automatizar algumas execuções nos bancos de dados. Além da seleção de dados (`SELECT`), quais outros procedimentos podem ser armazenados?

Para exibir todos os procedimentos desenvolvidos que estejam armazenados, deve ser utilizada a seguinte sintaxe SQL:

```
SHOW PROCEDURE STATUS;
```

O retorno desse comando demonstra todos os procedimentos armazenados criados dentro do SGBD, conforme pode ser observado na Figura 4.36.

Figura 4.36 | Exemplo dos *status* das `PROCEDURE`



Name	Type	Definer	Modified
proc_MediaExame	PROCEDURE	root@localhost	2018-07-06 20:55:43
proc_tipo	PROCEDURE	root@localhost	2016-11-14 13:23:16
consul_nome	PROCEDURE	root@localhost	2016-11-07 13:28:39
proc_aposentado	PROCEDURE	root@localhost	2016-11-07 13:28:39
proc_ativos	PROCEDURE	root@localhost	2016-11-07 13:28:39
proc_delete	PROCEDURE	root@localhost	2016-11-07 13:34:04
ver_total	PROCEDURE	root@localhost	2016-11-07 13:28:39
ver_nome	PROCEDURE	root@localhost	2018-07-06 19:36:04

Fonte: captura de tela do MySQL, elaborada pelo autor.

O primeiro registro demonstra a `PROCEDURE` desenvolvida no exemplo. Para excluir um procedimento armazenado, deve ser utilizada a seguinte sintaxe SQL:

```
DROP PROCEDURE nome_do_procedure;
```

Pesquise mais

O PostgreSQL é um sistema de banco de dados objeto-relacional, de código-fonte aberto, desenvolvido em 1986 como parte do projeto POSTGRES da Universidade da Califórnia em Berkeley e têm mais de 30 anos de desenvolvimento ativo na plataforma central (POSTGRESQL, 2018). A linguagem utilizada para o desenvolvimento é muito próxima à linguagem SQL "pura". O artigo do site DevMedia demonstra a sintaxe para se desenvolver e utilizar um procedimento armazenado.

DIONISIO, E. J. Trabalhando com Stored Procedures no PostgreSQL. **DevMedia**, 18 set. 2015. Disponível em: <https://www.devmedia.com.br/trabalhando-com-stored-procedures-no-postgresql/33354>. Acesso em: 19 set. 2018.

Sem medo de errar

Você é o responsável pelos desenvolvimentos nos bancos de dados da prefeitura de uma cidade litorânea e, atualmente, deve desenvolver uma solução no banco de dados que efetua a venda dos ingressos para os passeios de escuna, em que o operador deve digitar o número da venda e o programa deve retornar o valor com 30% de desconto.

Essa necessidade de implementar essa funcionalidade surgiu em uma reunião com o prefeito e os donos de escuna que, na baixa temporada, viram o número de turistas cair, então, a solução encontrada foi fazer uma promoção nessa época do ano. Para isso, primeiramente foi necessário desenvolver uma função por meio da sintaxe SQL:

```
CREATE FUNCTION fn_desc(x DECIMAL(5,2), y
INT)
RETURNS DECIMAL(5,2)
RETURN ((x * y)*0.7);
```

Em que:

- `(x DECIMAL(5,2), y INT)`: recebe o valor do passeio na tabela destino e a quantidade de ingressos disponível na tabela vendas.
- `RETURNS DECIMAL(5,2)`: retorna um valor com o desconto de 30%, no formato de moeda, conforme exemplo: 255,00.
- `RETURN ((x * y)*0.7);`: efetua o cálculo aritmético da função chamada `fn_desc`, nesse caso: `((ValorDoDestino * QuantidadeDeIngressos)* Desconto)`.

Posteriormente, para que possamos utilizar essa função, será necessário inserir somente o número da venda e retornar o valor do passeio com o desconto de 30%. Dessa forma, é necessário desenvolver um procedimento armazenado, sendo desenvolvido o comando SQL:

```
    CREATE PROCEDURE proc_desc (VAR_VendasNumero
    INT)
    SELECT (fn_desc(destino.valor, Vendas.Qtd))
AS "Valor com desconto", destino.Nome AS "Destino",
vendas.Qtd AS "Passagens", vendas.Embarque
    FROM Vendas INNER JOIN destino
    ON Vendas.DestinoId = destino.Id
    WHERE Numero = var_VendasNumero;
```

Em que:

- `(var_VendasNumero INT)`: é a variável que vai ser chamada no procedimento armazenado por meio do `CALL`.
- `(fn_desc(destino.valor, Vendas.Qtd))`: utiliza a função desenvolvida, tomando como parâmetros os campos "valor", da tabela destino, e "quantidade" (Qtd), da tabela Vendas.

Para fins de testes, foram inseridos três registros na tabela Vendas, conforme pode ser observado na Figura 4.37:

Figura 4.37 | *Inserts* na tabela Vendas

```
+---------+------------+-------------+------+
| Numero  | DestinoId  | Embarque    | Qtd  |
+---------+------------+-------------+------+
|      1  |          1 | 2018-02-03  |    3 |
|      2  |          7 | 2018-02-03  |    2 |
|      3  |          5 | 2018-02-03  |    1 |
+---------+------------+-------------+------+
3 rows in set (0.00 sec)
```

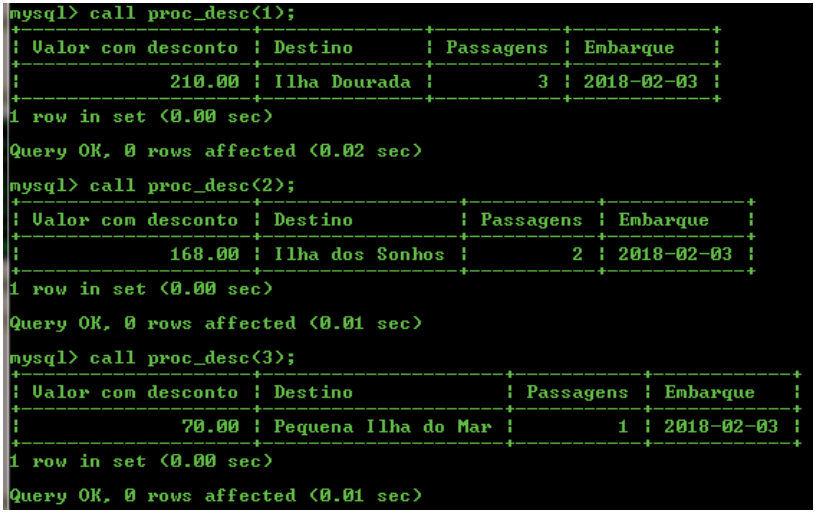
Fonte: captura de tela do MySQL, elaborada pelo autor.

Assim, é possível testar a função e o procedimento desenvolvido por meio da sintaxe SQL descrito a seguir:

```
    CALLproc_desc(1);
    CALLproc_desc(2);
    CALLproc_desc(3);
```

Gerando a saída demonstrada na Figura 4.38.

Figura 4.38 | Teste do desconto nas vendas dos passeios de escuna



Fonte: captura de tela do MySQL, elaborada pelo autor.

Assim, podemos visualizar o valor dos ingressos com o desconto, o destino do passeio, o número de passagens por compra e a data de embarque do passeio.

Avançando na prática

Criando mecanismos de cálculo de desconto

Descrição da situação-problema

Você trabalha em uma empresa de desenvolvimento de sistemas, e é responsável pelos bancos de dados dos projetos. Muito tempo atrás, o dono de uma vinícola solicitou um sistema para gerenciar a venda dos vinhos produzidos e, na época, foi desenvolvido o banco de dados com as seguintes tabelas e campos:

- Tabela Vinho (CodBar, Nome, Preço)
- Tabela Cliente (Numero, Nome, Endereço)
- Tabela Vendas (Id, ClienteNumero, VinhoCodBar)

Agora, essa vinícola quer realizar uma promoção de final de ano, em que o cliente recebe um desconto de 10% em compras com pagamento à vista. Para isso, eles solicitaram que sua empresa desenvolva um mecanismo no banco de dados em que o CodBar do vinho seja inserido e o valor do vinho seja retornado ao cliente já com 10%, se o cliente escolher o pagamento à vista. Você foi encarregado de realizar esta tarefa.

Resolução da situação-problema

Para resolver esse problema, inicialmente você deverá desenvolver uma função para que sejam calculados os descontos, como por meio da sintaxe SQL representada a seguir:

```
CREATE FUNCTION fn_desconto (x DECIMAL(5,2),
y FLOAT)
RETURNS DECIMAL(5,2)
RETURN (x * y);
```

Para que seja possível dar entrada no vinho pelo código de barra e seja concedido o devido desconto, será necessário desenvolver um procedimento armazenado. Isso poderá ser feito por meio da sintaxe SQL:

```
CREATE PROCEDURE proc_desconto (VAR_
VinhoCodBar INT)
SELECT (fn_desconto(Preco, 0.90)) AS "Valor
com desconto", Nome AS "Vinho"
FROM Vinho
WHERE CodBar = var_VinhoCodBar;
```

Para utilizar o recurso, utilize a sintaxe SQL:

```
CALL proc_desconto(19999);
```

Que irá gerar a saída representada na Figura 4.39.

Figura 4.39 | Desconto promocional Vinho_DDD

Fonte: captura de tela do MySQL, elaborada pelo autor.

Agora, para saber o valor do vinho com o desconto, bastará substituir o código de barras (variável CodBar) por um dos vinhos cadastrados no banco de dados. Você deverá obter uma situação como a descrita na Figura 4.40 para o vinho de código de barra 15555.

Figura 4.40 | Segundo exemplo para Desconto promocional

```
mysql> call proc_desconto(15555);
+---------------------+-----------+
| Valor com desconto  | Vinho     |
+---------------------+-----------+
|              162.00 | Vinho_AAA |
+---------------------+-----------+
```

Fonte: captura de tela do MySql, elaborada pelo autor.

Com isso, após desenvolver uma função (`fn_desconto`) para que ocorressem os cálculos matemáticos e incidissem os descontos, foi criado um procedimento armazenado (`proc_desconto`), que possibilita entrar com o código do vinho, retornando o seu respectivo valor com o desconto. Dessa forma, foi possível automatizar uma rotina no banco de dados.

Faça valer a pena

1. As funções e os procedimentos armazenados são ferramentas computacionais relacionadas aos bancos de dados, muito utilizados para recalcular alguns valores, conforme pode ser observado na Figura abaixo.

Figura | Campo para inserção de *voucher* de desconto em site

Fonte: <posanhanguera.com.br>. Acesso em: 7 jul. 2018.

Nesse exemplo, o site recalcula valor das parcelas da mensalidade de um curso de pós-graduação, concedendo ao aluno, no momento da inscrição, um desconto de 35% nas parcelas.

Observe as afirmativas a seguir:

I. Uma função em banco de dados utiliza expressões aritméticas para efetuar os cálculos, com base nos dados disponíveis nas tabelas.

II. A chamada de uma função é dada pela expressão SQL: `CALL nome_da_função(tipo);`
III. O procedimento armazenado permite que apenas seleções de dados sejam armazenadas em seu interior.

Com relação às afirmativas do texto acima, assinale a alternativa correta.

a) Somente a alternativa I está correta.
b) Somente as alternativas I e II estão corretas.
c) Somente a alternativa II está correta.
d) Somente a alternativa III está correta.
e) Somente as alternativas I, II e III estão corretas.

2. O desenvolvimento de sistemas computacionais é baseado na matemática, sendo que em algumas áreas essa relação é mais evidente. Uma das grandes referências da história que remete diretamente ao uso de matemática e computação foi Alan Turing, que desenvolveu aplicações computacionais para decifrar mensagens enviadas no período de guerra. Os conhecimentos matemáticos são de vital importância para os desenvolvimentos computacionais, inclusive para a programação em banco de dados, que utiliza conceitos da teoria dos conjuntos.

Observe a sintaxe SQL a seguir:

```
CREATE TABLE teste(
       a int,
       b int,
       c int
);
INSERT teste VALUES(4, 2, 3),
(2, 4, 3),
(3, 10, 5);
CREATE FUNCTION fn_teste (x INT, y INT, z INT)
RETURNS FLOAT
RETURN ((x * y)/z);
SELECT fn_teste(b, c, a) AS “Resultado”
FROM teste;
```

Assinale a alternativa que representa corretamente os resultados das saídas geradas pelo código acima.

a) ≅ 2,66 – ≅ 2,66 – 6.
b) 12 – 6 – 3.
c) 4 – 12 – 6.
d) 1.5 – 6 – ≅16,66.
e) 5 – 15 – 2.

3. Armazenar alguns procedimentos dentro do banco de dados os torna ferramentas muito úteis, que podem proporcionar:

- Códigos SQL mais simples para executar uma tarefa, por meio de um CALL;
- Menor taxa de processamento, pois s execução já está pré-configurada;
- Fazer seleção, inserção, alteração e exclusão de dados em uma ou mais tabelas.

A sintaxe SQL deve ser:

```
CREATE PROCEDURE nome_do_procedure (var_nome tipo)
declarações.
```

Já para utilizar um procedimento armazenado, a seguinte sintaxe SQL deve ser:

```
CALL nome_do_procedure (var_nome);
```

Observe o código SQL a seguir:

```
CREATE TABLE exemplo(
   a INT,
   b VARCHAR(3)
   );
   CREATE PROCEDURE proc_exemplo1(IN var_a INT, IN var_b
   VARCHAR(3))
   INSERT exemplo VALUES (var_a, var_b);
   CALL proc_exemplo1(1, "AAA");
   CALL proc_exemplo1(2, "BBB");
   CALL proc_exemplo1(3, "CCC");

   CREATE PROCEDURE proc_exemplo2 (var_a INT)
   DELETE FROM exemplo
   WHERE a = var_a;
```

```
CALL proc_exemplo2(3);

CREATE PROCEDURE proc_exemplo3(IN var_a INT, IN var_b
VARCHAR(3))
UPDATE exemplo SET b = "ZZZ"
WHERE a = var_a AND b = var_b AND b = "BBB";
CALL proc_exemplo3(2,"BBB");
```

Dados os procedimentos demonstrados no código, assinale a alternativa que descreve corretamente as saídas para os comandos: "`SELECT b FROM exemplo WHERE a = 2`"; e "`SELECT b FROM exemplo WHERE a = 3;`".

a) BBB e AAA.
b) BBB e Empty Set.
c) AAA e Empty Set.
d) BBB e CCC.
e) ZZZ e Empty Set.

Referências

ALVES, W. P. **Banco de dados**. São Paulo: Érica, 2014.

DATE, C. J. **Introdução à Sistema de Banco de Dados**. 8a ed. São Paulo: Elsevier, 2012. 921 p.

MANZANO, J. A. N. G. **Linguagem C**: acompanhada de uma xícara de café. São Paulo: Érica, 2015.

MICROSOFT. **CREATE PROCEDURE (Transact-SQL).** 5 set. 2017. Disponível em: <https://docs.microsoft.com/pt-br/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-2017>. Acesso em: 19 set. 2018.

_____. **Funções do Excel (por categoria)**. [S.d.]. Disponível em: <https://support.office.com/pt-br/article/fun%C3%A7%C3%B5es-do-excel-por-categoria-5f91f4e9-7b42-46d2-9bd1-63f26a86c0eb>. Acesso em: 15 jul. 2018.

_____. **Introdução aos bancos de dados**. [S.d.]. Disponível em: <https://support.office.com/pt-br/article/v%C3%ADdeo-introdu%C3%A7%C3%A3o-aos-bancos-de-dados-457013e7-f75d-48a9-bc8a-4b816436a5a0?ui=pt-BR&rs=pt-BR&ad=BR>. Acesso em: 30 jul. 2018.

_____. **PROCV (Função PROCV)**. [S.d.]. Disponível em: <https://support.office.com/pt-br/article/procv-fun%C3%A7%C3%A3o-procv-0bbc8083-26fe-4963-8ab8-93a18ad188a1>. Acesso em: 15 jul. 2018.

_____. **Treinamento em vídeo do Word para Windows**. [S.d.]. 2018. Disponível em: <https://support.office.com/pt-br/article/treinamento-em-v%C3%ADdeo-do-word-para-windows-7bcd85e6-2c3d-4c3c-a2a5-5ed8847eae73?ui=pt-BR&rs=pt-BR&ad=BR>. Acesso em: 1 jul. 2018.

_____. **Usar Minigráficos para mostrar tendências de dados**. [S.d.]. Disponível em: <https://support.office.com/pt-br/article/v%C3%ADdeo-usar-minigr%C3%A1ficos-para-mostrar-tend%C3%AAncias-de-dados-8d2399ed-748e-4fb5-95c9-eed8177f116d?ui=pt-BR&rs=pt-BR&ad=BR>. Acesso em: 30 jul. 2018.

POSTGRESQL. Create Function. In: **PostgreSQL 9.1.24 Documentation.** 2016. Disponível em: <https://www.postgresql.org/docs/9.1/static/sql-createfunction.html>. Acesso em: 19 set. 2018.

_____. **WHAT IS POSTGRESQL?** [S.d.]. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 19 set. 2018.

SILBERSCHATZ, A.; KORTH, H.; SUNDARSHAN, S. **Sistema de Banco de Dados**. 5. ed. Rio de Janeiro: Elsevier, 2010. 781 p.

SILVA, M. G. **Informática –** Terminologia - Microsoft Windows 8, Internet, Segurança, Microsoft Word 2013, Microsoft Excel 2013, Micro- soft PowerPoint 2013, Microsoft Access 2013. 1. ed. São Paulo: Érica, 2014.

ISBN 978-85-522-1165-5