

Banco de Dados

Vanderson José Ildefonso Silva



INSTITUTO FEDERAL
ESPÍRITO SANTO

Colatina - ES
2011

© Instituto Federal do Espírito Santo

Este Caderno foi elaborado em parceria entre o Instituto Federal do Espírito Santo e a Universidade Federal de Santa Catarina para o Sistema Escola Técnica Aberta do Brasil – e-Tec Brasil.

Equipe de Elaboração

Instituto Federal do Espírito Santo – IFES

Coordenação do Curso

Allan Francisco Forzza Amaral/IFES

Professores-autores

Vanderson José Ildefonso Silva/IFES

Comissão de Acompanhamento e Validação

Universidade Federal de Santa Catarina – UFSC

Coordenação Institucional

Araci Hack Catapan/UFSC

Coordenação do Projeto

Sílvia Modesto Nassar/UFSC

Coordenação de Design Instrucional

Beatriz Helena Dal Molin/UNIOESTE e UFSC

Coordenação de Design Gráfico

André Rodrigues/UFSC

Design Instrucional

Gustavo Pereira Mateus/UFSC

Web Master

Rafaela Lunardi Comarella/UFSC

Web Design

Beatriz Wilges/UFSC

Mônica Nassar Machuca/UFSC

Diagramação

André Rodrigues/UFSC

Bárbara Zardo/UFSC

Juliana Tonietto/UFSC

Nathalia Takeuchi/UFSC

Revisão

Júlio César Ramos/UFSC

Projeto Gráfico

e-Tec/MEC

S586b

Silva, Vanderson José Ildefonso

**Banco de dados : Curso Técnico de Informática / Vanderson José Ildefonso Silva. – Colatina: Ifes/CEAD, 2011.
176 p. : il.**

ISBN: 978-85-62934-73-5

1. Banco de dados. 2. Informática. I. Instituto Federal do Espírito Santo. II. Título.

CDD: 005.74

Apresentação e-Tec Brasil

Prezado estudante,

Bem-vindo ao e-Tec Brasil!

Você faz parte de uma rede nacional pública de ensino, a Escola Técnica Aberta do Brasil, instituída pelo Decreto nº 6.301, de 12 de dezembro 2007, com o objetivo de democratizar o acesso ao ensino técnico público, na modalidade a distância. O programa é resultado de uma parceria entre o Ministério da Educação, por meio das Secretarias de Educação a Distância (SEED) e de Educação Profissional e Tecnológica (SETEC), as universidades e escolas técnicas estaduais e federais.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

O e-Tec Brasil leva os cursos técnicos a locais distantes das instituições de ensino e para a periferia das grandes cidades, incentivando os jovens a concluir o ensino médio. Os cursos são ofertados pelas instituições públicas de ensino e o atendimento ao estudante é realizado em escolas-polo integrantes das redes públicas municipais e estaduais.

O Ministério da Educação, as instituições públicas de ensino técnico, seus servidores técnicos e professores acreditam que uma educação profissional qualificada – integradora do ensino médio e educação técnica, – é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação
Janeiro de 2010

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.



Atenção: indica pontos de maior relevância no texto.



Saiba mais: oferece novas informações que enriquecem o assunto ou “curiosidades” e notícias recentes relacionadas ao tema estudado.



Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.



Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.



Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor	9
Apresentação da disciplina	11
Projeto instrucional	13
Aula 1 – Conceitos de Bancos de Dados	17
1.1 Definição	17
1.2 Objetivos de Banco de Dados	21
1.3 Usuários de Bancos de Dados	22
1.4 Modelos de Bancos de Dados	23
Aula 2 – Modelagem de Dados:	
Modelo Entidade-Relacionamento	35
2.1 Cardinalidade de relacionamentos	37
2.2 Exemplos do modelo entidade-relacionamento	42
2.3 Generalização/especialização de entidades	47
2.4 Entidades fracas	48
Aula 3 – Modelagem de Dados:	
Modelo Relacional Normalizado	51
3.1 Chave primária	51
3.3 Exemplos de diagrama do MRN	57
3.4 Formas normais	60
Aula 4 – Criação de Banco de Dados	69
4.1 Linguagem SQL	69
4.2 MySQL	71
4.3 Como criar um Banco de Dados	72
4.4 Criação de tabelas	74
4.5 Inserção de registro em tabela	88
4.6 Alteração de registro em tabela	94
4.7 Exclusão de registro em tabela	98

Aula 5 – Consultas SQL	103
5.1 Consultas básicas	103
5.2 Consultas com cláusula ORDER BY	105
5.3 Consultas com cláusula WHERE	109
5.4 Consultas com funções e agrupamento	112
5.5 Consultas com mais de uma tabela	119
5.6 Outras consultas	124
Aula 6 – Segurança da Informação	127
6.1 Preparando o terreno	127
6.2 Alterando a senha do administrador do BD	128
6.3 Criando novas contas de usuário e definindo seus privilégios de segurança	130
6.4 Criando visões	136
Aula 7 – Procedimentos Armazenados e Gatilhos	141
7.1 Importância de rotinas armazenadas	141
7.2 Procedimentos armazenados	142
Aula 8 – Criação de índices	159
8.1 Índices	159
8.2 Criando um índice durante a criação da tabela	161
8.3 Criando um índice para tabela preexistente	163
Aula 9 – Transações em Banco de Dados	165
9.1 Conceito de transação	165
9.2 Transações no MySQL	170
Referências	174
Currículo do professor-autor	175

Palavra do professor-autor

Olá caro estudante!

A tecnologia de Bancos de Dados é quase onipresente em sistemas de informação. Pode ser encontrada em *sítes* de comércio eletrônico e sistemas de georreferenciamento. Muitas vezes utilizamos um Banco de Dados indiretamente e sem perceber. Por exemplo, sempre que fazemos uso da máquina de busca do Google ou mesmo do Google Maps, acessamos um Banco de Dados.

Banco de dados é uma fascinante área de estudo. É impossível não se deixar contagiar por sua elegância e potencial tecnológico. Porém, devo lembrá-lo que cada novo conhecimento adquirido encontra-se intimamente associado ao abordado anteriormente. Portanto, não acumule conceitos e técnicas sem a devida cota de estudo diário. O fraco entendimento de um conteúdo pode resultar em grave dificuldade de assimilação do seguinte. Organize o seu estudo e mantenha-se atualizado com o desenvolvimento das aulas.

Assim como nas disciplinas de programação de computadores, em Banco de Dados somente é possível “aprender fazendo”. Logo, é importante que se esforce na resolução dos exercícios propostos e até se aventure em voos solo. Estou certo de que inúmeras ideias e possibilidades aparecerão em sua mente à medida que for se aprofundando nos conceitos e técnicas apresentados. Vale a pena dar vazão à sua criatividade. Talvez trabalhe em alguma atividade que possa se beneficiar com a criação de um Banco de Dados adaptado à sua realidade, ou talvez à de uma empresa de um conhecido. Apenas ler este material sem praticar os conteúdos não trará os resultados que eu e você esperamos.

Bons estudos!

Vanderson José Ildefonso Silva

Apresentação da disciplina

Nesta disciplina você será apresentado aos conceitos básicos de Bancos de Dados e aprenderá a empregá-los como uma importante ferramenta para a manutenção de dados em sistemas de informação (persistência de dados). Com a crescente utilização da Tecnologia da Informação pelas organizações e pessoas físicas, os Bancos de Dados converteram-se em uma importante tecnologia.

Embora já seja uma tecnologia madura, com décadas de existência, ainda demonstra grande valor como suporte a tecnologias mais recentes, como de *Data Warehouse* (Armazém de Dados) e *Data Mining* (Mineração de Dados), e é imprescindível para grande parcela dos profissionais da área de informática.

Este material tem como objetivo orientá-lo no estudo da disciplina Banco de Dados, por meio de dicas e sugestões, com destaque para os aspectos mais importantes. Aqui você encontrará conceitos com os quais trabalharemos ao longo do curso, o que não dispensa, é claro, a utilização de outros livros também usados para a confecção deste trabalho, que trazem diversos exemplos adicionais e aprofundamento maior em vários aspectos. É importante esclarecer que esses outros livros foram consultados para complementar alguns conceitos, a fim de facilitar o seu entendimento, e constam nas referências deste material.

Bons Estudos!

Prof. Vanderson José Ildefonso Silva

Projeto instrucional

Disciplina: Banco de Dados (carga horária: 90 horas).

Ementa: Arquitetura de Sistemas Gerenciadores de Banco de Dados. Modelo de Dados. Integridade Referencial. Linguagens de Definição, Manipulação e Controle de Dados. Segurança e Integridade. Controle de Transações.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. Conceitos de Banco de Dados	<ul style="list-style-type: none">- Definir conceitos básicos de Bancos de Dados.- Identificar problemas relativos à tecnologia da informação que induziram ao desenvolvimento de Sistemas Gerenciadores de Banco de Dados (SGBDs).- Enumerar os principais objetivos da tecnologia de Banco de Dados.- Classificar os diferentes tipos de usuários de Banco de Dados.	Este caderno e outros livros indicados nas referências.	10
2. Modelagem de Dados: MER	<ul style="list-style-type: none">- Definir entidades e seus atributos.- Estabelecer relacionamentos entre entidades.- Ajustar a cardinalidade dos relacionamentos.- Interpretar Modelos de Entidade e Relacionamento (MER).- Modelar conceitualmente sistemas de informação.	Este caderno e outros livros indicados nas referências.	10
3. Modelagem de Dados: MRN	<ul style="list-style-type: none">- Interpretar um Modelo Relacional Normalizado (MRN).- Implementar um MRN a partir de um Modelo Entidade Relacionamento (MER) preexistente.- Definir chaves primárias e estrangeiras para uma entidade.- Efetuar a normalização do modelo de dados, obedecendo às três primeiras Formas Normais (FN).	DB Designer.	10
(continua)			

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
4. Criação de um Banco de Dados	<ul style="list-style-type: none"> - Utilizar o <i>Data Definition Language</i> (DDL) da SQL. - Definir chaves primárias e chaves estrangeiras. - Estabelecer relacionamentos entre tabelas. - Alterar tabelas. - Excluir tabelas. - Inserir novos registros em tabelas. - Alterar registros. - Excluir registros de tabelas. 	SGBD MySQL.	10
5. Consultas SQL	<ul style="list-style-type: none"> - Recuperar dados armazenados em tabelas através de consultas SQL. - Definir colunas cujos dados serão visualizados na consulta. - Consultar simultaneamente informações armazenadas em várias tabelas relacionadas. - Estabelecer "filtros" que separem os registros relevantes para o contexto da consulta daqueles que não o são. - Utilizar funções especiais disponíveis. 	SGBD MySQL.	10
6. Segurança da Informação	<ul style="list-style-type: none"> - Alterar a senha do Administrador do Banco de Dados. - Criar novas contas de usuário. - Definir privilégios de segurança diferenciados para cada conta de usuário. - Criar visões diferentes para o mesmo Banco de Dados. - Utilizar as visões para definir melhor a segurança das informações armazenadas. 	SGBD MySQL.	10
(continua)			

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
7. Procedimentos Armazenados e Gatilhos	<ul style="list-style-type: none"> - Diferenciar um procedimento armazenado de um gatilho. - Identificar situações em que é justificada a utilização de procedimentos armazenados e gatilhos. - Criar procedimentos armazenados. - Executar procedimentos armazenados. - Apagar procedimentos armazenados. - Efetuar passagens de parâmetros em procedimentos armazenados. - Criar gatilhos que disparem antes ou após um evento. 	SGBD MySQL.	10
8. Criação de Índices	<ul style="list-style-type: none"> - Definir o conceito de índice no contexto da tecnologia de Banco de Dados. - Descrever a importância dos índices para os sistemas de Bancos de Dados. - Criar índices durante a criação das tabelas. - Criar novos índices alterando tabelas existentes. 	SGBD MySQL.	10
9. Transações em Banco de Dados.	<ul style="list-style-type: none"> - Conceituar transação no contexto dos sistemas de Banco de Dados. - Implementar transações no SGBD MySQL através de comandos SQL. 	SGBD MySQL.	10
(conclusão)			

Aula 1 – Conceitos de Bancos de Dados

Objetivos

Definir conceitos básicos de Bancos de Dados.

Identificar os problemas relativos à tecnologia da informação que induziram ao desenvolvimento de Sistemas Gerenciadores de Banco de Dados (SGBDs).

Enumerar os principais objetivos da tecnologia de Banco de Dados.

Classificar os diferentes tipos de usuários de Banco de Dados.

1.1 Definição

A humanidade sempre se dedicou à produção de signos. Um signo ou símbolo corresponde a qualquer coisa que represente algo para alguém. Da mesma forma que um mapa representa um dado terreno, um brasão pode representar o time de futebol. Uma canção pode fazê-lo lembrar uma fase específica de sua vida ou uma paixão. Tudo isso e muitos outros são signos. Nesse sentido, o que para você nada significa pode representar algo para outros. Portanto, todo signo é subjetivo e pode ser de difícil interpretação.

Até hoje, pesquisadores debruçam-se sobre pinturas feitas em cavernas há pelo menos 15.000 anos e debatem sobre seus significados. Apesar de representarem animais ou cenas de caçadas na pré-história, escapa-nos sua motivação. Faziam parte de algum ritual religioso, eram o registro de uma importante atividade social ou apenas a expressão da subjetividade de seu pintor?

A oralidade primária precede a invenção da escrita e se caracteriza pelo uso da palavra falada como a única forma de gerir a memória social das comunidades. A cultura narrativa valorizava as parábolas, fábulas e mitos como veículos naturais do conhecimento que se pretendia preservar de uma geração para outra. Histórias e imagens eram associadas a algum fato que se buscava memorizar.

O mito codifica, sob forma de narrativa, algumas das representações que parecem essenciais aos membros de uma sociedade. Dado o funcionamento da memória humana, e na ausência de técnicas de fixação da informação como a escrita [...], as representações que têm mais chances de sobreviver em um ambiente composto quase que unicamente por memórias humanas são aquelas que estão codificadas em narrativas dramáticas, agradáveis de serem ouvidas, trazendo uma forte carga emotiva e acompanhadas de músicas e rituais diversos. (LÉVY, 1993, p. 82).

Ao desenvolverem a escrita, as primeiras civilizações melhoraram a utilidade dos símbolos. Afinal, o surgimento do Estado trouxe consigo a inevitável cobrança de impostos. Com a escrita, a contabilidade veio a possuir meios objetivos para registrar tributos devidos e valores pagos. Desde então, o monarca passou a contar com registros precisos de seu tesouro e dos súditos devedores.

Com o capitalismo e o consequente progresso técnico, a administração das organizações tornou-se mais complexa, demandando, por exemplo, maior controle da atividade produtiva: do gerenciamento de estoques, recursos humanos e financeiros. O grande volume de informações registradas em papel dificultava consideravelmente seu gerenciamento e atualização. Então, com os primeiros computadores migraram-se essas informações para dispositivos eletrônicos.

De início, a migração foi implementada de modo pouco organizado, usando-se de sistemas de arquivos tradicionais. Cada aplicação do sistema de informações era tratada isoladamente pela equipe de desenvolvedores. Então, cada aplicação tinha seus próprios arquivos e a redundância de informações era normal. Uma aplicação para o controle da frequência dos funcionários, por exemplo, tinha seu próprio arquivo com dados dos empregados em atividade. Esse arquivo podia não ser compartilhado com a aplicação de controle das férias desses mesmos empregados.

Por isso, dados como nome, número de matrícula e departamento de trabalho podiam facilmente estar duplicados nos diferentes arquivos. Com a multiplicação de aplicações e, assim, de arquivos com redundância de dados, o risco de inconsistências de dados entre eles crescia exponencialmente. Considere, por exemplo, uma funcionária que se casou e mudou de nome. Uma eventual falha humana podia levar a uma situação em que seu nome fosse alterado apenas em alguns desses arquivos, mas não em todos eles.

Os primeiros **Bancos de Dados** surgiram no mercado como uma resposta a esses problemas. Um **Sistema de Gerenciamento de Banco de Dados** (SGBD) consiste em um conjunto de arquivos estruturados e de programas que respondem pelo acesso e manipulação de tais arquivos. Diferente dos sistemas de arquivos tradicionais, o Banco de Dados (BD) favorece o inter-relacionamento dos arquivos; portanto, pode ser definido como “uma coleção de dados inter-relacionados e um conjunto de programas para acessá-los” (KORTH; SILBERSCHATZ; SUDARSHAN, 2006, p. 1).

Simplificando, pode-se definir SGBD como um *software* desenvolvido para gerenciar grandes volumes de informações. O objetivo principal reside em superar problemas comuns aos sistemas de arquivos tradicionais. Tais problemas ou desvantagens (KORTH; SILBERSCHATZ; SUDARSHAN, 2006) são:

- (1) Redundância e inconsistência de dados.** Equipes de desenvolvedores criam diferentes aplicações/sistemas ao longo do tempo. De maneira análoga, em uma mesma organização, diferentes linguagens de programação, por exemplo, podem ser usadas no desenvolvimento de diversos sistemas de informações. Logo, dados distintos podem estar duplicados. Tal redundância conduz a altos custos de armazenamento e dificuldade de atualização das informações.
- (2) Dificuldade no acesso aos dados.** Dados espalhados em arquivos isolados não apresentam as facilidades de acesso e processamento das informações dos BD. Há pouca flexibilidade em relação a demandas que não tenham sido antecipadas quando o sistema foi projetado. Por exemplo, uma vez desenvolvido o sistema, caso haja a necessidade de gerar relatórios com os nomes de todos os empregados com idade igual ou superior a 40 anos, a ausência de uma aplicação específica para esse objetivo traz sérios inconvenientes. Isso porque quando os usuários do sistema solicitarem esse novo aplicativo, sua implementação demandará tempo e recursos dos programadores para a geração de um relatório muito específico que raramente será usado. Outra solução seria conseguir a impressão de uma listagem com todos os empregados existentes para posterior extração manual da informação desejada. Essa solução também não é satisfatória, pois o trabalho manual está sujeito a erros humanos.
- (3) Isolamento de dados.** Dados espalhados em diferentes arquivos e formatos dificultam a criação de novos aplicativos para sua recuperação.

A-Z

Banco de Dados (BD)

É um conjunto de dados integrados reunidos com o intuito de suportar o funcionamento de sistemas de informação.

Sistema Gerenciador de Banco de Dados (SGBD)

É um *software* de caráter geral para a manipulação eficiente de grandes coleções de informações estruturadas e armazenadas de uma forma consistente e integrada.

(4) Anomalias de acesso concorrente. Inúmeros sistemas de informação permitem que múltiplos usuários acessem e atualizem dados simultaneamente. A inexistência de sofisticados mecanismos de gerenciamento de atualizações concorrentes pode resultar em dados inconsistentes. Considere o exemplo de uma conta bancária conjunta com saldo de R\$ 600,00, caso dois clientes saquem dinheiro dessa mesma conta simultaneamente. Suponha que o cliente A saque R\$ 50,00 ao mesmo tempo em que o cliente B saca R\$ 100,00. Então, o programa aplicativo lê o saldo da conta de R\$ 600,00. Em seguida, o valor do saque é diminuído do saldo lido, gerando assim o saldo atualizado. Então, para o aplicativo de A, temos $(600 - 50) = 550$, enquanto para o aplicativo de B, temos $(600 - 100) = 500$. Supondo que o aplicativo de A atualize o arquivo antes do aplicativo de B, o saldo da conta registrará R\$ 500,00. Afinal, no arquivo, os dados do aplicativo de B sobreporão os dados do aplicativo de A. Repare que essa informação está errada – inconsistente – pois havia R\$ 600,00 e foram retirados R\$ 150,00. Portanto, o saldo deveria ser de R\$ 450,00.



A tecnologia de BD trouxe maior produtividade para o desenvolvimento de sistemas de informação. Afinal, nos sistemas tradicionais de arquivos, quando implementadas, as características inauguradas com o surgimento dos BD eram incorporadas às aplicações isoladamente. Logo, a complexidade do desenvolvimento de sistemas de informação aumentava consideravelmente, resultando em:

- (1) Prazos maiores para a conclusão dos sistemas de informação e, consequentemente, elevação dos custos envolvidos no desenvolvimento.
- (2) Obrigação de as equipes desenvolvedoras (analistas e programadores) dedicarem mais tempo a programas que não tratavam diretamente do sistema de informação em questão, mas sim de funcionalidades que forneceriam suporte a esse mesmo sistema.
- (3) Ocorrência de problemas não previstos, favorecidos pela ausência de padronização das funcionalidades de suporte.

(5) Problemas de segurança. O acesso a determinados dados deve ser restrito para alguns usuários do sistema de informações. Os dados relativos ao contracheque dos empregados não podem ser disponibilizados a todos os usuários indistintamente, mas apenas aos usuários responsáveis pela folha de pagamento. Entretanto, quando programas aplicativos são instalados de maneira arbitrária, ou sem acompanhamento e controle necessários, não há como assegurar tais restrições de segurança.

(6) Problemas de integridade. Restrições de consistência são impostas aos dados armazenados. Elas são normalmente regras de negócio. O saldo de uma conta bancária, por exemplo, não deve cair abaixo de um valor predeterminado. Restrições de consistência como essa, nos sistemas de arquivos tradicionais, são incorporadas aos códigos dos programas aplicativos. Um problema grave ocorre quando a adição de novas restrições implica a necessidade de alteração de vários programas aplicativos. Sempre há o risco de esquecimento de algum desses programas, ameaçando a integridade dos dados.

Essas seis dificuldades levaram ao desenvolvimento dos SGBD. Ao longo do curso veremos algumas das estratégias criativamente engendradas para a solução das desvantagens dos sistemas de arquivos tradicionais.

Os sistemas de BD atualmente existentes no mercado aliviam as equipes desenvolvedoras de preocupações do que não esteja relacionado aos seus objetivos, permitindo assim a codificação de sistemas de informação mais robustos e confiáveis.

Crie pelo menos dois possíveis exemplos para as desvantagens/problemas trazidos pelo uso de sistemas tradicionais de arquivos.



1.2 Objetivos de Banco de Dados

Sistemas Gerenciadores de Bancos de Dados (SGBD) têm o objetivo de prover mecanismos adequados ao armazenamento e acesso seguro e eficiente de dados em Sistemas de Informação (SI). Mais detalhadamente, os BD têm por objetivo:

- fornecer interfaces amigáveis e padronizadas para o armazenamento e acesso aos dados, poupando os usuários dos detalhes da implementação interna;
- assegurar a privacidade dos dados através de medidas de segurança como: atribuição de permissões de acesso; criação de visões; e fornecimento de senhas de acesso, evitando o acesso a dados por pessoas não autorizadas;
- administrar acessos concorrentes aos dados, permitindo que diferentes usuários compartilhem simultaneamente a mesma coleção de dados;
- prover mecanismos para a recuperação de dados em caso de eventuais paradas e falhas do sistema, as quais podem ocorrer por causa de erros de *software*, interrupção no suprimento de energia, defeito de *hardware*, queda na comunicação com o servidor, etc. Qualquer falha pode resultar na perda de dados processados pelo SGBD no momento da parada. A perda desses dados pode levar o BD a uma condição de inconsistência que, se não evitada, torna a tecnologia pouco confiável. Portanto, durante uma venda, o estoque deve ser atualizado e, se uma falha ocorre após a venda, mas antes da atualização do estoque, o BD ficará inconsistente. Afinal, a quantidade em estoque não será real para o produto em questão.

Um SGBD foi definido como “uma coleção de dados inter-relacionados e um conjunto de programas para acessá-los” (KORTH; SILBERSCHATZ; SUDARSHAN, 2006, p. 1). Entretanto, os sistemas tradicionais de arquivo que precederam a tecnologia de BD falhavam exatamente nesses dois aspectos: (I) manter dados



inter-relacionados e (II) fornecer um conjunto de programas voltados à manutenção de tais dados.

- a) Comente as vantagens que a existência de **dados inter-relacionados** traz aos sistemas de informação.
- b) Uma das principais funcionalidades de um Banco de Dados é a chamada **restrição de consistências**. Explique os riscos que a ausência dessa funcionalidade trazia aos sistemas tradicionais de arquivos.

1.3 Usuários de Bancos de Dados

Basicamente são quatro os tipos de usuários de sistemas de Bancos de Dados:

- **Usuários leigos:** profissionais de outras áreas cujos conhecimentos de informática se restringem ao básico e que interagem com o BD por meio de aplicações escritas pelos programadores de aplicações. As aplicações fazem a intermediação entre esses usuários e o BD, pois através de telas ou páginas tal usuário acessa seus dados.
- **Usuários avançados:** usuários com maior nível de independência em relação aos programadores de aplicações: interagem com os Bancos de Dados por meio de interfaces disponíveis no ambiente e escrevem consultas para relatórios com certa facilidade, sem a necessidade de um programador escrever uma nova aplicação.
- **Programadores de aplicações:** profissionais formados em computação que constroem aplicações/programas de computador com interfaces intuitivas e amigáveis (formulários e relatórios acessando o BD) para os usuários leigos.
- **Analistas de sistemas:** profissionais responsáveis por traduzir as necessidades dos usuários leigos e avançados em uma especificação racional de um sistema de informação. Definem o projeto do sistema de informação que especifica as aplicações e a estrutura do Banco de Dados e que será rigorosamente seguido pelos programadores, desenvolvendo programas que acessem o Banco de Dados;
- **Administrador de BD (Database Administrator – DBA):** usuário mais especializado responsável por administrar as bases de dados; ocupa-se com:

- (1) a atribuição de permissões de acesso adequadas a cada usuário;
- (2) a geração de cópias de segurança (*backups*) como contingência contra falhas;
- (3) o monitoramento do ambiente como forma de assegurar a disponibilidade do Banco de Dados pelo maior tempo possível;
- (4) a otimização de recursos de infraestrutura (disco, memória, processador) para assegurar o desempenho satisfatório do BD;
- (5) o suporte à equipe de desenvolvimento (analistas e programadores).

Em resumo, o **DBA** deve ser o profissional que zela pela implementação adequada do BD, assegurando um funcionamento eficiente que prime pelo desempenho, escalabilidade, flexibilidade e confiabilidade.



Algumas organizações preferem manter um profissional que acumule as funções do analista de sistemas e do programador de aplicações. Isso lhes permite reduzir custos trabalhistas e evitar "ruidos de comunicação" que comprometam a eficiência dos sistemas de informações. Afinal, sempre há a possibilidade de que o analista de sistemas não seja plenamente entendido pelo programador de aplicações, gerando um sistema de informação muito diferente do que o usuário requisitara inicialmente. Assim, as aplicações teriam de ser reescritas, tornando o processo mais caro e menos eficiente.

1.4 Modelos de Bancos de Dados

Os modelos de BD definem a forma como os dados estão organizados internamente. Em ordem cronológica, são assim classificados: em redes, hierárquicos, relacionais, objeto-relacionais e orientados a objetos.

1.4.1 Modelo em rede

Um BD em rede é uma coleção de registros concatenados uns aos outros por meio de ligações. Semelhantes ao conceito de ponteiros, elas são entendidas como endereços de memória que "apontam" a localização de um registro associado.

Na Figura 1.1, há um modelo em rede com dois diferentes tipos de registros: cliente e conta. O registro de cliente apresenta três atributos: nome, cidade e sexo. Por sua vez, o registro de conta possui apenas dois atributos: número e saldo.

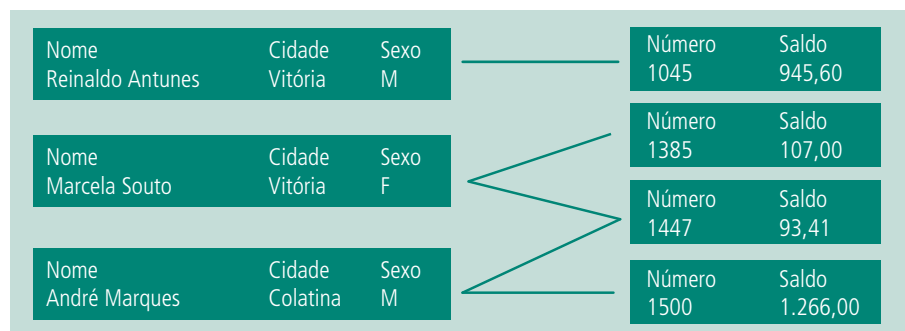


Figura 1.1: Exemplo de Banco de Dados – modelo em rede

Fonte: Elaborada pelo autor

Ligações associam registros de clientes aos de contas. Então, sabemos que o cliente Reinaldo Antunes tem uma conta de número 1045 e saldo de R\$ 945,60; e Marcela Souto e André Marques possuem uma conta conjunta de número 1447, com R\$ 93,41. Porém, Marcela tem ainda outra conta número 1358 e saldo R\$ 107,00; e André Marques também tem outra de número 1500 com R\$ 1.266,00.

Um conjunto de diferentes tipos de registros relacionados entre si por meio de um emaranhado de ligações forma uma estrutura de dados semelhante a uma rede.



Seguindo o modelo da Figura 1.1, elabore um exemplo de BD no modelo em rede envolvendo os seguintes dados: Funcionário {matrícula, nome, salário, função} e Filial {cidade, bairro, telefone}. O objetivo dessa coleção de dados é armazenar informações sobre todos os funcionários de um supermercado e de suas filiais de diversas localidades. Por meio dessa coleção deve ser possível determinar as filiais em que cada funcionário está lotado.

Esse exemplo deverá apresentar pelo menos duas filiais com um mínimo de cinco funcionários por elas distribuídos. Use sua criatividade para dar nomes aos funcionários e preencher os outros dados sobre eles e as filiais nas tabelas. Lembre-se de que um funcionário somente poderá estar lotado em uma filial.

1.4.2 Modelo hierárquico

Uma evolução do modelo em rede, os BD hierárquicos são uma coleção de registros relacionados uns aos outros por meio de ligações semelhantes a ponteiros. Porém, se diferenciam de seu antecessor na organização seus registros. Enquanto no modelo em rede os registros são distribuídos conforme a lógica de grafos arbitrários, no hierárquico eles são dispostos como uma coleção de árvores. A Figura 1.1 ficaria então traduzida como mostra a Figura 1.2 a seguir.

Árvores binárias são um dos temas tratados na disciplina referente à Programação em Estrutura de Dados.

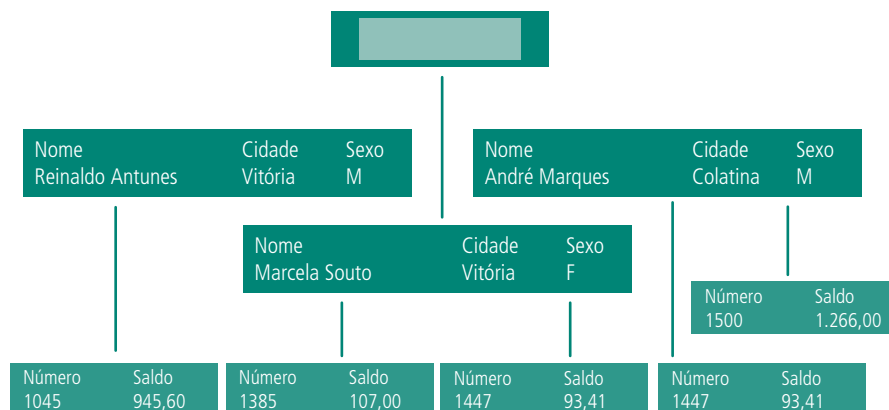


Figura 1.2: Exemplo de Banco de Dados – modelo hierárquico

Fonte: Elaborada pelo autor

Um registro isolado no topo da Figura 1.2 encontra-se associado a todos os registros de Clientes. Esse é o registro do tipo “raiz”, o ponto de partida da árvore de registros. Ele está no nível mais elevado da estrutura de dados e pode ser ligado a nenhum, um ou vários registros no nível inferior. Em nosso exemplo, os registros conectados ao registro “raiz” são sempre registros de Clientes que, por sua vez, estão interligados a registros de Contas no nível imediatamente abaixo.

Generalizando, existem camadas, níveis ou hierarquias em uma árvore. Os registros de um nível sempre se associam aos do imediatamente inferior e nunca aos do mesmo nível. No primeiro nível da árvore há sempre um único registro, a “Raiz”, que representa o ponto de partida para acessar os demais registros.

Cada nível da árvore comporta apenas um tipo de registro. No exemplo, o nível abaixo da raiz comporta registros do tipo Cliente; e o seguinte, do tipo Conta. Ainda poderiam existir outros níveis se houvesse registros de outros tipos.

Na árvore, um registro de nível K pode estar associado a, no máximo, um registro do nível imediatamente acima ($K - 1$). Contudo, um registro de nível superior pode estar associado a mais de um registro do nível imediatamente abaixo. Note que o registro da cliente Marcela Souto liga-se a dois outros registros de conta, mas a conta de número 1447 teve de ser duplicada para que a regra fosse preservada, replicando o registro para que André Marques compartilhasse essa conta com Marcela Souto, já que um registro de conta não pode se associar simultaneamente a dois de clientes.

A estrutura hierárquica do BD define o caminho de acesso a registros. A busca ao registro sempre começa pela raiz até o nível correspondente ao tipo procurado.



Converta o exemplo de BD no modelo em rede da atividade anterior em um exemplo equivalente no modelo hierárquico.

1.4.3 Modelo relacional

Na década de 1970, o modelo relacional de BD estabeleceu-se como o preferencial para aplicações comerciais. Na década seguinte já era o padrão no mercado corporativo.



É comum encontrar na literatura as denominações Pai e Filho para distinguir a relação existente entre registros de diferentes camadas da árvore. Registros de uma camada K geralmente são designados Filhos dos registros de uma camada (K – 1). Em contrapartida, os dessa camada (K – 1) são considerados Pais dos registros da camada K que a eles estiverem conectados. Portanto, em nosso exemplo da Figura 1.2, os registros de Conta são Filhos dos registros de Clientes.

No modelo relacional, os registros são organizados em tabelas e cada linha representa uma relação entre os valores armazenados em diferentes colunas. Assim, na tabela de CLIENTE (Figura 1.3), seguindo o exemplo desenvolvido até aqui, temos cada registro subdividido em três colunas: NOME, CIDADE e SEXO. Toda linha existente na tabela de CLIENTES representa um conjunto de valores inter-relacionados. Ou seja, cada linha da tabela armazena os dados de um cliente específico organizados em diferentes colunas.

CLIENTE		
NOME	CIDADE	SEXO
Reinaldo Antunes	Vitória	M
Marcela Souto	Colatina	F
André Marques	Colatina	M

Figura 1.3: Exemplo de tabela CLIENTE em um modelo relacional de Banco de Dados
Fonte: Elaborada pelo autor



Em 1970, Edgar Frank Codd, um pesquisador da IBM, propôs o modelo relacional de BD tratado aqui. Esse novo modelo trouxe uma importante contribuição ao dissociar a estrutura lógica do BD dos métodos de armazenamento físico dos dados – algo impossível para os modelos em rede e hierárquico. Ao fazê-lo, tornou os SGBD mais “amigáveis” (fáceis de utilizar).

Assim, sabemos que o cliente “Reinaldo Antunes” mora em “Vitória” e é do sexo masculino (“M”), pois esses dados estão na mesma linha da tabela. Também sabemos que os clientes “Marcela Souto” e “André Marques” residem na cidade de “Colatina” e são, respectivamente, do sexo feminino e masculino.

O modelo relacional de BD apresenta esse nome em razão da relação existente entre as colunas de uma mesma tabela, e também da possibilidade de estabelecer relacionamentos entre diferentes tabelas. Para que o exemplo de clientes e contas fique completo, precisamos relacionar registros de uma

tabela com os de outra. Caso contrário, não teríamos como saber com precisão a quais clientes cada conta pertence. Porém, as associações entre registros não são implementadas mediante “ponteiros”. Na verdade, o modelo relacional emprega a duplicação de uma ou mais colunas em uma tabela distinta daquela a que pertencem originalmente.

Ponteiros é tema da disciplina de Programação em Estrutura de Dados.



Na Figura 1.4 temos duas tabelas – FILME e GENERO – relacionadas entre si através da coluna COD_GENERO. Essa coluna é original da tabela GENERO, mas foi duplicada na tabela FILME para estabelecer uma associação lógica entre os registros das duas tabelas. Logo, sabemos que o filme “*Star Trek*” é classificado como ficção científica, pois apresenta o valor “FIC” em sua coluna COD_GENERO, e o mesmo valor na coluna correspondente da tabela GENERO está associada à descrição “Ficção Científica”.

NUMERO	TITULO	ANO	COD_GENERO
1099	El Cid	1961	DRA
1100	Star Wars	1977	FIC
1101	Star Trek	2009	FIC
1103	Transformers 2	2009	FIC

COD_GENERO	DESCRICAO
DRA	Drama
FIC	Ficção

Figura 1.4: Exemplo de relacionamento entre as tabelas FILME e GENERO

Fonte: Elaborada pelo autor

Há quem não compreenda a necessidade de duas tabelas em situações como essa. Afirmam ser uma solução complicada e perguntam: A coluna descrição não poderia simplesmente existir apenas na tabela FILME para não haver a tabela GENERO e nem as colunas duplicadas de COD_GENERO? A princípio essa parece uma solução mais simples e, portanto, melhor, mas a Figura 1.5 procura exemplificar sua vulnerabilidade.

FILME			
NUMERO	TITULO	ANO	DESCRICAO_GENERO
1099	El Cid	1961	Drama
1100	Star Wars	1977	Ficção - Científica
1101	Star Trek	2009	Ficção
1103	Transformers 2	2009	Ficção - Científica

Figura 1.5: Vulnerabilidade da fusão equivocada das tabelas FILME e GENERO

Fonte: Elaborada pelo autor

Os valores da coluna DESCRICAO_GENERO podem se repetir em diferentes registros (linhas). Na tabela da Figura 1.5 há três filmes do gênero ficção científica. Contudo, cada filme apresenta valores diferenciados para essa coluna. No filme “*Transformers 2*” não há hífen, mas em “*Star Wars*” esse caractere está presente. Já no filme “*Star Trek*”, não somente houve a omissão do hífen como também a supressão da palavra “Científica”. Algo assim é perfeitamente possível de acontecer. Usuários diferentes podem ter cadastrado cada um desses filmes, ou ainda um mesmo usuário em diferentes ocasiões. Acontece que, em um momento, esse usuário encontrava-se um tanto cansado e registrou apenas “ficção”. Mais tarde, por um erro de digitação, passou a usar o hífen.

A possibilidade de ocorrer tal situação demonstra a vulnerabilidade da solução baseada em uma única tabela. Caso uma consulta seja efetuada, o resultado pode diferir da realidade que a base de dados deveria espelhar. Suponha que um usuário submeta uma consulta ao BD usando como critério de busca a coluna DESCRICAO_GENERO com o valor “Ficção Científica”. O resultado da consulta não incluirá os registros de “*Star Trek*” e “*Star Wars*” e ele pensará que existe apenas um filme desse gênero: “*Transformers 2*”.

Ao tratarmos do conceito de integridade referencial, a solução inicial (Figura 1.5) é a mais indicada para evitar tais problemas. Ocorre que a tecnologia de BD relacionais possui recursos suficientes para assegurar que a coluna COD_GENERO da tabela FILME apresente somente valores já existentes na coluna COD_GENERO da tabela GENERO. Logo, nenhum usuário poderá cadastrar um novo filme ou alterar um velho filme com um código inexistente na tabela de origem (GENERO). O SGBD relacional simplesmente não permitirá: não se consegue usar um novo código de gênero para um Filme

sem antes criá-lo na tabela de GENERO. No máximo, um usuário descuidado poderia lançar um código errado para um filme.

Retomando o exemplo do BD para clientes e contas bancárias, a Figura 1.6 apresenta a configuração ideal para o modelo relacional. Uma terceira tabela CONTAS_CLIENTE precisou ser criada para relacionar CLIENTE e CONTA. No exemplo sobre FILME e GENERO, não foi necessário, bastou replicar a coluna COD_GENERO na tabela FILME. Porém, a duplicação da coluna ID_CLIENTE na tabela CONTA ou a da coluna NUMERO_CONTA na tabela CLIENTE não seria uma solução eficiente. Por isso, criamos uma tabela apresentando duas colunas e nenhuma delas é originária da nova tabela (CONTAS_CLIENTE). As colunas ID_CLIENTE e NUMERO_CONTA são replicadas respectivamente nas tabelas Cliente e Conta.

CLIENTE			
ID_CLIENTE	NOME	Cidade	SEXO
10	Reinaldo Antunes	Vitória	M
15	Marcela Souto	Colatina	F
37	André Marques	Colatina	M

CONTAS_CLIENTE		CONTA	
NUMERO_CONTA	ID_CLIENTE	NUMERO_CONTA	SALDO
1045	10	1045	945,60
1385	15	1385	107,00
1447	15	1447	93,41
1447	37	1500	1.266,00
1500	37		

Figura 1.6: Exemplo de Banco de Dados – modelo relacional

Fonte: Elaborada pelo autor

Caso duplicássemos ID_CLIENTE na tabela CONTA, haveria um problema ao cadastrar a conta número 1447, como mostrado na Figura 1.7. Essa conta pertence a dois clientes, é uma conta conjunta: André (ID_CLIENTE = 37) e Marcela (ID_CLIENTE = 15). Como a coluna ID_CLIENTE na tabela CONTA comporta apenas um valor por linha, teríamos de cadastrar duas linhas para a Conta de número 1447.

CONTAS_CLIENTE		
NUMERO_CONTA	SALDO	ID_CLIENTE
1045	945,60	10
1385	107,00	15
1447	93,41	15
1447	93,41	37
1500	1.266,00	37




Figura 1.7: O problema de duplicar ID_CLIENTE na tabela CONTA

Fonte: Elaborada pelo autor

Por outro lado, se optássemos por duplicar NUMERO_CONTA na tabela CLIENTE, teríamos outro problema para registrar as duas contas de “Marcela Souto” – números 1385 e 1447 (Figura 1.8), pois teríamos de cadastrar duas linhas exatamente iguais para Marcela na tabela CLIENTE, exceto pelo conteúdo da coluna duplicada de NUMERO_CONTA.

CONTAS_CLIENTE				
ID_CLIENTE	NOME	ID_CLIENTE	SEXO	NUMERO_CONTA
10	Reinaldo Antunes	Vitória	M	1045
15	Marcela Souto	Colatina	F	1385
15	Marcela Souto	Colatina	F	1447
37	André Marques	Colatina	M	1447
7	André Marques	Colatina	M	1500



Figura 1.8: O problema de duplicar NUMERO_CONTA na tabela CONTA

Fonte: Elaborada pelo autor

As consequências mais sérias para essas tentativas equivocadas de relacionamento das tabelas CLIENTE e CONTA seriam:

- (1) Uso ineficiente dos recursos de armazenamento: redundância de dados consumiria desnecessariamente o espaço disponível no disco rígido.
- (2) Maior complexidade para a manutenção das informações: a atualização do saldo em uma conta (Figura 1.7) ou a mudança de cidade para um cliente (Figura 1.8) poderia alterar mais de um registro simultaneamente.

Por mais que a nova tabela CONTAS_CLIENTE (Figura 1.6) também seja uma redundância de dados, quando comparada às outras situações (Figuras 1.7 e 1.8), ela representa uma replicação controlada e, portanto, mais eficiente.

Na Figura 1.8, a duplicação de dados abrange quatro colunas (ID_CLIENTE, NOME, CIDADE e SEXO), e na Figura 1.6, envolve duas colunas (ID_CLIENTE e NUMERO_CONTA), mas nenhuma combinação se repete na tabela CONTAS_CLIENTE.

Como a duplicação de dados é inerente ao modelo relacional – o que é inevitável – a situação representada pela Figura 1.6 mostra-se plenamente aceitável e preferível às demais situações representadas pelas Figuras 1.7 e 1.8.

A maneira de definir quais colunas e em quais tabelas as mesmas deverão ser replicadas será tratada adiante quando abordarmos conceitos como chaves primárias, chaves estrangeiras e integridade referencial. Também veremos com detalhes a linguagem de consulta e manipulação de BDs relacionais conhecida como *Structured Query Language* (SQL), padrão desde 1980.



Converta o exemplo de BD das Atividades anteriores em um exemplo equivalente no Modelo Relacional.



1.4.4 Modelo objeto-relacional

SGBDs que adotam o modelo Objeto-Relacional (OR) aproveitam a estrutura básica do modelo relacional com algumas características próprias da orientação a objetos. Porém, esse modelo híbrido não deve ser confundido com o Orientado a Objetos. Dentre as características da orientação a objetos incorporadas pelo modelo Objeto-Relacional merecem destaque: a herança de tipos e tabelas e a definição de novos tipos complexos.

O modelo Objeto-Relacional é conhecido como modelo relacional estendido. Sua linguagem de consulta foi adaptada para abranger objetos, atributos multivalorados, dados abstratos, métodos e funções como predicados de busca.

O padrão ANSI SQL-99 ou SQL-3, caracterizado como SQL orientada a objetos, trouxe inovações em relação à SQL-92. Ela é a base de vários SGBDs OR, como o Oracle 11g, o IBM DB2 Universal Database e o Informix Universal Server.

1.4.5 Modelo orientado a objetos

Os modelos em rede, hierárquico e relacional trouxeram importantes contribuições para a tecnologia de BD, principalmente os comerciais. Qualquer um desses modelos foi um avanço em relação aos sistemas tradicionais de arquivos. Dentre os três modelos acima, o relacional merece destaque, pois, desde a década de 1980, tornou-se padrão de mercado no desenvolvimento de aplicações

comerciais, como controle de estoques, contas a pagar e a receber, frente de loja, recursos humanos, etc. Contudo, a complexidade de novas demandas tecnológicas como os sistemas de informações geográficas e multimídias evidenciaram as limitações desses modelos. As novas aplicações precisavam suportar estruturas complexas de dados para objetos, assim como o armazenamento de imagens digitalizadas e textos muito longos (ELMASRI; NAVATHE, 2005).

Outro fator que pressionou o desenvolvimento de um modelo Orientado a Objeto (OO) para BD é a predominância de linguagens de Programação Orientadas a Objeto (POO). Logo, programadores de aplicações convivem com os dois paradigmas de desenvolvimento: modelo relacional para BD e orientação a objetos para os programas que trabalham com esses mesmos bancos. Tudo seria mais simples se aplicações e bancos que dão suporte à persistência de seus dados fossem igualmente orientados a objetos, não sendo mais necessário o mapeamento objeto-relacional para combinar as duas tecnologias.

Infelizmente, BDs orientados a objetos ainda não foram bem aceitos no mercado. Tal rejeição é explicada pela simplicidade e popularidade do modelo relacional junto aos profissionais de informática. Portanto, muitos desenvolvedores optam por uma solução híbrida como o modelo Objeto-Relacional ou modelo relacional estendido.



Modelo Orientado a Objetos é abordado na disciplina referente à Análise e Projeto de Sistemas.

Resumo

Nesta aula você se familiarizou com conceitos básicos de Banco de Dados. Esses conceitos serão desenvolvidos com mais detalhes ao longo do material. Vimos como as exigências modernas de um maior controle sobre os dados armazenados levaram ao surgimento de sistemas de informação; inicialmente, desenvolvidos como sistemas de arquivos tradicionais que constituíam um grande avanço em relação os mecanismos manuais de registro das informações, mas apresentavam sérias limitações que somente foram corrigidas com a tecnologia de Banco de Dados. Também abordamos os diferentes tipos de usuários para Bancos de Dados. Os sistemas gerenciadores de Bancos de Dados (SGBDs) foram implementados em diferentes modelos ao longo do tempo. Inicialmente surgiram os SGBDs de modelo em rede. Com a evolução tecnológica, sucederam-no os modelos hierárquico, relacional e o objeto-relacional. Um modelo orientado a objetos é esperado para breve, mas ainda não se estabeleceu no mercado.

Atividades de aprendizagem

1. Liste em ordem cronológica os modelos de BD.
2. Cite causas que levaram ao surgimento do modelo Orientado a Objetos de BD.
3. Por que BDs orientados a objetos não se tornaram popular no mercado? Poderíamos afirmar que a tecnologia de BD estagnou no modelo relacional incorporando poucas inovações desde o início da década de 1970?
4. As tabelas mostradas na Figura 1.9 correspondem ao modelo relacional de BD.

Funcionarios				Cargos	
MATRICULA	NOME	CPF	CARGO	CODIGO	NOME_CARGO
01	Ana	123	02	01	Programador
02	Maria	234	01	02	Topógrafo
03	José	245	03	03	Engenheiro
04	Pedro	125	01		

Figura 1.9: Exemplo de BD de uma empresa
Fonte: Elaborada pelo autor

Converta esses dados:

- a) para o modelo redes, e
 - b) para o modelo hierárquico (lembre-se de começar dos dados mais gerais para os mais específicos dentro da árvore).
5. (FCC 2006 – SEFAZ PB – Auditor Fiscal de Tributos Estaduais) Um gerenciador de Banco de Dados relacional:
- a) Identifica a relação entre seus registros a partir de ponteiros no sentido filho-pai, unicamente.
 - b) Identifica a relação entre dois ou mais registros a partir da sua justaposição.
 - c) Não contempla a definição de dados pertinentes às tabelas.
 - d) Identifica a relação entre seus registros a partir de ponteiros no sentido pai-filho, unicamente.
 - e) Deve possibilitar a identificação única de uma linha de uma tabela.

Aula 2 – Modelagem de Dados: Modelo Entidade-Relacionamento

Objetivos

- Definir entidades e seus atributos.
- Estabelecer relacionamentos entre entidades.
- Ajustar a cardinalidade dos relacionamentos.
- Interpretar Modelos de Entidade e Relacionamento (MER).
- Modelar conceitualmente sistemas de informação.

O Modelo Entidade-Relacionamento (MER) é uma técnica criada em 1976 por Peter Chen (CHEN, 1990) que expressa graficamente a estrutura lógica de um BD. A aplicação da técnica resulta em um diagrama com os seguintes elementos:

(1) Entidades: representadas graficamente por um retângulo, são “coisas” ou “objetos” armazenados em um BD. MEDICO e PACIENTE, por exemplo, são entidades em um BD projetado para suportar um sistema de agendamento de consultas (Figura 2.1). Afinal, ele é implementado para armazenar dados sobre médicos da clínica e pacientes que agendam consultas.

MEDICO

PACIENTE

Figura 2.1: Representação gráfica das entidades MEDICO e PACIENTE

Fonte: Elaborada pelo autor

(2) Atributos: são características ou propriedades relevantes de uma entidade. Por “relevantes” destacamos que nem todas as informações sobre uma entidade são pertinentes para o modelo: saber qual a cor do cabelo do paciente ou o time de futebol para o qual um médico torce em nada contribui para a marcação de uma consulta. Por outro lado, o nome do paciente, sua data de nascimento (forma de manter atualizada sua idade), um telefone para contato, o endereço e o sexo são informações relevantes para o contexto (Figura 2.2).

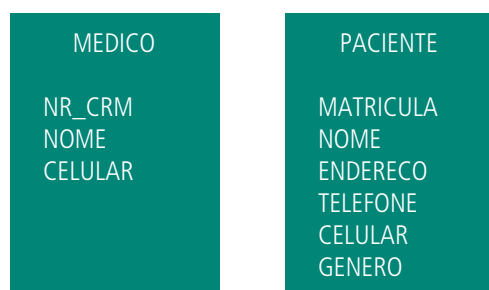


Figura 2.2: Entidades e seus respectivos atributos

Fonte: Elaborada pelo autor

(3) Relacionamentos: são vínculos ou associações lógicas entre duas ou mais entidades. Em alguns casos, um relacionamento pode ser estabelecido entre uma entidade e ela mesma (autorrelacionamento). Porém, a forma mais comum de relacionamento é entre duas entidades. As Figuras 2.3, 2.4 e 2.5 apresentam exemplos de relacionamentos. Eles são representados por uma linha com um losango sobreposto e um verbo que o identifica. No interior desse losango há geralmente uma seta indicando o sentido da leitura do verbo. Na Figura 2.3, por exemplo, lemos que “médico atende paciente”. Por sua vez, a Figura 2.4 demonstra um relacionamento entre três entidades simultaneamente (relacionamento ternário). Nesse caso não usamos o verbo nem a seta, apenas o losango na junção das linhas. A vantagem desse relacionamento é que se torna possível saber qual o plano de saúde utilizado pelo paciente em uma determinada consulta com um dado médico. A Figura 2.5 exemplifica um relacionamento entre uma entidade e ela mesma (o autorrelacionamento) para que um funcionário possa coordenar os trabalhos de outros, permitindo identificar quais funcionários são coordenados.

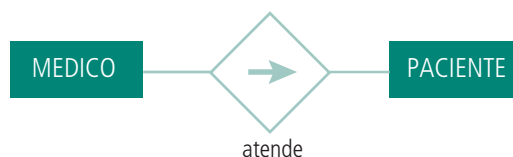


Figura 2.3: Relacionamento entre duas entidades

Fonte: Elaborada pelo autor

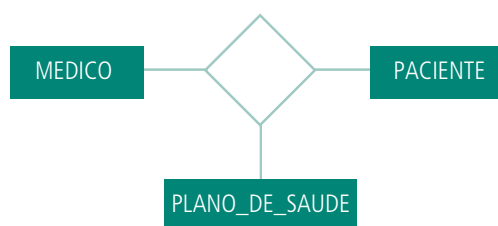


Figura 2.4: Relacionamento entre três entidades (ternário)

Fonte: Elaborada pelo autor



Figura 2.5: Relacionamento de uma entidade consigo mesma (Autorrelacionamento)

Fonte: Elaborada pelo autor

Faça um diagrama que demonstre relacionamentos coerentes entre as seguintes entidades de um restaurante: MESA, PRATO, GARCOM e COMANDA (pedidos feitos pelos ocupantes da mesa). Indique para cada entidade um conjunto de atributos adequados.



2.1 Cardinalidade de relacionamentos

A cardinalidade é uma característica de todo relacionamento no Modelo Entidade-Relacionamento (MER). Indica com quantas ocorrências de uma entidade as ocorrências de outra entidade podem se relacionar.

A Figura 2.6 mostra um autorrelacionamento sem a indicação da cardinalidade, utilizamos os chamados indicadores de papéis ("Marido" e "Esposa"). O relacionamento pode ser lido como "pessoa casa com pessoa". A indicação dos papéis apenas torna claro que, no casamento, pessoas têm o papel ou de marido ou de esposa.

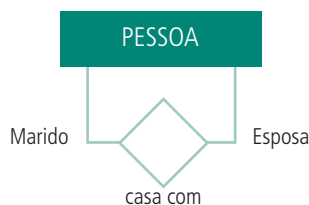


Figura 2.6: Autorrelacionamento com indicação de papéis

Fonte: Elaborada pelo autor

A ausência de cardinalidade nos impossibilita uma melhor compreensão das regras envolvidas nesse casamento entre pessoas. Em algumas sociedades é permitido que um homem se case com várias mulheres. Caso nosso Banco de Dados precise refletir esse costume, devemos impor um relacionamento de cardinalidade um-para-muitos (1:N).

O relacionamento da Figura 2.7 exemplifica as regras de uma sociedade poligâmica, na qual um marido pode ter várias esposas. Então lemos que "uma (1) pessoa (Marido) pode casar-se com muitas (N) pessoas (Esposas)".

Define, também, que “uma pessoa (Esposa) somente pode casar-se com uma pessoa (Marido)”. Portanto, os direitos não são iguais nessa sociedade e a cardinalidade define isso com clareza. Um homem pode ter várias mulheres, mas uma mulher pode ter apenas um homem como marido.

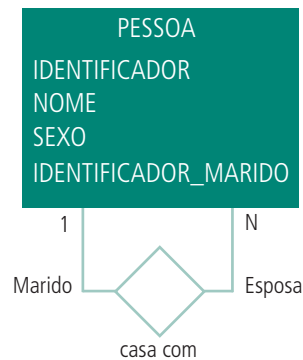


Figura 2.7: Cardinalidade um-para-muitos (1:N)

Fonte: Elaborada pelo autor

Os atributos da entidade PESSOA evidenciados na Figura 2.7 são:

- (1) IDENTIFICADOR:** código que possibilite distinguir uma pessoa de outra sem confusões.
- (2) NOME:** nome completo da pessoa. Não serve para identificar com precisão dada a possibilidade da ocorrência de homônimos (pessoas de mesmo nome. Ex.: José da Silva).
- (3) SEXO:** classificação do sexo das pessoas: “F” para feminino e “M” para masculino.
- (4) IDENTIFICADOR_MARIDO:** atributo que será utilizado apenas pelas pessoas do sexo feminino. Geralmente, sociedades poligâmicas não toleram casamentos de pessoas do mesmo sexo. Esse atributo ou terá valor nulo (para homens e mulheres solteiras) ou conterá o identificador de uma pessoa do sexo masculino (o marido).

A Figura 2.8 exemplifica uma tabela em BD relacional com a implementação do diagrama anterior, onde um homem pode ter várias esposas, mas uma mulher, só um marido.

IDENTIFICADOR	NOME	SEXO	IDENTIFICADOR_MARIDO
1	Altair Souza	M	
2	Karla Silva	F	
3	Martha Souza	F	1
4	Sueli Souza	F	1
5	Ricardo Vitali	M	
6	Rita Vitali	F	5
7	Ângela Souza	F	1

Figura 2.8: Tabela PESSOA correspondente à entidade PESSOA

Fonte: Elaborada pelo autor

Observe que, na tabela PESSOA, “Altair” tem três esposas (Martha, Sueli e Ângela). Elas possuem o número identificador dele na coluna/atributo IDENTIFICADOR_MARIDO. Por outro lado, “Karla” apresenta esse atributo sem a indicação de qualquer número (valor nulo). Significa que “Karla” é solteira. “Rita” (identificador = 6) é casada com “Ricardo” (identificador = 5).

Devemos destacar que a cardinalidade indicada não implica a obrigatoriedade de um homem ter mais de uma esposa. No exemplo, “Ricardo” tem apenas uma esposa e poderiam ainda existir homens solteiros.

Considere agora outra sociedade onde os homens poderão se casar apenas com uma mulher, mas uma mulher poderá casar-se com muitos homens (poliandria). Nosso MER sofreria uma alteração em sua cardinalidade, deixando de apresentar um relacionamento um-para-muitos (1:N) para assumir um relacionamento muitos-para-um (N:1).

A Figura 2.9 é o resultado de uma sociedade poliândrica. Curiosamente, ela quase não difere da Figura 2.7. As únicas alterações perceptíveis são a troca de posições entre “N” e “1” na indicação da cardinalidade, e a substituição do atributo IDENTIFICADOR_MARIDO pelo IDENTIFICADOR_ESPOSA. Portanto, a ordem dos fatores influi sobre o resultado final. O relacionamento agora indica que “uma pessoa (Marido) pode casar-se com apenas uma pessoa (Esposa)” e que “uma pessoa (Esposa) pode casar-se com muitas pessoas (Maridos)”.

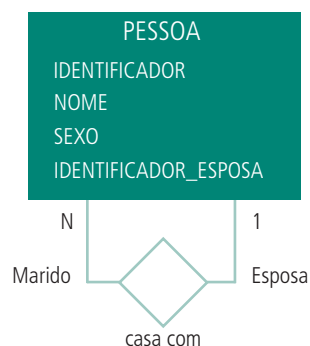


Figura 2.9: Cardinalidade muitos-para-um (N:1)

Fonte: Elaborada pelo autor

Agora consideremos que homens e mulheres negociaram uma saída mais equilibrada, tornando essa sociedade monogâmica (Figura 2.10).



Figura 2.10: Cardinalidade um-para-um (1:1)

Fonte: Elaborada pelo autor

Mais uma vez nos deparamos com pequenas mudanças: o atributo IDENTIFICADOR_CONJUGE substitui o atributo IDENTIFICADOR_ESPOSA ou IDENTIFICADOR_MARIDO; e a cardinalidade do relacionamento agora é um-para-um (1:1). Não temos mais o indicador "N" (que simboliza "muitos/vários"). Dessa forma, agora um homem pode casar-se com apenas uma mulher e uma mulher, com somente um homem por vez.

Esgotamos todas as possibilidades de cardinalidade para esse relacionamento? Certamente não. Há a possibilidade de que um homem possa casar-se com muitas mulheres e que também uma mulher possa casar-se com muitos homens. Esse seria um relacionamento de cardinalidade muitos-para-muitos (N:N). Nesse caso ocorreriam mudanças substanciais.

Na Figura 2.11, por exemplo, percebemos que um retângulo tracejado aparece sobre o losango do relacionamento em consequência direta de relacionamentos de cardinalidade muitos-para-muitos. Esse retângulo tracejado corresponde a uma relação e possui nome (Casamento) e atributos (IDENTIFICADOR_MARIDO e IDENTIFICADOR_ESPOSA).

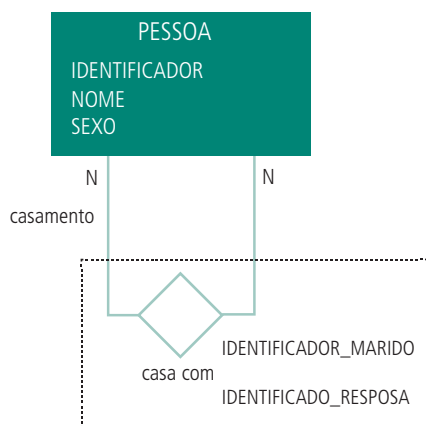


Figura 2.11: Cardinalidade muitos-para-muitos (N:N)

Fonte: Elaborada pelo autor

Em um BD relacional, essa situação precisa de duas tabelas: PESSOA e CASAMENTO, como exemplificado na Figura 2.12.

PESSOA		
IDENTIFICADOR	NOME	SEXO
1	Eduardo	M
2	Ana Paula	F
3	Ronaldo	M
4	Cibele	F
5	Anabela	F
6	Cecília	F
7	Sérgio	M

CASAMENTO	
IDENTIFICADOR_MARIDO	IDENTIFICADOR_ESPOSA
1	2
1	6
3	2
3	4
3	6

Figura 2.12: Banco de dados equivalente ao modelo ER da Figura 2.11

Fonte: Elaborada pelo autor

Conforme a Figura 2.12, “Eduardo” (IDENTIFICADOR=1) possui duas ocorrências na tabela CASAMENTO. Uma vez que é do sexo masculino, seu identificador aparece na coluna (atributo) IDENTIFICADOR_MARIDO. Significa que

“Eduardo” é casado com duas mulheres: “Ana Paula” (IDENTIFICADOR=2) e “Cecília” (6). Como elas são do sexo feminino, seus identificadores são reproduzidos na coluna IDENTIFICADOR_ESPOSA da tabela CASAMENTO. Por analogia, “Sérgio” (IDENTIFICADOR=7) continua solteiro, por não ter ocorrências na tabela CASAMENTO.



Acrescente a cardinalidade aos relacionamentos da Atividade anterior.

2.2 Exemplos do modelo entidade-relacionamento

Considere um BD para manter informações atualizadas sobre os projetos desenvolvidos por uma empresa e as equipes responsáveis por tais projetos. Todos os integrantes das equipes são funcionários da empresa. Cada projeto é gerenciado por um funcionário e as equipes podem ser formadas por vários funcionários. Independentemente dos projetos, cada funcionário possui uma função na empresa (diretor, analista de sistemas, contador, engenheiro, etc.).

De cada projeto importa armazenar seu número identificador, nome, data de início, data prevista de conclusão e data efetiva de conclusão. Em relação a cada funcionário é importante guardar sua matrícula, nome, sexo e função.

A Figura 2.13 apresenta um diagrama ER que modela o BD proposto. Nela há três relacionamentos, três entidades e uma relação. Observe que um funcionário pode gerenciar muitos projetos, mas um projeto pode ser gerenciado apenas por um funcionário. Entretanto, um funcionário pode trabalhar em muitos projetos simultaneamente e cada projeto pode ter muitos funcionários nele trabalhando. Do mesmo modo, um funcionário possui apenas uma função, mas uma função pode pertencer a muitos funcionários. Portanto, um funcionário não pode ser programador de computadores e contador ao mesmo tempo, mas a empresa pode ter mais de um programador ou contador dentre seus funcionários.

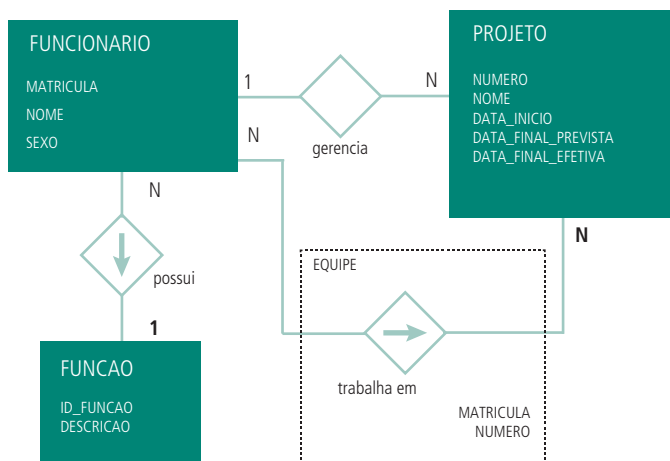


Figura 2.13: Exemplo de Diagrama ER

Fonte: Elaborada pelo autor

Como indicam os atributos do modelo, um projeto ainda em andamento apresentaria a DATA_FINAL_EFETIVA com valor nulo. A existência de DATA_FINAL_PREVISTA e de DATA_FINAL_EFETIVA permite avaliar o atraso do projeto em andamento ou concluído.

Consideremos agora a modelagem de um BD desenvolvido para o gerenciamento de uma biblioteca. Para simplificar, a biblioteca empresta apenas livros. Não há revistas ou DVDs em seu acervo. Também é vetado ao usuário dessa biblioteca reservar um livro indisponível no momento como nas bibliotecas em que o usuário pode entrar em uma fila de reservas, de modo que, ao ser devolvido um exemplar do livro reservado, os usuários que fizeram a reserva podem, conforme sua ordem na fila, requisitar o livro. Por outro lado, o BD modelado deve prover os seguintes requisitos:

- (1)** Manter dados sobre as obras do acervo, como nome dos autores, data da aquisição, editora, edição e título.
- (2)** Manter registro de todos os livros de uma mesma obra.
- (3)** Efetuar o empréstimo e devolução de livros, mantendo registros detalhados ao longo do tempo.
- (4)** Consultar livros por código de identificação, título da obra, nome de autores e gênero da obra (autoajuda, dicionário, literatura, etc.).
- (5)** Manter dados sobre os usuários da biblioteca: matrícula, nome, sexo, data de nascimento, endereço e telefone.

A Figura 2.14 mostra o diagrama ER para o sistema de informação dessa biblioteca.

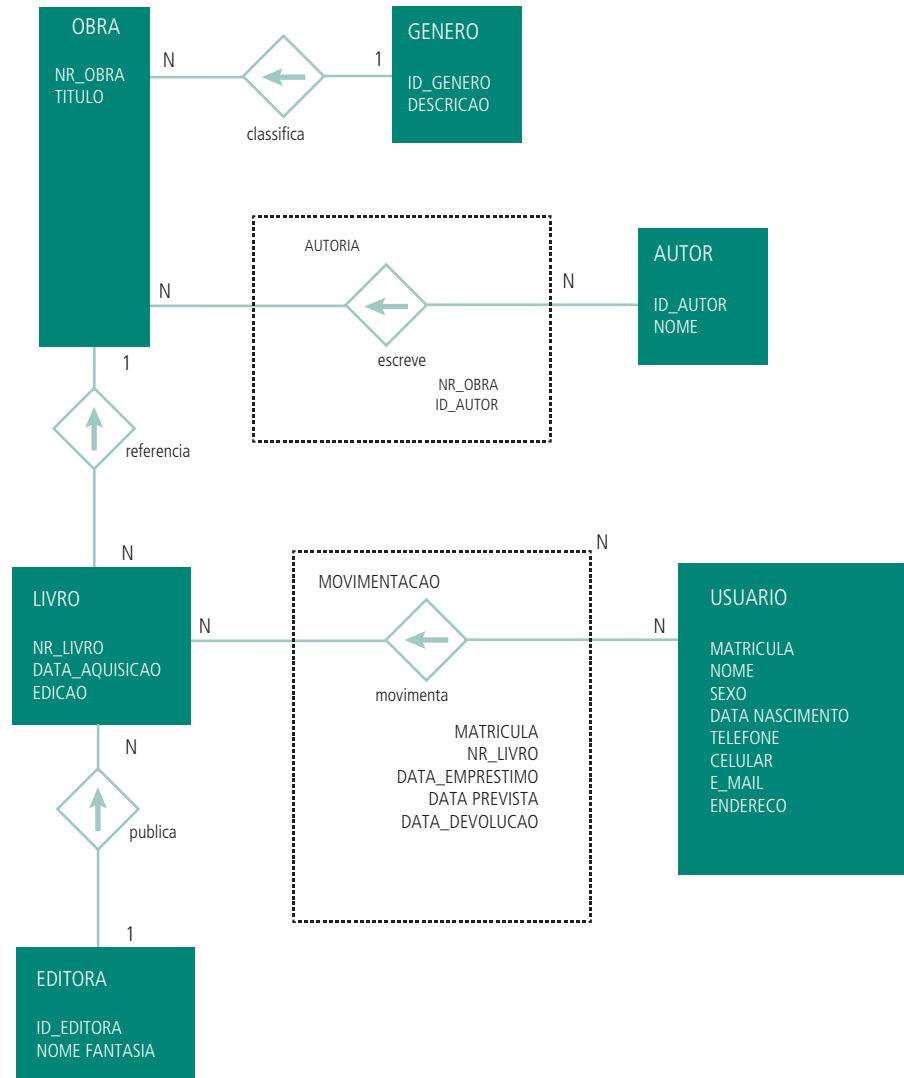


Figura 2.14: Diagrama ER do Sistema de Gerenciamento de Biblioteca

Fonte: Elaborada pelo autor

Nela foram usadas duas entidades: OBRA e LIVRO. A entidade LIVRO, por exemplo, refere-se a uma edição em particular de uma obra. Considere uma obra do século XIX, como *Helena*, de Machado de Assis. Ela já foi publicada por diferentes editoras em mais de um século de existência. Portanto, a entidade EDITORA não poderia estar relacionada à entidade OBRA, mas sim à entidade LIVRO. Se fosse diferente, essa obra somente poderia ser publicada por uma editora.

Por analogia, o atributo EDICAO não pode estar na entidade OBRA, mas em LIVRO. Afinal, somente livros publicados de uma mesma obra podem ter edições diferentes. No exemplo, podem existir 30 livros da obra Helena, cada um publicado por editoras diferentes ou com edições diferentes da mesma editora.

A entidade AUTOR se relaciona à entidade OBRA, pois se relacionada a LIVRO implicaria o relacionamento pouco justificável de seus autores a cada livro. Nesse caso, se 50 livros da obra de Alexandre Dumas, *Os Três Mosqueteiros*, fossem doados para a biblioteca, um funcionário deveria relacionar o nome do autor a cada uma das 50 ocorrências de LIVROS. Com o relacionamento de AUTOR com OBRA, Alexandre Dumas seria relacionado a uma única obra e esta, por sua vez, aos 50 livros dessa mesma obra.

Ainda observando a Figura 2.14, notamos que um gênero pode classificar várias obras, mas uma obra pode ser classificada por um gênero. Assim, *Helena* e *Os Três Mosqueteiros* podem ser classificadas como literatura. Porém, nenhuma dessas obras pode ser classificada em mais de um gênero simultaneamente.

Um autor pode escrever muitas obras, e uma obra, ser escrita por muitos autores. Devido à cardinalidade N:N surgiu a relação AUTORIA (retângulo tracejado), possibilitando obras com mais de um autor, como em livros didáticos ou coletâneas de artigos.

Um livro pode referenciar apenas uma obra, mas uma obra pode ser referenciada por muitos livros. Por outro lado, Uma editora pode publicar muitos livros, mas um livro pode ser publicado apenas por uma editora.

Um usuário pode movimentar vários livros e um mesmo livro pode ser movimentado por vários usuários ao longo do tempo. Mais uma vez a cardinalidade N:N gerou a relação MOVIMENTACAO, a qual armazena empréstimos e devoluções de livros.

Ao emprestar um livro a um usuário, uma ocorrência de movimentação é gerada com o atributo DATA_DEVOLUCAO nulo e DATA_EMPRESTIMO com a data atual do sistema. Na devolução do livro essa movimentação é atualizada com a alteração do atributo DATA_DEVOLUCAO para a data corrente do sistema.

No exemplo de um diagrama ER da Figura 2.15, temos um BD modelado para suportar um sistema de gerenciamento da marcação de consultas médicas.

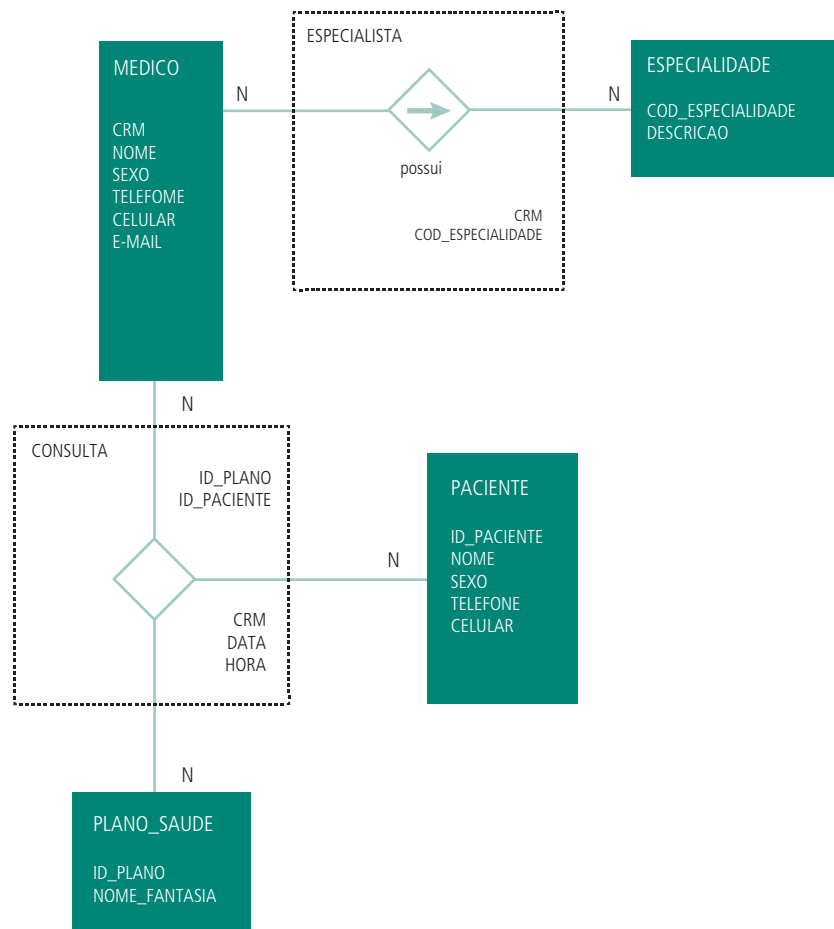


Figura 2.15: Diagrama ER para marcação de consultas médicas

Fonte: Elaborada pelo autor

O diagrama foi modelado a partir dos seguintes requisitos:

- (1) Armazenar dados de médico: número de registro no Conselho Regional de Medicina (CRM), nome completo, sexo, telefone fixo, celular e *e-mail* para contato.
- (2) Armazenar informações de paciente: número identificador, nome completo, sexo, telefone fixo e celular.
- (3) Identificar as especialidades de cada médico (um mesmo médico pode atuar, por exemplo, como clínico geral e dermatologista).
- (4) Agendar consultas para um paciente, especificando o plano de saúde e o médico.

- (5) Como simplificação do modelo não será necessário armazenar os horários de atendimento semanal para cada médico. Consideraremos apenas que todos os médicos atendem todos os dias úteis da semana em um horário fixo e igual para todos.

Note que Consulta é uma relação gerada a partir de um relacionamento ternário entre as entidades MEDICO, PACIENTE e PLANO_SAUDE.

2.3 Generalização/especialização de entidades

Existem situações em que diferentes entidades apresentam algumas características em comum, divergindo apenas em outras características. Na Figura 2.16, por exemplo, temos duas entidades nessa condição de semelhança: MEDICO e PACIENTE.

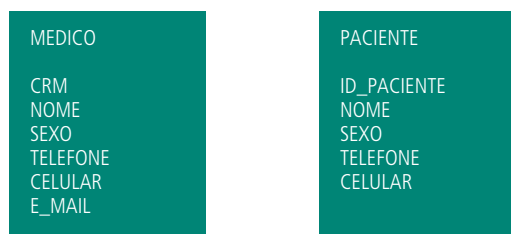


Figura 2.16: Entidades com atributos em comum

Fonte: Elaborada pelo autor

Repare que ambas apresentam quatro atributos em comum: NOME, SEXO, TELEFONE e CELULAR. Médico ainda possui dois atributos exclusivos: CRM e E_MAIL. Por sua vez, a entidade Paciente apresenta um único atributo somente seu: ID_PACIENTE.

Nesses casos, podemos lançar mão de um recurso previsto na modelagem de dados, conhecido como generalização. Então, uma nova entidade, mais genérica ou menos especializada, deve ser criada para agregar os atributos comuns.

Na Figura 2.17 temos uma entidade Pessoa que assume o papel de entidade menos especializada. Ela chama para si os atributos comuns das entidades mais especializadas: Médico e Paciente. O símbolo triangular que interliga essas entidades indica que os atributos de Pessoa são compartilhados por MEDICO e PACIENTE, ou seja, MEDICO e PACIENTE herdam os atributos de PESSOA. Entretanto, o atributo de ID_PACIENTE pertence somente a PACIENTE; e CRM e E_MAIL são atributos exclusivos de MEDICO.

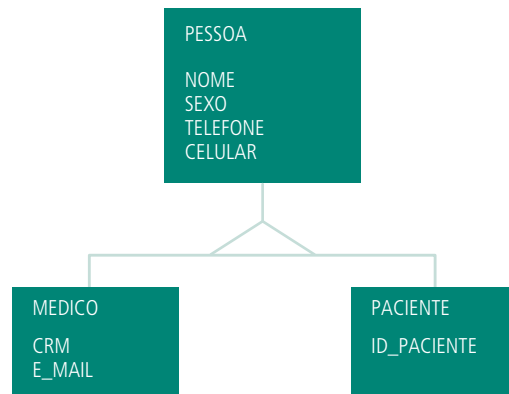


Figura 2.17: Exemplo de generalização ou especialização

Fonte: Elaborada pelo autor



Generalização: entidades de um nível mais baixo de abstração com características comuns; são agrupadas originando uma entidade de nível mais alto.

Especialização: uma entidade de nível mais alto de abstração é desmembrada em várias entidades de nível mais baixo, com características parcialmente distintas.



Construa um diagrama ER completo para um sistema de supermercado que permita as seguintes funcionalidades:

- a) Manter informações sobre produtos (código, nome, preço unitário, quantidade em estoque e estoque mínimo).
- b) Registrar vendas de produtos nos caixas.
- c) Registrar recebimentos – discriminando o tipo: em dinheiro, cheque, cartão de débito, ou cartão de crédito.

2.4 Entidades fracas

Uma entidade fraca é qualquer uma cuja existência não se justifica. Surge apenas em razão de seu relacionamento com outra entidade, considerada forte. A entidade fraca deve apresentar identificadores compostos (formados pelo menos por dois atributos). Um deles deve ser originário da forte. Ele é replicado na fraca para estabelecer uma ligação entre as ocorrências das duas.

Um exemplo de entidade fraca é demonstrado na Figura 2.18. Dependente é a entidade fraca, pois a identificação de um dependente é impossível sem a do sócio. Na verdade, os dependentes não existiriam sem os sócios.

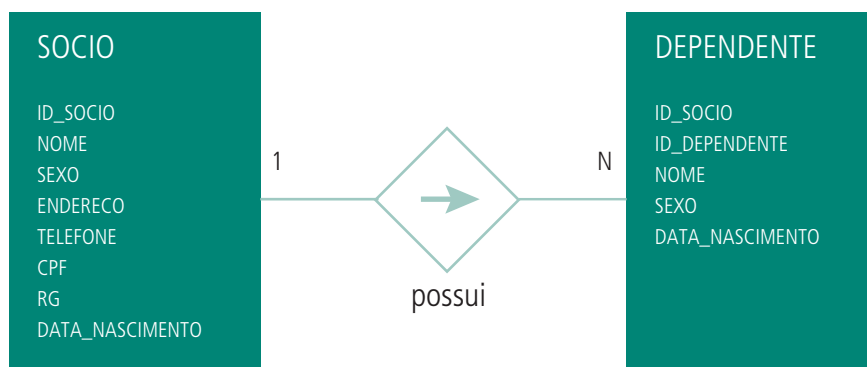


Figura 2.18: Exemplo de entidade fraca (dependente)

Fonte: Elaborada pelo autor

Resumo

Antes de sair criando tabelas, relacionamentos e consultas, devemos projetar um BD que atenda satisfatoriamente aos objetivos de nossos usuários.

Um BD bem modelado demandará pouca ou nenhuma manutenção corretiva no futuro. Por isso, é conveniente despendar parte do tempo inicial projetando e modelando o BD propriamente dito. Uma vez concluída essa etapa, pode-se dedicar tempo à criação das tabelas, relacionamentos, consultas, visões, procedimentos, etc.

Na modelagem de um BD, nos concentramos nos objetivos a que pretendemos atender com essa tecnologia. Se pretendemos criar um BD para um sistema de controle acadêmico que permita armazenar dados sobre alunos, turmas, disciplinas, notas e frequências, quais serão as tabelas necessárias e seus respectivos atributos?

E se o sistema objetivar gerenciar o atendimento de consultas médicas em uma clínica? Certamente, serão outras as tabelas e atributos necessários. Mas, quais serão essas tabelas e como se relacionarão entre si?

Neste capítulo abordamos a modelagem conceitual de dados. Um modelo conceitual objetiva projetar os requisitos de negócio segundo a perspectiva do usuário final. Tendo em vista apenas o modelo relacional de BD, não precisamos ainda determinar qual SGBD utilizaremos para implementar nosso BD, não importando se o SGBD será o Oracle, o Microsoft SQL-Server, o Sybase, o PostgreSQL, o Firebird ou o MySQL. O que modelaremos pode ser implementado em qualquer SGBD relacional.

Atividades de aprendizagem

Na transformação do modelo conceitual Entidade-Relacionamento (MER) em um modelo lógico relacional (MRN), as cardinalidades do relacionamento entre as entidades exercem papel importante.

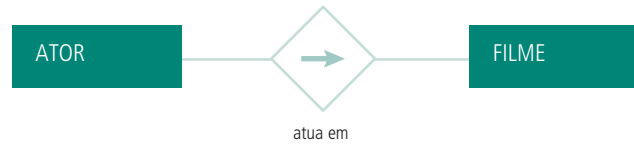


Figura 2.19: Modelo Entidade-Relacionamento: relação entre ATOR e FILME

Fonte: Elaborada pelo autor

Para o modelo apresentado na Figura 2.19 marque a opção **correta** dentre as apresentadas a seguir:

- a) N:N, não gerando uma tabela para o relacionamento.
- b) N:N, gerando uma tabela para o relacionamento.
- c) N:1, não gerando uma tabela para o relacionamento.
- d) N:1, gerando uma tabela para o relacionamento.
- e) 1:N, não gerando uma tabela para o relacionamento.

Aula 3 – Modelagem de Dados:

Modelo Relacional Normalizado

Objetivos

Interpretar um Modelo Relacional Normalizado (MRN).

Implementar um MRN a partir de um MER preexistente.

Definir chaves primárias e estrangeiras para uma entidade.

Efetuar a normalização do MRN, obedecendo às três Formas Normais (FN).

O Modelo Entidade-Relacionamento (MER) é a primeira etapa da modelagem de dados para a implementação de um BD. Desenvolvido pelo analista de sistemas, ele é considerado um modelo conceitual do BD em implementação. Logo, pode ser elaborado antes mesmo da definição do SGBD em que a solução será implementada. A única restrição imposta pelo modelo ER é a escolha de um SGBD também relacional.

Uma vez concluída a etapa, a equipe de analistas de sistemas deve ocupar-se do projeto lógico do BD: o Modelo Relacional Normalizado (MRN). Até aqui ignoramos alguns importantes conceitos que, agora, durante o modelo lógico, devemos considerar. O mais básico desses conceitos é a chave primária.

3.1 Chave primária

Em BD relacionais as informações são armazenadas em tabelas, sendo essencialmente organizadas em linhas e colunas. Cada linha de uma tabela corresponde a uma ocorrência da informação. Assim, em uma tabela de alunos (Figura 3.1), cada linha (ou registro) corresponde ao conjunto de informações inter-relacionadas de um aluno. As colunas (ou campos) correspondem a subdivisões lógicas ou compartimentos para diferentes informações contidas em uma mesma linha.

Na tabela ALUNO na Figura 3.1, cada linha foi subdividida em cinco colunas (MATRICULA, NOME, SEXO, NASCIMENTO e CURSO). No exemplo existem seis linhas armazenando as informações de seis diferentes alunos de uma instituição de ensino. Os dados armazenados nas colunas de uma mesma linha são

relativos a um mesmo aluno. Assim, podemos afirmar que o aluno de nome Rodrigo Bivar tem matrícula 1099, é do sexo masculino, nasceu em 25/12/1992 e cursa Informática. Também podemos confirmar que a aluna Karina Gonçalves encontra-se matriculada no curso de Biologia com a matrícula 1203.

ALUNO				
MATRICULA	NOME	SEXO	NASCIMENTO	CURSO
1099	Rodrigo Bivar	M	25/12/1992	Informática
1203	Karina Gonçalves	F	03/11/1998	Biologia
1399	José da Silva	M	27/02/1995	Matemática
1500	Cesário Antunes	M	16/12/1999	Farmácia
1701	José da Silva	M	03/10/1991	Informática

Figura 3.1: Tabela ALUNO de uma escola

Fonte: Elaborada pelo autor

Uma tabela pode armazenar milhares de linhas (registros). Imaginar a complexidade de localizar um registro específico em meio a milhares de outros não é difícil. As colunas podem ajudar a identificar uma linha específica, distinguindo-a das demais.

Supondo que iremos acessar os dados de um determinado aluno; a busca pela linha deve ser parametrizada pelo valor de uma ou mais colunas, mas não de qualquer coluna. Afinal, diferentes alunos podem, por exemplo, ter o mesmo nome. Na Figura 3.1, essa situação encontra-se exemplificada nas linhas 3 e 5 da tabela, onde estão registrados dois alunos de nome José da Silva. A possibilidade de alunos homônimos torna a coluna Nome inadequada para a identificação de um registro em particular.

A Figura 3.2 mostra o resultado da busca na tabela ALUNO com o critério de pesquisa **NOME = "José da Silva"**; o fato de essa busca/consulta resultar em mais de uma linha evidencia a inadequação da coluna **NOME** na identificação de uma linha em particular.

MATRICULA	NOME	SEXO	NASCIMENTO	CURSO
1399	José da Silva	M	27/02/1995	Matemática
1701	José da Silva	M	03/10/1991	Informática

Figura 3.2: Resultado da busca por aluno de Nome = "José da Silva"

Fonte: Elaborada pelo autor

Por outro lado, a utilização da coluna MATRICULA como critério de pesquisa mostra-se adequada para a identificação de uma única linha. Afinal, não podem existir dois alunos com um mesmo número de MATRICULA **1701**. A Figura 3.3 exemplifica o resultado de uma busca utilizando a coluna MATRICULA como critério de pesquisa.

MATRICULA	NOME	SEXO	NASCIMENTO	CURSO
1701	José da Silva	M	03/10/1991	Informática

Figura 3.3: Resultado da busca por Aluno de MATRICULA = 1701

Fonte: Elaborado pelo autor

Na Figura 3.4 é apresentada a entidade FUNCIONARIO dotada de 12 atributos. Desses, podemos selecionar algumas **superchaves**: CARTEIRA_IDENTIDADE é uma superchave, pois não existe mais de um funcionário com o mesmo número da carteira de identidade. Entretanto, não é a única superchave. CPF também é uma, assim como a combinação de atributos **{NOME e CARTEIRA_IDENTIDADE}**, ou **{NOME e CPF}**, ou ainda **{NOME, CARTEIRA_IDENTIDADE e CPF}** e **{NOME e TELEFONE}**; lembrando que o atributo NOME isoladamente não pode ser considerado uma superchave.

FUNCIONARIO	
NOME	
SEXO	
DATA_NASCIMENTO	
CARTEIRA_IDENTIDADE	
CPF	
LOGRADOURO	
COMPLEMENTO	
BAIRRO	
CIDADE	
UF	
CEP	
TELEFONE	

Figura 3.4: Atributos da entidade FUNCIONARIO

Fonte: Elaborada pelo autor

Portanto, a combinação de atributos {NOME, CPF}, apesar de ser superchave, não pode ser uma **chave candidata**, pois {CPF} também é superchave e subconjunto da combinação. O mesmo vale para a combinação {CARTEIRA_IDENTIDADE, CPF, NOME}, que não é chave candidata, pois existem dois subconjuntos que são superchaves.

A-Z

Superchave

É um conjunto de um ou mais atributos que permite identificar com precisão uma única linha de uma tabela.

A-Z

Chaves candidatas

São um subconjunto específico das superchaves. Somente podem ser chaves candidatas as superchaves mínimas. Ou seja, uma superchave formada pela combinação de mais de um atributo pode ser classificada como uma chave candidata apenas se essa combinação não possuir qualquer subconjunto próprio que seja ele mesmo uma superchave.

A entidade **FUNCIONARIO** possui várias chaves candidatas: **{CPF}** e **{CARTEIRA_IDENTIDADE}** e **{NOME e TELEFONE}**.

A-Z

Chave primária

É uma chave candidata escolhida pelo projetista de BD. É o principal meio de identificação unívoca de uma linha em uma entidade. Toda chave primária obedece às seguintes regras:

(1) Não pode conter valor nulo (a chave primária é de preenchimento obrigatório); e

(2) Não pode conter valores duplicados (não pode existir na entidade mais de uma linha com o mesmo valor de chave primária).

A Figura 3.5 mostra as tabelas do sistema de marcação de consultas com suas respectivas **chaves primárias**. Graficamente, elas são representadas em uma faixa estreita abaixo do nome das entidades. Todo atributo no interior dessa faixa é parte da chave primária.



Figura 3.5: Chaves primárias nas entidades do sistema de marcação de consultas

Fonte: Elaborada pelo autor

O atributo CRM é obviamente a chave primária de MEDICO, pois médicos não possuem o mesmo registro no Conselho Regional de Medicina.

Código da especialidade é chave primária de Especialidade. Embora a descrição também seja única, o código é de menor tamanho: demanda menor esforço de digitação e memorização e ocupa menos espaço de armazenamento. O mesmo ocorre com a identificação do plano de saúde, que é preferível para ser a chave primária.

A entidade Especialista não apresenta atributos próprios. CRM é originário da entidade MEDICO e o código da especialidade, de Especialidade. Logo, eles foram combinados para a formação de uma chave primária composta, pois isoladamente não funcionam como chave primária adequada. Se apenas o CRM fizesse parte da chave primária, não teríamos como registrar um médico com mais de uma especialidade; precisaríamos repetir o CRM e chaves primárias não aceitam duplicação de valores. Por outro lado, se apenas o código da especialidade fizesse parte da chave primária, não teríamos como registrar mais de um médico para uma mesma especialidade. A solução é a combinação dos dois atributos para a chave primária. Nesse caso, o que não pode repetir é a combinação dos valores dos atributos. Ficamos impedidos de registrar um médico e sua especialidade mais de uma vez, mas nada nos impede de registrar um médico com várias especialidades ou uma especialidade para vários médicos.

Qualquer entidade sempre apresentará uma e apenas uma **chave primária**. Contudo, uma chave primária pode ser formada por um único atributo (**chave primária simples**) ou combinação de dois ou mais atributos (**chave primária composta**).



Consulta tem dois atributos próprios: data e hora. Porém, sua chave primária também é composta de atributos de outras entidades: ID_PACIENTE, CRM e DATA. Assim como no caso de Especialista, a replicação dos valores de um dos atributos isoladamente é possível. Somente a repetição dos três atributos simultaneamente seria um problema.

3.2 Chave estrangeira

O conceito de **chave estrangeira** é tão importante quanto o conceito de **chave primária**.

Na Figura 3.6 é apresentado um **diagrama ER** para uma escola com cursos variados. Cada aluno pode matricular-se em apenas um curso, mas cada curso pode ter muitos alunos. Caso um aluno resolva fazer mais de um curso, precisará de nova matrícula. Ao longo do curso, o aluno pode frequentar muitas disciplinas e uma disciplina pode ser frequentada por muitos alunos. Desse relacionamento de cardinalidade **N:N**, surge a relação HISTORICO: para registrar notas e frequências do aluno.

A-Z

Chave estrangeira

É um atributo ou conjunto de atributos originário de uma entidade que é replicado em outra. Essa replicação tem por objetivo estabelecer uma associação entre linhas das duas entidades. Sem as chaves estrangeiras seria impossível criar relacionamentos entre entidades. Uma chave estrangeira em uma determinada entidade sempre será uma chave primária em sua entidade de origem.

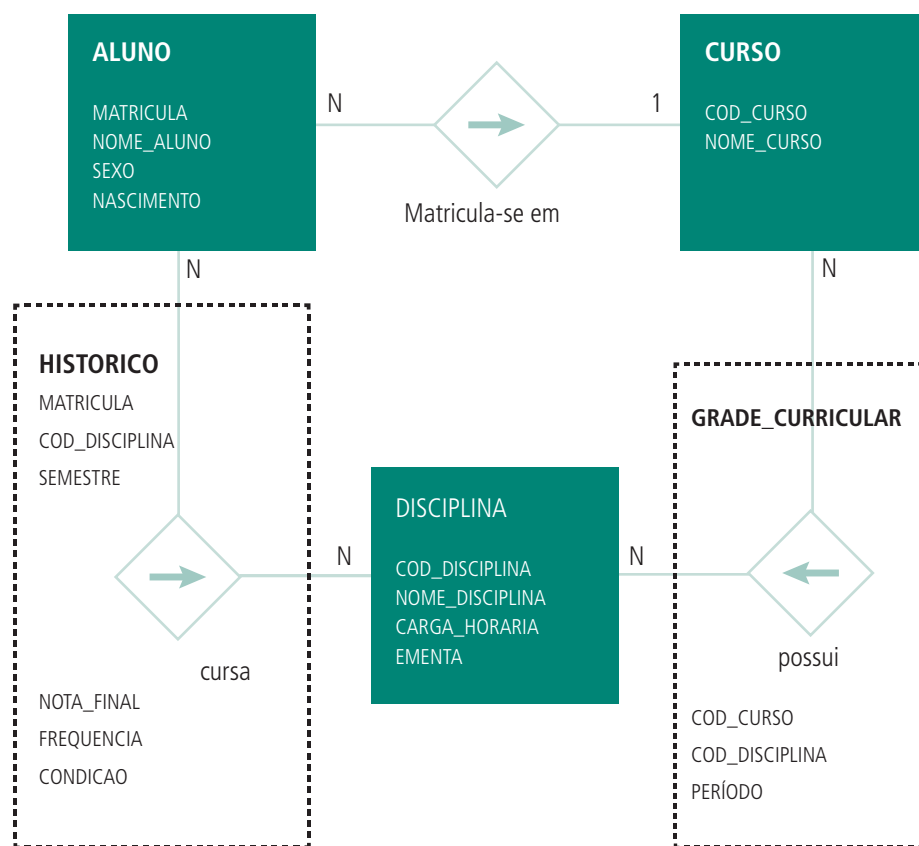


Figura 3.6: Diagrama entidade-relacionamento

Fonte: Elaborada pelo autor

Um curso pode possuir muitas disciplinas e uma disciplina pode pertencer a muitos cursos. Desse relacionamento de cardinalidade **N:N** surge a relação GRADE_CURRICULAR, com informações sobre as disciplinas de cada curso e o período a que pertencem.

Observe que no relacionamento **N:1** entre ALUNO e CURSO não aparecem chaves estrangeiras. No MER, a **chave estrangeira** não aparece explicitamente. Ela é sugerida pela existência do relacionamento. Uma exceção ocorre nos relacionamentos de cardinalidade **N:N**. Histórico apresenta atributos de outras entidades: MATRICULA é proveniente da entidade ALUNO e COD_DISCIPLINA, da DISCIPLINA. O mesmo com GRADE_CURRICULAR: COD_CURSO vem da entidade CURSO e COD_DISCIPLINA, da DISCIPLINA.



Existe uma regra para a determinação de qual tabela deve apresentar a chave estrangeira no relacionamento de cardinalidade 1:N ou N:1. A tabela do lado “N” sempre fica com a chave estrangeira (a chave primária do lado “1”). Quando a cardinalidade é **N:N**, a **chave estrangeira** fica com nenhuma das tabelas

localizadas na extremidade do relacionamento. Afinal todas as extremidades são “N”. Sob essa situação é natural que as **chaves estrangeiras** das tabelas das extremidades fiquem na relação que emerge no centro do relacionamento.

Na Figura 3.7, a tabela ALUNO tem agora a coluna COD_CURSO como chave estrangeira, pois, no relacionamento com CURSO, encontra-se do lado “N” da cardinalidade 1:N.

ALUNO				
MATRICULA	NOME ALUNO	SEXO	NASCIMENTO	COD_CURSO
12359	José da Silva	M	14/02/1985	INF
12361	Andressa Simões	F	25/08/1986	SAN
13000	Lina Silva	F	22/09/1996	INF

CURSO	
COD_CURSO	NOME_CURSO
INF	Informática
SAN	Saneamento Ambiental

Figura 3.7: Tabelas ALUNO e CURSO (destaque para a chave estrangeira)

Fonte: Elaborada pelo autor

Diferente da chave primária, a estrangeira pode conter valores nulos.

Determine as chaves primárias e estrangeiras para as entidades do diagrama ER desenvolvido na primeira atividade de aprendizagem da Aula 2 (restaurante).



3.3 Exemplos de diagrama do MRN

O **Modelo Relacional Normalizado (MRN)**, parte do projeto lógico de um BD, é construído a partir do **Modelo Entidade-Relacionamento (MER)**, que é parte do modelo conceitual. Por ser derivado do **MER**, o **MRN** apresenta-se como mais detalhado e próximo da efetiva implementação em um **SGBD**.

Alguns desenvolvedores insistem em saltar a etapa do modelo conceitual (MER) direto para o projeto lógico (MRN). Entretanto, é uma péssima prática no desenvolvimento de BDs relacionais.



O **MRN** se distingue do **MER** pelas seguintes razões:

(1) Indica as chaves primárias de todas as entidades.

- (2) Indica, explicitamente, as chaves estrangeiras existentes nas entidades.
- (3) Transforma todos os relacionamentos de cardinalidade **N:N** em dois relacionamentos de cardinalidade **1:N**.
- (4) As relações que no MER surgem dos relacionamentos de cardinalidade **N:N** deixam de ser diferenciadas das demais entidades.
- (5) Os relacionamentos deixam de apresentar verbos identificadores, losangos e setas.
- (6) Utiliza novas formas de representação gráfica para a cardinalidade dos relacionamentos. O “N” passa a ser representado por um tridente ou “pé de galinha”, enquanto o “1” é substituído por um simples traço sobre a linha do relacionamento.
- (7) O nome dos atributos obrigatoriamente obedecem a regras próprias dos SGBDs: sem conter espaços em branco entre partes do nome, não começar nome com números, nem empregar cedilha ou acentuação própria da língua portuguesa.



Para facilitar a compreensão do modelo, adotamos a seguinte convenção para as chaves estrangeiras: são atributos sublinhados em que o nome começa com a indicação da entidade de origem, seguido de um ponto e do próprio nome.

A Figura 3.8 apresenta o **MRN** resultante da adaptação do **diagrama ER** da Figura 3.6. **GRADE_CURRICULAR** e **HISTORICO** passaram a ser representados graficamente como as demais entidades do modelo. Até mesmo porque a cardinalidade **N:N** foi substituída por dois relacionamentos **1:N**.

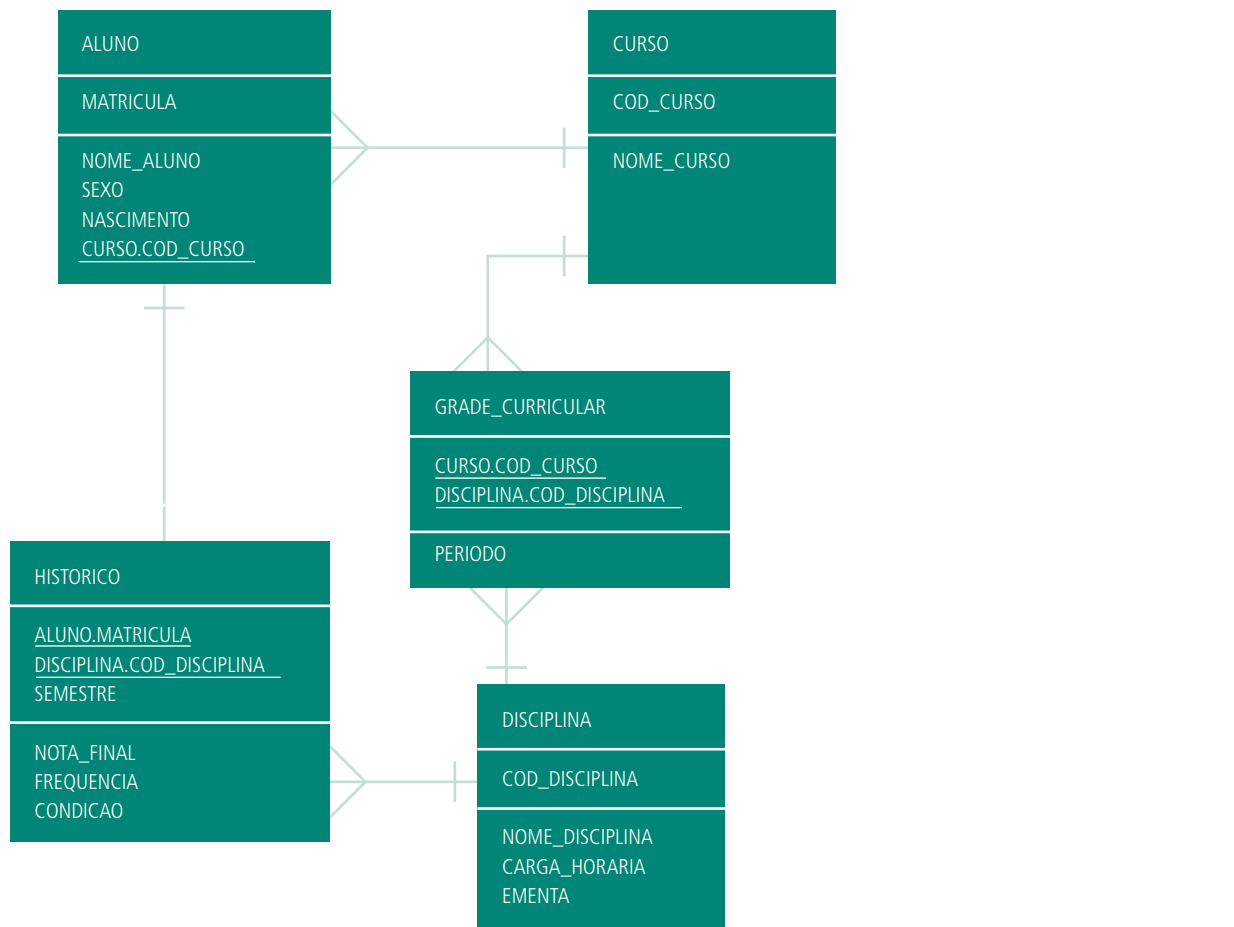


Figura 3.8: Exemplo de Modelo Relacional Normalizado

Fonte: Elaborada pelo autor

Em HISTORICO a chave primária é composta por três atributos: um próprio da entidade (SEMESTRE) e dois de outras entidades. O nome do atributo **ALUNO.MATRICULA** indica que ele é uma **chave estrangeira** vinda de ALUNO. Por analogia, **DISCIPLINA.COD_DISCIPLINA** é igualmente uma **chave estrangeira** proveniente da entidade DISCIPLINA.

A Figura 3.9 mostra o **MRN** de um outro **diagrama ER** já visto na Figura 2.13 da Aula 2 (Exemplo de Diagrama ER). Na entidade Projeto, agora aparece a **chave estrangeira** **FUNCIONARIO.MATRICULA_GERENTE**, consequência direta do relacionamento **1:N** entre FUNCIONARIO e PROJETO. As **chaves primárias** de cada entidade também foram destacadas e nenhum relacionamento de cardinalidade **N:N** pode mais ser encontrado.

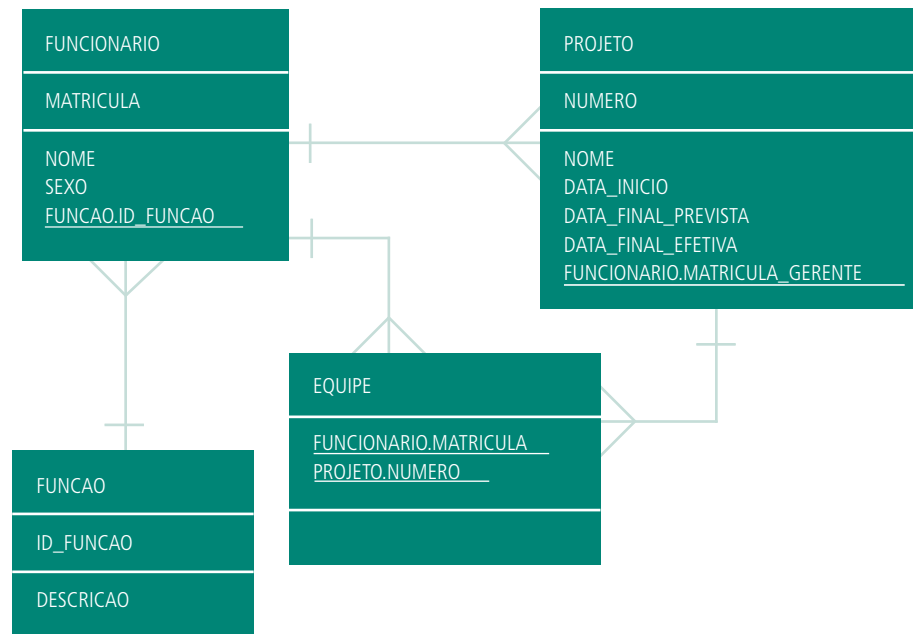


Figura 3.9: MRN de gerenciamento de projetos

Fonte: Elaborada pelo autor



1. Converta os diagramas ER das atividades da Aula 2 (o restaurante e o supermercado) em Modelos Relacionais Normalizados (MRNs) equivalentes.
2. Elabore um exemplo envolvendo três tabelas preenchidas inteiramente, baseado em um dos diagramas do exercício anterior. Destaque as chaves estrangeiras existentes.

3.4 Formas normais

Ao se encerrar o **projeto lógico do Banco de Dados**, resta ainda uma etapa de validação para cada entidade/tabela, conhecida como **Normalização**, na qual o projetista verifica um conjunto de regras ou normas que devem ser observadas, denominadas “**Formas Normais**”. Embora existam inúmeras regras, nos ateremos apenas às três primeiras.

Considere a Figura 3.10 que mostra dados sobre médicos em um hospital. Porém, apresenta uma particularidade que fere uma das **Formas Normais**.

MEDICO					
ID	NOME	SEXO	TELEFONES		
1001	Ostrogenes Ribeiro	M	3711-2278	9947-5611	3345-0587
1023	Christine Álvares	F		8881-0099	3335-7210
1066	Renan Rui Monteforte	M		9967-0909	

Figura 3.10: Exemplo de tabela MEDICO que fere a 1FN

Fonte: Elaborada pelo autor

A tabela MEDICO fere a **1FN** por apresentar um **atributo multivalorado** (até três diferentes telefones de um médico no mesmo campo). Os demais atributos não transgridem a **1FN**, pois apresentam apenas **valores atômicos**.

A Figura 3.11 mostra uma solução alternativa que não fere a **1FN**. Para tanto, o atributo Telefone tornou-se uma tabela (TELEFONE) que se relaciona com a tabela MEDICO. Assim, cada médico pode ter mais de um número de telefone para contato. Contudo, caso seja interessante armazenar também os telefones dos pacientes, essa última solução deixa de ser adequada, mas pode ser facilmente remediada.

MEDICO			TELEFONE	
ID_MEDICO	NOME	SEXO	ID_MEDICO	TELEFONES
1001	Ostrogenes Ribeiro	M	1001	3711-2278
1023	Christine Álvares	F	1001	9947-5611
1066	Renan Rui Monteforte	M	1001	3345-0587
			1023	8881-0099
			1023	3335-7210
			1066	9967-0909

Figura 3.11: Solução que NÃO fere a 1FN

Fonte: Elaborada pelo autor

Na Figura 3.12 há um **MRN** ilustrando uma terceira solução, onde médicos e pacientes podem registrar inúmeros telefones. A diferença é a utilização da **generalização/especialização**, surgindo uma entidade mais geral chamada PESSOA, a qual agrega os atributos comuns a MEDICO e PACIENTE. Portanto, MEDICO e PACIENTE são especializações da entidade PESSOA que se relaciona com a entidade TELEFONE. A solução também tem o mérito de não ferir a **1FN**. Tanto MEDICO quanto PACIENTE possuem como atributo uma chave estrangeira: ID_PESSOA.

A-Z

Primeira Forma Normal (1FN)

Determina que nenhum atributo de uma entidade deve ser **multivalorado**. Todos os atributos existentes – sem exceção – devem conter apenas **valores atômicos**.

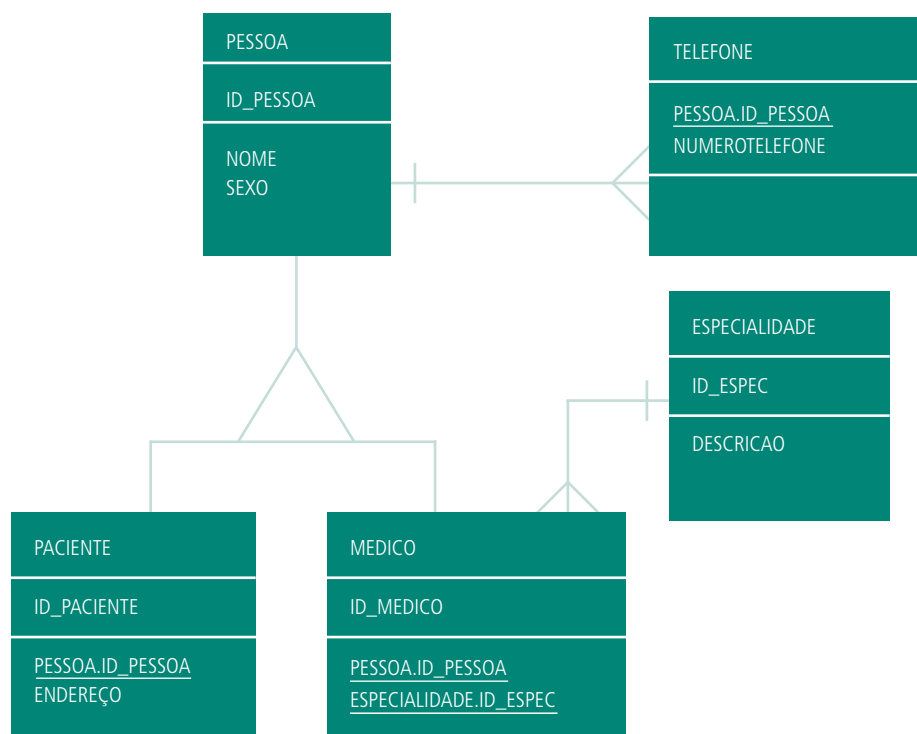


Figura 3.12: MRN de solução que NÃO viola 1FN

Fonte: Elaborada pelo autor.

A-Z

Segunda Forma Normal (2FN)

Determina que tabelas com **chaves primárias compostas** devem sempre ter os demais atributos dependentes de toda essa **chave primária**, e não de apenas parte dela. Uma **tabela** que apresente um ou mais **atributos** que dependam apenas de parte da **chave primária composta** viola a **2FN**.

A **2FN** trata especificamente de quais atributos pertencem a quais entidades. A Figura 3.13 mostra essa violação, apresentando redundância de dados. Há, por exemplo, dois registros/linhas relativos ao filme *Transformers*, de 2007. Os atributos NR_FILME, TITULO e ANO_PRODUCAO tiveram seus valores replicados para que cadastrássemos dois atores nesse filme: Shia LaBeouf e Megan Fox. E esse não é o único problema da tabela. Para registrar a participação de um mesmo ator em mais de um filme, repetimos seu nome e identificador duas vezes. Foi o que aconteceu ao ator Zachary Quinto. Essa também é uma transgressão à **2FN**.

FILME				
NR_FILME	TITULO	ID_ATOM	NOME_ATOM	ANO_PRODUCAO
907	Transformers	SLB	Shia LaBeouf	2007
907	Transformers	MFx	Megan Fox	2007
956	Star Trek	ZQT	Zachary Quinto	2009
998	Heroes	ZQT	Zachary Quinto	2006

Figura 3.13: Exemplo de tabela FILME que viola a 2FN

Fonte: Elaborada pelo autor

Supondo que todos os cinco atributos pertençam à tabela FILME, nenhum deles pode, sozinho, ser a **chave primária**. Afinal, em todas as colunas temos valores replicados. Portanto, essa **chave primária** deve ser **composta**; formada pela combinação de dois ou mais atributos. Poderia ser NR_FILME e ID_ATOR. Para essa combinação em particular não encontramos linhas com valores duplicados. Assim, a combinação {NR_FILME + ID_ATOR} é a chave primária.

Ainda assim, a tabela continua violando a 2FN. A coluna/atributo TITULO, por exemplo, não depende de toda a chave primária composta, nem da combinação de atributos {NR_FILME + ID_ATOR}, mas do atributo NR_FILME. De maneira semelhante, o atributo NOME_ATOR não depende da chave primária composta por inteiro, mas simplesmente do atributo ID_ATOR (que é apenas parte da chave primária).

A solução para essa violação da **2FN** está na Figura 3.14, onde os atributos da antiga tabela FILME são redistribuídos entre ela e a tabela ATOR. Outra tabela, ELENCO, tornou-se necessária em razão da cardinalidade **N:N** do relacionamento entre as duas tabelas. Afinal, um ator pode atuar em muitos filmes e um filme pode contar com muitos atores. A entidade ELENCO contém apenas dois atributos, ambos chaves estrangeiras. Em FILME a chave primária é o atributo **NR_FILME**. Na ATOR, **ID_ATOR**. Já em Elenco, é composta pela combinação de atributos {**NR_FILME, ID_ATOR**}.

FILME			ELENCO		ATOR	
NR_FILME	TITULO	ANO_PRODUCAO	NR_FILME	ID_ATOR	ID_ATOR	NOME_ATOR
907	Transformers	2007	907	SLB	SLB	Shia LaBeouf
956	Star Trek	2009	907	MFY	MFY	Megan Fox
998	Heroes	2006	956	ZQT	ZQT	Zachary Quinto
			998	ZQT		

Figura 3.14: Solução que NÃO fere a 2FN

Fonte: Elaborada pelo autor

A Figura 3.15 apresenta um exemplo de tabela que viola a **3FN**. A tabela VENDA apresenta cinco atributos: NR_PEDIDO, COD_PRODUTO, QUANTIDADE, PRECO_UNITARIO e SUBTOTAL. A chave primária composta é formada pela combinação de atributos {**NR_PEDIDO + COD_PRODUTO**}. Afinal, um mesmo pedido pode conter variados produtos simultaneamente. Ela viola a **3FN** em razão da existência da coluna SUBTOTAL. Esse **atributo** depende de outros dois que não fazem parte da **chave primária**: SUBTOTAL é o resultado da multiplicação de **QUANTIDADE** e **PRECO_UNITARIO**. Ou seja, o valor de SUBTOTAL depende dos valores

A-Z

Terceira Forma Normal (3FN)

Determina que não deve haver interdependência entre os atributos que não fazem parte da chave primária. Todos os atributos de uma entidade devem depender apenas da chave primária dessa mesma entidade e de nenhum outro atributo. Entretanto, para estar na 3FN uma tabela também deve estar necessariamente na 2FN.

desses dois atributos e nenhum deles é parte da **chave primária**. Nesse caso, a solução passa pela eliminação da coluna SUBTOTAL. Somente assim a tabela **VENDA** estará em conformidade com a **3FN**.

VENDA				
NR_PEDIDO	COD_PRODUTO	QUANTIDADE	PRECO_UNITARIO	SUBTOTAL
1035	10-900	10	40,99	409,90
1035	10-991	5	10,00	50,00
1213	20-560	3	35,20	105,60

Figura 3.15: Exemplo de tabela VENDA, com violação da 3FN

Fonte: Elaborada pelo autor

A Figura 3.16 mostra outra violação da **3FN**. A tabela **FILME** refere-se a uma **videolocadora**, em que o valor da locação de um filme depende da categoria em que ele está classificado. Se o filme é um lançamento, seu preço é de R\$ 4,00; se é da categoria ouro, é R\$ 3,00; e da prata, é R\$ 2,00.

FILME			
NR_FILME	TITULO	CATEGORIA	PRECO
5598	Batman - The Dark Knight	Prata	2,00
5600	Wall-E	Ouro	3,00
5601	Se Beber, Não Case	Lançamento	4,00
5603	Avatar	Lançamento	4,00

Figura 3.16: Outro Exemplo de violação da 3FN

Fonte: Elaborada pelo autor

O problema apresentado na Figura 3.16 é que o atributo “PRECO” apresenta uma interdependência em relação ao atributo CATEGORIA, que não faz parte da chave primária. A solução é a criação de uma tabela de CATEGORIA e a migração para lá do atributo PRECO, obedecendo então a **3FN**. A entidade FILME se relacionaria com ela e, portanto, teria uma coluna de categoria (ID_CATEGORIA). Nessa solução, o reajuste de preços é simples. Basta atualizar a tabela CATEGORIA e todos os Filmes estarão com novos preços, conforme Figura 3.17 a seguir.

FILME			CATEGORIA	
NR_FILME	TITULO	CATEGORIA	ID_CATEGORIA	PRECO
5598	Batman - The Dark Knight	Prata	Lançamento	4,00
5600	Wall-E	Ouro	Ouro	3,00
5601	Se Beber, Não Case	Lançamento	Prata	2,00
5603	Avatar	Lançamento		

Figura 3.17: Tabelas em conformidade com a 3FN

Fonte: Elaborado pelo autor

Altere um dos Modelos Relacionais Normalizados feitos anteriormente (o supermercado ou o restaurante) para que viole a 2FN ou a 3FN. Em seguida explique o problema dessa violação no contexto escolhido (do supermercado ou do restaurante).



3.5 Ferramentas CASE

Existem *softwares* que auxiliam a **modelagem de dados** para BDs e sistemas de informação: as **ferramentas CASE**. Alguns são *softwares* **proprietário**; outros, **livre**. Em alguns casos existem diferentes versões da mesma ferramenta, sendo algumas mais simples, com menos recursos e gratuitas; outras, sem restrições de uso e pagas.

Algumas dessas ferramentas CASE não somente permitem a modelagem dos dados, como, uma vez concluída a modelagem, oferecem a possibilidade de criação do próprio BD em diferentes **SGBDs**.

Há ferramentas CASE cuja sofisticação permite até mesmo a **engenharia reversa**. Em vez de produzirmos um modelo apoiado por esses *softwares* para somente então gerarmos o BD, na engenharia reversa o banco já está pronto e pretendemos, a partir dele, gerar o modelo dos dados. Isso é útil para organizações que herdaram ou adquiriram sistemas com nenhuma documentação. A compreensão das partes fica limitada para as equipes responsáveis pela manutenção e pela ampliação dos sistemas.



O MySQL Workbench está disponível para *download* em <http://wb.mysql.com/>

Dentre as ferramentas CASE existentes, destacamos:

- (1) Doctor Case,
- (2) ERWin,
- (3) DB Design,
- (4) brModelo,
- (5) MySQL Workbench
- (6) System Architect.



Muitos desses *softwares* podem ser conseguidos por meio de *downloads* na internet. Se estiver interessado em desenvolver sistemas de informação para a *web*, um *software* interessante é o WampServer (disponível para *download* em <http://www.wampserver.com/>). Ele instala de maneira integrada o SGBD MySQL, a linguagem PHP para programação de *sites* dinâmicos para a internet e o servidor Apache para executar os *sites* criados.

Resumo

Nesta aula abordamos a modelagem lógica de dados. Um modelo lógico objetiva adaptar o modelo conceitual (visto na aula anterior) à teoria de sistemas relacionais, definindo chaves primárias (simples ou compostas) e chaves estrangeiras. As chaves primárias são responsáveis por identificar univocamente um registro de uma tabela (entidade).

Assim enfatizamos a conversão de Modelos de Entidade e Relacionamentos (MER) em Modelos Relacionais Normalizados (MRN). Para tanto, os modelos gerados precisaram obedecer a um certo número de normas, identificadas como Formas Normais (FN), das quais aqui nos restringimos às três primeiras

Atividades de aprendizagem

1. Muitas vezes, o administrador de DB se depara com situações em que possui o esquema do BD, porém não tem o modelo conceitual equivalente. Baseado nisso, dada a definição do esquema abaixo, gere o modelo conceitual equivalente (processo de engenharia reversa).

```
FABRICANTE {IDFABR, NOMEFABR}  
MODELO {IDMODELO, NOMEMODELO, PRECO}  
COR {IDCOR, DESCRICAOCOR}  
AUTOMOVEL {IDAUTOM, IDFABR, IDMODELO, NRPLACA, NRCHASSI,  
IDCOR}  
CLIENTE {IDCLIENTE, NOMECLIENTE, TELEFONE}  
VENDA {IDCLIENTE, IDAUTOM, DATAVENDA, VALOR, DATAENTREGA}
```

2. O pequeno diagrama de MRN da Figura 3.18 apresenta alguns problemas graves. Identifique-os um a um, explique as dificuldades que causam e, finalmente, corrija-o.

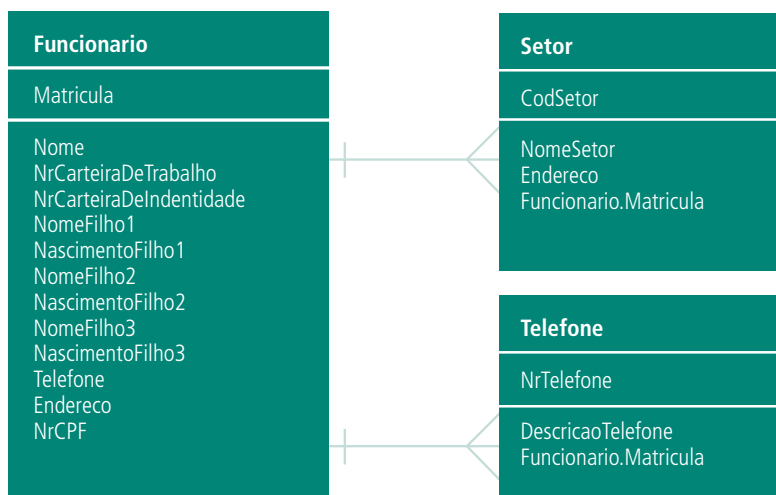


Figura 3.18: Diagrama de MRN problemático

Fonte: Elaborado pelo autor

3. Analise os relacionamentos no diagrama mostrado na Figura 3.19 a seguir.

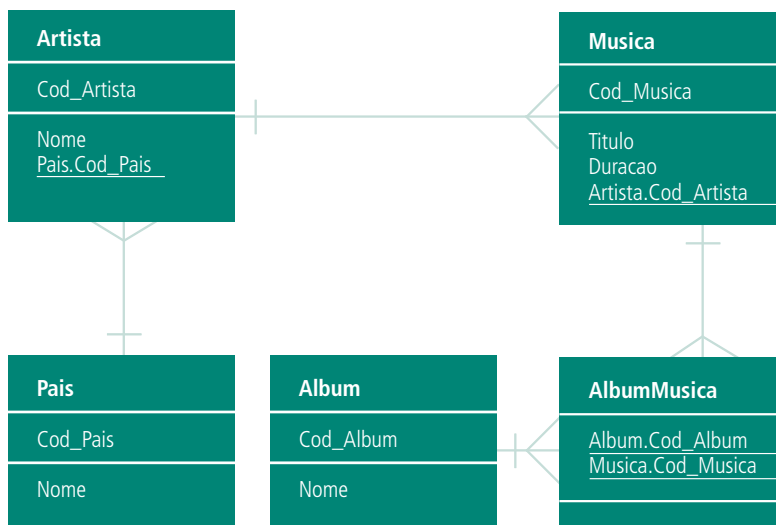


Figura 3.19: Diagrama de MRN de álbuns de música

Fonte: Elaborada pelo autor

Assinale a afirmação **incorreta** dentre as seguintes opções:

- a) Uma mesma música pode fazer parte de álbuns diferentes.
- b) Um artista, obrigatoriamente, possui um país de origem.
- c) Não há artista sem música.
- d) Toda música tem um artista.
- e) Podem existir várias músicas em um álbum.

Aula 4 – Criação de Banco de Dados

Objetivos

Utilizar o *Data Definition Language* (DDL) da SQL para a criação de tabelas.

Definir chaves primárias e chaves estrangeiras.

Estabelecer relacionamentos entre tabelas.

Alterar e excluir tabelas.

Inserir novos registros em tabelas.

Alterar e excluir registros de tabelas.

4.1 Linguagem SQL

Considerada pelo *American National Standard Institute* (ANSI) uma linguagem de consulta padrão de BDs desde 1986, a SQL surgiu no início da década de 1970 como uma interface para o SYSTEM R – um SGBD relacional da IBM. A princípio foi batizada de *Structured English Query Language* (**Sequel**), mas logo teve seu nome alterado para *Structured Query Language* (**SQL**). Uma grande variedade de SGBDs no mercado suporta a linguagem SQL. Isso a tornou uma linguagem padrão de BDs relacionais e levou o ANSI a publicar um padrão SQL.

O aprendizado da linguagem SQL é estratégico para o profissional de informática que lida com a tecnologia de BD. Seu conhecimento permite manipular não apenas um SGBD ou um pequeno subconjunto deles, mas sim uma grande variedade. Dentre os SGBDs que usam o SQL vale destacar: Oracle, DB-II, MS SQL-Server, Interbase, Firebird, MySQL, PostgreSQL e Sybase.



As principais características da linguagem SQL são:

- (1) Apresenta comandos para criação, alteração e exclusão de tabelas, relacionamentos e índices. Esse subconjunto de comandos, o **Data Definition Language (DDL)**, será o nosso ponto de partida nesta aula.
- (2) Possui um subconjunto de comandos para a manipulação de dados, o **Data Manipulation Language (DML)**, que permite inserir, alterar ou excluir os registros/linhas de uma tabela.
- (3) Permite **gerenciar a segurança das informações** no BD, atribuindo níveis diferenciados de autorização a usuários ou grupos de usuários.
- (4) Define regras sobre informações armazenadas pela especificação de **restrições de integridade**.
- (5) Cria **visões**. Nem sempre é desejável que um grupo de usuários tenha acesso a todas as colunas ou linhas de uma tabela. A SQL permite que se restrinja esse acesso criando “**visões**”. Suponha, por exemplo, uma tabela de FUNCIONARIOS que tenha como colunas {MATRICULA, NOME, SENHA, CODDEPARTAMENTO}. Pode-se criar uma visão que permita a visualização de todas as colunas, exceto a senha. Pode-se também criar uma visão que permita a visualização das linhas de Funcionários cuja coluna COD-DEPARTAMENTO tenha valor “Diretoria”. Em seguida, o administrador do BD pode conceder a grupos de usuários a permissão de acesso a uma ou outra visão, e revogar a permissão de acesso direto à tabela. Assim, alguns usuários poderão acessar {MATRICULA, NOME, CODDEPARTAMENTO}, mas não {SENHA}. Outros poderão acessar todas as colunas da tabela, mas não todas as linhas. Poderão visualizar ou manipular apenas os dados dos funcionários lotados na “Diretoria”.
- (6) Permite melhor controlar transações (**Transaction Control**). Uma **transação** corresponde à manipulação de vários itens de dados que, para manter a integridade, destes, exige que nenhum ou todos sejam atualizados. A clássica transferência de valores de uma conta bancária A para a conta B é um bom exemplo de **transação**. Ela funciona como um mecanismo para recuperação de paradas e falhas em BDs. Caso o valor seja debitado da conta A, deve ser creditado na conta B. Uma eventual falha que ocorra no meio do processo não pode resultar na atualização apenas da conta A, pois a tecnologia de transações em BD evita problemas de integridade em razão da atualização parcial de dados.

(7) Manipula dados por uma linguagem embutida (***Embedded Data Manipulation Language***) especialmente projetada para o acesso ao BD por meio de linguagens de programação como C, Cobol, Fortran, Java e PHP. É esse recurso que permite a programas aplicativos acessarem e manipularem o BD.

Apenas as duas primeiras vantagens serão abordadas nesta aula. As demais serão tratadas mais adiante.

4.2 MySQL

O **MySQL**, no Brasil pronuncia-se “Mai Ésse Que Éle”, é um **SGBD** desenvolvido pela MySQL AB, que agora é propriedade da Sun Microsystems. Em 2009, a Sun foi adquirida pela Oracle, uma reconhecida desenvolvedora de SGBDs.

Rapidamente o MySQL converteu-se em um dos SGBDs mais populares no mundo – especialmente para desenvolvimento de aplicações na *web*. Grande parte dessa popularidade se deve ao fato de apresentar a opção **gratuita, software livre**. Porém, essa não é a única razão de sua aceitação, pois o **PostgreSQL**, seu principal concorrente **open source**, apresenta mais recursos.

Algumas características do MySQL que contribuem para seu sucesso são:

- (1) **Facilidade** de instalação.
- (2) É um **software “enxuto”**, de tamanho reduzido, facilmente executado em *hardware* de configuração modesta e, conseqüentemente, com grande velocidade de recuperação de informações.
- (3) A cada nova versão traz novidades interessantes para a comunidade de seus usuários (**responsividade** para a comunidade).
- (4) Apresenta interface de fácil utilização para outros *softwares*, como as linguagens de programação **C, Java, Perl, PHP, Python, Ruby**. O mesmo é válido para as linguagens **Microsoft.NET**. Todos esses *softwares* apresentam bibliotecas de funções para utilização do MySQL. Se não possuírem essas bibliotecas, o MySQL também suporta a Conectividade de Banco de Dados Aberto (ODBC) padrão.
- (5) Apresenta modalidades diferenciadas de licença: uma **versão gratuita** e **outra paga**.



O MySQL encontra-se na versão 5.4 e pode ser obtido por meio de download na url: <http://dev.mysql.com/downloads/mysql/5.4.html>.

(6) Dispõe de **suporte técnico** para usuários.

(7) Utiliza a linguagem **SQL**.

(8) Apresenta **interface** de linha de código (no estilo **prompt** do **DOS**), mas o MySQL possui inúmeras ferramentas com interface gráfica baseada em janelas como a do Windows adquiridas separadamente: algumas gratuitas, outras não.

4.3 Como criar um Banco de Dados

Uma vez instalado, o servidor do MySQL pode ser executado facilmente no Linux ou no Windows. A Figura 4.1 mostra a tela inicial do MySQL, quando uma senha é solicitada ao usuário: “Enter password:”

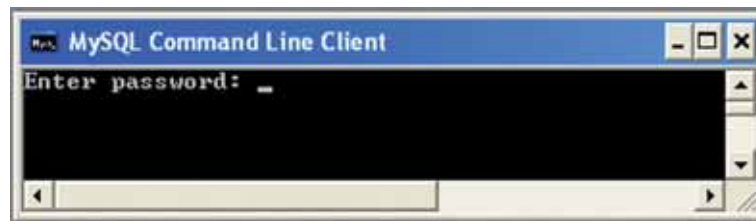


Figura 4.1: Tela inicial do SGBD MySQL

Fonte: Print screen, Windows XP

Diante das palavras “Enter password:” (“Entre com a senha:”) digite **root** (com letras minúsculas). Essa é a senha padrão inicial para o usuário de mesmo nome: **root**. Logicamente, em uma organização, a senha não pode permanecer com esse valor. Mesmo porque o usuário **root** é o que apresenta maior nível de autorização no MySQL.

Após digitar **root** e teclar <ENTER> podemos criar um BD novo com a sintaxe:

```
CREATE DATABASE <NOME DO BANCO DE DADOS>;
```

Como exemplo, vamos criar um BD de nome **BIBLIOTECA** com o comando:

```
CREATE DATABASE BIBLIOTECA;
```

E teclar <ENTER>. Para comprovar sua criação executamos o comando:

```
SHOW DATABASES;
```

O resultado desse comando é a apresentação de uma lista com todos os BDs existentes em sua máquina. Você pode ter vários **Bancos de Dados** MySQL criados em seu computador, como na Figura 4.2.


```
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.39-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

mysql> CREATE DATABASE BIBLIOTECA;
Query OK, 1 row affected (0.00 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| biblioteca        |
| mysql             |
| test              |
+-----+
4 rows in set (0.03 sec)
```

Figura 4.2: Resultado do comando “SHOW DATABASES;” após criação de BIBLIOTECA

Fonte: Elaborada pelo autor

Note que a senha informada pelo usuário diante dos dizeres “Enter Password:” é mascarada por caracteres “*” (um para cada caractere da senha). Essa é uma medida de segurança, pois os comandos executados no MySQL permanecem visíveis na tela do computador, e podem ser visualizados por qualquer pessoa que passe no momento.

O MySQL devolve uma resposta – em inglês – em que:

- Saúda o usuário final com um “**Bem-vindo ao MySQL Monitor**” (“**Welcome to the MySQL monitor**”).
- Alerta que cada comando no MySQL deve ser finalizado com um ponto e vírgula ou uma contrabarra seguida do caractere “g” (“**Commands end with ; or \g.**”).
- Informa que a identificação da sua conexão é 1 (“**Your MySQL connection id is 1.**”). Esse número pode variar, dependendo de quantos usuários estiverem conectados ao BD no momento (incluindo aplicativos em execução).
- Esclarece a versão do MySQL que está sendo executada (5.1.39), bem como a sua licença de uso (GPL) (“**Server version: 5.1.39-community MySQL Community Server (GPL)**”).
- Informa comandos para acessar a ajuda (**help**) ou limpar a tela: “**Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.**”. Caso o usuário final digite “**help;**” ou “**\h**”, o MySQL apresentará uma tela de ajuda com uma lista de comandos disponíveis acompanhada de uma breve descrição para cada um. Por outro lado, se o usuário final digitar “**\c**” o MySQL limpará a tela do monitor.

Em seguida, o usuário se depara com o *prompt* do MySQL (**"mysql>"**) seguido do cursor piscando à sua direita. Então, o comando **"CREATE DATABASE BIBLIOTECA;"** foi digitado pelo usuário. O MySQL lhe devolve uma resposta: **"Query OK, 1 row affected (0.05 sec)"**. Nessa mensagem, o usuário é informado de que o comando foi executado com sucesso (**"Query OK"**), de que uma linha foi afetada no BD (**"1 row affected"**) e de que sua execução gastou 0,05 segundos (**"0.05 sec"**). Logo, essa mensagem indica que o Banco de Dados **BIBLIOTECA** foi criado com sucesso.

Desconfiado, o usuário digitou o comando **"SHOW DATABASES;"**. Em resposta, o MySQL lhe devolve uma tabela com uma coluna chamada *databases* (Bancos de Dados). Cada linha da tabela traz a lista de nomes de BDs existentes no servidor.

Observe que, na Figura 4.2, existem quatro Bancos de Dados gravados. O BD **BIBLIOTECA**, que acabamos de criar; e os **INFORMATION_SCHEMA**, **MYSQL** e **TEST** que são BDs criados pelo instalador do SGBD MySQL.

Criamos o BD BIBLIOTECA, mas ainda não o abrimos. Para efetuar qualquer operação no BD BIBLIOTECA, como criar as tabelas e nelas inserir dados, precisamos antes "abrir" ou "entrar" nesse BD utilizando o comando **USE** do MySQL:

```
USE <NOME-DO-BANCO DE DADOS>;
```

Ou seja, digitamos **USE**, o nome do BD que abriremos e o ponto e vírgula:

```
mysql> USE BIBLIOTECA;  
Database changed
```

Ao pressionar a tecla <ENTER>, o MySQL devolveu a seguinte resposta: **"Database changed"**, o usuário obteve sucesso e abriu ou entrou no BD BIBLIOTECA. Agora, uma ação do usuário afetará somente esse BD.

4.4 Criação de tabelas

Uma vez dentro do BD BIBLIOTECA e de posse do **MER** e **MRN**, podemos criar tabelas e relacionamentos. A Figura 4.3 é o diagrama de nossos exemplos.

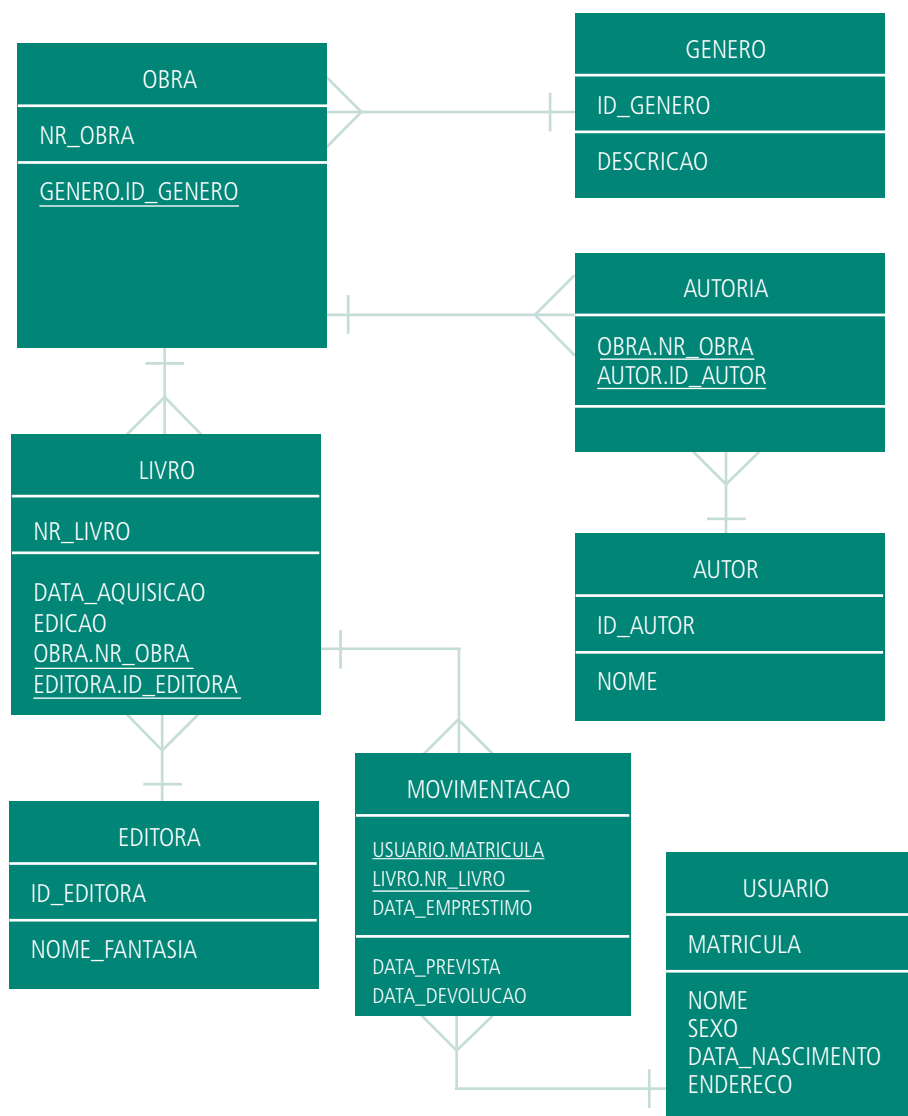


Figura 4.3: Diagrama do Sistema de Gerenciamento de BIBLIOTECA

Fonte: Elaborada pelo autor

A entidade USUARIO foi levemente alterada. Os atributos CEP, E_MAIL, TELEFONE e CELULAR foram omitidos no intuito de simplificar o exemplo.

Veja um exemplo simplificado do comando SQL para a criação de uma tabela:

```
mysql> CREATE TABLE GENERO (
  -> ID_GENERO CHAR(3) NOT NULL PRIMARY KEY,
  -> DESCRICAO VARCHAR(25) NOT NULL );
```

Query OK, 0 rows affected (0.44 sec)

Criamos então a tabela GÊNERO “**CREATE TABLE GÊNERO (...)**”. O parêntese aberto logo após o nome da tabela indica que, em seguida, descreveremos uma lista de colunas/atributos. A definição de cada atributo deve ser separada de outra por vírgula.

Assim, o trecho “**ID_GÊNERO CHAR(3) NOT NULL PRIMARY KEY,**” define uma coluna com as seguintes características:

- Nome da coluna é **ID_GÊNERO**.
- Tipo de dado: **caractere** (com tamanho máximo delimitado em três caracteres): “**CHAR(3)**”. O tipo caractere aceita combinações de algarismos (de “0” a “9”), letras (de “A” a “Z” ou de “a” a “z”) e símbolos como “#”, “@”, “!”, “.” ou mesmo espaços em branco (“ ”). A delimitação de tamanho igual a 3 significa que essa coluna pode armazenar combinações de até três caracteres. Exemplo: “LIT”, “CIE”, “AA”, “F” ou “ ”.
- A expressão “**NOT NULL**” (“**NÃO NULO**”) determina que o preenchimento é obrigatório. Em outras palavras, o SGBD não permitirá que se tente inserir uma linha (registro) nessa tabela sem que se informe algum valor para essa coluna. Essa expressão não é obrigatória em todos os atributos, pois eventualmente podem existir colunas cujo preenchimento não seja obrigatório.
- A expressão “**PRIMARY KEY**” (“**CHAVE PRIMÁRIA**”) define essa coluna como a chave primária simples dessa tabela. Mais adiante mostraremos como definir chaves primárias compostas.

Após a vírgula, definimos a segunda coluna da tabela GÊNERO (“**DESCRICAO VARCHAR(25) NOT NULL;**”). Seu nome é “DESCRICAO” e o tipo de dado é “**VARCHAR**” de tamanho 25 caracteres. Seu preenchimento também é obrigatório (“**NOT NULL**”) – afinal, não faz sentido criar um identificador de gênero sem a respectiva descrição. Como essa é a última coluna da tabela, sua definição não é sucedida por vírgula. Em seu lugar, fechamos o parêntese e acrescentamos um ponto e vírgula. O parêntese fechado indica que a lista de colunas da tabela foi encerrada e o ponto e vírgula sinaliza a conclusão do comando.



O tipo de dados VARCHAR (caractere variável) é semelhante ao CHAR (caractere). A diferença é que CHAR armazena dados de tamanho fixo, enquanto VARCHAR, dados com tamanho variável. Portanto, se atribuirmos o conte-

údo “**Ana Souza**” para uma coluna VARCHAR que pode armazenar até **45 caracteres**, o armazenamento desse conteúdo ocupará apenas o espaço de nove caracteres. Dessa forma estaremos economizando no registro o espaço de 36 caracteres. Em outras palavras, a coluna poderia armazenar até 45 caracteres (sua capacidade máxima), mas o conteúdo “**Ana Souza**” precisa apenas de nove (incluindo o espaço entre “Ana” e “Souza”). No VARCHAR utiliza-se apenas a quantidade necessária de espaço para o armazenamento.

O CHAR, por outro lado, é um tipo de dado mais antigo e menos otimizado. O armazenamento do conteúdo “**Ana Souza**” – de nove caracteres – em uma coluna CHAR de tamanho 45 caracteres, irá efetivamente ocupar o espaço de armazenamento relativo a todos os 45 caracteres. Nenhuma economia de espaço em disco ocorrerá, pois o espaço de 36 caracteres não utilizado ainda estará reservado para a coluna.

Ao final – após o usuário teclar <ENTER> – o MySQL retorna a mensagem de que o comando foi executado com sucesso (“**Query OK**”), nenhuma linha foi afetada (“**0 rows affected**”) e sua execução despendeu 0,44 segundos (“**(0.44 sec)**”).

Em seguida é apresentado o comando para a criação da tabela AUTOR:

```
mysql> CREATE TABLE AUTOR (  
-> ID_AUTOR INTEGER NOT NULL PRIMARY KEY,  
-> NOME VARCHAR(40) NOT NULL );
```

Query OK, 0 rows affected (0.01 sec)

Note que a coluna “**ID_AUTOR**” foi definida como do tipo inteiro (**INTEGER**), de preenchimento obrigatório (“**NOT NULL**”) e como chave primária simples da tabela **AUTOR** (“**PRIMARY KEY**”). O tipo inteiro aceita apenas valores numéricos inteiros. Letras, símbolos, espaço em branco e mesmo números reais – com casas decimais – não são aceitos como valores válidos. A coluna “**NOME**” foi definida como do tipo **VARCHAR** de tamanho 40 caracteres e preenchimento obrigatório (“**NOT NULL**”).

A seguir está a criação da tabela “**USUARIO**” com suas cinco colunas.

```
mysql> CREATE TABLE USUARIO (
-> MATRICULA INTEGER NOT NULL PRIMARY KEY
-> AUTO_INCREMENT,
-> NOME VARCHAR(50) NOT NULL,
-> SEXO CHAR(1) NOT NULL,
-> DATA_NASCIMENTO DATE NOT NULL,
-> ENDERECO VARCHAR(60) NOT NULL );
```

Query OK, 0 rows affected (0.02 sec)

As novidades nesse comando são:

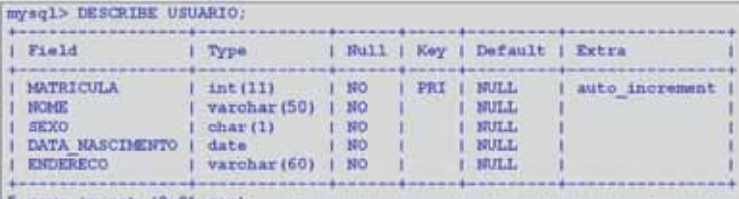
- O atributo **AUTO_INCREMENT** na definição da coluna MATRICULA permite que o valor desta última seja omitido no momento de inserção de um novo registro. Caso não seja informado um valor para essa coluna durante a inclusão da nova linha (registro), o BD, automaticamente, gera um valor que equivale ao maior valor para essa coluna já existente na tabela, acrescido de uma unidade. Assim, caso a maior matrícula na tabela seja 1099, o BD atribuirá o valor 1100 (= 1099 + 1) para o próximo registro de MATRICULA.
- A coluna DATA_NASCIMENTO foi definida como do tipo **DATE**. Portanto, ela está preparada para armazenar somente dados do tipo data.

O comando **DESCRIBE** nos permite checar as características de uma tabela criada:

```
DESCRIBE <NOME_TABELA>;
```

Para acessar as informações sobre a tabela USUARIO, por exemplo, devemos digitar **DESCRIBE USUARIO;** e teclar <ENTER>.

O resultado do comando **DESCRIBE** é a apresentação de uma tabela descritiva (Figura 4.4) em que são apresentadas as seguintes informações:



```
mysql> DESCRIBE USUARIO;
```

Field	Type	Null	Key	Default	Extra
MATRICULA	int(11)	NO	PRI	NULL	auto_increment
NOME	varchar(50)	NO		NULL	
SEXO	char(1)	NO		NULL	
DATA_NASCIMENTO	date	NO		NULL	
ENDERECO	varchar(60)	NO		NULL	

5 rows in set (0.01 sec)

Figura 4.4: Resultado do Comando “DESCRIBE USUARIO;”

Fonte: Elaborada pelo autor

- A coluna **Field (Campo)** com os nomes das colunas da tabela em questão.
- A coluna **Type (Tipo)** com os tipos de dados de cada coluna da tabela.
- A coluna **Null (Nulo)** indica, para cada coluna, quais são as de preenchimento obrigatório ou não.
- A coluna **Key (Chave)** especifica quais colunas fazem parte da chave primária. Acima, apenas a coluna MATRICULA tem o valor “PRI” e, portanto, faz parte da chave-primária da tabela.
- A coluna **Default (Padrão)** relaciona os valores *defaults* de cada coluna da tabela. No exemplo acima, nenhuma coluna da tabela possui um valor padrão. Caso determinássemos “F” como o valor default para a coluna SEXO da tabela USUARIO, todo registro teria automaticamente esse valor, exceto quando indicássemos explicitamente outro valor.
- A coluna **Extra** traz informações especiais quando elas existem. No exemplo acima, ela indica o atributo **AUTO_INCREMENT** da coluna MATRICULA.

A próxima tabela a ser definida é a que armazena dados das editoras:

```
mysql> CREATE TABLE EDITORA (
  -> ID_EDITORA CHAR(5) NOT NULL,
  -> NOME_FANTASIA VARCHAR(60) NOT NULL,
  -> PRIMARY KEY(ID_EDITORA) );
```

Query OK, 0 rows affected (0.38 sec)

O comando acima apresenta a chave primária pouco antes do final do comando. Até aqui indicamos a chave primária simples na declaração da própria coluna qualificada, como apresentado no exemplo abaixo, mas ambos estão corretos.

```
mysql> CREATE TABLE EDITORA (
  -> ID_EDITORA CHAR(5) NOT NULL PRIMARY KEY,
  -> NOME_FANTASIA VARCHAR(60) NOT NULL );
```

Query OK, 0 rows affected (0.38 sec)

Porém, para chaves primárias compostas estudadas na Aula 3, somente poderá ser executado com a cláusula **PRIMARY KEY()** em separado.

Seja qual for a forma escolhida para a criação da tabela EDITORA, o comando **DESCRIBE** nos revelará a estrutura da Figura 4.5 a seguir.

```
mysql> DESCRIBE EDITORA;
```

Field	Type	Null	Key	Default	Extra
ID_EDITORA	char(5)	NO	PRI	NULL	
NOME_FANTASIA	varchar(60)	NO		NULL	

2 rows in set (0.01 sec)

Figura 4.5: Resultado do comando “DESCRIBE EDITORA;”

Fonte: Elaborada pelo autor



Até aqui criamos tabelas sem **chaves estrangeiras**. Isso foi proposital, pois quando definirmos a chave estrangeira da tabela OBRA, por exemplo, a tabela GENERO necessariamente já deverá existir em nosso BD. Somente assim poderemos estabelecer o relacionamento entre a coluna ID_GENERO da tabela GENERO e a coluna ID_GENERO da tabela OBRA (ver Figura 4.3). Uma forma alternativa seria criar as tabelas OBRA e GENERO sem estabelecer o **relacionamento** no ato da criação de OBRA. Esse relacionamento somente seria feito em um momento posterior pela alteração da tabela OBRA por meio do comando **ALTER TABLE** (que será abordado mais adiante).

O comando para a criação da tabela OBRA é:

```
mysql> CREATE TABLE OBRA (  
-> NR_OBRA INTEGER NOT NULL AUTO_INCREMENT,  
-> TITULO VARCHAR (60) NOT NULL,  
-> ID_GENERO CHAR (3) NOT NULL,  
-> PRIMARY KEY (NR_OBRA),  
-> FOREIGN KEY (ID_GENERO)  
-> REFERENCES GENERO (ID_GENERO) ON DELETE RESTRICT );
```

Query OK, 0 rows affected (0.09 sec)

Esse comando SQL apresenta a cláusula **FOREIGN KEY (Chave Estrangeira)** para especificar que a coluna ID_GENERO da tabela OBRA faz referência à da tabela GENERO. Portanto, ID_GENERO é **chave primária** na tabela GENERO e **chave estrangeira** na tabela OBRA. Ao estabelecermos ID_GENERO como **chave estrangeira** em OBRA, o SGBD assegura a **integridade referencial** dos dados. Isso significa que:

- ID_GENERO em OBRA somente poderá conter valores previamente inseridos em ID_GENERO de GENERO. Em outras palavras, não podemos ca-

dastrar uma OBRA com ID_GENERO = "LIT" se não houver em GENERO um ID_GENERO = "LIT".

- O uso da expressão "**ON DELETE RESTRICT**" impede que o registro de GENERO com um dado ID_GENERO seja excluído se o mesmo valor de ID_GENERO existir em pelo menos um registro de OBRA. Assim, não podemos excluir um GENERO com ID_GENERO = "DID" se existir pelo menos um registro de OBRA com ID_GENERO = "DID".

Usando o comando "DESCRIBE OBRA", obtemos o resultado da Figura 4.6 a seguir.

```
mysql> DESCRIBE OBRA;
```

Field	Type	Null	Key	Default	Extra
NR_OBRA	int(11)	NO	PRI	NULL	auto_increment
TITULO	varchar(60)	NO		NULL	
ID_GENERO	char(3)	NO	MUL	NULL	

3 rows in set (0.00 sec)

Figura 4.6: Resultado do comando "DESCRIBE OBRA;"

Fonte: Elaborada pelo autor

Note que a coluna **Key (Chave)** para ID_GENERO apresenta o valor "**NULL**" indicando seu caráter de chave que aceita valores múltiplos. Em sua tabela de origem – GENERO – esse campo é uma chave primária e, portanto, não aceita valores duplicados: não pode existir mais de um registro na tabela com o mesmo valor. Porém, em OBRA, esse campo não é a **chave primária**, mas sim a **chave estrangeira**. Por isso, pode apresentar inúmeros registros com o mesmo valor.

O comando SQL para criar a tabela LIVRO é:

```
mysql> CREATE TABLE LIVRO (  
-> NR_LIVRO INTEGER NOT NULL AUTO_INCREMENT,  
-> DATA_AQUISICAO DATE NOT NULL,  
-> EDICAO INTEGER,  
-> NR_OBRA INTEGER NOT NULL,  
-> ID_EDITORA CHAR(5) NOT NULL,  
-> PRIMARY KEY (NR_LIVRO),  
-> FOREIGN KEY (NR_OBRA) REFERECES OBRA (NR_OBRA)  
-> ON DELETE RESTRICT,  
-> FOREIGN KEY (ID_EDITORA) REFERENCES  
-> EDITORA (ID_EDITORA) ON DELETE RESTRICT );
```

Query OK, 0 rows affected (0.39 sec)

Aqui a tabela LIVRO possui duas chaves estrangeiras; relacionando-a a duas tabelas. **NR_OBRA** relaciona LIVRO à tabela OBRA e ID_EDITORA o relaciona à EDITORA.

O comando SQL para a criação da tabela AUTORIA é o seguinte:

```
mysql> CREATE TABLE AUTORIA (  
-> NR_OBRA INTEGER NOT NULL,  
-> ID_AUTOR INTEGER NOT NULL,  
-> PRIMARY KEY (NR_OBRA, ID_AUTOR),  
-> FOREIGN KEY (NR_OBRA) REFERENCES  
-> OBRA (NR_OBRA) ON DELETE RESTRICT,  
-> FOREIGN KEY (ID_AUTOR) REFERENCES  
-> AUTOR (ID_AUTOR) ON DELETE CASCADE );
```

Query OK, 0 rows affected (0.02 sec)

Mais uma vez, temos duas chaves estrangeiras em uma tabela. Na verdade, poderiam existir tabelas com um número bem maior de chaves estrangeiras, mas as novidades desse comando SQL de criação de tabela são:

- A ocorrência de uma **chave primária** composta. **NR_OBRA** e **ID_AUTOR** formam a chave primária e, por isso, são listados no interior dos parênteses de PRIMARY KEY() e os nomes dos campos (colunas) são separados por uma vírgula.
- Na definição da **chave estrangeira (FOREIGN KEY) ID_AUTOR** a expressão "**ON DELETE CASCADE**" substitui a "**ON DELETE RESTRICT**". Significa que o SGBD não impedirá a exclusão de um registro de AUTOR cujo valor de ID_AUTOR exista na coluna de mesmo nome da tabela AUTORIA. Em vez disso, ele propagará o efeito da exclusão em AUTOR sobre todos os registros relacionados em AUTORIA. Assim, se excluirmos o registro de AUTOR com ID_AUTOR = 135, o SGBD se encarregará de, automaticamente, excluir todos os registros de AUTORIA com ID_AUTOR = 135 – sejam eles quantos forem.

O comando SQL para a criação da tabela MOVIMENTACAO é:

```
mysql> CREATE TABLE MOVIMENTACAO (
-> MATRICULA INTEGER NOT NULL,
-> NR_LIVRO INTEGER NOT NULL,
-> DATA_EMPRESTIMO DATE NOT NULL,
-> DATA_PREVISTA DATE NOT NULL,
-> DATA_DEVOLUCAO DATE,
-> MULTA NUMERIC(5,2),
-> PRIMARY KEY (MATRICULA, NR_LIVRO, DATA_EMPRESTIMO ),
-> FOREIGN KEY (MATRICULA) REFERECES
-> USUARIO (MATRICULA) ON DELETE RESTRICT,
-> FOREIGN KEY (NR_LIVRO) REFERENCES
-> LIVRO (NR_LIVRO) ON DELETE RESTRICT);
```

Query OK, 0 rows affected (0.03 sec)

Nessa tabela a chave primária composta é formada por: **MATRICULA**, **NR_LIVRO** e **DATA_EMPRESTIMO**. Portanto, individualmente, esses campos podem apresentar valores repetidos, mas jamais poderá haver um registro de **MOVIMENTACAO** com repetição de valores para a combinação dos três campos.

Talvez você tenha notado que essa tabela **MOVIMENTACAO** não é exatamente igual à entidade de mesmo nome no diagrama do **MRN** da Figura 4.3. Acrescentamos o campo **MULTA** apenas para exemplificar a utilização do tipo de dado **NUMERIC**.



A coluna (campo) **MULTA** foi definida como sendo **NUMERIC(5,2)**. Esse tipo de dado aceita números com casas decimais. Nesse caso o valor numérico tem capacidade para armazenar até cinco casas, sendo duas após a vírgula. Portanto, o valor máximo que pode ser armazenado é 999,99 – um valor bem alto para uma multa de biblioteca.

O comando **DESCRIBE** para a tabela **MOVIMENTACAO** resulta na Figura 4.7.

```
mysql> DESCRIBE MOVIMENTACAO;
```

Field	Type	Null	Key	Default	Extra
MATRICULA	int(11)	NO	PRI	NULL	
NR_LIVRO	int(11)	NO	PRI	NULL	
DATA_EMPRESTIMO	date	NO	PRI	NULL	
DATA_PREVISTA	date	NO		NULL	
DATA_DEVOLUCAO	date	YES		NULL	
MULTA	decimal(5,2)	YES		NULL	

6 rows in set (0.02 sec)

Figura 4.7: Resultado do comando “DESCRIBE MOVIMENTACAO;”

Fonte: Elaborada pelo autor

Note que a DATA_DEVOLUCAO e a MULTA não são campos de preenchimento obrigatório. Apresentam valor “**yes**” na coluna **Null**. Afinal, a DATA_DEVOLUCAO permanecerá nula (sem preenchimento) até que o livro seja finalmente devolvido pelo usuário e a MULTA somente será preenchida quando DATA_DEVOLUCAO estiver preenchida e for posterior à DATA_PREVISTA para a devolução do livro.

O comando **SHOW TABLES;** (Figura 4.8) listará o nome de todas as tabelas criadas em nosso banco BIBLIOTECA.

```
mysql> SHOW TABLES;
+-----+
| Tables_in_biblioteca |
+-----+
| autor                 |
| autoria               |
| editora               |
| genero                |
| livro                 |
| movimentacao          |
| obra                  |
| usuario               |
+-----+
8 rows in set (0.00 sec)
```

Figura 4.8: Resultado do comando “SHOW TABLES;” após criação de MOVIMENTACAO

Fonte: Elaborada pelo autor

O comando **DROP TABLE** deve ser usado sempre que quisermos apagar uma tabela completamente: seus registros e sua estrutura.

```
mysql> DROP TABLE MOVIMENTACAO;
```

```
Query OK, 0 rows affected (0.05 sec)
```

O comando acima exclui a tabela MOVIMENTACAO. Após executado, o comando “SHOW TABLES;” exibe uma lista de sete tabelas e não oito (Figura 4.9).

```
mysql> show tables;
+-----+
| Tables_in_biblioteca |
+-----+
| autor                 |
| autoria               |
| editora               |
| genero                |
| livro                 |
| obra                  |
| usuario               |
+-----+
7 rows in set (0.00 sec)
```

Figura 4.9: Resultado do comando "SHOW TABLES;" após a Exclusão de MOVIMENTACAO

Fonte: Elaborada pelo autor

Vamos recriar a tabela MOVIMENTACAO ligeiramente diferente para introduzirmos o comando de alteração de tabelas (ALTER TABLE).

```
mysql> CREATE TABLE MOVIMENTACAO (
-> MATRICULA INTEGER NOT NULL,
-> NR_LIVRO INTEGER NOT NULL,
-> DATA_EMPRESTIMO DATE NOT NULL,
-> DATA_PREVISTA DATE NOT NULL,
-> DATA_DEVOLUCAO DATE,
-> PRIMARY KEY (MATRICULA, NR_LIVRO, DATA_EMPRESTIMO));
```

Query OK, 0 rows affected (0.00 sec)

Não incluímos o campo MULTA, nem chaves estrangeiras (FOREIGN KEY). Se percebermos o erro depois de criar a tabela, restam-nos duas alternativas:

- Destruir a tabela (DROP TABLE) para, em seguida, recriá-la sem erros.
- Alterar a tabela (ALTER TABLE), fazendo os devidos acréscimos.

Vamos começar acrescentando o campo MULTA.

```
mysql> ALTER TABLE MOVIMENTACAO ADD MULTA NUMERIC(5,2);
```

Query OK, 0 rows affected (0.41 sec)

Records: 0 Duplicates: 0 Warnings: 0

Repare que a estrutura básica do comando é

```
ALTER TABLE <nome-tabela> ADD <acréscimos>;
```

Vamos agora acrescentar a chave estrangeira da tabela USUARIO:

```
mysql> ALTER TABLE MOVIMENTACAO ADD FOREIGN KEY (MATRICULA)  
REFERENCES USUARIO (MATRICULA) ON DELETE RESTRICT;
```

Query OK, 0 rows affected (0.41 sec)

Records: 0 Duplicates: 0 Warnings: 0

Agora, acrescentaremos a outra chave estrangeira:

```
mysql> ALTER TABLE MOVIMENTACAO ADD FOREIGN KEY (NR_LIVRO)  
REFERENCES LIVRO (NR_LIVRO) ON DELETE RESTRICT;
```

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0



1. Escreva comandos SQL para (1) criar um Banco de Dados de nome loja; (2) “abrir” esse mesmo Banco de Dados e (3) criar tabelas conforme o especificado pelo diagrama abaixo. Utilize o bom senso para definir os tipos de dados para cada coluna.

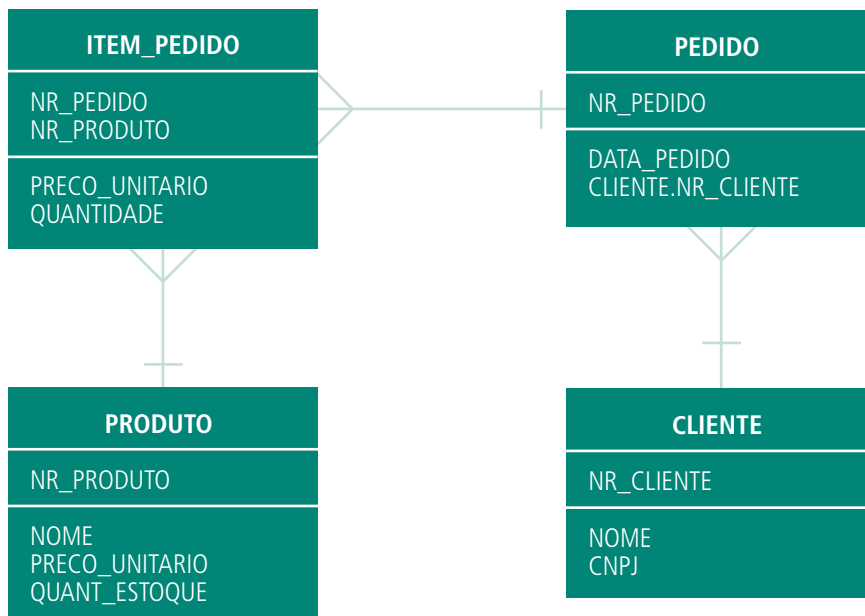


Figura 4.10: Diagrama MRN de PEDIDO de CLIENTE

Fonte: Elaborada pelo autor

2. Escreva um comando SQL para alterar a tabela PEDIDO, acrescentando um campo de nome OBSERVACAO e tipo VARCHAR de tamanho 90.
3. Descreva um comando SQL para apagar a tabela PRODUTO.
4. Explique o que ocorreria se tentássemos excluir um registro de CLIENTE quando a tabela PEDIDO foi criada usando a seguinte cláusula para a definição da chave estrangeira: **FOREIGN KEY (NR_CLIENTE) REFERENCES CLIENTE (NR_CLIENTE) ON DELETE CASCADE**.
5. Diferencie a expressão **"ON DELETE CASCADE"** da expressão **"ON DELETE RESTRICT"**. Explique as consequências de se adotar uma no lugar de outra, quando definimos uma chave estrangeira.

4.5 Inserção de registro em tabela

A sintaxe **SQL** para a inserção de um novo registro em uma tabela é:

```
INSERT INTO <NOME-DA-TABELA> (<LISTA-NOME-CAMPOS>)  
VALUES (<LISTA-VALORES-CAMPOS>);
```

Após a palavra-chave “**INSERT INTO**”, informamos o nome da tabela onde o registro será inserido; depois, separados por vírgulas, a lista de nomes de campos em que serão inseridos valores. Campos de preenchimento obrigatório não poderão ser omitidos para não resultar em falha de execução, pois uma mensagem de erro aparecerá na tela e o registro não será inserido.

Após a lista de nomes de campos, acrescentamos a palavra-chave “**VALUES**” e a lista de valores a serem inseridos nos campos separados por vírgulas obedecendo à especificidade de cada tipo de dado. Os valores do tipo DATE devem ser informados entre aspas simples, CHAR e VARCHAR devem ser apresentados entre aspas duplas e números devem ser escritos sem delimitadores (aspas). Número com casas decimais não devem ser formatados com vírgula, mas sim com ponto.

As duas listas, de nomes e de valores de campos, devem ser apresentadas com a mesma sequência. O primeiro nome de campo receberá o primeiro valor de campo indicado, o segundo receberá o segundo valor de campo, e assim por diante.

```
mysql> INSERT INTO GENERO (ID_GENERO, DESCRICAO)  
VALUES ("LIT", "LITERATURA");
```

```
Query OK, 1 row affected (0.02 sec)
```

O comando **SQL** anterior exemplifica a inserção de um novo registro na tabela GENERO (“**INSERT INTO GENERO**”). A lista de nomes de campos (“ID_GENERO, DESCRICAO”) indica a ordem em que os valores serão informados (“**VALUES (“LIT”, “LITERATURA”);**”). Portanto, esse comando insere um registro (linha) na tabela GENERO onde o ID_GENERO recebe valor “LIT” e a DESCRICAO recebe valor “LITERATURA”.


```
mysql> INSERT INTO GENERO (ID_GENERO, DESCRICAO) VALUES  
("DID", "DIDATICO E TECNICO");
```

Query OK, 1 row affected (0.02 sec)

Um novo registro foi inserido na tabela GENERO. Agora o ID_GENERO recebe o valor "DID" e a DESCRICAO recebe o valor "DIDATICO E TECNICO".

```
mysql> INSERT INTO GENERO (DESCRICAO, ID_GENERO) VALUES  
("AUTO-AJUDA", "AAJ");
```

Query OK, 1 row affected (0.01 sec)

Repare uma sutil mudança neste comando. Alteramos a ordem na lista de nomes de campos e tivemos de alterar a sequência na lista de valores. Primeiro informamos a DESCRICAO e depois o ID_GENERO.

```
mysql> INSERT INTO GENERO (DESCRICAO, ID_GENERO) VALUES  
("RELIGIOSO", "AAJ");
```

ERROR 1062 (23000): Duplicate entry 'AAJ' for key 1

O comando acima gerou um erro e, em consequência, o registro com DESCRICAO = "RELIGIOSO" não foi inserido na tabela GENERO. A razão está na tentativa de violação da **chave primária**. Já existe um registro com ID_GENERO = "AAJ" na tabela GENERO e ID_GENERO, como toda **chave primária**, não aceita valores nulos ou duplicados. Nesse caso, o SGBD impediu, automaticamente, a inclusão de um novo registro com **chave primária duplicada**.

```
mysql> INSERT INTO GENERO (DESCRICAO, ID_GENERO) VALUES  
("RELIGIOSO", "REL");
```

Query OK, 1 row affected (0.00 sec)

Para corrigir o problema da chave primária duplicada, alteramos o valor de

ID_GENERO para "REL".

A sintaxe "**SELECT * FROM <NOME-DA-TABELA>;**" é uma forma básica de consulta aos registros de uma tabela que exibe todas as colunas e as linhas da tabela. Como escrevemos "**SELECT * FROM GENERO;**" (Figura 4.11), todos os registros da tabela GENERO nos são apresentados na tela. Nesse caso, quatro linhas (rows).

```
mysql> SELECT * FROM GENERO;
+-----+-----+
| ID_GENERO | DESCRICAO |
+-----+-----+
| AAJ      | AUTOAJUDA |
| DID      | DIDATICO E TECNICO |
| LIT      | LITERATURA |
| REL      | RELIGIOSO |
+-----+-----+
4 rows in set (0.00 sec)
```

Figura 4.11: Resultado do comando "SELECT * FROM GENERO;"

Fonte: Elaborada pelo autor

```
mysql> INSERT INTO OBRA (ID_GENERO, TITULO) VALUES
("LIT", "HELENA");
```

Query OK, 1 row affected (0.02 sec)

O comando acima nos traz uma novidade: o campo **NR_OBRA**(chave **primária** de OBRA) não teve seu valor discriminado no comando de inserção. Sabemos que ID_GENERO = "LIT" e TITULO = "HELENA", mas qual o valor de NR_OBRA? Por ser a chave primária, seu valor não pode ser nulo nem duplicado. Mas, na tabela OBRA, definimos a coluna NR_OBRA com o atributo AUTO_INCREMENT. Portanto, o SGBD gera seu valor automaticamente. Como ele é o primeiro registro, NR_OBRA=1.

```
mysql> INSERT INTO OBRA (TITULO, ID_GENERO) VALUES ("DOM
CASMURRO", "LIT");
```

Query OK, 1 row affected (0.00 sec)

O registro da obra de TITULO = "DOM CASMURRO" terá NR_OBRA = 2 (também gerado automaticamente pelo SGBD).

```
mysql> INSERT INTO OBRA (TITULO, ID_GENERO) VALUES ("TEORIA DO CAOS", "XYZ");
```

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`biblioteca/obra`, CONSTRAINT `obra_ibfk_1` FOREIGN KEY (`ID_GENERO`) REFERENCES `genero` (`ID_GENERO`))
```

O comando acima gerou um erro por violar a **restrição de integridade** da **chave estrangeira**. Observe que não existe em GENERO um registro com ID_GENERO = "XYZ". Portanto, ID_GENERO em OBRA também não pode conter o valor "XYZ". O SGBD tratou de impedir a inclusão desse registro e alertou o usuário sobre o erro.

```
mysql> INSERT INTO OBRA (TITULO, ID_GENERO) VALUES ("TEORIA DO CAOS", "DID");
```

```
Query OK, 1 row affected (0.00 sec)
```

Alertado sobre o problema, o usuário corrige o conteúdo de ID_GENERO para um valor válido. No comando acima, ID_GENERO = "DID" substitui o valor "XYZ". Como "DID" é um valor previamente cadastrado em GENERO, o SGBD não impede a inclusão do registro de OBRA.

```
mysql> INSERT INTO GENERO VALUES ("CIE", "DIVULGACAO CIENTIFICA");
```

```
Query OK, 1 row affected (0.02 sec)
```

O comando SQL acima volta a inserir um novo registro na tabela GENERO. Contudo, o faz sutilmente diferente. A lista de nomes de campos foi omitida. Isso é possível, se:

1. os valores estejam na ordem de criação dos campos na tabela e que
2. o valor de nenhum campo, mesmo de preenchimento não obrigatório, seja omitido.

Assim, o SGBD sabe automaticamente que ID_GENERO = "CIE" e DESCRICAO = "DIVULGACAO CIENTIFICA".

```
mysql> INSERT INTO OBRA VALUES ("5","OS TRÊS  
MOSQUETEIROS","LIT");
```

Query OK, 1 row affected (0.02 sec)

Note que o mesmo foi feito no comando acima. Para inserir um novo registro de OBRA, omitimos a lista de campos, mas informamos explicitamente o valor de NR_OBRA = "5". De outra maneira, mesmo apesar do atributo AUTO_INCREMENT dessa coluna, esse comando resultaria em um erro.

```
mysql> INSERT INTO OBRA VALUES (6,"A HISTORIA PERDIDA E  
REVELADA","REL");
```

Query OK, 1 row affected (0.02 sec)

No comando SQL acima, o valor de NR_OBRA foi informado sem aspas (NR_OBRA = 6). Isso somente é possível porque a coluna NR_OBRA foi definida na criação da tabela OBRA como tipo **INTEGER** (inteiro). Valores numéricos dispensam aspas.

```
mysql> INSERT INTO OBRA VALUES (6,"O CONDE DE  
MONTECRISTO","LIT");
```

ERROR 1062 (23000): Duplicate entry '6' for key 'PRIMARY'

A execução do comando acima resultou em um erro porque o valor da chave primária NR_OBRA foi duplicado. A OBRA de TITULO = "O CONDE DE MONTECRISTO" não pode ter NR_OBRA = 6, pois já existe um registro com esse mesmo valor.

```
mysql> INSERT INTO OBRA VALUES (7,"O CONDE DE  
MONTECRISTO","LIT");
```

Query OK, 1 row affected (0.00 sec)

Agora, sim, a OBRA de TITULO = "O CONDE DE MONTECRISTO" pode ser inserida, pois NR_OBRA = 7 é um valor inédito nessa tabela.

```
mysql> INSERT INTO OBRA VALUES (8, "SISTEMAS DE BANCO DE DADOS", "DID");
```

Query OK, 1 row affected (0.02 sec)

O mesmo é válido para a OBRA de TITULO = "SISTEMAS DE BANCO DE DADOS". Conforme demonstrado pela consulta da Figura 4.12, a tabela OBRA agora contém sete registros (*records*) ou linhas (*rows*) cadastrados.

```
mysql> SELECT * FROM OBRA;
```

NR_OBRA	TITULO	ID_GENERO
1	HELENA	LIT
2	DOM CASMURRO	LIT
4	TEORIA DO CAOS	DID
5	OS TRÊS MOSQUETEIROS	LIT
6	A HISTORIA PERDIDA E REVELADA	REL
7	O CONDE DE MONTECRISTO	LIT
8	SISTEMAS DE BANCO DE DADOS	DID

7 rows in set (0.02 sec)

Figura 4.12: Resultado do comando "SELECT * FROM OBRA;" após inserções

Fonte: Elaborada pelo autor.

O comando SQL abaixo demonstra a inserção de um registro na tabela USUARIO com um campo do tipo **DATE** (data). O dado relativo à data de nascimento é informado no formato "yyyy-mm-dd", ou seja, quatro dígitos para o ano, traço, dois dígitos para o mês, traço e dois dígitos para o dia.

```
mysql> INSERT INTO USUARIO VALUES (100, "ALBERTO ROBERTO RAVONE", "M", "1979-12-20", "AV. PATOS, 331 APT.201, VILA SANTANA");
```

Query OK, 1 row affected (0.00 sec)

1. Com base nas tabelas criadas na atividade anterior, escreva comandos SQL para a inserção dos seguintes registros.



NR_CLIENTE	NOME	CNPJ
1	Casas Gonçalves Ltda	111.222.445-8
2	Sisne Informática	212.331.890-0

NR_PEDIDO	DATA_PEDIDO	NR_CLIENTE
1001	22-09-2009	1
1023	03-10-2009	2
1024	03-10-2009	1

Figura 4.13: Dados a serem inseridos nas tabelas do BD

Fonte: Elaborado pelo autor

2. Explique por que, uma vez feitas as inserções de registros do item anterior, as seguintes inserções seriam impossíveis:

NR_PEDIDO	DATA_PEDIDO	NR_CLIENTE
1001	30-01-2010	1
1099	31-01-2010	9

Figura 4.14: Dados impossíveis de serem inseridos

Fonte: Elaborado pelo autor

4.6 Alteração de registro em tabela

A sintaxe para a alteração de um registro é:

```
UPDATE <nome-da-tabela> SET <alteração> WHERE <condição>;
```

Assim, por exemplo, se quisermos alterar o identificador de gênero (ID_GENERO) da OBRA de número 4 (NR_OBRA = 4) para que deixe de ser "DID" (Didático e Técnico) para ser "CIE" (Divulgação Científica), podemos escrever o seguinte comando:

```
mysql> UPDATE OBRA SET ID_GENERO="CIE" WHERE NR_OBRA = 4;
```

Query OK, 1 row affected (0.03 sec)

Rows matched: 1 Changed: 1 Warnings: 0

O SGBD retornou a seguinte mensagem **"Query OK"** (comando bem-sucedido), **"1 row affected (0.03 sec)"** (uma linha foi afetada e levou 0,03

segundos para ser executado), "**Rows matched: 1 Changed: 1 Warnings: 0**" (Linhas encontradas= 1, alteradas= 1 e Advertências= 0). Por essa razão sabemos que a alteração foi efetuada. Mesmo assim, executamos uma consulta SQL para comprovação (Figura 4.15).

```
mysql> SELECT * FROM OBRA;
```

NR_OBRA	TITULO	ID_GENERO
1	HELENA	LIT
2	DOM CASMURRO	LIT
4	TEORIA DO CAOS	CIE
5	OS TRÊS MOSQUETEIROS	LIT
6	A HISTORIA PERDIDA E REVELADA	REL
7	O CONDE DE MONTECRISTO	LIT
8	SISTEMAS DE BANCO DE DADOS	DID

7 rows in set (0.00 sec)

Figura 4.15: Resultado do comando "SELECT * FROM OBRA;" após alteração de um registro

Fonte: Elaborada pelo autor

Conforme demonstrado pela consulta acima, o valor do campo (coluna) ID_GENERO para a linha em que NR_OBRA = 4 é agora "CIE".

Para afetar uma única linha (registro) da tabela usamos uma condição baseada em sua **chave primária** (WHERE NR_OBRA = 4), onde apenas o registro com NR_OBRA igual a 4 deveria ser alterado. Como uma chave primária não pode conter valores duplicados, temos a certeza de que apenas dois resultados são possíveis:



1. Um único registro será alterado (caso exista um registro com NR_OBRA = 4);
2. Nenhum registro será alterado (caso não exista um registro com NR_OBRA = 4).

```
mysql> UPDATE OBRA SET ID_GENERO="ABC" WHERE NR_OBRA = 6;
```

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`Biblioteca/Obra`, CONSTRAINT `obra_ibfk_1` FOREIGN KEY (`ID_GENERO`) REFERENCES `GENERO` (`ID_GENERO`))
```

Acima temos um exemplo de comando **UPDATE** malsucedido. Tentou-se alterar o conteúdo de ID_GENERO para "ABC" e não existe uma coluna com esse valor na tabela GENERO. O SGBD identificou uma tentativa de violação da **integridade referencial** e "abortou" a execução desse comando.

```
mysql> UPDATE OBRA SET ID_GENERO="CIE" WHERE NR_OBRA = 6;
```

Query OK, 1 row affected (0.02 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Note que, ao mudarmos o valor de "ABC" para "CIE", o comando funcionou. Agora, a OBRA de número 6 apresenta ID_GENERO igual a "CIE" em vez de "REL".



Cuidado: a utilização do comando UPDATE sem a cláusula WHERE resultará na alteração de todos os registros da tabela. Por exemplo, o comando "UPDATE OBRA SET ID_GENERO = "LIT";" acabaria por alterar todos os registros da tabela OBRA que passariam a ter o gênero "literatura".

A cláusula SET do comando UPDATE pode ser empregada para alterar mais de uma coluna simultaneamente.

```
mysql> UPDATE obra SET ID_GENERO="LIT", TITULO =  
-> "ALICE NO PAÍS DAS MARAVILHAS"  
-> WHERE NR_OBRA = 5;
```

Query OK, 1 row affected (0.02 sec)

Rows matched: 1 Changed: 1 Warnings: 0

No comando anterior duas colunas foram alteradas simultaneamente: ID_GENERO e TITULO. A cláusula WHERE pode trazer mais de um argumento para as condições.

```
mysql> UPDATE OBRA SET ID_GENERO="CIE"  
-> WHERE NR_OBRA > 5 AND ID_GENERO = "DID";
```

Query OK, 1 row affected (0.02 sec)

Rows matched: 1 Changed: 1 Warnings: 0

No comando acima serão alterados todos os registros que satisfaçam simultaneamente as duas condições: (1ª) número da obra maior que 5 (NR_OBRA > 5); e (2ª) o ID_GENERO igual a "DID". O operador **AND** exige que ambas as condições sejam verdadeiras. Um registro com apenas uma das condições verdadeiras não será alterado.


```
mysql> UPDATE OBRA SET ID_GENERO="DID"  
-> WHERE NR_OBRA <= 5 OR ID_GENERO = "CIE";
```

Query OK, 6 row affected (0.02 sec)

Rows matched: 6 Changed: 6 Warnings: 0

Novamente fornecemos duas condições: (1) número da obra menor ou igual a 5, ou (2) ID_GENERO igual a "CIE". O operador OR aceita que ambas as condições sejam verdadeiras ou apenas uma delas. Portanto, ainda que a obra seja nº 290 (1ª condição falsa), se o ID_GENERO for igual a "CIE" (2ª condição verdadeira), a alteração será efetuada no registro. Da mesma maneira, se a obra for nº 2 (1ª condição verdadeira) e ID_GENERO = "REL" (2ª condição falsa), o registro também será alterado. Por outro lado, se NR_OBRA = 78 e ID_GENERO= "LIT" (sendo falsas as duas condições), o registro **não** será alterado.

Com base nas tabelas (Figura 4.16), informe quantos registros (linhas) serão alterados pelos seguintes comandos.



ALUNO			
MATRICULA	NOME	SEXO	ID_CURSO
1	Adão Montenegro	M	INF
2	Sônia Lopes	F	SAN
3	Mariana Andrade Lopes	F	INF
4	Sérgio Gomes Xavier	M	COM
5	Tâmara Montenegro	F	ECO

CURSO	
ID_CURSO	NOME_CURSO
COM	Contabilidade
ECO	Economia
INF	Informática
SAN	Saneamento Ambiental

Figura 4.16: Tabelas ALUNO e CURSO de uma escola

Fonte: Elaborada pelo autor

- a) UPDATE aluno SET NOME = "Adão Alves Montenegro" WHERE MATRICULA = 1;
- b) UPDATE curso SET NOME_CURSO = "Ciências Econômicas" WHERE ID_CURSO = "ECO";

- c) UPDATE ALUNO SET ID_CURSO = "CON" WHERE SEXO = "F";
- d) UPDATE CURSO SET NOME_CURSO = "CURSO DE VERÃO";
- e) UPDATE aluno SET ID_CURSO = "ECO" WHERE MATRICULA >= 3 OR ID_CURSO = "CON";>

4.7 Exclusão de registro em tabela

A sintaxe SQL para a exclusão de registros em uma tabela é:

```
DELETE FROM <NOME-DA-TABELA> WHERE <CONDIÇÃO>;
```

Após as palavras-chave "**DELETE FROM**" acrescentamos o nome da tabela de onde pretendemos excluir registros. A cláusula "**WHERE**" é opcional. Sua omissão levará à exclusão de todos os registros da tabela em questão. Porém, o uso da cláusula "**WHERE**" permite especificar melhor os registros que desejamos excluir.

```
mysql> DELETE FROM GENERO WHERE ID_GENERO="AAJ";
```

```
Query OK, 1 row affected (0.02 sec)
```

O comando acima exclui o registro da tabela GENERO cujo ID_GENERO é "**AAJ**". Como ID_GENERO corresponde à chave primária da tabela, temos certeza de que apenas um registro será excluído (caso exista) ou que nenhum registro será excluído (se não houver linha com ID_GENERO = "AAJ").

```
mysql> DELETE FROM GENERO WHERE ID_GENERO="LIT";
```

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`biblioteca`.`obra`, CONSTRAINT `obra_ibfk_1` FOREIGN KEY (`ID_GENERO`) REFERENCES `genero` (`ID_GENERO`))
```

Um erro aconteceu ao se tentar apagar um registro da tabela GENERO com a coluna ID_GENERO igual a "LIT". De fato esse registro existe na tabela GENERO e também na tabela OBRA, em que quatro dos sete registros inseridos têm ID_GENERO = "LIT". Como estabelecemos um relacionamento entre GENERO e OBRA através da chave estrangeira (**FOREIGN KEY**) ID_GENERO em OBRA e usamos a opção de **exclusão restrita** ("**ON DELETE RESTRICT**"), o SGBD nos impede de excluir qualquer registro de GENERO (Figura 4.17) que esteja relacionado a pelo menos um registro em OBRA como no exemplo acima, com quatro

registros "LIT" em OBRA. Caso tivéssemos optado por **exclusão em cascata** ("**ON DELETE CASCADE**") o SGBD não impediria a exclusão do registro, porém acabaria por apagar também todos os registros de OBRA relacionados a esse registro de GENERO apagado.

```
mysql> SELECT * FROM GENERO;
```

ID_GENERO	DESCRICAO
CIE	DIVULGACAO CIENTIFICA
DID	DIDATICO E TECNICO
LIT	LITERATURA
REL	RELIGIOSO

4 rows in set (0.00 sec)

Figura 4.17: Conteúdo da tabela GENERO após uma tentativa de exclusão de registro

Fonte: Elaborada pelo autor

De acordo com a consulta SQL, o registro com ID_GENERO = "LIT" não foi excluído. Mas, o registro de GENERO com ID_GENERO = "AAJ" foi.

Suponha que a tabela OBRA esteja com os registros da Figura 4.15 (desconsiderando todas as alterações feitas após sua exibição com o comando SELECT). Caso executemos o seguinte comando SQL:

```
mysql> DELETE FROM OBRA WHERE ID_GENERO="LIT";
Query OK, 4 rows affected (0.02 sec)
```

Teremos quatro registros excluídos com ID_GENERO = "LIT" (Figura 4.18).

```
mysql> SELECT * FROM OBRA;
```

NR_OBRA	TITULO	ID_GENERO
4	TEORIA DO CAOS	DID
6	A HISTORIA PERDIDA E REVELADA	REL
8	SISTEMAS DE BANCO DE DADOS	DID

3 rows in set (0.00 sec)

Figura 4.18: Conteúdo da tabela OBRA após exclusão de registros

Fonte: Elaborada pelo autor

Observe que a tabela passou a apresentar apenas três registros em vez de sete. Nenhum dos registros remanescentes possui ID_GENERO = "LIT".

Agora sim poderíamos excluir o registro de GENERO com ID_GENERO = "LIT". Nenhum erro aconteceria, pois já não existem seus registros na tabela OBRA.



```
mysql> DELETE FROM OBRA;  
Query OK, 3 rows affected (0.02 sec)
```

O comando SQL acima excluiu todos os registros da tabela OBRA. Afinal, não informamos qualquer cláusula "WHERE". Porém, a estrutura da tabela OBRA permanece intacta e pronta para receber novos registros. Tanto que uma consulta SQL nos revela uma tabela OBRA vazia (*empty*):

```
mysql> SELECT * FROM OBRA;
```

```
Empty set (0.00 sec)
```



Note que o comando DELETE é diferente do comando DROP. "DROP TABLE" destrói a tabela com todos os registros que eventualmente possua. "DELETE FROM" apaga apenas os registros e mantém intacta a estrutura da tabela: os registros são apagados, mas a tabela continua a existir.

Assim como no comando UPDATE, a cláusula "WHERE" do comando DELETE pode apresentar mais de um argumento para definir a condição de exclusão.

```
mysql> DELETE FROM USUARIO WHERE MATRICULA < 1015 OR  
-> DATA_NASCIMENTO = "1985-03-25";
```

```
Query OK, 1 row affected (0.02 sec)
```

Acima, por exemplo, serão excluídos apenas os registros de USUARIO com MATRICULA menor que 1015 (1º argumento) ou com DATA_NASCIMENTO igual a 25 de março de 1985 (2º argumento). O uso do operador OR faz com que baste a satisfação dos dois ou de um argumento para ocorrer a exclusão.

Resumo

Nesta aula abordamos os comandos da linguagem de definição e manipulação de dados da SQL, que surgiu na década de 1970 e tornou-se padrão para BDs nos anos 1980. Damos especial destaque para os comandos de criação, alteração e exclusão de tabelas; definição de relacionamentos entre tabelas; além de comandos SQL para a inserção, alteração e exclusão de registros em tabelas. Também utilizamos comandos para criar BDs e efetuar consultas simples. Entretanto a próxima aula será exclusivamente para a geração de consultas básicas e avançadas por meio da SQL.

Atividades de aprendizagem

Qual o risco que corremos quando esquecemos a cláusula WHERE em uma instrução DELETE?

Aula 5 – Consultas SQL

Objetivos

Recuperar dados armazenados em tabelas por meio de consultas SQL.

Definir colunas cujos dados serão visualizadas na consulta.

Consultar simultaneamente informações armazenadas em várias tabelas relacionadas.

Estabelecer “filtros” que separem os registros relevantes para o contexto da consulta daqueles que não o são.

Utilizar funções especiais disponíveis.

Até aqui (1) criamos um BD, (2) construímos tabelas dentro desse BD, (3) inserimos e alteramos registros nessas mesmas tabelas e delas os excluímos. Porém, de nada adianta “escrever” dados em tabelas se não sabemos como “ler” ou recuperar esses dados. Isso é feito por meio de consultas SQL.

5.1 Consultas básicas

A estrutura básica de uma expressão de consulta SQL consiste em duas cláusulas: **SELECT** e **FROM**.

```
SELECT <lista-de-colunas> FROM <lista-de-tabelas>;
```

A cláusula **SELECT** serve para definir as colunas que terão seus valores exibidos na consulta e a ordem de apresentação dessas colunas. A cláusula **FROM**, por sua vez, serve para indicar as tabelas de origem das referidas colunas.

A modalidade mais simples de consulta SQL foi explorada na aula anterior. Nela a lista de colunas é substituída por um asterisco (“*”) e a de tabelas se resume a uma única tabela. O asterisco indica que todas as colunas das tabelas do **FROM** serão exibidas.



Após executar o MySQL e informar a senha (root), devemos “abrir” o BD em que estamos trabalhando (BIBLIOTECA). Para tanto, execute o comando:

```
mysql> USE BIBLIOTECA;  
Database changed  
mysql>
```

A consulta da Figura 5.1 exibe todos os registros da tabela GENERO, com suas colunas (antes das modificações e exclusões).

```
mysql> SELECT * FROM GENERO;  
+-----+-----+  
| ID_GENERO | DESCRICAO |  
+-----+-----+  
| AAJ      | AUTOAJUDA |  
| CIE      | DIVULGACAO CIENTIFICA |  
| DID      | DIDATICO  |  
| LIT      | LITERATURA |  
| REL      | RELIGIOSO |  
+-----+-----+  
5 rows in set (0.00 sec)
```

Figura 5.1: Resultado da consulta de todos os registros da tabela GENERO

Fonte: Elaborada pelo autor

O mesmo vale para a consulta da Figura 5.2, que exibe todos os registros da tabela OBRA sem erros, antes das modificações e exclusões.

```
mysql> select * from obra;  
+-----+-----+-----+  
| NR_OBRA | TITULO | ID_GENERO |  
+-----+-----+-----+  
| 1 | HELENA | LIT |  
| 2 | DOM CASMURRO | LIT |  
| 3 | TEORIA DO CAOS | DID |  
| 4 | OS TRÊS MOSQUETEIROS | LIT |  
| 5 | A HISTORIA PERDIDA E REVELADA | REL |  
| 6 | O CONDE DE MONTECRISTO | LIT |  
| 7 | SISTEMAS DE BANCO DE DADOS | DID |  
+-----+-----+-----+  
7 rows in set (0.00 sec)
```

Figura 5.2: Resultado da consulta de todos os registros da tabela OBRA

Fonte: Elaborada pelo autor

Observe que a consulta da Figura 5.3 restringe a quantidade de colunas da tabela OBRA que devem ser exibidas e altera a ordem de exibição dessas colunas.

```
mysql> SELECT TITULO, NR_OBRA FROM OBRA;
```

TITULO	NR_OBRA
HELENA	1
DOM CASMURRO	2
TEORIA DO CAOS	3
OS TRÊS MOSQUETEIROS	4
A HISTORIA PERDIDA E REVELADA	5
O CONDE DE MONTECRISTO	6
SISTEMAS DE BANCO DE DADOS	7

7 rows in set (0.00 sec)

Figura 5.3: Resultado da consulta de alguns registros da tabela OBRA

Fonte: Elaborada pelo autor.

Note que a coluna ID_GENERO foi omitida na cláusula SELECT e seus dados não aparecem no resultado da consulta. Os dados de TITULO também são apresentados antes de NR_OBRA, conforme sequência determinada pelo SELECT.

5.2 Consultas com cláusula ORDER BY

A cláusula ORDER BY objetiva estabelecer a ordem de apresentação dos registros em uma consulta SQL. Ela deve ser incluída no código da consulta após as cláusulas SELECT e FROM. ORDER BY normalmente é a última cláusula de uma consulta.

```
SELECT <LISTA-COLUNAS>  
FROM <LISTA-TABELAS>  
ORDER BY <LISTA-COLUNAS>;
```

Na consulta realizada na Figura 5.4, os registros de OBRA foram apresentados em ordem crescente de NR_OBRA (**chave primária** da tabela OBRA) porque sem a cláusula ORDER BY a consulta geralmente exibe os dados conforme a ordem crescente da **chave primária**. Repare que os registros da Figura 5.4 estão ordenados alfabeticamente por TITULO.

```
mysql> SELECT TITULO, NR_OBRA
-> FROM OBRA
-> ORDER BY TITULO;
```

TITULO	NR_OBRA
A HISTORIA PERDIDA E REVELADA	5
DOM CASMURRO	2
HELENA	1
O CONDE DE MONTECRISTO	6
OS TRÊS MOSQUETEIROS	4
SISTEMAS DE BANCO DE DADOS	7
TEORIA DO CAOS	3

7 rows in set (0.00 sec)

Figura 5.4: Resultado de uma consulta com a Cláusula "ORDER BY"

Fonte: Elaborada pelo autor

Na consulta da Figura 5.5 adicionamos o operador DESC ao final da lista de ORDER BY. O efeito dessa mudança foi a inversão da ordem de apresentação dos registros. Com o operador DESC, os registros passam a ser apresentados em ordem decrescente. "A HISTÓRIA PERDIDA E REVELADA" passou a ser o último registro apresentado.

```
mysql> SELECT TITULO, NR_OBRA
-> FROM OBRA
-> ORDER BY TITULO DESC;
```

TITULO	NR_OBRA
TEORIA DO CAOS	3
SISTEMAS DE BANCO DE DADOS	7
OS TRÊS MOSQUETEIROS	4
O CONDE DE MONTECRISTO	6
HELENA	1
DOM CASMURRO	2
A HISTORIA PERDIDA E REVELADA	5

7 rows in set (0.06 sec)

Figura 5.5: Resultado de uma consulta com a Cláusula "ORDER BY ... DESC"

Fonte: Elaborada pelo autor

Na consulta da Figura 5.6, utilizamos mais de uma coluna na cláusula ORDER BY: ID_GENERO e TITULO. Significa que os registros são primeiro ordenados por ID_GENERO e, depois, por TITULO. Note que os registros com ID_GENERO = "DID" aparecem todos no início da consulta. Dentre estes, "SISTEMAS DE BANCO DE DADOS" aparece antes de "TEORIA DO CAOS". Portanto, dentro do conjunto de registros "DID" ocorreu um segundo ordenamento, por TITULO.

```
mysql> SELECT ID_GENERO, TITULO, NR_OBRA
-> FROM OBRA
-> ORDER BY ID_GENERO, TITULO;
```

ID_GENERO	TITULO	NR_OBRA
DID	SISTEMAS DE BANCO DE DADOS	7
DID	TEORIA DO CAOS	3
LIT	DOM CASMURRO	2
LIT	HELENA	1
LIT	O CONDE DE MONTECRISTO	6
LIT	OS TRÊS MOSQUETEIROS	4
REL	A HISTORIA PERDIDA E REVELADA	5

7 rows in set (0.00 sec)

Figura 5.6: Resultado de uma consulta ordenada primeiro por ID_GENERO

Fonte: Elaborada pelo autor

Ao invertermos a sequência de colunas da cláusula ORDER BY, fazendo o TITULO vir antes de ID_GENERO, alteramos também o ordenamento dos registros. Agora, os registros são ordenados primeiro com base nos valores da coluna TITULO e somente então nos valores da coluna ID_GENERO (Figura 5.7).

```
mysql> SELECT *
-> FROM OBRA
-> ORDER BY TITULO, ID_GENERO;
```

NR_OBRA	TITULO	ID_GENERO
5	A HISTORIA PERDIDA E REVELADA	REL
2	DOM CASMURRO	LIT
1	HELENA	LIT
6	O CONDE DE MONTECRISTO	LIT
4	OS TRÊS MOSQUETEIROS	LIT
7	SISTEMAS DE BANCO DE DADOS	DID
3	TEORIA DO CAOS	DID

7 rows in set (0.00 sec)

Figura 5.7: Resultado de uma consulta ordenada primeiro por TITULO

Fonte: Elaborada pelo autor

Com os registros ordenados por MATRICULA e DATA_EMPRESTIMO, podemos visualizar todos os empréstimos efetuados por um USUARIO (dados foram acrescentados aleatoriamente nas tabelas EDITORA, LIVRO, MOVIMENTACAO, etc. para os exemplos de movimentação, usando os comandos ensinados na seção 4.5 da Aula 4). Primeiro visualizamos os empréstimos do USUARIO 103 e somente depois visualizamos os do USUARIO 104 (Figura 5.8). Nenhum outro USUARIO cadastrado efetuou qualquer empréstimo de livro. Caso contrário, seus registros também seriam exibidos próximos uns dos outros.

```
mysql> SELECT MATRICULA, DATA_EMPRESTIMO, NR_LIVRO, MULTA
-> FROM MOVIMENTACAO
-> ORDER BY MATRICULA, DATA_EMPRESTIMO;
```

MATRICULA	DATA_EMPRESTIMO	NR_LIVRO	MULTA
103	2010-01-30	2	40.00
103	2010-02-10	7	0.00
103	2010-02-10	8	30.00
103	2010-02-10	1	0.00
104	2010-03-25	2	80.00
104	2010-03-25	5	0.00
104	2010-03-25	7	0.00

7 rows in set (0.00 sec)

Figura 5.8: Resultado da consulta de empréstimos

Fonte: Elaborada pelo autor.

Para cada conjunto de registros de uma mesma MATRICULA na tabela de MOVIMENTACAO, os registros são ordenados por DATA_EMPRESTIMO. Assim fica fácil perceber que o USUARIO com MATRICULA 103 fez um empréstimo em 30/janeiro/2010 e três empréstimos em 10/fevereiro/2010. Também podemos apurar sem maiores esforços que a MATRICULA = 104 fez três empréstimos na data de 25/março/2010.



Conforme as tabelas da Figura 5.9, descreva o comando SQL para as consultas que seguem.

MEDICO	
ID_MEDICO	inteiro
NOME	texto (40)
COD_ESPECIALIDADE	texto (3)
SEXO	texto (1)

ESPECIALIDADE	
COD_ESPECIALIDADE	texto (3)
DESCRICAO	texto (35)

AGENDA	
DATA	data
ID_MEDICO	inteiro
ID_CLIENTE	inteiro
ID_CONVENIO	numero
VALOR_CLIENTE	moeda
VALOR_CONVENIO	moeda

CONVENIO	
ID_CONVENIO	inteiro
NOME	texto (60)

CLIENTE	
ID_CLIENTE	inteiro
NOME	texto (40)
SEXO	texto (1)
NASCIMENTO	data
TELEFONE	texto (16)

Figura 5.9: Tabelas de um BD de consultas médicas

Fonte: Elaborada pelo autor

- a) Exibir o nome e o código da especialidade para todos os médicos e ordenar o resultado por nome do médico em ordem alfabética decrescente.
- b) Exibir o nome e a data de nascimento para todos os clientes e ordenar o resultado por data de nascimento.

5.3 Consultas com cláusula WHERE

A cláusula WHERE filtra registros de consultas, permitindo-nos exibir alguns registros enquanto deixamos outros de fora da consulta. Sua sintaxe básica é:

```
SELECT <LISTA-COLUNAS> FROM <LISTA-TABELAS> WHERE <CONDIÇÕES>;
```

A cláusula WHERE deve estar sempre após o SELECT e FROM. Quando houver ORDER BY o WHERE deverá ser posto antes dele. Assim, teríamos a seguinte sintaxe:

```
SELECT <LISTA-COLUNAS>  
FROM <LISTA-TABELAS>  
WHERE <CONDIÇÕES>  
ORDER BY <LISTA-COLUNAS>;
```

A consulta SQL da Figura 5.10 lista os registros da tabela OBRA, com todas as suas colunas onde ID_GENERO = "LIT".

```
mysql> SELECT *  
-> FROM OBRA  
-> WHERE ID_GENERO = "LIT";
```

NR_OBRA	TITULO	ID_GENERO
1	HELENA	LIT
2	DOM CASMURRO	LIT
4	OS TRÊS MOSQUETEIROS	LIT
6	O CONDE DE MONTECRISTO	LIT

```
4 rows in set (0.11 sec)
```

Figura 5.10: Resultado da consulta de registros da tabela OBRA limitados por gênero

Fonte: Elaborada pelo autor

Já a Figura 5.11 resultou apenas no registro de TITULO= "DOM CASMURRO".

```
mysql> SELECT * FROM OBRA
-> WHERE TITULO = "DOM CASMURRO";
```

NR_OBRA	TITULO	ID_GENERO
2	DOM CASMURRO	LIT

```
1 row in set (0.00 sec)
```

Figura 5.11: Resultado da consulta de registros da tabela OBRA limitado por título

Fonte: Elaborada pelo autor

Porém, se nossa cláusula WHERE fosse TITULO = "DOM", nenhum registro seria apresentado, pois o operador "=" exige que o valor do campo seja exatamente igual ao do argumento de pesquisa ("DOM"). Como nenhum registro satisfaz essa condição, a consulta resulta em um conjunto vazio (*Empty set*).

```
mysql> SELECT * FROM OBRA WHERE TITULO = "DOM";
```

Empty set (0.00 sec)

Para fazermos uma consulta por valor parcial de uma coluna do tipo VARCHAR ou CHAR, devemos utilizar o operador LIKE no lugar do "=".

A consulta da Figura 5.12 fez uma busca por TITULO que comece com "A HISTORIA". O caractere curinga "%" indica que qualquer conteúdo pode estar nessa posição. Portanto, o argumento "A HISTORIA%" indica um conteúdo apenas começado com "A HISTORIA".

```
mysql> SELECT * FROM OBRA
-> WHERE TITULO LIKE "A HISTORIA%";
```

NR_OBRA	TITULO	ID_GENERO
5	A HISTORIA PERDIDA E REVELADA	REL

```
1 row in set (0.01 sec)
```

Figura 5.12: Resultado da consulta de registros da tabela OBRA começados por "A Historia"

Fonte: Elaborada pelo autor

Se pretendêssemos consultar registros de OBRA com TITULO terminado com "A HISTORIA" bastaria alterar a posição do caractere curinga:

```
mysql> SELECT * FROM OBRA
-> WHERE TITULO LIKE "%A HISTORIA";
```

Empty set (0.00 sec)

Nenhum registro com TITULO terminado com "A HISTÓRIA" foi encontrado. A consulta retornou como resposta um conjunto vazio (*Empty set*).

Uma pesquisa mais ampla pode ser feita empregando duas vezes o caractere curinga. Na consulta da Figura 5.13, procuramos por registros da tabela OBRA em que TITULO tenha "%SISTEMA%" em qualquer parte de seu conteúdo.

```
mysql> SELECT ID_GENERO, TITULO, NR_OBRA
-> FROM OBRA
-> WHERE TITULO LIKE "%SISTEMA%";
```

ID_GENERO	TITULO	NR_OBRA
DID	SISTEMAS DE BANCO DE DADOS	7

1 row in set (0.01 sec)

Figura 5.13: Resultado de consulta com dois caracteres curingas

Fonte: Elaborada pelo autor

Para a Figura 5.14, o critério de filtragem dos registros baseou-se na coluna NR_OBRA. Essa consulta trouxe os registros de OBRA com valor de NR_OBRA superior a 3.

```
mysql> SELECT * FROM OBRA
-> WHERE NR_OBRA > 3;
```

NR_OBRA	TITULO	ID_GENERO
4	OS TRÊS MOSQUETEIROS	LIT
5	A HISTORIA PERDIDA E REVELADA	REL
6	O CONDE DE MONTECRISTO	LIT
7	SISTEMAS DE BANCO DE DADOS	DID

4 rows in set (0.02 sec)

Figura 5.14: Resultado de consulta com NR_OBRA > 3

Fonte: Elaborada pelo autor

A consulta da Figura 5.15 exibe os registros de OBRA em que a coluna NR_OBRA possui valor maior que 2 e menor que 6. Como o operador AND foi utilizado, os dois critérios devem ser satisfeitos simultaneamente.

```
mysql> SELECT * FROM OBRA
-> WHERE NR_OBRA > 2 AND NR_OBRA < 6
-> ORDER BY TITULO;
```

NR_OBRA	TITULO	ID_GENERO
5	A HISTORIA PERDIDA E REVELADA	REL
4	OS TRÊS MOSQUETEIROS	LIT
3	TEORIA DO CAOS	DID

3 rows in set (0.02 sec)

Figura 5.15: Resultado de consulta com NR_OBRA entre 2 e 6

Fonte: Elaborada pelo autor.

Em algumas situações, é interessante que a consulta retorne um único registro. Nesses casos devemos usar a chave primária na cláusula WHERE. Foi exatamente o que fizemos na Figura 5.16. O critério de filtragem é NR_OBRA = 7. Como essa coluna é a chave primária da tabela OBRA, o resultado dessa consulta somente poderia ser um único registro ou nenhum.

```
mysql> SELECT * FROM OBRA WHERE NR_OBRA = 7;
```

NR_OBRA	TITULO	ID_GENERO
7	SISTEMAS DE BANCO DE DADOS	DID

```
1 row in set (0.00 sec)
```

Figura 5.16: Resultado de consulta com NR_OBRA igual a 7

Fonte: Elaborada pelo autor



Utilizando as mesmas tabelas do exercício anterior (BD para Agendamento de Consultas Médicas), descreva o comando SQL para as seguintes consultas:

- Exibir nome e data de nascimento para todo cliente do sexo feminino (F).
- Exibir todas as informações dos clientes cujo nome termine com Silva.
- Exibir nome e telefone de todo cliente que nasceu em 17 de maio de 1969 e é do sexo masculino (M).
- Exibir nome de todo médico do sexo feminino ou especialidade de geriatria (GER).

5.4 Consultas com funções e agrupamento

A linguagem SQL conta com várias funções para consultas. Existe, por exemplo, a função COUNT() que retorna a contagem de registros como mostra a Figura 5.17 - na tabela GENERO existem cinco registros ou linhas.

```
mysql> SELECT COUNT(*) FROM GENERO;
```

COUNT(*)
5

```
1 row in set (0.06 sec)
```

Figura 5.17: Resultado de consulta de contagem de registros

Fonte: Elaborada pelo autor

Na consulta da Figura 5.18 utilizamos **AS** como forma de alterar o cabeçalho da coluna que deixou de ser "COUNT(*)" e passou a ser "TOTAL_REGISTROS". Isso pode ser feito com qualquer coluna da cláusula SELECT. A consulta abaixo tenta apurar a quantidade de registros de OBRA para cada diferente valor de ID_GENERO existente na tabela.

```
mysql> SELECT COUNT(*) AS TOTAL_REGISTROS
-> FROM GENERO;
+-----+
| TOTAL_REGISTROS |
+-----+
|                5 |
+-----+
1 row in set (0.00 sec)
```

Figura 5.18: Resultado de consulta de contagem de registros com título

Fonte: Elaborada pelo autor

No comando a seguir:

```
mysql> SELECT ID_GENERO, COUNT(*) AS TOTAL_OBRAS FROM OBRA;
```

ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(), COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

Falha em razão de misturarmos pelo menos uma coluna com a função sem utilizar a cláusula GROUP BY. Anteriormente, não houve necessidade da cláusula GROUP BY. Porém, se a referida função estiver acompanhada de pelo menos uma coluna no SELECT, a linguagem SQL exige a inclusão do GROUP BY acompanhado das colunas que precedem a função SELECT.

A Figura 5.19 agrupou a contagem (GROUP BY) por ID_GENERO. Em consequência, obtemos quantos registros de OBRA existem para cada GENERO.

```
mysql> SELECT ID_GENERO, COUNT(*) AS TOTAL_OBRAS
-> FROM OBRA
-> GROUP BY ID_GENERO;
+-----+-----+
| ID_GENERO | TOTAL_OBRAS |
+-----+-----+
| DID      |            2 |
| LIT      |            4 |
| REL      |            1 |
+-----+-----+
3 rows in set (0.00 sec)
```

Figura 5.19: Contagem de linhas da tabela OBRA agrupadas por gênero

Fonte: Elaborada pelo autor

Num outro exemplo de contagem (Figura 5.20), sabemos que a MATRICULA = 103 efetuou quatro empréstimos de livros, enquanto a MATRICULA = 104 realizou três. Portanto, nosso BD registra sete empréstimos apenas.

```
mysql> SELECT MATRICULA, COUNT(*) AS TOTAL_EMPRESTIMOS
-> FROM MOVIMENTACAO
-> GROUP BY MATRICULA;
```

MATRICULA	TOTAL_OBRAS
103	4
104	3

2 rows in set (0.00 sec)

Figura 5.20: Contagem de empréstimos agrupados por matrícula

Fonte: Elaborada pelo autor

A próxima consulta (Figura 5.21) consegue um detalhamento ainda maior da informação ao combinar duas colunas na cláusula GROUP BY: MATRICULA e DATA_EMPRESTIMO.

```
mysql> SELECT MATRICULA, DATA_EMPRESTIMO, COUNT(*) AS TOTAL_EMPRESTIMOS
-> FROM MOVIMENTACAO
-> GROUP BY MATRICULA, DATA_EMPRESTIMO;
```

MATRICULA	DATA_EMPRESTIMO	TOTAL_OBRAS
103	2010-01-30	1
103	2010-02-10	3
104	2010-03-25	3

3 rows in set (0.00 sec)

Figura 5.21: Contagem de registros agrupados por matrícula e data do empréstimo

Fonte: Elaborada pelo autor

Agora sabemos que, dentre os quatro empréstimos da MATRICULA = 103, um foi feito em 30/janeiro/2010 e três, em 10/fevereiro/2010. Também vemos que todos os três empréstimos da MATRICULA = 104 foram efetuados na mesma data.

Existem ainda outras funções que merecem atenção. A função MAX(), por exemplo, retorna o maior valor existente em uma determinada coluna informada nos parênteses.

No comando da Figura 5.22, a execução da função MAX(MULTA) retorna o maior valor existente na coluna MULTA da tabela MOVIMENTACAO.

```
mysql> SELECT MAX(MULTA) AS MAIOR_MULTA
-> FROM MOVIMENTACAO;
+-----+
| MAIOR_MULTA |
+-----+
|          80.00 |
+-----+
1 row in set (0.00 sec)
```

Figura 5.22: Resultado de consulta do maior valor de multa já aplicada

Fonte: Elaborada pelo autor

Existe também uma função MIN() que retorna o menor valor de uma coluna de tipo numérico (INTEGER, NUMERIC e DECIMAL) como mostra a Figura 5.23 a seguir.

```
mysql> SELECT MIN(MULTA) AS MENOR_MULTA
-> FROM MOVIMENTACAO;
+-----+
| MENOR_MULTA |
+-----+
|          0.00 |
+-----+
1 row in set (0.00 sec)
```

Figura 5.23: Resultado de consulta do menor valor de multa já aplicada

Fonte: Elaborada pelo autor

A função SUM() soma os valores de uma coluna numérica declarada entre os parênteses da função. A expressão SUM(MULTA) efetua a soma de valores da coluna MULTA para todos os registros de MOVIMENTACAO (Figura 5.24).

```
mysql> SELECT SUM(MULTA) AS SOMA_MULTAS
-> FROM MOVIMENTACAO;
+-----+
| SOMA_MULTAS |
+-----+
|          150.00 |
+-----+
1 row in set (0.17 sec)
```

Figura 5.24: Resultado da soma de multas

Fonte: Elaborada pelo autor

Na Figura 5.25, o resultado da soma das multas é agrupado (GROUP BY) por MATRICULA. Assim, temos que o usuário com MATRICULA 103 pagou R\$ 70,00, enquanto o 104 pagou R\$ 80,00. O somatório resulta em R\$ 150,00 – o valor da consulta da Figura anterior (Figura 5.24).

```
mysql> SELECT MATRICULA, SUM(MULTA) AS SOMA_MULTAS
-> FROM MOVIMENTACAO
-> GROUP BY MATRICULA;
```

MATRICULA	SOMA_MULTAS
103	70.00
104	80.00

```
2 rows in set (0.00 sec)
```

Figura 5.25: Resultado da soma de multas agrupada por matrícula

Fonte: Elaborada pelo autor

Também é possível conseguir a média dos valores de uma coluna de tipo numérico com a função `AVG()`, abreviatura de *average* (média em inglês). O resultado da consulta da Figura 5.27 é a apresentação da média das multas, que poderiam ser agrupadas por `MATRICULA`, como na Figura 5.26 a seguir.

```
mysql> SELECT AVG(MULTA) AS MEDIA_MULTA
-> FROM MOVIMENTACAO;
```

MEDIA_MULTA
21.428571

```
1 row in set (0.00 sec)
```

Figura 5.26: Resultado de média das multas

Fonte: Elaborada pelo autor

```
mysql> SELECT MATRICULA, AVG(MULTA) AS MEDIA_MULTA
-> FROM MOVIMENTACAO
-> GROUP BY MATRICULA;
```

MATRICULA	MEDIA_MULTA
103	17.500000
104	26.666667

```
2 rows in set (0.00 sec)
```

Figura 5.27: Resultado de média das multas agrupadas por matrícula

Fonte: Elaborada pelo autor

Existem ainda as funções relativas a colunas do tipo `DATE`. A consulta da Figura 5.28, por exemplo, seleciona a `DATA_NASCIMENTO` e a separa em ano (função `YEAR()`), mês (função `MONTH()`) e dia (função `DAY()`).

```
mysql> SELECT DATA_NASCIMENTO,
-> YEAR(DATA_NASCIMENTO) AS ANO_NASCIMENTO,
-> MONTH(DATA_NASCIMENTO) AS MES_NASCIMENTO,
-> DAY(DATA_NASCIMENTO) AS DIA_NASCIMENTO
-> FROM USUARIO
-> ORDER BY NOME;
```

DATA_NASCIMENTO	ANO_NASCIMENTO	MES_NASCIMENTO	DIA_NASCIMENTO
1992-12-31	1992	12	31
1985-03-25	1985	3	25
1995-08-21	1995	8	21
1992-05-21	1992	5	21
1981-01-17	1981	1	17

5 rows in set (0.19 sec)

Figura 5.28: Resultado da separação das partes que compõem a data de nascimento

Fonte: Elaborada pelo autor

Na Figura 5.29, a função YEAR() foi aplicada para a DATA_NASCIMENTO na cláusula WHERE. Os registros de USUARIO exibidos referem-se apenas aos que possuem ano de nascimento maior que 1990.

```
mysql> SELECT NOME, DATA_NASCIMENTO
-> FROM USUARIO
-> WHERE YEAR (DATA_NASCIMENTO) > 1990
-> ORDER BY NOME;
```

NOME	DATA_NASCIMENTO
MARCELA GONCALVES	1992-12-31
RENAN BELISARIO	1995-08-21
RENATO PEREIRA	1992-05-21

3 rows in set (0.01 sec)

Figura 5.29: Resultado de consulta dos usuários nascidos depois de 1990

Fonte: Elaborada pelo autor

Existem ainda outras funções de data que o SGBD MySQL disponibiliza. Dentre elas há a função CURDATE(), que retorna a data corrente (data atual fornecida pelo sistema). Na consulta da Figura 5.30 calculamos a idade de cada USUARIO. Para tanto, utilizamos a função RIGHT(), além da função CURDATE(). A expressão YEAR(CURDATE()) retorna o ano atual – no caso, 2009. Analogamente, a expressão YEAR(DATA_NASCIMENTO) retorna o ano de nascimento do USUARIO. Note que, na cláusula SELECT, o ano de nascimento é subtraído do ano atual, na expressão:

`(YEAR(CURDATE()) – YEAR(DATA_NASCIMENTO))`

```
mysql> SELECT CURDATE() AS DATA_ATUAL, NOME, DATA_NASCIMENTO,
-> (YEAR(CURDATE()) - YEAR(DATA_NASCIMENTO)) -
-> (RIGHT(CURDATE(),5) < RIGHT(DATA_NASCIMENTO,5)) AS IDADE
-> FROM USUARIO ORDER BY NOME;
```

DATA_ATUAL	NOME	DATA_NASCIMENTO	IDADE
2009-11-26	MARCELA GONCALVES	1992-12-31	16
2009-11-26	MARCIO ZORZANELLI	1985-03-25	24
2009-11-26	RENAN BELISARIO	1995-08-21	14
2009-11-26	RENATO PEREIRA	1992-05-21	17
2009-11-26	SONIA VIEIRA	1981-01-17	28

5 rows in set (0.02 sec)

Figura 5.30: Resultado de cálculo das idades de usuários

Fonte: Elaborada pelo autor

A função `RIGHT()` é usada para retornar uma determinada quantidade de caracteres mais à direita. A expressão **`RIGHT(CURDATE(), 5)`**, por exemplo, retorna os cinco caracteres mais à direita da data corrente. Supondo que esta seja 26 de novembro de 2009 ("2009-11-26"), a referida expressão retornará "11-26". De maneira semelhante, a expressão `RIGHT(DATA_NASCIMENTO)` retornará para o registro do `USUARIO` de nome "Marcela Goncalves" o valor "12-31".

Dessa maneira, o resultado da última coluna da cláusula `SELECT` para o primeiro registro exibido fica sendo o seguinte cálculo:

$$\text{IDADE} = (\text{Year}(\text{Curdate}()) - \text{Year}(\text{Data_Nascimento})) -$$

$$(\text{Right}(\text{Curdate}(), 5) < \text{Rigth}(\text{Data_Nascimento}, 5))$$

$$\text{IDADE} = (\text{Year}("2009-11-26") - \text{Year}("1992-12-31")) -$$

$$(\text{Rigth}("2009-11-26", 5) < \text{Rigth}("1992-12-31", 5))$$

$$\text{IDADE} = (2009 - 1992) - ("11-26" < "12-31")$$

$$\text{IDADE} = (17) - (1)$$

$$\text{IDADE} = 16$$

A idade do `USUARIO` "MARCELA GONCALVES" é 16 anos. A expressão `(RIGHT(CURDATE(),5) < (RIGTH(DATA_NASCIMENTO))` avalia se o segmento "MM_DD" (mês-dia) da data corrente é menor que o da `DATA_NASCIMENTO`. A resposta a essa avaliação pode ser apenas um de dois resultados possíveis: 1 caso a expressão seja verdadeira ou 0 (zero) se a expressão for falsa.

Em nosso exemplo, em um dado momento, a expressão fica resumida a **("11-26" < "12-31")**. Como 26 de novembro é realmente menor que 31 de dezembro, o resultado dessa avaliação é verdadeiro. Então a expressão

retorna **1** que é subtraído de (2009 - 1992). "MARCELA GONCALVES" ainda não completou 17 anos. A data atual é 26/novembro e ela fará aniversário em 31/dezembro. Portanto, por enquanto, ela ainda tem 16 anos (17 - 1).

Porém, se "MARCELA GONCALVES" tivesse outra DATA_NASCIMENTO, digamos "1992-10-03" (3 de outubro de 1992), o resultado da expressão seria:

```
IDADE = (YEAR(CURDATE( )) - YEAR(DATA_NASCIMENTO)) -  
        (RIGHT(CURDATE( ), 5) < (RIGTH(DATA_NASCIMETO,5))
```

```
IDADE = (YEAR("2009-11-26") - YEAR("1992-10-03")) -  
        (RIGTH("2009-11-26",5) < RIGTH("1992-10-03",5))
```

```
IDADE = (2009 - 1992) - ("11-26" < "10-03")
```

```
IDADE = (17) - (0)
```

```
IDADE = 17
```

Nesse caso, "MARCELA GONCALVES" já teria feito aniversário. Então, a expressão ("11-26" < "10-03") resultaria em 0 (falso). Afinal, 26/Nov. não é menor que 3/Out. Assim, "MARCELA" já teria 17 anos completos na data atual.

Ainda usando as tabelas de Agendamento de Consultas Médicas, descreva o comando SQL para as seguintes consultas:



1. Exibir o nome e a data de nascimento para todo cliente nascido no mês de outubro. O resultado da pesquisa deve estar ordenado alfabeticamente por nome do médico.
2. Exibir o ano de nascimento e a quantidade de clientes nascidos nesse mesmo ano.
3. Exibir a soma de valores pagos pelos clientes do médico com identificador 1035 desde 10/dezembro/2009 até o dia de hoje.

5.5 Consultas com mais de uma tabela

Todas as nossas consultas, até aqui, envolveram sempre uma única tabela. Entretanto, na realidade dos profissionais de informática que trabalham com BDs normalmente elaboram-se consultas com mais de uma tabela simultaneamente.

Considere uma consulta em que sejam exibidos simultaneamente o TITULO (coluna de OBRA – Figura 5.31) e a DESCRICAO (coluna de GENERO – Figura 5.32) conforme Figura 5.33.

```
mysql> SELECT * FROM GENERO;
```

ID_GENERO	DESCRICAO
AAJ	AUTOAJUDA
CIE	DIVULGACAO CIENTIFICA
DID	DIDATICO
LIT	LITERATURA
REL	RELIGIOSO

5 rows in set (0.09 sec)

Figura 5.31: Conteúdo da tabela GENERO

Fonte: Elaborada pelo autor

```
mysql> SELECT * FROM OBRA;
```

NR_OBRA	TITULO	ID_GENERO
1	HELENA	LIT
2	DOM CASMURRO	LIT
3	TEORIA DO CAOS	DID
4	OS TRÊS MOSQUETEIROS	LIT
5	A HISTORIA PERDIDA E REVELADA	REL
6	O CONDE DE MONTECRISTO	LIT
7	SISTEMAS DE BANCO DE DADOS	DID

7 rows in set (0.03 sec)

Figura 5.32: Conteúdo da tabela OBRA

Fonte: Elaborada pelo autor


```
mysql> SELECT TITULO, DESCRICAO FROM GENERO, OBRA;
```

TITULO	DESCRICAO
HELENA	AUTOAJUDA
HELENA	DIVULGACAO CIENTIFICA
HELENA	DIDATICO
HELENA	LITERATURA
HELENA	RELIGIOSO
DOM CASHURRO	AUTOAJUDA
DOM CASHURRO	DIVULGACAO CIENTIFICA
DOM CASHURRO	DIDATICO
DOM CASHURRO	LITERATURA
DOM CASHURRO	RELIGIOSO
TEORIA DO CAOS	AUTOAJUDA
TEORIA DO CAOS	DIVULGACAO CIENTIFICA
TEORIA DO CAOS	DIDATICO
TEORIA DO CAOS	LITERATURA
TEORIA DO CAOS	RELIGIOSO
OS TRÊS MOSQUETEIROS	AUTOAJUDA
OS TRÊS MOSQUETEIROS	DIVULGACAO CIENTIFICA
OS TRÊS MOSQUETEIROS	DIDATICO
OS TRÊS MOSQUETEIROS	LITERATURA
OS TRÊS MOSQUETEIROS	RELIGIOSO
A HISTORIA PERDIDA E REVELADA	AUTOAJUDA
A HISTORIA PERDIDA E REVELADA	DIVULGACAO CIENTIFICA
A HISTORIA PERDIDA E REVELADA	DIDATICO
A HISTORIA PERDIDA E REVELADA	LITERATURA
A HISTORIA PERDIDA E REVELADA	RELIGIOSO
O CONDE DE MONTECRISTO	AUTOAJUDA
O CONDE DE MONTECRISTO	DIVULGACAO CIENTIFICA
O CONDE DE MONTECRISTO	DIDATICO
O CONDE DE MONTECRISTO	LITERATURA
O CONDE DE MONTECRISTO	RELIGIOSO
SISTEMAS DE BANCO DE DADOS	AUTOAJUDA
SISTEMAS DE BANCO DE DADOS	DIVULGACAO CIENTIFICA
SISTEMAS DE BANCO DE DADOS	DIDATICO
SISTEMAS DE BANCO DE DADOS	LITERATURA
SISTEMAS DE BANCO DE DADOS	RELIGIOSO

35 rows in set (0.02 sec)

Figura 5.33: Consulta malsucedida de título da obra e seu gênero literário

Fonte: Elaborada pelo autor

Apesar de nenhuma mensagem de erro ser exibida, a consulta a seguir está longe do que classificaríamos como bem-sucedida. Eis um tipo de erro muito perigoso. Se não tivermos atenção, nem mesmo percebemos sua ocorrência. Repare que para cada TITULO de OBRA, nossa consulta estabeleceu uma ligação com toda DESCRICAO de GENERO existente. Assim, por exemplo, “HELENA” aparece classificada como “AUTOAJUDA”, “DIDATICO E TECNICO”, “DIVULGACAO CIENTIFICA”, “LITERATURA” e “RELIGIOSO”. Porém, na prática sabemos que a famosa obra de Machado de Assis não pode ser tudo isso simultaneamente. Em nosso BD ela foi cadastrada como uma OBRA do GENERO “LITERATURA”.



Uma maneira de consertar o erro da consulta anterior é forçar o relacionamento entre as duas tabelas com a cláusula WHERE definindo que a coluna ID_GENERO da tabela GENERO deve ser igual à coluna ID_GENERO da tabela OBRA como na Figura 5.34 a seguir.

```
mysql> SELECT TITULO, DESCRICAO FROM GENERO, OBRA
-> WHERE GENERO.ID_GENERO = OBRA.ID_GENERO
-> ORDER BY TITULO;
```

TITULO	DESCRICAO
A HISTORIA PERDIDA E REVELADA	RELIGIOSO
DOM CASMURRO	LITERATURA
HELENA	LITERATURA
O CONDE DE MONTECRISTO	LITERATURA
OS TRÊS MOSQUETEIROS	LITERATURA
SISTEMAS DE BANCO DE DADOS	DIDATICO
TEORIA DO CAOS	DIDATICO

```
7 rows in set (0.03 sec)
```

Figura 5.34: Consulta bem-sucedida de título da obra e seu gênero literário

Fonte: Elaborada pelo autor

Em GENERO, ID_GENERO é a chave primária e em OBRA ela é uma chave estrangeira usada para relacionar os registros desta tabela com os da outra.

Do mesmo modo, a consulta seguinte procura exibir NOME do USUARIO, DATA_EMPRESTIMO, NR_LIVRO e TITULO da obra tomada emprestada da biblioteca para o USUARIO de MATRICULA igual a 105.

Vamos construir essa consulta gradativamente, exibindo primeiro a coluna NOME da tabela USUARIO.

```
SELECT NOME FROM USUARIO;
```

Ela também deve exibir a DATA_EMPRESTIMO da tabela MOVIMENTACAO.

```
SELECT NOME, DATA_EMPRESTIMO FROM USUARIO, MOVIMENTACAO
WHERE USUARIO.MATRICULA = MOVIMENTACAO.MATRICULA;
```

Também introduzimos a cláusula WHERE para “forçar” o relacionamento entre as duas tabelas. Assim, temos que a coluna MATRICULA em USUARIO (a **chave primária**) deve ser igual à coluna MATRICULA em MOVIMENTACAO (a **chave estrangeira**).

Agora adicionamos a coluna NR_LIVRO da tabela MOVIMENTAÇÃO:

```
SELECT NOME, DATA_EMPRESTIMO, NR_LIVRO
FROM USUARIO U, MOVIMENTACAO M
WHERE U.MATRICULA = M.MATRICULA;
```

Além de acrescentarmos a coluna NR_LIVRO na cláusula SELECT, criamos apelidos ou nomes curtos para designar as tabelas: U para USUARIO e M para MOVIMENTACAO de modo a reescrever a cláusula WHERE com os apelidos.

Em termos práticos, o uso desses apelidos ou nomes curtos para identificar as tabelas não produz qualquer diferença no resultado de nossa consulta. A vantagem de sua utilização está na simplicidade de escrever os comandos. No lugar de MOVIMENTACAO escrevemos simplesmente M; e substituímos USUARIO por U.



O próximo passo de nossa empreitada é fazer com que o TITULO da OBRA também seja exibido por nossa consulta. O problema reside no fato de que nenhuma de nossas tabelas listadas no FROM se relaciona diretamente com OBRA. Na verdade, MOVIMENTACAO relaciona-se com a tabela LIVRO e esta sim com a tabela OBRA.

```
SELECT nome, DATA_EMPRESTIMO , m.nr_livro, o.titulo
FROM usuario u, movimentacao m, livro l, obra o
WHERE u.matricula = m.matricula
AND m.nr_livro = l.nr_livro
AND l.nr_obra = o.nr_obra;
```

Observe que a cláusula SELECT sofreu duas alterações:

- (1) Acréscimo da coluna TITULO proveniente da tabela OBRA.
- (2) Identificação da coluna NR_LIVRO como M.NR_LIVRO. Isso agora é necessário, pois em FROM há duas tabelas com coluna de mesmo nome (MOVIMENTACAO e LIVRO). Nesse caso, sempre que indicarmos a referência da coluna somos obrigados a declarar sua procedência. Assim definimos que o NR_LIVRO do SELECT é oriundo da tabela MOVIMENTACAO, mas também poderia ser da tabela LIVRO.

A cláusula FROM recebeu como acréscimos as tabelas LIVRO (L) e OBRA (O). Precisamos das duas tabelas para exibir a coluna TITULO. Em WHERE, a condição **M.NR_LIVRO = L.NR_LIVRO** “força” o relacionamento entre MOVIMENTACAO e LIVRO. Todavia, a coluna TITULO não está em LIVRO, mas sim em OBRA. Portanto, devemos também “forçar” o relacionamento entre as tabelas LIVRO e OBRA, através da condição **L.NR_OBRA = O.NR_OBRA**.

Para que concluir, acrescentamos a condição ao WHERE que filtra os registros da tabela MOVIMENTACAO para reter os relativos ao USUARIO de matricula = 105.

```
SELECT NOME, DATA_EMPRESTIMO , M.NR_LIVRO, O.TITULO
FROM USUARIO U, MOVIMENTACAO M, LIVRO L, OBRA O
WHERE U.MATRICULA = M.MATRICULA
AND M.NR_LIVRO = L.NR_LIVRO
AND L.NR_OBRA = O.NR_OBRA
AND U.MATRICULA = 105;
```

Utilizamos a coluna MATRICULA de USUARIO (U.Matricula), mas também poderíamos usar a coluna da tabela MOVIMENTACAO (M.MATRICULA).

Desde a versão 3.23.17 o SGBD MySQL aceita os comandos **INNER JOIN** e **LEFT JOIN**. O comando **INNER JOIN** “força” o relacionamento entre as tabelas GENERO e OBRA (“**FROM GENERO INNER JOIN OBRA**”) pela coluna ID_GENERO (“**USING (ID_GENERO);**”). Afinal, essa coluna existe nas duas tabelas. ID_GENERO é **chave primária** em GENERO e **chave estrangeira** em OBRA.

```
SELECT DESCRICAO, TITULO
FROM GENERO
INNER JOIN OBRA
USING (ID_GENERO);
```

5.6 Outras consultas

A cláusula LIMIT permite controlar a quantidade de registros de uma consulta, limitando o número de registros que devem ser retornados.

```
SELECT * FROM MOVIMENTACAO LIMIT 10;
```

Nesse exemplo o resultado da consulta é restringido a 10 linhas. Significa que apenas os 10 primeiros registros da consulta serão exibidos.

```
SELECT DISTINCT NOME FROM USUARIO;
```

O comando DISTINCT serve para evitar repetições de valores. A consulta acima pretende listar os nomes de todos na tabela USUARIO. Como DISTINCT foi posto imediatamente antes da coluna NOME, o resultado da consulta evitará a repetição de nomes. Portanto, se na tabela USUARIO existirem 13 registros com o nome "JOSÉ DA SILVA", ainda assim tal nome aparecerá uma única vez na consulta.

O comando da Figura 5.35 exemplifica a utilização do operador BETWEEN ("WHERE NR_OBRA BETWEEN 3 AND 6;"). Nesse caso, a cláusula WHERE filtra os registros a partir dos valores da coluna NR_OBRA. O operador BETWEEN (entre, E em inglês) define que somente interessam os registros com NR_OBRA maior ou = a 3 e menor ou = a 6.

```
mysql> SELECT * FROM OBRA WHERE NR_OBRA BETWEEN 3 AND 6;
```

NR_OBRA	TITULO	ID_GENERO
3	TEORIA DO CAOS	DID
4	OS TRÊS MOSQUETEIROS	LIT
5	A HISTORIA PERDIDA E REVELADA	REL
6	O CONDE DE MONTECRISTO	LIT

```
4 rows in set (0.06 sec)
```

Figura 5.35: Resultado de consulta com NR_OBRA entre 3 e 6

Fonte: Elaborada pelo autor

Resumo

Nesta aula tratamos de variados tipos de consultas SQL envolvendo apenas uma tabela ou várias tabelas simultaneamente, utilizando funções especiais como COUNT, SUM, MAX, MIN, AVG, CURDATE, YEAR, MONTH e DAY. Vimos também como definir as colunas das tabelas que devem ser visualizadas pela cláusula SELECT da SQL. Também abordamos como filtrar os registros que devem aparecer pela cláusula WHERE e como ordenar o resultado final da consulta pela cláusula ORDER BY. Usamos também o operador LIKE para consultas aproximadas de *substrings*.

Atividades de aprendizagem

Novamente, utilizando as tabelas de agendamento de consultas médicas, descreva o comando SQL para as seguintes consultas:

- a) Exibir o nome do convênio, o nome do cliente e a data da consulta para toda consulta agendada entre 20/fevereiro/2010 e 25/março/2010.
- b) Exibir o nome do convênio e a quantidade de consultas agendadas para a data de 03 de abril de 2010.
- c) Nome dos médicos e a quantia gerada em dinheiro por todas as consultas atendidas ($\text{VALOR_CLIENTE} + \text{VALOR_CONVENIO}$) no mês de janeiro de 2010. Observe que para cada consulta há a possibilidade de um Plano de Saúde pagar parte do valor e o cliente pagar a outra parcela. Para cliente particular, o valor pago pelo convênio é sempre zero.

Aula 6 – Segurança da Informação

Objetivos

Alterar a senha do administrador do Banco de Dados.

Criar novas contas de usuário.

Definir privilégios de segurança diferenciados para cada conta de usuário.

Criar visões diferentes para o mesmo Banco de Dados.

Utilizar as visões para definir melhor a segurança das informações armazenadas.

6.1 Preparando o terreno

Inicialmente, vamos criar outro BD. Após executar o SGBD MySQL e informar a senha padrão para o administrador do BD ("**root**"), usamos novamente o comando CREATE DATABASE para criar o banco EXEMPLO.

```
mysql> CREATE DATABASE EXEMPLO;  
Query OK, 1 row affected (0.03 sec)
```

Em seguida, entramos no Banco de Dados recém-criado:

```
mysql> USE EXEMPLO;  
Database changed
```

Agora criamos a tabela FILMES com quatro colunas: ID (a chave primária), TITULO, ANO e BILHETERIA.

```
mysql> CREATE TABLE FILMES(  
  ID INTEGER NOT NULL PRIMARY KEY,  
  TITULO VARCHAR(60) NOT NULL,  
  ANO INTEGER NOT NULL, BILHETERIA NUMERIC(6,2) NOT NULL);  
Query OK, 0 rows affected (0.09 sec)
```


Após inserir quatro registros de FILMES:

```
mysql> INSERT INTO FILMES VALUES (1,"TRANSFORMERS",2007,390.89);  
Query OK, 1 row affected (0.05 sec)
```

```
mysql> INSERT INTO FILMES VALUES (2,"TRANSFORMERS  
2",2009,990.90);  
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO FILMES VALUES (3,"STAR TREK",2009,598.99);  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO FILMES VALUES (4,"WOLVERINE - ORI-  
GENS",290,391.00);  
Query OK, 1 row affected (0.02 sec)
```

Executamos uma consulta sobre a mesma tabela usando o comando SELECT e listamos sua estrutura pelo comando DESCRIBE (Figura 6.1).

```
mysql> SELECT * FROM FILMES;
```

ID	TITULO	ANO	BILHETERIA
1	TRANSFORMERS	2007	390.89
2	TRANSFORMERS 2	2009	990.90
3	STAR TREK	2009	598.99
4	WOLVERINE - ORIGENS	2009	391.00

4 rows in set (0.02 sec)

```
mysql> DESCRIBE FILMES;
```

Fields	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI		
TITULO	varchar(60)	NO			
ANO	int(11)	NO			
BILHETERIA	decimal(6,2)	NO			

4 rows in set (0.02 sec)

Figura 6.1: Conteúdo da tabela FILMES e exibição de sua estrutura

Fonte: Elaborada pelo autor

6.2 Alterando a senha do administrador do BD

Até o momento sempre acessamos o BD como administrador (**login**: root e **password**: root). Comentamos, anteriormente, que não é interessante manter essa senha (**password**). Afinal, qualquer um poderia acessar seu BD como se fosse o próprio administrador, com plenos poderes sobre tudo o que nele existe.

A senha (**password**) do administrador do Banco de Dados (o usuário root) deve ser de conhecimento apenas dele. Na verdade, cada usuário do BD deve possuir uma senha que seja, exclusivamente, de seu conhecimento.



Para mudar a senha do administrador, acessamos o MYSQL como o administrador (usuário root) e em seguida entramos no Banco de Dados MYSQL ("**USE MYSQL;**").

```
Enter password: ****
```

```
Welcome to the MySQL monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 1
```

```
Server version: 5.0.41-community-nt MySQL Community Edition (GPL)
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> USE MYSQL;
```

```
Database changed
```

Em seguida executamos o comando UPDATE sobre a tabela USER do BD MYSQL:

```
mysql> UPDATE USER SET PASSWORD=PASSWORD("1234") WHERE  
USER="ROOT";
```

```
Query OK, 1 row affected (0.02 sec)
```

```
Rows matched: 1 Changed: 1 Warnings: 0
```

Esse comando altera a tabela USER ("UPDATE USER") alterando o valor da coluna PASSWORD ("**SET PASSWORD = PASSWORD('1234')**") para "1234". Foi usada uma função PASSWORD() para mascarar os dados da coluna e restringimos os registros a serem alterados através da cláusula WHERE ("**WHERE USER = 'root';**"). Ou seja, apenas o usuário USER = "root" terá sua senha alterada para "1234".

Cuidado para não esquecer a senha alterada do administrador do BD.



Em seguida executamos o comando "**FLUSH PRIVILEGES;**" para confirmar a mudança e saímos do MYSQL com o comando "**QUIT**".

```
mysql> FLUSH PRIVILEGES;
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> quit
```

6.3 Criando novas contas de usuário e definindo seus privilégios de segurança

Após entrar no Banco de Dados EXEMPLO, vamos criar uma nova conta de usuário:

```
mysql> GRANT SELECT, INSERT, UPDATE ON EXEMPLO.* TO COMUM IDENTIFIED BY '1234';
```

Query OK, 0 rows affected (0.03 sec)

Esse comando autoriza (**GRANT**) selecionar (**SELECT**), inserir (**INSERT**) e alterar (**UPDATE**) registros de todas as tabelas do BD Exemplo ("**ON EXEMPLO.***") para o usuário comum ("**TO COMUM**") identificado pela senha "1234" ("**IDENTIFIED BY '1234';**").

Agora, além de root também temos um usuário chamado COMUM com senha **1234**. Ele não está autorizado a excluir registros (**DELETE**) de nenhuma tabela do Banco de Dados EXEMPLO, pois simplesmente lhe omitimos esse privilégio.

Para testarmos o nível de autorização desse usuário, devemos sair do MYSQL, que, no momento, ainda estamos como Administrador do BD. Em seguida, devemos executar o MYSQL de uma maneira diferente. A partir do botão INICIAR do Windows XP clique na opção EXECUTAR e digite:

```
mysql -h localhost -uCOMUM -p1234
```

Então, entramos no servidor MYSQL local ("**MYSQL-H LOCALHOST**") como o usuário COMUM ("**-UCOMUM**") passando a senha 1234 ("**-P1234**").

Caso esse comando falhe se o Windows não encontrar o MYSQL, você deve executar o *prompt* de comando (que geralmente fica em acessórios):

```
Microsoft Windows XP [versão 5.1.2600] (C) Copyright 1985-2001  
Microsoft Corp.  
C:\Documents and Settings\Silva>
```

Então, digite o comando "**cd**" e tecla <ENTER> para chegar ao diretório raiz ("**c:\>**").

```
C:\Documents and Settings\Silva>cd\  
C:\>
```

Em seguida digite "dir *." E tecla <ENTER>. O resultado será a lista de diretórios (pastas) existentes na unidade "C" do disco rígido, como mostrado logo abaixo:

```
C:\>DIR *.
O volume na unidade C é ACER
O número de série do volume é 406D-517A
Pasta de C:\

12/10/2009 23:40 <DIR>  Arquivos de programas
23/07/2009 11:56 <DIR>  Book
03/10/2009 22:58 <DIR>  Documents and Settings
11/10/2009 00:07 <DIR>  i386
17/03/2009 19:48 <DIR>  Intel
10/10/2009 21:53 <DIR>  Python26
05/10/2009 09:24 <DIR>  Sun
17/03/2009 20:07 <DIR>  users
12/09/2008 00:53 <DIR>  VALUEADD
22/11/2009 17:53 <DIR>  WINDOWS

    0 arquivo(s)      0 bytes
    10 pasta(s) 120.542.089.216 bytes disponíveis
C:\>
```

O MySQL está dentro do diretório (ou pasta) "Arquivos de Programas". Por isso, vamos entrar nesse diretório com o comando "**cd Arquivos de Programas**".

```
C:\>cd arquivos de programas
```

Vamos agora listar os diretórios dentro de "Arquivos de Programas".

```
C:\Arquivos de programas>dir *.
O volume na unidade C é ACER
O número de série do volume é 406D-517A
Pasta de C:\Arquivos de programas

12/10/2009 23:40 <DIR> .
12/10/2009 23:40 <DIR> ..
23/07/2009 11:54 <DIR> Acer
10/10/2009 21:56 <DIR> fabFORCE
```

```
17/03/2009 19:50 <DIR> Intel
17/03/2009 19:12 <DIR> Internet Explorer
05/10/2009 19:58 <DIR> Java
05/10/2009 02:01 <DIR> MySQL
17/03/2009 19:02 <DIR> NetMeeting
```

```
0 arquivo(s) 0 bytes
9 pasta(s) 120.522.969.088 bytes disponíveis
C:\Arquivos de programas>
```

Observe que há um diretório do “MySQL” dentro do diretório “Arquivos de Programas”. Devemos entrar nele com o comando “**cd mysql**”.

```
C:\Arquivos de programas>cd mysql
C:\Arquivos de programas\mysql>
```

Entretanto, esse ainda não é o diretório onde podemos encontrar o arquivo executável (programa). Se digitarmos “**dir *.***” e, em seguida, teclarmos <ENTER>, então teremos acesso à lista de arquivos e diretórios existentes dentro do diretório “MySQL”.

```
C:\Arquivos de programas\mysql>dir *.*
O volume na unidade C é ACER
O número de série do volume é 406D-517A
Pasta de C:\Arquivos de programas\MySQL

05/10/2009 02:01 <DIR> .
05/10/2009 02:01 <DIR> ..
05/10/2009 02:12 <DIR> MySQL Server 5.1

0 arquivo(s) 0 bytes
3 pasta(s) 120.522.416.128 bytes disponíveis
C:\Arquivos de programas\mysql>
```

Há um diretório de nome “MySQL Server 5.1”. Vamos acessá-lo pelo comando “**cd MySQL Server 5.1**”. Em seguida entraremos no diretório “bin” com o comando “**cd bin**” e acessaremos o MySQL como usuário COMUM.

```
C:\Arquivos de programas\MySQL>cd mysql server 5.1
C:\Arquivos de programas\MySQL\MySQL Server 5.1>
```

```
C:\Arquivos de programas\MySQL\MySQL Server 5.1>dir *.*
```

O volume na unidade C é ACER

O número de série do volume é 406D-517A

Pasta de C:\Arquivos de programas\MySQL\MySQL Server 5.1

```
05/10/2009 02:12 <DIR>      .
05/10/2009 02:12 <DIR>      ..
05/10/2009 02:01 <DIR>      bin
04/09/2009 21:37      19.071 COPYING
04/09/2009 21:37      5.139 EXCEPTIONS-CLIENT
05/10/2009 02:01 <DIR>      lib
12/11/2008 04:13      71.772 loginhdr.jpg
04/09/2009 21:37      4.907 my-huge.ini
04/09/2009 21:37      20.646 my-innodb-heavy-4G.ini
04/09/2009 21:37      4.881 my-large.ini
04/09/2009 21:37      4.890 my-medium.ini
04/09/2009 21:37      2.462 my-small.ini
12/11/2008 04:13      13.129 my-template.ini
05/10/2009 02:12      8.967 my.ini
05/10/2009 02:01      5.531 register.htm
05/10/2009 02:01 <DIR>      share
05/10/2009 02:01      1.281 svctag.xml
```

12 arquivo(s) 162.676 bytes

5 pasta(s) 120.522.326.016 bytes disponíveis

```
C:\Arquivos de programas\MySQL\MySQL Server 5.1>cd bin
```

```
C:\Arquivos de programas\MySQL\MySQL Server 5.1\bin>dir *.*
```

O volume na unidade C é ACER

O número de série do volume é 406D-517A

Pasta de C:\Arquivos de programas\MySQL\MySQL Server 5.1\bin

```
05/10/2009 02:01 <DIR>      .
05/10/2009 02:01 <DIR>      ..
04/09/2009 21:37  2.359.296 libmysql.dll
04/09/2009 21:37  2.089.600 myisamchk.exe
04/09/2009 21:37  1.979.008 myisamlog.exe
04/09/2009 21:37  2.011.776 myisampack.exe
04/09/2009 21:37  1.966.720 myisam_ftdump.exe
04/09/2009 21:37  2.347.648 mysql.exe
04/09/2009 21:37  5.664.768 mysql.pdb
```

```

04/09/2009 21:37 2.282.112 mysqladmin.exe
04/09/2009 21:37 5.459.968 mysqladmin.pdb
04/09/2009 21:37 2.372.224 mysqlbinlog.exe
04/09/2009 21:37 5.926.912 mysqlbinlog.pdb
04/09/2009 21:37 2.278.016 mysqlcheck.exe
04/09/2009 21:37 6.041.600 mysqld.exe
04/09/2009 21:37 5.708.778 mysqld.map
04/09/2009 21:37 21.385.216 mysqld.pdb
04/09/2009 21:37 2.335.360 mysqldump.exe
04/09/2009 21:37 5.599.232 mysqldump.pdb
04/09/2009 21:37 2.273.920 mysqlimport.exe
04/09/2009 21:37 5.427.200 mysqlimport.pdb
13/08/2009 23:43 2.972.800 MySQLInstanceConfig.exe
04/09/2009 21:37 2.278.016 mysqlshow.exe
04/09/2009 21:37 5.427.200 mysqlshow.pdb
04/09/2009 21:37 1.802.880 mysql_upgrade.exe
04/09/2009 21:37 1.728.512 my_print_defaults.exe
04/09/2009 21:37 1.716.224 perror.exe
04/09/2009 21:37 1.708.032 resolveip.exe

26 arquivo(s) 103.143.018 bytes
2 pasta(s) 120.522.326.016 bytes disponíveis
C:\Arquivos de programas\MySQL\MySQL Server 5.1\bin>mysql -h localhost
-uCOMUM -p1234

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.39-community MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

```



Conforme o sistema operacional utilizado e sua versão, a sequência de passos pode diferir em muito da utilizada aqui. Esteja atento!

Acessado o MySQL como o usuário COMUM, tentaremos a inclusão de um registro na tabela FILMES para testar o privilégio INSERT sobre as tabelas do BD EXEMPLO.

```
mysql> INSERT INTO filmes VALUE (5,"O PODEROSO CHEFAO",1972,600);
Query OK, 1 row affected (0.00 sec)
```

Obtivemos sucesso na tentativa de inserir um novo registro de FILMES. Resta testar nosso privilégio de alterar (UPDATE) e consultar (SELECT) as tabelas (Figura 6.2).

```
mysql> UPDATE FILMES SET TITULO = "THE GODFATHER" WHERE ID = 5;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 changed: 1 Warnings: 0
mysql> SELECT * FROM FILMES;
```

ID	TITULO	ANO	BILHETERIA
1	TRANSFORMERS	2007	390.89
2	TRANSFORMERS 2	2009	990.90
3	STAR TREK	2009	598.99
4	WOLVERINE - ORIGENS	2009	391.00
5	THE GODFATHER	1972	600.00

```
5 rows in set (0.02 sec)
```

Figura 6.2: Alteração do título de filme de "O PODEROSO CHEFAO" para "THE GODFATHER"

Fonte: Elaborada pelo autor

Mais uma vez obtivemos o sucesso esperado. Testamos agora o que não podemos fazer nesse BD como usuário COMUM, por exemplo, a excluir registros (DELETE).

```
mysql> DELETE FROM FILMES WHERE ID = 5;
ERROR 1142 (42000): DELETE command denied to user
'COMUM'@'localhost' for table
```

De fato, o MySQL bloqueou nossa tentativa. Note a mensagem de erro gerada: o comando DELETE foi negado para o usuário COMUM.

O comando **GRANT** atribui autorizações para usuários, o comando **REVOKE** as revoga. Para executá-lo devemos primeiro reacessar o MySQL como usuário root.

```
mysql> REVOKE INSERT ON EXEMPLO.* FROM COMUM;
Query OK, 0 rows affected (0.00 sec)
```

Dessa forma acabamos de revogar o privilégio de INSERT sobre todas as tabelas do Banco de Dados EXEMPLO para o usuário COMUM. Para confirmar essa mudança devemos novamente executar o comando:

```
mysql> flush privileges;  
Query OK, 0 rows affected (0.11 sec)
```

Agora voltemos a entrar no MySQL como o usuário COMUM para testar a perda do privilégio de inserir novos registros nas tabelas de EXEMPLO.

```
mysql> INSERT INTO filmes VALUES (6,"AVATAR",2009,999);  
ERROR 1142 (42000): INSERT command denied to user  
'COMUM'@'localhost' for table 'filmes'
```

De fato, perdemos o privilégio de inserir novos registros de filmes.



Comente o risco de se manterem vários usuários acessando o BD com a mesma senha e a mesma conta (root).

6.4 Criando visões

A-Z

Visão (VIEW)

É uma forma de visualização dos dados de uma ou mais tabelas. É uma espécie de tabela virtual formada a partir de outras tabelas ou mesmo de outras visões. Operações de INSERT, UPDATE, DELETE e SELECT podem ser aplicadas sobre uma visão. Porém, uma visão não deve ser confundida com uma tabela, pois ao contrário destas, não ocupa espaço de memória secundária para armazenamento de dados.

Considere que, por alguma razão, o usuário COMUM não deva ter acesso sobre todas as colunas e linhas da tabela FILMES. Então resolvemos lhe impor algumas restrições: somente acessará FILMES produzidos de 2009 em diante (restrição de linhas) e não visualizará a coluna BILHETERIA (restrição de coluna).

Inicialmente, revogamos todos os privilégios do usuário COMUM sobre FILMES.

```
mysql> REVOKE ALL ON EXEMPLO.* FROM COMUM;  
Query OK, 0 rows affected (0.00 sec)
```

O comando revoga todos os privilégios ("**REVOKE ALL**") do usuário COMUM ("**FROM COMUM**") sobre tudo que há no BD EXEMPLO ("**ON EXEMPLO.***"). Contudo, para que essa mudança se efetive não podemos nos esquecer do comando:

```
mysql> FLUSH PRIVILEGES;  
Query OK, 0 rows affected (0.11 sec)
```

Finalmente, criamos uma visão (VIEW) com as restrições anteriormente sugeridas:

```
CREATE VIEW FILMES_NOVOS AS SELECT ID, TITULO, ANO FROM FILMES  
WHERE ANO >= 2009;
```

```
Query OK, 0 rows affected (0.05 sec)
```


Repare que a visão criada recebeu o nome FILMES_NOVOS (“CREATE VIEW FILMES_NOVOS”). Em seguida, usamos uma consulta SQL para definir as restrições:

- (1)** Omitimos a coluna BILHETERIA na cláusula SELECT da consulta SQL que define a visão. Portanto, essa coluna não será acessível pela visão FILMES_NOVOS.
- (2)** Estabelecemos como condição da cláusula WHERE que apenas filmes com ano de produção maior ou igual a 2009 poderão ser acessados pela visão FILMES_NOVOS.

O comando seguinte define os privilégios do usuário COMUM sobre a visão (VIEW):

```
mysql> GRANT UPDATE, SELECT ON EXEMPLO.FILMES_NOVOS TO CO-  
MUM;  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> FLUSH PRIVILEGES;  
Query OK, 0 rows affected (0.11 sec)
```

Portanto, o usuário COMUM pode consultar e alterar os registros da visão FILMES_NOVOS. Por outro lado, fica impedido de apagar (DELETE) e de inserir (INSERT) registros na mesma visão.

Para verificar os privilégios de segurança do usuário COMUM, acessamos o MySQL com sua conta de usuário. Iniciaremos pelo comando SELECT.

```
mysql> SELECT * FROM FILMES;  
ERROR 1142 (42000): SELECT command denied to user  
'COMUM'@'localhost' for table 'filmes'
```

Confirmado: o usuário COMUM não pode mais efetuar uma simples consulta relacionada à tabela FILMES (revogamos seus privilégios com o comando REVOKE).

Porém, COMUM pode efetuar consultas à visão FILMES_NOVOS (Figura 6.3).

```
mysql> SELECT * FROM FILMES_NOVOS;
+----+-----+-----+
| ID | TITULO                | ANO |
+----+-----+-----+
| 2  | TRANSFORMERS 2        | 2009 |
| 3  | STAR TREK              | 2009 |
| 4  | WOLVERINE - ORIGENS    | 2009 |
+----+-----+-----+
3 rows in set (0.20 sec)
```

Figura 6.3: Conteúdo da visão FILMES_NOVOS

Fonte: Elaborada pelo autor

Repare que a consulta acima trouxe todos os registros de FILMES_NOVOS e isso não equivale a todos os registros de FILMES. Os FILMES com IDs 1 e 5 não aparecem no resultado dessa pesquisa, pois foram produzidos antes de 2009. Apesar de termos usado o caractere "*" na cláusula SELECT, a coluna BILHETERIA não aparece, pois ela não foi usada para a criação da visão FILMES_NOVOS.

```
mysql> INSERT INTO FILMES_NOVOS VALUES (1,"MATRIX", 1999);
ERROR 1142 (42000): INSERT command denied to user
'COMUM'@'localhost' for table 'filmes_novos'
```

COMUM também não mais pode inserir novos registros na tabela FILMES. Também não pode fazê-lo com a visão FILMES_NOVOS. O único outro privilégio de segurança que lhe restou é o de alterar registros de FILMES_NOVOS (Figura 6.4).

```
mysql> UPDATE FILMES_NOVOS SET TITULO = "X-MEN ORIGENS: WOLVERINE"
-> WHERE ID = 4;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1 changed: 1 Warnings: 0
mysql> SELECT * FROM FILMES_NOVOS;
+----+-----+-----+
| ID | TITULO                | ANO |
+----+-----+-----+
| 2  | TRANSFORMERS 2        | 2009 |
| 3  | STAR TREK              | 2009 |
| 4  | X-MEN ORIGENS: WOLVERINE | 2009 |
+----+-----+-----+
3 rows in set (0.02 sec)
```

Figura 6.4: Conteúdo da visão FILMES_NOVOS após alteração do título de filme de "WOLVERINE – ORIGENS" para "X-MEN ORIGENS: WOLVERINE"

Fonte: Elaborada pelo autor

Note que COMUM conseguiu alterar o TITULO em FILMES_NOVOS do registro com ID = 4. O TITULO passou de "**WOLVERINE – ORIGENS**" para "**X-MEN ORIGENS: WOLVERINE**". E essa alteração não se restringe aos dados da visão. Esse registro também foi alterado na tabela FILMES que serviu de base para a geração da visão FILMES_NOVOS.

Lembre que FILMES_NOVOS não é uma tabela, mas uma visão de parte dos dados da tabela FILMES. Assim, FILMES_NOVOS não implica uma maior ocupação da memória secundária. Os registros de FILMES_NOVOS são os mesmos de FILMES visualizados diferentemente.

Resumo

Nesta aula tratamos de questões relacionadas à segurança da informação em um Banco de Dados MySQL. Mais uma vez não esgotamos o assunto, mas vimos como alterar a senha do administrador do Banco de Dados, como criar novas contas de usuários e definir seus privilégios de segurança diferenciadamente. Também vimos como criar e utilizar as visões em um Banco de Dados.

Atividades de aprendizagem

Considere uma tabela de nome FUNCIONARIO com as seguintes colunas:

MATRICULA **INTEGER NOT NULL PRIMARY KEY**,
NOME **VARCHAR(50) NOT NULL**,
SEXO **CHAR(1) NOT NULL**,
TELEFONE **VARCHAR(15)**,
SETOR **INTEGER NOT NULL**,
SALARIO **NUMERIC(6,2)**, e
SENHA **VARCHAR(20)**.

A empresa proprietária desse BD está preocupada com a segurança dessa tabela e lhe passa a incumbência de:

- a) criar uma visão chamada FUNC em que não seja possível visualizar a SENHA e o SALARIO de qualquer funcionário e nem os dados daqueles que trabalham nos setores 1, 2 e 3.
- b) criar nova conta de usuário com privilégios apenas para consultar e inserir novos registros em FUNC.
- c) revogar todos os privilégios sobre a tabela FUNCIONARIO para esse novo usuário.

Escreva corretamente os comandos que atenderão às incumbências acima listadas.

Aula 7 – Procedimentos Armazenados e Gatilhos

Objetivos

Diferenciar um procedimento armazenado de um gatilho.

Identificar situações que justifiquem o uso de procedimentos armazenados e gatilhos.

Criar procedimentos armazenados.

Executar procedimentos armazenados.

Apagar procedimentos armazenados.

Efetuar passagens de parâmetros em procedimentos armazenados.

Criar gatilhos que disparem antes ou após um evento.

Alguns SGBDs – e o MySQL é um deles – permitem armazenar não somente dados, mas também pequenos programas que acessem e manipulem esses mesmos dados. Esses miniprogramas são geralmente identificados como rotinas armazenadas e podem ser basicamente de dois tipos: procedimentos armazenados ou gatilhos.

7.1 Importância de rotinas armazenadas

Rotinas armazenadas apresentam como vantagens no contexto de um Banco de Dados:

- (1)** Centralizam as operações críticas do BD. É o que ocorre quando programas aplicativos escritos em diferentes linguagens de programação e rodando sob diferentes plataformas precisam executar as mesmas operações de BD.
- (2)** Padronizam as operações de BD, que deixam de depender, excessivamente, da memória e da disciplina de usuários humanos. Se todo dia, por exemplo, há a necessidade de executar uma sequência de operações que envolve a inclusão de registros em uma tabela T1 e a consequente atualiza-

A-Z

Rotinas armazenadas

São um conjunto de comandos SQL que podem convenientemente ser armazenados em um servidor.

ção de registros em uma tabela T2 para, somente então, gerar um relatório impresso R1 com base nos dados de T1 e T2, uma rotina armazenada pode ser útil. Executada todo dia, assegura que T2 seja atualizado apenas após a inclusão de registros em T1 e que R1 seja gerado apenas após a atualização de T2. Reduz-se assim a margem para erros humanos.

- (3) Os usuários finais deixam de manipular ou acessar diretamente as tabelas e passam a fazê-lo somente pelas rotinas armazenadas, que foram testadas antes de entrar em produção e fazem validação dos dados manipulados.
- (4) Podem melhorar a *performance* do BD, providenciando um tempo de resposta menor para operações que exigem intensa troca de informações do Servidor com as aplicações Clientes: validações de informações que exigiriam um pesado intercâmbio de dados passam a ser realizadas no Servidor de Banco de Dados em vez de nos Clientes. Um bom exemplo é quando um Cliente solicita ao Servidor que registre a venda de um produto (inclusão na tabela VENDAS). Após o Servidor confirmar a conclusão da operação, o Cliente solicita que o estoque do produto seja atualizado pelo Servidor (atualização na tabela PRODUTO). O fluxo excessivo de informações entre o Servidor e seus Clientes pode ser reduzido quando tudo que o Cliente deve fazer é executar uma rotina armazenada no Servidor. Esta se encarrega de inserir registros na tabela de VENDAS e atualizar os registros na tabela de PRODUTOS, sem que Servidor e Clientes necessitem trocar mais informações.

7.2 Procedimentos armazenados

Novamente entramos no BD EXEMPLO, acessando o MySQL como root (Administrador). Para a criação de um **procedimento armazenado (stored procedure)**, utilizamos o comando CREATE PROCEDURE, conforme exemplo abaixo:

```
mysql> DELIMITER //

mysql> CREATE PROCEDURE bilheteriaMedia()
-> BEGIN
-> SELECT ANO, AVG(BILHETERIA) AS BILHETERIA_MEDIA
-> FROM FILMES
-> GROUP BY ANO;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```

A-Z

Procedimentos armazenados (stored procedure)

São rotinas armazenadas que devem ser executadas por intermédio de invocações explícitas da parte do usuário (comando **CALL**).

Antes do comando `CREATE PROCEDURE`, executamos o comando `"DELIMITER //"`, que substitui temporariamente o caractere `";"` pelos `"//"` como finalizador de comandos no MySQL. Significa que ao deparar com um ponto e vírgula, o interpretador de comandos do MySQL não "entende" que o usuário terminou a digitação do comando. Útil, por exemplo, na linha do `"GROUP BY ANO;"`. Sem o comando `"DELIMITER //"` antes, certamente haveria um erro nessa linha. A definição do procedimento armazenado termina com o comando `"END//"`. Então, o interpretador de comandos do MySQL "entende" que o usuário já completou o comando. Após a criação do procedimento, executamos o comando `"DELIMITER;"` para que o ponto e vírgula volte a ser o finalizador de comandos.

O procedimento armazenado acima foi criado com o nome de `bilheteriaMedia` (`"CREATE PROCEDURE BilheteriaMedia()"`). Os comandos do procedimento propriamente começam com o comando `"BEGIN"` (Início). Em seguida, há uma consulta SQL normal que começa com `"SELECT"` e termina com `"GROUP BY ANO;"`. Em seguida, o comando `"END//"` fecha o bloco de comando iniciado com `"BEGIN"`. Ou seja, para todo comando `"BEGIN"` deve haver um comando `"END"`.

O objetivo desse procedimento é exibir a média das bilheterias por ano de produção dos FILMES. Para executá-la, utilizamos o comando `"CALL BILHETERIAMEDIA()"` conforme mostrado na Figura 7.1 a seguir.

```
mysql> CALL BILHETERIAMEDIA;
+-----+-----+
| ANO  + BILHETERIAMEDIA |
+-----+-----+
| 1972 |      600.000000 |
| 2007 |      390.890000 |
| 2009 |      660.296667 |
+-----+-----+
3 rows in set (0.00 sec)
Query OK, 0 rows affected (0.02 sec)
```

Figura 7.1: Resultado do Comando "CALL BILHETERIAMEDIA;"

Fonte: Elaborada pelo autor

Geralmente, criam-se procedimentos armazenados para mantê-los no Servidor de BD. Porém, pode-se excluir o procedimento com o comando `DROP PROCEDURE`.

```
mysql> DROP PROCEDURE BILHETERIAMEDIA;  
Query OK, 0 rows affected (0.05 sec)
```

Observe que omitimos os parênteses que acompanham o nome do procedimento para a passagem de parâmetros, mas eles podem receber parâmetros de entrada ou saída entre os parênteses após o seu nome.

```
mysql> DELIMITER //  
  
mysql> CREATE PROCEDURE ANALISE_BILHETERIA  (  
-> OUT MAIOR_BIL NUMERIC(6,2),  
-> OUT MENOR_BIL NUMERIC(6,2),  
-> OUT MEDIA_BIL NUMERIC(6, 2))  
-> BEGIN  
-> SELECT MIN(BILHETERIA) INTO MENOR_BIL FROM FILMES;  
-> SELECT MAX(BILHETERIA) INTO MAIOR_BIL FROM FILMES;  
-> SELECT AVG(BILHETERIA) INTO MEDIA_BIL FROM FILMES;  
-> END//  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> DELIMITER ;
```

O comando anterior cria um procedimento armazenado (*stored procedure*) de nome ANALISE_BILHETERIA. Ao contrário do anterior, ele recebe três parâmetros de saída (OUT): MAIOR__BIL, MENOR_BIL, e MEDIA_BIL. Todos eles foram declarados do mesmo tipo que a coluna BILHETERIA em FILMES (**NUMERIC(6,2)**). Esses parâmetros não levam valores para dentro do procedimento armazenado, servem para levar para fora do procedimento armazenado os resultados gerados.

O operador INTO usado nas cláusulas SELECT armazena os resultados lidos nos respectivos parâmetros de saída. No comando “**SELECT MAX(BILHETERIA) INTO MAIOR_BIL**”, por exemplo, o operador INTO armazena o valor encontrado da maior bilheteria no parâmetro MAIOR_BIL.

Antes de executar o procedimento armazenado por meio do comando CALL, devemos criar variáveis para armazenar seus resultados.


```
mysql> SET @MAIOR = 0;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SET @MENOR = 0;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SET @MEDIA = 0;  
Query OK, 0 rows affected (0.00 sec)
```

Nos comandos acima, foram criadas três variáveis: @MAIOR, @MENOR e @MEDIA. Todas foram inicializadas com valor zero. O caractere arroba (“@”) utilizado no início de cada nome de variável é obrigatório.

```
mysql> CALL ANALISE_BILHETERIA(@MAIOR, @MENOR, @MEDIA);  
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

À primeira vista, a execução desse procedimento armazenado não parece produzir qualquer resultado. Se não fosse a mensagem de “Query Ok” gerada pelo MySQL, não teríamos como ter certeza de sua execução, uma vez que para exibir o resultado teríamos que executar o comando da Figura 7.2.

```
mysql> SELECT @MAIOR AS MAIOR_BILHETERIA, @MENOR AS MENOR_BILHETERIA,  
@MEDIA AS BILHETERIA_MEDIA;  
+-----+-----+-----+  
| MAIOR_BILHETERIA | MENOR_BILHETERIA | BILHETERIA_MEDIA |  
+-----+-----+-----+  
|          990.90 |          390.89 |          594.36 |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

Figura 7.2: Resultado do Comando para analisar as bilheterias

Fonte: Elaborada pelo autor

Repare que o comando SELECT não busca os dados em tabelas (ausência da cláusula FROM), mas apenas nas variáveis que criamos e passamos ao procedimento armazenado como parâmetros de saída.

A execução desse procedimento pareceu complicada. No entanto, podemos suavizá-la com a criação de um outro procedimento que faça uma chamada a este último.

```
mysql> DELIMITER //

mysql> CREATE PROCEDURE ANALISE_RESULTADOS()
-> BEGIN
-> SET @MAIOR = 0;
-> SET @MENOR = 0;
-> SET @MEDIA = 0;
-> CALL ANALISE_BILHETERIA(@MAIOR, @MENOR, @MEDIA);
-> SELECT @MAIOR AS MAIOR_BILHETERIA,
-> @MENOR AS MENOR_BILHETERIA,
-> @MEDIA AS BILHETERIA_MEDIA;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```

Repare que toda a sequência de passos necessária para os resultados foi encapsulada por um novo procedimento: ANALISE_RESULTADOS. É ele que cria as variáveis de memória (@MAIOR, @MENOR e @MEDIA) e executa ANALISE_BILHETERIA, para depois exibir os resultados gerados por este último procedimento (Figura 7.3).

```
mysql> CALL ANALISE_RESULTADOS( );
+-----+-----+-----+
| MEIOR_BILHETERIA | MENOR_BILHETERIA | BILHETERIA_MEDIA |
+-----+-----+-----+
|          990.90 |          390.89 |          594.36 |
+-----+-----+-----+
1 row in set (0.00 sec)
Query OK, 0 rows affected, 1 warning (0.02 sec)
```

Figura 7.3: Resultado do Comando “CALL ANALISE_RESULTADOS();”

Fonte: Elaborada pelo autor

Agora, com a criação do novo procedimento, o usuário final não precisa saber ou lembrar que variáveis devem ser criadas, ou mesmo a sequência correta de passagem dessas variáveis como parâmetros, ou ainda, que é necessário executar um SELECT ao final de tudo. O outro procedimento se encarrega de tudo isso.

```
mysql> DELIMITER //

mysql> CREATE PROCEDURE BILHETERIA_ANO( IN ANOP INT )
-> BEGIN
-> SELECT ANO, MAX(BILHETERIA) AS MAIOR_BILHETERIA,
-> MIN(BILHETERIA) AS MENOR_BILHETERIA,
```

```
-> AVG(BILHETERIA) AS MEDIA_BILHETERIA
-> FROM FILMES
-> WHERE ANO = ANOP
-> GROUP BY ANO;
-> END//
```

Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;

O procedimento armazenado BILHETERIA_ANO recebe como parâmetro de entrada (IN) o ano de produção dos FILMES (ANOP) declarado como inteiro (INT). Observe que o parâmetro de entrada ANOP não serve para levar valores para fora do procedimento, mas apenas para trazer uma informação para dentro do procedimento.

O comando da Figura 7.4 gera os resultados somente para o ano de 2009. Caso pretendamos exibir os resultados para o ano de 1972, devemos chamar o procedimento BILHETERIA_ANO com o ano de 1972 como parâmetro (Figura 7.5).

```
mysql> CALL BILHETERIA_ANO(2009);
```

ANO	MEIOR_BILHETERIA	MENOR_BILHETERIA	MEDIA_BILHETERIA
2009	990.90	391.00	660.296667

```
1 row in set (0.01 sec)
Query OK, 0 rows affected (0.01 sec)
```

Figura 7.4: Resultado do Comando "CALL BILHETERIA_ANO(2009);"

Fonte: Elaborada pelo autor

Como existe apenas um filme com ANO de produção igual a 1972, os resultados para MAIOR_BILHETERIA, MENOR_BILHETERIA e MEDIA_BILHETERIA na Figura 7.5 são idênticos (R\$ 600.000,00).

```
mysql> CALL BILHETERIA_ANO(1972);
```

ANO	MEIOR_BILHETERIA	MENOR_BILHETERIA	MEDIA_BILHETERIA
2009	600.00	600.00	600.00

```
1 row in set (0.00 sec)
```

Figura 7.5: Resultado do Comando "CALL BILHETERIA_ANO(1972);"

Fonte: Elaborada pelo autor

Se quiser rever o conteúdo de um procedimento armazenado, basta usar o comando SHOW CREATE PROCEDURE:

```
mysql> SHOW CREATE PROCEDURE BILHETERIA_ANO;

CREATE DEFINER=`root`@`localhost` PROCEDURE `BILHETERIA_ANO`( IN
ANO INT )
BEGIN
  SELECT ANO, MAX(BILHETERIA) AS MAIOR_BILHETERIA,
         MIN(BILHETERIA) AS MENOR_BILHETERIA,
         AVG(BILHETERIA) AS MEDIA_BILHETERIA
  FROM FILMES
  WHERE ANO = ANOP;
END

1 row in set (0.00 sec)
```

Considere agora uma tabela chamada CONTA_CORRENTE:

```
mysql> CREATE TABLE CONTA_CORRENTE (
-> NR_CONTA INTEGER NOT NULL,
-> ID_AGENCIA INTEGER NOT NULL,
-> ID_CLIENTE INTEGER NOT NULL,
-> SALDO NUMERIC(8,2) NOT NULL,
-> PRIMARY KEY(NR_CONTA)
-> );
Query OK, 0 rows affected (0.09 sec)
```

Para transferir uma quantia de uma conta para outra, vamos criar um procedimento armazenado, conforme especificado logo abaixo.

```
mysql> DELIMITER //

mysql> CREATE PROCEDURE TRANSFERENCIA (
-> IN CONTA_ORIGEM INT,
-> IN CONTA_DESTINO INT,
-> IN VALOR NUMERIC(8,2) )
-> BEGIN
-> DECLARE SALDO_ORIGEM NUMERIC(8,2);
-> DECLARE SALDO_DESTINO NUMERIC(8,2);
-> SELECT SALDO INTO SALDO_ORIGEM
-> FROM CONTA_CORRENTE
-> WHERE NR_CONTA = CONTA_ORIGEM;
-> IF SALDO_ORIGEM < VALOR THEN
```

```

-> SELECT "FALHA NA TRANSFERENCIA – SALDO INSUFICIENTE";
-> ELSE
-> SELECT "TRANSFERENCIA AUTORIZADA – SALDO SUFICIENTE";
-> UPDATE CONTA_CORRENTE SET SALDO = (SALDO_ORIGEM - VALOR)
-> WHERE NR_CONTA = CONTA_ORIGEM;
-> SELECT "VALOR R$ ", VALOR, " RETIRADO DA CONTA ", CONTA_ORIGEM;
-> SELECT SALDO INTO SALDO_DESTINO
-> FROM CONTA_CORRENTE
-> WHERE NR_CONTA = CONTA_DESTINO;
-> UPDATE CONTA_CORRENTE SET SALDO = (SALDO_DESTINO+VALOR)
-> WHERE NR_CONTA = CONTA_DESTINO;
-> SELECT "VALOR R$ ", VALOR, " DEPOSITADO NA CONTA ", CON-
TA_DESTINO;
-> END IF;
-> END//
Query OK, 0 rows affected (0.16 sec)

mysql> DELIMITER ;

```

O procedimento armazenado recebe três parâmetros de entrada: CONTA_ORIGEM (número da conta-corrente de onde o valor será retirado), CONTA_DESTINO (número da conta-corrente onde o valor será depositado) e VALOR (quantia em dinheiro transferida de uma conta-corrente para outra).

Duas variáveis de memória foram criadas: SALDO_ORIGEM e SALDO_DESTINO. Ambas foram definidas como NUMERIC(8,2).

Em seguida, o procedimento armazenado executa uma consulta SQL (SELECT) em que a variável de memória SALDO_ORIGEM recebe o valor do SALDO da CONTA_CORRENTE em que NR_CONTA = CONTA_ORIGEM.

Um comando condicional IF avalia se SALDO_ORIGEM é menor que VALOR. Caso essa condição seja verdadeira, uma mensagem é exibida, informando que ocorreu uma **"FALHA NA TRANSFERÊNCIA – SALDO INSUFICIENTE"**. Por outro lado, se a condição for falsa, ou seja, se SALDO_ORIGEM for igual ou maior que o VALOR a ser transferido, outra mensagem informa que **"TRANSFERENCIA AUTORIZADA – SALDO SUFICIENTE"**. Se esta condição for falsa, o procedimento armazenado também altera o valor do SALDO da CONTA_CORRENTE em que NR_CONTA = CONTA_ORIGEM. Na verdade, o VALOR é retirado do SALDO (**"SET SALDO = (SALDO_ORIGEM – VALOR)"**).00

Então uma mensagem é exibida na tela informando que uma quantia igual a VALOR foi retirada da CONTA_CORRENTE de número igual a CONTA_ORIGEM.

A variável de memória SALDO_DESTINO é alimentada por uma consulta SQL com o SALDO da CONTA_CORRENTE em que NR_CONTA = CONTA_DESTINO. Ele é então atualizado com a soma de SALDO_DESTINO com VALOR transferido. Uma mensagem é exibida na tela para informar que o VALOR foi depositado na CONTA_CORRENTE em que NR_CONTA = CONTA_DESTINO.

Para testar esse procedimento armazenado, precisaremos antes inserir alguns registros na tabela CONTA_CORRENTE, conforme os comandos abaixo:

```
mysql> INSERT INTO CONTA_CORRENTE VALUES (107,662,335,2900.80);
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO CONTA_CORRENTE VALUES (129,662,335,200.10);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO CONTA_CORRENTE VALUES (108,662,343,55200.00);
Query OK, 1 row affected (0.00 sec)
```

Passamos a ter três contas correntes na tabela. Duas pertencem ao mesmo cliente.

```
mysql> SELECT * FROM CONTA_CORRENTE;
```

NR_CONTA	ID_AGENCIA	ID_CLIENTE	SALDO
107	662	335	2900.80
108	662	343	55200.00
129	662	335	200.10

3 rows in set (0.00 sec)

Figura 7.6: Conteúdo da tabela CONTA_CORRENTE

Fonte: Elaborada pelo autor

Para testar o procedimento armazenado, vamos tentar a transferência de R\$ 300,00 da conta 107 (com SALDO de R\$ 2.900,80) para a conta 129 (com SALDO de R\$ 200,10) com o comando da Figura 7.7.

```
mysql> CALL TRANSFERENCIA(107,129,300);
+-----+
| TRANSFERENCIA AUTORIZADA - SALDO SUFICIENTE |
+-----+
| TRANSFERENCIA AUTORIZADA - SALDO SUFICIENTE |
+-----+
1 row in set (0.06 sec)
+-----+
| VALOR RS | VALOR | RETIRADO DA CONTA | CONTA_ORIGEM |
+-----+
| VALOR RS | 300.00 | RETIRADO DA CONTA | 107 |
+-----+
1 row in set (0.08 sec)
+-----+
| VALOR RS | VALOR | DEPOSITADO NA CONTA | CONTA_DESTINO |
+-----+
| VALOR RS | 300.00 | DEPOSITADO NA CONTA | 129 |
+-----+
1 row in set (0.09 sec)
Query OK, 0 rows affected (0.11 sec)
```

Figura 7.7 Resultado do Comando “CALL TRANSFERENCIA(107,129,300);”

Fonte: Elaborada pelo autor

Como a consulta da Figura 7.8 comprova, tudo transcorreu como esperávamos. A conta 107, após a retirada de R\$ 300,00, ficou com SALDO de R\$ 2.600,00. Por sua vez, a conta 129 teve seu SALDO alterado para R\$ 500,10.

```
mysql> SELECT * FROM CONTA_CORRENTE;
+-----+
| NR_CONTA | ID_AGENCIA | ID_CLIENTE | SALDO |
+-----+
| 107 | 662 | 335 | 2600.80 |
| 108 | 662 | 343 | 55200.00 |
| 129 | 662 | 335 | 500.10 |
+-----+
3 rows in set (0.00 sec)
```

Figura 7.8: Conteúdo da tabela CONTA_CORRENTE atualizada

Fonte: Elaborada pelo autor

Ainda testando o procedimento armazenado TRANSFERENCIA (Figura 7.9), tentaremos transferir R\$ 900,00 da conta 129 para a conta 107. Como o VALOR a ser transferido é superior ao existente na conta 129, a transferência não poderá ocorrer.


```
mysql> CALL TRANSFERENCIA(129,107,900);
+-----+
| FALHA NA TRANSFERENCIA - SALDO INSUFICIENTE |
+-----+
| FALHA NA TRANSFERENCIA - SALDO INSUFICIENTE |
+-----+
1 row in set (0.06 sec)
Query OK, 0 rows affected (0.02 sec)
mysql> SELECT * FROM CONTA_CORRENTE;
+-----+-----+-----+-----+
| NR_CONTA | ID_AGENCIA | ID_CLIENTE | SALDO |
+-----+-----+-----+-----+
| 107 | 662 | 335 | 2600.80 |
| 108 | 662 | 343 | 55200.00 |
| 129 | 662 | 335 | 500.10 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Figura 7.9: Resultado do Comando "CALL TRANSFERENCIA(129,107,900);"

Fonte: Elaborada pelo autor

De fato, o procedimento armazenado TRANSFERENCIA não apenas exibiu uma mensagem informando que a transferência falhou por insuficiência de SALDO, como não permitiu alterações nos saldos das contas envolvidas.



Crie um procedimento armazenado de nome CONSULTA_FUNCIONARIO, com parâmetro de entrada MATRICULA e exiba os dados relativos ao funcionário dessa matrícula. O procedimento deverá lidar com a visão FUNC criada na atividade anterior (6.2) em vez da tabela FUNCIONARIO.

A-Z

Gatilhos (triggers)

São rotinas armazenadas que, associadas a eventos percebidos automaticamente pelo MySQL, não precisam ser invocadas explicitamente pelo usuário para serem executadas. Quando o evento programado acontece, o gatilho é disparado automaticamente.

7.3 Gatilhos

Os **gatilhos (triggers)** diferenciam dos procedimentos armazenados por não necessitarem do comando CALL para sua execução; ela fica condicionada à ocorrência de um evento predefinido. Ao ocorrer, o gatilho é "disparado" automaticamente.

Vamos criar duas novas tabelas em nosso BD EXEMPLO: PRODUTO e VENDA.

```
mysql> CREATE TABLE PRODUTO (
-> ID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
-> NOME VARCHAR(40) NOT NULL,
-> ESTOQUE INTEGER NOT NULL,
-> PRECO NUMERIC(6,2));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE VENDA (
-> NR_VENDA INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
```



```
-> ID INTEGER NOT NULL,  
-> QUANTIDADE INTEGER NOT NULL,  
-> PRECO NUMERIC(6,2));  
Query OK, 0 rows affected (0.02 sec)
```

Em seguida, vamos inserir dois novos registros na tabela PRODUTO.

```
mysql> INSERT INTO PRODUTO VALUES (1,"DVD PLAYER", 40,299.99);  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO PRODUTO VALUES (2,"APARELHO DE TELEVISÃO  
LCD", 20,1429.99);  
Query OK, 1 row affected (0.05 sec)
```

Assim, se fizermos uma consulta à tabela PRODUTO, obteremos o resultado da Figura 7.10, o qual demonstra que, no estoque da loja, existem 40 aparelhos de DVD e 20 aparelhos de TV com tela LCD.

```
mysql> START TRANSACTION;  
Query OK, 0 rows affected (0.00 sec)  
mysql> SELECT * FROM PRODUTO;  
+-----+-----+-----+-----+  
| ID | NOME | ESTOQUE | PRECO |  
+-----+-----+-----+-----+  
| 1 | DVD PLAYER | 30 | 299.99 |  
| 2 | APARELHO DE TELEVISAO LCD | 20 | 1429.99 |  
+-----+-----+-----+-----+  
2 rows in set (0.00 sec)  
mysql> INSERT INTO VENDA VALUSE (2,1,5,290);  
Query OK, 1 row affected (0.41 sec)  
mysql> UPDATE PRODUTO SET ESTOQUE = 25 WHERE ID = 1;;  
Query OK, 0 rows affected (0.00 sec)  
Rows matched: 1 Changed: 0 Warnings: 0  
mysql> COMMIT;  
Query OK, 0 rows affected (0.00 sec)
```

Figura 7.10: Conteúdo da tabela PRODUTO

Fonte: Elaborada pelo autor

Antes de inserirmos registros em VENDA, criaremos um gatilho (*trigger*) que será executado sempre que um novo registro de VENDA for inserido no BD.

```
mysql> DELIMITER //
```

```
mysql> CREATE TRIGGER ATUALIZANDO_ESTOQUE  
-> AFTER INSERT ON VENDA FOR EACH ROW BEGIN  
-> SET @VELHO_ESTOQUE = 0;  
-> SELECT ESTOQUE INTO @VELHO_ESTOQUE FROM PRODUTO WHERE
```

```

ID = NEW.ID;
-> UPDATE PRODUTO SET ESTOQUE = @VELHO_ESTOQUE - NEW.
QUANTIDADE WHERE ID = NEW.ID;
-> END//
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER ;

```

O gatilho (*trigger*) foi criado com o nome **ATUALIZANDO_ESTOQUE** ("**CREATE TRIGGER ATUALIZANDO_ESTOQUE**") e o evento que o dispara é a inserção de um novo registro na tabela **VENDA** ("**AFTER INSERT ON VENDA**"). Para cada linha ("**FOR EACH ROW**") inserida na tabela **VENDA** o gatilho cria uma variável chamada **@VELHO_ESTOQUE**. Essa variável é inicializada com zero, mas em seguida é atualizada com a quantidade antes da venda. **NEW.ID** corresponde ao valor da coluna ID inserido no novo ("NEW") registro de **VENDA**.

Em seguida, a coluna **ESTOQUE** da tabela **PRODUTO** é alterada com o resultado da subtração da quantidade vendida da quantidade em estoque do produto.

Agora vamos inserir um registro em **VENDA** e verificar as consequências dessa ação.

```

mysql> INSERT INTO VENDA VALUSE (1,1,10,298.50);
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM VENDA;
+-----+-----+-----+-----+
| NR_VENDA | ID | QUANTIDADE | PRECO |
+-----+-----+-----+-----+
|          1 | 1 |          10 | 298.50 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql> SELECT * FROM PRODUTO;
+-----+-----+-----+-----+
| ID | NOME | ESTOQUE | PRECO |
+-----+-----+-----+-----+
| 1 | DVD PLAYER | 30 | 299.99 |
| 2 | APARELHO DE TELEVISAO LCD | 20 | 1429.99 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

Figura 7.11: Conteúdo das tabelas **VENDA e **PRODUTO** após a venda de dez DVD players**

Fonte: Elaborada pelo autor

Observe que foram vendidos 10 aparelhos de DVD, descontados da coluna **ESTOQUE** em **PRODUTO**. Antes, havia 40 aparelhos de DVD e agora restam apenas 30. Não precisamos alterar o estoque, pois o gatilho nos poupou dessa tarefa.

Para o nosso próximo exemplo de gatilho, precisamos criar uma tabela para armazenar alterações efetuadas em registros da tabela FILMES.

```
mysql> CREATE TABLE FILMES_LOG(  
-> NR_ENTRADA INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,  
-> ID_VELHO INTEGER,  
-> ID_NOVO INTEGER,  
-> TITULO_VELHO VARCHAR(60),  
-> TITULO_NOVO VARCHAR(60),  
-> ANO_VELHO INTEGER,  
-> ANO_NOVO INTEGER,  
-> BILHETERIA_VELHA NUMERIC(6,2),  
-> BILHETERIA_NOVA NUMERIC(6,2)      );  
Query OK, 0 rows affected (0.00 sec)
```

Assim, a tabela FILMES_LOG armazena os dados de antes da alteração do registro (ID_VELHO, TITULO_VELHO, ANO_VELHO e BILHETERIA_VELHA), assim como os dados após essa alteração (ID_NOVO, TITULO_NOVO, ANO_NOVO, BILHETERIA_NOVA).

```
mysql> DELIMITER //  
  
mysql> CREATE TRIGGER MANTER_LOG_ALTERACAO  
-> AFTER UPDATE ON FILMES FOR EACH ROW BEGIN  
-> INSERT INTO FILMES_LOG ( ID_VELHO, ID_NOVO, TITULO_VELHO,  
TITULO_NOVO, ANO_VELHO, ANO_NOVO, BILHETERIA_VELHA, BILHETE-  
RIA_NOVA )  
-> VALUES ( OLD.ID, NEW.ID, OLD.TITULO, NEW.TITULO, OLD.ANO,  
NEW.ANO, OLD.BILHETERIA, NEW.BILHETERIA );  
-> END //  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> DELIMITER ;
```

O gatilho acima dispara sempre que uma alteração é produzida em algum registro da tabela FILMES, inserindo um novo registro na tabela FILMES_LOG. Para obter o valor antigo de uma coluna, utilizamos o prefixo "**OLD.**". Analogamente, para obter o novo valor de uma coluna utilizamos o prefixo "**NEW**". Alteramos simultaneamente o TITULO e ANO de produção do FILME com ID = 5 (Figura 7.12).

```
mysql> SELECT * FROM FILMES;
+----+-----+-----+-----+
| ID | TITULO          | ANO  | BILHETERIA |
+----+-----+-----+-----+
| 1  | TRANSFORMERS    | 2007 | 390.89      |
| 2  | TRANSFORMERS 2  | 2009 | 990.90      |
| 3  | STAR TREK       | 2009 | 598.99      |
| 4  | WOLVERINE - ORIGENS | 2009 | 391.00      |
| 5  | THE GODFATHER   | 1972 | 600.00      |
+----+-----+-----+-----+
5 rows in set (0.02 sec)
mysql> UPDATE FILMES SET TITULO = "O PODEROSO CHEFAO", ANO = 1973
-> WHERE ID = 5;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Figura 7.12: Conteúdo da tabela FILMES antes da alteração de um registro

Fonte: Elaborada pelo autor

A alteração foi bem-sucedida (Figura 7.13) e o gatilho gerou um novo registro na tabela – criada de FILMES_LOG (Figura 7.14).

```
mysql> SELECT * FROM FILMES;
+----+-----+-----+-----+
| ID | TITULO          | ANO  | BILHETERIA |
+----+-----+-----+-----+
| 1  | TRANSFORMERS    | 2007 | 390.89      |
| 2  | TRANSFORMERS 2  | 2009 | 990.90      |
| 3  | STAR TREK       | 2009 | 598.99      |
| 4  | WOLVERINE - ORIGENS | 2009 | 391.00      |
| 5  | O PODEROSO CHEFAO | 1973 | 600.00      |
+----+-----+-----+-----+
5 rows in set (0.02 sec)
```

Figura 7.13: Conteúdo da tabela FILMES após a alteração de um registro

Fonte: Elaborada pelo autor

```
mysql> SELECT TITULO_NOVO, TITULO_VELHO, ANO_NOVO, ANO_VELHO
FROM FILMES_LOG;
+-----+-----+-----+-----+
| TITULO_NOVO | TITULO_VELHO | ANO_NOVO | ANO_VELHO |
+-----+-----+-----+-----+
| O PODEROSO CHEFAO | THE GODFATHER | 1973 | 1972 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Figura 7.14: Conteúdo de alguns atributos da tabela FILMES_LOG

Fonte: Elaborada pelo autor

Um último exemplo de gatilho traz como novidade o evento antes da exclusão na tabela FILMES (“**BEFORE DELETE ON FILMES**”).

```
mysql> DELIMITER //
```

```
mysql> CREATE TRIGGER MANTER_LOG_EXCLUSAO
-> BEFORE DELETE ON FILMES FOR EACH ROW BEGIN
-> INSERT INTO FILMES_LOG (ID_VELHO, TITULO_VELHO,
-> ANO_VELHO, BILHETERIA_VELHA)
```

```
-> VALUES (OLD.ID, OLD.TITULO, OLD.ANO, OLD.BILHETERIA);  
-> END //
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> DELIMITER ;
```

Antes da exclusão de registro da tabela FILMES_LOG (Figura 7.15), o gatilho MANTER_LOG_EXCLUSAO é efetuado, inserindo um novo registro na tabela FILMES_LOG.

```
mysql> DELETE FROM FILMES WHERE ID = 3;  
Query OK, 1 row affected (0.01 sec)  
mysql> SELECT * FROM FILMES;  
+-----+-----+-----+-----+  
| ID | TITULO           | ANO | BILHETERIA |  
+-----+-----+-----+-----+  
| 1 | TRANSFORMERS     | 2007 | 390.89 |  
| 2 | TRANSFORMERS 2    | 2009 | 990.90 |  
| 4 | WOLVERINE - ORIGENS | 2009 | 391.00 |  
| 5 | O PODEROSO CHEPAO | 1973 | 600.00 |  
+-----+-----+-----+-----+  
4 rows in set (0.00 sec)  
mysql> SELECT ID_VELHO, TITULO_VELHO, ANO_VELHO,  
BILHETERIA_VELHA FROM FILMES_LOG;  
+-----+-----+-----+-----+  
| ID_VELHO | TITULO_VELHO | ANO_NOVO | BILHETERIA_VELHA |  
+-----+-----+-----+-----+  
| 5 | THE GODFATHER   | 1972 | 600.00 |  
| 3 | STAR TREK       | 2009 | 598.99 |  
+-----+-----+-----+-----+  
2 rows in set (0.02 sec)
```

Figura 7.15: Conteúdo das tabelas FILMES e FILMES_LOG após exclusão de registro

Fonte: Elaborada pelo autor

Resumo

Nesta aula aprendemos a criar rotinas em BDs: procedimentos armazenados (*Stored Procedure*) ou gatilhos (*Triggers*). Também tratamos da conveniência do uso dessas rotinas. Vimos que enquanto um procedimento armazenado deve ser explicitamente invocado para que seja executado, os gatilhos são associados a eventos de BDs, não precisando da instrução CALL; eles são executados, automaticamente, pelo SGBD antes ou depois da ocorrência de um dado evento.

Atividades de aprendizagem

Crie um gatilho (*trigger*) a ser disparado sempre que uma exclusão for efetuada na tabela FUNCIONARIO (a mesma Atividade do final da Aula 6). O gatilho deverá armazenar os dados do registro excluído na tabela FUNCIONARIOS_EXCLUIDOS, a qual deverá apresentar as seguintes colunas:

```
NR_EXCLUSAO INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,  
DATA_EXCLUSAO DATE NOT NULL,  
MATRICULA INTEGER NOT NULL PRIMARY KEY,  
NOME VARCHAR(50) NOT NULL,  
SEXO CHAR(1) NOT NULL,  
TELEFONE VARCHAR(15),  
SETOR INTEGER NOT NULL,  
SALARIO NUMERIC(6,2), e  
SENHA VARCHAR(20).
```

Aula 8 – Criação de índices

Objetivos

Definir o conceito de índice no contexto da tecnologia de Banco de Dados.

Descrever a importância dos índices para os sistemas de Bancos de Dados.

Criar índices durante a criação das tabelas.

Criar novos índices alterando tabelas existentes.

Agora vamos tratar dos “bastidores” ou “subterrâneos” da tecnologia de BD: parte do trabalho que ninguém vê e poucos valorizam, mas que, quando falha, resulta em muitas reclamações por parte dos usuários finais.

O resultado da criação de um índice não é algo que salte aos olhos como em uma consulta SQL. Porém, a ausência de um índice pode acarretar em demora por parte do BD para retornar o resultado de uma pesquisa. O usuário final pode até não entender a causa dessa demora, mas certamente vai se impacientar.

8.1 Índices

Quando fazemos uma consulta em uma ou mais tabelas, o BD pode ter de lidar com uma massa considerável de registros. Imagine o trabalho que gera a busca de um registro em uma tabela com 7 mil registros. Se a busca for sequencial e o registro procurado estiver ocupando a posição 5.354, o banco começará a busca a partir do primeiro registro e verificará, um a um, cada registro, até que finalmente encontre o registro desejado ou chegue ao final da tabela. Nesse caso, o BD efetuará a leitura em exatamente 5.354 registros até finalmente encontrar o procurado.

A-Z

Índice (INDEX)

É um recurso utilizado pelos BDs para a rápida localização de registros em tabelas; trata-se de um arquivo que funciona de maneira semelhante ao índice de um livro ou o catálogo de uma biblioteca. Conforme Korth, Silberschatz e Sudarshan (2006), quando procuramos um livro por um determinado autor, recorremos a um catálogo de autores e uma ficha de papel nos diz exatamente onde localizá-lo. Para facilitar nossa pesquisa, as fichas do catálogo são mantidas em ordem alfabética. Isso evita que tenhamos de olhar ficha a ficha. Normalmente pegamos qualquer ficha, avaliamos se ela está alfabeticamente antes ou depois da ficha que procuramos. Se estiver antes, então descartamos as que estão antes da ficha em avaliação e nos dedicamos a procurar nas fichas subsequentes. Dessa forma, a busca é mais eficiente. Localizamos a ficha mais rapidamente. Em BD, em vez de catálogos, há tabelas e no lugar de fichas de papel temos registros.

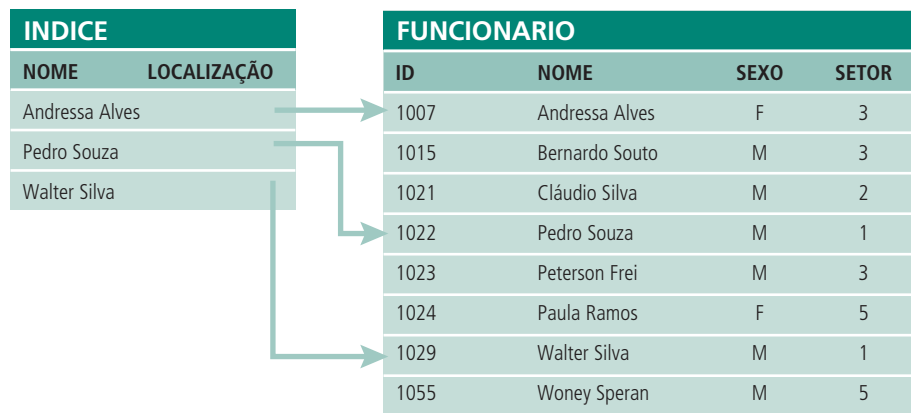


Figura 8.1: Exemplo de Banco de Dados – modelo em rede

Fonte: Elaborada pelo autor

A Figura 8.1 mostra um esquema de sua tabela (FUNCIONARIO) e respectivo índice. Esse índice é relativo à coluna NOME da tabela FUNCIONARIO.

```
SELECT *
FROM FUNCIONARIO
WHERE NOME = "WOLNEY SPERAN";
```

Uma busca sequencial, sem uso do arquivo de índice, pelo funcionário de NOME = "WOLNEY SPERAN" resultaria na leitura de oito registros da tabela. A consulta SQL começaria pelo primeiro registro (NOME = "ANDRESSA ALVES") e, um a um, passaria por todos até encontrar o FUNCIONARIO ou chegar ao final da tabela.

Por outro lado, se o projetista desse BD tiver criado um arquivo de índice para a coluna NOME na tabela FUNCIONARIO, a consulta deixa de ser sequencial e o registro é localizado mais rapidamente. Os arquivos de índices são menores que as tabelas e, por essa razão, ocupam relativamente pouco espaço de memória principal. Aproveitando-se dessa vantagem, sempre que abrimos uma tabela ("FROM FUNCIONARIO"), todos os seus arquivos de índices são carregados para a memória principal.

Uma consulta que busque o FUNCIONARIO "WOLNEY SPERAN" (**WHERE NOME = "WOLNEY SPERAN"**) começa por seu arquivo de índice (quando este existe). A primeira linha do arquivo de índice apresenta o nome "ANDRESSA ALVES" e um ponteiro com o endereço do respectivo registro na tabela FUNCIONARIO na memória secundária. Como "WOLNEY SPERAN" (o registro procurado) está alfabeticamente após "ANDRESSA ALVES", o BD segue para a próxima linha do arquivo de índices. Como "WOLNEY SPERAN" também está alfabeticamente após "PEDRO SOUZA", o BD passa para a próxima linha do arquivo de índices.

Apesar de “WOLNEY SPERAN” também estar alfabeticamente após “WALTER SILVA”, não há mais linhas para prosseguirmos. Com isso, o banco de dado lê a tabela FUNCIONARIO na posição indicada pela coluna LOCALIZACAO. Assim, o primeiro registro lido por nossa consulta na tabela FUNCIONARIO tem NOME = “WALTER SILVA”. A partir desse ponto faz-se uma busca sequencial do registro com NOME = “WOLNEY SPERAN”.

Ao usar o arquivo de índice, o BD evitou ler oito registros da tabela FUNCIONARIO. No exemplo acima, apenas dois registros foram lidos. Portanto, o arquivo de índice tornou a consulta mais eficiente em termos do tempo de resposta.

De maneira análoga, a consulta exemplificada abaixo resultará na leitura de um registro na tabela FUNCIONARIO, em vez da leitura de quatro registros de uma busca sequencial:

```
SELECT *  
FROM FUNCIONARIO  
WHERE NOME = “PEDRO SOUZA”;
```

Em resumo, um índice de BD é semelhante ao índice ou sumário de um livro o qual apresenta bem menos páginas que o restante do livro, e cada informação corresponde ao nome de um capítulo ou tópico com a indicação do número da página de seu começo.

8.2 Criando um índice durante a criação da tabela

Toda chave primária possui um arquivo de índice (INDEX) para as colunas que a formam, já criado, automaticamente, pelo próprio BD. Mas, colunas que não fazem parte da chave primária, mas que provavelmente serão muito utilizadas para a localização de registros nas consultas SQL, merecem a criação de um índice.

```
mysql> CREATE TABLE ARTISTA (  
-> ID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,  
-> NOME VARCHAR(40) NOT NULL, INDEX(NOME));  
Query OK, 0 rows affected (0.13 sec)
```

O comando acima cria um índice (INDEX) para a coluna NOME durante a criação da tabela ARTISTA. A coluna ID, por ser a chave primária já terá um índice para si.

A criação de um índice para NOME foi estratégica. Afinal, não é difícil imaginar que um usuário não acostumado com os identificadores (ID) criados para cada ARTISTA, venha a solicitar buscas pela coluna NOME:

```
mysql> SELECT * FROM ARTISTA WHERE NOME LIKE "%LENNON%";  
Empty set (0.08 sec)
```

O critério de pesquisa utilizado é o de NOME com a *string* "LENNON" em qualquer parte de seu conteúdo. Como criamos um índice, a leitura não será sequencial.

```
mysql> SHOW INDEX FROM ARTISTA;
```

O comando "**SHOW INDEX FROM <NOME-DA-TABELA>;**" exibe dados de todos os índices criados para uma tabela específica (Figura 8.2).

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment
ARTISTA	0	PRIMARY	1	ID	A	0			NULL	BTREE	
ARTISTA	1	NOME	1	NOME	A	0			NULL	BTREE	

Figura 8.2: tabela do Comando SHOW INDEX

Fonte: Elaborada pelo autor

Os dados podem aparecer um tanto bagunçados em razão das restrições de tamanho da tela e do papel. Mas, mesmo assim, podemos recuperar algumas informações sobre os índices da tabela ARTISTA. Repare que há uma coluna de nome *Column_name* em que aparecem os valores "ID" (na primeira linha) e "NOME" (na outra linha). O índice para ID é um índice que aceita somente valores únicos (não repetidos): *NON_UNIQUE* = 0, uma vez que ID é a chave primária de ARTISTA. Por outro lado, o índice baseado na coluna NOME aceita repetições de valores (*NON_UNIQUE* = 1).

8.3 Criando um índice para tabela preexistente

Podemos criar um índice (INDEX) para uma tabela já existente contendo ou não registros. O índice pode ser criado em qualquer fase da existência de uma tabela.

Vamos supor que a tabela ARTISTA já tivesse sido criada sem a definição de um índice para a coluna NOME.

```
mysql> CREATE TABLE ARTISTA (ID INTEGER NOT NULL PRIMARY KEY  
AUTO_INCREMENT, NOME VARCHAR(40) NOT NULL);  
Query OK, 0 rows affected (0.13 sec)
```

Em um momento posterior ao da construção da tabela, resolvemos criar o índice (INDEX) para a coluna NOME da seguinte maneira:

```
mysql> CREATE INDEX INDICE_NOME ON ARTISTA (NOME);  
Query OK, 0 rows affected (0.01 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

O comando anterior cria um índice de nome INDICE_NOME ("**CREATE INDEX INDICE_NOME**") a partir da tabela ARTISTA ("**ON ARTISTA**"), usando a coluna NOME como base ("**(NOME);**").

Também podemos criar um índice baseado apenas em parte do NOME do ARTISTA.

```
mysql> CREATE INDEX INDICE_NOME_ABREV ON ARTISTA (NOME(10));  
Query OK, 0 rows affected (0.03 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

O comando acima cria um índice com os dez primeiros caracteres do conteúdo da coluna NOME.

Resumo

Nesta aula destacamos a importância dos índices (*index*) para a tecnologia de BD. Vimos como criar índices ainda durante a criação de tabelas e como criá-los posteriormente, alterando a definição das tabelas.

Atividades de aprendizagem

A tabela com nome EXAME_ LABORATORIAL apresenta cinco colunas (Figura 8.3). São elas: ID_EXAME (Identificador do Exame – nossa chave primária), COD_AMB (código usado pela Associação Médica Brasileira para identificar o exame), NOME_CIENTIFICO, NOME (o nome abreviado do exame) e DESCRICAO_MATERIAL (descrição do material utilizado no exame: sangue, fezes, urina, etc.).

EXAME_LABORATORIAL
ID_EXAME
COD_AMB
NOME_CIENTIFICO
NOME
DESCRICAO_MATERIAL

Figura 8.3: Exemplo de tabela de EXAME_LABORATORIAL

Fonte: Elaborado pelo autor

Como regra geral, sabe-se que as colunas frequentemente utilizadas para a filtragem de registros em consultas SQL (cláusula WHERE) precisam de índices para agilizar o tempo de resposta dessas consultas.

Com base na tabela EXAME_LABORATORIAL, escreva os comandos de criação de índices para as colunas em que tal procedimento seja justificável.

Aula 9 – Transações em Banco de Dados

Objetivos

Conceituar transação no contexto dos sistemas de Banco de Dados.

Implementar transações no SGBD MySQL através de comandos SQL.

Quando, no início deste material, argumentamos que a tecnologia de BD veio a substituir os sistemas de arquivos tradicionais, suprimindo as deficiências destes últimos, vimos que uma dessas deficiências era a impossibilidade de assegurar a integridade dos dados armazenados, a qual pode ser remediada por meio de transações.

9.1 Conceito de transação

Os sistemas de informação estão sujeitos a paradas e falhas de variadas causas: defeito de *hardware*, interrupção do fornecimento de energia, erros de *software*, falha de comunicação entre dispositivos distribuídos, incêndio nas instalações, sabotagem, etc.

Em termos da tecnologia de BD é fundamental assegurar a consistência dos dados; mesmo que algumas dessas falhas ocorram. Suponha que o registro de uma VENDA tenha sido efetuado em nosso BD. A empresa servida por essa tecnologia vendeu uma unidade de *notebook*. Como existiam 20 unidades antes da VENDA, após esta, deverão existir no ESTOQUE apenas 19 unidades de *notebooks*. Se após a VENDA constar nos registros qualquer outra quantidade de *notebooks* no ESTOQUE, podemos afirmar que o BD não mais se encontra em situação consistente. A integridade dos dados não foi assegurada. Como isso poderia ter acontecido?

Imagine que após o registro da VENDA, mas imediatamente antes da atualização do ESTOQUE, ocorra uma interrupção no suprimento de energia. Com isso, os dados que ainda estavam na memória principal ou nos registradores foram perdidos. A VENDA já havia sido devidamente armazenada na memória secundária, mas a mudança no ESTOQUE ainda não. Em consequência há uma situação de inconsistência dos dados.

Sempre que um programa aplicativo acessar e atualizar diversos registros de uma ou mais tabelas, seu programador tem por responsabilidade defini-lo como uma transação, pois o esquecimento pode implicar um grave risco para a integridade de um sistema.

A-Z

Transação

Pode ser definida como um conjunto de comandos de programação que, após sua execução, ainda preserva a consistência do BD. Caso o BD estivesse consistente antes da execução da transação, podemos estar certos de que permanecerá consistente após sua execução. Conforme Korth, Silberschatz e Sudarshan (2006), toda transação é atômica: ou todas as instruções de uma transação são executadas, ou então nenhuma delas é. A possibilidade de apenas parte das instruções associadas à transação ser executada é simplesmente inexistente. É uma questão de “tudo ou nada”.

Quando uma **transação** completa sua execução com sucesso, dizemos que ela foi **compromissada**. Por outro lado, quando não consegue concluir sua execução com sucesso, a **transação** assume a condição de **abortada**. A regra da atomicidade exige que todas as ações perpetradas por uma **transação abortada** devem ser **desfeitas**, de maneira que essa transação não tenha efeito sobre o estado do Banco de Dados.

Para assegurar a consistência dos dados, transações costumam implementar um histórico incremental com atualizações adiadas. Considere nosso exemplo da venda de um *notebook* quando, antes da venda ser efetuada, a situação no BD fosse a da Figura 9.1.

PRODUTO	
DESCRICAO	ESTOQUE
Notebook	20
TV Plasma	13

VENDA		
NR VENDA	DESCRICAO	QUANTIDADE

LOG DO SISTEMA	

Figura 9.1: Situação anterior à transação

Fonte: Elaborada pelo autor

O cliente chega ao caixa para pagar o *notebook* e a transação é iniciada: um registro é inserido no arquivo de LOG_DO_SISTEMA (Figura 9.2) para sinalizar que a transação **T1** foi inicializada (“<**T1**, INÍCIO>”).

A transação **T1** foi iniciada, mas ainda não foi concluída. Então o caixa registra a venda do *notebook* (um registro deve ser gravado na tabela VENDA). Porém, antes que isso aconteça, um novo registro é inserido no arquivo de LOG_DO_SISTEMA, informando a inserção na tabela de VENDA.

PRODUTO	
DESCRICAO	ESTOQUE
Notebook	20
TV Plasma	13

VENDA		
NR VENDA	DESCRICAO	QUANTIDADE

LOG DO SISTEMA	
<T1, INÍCIO>	

Figura 9.2: Início da transação T1

Fonte: Elaborada pelo autor

Note na Figura 9.3 o formato do registro no arquivo de LOG_DO_SISTEMA ("**<T1, Venda, Notebook, 1>**"). Ele indica que a transação T1 inseriu um registro de venda de 1 unidade de *notebook*.

PRODUTO	
DESCRICAO	ESTOQUE
Notebook	20
TV Plasma	13

VENDA		
NR VENDA	DESCRICAO	QUANTIDADE

LOG DO SISTEMA	
<T1, INÍCIO>	
<T1, Venda, Notebook, 1>	

Figura 9.3: Transação em andamento - log do sistema atualizado

Fonte: Elaborada pelo autor

Suponha que uma falha grave ocorra nesse exato momento e o BD tenha de ser reiniciado. A transação ainda não foi comprometida e o BD tem "consciência" de sua condição, pois há um registro **<T1, INÍCIO>** no arquivo LOG_DO_SISTEMA, mas não um registro **<T1, FIM>**. Portanto, a transação não foi concluída/comprometida.

Então, o BD aborta a transação T1. Contudo, nenhuma transação deve ser apenas parcialmente executada – como determina a regra do "tudo ou nada". Assim, o BD cancela todas as operações executadas por T1. No arquivo LOG_DO_SISTEMA encontra uma referência à inclusão de um registro na tabela VENDA. Contudo, como não encontra esse registro na tabela, não precisa excluí-lo.

Porém, vamos imaginar que nenhuma falha ocorreu até aqui. O próximo passo da transação T1 é incluir o registro na tabela VENDA (Figura 9.4).

PRODUTO		LOG DO SISTEMA	
DESCRICAO	ESTOQUE	<T1, INÍCIO>	
Notebook	20	<T1, Venda, Notebook, 1>	
TV Plasma	13		

VENDA		
NR VENDA	DESCRICAO	QUANTIDADE
1	Notebook	1

Figura 9.4: Transação em andamento: atualização da tabela VENDA

Fonte: Elaborada pelo autor

Se alguma falha ocorrer nesse exato momento, o Banco de Dados perceberá que a transação T1 não foi concluída e tratará de excluir o registro inserido em VENDA.

Por outro lado, se nenhuma falha ocorrer até aqui, o arquivo LOG_DO_SISTEMA (Figura 9.5) receberá um novo registro (“<T1, Produto, Notebook, 20, 19>”). Este sinaliza que o registro da tabela PRODUTO relativo ao *notebook* será alterado de tal forma que a coluna ESTOQUE – que tinha valor 20 – pasará a ter valor 19.

PRODUTO		LOG DO SISTEMA	
DESCRICAO	ESTOQUE	<T1, INÍCIO>	
Notebook	20	<T1, Venda, Notebook, 1>	
TV Plasma	13	<T1, Produto, Notebook, 20, 19>	

VENDA		
NR VENDA	DESCRICAO	QUANTIDADE
1	Notebook	1

Figura 9.5: Transação em andamento: nova atualização do log do sistema

Fonte: Elaborada pelo autor

Em seguida, a tabela PRODUTO é de fato alterada e o ESTOQUE de Notebook passa a apresentar valor 19. Embora todas as operações da transação tenham sido efetivamente executadas, a transação ainda não foi compromissada, pois o registro **<T1, FIM>** ainda não foi inserido no arquivo LOG_DO_SISTEMA.

Por essa razão, caso uma falha ocorra nesse momento, o Banco de Dados ainda irá desfazer todas as operações associadas a **T1**. Ao ler o registro **<T1, Venda, <IT>Notebook<IT>, 1>** o Banco de Dados entenderá que deve excluir o registro de VENDA relativo ao Notebook. Por sua vez, ao ler o registro **<T1, Produto, <IT>Notebook<IT>, 20, 19>** o banco resolve alterar o registro de PRODUTO relativo ao Notebook, substituindo o novo valor de ESTOQUE (=19) pelo velho valor da mesma coluna (=20) (Figura 9.6).

Dessa forma, todas as operações levadas a cabo pela transação T1 serão desfeitas. Para todos os efeitos, é como se a transação jamais tivesse existido. O Banco de Dados permanece consistente, pois o ESTOQUE terá um valor condizente com a realidade.

PRODUTO	
DESCRICAO	ESTOQUE
Notebook	19
TV Plasma	13

VENDA		
NR_VENDA	DESCRICAO	QUANTIDADE
1	Notebook	1

LOG_DO_SISTEMA
<T1, INÍCIO>
<T1, Venda, Notebook, 1>
<T1, Produto, Notebook, 20, 19>

Figura 9.6: Transação em vias de ser compromissada

Fonte: Elaborada pelo autor

No fim da transação, um registro **<T1, FIM>** é inserido no arquivo LOG_DO_SISTEMA. De agora em diante, qualquer falha ocorrida não levará o Banco de Dados a desfazer as operações de **T1** (Figura 9.7).

PRODUTO		
DESCRICAO	ESTOQUE	
Notebook	19	
TV Plasma	13	

VENDA		
NR_VENDA	DESCRICAO	QUANTIDADE
1	Notebook	1

LOG_DO_SISTEMA	
<T1, INÍCIO>	
<T1, Venda, Notebook, 1>	
<T1, Produto, Notebook, 20, 19>	
<T1, FIM>	

Figura 9.7: Transação finalmente compromissada (COMMIT)

Fonte: Elaborada pelo autor

A simples existência dos registros <T1,INICIO> e <T1,FIM> caracteriza a transação T1 como compromissada.

9.2 Transações no MySQL

Por padrão, o SGBD MySQL apresenta configuração no modo AUTOCOMMIT. Isso significa que qualquer atualização de tabela – INSERT, DELETE e UPDATE – é acompanhada da execução automática do comando COMMIT (compromissar). Portanto, cada vez que um comando INSERT, DELETE ou UPDATE é executado, o MySQL trata de confirmar as alterações automaticamente, tornando as transações compromissadas.

```
SET AUTOCOMMIT=0
```

O comando acima altera a configuração do MySQL, desabilitando o AUTOCOMMIT. Por outro lado, o comando abaixo volta a habilitar o AUTOCOMMIT.

```
SET AUTOCOMMIT=1
```

O comando START TRANSACTION inicializa uma transação, enquanto o comando COMMIT leva a transação ao estado de compromissada e o comando ROLLBACK aborta a transação.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM PRODUTO;
```

ID	NOME	ESTOQUE	PRECO
1	DVD PLAYER	30	299.99
2	APARELHO DE TELEVISAO LCD	20	1429.99

```
2 rows in set (0.00 sec)
mysql> INSERT INTO VENDA VALUSE (2,1,5,290);
Query OK, 1 row affected (0.41 sec)
mysql> UPDATE PRODUTO SET ESTOQUE = 25 WHERE ID = 1;;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1 Changed: 0 Warnings: 0
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Figura 9.8: Sequência de comandos para a realização de transações no MySQL

Fonte: Elaborada pelo autor

A sequência de comandos da Figura 9.8 exemplifica a utilização de transações no MySQL. O comando "START TRANSACTION;" inicia a transação. Em seguida fazemos uma consulta (SELECT) para visualizar o ESTOQUE dos produtos. Então, inserimos um novo registro na tabela VENDA (INSERT) e atualizamos (UPDATE) um registro de PRODUTO. Finalmente, confirmamos todas as atualizações com o comando COMMIT ("compromissar"). Se tivéssemos concluído a transação com o comando ROLLBACK, todas as atualizações teriam sido desfeitas.

Se uma falha qualquer ocorresse entre os comandos INSERT e UPDATE, o próprio MySQL executaria o comando ROLLBACK. Porém, isso somente seria possível se, antes dos comandos INSERT e UPDATE, o comando START TRANSACTION fosse executado. Se o usuário do BD se esquecer desse último comando, a transação simplesmente não existe e falhas poderiam conduzir o BD a uma situação de inconsistência.

Resumo

Nesta aula conhecemos o importante conceito de transação em Banco de Dados. Notamos como essa tecnologia é de vital importância para a integridade dos dados e para aquelas situações em que devemos garantir que os dados de diferentes tabelas sejam modificados simultaneamente. Em tais situações ou todas as tabelas envolvidas são atualizadas ou nenhuma delas deve ser atualizada. Também exploramos a sintaxe SQL para a implementação de transações em Bancos de Dados MySQL. Como visto na atividade proposta ao final desta aula, as transações podem ser utilizadas em procedimentos armazenados ou gatilhos.

Atividades de aprendizagem

Na Aula 7, foi apresentado um exemplo de procedimento armazenado (*stored procedure*) para a transferência de valores de uma conta-corrente para outra.

Esse procedimento armazenado de nome TRANSFERENCIA recebe como parâmetros de entrada (1) o número da conta-corrente de onde o dinheiro será retirado, (2) o número da conta onde será depositado e (3) o valor a ser transferido.

Apesar de pronto e testado, esse procedimento armazenado não faz uso de transação. Sua tarefa é alterá-lo para que passe a trabalhar com uma transação. Ela impedirá a possibilidade de um valor debitado em uma conta não ser creditado em outra.

<Solução da Atividade 9.1:

DELIMITER //

CREATE PROCEDURE TRANSFERENCIA (

-> IN CONTA_ORIGEM INT,

-> IN CONTA_DESTINO INT,

-> IN VALOR NUMERIC(8,2))

-> BEGIN

-> DECLARE SALDO_ORIGEM NUMERIC(8,2);

-> DECLARE SALDO_DESTINO NUMERIC(8,2);

-> SELECT SALDO INTO SALDO_ORIGEM

-> FROM CONTA_CORRENTE

-> WHERE NR_CONTA = CONTA_ORIGEM;

-> IF SALDO_ORIGEM < VALOR THEN

-> SELECT "FALHA NA TRANSFERENCIA – SALDO

-> INSUFICIENTE";

-> ELSE

-> SELECT "TRANSFERENCIA AUTORIZADA – SALDO

-> SUFICIENTE";

-> START TRANSACTION;

-> UPDATE CONTA_CORRENTE SET SALDO =

-> (SALDO_ORIGEM - VALOR)

-> WHERE NR_CONTA = CONTA_ORIGEM;

-> SELECT "VALOR R\$",VALOR,"RETIRADO DA CONTA"

-> ,CONTA_ORIGEM;

-> SELECT SALDO INTO SALDO_DESTINO

-> FROM CONTA_CORRENTE

-> WHERE NR_CONTA = CONTA_DESTINO;

-> UPDATE CONTA_CORRENTE SET SALDO = (SALDO_DESTINO

-> + VALOR)

-> WHERE NR_CONTA = CONTA_DESTINO;

-> COMMIT;

-> SELECT "VALOR R\$ ",VALOR, " DEPOSITADO NA CONTA",

-> CONTA_DESTINO;

-> END IF;

-> END//

DELIMITER ;>

Referências

CHEN, Peter. **Modelagem de dados:** a abordagem entidade-relacionamento para projeto lógico. Tradução de Cecília Camargo Bartalotti. São Paulo: Makron Books, 1990.

DATE, C. J. **Bancos de dados:** introdução aos sistemas de bancos de dados. Tradução de Hélio Auro Golveia. 8. ed. Rio de Janeiro: Campus, 2004.

ELMASRI, R.; NAVATHE, E. **Sistemas de Bancos de Dados.** Tradução de Marília Guimarães Pinheiro. 4. ed. São Paulo: Pearson Addison Wesley, 2005.

FALBO, R. A. **Projeto de sistemas:** notas de aula. Disponível em: <<http://ebookbrowse.com/projeto-estruturado-de-sistemas-programa-ricardo-falbo-2010-1-pdf-d53398277>>. Acesso em: 28 jun. 2011.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados.** Porto Alegre: Sagra Luzzato, 2004.

KORTH, Henry F.; SILBERSCHATZ, Abraham; SUDARSHAN, S. **Sistema de Banco de Dados.** Tradução de Marília Guimarães Pinheiro e Cláudio César Canhette. Rio de Janeiro: Campus, 2006.

KORTH, Henry F.; SILBERSCHATZ, Abraham; SUDARSHAN, S. Database System Concepts. Disponível em: <<http://www.db-book.com>>. Acesso em: 23 mar. 2011.

LÉVY, Pierre. **As tecnologias da inteligência:** o futuro do pensamento na era da informática. Tradução de Carlos Irineu da Costa. Rio de Janeiro: Editora 34, 1993.

TAHAGHOGHI, S.; WILLIAMS, H. **Aprendendo MySQL.** Tradução de Alonso Dias. Rio de Janeiro: Alta Books, 2007.

Currículo do professor-autor

Vanderson José Ildefonso Silva é mestre em Informática pela Universidade Federal do Espírito Santo (UFES). Graduado em Ciências Econômicas e pós-graduado em Análise de Sistemas pela mesma Instituição de Ensino Superior. Atua na educação a distância desde 2009, inicialmente como professor especialista e conteudista da disciplina de Economia e Finanças (CEAD) e posteriormente como professor especialista e conteudista da disciplina de Banco de Dados (e-Tec). Também é professor do IFES (Campus Colatina) desde 1996, trabalhando nos Cursos Técnicos de Informática, Superior em Tecnologia de Redes de Computadores e Bacharelado em Sistemas de Informação.



