

PÓS-Graduação em Distância



Banco de Dados

Prof. Paulo Sergio Borba



Sumário

Apresentação	5
Módulo 1 – Introdução a banco de dados	6
Aula 1 – Banco de dados relacionais	6
Conceito de banco de dados	6
SGBD – sistema gerenciador de banco de dados	6
Vantagens em utilizar banco de dados	7
Banco de dados relacional	8
Arquitetura de sistemas de banco de dados	8
Aula 2 – MySQL	10
História do MySQL	10
Características principais do MySQL	10
Instalando o MySQL	10
MySQL Workbench	15
Aula 3 – A linguagem SQL	17
SQL – Linguagem para banco de dados relacional	17
Principais comandos SQL	18
Exercícios do Módulo 1	18
Módulo 2 – Modelagem de dados	21
Aula 4 – Modelos de dados e elementos do modelo relacional	21
Modelos de dados	21
Elementos de um modelo relacional	22
Chaves primária e estrangeira	23
Aula 5 – Organização dos dados em tabelas	24
Estruturando dados em tabelas	24
Diagrama entidade relacionamento (DER)	26
Cardinalidade	27
Regra para determinar a cardinalidade	27
Exemplo de um DER	28
Mapeamento	29
Modelo de dados	29
Tipo de dado	31
Normalização de dados	32
Regras de integridade e consistência	36
Exercícios do Módulo 2	38
Módulo 3 – Criação de tabelas e manipulação de dados	42
Aula 6 – Criando um banco de dados e objetos	42
Criando um banco de dados – projeto curso	42
Criando tabelas no banco de dados	44
O comando <i>insert</i> – inserindo dados em uma tabela	50
O comando <i>select</i> – consultando dados em uma tabela	51
Aula 7 – Mais opções de uso dos comandos <i>insert</i> e <i>select</i>	56
Criando tabelas com relacionamentos	56
JOIN - Consultando dados de várias tabelas em um <i>select</i>	58
Select distinct — suprimindo repetições	61
Aula 8 – Alterando e excluindo dados em tabelas	62
Update – alterando dados de tabelas	62

Delete – eliminando dados de tabelas	63
O comando <i>truncate</i> – limpando os dados de uma tabela	64
Exercícios do módulo 3	65
Módulo 4 – recursos do MySQL	69
Aula 9 – Funções do MySQL	69
Funções matemáticas	69
Funções para manipulação de <i>string</i>	70
Funções de data e hora	71
Funções de conversão	73
Funções estatísticas	74
Aula 10 – Exercitando <i>join</i> e <i>group by</i> em um novo DB	75
Carregando uma massa de dados para um novo banco de dados	75
Join com várias tabelas em um comando SELECT	75
GROUP BY – cláusula de agrupamento em um SELECT	79
Usando a cláusula HAVING em um comando SELECT	85
Aula 11 – Objetos de código armazenado: <i>view</i> - <i>procedure</i> - <i>trigger</i>	87
Create view – criação e uso de visões	87
Create procedure – criação e uso de <i>stored procedures</i>	93
Create trigger	99
Exercícios do Módulo 4	109
Considerações Finais	113
Módulo 5 – mais comandos na prática	114
Aula 12 – Utilizando sub-select	114
Sub-select com IN e NOT IN	114
Sub-select com Join	118
Aula 13 – A cláusula CASE dentro de um SELECT	120
Aula 14 – Variações da cláusula JOIN	123
Tipos de JOIN	123
Aula 15 – Alguns casos especiais de modelagem e DML	128
Autorrelacionamento	128
Especialização	132
Duplo relacionamento	135
Exercícios do Módulo 5	140
Respostas Comentadas dos Exercícios	142
Módulo 1	142
Módulo 2	143
Módulo 3	148
Módulo 4	152
Módulo 5	160
Bibliografia	164

Apresentação

Caro(a) aluno(a), você já cumpriu etapas importantes deste curso e esta disciplina ajudará você com soluções ligadas a banco de dados.

Cada vez mais as informações são úteis às empresas e são cada vez mais valorizadas. Cresce a necessidade de se armazenar mais e mais informações para que sejam exploradas das mais diversas formas. Passamos a guardar quase tudo eletronicamente, reduzindo ou eliminando a utilização e o uso de documentos físicos. Isso gerou a necessidade de dispositivos de hardware mais modernos, com maior capacidade de armazenamento e, claro, mais velozes.

Com o software não foi diferente e as aplicações também evoluíram tanto para armazenar e manipular quantidades enormes de informação quanto para fazer o tratamento e o processamento de imagens.

Os bancos de dados nasceram e evoluíram justamente para tratar volumes maiores de informação e se transformaram em um componente tecnológico quase indispensável para a maioria das organizações. O mundo hoje é dependente dos sistemas de transações que manipulam dados. Quase a unanimidade desses sistemas apoiam-se em sistemas gerenciadores de banco de dados para o tratamento dos dados das mais diversas aplicações e com as mais diversas finalidades.

O objetivo desta disciplina é apresentar o banco de dados como uma ferramenta ou componente do desenvolvimento de sistemas e para isso estudaremos os conceitos básicos de bancos de dados, a modelagem dos dados (projeto de banco de dados), os SGBDs e comandos para a criação de banco de dados e a manipulação dos dados através da linguagem SQL.

Paulo Borba

Módulo 1 – Introdução a banco de dados

Caros alunos, este módulo apresenta uma introdução à tecnologia de banco de dados, introduzindo sua conceituação, vantagens de uso e o modelo relacional, que é o modelo utilizado de forma quase unânime atualmente.

Aula 1 – Banco de dados relacionais

Conceito de banco de dados

Um banco de dados (BD) é um sistema de computador capaz de armazenar dados e informações para posterior busca e recuperação desses dados.

Os bancos de dados permitem armazenar grandes quantidades de dados organizados de forma inteligente a fim de permitir facilmente sua posterior busca e recuperação. O banco de dados é muito mais eficiente e proporciona muito mais recursos à manipulação de informações que os arquivos isolados.

SGBD – sistema gerenciador de banco de dados

O SGBD é um software que gerencia todo o qualquer acesso externo aos dados armazenados no banco de dados.

Na figura 1.2, podemos ver que os dados ficam armazenados em tabelas dentro do banco de dados. Os softwares, aplicativos que precisam efetuar operações com dados (seja busca de dados já existentes, inclusão de novos dados, alteração ou exclusão dos dados já existentes), não acessam diretamente as tabelas de dados.

Os aplicativos devem se conectar ao banco de dados através de um usuário e senha (autenticação) e solicitar ao SGBD que execute sobre os dados os acessos que precisam.

Desta forma, se um aplicativo necessita buscar uma informação no banco de dados, ele não acessa diretamente os dados, mas faz a solicitação ao SGBD que utilizará seus mecanismos próprios para acessar os dados e devolvê-los ao aplicativo. O mesmo vale para uma inclusão, alteração ou exclusão de dados.

SGBD (Sistema Gerenciador de Banco de Dados)

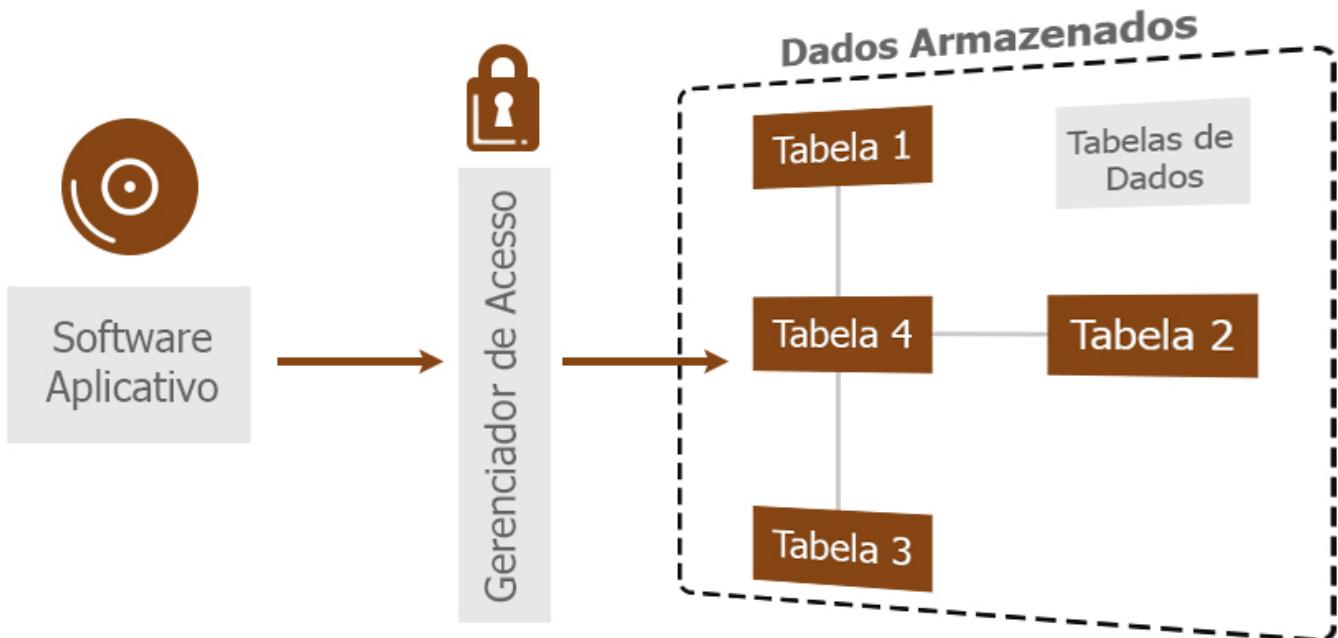


Figura 1.2 – SGBD: Controle de acesso aos dados. Fonte: adaptado de DATE (2000).

Vantagens em utilizar banco de dados

A utilização de tecnologia de banco de dados nos proporciona uma série de vantagens, entre elas as listadas no que se segue:

Tipo	Descrição
Dados podem ser compartilhados	Uma mesma tabela de dados ou conjunto de tabelas pode ser acessado por várias aplicações que podem ter visões parciais diferentes desses dados, dependendo dos objetivos de cada aplicação. Com isso é comum reduzirmos o esforço de manter os dados e possibilita o benefício de permitir que mais usuários os explorem.
Redução de redundâncias	O compartilhamento permite que várias aplicações acessem os mesmos dados ou parte deles. Isso evita que cada aplicação precise manter um conjunto próprio de arquivos isolados e cativos da aplicação. Com isso, informações repetidas que estariam em arquivos isolados de aplicações, passam a ser centralizadas e armazenadas uma única vez.
Independência dos aplicativos das estruturas de dados	As aplicações não precisam se preocupar em cuidar da estrutura de dados e parte da manutenção de dados é realizada pelo banco de dados. Assim os aplicativos: vêm apenas os dados que lhes interessam; não precisam saber detalhes de como seus dados estão fisicamente gravados; não precisam ser modificados se a estrutura de dados for alterada e não afetar diretamente a porção de dados acessada.
Mecanismos de acesso	O SGBD já possui mecanismos de acesso às tabelas de dados. Assim, os aplicativos devem apenas fazer as solicitações ao SGBD, em alto nível, que se ocupará de tratar os “detalhes” do acesso aos dados e retornará às informações solicitadas pelos aplicativos.

Tipo	Descrição
Segurança	O banco de dados não permite acesso direto aos dados, permite apenas acesso aos dados a partir de uma conexão com usuário e senha. Para cada usuário é permitido acesso a uma parte dos dados e esse acesso pode ser apenas de leitura ou de leitura e gravação, dependendo do acesso necessário para aquele usuário.
Consistência dos dados	Ao se inserir ou alterar dados, devem ser observadas algumas regras de validação, como o tipo de dado (numérico ou <i>string</i>), tamanho e conteúdo. O banco de dados efetua uma série de verificações, automaticamente, dispensando que todas elas tenham que ser feitas nos programas aplicativos.

Banco de dados relacional

Alguns tipos de banco de dados surgiram no mercado, como o hierárquico e o de rede. Porém, o banco de dados relacional predominou e, atualmente, é praticamente o único tipo utilizado. O BD relacional armazena os dados em **tabelas**, que têm sua estrutura formada por **linhas** e **colunas**. Cada tabela abriga informações de um determinado **assunto**. Assim, deve-se criar uma tabela para cada assunto que se deseja armazenar, por exemplo: **cliente, vendedor, pedido**.

Colunas	Indicam a estrutura da tabela ou que tipo de informações a tabela abriga a respeito do assunto, ou seja, de cada elemento da tabela. Na linguagem relacional, também chamamos de atributos da tabela ou atributos dos elementos da tabela (por exemplo: nome, idade, sexo...).	
Linhas	As linhas da tabela possuem dados a respeito de seus elementos. Cada linha de uma tabela tem dados de um elemento.	

Quando um pedido novo é incluído não é necessário constar o nome do cliente por extenso, permanece apenas uma referência (código) do cliente que se relaciona à tabela — na qual está o cadastro completo do cliente. O mesmo se aplica ao vendedor.

As vantagens de trabalhar desta forma são:

- redução de redundância — escreve-se menos, sem a necessidade de se repetir informação;
- integridade — garante-se que o nome estará igual em todas as ocorrências (se o nome fosse digitado N vezes, poderia ser digitado de forma diferente por um erro de digitação);
- redução de espaço — ao contrário do que pode parecer, ao segregar a informação em várias tabelas, o tamanho total em bytes se reduz (por exemplo, ao invés de escrever o nome várias vezes, escreve-se uma vez na tabela de cadastro e nas demais apenas um código que faz referência à primeira tabela).

Arquitetura de sistemas de banco de dados

De acordo com Date (2000), a arquitetura de um sistema de banco de dados tem como principal objetivo separar aplicações do usuário dos dados físicos e se divide em três níveis de representação, ilustrados na figura 1.5.

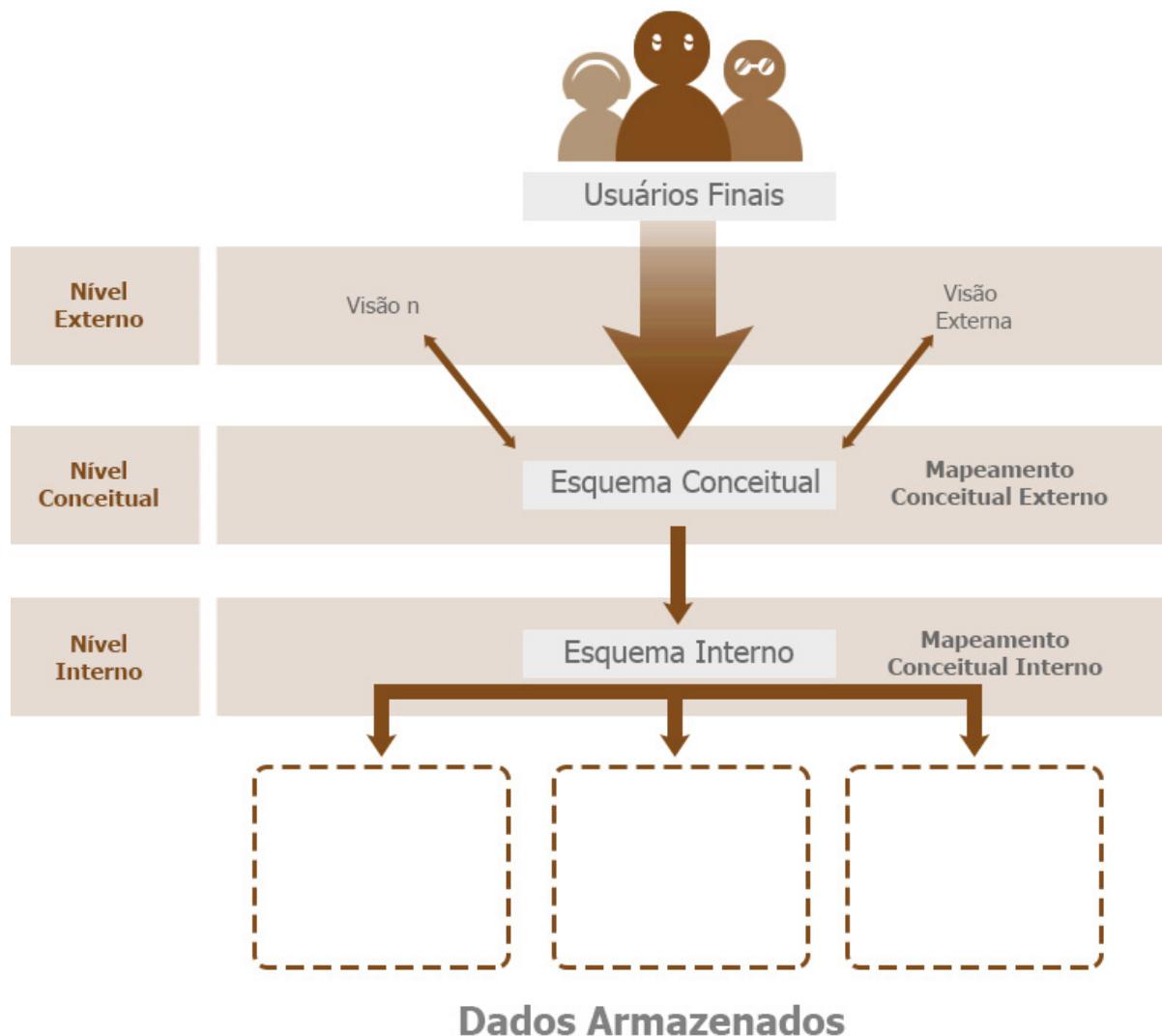


Figura 1.5 – Arquitetura de Um SGBD.

De acordo com Date (2000), estes níveis (ou esquemas) são assim definidos:

Nível externo	É o nível mais distante da implementação física. Apresenta visões parciais, do ponto de vista do usuário, do que está nos níveis abaixo. Há usuários que, dependendo de seu perfil, têm visões distintas da informação.
Nível conceitual	É o nível em que representamos as tabelas e objetos que estão no banco de dados físico. Podemos utilizar vários tipos de representação, a maioria delas gráficas.
Nível interno	É o nível de armazenamento, ou seja, nível físico. É o formato como os dados estão no banco de dados. Utilizamos comandos da DDL para definir as tabelas e demais objetos armazenados no banco de dados.

Aula 2 – MySQL

História do MySQL

De acordo com Manzano (2007), o MySQL surgiu na Suécia criado pelos suecos David Axmark, Allan Larsson e pelo finlandês Michael “Monty” Widenius. As primeiras ideias são de 1979. Juntos criaram um programa de BD, o UniReg, que não era padrão SQL, escrito em *basic*. Em 1994, Monty começou a trabalhar no desenvolvimento de um banco de dados de código aberto, baseado no UniReg e no mSQL. Isto inspirou o trio a iniciar o desenvolvimento do MySQL, em 1995, constituindo a empresa chamada MySQL AB.



O MySQL foi se popularizando e tornou-se um dos softwares de gerenciamento de banco de dados mais utilizados no planeta. Desde sua versão inicial até as versões mais atuais, foi incorporando recursos que não tinha, como integridade referencial (abordaremos mais à frente).

Em 2008, a MySQL AB foi vendida para a Sun Microsystems que passou a deter os direitos do MySQL, mesmo sendo um software de código aberto e licença livre. No ano seguinte, a Oracle comprou a Sun Microsystems, com todos seus produtos, e passou a ser o detentor do MySQL.

Características principais do MySQL

O MySQL é um SGBD completo e um dos mais utilizados atualmente. De acordo com Manzano (2007) e Milani (2007), entre suas principais características estão:

- excelente desempenho e estabilidade;
- pouco exigente quanto a recursos de hardware;
- oferece portabilidade, pois suporta as principais plataformas como windows, linux, unix, mac os server;
- oferece suporte a desenvolvimento, pois integra-se com diversas linguagens de programação como Java, PHP, C/C++, C#, Python, Perl, ASP e Ruby;
- oferece facilidade no manuseio;
- é um software livre;
- pode utilizar vários *engines* de armazenamento de dados como MyISam, InnoDB, Falcon, BDB, Archive, Federated, CSV, Solid.

Instalando o MySQL

Os downloads do MySQL são gratuitos e há mais de um lugar em que você pode obtê-lo, caro aluno. Você pode obter os arquivos de instalação diretamente do site do MySQL: <<http://www.mysql.com/downloads/>>.

Para este curso, fizemos a opção de fazer o download de um conjunto chamado XAMPP. Isto porque, além do MySQL, ele traz alguns recursos adicionais e uma interface web simples e prática para se trabalhar. Faremos a instalação em Windows 7 e todos os exemplos mostrados utilizarão este ambiente. Faça as adaptações, caso esteja usando uma plataforma diferente.

Siga as etapas a seguir para instalar o XAMPP (MySQL).

- 1** Faça download do XAMPP. Há vários sites de onde se pode baixá-lo gratuitamente. A versão que utilizamos é: **xampp-win32-1.8.3-2-VC11-installer.exe** (caso esta versão tenha sido substituída por outra mais recente, utilize-a).
- 2** Proceda à instalação do XAMPP, executando o arquivo que você obteve. A instalação é simples. Siga os passos do instalador conforme ilustra a figura 2.3.



Figura 2.3 – Etapas de Instalação.

A Oracle TM adquiriu os direitos sobre o MySQL e passou a ser responsável por sua evolução. Veja as motivações de se optar pelo MySQL. Aponte sua curiosidade para: <<http://www.mysql.com/why-mysql/>>. Acesso em: 28 maio 2014. Veja também o documento: <<http://www.mysql.com/why-mysql/white-papers/top-10-reasons-to-choose-mysql-for-next-generation-web-applications/>>. Acesso em: 28 de maio 2014.

Saiba Mais

Carregando o ambiente do MySQL

Para usar o MySQL é necessário carregar seu ambiente MySQL. Siga os passos:

- 1 Para começar a trabalhar com o MySQL, ative os serviços na Central do XAMPP e ative os serviços do Apache e do MySQL, conforme mostram as figuras 2.4a e 2.4b.

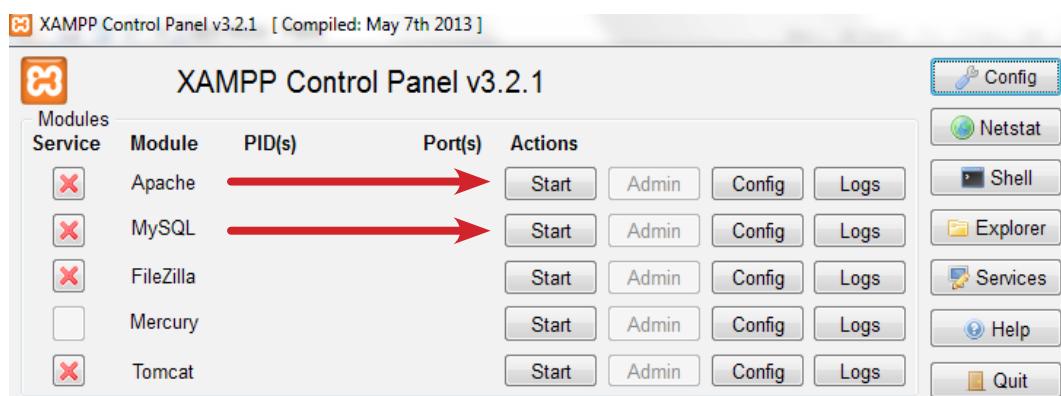
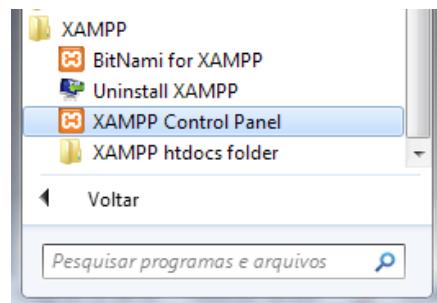


Figura 2.4a – Ativando Serviços do Apache e do MySQL.

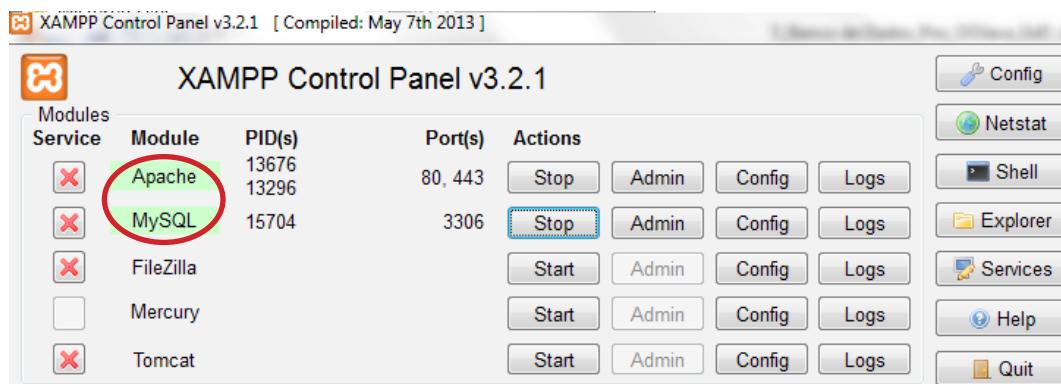


Figura 2.4b – Status após Ativação dos Serviços.

Para acessar a interface do MySQL, utilize um browser com o endereço “**localhost**”. Depois acesse a opção do MySQL no menu “Tools”: **phpMyAdim** (veja figura 2.4c).

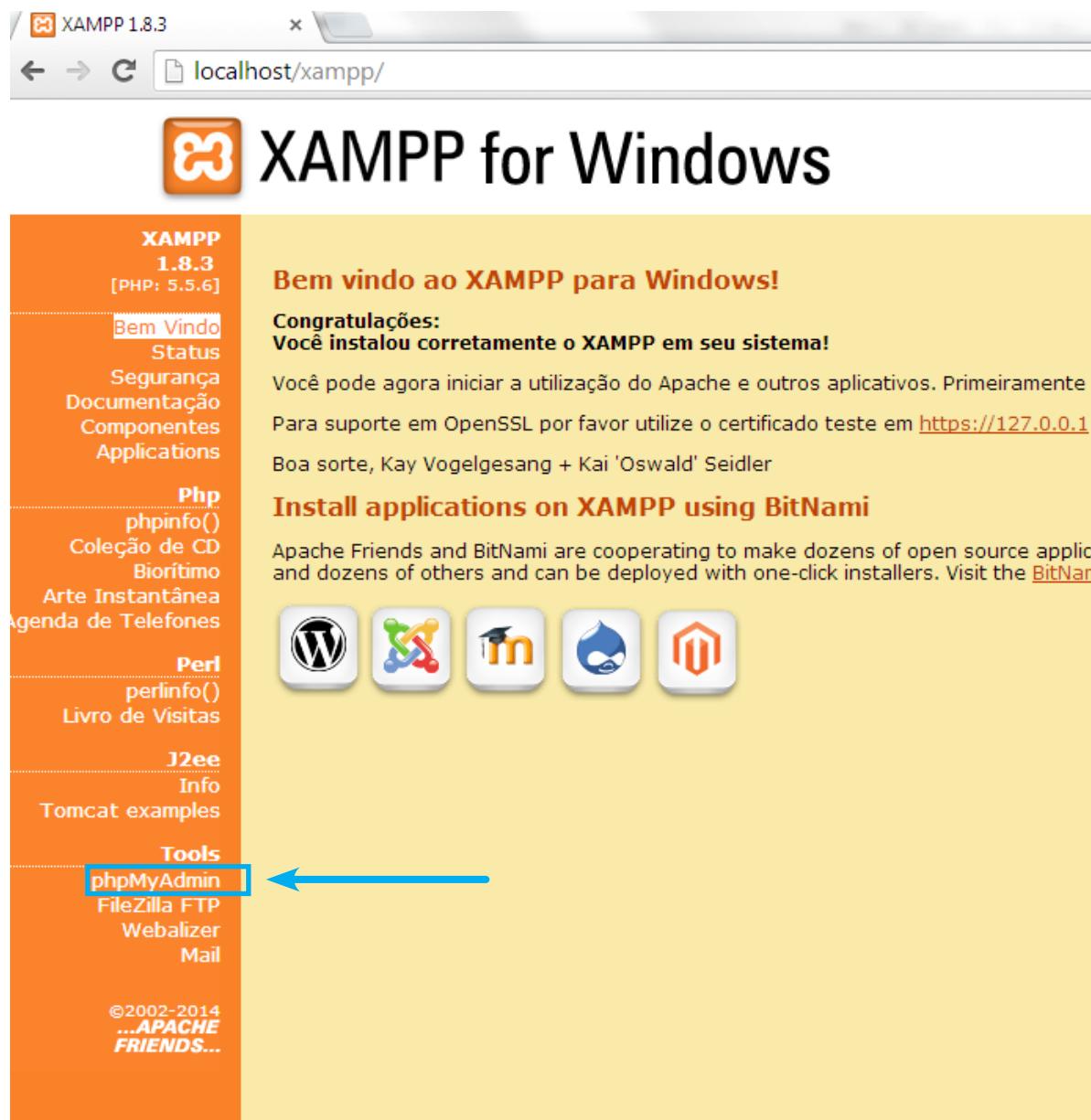


Figura 2.4c – Menu Geral do XAMPP.

A figura 2.4d mostra a tela principal do **phpMyAdim**. Aqui você tem uma interface interessante para criar suas tabelas e testar seus comandos MySQL.

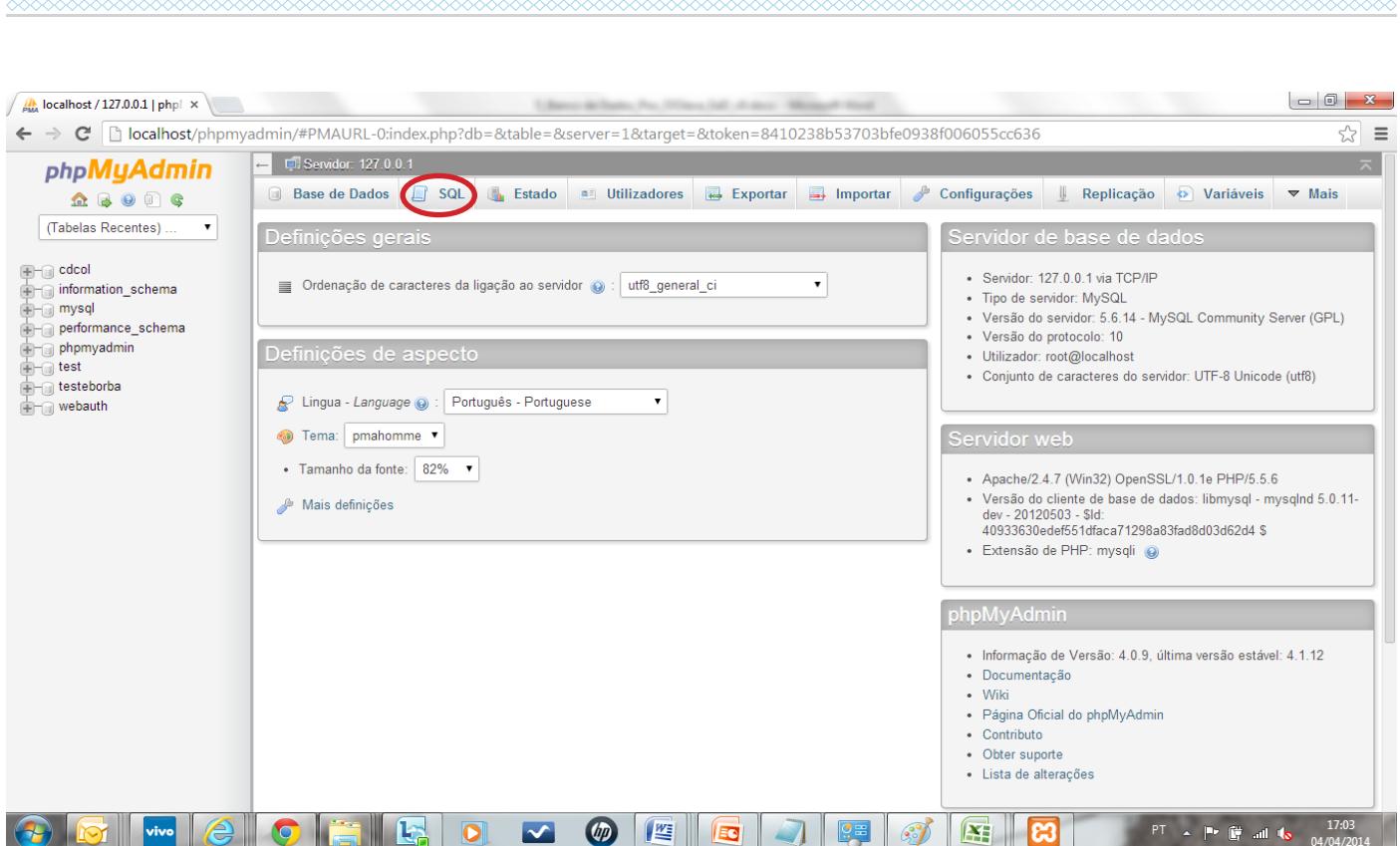


Figura 2.4d – Tela Principal da Interface phpMyAdmin. Fonte: elaborado pelo autor.

Acessando o Menu SQL, você tem acesso a uma janela na qual pode digitar e executar os comandos SQL (veja figura 2.4e). Usaremos este espaço para testar nossos comandos.

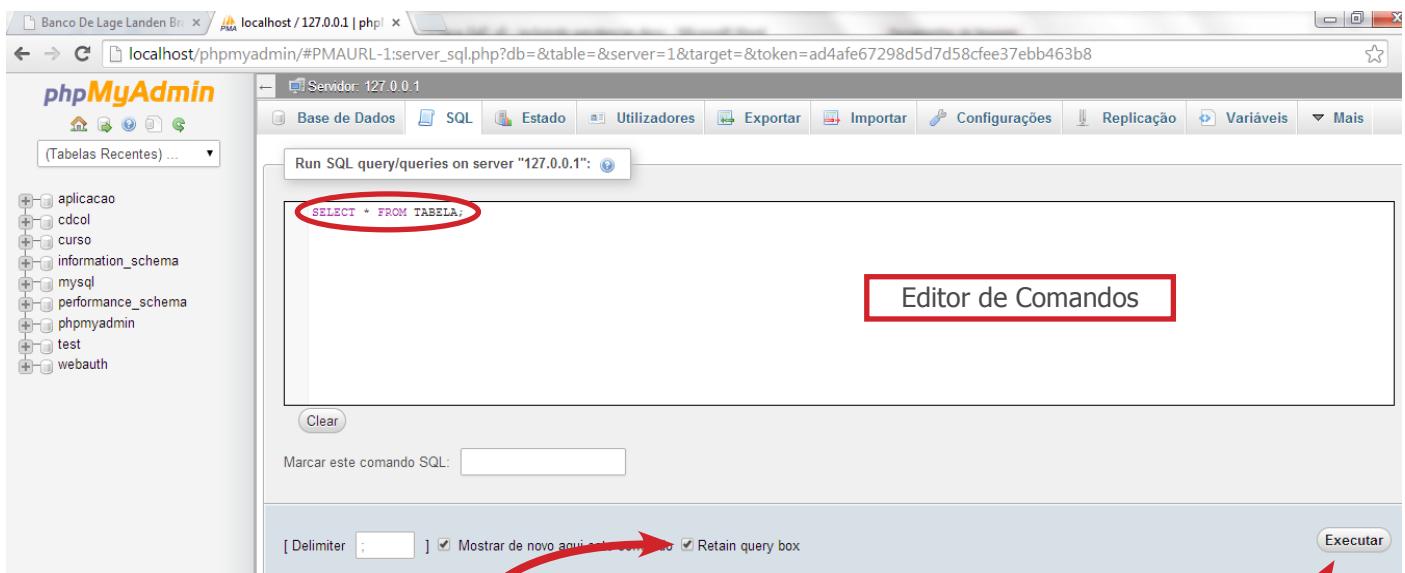


Figura 2.4e – Interface para digitação e execução de comandos SQL.

Deixe esta caixa de opção sempre marcada para preservar o último comando executado.

Clique neste botão para executar os comandos descritos na caixa de diálogo acima.

MySQL Workbench

Na elaboração de um sistema, antes de criarmos as estruturas de tabelas e outros objetos dentro do SGBD, é necessário fazer um projeto do banco de dados, criando um modelo de dados para normalizar suas tabelas. Este modelo pode ser preparado manualmente, mas há ferramentas apropriadas para criação destes modelos que apresentam inúmeras vantagens, além da manutenção futura do modelo.

Há várias ferramentas no mercado com essas características. Uma das melhores está na *suite* do MySQL: o MySQL WorkBench, conforme ilustra a figura 2.5a.

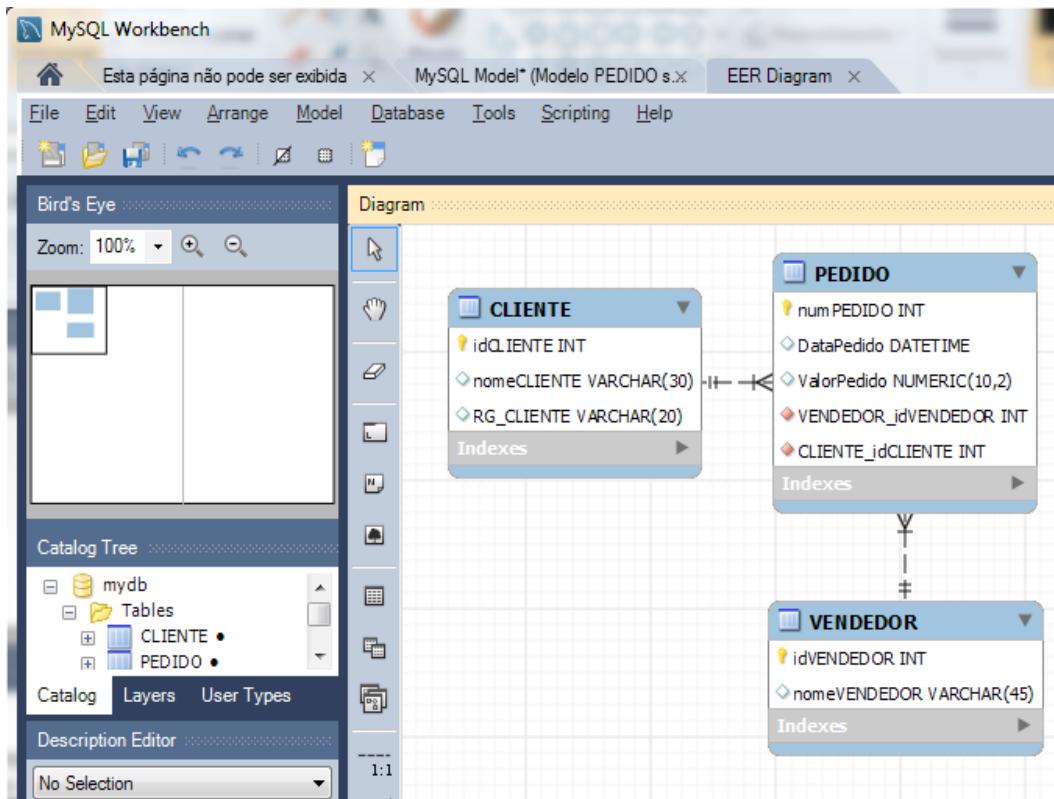


Figura 2.5a – imagem ilustrativa do WorkBench.

Para obter este software, acesse: <<http://www.mysql.com/downloads/>>. Acesso em: 28 maio 2014.

Dica: desça até o rodapé da página e selecione na lista a opção “MySQL Workbench”, faça o download da versão adequada para sua plataforma.

Neste curso, instalamos a versão deste executável: **mysql-workbench-community-6.0.9-win32.msi**. Para instalá-lo, dê duplo clique sobre este arquivo. (Utilize sempre a versão mais recente, caso esta tenha sido substituída).

Há alguns pré-requisitos para a instalação do MySQL Workbench. O mais importante é o Visual C++. Caso você não tenha este software instalado em seu computador, durante a instalação do MySQL WorkBench você

receberá uma mensagem de aviso sobre isso. A figura 2.5b ilustra a sequência de instalação. Siga os passos do instalador.

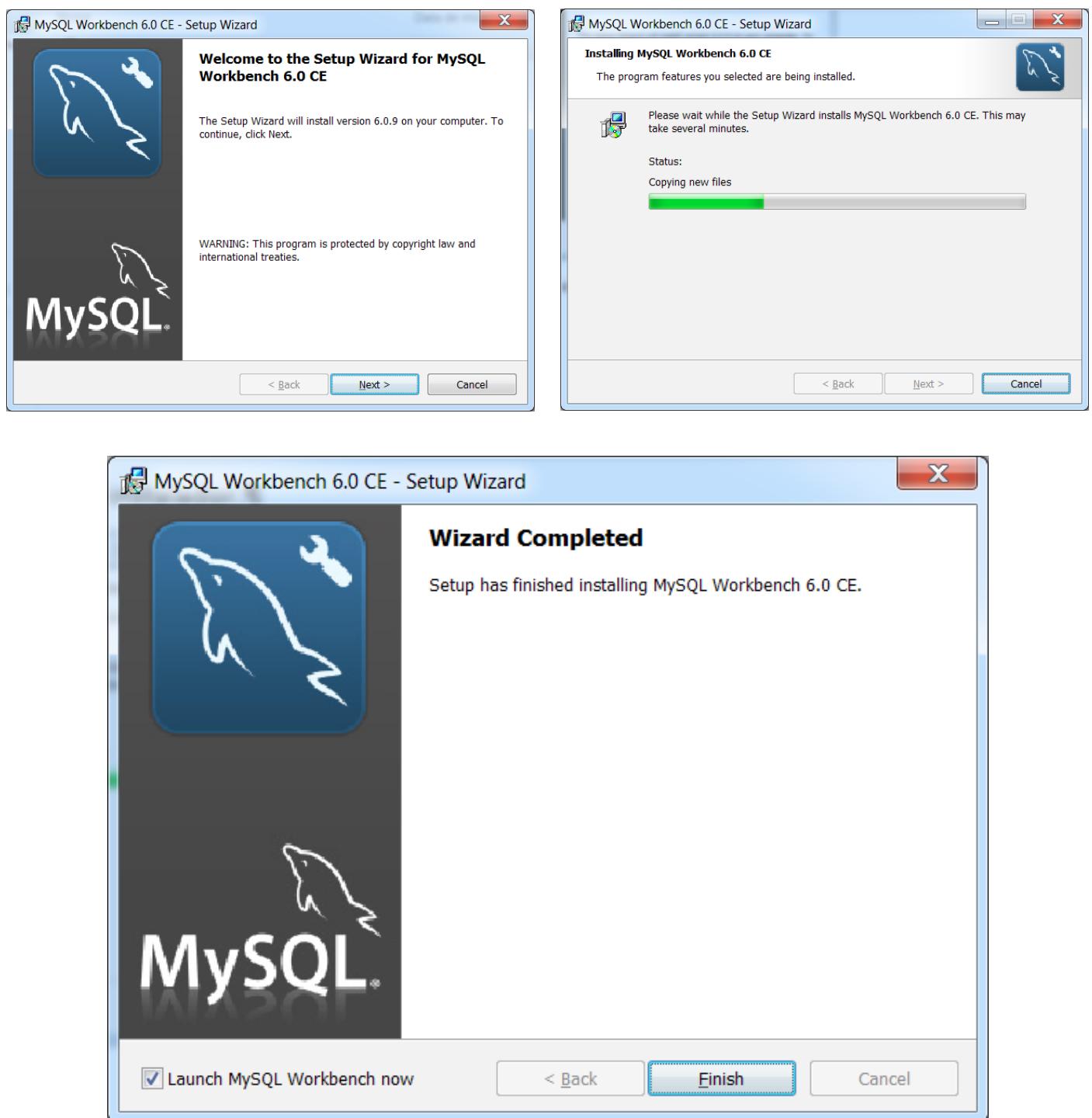
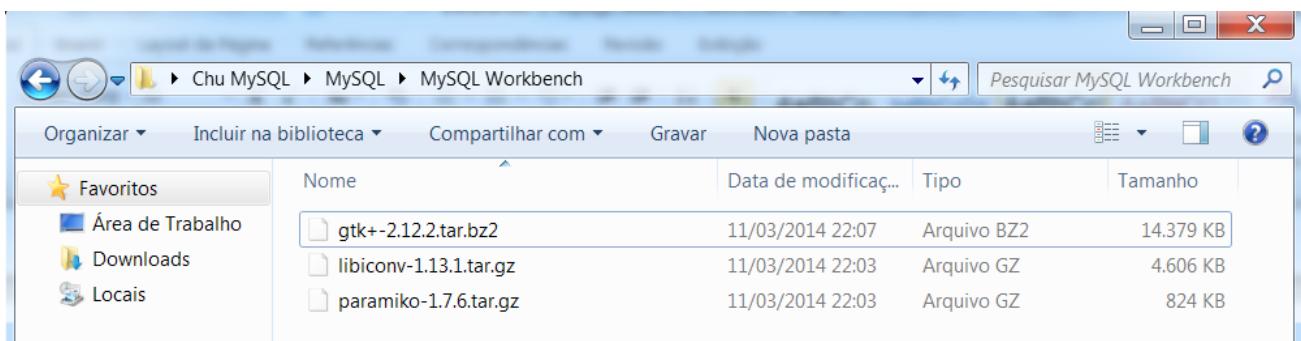


Figura 2.5b – Etapas da instalação do MySQL WorkBench.

Podem ser necessários alguns arquivos adicionais instalados em seu computador.

Os arquivos abaixo podem ser requeridos:



Se for o caso, você pode obtê-los diretamente no site de instalação do MySQL Workbench: <<http://dev.mysql.com/doc/workbench/en/wb-starting.html>>. Acesso em: 14 maio 2014.

Aula 3 – A linguagem SQL

SQL – Linguagem para banco de dados relacional

Um banco de dados relacional tem uma linguagem padrão para tratamento dos dados chamada SQL (*Structured Query Language*) ou linguagem estruturada de consulta. A recuperação e manipulação dos dados armazenados no banco de dados é o principal objetivo da linguagem SQL.

Esta linguagem surgiu nos anos 1970, inspirada em um artigo de Edgard F. Codd e foi desenvolvida nos laboratórios da IBM, direcionada ao tratamento relacional.

(IBM, 2003).

Surgiram outras linguagens com objetivos similares, mas a SQL tornou-se a mais utilizada, graças à sua facilidade de uso. Surgiram também algumas variações da linguagem SQL, gerando a necessidade de padronização.

Em 1986, o ANSI — *American National Standard Institute* — tornou-a padrão para aplicações de banco de dados. Em 1987, a ISO — *International Organization for Standardization* — ratificou a padronização. O SQL padrão ANSI foi revisado em algumas oportunidades, como em 1992, 1999 e 2003. (ORACLE, 2003)

Mesmo com a padronização, vários fabricantes lançaram versões SQL de seus SGBDs. Quase todas elas contêm o conjunto de comandos SQL ANSI, além de comandos proprietários.

Os comandos da linguagem SQL são divididos em dois grupos principais:

DDL – Data Definition Language	Grupo de comandos utilizados para a criação e manutenção de tabelas e outros objetos no banco de dados.
DML – Data Manipulation Language	Grupo de comandos utilizados para manipular dados: incluir, alterar, excluir e consultar dados.

Entre os vários SGBDs que adotam a SQL como linguagem, há um grande conjunto de comandos comuns, principalmente no que diz respeito aos comandos DML (manipulação de dados). Geralmente, as

principais variações entre os SGBDs estão nos comandos de administração do banco de dados, que cuidam mais das configurações do banco de dados e de seu ambiente.

Após aprender a trabalhar com um SGBD que utiliza SQL, torna-se bem fácil migrar para outro justamente pela similaridade e padronização de grande parte da SQL.

Principais comandos SQL

As principais operações efetuadas em banco de dados são relativas à manipulação de dados. São elas:

- inclusão de dados no banco de dados (INSERT);
- alteração de dados já existentes no banco de dados (UPDATE);
- exclusão de dados já existentes no banco de dados (DELETE);
- pesquisa e recuperação de dados já existentes no banco de dados (SELECT).

No decorrer deste curso, caro aluno, veremos esses comandos em detalhes.

Exercícios do Módulo 1

1) Qual a melhor definição para um banco de dados?

- a. É uma coleção de dados armazenada em um único arquivo físico.
- b. É um software com inteligência para manipular dados.
- c. É um conjunto de dados, armazenados em uma estrutura inteligente composta de um software especializado em manipulação e gerenciamento de grandes volumes de dados.
- d. É um conjunto de arquivos e tabelas eletrônicas, com chaves-primárias e relacionamentos 1 x N, onde podemos armazenar todos os tipos de informação.
- e. É um conjunto de tabelas, relacionamentos e chaves-primárias.

2) Antes da tecnologia de banco e dados, o armazenamento de dados era feito através de “sistema de arquivos”, ou seja, através de arquivos isolados, sem gerenciamento centralizado e todo o trabalho de consistência e integridade desses arquivos cabia à aplicação que os acessava. Comparando “Sistema de Arquivos” e “Sistema Gerenciador de Banco de Dados”, qual afirmação é verdadeira?

- a. No “Sistema de Arquivos” os dados são íntegros, já que os dados são armazenados em um único local bem especificado.
- b. No “Sistema de Arquivos” ocorre uma redução da duplicação de dados, pois como os dados estão armazenados em um único local, existem menos chances para os dados terem múltiplas cópias.
- c. O “Sistema Gerenciador de Banco de Dados” possui uma dependência com o programa de aplicação bem maior que a dependência que um “Sistema de Arquivos” possui, já que o SGBD facilita toda a interface do “Sistema de Arquivos” com o programa.

- d.** "Sistema de Arquivos" grava seus dados em disco, segundo estruturas de dados próprias. Para acessar estes dados é necessário conhecer essas estruturas, portanto se vários programas compartilham os mesmos dados, então todos devem conhecer e manipular as mesmas estruturas.
- e.** No "Sistema de Arquivos" cada programa vê apenas os dados que o interessam e não precisa ser modificado se a estrutura de dados for alterada.

3) Sobre SGBD, podemos afirmar:

- a.** É uma camada inteligente de dados que gerencia o software de manipulação de dados.
- b.** É composto, basicamente, por duas camadas distintas: uma que gerencia os dados já armazenados no BD e outra que gerencia os programas que manipulam esses dados.
- c.** É um software que tem por especialidade a recuperação de dados, seu armazenamento através de sistemas aplicativos, passando por uma camada inteligente.
- d.** É composto por dados e software. Os dados estão armazenados em arquivos físicos, dependentes dos sistemas aplicativos, enquanto o software tem portabilidade assegurada pela tecnologia plena de banco de dados.
- e.** É um software especializado exclusivamente no tratamento de dados.

4) Classifique as afirmações abaixo em (V)erdadeira ou (F)alsa:

- () Quando um banco de dados se relaciona com outro banco de dados nós podemos chamá-lo de banco de dados relacional.
- () Uma das vantagens de se usar banco de dados é que se pode evitar mais facilmente redundâncias.
- () O uso de banco de dados simplifica o trabalho dos sistemas aplicativos.
- () Usar banco de dados é importante, mas diminui a segurança dos dados.
- () Uma grande vantagem do banco de dados é a facilidade de se compartilhar dados entre aplicações.

5) Em banco de dados relacionais, as informações são acomodadas em _____ de acordo com seu assunto. A estrutura de uma tabela é formada pelas _____. Cada ocorrência de uma tabela representa um elemento; dentro da tabela, no modelo relacional, cada elemento é uma _____.

6) Em um banco de dados relacional, as informações ficam organizadas em tabelas que se relacionam. Para esta forma de organizar/armazenar dados, avalie as afirmações abaixo em (V)erdadeiras ou (F)alsas:

- () escreve-se menos, sem a necessidade de se repetir informações, reduzindo a redundância.
- () centralizando a informação em uma tabela e referenciando-se pelo código em outras, garantindo que a informação estará igual em todas as ocorrências o que poderia não acontecer se fosse necessário digitá-la várias vezes (erros de digitação).
- () segregando-se as informações em mais tabelas, aumentamos consideravelmente o espaço necessário para se armazenar a mesma informação.

7) Relacione as colunas:

- (1) SQL () Utilizada para a criação e manutenção de tabelas e outros objetos no banco de dados.
(2) DDL () Utilizada para manipular dados: incluir, alterar, excluir e consultar dados.
(3) DML () Linguagem estruturada de consulta. Tem um padrão internacional de comandos.
(4) MySQL () É um dos SGBDs mais modernos e mais utilizado atualmente no mundo.

8) Relacione os principais comandos de Banco de Dados com sua função:

- (1) INSERT () Consulta dados já armazenados em tabelas. É o comando mais utilizado.
(2) UPDATE () Exclui dados já armazenados em uma tabela do banco de dados.
(3) DELETE () Inclui dados em uma tabela do banco de dados.
(4) SELECT () Altera dados já armazenados em uma tabela do banco de dados

Módulo 2 – Modelagem de dados

Caro(a) aluno(a), neste módulo discutiremos como projetar um banco de dados em sua estrutura de tabelas. É uma etapa substancialmente importante, pois as tabelas e demais objetos serão criados na base de dados a partir de um modelo que criamos. Serão apresentados os principais conceitos de modelagem de dados relacional, normalização de dados e criação de diagramas para modelar os dados. Nesta etapa, podemos utilizar a ferramenta MySQL Workbench para nos apoiarmos na construção desses modelos.

Aula 4 – Modelos de dados e elementos do modelo relacional

Modelos de dados

Os modelos de dados são representações que criamos das tabelas que figurarão no banco de dados e de como estas se relacionam. É um instrumento poderoso para documentar nosso banco de dados e para garantir que a estrutura está sendo criada da forma correta, sem vícios.

Fazendo uma analogia, antes de se construir uma casa é necessário que um engenheiro elabore sua planta, a fim de obter uma visão geral e detalhada e garantir a consistência da construção, evitando itens não previstos que podem acarretar retrabalho (a reconstrução de uma parte da casa, por exemplo).

Da mesma forma, antes de criarmos as estruturas de dados dentro do SGBD, devemos desenhar sua estrutura, avaliando de forma global que dados são necessários e como devem estar organizados.

As vantagens de se elaborar um modelo de dados são:

- Estruturas de dados mal organizadas podem gerar trabalho adicional quando da criação dos programas que acessam esses dados, ao passo que um banco de dados bem projetado, traz facilidades ao desenvolvedor que cria programas acessando esta base de dados.
- Estruturas de dados mal projetadas podem, inclusive, impactar na performance do sistema, pois podem exigir que a aplicação faça esforços maiores que o necessário para recuperar ou atualizar dados.
- Em desenvolvimento de software, sempre gastamos mais tempo alterando sistemas já criados do que criando sistemas novos. Isto significa que muito provavelmente faremos alterações na base de dados já criada anteriormente. Uma base de dados bem projetada facilita a manutenção futura, poupando tempo de manutenção e preservando a integridade do projeto.

Um modelo de dados bem elaborado:

- poupa tempo de desenvolvimento (ou evita desperdício com retrabalho);
- permite uma programação mais limpa para acesso aos dados;
- facilita manutenção futura; e
- documenta as estruturas de dados do sistema (parte da especificação técnica do sistema).



Elementos de um modelo relacional

Vimos que o modelo relacional é baseado em tabelas que, por sua vez, são compostas por linhas e colunas. Um modelo relacional é composto pelos seguintes elementos:

- tabelas (ou entidades ou relações);
- relacionamentos;
- atributos.

Veja o exemplo de relacionamento entre tabela de dados na figura 4.2a.

Tabela de Vendas					Tabela Vendedor	
NumVenda	DataVenda	Cliente	Valor	CodVendedor	CodVendedor	NomeVendedor
150	15/01/2013	Pedro	1.500,00	V3	V1	Obama
151	25/01/2013	Tiago	1.950,00	V2	V2	Putin
152	04/02/2013	João	975,00	V6	V3	Angela
153	14/02/2013	Maria	1.267,50	V1	V4	Fernando
154	24/02/2013	Pedro	633,75	V5	V5	Faustão
155	06/03/2013	Carlos	823,88	V4	V6	Jô Soares
156	16/03/2013	José	411,94	V1		
157	26/03/2013	João	535,52	V6		

Figura 4.2a – Exemplo de tabelas com atributos (colunas) e relacionamentos.

O exemplo da figura 4.2a apresenta duas **tabelas**, uma para o assunto “Vendas” e outra para o assunto “Vendedor”.

Em tabelas relacionais, os dados estão distribuídos em linhas e colunas. A primeira tabela do exemplo registra as vendas, com o número de identificação da venda, a data, o cliente, o valor da venda e o código de identificação do vendedor. Os dados do vendedor estão na segunda tabela.

Note que cada linha da tabela vendas possui o registro de uma venda e cada linha da tabela vendedor registra um vendedor. Assim, cada linha de uma tabela registra os dados a respeito de um **elemento** desta tabela.

Note também que cada coluna registra um tipo de informação a respeito de cada elemento da tabela. As colunas são os atributos que temos a respeito dos elementos da tabela. Nas tabelas da figura 4.2a, podemos exemplificar com atributos o que é mostrado na figura 4.2b.

Tabela de Vendas					Tabela Vendedor	
NumVenda	DataVenda	Cliente	Valor	CodVendedor	CodVendedor	NomeVendedor

Figura 4.2b – Atributos de Tabelas.

O conjunto de colunas (**atributos**) de uma tabela compõe a **estrutura da tabela**.

Observe também que a tabela vendas não contém o nome do vendedor, apenas seu código. O nome do vendedor está em outra tabela e conseguimos localizá-lo a partir do código do vendedor, que é comum às duas tabelas. Isto caracteriza um **relacionamento** entre tabelas, como exemplificado na figura 4.2c.



Chaves primária e estrangeira

Para relacionar duas tabelas, precisamos ter colunas em ambas as tabelas que sejam correspondentes e equivalentes. Essas colunas são consideradas **chaves**, e podem ser de dois tipos: **primária** e **estrangeira**.

Chave primária (PK)

Em uma tabela relacional, precisamos eleger uma informação que identifique o elemento da tabela de uma forma única. Por exemplo, na tabela vendas, a informação que temos certeza que não se repete é a coluna NumVenda (número da venda). Assim, elegemos esta coluna como sendo a **chave primária** da tabela vendas.

Chamamos a chave primária de **PK** (*Primary Key*). Usaremos esta nomenclatura em nossos exemplos.

Características de uma chave primária (PK):

- é a informação que identifica, de forma única, uma linha da tabela;
- pode ser composta por um ou mais campos da tabela;
- uma chave primária não pode ter valores repetidos (ou seja, não se podem ter 2 linhas da tabela com valores iguais na chave primária).



Exemplos de chaves primárias:

Tipo	Coluna	Descrição
Simples	CPF	identifica um contribuinte de forma única do cadastro da Receita Federal.
Composta	Núm.Agência + Núm. Conta Corrente	identifica uma conta dentro de um banco de forma única.

No caso da conta corrente, são necessárias duas informações para identificar uma conta. Além do número da conta é necessário o número da agência, pois no cadastro de um banco, uma conta de número 1500, por exemplo, pode existir mais de uma vez em agências diferentes.

Desta forma, as duas informações, em conjunto, são necessárias para identificação única da conta, formando o que chamados de **chave primária composta**.

No exemplo de tabelas apresentado na figura 4.3, as colunas NumVenda e CodVendedor são as PKs eleitas para identificar de forma única suas respectivas tabelas.

Tabela de Vendas					Tabela Vendedor	
NumVenda	DataVenda	Cliente	Valor	CodVendedor	CodVendedor	NomeVendedor
150	15/01/2013	Pedro	1.500,00	V3	V1	Obama
151	25/01/2013	Tiago	1.950,00	V2	V2	Putin
152	04/02/2013	João	975,00	V6	V3	Angela
153	14/02/2013	Maria	1.267,50	V1	V4	Fernando
154	24/02/2013	Pedro	633,75	V5	V5	Faustão
155	06/03/2013	Carlos	823,88	V4	V6	Jô Soares
156	16/03/2013	José	411,94	V1		
157	26/03/2013	João	535,52	V6		

Figura 4.3 – Exemplos de PKs e FKs.

Chave estrangeira (FK)

Na figura 4.3, observe que a coluna CodVendedor liga as duas tabelas. Dissemos que para relacionar duas tabelas é necessário que tenhamos coluna(s) correspondente(s) entre as duas tabelas. Essas colunas correspondentes precisam ter as mesmas características: tamanho, tipo de dados (numérico, string, etc.).

Quando relacionamos duas tabelas, tomamos as características de determinada coluna de uma tabela e criamos uma coluna em outra tabela com as mesmas características. Nesta relação, sempre teremos os dois tipos de chave: chave primária e chave estrangeira.

Uma tabela terá a chave primária (PK que a identifica de forma única) e a outra, que recebe as características da coluna, terá a **chave estrangeira**, pois recebe uma coluna cujas características vieram de outra tabela.

Em nosso exemplo da figura 4.3, a coluna original está na tabela Vendedor e esta coluna é “exportada” para a tabela vendas. Essa coluna que “veio” de outra tabela chamamos de chave estrangeira. Chamamos a chave estrangeira de **FK** (*Foreign Key*). Usaremos esta nomenclatura algumas vezes em nossos exemplos.

Características de uma chave estrangeira (FK):

- Coluna(s) de tabela que referencia (m) a chave primária de outra tabela;
- A chave estrangeira liga duas tabelas verificando a integridade de dados entre elas;
- Pode ou não ser parte da chave primária da tabela onde está;
- Pode ter o valor repetido na tabela onde é chave estrangeira se não for coluna(s) exclusiva(s) na PK da tabela.



Aula 5 – Organização dos dados em tabelas

Estruturando dados em tabelas

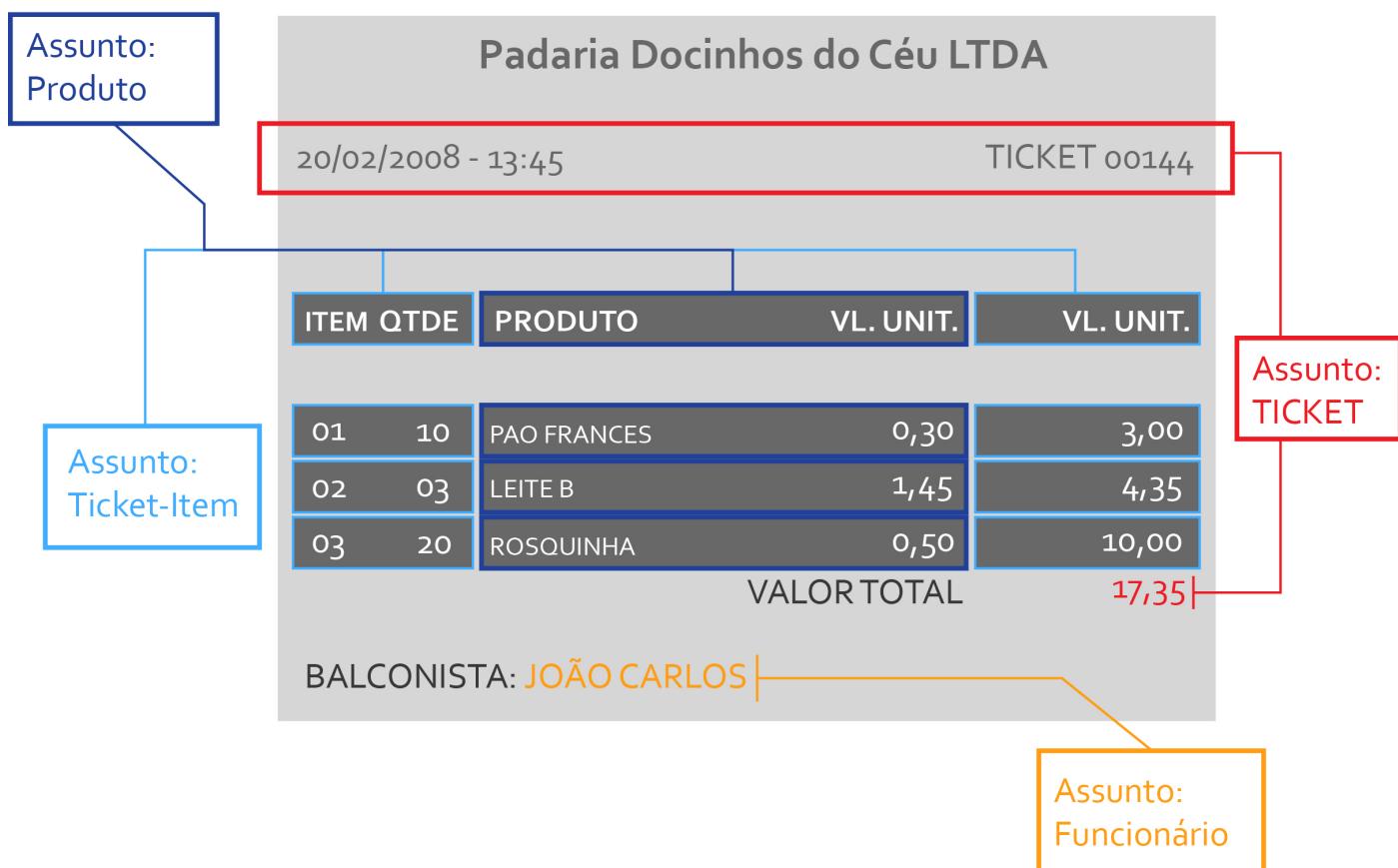


Para que tenhamos armazenamento e recuperação de dados eficientes, é necessário que o banco de dados seja bem projetado, como discutimos anteriormente. A estruturação dos dados se dá segregando as informações em tabelas. O critério para esta segregação é separar os assuntos em tabelas distintas.

Vamos discutir este assunto com um exemplo. Acompanhe o quadro a seguir que toma como exemplo um ticket de caixa gerado em uma padaria. Os dados que compõem o ticket podem ser organizados em várias tabelas.

Exemplo de modelagem: **Ticket de Padaria**

Segregação por Assunto



4 assuntos identificados: TICKET / ITEM de TICKET / PRODUTO / FUNCIONÁRIO

Após identificar os assuntos/tabelas, separamos as informações pelas tabelas, considerando cada tipo de informação de uma coluna, de uma tabela.

Pode ser necessário incluir colunas artificialmente, principalmente nos casos onde não exista uma coluna que possa ser eleita como **PK** (identificar a tabela de forma única).

Veja no quadro a seguir um exemplo de como podemos distribuir as informações em tabelas.

1. Distribuição dos dados em tabelas.
2. Criação de Chaves Primárias.
3. Tabelas não têm boa chave primária, exigem a criação de colunas artificiais.
4. Ao relacionar as tabelas, criamos as Chaves Estrangeiras.
5. Criação de Chave Composta para tabela de Item.

TICKET	ITEM_TICKET	PRODUTO	FUNCIONARIO
NumTicket	NumTicket	CodProduto	CodFunc
DataTicket	NumItem	NomeProduto	NomeFunc
ValorTotal	Qtde	ValorUnitario	
CodFunc	ValorItem		
	CodProduto		

Diagrama entidade relacionamento (DER)

O diagrama entidade relacionamento (DER), é uma representação abstrata das tabelas e relacionamentos de um banco de dados. Os elementos do DER são os mesmos elementos do relacional: entidades (tabelas), relacionamentos e atributos.

DER é uma representação gráfica e seus elementos são representados por símbolos. Na nossa disciplina adotaremos os símbolos mostrados na figura 5.2.

- ENTIDADES (tabelas)

Nome Entidade

- RELACIONAMENTOS



- ATRIBUTOS

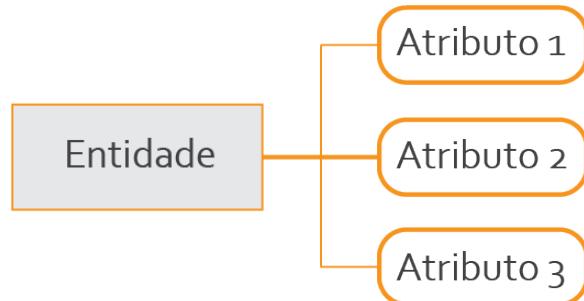


Figura 5.2 – Elementos de um DER.

Cardinalidade

A cardinalidade é um conceito que indica quantas vezes um elemento singular de uma tabela pode se relacionar com um ou mais elementos de outra tabela.

A cardinalidade é baseada na teoria de conjuntos e nas funções que ligam elementos de um a outro conjunto. Vamos esquematizar um exemplo de conjuntos para explicar melhor, como apresentado na figura 5.3.

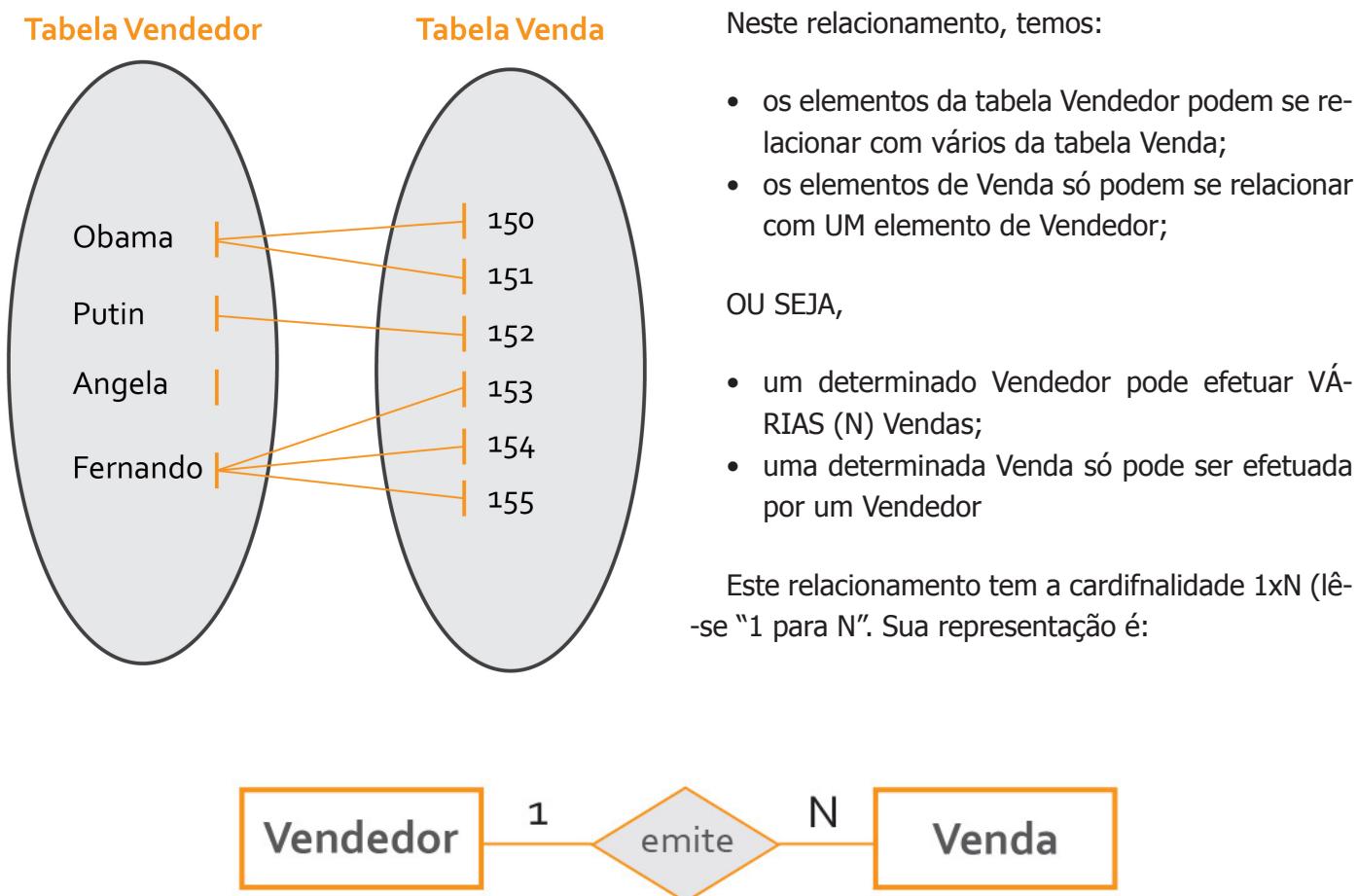


Figura 5.3 – Exemplo de determinação da cardinalidade.

Regra para determinar a cardinalidade

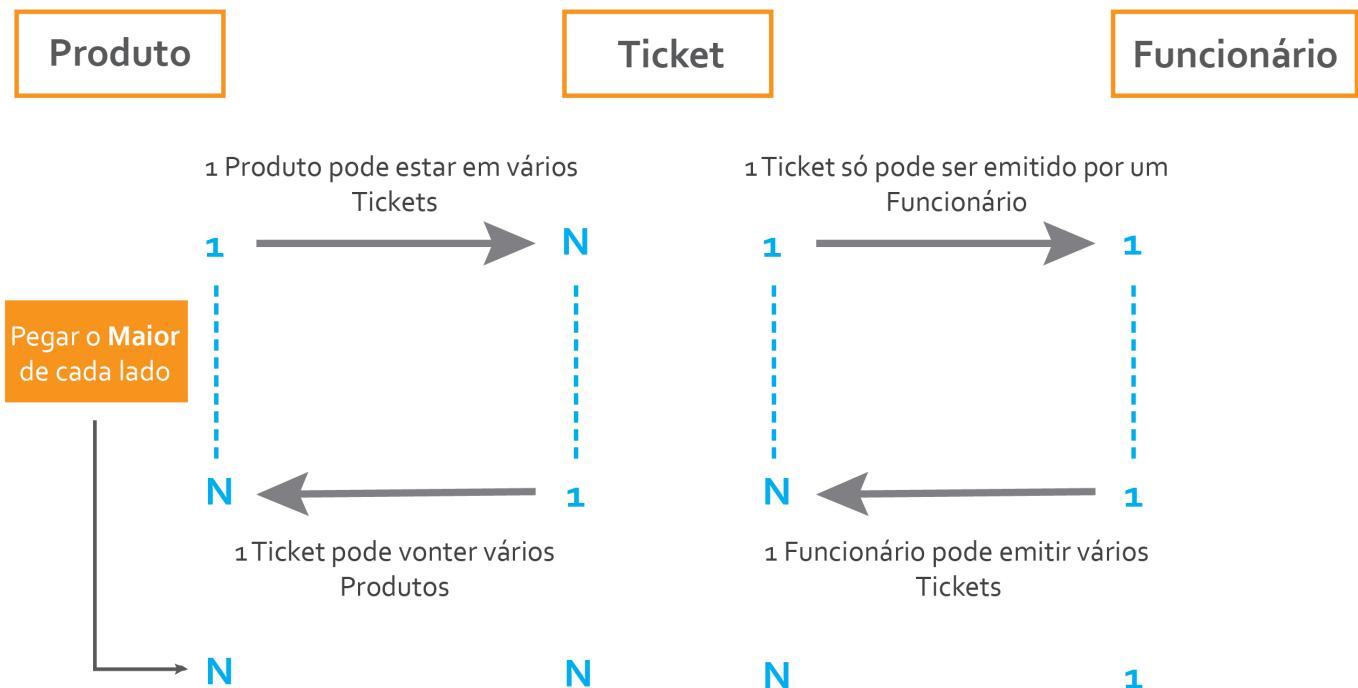
Para determinar a cardinalidade de um relacionamento, faça-se a pergunta: quantas vezes um elemento (uma linha) de uma tabela poderia se relacionar com linhas de outra tabela? Refaça a pergunta no sentido inverso: quantas vezes um elemento da segunda tabela poderia se relacionar com elementos da primeira tabela?

A Cardinalidade é determinada lendo-se os relacionamentos em ambos os sentidos. Torna-se a MAIOR cardinalidade de cada lado, como a cardinalidade final da relação. Veja o exemplo:

- 1) Considere o Diagrama abaixo, com 3 tabelas e 2 relacionamentos entre elas. Vamos determinar sua cardinalidade:



2) Avaliação da Cardinalidade (passo a passo):



Exemplo de um DER

Considerando o exemplo da animação 4, podemos esquematizar em um DER as tabelas, os relacionamentos entre elas, as cardinalidades e os atributos das tabelas, com suas PKs e FKs (conforme mostra a figura 5.4).

Veja a animação a seguir para entender a montagem do DER para as tabelas e atributos identificados na animação anterior.

1. Tabelas mapeadas (assuntos mapeados);
2. Relacionamentos mapeados entre as tabelas;
3. Cardinalidades entre as Tabelas;
4. Atributos (colunas) das tabelas;
5. Destacando FKs

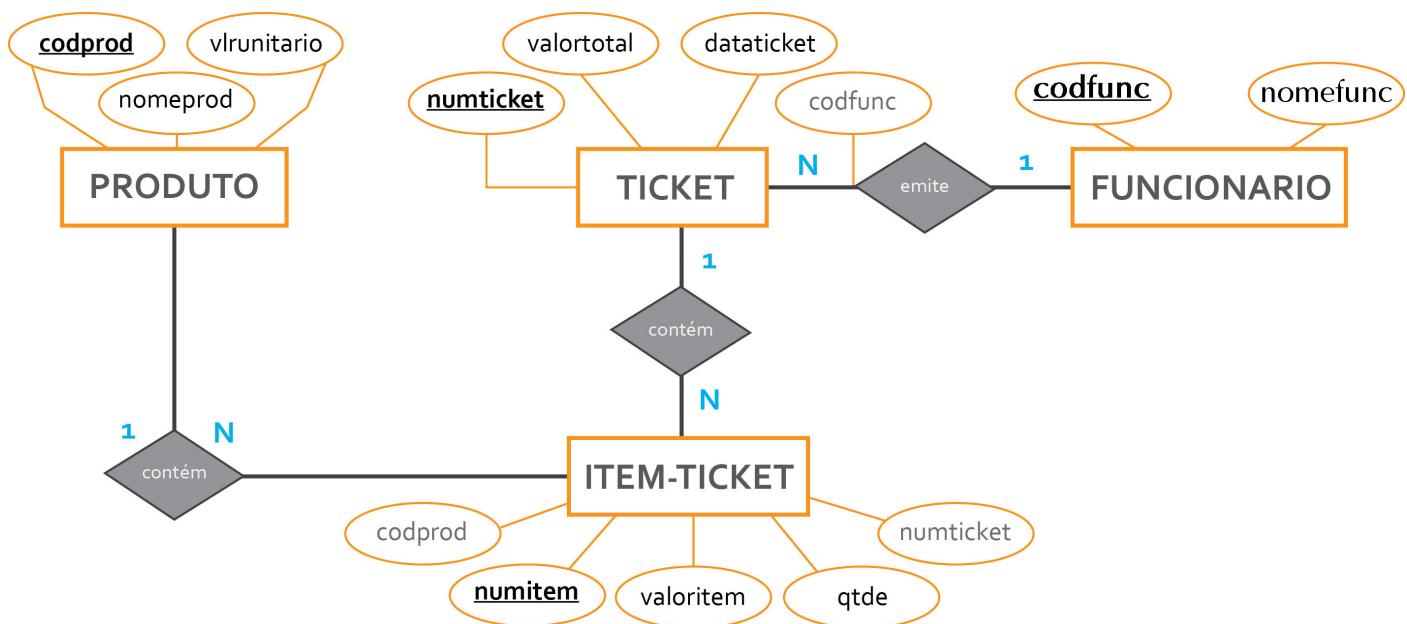


Figura 5.4 – DER de exemplo de Ticket.

Observe na figura 5.4 que, para cada tabela, existe(m) coluna(s) sublinhadas. Estas colunas são as PKs dessas tabelas e assim são indicadas em um DER.

Da mesma forma, as FKs são destacadas com uma linha acima de seu nome. São as FKs que assim são representadas neste diagrama.

Mapeamento

O DER é muito valioso para visualizarmos tabelas e seus relacionamentos. Mas não é o diagrama mais adequado quando queremos verificar as colunas das tabelas e as PKs e FKs. Uma forma simplificada de demonstrar as tabelas e suas colunas é o mapeamento.

Com ele indicamos simplesmente o nome da tabela e, entre parênteses, as colunas desta tabela. Veja como ficaria o mapeamento das tabelas e atributos de nosso exemplo:

Ticket (NumTicket, DataTicket, ValorTotal, CodFunc)
Item_Ticket (NumTicket, NumItem, Qtde, ValorItem, CodProduto)
Produto (CodProduto, NomeProduto, ValorUnitario)
Funcionario (CodFunc, NomeFunc)

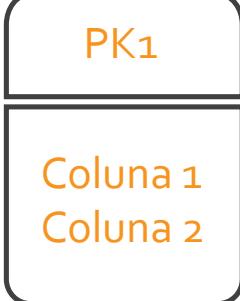
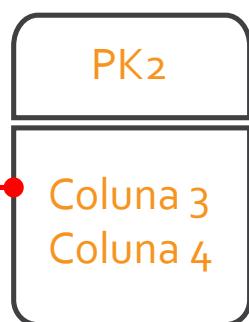
Modelo de dados

Além do Mapeamento, que é uma representação bastante simples, temos o modelo de dados que apresenta com detalhes tabelas, atributos, relacionamentos, PKs e FKs.

No modelo de dados, representamos as tabelas como retângulos, com o nome acima e, dentro do retângulo, as colunas da tabela. A PK fica separada das demais colunas da tabela por uma linha. Veja figura 5.6a.



Figura 5.6a – Símbolo de tabelas.

Tabela 1**Tabela 2**

Os relacionamentos são apresentados por linhas que ligam duas tabelas. A cardinalidade **1 x N** é representada, do lado do N, por um tripé (como um pé de galinha) ou por um pequeno círculo preenchido.

Veja as duas notações na figura 5.6b. São equivalentes e podem ser utilizadas indistintamente. Ambas representam um relacionamento entre as tabelas 1 e 2, com cardinalidade 1 x N. A TABELA1 é o lado **1** e a TABELA2 é o lado do **N**.

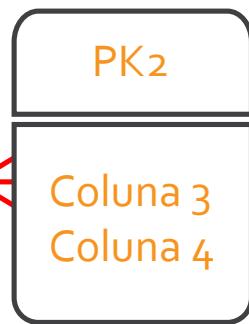
Tabela 1**Tabela 2**

Figura 5.6b – Símbolos de relacionamentos.

Voltando ao nosso exemplo, a figura 5.6c mostra como ficaria o DER da figura 5.4 convertido em modelo de dados:

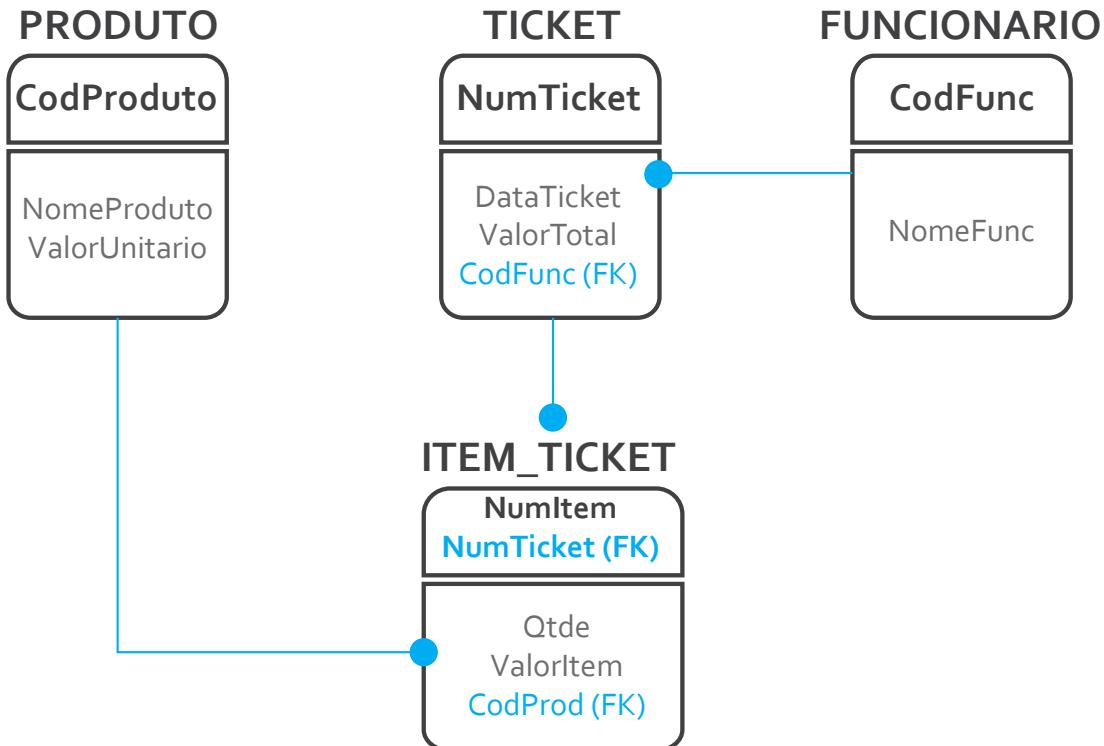


Figura 5.6c – Modelo de dados.

Para este tipo de diagrama, podemos utilizar ferramentas de modelagem como o MySQL WorkBench. O modelo de dados, representado nesta ferramenta ficaria como mostrado na figura 5.6d.

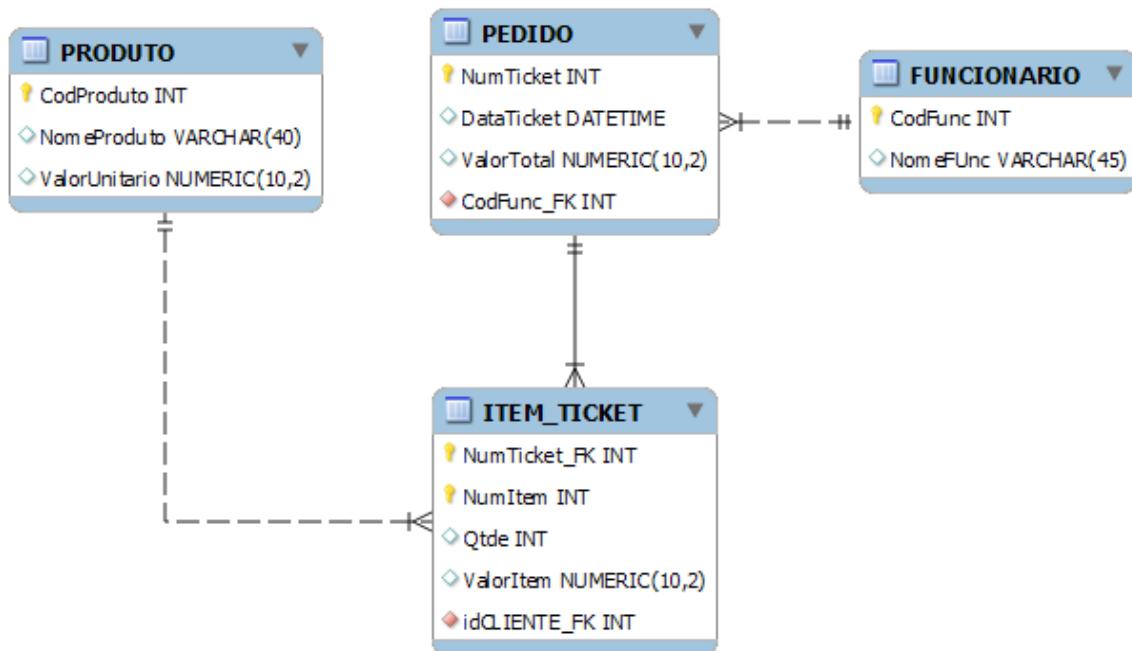


Figura 5.6d – Modelo de Dados feito com MySQL WorkBench.

Note que a apresentação é ligeiramente diferente do que fizemos em 5.6c. O MySQL WorkBench (chamaremos de MS-WB) pode ser configurado para apresentar o mesmo modelo com diferentes metodologias de modelagem (a apresentação varia um pouco de uma para a outra, mas o sentido é sempre o mesmo).

Na representação da figura 5.6d, veja que as PKs não estão separadas das demais colunas por um traço que divide a tabela. Neste caso, há um ícone de uma chave amarela ao lado esquerdo da(s) coluna(s) que é (são) PK.

Para as colunas normais da tabela, o ícone à esquerda é um losango branco. E para as FKs o ícone é um losango vermelho. Nesta versão, opcionalmente, configuramos o MS-WB para acrescentar o sufixo “_FK” ao final de cada nome de Foreign Key.

Tipo de dado

Para criarmos as tabelas no banco de dados é necessário que qualifiquemos as colunas que vamos criar. A primeira qualificação que precisamos indicar é o tipo de dado e, se for o caso, o tamanho em bytes que deve ser reservado para aquela coluna.

Ao criarmos uma coluna, devemos indicar que tipo de informação será ali gravada, se a informação será numérica (neste caso se é um valor inteiro ou se pode ser decimais), se é um *string* (neste caso é necessário indicar o tamanho em bytes), se é um campo de armazenamento de data (este é um tipo específico para armazenamento de conteúdo de data e hora; este tipo permite, inclusive, que façamos contas entre datas para calcular, por exemplo, quantos dias há entre duas datas).

Note que na figura 5.6d, após cada nome de coluna, há uma informação indicando o tipo de dado que deve ser utilizado para cada coluna. Na tabela produto, por exemplo, suas três colunas têm tipos de dados diferentes, veja:

CodProduto	É um número inteiro, sem decimais.
NomeProduto	É um valor <i>string</i> . Neste caso, com 45 posições, no máximo. Utilizamos o tipo VARCHAR, ao invés de CHAR, pois se usamos CHAR, ocuparemos exatamente 45 posições. Usando o VARCHAR, ele permite utilizar no máximo 45 posições, mas se forem utilizadas apenas 25, por exemplo, o armazenamento não consumirá as 45 posições, poupando espaço de armazenamento.
Valor unitário	É um campo numérico, mas desta vez é necessário que se permitam valores decimais (neste caso, centavos de um valor monetário). Indica-se o tamanho total da coluna e, separado por vírgula, quantas destas posições são dedicadas a casas decimais.

Normalização de dados

A normalização é um conjunto de regras formais para garantir que um conjunto de tabelas está bem projetado, eliminando redundâncias e facilitando o trabalho do programador.

O processo de normalização ocorre em etapas sucessivas. Cada etapa corresponde a uma determinada forma normal, que apresenta um progressivo refinamento na estrutura das tabelas. Uma tabela precisa atender às 3 etapas de normalização para que possa ser considerada como normalizada. Essas etapas são nomeadas como:

1FN	Primeira Forma Normal
2FN	Segunda Forma Normal
3FN	Terceira Forma Normal

A seguir, abordaremos cada uma delas e as regras para que a tabela atenda a essas formas normais e à formalização.

1FN – Primeira forma normal

Regra para estar na 1FN:

Para uma tabela estar na 1FN a tabela deve ter apenas atributos atômicos, ou seja, a tabela não deve conter dados repetidos em sua estrutura.

Para uma tabela estar na 1FN a tabela deve ter apenas atributos atômicos, ou seja, a tabela não deve conter dados repetidos em sua estrutura.

Caso haja dados repetidos em sua estrutura, dizemos que há a coluna é “multivvalorada”. Neste caso, devemos segregar em uma nova tabela esta coluna.

A tabela de livros a seguir é um exemplo de tabela com atributos multivalorados.

Tabela “Livros”

IdLivro	Título	Assunto	Autor1	Autor2	Autor3
21237	Os Sertões	Ficção	E. Cunha		
33455	Eletricidade Básica	Física	A. Silva	B. Santos	
12312	Atlas do Brasil	Geografia	IBGE		

Atributos Multivalorados

Veja que a tabela **livros** contém 3 colunas para o mesmo tipo de informação: autor. Isto é de pouca valia, pois havendo menos de 3 autores, colunas ficarão vazias, sendo desperdiçadas. Ao mesmo tempo, caso haja um livro com mais de 3 autores, esta estrutura não permite seu armazenamento.

A solução para isto é segregar as colunas de autor (coluna multivalorada) em uma tabela à parte que seja ligada à tabela principal. Neste exemplo, a tabela de “Autorias”, veja:

Tabela “Autorias”

IdLivro	Título	Assunto
21237	Os Sertões	Ficção
33455	Eletricidade Básica	Física
12312	Atlas do Brasil	Geografia

Tabela “Autorias”

IdLivro	Autor1
21237	E. Cunha
33455	A. Silva
33455	B. Santos
12312	IBGE

Nesta nova disposição, criamos uma tabela nova: autorias, que abrigará os nomes dos autores. Essa nova tabela está ligada à primeira tabela. Veja que agora um livro pode ter quantos autores forem que a estrutura permite seu armazenamento.

Agora podemos dizer: **Estrutura Normalizada para 1FN**, pois agora todos os atributos das tabelas são atributos atômicos.

2FN – Segunda forma normal

Regra para 2FN:

Para estar na 2FN a tabela deve:

- estar na 1FN;
- não possuir dependência parcial da PK, ou seja:
- atributos não-chave não podem se referir a apenas uma parte da chave primária (o que caracteriza dependência funcional parcial);
- um valor que toda vez que apareça tenha outro valor associado sempre de valor igual.

Vamos a um exemplo:

a) Estrutura Original

PK					
NumeroVenda	CodProd	NomeProd	VirUnit	Qtde	VlrTot
15	50	DVD	3.20	20	64.00
22	88	Cartucho	76.00	2	152.00
25	43	Papel A4	12.50	10	125.00
30	50	DVD	3.20	10	32,00
38	50	DVD	3.20	15	48.00
40	88	Cartucho	76.00	1	76.00
42	21	Marca Texto	2.80	8	22.40

b) Há colunas que dependem de PARTE da Chave

NumeroVenda	CodProd	NomeProd	VirUnit	Qtde	VlrTot
15	50	DVD	3.20	20	64.00
22	88	Cartucho	76.00	2	152.00
25	43	Papel A4	12.50	10	125.00
30	50	DVD	3.20	10	32,00
38	50	DVD	3.20	15	48.00
40	88	Cartucho	76.00	1	76.00
42	21	Marca Texto	2.80	8	22.40

Dependência de parte da chave

Dependência de parte da chave

Esta tabela atende a 1FN (não contém colunas multivaloradas) mas as colunas NomeProd e VlrUnit dependem da coluna CodProd (alteram seu valor dependendo desta). Ou seja, a tabela não atende às regras da 2FN.

Para que atenda, será necessário separar em outra tabela os dados dependentes de parte da chave.

c) Estrutura Normalizada para (2FN)

Venda			
NumeroVenda	CodProd	Qtde	VlrTot
15	50	20	64.00
22	88	2	152.00
25	43	10	125.00
30	50	10	32,00
38	50	15	48.00
40	88	1	76.00
42	21	8	22.40

Produto

CodProd	NomeProd	VlrTot
21	Marca Texto	2,80
43	Papel A4	12,50
50	DVD	3,20
88	Cartucho	76,00

Nesta nova apresentação, não há dados que dependem parcialmente da chave primária (PK) da tabela Venda, nem da tabela Produto. Agora podemos dizer:

Estrutura Normalizada para 2FN

3FN – Terceira forma normal

Regra 3FN:

Para estar na 3FN a tabela deve:

- estar na 2FN;
- seus atributos não devem depender de atributo(s) que não seja(m) PK (dependência transitiva em relação a outra coluna não-PK), ou seja:
- atributos não-chave não podem se referir a outros atributos não-chave ou só parte da chave.

Vamos um exemplo:

a) Estrutura Original

PK	RegFunc	NomeFunc	Endereco	CodCargo	NomeCargo	Salario
	15	Helio	Av. Paulista	18	Analista	1200,00
	22	Frank	Rua Mauro	23	Suporte	750,00
	25	Flavio	Al. Aicás	38	Comercial	2000,00
	30	Carmela	Av. Pavão	18	Analista	1200,00
	38	Mauro	Rua Cravo	15	Programador	980,00
	40	Vanuza	Rua Inglaterra	23	Suporte	750,00
	42	Mituo	Rua Suécia	23	Suporte	750,00

b) Colunas que dependem de outra coluna que NÃO é PK



PK	RegFunc	NomeFunc	Endereco	CodCargo	NomeCargo	Salario
	15	Helio	Av. Paulista	18	Analista	1200,00
	22	Frank	Rua Mauro	23	Suporte	750,00
	25	Flavio	Al. Aicás	38	Comercial	2000,00
	30	Carmela	Av. Pavão	18	Analista	1200,00
	38	Mauro	Rua Cravo	15	Programador	980,00
	40	Vanuza	Rua Inglaterra	23	Suporte	750,00
	42	Mituo	Rua Suécia	23	Suporte	750,00

dependência em relação a outro atributo não chave

c) Estrutura Normalizada (3FN)

FUNCIONARIO

RegFunc	NomeFunc	Endereco	CodCargo
15	Helio	Av. Paulista	18
22	Frank	Rua Mauro	23
25	Flavio	Al. Aicás	38
30	Carmela	Av. Pavão	18
38	Mauro	Rua Cravo	15
40	Vanuza	Rua Inglaterra	23
42	Mituo	Rua Suécia	23

CARGO

CodCargo	NomeCargo	Salario
15	Programador	980,00
18	Analista	1200,00
23	Suporte	750,00
38	Comercial	2000,00

Com essa estrutura, as colunas que não são PK dependem exclusivamente da PK.

Agora podemos dizer: **Estrutura Normalizada para 3FN**

Se uma tabela está na 3FN atende 1FN, 2FN e 3FN.
Uma tabela que está na 3FN está **normalizada!!**
Elá está adequada para ser criada no Banco de Dados!



Regras relativas as PKs

As chaves primárias identificam de forma única uma tabela. Uma vez criada uma PK em uma tabela, o banco de dados não permitirá conteúdo duplicado na(s) coluna(s) que for(em) PK.

Produto



Exemplo: Considere que em uma tabela produto, o código do produto seja a PK. Neste caso, não poderão existir dois produtos com o mesmo código.

Caso uma aplicação tente gravar um código duplicado na tabela produto, o banco de dados impede sua gravação, garantindo a consistência dos dados ali gravados. Chamamos esta validação de *constraint*. As *constraints* são verificadas sempre que houver uma tentativa e inclusão ou alteração de dados no banco de dados.

Quando criamos uma PK em uma tabela, automaticamente é criada uma constraint de banco de dados para efetuar esta validação que impede a duplicidade de dados na PK.

Observação: caso a PK seja composta por mais de uma coluna, não se admite duplicidade para o conjunto dessas colunas. Por exemplo, no cadastro de contas correntes de um banco pode haver várias contas da agência 10 e pode haver várias contas com o número 1000. Porém, esses números, em conjunto, só poderão ocorrer uma única vez, ou seja, a conta 1000 na agência 10 só pode ser cadastrada uma vez.



Regras de integridade e consistência

A criação das chaves em um banco de dados traz vantagens de qualidade para os dados que ali serão gravados. Tanto PKs como FKs fazem o banco de dados efetuar validações nos dados que se tenta gravar no banco. Desta forma, mesmo que a aplicação não efetue as validações necessárias, o banco de dados não permite a gravação de dados que firam a integridade e a consistência do banco.

CONTA

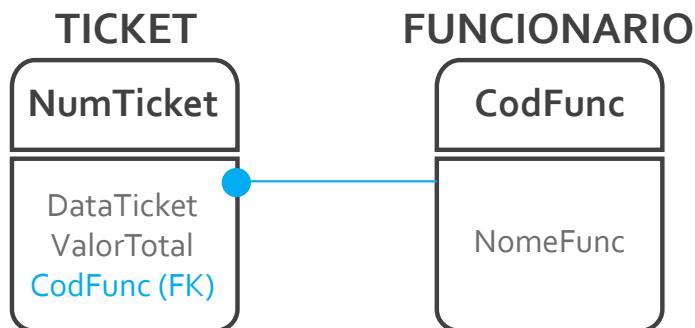


Regras relativas a FKs

As FKs são produtos de relacionamentos entre tabelas. No exemplo a seguir, em que relaciona-se funcionário a um ticket emitido em uma padaria, o código do funcionário é gravado no ticket emitido.

Suponhamos que haja 3 funcionários cadastrados na tabela funcionário:

- 1 Pedro
- 2 Tiago
- 3 João



Ao preencher os dados de um novo ticket (ou ao alterar um já existente), só será possível preencher a coluna CodFunc com os valores 1, 2 ou 3. Caso uma aplicação ou qualquer outro meio tente preencher esta coluna com um valor 4, por exemplo, o banco de dados impedirá a gravação, pois não há nenhum elemento na tabela funcionário cuja PK seja 4.

Este mecanismo também é uma **constraint** de banco de dados e é criado quando criamos o relacionamento entre as duas tabelas no banco de dados.

Este tipo de validação é chamada de **integridade referencial**. Recebe este nome, pois a FK de uma tabela referencia a PK de outra tabela. E justamente o que se pretende garantir aqui é que os dados estejam consistentes e íntegros entre estas tabelas.

Exercícios do Módulo 2

1) Qual alternativa ilustra melhor os principais objetivos da modelagem relacional?

- a. Preparar um desenho ótimo do banco de dados, separando a informação em tabelas e formatando todas as chaves-primárias e estrangeiras.
- b. Criar um modelo que permita o armazenamento da informação sem redundâncias, com integridade de dados e permitindo a extração de dados para sistemas transacionais e grandes sistemas de apoio à decisão através de ferramentas de BI.
- c. Garantir que não haverá redundância alguma no banco de dados.
- d. Criar um modelo que será implementado posteriormente em um sistema gerenciador de banco de dados.
- e. Projetar um banco de dados otimizado com o mínimo de redundância possível, garantindo a integridade dos dados e organizando a informação de modo que possa ser recuperada corretamente, principalmente por sistemas transacionais.

2) Um modelo de dados é:

- a. Um conjunto de relações em um banco de dados.
- b. Uma coleção de dados armazenados
- c. A descrição das informações que estarão armazenadas no banco de dados.
- d. Um modelo de dados armazenados.
- e. Representação abstrata do banco de dados, com tabelas e relacionamentos.

3) Qual das seguintes funções NÃO pode ser atribuída a um modelo de dados conceitual?

- a. Pode ser utilizado em uma conversa entre o analista de sistemas e os usuários finais para ajudar na discussão de como funciona o negócio do cliente.
- b. Representar os relacionamentos entre tabelas.
- c. Permitir que os desenvolvedores de programas entendam a estrutura de tabelas do banco de dados.
- d. Mostrar quais dados estão armazenados em um banco de dados.
- e. É uma ideia abstrata do banco de dados que deve ser implementada em um SGBD.

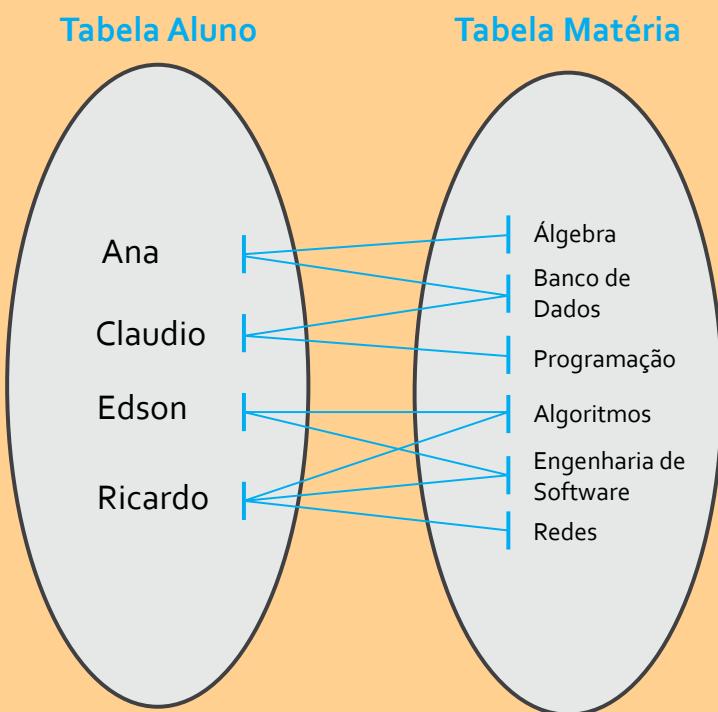
4) Classifique as afirmações abaixo em (V)erdadeira ou (Falsa):

- () Uma PK representa uma estrutura externa à tabela que a identifica de forma única, não admitindo repetição.
- () Uma FK é uma chave primária, que é composta por uma ou mais colunas da tabela e que a identifica de forma única.
- () Uma FK é composta por uma ou mais colunas, trazidas de outra tabela quando da criação de um relacionamento entre as duas tabelas. A(s) coluna(s) trazida(s) representam a chave primária da outra tabela.
- () Uma PK não existe se não tiver uma FK associada.

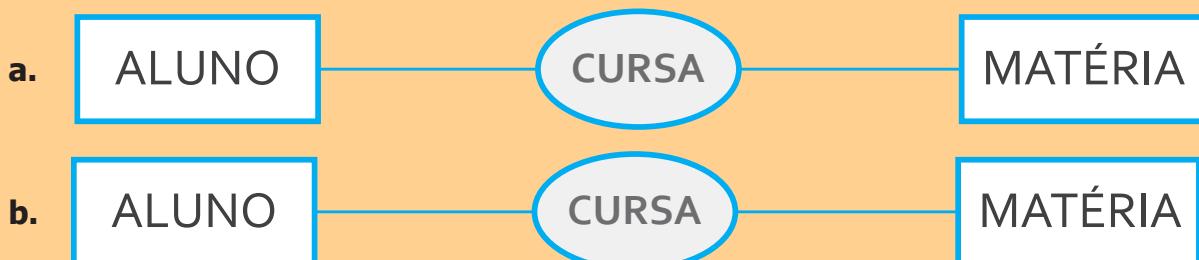
5) Um conceito muito importante no modelo relacional é a cardinalidade, que indica:

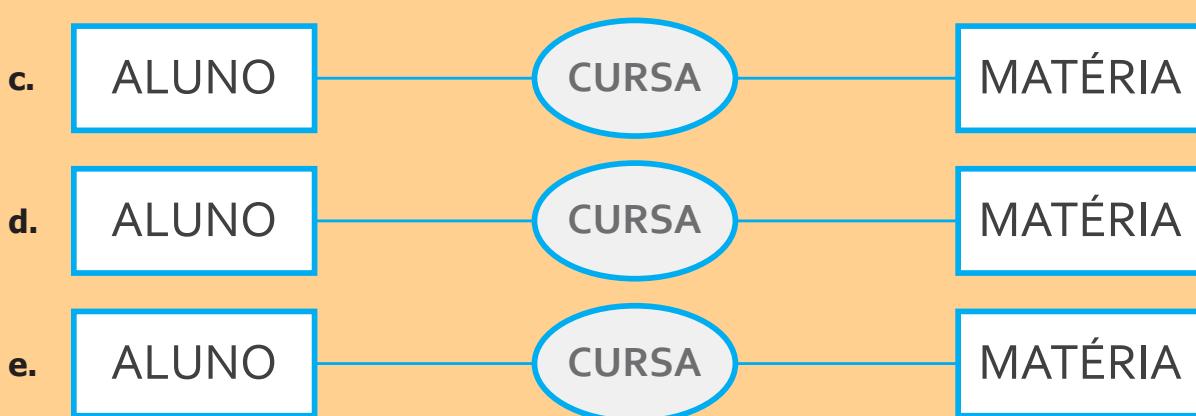
- a. A quantidade máxima de linhas suportada por uma tabela.
- b. A quantidade de campos permitidos em uma chave-primária.
- c. Quantas vezes a chave-primária de uma tabela pode ser relacionada com os atributos desta tabela.
- d. Em uma relação entre duas tabelas, quantas vezes um mesmo elemento de uma tabela pode estar relacionado a elementos de outra tabela.
- e. Em uma relação entre duas colunas de uma tabela, quantas vezes uma coluna pode estar relacionada a outra coluna.

6) Considere a figura abaixo:



Um banco de dados é representado pelas entidades aluno e matéria (disciplina), como visualizado na figura acima. De acordo com a figura, qual a representação correta do Modelo Entidade-Relacionamento (MER):





7) Uma escola possui um sistema informatizado para controle de informações. Neste sistema, os alunos são cadastrados com nome, número de matrícula, data de nascimento, e-mail e telefones de contato. As disciplinas são cadastradas com nome, carga horária, código e sempre estão associadas a um curso. Os cursos são cadastrados com nome, código e identificação de tipo (bacharelado, tecnólogo ou licenciatura).

A cada semestre o aluno efetua sua matrícula. No ato da matrícula, o aluno deve indicar quais disciplinas cursará naquele semestre.

A partir da descrição acima, elabora o DER e modelo de dados. Indique claramente as tabelas e os relacionamentos. No modelo de dados, evidencie os atributos, chaves primárias e estrangeiras.

De preferência, produza este modelo utilizando o MySQL WorkBench.

8) Considere a descrição a seguir para elaborar o DER e Modelo de Dados. Indique claramente as tabelas e os relacionamentos. No Modelo de Dados, evidencie os atributos, chaves primárias e estrangeiras.

Uma empresa registra em seus sistemas os cargos de seus funcionários. Registre também, uma tabela de Departamento, com Nome e Código dos funcionários, registre nome completo, data de nascimento, CPF e Salário Atual. Cada funcionário está associado a um cargo e a um departamento. A empresa é do ramo de Engenharia e executa vários projetos para clientes. Os clientes são registrados com CNPJ e Nome Completo. Os projetos são cadastrados com Nome, Data de Início, Data Fim. Há uma associação entre Projetos e Clientes (um projeto só pode ser de um cliente). Há associações entre Funcionários e Projetos (um projeto pode ter participação de mais de um funcionário).

9) A tabela abaixo está em que forma normal?

ALUNO	
idALUNO	INT
Nom eAluno	VARCHAR(45)
D tNascimento	DATE
CPF	INT (11)
RG	VARCHAR(15)
Fone1	VARCHAR(45)
Fone2	VARCHAR(45)
Fone3	VARCHAR(45)

- a.** 1FN pois possui colunas multivaloradas
- b.** 2FN pois possui colunas multivaloradas
- c.** 3FN pois possui apenas uma PK
- d.** Nenhuma, pois possui colunas multivaloradas
- e.** Não é possível determinar.

10) Classifique em (V)erdadeira ou (Falsa) as afirmações abaixo:

- () PKs e FKs são objetos do Banco de Dados.
- () Constraints são regras de validação que o MySQL executa após incluir uma informação em uma Tabela do Banco de Dados.
- () PKs e FKs são Constraints.
- () Considerando que temos uma tabela A relacionada a uma tabela B e que a PK de A é uma FK em B. Quando tenta-se incluir um valor na FK de B é verificado se este valor existe na PK de alguma linha de A. Isto é chamado de Integridade Referencial.
- () Considerando que temos uma tabela A relacionada a uma tabela B e que a PK de A é uma FK em B. Quando tenta-se incluir um valor na PK de B é verificado se este valor existe na PK de alguma linha de A. Isto é chamado de Integridade Referencial.

11) Considere a descrição a seguir para elaborar o DER e modelo de dados. Indique claramente as tabelas e os relacionamentos. No Modelo de Dados, evidencie os atributos, chaves primárias e estrangeiras.

Uma biblioteca cadastra os livros com título, número do tombo (código para localização), ano de publicação, edição e informação de editora e autor(es). Uma editora é cadastrada com nome e UF e pode editar vários livros. Os autores são cadastrados com nomes completos. Um autor pode escrever vários livros e um livro pode ter mais de um autor. Quando a biblioteca efetua o empréstimo do livro, registra a data do empréstimo, o número do tombo e a data prevista de devolução. Quando o livro é devolvido, é registrada a data efetiva de devolução.

12) Mapeamento: Represente as tabelas do modelo de dados do exercício 11 como mapeamento.

Módulo 3 – Criação de tabelas e manipulação de dados

Caro(a) aluno(a), mantenha-se animado! Neste módulo que iniciamos agora trataremos diretamente com o SGBD, criando tabelas no banco de dados além de manipular dados dentro destas tabelas. Começaremos com os comandos DDL (para criar as tabelas que precisaremos) e usaremos comandos DML para manipular dados (incluir dados em tabelas, recuperá-los através de consultas, alterá-los e excluí-los). É um módulo bem interessante e você poderá utilizar seu computador com o MySQL instalado para reproduzir os exemplos, tornando seu estudo mais atraente e aprendendo mais! É importante você criar outros exemplos além dos que aqui são apresentados para aprender mais. Então vamos lá e bons estudos!

Aula 6 – Criando um banco de dados e objetos

Criando um banco de dados – projeto curso

O MySQL permite a criação de vários bancos de dados ou *databases* de trabalho. Ele já possui alguns *defaults*, mas podemos criar databases (DBs) separados para organizar nossos trabalhos por assunto ou por sistema.

Entre no MySQL, acesse a Aba de banco de dados e veja a lista de DBs já criados, como mostra a figura 6.1.

The screenshot shows the phpMyAdmin interface with the following details:

- Top Navigation Bar:** Shows "Servidor 127.0.0.1" and tabs for "Base de Dados", "SQL", "Estado", "Utilizadores", and "Expo".
- Left Sidebar:** Displays a tree view of databases: cdcol, information_schema, mysql, performance_schema, phpmyadmin, test, and webauth.
- Main Content Area:**
 - Header:** "Base de Dados" with a "Create database" button and a dropdown for "Agrupamento (Collation)".
 - Table:** Shows a list of existing databases with checkboxes and "Verificar Privilégios" links. The "mysql" database is currently selected.
 - Total:** Shows a total of 7 databases.
 - Bottom Buttons:** Includes "Todos", "Com os seleccionados:", "Eliminar", and "Activar as estatísticas".

Figura 6.1 – MySQL - Aba de Base de Dados.



Vamos criar um DB para trabalhar nossos exemplos de aula, dando início ao nosso projeto curso. Desta forma vamos aprender na prática. Introduziremos a parte teórica e faremos a parte prática compondo o nosso projeto.

Os nomes do DB, das tabelas e das colunas são de escolha do usuário (desenvolvedor), mas sugerimos que você utilize os mesmos nomes apresentados nos exemplos de aula para que você possa comparar mais facilmente seus resultados.

Importante: O SQL não diferencia letras maiúsculas de letras minúsculas.



Vamos criar nosso database curso:

- 1) Preencha o nome do curso no campo *create database* e clique em CRIAR:



- 2) Será criado um novo DB denominado curso e apresentado nas listas de DBs. Veja:

Base de Dados	Ações
cdccl	Verificar Privilégios
CURSO	Verificar Privilégios
information_schema	Verificar Privilégios
mysql	Verificar Privilégios
performance_schema	Verificar Privilégios
phpmyadmin	Verificar Privilégios
test	Verificar Privilégios
webauth	Verificar Privilégios
Total: 8	

Criando tabelas no banco de dados

O phpMyAdmin tem interface amigável para criação de tabelas, mas vamos abordar aqui a criação das tabelas através de comandos DDL da SQL.

Comandos *create table* e *describe*

Para se criar uma tabela em um banco de dados usando o SQL, deve-se usar o comando *create table*. A sintaxe do *create table* é:

CREATE TABLE	Cria tabelas no banco de dados
	<pre>CREATE TABLE nome_tabela (nome_atributo1 datatype [NULL/NOTNULL] [DE- FAULT valor] [AUTO_INCREMENT], nome_atributo2 datatype ... : nome_atributoN datatype ... [PRIMARY KEY (nome_atributoX)], [FOREIGN KEY (nome_atributoY) REFERENCES nome_tabela_origem (nome_atributo_ori- gem)])</pre>
Sintaxe:	:

Na sintaxe dos comandos, o texto entre colchetes [] é opcional e seu uso depende da necessidade de sua aplicação.

Elementos com comando *create table*:

nome_tabela	Nome que você determina para a tabela a ser criada.
nome_atributoX	Nome de cada coluna que comporá a tabela.
datatype	É o tipo de dado a ser utilizado na coluna (Inteiro, String, ...).

NULL / NOT NULL	Indica se a coluna pode ser gravada com valor nulo (NULL) ou se seu preenchimento é obrigatório (NOT NULL).
DEFAULT valor	Indica um conteúdo que será inserido automaticamente naquela coluna se não for informado outro conteúdo.
AUTO_INCREMENT	Cria uma numeração automática na coluna; não sendo mais necessário indicar valor para a coluna. É usado com chaves primárias para numerar automaticamente os elementos de uma tabela.
PRIMARY KEY	Uma ou mais colunas que compõem a PK da tabela.
FOREIGN KEY	Uma ou mais colunas que relacionadas a PK de outra tabela (indicada após REFERENCES). Essa(s) coluna(s) forma(m) a FK da tabela.

- Vamos a um exemplo do comando *create table*. Vamos criar uma tabela com algumas informações de alunos:

Aluno		
Coluna	Tipo	Tamanho
Matrícula	INT	15
Nome	VARCHAR	40
CPF	INT	11
DtNascimento	DATE	
Idade	INT	3

*String de tamanho 40
Coluna tipo data (dia, mês, ano).*

O comando SQL para criação desta tabela seria:

```
CREATE TABLE ALUNO (
    Matricula      INT(15)      NOT NULL,
    Nome           VARCHAR(40)   NULL,
    CPF            INT(11)      NULL,
    DtNascimento   DATE        NULL,
    Idade          INT(3)       NULL,
    PRIMARY KEY (Matricula)
);
```

No comando *create table*, dentro dos parênteses são declaradas as colunas que farão parte da tabela. É obrigatório separar as colunas, umas das outras, por vírgula, mas não é necessário que fiquem em linhas separadas. A linguagem SQL aceita que os comandos sejam escritos em uma única linha ou em várias linhas. Para o MySQL, o ponto e vírgula indica o final do comando.

À frente de cada coluna, é especificado o tipo de dado, ou seja, o tipo de informação que será armazenada naquela coluna. Informamos também se a coluna pode ter valor nulo ou não.

Algumas observações sobre alguns tipos de dados:

CHAR(N) ou VARCHAR(N)	N é a quantidade de bytes reservado para conteúdo. Ambos admitem conteúdo <i>string</i> . CHAR é usado para uma quantidade exata de caracteres e VARCHAR é usado para indicar um tamanho máximo. Se CHAR for usado, o espaço ocupado será exatamente igual ao indicado. Com VARCHAR, se não forem utilizadas todas as posições pelo conteúdo, será poupado espaço.
DATE	É um tipo próprio para armazenar datas, efetuando consistências como mês de 1 a 12, dias do mês de acordo com o mês (por exemplo, não aceita dia 31 de abril, nem dia 30 de fevereiro).
DECIMAL (T,D)	Admite apenas valores numéricos e permite ponto flutuante. T é a quantidade total de dígitos e D é a quantidade de dígitos após a cada decimal.

Vamos criar, agora, duas tabelas com relacionamento entre elas e poderemos observar a ligação de uma FK com uma PK.

Para melhor entendimento, você deve executar os comandos no MySQL. Vamos usar o banco de dados curso, que você já criou, para prosseguirmos em nosso projeto. Para isso, clique sobre o nome do DB (curso). Vamos criar as tabelas abaixo utilizando o comando DLL do SQL *create table*, de acordo com as especificações do modelo de dados da figura 6.2.1.

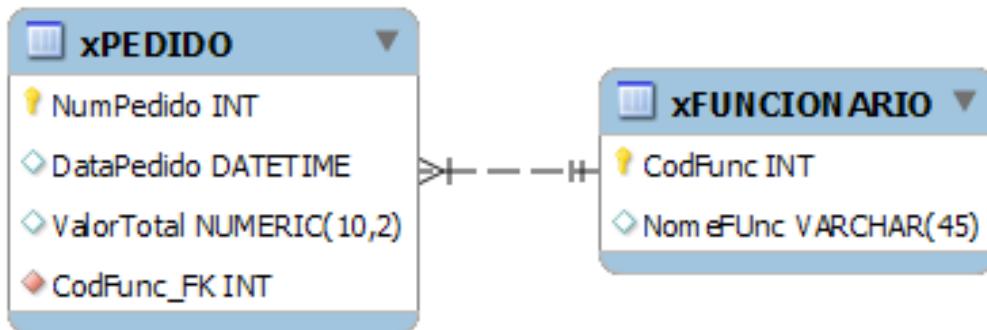


Figura 6.2.1 – Modelo de Dados Pedido X Funcionário.

O quadro 6.2.1 mostra o código que deve ser inserido para a criação das tabelas.

Quadro 6.2.1- Usando `create table` para criar as tabelas do DB CURSO

```

CREATE TABLE xFUNCIONARIO (
    CodFunc INT NOT NULL,
    NomeFunc VARCHAR(45) NULL,
    PRIMARY KEY (CodFunc)
);

CREATE TABLE xPEDIDO (
    NumPedido INT NOT NULL,
    DataPedido DATETIME NULL,
    ValorTotal DECIMAL(10,2) NULL,
    CodFunc_FK INT NULL,
    PRIMARY KEY (NumPedido),
    FOREIGN KEY (CodFunc_FK)
        REFERENCES xFUNCIONARIO (CodFunc)
);

```

Veja que com os comandos do quadro 6.2.1 criamos:

- a tabela xFUNCIONARIO
 - ◊ sua PK é CodFunc;
- a tabela xPEDIDO;
 - ◊ sua PK é NumPedido;
 - ◊ sua FK é CodFunc_FK que aponta para a tabela xFUNCIONARIO, coluna CodFunc.

Você pode ver a estrutura da tabela que você criou ou de outra tabela já existente usando o comando `describe`. Este comando exibe a estrutura da tabela. No editor de comandos, digite e execute o comando seguinte para ver a estrutura da tabela x FUNCIONÁRIO:

DESCRIBE x FUNCIONÁRIO;

Resultado:

Field	Type	Null	Key	Default	Extra
CodFunc	int(11)	NO	PRI	NULL	
NomeFunc	varchar(45)	YES		NULL	

O comando *alter table*

Para se alterar a estrutura de uma tabela, para acrescentar ou excluir colunas ou chaves, em um banco de dados usando o SQL, deve-se usar o comando **ALTER TABLE**. A sintaxe do *alter table* é:

ALTER TABLE	Altera a estrutura de uma tabela já criada.
Sintaxe: <pre> ALTER TABLE nome_tabela ([ADD COLUMN nome_ atributoX datatype [N U L L / N O T N U L L] [DEFAULT valor][AUTO_INCREMENT] [PRIMARY KEY] ,] [DROP COLUMN nome_atributoY] [ADD PRIMARY KEY (nome_atributoX)]] [DROP PRIMARY KEY], [ADD FOREIGN KEY (nome_atributoY) REFERENCES nome_tabela_origem (nome_atributo_ori- gem)], [DROP FOREIGN KEY (nome_atributoY)]) </pre>	ALTER TABLE nome_tabela ([ADD COLUMN nome_ atributoX datatype [N U L L / N O T N U L L] [DEFAULT valor][AUTO_INCREMENT] [PRIMARY KEY] ,] [DROP COLUMN nome_atributoY] [ADD PRIMARY KEY (nome_atributoX)]] [DROP PRIMARY KEY], [ADD FOREIGN KEY (nome_atributoY) REFERENCES nome_tabela_origem (nome_atributo_ori- gem)], [DROP FOREIGN KEY (nome_atributoY)])

Você pode alterar a estrutura da tabela xFUNCIONARIO que você criou. Como exemplo, vamos incluir uma nova coluna nesta tabela. Para isso, entre no editor de comandos, digite e execute o comando abaixo:

ALTER TABLE xFUNCIONARIO

ADD COLUMN Idade INT NULL;

Após este comando, a estrutura da tabela (verificada pelo DESCRIBE) fica assim:

Field	Type	Null	Key	Default	Extra
CodFunc	int(11)	NO	PRI	<i>NULL</i>	
NomeFunc	varchar(45)	YES		<i>NULL</i>	
Idade	int(11)	YES		<i>NULL</i>	

Você pode também excluir uma coluna de uma tabela. Como exemplo, vamos excluir a coluna que acabamos de incluir. No editor de comandos, digite e execute o comando abaixo:

ALTER TABLE xFUNCIONARIO

DROP COLUMN **Idade;**

Após este comando, a estrutura da tabela (verificada pelo DESCRIBE) fica assim:

Field	Type	Null	Key	Default	Extra
CodFunc	int(11)	NO	PRI	<i>NULL</i>	
NomeFunc	varchar(45)	YES		<i>NULL</i>	

O comando drop table e o comando show table

Para se excluir uma tabela em um banco de dados usando o SQL, deve-se usar o comando **drop table**. A sintaxe do *drop table* é:

DROP TABLE

Exclui uma tabela do banco de dados.

Sintaxe: **DROP TABLE nome_tabela;**

Para exemplificarmos o uso do *drop table*, antes de excluirmos uma tabela de nosso banco de dados curso, vamos criar uma cópia de uma das tabelas que criamos.

O comando a seguir copia a estrutura e os dados da tabela original xPEDIDO para a tabela destino x pedidocopy:

create table x pedidocopy select * from xpedido;

Vamos usar o comando *describe* para verificarmos que a estrutura da tabela x pedidocopy foi realmente copiada da x pedido. Utilize o comando e veja seu resultado:

describe x pedidocopy;

Field	Type	Null	Key	Default	Extra
NumPedido	int(11)	NO		<i>NULL</i>	
DataPedido	datetime	YES		<i>NULL</i>	
ValorTotal	decimal(10,2)	YES		<i>NULL</i>	
CodFunc_FK	int(11)	YES		<i>NULL</i>	

Você também pode verificar que a tabela foi copiada listando as tabelas existentes no banco de dados usando o comando *show tables*:

Show Tables;

O resultado é uma lista com as tabelas existentes no banco de dados que estamos usando:

Tables_in_curso
xfuncionario
xpedido
xpedidocopy

Agora vamos excluir a tabela x pedidocopy, praticando o comando drop table:

drop table x pedidocopy;

Para verificar a eficácia deste comando, vamos listar novamente as tabelas existentes no DB curso:

Show Tables;

Tables_in_curso
xfuncionario
xpedido

Caro(a) aluno(a), a partir daqui estamos aptos a incluir dados nas tabelas que criamos para podermos entender melhor como funcionam as restrições (constraints) nela inseridas.

O comando *insert* – inserindo dados em uma tabela

Para inserirmos dados em uma tabela de um banco de dados usando o SQL, devemos usar o comando **INSERT**. A sintaxe do INSERT é:

INSERT Inclui elementos novos em uma tabela	
Sintaxe:	INSERT [INTO] nome_tabela
	[(coluna1, coluna2, ..., colunaN)]
	VALUES (conteúdo1, conteúdo2, ..., conteúdoN)

Também na sintaxe do comando *insert*, o texto entre colchetes **[]** é opcional e seu uso depende da necessidade da aplicação. Os campos do comando são assim definidos:

nome_tabela	Nome da tabela na qual o novo elemento será inserido.
colunaX	Nomes das colunas, não é necessário que estejam na mesma ordem na tabela.
conteúdoX	É o conteúdo que será gravado nas colunas correspondentes da tabela, os conteúdos devem obedecer a sequência em que as colunas estão apresentadas.

Para exercitarmos o comando *insert*, vamos incluir os seguintes elementos na tabela x funcionário:

Para incluir estes dados na tabela x funcionário, entre no editor de comandos, digite e execute os comandos apresentados no quadro 6.3. Observe que usaremos variações do comando *insert*.

Quadro 6.3 – Inserindo dados na tabela x funcionário com *insert*

```

INSERT INTO x FUNCIONARIO (CodFunc, NomeFunc) VALUES (1, 'Pedro');

INSERT x FUNCIONARIO (CodFunc, NomeFunc) VALUES (2, 'Paulo');

INSERT INTO x FUNCIONARIO VALUES (3, 'João');

INSERT INTO x FUNCIONARIO (NomeFunc, CodFunc) VALUES ('Maria', 4);

INSERT INTO x FUNCIONARIO (CodFunc, NomeFunc) VALUES (5, 'Antonia');
  
```

Cada uma das linhas do quadro é um comando que insere dados na tabela. Assim, 5 elementos foram inseridos ao se executar esses 5 comandos.

Observe no quadro 6.3 que o primeiro comando apresenta a sintaxe mais completa. Os itens destacados em azul nesta primeira linha são opcionais, dependendo de algumas condições. O INTO é opcional no MySQL. A lista de campos a ser incluída (**CodFunc**, **NomeFunc**) é dispensável se:

- todas as colunas receberem um conteúdo em *values*;
- os conteúdos em *values* estiverem na mesma ordem em que as colunas estão criadas na tabela.

Na quarta linha do quadro 6.3, observe que as colunas foram listadas em ordem inversa (**NomeFunc**, **CodFunc**). Os conteúdos foram invertidos também (**'Maria'**, **4**), respeitando essa ordem.

Insert – Exemplos de uso incorreto

Vamos ver agora algumas situações em que o INSERT é usado de forma errada. O comando a seguir tem problemas que foram inseridos de forma proposital para alertá-lo, caro aluno, de alguns erros comumente cometidos.

```
INSERT INTO xFUNCIONARIO (CodFunc, NomeFunc) VALUES (1, 'Pedro');
```

Se o comando for executado de forma repetida, dará certo na primeira vez, mas na segunda vez apresentará o seguinte erro: “**#1062 - Duplicate entry ‘1’ for key ‘PRIMARY’**”. Esta mensagem significa que foi ferida a regra da PK. Lembre-se que a PK não pode ter valores repetidos. Neste caso, como já tínhamos na tabela um Funcionário com o Código “1”, a inclusão não é permitida.

Toda vez que executamos um comando INSERT, antes de efetuar a inclusão, o MySQL efetua a verificação das CONSTRAINTs a fim de não permitir a inclusão de dados inconsistentes no DB. No caso apresentado, o comando INSERT não é concluído e os dados NÃO são gravados na tabela.

```
INSERT xFUNCIONARIO VALUES (8);
```

Se este comando for executado da forma como está, gerará o seguinte erro:

“**#1136 - Column count doesn’t match value count at row 1**”. Isto acontece porque se não forem declaradas as colunas que receberão dados, o MySQL assume que serão todas as colunas e na ordem em que estão criadas no banco de dados. Neste caso, o comando tenta incluir apenas um conteúdo e o MySQL está esperando dois conteúdos, pois é a quantidade de colunas da tabela.

Há algumas outras variações do comando INSERT que trataremos mais à frente.

O comando select – consultando dados em uma tabela

Vamos aprender, agora, caro aluno, como consultar dados de tabelas. Iniciaremos com consultas simples e depois passaremos as consultas mais elaboradas, envolvendo várias tabelas. Sem dúvida, o comando SELECT é o comando mais utilizado, uma vez que, em geral, os sistemas utilizam muito mais vezes comandos de consulta de dados do que comandos que incluem ou alteram ou excluem dados.

Para consultarmos dados em uma tabela de um banco de dados usando o SQL, devemos usar o comando `select`. A sintaxe do `select` é:

SELECT	Consulta dados de uma ou mais tabelas
Sintaxe:	<pre> SELECT [DISTINCT] nome_atributo1 ou Expressão1, nome_atributo2 ou Expressão2, ..., nome_atributoN ou ExpressãoN FROM nome_tabela1, ... nome_tabelaN [WHERE (condições)] [GROUP BY nome_atributo1,... nome_atributoN] [HAVING (condições)] [{INTERSECT MINUS UNION} comando_select] [{ORDER BY nome_atributo {ASC DESC}]</pre>

Também na sintaxe do comando `select`, o texto entre colchetes [] é opcional e seu uso depende da necessidade da aplicação. Os campos do comando são assim definidos:

nome_atributoX ou ExpressãoX	Após o <code>select</code> , podemos declarar nomes de colunas de tabelas ou expressões (exemplo: <code>CodFunc + 2</code>). Cada coluna/expressão deve ser separada de outra por uma vírgula. Podem estar na mesma linha ou serem divididos em várias linhas. Você pode escolher a ordem em que as colunas serão apresentadas pelo comando <code>select</code> , independente da ordem em que as colunas tenham sido criadas na tabela. Caso queiramos todas as colunas de uma tabela, na ordem em que estão na tabela, basta utilizarmos um "*" no lugar das colunas.
DISTINCT	Esta opção não exibe linhas com conteúdos repetidos.
nome_tabela	Nome da tabela que está sendo consultada.
condições	É comum querermos consultar uma parte dos dados da tabela. Para isto, podemos filtrar os resultados utilizando a cláusula WHERE . Esta é seguida de uma condição (simples ou composta) que será aplicada aos dados consultados, filtrando o resultado e só apresentando os dados que atendem às condições declaradas.
GROUP BY	Agrupa os resultados consolidando-os por um ou mais atributos. Esta cláusula é utilizada junto com funções estatísticas que veremos mais à frente.
HAVING	Funciona como um filtro similar ao <code>where</code> , mas é aplicado após a consolidação dos resultados de um group by .
INTERSECT ou MINUS ou UNION	Executam operações de conjuntos entre vários comandos <code>select</code> .
ORDER BY	Classifica o resultado apresentado pelo <code>select</code> , alterando sua ordem, independente da ordem em que os dados (elementos) estejam gravados nas Tabelas. É possível escolher uma ou mais colunas para ordenar a apresentação dos resultados e indicar se queremos que os dados sejam exibidos por em ordem ascendente (ASC) ou descendente ou (DESC) dessas colunas.

Vamos ver alguns exemplos com a tabela xfuncionario. Como ela é pequena, os exemplos serão limitados e mais simplificados. Quando criarmos mais tabelas em nossa base de dados faremos uso de outras opções do comando select.

Para você testar estes comandos SELECT, entre no editor de comandos, digite e execute um a um os comandos a seguir, observando seus resultados.

a. Listando todos os elementos da tabela

```
SELECT * FROM xFUNCIONARIO;
```

Resultado:

CodFunc	NomeFunc
1	Pedro
2	Paulo
3	João
4	Maria
5	Antonia

b. Exibindo colunas em outra ordem

```
SELECT NomeFunc, CodFunc FROM xFUNCIONARIO;
```

Resultado:

NomeFunc	CodFunc
Pedro	1
Paulo	2
João	3
Maria	4
Antonia	5

Operadores do MySQL – Aritméticos, Lógicos e de Comparação

Há operadores que nos auxiliam na composição de alguns comandos. Vamos conhecê-los agora. Eles podem nos ajudar, entre outras coisas, nas condições de filtro em um comando select. Apresentamos esses operadores.

c. Exibindo os funcionários em ordem alfabética de nome

```
SELECT * FROM xFUNCIONARIO ORDER BY NomeFunc;
```

Resultado:

CodFunc	NomeFunc
5	Antonia
3	João
4	Maria
2	Paulo
1	Pedro

Neste caso, os dados são apresentados em ordem ascendente (crescente de nome), independente da ordem em que estão armazenados no banco de dados. O default de **ORDER BY** é **ASC** (ascendente).

Exibindo os funcionários em ordem decrescente de nome

```
SELECT * FROM xFUNCIONARIO ORDER BY NomeFunc DESC;
```

Resultado:

CodFunc	NomeFunc
1	Pedro
2	Paulo
4	Maria
3	João
5	Antonia

Aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão

Lógicos

AND ou &&	"E" em condição composta
OR ou 	"OU" em condição composta
NOT ou !	Negação; inverte o valor lógico da condição

Comparação

>	Maior
>=	Maior ou igual
<	Menor
<=	Menor ou igual
<> ou !=	Diferente
LIKE	Valor similar
BETWEEN	Faixa de valores
IN	Quem estiver na lista
NOT IN	Quem não estiver na lista

Para você testar estes operadores no comando SELECT, execute os comandos apresentados nos itens a seguir.

- Filtrando dados: só códigos maiores que 3.**

```
SELECT * FROM xFUNCIONARIO WHERE CodFunc > 3;
```

Resultado:

CodFunc	NomeFunc
4	Maria
5	Antonia

O resultado é filtrado de acordo com a condição, apresentando apenas os registros que têm código de funcionário maior que 3.

- Filtrando dados: condição composta apresenta códigos entre 1 e 4.**

```
SELECT * FROM xFUNCIONARIO WHERE CodFunc > 1 AND CodFunc < 4;
```

Resultado:

CodFunc	NomeFunc
2	Paulo
3	João

- Quando queremos um range de valores podemos usar BETWEEN para facilitar nossa consulta.**

```
SELECT * FROM xFUNCIONARIO WHERE CodFunc BETWEEN 2 AND 3;
```

Resultado:

CodFunc	NomeFunc
2	Paulo
3	João

O resultado é o mesmo do anterior, mas escrevemos menos. Note que os valores limite declarados após between são incluídos no resultado. Ou seja, este comando é equivalente ao seguinte comando:

```
SELECT      * FROM xFUNCIONARIO
WHERE      CodFunc >= 2
AND        CodFunc <= 3;
```

Filtrando strings com “%” em um select

Podemos filtrar os dados usando condições que façam comparações aproximadas, ou seja, filtrem partes de uma coluna *string*. Para tanto, utilizamos `like` no lugar do operador `=` e o caractere “`%`” que funciona como um “coninga”.

- Filtrando dados: todos os nomes que começem com ‘P’**

```
SELECT * FROM xFUNCIONARIO WHERE
NomeFunc LIKE 'P%';
```

Resultado:

CodFunc	NomeFunc
1	Pedro
2	Paulo

- Filtrando dados: todos os nomes que terminem com ‘A’.**

```
SELECT * FROM xFUNCIONARIO WHERE
NomeFunc LIKE '%A';
```

Resultado:

CodFunc	NomeFunc
4	Maria
5	Antonia

- Filtrando dados com partes em qualquer lugar do string.**

```
SELECT * FROM xFUNCIONARIO WHERE
NomeFunc LIKE '%R%';
```

Resultado:

CodFunc	NomeFunc
1	Pedro
4	Maria

- Filtrando dados com mais de um caractere**

```
SELECT * FROM xFUNCIONARIO WHERE
NomeFunc LIKE '%IA%';
```

Resultado:

CodFunc	NomeFunc
4	Maria
5	Antonia

Filtrando a partir de uma lista usando *in* ou *not in* em um *select*

Podemos consultar filtrando vários conteúdos de uma única vez. Se quisermos, por exemplo, os funcionários com códigos 1 ou 3 ou 5, podemos usar o comando:

```
SELECT      * FROM xFUNCIONARIO
WHERE      CodFunc      =      1
OR          CodFunc      =      3
OR          CodFunc      =      5;
```

Resultado:

CodFunc	NomeFunc
1	Pedro
3	João
5	Antonia

Mas podemos, também, fazer uso de **IN** ou **NOT IN** para estabelecer uma lista de valores, trazendo exatamente o mesmo resultado:

```
SELECT      * FROM xFUNCIONARIO
WHERE      CodFunc      IN (1, 3, 5);
```

Resultado:

CodFunc	NomeFunc
1	Pedro
3	João
5	Antonia

IN funciona como se houvesse vários operadores lógicos **ou** na condição. Assim, qualquer elemento que atenda a um dos conteúdos listados em IN serão considerados.

A negativa **not in** também pode ser usada e é uma consulta muito prática, como no comando:

```
SELECT * FROM xFUNCIONARIO
WHERE CodFunc NOT IN (1, 3, 5);
```

Resultado:

CodFunc	NomeFunc
2	Paulo
4	Maria

O resultado, claro, é o inverso, trazendo todos os elementos que não atendem aos itens listados em NOT IN. Um comando equivalente, produzindo o mesmo resultado, sem usar NOT IN, usando o operador lógico AND seria:

```
SELECT * FROM xFUNCIONARIO
WHERE CodFunc <> 1
AND CodFunc <> 3
AND CodFunc <> 5;
```

Resultado:

CodFunc	NomeFunc
2	Paulo
4	Maria

Aula 7 – Mais opções de uso dos comandos *insert* e *select*

Criando tabelas com relacionamentos

Além da tabela XFUNCIONARIO, com a qual estamos trabalhando, vamos inserir dados na tabela xPEDIDO e criarmos um relacionamento desta com a primeira. Isto nos permitirá observar situações adicionais.

Vamos inserir os seguintes dados na tabela xPEDIDO:

NumPedido	DataPedido	ValorTotal	CodFunc
100	10/01/2014	1.000,00	1
101	15/01/2014	500,25	2
102	20/02/2014	1.700,00	5
103	30/02/2014	800,00	1
104	05/02/2014	250,00	3
105	05/02/2014	1.314,50	1
106	15/02/2014	1.292,00	2
107	20/02/2014	915,25	5
108	08/03/2014	978,00	4
109	09/03/2014	1.250,00	5

Para isso, vamos usar os comandos INSERT mostrados no quadro 7.1.

Quadro 7.1- Comandos para inclusão de dados na tabela xPEDIDO

```

INSERT INTO xPEDIDO VALUES (100, '2014-01-10', 1000 , 1);

INSERT INTO xPEDIDO VALUES (101, '2014-01-15', 500.25, 2);

INSERT INTO xPEDIDO VALUES (102, '2014-01-20', 1700 , 5);

INSERT INTO xPEDIDO VALUES (103, '2014-01-30', 800 , 1);

INSERT INTO xPEDIDO VALUES (104, '2014-02-05', 250 , 3);

INSERT INTO xPEDIDO VALUES (105, '2014-02-05', 1314.5 , 1);

INSERT INTO xPEDIDO VALUES (106, '2014-02-15', 1292 , 2);

INSERT INTO xPEDIDO VALUES (107, '2014-02-25', 915.25, 5);

INSERT INTO xPEDIDO VALUES (108, '2014-03-08', 978 , 4);

INSERT INTO xPEDIDO VALUES (109, '2014-03-09', 1250 , 5);

```

A partir dos novos dados, vamos selecionar e apresentar seu conteúdo usando o comando:

```
SELECT * FROM xPEDIDO;
```

Resultado:

NumPedido	DataPedido	ValorTotal	CodFunc_FK
100	2014-01-10 00:00:00	1000.00	1
101	2014-01-15 00:00:00	500.25	2
102	2014-01-20 00:00:00	1700.00	5
103	2014-01-30 00:00:00	800.00	1
104	2014-02-05 00:00:00	250.00	3
105	2014-02-05 00:00:00	1314.50	1
106	2014-02-15 00:00:00	1292.00	2
107	2014-02-25 00:00:00	915.25	5
108	2014-03-08 00:00:00	978.00	4
109	2014-03-09 00:00:00	1250.00	5

Integridade referencial

Como comentamos anteriormente, ao criarmos um relacionamento entre duas tabelas, uma tabela recebe a PK de outra, configurando uma FK. Neste caso, a coluna FK será validada quando da inclusão de um elemento na tabela. O valor que se tenta gravar nesta coluna é validado contra a PK da tabela relacionada.

Vamos ver o que pode acontecer ao executarmos o comando:

```
INSERT INTO xPEDIDO VALUES (110, '2014-03-09', 1250 , 6);
```

Como resultado, temos a seguinte mensagem de erro:

#1452 - Cannot add or update a child row: a foreign key constraint fails (`curso`.`xpedido`, CONSTRAINT `xpedido_ibfk_1` FOREIGN KEY (`CodFunc_FK`) REFERENCES `xfuncionario`(`CodFunc`))

Veja que a mensagem referencia erro de **foreign key constraint**, apontando para a coluna CodFunc_FK que, por sua vez, referencia a tabela xFUNCIONARIO. O erro aconteceu porque o valor "6" não existe na PK de xFUNCIONARIO, ou seja, não existe nenhum funcionário com este código.

Desta forma, a inclusão **não** foi efetuada, pois a informação estava inconsistente com as regras de criação dessa tabela. A inclusão não foi efetuada, pois feriria a **integridade referencial**.

JOIN - Consultando dados de várias tabelas em um select

Uma variação do comando *select* inclui a cláusula *join*. Esta cláusula é usada quando desejamos recuperar dados de mais de uma tabela que mantenham, entre si, relacionamento. Todas as demais opções do comando *select* permanecem válidas com o uso do *join*.

A sintaxe do comando *select* com *join* é:

SELECT com JOIN	Consulta dados de tabelas que mantém relacionamento
Sintaxe: <pre> SELECT nome_atributo1 ou Expressão1, nome_atributo2 ou Expressão2, ..., nome_atributoN ou ExpressãoN FROM n o m e _ t a b e l a 1 JOIN nome_tabela1 ON tabela_Pai.PK = tabela_Filha.FK </pre>	SELECT nome_atributo1 ou Expressão1, nome_atributo2 ou Expressão2, ..., nome_atributoN ou ExpressãoN FROM n o m e _ t a b e l a 1 JOIN nome_tabela1 ON tabela_Pai.PK = tabela_Filha.FK

Os elementos são:

nome_tabela1 e nome_tabela2

São as tabelas envolvidas na consulta e que se relacionam. Ou seja, uma é tabela “pai” e outra tabela “filha” no relacionamento. A ordem em que são declaradas não importa.

ON

Indica quais colunas devem ser utilizadas para relacionar as tabelas.

Utilize aqui sempre:

PK da tabela “pai” e
FK da tabela “filha”

Novamente a ordem das tabelas não é importante, ou seja, tanto faz indicar “tabela_Pai.PK = tabela_Filha.FK” ou “tabela_Filha.FK = tabela_Pai.PK”

Em um mesmo *select* podem ser incluídas várias tabelas que se relacionam. Para cada tabela acrescentada após a primeira, use mais uma vez a cláusula **JOIN** com **ON**. Assim, se tivermos 5 tabelas, teremos 4 vezes o uso do JOIN.

Vamos a um exemplo de consulta envolvendo as tabelas xPEDIDO e xFUNCIONARIO. Veja que utilizamos a coluna CodFunc (PK da tabela “pai”, neste caso xPEDIDO) e CodFunc_FK (FK na tabela “filha”, neste caso xFUNCIONARIO). Execute o comando abaixo, caro aluno, e observe o resultado.

```
SELECT      *
FROM        xPEDIDO
JOIN        xFUNCIONARIO
ON          xPEDIDO.CodFunc_FK = xFUNCIONARIO.CodFunc;
```

Resultado:

NumPedido	DataPedido	ValorTotal	CodFunc_FK	CodFunc	NomeFunc
100	2014-01-10 00:00:00	1000.00	1	1	Pedro
103	2014-01-30 00:00:00	800.00	1	1	Pedro
105	2014-02-05 00:00:00	1314.50	1	1	Pedro
101	2014-01-15 00:00:00	500.25	2	2	Paulo
106	2014-02-15 00:00:00	1292.00	2	2	Paulo
104	2014-02-05 00:00:00	250.00	3	3	João
108	2014-03-08 00:00:00	978.00	4	4	Maria
102	2014-01-20 00:00:00	1700.00	5	5	Antonia
107	2014-02-25 00:00:00	915.25	5	5	Antonia
109	2014-03-09 00:00:00	1250.00	5	5	Antonia

Nós podemos utilizar um **alias** (apelido) para cada tabela a fim de escrever menos na cláusula **on**. Veja o exemplo em que colocamos um **alias** de uma letra para cada uma das tabelas e depois as utilizamos no **on**. Ao executarmos este comando, o resultado será o mesmo do *select* anterior.

```

SELECT      *
FROM        xPEDIDO          P
JOIN        xFUNCIONARIO      F
ON          P.CodFunc_FK     =     F.CodFunc;

```

Podemos escolher as colunas a apresentá-las na consulta, como no comando:

```

SELECT      NumPedido, DataPedido, ValorTotal, NomeFunc
FROM        xPEDIDO          P
JOIN        xFUNCIONARIO      F
ON          P.CodFunc_FK     =     F.CodFunc;

```

Resultado:

NumPedido	DataPedido	ValorTotal	NomeFunc
100	2014-01-10 00:00:00	1000.00	Pedro
103	2014-01-30 00:00:00	800.00	Pedro
105	2014-02-05 00:00:00	1314.50	Pedro
101	2014-01-15 00:00:00	500.25	Paulo
106	2014-02-15 00:00:00	1292.00	Paulo
104	2014-02-05 00:00:00	250.00	João
108	2014-03-08 00:00:00	978.00	Maria
102	2014-01-20 00:00:00	1700.00	Antonia
107	2014-02-25 00:00:00	915.25	Antonia
109	2014-03-09 00:00:00	1250.00	Antonia

Uma opção que temos é mudar o nome da coluna quando de sua exibição. Isso é feito acrescentando **AS** após o nome da coluna e indicando qual o *label* queremos que seja exibido.

```

SELECT      NumPedido, DataPedido, ValorTotal, NomeFunc AS Vendedor
FROM        xPEDIDO          P
JOIN        xFUNCIONARIO      F
ON          P.CodFunc_FK     =     F.CodFunc;

```

Resultado:

NumPedido	DataPedido	ValorTotal	Vendedor
100	2014-01-10 00:00:00	1000.00	Pedro
103	2014-01-30 00:00:00	800.00	Pedro
105	2014-02-05 00:00:00	1314.50	Pedro
101	2014-01-15 00:00:00	500.25	Paulo
106	2014-02-15 00:00:00	1292.00	Paulo
104	2014-02-05 00:00:00	250.00	João
108	2014-03-08 00:00:00	978.00	Maria
102	2014-01-20 00:00:00	1700.00	Antonia
107	2014-02-25 00:00:00	915.25	Antonia
109	2014-03-09 00:00:00	1250.00	Antonia

Select distinct — suprimindo repetições

Para entender o uso de **distinct**, vamos começar exibindo os nomes dos vendedores que efetuaram algum pedido:

```
SELECT      NomeFunc AS Vendedor
FROM        xPEDIDO          P
JOIN        xFUNCIONARIO      F
ON          P.CodFunc_FK    =    F.
CodFunc;
```

Resultado:

Vendedor
Pedro
Pedro
Pedro
Paulo
Paulo
João
Maria
Talita
Talita
Talita

Para obtermos uma lista simples dos vendedores que efetuaram algum pedido, precisamos eliminar as repetições na exibição do *select*.

Para isso, utilizamos a cláusula **distinct**. A opção *distinct* do comando *select* elimina exibições repetidas.

O comando a seguir suprime as repetições nos dados apresentados:

```
SELECT      DISTINCT     NomeFunc AS Vendedor
FROM        xPEDIDO          P
JOIN        xFUNCIONARIO      F
ON          P.CodFunc_FK    =    F.
CodFunc;
```

Resultado:

Vendedor
Pedro
Paulo
João
Maria
Talita

Observe, caro aluno, que os nomes aparecem de forma repetida, pois há vendedores que efetuaram mais de um pedido.

Aula 8 – Alterando e excluindo dados em tabelas

Update – alterando dados de tabelas

Para alterarmos dados já armazenados em tabelas, utilizamos o comando `update`.

A sintaxe do comando *update* é:

UPDATE	Altera dados já existentes em uma tabela		
	UPDATE	nome_tabela	
Sintaxe:	SET	nome_atributo1	= conteúdo1,
		nome_atributo2	= conteúdo2, ...,
		nome_atributoN	= conteúdoN
	[WHERE	(condições)]	

Os elementos do comando são assim definidos:

nome_tabela	É a tabela que terá seus dados alterados.
nome_atributoX	Nomes de colunas que devem ter seu conteúdo alterado.
ConteúdoX	Conteúdo que deve ser armazenado no atributo, substituindo seu valor original.
condições	Fazem parte da cláusula WHERE, que é opcional. São os critérios de filtro que selecionam os elementos da tabela que devem ser alterados.

Vamos mostrar alguns exemplos alterando os dados que já temos em nossas tabelas.

a. Trocando o nome de um funcionário: de Antonia para Talita.

```
UPDATE      xFUNCIONARIO  
  
SET        NomeFunc    =      'Talita'  
  
WHERE     CodFunc      =      5;  
  
SELECT      * FROM xFUNCIONARIO;
```

Resultado:

CodFunc	NomeFunc
1	Pedro
2	Paulo
3	João
4	Maria
5	Talita

Veja que o nome do último funcionário foi alterado.

A condição CodFunc = 5 restringiu a alteração apenas à linha da Antonia. Caso contrário, todas as linhas seriam alteradas.

b. Alterando um valor de um dos pedidos

```
UPDATE xPEDIDO
SET ValorTotal = 5000
WHERE NumPedido = 103;

SELECT * FROM xPEDIDO;
```

Resultado:

NumPedido	DataPedido	ValorTotal	CodFunc_FK
100	2014-01-10 00:00:00	1000.00	1
101	2014-01-15 00:00:00	500.25	2
102	2014-01-20 00:00:00	1700.00	5
103	2014-01-30 00:00:00	5000.00	1
104	2014-02-05 00:00:00	250.00	3
105	2014-02-05 00:00:00	1314.50	1
106	2014-02-15 00:00:00	1292.00	2
107	2014-02-25 00:00:00	915.25	5
108	2014-03-08 00:00:00	978.00	4
109	2014-03-09 00:00:00	1250.00	5

A alteração pode envolver várias linhas (desde que atendam à condição) e o conteúdo pode ser uma expressão que envolva cálculo e não apenas um valor fixo.

c. Duplicando o valor do pedido para os pedidos que sejam menores de R\$ 1.000,00 (acrescente o comando SELECT para ver os resultados)

```
UPDATE xPEDIDO
SET ValorTotal = ValorTotal * 2
WHERE ValorTotal < 1000;
```

Resultado:

NumPedido	DataPedido	ValorTotal	CodFunc_FK
100	2014-01-10 00:00:00	1000.00	1
101	2014-01-15 00:00:00	1000.50	2
102	2014-01-20 00:00:00	1700.00	5
103	2014-01-30 00:00:00	5000.00	1
104	2014-02-05 00:00:00	500.00	3
105	2014-02-05 00:00:00	1314.50	1
106	2014-02-15 00:00:00	1292.00	2
107	2014-02-25 00:00:00	1830.50	5
108	2014-03-08 00:00:00	1956.00	4
109	2014-03-09 00:00:00	1250.00	5

Veja que podemos fazer cálculos com as próprias colunas da tabela, mesmo que a própria coluna faça parte da condição que filtra os dados.

Delete – eliminando dados de tabelas

O **delete** é, sem dúvida, um dos comandos menos utilizados na elaboração de programas. Até porque muitas vezes nós podemos apenas mudar uma informação ou estado do elemento, por exemplo, para cliente **inativo**. Isso é muito mais comum do que a exclusão pura e simples da informação. Mas há casos em que precisamos eliminar um ou mais

elementos de uma tabela. Para isso, utilizamos o comando delete.

A sintaxe do comando delete é:

Delete | Elimina dados já existentes de uma tabela.

Sintaxe:

```
DELETE      FROM      nome_tabela
              [WHERE      (condições)]
```

Os elementos do comando são assim definidos:

nome_tabela

É a tabela que terá seus dados excluídos.

condições

Fazem parte da cláusula *where*, que é opcional. São os critérios de filtro que selecionam os elementos da tabela que devem ser excluídos.

Vamos fazer alguns exemplos de exclusão de dados.

a. Excluindo uma linha: pedido número 105.

```
DELETE      FROM      xPEDIDO      WHERE
NumPedido = 105;

SELECT      *      FROM      xPEDIDO;
```

Resultado:

NumPedido	DataPedido	ValorTotal	CodFunc_FK
100	2014-01-10 00:00:00	1000.00	1
101	2014-01-15 00:00:00	1000.50	2
102	2014-01-20 00:00:00	1700.00	5
103	2014-01-30 00:00:00	5000.00	1
104	2014-02-05 00:00:00	500.00	3
106	2014-02-15 00:00:00	1292.00	2
107	2014-02-25 00:00:00	1830.50	5
108	2014-03-08 00:00:00	1956.00	4
109	2014-03-09 00:00:00	1250.00	5

Linha do pedido 105 não existe mais.

Excluindo várias linhas: pedido com número maior que 104.

```
DELETE      FROM      xPEDIDO      WHERE Num-
Pedido > 104;
```

Resultado:

NumPedido	DataPedido	ValorTotal	CodFunc_FK
100	2014-01-10 00:00:00	1000.00	1
101	2014-01-15 00:00:00	1000.50	2
102	2014-01-20 00:00:00	1700.00	5
103	2014-01-30 00:00:00	5000.00	1
104	2014-02-05 00:00:00	500.00	3

- a. Excluindo todas as linhas de uma tabela: basta fazer o *delete* sem a cláusula *where*.

```
DELETE      FROM      xPEDIDO;
```

Resultado após o comando SELECT:

MySQL não retornou nenhum registro.

Atenção! Muito cuidado ao usar este último comando sem o *where*, pois todos os dados serão eliminados da tabela, deixando-a vazia!



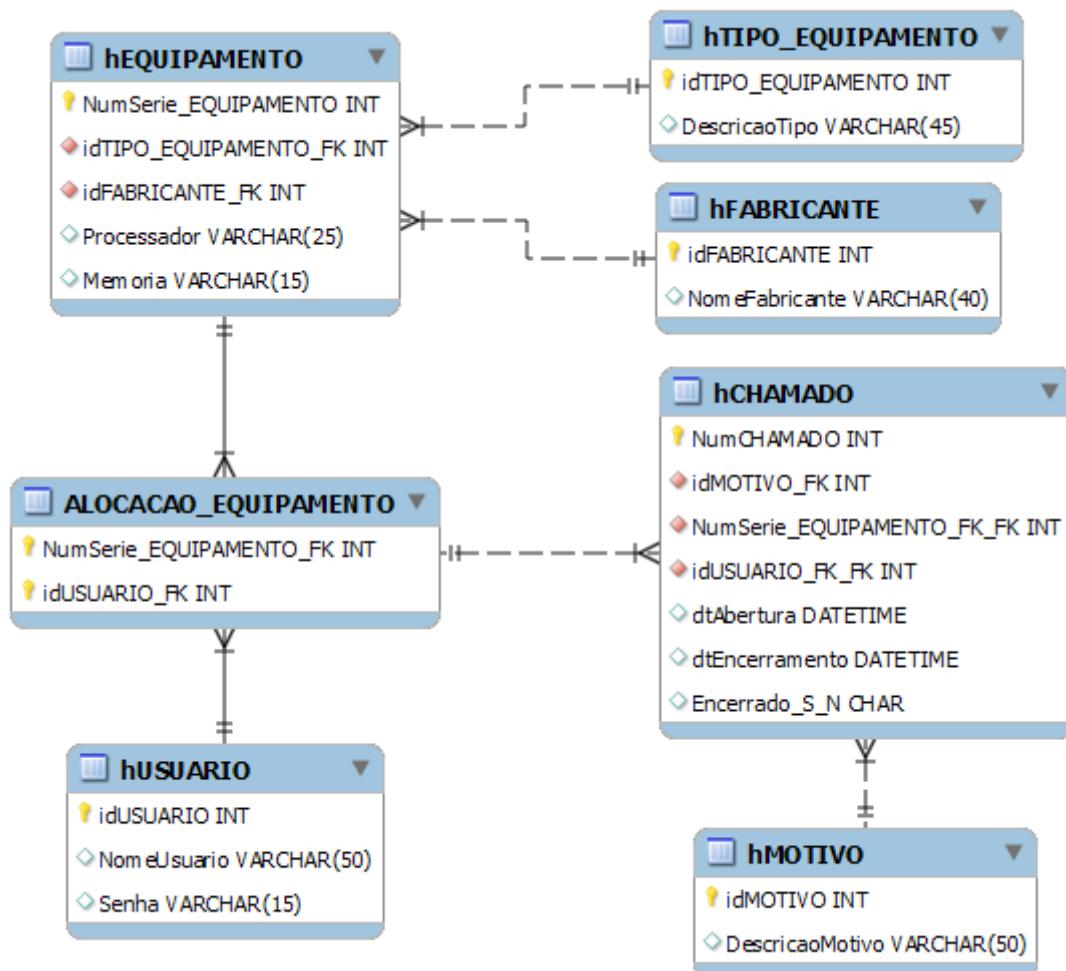
O comando *truncate* – limpando os dados de uma tabela

O *delete* sem a cláusula *where* elimina todas as linhas de uma tabela. O comando **TRUNCATE** faz o mesmo trabalho, mas de maneira mais rápida. Isso ocorre porque ao invés de apagar linha a linha da tabela, o *truncate* recria a tabela com sua estrutura e sem nenhuma linha. Sua sintaxe é simples e só tem uma forma de uso:

TRUNCATE | Limpa todos os dados de uma tabela.

Sintaxe: TRUNCATE nome_tabela

Modelo de dados de help desk (SIMPLIFICADO)



Exercícios do módulo 3

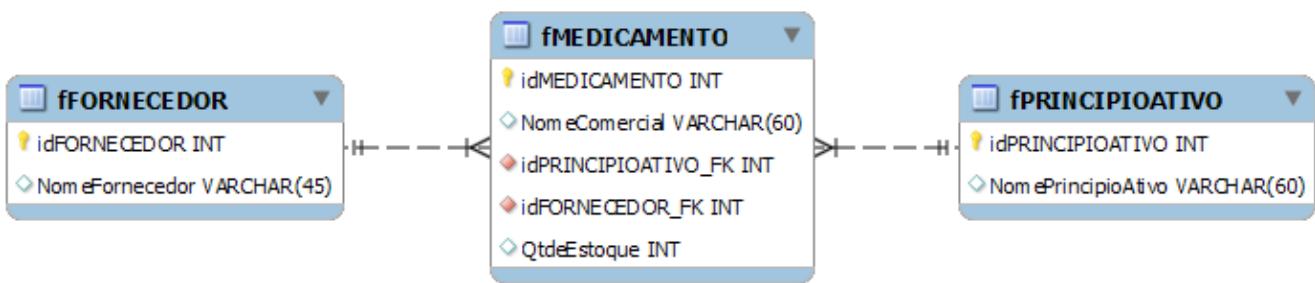
Para os exercícios deste módulo, crie um banco de dados para você praticar. Sugestão de nome para o banco de dados: "MODULO3".

1) Crie as tabelas do modelo abaixo com os comandos MySQL que estudamos.

Crie as chaves e os relacionamentos entre as tabelas como apresentado na figura.

Teste seus comandos no banco de dados MODULO3.

Teste alguns comandos de inclusão de dados nestas tabelas.



2) O modelo de dados a seguir implementa tabelas para controlar chamados de um help desk. A descrição resumida abaixo dá o contexto das informações que são armazenadas nestas tabelas:

O sistema de help desk registra os chamados abertos pelos usuários.

Na abertura do chamado, é utilizado um cadastro de equipamentos. Os equipamentos têm relacionamentos com as tabelas de fabricante e de tipo de equipamento (micro, impressora, scanner, webcam, ...). Cada equipamento é identificado por seu número de série

O sistema controla um cadastro de usuários que podem abrir chamados.

Outra informação armazenada são os equipamentos estão sob a responsabilidade de cada usuário (tabela ALOCACAO_EQUIPAMENTO).

Os chamados devem ser cadastrados com a Data de Solicitação (data de abertura do chamado), o número de série do equipamento e o usuário que está abrindo o chamado.

Quando o chamado é encerrado, é inserida a data de conclusão e é alterado o valor de uma flag que indica que o chamado foi encerrado.

Para o modelo de dados a seguir e considerando a descrição de negócios acima, crie as tabelas do Modelo abaixo com os comandos MySQL que estudamos.

Crie as chaves e os relacionamentos entre as tabelas como apresentado na figura.

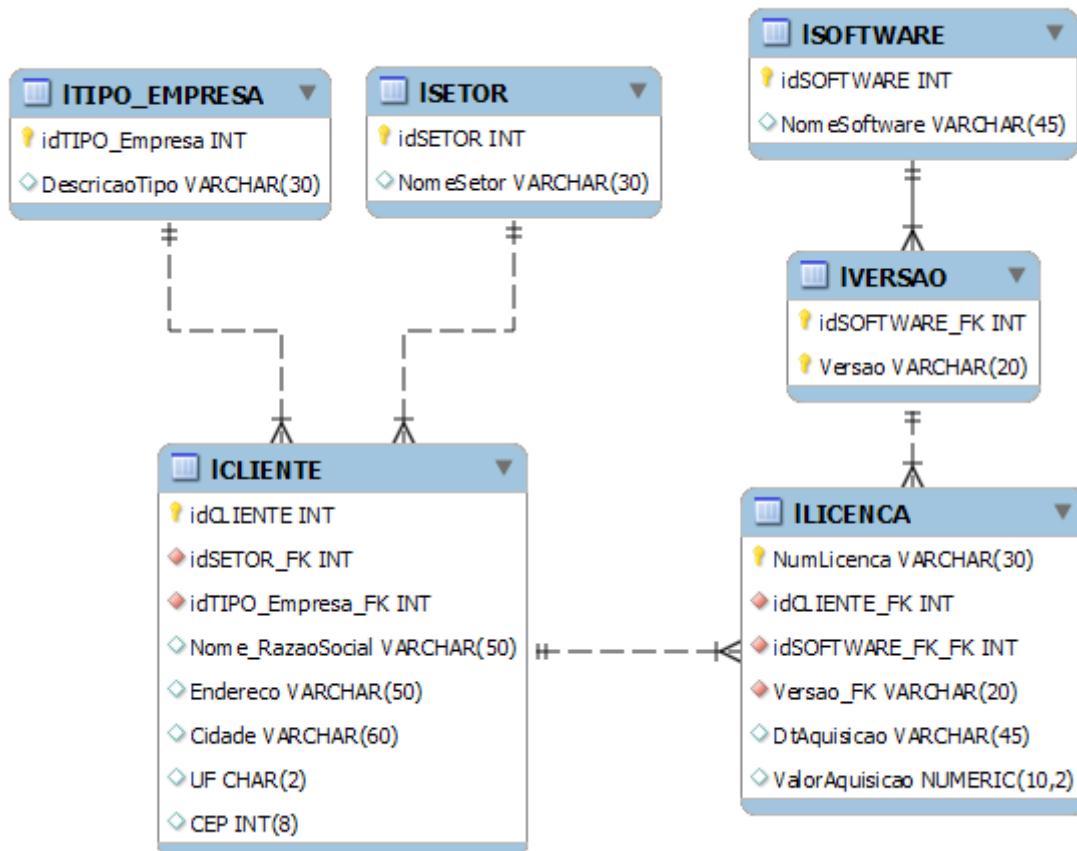
Teste seus comandos no banco de dados MODULO3.

Teste alguns comandos de inclusão de dados nestas tabelas.

3) O modelo de dados a seguir implementa tabelas para controlar licenças de software vendidas a empresas do mercado. Os softwares são registrados com seus nomes e suas versões.

As empresas são registradas com suas informações cadastrais e são classificadas por seu tipo de empresa e pelo setor da economia no qual atuam (há tabelas para ambas as classificações).

As licenças de software são adquiridas por empresas (clientes) e recebem a identificação do software e versão da licença, além de conter a data da aquisição da licença e o seu valor.



Para criar as Tabelas e demais objetos deste modelo e ainda carregar os dados para os exercícios, utilize os arquivos do "ANEXO – EXERCICIOS 3.ZIP".

Você encontrará:

- o desenho do modelo de dados;
- script para criação das tabelas do modelo;
- script para carga dos dados nas tabelas.

Após criar e carregar as tabelas de dados, responda aos exercícios a seguir. Teste seus comandos (respostas) no seu banco de dados.

Responda com os comandos MySQL para efetuar o que está sendo pedido em cada item:

- Liste os clientes exibindo nome, cidade de UF.
- Exiba as licenças de uso que contenham "A" no número de licença.
- exiba os clientes cujo nome comece com "P". Ordene por nome.
- Exiba os clientes cujo nome termina com "AR". Ordene por nome em ordem decrescente.

- e) Exiba os clientes que contém "W" ou "Y" no nome.
- f) Exiba as licenças de software com valor igual ou superior a R\$ 1.200,00. Ordene por valor de forma decrescente.
- g) Exiba os clientes cuja identificação (código) seja maior que 150 e menor que 200.
- h) Exiba as licenças de software com valor entre R\$ 250,00 e R\$ 500,00. Ordene em ordem crescente de valor de licença.
- i) Exiba as licenças que foram comercializadas após 2008 e cujo valor esteja entre R\$ 300 e R\$ 450 ou cujo valor esteja entre R\$ 600 e R\$ 800.
- j) Exiba os clientes que pertençam a uma das UFs: "SP", "RS", "PR", MG".
- k) Exiba os clientes que NÃO pertençam a nenhuma das UFs: "RJ", "ES", "SP", MG".
- l) Sua empresa virou cliente e vai comprar licenças de software. Inclua sua empresa na tabela de clientes e associe-a a um setor e a um tipo de empresa.
- m) Inclua um novo Software e versão nas tabelas apropriadas.
- n) Inclua 3 aquisições de licença para sua empresa. Uma delas deve ser do software/versão que você incluiu no item anterior.
- o) A tabela de preços subiu de última hora. Altere o valor da Licença que você inclui para o novo software incluído. Aumente o valor em 12,5%.
- p) Altere a descrição do tipo de empresa de código 6 para "Governo".
- q) As Licenças comercializadas terão que pagar imposto. Exiba o número da licença, a data de aquisição, o valor da licença, o imposto (10% sobre o Valor da Licença) e o Valor Líquido (Valor da Licença – Imposto). Ordene pela data de aquisição. Só as licenças comercializadas a partir de 2011 pagam impostos. Exiba apenas estas.

Módulo 4 – recursos do MySQL

A essa altura, caro(a) aluno(a), já aprendemos os comandos básicos e mais utilizados da linguagem SQL. O MySQL tem muitos recursos e neste capítulo vamos estudar alguns deles, além de aprofundar um pouco mais o conhecimento dos comandos que já conhecemos. Neste módulo também exercitaremos vários itens com a ferramenta MySQL. Então continue com sua animação e vamos aprender mais!

Aula 9 – Funções do MySQL

Além dos comandos, o MySQL disponibiliza ao desenvolvedor uma série de funções para processar informações. O conceito é o tradicional de funções: passamos parâmetros e temos um conteúdo de retorno. Estas funções tanto podem ser utilizadas com valores de parâmetros literais quanto sobre colunas de uma tabela ou expressões.

As funções podem ser utilizadas dentro de comandos DML, como em um *select*.

Vamos conhecer as principais funções do MySQL no que se segue.

Funções matemáticas

Algumas das principais funções matemáticas do MySQL são:

Função	Descrição	Exemplo de uso	Exemplo de resultado
ABS(x)	Retorna o valor absoluto de um número (sempre retorna positivo).	SELECT ABS(5); SELECT ABS (-10);	5 10
MOD(X,Y) ou %	Retorna o RESTO da divisão inteira de X por Y	SELECT (11, 3); ou SELECT 11 MOD 3; ou SELECT 11 % 3;	2 (A divisão inteira de 11 por 3 resulta 3 e o resto é 2)
ROUND (X, Y)	Arredonda o valor decimal X com a quantidade de casas decimais Y	SELECT ROUND(8.543); SELECT ROUND(8.543, 2); SELECT ROUND(8.543, 1); SELECT ROUND(8.123, 2); SELECT ROUND(-1.23);	9 8.54 8.5 8.12 -1
TRUNCATE (X, Y)	Trunca o valor decimal X com a quantidade de casas decimais Y	SELECT TRUNCATE(8.583, 2); SELECT TRUNCATE(8.583, 1); SELECT TRUNCATE(8.123, 0);	8.58 8.5 8
POW(X, Y) ou POWER (X,Y)	Retorna o valor da potência: X elevado a Y	SELECT POW(2,3); SELECT POWER (4, 4); SELECT POW(2.5, 3);	8 256 15.625
SQRT (x)	Retorna a raiz quadrada de X	SELECT SQRT (144); SELECT SQRT (81);	12 9

Função	Descrição	Exemplo de uso	Exemplo de resultado
LEAST (lista)	Retorna o menor dentre uma lista de valores (pode ser uma lista com numéricos ou com string).	SELECT LEAST(8, 5, -1); SELECT LEAST("Pedro","Abel","Luis");	-1 Abel
GREATEST (lista)	Retorna o maior valor dentre uma lista de valores (pode ser uma lista com numéricos ou com string).	SELECT GREATEST(8, 5, -1); SELECT GREATEST("Pedro","Abel","Luis");	8 Pedro

Funções para manipulação de *string*

Algumas das principais funções para manipulação de *strings* do MySQL são:

Função	Descrição	Exemplo de uso	Exemplo de resultado
CONCAT (S1, S2)	Concatena (une) duas <i>strings</i> , retornando uma única.	SELECT CONCAT ('Ja', 'va'); SELECT CONCAT ('Especia', 'lização');	Java Especialização
LENGTH (S)	Retorna o tamanho em bytes da string S	SELECT LENGTH ('Especialização');	16
LEFT (S, X)	Retorna X caracteres mais à esquerda da string S	SELECT LEFT ('Especialização', 5);	Espec
RIGHT (S, X)	Retorna X caracteres mais à direita da string S	SELECT RIGHT ('Especialização', 5);	zação
SUBSTRING (S,INI,[TAM])	Retorna uma parte de uma string S, a partir da posição inicialINI. Retornará a quantidade de caracteres TAM. Se TAM não for usado, retorna os caracteres restantes.	SELECT SUBSTRING('Especialização',3,6); SELECT SUBSTRING('Especialização',3);	pecial pecialização
MID	É o mesmo que SUBSTRING		
REPLACE (S, S1, S2)	Dentro da <i>string</i> S, substitui as ocorrências do string S1 por S2.	SELECT REPLACE('1.234.567,89', '.', ','); SELECT REPLACE('000012345678', '0', '');	1,234,567,89 12345678
LOCATE (SB, S)	Retorna a posição que a substring SB aparece dentro da string S	SELECT LOCATE('e', 'Banco de Dados');	8
LTRIM (S)	Retorna a string S sem espaços em branco à esquerda.	SELECT LTRIM (' Java ');	'Java '
RTRIM (S)	Retorna a string S sem espaços em branco à direita.	SELECT RTRIM (' Java ');	' Java'
TRIM (S)	Retorna a string S sem espaços em branco à direita e à esquerda.	SELECT TRIM (' Java ');	'Java'
UPPER (S)	Retorna a string S com seus caracteres maiúsculos.	SELECT UPPER('mysql');	MYSQL
LOWER (S)	Retorna a string S com seus caracteres minúsculos.	SELECT LOWER('MYSQL');	mysql

Funções de data e hora

Vamos estudar algumas das principais funções para manipulação de data e hora do MySQL. Caso você tenha excluído os dados da tabela xPEDIDO, repita o conjunto de comandos *insert* para repor os dados.

```
INSERT INTO xPEDIDO VALUES (100, '2014-01-10', 1000 , 1);

INSERT INTO xPEDIDO VALUES (101, '2014-01-15', 500.25, 2);

INSERT INTO xPEDIDO VALUES (102, '2014-01-20', 1700 , 5);

INSERT INTO xPEDIDO VALUES (103, '2014-01-30', 800 , 1);

INSERT INTO xPEDIDO VALUES (104, '2014-02-05', 250 , 3);

INSERT INTO xPEDIDO VALUES (105, '2014-02-05', 1314.5 ,
1);

INSERT INTO xPEDIDO VALUES (106, '2014-02-15', 1292 , 2);

INSERT INTO xPEDIDO VALUES (107, '2014-02-25', 915.25, 5);

INSERT INTO xPEDIDO VALUES (108, '2014-03-08', 978 , 4);

INSERT INTO xPEDIDO VALUES (109, '2014-03-09', 1250 , 5);
```

Vamos utilizar esses valores para testar algumas funções de data, hora e tempo.

Função	Descrição	Exemplo de uso	Exemplo de Resultado
YEAR (Data)	Retorna o ano de uma data.	SELECT YEAR ('2014-12-31');	2014
MONTH (data)	Retorna o mês de uma data.	SELECT MONTH('2014-12-31');	12
DAY (data)	Retorna o dia de uma data.	SELECT DAY ('2014-12-31');	31
DAYOFWEEK (data)	Retorna o dia da semana de uma data (1 = domingo, 2 = 2 ^a feira, 3 = 3 ^a feira, ...).	SELECT DAYOFWEEK ('2014-04/19');	7
HOUR (hora)	Retorna a hora de um horário.	SELECT HOUR ('17:43:15');	17
MINUTE (hora)	Retorna os minutos de um horário.	SELECT MINUTE ('17:43:15');	43
SECOND (hora)	Retorna os segundos de um horário.	SELECT SECOND ('17:43:15');	15
NOW()	Retorna a data atual do sistema, com horário. (Teste agora!)	SELECT NOW();	2014-04-19 17:47:17
CURDATE()	Retorna a data atual do sistema. (Teste agora!)	SELECT CURDATE();	2014-04-19

Função	Descrição	Exemplo de uso	Exemplo de Resultado
CURTIME()	Retorna a hora atual do sistema. (Teste agora!)	SELECT CURTIME();	17:47:17
DATEDIFF (data1, data2)	Retorna a diferença em dias entre DATA1 e DATA2.	SELECT DATEDIFF ('2014-04-19', '2014-01-28');	81

Vamos executar este comando para exemplificar o uso do que aprendemos sobre a tabela xPEDIDO que contém uma coluna com datas:

```
SELECT      NumPedido, DataPedido,
            YEAR (DataPedido)  as ANO,
            MONTH(DataPedido)  as MES,
            DAY  (DataPedido)  as DIA
        FROM          xPEDIDO;
```

Resultado:

NumPedido	DataPedido	ANO	MES	DIA
100	2014-01-10 00:00:00	2014	1	10
101	2014-01-15 00:00:00	2014	1	15
102	2014-01-20 00:00:00	2014	1	20
103	2014-01-30 00:00:00	2014	1	30
104	2014-02-05 00:00:00	2014	2	5
105	2014-02-05 00:00:00	2014	2	5
106	2014-02-15 00:00:00	2014	2	15
107	2014-02-25 00:00:00	2014	2	25
108	2014-03-08 00:00:00	2014	3	8
109	2014-03-09 00:00:00	2014	3	9

Estas funções permitem efetuarmos seleções ou manipulações que usem como filtro parte da data. Por exemplo, o comando a seguir lista quais foram os pedidos de janeiro de 2014:

```
SELECT      *
        FROM          xPEDIDO
        WHERE      YEAR (DataPedido) =      2014
                    AND MONTH(DataPedido) =      1;
```

Resultado:

NumPedido	DataPedido	ValorTotal	CodFunc_FK
100	2014-01-10 00:00:00	1000.00	1
101	2014-01-15 00:00:00	500.25	2
102	2014-01-20 00:00:00	1700.00	5
103	2014-01-30 00:00:00	800.00	1

Retorna apenas os pedidos de Janeiro/2014.

Vamos ver um exemplo de diferença de datas usando a tabela xPEDIDO. O resultado é retornado em dias. Vamos ao comando:

```
SELECT      NumPedido, NOW(), DataPedi-
do,
            DATEDIFF (NOW(), DataPedi-
do) as QtdeDias
FROM        xPEDIDO;
```

Resultado:

NumPedido	NOW()	DataPedido	QtdeDias
100	2014-04-19 18:28:40	2014-01-10 00:00:00	99
101	2014-04-19 18:28:40	2014-01-15 00:00:00	94
102	2014-04-19 18:28:40	2014-01-20 00:00:00	89
103	2014-04-19 18:28:40	2014-01-30 00:00:00	79
104	2014-04-19 18:28:40	2014-02-05 00:00:00	73
105	2014-04-19 18:28:40	2014-02-05 00:00:00	73
106	2014-04-19 18:28:40	2014-02-15 00:00:00	63
107	2014-04-19 18:28:40	2014-02-25 00:00:00	53
108	2014-04-19 18:28:40	2014-03-08 00:00:00	42
109	2014-04-19 18:28:40	2014-03-09 00:00:00	41

Funções de conversão

São funções que transformam conteúdos de um determinado tipo em outro tipo. Os tipos (DATATYPE) mais usados para conversão são:

- CHAR
- DATE
- DATETIME
- TIME

Função	Descrição
CAST (EXPR AS DATATYPE)	Retorna o conteúdo da expressão / coluna / valor EXPR convertida para o tipo determinado DATATYPE.
CONVERT (EXPR, DATATYPE)	

Exemplos de uso de funções de conversão:

Exemplos	Resultados
SELECT CAST('2014-04-19' AS DATE);	2014-04-19
SELECT CAST(1548.16 AS CHAR);	1548.16
SELECT CAST(NOW() AS DATE);	2014-04-20
SELECT CAST(NOW() AS TIME);	11:52:54
SELECT CAST(NOW() AS DATETIME);	2014-04-20 11:52:54
SELECT CAST('2014-04-19' AS DATETIME);	2014-04-19 00:00:00

Mais um exemplo de uso de função de conversão de tipos de dados:

```
SELECT      NumPedido,
            CAST(ValorTotal AS CHAR) as
VlrChar
FROM        xPEDIDO;
```

Resultado:

NumPedido	VlrChar
100	1000.00
101	500.25
102	1700.00
103	800.00
104	250.00
105	1314.50
106	1292.00
107	915.25
108	978.00
109	1250.00

Funções estatísticas

Chegou a vez de falarmos sobre as funções estatísticas. Elas retornam valores estatísticos e são especialmente usadas sobre tabelas de dados. Neste tópico, vamos ver sua forma mais básica, no entanto mais à frente veremos estas funções apresentadas em conjunto com a cláusula GROUP BY do SELECT, que as tornam mais completas e nos fornecem recursos interessantes.

Função	Descrição	Exemplo de uso	Exemplo de resultado
COUNT(*)	Retorna a quantidade de linhas de uma consulta.	SELECT COUNT(*) FROM xPEDIDO;	10
SUM (coluna)	Retorna a soma dos valores da coluna da tabela.	SELECT SUM(ValorTotal) FROM xPEDIDO;	10000.00
AVG (coluna)	Retorna a média dos valores da coluna da tabela.	SELECT AVG(ValorTotal) FROM xPEDIDO;	1000.00
MAX (coluna)	Retorna o maior valor da coluna da tabela.	SELECT MAX(ValorTotal) FROM xPEDIDO;	1700.00
MIN (coluna)	Retorna o menor valor da coluna da tabela.	SELECT MIN(ValorTotal) FROM xPEDIDO;	250.00
STD (coluna)	Retorna o desvio padrão dos valores da coluna da tabela.	SELECT STD(ValorTotal) FROM xPEDIDO;	399.5736321380579

Os exemplos foram apresentados com *select simples*, mas podem usar filtros *where*. Exemplo:

```
SELECT      AVG (ValorTotal)      FROM      xPEDIDO
WHERE      MONTH (DataPedido) =      2;
```

Resultado: **AVG (ValorTotal)** Retorna a média apenas dos valores das linhas que atendem ao filtro.
942.937500

Além disso, podem ser usados em conjunto em um mesmo *select*, desde que o filtro seja o mesmo, veja:

```
SELECT      COUNT (*)      as      Qtde,
           SUM(ValorTotal)      as      Soma,
           AVG(ValorTotal)      as      Media,
           MAX(ValorTotal)      as      Maior,
           MIN(ValorTotal)      as      Menor
FROM      xPEDIDO;
```

Resultado:

Qtde	Soma	Media	Maior	Menor
10	10000.00	1000.000000	1700.00	250.00

Aula 10 – Exercitando *join* e *group by* em um novo DB

Caro(a) aluno(a), vamos criar mais um *database*, agora com mais tabelas e mais dados, a fim de exercitarmos melhor alguns conceitos que já estudamos.

Carregando uma massa de dados para um novo banco de dados

Siga os passos para obter os arquivos necessários, criar um novo DB com tabelas novas e carregar dados em nessas tabelas. O novo *database* será chamado APLICACAO. Siga as etapas:

1. No MySQL, crie um *database* novo chamado APLICACAO.
2. Utilize os arquivos do anexo: **Anexo Modelo APLICACAO.zip**
3. Leia o contexto e observe o modelo de dados do arquivo: **Modelo APLICACAO - Modelo de Dados e Contexto.pdf**
4. Crie as tabelas dentro do *database* APLICACAO. Use para isso o arquivo de comandos DDL: **Modelo APLICACAO - Criacao de Tabelas.sql**
5. Carregue os dados nas tabelas a partir do arquivo de comandos INSERTs: **Modelo APLICACAO - Carga de Dados.sql**
6. Faça comandos SELECTs sobre as tabelas para você conhecer os dados que carregamos nas tabelas.

Join com várias tabelas em um comando SELECT

Vamos exercitar a consulta de dados de várias tabelas deste novo BD APLICACAO. Para tanto, vamos realizar exemplos listados a seguir.

a. Listando as aplicações sem relacionamentos:

```
SELECT * FROM aAPLICACAO;
```

Resultado: (5 primeiras linhas)

NumAplicacao	CodCli	NumFundo	NumConsultor	DtAplicacao	ValorAplicacao
1	25	2600	11	2007-04-01 00:00:00	15000.00
2	70	100	13	2007-04-15 00:00:00	48000.00
3	111	2000	14	2007-04-20 00:00:00	200000.00
4	211	1400	13	2007-04-29 00:00:00	35000.00
5	70	200	11	2007-05-10 00:00:00	10000.00

b. Apresentando as aplicações agora com o nome do cliente no lugar de seu código:

```

SELECT      NumAplicacao as NumAplic, NomeCli, NumFundo, NumConsultor,
            DATE(DtAplicacao) as DataAplic, ValorAplicacao
FROM        aAPLICACAO A
JOIN        aCLIENTE     C      ON      A.CodCli      =      C.CodCli
ORDER BY    NumAplicacao;

```

Resultado (10 primeiras linhas):

NumAplic	NomeCli	NumFundo	NumConsultor	DataAplic	ValorAplicacao
1	LUCIO HENRIQUE PERES	2600	11	2007-04-01	15000.00
2	LUCAS MORAES DA SILVA	100	13	2007-04-15	48000.00
3	ANDRE TAVARES DE OLIVEIRA	2000	14	2007-04-20	200000.00
4	ANA CRISTINA DE PAIVA	1400	13	2007-04-29	35000.00
5	LUCAS MORAES DA SILVA	200	11	2007-05-10	10000.00
6	ODIMAR CLAUDIO DA SIVA DE CARVALHO	1400	13	2007-05-12	29000.00
7	Paulo Jose da Silva	2000	12	2007-05-15	350000.00
8	TARCISIO HENRIQUE DA SILVA PAULA	1800	11	2007-05-28	28300.00
9	SERGIO VIANA JUNIOR	1800	11	2007-06-06	13100.00
10	GILBERTO NEPOMUCENO	1800	11	2007-06-18	31600.00

c. Apresentando os mesmos dados, mas agora substituindo o código do consultor financeiro por seu nome.

(Isto exigirá que incluamos mais uma tabela no select e mais um join para exercer o relacionamento entre essas tabelas)

```

SELECT      NumAplicacao as NumAplic, NomeCli, NumFundo, NomeConsul-
tor,
            DATE(DtAplicacao) as DataAplic, ValorAplicacao
FROM        aAPLICACAO          A
JOIN        aCLIENTE           C      ON      A.CodCli      =      C.CodCli
JOIN        aCONSULTORINVESTIMENTO CO
            ON      A.NumConsultor      =      CO.NumConsultor
ORDER BY    NumAplicacao;

```

Resultado (10 primeiras linhas):

NumAplic	NomeCli	NumFundo	NomeConsultor	DataAplic	ValorAplicacao
1	LUCIO HENRIQUE PERES	2600	Marie Anne Goffaux	2007-04-01	15000.00
2	LUCAS MORAES DA SILVA	100	Edson Martins Dantas	2007-04-15	48000.00
3	ANDRE TAVARES DE OLIVEIRA	2000	Pedro Sato Junior	2007-04-20	200000.00
4	ANA CRISTINA DE PAIVA	1400	Edson Martins Dantas	2007-04-29	35000.00
5	LUCAS MORAES DA SILVA	200	Marie Anne Goffaux	2007-05-10	10000.00
6	ODIMAR CLAUDIO DA SIVA DE CARVALHO	1400	Edson Martins Dantas	2007-05-12	29000.00
7	Paulo Jose da Silva	2000	Marina de Oliveira	2007-05-15	350000.00
8	TARCISIO HENRIQUE DA SILVA PAULA	1800	Marie Anne Goffaux	2007-05-28	28300.00
9	SERGIO VIANA JUNIOR	1800	Marie Anne Goffaux	2007-06-06	13100.00
10	GILBERTO NEPOMUCENO	1800	Marie Anne Goffaux	2007-06-18	31600.00

Substituindo o Número do Fundo por seu nome.(Novamente teremos mais um *join* para exercer o relacionamento entre essas tabelas)

```

SELECT      NumAplicacao as Num, NomeCli, NomeComlFundo as NomeFundo,
            NomeConsultor, DATE(DtAplicacao) as DataAplic,
            ValorAplicacao

FROM        aAPLICACAO          A

JOIN        aCLIENTE           C    ON      A.CodCli      =      C.CodCli
JOIN        aCONSULTORINVESTIMENTO CO
            ON      A.NumConsultor      =      CO.NumConsultor
JOIN        aFUNDOINVESTIMENTO F
            ON      A.NumFundo          =      F.NumFundo

ORDER BY    NumAplicacao;
  
```

Resultado (10 primeiras linhas):

Num	NomeCli	NomeFundo	NomeConsultor	DataAplic	ValorAplicacao
1	LUCIO HENRIQUE PERES	FIA Petrobras	Marie Anne Goffaux	2007-04-01	15000.00
2	LUCAS MORAES DA SILVA	BETA FIC Curto Prazo	Edson Martins Dantas	2007-04-15	48000.00
3	ANDRE TAVARES DE OLIVEIRA	BETA FI Multimercado Principal Protegido	Pedro Sato Junior	2007-04-20	200000.00
4	ANA CRISTINA DE PAIVA	BETA FIC Renda Fixa Saturno	Edson Martins Dantas	2007-04-29	35000.00
5	LUCAS MORAES DA SILVA	BETA FIC Referenciado DI HiperFundo	Marie Anne Goffaux	2007-05-10	10000.00
6	ODIMAR CLAUDIO DA SIVA DE CARVALHO	BETA FIC Renda Fixa Saturno	Edson Martins Dantas	2007-05-12	29000.00
7	Paulo Jose da Silva	BETA FI Multimercado Principal Protegido	Marina de Oliveira	2007-05-15	350000.00
8	TARCISIO HENRIQUE DA SILVA PAULA	BETA FIC Multimercado Iden Profit Moderado	Marie Anne Goffaux	2007-05-28	28300.00
9	SERGIO VIANA JUNIOR	BETA FIC Multimercado Iden Profit Moderado	Marie Anne Goffaux	2007-06-06	13100.00
10	GILBERTO NEPOMUCENO	BETA FIC Multimercado Iden Profit Moderado	Marie Anne Goffaux	2007-06-18	31600.00

d. Incluindo a descrição do tipo de fundo que se está aplicando.

(Vamos incluir a tabela aTIPOFUNDO, desta vez ligada à tabela aFUNDOINVESTIMENTO)

```

SELECT      NumAplicacao as Num, NomeCli, NomeComlFundoo as NomeFundoo,
            DescrTipoFundoo,     NomeConsultor,
            DATE(DtAplicacao) as DataAplic, ValorAplicacao

        FROM      aAPLICACAO          A

        JOIN      aCLIENTE           C      ON      A.CodCli      =      C.CodCli
        JOIN      aCONSULTORINVESTIMENTO CO
            ON      A.NumConsultor      =      CO.NumConsultor
        JOIN      aFUNDOINVESTIMENTO F
            ON      A.NumFundoo        =      F.NumFundoo
        JOIN      aTIPOFUNDOD          T
            ON      F.CodTipoFundoo    =      T.CodTipoFundoo

        ORDER BY  NumAplicacao;
    
```

Resultado (10 primeiras linhas):

Num	NomeCli	NomeFundoo	DescrTipoFundoo	NomeConsultor	DataAplic	ValorAplicacao
1	LUCIO HENRIQUE PERES	FIA Petrobras	Ações Outros	Marie Anne Goffaux	2007-04-01	15000.00
2	LUCAS MORAES DA SILVA	BETA FIC Curto Prazo	Curto Prazo	Edson Martins Dantas	2007-04-15	48000.00
3	ANDRE TAVARES DE OLIVEIRA	BETA FI Multimercado Principal Protegido	Capital Protegido	Pedro Sato Junior	2007-04-20	200000.00
4	ANA CRISTINA DE PAIVA	BETA FIC Renda Fixa Saturno	Renda Fixa	Edson Martins Dantas	2007-04-29	35000.00
5	LUCAS MORAES DA SILVA	BETA FIC Referenciado DI HiperFundo	Referenciado DI	Marie Anne Goffaux	2007-05-10	10000.00
6	ODIMAR CLAUDIO DA SIVA DE CARVALHO	BETA FIC Renda Fixa Saturno	Renda Fixa	Edson Martins Dantas	2007-05-12	29000.00
7	Paulo Jose da Silva	BETA FI Multimercado Principal Protegido	Capital Protegido	Marina de Oliveira	2007-05-15	350000.00
8	TARCISIO HENRIQUE DA SILVA PAULA	BETA FIC Multimercado Iden Profit Moderado	Multimercado	Marie Anne Goffaux	2007-05-28	28300.00
9	SERGIO VIANA JUNIOR	BETA FIC Multimercado Iden Profit Moderado	Multimercado	Marie Anne Goffaux	2007-06-06	13100.00
10	GILBERTO NEPOMUCENO	BETA FIC Multimercado Iden Profit Moderado	Multimercado	Marie Anne Goffaux	2007-06-18	31600.00

Observe, caro(a) aluno(a) que neste resultado apresentamos dados de 5 tabelas:

- a 1ª e as duas últimas colunas são da tabela aAPLICACAO;
- a 2ª da tabela aCLIENTE;
- a 3ª da tabela aFUNDOINVESTIMENTO;
- a 4ª da tabela aTIPOFUNDOD;
- a 5ª da tabela aCONSULTORINVESTIMENTO.

É comum, ao consultarmos dados de um Database, buscarmos dados em várias tabelas. Para tanto, a cláusula *join* é usada para agregar ao resultado seus dados através dos relacionamentos das tabelas.

Observe que a cada *join*, sempre são indicadas PK e FK na cláusula *on*. A cláusula *on* do *join* indica quais colunas o MySQL deve utilizar para relacionar as tabelas. Mesmo que exista uma FK criada para relacionar as tabelas, é necessário especificar a PK de uma tabela e igualá-la à FK de outra tabela.

No item *group by* faremos algumas consultas agrupando dados. Usaremos a mesma lista de tabelas declaradas em *from / join* no quadro de comandos do item “e” que apresentamos há pouco.

Copiamos parte destes comandos para o quadro 10.2 e destacamos em azul o trecho que pretendemos repetir quando usarmos algumas das consultas com *group by*. Podemos alterar os dados que queremos que sejam apresentados alterando os itens imediatamente após o *select*.

Quadro 10.2 – Tabelas e relacionamentos que usaremos repetidas vezes

```

SELECT      (colunas / expressões)

FROM        aAPLICACAO          A

JOIN        aCLIENTE            C    ON      A.CodCli      =      C.CodCli

JOIN        aCONSULTORINVESTIMENTO CO

                      ON      A.NumConsultor      =      CO.NumConsultor

JOIN        aFUNDODINVESTIMENTO F

                      ON      A.NumFundo      =      F.NumFundo

JOIN        aTIPOFUNDO          T

                      ON      F.CodTipoFundo      =      T.CodTipoFundo

ORDER BY    NumAplicacao;
  
```

GROUP BY – cláusula de agrupamento em um SELECT

A cláusula *group by* do *select* permite agruparmos resultados de uma consulta em tabela(s) de dados. É especialmente útil quando a associamos com as funções estatísticas que estudamos anteriormente.

Já apresentamos estatísticas utilizando estas funções sobre a tabela xPEDIDO. Vamos repetir esta pesquisa, agora sobre a tabela aAPLICACAO:

```
SELECT COUNT(*) as Qtde,
       SUM(ValorAplicacao) as Soma,
       AVG(ValorAplicacao) as Media,
       MAX(ValorAplicacao) as Maior,
       MIN(ValorAplicacao) as Menor
  FROM aAPLICACAO;
```

Resultado:

Qtde	Soma	Media	Maior	Menor
120	3633000.00	30275.000000	350000.00	1800.00

Vamos refazer esta consulta e utilizar outras, agora sempre utilizando o trecho destacado em azul no quadro 10.2 que indica a lista de tabelas envolvidas e efetua as junções (JOINS) entre estas tabelas.

```

SELECT COUNT(*) as Qtde,
       SUM(ValorAplicacao) as Soma,
       AVG(ValorAplicacao) as Media,
       MAX(ValorAplicacao) as Maior,
       MIN(ValorAplicacao) as Menor
FROM aAPLICACAO A
JOIN aCLIENTE C ON A.CodCli = C.CodCli
JOIN aCONSULTORINVESTIMENTO CO
ON A.NumConsultor = CO.NumConsultor
JOIN aFUNDOINVESTIMENTO F
ON A.NumFundo = F.NumFundo
JOIN aTIPOFUNDO T
ON F.CodTipoFundo = T.CodTipoFundo
ORDER BY NumAplicacao;

```

Resultado:

Qtde	Soma	Media	Maior	Menor
120	3633000.00	30275.000000	350000.00	1800.00

O resultado é exatamente o mesmo, pois as contas foram feitas sobre a coluna ValorAplicacao que já existia na consulta simples feita só com a tabela aAPLICACAO.

Mas a inclusão de outras tabelas nos dá algumas facilidades para gerar informação, classificando nossa informação a partir dos dados de outras tabelas. Vejamos os exemplos a seguir.

a. Consultando o total de aplicações, mas agora para cada um dos Consultores de Investimento.

(Queremos saber o total de valor aplicado efetuado por cada um dos consultores de investimento)

```

SELECT      NomeConsultor,  SUM(ValorAplicacao)  TotalAplic

FROM        aAPLICACAO          A

JOIN        aCLIENTE            C    ON      A.CodCli      =      C.CodCli

JOIN        aCONSULTORINVESTIMENTO CO
ON          A.NumConsultor     =      CO.NumConsultor

JOIN        aFUNDODINVESTIMENTO F
ON          A.NumFundo         =      F.NumFundo

JOIN        aTIPOFUNDO          T
ON          F.CodTipoFundo     =      T.CodTipoFundo

GROUP BY  NomeConsultor;

```

Resultado:

NomeConsultor	TotalAplic
Edson Martins Dantas	566400.00
Fatima da Silva	605500.00
Marie Anne Goffaux	727100.00
Marina de Oliveira	933100.00
Pedro Sato Junior	800900.00

Observe que o total geral está distribuído em 5 subtotais, um para cada consultor de investimento.

Note que com a cláusula GROUP BY temos algumas regras a seguir:

- no GROUP BY devemos especificar qual coluna desejamos que o MySQL utilize como “quebra” da informação em subtotais;
- imediatamente após escrevermos o SELECT, devem ser utilizadas apenas funções estatísticas e as colunas que estão declaradas no GROUP BY;
- ou seja, as colunas que individualizam a informação, como NumAplicacao, não podem ser utilizadas, pois impediriam o agrupamento da informação.

b. Podemos apresentar estatísticas também por clientes. Vamos incrementar a consulta pedindo, além do total de valor, a quantidade de aplicações de cada cliente e o valor médio dessas aplicações:

```

SELECT      NomeCli, SUM(ValorAplicacao) TotalAplic,
            COUNT(*) as QtdeAplic, AVG(ValorAplicacao) MediaAplic
FROM        aAPLICACAO          A
JOIN        aCLIENTE           C   ON      A.CodCli      =      C.CodCli
JOIN        aCONSULTORINVESTIMENTO CO
ON          A.NumConsultor    =      CO.NumConsultor
JOIN        aFUNDODINVESTIMENTO F
ON          A.NumFundo        =      F.NumFundo
JOIN        aTIPOFUNDO         T
ON          F.CodTipoFundo    =      T.CodTipoFundo
GROUP BY    NomeCli;

```

Resultado:

NomeCli	TotalAplic	QtdeAplic	MediaAplic
ALLAN PEREIRA DOS SANTOS	20400.00	1	20400.000000
ALMIR ANTONIO DE SOUZA	22800.00	1	22800.000000
ANA CRISTINA DE PAIVA	35000.00	1	35000.000000
ANDRE TAVARES DE OLIVEIRA	312700.00	5	62540.000000
CARLOS TAVARES DE FIGUEIREDO	97000.00	3	32333.333333
CICERO SILVA FRANCO	16800.00	1	16800.000000
CLAUDIO FRANCISCO MENDES	46700.00	1	46700.000000
CLEA CRISTINA CALIMAN	105700.00	3	35233.333333
DALILA LIMA DOS SANTOS	13000.00	1	13000.000000
DENIS DE LIMA GONÇALVES	132800.00	3	44266.666667
DIEGO GONCALVES DE OLIVEIRA	228700.00	12	19058.333333
FABIO BITTENCOURT VIDIGAL LEITAO	46800.00	1	46800.000000
FAGNER SILVA DOS SANTOS - Canc	23200.00	1	23200.000000
FLAVIO MACEDO DE GOIS	119500.00	5	23900.000000
GERCINO DE BRITO	85900.00	3	28633.333333

c. Podemos ainda utilizar mais de uma coluna como critério de agrupamento da informação.
Vamos agrupar a informação em 2 níveis: primeiro por TIPO de FUNDO de depois por NOMEFUNDO. Para isto, utilizamos duas colunas em GROUP BY como critério de agrupamento:

```

SELECT      T.CodTipoFundo as CodTipo, DescrTipoFundo as TipoFundo,
            F.NumFundo, NomeComlFundo as NomeFundo,
            SUM(ValorAplicacao) TotalAplic, COUNT(*) as QtdeAplic,
            AVG(ValorAplicacao) MediaAplic

FROM        aAPLICACAO          A
            JOIN     aCLIENTE           C   ON     A.CodCli      =      C.CodCli
            JOIN     aCONSULTORINVESTIMENTO CO
            ON       A.NumConsultor    =      CO.NumConsultor
            JOIN     aFUNDOINVESTIMENTO F
            ON       A.NumFundo        =      F.NumFundo
            JOIN     aTIPOFUNDO         T
            ON       F.CodTipoFundo    =      T.CodTipoFundo
GROUP BY   DescrTipoFundo, NomeComlFundo
ORDER BY   T.CodTipoFundo, F.NumFundo;

```

Resultado:

CodTipo	TipoFundo	NumFundo	NomeFundo	TotalAplic	QtdeAplic	MediaAplic
1	Curto Prazo	100	BETA FIC Curto Prazo	114900.00	5	22980.000000
2	Referenciado DI	200	BETA FIC Referenciado DI HiperFundo	119600.00	5	23920.000000
2	Referenciado DI	300	BETA FIC Referenciado DI Safira	72700.00	2	36350.000000
2	Referenciado DI	400	BETA FIC Referenciado DI Nikkei	199000.00	7	28428.571429
2	Referenciado DI	500	BETA FIC Referenciado DI Brilhante	76800.00	3	25600.000000
2	Referenciado DI	600	BETA FIC Referenciado DI Topázio	48500.00	3	16166.666667
2	Referenciado DI	700	BETA FIC Referenciado DI Platinum	79600.00	3	26533.333333
2	Referenciado DI	800	BETA FIC Referenciado DI Federal	41700.00	3	13900.000000
3	Cambial Indexado Dólar	1000	BETA FIC Cambial Dólar Special	69300.00	2	34650.000000
4	Renda Fixa	1100	BETA FIC Renda Fixa Vénus	153600.00	7	21942.857143
4	Renda Fixa	1200	BETA FIC Renda Fixa FIC Mais	30800.00	1	30800.000000
4	Renda Fixa	1300	BETA FIC Renda Fixa Mercúrio	81100.00	3	27033.333333
4	Renda Fixa	1400	BETA FIC Renda Fixa Saturno	132100.00	4	33025.000000
4	Renda Fixa	1500	BETA FIC Renda Fixa Marte	123400.00	5	24680.000000
5	Renda Fixa Multi-Índices	1600	BETA FIC Renda Fixa Multi-Índices	8000.00	2	4000.000000
6	Multimercado	1700	BETA FIC Multimercado ;lden Profit Conservador	35700.00	1	35700.000000
6	Multimercado	1800	BETA FIC Multimercado ;lden Profit Moderado	94700.00	4	23675.000000
6	Multimercado	1900	BETA FIC Multimercado ;lden Profit Dinâmico	67500.00	5	13500.000000
7	Capital Protegido	2000	BETA FI Multimercado Principal Protegido	707700.00	7	101100.000000
7	Capital Protegido	2100	BETA FI Multimercado Principal Protegido 1	135200.00	5	27040.000000
8	Ações IBOVESPA Indexado	2200	BETA FIC de FIA Ibovespa Indexado	202700.00	6	33783.333333

O resultado está agrupado em dois níveis: primeiro por tipo de fundo (incluímos os códigos a fim de facilitar a visualização) e depois por nome do fundo.

Os valores apresentados estão agrupados pelo segundo nível (nome fundo).

Há um detalhe a ressaltar neste comando SELECT: antes das colunas CodTipoFundo e NumFundos utilizamos o ALIAS de uma tabela (T.CodTipoFundo e F.NumFundos).

Isto é necessário porque quando fazemos JOIN de duas ou mais tabelas, todos as colunas das tabelas envolvidas ficam disponíveis, embora só sejam apresentadas as colunas que indicamos no SELECT.

Quando fazemos um SELECT JOIN simples, se utilizamos * no SELECT, todas as colunas são apresentadas. No caso do CodTIpoFundo, haveria duas colunas, com o mesmo nome, disponíveis no SELECT.

Isto ocorre porque uma coluna veio da tabela aFUNDOINVESTIMENTO e outra da tabela aTIPOFUNDO.

O mesmo ocorre com a coluna NumFundos.

Deste modo, é necessário indicar para o MySQL de que tabela queremos que seja apresentada a coluna (embora na prática dê no mesmo, uma vez que as igualamos no ON).

Caso não utilizemos a identificação da tabela ou ao menos seu ALIAS, o MySQL retornará um erro de ambiguidade, pois não saberá qual das colunas (de mesmo nome) deverá apresentar no resultado.



Usando a cláusula HAVING em um comando SELECT

A subcláusula *having* funciona de forma muito similar à cláusula *where*. Mas as condições que estipulamos nesta subcláusula valem para o conjunto de dados gerado pelo SELECT/GROUP BY.

Na figura 10.4, veja que o comando *select*, em sua parte inicial, extrai um conjunto de dados de tabelas a partir de filtros (declarados no WHERE). Este resultado fica disponível em memória e serão apresentados ao usuário os itens solicitados imediatamente após a palavra SELECT.

Ao usar o HAVING, este resultado em memória é filtrado novamente a partir de novas condições que não são aplicadas quando na consulta às tabelas, mas são aplicadas sobre o resultado parcial produzido e que está na memória.



Figura 10.4 – Resultado de Pesquisa antes e após HAVING

Vamos repetir a consulta da letra c), efetuada no item anterior, mas agora filtrando apenas os fundos que tiveram mais de 5 aplicações. Esta consulta é feita usando-se os resultados de COUNT / GROUP BY usados no SELECT:

```

SELECT      T.CodTipoFundo as CodTipo, DescrTipoFundo as TipoFundo,
            F.NumFundos, NomeComlFundos as NomeFundos,
            SUM(ValorAplicacao) TotalAplic, COUNT(*) as QtdeAplic,
            AVG(ValorAplicacao) MediaAplic
FROM        aAPLICACAO          A
JOIN        aCLIENTE           C      ON      A.CodCli      =      C.CodCli
JOIN        aCONSULTORINVESTIMENTO CO
          ON      A.NumConsultor      =      CO.NumConsultor
JOIN        aFUNDOINVESTIMENTO   F
          ON      A.NumFundos       =      F.NumFundos
JOIN        aTIPOFUNDO          T
          ON      F.CodTipoFundo     =      T.CodTipoFundo
GROUP BY    DescrTipoFundo, NomeComlFundos
HAVING      QtdeAplic > 5
ORDER BY    T.CodTipoFundo, F.NumFundos;
  
```

Resultado:

CodTipo	TipoFundo	NumFundo	NomeFundo	TotalAplic	QtdeAplic	MediaAplic
2	Referenciado DI	400	BETA FIC Referenciado DI Nikkei	199000.00	7	8428.571429
4	Renda Fixa	1100	BETA FIC Renda Fixa Vênus	153600.00	7	11942.857143
7	Capital Protegido	2000	BETA FI Multimercado Principal Protegido	707700.00	7	101100.000000
8	Ações IBOVESPA Indexado	2200	BETA FIC de FIA Ibovespa Indexado	202700.00	6	33783.333333
9	Ações IBOVESPA Ativo	2300	BETA FIC de FIA Ibovespa Ativo	220700.00	9	24522.222222
10	Ações Outros	2500	BETA	365000.00	14	26071.428571

Só os fundos que tiveram mais que 5 aplicações aparecem no resultado desta pesquisa. Veja que o *having* faz o filtro (aplica condições) sobre os dados gerados pelo SELECT / GROUP BY. Esses dados não existem nas tabelas originais de dados, são dados intermediários gerados em memória quando da execução do select.

Sobre esses dados é aplicada a condição do *having*, filtrando o resultado a ser exibido.

Aula 11 – Objetos de código armazenado: *view - procedure - trigger*

Create view – criação e uso de visões

Outro recurso bastante utilizado em MySQL são as visões, ou VIEWS. **VIEW** é um objeto de banco de dados (como tabelas, PKs, FKs ...) e funciona como uma “tabela virtual”. É um objeto que referencia outra(s) tabela(s) mas que não possui dados em sua estrutura. Ou seja, é uma visão virtual de dados.

A sintaxe do comando **create view** é:

CREATE VIEW	Cria uma visão de tabela(s)
	CREATE VIEW nome_da_view
Sintaxe:	AS
	<comando SELECT>
nome_da_view	É o nome da view que será criada.
Comando SELECT	É um comando SELECT válido e que pode ser executado sobre as tabelas disponíveis no banco de dados.

Este comando é especialmente útil para retornar dados em um formato que não existe fisicamente na base de dados.

Vamos considerar esta consulta similar às que fizemos anteriormente:

```

SELECT      NumAplicacao as Num, NomeCli, NomeComlFundo as NomeFundo,
            CodTipoFundo, NomeConsultor,
            DATE(DtAplicacao) as DataAplic, ValorAplicacao

FROM        aAPLICACAO          A

JOIN        aCLIENTE           C    ON     A.CodCli      =      C.CodCli

JOIN        aCONSULTORINVESTIMENTO CO
            ON     A.NumConsultor      =      CO.NumConsultor

JOIN        aFUNDOINVESTIMENTO F
            ON     A.NumFundo          =      F.NumFundo

ORDER BY    NumAplicacao;

```

Resultado (**6 primeiras linhas**):

Num	NomeCli	NomeFundo	CodTipoFundo	NomeConsultor	DataAplic	ValorAplicacao
1	LUCIO HENRIQUE PERES	FIA Petrobras	10	Marie Anne Goffaux	2007-04-01	15000.00
2	LUCAS MORAES DA SILVA	BETA FIC Curto Prazo	1	Edson Martins Dantas	2007-04-15	48000.00
3	ANDRE TAVARES DE OLIVEIRA	BETA FI Multimercado Principal Protegido	7	Pedro Sato Junior	2007-04-20	200000.00
4	ANA CRISTINA DE PAIVA	BETA FIC Renda Fixa Saturno	4	Edson Martins Dantas	2007-04-29	35000.00
5	LUCAS MORAES DA SILVA	BETA FIC Referenciado DI HiperFundo	2	Marie Anne Goffaux	2007-05-10	10000.00
6	ODIMAR CLAUDIO DA SIVA DE CARVALHO	BETA FIC Renda Fixa Saturno	4	Edson Martins Dantas	2007-05-12	29000.00



Se este tipo de seleção for usada constantemente, ao invés de escrevermos o comando SELECT/JOIN várias vezes, podemos “deixar pronta” uma consulta que agregue estes valores. Fazemos isto criando um objeto *view* no banco de dados que armazene esta consulta, usando o comando:

```
CREATE VIEW aVISA0
AS
SELECT      NumAplicacao as Num, NomeCli, NomeComlFundo as NomeFundo,
            CodTipoFUndo, NomeConsultor,
            DATE(DtAplicacao) as DataAplic, ValorAplicacao
FROM        aAPLICACAO          A
JOIN        aCLIENTE           C    ON      A.CodCli      =      C.CodCli
JOIN        aCONSULTORINVESTIMENTO CO
ON          A.NumConsultor     =      CO.NumConsultor
JOIN        aFUNDOINVESTIMENTO   F
ON          A.NumFundo         =      F.NumFundo;
```

O comando *create view* cria a visão com a mesma regra da consulta anterior. Esta *view* é, inclusive, listada entre os objetos tabela do banco de dados, veja:

SHOW TABLES;

Tables_in_aplicacao
aaplicacao
cliente
aconsultorinvestimento
afundoinvestimento
atipofundo
aviso

Para efeito de consulta, as *views* funcionam como tabelas. Podemos fazer SELECT sobre elas e fazer JOIN delas com outras tabelas. Veja o exemplo:

```
SELECT      *      FROM    aVISA0
ORDER BY    Num;
```

Resultado (6 primeiras linhas):

Num	NomeCli	NomeFundo	CodTipoFund	NomeConsultor	DataAplic	ValorAplicacao
1	LUCIO HENRIQUE PERES	FIA Petrobras	10	Marie Anne Goffaux	2007-04-01	15000.00
2	LUCAS MORAES DA SILVA	BETA FIC Curto Prazo	1	Edson Martins Dantas	2007-04-15	48000.00
3	ANDRE TAVARES DE OLIVEIRA	BETA FI Multimercado Principal Protegido	7	Pedro Sato Junior	2007-04-20	200000.00
4	ANA CRISTINA DE PAIVA	BETA FIC Renda Fixa Saturno	4	Edson Martins Dantas	2007-04-29	35000.00
5	LUCAS MORAES DA SILVA	BETA FIC Referenciado DI HiperFundo	2	Marie Anne Goffaux	2007-05-10	10000.00
6	ODIMAR CLAUDIO DA SIVA DE CARVALHO	BETA FIC Renda Fixa Saturno	4	Edson Martins Dantas	2007-05-12	29000.00

Veja que o resultado é o mesmo se fazemos um *select* sobre a *view* ou se utilizamos o *select* inicial. Observe ainda que, dentro da *view*, as colunas são apresentadas com os nomes que foram utilizados na criação da *view* e não com os nomes que estavam nas tabelas originais.

Observação: dentro da *view* os dados não são considerados de forma ordenada. assim, ao efetuarmos o *select* sobre a *view*, é necessário incluirmos o *order by* se quisermos o resultado ordenado.

Podemos usar *select / join* com *views* da mesma forma que utilizamos sobre tabelas. Podemos, inclusive, aproveitar uma visão para simplificar eventuais *joins* compostos. Por exemplo, se temos a consulta *select / join* a seguir:

```

SELECT      NumAplicacao as Num, NomeCli, NomeComlFund as NomeFundo,
            T.CodTipoFund, DescrTipoFund, NomeConsultor,
            DATE(DtAplicacao) as DataAplic, ValorAplicacao
FROM        aAPLICACAO      A
JOIN        aCLIENTE       C    ON    A.CodCli      =      C.CodCli
JOIN        aCONSULTORINVESTIMENTO CO
          ON    A.NumConsultor  =      CO.NumConsultor
JOIN        aFUNDOINVESTIMENTO F
          ON    A.NumFund       =      F.NumFund
JOIN        aTIPOFUND       T
          ON    F.CodTIpofund  =      T.CodTipoFund
ORDER BY    NumAplicacao;
  
```

Veja que em relação ao que já temos na *view*, foram acrescentados os itens em cor azul. Assim, para obtermos as mesmas informações, podemos utilizar a *view* aVISA0 com os dados de tabelas já agregados, e simplificar bastante nossa consulta. Observe que esta consulta apresenta *join* de 5 tabelas. A *view* aVISA0 já agrupa 4 dessas tabelas. Então, não é necessário repetir todas. Podemos utilizar a *view* para simplificar nossa consulta.

```

SELECT      Num, NomeCli, NomeFundo,
            T.CodTipoFundo, DescrTipoFundo, NomeConsultor,
            DataAplic, ValorAplicacao

FROM        aVISAO          V

JOIN        aTIPOFUNDO       T

ON          V.CodTipoFundo = T.CodTipoFundo

ORDER BY    Num;

```

O resultado das duas consultas é o mesmo.

Resultado (apenas primeiras linhas):

Num	NomeCli	NomeFundo	CodTipoFundo	DescrTipoFundo	NomeConsultor	DataAplic	ValorAplicacao
1	LUCIO HENRIQUE PERES	FIA Petrobras	10	Ações Outros	Marie Anne Goffaux	2007-04-01	15000.00
2	LUCAS MORAES DA SILVA	BETA FIC Curto Prazo	1	Curto Prazo	Edson Martins Dantas	2007-04-15	48000.00
3	ANDRE TAVARES DE OLIVEIRA	BETA FI Multimercado Principal Protegido	7	Capital Protegido	Pedro Sato Junior	2007-04-20	200000.00
4	ANA CRISTINA DE PAIVA	BETA FIC Renda Fixa Saturno	4	Renda Fixa	Edson Martins Dantas	2007-04-29	35000.00
5	LUCAS MORAES DA SILVA	BETA FIC Referenciado DI HiperFundo	2	Referenciado DI	Marie Anne Goffaux	2007-05-10	10000.00
6	ODIMAR CLAUDIO DA SIVA DE CARVALHO	BETA FIC Renda Fixa Saturno	4	Renda Fixa	Edson Martins Dantas	2007-05-12	29000.00

Observe que, para a consulta funcionar, temos que respeitar os nomes de colunas da *view* (em cor azul) e não de suas tabelas originais.

Os itens em cor vermelho destacam os itens acrescentados para se realizar o *join* entre a *view* e a tabela aTIPOFUNDO. Veja que aVisao funciona aqui exatamente como uma tabela, inclusive ganhando a letra v como aliás para sua identificação.

Visões são especialmente úteis para dois propósitos:

- evitar repetição de códigos de consulta, agilizando a tarefa de programação, pois podemos deixar criada uma VIEW na base de dados com consultas que realizamos frequentemente;
- utilizar em interface com outros sistemas: os sistemas raramente trabalham de forma isolada e é muito comum termos necessidade de integrá-los. Uma forma de permitir que outro sistema accesse nosso Banco de Dados, sem ter a necessidade de conhecer toda sua estrutura, é criar uma VIEW para que o outro sistema accesse nossa base. Inclusive, podemos dar acesso apenas a essa VIEW, sem a necessidade de dar acesso a outros objetos da base de dados.



Importante: Ao pedimos a descrição da view, a estrutura apresentada é virtual. Apenas a estrutura existe, pois os dados **não** estão armazenados na VIEW. Ou seja, a view só pode ser executada se as tabelas referenciadas pela VIEW existirem.



A estrutura apresentada é a criada no comando *create view*, inclusive com os títulos de colunas que utilizamos quando de sua criação.

DESCRIBE aVISAO;

Field	Type	Null	Key	Default	Extra
Num	int(11)	NO		<i>NULL</i>	
NomeCli	varchar(50)	YES		<i>NULL</i>	
NomeFundo	varchar(60)	YES		<i>NULL</i>	
CodTipoFundo	int(11)	YES		<i>NULL</i>	
NomeConsultor	varchar(30)	YES		<i>NULL</i>	
DataAplic	date	YES		<i>NULL</i>	
ValorAplicacao	decimal(15,2)	YES		<i>NULL</i>	

Views podem também ter colunas criadas só na view. Vamos supor a consulta a seguir sobre a tabela aAPLICACAO, admitindo que para cada aplicação haja retenção de 1% de imposto. Podemos criar uma view que tenha uma coluna virtual chamada **imposto**, veja:

```
CREATE      VIEW    aIMPOSTO
AS
SELECT      NumAplicacao, ValorAplicacao,
            ValorAplicacao * 0.01 as Imposto
FROM        aAPLICACAO;
```

A view é criada com uma coluna virtual, que não existe fisicamente em nenhuma tabela. Toda vez que selecionarmos os dados desta VIEW, esta coluna virtual será calculada e apresentada:

```
SELECT      *      FROM    aIMPOSTO
ORDER BY    NumAplicacao;
```

Resultado (primeiras linhas):

NumAplicacao	ValorAplicacao	Imposto
1	15000.00	150.0000
2	48000.00	480.0000
3	200000.00	2000.0000
4	35000.00	350.0000
5	10000.00	100.0000
6	29000.00	290.0000
7	350000.00	3500.0000
8	28300.00	283.0000
9	13100.00	131.0000
10	31600.00	316.0000
11	32400.00	324.0000
12	41700.00	417.0000
13	28700.00	287.0000
14	5500.00	55.0000
15	48600.00	486.0000

Create procedure – criação e uso de *stored procedures*

Procedures são procedimentos armazenados, chamados *Stored Procedures* (SPs). São comandos de consulta e atualização que podem ser armazenados na base de dados para uso em várias oportunidades.

Procedures são objetos de banco de dados que contêm comandos MySQL armazenados sob o nome de uma *procedure*. Sua criação e armazenamento são similares a de uma *view*. As *procedures* podem ou não receber parâmetros para sua execução.

A sintaxe do comando **create procedure** é:

CREATE PROCE- DURE	Cria uma Visão de Tabela(s)
Sintaxe:	<pre>CREATE PROCEDURE nome_da_proc [(IN param1 datatype, IN param2 datatype, ..., IN paramN datatype @)] <comandos MySQL></pre>

nome_da_proc	É o nome da <i>stored procedure</i> que será criada.
paramX	O parâmetro, se declarado, recebe um conteúdo externo, enviado quando a procedure é chamada. O conteúdo desse parâmetro será utilizado durante a execução da procedure e deve ter o mesmo <i>datatype</i> declarado na criação da procedure.
datatype	É o tipo de dado que é esperado para o parâmetro.
comandos MySQL	É um conjunto de comandos que devem ser executados em sequência quando a Procedure for acionada.

Podemos criar uma Procedure para executar um comando *select*, como se fosse uma *view*, veja:

```

CREATE PROCEDURE      aPROC ()

SELECT      NumAplicacao as Num, NomeCli, NomeComlFundo as NomeFundo,
            CodTipoFundo, NomeConsultor,
            DATE(DtAplicacao) as DataAplic, ValorAplicacao

FROM        aAPLICACAO          A
JOIN        aCLIENTE           C     ON      A.CodCli      =      C.CodCli
JOIN        aCONSULTORINVESTIMENTO CO
ON          A.NumConsultor    =      CO.NumConsultor
JOIN        aFUNDOINVESTIMENTO F
ON          A.NumFundo         =      F.NumFundo;

```

Para executar uma procedure armazenada, utilizamos o comando **CALL**:

```
CALL      aPROC();
```

O resultado da execução da procedure é o resultado de um *select*.

Veja as primeiras linhas:

Num	NomeCli	NomeFundo	CodTipoFundo	NomeConsultor	DataAplic	ValorAplicacao
19	ALLAN PEREIRA DOS SANTOS	BETA FIC Referenciado DI Federal	2	Fatima da Silva	2007-09-13	20400.00
30	DIEGO GONCALVES DE OLIVEIRA	BETA FIC Multimercado Iden Profit Dinâmico	6	Fatima da Silva	2008-01-16	13100.00
33	WENDELL CHAVES AGRA	BETA FIC Curto Prazo	1	Fatima da Silva	2008-02-14	11300.00
34	TIAGO VIEIRA BARBOSA	BETA FI Dívida Externa	11	Fatima da Silva	2008-02-18	25800.00
37	WENDELL CHAVES AGRA	FIA Vale do Rio Doce	10	Fatima da Silva	2008-03-13	37800.00
45	PETERSON REIMBERG DA SILVA	BETA FI Multimercado Principal Protegido	7	Fatima da Silva	2008-05-28	45400.00
50	RENATA OLIVEIRA DOS SANTOS	BETA FIC Referenciado DI HiperFundo	2	Fatima da Silva	2008-07-25	36700.00
54	JOSE GERALDO DE FREITAS	BETA FI Dívida Externa	11	Fatima da Silva	2008-09-10	38300.00
59	ANDRE TAVARES DE OLIVEIRA	BETA	10	Fatima da Silva	2008-10-28	38200.00
65	LUIS PAULO SOUZA	BETA FIC de FIA Ibovespa Ativo	9	Fatima da Silva	2008-12-11	5900.00
66	DIEGO GONCALVES DE OLIVEIRA	BETA	10	Fatima da Silva	2008-12-25	36500.00

Procedure que recebe parâmetros

Vamos ver um exemplo de *procedure* que recebe e usa parâmetros em sua execução. A procedure ainda é simples, tendo apenas um *select*. Mas os parâmetros de entrada serão utilizados no filtro do *where*, veja:

```
CREATE PROCEDURE aPROCparam (IN ano INTEGER, IN mes INTEGER)

SELECT NumAplicacao AS Num, NomeCli, NomeComlFundo AS NomeFundo,
       CodTipoFundo, NomeConsultor,
       DATE( DtAplicacao ) AS DataAplic, ValorAplicacao
FROM   aAPLICACAO A
JOIN  aCLIENTE C    ON A.CodCli      = C.CodCli
JOIN  aCONSULTORINVESTIMENTO CO
      ON A.NumConsultor = CO.NumConsultor
JOIN  aFUNDOINVESTIMENTO F
      ON A.NumFundo     = F.NumFundo
WHERE YEAR(DtAplicacao) = ano
AND   MONTH(DtAplicacao) < mes;
```

Para sua execução, é necessário passar como parâmetros ano e mês:

```
CALL aPROCparam (2008, 4);
```

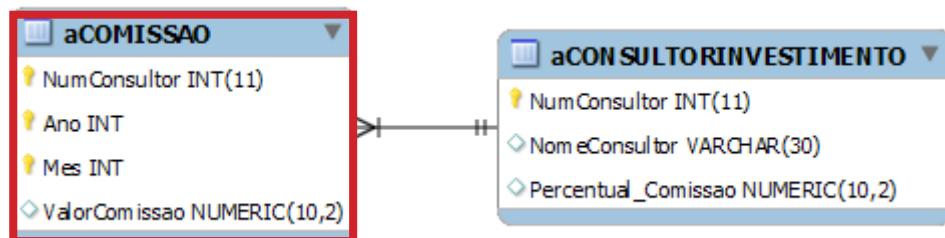
Resultado:

Num	NomeCli	NomeFundo	CodTipoFundo	NomeConsultor	DataAplic	ValorAplicacao
30	DIEGO GONCALVES DE OLIVEIRA	BETA FIC Multimercado ;Iden Profit Dinâmico	6	Fatima da Silva	2008-01-16	3100.00
33	WENDELL CHAVES AGRA	BETA FIC Curto Prazo	1	Fatima da Silva	2008-02-14	1300.00
34	TIAGO VIEIRA BARBOSA	BETA FI Dívida Externa	11	Fatima da Silva	2008-02-18	2300.00
37	WENDELL CHAVES AGRA	FIA Vale do Rio Doce	10	Fatima da Silva	2008-03-13	3700.00
35	DIEGO GONCALVES DE OLIVEIRA	BETA	10	Marie Anne Goffaux	2008-02-25	4300.00
29	CLEA CRISTINA CALIMAN	BETA FIC Referenciado DI Platinum	2	Marina de Oliveira	2008-01-04	24000.00
32	MARIA LUCIA DE PAIVA	BETA FI Multimercado Principal Protegido	7	Marina de Oliveira	2008-02-05	30200.00
36	MARIA LUCIA DE PAIVA	BETA FIC de FIA Ibovespa Ativo	9	Marina de Oliveira	2008-03-11	34600.00
38	CLAUDIO FRANCISCO MENDES	BETA FIC Referenciado DI HiperFundo	2	Edson Martins Danta	2008-03-19	6700.00
31	FLAVIO MACEDO DE GOIS	BETA FIC Renda Fixa Saturno	4	Pedro Sato Junior	2008-01-24	21800.00

O resultado foi filtrado a partir dos parâmetros passados na chamada da procedure.

Procedure com diversos comandos – uso de *delimiter begin end*

Vamos ver um outro exemplo de procedure que grava dados em uma tabela. Para isso, vamos criar uma tabela nova no DB APPLICACAO: uma tabela de comissões, denominada aCOMISSAO para armazenar valores mensais de comissão de cada consultor de investimento em cada ano/mês.



Utilize o comando a seguir para a criação da tabela aCOMISSAO:

```
CREATE TABLE aCOMISSAO (
    NumConsultor      INT(11)          NOT NULL,
    Ano                INT              NOT NULL,
    Mes                INT              NOT NULL,
    ValorComissao     DECIMAL(10,2)    NULL,
    PRIMARY KEY (NumConsultor, Ano, Mes),
    FOREIGN KEY (NumConsultor)
        REFERENCES aCONSULTORINVESTIMENTO (NumConsultor)
);
```

Sobre a tabela aCOMISSAO, vamos criar uma *procedure* que realize os seguintes passos:

1. Receba como parâmetros ano e mês de referência;
2. Limpe os valores de todos os consultores de investimento para aquele ano/mês;
3. Calcule a comissão de cada consultor de investimento para o ano/mês e inclua registros na tabela aCOMISSAO

Veja que na definição da *procedure* aPROCPARAM, utilizamos apenas uma vez o ponto e vírgula ao final do comando *select*, para indicar que o código da procedure está encerrado. Mas para criarmos uma *procedure* que efetue as tarefas 1. 2. e 3. é necessário indicar ao MySQL onde os comandos se iniciam e onde eles terminam, pois dentro do código da *procedure* haverá mais de um ponto e vírgula.

Para tanto, utilizamos dois conjuntos de “comandos de bloco”, ambos indicando início e fim:

- os comandos da Procedure ficam entre os comandos **begin end**;
- e tudo fica entre as entradas **delimiter**.

A sintaxe de **delimiter** é:

DELIMI- TER	
Sintaxe: <pre> DELIMITER <caracter delimitador> CREATE PROCEDURE <proc_name> ([parâmetros]) BEGIN <comando SQL1>; <comando SQL2>; <comando SQLN>; END<caracter delimitador> DELIMITER ; </pre>	

caracter delimitador É o caracter utilizado para indicar o início e o final dos comandos da *procedure*.

proc_name Nome do objeto *procedure*, tal como mostrado anteriormente.

comando SQLX Um dos comandos que compõem a *procedure*.

Nossa *procedure*, executando os passos requeridos e seguindo as regras de bloco, ficaria assim:

```

DELIMITER //

CREATE PROCEDURE      aCALCCOMISSAO_MES

    (IN pANO INTEGER, IN pMES INTEGER)

BEGIN

    -- limpa os valores do Ano/Mês do parâmetro para regravá-los

    DELETE      FROM  aCOMISSAO

    WHERE Ano    =      pANO
        AND   Mes    =      pMES;

```

```
-- Calcula e grava na tabela COMISSAO os totais por Consultor/
Mês

INSERT      INTO  aCOMISSAO

SELECT      A.NumConsultor, YEAR(DtAplicacao) ,
            MONTH(DtAplicacao) ,
            SUM(ValorAplicacao * Percentual_Comissao / 100)

FROM        aAPLICACAO          A

JOIN        aCONSULTORINVESTIMENTO C

ON          A.NumConsultor      =      C.NumConsultor

WHERE       YEAR(DtAplicacao) =      pANO

AND         MONTH(DtAplicacao)=      pMES

GROUP BY    A.NumConsultor, YEAR(DtAplicacao) ,
            MONTH(DtAplicacao) ;

END //;

DELIMITER ;
```

Para executá-la, utilizamos o *call*, passando os parâmetros ano e mês:

CALL aCALCCOMISSAO_MES (2008, 4);
--

Repita a chamada com outros parâmetros ano / mês, para você obter mais resultados. Por exemplo, faça as seguintes chamadas:

```
CALL      aCALCCOMISSAO_MES (2007, 8);
CALL      aCALCCOMISSAO_MES (2007,11);
CALL      aCALCCOMISSAO_MES (2008, 1);
CALL      aCALCCOMISSAO_MES (2008, 3);
CALL      aCALCCOMISSAO_MES (2008, 4);
CALL      aCALCCOMISSAO_MES (2008, 7);
```

Depois faça um *select* na tabela aCOMISSAO:

```
SELECT      Ano, Mes, NumConsultor, ValorComissao      FROM aCOMISSAO
ORDER BY Ano, Mes, NumConsultor;
```

Resultado:

Ano	Mes	NumConsultor	ValorComissao
2007	8	11	1630.00
2007	11	12	261.00
2007	11	13	756.00
2007	11	14	354.00
2008	1	10	196.50
2008	1	12	432.00
2008	1	14	327.00
2008	3	10	567.00
2008	3	12	622.80
2008	3	13	934.00
2008	4	12	792.00
2008	4	13	1578.00
2008	7	10	550.50
2008	7	12	437.40

Create trigger

Triggers são objetos que contêm procedimentos armazenados, similares às *procedures*. A diferença é que são executadas automaticamente quando ocorre um evento com uma tabela ao invés de serem executadas por um comando *call* como as *procedures*.

As *triggers* sempre são associadas a uma determinada tabela e podem ser associadas aos seguintes eventos desta tabela:

- inclusão de linhas;
- alteração em alguma linha;
- exclusão de alguma linha.

Devem ser criadas *triggers* diferentes para cada tipo de evento da tabela (inclusão, alteração, exclusão), embora seu código seja similar.

Há vários usos para *triggers*, mas uma utilização comum é para gravação do arquivo de Log de manutenções sobre o conteúdo de uma tabela. É comum nos sistemas a necessidade de gravar um Log para que o sistema possa ser auditado.

Vamos estudar *triggers* de Inclusão, Alteração e Exclusão com a apresentação da sintaxe e de exemplos para cada tipo de evento. Para exemplificar os diferentes *triggers*, vamos criar uma tabela de Log no DB APLICACAO, denominada aLOG. Essa tabela armazenará o log de manutenções em dados da tabela aCLIENTE. Crie esta tabela através do comando:

CREATE TABLE aLOG (
Tabela	VARCHAR(40)	NULL,
DataLog	DATETIME	NULL,
Evento	VARCHAR(10)	NULL,
UsuarioDB	VARCHAR(20)	NULL,
CodCli	INTEGER	NULL,
NomeCli_OLD	VARCHAR(50)	NULL,
DtNasc_OLD	DATETIME	NULL,
CPF_OLD	VARCHAR(11)	NULL,
NomeCli_NEW	VARCHAR(50)	NULL,
DtNasc_NEW	DATETIME	NULL,
CPF_NEW	VARCHAR(11)	NULL
);		

Trigger de inclusão

As triggers de inclusão são disparadas quando ocorre um comando *insert* sobre a tabela.

A sintaxe de **create trigger** após um *insert* é:

CREATE TRIGGER - INSERT	Cria uma trigger de inclusão sobre uma tabela
Sintaxe:	DELIMITER //
	CREATE TRIGGER nome_trigger_inc AFTER INSERT ON
	nome_tabela
	FOR EACH ROW
	BEGIN
	<comandos SQL>
	END
	//
	DELIMITER ;
nome_trigger_inc	É o nome da trigger que será criada.
nome_tabela	Nome da tabela que ficará associada à trigger.
comandos SQL	São os comandos SQL que devem ser executados quando a trigger for acionada.

Veja que utilizamos *delimiter* e *begin end* de forma muito similar ao que usamos com *procedures*.

Vamos agora criar uma *trigger* de Inclusão através do comando:

```

DELIMITER //

CREATE TRIGGER aCLIENTE_INCLUSAO AFTER INSERT ON aCLIENTE
FOR EACH ROW
BEGIN
    INSERT INTO aLOG SET
        Tabela      = 'aCLIENTE' ,
        DataLog     = NOW() ,
        Evento      = 'Inclusao' ,
        UsuarioDB   = USER() ,
        Código       = NEW.CodCli ,
        NomeCli_NEW = NEW.NomeCli ,
        DtNasc_NEW   = NEW.DtNascimento ,
        CPF_NEW      = NEW.CPF
;
END
//

DELIMITER ;

```

Esta *trigger* grava uma nova linha na tabela aLOG a cada inclusão que ocorrer na tabela aCLIENTE. Na inclusão em aLOG, observe que as 4 últimas colunas são gravadas com o conteúdo que está sendo incluído na tabela aCLIENTE.

Ao usarmos **NEW**, antes de uma coluna, indicamos para o MySQL que deve ser considerado o conteúdo que ficará gravado em aCLIENTE, após a execução do comando *insert* nesta tabela.

Para testarmos esta *trigger*, vamos incluir 2 novos clientes na tabela aCLIENTE e consultar o efeito na tabela aLOG:

```
INSERT INTO aCLIENTE
VALUES (1002, 'Diego Costa', '1985-05-21', '12345678900');

INSERT INTO aCLIENTE
VALUES (1005, 'Mourinho', '1974-08-10', '76576576576');
```

Consultando a tabela aLOG:

```
SELECT * FROM aLOG;
```

Resultado:

Tabela	DataLog	Evento	UsuarioDB	Codigo	NomeCli_OLD	DtNasc_OLD	CPF_OLD	NomeCli_NEW	DtNasc_NEW	CPF_NEW
aCLIENTE	2014-04-27 16:56:45	Inclusao	root@localhost	1002	NULL	NULL	NULL	Diego Costa	1985-05-21 00:00:00	12345678900
aCLIENTE	2014-04-27 16:56:45	Inclusao	root@localhost	1005	NULL	NULL	NULL	Mourinho	1974-08-10 00:00:00	76576576576

Observamos que são criadas duas linhas na tabela aLOG, pois a *trigger* foi disparada duas vezes. As 3 últimas colunas contêm os dados gravados na tabela aCLIENTE, além do código.

Trigger de alteração

As *triggers* de alteração são disparadas quando ocorre um comando *update* sobre a tabela.

A sintaxe de **create trigger** após um UPDATE é:

CREATE TRIGGER – UPDATE	<p>Cria uma Trigger de Alteração sobre uma Tabela</p> <p>DELIMITER //</p> <p>CREATE TRIGGER nome_trigger_alt BEFORE UPDATE ON nome_tabela</p> <p>FOR EACH ROW</p> <p>Sintaxe:</p> <p style="text-align: center;">BEGIN <comandos SQL> END // DELIMITER ;</p>
--	---

nome_trigger_alt É o nome da Trigger que será criada.

nome_tabela Nome da tabela que ficará associada à Trigger.

comandos SQL São os comandos SQL que devem ser executados quando a Trigger for acionada.

Vamos criar esta *trigger* de alteração:

```

DELIMITER //

CREATE TRIGGER aCLIENTE_ALTERACAO BEFORE UPDATE ON aCLIENTE
FOR EACH ROW
BEGIN
    INSERT INTO aLOG SET
        Tabela      = 'aCLIENTE',
        DataLog     = NOW(),
        Evento      = 'Alteracao',
        UsuarioDB   = USER(),
       Codigo       = NEW.CodCli,
        NomeCli_OLD = OLD.NomeCli,
        DtNasc_OLD  = OLD.DtNascimento,
        CPF_OLD     = OLD.CPF,
        NomeCli_NEW = NEW.NomeCli,
        DtNasc_NEW  = NEW.DtNascimento,
        CPF_NEW     = NEW.CPF
    ;
END
//  

DELIMITER ;

```

Vamos efetuar 2 alterações na tabela aCLIENTE:

```

UPDATE      aCLIENTE
SET          NomeCli           =      'Miranda',
             DtNascimento     =      '1982-02-20'
WHERE        CodCli            =      1005;

UPDATE      aCLIENTE
SET          CPF               =      '98798798798'
WHERE        CodCli            =      1005;
  
```

Agora vamos verificar como ficou a tabela aLOG após estas alterações:

```
SELECT * FROM aLOG;
```

Tabela	DataLog	Evento	UsuarioDB	Codigo	NomeCli_OLD	DtNasc_OLD	CPF_OLD	NomeCli_NEW	DtNasc_NEW	CPF_NEW
aCLIENTE	2014-04-27 16:56:45	Inclusao	root@localhost	1002	NULL	NULL	NULL	Diego Costa	1985-05-21 00:00:00	12345678900
aCLIENTE	2014-04-27 16:56:45	Inclusao	root@localhost	1005	NULL	NULL	NULL	Mourinho	1974-08-10 00:00:00	76576576576
aCLIENTE	2014-04-27 17:05:45	Alteracao	root@localhost	1005	Mourinho	1974-08-10 00:00:00	76576576576	Miranda	1982-02-20 00:00:00	76576576576
aCLIENTE	2014-04-27 17:05:45	Alteracao	root@localhost	1005	Miranda	1982-02-20 00:00:00	76576576576	Miranda	1982-02-20 00:00:00	98798798798

Observamos que são criadas duas linhas na tabela aLOG, pois a *trigger* foi disparada duas vezes, uma a cada *update*. As colunas de sufixo OLD contêm os dados da linha antes da alteração e as colunas de sufixo new contêm os dados das colunas após a alteração.

Trigger de exclusão

As *triggers* de exclusão são disparadas quando ocorre um comando *delete* sobre a tabela.

A sintaxe de **create trigger** após um *delete* é:

CREATE TRIGGER – DELETE	Cria uma Trigger de Exclusão sobre uma Tabela
	DELIMITER //
	CREATE TRIGGER nome_trigger_exc BEFORE DELETE ON nome_tabela
	FOR EACH ROW
	Sintaxe:
	BEGIN <comandos SQL>
	END
	//
	DELIMITER ;
nome_trigger_exc	É o nome da <i>trigger</i> que será criada.
nome_tabela	Nome da tabela que ficará associada à <i>trigger</i> .
comandos SQL	São os comandos SQL que devem ser executados quando a <i>trigger</i> for acionada.

Vamos criando uma *trigger* de exclusão:

```

DELIMITER //

CREATE TRIGGER aCLIENTE_EXCLUSAO BEFORE DELETE ON aCLIENTE
FOR EACH ROW
BEGIN
    INSERT INTO aLOG SET
        Tabela      = 'aCLIENTE',
        DataLog     = NOW(),
        Evento      = 'Exclusao',
        UsuarioDB   = USER(),
       Codigo       = OLD.CodCli,
        NomeCli_OLD = OLD.NomeCli,
        DtNasc_OLD  = OLD.DtNascimento,
        CPF_OLD     = OLD.CPF
    ;
END
//  

DELIMITER ;

```

Vamos excluir alguns registros na tabela aCLIENTE e observar aLOG após as exclusões:

```

DELETE FROM aCLIENTE
WHERE CodCli      =
1005;

DELETE FROM aCLIENTE
WHERE CodCli      =
1002;

```

```
SELECT * FROM aLOG;
```

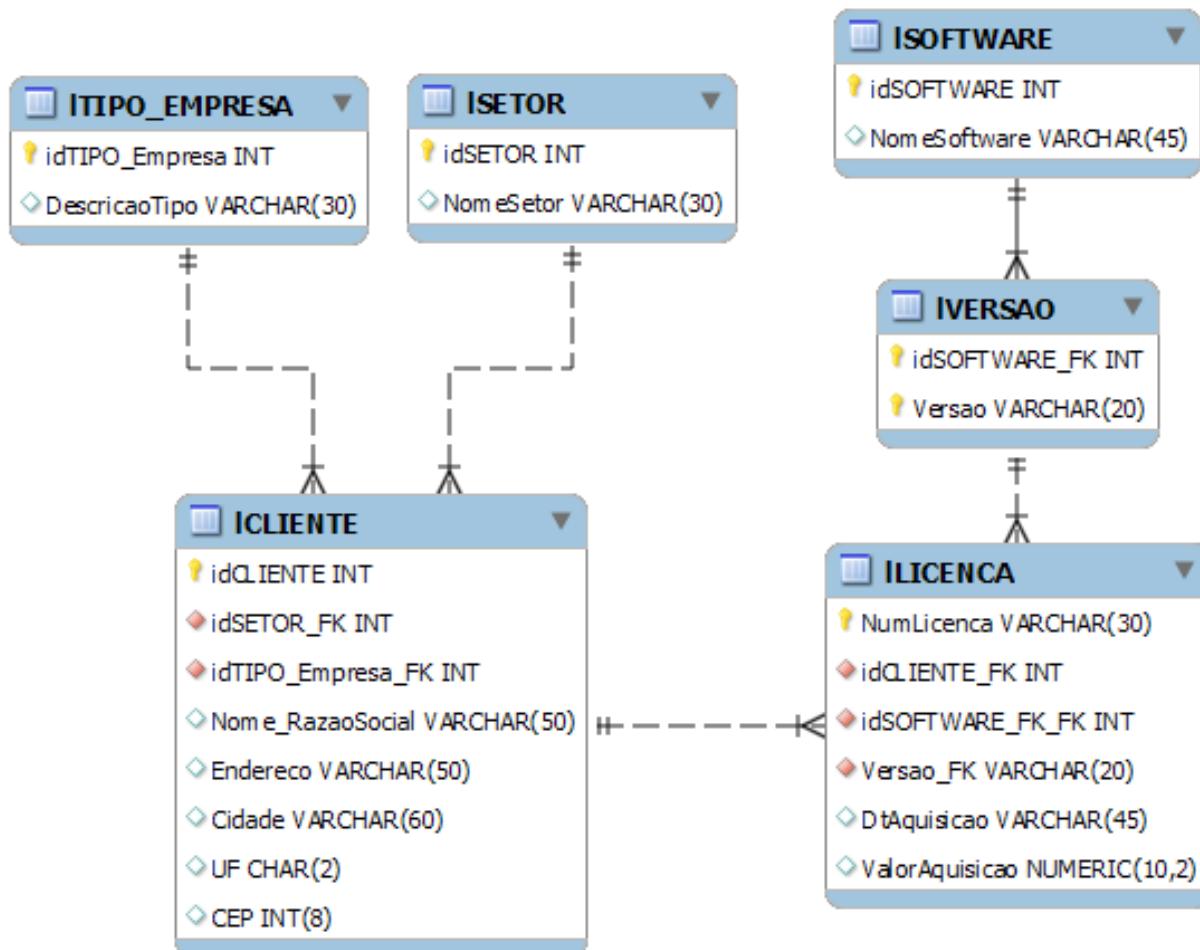
Tabela	DataLog	Evento	UsuarioDB	Codigo	NomeCli_OLD	DtNasc_OLD	CPF_OLD	NomeCli_NEW	DtNasc_NEW	CPF_NEW
aCLIENTE	2014-04-27 16:56:45	Inclusao	root@localhost	1002	NULL	NULL	NULL	Diego Costa	1985-05-21 00:00:00	12345678900
aCLIENTE	2014-04-27 16:56:45	Inclusao	root@localhost	1005	NULL	NULL	NULL	Mourinho	1974-08-10 00:00:00	76576576576
aCLIENTE	2014-04-27 17:05:45	Alteracao	root@localhost	1005	Mourinho	1974-08-10 00:00:00	76576576576	Miranda	1982-02-20 00:00:00	76576576576
aCLIENTE	2014-04-27 17:05:45	Alteracao	root@localhost	1005	Miranda	1982-02-20 00:00:00	76576576576	Miranda	1982-02-20 00:00:00	98798798798
aCLIENTE	2014-04-27 17:15:22	Exclusao	root@localhost	1005	Miranda	1982-02-20 00:00:00	98798798798	NULL	NULL	NULL
aCLIENTE	2014-04-27 17:15:22	Exclusao	root@localhost	1002	Diego Costa	1985-05-21 00:00:00	12345678900	NULL	NULL	NULL

Observamos que são criadas duas linhas na tabela aLOG, pois a *trigger* foi disparada duas vezes, uma a cada *delete*. As colunas de sufixo *old* contêm os dados da linha antes de sua exclusão da tabela.

Exercícios do Módulo 4

Para os exercícios deste módulo, você pode criar o mesmo banco de dados MODULO3 já criado nos exercícios do módulo 3.

- 1) Utilizaremos o mesmo modelo de dados que armazena informações de licenças de software utilizado no módulo anterior de nosso curso:**



Considerando este modelo e os comandos MySQL que aprendemos neste módulo, responda as questões abaixo. Teste seus comandos no banco de dados:

FUNÇÕES

- a) Liste a quantidade de clientes cadastrados;
- b) Liste as aquisições que foram feitas no mesmo mês que você nasceu (Dica: use a função MONTH)
- c) Liste as aquisições de licenças feitas no mês de maio de 2007 (use a função MONTH e YEAR para obter o mês e o ano da data:);
- d) Para os tipos de empresa, exiba sua identificação, sua descrição e as 5 primeiras letras da descrição. Ordene a saída pela descrição em ordem crescente.
- e) Para os tipos de empresa, exiba sua identificação, sua descrição e as 5 últimas letras da descrição. Ordene a saída pela descrição em ordem decrescente.
- f) Para os tipos de empresa, exiba sua identificação, sua descrição e as letras da descrição da 6^a a 10^a posições.
- g) Liste os nomes dos clientes e quantidade de bytes que cada nome contém. Ordene pelo nome.
- h) Exiba todas as licenças com número da Licença, data de aquisição e quantos dias já decorreram da aquisição até a data atual.
- i) Exiba os nomes setores da economia duas vezes: uma em letras maiúsculas e outra em letras minúsculas.

JOIN

- j) Exiba todos softwares com todas as suas versões. Ordene por nome e versão do software.
- k) Exiba os clientes com a descrição de seu tipo e com o nome do seu setor. Ordene por tipo de empresa e setor.
- l) Exiba os clientes (identificação e nome) e as licenças adquiridas (número, data aquisição e valor).
- m) Liste os clientes e os nomes dos softwares que cada um adquiriu. Ordene por cliente e software e iniba as repetições.
- n) Exiba os clientes com a descrição de seu tipo e com o nome do seu setor. Exiba apenas os clientes que pertençam a uma das UFs: "SP", "RS", "PR", MG".
- o) Exiba software, versão, nome do cliente, descrição do tipo do cliente, nome do setor do cliente e as licenças adquiridas pelo cliente com seu número, data e valor de aquisição. Ordene por software, versão, data, cliente.

FUNÇÕES ESTATÍSTICAS COM JOIN

- p) Lista a quantidade de licenças vendidas.
- q) Sobre os valores das aquisições, exiba o valor total das licenças, o valor médio, o maior valor e o menor valor.

- r) Liste a quantidade de clientes cadastrados, mas considere apenas os clientes do setor de "farmacêutica".
- s) Liste a quantidade de licenças para cada cliente. Apresente o nome do cliente. Ordene pelo nome do cliente.
- t) Sobre os valores das aquisições, exiba o valor total e o valor médio das licenças adquiridas por cliente. Ordene por nome de cliente.
- u) Exiba o valor total de licenças comercializadas por setor da economia. Ordene por setor.
- v) Exiba o valor total de licenças comercializadas por tipo de empresa. Ordene por tipo.
- w) Exiba o valor total de licenças comercializadas por software / versão. Ordene por software / versão.
- x) Exiba a quantidade de licenças adquiridas por cada empresa. Para isso, apresente nome do software, nome da empresa (cliente) e a quantidade de licenças adquiridas (de qualquer versão).
- y) Apresente os nomes dos clientes e a quantidade de licenças adquiridas. Liste apenas as empresas que compraram mais que 10 licenças.

OBJETOS DE CÓDIGO ARMAZENADO

- z) Crie uma view que mostre todos os dados do cliente, do setor da economia e do tipo de empresa.
- aa) Crie uma procedure que receba como parâmetros data de início e fim. Sabendo que a PROC deve exibir o valor de comissão que cada licença de software gerou, exiba as seguintes colunas:
- ◊ Nome do cliente;
 - ◊ Setor da economia;
 - ◊ Software e versão;
 - ◊ Número da Licença;
 - ◊ Data da aquisição;
 - ◊ Valor da Licença;
 - ◊ Valor da comissão (3% sobre o Valor da Licença).

A PROC deve exibir a saída em ordem de cliente, software, versão.

- ab) Crie uma procedure que receba como parâmetros data de início e fim. Sabendo que a PROC deve exibir os totais de valores vendidos por ano, de cada software, exiba as seguintes colunas:
- ◊ Ano da aquisição da licença;
 - ◊ Nome do software;
 - ◊ Valor total de licenças vendidas no ano para aquele software.

A PROC deve exibir a saída em ordem de ano, software.

- ac) Crie triggers de inclusão, alteração e exclusão para a tabela ISOFTWARE. Essas Triggers deverão armazenar as manutenções nesta tabela em uma tabela de LOG (ILOG) com as seguintes colunas: ILOG (NomeTabela, Tipo_Operacao, Identificação do Software, Data da Operação, Hora da Operação).

A coluna Tipo_Operação pode receber um dos valores: “Inclusão”, “Alteração”, “Exclusão”.

- ◊ Crie a tabela ILOG;
- ◊ Crie as triggers de inclusão, alteração e exclusão.
- ◊ Inclua, altere e exclua um novo software e verifique se estas operações foram logadas corretamente.

Considerações Finais

Caro(a) aluno(a), estamos encerrando mais uma etapa de nosso curso! Foi uma alegria estudar com você os assuntos desta disciplina. Sinceramente espero que você tenha expandido seus conhecimentos e aprendido mais recursos para utilizar em sua vida profissional.

Você notou que esta é uma disciplina prática e exercitar o aprendizado é e continuará sendo essencial para que você aprimore seu conhecimento.

No início da disciplina estudamos alguns conceitos importantes sobre bancos de dados. Depois passamos a estudar modelagem de dados que é essencial para construirmos um *database* estruturado, com tabelas bem planejadas e organizadas, proporcionando mais facilidades para desenvolver softwares que se utilizam destas tabelas.

Depois passamos a uma parte mais prática e começamos a utilizar o MySQL. Seus recursos são similares a outros SGBDs de mercado, principalmente os comandos DDL e DML que estudamos, mas certamente você percebeu a facilidade de se obter este software e as facilidades que ele nos oferece com uso simples e ágil.

Os comandos MySQL apresentados possuem algumas opções adicionais além das apresentadas nesta disciplina, mas você aprendeu o suficiente para incrementar a criação de seus programas, indo além, caso deseje.

Ferramentas de banco de dados são poderosas e indispensáveis na criação de bons sistemas. À medida que você progredir no nosso curso, fará uso conjunto de várias tecnologias, utilizando banco de dados MySQL com outros programas e ferramentas que você estiver aprendendo, especialmente da plataforma Java.

Gostaria de sugerir que você revisasse periodicamente o conteúdo de banco de dados, especialmente re fazendo alguns exercícios. Isto o manterá afiado no assunto. Além disso, frequentemente são lançadas novas versões dos softwares, inclusive do MySQL. Fique atento aos lançamentos e às novidades que são incluídas a cada nova versão.

Bons estudos e sucesso em sua vida profissional.

Prof. Paulo Borba



Caro aluno, veja abaixo e faça o conteúdo do **módulo 5** desta disciplina. Este material é complementar aos seus estudos e não será avaliado.

Recomendo que você estude este material para avançar mais em seu conhecimento de SQ e MySQL. Neste módulo há exercícios propostos cujas respostas também são apresentadas. Tente fazer sozinho, depois você pode conferir as respostas, ok?

Módulo 5 – mais comandos na prática

Caro(a) aluno(a), estamos quase finalizando nossa disciplina! Os principais comandos e recursos de banco de dados e do MySQL foram apresentados nos módulos anteriores. Este é um módulo diferente dos demais. Vamos apresentar aqui algumas variações dos comandos já apresentados com recursos adicionais. Algumas situações particulares que podem ser muito úteis serão abordadas. É o módulo mais prático de nossa disciplina!

Aula 12 – Utilizando sub-select

Um recurso interessante do MySQL é a utilização de sub-selects que são SELECTs dentro de outro SELECT.

Sub-select com IN e NOT IN

Um uso comum de **sub-select** é quando utilizamos o retorno de um SELECT como parte da condição de um outro SELECT. A sintaxe do comando SELECT ficaria assim neste caso:

SELECT	<colunas>	FROM	<tabela1>
WHERE	<coluna>	IN	
(SELECT <coluna> FROM tabela2 [WHERE <condição>])			

Para exemplificar, vamos listar os clientes, da tabela aCLIENTE do nosso BD APPLICACAO, que já fizeram alguma aplicação financeira. Uma opção é fazer JOIN entre as tabelas aCLIENTE e aAPPLICACAO com DISTINCT no nome do cliente. Outra opção é fazer uma pesquisa com Sub-Select, como no seguinte comando:

```

SELECT      *      FROM  aCLIENTE
WHERE      CodCli IN
(
SELECT      DISTINCT      CodCli      FROM  aAPLICACAO
);

```

Os comandos produzem o seguinte efeito:

- o SELECT mais interno retorna os códigos de cliente de todos os clientes que fizeram alguma aplicação;
- o SELECT mais externo utiliza este retorno na cláusula WHERE para filtrar sua exibição, como são vários valores é necessário usar o IN, pois o "=" daria erro.

Resultado (apenas primeiras linhas):

CodCli	NomeCli	DtNascimento	CPF
12	TARCISIO HENRIQUE DA SILVA PAULA	1978-09-07 00:00:00	77110812162
14	SERGIO VIANA JUNIOR	1980-01-02 00:00:00	35523514712
16	WILSON PAULINO DE FARIA	1984-09-02 00:00:00	26406316777
25	LUCIO HENRIQUE PERES	1983-01-09 00:00:00	61603425520
37	MARIA LUCIA DE PAIVA	1969-09-08 00:00:00	58168237073
38	WENDELL CHAVES AGRA	1958-09-06 00:00:00	75889838216
39	MARCUS VINICIUS DE CASTRO FERRAZ	1978-01-07 00:00:00	28714439370
40	MARCOS ANTONIO OSORIO	1984-01-06 00:00:00	26141240339
50	GILSON FERREIRA DE SOUZA	1960-01-09 00:00:00	54200050104

CodCli
12
12
14
16
16
16
16
16
16
16
16
16
25
37

Note que o SELECT mais interno retorna uma lista de códigos, como na lista CodCli à esquerda.

Esta lista é o retorno parcial do SELECT mais interno.

Veja que só os códigos dos clientes são exibidos por este SELECT.

Então, o SELECT externo ficaria assim:

```
SELECT *      FROM aCLIENTE
WHERE CodCli IN
(
  12, 12, 14, 16, 16, ..., 16, 25, 37, ...
);
```

Assim, o SELECT externo apresentará os clientes cujo código fazem parte desta lista de códigos que foi retornada do SELECT interno.

O uso deste sub-select fica mais importante quando queremos **selecionar** os clientes que **não** fizeram aplicações. Para isso, usamos a mesma estrutura, mas agora com NOT IN, veja:

```
SELECT      *      FROM aCLIENTE
WHERE      CodCli NOT IN
(
  SELECT DISTINCT      CodCli      FROM aAPLICACAO
);
```

O retorno do SELECT interno é o mesmo, mas desta vez a condição do SELECT externo filtra os clientes que **não** estão contidos na lista, resultando em:

CodCli	NomeCli	DtNascimento	CPF
184	CLAYTON GOMES DA CUNHA	1980-06-04 00:00:00	20960184888
269	DOUGLAS MARTINS DOS SANTOS	1978-06-05 00:00:00	89459269572
412	MARCIO RODRIGO GOIS	1984-05-09 00:00:00	22847412403
428	RONNALDO KRAFT DA SILVA	1969-05-02 00:00:00	99382428963
458	IVE SANDRA ALMEIDA JARDIM	1958-05-07 00:00:00	44413458487
860	RODRIGO ALVES SOARES	1984-08-04 00:00:00	34394860428
893	ALVARO MOREIRA LUNA	1958-08-08 00:00:00	13590893211
927	ANGELO TAVARES DA CONCEICAO	1980-08-09 00:00:00	79923927562

O mesmo processo poderia ser aplicado se desejássemos contar os clientes que não fizeram aplicações financeiras. Neste caso, bastaria trocar, no SELECT, o * por COUNT(*):

```

SELECT COUNT(*) FROM aCLIENTE
WHERE CodCli NOT IN (
    SELECT DISTINCT CodCli FROM aAPLICACAO
);

```

Resultado: count(*)

8

Podemos, ainda, utilizar **sub-select** em mais de um nível! Por exemplo, se quisermos o seguinte retorno: exibir os clientes que aplicaram em algum fundo de investimento que seja do tipo renda fixa; o comando SQL ficaria assim:

```

1   SELECT * FROM aCLIENTE
    WHERE CodCli IN (
        2   SELECT DISTINCT CodCli FROM aAPLICACAO
        WHERE NumFundo IN (
            3   SELECT NumFundo FROM aFUNDOINVESTIMENTO
            WHERE CodTipoFundo= 4
        )
    )
;

```

Neste exemplo, há um sub-select dentro de outro. A forma de se interpretar esses comandos são do SELECT mais interno para o mais externo. Isto porque para a execução do mais externo há dependência do resultado do mais interno. Os resultados parciais (3 e 2) e o final (1) são:

1 Resultado: <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>NumFundo</th> </tr> </thead> <tbody> <tr><td>1100</td></tr> <tr><td>1200</td></tr> <tr><td>1300</td></tr> <tr><td>1400</td></tr> <tr><td>1500</td></tr> </tbody> </table>	NumFundo	1100	1200	1300	1400	1500	2 Resultado: <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>CodCli</th> </tr> </thead> <tbody> <tr><td>39</td></tr> <tr><td>389</td></tr> <tr><td>111</td></tr> <tr><td>16</td></tr> <tr><td>86</td></tr> <tr><td>137</td></tr> <tr><td>710</td></tr> <tr><td>722</td></tr> <tr><td>331</td></tr> <tr><td>211</td></tr> <tr><td>377</td></tr> <tr><td>500</td></tr> <tr><td>38</td></tr> <tr><td>37</td></tr> </tbody> </table>	CodCli	39	389	111	16	86	137	710	722	331	211	377	500	38	37																																							
NumFundo																																																													
1100																																																													
1200																																																													
1300																																																													
1400																																																													
1500																																																													
CodCli																																																													
39																																																													
389																																																													
111																																																													
16																																																													
86																																																													
137																																																													
710																																																													
722																																																													
331																																																													
211																																																													
377																																																													
500																																																													
38																																																													
37																																																													
Números dos fundos de investimento que são do Tipo 4 "Renda Fixa".	Códigos dos clientes que aplicaram em algum dos fundos da lista ao lado esquerdo.																																																												
3 Resultado: Lista de clientes que aplicaram em fundos de renda fixa.	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>CodCli</th><th>NomeCli</th><th>DtNascimento</th><th>CPF</th></tr> </thead> <tbody> <tr><td>39</td><td>MARCUS VINICIUS DE CASTRO FERRAZ</td><td>1978-01-07 00:00:00</td><td>28714439370</td></tr> <tr><td>389</td><td>CICERO SILVA FRANCO</td><td>1983-06-07 00:00:00</td><td>78015389474</td></tr> <tr><td>111</td><td>ANDRE TAVARES DE OLIVEIRA</td><td>1978-02-09 00:00:00</td><td>32912111927</td></tr> <tr><td>16</td><td>WILSON PAULINO DE FARIA</td><td>1984-09-02 00:00:00</td><td>26406316777</td></tr> <tr><td>86</td><td>FLAVIO MACEDO DE GOIS</td><td>1978-09-03 00:00:00</td><td>47464586179</td></tr> <tr><td>137</td><td>DIEGO GONCALVES DE OLIVEIRA</td><td>1984-02-06 00:00:00</td><td>45470137269</td></tr> <tr><td>710</td><td>JOSE GERALDO DE FREITAS</td><td>1984-08-04 00:00:00</td><td>31678710396</td></tr> <tr><td>722</td><td>RAFAEL OLIVEIRA LEITE</td><td>1969-08-07 00:00:00</td><td>22869722167</td></tr> <tr><td>331</td><td>DENIS DE LIMA GONÇALVES</td><td>1969-06-04 00:00:00</td><td>61757331137</td></tr> <tr><td>211</td><td>ANA CRISTINA DE PAIVA</td><td>1983-06-05 00:00:00</td><td>66477211111</td></tr> <tr><td>377</td><td>ODIMAR CLAUDIO DA SIVA DE CARVALHO</td><td>1980-06-04 00:00:00</td><td>26745377278</td></tr> <tr><td>500</td><td>GILBERTO NEPOMUCENO</td><td>1978-05-06 00:00:00</td><td>46128500591</td></tr> <tr><td>38</td><td>WENDELL CHAVES AGRA</td><td>1958-09-06 00:00:00</td><td>75889838216</td></tr> <tr><td>37</td><td>MARIA LUCIA DE PAIVA</td><td>1969-09-08 00:00:00</td><td>58168237073</td></tr> </tbody> </table>	CodCli	NomeCli	DtNascimento	CPF	39	MARCUS VINICIUS DE CASTRO FERRAZ	1978-01-07 00:00:00	28714439370	389	CICERO SILVA FRANCO	1983-06-07 00:00:00	78015389474	111	ANDRE TAVARES DE OLIVEIRA	1978-02-09 00:00:00	32912111927	16	WILSON PAULINO DE FARIA	1984-09-02 00:00:00	26406316777	86	FLAVIO MACEDO DE GOIS	1978-09-03 00:00:00	47464586179	137	DIEGO GONCALVES DE OLIVEIRA	1984-02-06 00:00:00	45470137269	710	JOSE GERALDO DE FREITAS	1984-08-04 00:00:00	31678710396	722	RAFAEL OLIVEIRA LEITE	1969-08-07 00:00:00	22869722167	331	DENIS DE LIMA GONÇALVES	1969-06-04 00:00:00	61757331137	211	ANA CRISTINA DE PAIVA	1983-06-05 00:00:00	66477211111	377	ODIMAR CLAUDIO DA SIVA DE CARVALHO	1980-06-04 00:00:00	26745377278	500	GILBERTO NEPOMUCENO	1978-05-06 00:00:00	46128500591	38	WENDELL CHAVES AGRA	1958-09-06 00:00:00	75889838216	37	MARIA LUCIA DE PAIVA	1969-09-08 00:00:00	58168237073
CodCli	NomeCli	DtNascimento	CPF																																																										
39	MARCUS VINICIUS DE CASTRO FERRAZ	1978-01-07 00:00:00	28714439370																																																										
389	CICERO SILVA FRANCO	1983-06-07 00:00:00	78015389474																																																										
111	ANDRE TAVARES DE OLIVEIRA	1978-02-09 00:00:00	32912111927																																																										
16	WILSON PAULINO DE FARIA	1984-09-02 00:00:00	26406316777																																																										
86	FLAVIO MACEDO DE GOIS	1978-09-03 00:00:00	47464586179																																																										
137	DIEGO GONCALVES DE OLIVEIRA	1984-02-06 00:00:00	45470137269																																																										
710	JOSE GERALDO DE FREITAS	1984-08-04 00:00:00	31678710396																																																										
722	RAFAEL OLIVEIRA LEITE	1969-08-07 00:00:00	22869722167																																																										
331	DENIS DE LIMA GONÇALVES	1969-06-04 00:00:00	61757331137																																																										
211	ANA CRISTINA DE PAIVA	1983-06-05 00:00:00	66477211111																																																										
377	ODIMAR CLAUDIO DA SIVA DE CARVALHO	1980-06-04 00:00:00	26745377278																																																										
500	GILBERTO NEPOMUCENO	1978-05-06 00:00:00	46128500591																																																										
38	WENDELL CHAVES AGRA	1958-09-06 00:00:00	75889838216																																																										
37	MARIA LUCIA DE PAIVA	1969-09-08 00:00:00	58168237073																																																										

Sub-select com Join

Podemos utilizar o recurso de sub-select para produzir uma tabela virtual e depois utilizar esta tabela virtual em um Join com outras tabelas. Veja o exemplo:

```

1   SELECT      C.CodCli, NomeCli, NumAplicacao, NumFundo, ValorAplicacao
    FROM       aCLIENTE      C
    JOIN
    (
2   SELECT      *      FROM aAPLICACAO
    WHERE      NumFundo      IN
    (
        SELECT      NumFundo      FROM aFUNDOINVESTIMENTO
        WHERE      CodTipoFundo=      4
    )
    )
    )      S
    ON      C.CodCli =      S.CodCli
;

```

O SUB-SELECT destacado em cor azul retorna um conjunto de dados. Este conjunto recebe um ALIAS (apelido), neste caso, **S**. Usamos um SELECT com dados da tabela aCLIENTE e do conjunto de dados retornado pelo SUB-SELECT. Para juntar os dois conjuntos de dados, utilizamos o JOIN/ON da mesma forma que usariamos para fazer JOIN de duas tabelas.

Resultado:

CodCli	NomeCli	NumAplicacao	NumFundo	ValorAplicacao
39	MARCUS VINICIUS DE CASTRO FERRAZ	11	1100	32400.00
389	CICERO SILVA FRANCO	17	1100	16800.00
111	ANDRE TAVARES DE OLIVEIRA	43	1100	15900.00
16	WILSON PAULINO DE FARIA	85	1100	18000.00
86	FLAVIO MACEDO DE GOIS	91	1100	30000.00
137	DIEGO GONCALVES DE OLIVEIRA	110	1100	8900.00
710	JOSE GERALDO DE FREITAS	116	1100	31600.00
722	RAFAEL OLIVEIRA LEITE	106	1200	30800.00
331	DENIS DE LIMA GONÇALVES	44	1300	46800.00
710	JOSE GERALDO DE FREITAS	87	1300	8900.00
86	FLAVIO MACEDO DE GOIS	111	1300	25400.00
211	ANA CRISTINA DE PAIVA	4	1400	35000.00
377	ODIMAR CLAUDIO DA SIVA DE CARVALHO	6	1400	29000.00
86	FLAVIO MACEDO DE GOIS	31	1400	21800.00
710	JOSE GERALDO DE FREITAS	82	1400	46300.00
111	ANDRE TAVARES DE OLIVEIRA	23	1500	18900.00
500	GILBERTO NEPOMUCENO	25	1500	14500.00
38	WENDELL CHAVES AGRA	27	1500	32200.00
37	MARIA LUCIA DE PAIVA	72	1500	28100.00
16	WILSON PAULINO DE FARIA	114	1500	29700.00

Aula 13 – A cláusula CASE dentro de um SELECT

A cláusula CASE é utilizada dentro de um SELECT, na lista de colunas/expressões, no lugar de uma coluna/expressão.

CASE – sintaxe 1	Valor condicional de uma coluna. Interno ao SELECT.
Sintaxe:	<pre> CASE <coluna ou expressão> WHEN <condição1> THEN <valor literal ou expressão1 > WHEN <condição2> THEN < valor literal ou expressão2> : ELSE < valor literal ou expressãoN> END [Label da Coluna] </pre>

Coluna ou expressão	Coluna ou expressão que terá o conteúdo avaliado pelas condições.
CondiçãoX	É o teste de condição do valor (numérico ou string) da coluna/expressão. Indica o valor a ser testado.
Valor literal ou expressãoX	É o valor que a coluna deve assumir se a condição do WHEN for atendida. Se nenhuma das condições do WHEN for atendida, será assumido o valor declarado após o ELSE.
Label da coluna	Toda coluna pode receber um ALIAS (um apelido para ser apresentado no lugar de seu nome original). Como com CASE será produzido um resultado diferente da coluna, é importante indicar aqui um ALIAS.

Para exemplificar esta forma do CASE, vamos utilizar a tabela aFUNDOINVESTIMENTO do DataBase API-CACAO. Para que tenhamos valores adequados ao exemplo, execute o seguinte UPDATE:

UPDATE	aFUNDOINVESTIMENTO		
SET	Ativo_S_N	=	'N'
WHERE	NumFundo	IN	(300,500);

Com este comando mudamos o status de 2 fundos para inativo.

Agora vamos exibir 3 colunas dessa tabela da forma original:

```

SELECT      NumFundo,
            NomeComlFundo,
            Ativo_S_N
FROM        aFUNDOINVESTIMENTO;

```

Resultado (primeiras linhas):

NumFundo	NomeComlFundo	Ativo_S_N
100	BETA FIC Curto Prazo	S
200	BETA FIC Referenciado DI HiperFundo	S
300	BETA FIC Referenciado DI Safira	N
400	BETA FIC Referenciado DI Nikkei	S
500	BETA FIC Referenciado DI Brilhante	N
600	BETA FIC Referenciado DI Topázio	S
700	BETA FIC Referenciado DI Platinum	S
800	BETA FIC Referenciado DI Federal	S

Vamos repetir o SELECT com as mesmas colunas só que utilizando um condicional para tratar o resultado da coluna Ativo_S_N:

```

SELECT      NumFundo,
            NomeComlFundo,
            CASE      Ativo_S_N
                        WHEN  'S'      THEN  'Ativo'
                        WHEN  'N'      THEN  'Inativo'
                        ELSE                'indefinido'
            END        Situação
FROM        aFUNDOINVESTIMENTO;

```

Resultado (primeiras linhas):

NumFundo	NomeComlFundo	Situacao
100	BETA FIC Curto Prazo	Ativo
200	BETA FIC Referenciado DI HiperFundo	Ativo
300	BETA FIC Referenciado DI Safira	Inativo
400	BETA FIC Referenciado DI Nikkei	Ativo
500	BETA FIC Referenciado DI Brilhante	Inativo
600	BETA FIC Referenciado DI Topázio	Ativo
700	BETA FIC Referenciado DI Platinum	Ativo
800	BETA FIC Referenciado DI Federal	Ativo

A 3^a coluna que tinha um *flag* de um byte passa a ser apresentada com um conteúdo tratado a partir das condições indicadas em CASE WHEN. Note que desde o CASE até o END é declarada uma ÚNICA coluna para exibição. O label SITUACAO logo após o END dá o rótulo a ser exibido para esta coluna criada pelo CASE.

CASE – sintaxe 2	Valor condicional de uma coluna. Interno ao SELECT.
Sintaxe:	<pre> CASE WHEN <condição1> THEN <valor literal ou expressão1 > WHEN <condição2> THEN < valor literal ou expressão2> : ELSE < valor literal ou expressãoN> END [Label da Coluna] </pre>

Para exemplificar esta forma do CASE, vamos utilizar a tabela aAPLICACAO do DataBase APPLICACAO. Vamos calcular um valor de comissão de acordo com o valor de cada aplicação, usando as seguintes regras de cálculo:

Valor a Aplicação	Percentual de Comissão
Até 45.000	Zero
> 45.000 e até 100.000	1%
> 100.000	3%

```

SELECT  NumAplicacao,
        CodCli AS Cliente,
        ValorAplicacao,
        CASE
            WHEN ValorAplicacao >      100000
                THEN  ValorAplicacao * 0.03
            WHEN ValorAplicacao <      45000
                THEN  0
            ELSE          ValorAplicacao * 0.01
            END          Situacao
FROM    aAPLICACAO
ORDER BY ValorAplicacao DESC;

```

Resultado (primeiras linhas):

NumAplicacao	Cliente	ValorAplicacao	Situacao
7	278	350000.00	10500.0000
3	111	200000.00	6000.0000
55	37	50100.00	501.0000
58	963	48800.00	488.0000
15	59	48600.00	486.0000
21	73	48400.00	484.0000
39	487	48400.00	484.0000
83	888	48400.00	484.0000
88	562	48400.00	484.0000
60	331	48200.00	482.0000
2	70	48000.00	480.0000
44	331	46800.00	468.0000
64	572	46800.00	468.0000
38	474	46700.00	467.0000
78	37	46300.00	463.0000
82	710	46300.00	463.0000
45	143	45400.00	454.0000
90	137	44100.00	0
42	365	44000.00	0
22	79	43900.00	0
103	520	42300.00	0
12	37	41700.00	0
61	37	41000.00	0

A última coluna é virtual, pois não existe na base de dados.

Ela é criada a partir de regras de cálculo que, por sua vez, dependem das condições que são tratadas pelo CASE.

Aula 14 – Variações da cláusula JOIN

A cláusula JOIN que aprendemos anteriormente possui algumas variações. Ao usarmos o JOIN da forma mais simples que já utilizamos, trazemos como retorno apenas as tuplas (linhas) nas quais a PK de uma tabela e a FK de outra casam-se perfeitamente. Mas o JOIN pode ser utilizado com algumas opções adicionais que apresentam resultados diferentes.

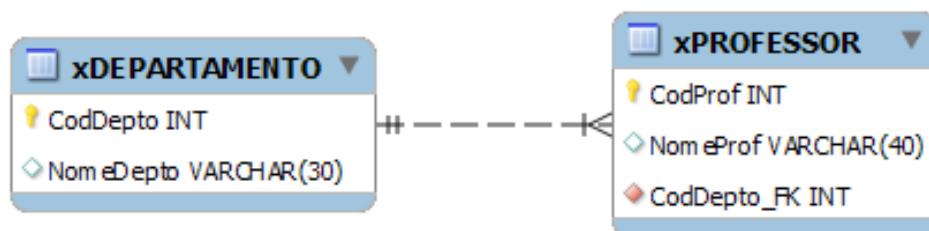
Tipos de JOIN

Os tipos de JOIN são aplicáveis com resultados distintos. Para ilustrar os tipos de JOIN listados a seguir, vamos imaginar que os comandos JOIN sejam executados sobre duas tabelas: tabA e tabB, sendo tabA a tabela “pai” (PK) e tabB a tabela “filha” (FK relacionada à PK de tabA):

INNER JOIN	Retorna às linhas em que haja coincidência exata entre PK de tabA e a FK de tabB. As demais linhas de tabA e tabB não são apresentadas. É o <i>default</i> . Se o JOIN for usado sozinho, o MySQL assume INNER JOIN.
LEFT JOIN	Retorna TODAS as linhas de tabA e as linhas de tabB cuja FK coincidam com a PK de tabA.
RIGHT JOIN	Retorna TODAS as linhas de tabB e as linhas de tabA cuja PK coincidam com a FK de tabB.

Para exemplificarmos, vamos criar duas tabelas novas e incluir alguns dados. Você pode usar o mesmo DataBase que vínhamos usando.

Vamos criar e popular as tabelas abaixo para usar nos exemplos:



Comandos de criação das tabelas:

```

CREATE TABLE xDEPARTAMENTO (
  CodDept INT NOT NULL,
  NomeDept VARCHAR(30) NULL,
  PRIMARY KEY (CodDept)
);

CREATE TABLE xPROFESSOR (
  CodProf INT NOT NULL,
  NomeProf VARCHAR(40) NULL,
  CodDept INT NULL,
  PRIMARY KEY (CodProf),
  FOREIGN KEY (CodDept) REFERENCES xDEPARTAMENTO (CodDept)
);
  
```

Carregando alguns dados:

```

INSERT INTO xDEPARTAMENTO      VALUES (100, 'TI');
INSERT INTO xDEPARTAMENTO      VALUES (200, 'Engenharia');
INSERT INTO xDEPARTAMENTO      VALUES (300, 'Direito');
INSERT INTO xDEPARTAMENTO      VALUES (400, 'Gestão');
INSERT INTO xDEPARTAMENTO      VALUES (500, 'Licenciaturas');

INSERT INTO xPROFESSOR        VALUES (1, 'Tarcisio', 100);
INSERT INTO xPROFESSOR        VALUES (2, 'Jorge' , 200);
INSERT INTO xPROFESSOR        VALUES (3, 'Nagata' , null);
INSERT INTO xPROFESSOR        VALUES (4, 'Hamilton', 100);
INSERT INTO xPROFESSOR        VALUES (5, 'Akamine' , 500);
INSERT INTO xPROFESSOR        VALUES (6, 'Marcos' , null);

```

Vamos, agora, executar as variações de JOIN sobre estas tabelas e observar os resultados:

a. JOIN tradicional que usamos anteriormente:

```

SELECT      *
FROM        xPROFESSOR          P
JOIN xDEPARTAMENTO           D   ON      P.CodDept = D.CodDept;

```

Resultado:

CodProf	NomeProf	CodDept	CodDept	NomeDept
1	Tarcisio	100	100	TI
4	Hamilton	100	100	TI
2	Jorge	200	200	Engenharia
5	Akamine	500	500	Licenciaturas

São retornadas as linhas em que há coincidência na coluna CodDept (FK de professor e PK de departamento).

b. INNER JOIN:

```

SELECT      *
FROM        xPROFESSOR      P
INNER JOIN  xDEPARTAMENTO    D
ON          P.CodDept       =       D.CodDept;

```

Resultado:

CodProf	NomeProf	CodDept	CodDept	NomeDept
1	Tarcisio	100	100	TI
4	Hamilton	100	100	TI
2	Jorge	200	200	Engenharia
5	Akamine	500	500	Licenciaturas

Idêntico ao anterior. Isso porque quando usamos o JOIN sem indicação do seu tipo é assumido o *default INNER*.

c. LEFT JOIN:

```

SELECT      *
FROM        xPROFESSOR      P
LEFT JOIN   xDEPARTAMENTO    D
ON          P.CodDept       =       D.CodDept;

```

Resultado:

CodProf	NomeProf	CodDept	CodDept	NomeDept
1	Tarcisio	100	100	TI
4	Hamilton	100	100	TI
2	Jorge	200	200	Engenharia
5	Akamine	500	500	Licenciaturas
3	Nagata	NULL	NULL	NULL
6	Marcos	NULL	NULL	NULL

Com LEFT JOIN, são apresentadas TODAS as linhas da tabela mais à esquerda (primeira tabela declarada), no caso xPROFESSOR (3 primeiras colunas).

São apresentadas também as linhas da segunda tabela (xDEPARTAMENTO) quando houver coincidência de PK com FK. Quando NÃO houver coincidência as colunas da segunda tabela são apresentadas com NULL (valor nulo).

d. RIGHT JOIN:

```

SELECT      *
FROM        xPROFESSOR      P
RIGHT JOIN  xDEPARTAMENTO    D
ON          P.CodDept       =       D.CodDept;

```

Resultado:

CodProf	NomeProf	CodDept	CodDept	NomeDept
1	Tarcisio	100	100	TI
4	Hamilton	100	100	TI
2	Jorge	200	200	Engenharia
NULL	NULL	NULL	300	Direito
NULL	NULL	NULL	400	Gestão
5	Akamine	500	500	Licenciaturas

Com RIGHT JOIN o efeito é invertido: são apresentadas TODAS as linhas da tabela mais à direita (segunda tabela declarada), no caso xDEPARTAMENTO (2 últimas colunas).

São apresentadas também as linhas da primeira tabela (xPROFESSOR) quando houver coincidência de PK com FK. Quando NÃO houver coincidência, as colunas da primeira tabela são apresentadas com NULL (valor nulo).

e. LEFT JOIN com RIGHT JOIN:

```

SELECT      *
FROM        xPROFESSOR      P
LEFT JOIN   xDEPARTAMENTO    D
ON          P.CodDept       =       D.CodDept
UNION
SELECT      *
FROM        xPROFESSOR      P
RIGHT JOIN  xDEPARTAMENTO    D
ON          P.CodDept       =       D.CodDept;

```

Resultado:

CodProf	NomeProf	CodDept	CodDept	NomeDept
1	Tarcisio	100	100	TI
4	Hamilton	100	100	TI
2	Jorge	200	200	Engenharia
5	Akamine	500	500	Licenciaturas
3	Nagata	NULL	NULL	NULL
6	Marcos	NULL	NULL	NULL
NULL	NULL	NULL	300	Direito
NULL	NULL	NULL	400	Gestão

O MySQL não tem o recurso do “FULL JOIN” existente em outros SGBDs. Mas conseguimos produzir o mesmo efeito usando 2 SELECTs (com RIGHT e com LEFT) e juntando os resultados com **UNION**.

Neste caso, o MySQL trará todas as linhas das duas tabelas. Fará o emparelhamento das linhas das duas tabelas em que há coincidência de PK com FK. Quando NÃO houver coincidência para algum dos lados, as colunas sem correspondente serão preenchidas com o valor NULL.

Aula 15 – Alguns casos especiais de modelagem e DML

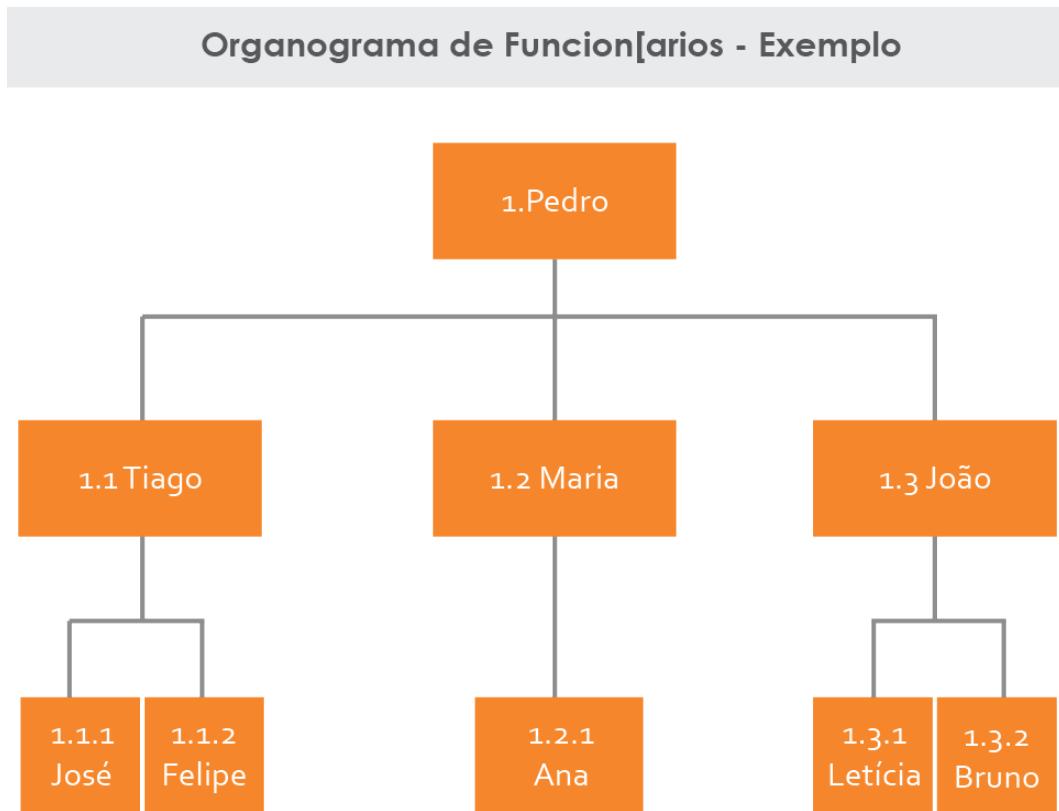
Vamos discutir alguns casos especiais de modelagem de dados como tratá-los com comandos DML, pois você poderá se deparar com alguns deles em sua vida profissional.

Autorrelacionamento

Ao preparar o modelo de dados, projetando as tabelas do banco de dados, podemos nos deparar com a necessidade de relacionar uma tabela com ela mesma.

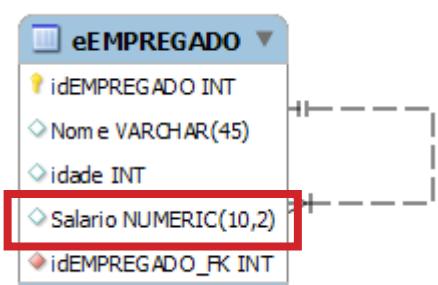
Um caso típico é quando preparamos um cadastro de empregados de uma empresa e tentamos acomodar também nos dados a hierarquia.

Vamos supor que uma empresa tenha o seguinte organograma de funcionários:



Fonte: elaborado pelo autor.

Para armazenar as informações de empregados e a posição hierárquica, indicando, para cada empregado, qual seu superior imediato, podemos preparar o modelo de dados da seguinte forma:



Há aqui uma particularidade, pois eEMPREGADO é, ao mesmo tempo, a "tabela pai" e a "tabela filha" no relacionamento. Chamamos isto de autorrelacionamento.

Veja que montamos como se fosse um relacionamento 1 x N, pois um superior pode ter vários subordinados.

Note que a PK desta tabela é "exportada" como FK para ela mesma, criando um relacionamento. Vamos criar esta tabela no MySQL:

```

CREATE TABLE eEMPREGADO (
    idEMPREGADO      INT          NOT NULL,
    Nome              VARCHAR(45)   NULL,
    idade             INT          NULL,
    Salario           DECIMAL(10,2) NULL,
    idEMPREGADO_FK   INT          NULL,
    PRIMARY KEY (idEMPREGADO),
    FOREIGN KEY (idEMPREGADO_FK)
        REFERENCES eEMPREGADO (idEMPREGADO)
);

```

Para armazenar os dados nesta tabela, usariamos os seguintes comandos MySQL:

```

INSERT INTO eEMPREGADO VALUES ( 1, 'Pedro' , 53, 25000, null);

INSERT INTO eEMPREGADO VALUES ( 11, 'Tiago' , 47, 18000, 1);

INSERT INTO eEMPREGADO VALUES (111, 'José' , 35, 12000, 11);

INSERT INTO eEMPREGADO VALUES (112, 'Felipe' , 29, 6500, 11);

INSERT INTO eEMPREGADO VALUES ( 12, 'Maria' , 41, 16500, 1);

INSERT INTO eEMPREGADO VALUES (121, 'Ana' , 30, 10000, 12);

INSERT INTO eEMPREGADO VALUES ( 13, 'João' , 45, 19000, 1);

INSERT INTO eEMPREGADO VALUES (131, 'Letícia', 19, 7500, 13);

INSERT INTO eEMPREGADO VALUES (132, 'Bruno' , 26, 8900, 13);

```

Eis os dados armazenados:

idEMPREGADO	Nome	idade	Salario	idEMPREGADO_FK
1	Pedro	53	25000.00	NULL
11	Tiago	47	18000.00	1
12	Maria	41	16500.00	1
13	João	45	19000.00	1
111	José	35	12000.00	11
112	Felipe	29	6500.00	11
121	Ana	30	10000.00	12
131	Letícia	19	7500.00	13
132	Bruno	26	8900.00	13

Os empregados e a hierarquia está armazenada nos dados da tabela eEMPREGADO.

Agora, para exibirmos os empregados e o nome de seu superior imediato, é necessário fazer um SELECT contendo um JOIN desta tabela com ela mesma:

```

SELECT      E.idEmpregado,
            E.Nome,
            E.Idade,
            E.Salario,
            S.Nome      SuperiorImediato
FROM        eEMPREGADO    E
LEFT JOIN    eEMPREGADO    S
ON          E.idEMPREGADO_FK = S.idEMPREGADO;

```

Resultado:

idEmpregado	Nome	Idade	Salario	SuperiorImediato
1	Pedro	53	25000.00	NULL
11	Tiago	47	18000.00	Pedro
12	Maria	41	16500.00	Pedro
13	João	45	19000.00	Pedro
111	José	35	12000.00	Tiago
112	Felipe	29	6500.00	Tiago
121	Ana	30	10000.00	Maria
131	Letícia	19	7500.00	João
132	Bruno	26	8900.00	João

Para efetuar este JOIN, utilizamos a tabela eEMPREGADO duas vezes. Aqui é essencial a utilização do ALIAS para indicar com que significado estamos pesquisando a tabela, se como empregado ou como superior imediato.

O ALIAS "E" foi usado no exemplo para indicar a linha em que está um empregado e o ALIAS "S" foi usado para indicar a linha em que está seu superior imediato.

Desta forma, com o JOIN, emparelhamos duas linhas da mesma tabela, em posições diferentes (linhas diferentes).

Note que para aparecer os dados do empregado com mais alto nível hierárquico é necessário a utilização de LEFT JOIN, pois ele não tem superior correspondente.

Para podermos emparelhar as linhas do Empregado com seu Superior, montamos o JOIN usando a FK do Empregado (como se fosse "tabela filha") com a PK do superior imediato (como se fosse "tabela pai").

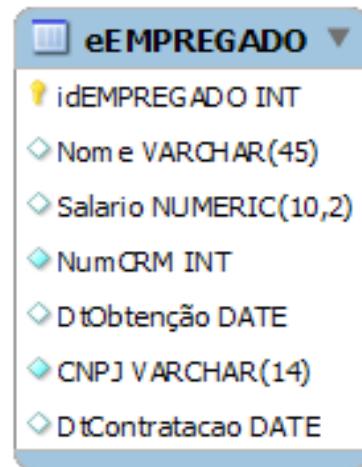


Especialização

Há algumas tabelas que gostaríamos de armazenar ou não algumas informações dependendo do caso.

Vamos supor, como exemplo, o cadastro de empregados de um hospital. Pode haver médicos contratados, neste caso é necessário informar seu CRM e data de obtenção do CRM. Pode haver médicos terceirizados como pessoa jurídica. Neste caso, é necessário indicar o CNPJ deste empregado e a data de contratação deste PJ. E pode haver, ainda, empregados que não são médicos e não possuem nenhuma dessas informações adicionais.

O modelo descrito no desenho abaixo poderia atender essa demanda:

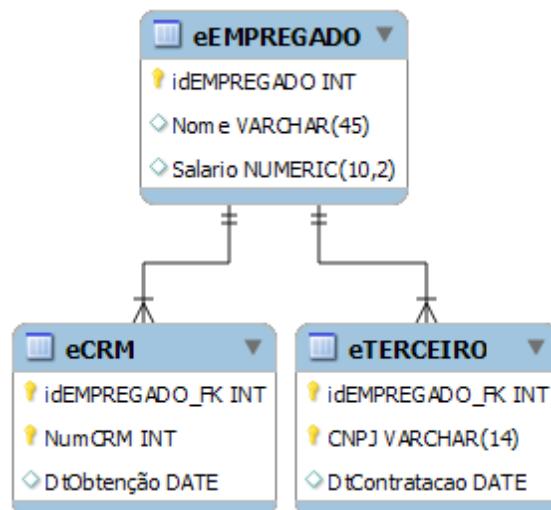


Uma forma é criar a tabela com todos os dados possíveis.

No caso de um médico, as duas últimas colunas ficariam sempre vazias. Se for uma pessoa jurídica, as colunas CRM e DtObtencao ficariam vazias.

Se for outro tipo de funcionário, as quatro últimas colunas ficariam vazias.

Não há problema prático com isso, mas tornamos a estrutura da tabela mais complexa e desperdiçamos espaço no DataBase.



Uma forma mais otimizada de construir este modelo seria o desenho apresentado à direita.

A estratégia aqui é colocar na tabela eEMPREGADO os dados comuns a todos os empregados. As ou-

tras tabelas (CRM e TERCEIRO) só seriam preenchidas quando for o caso e só com os dados de médico ou de terceiro.

Um empregado de outro tipo não teria nenhuma dessas tabelas adicionais preenchidas.

Criação das tabelas:

```
-- Table eEMPREGADO
CREATE TABLE      eEMPREGADO  (
    idEMPREGADO      INT          NOT NULL,
    Nome              VARCHAR(45)   NULL,
    Salario            DECIMAL(10,2)  NULL,
    PRIMARY KEY (idEMPREGADO)
);

-- Table eCRM
CREATE TABLE      eCRM (
    idEMPREGADO_FK  INT          NOT NULL,
    NumCRM            INT          NOT NULL,
    DtObtenção        DATE         NULL,
    PRIMARY KEY (idEMPREGADO_FK, NumCRM),
    FOREIGN KEY (idEMPREGADO_FK)
        REFERENCES eEMPREGADO (idEMPREGADO)
);

-- Table eTERCEIRO
CREATE TABLE      eTERCEIRO (
    idEMPREGADO_FK  INT          NOT NULL,
    CNPJ               VARCHAR(14)  NOT NULL,
    DtContratacao      DATE         NULL,
    PRIMARY KEY (idEMPREGADO_FK, CNPJ),
    FOREIGN KEY (idEMPREGADO_FK)
        REFERENCES eEMPREGADO (idEMPREGADO)
);
```

Vamos supor um conjunto de dados para incluir nestas tabelas:

idEMPREGADO	Nome	Salario	NumCRM	DtObtencao	CNPJ	DtContratacao	Tipo de Funcionário
1	José	3.500,00					outro tipo
2	Maria	5.000,00					outro tipo
3	João	6.500,00					outro tipo
4	Gabriela	7.000,00	72.401	10/01/99			médico
5	Cristine	9.000,00	70.418	15/12/94			médico
6	Wagner	8.000,00	55.715	20/12/84			médico
7	Karina	4.500,00			99999999000122	20/05/13	Pessoa Jurídica
8	Marcelo	9.000,00			12345678000155	15/01/05	Pessoa Jurídica

Os comandos INSERT abaixo seriam utilizados para incluir esses dados. Observe que para médicos e pessoas jurídicas são necessários 2 INSERTs para inclusão dos dados:

```

INSERT INTO eEMPREGADO    VALUES (1, 'José' , 3500);
INSERT INTO eEMPREGADO    VALUES (2, 'Maria' , 5000);
INSERT INTO eEMPREGADO    VALUES (3, 'João' , 6500);

INSERT INTO eEMPREGADO    VALUES (4, 'Gabriela', 7000);
INSERT INTO eCRM           VALUES (4, 72401, '1999-01-10');

INSERT INTO eEMPREGADO    VALUES (5, 'Cristine', 9000);
INSERT INTO eCRM           VALUES (5, 70418, '1994-12-15');

INSERT INTO eEMPREGADO    VALUES (6, 'Wagner' , 8000);
INSERT INTO eCRM           VALUES (6, 55715, '1984-01-20');

INSERT INTO eEMPREGADO    VALUES (7, 'Karina' , 4500);
INSERT INTO eTERCEIRO      VALUES (7, 99999999000122, '2013-05-20');

INSERT INTO eEMPREGADO    VALUES (8, 'Marcelo' , 9000);
INSERT INTO eTERCEIRO      VALUES (8, 12345678000155, '2005-01-15');

```



Para exibir os dados dos empregados, fazemos JOIN das 3 tabelas. Devemos usar LEFT JOIN na tabela eEMPREGADO para que sejam exibidos todos os empregados, tendo ou não os dados adicionais:

```

SELECT      *
FROM        eEMPREGADO    E
LEFT JOIN   eCRM          C
ON          E.idEMPREGADO = C.idEMPREGADO_FK
LEFT JOIN   eTERCEIRO     T
ON          E.idEMPREGADO = T.idEMPREGADO_FK
ORDER BY    E.idEMPREGADO;
  
```

Veja que é importante utilizar o LEFT JOIN para que todos os empregados sejam exibidos. A tabela eEMPREGADO está relacionada às tabelas eCRM e eTERCEIRO.

Veja a composição dos dados que são originados a partir de 3 tabelas:

			eEMPREGADO			eCRM			eTERCEIRO		
idEMPREGADO	Nome	Salario									
1	José	3500.00		NULL	NULL	NULL		NULL	NULL	NULL	
2	Maria	5000.00		NULL	NULL	NULL		NULL	NULL	NULL	
3	João	6500.00		NULL	NULL	NULL		NULL	NULL	NULL	
4	Gabriela	7000.00		4	72401	1999-01-10		NULL	NULL	NULL	
5	Cristine	9000.00		5	70418	1994-12-15		NULL	NULL	NULL	
6	Wagner	8000.00		6	55715	1984-01-20		NULL	NULL	NULL	
7	Karina	4500.00		NULL	NULL	NULL		7	99999999000122	2013-05-20	
8	Marcelo	9000.00		NULL	NULL	NULL		8	12345678000155	2005-01-15	

Todas as linhas estão preenchidas para a tabela eEMPREGADO.

Na tabela eCRM apenas os médicos têm os dados preenchidos.

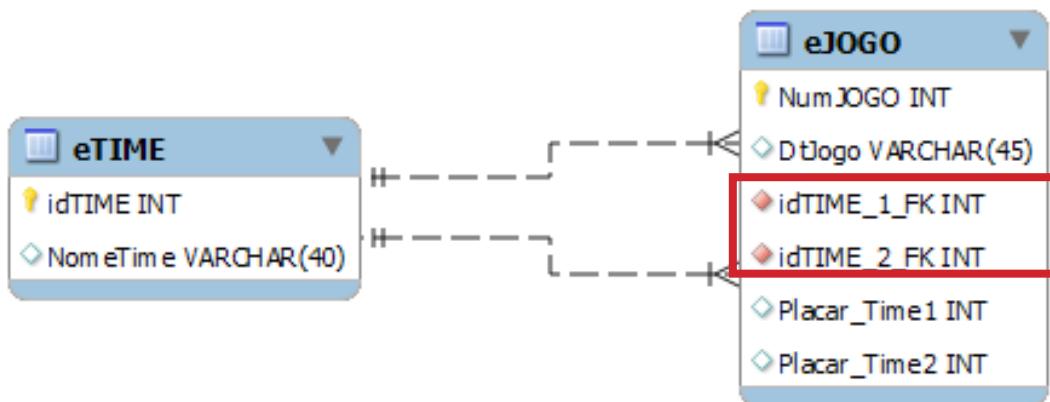
Na tabela eTERCEIRO apenas as pessoas jurídicas têm os dados preenchidos.

Duplo relacionamento

Há casos em que uma tabela se relaciona com outra tabela mais de uma vez. Vamos ver um caso:

Suponha que temos uma tabela com times de futebol e estejamos criando outra tabela para armazenar os dados dos jogos entre esses times. Na tabela de jogos, a cada jogo, sempre haverá dois times envolvidos. Como os times estão cadastrados na tabela de TIMES é necessário criar dois relacionamentos entre essas tabelas.

O modelo de dados resultante ficaria assim:



Veja que a tabela **eJOGO** recebe duas FKs relacionadas à PK de **eTIME**. Como as FKs são colunas de **eJOGO**, é claro que precisam ter nomes distintos. O MySQL WorkBench cria uma FK com nome distinto a cada relacionamento criado. Você pode renomear as colunas FK deixando-as com nomes adequados (por exemplo, **Time1** e **Time2**).

Criando as tabelas no DataBase:

```
-- Table eTIME

CREATE TABLE eTIME (
    idTIME INT NOT NULL,
    NomeTime VARCHAR(40) NULL,
    PRIMARY KEY (idTIME)
);

-- Table eJOGO

CREATE TABLE eJOGO (
    NumJOGO INT NOT NULL,
    DtJogo VARCHAR(45) NULL,
    idTIME_1_FK INT NOT NULL,
    idTIME_2_FK INT NOT NULL,
    Placar_Time1 INT NULL,
    Placar_Time2 INT NULL,
    PRIMARY KEY (NumJOGO),
    FOREIGN KEY (idTIME_1_FK) REFERENCES eTIME (idTIME),
    FOREIGN KEY (idTIME_2_FK) REFERENCES eTIME (idTIME)
);
```

A tabela eJOGO fica com 2 FKs, ambas apontadas para a PK de eTIME.

Dados para incluir nas tabelas:

eTIME		eJOGO					
idTIME	NomeTime	NumJogo	DtJogo	idTIME_1	idTIME_2	Placar_Time1	Placar_Time2
1	Flamengo	1	20/04/14	7	2	0	0
2	Corinthians	2	20/04/14	3	9	3	0
3	São Paulo	3	20/04/14	5	6	1	1
4	Palmeiras	4	20/04/14	10	4	3	2
5	Vasco da Gama	5	26/04/14	8	5	2	2
6	Internacional	6	26/04/14	6	3	1	1
7	Atlético-MG	7	26/04/14	2	9	3	1
8	Grêmio	8	04/05/14	1	4	4	2
9	Botafogo	9	04/05/14	10	3	2	2
10	Bahia	10	04/05/14	6	7	2	1

A seguir os INSERTs para carregar esses dados para as tabelas:

- TIMES:

```
INSERT INTO eTIME VALUES ( 1, 'Flamengo');

INSERT INTO eTIME VALUES ( 2, 'Corinthians');

INSERT INTO eTIME VALUES ( 3, 'São Paulo');

INSERT INTO eTIME VALUES ( 4, 'Palmeiras');

INSERT INTO eTIME VALUES ( 5, 'Vasco da Gama');

INSERT INTO eTIME VALUES ( 6, 'Internacional');

INSERT INTO eTIME VALUES ( 7, 'Atlético-MG');

INSERT INTO eTIME VALUES ( 8, 'Grêmio');

INSERT INTO eTIME VALUES ( 9, 'Botafogo');

INSERT INTO eTIME VALUES (10, 'Bahia');
```

- JOGOS:

```
INSERT INTO eJOGO VALUES ( 1, '2014-04-20', 7, 2, 0, 0);

INSERT INTO eJOGO VALUES ( 2, '2014-04-20', 3, 9, 3, 0);

INSERT INTO eJOGO VALUES ( 3, '2014-04-20', 5, 6, 1, 1);

INSERT INTO eJOGO VALUES ( 4, '2014-04-20', 10, 4, 3, 2);

INSERT INTO eJOGO VALUES ( 5, '2014-04-26', 8, 5, 2, 2);

INSERT INTO eJOGO VALUES ( 6, '2014-04-26', 6, 3, 1, 1);

INSERT INTO eJOGO VALUES ( 7, '2014-04-26', 2, 9, 3, 1);

INSERT INTO eJOGO VALUES ( 8, '2014-05-04', 1, 4, 4, 2);

INSERT INTO eJOGO VALUES ( 9, '2014-05-04', 10, 3, 2, 2);

INSERT INTO eJOGO VALUES (10, '2014-05-04', 6, 7, 2, 1);
```

Vamos agora exibir os jogos com os nomes dos times e os placares:

```

SELECT      NumJogo,
            DtJogo,
            T1.NomeTime AS Time1,
            Placar_Time1,
            ' x ' AS 'x',
            Placar_Time2,
            T2.NomeTime AS Time2
FROM        eJOGO J
JOINeTIME T1    ON    J.idTIME_1_FK = T1.idTIME
JOINeTIME T2    ON    J.idTIME_2_FK = T2.idTIME;

```

Note que a tabela de TIMES está em 2 JOINS com a tabela de JOGOS para que os nomes de ambos os times possam ser apresentados.

Novamente os ALIAS são fundamentais para identificar os dois usos da tabela de TIMES.

Quando o JOIN for efetuado, o conjunto de dados resultante (não o exibido) terá uma linha da tabela de JOGOS, uma linha da tabela de TIMES (apontando para o Time1) e outra linha diferente da tabela de TIMES (apontando para o Time2).

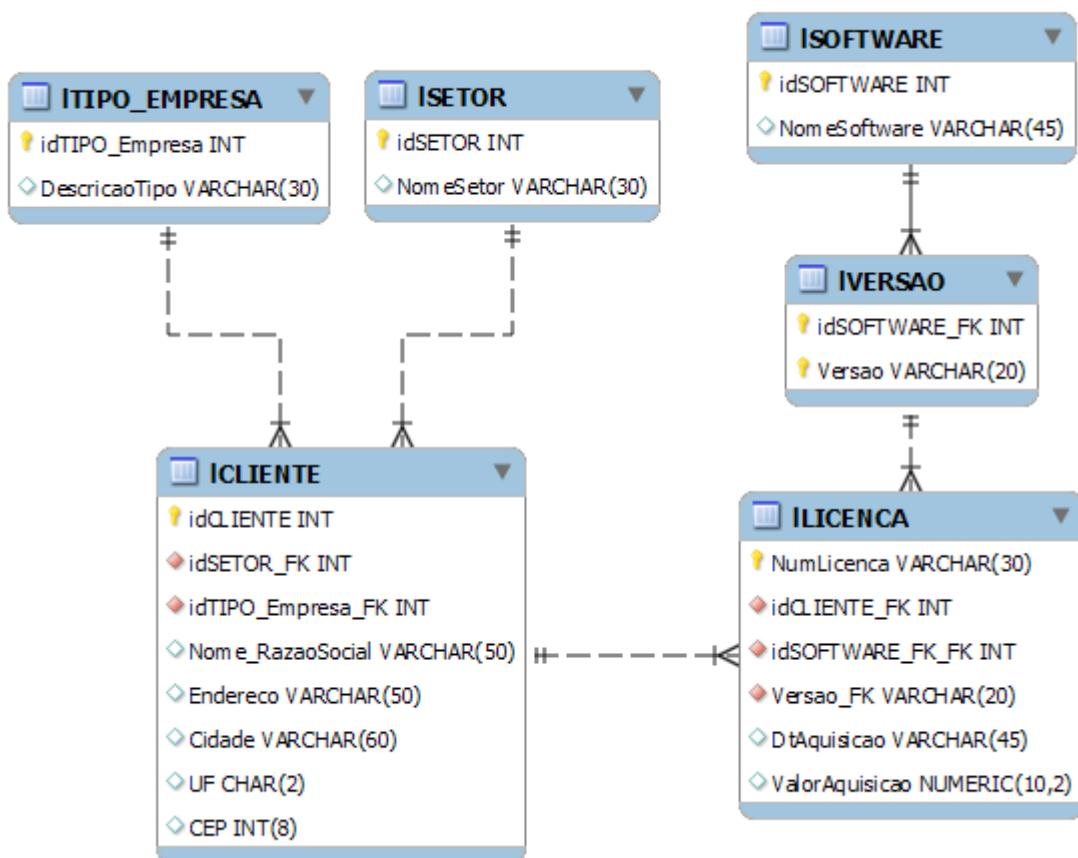
Resultado do SELECT / JOIN:

NumJogo	DtJogo	Time1	Placar_Time1	x	Placar_Time2	Time2
1	2014-04-20	Atlético-MG	0	x	0	Corinthians
2	2014-04-20	São Paulo	3	x	0	Botafogo
3	2014-04-20	Vasco da Gama	1	x	1	Internacional
4	2014-04-20	Bahia	3	x	2	Palmeiras
5	2014-04-26	Grêmio	2	x	2	Vasco da Gama
6	2014-04-26	Internacional	1	x	1	São Paulo
7	2014-04-26	Corinthians	3	x	1	Botafogo
8	2014-05-04	Flamengo	4	x	2	Palmeiras
9	2014-05-04	Bahia	2	x	2	São Paulo
10	2014-05-04	Internacional	2	x	1	Atlético-MG

Exercícios do Módulo 5

Para os exercícios deste módulo, você pode criar o mesmo banco de dados Módulo 3 já criado anteriormente.

- 1) Utilizando o Modelo de Dados abaixo e as tabelas já criadas e populadas anteriormente, responda com os comandos adequados:**



SUB-SELECT / CASE / JOINS

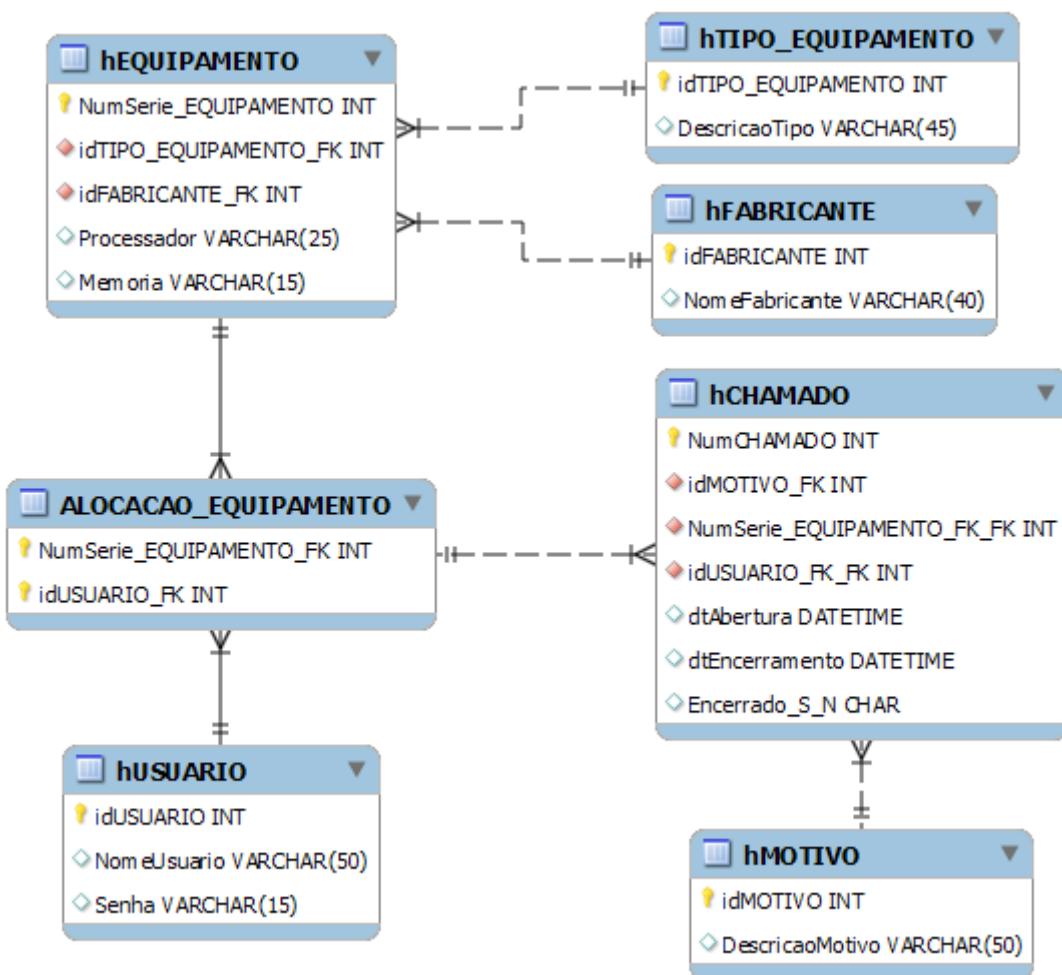
- Liste os nomes dos clientes que já compraram ao menos uma licença de Software;
- Liste os nomes dos clientes que NÃO compraram nenhuma licença de Software;
- Exiba todos os clientes com seu respectivo setor da economia e com seu tipo de empresa. Exiba todas as linhas dessas 3 tabelas, mesmo que não haja correspondente de PK / FK.
- Exiba o software / versão que teve o maior número de licenças adquiridas.
- Exiba a licença que foi vendida pelo maior valor.
- Para cada licença vendida, exiba: o número da Licença, a data da aquisição, o trimestre da aquisição e o valor da aquisição. Ordene por ano e trimestre.
obs.: Jan/Fev/Mar = 1oTri, ABr/Mai/Jun = 2oTri, ...

g) Liste os clientes com código, nome, cidade, UF e região.

A região deve ser considerada:

- SUL - RS, SC, PR;
- SUDESTE – SP, RJ, MG, ES;
- CENTRO – GO, DF, MS, MT;
- NORTE/NORDESTE – todos as demais UFs.

Ordene por região, UF, cidade, nome cliente.



2) Para o modelo de dados acima que você criou e populou as tabelas anteriormente, responda à questões abaixo com os comandos adequados.

- Exiba TODOS os Motivos cadastrados. Quando possível, exiba os dados dos chamados de cada motivo (número e data). Ordene por motivos.
- Exiba todos os fabricantes e seus equipamentos. Exiba mesmo os Fabricantes sem equipamento associado.

Respostas Comentadas dos Exercícios

Caro aluno, segue abaixo as respostas comentadas das listas de exercícios dos módulos desta disciplina.

Recomendo que você tente primeiro fazer sozinho as tarefas e use os chats e fóruns para tirar suas dúvidas. Depois você pode conferir as respostas, ok?

Módulo 1

- 1) C
- 2) D
- 3) C
- 4)

(F) Quando um Banco de Dados se relaciona com outro Banco de Dados nós podemos chamá-lo de Banco de Dados Relacional.

(V) Uma das vantagens de se usar Banco de Dados é que se pode evitar mais facilmente redundâncias.

(V) O uso de Banco de Dados simplifica o trabalho dos sistemas aplicativos.

(F) Usar Banco de Dados é importante mas diminui a segurança dos dados.

(V) Uma grande vantagem do Banco de Dados é a facilidade de se compartilhar dados entre aplicações.

- 5) tabelas / colunas / linhas:

- 6)

(V) escreve-se menos, sem a necessidade de se repetir informações, reduzindo a redundância.

(V) integridade: centralizando a informação em uma tabela e referenciando-se pelo código em outras, garante-se que a informação estará igual em todas as ocorrências o que poderia não acontecer se fosse necessário digitá-la várias vezes (erros de digitação).

(F) Segregando-se as informações em mais tabelas, aumentamos consideravelmente o espaço necessário para se armazenar a mesma informação.

- 7) Relacione as colunas:

- | | | |
|-------|-------|--|
| (1) | SQL | (2) Utilizada para a criação e manutenção de tabelas e outros objetos no Banco de Dados. |
| (2) | DDL | (3) Utilizada para manipular dados: incluir, alterar, excluir e consultar dados. |
| (3) | DML | (1) Structured Query Language, ou linguagem estruturada de pesquisas. Tem um padrão internacional de comandos. |
| (4) | MySQL | (4) É um dos SGBD mais modernos e mais utilizado atualmente no mundo. |

- 8) Relacione os principais comandos de Banco de Dados com sua função:

- (1) INSERT (4) Consulta dados já armazenados em tabelas. É o comando mais utilizado.
- (2) UPDATE (3) Exclui dados já armazenados um uma tabela do Banco de Dados.
- (3) DELETE (1) Inclui dados em uma Tabela do Banco de Dados.
- (4) SELECT (2) Altera dados já armazenados um uma Tabela do Banco de Dados

Módulo 2

- 1) E
2) E
3) D

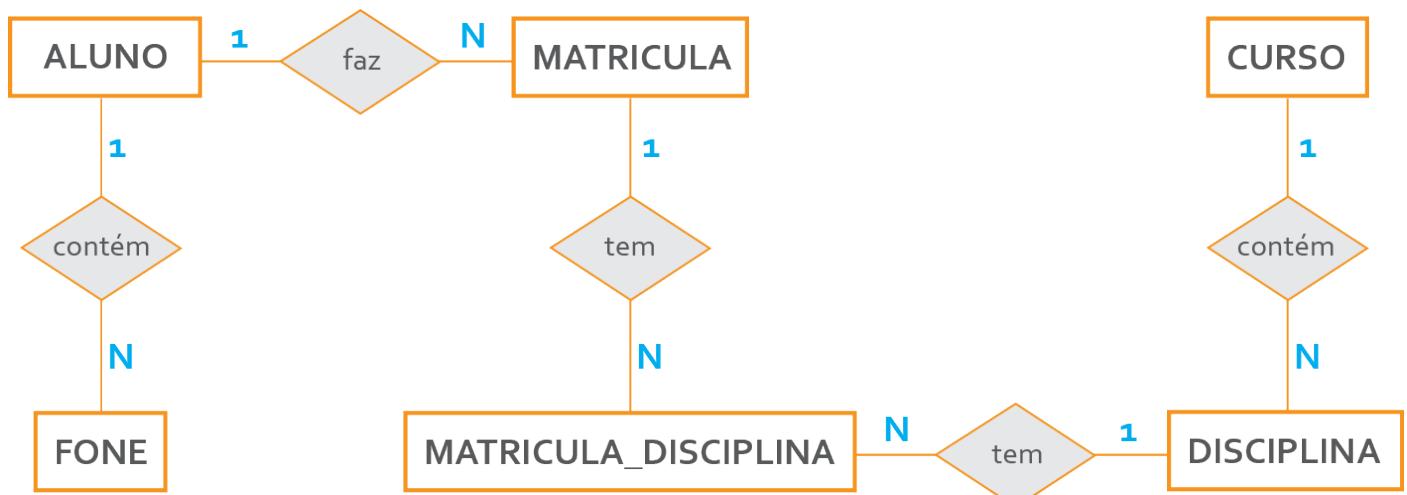
4) Quais são os tipos de visibilidade de um atributo?

- (F) Uma PK representa uma estrutura externa à tabela que a identifica de forma única, não admitindo repetição.
- (F) Uma FK é uma Chave Primária, que é composta por um ou mais colunas da tabela e que a identifica de forma única.
- (V) Uma FK é composta por uma ou mais colunas, trazidas de outra tabela quando da criação de um relacionamento entre as duas tabelas. A(s) coluna(s) trazida(s) representam a Chave Primária da outra tabela.
- (F) Uma PK não existe se não tiver uma FK associada.
- (V) Uma FK é uma Chave Primária, que é composta por um ou mais colunas da tabela e que a identifica de forma única.

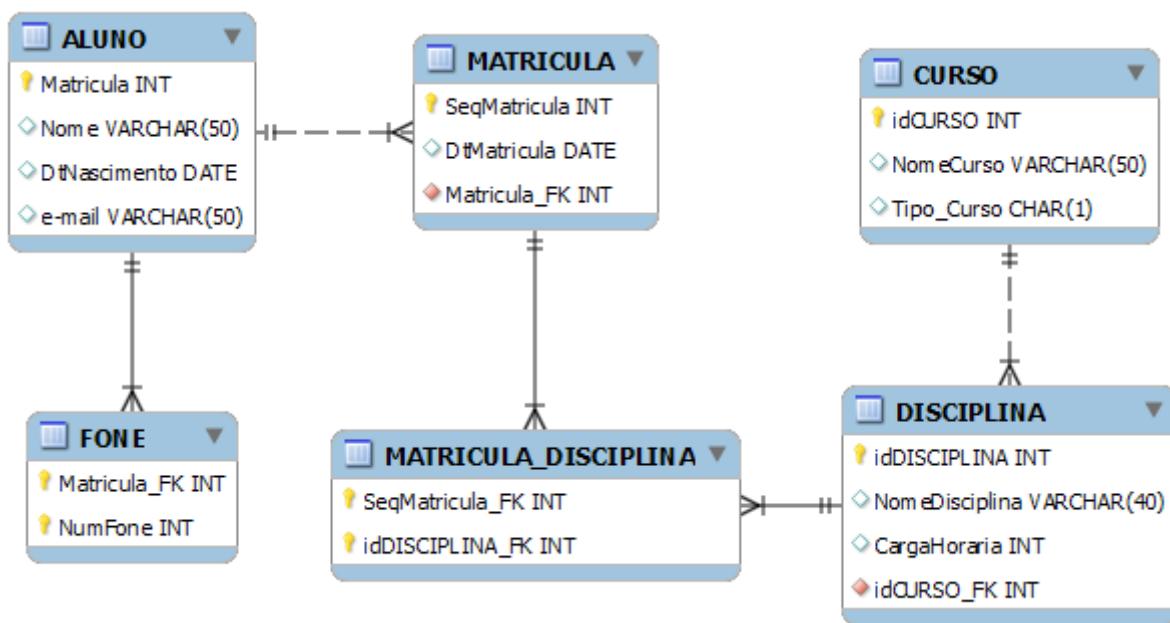
- 5) D
6) E

7) Para a descrição apresentada, são os seguintes DER e Modelo de Dados:

DER



Modelo de Dados

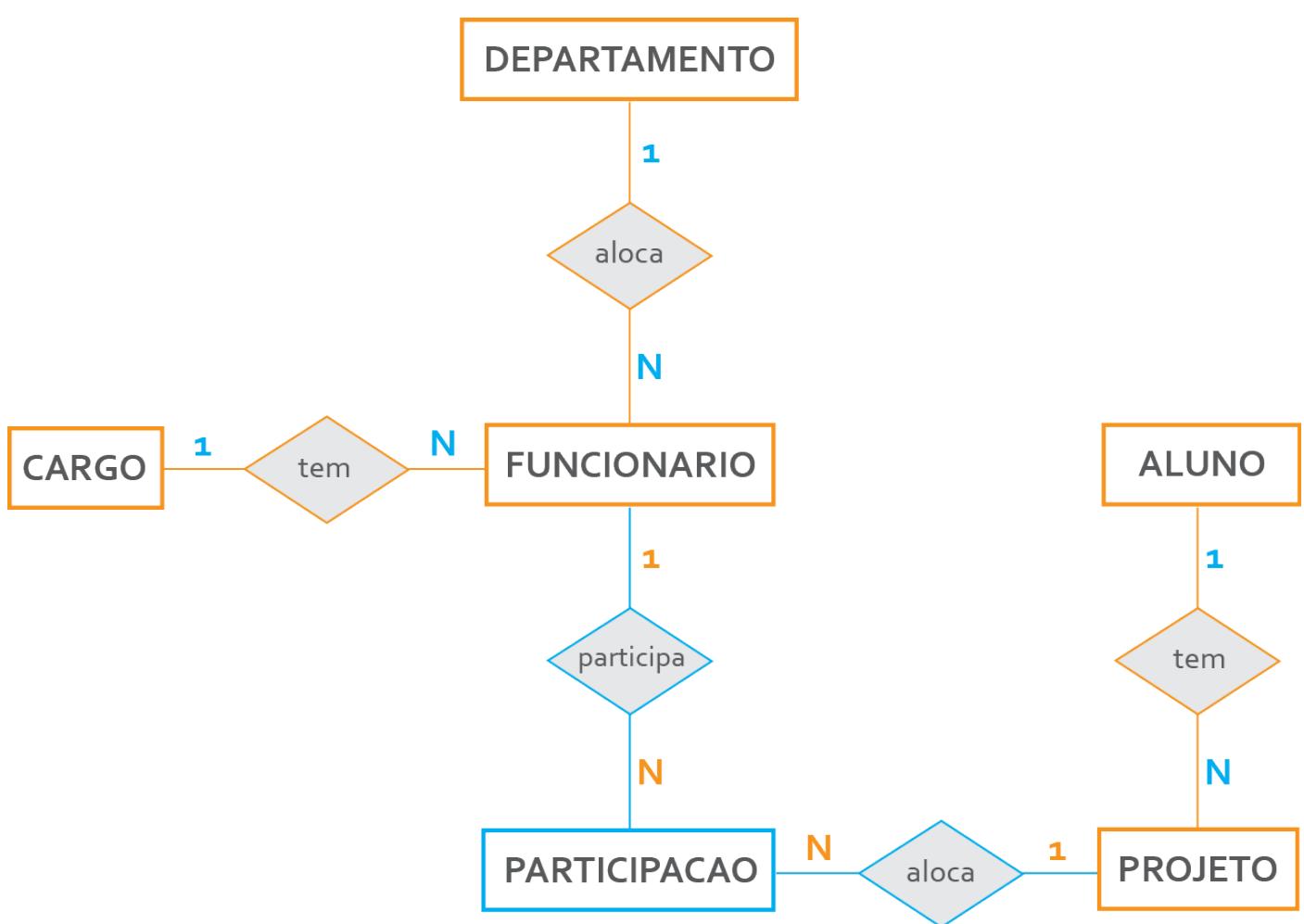


Comentários e dicas:

- quando modelamos um aluno, ele pode ter vários fones. Se deixássemos em uma única tabela haveria uma coluna multivvalorada (Fone1, Fone2, ...). Então separamos os fones em uma tabela à parte;
- Disciplina fica subordinada à tabela de Curso;
- Um aluno pode se matricular várias vezes, por isso a relação ALUNO x MATRICULA é 1 para N;
- Uma matrícula tem várias disciplinas e uma disciplina pode estar em várias matrículas. Isso daria uma relação N para N. Para resolver isto, criamos uma tabela de ligação entre as duas tabelas e “quebramos” o relacionamento N para N em dois relacionamentos 1 para N.

8) Para a descrição apresentada, são os seguintes DER e Modelo de Dados:

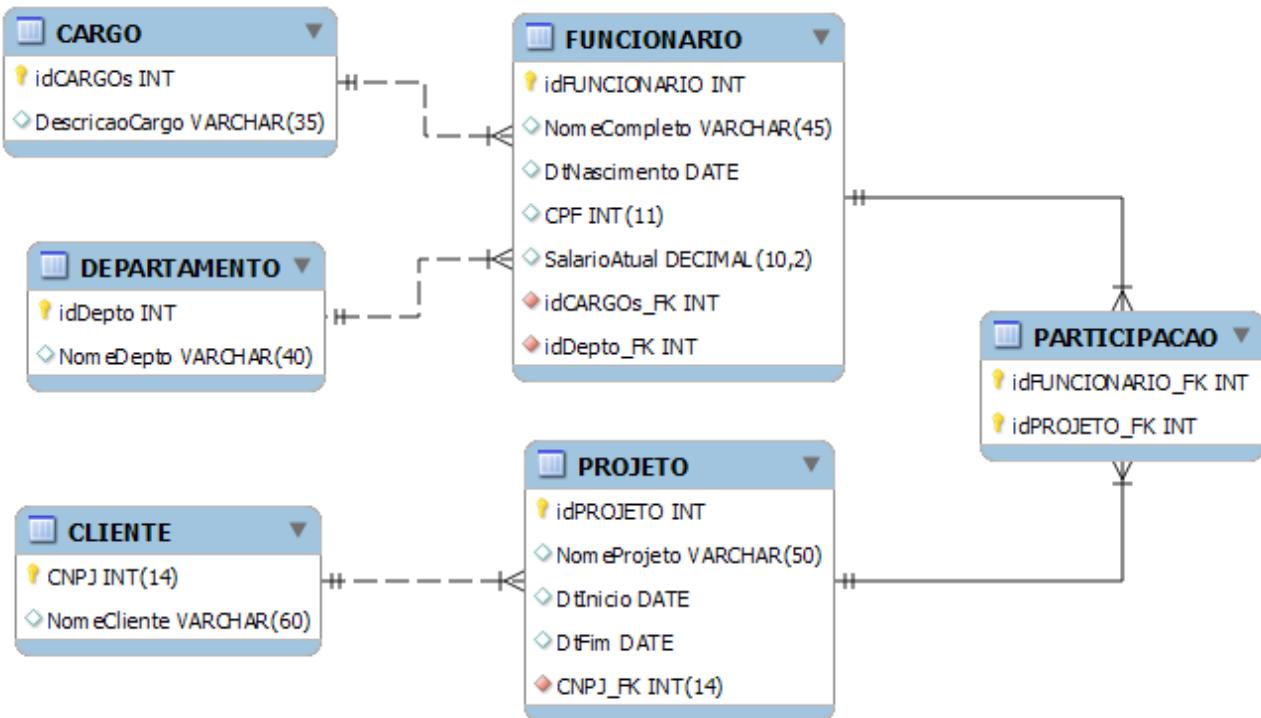
DER



Comentários e dicas:

- Funcionário está ligado a um departamento e a um cargo. Naturalmente um funcionário só pode estar ligado a um Cargo e estar alocado apenas em um Departamento. Mas Cargo e Departamento podem estar relacionados a vários Funcionários. Por isso, estes relacionamentos são 1 x N.;
- Um cliente pode ter vários projetos mas um Projeto só pode ser de um Cliente, Relação 1 x N de Cliente par Projeto;
- Os relacionamentos destacados em vermelho indicam um relacionamento N X N que foi “quebrado” em dois relacionamentos 1 x N. Um Funcionário pode participar de vários projetos. E um projeto pode ter a participação de vários funcionários, o que caracterizaria um relacionamento N x N. Para resolver isto, foi criada uma nova Tabela, chamada tabela que é filha dessas duas tabelas.

Modelo de Dados



9) D

- (V) PKs e FKs são objetos do Banco de Dados.

Constraints são regras de validação que o MySQL executa após incluir uma informação em uma

- (F) Tabela do Banco de Dados. (são executadas ANTES da inclusão e se as regras de validação não forem atendidas, a inclusão não é efetuada)

- (V) PKs e FKs são Constraints.

Considerando que temos uma tabela A relacionada a uma tabela B e que a PK de A é uma FK em

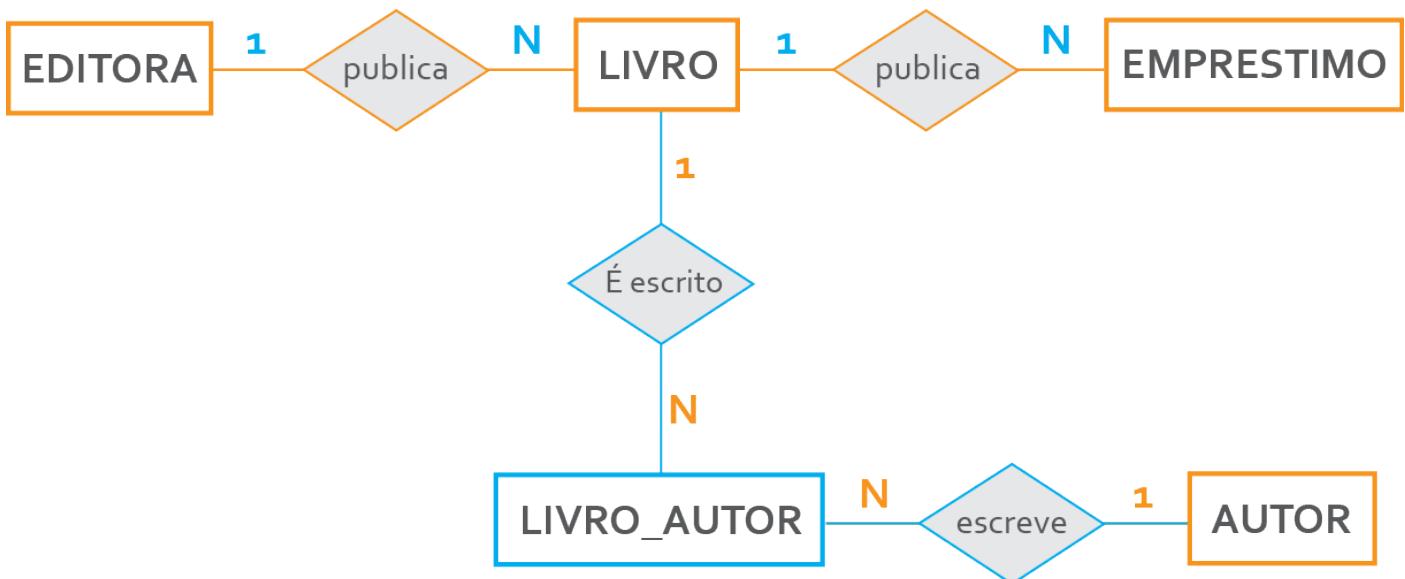
- (V) B. Quando tenta-se incluir um valor na FK de B é verificado se este valor existe na PK de alguma linha de A. Isto é chamado de Integridade Referencial.

Considerando que temos uma tabela A relacionada a uma tabela B e que a PK de A é uma FK em

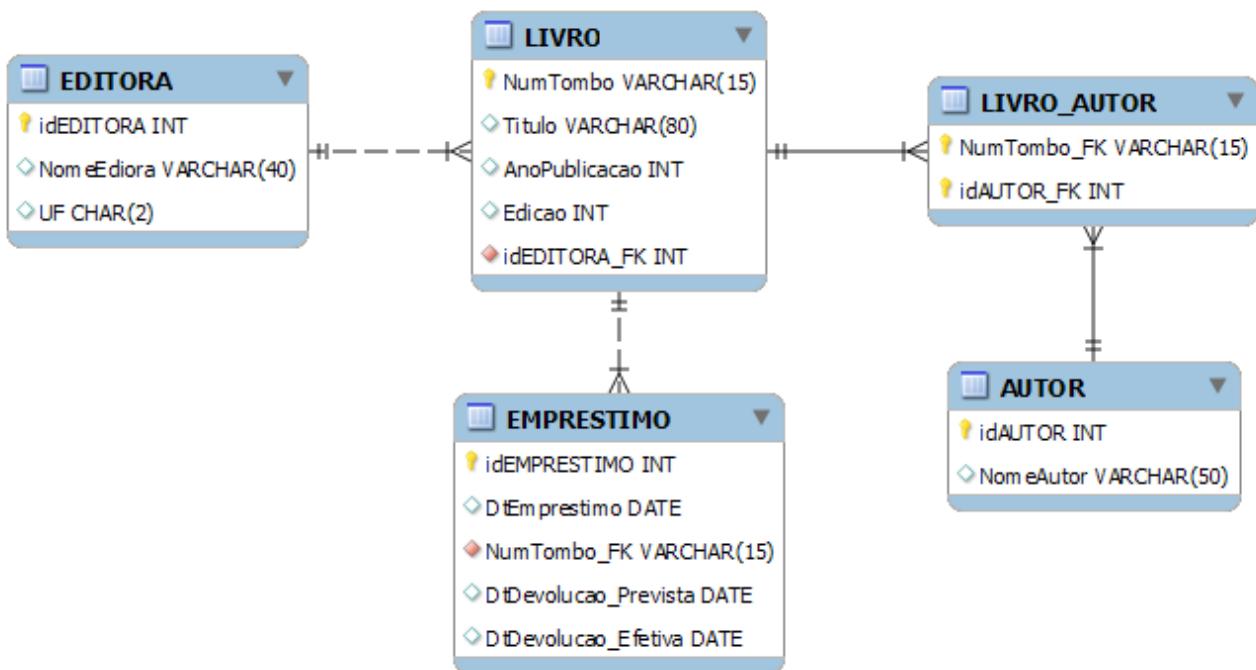
- (V) B. Quando tenta-se incluir um valor na PK de B é verificado se este valor existe na PK de alguma linha de A. Isto é chamado de Integridade Referencial.

Para a descrição apresentada, são os seguintes DER e Modelo de Dados:

DER



Modelo de Dados



Comentários e dicas:

O relacionamento entre Livro e Autor seria N x N, pois um Livro pode ter vários Autores e um Autor pode escrever vários Livros. Este relacionamento foi transformado em dois relacionamentos 1 x N, quando criamos uma tabela para referenciar as duas: LIVRO_AUTOR.

12) O Mapeamento é uma simplificação do DER e do Modelo de Dados.

Editora (idEditora, NomeEditora, UF)

Livro (NumTombo, Titulo, AnoPublicacao, Edicao, idEditora_FK)

Autor (idAutor, NomeAutor)

Livro_Autor (NumTombo_FK, idAutor_FK)

Emprestimo (idEmprestimo, DtEmprestimo, NumTombo_FK, DtDevolucao_Prevista, DtDevolucao_Efetiva)

Módulo 3

1) Comandos para criação das tabelas

```
-- Tabela fFORNECEDOR
CREATE TABLE fFORNECEDOR (
    idFORNECEDOR           INT          NOT NULL,
    NomeFornecedor          VARCHAR(45)   NULL,
    PRIMARY KEY (idFORNECEDOR)
);

-- Tabela fPRINCIPIOATIVO
CREATE TABLE fPRINCIPIOATIVO (
    idPRINCIPIOATIVO        INT          NOT NULL,
    NomePrincipioAtivo       VARCHAR(60)   NULL,
    PRIMARY KEY (idPRINCIPIOATIVO)
);

-- Tabela fMEDICAMENTO
CREATE TABLE fMEDICAMENTO (
    idMEDICAMENTO            INT          NOT NULL,
    NomeComercial             VARCHAR(60)   NULL,
    idPRINCIPIOATIVO_FK      INT          NULL,
    idFORNECEDOR_FK          INT          NULL,
    QtdeEstoque                INT          NULL,
    PRIMARY KEY (idMEDICAMENTO),
    FOREIGN KEY (idFORNECEDOR_FK)
        REFERENCES fFORNECEDOR          (idFORNECEDOR),
    FOREIGN KEY (idPRINCIPIOATIVO_FK)
        REFERENCES fPRINCIPIOATIVO (idPRINCIPIOATIVO)
);
```

```

-----  

-- Tabela hTIPO_EQUIPAMENTO  

-----  

CREATE TABLE hTIPO_EQUIPAMENTO (
    idTIPO_EQUIPAMENTO      INT          NOT NULL,
    DescricaoTipo            VARCHAR(45)  NULL,
    PRIMARY KEY (idTIPO_EQUIPAMENTO)
);  

-----  

-- Tabela hFABRICANTE  

-----  

CREATE TABLE hFABRICANTE (
    idFABRICANTE           INT          NOT NULL,
    NomeFabricante          VARCHAR(40)  NULL,
    PRIMARY KEY (idFABRICANTE)
);  

-----  

-- Tabela hEQUIPAMENTO  

-----  

CREATE TABLE hEQUIPAMENTO (
    NumSerie_EQUIPAMENTO    INT          NOT NULL,
    idTIPO_EQUIPAMENTO_FK   INT          NOT NULL,
    idFABRICANTE_FK         INT          NOT NULL,
    Processador              VARCHAR(25)  NULL,
    Memoria                 VARCHAR(15)  NULL,
    PRIMARY KEY (NumSerie_EQUIPAMENTO),
    FOREIGN KEY (idTIPO_EQUIPAMENTO_FK)
        REFERENCES hTIPO_EQUIPAMENTO (idTIPO_EQUIPAMENTO),
    FOREIGN KEY (idFABRICANTE_FK)
        REFERENCES hFABRICANTE (idFABRICANTE)
);  

-----  

-- Tabela hUSUARIO  

-----  

CREATE TABLE hUSUARIO (
    idUSUARIO               INT          NOT NULL,
    NomeUsuario              VARCHAR(50)  NULL,
    Senha                   VARCHAR(15)  NULL,
    PRIMARY KEY (idUSUARIO)
);

```

```

-- -----
-- Tabela ALOCACAO_EQUIPAMENTO

CREATE TABLE ALOCACAO_EQUIPAMENTO (
    NumSerie_EQUIPAMENTO_FK INT NOT NULL,
    idUSUARIO_FK INT NOT NULL,
    PRIMARY KEY (NumSerie_EQUIPAMENTO_FK, idUSUARIO_FK),
    FOREIGN KEY (NumSerie_EQUIPAMENTO_FK)
        REFERENCES hEQUIPAMENTO (NumSerie_EQUIPAMENTO),
    FOREIGN KEY (idUSUARIO_FK)
        REFERENCES hUSUARIO (idUSUARIO)
);

-- -----
-- Tabela hMOTIVO

CREATE TABLE hMOTIVO (
    idMOTIVO INT NOT NULL,
    DescricaoMotivo VARCHAR(50) NULL,
    PRIMARY KEY (idMOTIVO)
);

-- -----
-- Tabela hCHAMADO

CREATE TABLE hCHAMADO (
    NumCHAMADO INT NOT NULL,
    idMOTIVO_FK INT NOT NULL,
    NumSerie_EQUIPAMENTO_FK_FK INT NOT NULL,
    idUSUARIO_FK_FK INT NOT NULL,
    dtAbertura DATETIME NULL,
    dtEncerramento DATETIME NULL,
    Encerrado_S_N CHAR NULL,
    PRIMARY KEY (NumCHAMADO),
    FOREIGN KEY (idMOTIVO_FK)
        REFERENCES hMOTIVO (idMOTIVO),
    FOREIGN KEY (NumSerie_EQUIPAMENTO_FK_FK , idUSUARIO_FK_FK)
        REFERENCES ALOCACAO_EQUIPAMENTO
            (NumSerie_EQUIPAMENTO_FK , idUSUARIO_FK)
);

```

3) Comandos SQL

a)	<pre>SELECT Nome_RazaoSocial, Cidade, UF FROM lCLIENTE;</pre>
b)	<pre>SELECT * FROM lLICENCA WHERE NumLicenca LIKE '%A%';</pre>
c)	<pre>SELECT * FROM lCLIENTE WHERE Nome_RazaoSocial LIKE 'P%' ORDER BY Nome_RazaoSocial;</pre>
d)	<pre>SELECT * FROM lCLIENTE WHERE Nome_RazaoSocial LIKE '%AR' ORDER BY Nome_RazaoSocial DESC;</pre>
e)	<pre>SELECT * FROM lCLIENTE WHERE Nome_RazaoSocial LIKE '%W%' OR Nome_RazaoSocial LIKE '%Y%';</pre>
f)	<pre>SELECT * FROM lLICENCA WHERE ValorAquisicao >= 1200 ORDER BY ValorAquisicao DESC;</pre>
g)	<pre>SELECT * FROM lCLIENTE WHERE idCLIENTE > 150 AND idCLIENTE < 200;</pre> <p>Neste caso o BETWEEN não funcionaria, pois ele incluiria o 150 e o 200, a não ser que fosse usado como: WHERE idCLIENTE BETWEEN 151 AND 199</p>
h)	<pre>SELECT * FROM lLICENCA WHERE ValorAquisicao BETWEEN 250 AND 500 ORDER BY ValorAquisicao;</pre>
i)	<pre>SELECT * FROM lLICENCA WHERE YEAR(DtAquisicao) > 2008 AND (ValorAquisicao BETWEEN 300 AND 450 OR ValorAquisicao BETWEEN 600 AND 800); </pre>
j)	<pre>SELECT * FROM lCLIENTE WHERE UF IN ('SP', 'RS', 'PR', 'MG');</pre>
k)	<pre>SELECT * FROM lCLIENTE WHERE UF NOT IN ('RJ', 'ES', 'SP', 'MG');</pre>

i)	<pre>INSERT INTO lCLIENTE values (1, 98, 2, 'ABYARA', 'AV. Paulista, 2100', 'São Paulo', 'SP', '01218100');</pre>
m)	<pre>INSERT INTO lSOFTWARE VALUES (20,'MySQL'); INSERT INTO lVERSAO VALUES (20,'5.6');</pre>
n)	<pre>INSERT INTO lLICENCA values ('P1094AE4', 700, 3, '2007', '2014-04-23', 560); INSERT INTO lLICENCA values ('P1102AE4', 700, 10, '4.0' , '2014-04-24', 340); INSERT INTO lLICENCA values ('P1110AE4', 700, 20, '5.6' , '2014-04-25', 1);</pre>
o)	<pre>UPDATE lLICENCA SET ValorAquisicao = ValorAquisicao * 1.125 WHERE NumLicenca IN ('P1094AE4', 'P1102AE4', 'P1110AE4');</pre>
p)	<pre>UPDATE lTIPO_EMPRESA SET DescricaoTipo = 'Governo' WHERE IdTIPO_EMPRESA = 6;</pre>
q)	<pre>SELECT NumLicenca, DtAquisicao, ValorAquisicao, ValorAquisicao * 0.10 as Imposto, ValorAquisicao - ValorAquisicao * 0.10 as ValorLiquido FROM lLICENCA WHERE YEAR(DtAquisicao) > 2010 ORDER BY DtAquisicao;</pre>

Módulo 4

a)	<pre>SELECT COUNT(*) as QtdeClientes FROM lCLIENTE;</pre>
b)	<pre>SELECT * FROM lLICENCA WHERE MONTH(DtAquisicao) = 5; -- mês de nascimento</pre>
c)	<pre>SELECT * FROM lLICENCA WHERE MONTH(DtAquisicao) = 3 AND YEAR(DtAquisicao) = 2007;</pre>
d)	<pre>SELECT idTIPO_EMPRESA, DescricaoTipo, LEFT(DescricaoTipo,5) FROM lTIPO_EMPRESA ORDER BY DescricaoTipo;</pre>
e)	<pre>SELECT idTIPO_EMPRESA, DescricaoTipo, RIGHT(DescricaoTipo,5) FROM lTIPO_EMPRESA ORDER BY DescricaoTipo DESC;</pre>
f)	<pre>SELECT idTIPO_EMPRESA, DescricaoTipo, SUBSTRING(DescricaoTipo, 6, 5) FROM lTIPO_EMPRESA;</pre>

g)	<pre>SELECT Nome_RazaoSocial, LENGTH (Nome_RazaoSocial) FROM tCLIENTE ORDER BY Nome_RazaoSocial;</pre>
h)	<pre>SELECT NumLicenca, DtAquisicao, DATEDIFF (NOW(), DtAquisicao) as DiasDecorridos FROM tLICENCA;</pre>
i)	<pre>SELECT UPPER(NomeSetor) as NomeMaiusculo, LOWER(NomeSetor) as NomeMinusculo FROM tSETOR;</pre>

JOIN

j)	<pre>SELECT S.idSOFTWARE, NomeSoftware, Versao FROM tSOFTWARE S JOIN tVERSAO V ON S.idSOFTWARE = V.idSOFTWARE_FK ORDER BY NomeSoftware, Versao;</pre>
k)	<pre>SELECT idCLIENTE, Nome_RazaoSocial, Endereco, Cidade, UF, DescricaoTipo, NomeSetor FROM tCLIENTE C JOIN tTIPO_EMPRESA T ON C.idTIPO_EMPRESA_FK = T.idTIPO_EMPRESA JOIN tSETOR S ON C.idSETOR_FK = S.idSETOR ORDER BY DescricaoTipo, NomeSetor;</pre>
l)	<pre>SELECT idCLIENTE, Nome_RazaoSocial, NumLicenca, DtAquisicao, ValorAquisicao FROM tCLIENTE C JOIN tLICENCA L ON C.idCLIENTE = L.idCLIENTE_FK;</pre>
m)	<pre>SELECT DISTINCT idCLIENTE, Nome_RazaoSocial, NomeSoftware FROM tCLIENTE C JOIN tLICENCA L ON C.idCLIENTE = L.idCLIENTE_FK JOIN tSOFTWARE S ON L.idSOFTWARE_FK_FK = S.idSOFTWARE ORDER BY Nome_RazaoSocial, NomeSoftware;</pre> <p>O DISTINCT é usado para eliminar repetições de software, pois um mesmo cliente pode ter adquiridos várias licenças no mesmo Software, inclusive em versões diferentes.</p>
n)	<pre>SELECT idCLIENTE, Nome_RazaoSocial, Endereco, Cidade, UF, DescricaoTipo, NomeSetor FROM tCLIENTE C JOIN tTIPO_EMPRESA T ON C.idTIPO_EMPRESA_FK = T.idTIPO_EMPRESA JOIN tSETOR S ON C.idSETOR_FK = S.idSETOR WHERE UF IN ('SP', 'RS', 'PR', 'MG');</pre>

o)

```

SELECT      NomeSoftware,
            Versao,
            Nome_RazaoSocial,
            DescricaoTipo,
            NomeSetor,
            NumLicenca,
            DtAquisicao,
            ValorAquisicao

FROM        lSOFTWARE      S

JOIN        lVERSAO          V
ON          S.idSOFTWARE     =      V.idSOFTWARE_FK

JOIN        lLICENCA         L
ON          V.idSOFTWARE_FK =      L.idSOFTWARE_FK_FK
AND         V.Versao         =      L.Versao_FK

JOIN        CLIENTE           C
ON          L.idCLIENTE_FK   =      C.idCLIENTE

JOIN        lTIPO_EMPRESA    T
ON          C.idTIPO_EMPRESA_FK =  T.idTIPO_EMPRESA

JOIN        lSETOR            SE
ON          C.idSETOR_FK      =      SE.idSETOR

ORDER BY    NomeSoftware, Versao, DtAquisicao, Nome_RazaoSocial;

```

Funções Estatísticas com JOIN

p)

```

SELECT      COUNT(*) AS QtdeLicencas
FROM        lLICENCA;
```

q)

```

SELECT      SUM(ValorAquisicao) as ValorTotal,
            AVG(ValorAquisicao) as ValorMedio,
            MAX(ValorAquisicao) as MaiorValor
FROM        lLICENCA;
```

r)

```

SELECT      COUNT(*) as QtdeClientes
FROM        lCLIENTE      C
JOIN        lSETOR         S
ON          C.idSETOR_FK   =      S.idSETOR
WHERE       NomeSetor      =      'Farmacêutica';
```

s)

```

SELECT      Nome_RazaoSocial as NomeCliente, COUNT(*) as QtdeLicencas
FROM        lLICENCA      L
JOIN        lCLIENTE      C
ON          L.idCLIENTE_FK =      C.idCLIENTE
GROUP BY   Nome_RazaoSocial
ORDER BY   Nome_RazaoSocial;
```

t)	<pre> SELECT Nome_RazaoSocial as NomeCliente, SUM(ValorAquisicao) as ValorTotal, AVG(ValorAquisicao) as ValorMedio FROM LLICENCA L JOIN LCLIENTE C ON L.idCLIENTE_FK = C.idCLIENTE GROUP BY Nome_RazaoSocial ORDER BY Nome_RazaoSocial; </pre>
u)	<pre> SELECT NomeSetor, SUM(ValorAquisicao) as ValorTotal FROM LLICENCA L JOIN LCLIENTE C ON L.idCLIENTE_FK = C.idCLIENTE JOIN LSETOR S ON C.idSETOR_FK = S.idSETOR GROUP BY NomeSetor ORDER BY NomeSetor; </pre>
v)	<pre> SELECT DescricaoTipo as TipoEmpresa, SUM(ValorAquisicao) as ValorTotal FROM LLICENCA L JOIN LCLIENTE C ON L.idCLIENTE_FK = C.idCLIENTE JOIN LTIPO_EMPRESA T ON C.idTIPO_Empresa_FK = T.idTIPO_Empresa GROUP BY DescricaoTipo ORDER BY DescricaoTipo; </pre>
w)	<pre> SELECT NomeSoftware, Versao, SUM(ValorAquisicao) as ValorTotal FROM LLICENCA L JOIN LVERSAO V ON L.idSOFTWARE_FK_FK = V.idSOFTWARE_FK AND L.Versao_FK = V.Versao JOIN LSOFTWARE S ON V.idSOFTWARE_FK = S.idSOFTWARE GROUP BY NomeSoftware, Versao ORDER BY NomeSoftware, Versao; </pre>
x)	<pre> SELECT Nome_RazaoSocial as NomeEmpresa, NomeSoftware, COUNT(*) as QtdeLicencas FROM LCLIENTE C JOIN LLICENCA L ON C.idCLIENTE = L.idCLIENTE_FK JOIN LSOFTWARE S ON L.idSOFTWARE_FK_FK = S.idSOFTWARE GROUP BY Nome_RazaoSocial, NomeSoftware ORDER BY Nome_RazaoSocial, NomeSoftware; </pre>
y)	<pre> SELECT Nome_RazaoSocial as NomeCliente, COUNT(*) as QtdeLicencas FROM LLICENCA L JOIN LCLIENTE C ON L.idCLIENTE_FK = C.idCLIENTE GROUP BY Nome_RazaoSocial HAVING QtdeLicencas > 10; </pre>

Objetos de Código Armazenado

```

CREATE VIEW vCLIENTE
AS
SELECT      *
FROM        tCLIENTE          C
JOIN        tSETOR             S
    ON      C.idSETOR_FK      =      S.idSETOR
JOIN        tTIPO_EMPRESA     T
    ON      C.idTIPO_Empresa_FK =      T.idTIPO_Empresa;

```

```

DELIMITER //
CREATE PROCEDURE pLICENCIAS_PERIODICO (IN DtInicio DATE, IN dtFIM DATE)
BEGIN

SELECT      Nome_RazaoSocial  as      NomeCliente,
            NomeSetor,
            NomeSoftware,
            Versao,
            NumLicenca,
            DtAquisicao,
            ValorAquisicao,
            ValorAquisicao * 0.03 as      Comissao
FROM        tCLIENTE          C
JOIN        tSETOR             SE
    ON      C.idSETOR_FK      =      SE.idSETOR
JOIN        tTIPO_EMPRESA     T
    ON      C.idTIPO_Empresa_FK =      T.idTIPO_Empresa
JOIN        tLICENCA          L
    ON      C.idCLIENTE       =      L.idCLIENTE_FK
JOIN        tVERSÃO            V
    ON      L.idSOFTWARE_FK_FK =      V.idSOFTWARE_FK
    AND     L.Versao_FK        =      V.Versao
JOIN        tSOFTWARE          S
    ON      V.idSOFTWARE_FK   =      S.idSOFTWARE
WHERE     DtAquisicao        BETWEEN      DtInicio AND dtFim
ORDER BY    Nome_RazaoSocial,
            NomeSetor,
            NomeSoftware;

END //
DELIMITER ;

```

Para executar a PROC (as datas são exemplos – formato aaa/mm/dd):
 CALL pLICENCIAS_PERIODICO ('2014-01-01', '2014-05-31');

RESPOSTA ALTERNATIVA (como há um comando só, podemos omitir o DELIMITER e BEGIN):

```

CREATE PROCEDURE pLICENCIAS_PERIODICO (IN DtInicio DATE, IN dtFIM DATE)

SELECT      Nome_RazaoSocial           as     NomeCliente,
            NomeSetor,
            NomeSoftware,
            Versao,
            NumLicenca,
            DtAquisicao,
            ValorAquisicao,
            ValorAquisicao * 0.03    as     Comissao
  FROM        tCLIENTE                 C
  JOIN        tSETOR                  SE
    ON         C.idSETOR_FK          =      SE.idSETOR
  JOIN        tTIPO_EMPRESA          T
    ON         C.idTIPO_Empresa_FK =      T.idTIPO_Empresa
  JOIN        tLICENCA                L
    ON         C.idCLIENTE          =      L.idCLIENTE_FK
  JOIN        tVERSAO                 V
    ON         L.idSOFTWARE_FK_FK   =      V.idSOFTWARE_FK
    AND        L.Versao_FK          =      V.Versao
  JOIN        tSOFTWARE               S
    ON         V.idSOFTWARE_FK      =      S.idSOFTWARE
 WHERE       DtAquisicao          BETWEEN      DtInicio AND dtFim
 ORDER BY    Nome_RazaoSocial,
            NomeSetor,
            NomeSoftware;

```

Para executar a PROC (as datas são exemplos – formato aaa/mm/dd):

```
CALL pLICENCIAS_PERIODICO ('2014-01-01', '2014-05-31');
```

```

DELIMITER //
CREATE PROCEDURE pLICENCIAS_ANO      (IN DtInicio DATE, IN dtFIM DATE)
BEGIN

SELECT          YEAR(DtAquisicao)  as    AnoAquisicao,
                NomeSoftware,
                SUM(ValorAquisicao)   as    ValorTotal
FROM            lLICENCA           L
JOIN            lSOFTWARE          S
    ON          L.idSOFTWARE_FK_FK = S.idSOFTWARE
WHERE          DtAquisicao     BETWEEN      DtInicio AND dtFim
GROUP BY        YEAR(DtAquisicao),
                NomeSoftware
ORDER BY        YEAR(DtAquisicao),
                NomeSoftware;

END //
DELIMITER ;

```

Para executar a PROC (as datas são exemplos – formato aaaa/mm/dd):

```
CALL pLICENCIAS_ANO ('2014-01-01', '2014-05-31');;
```

Criação da Tabela de LOG

```

ac) CREATE TABLE      lLOG (
    Tabela          VARCHAR(40) NULL,
    TipoOperacao   VARCHAR(10) NULL,
    IdSOFTWARE      INT        NULL,
    DataLog         DATE       NULL,
    HoraLog         TIME       NULL
);

```

Criação da TRIGGER de Inclusão

```

DELIMITER //
CREATE TRIGGER lSOFTWARE_INCLUSAO AFTER INSERT ON lSOFTWARE
FOR EACH ROW
BEGIN
    INSERT INTO lLOG SET
        Tabela      = 'lSOFTWARE',
        TipoOperacao = 'Inclusao',
        IdSOFTWARE  = NEW.idSOFTWARE,
        DataLog     = CURDATE(),
        HoraLog     = CURTIME()
    ;
END
// 
DELIMITER ;

```

Criação da TRIGGER de Alteração

```

DELIMITER //
CREATE TRIGGER 1SOFTWARE_ALTERACAO BEFORE UPDATE ON 1SOFTWARE
FOR EACH ROW
BEGIN
    INSERT INTO 1LOG SET
        Tabela          = '1SOFTWARE',
        TipoOperacao   = 'Alteracao',
        IdSOFTWARE     = NEW.idSOFTWARE,
        DataLog         = CURDATE(),
        HoraLog         = CURTIME()
    ;
END
//
DELIMITER ;

```

Criação da TRIGGER de Exclusão

```

DELIMITER //
CREATE TRIGGER 1SOFTWARE_EXCLUSAO BEFORE DELETE ON 1SOFTWARE
FOR EACH ROW
BEGIN
    INSERT INTO 1LOG SET
        Tabela          = '1SOFTWARE',
        TipoOperacao   = 'Exclusao',
        IdSOFTWARE     = OLD.idSOFTWARE,
        DataLog         = CURDATE(),
        HoraLog         = CURTIME()
    ;
END
//
DELIMITER ;

```

Comandos de Inclusão, Alteração e Exclusão

```

-- Inclusão id=50
INSERT      INTO 1SOFTWARE VALUES      (50, 'MeuSoft');

-- Alteração do nome do software
UPDATE      1SOFTWARE
SET          NomeSoftware      =      'MySoft'
WHERE        idSOFTWARE       =      50;

-- Exclusão do Software
DELETE      FROM 1SOFTWARE
WHERE        idSOFTWARE      =      50;

```

Listando a Tabela de LOG:

```
SELECT * FROM LLOG;
```

Resultado:

Tabela	TipoOperacao	IdSOFTWARE	DataLog	HoraLog
ISOFTWARE	Inclusao	50	2014-06-16	02:06:09
ISOFTWARE	Alteracao	50	2014-06-16	02:15:37
ISOFTWARE	Exclusao	50	2014-06-16	02:20:21

Módulo 5

1)

a)

```
SELECT      Nome_RazaoSocial as NomeCliente
FROM        lCLIENTE
WHERE       idCLIENTE    IN
            (SELECT      DISTINCT      idCLIENTE_FK
             FROM        LLICENCA
            )
```

Solução com SUB-SELECT. O SELECT mais interno lista os códigos de Clientes que compraram alguma Licença de Software. Tiramos os códigos repetidos com o DISTINCT.

O resultado é uma lista de códigos de clientes.

Usamos então esta lista para filtrar os resultados do SELECT mais externo, de forma que só sejam exibidos os clientes que compraram alguma Licença de Software.

b)

```
SELECT      Nome_RazaoSocial as NomeCliente
FROM        lCLIENTE
WHERE       idCLIENTE    NOT IN
            (SELECT      DISTINCT      idCLIENTE_FK
             FROM        LLICENCA
            )
```

Similar ao anterior, mas aqui usamos a negativa para exibir apenas os Clientes que não compraram nenhuma Licença de Software.

```

SELECT      idCLIENTE, Nome_RazaoSocial as NomeCliente,
            idSETOR, NomeSetor,
            idTIPO_Empresa, DescricaoTipo as TipoEmpresa
FROM        lCLIENTE          C
RIGHT JOIN lSETOR           S
    ON      C.idSETOR_FK      =      S.idSETOR
RIGHT JOIN lTIPO_EMPRESA    T
    ON      C.idTIPO_EMPRESA_FK =      T.idTIPO_EMPRESA

UNION

c)   SELECT      idCLIENTE, Nome_RazaoSocial as NomeCliente,
            idSETOR, NomeSetor,
            idTIPO_Empresa, DescricaoTipo as TipoEmpresa
FROM        lCLIENTE          C
LEFT JOIN  lSETOR           S
    ON      C.idSETOR_FK      =      S.idSETOR
LEFT JOIN  lTIPO_EMPRESA    T
    ON      C.idTIPO_EMPRESA_FK =      T.idTIPO_EMPRESA
;

```

Note que para garantir que todos os registros das 3 tabelas tenham sido incluídos, precisamos usar ambos: RIGHT JOIN e LEFT JOIN. Para uni-los, utilizamos a cláusula UNION.

```

SELECT      NomeSoftware, Versao, COUNT(*) as QtdeLicencas
FROM        lLICENCA          L
JOIN        lVERSAO           V
    ON      L.idSoftware_FK_FK =      V.idSoftware_FK
    AND     L.Versao_FK       =      V.Versao
JOIN        lSOFTWARE          S
    ON      V.idSoftware_FK      =      S.idSoftware
GROUP BY    NomeSoftware, Versao

HAVING      QtdeLicencas      =

d)   (
        SELECT      MAX(QtdeLicencas)
        FROM

            (
                SELECT      idSoftware_FK_FK,
                            Versao_FK,
                            COUNT(*) as QtdeLicencas
                FROM        lLICENCA
                GROUP BY    idSoftware_FK_FK, Versao_FK
            ) Q

        )
;

```

Os Sub-Selects são mais fáceis de ser lidos de dentro para fora.

- (1) Este primeiro SELECT, mais interno (destacado em azul), gera um conjunto de dados com a quantidade de licenças para cada Software/Versão;
- (2) O segundo SELECT, intermediário (destacado em vermelho), pesquisa dados de uma tabela virtual. Esta tabela virtual foi produzida pelo SELECT interno.

Este SELECT intermediário seleciona o maior valor da coluna QtdeLicencas produzida no SELECT interno. Observe que o SELECT interno funciona como uma tabela para o segundo SELECT e está na cláusula FROM dele como tal. Para isto funcionar é necessário que o SELECT interno tenha um ALIAS (apelido).

Este ALIAS é a letra "Q" que está logo após o fim do SELECT interno, ou seja, é como se fosse a tabela Q (quantidades). O resultado do SELECT intermediário (que engloba o interno) é um valor equivalente à maior quantidade de Licenças de um Software/Versao.

Mas observe que apenas o valor é retornado, sem a identificação de qual Software/Versão ele se refere.

- (3) O SELECT mais externo (em preto) é quase uma reprodução do SELECT mais interno, mas agora com a apresentação dos Nomes do Software e Versão.

O conjunto de dados retornado da contagem é agrupado por Software/Versão.

Após a obtenção deste conjunto de dados (Software, Versão, Quantidade de Licenças), o conteúdo é filtrado. Lembre-se que após o GROUP BY o filtro deve ser feito com o HAVING e não com o WHERE.

Ao filtrar, pedimos que seja apresentado apenas a linha que contém o valor de quantidade de Licenças igual ao retornado pelo SELECT intermediário, ou seja, a maior quantidade.

Para você ter melhor entendimento, execute os SELECTs separadamente e observe os resultados:

- I. rode o SELECT interno e observe que são retornadas as quantidades de licenças por código de software e versão;
- II. rode o SELECT intermediário (abrange o interno também). o valor retornado é a maior quantidade de licenças comercializadas para um software;
- III. rode o select externo de forma parcial, executando o código apenas até antes do HAVING; você verá uma lista muito similar à do SELECT interno, mas agora como nome do software;
- IV. Enfim, execute o SELECT externo completo (abrange todos os demais). E você obterá qual software e versão teve mais licenças vendidas e qual a quantidade de licenças foram vendidas para aquele software.

De forma simples:

```
SELECT      *      FROM  LLICENCA
WHERE      ValorAquisicao  =
(
    SELECT      MAX(ValorAquisicao)      FROM  LLICENCA
);
```

Ou trazendo os dados do Software e Empresa:

```
SELECT      NumLicenca, Nome_RazaoSocial as NomeCliente,
           NomeSoftware, Versao, DtAquisicao, ValorAquisicao
FROM        LLICENCA      L
JOIN        LVERSAO          V
ON          L.idSoftware_FK_FK      =      V.idSoftware_FK
AND         L.Versao_FK            =      V.Versao
JOIN        LSOFWARE          S
ON          V.idSoftware_FK      =      S.idSoftware
JOIN        LCLIENTE          C
ON          L.idCLIENTE_FK      =      C.idCLIENTE
WHERE       ValorAquisicao  =
(
    SELECT      MAX(ValorAquisicao)      FROM  LLICENCA
);
```

e)

```

SELECT      NumLicenca, DtAquisicao, YEAR(DtAquisicao) as Ano,
CASE        MONTH(DtAquisicao)
    WHEN    1      THEN  '1oTri'
    WHEN    2      THEN  '1oTri'
    WHEN    3      THEN  '1oTri'
    WHEN    4      THEN  '2oTri'
    WHEN    5      THEN  '2oTri'
    WHEN    6      THEN  '2oTri'
    WHEN    7      THEN  '3oTri'
    WHEN    8      THEN  '3oTri'
    WHEN    9      THEN  '3oTri'
    WHEN   10      THEN  '4oTri'
    WHEN   11      THEN  '4oTri'
    WHEN   12      THEN  '4oTri'
END
                           as Trimestre,
ValorAquisicao
FROM  LLICENCA     L
ORDER BY     YEAR(DtAquisicao), Trimestre;

```

Ou usando a outra sintaxe de CASE

```

SELECT      NumLicenca, DtAquisicao, YEAR(DtAquisicao) as Ano,
CASE
    WHEN    MONTH(DtAquisicao) IN ( 1,  2,  3) THEN  '1oTri'
    WHEN    MONTH(DtAquisicao) IN ( 4,  5,  6) THEN  '2oTri'
    WHEN    MONTH(DtAquisicao) IN ( 7,  8,  9) THEN  '3oTri'
    WHEN    MONTH(DtAquisicao) IN (10, 11, 12) THEN  '4oTri'
END
                           as Trimestre,
ValorAquisicao
FROM  LLICENCA     L
ORDER BY     YEAR(DtAquisicao), Trimestre;

```

```

SELECT      idCLIENTE,
Nome_RazaoSocial AS NomeCliente,
Cidade,
UF,
CASE
    WHEN UF IN ('RS', 'SC', 'PR')          THEN  'SUL'
    WHEN UF IN ('SP', 'RJ', 'MG', 'ES')    THEN  'SUDESTE'
    WHEN UF IN ('GO', 'DF', 'MS', 'MT')    THEN  'CENTRO'
    ELSE                                     'NORTE/NORDESTE'
END AS Regiao
FROM  LCLIENTE;

```

2)

```

h)      SELECT      idMOTIVO, DescricaoMotivo, NumChamado, dtAbertura
        FROM       hMOTIVO      M
        LEFT JOIN hCHAMADO    C
        ON         M.idMOTIVO = C.idMOTIVO_FK;

```

```

i)      SELECT      *
        FROM       hFABRICANTE    F
        LEFT JOIN hEQUIPAMENTO  E
        ON         F.idFABRICANTE = E.idFABRICANTE_FK;

```

Bibliografia

DUBOIS, P.; HINZ, S.; PEDERSEN, C. **MYSQL: Guia de estudo para certificação**. Rio de Janeiro: Ciência Moderna, 2005.

MILANI, A. **MySQL: Guia do programador**. São Paulo: Novatec, 2007.

DATE, C. J. **Introdução a sistemas de bancos de dados**. 7ª ed. Rio de Janeiro: Campus, 2000.

IBM. **Edgar F. Codd**. 2003. Disponível em: <http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html>. Acesso em: 14 mai. 2014.

MILANI, A. **Capítulo 1. - Bem-vindo ao MYSQL**. In: MySQL: Guia do Programador. São Paulo: Novatec, 2007. Disponível em: <https://novatec.com.br/livros/mysqlcompleto/capitulo8575221035.pdf>. Acesso em: 7 mai. 2014.

MySQL. **Manual de referência do MySQL – sintaxe de comandos**. 2014. Disponível em: <<http://dev.mysql.com/doc/refman/5.6/en/index.html>>. Acesso em: 5 mai. 2014.

MySQL. **MySQL downloads**. 2014. Disponível em: <<http://www.mysql.com/downloads>>. Acesso em: 7 mai. 2014.

Microsoft. **SQL: conceitos básicos, vocabulário e sintaxe**. 2014. Disponível em: <<http://office.microsoft.com/pt-br/access-help/access-sql-conceitos-basicos-vocabulario-e-sintaxe-HA010256402.aspx>>. Acesso em: 6 mai. 2014.

MANZANO, J. A. N. G. **MySQL 5 Interativo**. São Paulo: Érica, 207

ORACLE. **Oracle database SQL reference – ANSI standards**. 2003. Disponível em: <http://docs.oracle.com/cd/B12037_01/server.101/b10759/ap_standard_sql001.htm>. Acesso em: 14 mai. 2014.

SUEHRING, S. **MySQL – A Bíblia**. Rio de Janeiro: Campus, 2002.

TAHAGHOGHI, S. M. M.; WILLIAMS, H. E. **Aprendendo MySQL**. Rio de Janeiro: Alta Books, 2007.

TEORY, T.; HIGHSTONE, S.; NARDEAU, T. **Projeto e modelagem de bancos de dados**. Rio de Janeiro: Elsevier, 2006.

1KEYDATA.COM. **Tutorial de SQL**. 2014. Disponível em: <<http://www.1keydata.com/pt/sql/>>. Acesso em: 6 mai. 2014.