Modelagem de Dados e Banco de Dados

do teórico à prática
sem complicações
introduzindo MySQL Server

Versão 2.0 – 26/06/13

Rudinei Pereira Dias

INTRODUÇÃO

Com a grande quantidade de dados nas organizações, toda a informação torna-se valiosa e é necessário uma forma organizada e controlada de manter esses dados.

A Modelagem de dados ensina como organizar e estruturar as informações em bancos de dados, de forma a permitir a utilização pelos SGBDs e a criação de sistemas que necessitem acessar esses dados.

A SQL ensina as estruturas de linguagem para controlar, acessar, consultar e manipular os dados armazenados em bancos de dados.

MySQL

MySQL é um SGBD (Sistema Gerenciador de Banco de Dados) relacional padrão SQL (Structured Query Language - Linguagem Estruturada para Consultas) robusto, rápido, multiusuário e multi-tarefa [1].

O MySQL é marca registrada da ORACLE e seu servidor de banco de dados vem sendo distribuído sobre uma Licença Dupla, uma Open Source/GNU GPL e também por uma licença comercial.

O site oficial do MySQL é http://www.mysql.com/.

Este livro aborda a versão 5x do MySQL para a prática de Banco de Dados e apresenta ferramentas para a sua utilização, não tendo a pretensão de ser completo, tampouco cobrir toda a *API* de comandos do MySQL. Para acesso a toda *API*, acesse http://dev.mysql.com/doc/refman/5.5/en/.

Banco *de Dados*

Banco de dados (ou base de dados), é um conjunto de registros dispostos em estrutura regular que possibilita a reorganização dos mesmos e produção de informação. Um banco de dados normalmente agrupa registros utilizáveis para um mesmo fim.

Um banco de dados é usualmente mantido e acessado por meio de um software conhecido como Sistema Gerenciador de Banco de Dados (SGBD). Normalmente um SGBD adota um modelo de dados, de forma pura, reduzida ou estendida. Muitas vezes o termo banco de dados é usado, de forma errônea, como sinônimo de SGDB.

O modelo de dados mais adotado hoje em dia é o modelo relacional, onde as estruturas têm a forma de tabelas, compostas por tuplas (linhas) e colunas.

Os bancos de dados são utilizados em muitas aplicações, abrangendo praticamente todo o campo dos programas de computador. Os bancos de dados são o método de armazenamento preferencial e baseiam-se em tecnologias padronizadas de bancos de dados.

Um banco de dados é um conjunto de informações com uma estrutura regular. Um banco de dados é normalmente, mas não necessariamente, armazenado em algum formato de máquina legível para um computador. Há uma grande variedade de bancos de dados, desde simples tabelas armazenadas em um único arquivo até gigantescos bancos de dados com muitos milhões de registos, armazenados em salas cheias de discos rígidos.

Bancos de dados caracteristicamente modernos são desenvolvidos desde os anos da década de 1960. Um pioneiro nesse trabalho foi Charles Bachman.

Apresentação dos dados geralmente

Apresentação dos dados geralmente é semelhante à de uma planilha eletrônica, porém os sistemas de gestão de banco de dados possuem características especiais para o armazenamento, classificação, gestão da integridade e recuperação dos dados. Com a evolução de padrões de conectividade entre as tabelas de um banco

de dados e programas desenvolvidos em linguagens como Java, Delphi, Visual Basic, C++, PHP, C#, etc, a apresentação dos dados, bem como a navegação, passou a ser definida pelo programador ou o designer de aplicações. Como hoje em dia a maioria das linguagens de programação fazem ligações a bancos de dados, a apresentação destes tem ficado cada vez mais a critério dos meios de programação, fazendo com que os bancos de dados deixem de restringir-se às pesquisas básicas, dando lugar ao compartilhamento, em tempo real, de informações, mecanismos de busca inteligentes e permissividade de acesso hierarquizada.

Modelos de base de dados

O modelo plano (ou tabular) consiste de matrizes simples, bidimensionais, compostas por elementos de dados: inteiros, números reais, etc. Este modelo plano é a base das planilhas eletrônicas.

O modelo em rede permite que várias tabelas sejam usadas simultaneamente através do uso de apontadores (ou referências). Algumas colunas contêm apontadores para outras tabelas ao invés de dados. Assim, as tabelas são ligadas por referências, o que pode ser visto como uma rede. Uma variação particular deste modelo em rede, o modelo hierárquico, limita as relações a uma estrutura semelhante a uma árvore (hierarquia - tronco, galhos), ao invés do modelo mais geral direcionado por grafos.

Bases de dados relacionais consistem, principalmente de três componentes: uma coleção de estruturas de dados, nomeadamente relações, ou informalmente tabelas; uma coleção dos operadores, a álgebra e o cálculo relacionais; e uma coleção de restrições da integridade, definindo o conjunto consistente de estados de base de dados e de alterações de estados. As restrições de integridade podem ser de quatro tipos: domínio (também conhecidas como type), atributo, relvar (variável relacional) e restrições de base de dados.

Diferentemente dos modelos hierárquico e de rede, não existem quaisquer apontadores, de acordo com o Princípio de Informação: toda informação tem de ser representada como dados; qualquer tipo de atributo representa relações entre conjuntos de dados. As bases de dados relacionais permitem aos utilizadores (incluindo programadores) escreverem consultas (queries) que não foram antecipadas por quem projetou a base de dados. Como resultado, bases de dados relacionais podem ser utilizadas por várias aplicações em formas que os projetistas originais não previram, o que é especialmente importante em bases de dados que podem ser utilizadas durante décadas. Isto tem tornado as bases de dados relacionais muito populares no meio empresarial.

O modelo relacional é uma teoria matemática desenvolvida por Edgard Frank Codd, matemático e pesquisador da IBM, para descrever como as bases de dados devem funcionar. Embora esta teoria seja a base para o software de bases de dados relacionais, muito poucos sistemas de gestão de bases de dados seguem o modelo de forma restrita ou a pé da letra - lembre-se das 13 leis do modelo relacional - e todos têm funcionalidades que violam a teoria, desta forma variando a complexidade e o poder. A discussão se esses bancos de dados merecem ser chamados de relacional ficou esgotada com o tempo, com a evolução dos bancos existentes. Os bancos de dados hoje implementam o modelo definido como objeto-relacional.

As 13 leis do modelo relacional

Em 1985, Edgar Frank Codd, criador do modelo relacional, publicou um artigo onde definia 13 regras para que um Sistema Gerenciador de Banco de Dados (SGBD) fosse considerado relacional:

1. Regra Fundamental: Um SGBD relacional deve gerir os seus dados usando apenas suas capacidades relacionais
2. Regra da informação: Toda informação deve ser representada de uma única forma, como dados em uma tabela
3. Regra da garantia de acesso: Todo o dado (valor atômico) pode ser acedido logicamente (e unicamente) usando o nome da tabela, o valor da chave primária da linha e o nome da coluna.

4. Tratamento sistemático de valores nulos: Os valores nulos (diferente do zero, da *string* vazia, da *string* de caracteres em brancos e outros valores não nulos) existem para representar dados não existentes de forma sistemática e independente do tipo de dado.
5. Catálogo dinâmico on-line baseado no modelo relacional: A descrição do banco de dados é representada no nível lógico como dados ordinários (isso é, em tabelas), permitindo que usuários autorizados apliquem as mesmas formas de manipular dados aplicada aos dados comuns ao consultá-las.
6. Regra da sub-linguagem abrangente: Um sistema relacional pode suportar várias linguagens e formas de uso, porém deve possuir ao menos uma linguagem com sintaxe bem definida e expressa por cadeia de caracteres e com habilidade de apoiar a definição de dados, a definição de visões, a manipulação de dados, as restrições de integridade, a autorização e a fronteira de transações.
7. Regra da atualização de visões: Toda visão que for teoricamente atualizável será também atualizável pelo sistema.
8. Inserção, atualização e eliminação de alto nível: Qualquer conjunto de dados que pode ser manipulado com um único comando para retornar informações, também deve ser manipulado com um único comando para operações de inserção, atualização e exclusão. Simplificando, significa dizer que as operações de manipulação de dados devem poder ser aplicadas a várias linhas de uma vez, ao invés de apenas uma por vez.
9. Independência dos dados físicos: Programas de aplicação ou atividades de terminal permanecem logicamente inalteradas quaisquer que sejam as modificações na representação de armazenagem ou métodos de acesso internos.
10. Independência lógica de dados: Programas de aplicação ou atividades de terminal permanecem logicamente inalteradas quaisquer que sejam as mudanças de informação que permitam teoricamente a não alteração das tabelas base.
11. Independência de integridade: As relações de integridade específicas de um banco de dados relacional devem ser definidas em uma sub-linguagem de dados e armazenadas no catálogo (e não em programas).
12. Independência de distribuição: A linguagem de manipulação de dados deve possibilitar que as aplicações permaneçam inalteradas estejam os dados centralizados ou distribuídos fisicamente.
13. Regra da Não-subversão: Se o sistema relacional possui uma linguagem de baixo nível (um registro por vez), não deve ser possível subverter ou ignorar as regras de integridade e restrições definidas no alto nível (muitos registros por vez).

Aplicações de bancos de dados

Sistemas Gerenciadores de Bancos de dados são usados em muitas aplicações, enquanto atravessando virtualmente a gama inteira de software de computador. Os Sistemas Gerenciadores de Bancos de dados são o método preferido de armazenamento/recuperação de dados/informações para aplicações multi-usuárias grandes onde a coordenação entre muitos usuários é necessária. Até mesmo usuários individuais os acham conveniente, entretanto, muitos programas de correio eletrônico e organizadores pessoais estão baseados em tecnologia de banco de dados.

Aplicativo de Banco de Dados

Um Aplicativo de Banco de dados é um tipo de software exclusivo para gerenciar um banco de dados.

O termo "Aplicativo de Banco de dados" usualmente se refere a softwares que oferecem uma interface para o banco de dados. O software que gerencia os dados é geralmente chamado de sistema gerenciador de banco de dados (SGBD) ou (se for embarcado) de "*database engine*". Exemplos de aplicativos de banco de dados são Microsoft Visual FoxPro, Microsoft Access, dBASE, FileMaker, (em certa medida) HyperCard, MySQL, PostgreSQL, Firebird, Microsoft SQL Server, Oracle, Informix, DB2, Caché e Sybase.

Transação

É um conjunto de procedimentos que é executado num banco de dados, que para o usuário é visto como uma única ação.

A integridade de uma transação depende de 4 propriedades, conhecidas como ACID.

• **Atomicidade:** Todas as ações que compõem a unidade de trabalho da transação devem ser concluídas com sucesso, para que seja efetivada. Qualquer ação que constitui falha na unidade de trabalho e a transação deve ser desfeita (rollback). Quando todas as ações são efetuadas com sucesso, a transação pode ser efetivada (commit).

• **Consistência:** Nenhuma operação do banco de dados de uma transação pode ser parcial. O status de uma transação deve ser implementado na íntegra. Por exemplo, um pagamento de conta não pode ser efetivado se o processo que debita o valor da conta corrente do usuário não for efetivado antes, nem vice-versa.

• **Isolamento:** Cada transação funciona completamente à parte de outras estações. Todas as operações são parte de uma transação única. O principio é que nenhuma outra transação, operando no mesmo sistema, pode interferir no funcionamento da transação corrente(é um mecanismo de controle). Outras transações não podem visualizar os resultados parciais das operações de uma transação em andamento.

• **Durabilidade**: Significa que os resultados de uma transação são permanentes e podem ser desfeitos somente por uma transação subseqüente. Por exemplo: todos os dados e status relativos a uma transação devem ser armazenados num repositório permanente, não sendo passíveis de falha por uma falha de hardware.

Na prática, alguns SGBDs relaxam na implementação destas propriedades buscando desempenho.

Controle de concorrência é um método usado para garantir que as transações sejam executadas de uma forma segura e sigam as regras ACID. Os SGBD devem ser capazes de assegurar que nenhuma ação de transações completadas com sucesso (committed transactions) seja perdida ao desfazer transações abortadas (rollback).

Uma transação é uma unidade que preserva consistência. Requeremos, portanto, que qualquer escalonamento produzido ao se processar um conjunto de transações concorrentemente seja computacionalmente equivalente a um escalonamento produzindo executando essas transações serialmente em alguma ordem. Diz-se que um sistema que garante esta propriedade assegura a seriabilidade.

Tabelas

Nos modelos de bases de dados relacionais, a tabela é um conjunto de dados dispostos em número finito de colunas e número ilimitado de linhas (ou tuplos).

As colunas são tipicamente consideradas os campos da tabela, e caracterizam os tipos de dados que deverão constar na tabela (numéricos, alfa-numéricos, datas, coordenadas, etc). O número de linhas pode ser interpretado como o número de combinações de valores dos campos da tabela, e pode conter linhas idênticas, dependendo do objectivo. A forma de referenciar inequivocamente uma única linha é através da utilização de uma chave primária.

Para além do tipo de dados inerente a todas as colunas de uma tabela, algumas podem ter associadas restrições: a unicidade (SQL: UNIQUE), proibição de valores NULL (SQL: NOT NULL), delimitação de valores, etc.

Estas restrições impedem que sejam inseridos valores não desejados que comprometam a validade e integridade dos dados.

O número de tuplos de uma tabela é virtualmente ilimitado, o que torna as pesquisas por valor potencialmente muito lentas. Para permitir agilizar estas consultas, podem ser associados índices à tabela, que são estruturas de dados independentes da forma e ordem como estão armazenados os dados, embora tenham relação direta com os mesmos. Como consequência, a cada alteração de dados, irá corresponder uma (ou mais) alterações em cada um dos índices, aumentando o esforço necessário ao sistema gestor de base de dados (SGBD) para gerir essa alteração, motivo pelo qual os índices não existam naturalmente para cada coluna. A estrutura usada para a elaboração do índice depende do SGBD e do tipo de dados das colunas usadas no índice: árvore B, árvore R, etc.

Não obstante o papel principal da tabela ser a de armazenamento de dados, é também utilizada como representação de relações, tipicamente de N para M. Nesse caso específico, essa tabela irá dispor obrigatoriamente de duas relações 1 para N — uma para a tabela N e outra para a tabela M — e, eventualmente, de atributos específicos à relação. Como consequência desta característica, este tipo de tabela nunca poderá conter linhas duplicadas. Um outro tipo de tabela especial — por não fazer armazenamento de dados — é a vista, cujas linhas são determinadas dinamicamente através de uma query (consulta) de tabelas reais (que armazenam os dados).

Chave primária

Chaves primárias (em inglês Primary Keys ou PK) sob o ponto de vista de um banco de dados relacional, referem-se às tuplas (conjuntos) de um ou mais campos, cujos valores, considerando a combinação de valores de todos os campos da tupla, nunca se repetem e que podem ser usadas como um índice para os demais campos da tabela do banco de dados. Em chaves primárias, não pode haver valores nulos nem repetição de tuplas.

Simplificando, quando a chave primária é simples, ou seja, é formada por um único campo da tabela, esse campo não pode ter dois ou mais registros de mesmo valor, e também não pode conter nenhum registro nulo. Se a chave primária é composta, ou seja, formada por mais de um campo, os valores de cada campo podem se repetir, mas não a combinação desses valores.

```
Exemplo: a tabela 'Livros_Autores' tem como chave primária (cod_livro, cod_autor).
```

Podem existir nessa tabela os registros:

```
(5, 9)
(5, 10)
(4, 9)
(9, 5)
```

Mas não podem existir dois registros (5, 9).

Ao criarmos uma chave primária, criamos automaticamente um índice do tipo aglomerado (CLUSTERED). Este é o tipo criado por padrão, mas caso já exista um índice desse tipo em sua tabela, então é necessário ser criado um índice do tipo não-aglomerado (NONCLUSTERED).

Podemos inserir uma chave primária durante ou após a criação da tabela. Com a tabela já criada, o campo que escolhermos para ser a chave primária deve ter a opção NOT NULL adicionada. Para inserirmos durante a criação usamos a seguinte sintaxe:

```
CREATE TABLE nome_tabela
(
    Codigo int NOT NULL PRIMARY KEY,
    Nome varchar(17)
)
```

Nessa estrutura escolhemos um índice do tipo CLUSTERED e resolvemos nomear com algum nome desejado a constraint de primary key. O índice poderia ser do tipo Nonclustered e poderíamos deixar o próprio SQL Server nomear a constraint, da seguinte forma:

```
CREATE TABLE nome_tabela
(
    Codigo int NOT NULL PRIMARY KEY,
    Nome varchar(17)
)
```

Além disso, pode-se definir a chave primária após a declaração dos campos, como segue:

```
CREATE TABLE nome_tabela
(
    campo1 <tipo> NOT NULL,
    campo2 <tipo> NOT NULL,
    campoX <tipo>,
    PRIMARY KEY (campo1, campo2)
)
```

Na definição de chave primária, usamos o comando ALTER TABLE para inserirmos e excluirmos uma primary key. As sintaxes respectivamente são:

```
ALTER TABLE nome_tabela
    ADD CONSTRAINT nome_constraint
    PRIMARY KEY NONCLUSTERED(nome_campo)

ALTER TABLE nome_tabela
    DROP CONSTRAINT nome_constraint
```

Uma chave candidata consiste em um atributo ou grupo de atributos cujo valor identifica unicamente cada tupla em uma relação e para o qual nenhum dos atributos pode ser removido sem destruir a identificação única.

Chave candidata

Uma chave candidata é um identificador único que garante que nenhuma tupla será duplicada; isto faz com que o relacionamento em algo denominado um multiconjunto, porque viola a definição básica de um conjunto. Uma chave pode ser composta, isto é, pode ser formada por vários atributos.

Ocorrem quando em uma relação existe mais de uma combinação de atributos para a identificação única do registro.

```
Ex: Matrícula, CPF, RG, Titulo Eleitor
```

Leve em consideração a regra de negócio: Para cada pedido pode existir um número infinito de itens(produtos), contudo o item não pode se repetir na lista de itens de um pedido, em caso da necessidade do mesmo item a quantidade deve ser alterada.

Considere a tabela abaixo:

```
pedidos(codPedido,valorTotal) PK - codPedido {Este número será único}
itensPedido(codPedido,codItem,quant,valorUnit) PK – codPedido
```

Suponhamos que a chave primária seja codPedido na tabela itensPedido, isso significa que este código deve ser único para os registro da tabela, contudo isso não pode ocorrer, pois exitem vários produtos para um pedido, neste caso outro campo deve ser candidato a chave também para unificar o registro.

Neste caso se definirmos como candidato o atributo codItem para compor a chave primária ficaria da seguinte forma:

```
PK - codPedido PK - codItem
```

Com esta chave candidata os itens do pedido não se repetirão e o codPedido poderá repetir,ficará conforme abaixo:

```
codPedido | codItem | quant | valorUnit
        1         1       2        2,50
        1         2       3        4,20
        1         3       3        1,50
```

Chave estrangeira

O conceito de Chave estrangeira em uso de banco de dados se refere ao tipo de relacionamento entre as tabelas de dados do banco de dados.

Uma chave estrangeira é chamada quando há o relacionamento entre duas tabelas.

Sempre em chave estrangeira vai haver relacionamentos entre tabelas, por exemplo, se uma tabela que tem uma chave primária de outra tabela.

Chave externas ou estrangeiras

Uma chave externa ou estrangeira é um atributo ou uma combinação de atributos numa relação R2, cujos valores são necessários para equivaler à chave primária de uma relação R1.

Uma chave estrangeira é um campo, que aponta para a chave primária de outra tabela. Ou seja, passa a existir uma relação entre essas duas tabelas. A finalidade da chave estrangeira é garantir a integridade dos dados referenciais, pois apenas serão permitidos valores que supostamente vão aparecer na Base de Dados.

Esse tipo de atributo não permite exclusão, modificação e/ou inserção de dados em tabelas que estejam dependentes umas das outras(foreign key), o que requer modificadores especiais, como cascade, por exemplo. Isso também exige uma maior atenção do administrador da base de dados, quanto à própria manipulação dos dados.

Fonte: wikipedia

Conceitos Fundamentais

Tabela, Entidade ***Table, Entity***	É uma estrutura em matriz bidimensional que representa o conjunto de informações de um mesmo tipo de elemento, indivíduo ou objeto. Contém campos que representam um tipo de informação e registros que representam todas as informações de um único elemento, indivíduo ou objeto. Ex: pessoas, produtos, notas fiscais, tipo de produto, classificação de animais, etc...
Linha, Registro ***Row / Line, Record***	É a linha da matriz que representa todas as informações distintas de um único elemento, indivíduo ou objeto. Ex: nome, endereço, código e data de nascimento do aluno “João de Almeida”

Coluna, Campo ***Column, Field***	É uma coluna da matriz que representa todo o conjunto de dados de uma única informação, e que define o significado daquele conjunto de dados. Para uma coluna é definido um único tipo de dado. Ex: data de nascimento, tipo *date*. Ex: nome do aluno, tipo *varchar*.
Chave Primária ***Primary Key (PK)***	É uma restrição (*Constraint*) aplicada à uma coluna ou conjunto de colunas, que tem o objetivo de criar um identificador único para cada um dos registros de uma entidade. Só é permitida uma chave primária por entidade. Nesta coluna / conjunto de colunas não pode haver repetição de dados.
Chave Estrangeira ***Foreign Key (FK)***	É uma restrição (*Constraint*) aplicada à uma coluna ou conjunto de colunas, que tem o objetivo de criar uma relação entre duas entidades, ligando um campo na entidade filha a um campo da entidade pai. Esta restrição garante que, o dado que for informado no campo *FK* deve existir obrigatoriamente no campo *PK* entidade pai.
NOT NULL	É a restrição de coluna que obriga o preenchimento de um dado na coluna à qual a restrição foi aplicada.
Chave Única ***Primary Key (PK)***	É uma restrição (*Constraint*) aplicada à uma coluna ou conjunto de colunas, que tem o objetivo de restringir a repetição de valores na(s) colunas associadas a esta restrição. Diferencia-se da chave primária por permitir a existência de mais de uma chave única por entidade e permitir o registro de NULL. Em muitos bancos de dados é possível utilizar de uma chave única para criar relação entre tabelas.

Tipos de Dados

O MySQL suporta tipos numéricos, data e hora e tipos string. Visto a extensão dos tipos de dados suportados pelo MySQL, esta apostila abordará somente os principais tipos.

Os campos são definidos no MySQL sendo necessário identificar sua precisão e regras de apresentação. Abaixo são demonstradas essas características, que serão usadas para a identificação dos tipos.

- **M**: Tamanho do campo, sendo que o máximo é 255.
- **D**: Número de casas decimais para tipos de ponto flutuante.
- **ZEROFILL:** Preenche automaticamente o campo numérico com zeros a esquerda até alcançar o tamanho máximo (M).
- **UNSIGNED**: Não permite a inserção de valores numéricos negativos.

Principais tipos Numéricos

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]` **-** Inteiro muito pequeno entre **-128 e 127** (com sinal) ou **0 a 255** (sem sinal). **BIT | BOOL | BOOLEAN** são sinônimos para TINYINT(1). Um tipo boolean verdadeiro será introduzido de acordo com o SQL-99.
- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]` - Inteiro pequeno entre **-32768 e 32767** ou **0 a 65535**.
- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]` **-** Inteiro médio entre **-8388608 a 8388607** ou **0 a 16777215**.

- **`INT[(M)] [UNSIGNED] [ZEROFILL]`** - Inteiro de tamanho normal entre **-2147483648 a 2147483647** ou **0 a 4294967295**. O tipo **INTEGER** é um sinônimo para **INT**.
- **`BIGINT[(M)] [UNSIGNED] [ZEROFILL]`** - Inteiro grande entre **-9223372036854775808 a 9223372036854775807** ou **0 a 18446744073709551615**.
- **`FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`** - Número de ponto flutuante pequeno (precisão simples) entre -3.402823466E+38 a -1.175494351E-38, 0 ou 1.175494351E-38 a 3.402823466E+38.
- **`DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`** - Número de ponto flutuante de tamanho normal (dupla-precisão) entre **-1.7976931348623157E+308 a -2.2250738585072014E-308, 0** ou **2.2250738585072014E-308 a 1.7976931348623157E+308**.

NOTA: Se você precisa armazenar um valor monetário, não use nenhum dos tipos acima pois o armazenamento ocorre com perda de precisão e arredondamentos para números muito grandes. Utilize o tipo **`DECIMAL`** ou **`NUMERIC`** que armazena em uma string sem perda de precisão.

Principais tipos de Data / Hora

- **`DATE`** - Data entre '1000-01-01' e '9999-12-31' no formato 'AAAA-MM-DD'.
- **`DATETIME`** - Data e hora e data entre '1000-01-01 00:00:00' e '9999-12-31 23:59:59'
- **`TIMESTAMP[(M)]`** – Um campo data-hora utilizando o formato Timestamp(unix) entre '1970-01-01 00:00:00' e o ano 2037, sendo exibidos nos formatos YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, ou YYMMDD, dependendo se M é 14, 12, 8 ou 6.
- **`TIME`** – Hora entre '-838:59:59' e '838:59:59'.
- **`YEAR[(2|4)]`** - Ano no formato de 2 ou 4 digitos.

Principais tipos String

- **`CHAR[(M)]`** – Caracter de tamanho M, atingindo o máximo de 255.
- **`VARCHAR[(M)]`** – Char de tamanho variável (simplificação para CHARACTER VARYING), de tamanho M, atingindo o máximo de 255.
- **`BLOB | TEXT`** - BLOB (Binary Large Object) ou TEXT com tamanho máximo de 65.535 caracteres.
- **`MEDIUMBLOB | MEDIUMTEXT`** - BLOB (Binary Large Object) ou TEXT com tamanho máximo de 16.777.215 caracteres.
- **`LONGBLOB | LONGTEXT`** - BLOB (Binary Large Object) ou TEXT com tamanho máximo de 4.294.967.295 caracteres.
- **`ENUM('valor1','valor2',...)`** - Tipo enumerado, ou seja, só aceita os valores definidos na lista pode ter até 65535 valores diferentes.
- **`SET('valor1','valor2',...)`** - Semelhante ao tipo enumerado podendo ter até 64 valores diferentes.
- **`DECIMAL[(M,D)] | NUMERIC[(M,D)]`** – Representa um número de M digitos sendo D decimais. M é limitado a 255 e D a 30. D não pose ser superior a M-2. **Ex.:** `salario DECIMAL(5,2)`

Tipos de Tabelas do MySQL

O MySQL tarabalha com vários tipos de tabela (*storage engines*), cada qual com suas aplicações, vantagens e desvantagens. A seguir veremos os tipos e suas funcionalidades:

- ISAM, MyISAM
- HEAP
- MERGE
- BDB, BerkeleyDB
- InnoDB

O perfil detalhado das tabelas do MySQL pode ser encontrado em http://dev.mysql.com/doc/refman/4.1/pt/storage-engines.html.

Detalhes com relação à criação de tabelas:

- o nome pode ter até 64 caracteres (numéricos e alfanuméricos), podendo conter números mas obrigatoriamente iniciando com caracteres
- o MySQL só diferenciará maiúsculas de minúsculas no nome da tabela caso o sistema operacional também diferencie (vide LINUX / UNIX /).

MyISAM

MyISAM é o tipo de tabela padrão no MySQL Versão 3.23 e é baseado no código ISAM, possuindo extensões úteis implementados pela MySQL.

O índice da tabela é armazenado em um arquivo com extensão .MYI e os dados são armazenados em um arquivo com a extensão .MYD.

Para reparar ou verificar o estado de uma tabela MyISAM, você pode utilizar o aplicativo myisamchk. Para mais informações consulte http://dev.mysql.com/doc/refman/4.1/pt/crash-recovery.html.

Ainda é possível compactar as tabelas MyISAM com myisampack para utilizar menos espaço, tornando somente leitura. Para mais informações consulte http://dev.mysql.com/doc/refman/4.1/pt/myisampack.html.

Para mais informações sobre MyISAM, consulte http://dev.mysql.com/doc/refman/4.1/pt/myisam-storage-engine.html.

HEAP

Tabelas HEAP usam índices hash e são armazenadas na memória. Isto as torna muito rápidas, mas se o MySQL falhar você irá perder todos os dados armazenados nela. HEAP é muito útil para tabelas temporárias.

As tabelas HEAP do MySQL utilizam hashing 100% dinâmico sem áreas em excesso. Não há espaços extras necessários para listas livres. Tabelas HEAP também não têm problemas com deleção + inserção, o que normalmente é comum em tabelas com hash:

```
mysql> CREATE TABLE test TYPE=HEAP SELECT ip,SUM(downloads) AS down
    ->    FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MERGE

Uma tabela MERGE (também conhecida como tabela MRG_MyISAM) é uma coleção de tabelas MyISAM idênticas que podem ser usada como uma. Você só pode fazer SELECT, DELETE, e UPDATE da coleção de tabelas. Se você fizer um DROP na tabela MERGE, você só está apagando a especificação de MERGE.

Com tabelas idênticas queremos dizer que todas as tabelas são criadas com informações de colunas e chaves idênticas. Você não pode fundir tabelas nas quais as colunas são empacotadas de forma diferente, não tenham as mesmas colunas ou tenham as chaves em ordem diferente.

Mais informações sobre tabelas merge, veja http://dev.mysql.com/doc/refman/4.1/pt/merge-storage-engine.html.

DBD, BerkeleyDB

BerkeleyDB, disponível em http://www.sleepycat.com/ tem provido o MySQL com um **mecanismo de armazenamento transacional**. O suporte para este mecanismo de armazenamento está incluído na distribuição fonte do MySQL a partir da versão 3.23.34 e está ativo no binário do MySQL-Max. Este mecanismo de armazenamento é chamado normalmente de BDB.

Tabelas BDB podem ter maior chance de sobrevivência a falhas e também são capazes de realizar operações COMMIT e ROLLBACK em transações. A distribuição fonte do MySQL vem com uma distribuição BDB que possui alguns pequenos patchs para faze-lo funcionar mais suavemente com o MySQL. Você não pode usar uma versão BDB sem estes patchs com o MySQL.

InnoDB

O InnoDB provê o MySQL com um mecanismo de armazenamento seguro **com transações** (compatível com ACID) com **commit**, **rollback**, e recuperação em caso de falhas. InnoDB faz bloqueio a nível de registro e também fornece uma leitura sem bloqueio em SELECT em um estilo consistente com Oracle. Estes recursos aumentam a performance e a concorrência de multi-usuários. InnoDB é o primeiro gerenciador de armazenamento no MySQL que suportam restrições FOREIGN KEY.

InnoDB foi desenvolvido para obter o máximo de performance ao processar grande volume de dados, sendo usado na produção de vários sites com banco de dados grandes e que necessitam de alto desempenho. O famoso site de notícias Slashdot.org utiliza InnoDB. Mytrix, Inc. armazena mais de 1 TB de dados em InnoDB, em outro site trata uma carga média de 800 inserções/atualizações por segundo em InnoDB.

Para maiores informações sobre o tipo, consulte http://dev.mysql.com/doc/refman/4.1/pt/innodb.html.

Comparativo de Tabelas MySQL

Tipo	Desempenho	Objetivo	Confiabi-lidade	Quantidade de dados	Chave Estrangeira	Transa-cional	ACID
MyISAM	+ Alta	Velocidade sem complexidade	Baixa	Média	Não	Não	Não
HEAP	Muito Alta	Tabelas Temporária	Alta sem preservação de dados	Média	Não	Não	Não
MERGE	+ Alta	Visão única de tabelas idênticas		Média	Não	Não	Não
DBD	- Alta	Desempenho Transacional	Alta	Grande	Não	Sim	Não
InnoDB	- Alta	Alta Confiabilidade	Alta	Grande >1TB	Sim	Sim	Sim

SQL – *STRUCTURED QUERY LANGUAGE*

A **SQL** (originalmente SEQUEL) é uma **Linguagem de Consulta Estruturada** (***Structured Query Language***), declarativa, para a utilização com bancos de dados no padrão relacional.

A SQL é desenvolvida desde a década de 1970 e foi concebida inicialmente pela IBM. Hoje a SQL é um padrão ANSI (American National Standards Institute – www.ansi.org/).

A linguagem possui diversos subconjuntos que categorizam sua temática, de acordo com as operações efetuadas sobre um banco de dados.

- DDL - Linguagem de Definição de Dados
- DCL - Linguagem de Controle de Dados
- DML - Linguagem de Manipulação de Dados
- DQL - Linguagem de Consulta de Dados
- DTL - Linguagem de Transação de Dados

NOTA: Todos os exemplos envolvidos neste livro aplicam-se principalmente ao banco de dados MySQL.

DDL - Linguagem de Definição de Dados

O subconjunto da SQL chamado de DDL (*Data Definition Language* - Linguagem de Definição de Dados) agrupa as instruções que permitem a definição de objetos no banco de dados, como o próprio banco de dados, tabelas, visões e índices, tendo como comandos principais:

- **CREATE**: cria um objeto.
- **ALTER**: altera um objeto existente.
- **DROP**: apaga um objeto.

Os comandos podem aplicar-se a estruturas como:

- **DATABASE**: banco de dados
- **TABLE**: tabela, podendo criar, remover ou adicionar elementos.
- **VIEW**: visão, permitido criar ou remover, sendo relacionados às tabelas.
- **INDEX**: índices de restrição ou ordenação, sendo inerentes às tabelas.

Banco de Dados – CREATE / ALTER / DROP DATABASE

A criação de banco de dados é descrita pela sintaxe abaixo:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification [, create_specification] ...]

    create_specification:
         [DEFAULT] CHARACTER SET charset_name
         | [DEFAULT] COLLATE collation_name

ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification [, alter_specification] ...

    alter_specification:
         [DEFAULT] CHARACTER SET charset_name
         | [DEFAULT] COLLATE collation_name

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

Tomando como base o banco de dados de nome **escola**, a criação da base de dados seria

```
CREATE DATABASE escola
  DEFAULT CHARACTER SET utf8;

CREATE DATABASE IF NOT EXISTS escola
  DEFAULT CHARACTER SET utf8;
```

Alterar a base, por exemplo, mudando o DEFAULT CHARACTER SET para latin1

```
ALTER DATABASE escola
```

```
  DEFAULT CHARACTER SET latin1;
```

Excluindo a base de dados.

```
DROP DATABASE escola;


DROP DATABASE IF EXISTS escola;
```

Definindo a base de dados como a ativa, numa conexão, para os comandos executados através desta.

```
USE escola;
```

Tabelas – CREATE / ALTER / DROP TABLE

A criação de tabelas e seus elementos é descrita pela sintaxe (simplificada) abaixo:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options];


create_definition:
    col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string']
      [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
      [STORAGE {DISK|MEMORY|DEFAULT}]
      [reference_definition]
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
      [index_option] ...
  | {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
      [index_name] [index_type] (index_col_name,...)
      [index_option] ...
  | CHECK (expr)


table_option:
    ENGINE [=] [ InnoDB | ISAM | MyISAM | HEAP ]
  | AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'      | MAX_ROWS [=] value      | MIN_ROWS [=] value
  | PASSWORD [=] 'string'     | UNION [=] (tbl_name[,tbl_name]...)
```

A alteração das tabelas é descrita pela sintaxe (simplificada) abaixo:

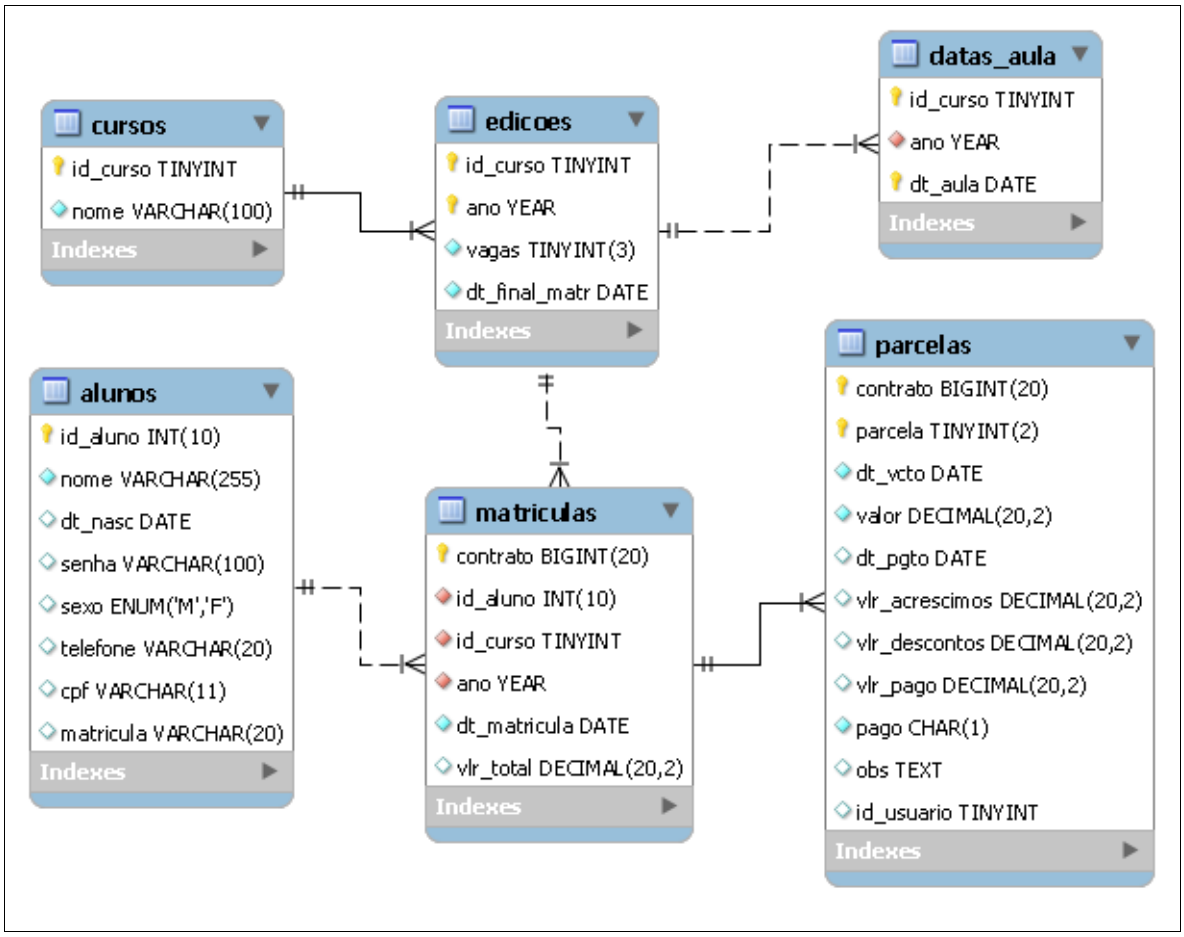
```
ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl_name
    [alter_specification [, alter_specification] ...]


alter_specification:
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name ]
  | ADD [COLUMN] (col_name column_definition,...)
```

```
    | ADD {INDEX|KEY} [index_name]
          [index_type] (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]] PRIMARY KEY
          [index_type] (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]]
          UNIQUE [INDEX|KEY] [index_name]
          [index_type] (index_col_name,...) [index_option] ...
    | ADD FULLTEXT [INDEX|KEY] [index_name]
          (index_col_name,...) [index_option] ...
    | ADD SPATIAL [INDEX|KEY] [index_name]
          (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]]
          FOREIGN KEY [index_name] (index_col_name,...)
          reference_definition
    | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
    | CHANGE [COLUMN] old_col_name new_col_name column_definition
          [FIRST|AFTER col_name]
    | MODIFY [COLUMN] col_name column_definition
          [FIRST | AFTER col_name]
    | DROP [COLUMN] col_name
    | DROP PRIMARY KEY   | DROP {INDEX|KEY} index_name     | DROP FOREIGN KEY fk_symbol
    | MAX_ROWS = rows    | DISABLE KEYS    | ENABLE KEYS    | RENAME [TO|AS] new_tbl_name
```

A exclusão de tabelas é descrita pela sintaxe (simplificada) abaixo:

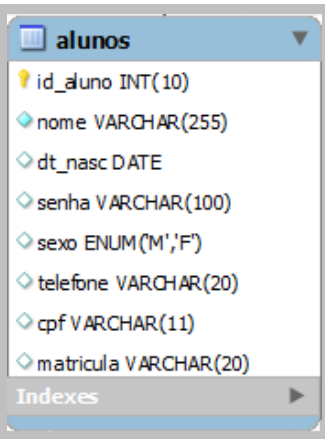
```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] …
```

Tomando por base o fragmento do modelo ER abaixo, escreveremos os *scripts* de criação das tabelas.



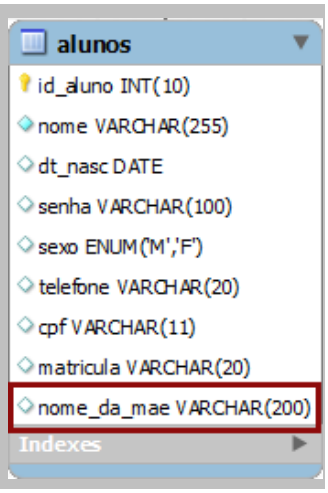
Exemplo de CREATE com a tabela alunos:

```
CREATE TABLE alunos (
  id_aluno int(10) unsigned NOT NULL AUTO_INCREMENT,
  nome varchar(255) NOT NULL,
  dt_nasc date,
  senha varchar(100),
  sexo enum('M','F'),
  telefone varchar(20),
  cpf varchar(11),
  matricula varchar(20),
  PRIMARY KEY (id_aluno)
) ENGINE=InnoDB
DEFAULT CHARSET=latin1;
```



Exemplos de ALTER com a tabela alunos:

```
ALTER TABLE alunos
  ADD nome_mae varchar(100) NOT NULL;

ALTER TABLE alunos
  CHANGE nome_mae nome_da_mae varchar(100) NOT NULL;

ALTER TABLE alunos
  MODIFY nome_da_mae varchar(200);

ALTER TABLE alunos
  ADD CONSTRAINT alunos_PK PRIMARY KEY(id_aluno);
```



Índices - INDEX

Índices são estruturas que agem como ponteiros apontando para linhas das tabelas, permitindo que uma consulta possa determinar rapidamente quais as linhas que correspondem a uma condição na cláusula WHERE, recuperando os valores destas linhas.

Os índices podem ser de chave primária, chaves estrangeiras e colunas, sendo que os dois primeiros são geridos automaticamente pela maioria dos bancos de dados.

A criação de índices de colunas é descrita pela sintaxe (simplificada) abaixo:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option] ...
```

```
index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value   | WITH PARSER parser_name   | COMMENT 'string'
```

Exemplo de CREATE INDEX a tabela alunos:

```
CREATE INDEX alunos_nome_IDX ON alunos (nome);

CREATE UNIQUE INDEX alunos_cpf_UK ON alunos (cpf);

CREATE INDEX matriculas_ano_aluno_IDX ON matriculas (ano,id_aluno);
```

Visões - VIEWS

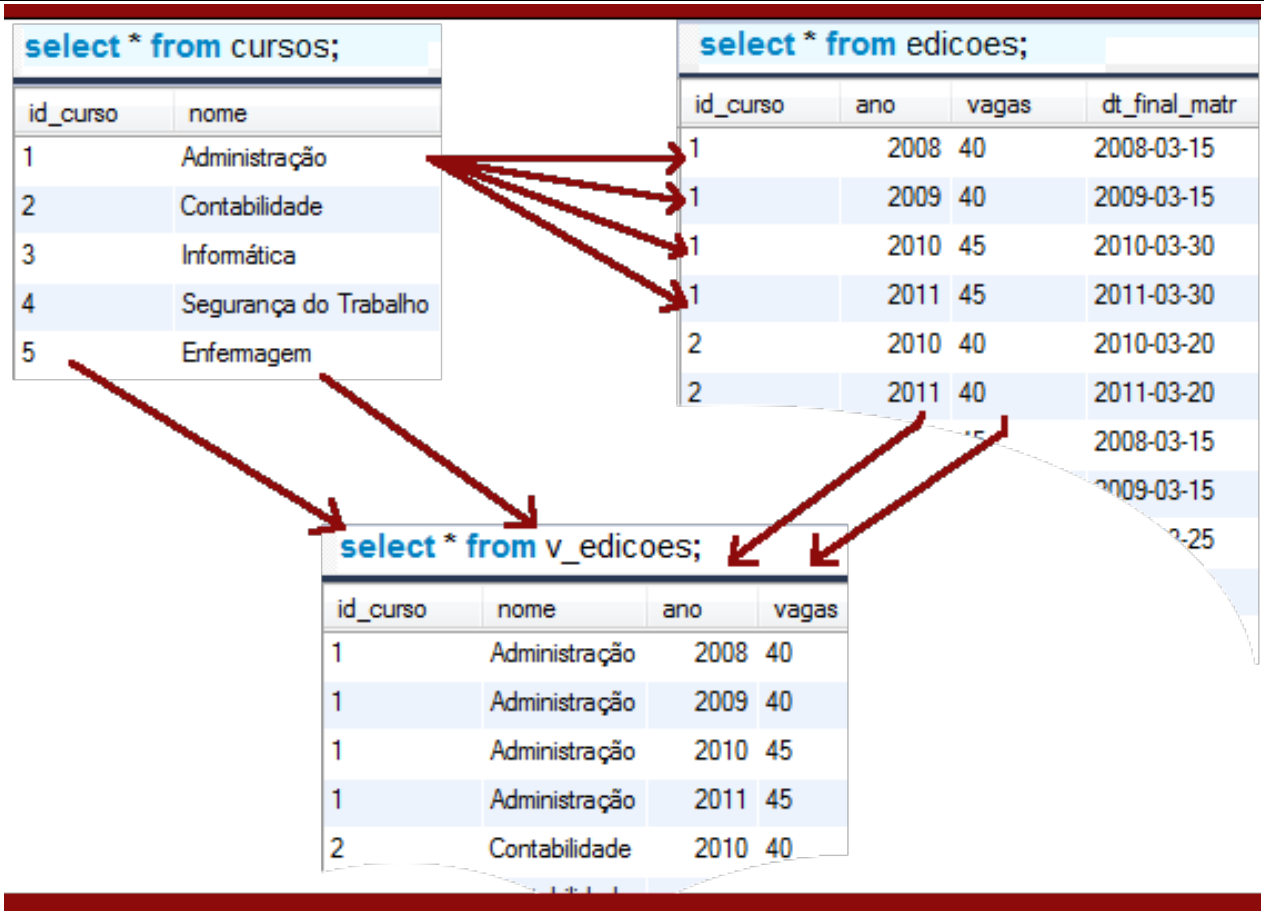
Visões (***VIEWS***) são consultas armazenadas, predefinidas, que quando chamadas apresentam seu resultado como uma tabela. Um ***view*** é considerada uma tabela virtual.

A sintaxe de CREATE VIEW é descrita abaixo:

```
CREATE [OR REPLACE]
    VIEW view_name [(column_list)]
AS select_statement;
```

Exemplo de CREATE VIEW:

```
CREATE OR REPLACE VIEW v_edicoes
AS SELECT c.id_curso, c.nome, e.ano, e.vagas
   FROM cursos c, edicoes e
   WHERE c.id_curso =e.id_curso;
```



Exercícios de DDL

Para estes exercícios você precisará iniciar o servidor SGBD MySQL e o MySQL Workbench (Para saber como utilizar, veja o Apêndice - Usando o MySQL Workbench). Abra o Workbench em modo *SQL Development*. No *SQL Editor*, crie uma nova *SQL Tab* para executar seus comandos *SQL Script*.

1. Exclua a base de dados **ESCOLA**.
2. Crie a base de dados **ESCOLA** com o ***caracter set utf8***.
3. Crie as entidades ***cursos***, ***edicoes***, ***alunos***, ***matriculas***, ***parcelas*** e ***datas_aula*** (nesta ordem).
4. Crie a FK entre ***edicoes*** e ***cursos***, alterando a endidade ***edicoes***.
5. Crie a FK entre ***datas_aula*** e ***edicoes***, alterando a endidade ***datas_aula***.
6. Crie a FK entre ***edicoes*** e ***matriculas***, alterando a endidade ***matriculas***.
7. Crie a FK entre ***alunos*** e ***matriculas***, alterando a endidade ***matriculas***.
8. Crie a FK entre ***matriculas*** e ***parcelas***, alterando a endidade ***parcelas***.
9. Altere a entidade ***matriculas*** adicionando o campo "***historico***" do tipo *mediumtext*.
10. Altere a entidade ***matriculas*** adicionando o campo "***id_usuario***" do tipo *tinyint(3) unsigned*.
11. Crie a FK entre ***matriculas*** e ***usuarios***, alterando a endidade ***matriculas***.
12. Crie o índice do tipo coluna, com o nome ***parcelas_dt_vcto_idx*** para a coluna ***dt_vcto*** da entidade ***parcelas***.
13. Crie o índice do tipo coluna, com o nome ***parcelas_dt_pgto_idx*** para a coluna ***dt_pgto*** da entidade ***parcelas***.
14. Crie uma view chamada ***v_parcelas_impagas*** com a segunte cláusula SQL

```
select a.id_aluno, a.nome, p.contrato, p.parcela, p.valor, p.dt_vcto
from alunos a, matriculas m, parcelas p
where a.id_aluno=m.id_aluno
  and m.contrato=p.contrato
  and p.pago<>'S';
```

15. Verifique os dados da view executando a consulta abaixo:

```
select * from v_parcelas_impagas;
```

DCL - Linguagem de Controle de Dados

A DCL (*Data Control Language* - Linguagem de Controle de Dados) controla os aspectos de autorização de acesso a recursos e dados do banco de dados de acordo com licenças concedidas a usuários, de forma a tornar o acesso aos dados mais seguros, tendo como comandos principais:

- **GRANT**: concede um privilégio de acesso.
- **REVOKE**: revoga um privilégio de acesso.

Os comandos podem aplicar-se a estruturas como:

- **DATABASE**: banco de dados, permitindo acessar.
- **TABLE**: tabela, permitido ver, inserir ou excluir registros, alterar ou remover views.
- **VIEW**: visão, permitido ver, inserir ou excluir registros, alterar ou remover views.

Níveis e Tipos de Privilégios

Os privilégios são divididos em níveis dependendo do contexto ao qual precisam ser aplicados. A concessão de permissões pode acontecer a nível *Global, Database, Table, Column* e *Stored Routines*.

Nota: o tópico *Stored Routines* não é abordado neste livro.

Nível de Privilégio Global - *Global*

São priviégios administrativos concedidos ou revogados, a todos os bancos de dados de um servidor. São associados utilizando a sintaxe:

```
ON *.*
```

Exemplos:

```
GRANT ALL ON *.* TO 'joao'@'localhost';
GRANT SELECT, INSERT ON *.* TO 'joao'@'localhost';
```

Os privilégios CREATE TABLESPACE, CREATE USER, FILE, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN, e SUPER são administrativos e somente podem ser associados neste nível.

Nível de Privilégio de Banco de Dados - *Database Privileges*

São priviégios aplicados a todas as tabelas de uma banco de dados. São associados utilizando a sintaxe:

```
ON db_name.*
```

Exemplos:

```
GRANT ALL ON escola.* TO 'joao'@'localhost';
GRANT SELECT, INSERT ON escola.* TO 'joao'@'localhost';
```

Os privilégios CREATE, DROP, EVENT, GRANT OPTION, e LOCK TABLES são associados no nível de banco de dados.

Nível de Privilégio de Tabelas - *Table Privileges*

São priviégios aplicados a tabelas específicas e são associadas a todas as colunas desta tabela.ão associados utilizando a sintaxe:

```
ON db_name.tbl_name
```

Exemplos:

```
GRANT ALL ON escola.alunos TO 'joao'@'localhost';
GRANT SELECT, INSERT ON escola.alunos TO 'joao'@'localhost';
```

Os tipos de privilégios possíveis neste nível são ALTER, CREATE VIEW, CREATE, DELETE, DROP, GRANT OPTION, INDEX, INSERT, SELECT, SHOW VIEW, TRIGGER e UPDATE.

Nível de Privilégio de Colunas - Column Privileges

São priviégios aplicados a colunas específicas de uma tabela. São associados utilizando o nome das colunas entre parêntesis.

Exemplo:

```
GRANT SELECT (nome), INSERT (id_aluno, nome) ON escola.alunos TO 'joao'@'localhost';
```

Pode ser aplicado às cláusulas INSERT, SELECT e UPDATE.

Tipos de Privilégio

Os tipos de privilégios (`priv_type`) mais comuns para *GRANT* e *REVOKE:*

- **SELECT:** habilita ao uso do comando *SELECT*;
- **DELETE:** habilita ao uso do comando *DELETE*;
- **INSERT:** habilita ao uso do comando *INSERT*;
- **UPDATE:** habilita ao uso do comando *UPDATE*;
- **DROP:** habilita a exclusão de *databases, tables e views*;
- **ALL [PRIVILEGES]:** concede todos os privilégios especificados pelo no nivel de acesso *GRANT OPTION.*
- **ALTER:** habilita ao uso do comando *ALTER TABLE*;
- **CREATE:** habilita a criação de *databases* e *tables*;
- **CREATE VIEW:** habilita a criação e alteração de *views*;
- **CREATE TEMPORARY TABLES:** habilita ao uso do comando *CREATE TEMPORARY TABLE;*
- **CREATE USER:** habilita ao uso do comando *CREATE USER, DROP USER, RENAME USER, e REVOKE ALL PRIVILEGES*
- **GRANT OPTION:** permite que os privilégios obtidos possam ser concedidos ou revogados de outras contas;
- **INDEX:** habilita a criação e exclusão de índices;

Acesse a lista completa em http://dev.mysql.com/doc/refman/5.5/en/grant.html#grant-privileges.

Concedendo Permissão - GRANT

O comando GRANT concede um privilégio de acesso.

A sintaxe básica de GRANT é descrita abaixo:

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_specification [, user_specification] ...
    [WITH with_option ...]

object_type:
    TABLE | FUNCTION | PROCEDURE

priv_level:
    *  | *.*  | db_name.*  | db_name.tbl_name  | tbl_name  | db_name.routine_name

user_specification:
    user IDENTIFIED BY [PASSWORD] 'password'

with_option:
    GRANT OPTION  | MAX_QUERIES_PER_HOUR count  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count  | MAX_USER_CONNECTIONS count
```

Nota: Os *wildcards* são caracteres que substituem valores definidos por genéricos, que podem ser processados de acordo com sua função:

***** - associado ao nível de permissão, pode representar qualquer banco de dados ou tabela.

% - associado às contas de usuário, pode representar qualquer usuário ou host.

Exemplos:

O exemplo abaixo cria o usuário joao, permitindo o acesso de qualquer host (%), dando-lhe a senha 123456 e concedendo o acesso de SELECT a todas as tabelas do banco de dados escola.

```
GRANT select ON escola.* TO 'joao'@'%'
   IDENTIFIED BY '123456';
```

O exemplo abaixo concede todos os privilégios ao usuário joao, sobre a base de dados escola, com opção deste poder conceder ou revogar acesso de outros usuários (permissão de gerência).

Nota: ***WITH GRANT OPTION*** – o usuário que for concedida a permissão de gerência, terá poderes de conceder ou revogar permissões de outros usuários.

```
GRANT ALL PRIVILEGES
ON escola.*
TO 'joao'@'localhost'
WITH GRANT OPTION;
```

Outros exemplos:

```
GRANT insert ON escola.livros TO 'rudinei'@'%';
GRANT select, insert, update ON escola.livros TO 'rudinei'@'%';
GRANT delete ON escola.livros TO 'rudinei'@'%';
```

Revogando Permissão - REVOKE

O comando REVOKE revoga um privilégio de acesso.

A sintaxe básica de REVOKE é descrita abaixo:

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...

REVOKE PROXY ON user
    FROM user [, user] ...
```

Exemplos:

O exemplo abaixo revoga a permissão de INSERT sobre todas as bases de dados e tabelas, da conta 'joao'@'localhost'.

```
REVOKE INSERT ON *.* FROM 'joao'@'localhost';
```

O exemplo abaixo revoga todos os privilégios e opção de grant dos usuário 'joao'@'localhost' e 'maria'@'localhost'.

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'joao'@'localhost', 'maria'@'localhost';
```

Outros exemplos:

```
REVOKE insert ON escola.* FROM 'joao'@'localhost';
REVOKE insert, update ON escola.livros FROM 'joao'@'localhost';
REVOKE delete ON escola.livros FROM 'rudinei'@'%';
```

Criando Contas de Usuários - CREATE USER / ALTER USER / SET PASSWORD

As contas de usuário normalmente são definidas com a associação do nome de usuário e o *host* pelo qual este pode acessar. A conta de usuário é definida pela sintaxe:

```
'user_name'@'host_name'
```

Podemos utilizar o *wildcard* % para definições genéricas.

Exemplos:

```
'joao'@'%.dominio.com'
'joao'@'200.174.33.%'
'joao'@'%'
```

Nota: O MySQL não suporta *wildcards* no nome do usuário. Para referenciar anonimamente o usuário, utiliza-se o vazio.

```
''@'%.dominio.com'
```

```
''@'localhost'
```

Um usuário pode ser criado juntamente com a concessão do privilégio (GRANT) ou utilizando-se do comando CREATE USER, definido pela sintaxe abaixo.

```
CREATE USER user_specification[, user_specification] …

user_specification:
    user [ IDENTIFIED BY [PASSWORD] 'password' ]
```

Exemplos:

```
CREATE USER 'joao'@'localhost';
CREATE USER 'joao'@'%';
CREATE USER 'joao'@'localhost' IDENTIFIED BY '123#456';
CREATE USER 'joao'@'%' IDENTIFIED BY '123#456';
```

No exemplo abaixo, a conta 'carminha'@'localhost' será criada automaticamente.

```
GRANT ALL ON escola.* TO 'carminha'@'localhost';
```

A alteração de conta de usuário permite que possa ser expirada a conta do usuário sem a necessidade de remover suas permissões ou a própria conta.

```
ALTER USER user_specification
    [, user_specification] ...

user_specification:
    user PASSWORD EXPIRE
```

Exemplo:

```
ALTER USER 'carminha'@'localhost' PASSWORD EXPIRE;
```

A alteração de senhas pode ser efetuada pelo comando *SET PASSWORD* com a sintaxe abaixo

```
SET PASSWORD [FOR user] =
    {
        PASSWORD('cleartext password')
      | OLD_PASSWORD('cleartext password')
      | 'encrypted password'
    }
```

Exemplo:

```
SET PASSWORD FOR 'carminha'@'localhost' = PASSWORD('palavraMagica');
```

Exercícios de DCL

1. Crie a conta de usuário **administrador** com permissão de acesso somente da máquina local, com a senha **super**.
2. Crie a conta de usuário **convidado** com permissão de acesso de qualquer máquina e para a máquina local, com a senha **guest**.
3. Crie a conta de usuário **funcionario** com permissão de acesso na rede **'10.1.1.%'**, com a senha **personal**.
4. Conceda a permissão de **SELECT, INSERT, UPDATE e DELETE** sobre todo os bancos de dados para a conta do usuário **administrador**.
5. Conceda a permissão de **SELECT, INSERT** sobre a entidade **livros** do banco de dados **escola** para a conta do usuário **convidado** do *host* local.
6. Conceda a permissão de **SELECT, INSERT, UPDATE e DELETE** sobre todo o banco de dados **escola** para a conta do usuário **funcionario**.
7. Remova a permissão de **INSERT** concedida para a conta do usuário **convidado**.
8. Conecte no banco de dados **escola** com o usuário **convidado** e, utilizando o *table edit*, tente inserir um registro na tabela livros. Qual o resultado que o banco de dados retorna?

DML - Linguagem de Manipulação de Dados

A DML (*Data Manipulation Language* - Linguagem de Manipulação de Dados) é o subconjunto de instruções da linguagem SQL que é utilizado para realizar inclusões, consultas, alterações e exclusões de dados nas entidades de um banco de dados, tendo como comandos principais:

- ***INSERT***: utilizado para adicionar registros novos numa entidade.
- ***UPDATE***: utilizado para atualizar registros em uma entidade.
- ***DELETE***: utilizado para remover registros de uma entidade.
- ***TRUNCATE***: utilizado para limpar/eliminar todos os registros uma entidade.

Inserindo Dados - *INSERT*

Sintaxe básica do *INSERT*:

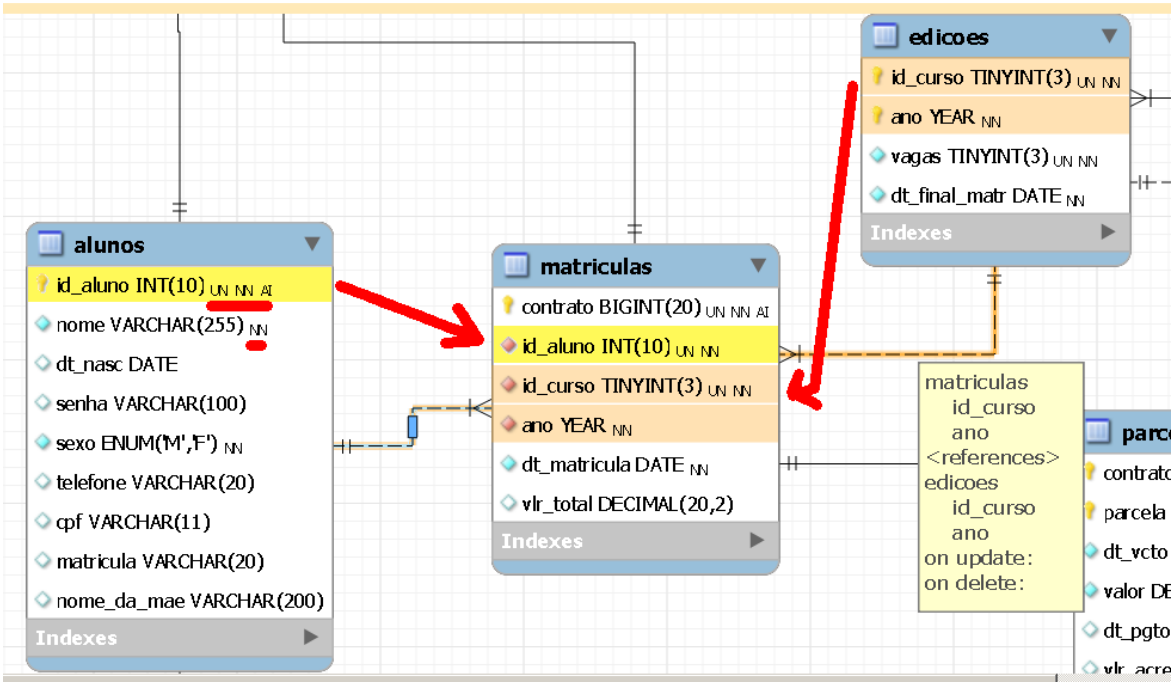
```
INSERT
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...;
```

O *insert* deve ser escrito declarando-se todas as colunas que precisarei afetar. Todas as colunas que possuem a restrição *NOT NULL* devem estar obrigatoriamente listadas na cláusula, salvo se for do tipo *auto_increment* ou possuir valor *default*.

Exemplos:

```
INSERT INTO alunos
(id_aluno, nome, dt_nasc, senha, sexo, telefone, cpf, matricula, nome_da_mae)
VALUES
(1999810,'Joao', '1983-10-13', NULL, 'M', NULL, '', 1235464, NULL);
```

Considerando o modelo abaixo, podemos identificar:



- Na entidade alunos:
  - id_aluno é a chave primária e *auto_increment.*
  - nome é *not null.*
  - sexo é not null, mas possui valor *default*, o que não é apresentado pelo modelo.
- Na entidade matrículas:
  - id_aluno é chave estrangeira, portanto precisa existir na entidades alunos para poder ser inserido na entidade matriculas.
  - id_curso e ano formam outra chave estrangeira, portanto precisam existir na entidades edições, na mesma combinação, para poder ser inserido na entidade matriculas.

```
INSERT INTO alunos
    (id_aluno, nome)
VALUES
    (1999810,'Joao');
-- 1 row(s) affected

select * from alunos where id_aluno=1999810;
```

id_aluno	nome	dt_nasc	senha	sexo	telefone	cpf	matricula	nome_da_mae
1999810	Joao	NULL	NULL	M	NULL	NULL	NULL	NULL

```
INSERT INTO alunos
     (nome)
VALUES
     ('JOAO');
-- 1 row(s) affected

SELECT LAST_INSERT_ID();
```

last_insert_id()
1999811

```
SELECT * FROM escola.alunos WHERE id_aluno=1999811;
```

id_aluno	nome	dt_nasc	senha	sexo	telefone	cpf	matricula	nome_da_mae
1999811	JOAO	(NULL)	(NULL)	M	(NULL)	(NULL)	(NULL)	(NULL)

Algumas regras da INSERÇÃO:

- Toda **STRING, DATA, HORA, DATAHORA** deve, obrigatoriamente, estar envolta de aspas.
- Todo número **INTEIRO** pode ser informado com ou sem aspas.
- Números **FRACIONÁRIOS** devem ser escritos com **PONTO DECIMAL**, e não vírgulas, **não devendo** estar.
- Campos **AUTO_INCREMENT** não devem ser informados no *insert*, pois assumirá o valor informado.
- Para não informar um valor, o atribua o valor ***NULL***, mas nunca deixe um campo vazio.
- Na inserção, campos de **chave estrangeira** devem ter seu valor existentes na entidade de origem da chave.

Atualizando Dados - *UPDATE*

Sintaxe básica do *UPDATE*:

```
UPDATE table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count];
```

Exemplos:

```
UPDATE alunos
SET
  telefone = '555133669955',
  cpf = '98765431221'
WHERE
  id_aluno = 1999810;
```

Excluindo Dados - *DELETE*

Sintaxe básica do *DELETE*:

```
DELETE FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count];
```

Exemplos:

```
DELETE FROM alunos
WHERE
  id_aluno = 1999810;
```

Eliminando todos os Dados - *TRUNCATE*

Sintaxe básica do *TRUNCATE*:

```
TRUNCATE [TABLE] tbl_name;
```

Exemplos:

```
TRUNCATE TABLE alunos;
```

Filtros *WHERE*

Os filtros são de suma importância no desenvolvimento de instruções SQL, pois eles que controlam quais os dados serão afetados pela instrução SQL, seja UPDATE, DELETE ou SELECT.

Os filtros são definidos na cláusula WHERE da instrução SQL:

```
SELECT * FROM LIVROS WHERE ID_LIVRO=1;

UPDATE livros SET titulo='book' WHERE ID_LIVRO=1;

DELETE FROM livros WHERE ID_LIVRO=1;
```

A cláusula ***where*** pode ser regida por uma série de instruções que visam identificar a range de dados que serão tratados pelo comando, definidas pelos operadores lógicos e de comparação abaixo ilustrados:

Operadores lógicos:

- e → **AND**
- ou → **OR**
- precedência → **( )**

Operadores de comparação:

- igualdade → **=**
- diferente → **<>**
- maior → **>**
- maior ou igual → **>=**
- menor → **<**
- menor ou igual → **<=**
- contido num conjunto → **IN(...)**
- não contido num conjunto → **NOT IN(...)**
- entre dois limitadores → **BETWEEN lim_1 AND lim_2**
- é nulo → **IS NULL**
- Não é nulo → **IS NOT NULL**

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO=1;
```

id_livro	titulo	isbn	ano	sumario
1	Corações Sujos	9788535919370	2000	A colônia japonesa e

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO<>1;
```

id_livro	titulo	isbn
2	Agosto	97885
3	Rápido e Devagar - Duas Formas de Pensar	97885
4	Guerra e paz	97885
5	1984	97885
6	Ulisses	97885

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO>5;
```

id_livro	titulo
6	Ulisses
7	Lolita
8	O som e a fúria
9	O homem invisível
10	Al faro
11	A ilíada e a odisséia
12	Orgulho e preconceito

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO<10;

SELECT * FROM LIVROS
  WHERE ID_LIVRO>=6;
```

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO<=9;
```

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO IN (1,3,6,10);
```

id_livro	titulo	isbn
1	Corações Sujos	97885359.
3	Rápido e Devagar - Duas Formas de Pensar	978853900
6	Ulisses	978856350
10	Al faro	(NULL)

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO NOT IN (1,3,6,10);
```

id_livro	titulo	isb
2	Agosto	978
4	Guerra e paz	978
5	1984	978
7	Lolita	978
8	O som e a fúria	978

```
SELECT * FROM LIVROS
  WHERE ID_LIVRO BETWEEN 3 AND 6;
```

```
SELECT * FROM LIVROS
  WHERE TITULO = 'DOM QUIXOTE';
```

id_livro	titulo
46	Dom Quixote

```
SELECT * FROM LIVROS
  WHERE TITULO LIKE 'LIVRO%';
```

id_livro	titulo
21	Livro das Religiões, O
38	Livro do Desassossego
57	LIVRO

```
SELECT * FROM LIVROS
  WHERE TITULO LIKE '%ICA';
```

id_livro	titulo
28	Genealogia da Moral: Uma Polêmica

```
SELECT * FROM LIVROS
  WHERE TITULO LIKE '%ORT%';
```

id_livro	titulo
29	Morte e Vida Severina
44	Cronica De Uma Morte Anunciada
45	A Morte e a Morte de Quincas Berro DÁgua

```
SELECT * FROM LIVROS
  WHERE TITULO LIKE '%M%R%T%';
```

id_livro	titulo
14	Mundo de Sofia, O - Romance da História da Filosofia
29	Morte e Vida Severina
31	Memórias de Minhas Putas Tristes
35	Salomão, O Homem Mais Rico que já Existiu
44	Cronica De Uma Morte Anunciada
45	A Morte e a Morte de Quincas Berro DÁgua
47	Poesia Completa de Alberto Caeiro
50	Cartas a um jovem escritor
58	Sofrimentos Do Jovem Werther, Os

```
DELETE FROM LIVROS
  WHERE ID_LIVRO IN (1,3,6,10);
```

```
DELETE FROM LIVROS
  WHERE TITULO LIKE 'HAM%';
```

```
ALTER TABLE LIVROS ADD emprestar ENUM('S','N');

UPDATE LIVROS
  SET EMPRESTAR='S'
  WHERE ID_LIVRO IN (1,3,6,10);

UPDATE LIVROS
  SET SUMARIO='Há alguns séculos atrás, sobre as ameias do castelo de Elsinor, na
          Dinamarca, os guardas reais viram o fantasma do rei Hamlet, que tinha ... '
  WHERE TITULO = 'HAMLET';
```

Condições complexas pode ser definidas utilizando os operadores AND (condição obrigatória) e OR (condições alterantivas)

```
SELECT *
FROM LIVROS
  WHERE TITULO LIKE 'LIVRO%'
    AND ID_LIVRO=21;
```

id_livro	titulo
21	Livro das Religiões, 0

```
SELECT *
FROM LIVROS
  WHERE TITULO LIKE 'LIVRO%'
    OR ID_LIVRO=10;
```

id_livro	titulo
10	Al faro
21	Livro das Religiões, 0
38	Livro do Desassossego
57	LIVRO

```
SELECT *
  FROM LIVROS
  WHERE ID_LIVRO IN (12,38,41,47)
  OR TITULO LIKE 'LIVRO%'
  OR ISBN='9788520925164';
```

id_livro	titulo
12	Orgulho e preconceito
21	Livro das Religiões, 0
38	Livro do Desassossego
41	Tieta do Agreste
47	Poesia Completa de Alberto Caeiro
57	LIVRO

```
SELECT *
  FROM LIVROS
  WHERE ID_LIVRO IN (12,38,41,47)
    AND (
         TITULO LIKE 'LIVRO%'
         OR
         ISBN='9788520925164'
    );
```

id_livro	titulo
12	Orgulho e preconceito
38	Livro do Desassossego

Operadores e Expressões Aritméticas

A SQL possibilita a execução de cálculos aritméticos associados ou não aos retornos de dados das estruturas dos bancos de dados. São operadores aritméticos:

- operador de subtração

+ operador de adição

***** operador de multiplicação

/ operador de divisão

DIV operador de divisão inteira

% ou MOD operador de módulo

Precedência dos operadores

- (menos unário)

***, /, DIV, %, MOD**

-, +

Exemplos:

```
SELECT 1+2, 1+2*3, (1+2)*3;

SELECT 11/3, 11 DIV 3, 11 MOD 3;

SELECT contrato, parcela, valor,
    valor*0.1 AS "Multa 10%"
FROM parcelas;
```

1+2	1+2*3	(1+2)*3
3	7	9

11/3	11 DIV 3	11 MOD 3
3.6667	3	2

contrato	parcela	valor	Multa 10%
1	5	52.89	5.289
1	6	52.87	5.287
1	50	133.00	13.300
1	51	133.00	13.300
1	52	133.00	13.300
2	1	343.77	34.377
2	2	343.77	34.377
2	3	343.77	34.377

Exercícios de DML

Responda as questões abaixo usando instruções SQL da DML.

1. Insira na entidade **cursos** um novo curso chamado **Culinária**, com o código **6**.
2. Insira na entidade **edicoes** uma nova edição para o curso acima, no ano de **2013**, máximo de vagas em **10** e a data limite de matrícula em **10 de Abril.**
3. Insira na entidade **alunos** um novo aluno, colocando seu **nome completo**, sua **data de nascimento**, seu **sexo**, sem informar o código do aluno (*auto_increment*).
4. Efetue uma consulta para identificar qual código o último *insert* gerou.
5. Insira na entidade **matriculas** uma nova matrícula, para o aluno inserido na questão 3, na edição inserida na questão 2, com a data de matrícula na data atual e o valor de R$ 1.600,00.
6. Atualize o número de vagas para 15 e a data limite de matrícula para 30/04, da edição inserida na questão 2.
7. Atualize o valor da matrícula para R$ 1.599,96.
8. Exclua a matrícula inserida na questão 5.
9. Exclua o aluno inserido na questão 3.
10. Exclua a edição inserida na questão 2.
11. Exclua o curso inserido na questão 1.

DQL - Linguagem de Consulta de Dados

A DQL (*Data Querie Language* - Linguagem de Consulta de Dados) é o subconjunto de instruções da linguagem SQL que é utilizado para realizar buscas e listagem dos dados nas entidades de um banco de dados, tendo como comando principal:

- ***SELECT***: utilizado para efetuar consultas em uma ou mais entidades.

Sintaxe geral simplificada da instrução:

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    select_expr [, select_expr ...]
[FROM table_references
[WHERE where_condition]
[GROUP BY {col_name | expr | position}
      [ASC | DESC]]
[HAVING where_condition]
[ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [FOR UPDATE]]
```

SELECT – Consultando Dados em Uma Única Entidade

A cláusula *select* possui uma série recursos, permitindo que diferentes combinações possam ser executadas, tornando-a sempre única, a saber:

- * - *wildcard* utilizado na cláusula SELECT que significa **todas as colunas**.
- % - *wildcard* utilizado na cláusula WHERE que significa **qualquer valor**, ou **qualquer parte** de um valor.

Exemplos:

```
SELECT * FROM alunos;                  -- seleciona todas as colunas da tabela alunos
SELECT * FROM escola.edicoes;          -- seleciona todas as colunas da tabela
                                       -- edicoes do banco de dados escola
SELECT m.* FROM escola.matriculas m;   -- seleciona todas as colunas exclusivamente
                                       -- da entidade referenciada pelo apelido m
```

Renome, Apelido e Referência Completa

Afim de facilitar a escrita da cláusula SQL é suportado o recurso de “apelido”, o que reduz consideravelmente a escrita da instrução.

Como forma de exibição ou desambiguação, podemos renomear uma coluna para o resultado de uma cláusula SQL

```
select id_aluno as ID, nome as Aluno
from alunos;
```

Id	Aluno
1	Carlos
2	João

Na consulta acima, apenas no resultado, o título das colunas id_aluno e nome foram modificados.

Quando precisamos definir uma formatação especial para uma coluna, com separação de palavras, utilizamos então aspas duplas.

```
select a.id_aluno as "ID", a.nome as "Aluno Especial"
from alunos as a;
```

ID	Aluno Especial
1	Carlos
2	João

Às entidades podemos empregar o uso de apelidos, muitas vezes necessário para a desambiguação de colunas com o mesmo nome em cláusulas com *join* e também como forma de reduzir a escrita. No exemplo acima, a entidade **alunos** foi apelidada de **a**, e os campos da entidade aluno passaram a ser referenciados com o **a.** na frente, como demonstrado com as colunas **a.id_aluno** e **a.nome**.

```
select a.id_aluno as "ID",
   a.nome as "Nome"
from escola.alunos as a;
```

ID	Nome
1	Carlos
2	João
3	Maria

No exemplo acima estamos usando uma referência completa identificando de qual base de dados uma entidade pertence. Para isso verificamos a referência **escola.alunos**, indicando que a entidade **alunos** utilizada é da base de dados **escola**.

Funções de Agregação e Agrupamentos

As funções de agregação e os agrupamentos servem para efetuar contagens, somas, médias, etc... em grupos de dados retornando dados sumarizados.

A sintaxe básica dos agrupamentos é

```
SELECT [col_name,] FUNÇÃO_AGREGADA(col_name) [, FUNÇÃO_AGREGADA(col_name)]
FROM table_references
[WHERE where_condition]
[GROUP BY {col_name [, col_name]}
[HAVING where_condition]
[ORDER BY {col_name [, col_name]}
```

As funções de agregação são:

- **MAX** – maior valor do grupo de valores de uma coluna.
- **MIN** – menor valor do grupo de valores de uma coluna.
- **AVG** – a média dos valores de um grupo de valores de uma coluna.
- **SUM** – a soma dos valores de um grupo de valores de uma coluna.
- **COUNT** - a contagem dos valores não nulos de um grupo de valores de uma coluna.

Exemplos:

```
SELECT
    COUNT(contrato) AS Contratos,
    SUM(valor) AS Total,
    SUM(vlr_pago) AS TotalPago,
    AVG(valor) AS ValorMedio,
    MIN(valor) AS MenorValor,
```

```
    MAX(valor) AS MaiorValor
FROM parcelas;
```

Contratos	Total	TotalPago	ValorMedio	MenorValor	MaiorValor
297	88283.83	54156.18	297.251953	52.87	486.41

Descrição: conte o número de contratos, some o valor dos contratos, some o valor pago, calcule o valor médio, obtenha o menor e o maior valor de parcela, de todos os contratos da entidade parcelas.

Agrupamentos

Os agrupamentos definem conjuntos de sumarização, ou seja, colunas/valores pelos quais as colunas com função de agregação deverão ser sumarizadas.

Os grupos de valores são definidos pelos valores distintos das colunas definidas pela cláusula **GROUP BY**.

Os filtros pré-agrupamento são definidos utilizando-se a cláusula **WHERE**.

Os filtros pós-agrupamento são definidos utilizando-se a cláusula **HAVING**.

Exemplos:

```
SELECT contrato, SUM(vlr_pago) AS total_pago
FROM parcelas
GROUP BY contrato;
```

contrato	total_pago
1	328.48
2	1810.18
3	1290.88
4	(NULL)
5	305.38

Descrição: para cada contrato, some todos os valores da coluna vlr_pago e agrupe pelo contrato.

```
SELECT contrato,
      (SUM(vlr_pago)/SUM(valor))*100 AS percentual_pago
FROM parcelas
GROUP BY contrato;
```

contrato	percentual_pago
1	45.856600
2	87.762473
3	54.240707
4	(NULL)
5	34.667605

Descrição: para cada contrato, divida a soma dos valores da coluna vlr_pago pela soma dos valores da coluna valor e multiplique o resultado por 100. Agrupe pelo contrato.

```
SELECT id_curso AS Curso,
       COUNT(id_aluno) AS Alunos,
       SUM(vlr_total) AS Total
FROM matriculas
GROUP BY id_curso;
```

Curso	Alunos	Total
1	19	31414.80
2	8	19913.60
3	22	36556.43

Os filtros pré-agrupamento delimitam os registros sobre os quais as funções de agrupamento atuarão sumarizando.

```
SELECT id_curso AS Curso,
       COUNT(id_aluno) AS Alunos,
       SUM(vlr_total) AS Total
FROM matriculas
WHERE id_curso IN (2,3)
```

Curso	Alunos	Total
2	8	19913.60
3	22	36556.43

```
GROUP BY id_curso ;
```

Descrição: para cada curso, conte quantos id_aluno válidos tem, some o valor total. Limite os registros aos cursos de código 2 e 3. Agrupe pelo código do curso.

Os filtros pós-agrupamento atuam sobre os resultados sumarizados, delimitando o que será exibido da sumarização efetuada.

```
SELECT id_curso AS Curso,
       COUNT(id_aluno) AS Alunos,
       SUM(vlr_total) AS Total
FROM matriculas
WHERE id_curso IN (2,3)
GROUP BY id_curso
HAVING COUNT(id_aluno) < 10;
```

Curso	Alunos	Total
2	8	19913.60

Descrição: para cada curso, conte quantos id_aluno válidos tem, some o valor total. Limite os registros aos cursos de código 2 e 3. Agrupe pelo código do curso. Exiba somente os cursos cuja quantidade de alunos seja inferior a 10.

Exercícios de DQL - Agrupamentos

Exercícios de DQL em uma entidade:

1. Obtenha os nomes de todos os alunos.
2. Obtenha o código e o nome de todos os alunos.
3. Obtenha o nome de todas os alunos em ordem alfabética.
4. Obtenha o nome do aluno código 15.
5. Obtenha o nome e a data de nascimento de todas os alunos com código maior que 2.
6. Obtenha o nome, o código e a matricula de todos os alunos que possuem numero de matricula.
7. Selecione o código da disciplina, o código do curso e a carga horária de todas as disciplinas do curriculo do curso 1.
8. Selecione o código dos alunos, o código do curso e a data de matrícula de todas as matrículas efetuadas entre entre o dia 1 e 30 de março de 2010.
9. Selecione o id e a quantidade de vagas de todas as edições de curso com 45 vagas.
10. Selecione o id e a quantidade de vagas de todas as edições de curso com mais de 45 vagas.
11. Selecione o id e a quantidade de vagas de todas as edições de curso com menos de 45 vagas.
12. Selecione o id e a quantidade de vagas de todas as edições com numero de vagas entre 20 e 30 vagas.
13. Selecione o id e a quantidade de vagas de todas as edições em que o número de vagas seja diferente de 45.

Exercícios de DQL em uma entidade com agrupamentos:

14. Obtenha o maior código da tabela alunos
15. Obtenha o próximo código (maior + 1) de alunos.

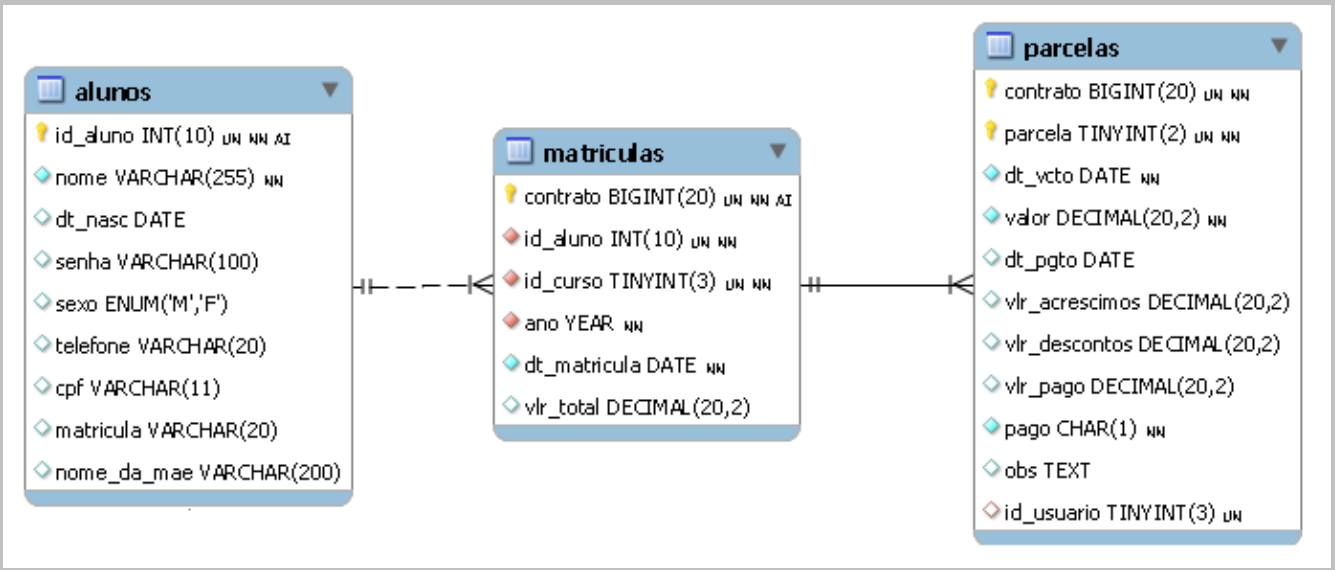
16. Selecione a soma total das cargas horarias das disciplinas do currículo do curso 2;
17. Selecione a média de preço das disciplinas do currículo do curso 3.
18. Selecione o id do curso, a soma das cargas horárias e a soma dos valores das disciplinas, agrupadas pelo id do curso, de todos os currículos cadastrados.
19. Selecione a quantidade de alunos matriculados no curso 3 (da tabela matrículas).
20. Calcule a soma das cargas horárias dos currículos, agrupado pelo curso e semestre
21. Busque a quantidade de alunos do sexo masculino e do sexo feminino, agrupando pela coluna sexo.
22. Selecione o código do curso, o ano, o valor total matriculado e o valor médio matriculado, de todas as matrículas desde que tenham mais de 3 alunos matriculados no curso/ano.

SELECT / JOINS – Consultando Dados em Mais de Uma Entidade

A consulta de dados apenas em uma entidade não é muito útil no dia-a-dia. A regra comum é que dados são cruzados pelos relacionamentos dos bancos de dados obtendo informações relacionadas das mais diversas lógicas de negócio.

O relacionamento numa consulta DQL é obtido através da comparação de chaves relacionadas (PK da entidade pai com a FK relacionada da entidade filha) como definido na criação das *contraints* de *foreign key*.

```
USE escola;
SELECT * FROM alunos a;        -- seleciona todas as colunas da entidade alunos
SELECT * FROM matriculas m;    -- seleciona todas as colunas da entidade matriculas
```




```
SELECT a.nome, m.id_curso, m.ano
FROM alunos a, matriculas m
WHERE a.id_aluno = m.id_aluno;       -- JOIN TRADICIONAL

SELECT a.nome, m.id_curso, m.ano
FROM alunos a, matriculas m
WHERE a.id_aluno = m.id_aluno        -- JOIN TRADICIONAL
  AND a.nome like '%A%';             -- FILTROS
```

```
SELECT a.nome, m.id_curso, m.ano, m.contrato, p.parcela, p.dt_vcto, p.valor
FROM alunos a, matriculas m, parcelas p
WHERE a.id_aluno = m.id_aluno        -- JOIN TRADICIONAL
  AND m.contrato = p.contrato
  AND a.nome like '%A%'              -- FILTROS
  AND p.pago = 'N'
ORDER BY a.id_aluno, m.ano, p.parcela
```

Sintaxe JOIN

A sintaxe JOIN é uma forma declarativa de demostrar os relacionamentos da consulta SQL, permitindo outras funções de consulta como OUTER JOIN (exclusiva).

```
SELECT a.nome, m.id_curso, m.ano
FROM alunos a
  JOIN matriculas m ON a.id_aluno = m.id_aluno;

SELECT a.nome, m.id_curso, m.ano
FROM alunos a
  JOIN matriculas m ON a.id_aluno = m.id_aluno
WHERE a.nome like '%A%';             -- FILTROS

SELECT a.nome, m.id_curso, m.ano, m.contrato, p.parcela, p.dt_vcto, p.valor
FROM alunos a
  JOIN matriculas m ON a.id_aluno = m.id_aluno
  JOIN parcelas p ON m.contrato = p.contrato
WHERE a.nome like '%A%'              -- FILTROS
  AND p.pago = 'N'
ORDER BY a.id_aluno, m.ano, p.parcela
```

Exercícios de DQL – Sintaxe JOIN

Resolva os exercícios de DQL abaixo utilizando a sintaxe tradicional e a sintaxe JOIN:

1. Obtenha o nome e data de nascimento do aluno, o código do curso, o ano, a data de matrícula e o valor total de todas as matrículas efetuadas pelos alunos de código inferior a 150.
2. Selecione o nome do curso, o ano e o número de vagas nas edições, de todos os cursos, retornando o resultado ordenado pelo ano da edição e após pelo nome dos cursos.
3. Liste o nome do autor e o nome dos livros que este escreveu.
4. Exiba o nome do curso, o nome das disciplinas deste curso com as respectivas cargas horárias do currículo, ordenando pelo nome do curso e após pelo semestre da disciplina.
5. Liste o nome, o id e o nome do curso de todas as pessoas e cursos, devidamente relacionados, cujo nome da pessoa termine com "A" e esteja matriculada no curso 1.

6. Liste a quantidade de pessoas matriculadas no curso 2 e que o nome contenha a letra A ('%A%').
7. Selecione o código da disciplina, o código do curso, o nome da disciplina e a carga horária de todas as disciplinas do curso 1;
8. Selecione o id e o nome do curso, a soma das cargas horárias e a soma dos valores das disciplinas, agrupadas pelo id do curso.
9. Selecione o código e nome das pessoas e o código do curso de todas as matrículas efetuadas entre entre o dia 1 e 30 de março de 2008.
10. Selecione o id, o nome e a quantidade de vagas de todos os cursos com 45 vagas.
11. Selecione o id, o nome e a quantidade de vagas de todos os cursos com 45 vagas e a quantidade de matriculas para cada ano.
12. Selecione o id, o nome, o ano da edição e a quantidade de vagas de todos os cursos com menos de 45 vagas cujo o número de vagas seja superior à quantidade de matrículas.
13. Selecione o id, o nome e a quantidade de vagas de todos os cursos com numero de vagas entre 41 e 49 vagas.
14. Selecione o id, o nome e a quantidade de vagas de todos os cursos em que o número de vagas seja diferente de 45.
15. Selecione o nome do curso e a média de preço das disciplinas do curso 3.
16. Selecione o nome do curso e a soma total das cargas horarias das disciplinas do curso 2.

Operações Complexas com SELECT

Muitas vezes em nossas operações precisamos confirmar e validar regras de negócio complexas após várias operações com DML / DQL.

Um exemplo simples deste cenário: Uma nota fiscal tem um total registrado numa entidade ***notafiscais*** e seus itens em uma entidade ***notafiscal_itens***. Como saber e validar se a soma total dos itens da nota fiscal equivale ao total da ota fiscal?

A solução para esta questão pode passar por um processo de consultas e cálculos programados em um sistema ou por consultas SQL.

No nosso modelo de dados “escola” tema deste livro, temos uma situação similar. A entidade ***matriculas*** possui um campo chamado ***vlr_total*** que representa o valor total de uma matrícula. A entidade ***parcelas*** possui a abertura deste valor em termo de parcelas cobradas mensalmente. Como saber se o valor total da matrícula equivale ao total incluído para aquele contrato? Analisemos por partes

A consulta abaixo retorna no valor total dos contratos 1 e 2:

```
SELECT m.contrato, m.vlr_total
FROM matriculas m
WHERE m.contrato IN (1,2);
```

contrato	vlr_total
1	317.32
2	2062.59

A consulta abaixo retorna no valor total dos contratos 1 e 2 do ponto de vista da soma das parcelas:

```
SELECT p.contrato, SUM(p.valor)
FROM parcelas p
WHERE p.contrato IN (1,2)
GROUP BY p.contrato;
```

contrato	SUM(p.valor)
1	716.32
2	2062.59

Numa análise visual, podemos ver que o contrato 1 possui uma diferença de valores entre o total e as suas parcelas. Então juntaremos as duas consultas numa só:

```
SELECT m.contrato, m.vlr_total, SUM(p.valor)
FROM matriculas m
  JOIN parcelas p ON m.contrato=p.contrato
WHERE m.contrato IN (1,2)
GROUP BY m.contrato, m.vlr_total;
```

contrato	vlr_total	sum(p.valor)
1	317.32	716.32
2	2062.59	2062.59

Na próxima consulta, o nosso desejo é que apenas os contratos que apresentem divergência entre o total e as parcelas sejam apresentados, então filtramos com o uso da cláusula *HAVING*.

```
SELECT m.contrato, m.vlr_total, SUM(p.valor)
FROM matriculas m
  JOIN parcelas p ON m.contrato=p.contrato
WHERE m.contrato IN (1,2)
GROUP BY m.contrato, m.vlr_total
HAVING m.vlr_total<>SUM(p.valor);
```

contrato	vlr_total	SUM(p.valor)
1	317.32	716.32

Podemos constatar então que com auxílio de uma consulta SQL relativamente simples podemos identificar dados que quebram regras de negócio.

Exercícios de DQL – Operações Complexas

SELECT / SUBSELECT – Consultando Dados com *Selects* Aninhados

É a técnica de gerar um conteúdo baseado em uma consulta SQL que serve como fonte de dados para outra consulta SQL na mesma instrução.

Consideremos o seguinte cenário: Precisamos obter toda a lista de parcelas dos contratos cujo valor total registrado na matrícula seja diferente da soma dos valores das parcelas daquele contrato. Já vimos como obter os contratos com esse tipo de problema, mas como listar as parcelas?

```
SELECT m.contrato, m.vlr_total, SUM(p.valor)
FROM matriculas m
  JOIN parcelas p ON m.contrato=p.contrato
GROUP BY m.contrato, m.vlr_total
HAVING m.vlr_total<>SUM(p.valor);
```

contrato	vlr_total	SUM(p.valor)
1	317.32	716.32
27	2422.41	2421.77
30	2918.36	2917.36
33	1541.54	1585.54
41	2062.59	2062.48

Então para identificarmos apenas os contratos, tiramos as colunas que não precisamos `(m.vlr_total e SUM(p.valor)`:

```
SELECT m.contrato
FROM matriculas m
  JOIN parcelas p ON m.contrato=p.contrato
GROUP BY m.contrato, m.vlr_total
HAVING m.vlr_total<>SUM(p.valor);
```

contrato
1
27
30
33
41

Tendo como origem de dados o SQL acima, podemos então buscar as parcelas com base nos contratos identificados, usando a consulta superior como *subselect*.

```
SELECT *
FROM parcelas
WHERE contrato IN
(
    SELECT m.contrato
    FROM matriculas m
      JOIN parcelas p ON m.contrato=p.contrato
    GROUP BY m.contrato, m.vlr_total
    HAVING m.vlr_total<>SUM(p.valor)
);
```

contrato	parcela	dt_vcto	valor	dt_pgto	vlr_acresc	vlr_des
1	50	2011-06-01	133.00	(NULL)	(NULL)	(NULL)
1	51	2011-07-01	133.00	(NULL)	(NULL)	(NULL)
1	52	2011-08-01	133.00	(NULL)	(NULL)	(NULL)
27	01	2010-04-08	403.74	2010-06-06	48.05	0.00
27	02	2010-05-08	403.74	2010-06-23	42.80	0.00
27	03	2010-06-08	403.10	2010-05-15	0.00	0.00
27	04	2010-07-08	403.74	2010-07-01	0.00	0.00

```
-- Obs.: Visualização parcial do retorno.
```

Na DQL acima, o código do contrato do *subselect* entrou como valores de um conjunto validados pela operação *IN*, podendo-se comparar com a instrução abaixo

```
SELECT *
FROM parcelas
WHERE contrato IN ( 1, 27, 30, 33, 41, 50, 55 )
```

O *subselect* pode também ser utilizado como uma tabela possível de *JOIN* com entidades existentes na base de dados. A SQL abaixo mostra a mesma solução da questão anterior utilizando este tipo de composição.

```
SELECT pa.*
FROM parcelas pa
    JOIN (
         SELECT m.contrato
         FROM matriculas m
           JOIN parcelas p ON m.contrato=p.contrato
         GROUP BY m.contrato, m.vlr_total
         HAVING m.vlr_total<>SUM(p.valor)
         ) X
    ON pa.contrato=X.contrato;
```

contrato	parcela	dt_vcto	valor	dt_pgto	vlr_acresc	vlr_des
1	50	2011-06-01	133.00	(NULL)	(NULL)	(NULL)
1	51	2011-07-01	133.00	(NULL)	(NULL)	(NULL)
1	52	2011-08-01	133.00	(NULL)	(NULL)	(NULL)
27	01	2010-04-08	403.74	2010-06-06	48.05	0.00
27	02	2010-05-08	403.74	2010-06-23	42.80	0.00
27	03	2010-06-08	403.10	2010-05-15	0.00	0.00
27	04	2010-07-08	403.74	2010-07-01	0.00	0.00

```
-- Obs.: Visualização parcial do retorno.
```

Na consulta acima, o ***subselect*** tornou-se uma tabela de nome X que foi juntada à entidade parcelas (***pa***) pela relação de igualdade entre **contrato** de **pa** e **X**. O resultado, neste caso, é exatamente igual à consulta anterior com ***IN***.

A vantagem da utilização de ***subselects*** em ***joins*** é que permite a utilização de colunas do ***subselect*** no resultado do ***select*** superior, como exemplificado abaixo:

```
SELECT pa.* , x.vlr_total, x.soma
FROM parcelas pa
    JOIN (
          SELECT m.contrato, m.vlr_total, SUM(p.valor) AS soma
          FROM matriculas m
            JOIN parcelas p ON m.contrato=p.contrato
          GROUP BY m.contrato, m.vlr_total
          HAVING m.vlr_total<>SUM(p.valor)
          ) X
    ON pa.contrato=X.contrato;

-- As colunas x.vlr_total e x.soma colocadas na cláusula select
-- são originadas no subselect.
```

contrato	parcela	dt_vcto	valor	dt_pgto	vlr_acresc	vlr_des	vlr_pago	pago	ob	i	vlr_total	soma
1	01	2008-04-08	52.89	2008-03-31	0.00	0.00	52.89	S		L	317.32	716.32
1	02	2008-05-08	52.89	2008-06-28	5.87	0.00	58.76	S		L	317.32	716.32
1	03	2008-06-08	52.89	2008-06-30	2.75	0.00	55.64	S		L	317.32	716.32
1	04	2008-07-08	52.89	2008-07-05	0.00	0.00	52.89	S		L	317.32	716.32
1	05	2008-08-08	52.89	2008-07-18	0.00	0.00	52.89	S		L	317.32	716.32
1	06	2008-09-08	52.87	2008-09-26	2.54	0.00	55.41	S		L	317.32	716.32
1	50	2011-06-01	133.00	(NULL)	(NULL)	(NULL)	(NULL)	N		L	317.32	716.32
1	51	2011-07-01	133.00	(NULL)	(NULL)	(NULL)	(NULL)	N		L	317.32	716.32
1	52	2011-08-01	133.00	(NULL)	(NULL)	(NULL)	(NULL)	N		L	317.32	716.32
27	01	2010-04-08	403.74	2010-06-06	48.05	0.00	451.79	S		L	2422.41	2421.77

Podemos ainda utilizar as colunas do *subselect* para efetuar filtros, como no exemplo abaixo:

```
SELECT pa.* , x.vlr_total, x.soma
FROM parcelas pa
JOIN ( SELECT m.contrato , m.vlr_total, SUM(p.valor) AS soma
       FROM matriculas m JOIN parcelas p ON m.contrato=p.contrato
       GROUP BY m.contrato, m.vlr_total
       HAVING m.vlr_total<>SUM(p.valor) ) X
ON pa.contrato=x.contrato
WHERE x.vlr_total > x.soma;
```

Exercícios de DQL - *Subselects*

Resolva os exercícios de DQL abaixo utilizando SUBSELECTS, a sintaxe tradicional e a sintaxe JOIN:

1. Liste o código e o nome dos alunos que tenham empréstimos de livros em seus nomes, sem repetir os nomes.

```
-- CERTO
SELECT a.id_aluno, a.nome
FROM alunos a
WHERE a.id_aluno
  IN (SELECT e.id_aluno FROM emprestimos e)
ORDER BY 2;
-- 100 rows

-- ERRADO
SELECT a.id_aluno, a.nome
FROM alunos a
  JOIN (SELECT e.id_aluno FROM emprestimos e) X
ON a.id_aluno=x.id_aluno
ORDER BY 2;
-- 112 rows
```

id_aluno	nome
15	Abadi C
444	Adahir Me
540	Affonso Te
250	Aldirio Qu
62	Aldoni Pi
245	Alirio Sa

id_aluno	nome
15	Abadi C
444	Adahir Me
540	Affonso Te
250	Aldirio Qu
250	Aldirio Qu
62	Aldoni Pi

2. Liste o nome do curso, o ano da edição, a quantidade de alunos matriculados e soma dos valores das parcelas agrupados por curso e edição. A soma dos valores e a quantidade de alunos deverão ser obtidos por *subselect*.

```
-- CERTO
SELECT c.nome, e.ano, X.num_alunos,
       Y.total
FROM cursos c
  JOIN edicoes e ON c.id_curso=e.id_curso
  JOIN (
    SELECT COUNT(m.id_aluno)
      AS num_alunos, m.ano, m.id_curso
    FROM matriculas m
    GROUP BY m.ano, m.id_curso
  ) X
  ON e.id_curso=X.id_curso AND e.ano=X.ano
  JOIN (
    SELECT SUM(p.valor) AS total, m.ano, m.id_curso
    FROM matriculas m JOIN parcelas p ON m.contrato=p.contrato
    GROUP BY m.ano, m.id_curso
  ) Y
  ON e.id_curso=Y.id_curso AND e.ano=Y.ano;
```

nome	ano	num_alunos	total
Administração	2008	6	10711.84
Informática	2008	5	8368.33
Administração	2009	6	13803.48
Informática	2009	6	9469.46
Administração	2010	4	5870.43
Contabilidade	2010	4	8240.16
Informática	2010	6	11274.41
Administração	2011	3	1427.94
Contabilidade	2011	4	11672.44
Informática	2011	5	7587.43

```
-- ERRADO
SELECT c.nome, e.ano, X.num_alunos, X.total
FROM cursos c
  JOIN edicoes e
    ON c.id_curso=e.id_curso
  JOIN (
    SELECT
     COUNT(m.id_aluno) AS num_alunos,
     SUM(p.valor) AS total,
     m.ano, m.id_curso
    FROM matriculas m
     JOIN parcelas p
     ON m.contrato=p.contrato
    GROUP BY m.ano, m.id_curso
) X
   ON e.id_curso=X.id_curso AND e.ano=X.ano;
```

nome	ano	num_alunos	total
Administração	2008	39	10711.84
Informática	2008	30	8368.33
Administração	2009	36	13803.48
Informática	2009	36	9469.46
Administração	2010	24	5870.43
Contabilidade	2010	24	8240.16
Informática	2010	36	11274.41
Administração	2011	18	1427.94
Contabilidade	2011	24	11672.44
Informática	2011	31	7587.43

SELECT / UNION – Consultando Dados em União de Resultados

A união (*UNION*) é um recurso da DQL que permite unir, em uma mesma visão de resultado, conjuntos de dados que podem ou não ter qualquer relação entre si. Enquanto a junção (*JOIN*) coloca colunas de entidades diferentes na mesma *querie*, a união une linhas de registros de conjuntos diferentes de resultados.

Atente para os seguintes exemplos:

Liste o número das parcelas, o vencimento, o valor e a situação de pagamento das parcelas do contrato de código 2.

```
SELECT parcela, dt_vcto, valor, pago
FROM parcelas
WHERE contrato=2
ORDER BY parcela;
```

parcela	dt_vcto	valor	pago
01	2009-04-08	343.77	S
02	2009-05-08	343.77	S
03	2009-06-08	343.77	S
04	2009-07-08	343.77	S
05	2009-08-08	343.77	S
06	2009-09-08	343.74	N

Liste o total pago e o total devido das parcelas do contrato 2.

```
SELECT SUM(valor) AS pago
FROM parcelas
WHERE contrato=2 AND pago='S'
GROUP BY pago;

SELECT SUM(valor) AS devido
FROM parcelas
WHERE contrato=2 AND pago='N'
GROUP BY pago;
```

pago
1718.85

No exemplo abaixo, com a utilização do *UNION*, teremos todas

```
SELECT parcela, dt_vcto, valor, pago
```

parcela	dt_vcto	valor	pago
1	2009-04-08	343.77	S
2	2009-05-08	343.77	S
3	2009-06-08	343.77	S
4	2009-07-08	343.77	S
5	2009-08-08	343.77	S
6	2009-09-08	343.74	N
999	Total Pago	1718.85	S
999	Total Devido	343.74	N

```
  FROM parcelas
  WHERE contrato=2
UNION
SELECT 999,'Total Pago',SUM(valor),pago
  FROM parcelas
  WHERE contrato=2 AND pago='S'
  GROUP BY pago
UNION
SELECT 999,'Total Devido',SUM(valor),pago
  FROM parcelas
  WHERE contrato=2 AND pago='N'
  GROUP BY pago
ORDER BY parcela;
```

Com o *UNION* podemos estruturar relatórios totalmente em SQL, desde que respeitadas as seguintes regras:

- Todas as consultas devem ter o mesmo número de colunas
- As colunas devem ser do mesmo tipo ou de tipos compatíveis de transformação. Alguns SGBD, como o MYSQL, permitem a utilização de números e texto na mesma coluna.
- Colunas vazias podem ser criadas para suplantar falta de dados em alguma das consultas.
- Os títulos da primeira consulta serão os títulos do resultado da consulta de união.

O exemplo abaixo cria um resultado de relatório com fontes de dados diferentes (tabela alunos e tabela parcelas) onde:

- a primeira linha de ordem 1 retorna o nome do aluno,
- as linhas de ordem 2 são as parcelas,
- as linhas de ordem 3 e 4 são os agrupamentos de totalização das parcelas.

Os títulos “Ordem”, “Dados” e “Valores” foram todos definidos na primeira consulta, sendo “Ordem” uma coluna “inventada” para a construção do relatório como desejamos. Ainda na primeira consulta deste *union* vemos a coluna “Valores” definida como vazia, somente para igualar ao número de colunas das próximas consultas neste *union*.

```
SELECT '1' AS Ordem, CONCAT('Aluno: ', a.id_aluno,' - ', a.nome) AS Dados, '' AS Valores
FROM alunos a JOIN matriculas m ON a.id_aluno=m.id_aluno
WHERE m.contrato=2
UNION
SELECT '2' AS Ordem, CONCAT('Parc: ', parcela,' Vcto: ', dt_vcto), valor
 FROM parcelas
 WHERE contrato=2
UNION
SELECT '3' AS Ordem, 'Total Pago',SUM(valor)
 FROM parcelas
 WHERE contrato=2 AND pago='S'
 GROUP BY pago
UNION
SELECT '4' AS Ordem, 'Total Devido',SUM(valor)
```

Ordem	Dados	Valores
1	Aluno: 1 - Carlos	
2	Parc: 01 Vcto: 2009-04-08	343.77
2	Parc: 02 Vcto: 2009-05-08	343.77
2	Parc: 03 Vcto: 2009-06-08	343.77
2	Parc: 04 Vcto: 2009-07-08	343.77
2	Parc: 05 Vcto: 2009-08-08	343.77
2	Parc: 06 Vcto: 2009-09-08	343.74
3	Total Pago	1718.85
4	Total Devido	343.74

```
FROM parcelas
WHERE contrato=2 AND pago='N'
GROUP BY pago
ORDER BY 1,2;
```

Exercícios de DQL - *UNION*

Resolva os exercícios de DQL abaixo utilizando UNION.

1. Selecione o nome do curso e o ano da edição em uma linha, listando abaixo as datas de aula conforme o exemplo abaixo, do curso 1 e ano da edição em 2008.

nome	ano
Administração	2008
	2008-02-25
	2008-02-26
	2008-02-27
	2008-02-28
	2008-02-29

2. Selecione o código do autor, o sobrenome e o nome concatenados em um único campo, a quantidade de empréstimos realizados para o autor e o valor ZERO num campo chamado ordem.
   Selecione o código do autor, o título do livro, a quantidade de empréstimos realizados para livro e o valor UM num campo chamado ordem.
   Una as duas consultas numa só apresentando um resultado semelhante ao abaixo.

ID	Autor	Emprestimos	Ordem
1	Autor: Moraes, Fernando	4	0
1	- Corações Sujos	4	1
2	Autor: Fonseca, Rubem	2	0
2	- Agosto	2	1
3	Autor: Kahneman, Daniel	1	0
3	- Rápido e Devagar - Duas Formas de Pensar	1	1
4	Autor: Tolstoi, Leon	1	0
4	- Guerra e paz	1	1
5	Autor: Orwells, George	4	0
5	- 1984	4	1

FUNÇÕES DA SQL (ANSI/MYSQL)

Nesta seção são apresentadas apenas algumas funções da API do MySQL. Para uma lista completa das funções verifique a documentação do site www.mysql.com.

LENGTH(string)

A função *length* retorna o tamanho de uma *string*, podendo ser um campo ou um dado informado.

```
SELECT LENGTH('abcdefg');
-> 7

SELECT nome
FROM alunos
WHERE LENGTH(nome)>10
```

CONCAT(str1, str2, str3,...)

A função *concat* concatena (junta) duas ou mais *strings* em uma só coluna.

```
SELECT CONCAT('My', 'S', 'QL');
```

```
-> MySQL
```

Obs.: Em muitos casos ocorrerá um problema de compatibilidade entre *collations* dos *caracter set* do banco de dados, especialmente se misturar campos de tabelas com *collation* diferentes ou dados fixos na SQL. Neste caso precisamos converter uma das partes da expressão para o mesmo *collation* da outra.

```
SELECT CONCAT(
     CONVERT('O aluno ' USING utf8),
     id_aluno,
     ' - ',
     CONVERT(nome USING utf8),
     CONVERT(' está devidamente matriculado na instituição.' USING utf8)
) AS dado_novo
FROM alunos;
```

dado_novo
O aluno 1 - Carlos está devidamente matriculado na instituição.
O aluno 2 - João está devidamente matriculado na instituição.
O aluno 3 - Maria está devidamente matriculado na instituição.
O aluno 4 - Gustavo está devidamente matriculado na instituição.
O aluno 5 - Rita está devidamente matriculado na instituição.
O aluno 6 - Tamara está devidamente matriculado na instituição.
O aluno 7 - Alberto está devidamente matriculado na instituição.

A função CONVERT converte o texto de entrada usando o *caracter set/collation* especificado.

CONCAT_WS(separator, str1, str2,...)

Similar à *concat*, a função *concat_ws* concatena duas ou mais *strings* adicionando um separador

```
SELECT CONCAT_WS(' - ', 'STRUCTURED', 'QUERY', 'LANGUAGE');
-> STRUCTURED - QUERY - LANGUAGE
```

CONV(N,from_base,to_base)

A função *conv* converte um valor de uma base numérica para outra

```
// converte o valor decimal 50 da base decimal para a base binária
SELECT CONV(50,10,2);
-> 110010
// converte o valor binário 1010 da base binária para a base decimal
SELECT CONV(110010,2,10);
-> 50
```

INSTR(str,substr);

A função *instr* verifica a ocorrência de uma *string* dentro de outra, retornando a posição da *substring* caso localizada ou ZERO caso não encontre.

```
SELECT INSTR('casa de carnes', 'car');
-> 9
SELECT INSTR('casa de carnes', 'boi');
-> 0
.. WHERE INSTR(campo_sql,'a')>0
```

LOWER(str)

A função *lower* converte uma *string* para caixa baixa.

```
SELECT LOWER('JavaScript');
-> javascript
```

UPPER(str)

A função *upper* converte uma *string* para caixa alta

```
SELECT UPPER('JavaScript');
-> JAVASCRIPT
```

REPEAT(str,count)

A função *repeat* repete uma *string* a quantidade de vezes definida no seu segundo parâmetro.

```
SELECT REPEAT('MySQL', 3);
-> MySQLMySQLMySQL
```

REPLACE(str,from_str,to_str)

Substitui uma *substring* por outra

```
SELECT REPLACE('x.mysql.com', 'x.', 'www.');
```

```
-> www.mysql.com
```

REVERSE(str)

A função *reverse* inverte o conteúdo de uma *string*

```
SELECT REVERSE('MySQL');
-> LQSyM
```

TRIM, RTRIM, LTRIM

As funções **trim* removem os espaços em branco de uma *string*, de acordo com a posição L (*left*), R (*right*) ou ambos os lados ao mesmo tempo.

```
SELECT TRIM('  MySQL   ');
-> 'MySQL'
SELECT RTRIM('  MySQL   ');
-> '  MySQL'
SELECT LTRIM('  MySQL   ');
-> 'MySQL   '
```

DATE_ADD, ADDDATE

A função *date_add* acrescenta períodos em datas.

```
SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
SELECT DATE_ADD('2007-01-31', INTERVAL 1 MONTH);
-> '2007-02-28'
SELECT DATE_ADD('2007-01-31', INTERVAL 2 MONTH);
-> '2007-03-31'
SELECT DATE_ADD('2007-01-31', INTERVAL 3 MONTH);
-> '2007-04-30'
```

DATE_SUB, SUBDATE

A função *date_sub* subtrai períodos de datas

```
SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
```

DATEDIFF

A função *datediff* demonstra a diferença de dias entre duas datas

```
SELECT DATEDIFF('1998-01-02', '1997-11-05');
```

```
-> 58
SELECT DATEDIFF('1997-11-05', '1998-01-02');
-> -58
```

ADDTIME()

A função *addtime* adiciona horas a uma hora

```
SELECT ADDTIME('2006-10-18 23:59:59', '0:15:11');
-> 2006-10-19 00:15:10
SELECT ADDTIME('14:00:00', '1:15:11');
-> 15:15:11
```

SUBTIME()

Subtrai horas a uma hora

```
SELECT SUBTIME('2006-10-18 23:59:59', '0:15:11');
-> 2006-10-19 23:44:48
SELECT SUBTIME('14:00:00', '1:15:11');
-> 12:44:49
```

TIMEDIFF

Diferença entre datas ou horas, contados em fração de tempo.

```
SELECT TIMEDIFF('2011-11-11 00:01:00', '2011-11-10 23:01:00');
-> 01:00:00
SELECT TIMEDIFF('00:01:00', '23:01:00');
-> -23:00:00
```

CURRENT_DATE(), CURDATE()

Obtém a data corrente.

```
SELECT CURRENT_DATE();
SELECT CURDATE();
-> 2006-10-19
```

CURRENT_TIME(), CURTIME()

Obtém a hora corrente.

```
SELECT CURRENT_TIME();
SELECT CURTIME();
-> 12:54:57
```

CURRENT_TIMESTAMP()

Obtém a data/hora corrente.

```
SELECT CURRENT_TIMESTAMP();
-> 2006-10-19 12:54:57
```

DATE_FORMAT(date, format)

Formata uma data ou hora.

```
SELECT DATE_FORMAT('2006-10-19', '%d/%m/%Y');
-> 19/10/2006
```

WEEKDAY(date)

Retorna o número do dia da semana (0 = Segunda, 1 = Terça, ... 6 = Domingo):

```
SELECT WEEKDAY('2006-10-18');
-> 2
SELECT WEEKDAY(CURDATE());
```

YEAR(date), MONTH(date), DAY(date)

Retorna o ano, o mês ou o dia de uma data

```
SELECT YEAR(CURRENT_DATE());
SELECT YEAR('2011-11-11');
-> 2011
SELECT MONTH('2011-11-11');
-> 11
SELECT DAY('2011-11-11');
-> 11
```

DAYNAME(date), MONTHNAME(date), DAY(date)

Retorna o nome do dia da semana ou do mês de uma data

```
SELECT DAYNAME('2011-11-11');
-> Friday
SELECT MONTHNAME('2011-11-11');
-> November
```

DAYOFWEEK(date), DAYOFMONTH(date), DAYOFYEAR(date)

Retorna o dia da semana, do mês ou do ano, dado uma dia

```
SELECT DAYOFWEEK('2011-11-11');
-> 6
SELECT DAYOFMONTH('2011-11-11');
-> 11
SELECT DAYOFYEAR('2011-11-11');
->315
```

EXTRACT(unit FROM date)

Extrai uma parte (*unit*) de uma data

```
SELECT EXTRACT(SECOND FROM '2011-11-11 23:59:59.999');
-> 59
SELECT EXTRACT(MICROSECOND FROM '2011-11-11 23:59:59.999');
-> 9990000
SELECT EXTRACT(MINUTE FROM '2011-11-11 23:59:59.999');
-> 59
SELECT EXTRACT(QUARTER FROM '2011-11-11 23:59:59.999');
-> 4
SELECT EXTRACT(QUARTER FROM '2011-01-11 00:14:59.999');
-> 1
SELECT EXTRACT(QUARTER FROM '2011-04-11');
-> 2
```

Obs.: Units (MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, SECOND_MICROSECOND, MINUTE_MICROSECOND, HOUR_MICROSECOND, HOUR_MINUTE, DAY_MICROSECOND, DAY_SECOND, DAY_MINUTE, DAY_HOUR, YEAR_MONTH)

LAST_DAY(date)

Retorna a última data do mês de uma data dada.

```
SELECT LAST_DAY('2011-11-11');
-> 2011-11-30
SELECT DAY(LAST_DAY('2011-11-11'));
-> 30
```

STR_TO_DATE(date_string, format)

Trasforma uma string contendo uma representação de data em uma data, dado um formato.

```
SELECT STR_TO_DATE('11/11/2011','%d/%m/%Y');
-> 2011-11-11
```

ENCODE(str,pass_str), DECODE(crypt_str,pass_str)

Codifica ou Descodifica uma string dada uma chave

```
update pessoas
set pes_senha=ENCODE('SuperPHP','chave1')
where pes_id=1;

select DECODE(pes_senha,'chave1')
from pessoas
where pes_id=1;
-> SuperPHP
```

LAST_INSERT_ID()

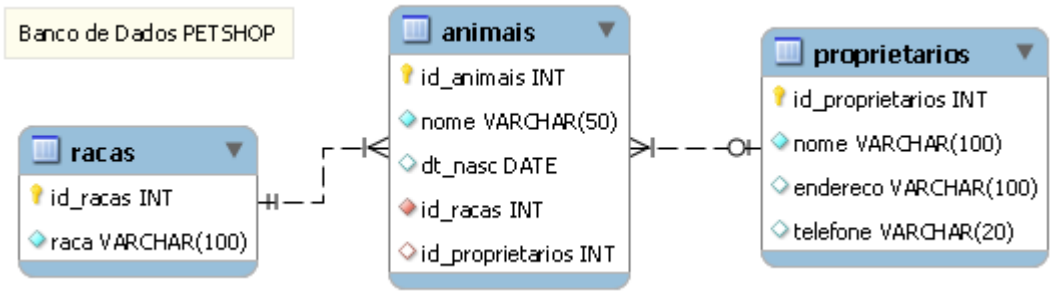
Retorna o último id de uma inserção em um campo autonumerado

```
SELECT LAST_INSERT_ID()
```

APÊNDICES

Exercícios Adicionais

Esta seção contém exercícios adicionais para a prática da SQL. Nesta seção você deverá criar um banco de dados completo, baseado no modelo abaixo, e realizar as interações propostas. As soluções estarão no final deste apêndice.



1. Crie uma base de dados chamada PETSHOP, baseada no ***character set*** utf8.
2. Crie as entidades **raças**, **proprietários** e **animais**, colocando a criação das chaves primárias e estrangeiras em instruções de ***alter table***.
3. Crie as seguintes contas de usuário com as permissões listadas abaixo:

Usuário	Maquina	Senha	Bancos	Permissões
carlos	qualquer	77@368	petshop	*Insert*, *update* e *delete* em animais
maria	10.63.75.8	uhameu	qualquer	*Select* em qualquer entidade

4. Altere a senha de cada uma das contas para @123456.
5. Popule com instruções de ***insert*** as entidades com os dados conforme demonstrados abaixo:

RACAS

id_racas	raca
1	Fox
2	Pug
3	Siamês
4	Himalaia

PROPRIETARIOS

id_proprietarios	nome	endereco	telefone
101	Ana	Rua A, 134	(NULL)
102	Carlos	(NULL)	32844433
103	Fabiana	(NULL)	(NULL)

ANIMAIS

id_animais	nome	dt_nasc	id_racas	id_proprietarios
11	Fifi	2010-08-13	3	101
12	Brucutu	2011-01-04	2	102
13	Joelma	2008-07-12	1	103
14	Max	2012-12-31	1	(NULL)
15	Jony	2012-07-31	4	(NULL)

6. Atualize as seguintes informações nas entidades:
   - A Fifi nasceu em 28/03/2011
   - O nome de Jony deve ser corrigido para Jhonny
7. Exclua o registro em animais com o nome de Joelma.

Exercícios Resolvidos

Esta seção aborda as soluções dos exercícios deste livro.

Exercícios de DDL

Esta seção aborda as soluções dos exercícios de *Data Definition Language*.

Exercícios de DDL da página 23

Para estes exercícios você precisará iniciar o servidor SGBD MySQL e o MySQL Workbench (Para saber como utilizar, veja o Apêndice - Usando o MySQL Workbench). Abra o Workbench em modo *SQL Development*. No *SQL Editor*, crie uma nova *SQL Tab* para executar seus comandos *SQL Script*.

8. Exclua a base de dados **ESCOLA**.

```
DROP DATABASE ESCOLA;
```

9. Crie a base de dados **ESCOLA** com o ***caracter set utf8***.

```
CREATE DATABASE escola DEFAULT CHARACTER SET  utf8;
```

10. Crie as entidades ***cursos***, ***edicoes***, ***alunos***, ***matriculas***, ***parcelas*** e ***datas_aula*** (nesta ordem).

```
CREATE TABLE cursos (
  id_curso tinyint(3) unsigned NOT NULL,
  nome varchar(100) NOT NULL,
  PRIMARY KEY (id_curso)
) ENGINE=InnoDB;

CREATE TABLE edicoes (
  id_curso tinyint(3) unsigned NOT NULL,
  ano year(4) NOT NULL,
  vagas tinyint(3) unsigned NOT NULL,
  dt_final_matr date NOT NULL,
  PRIMARY KEY (id_curso,ano)
) ENGINE=InnoDB;

CREATE TABLE alunos (
  id_aluno int(10) unsigned NOT NULL AUTO_INCREMENT,
  nome varchar(255) NOT NULL,
  dt_nasc date DEFAULT NULL,
  senha varchar(100) DEFAULT NULL,
  sexo enum('M','F') DEFAULT NULL,
  telefone varchar(20) DEFAULT NULL,
  cpf varchar(11) DEFAULT NULL,
  matricula varchar(20) DEFAULT NULL,
  PRIMARY KEY (id_aluno)
) ENGINE=InnoDB;

CREATE TABLE matriculas (
  contrato bigint(20) unsigned NOT NULL AUTO_INCREMENT
```

```
  id_aluno int(10) unsigned NOT NULL,
  id_curso tinyint(3) unsigned NOT NULL,
  ano year(4) NOT NULL,
  dt_matricula date NOT NULL,
  vlr_total decimal(20,2) DEFAULT NULL,
  PRIMARY KEY (contrato)
) ENGINE=InnoDB;

CREATE TABLE parcelas (
  contrato bigint(20) unsigned NOT NULL,
  parcela tinyint(2) unsigned zerofill NOT NULL,
  dt_vcto date NOT NULL,
  valor decimal(20,2) NOT NULL,
  dt_pgto date DEFAULT NULL,
  vlr_acrescimos decimal(20,2) DEFAULT NULL,
  vlr_descontos decimal(20,2) DEFAULT NULL,
  vlr_pago decimal(20,2) DEFAULT NULL,
  pago char(1) NOT NULL DEFAULT 'N',
  obs text,
  id_usuario tinyint(3) unsigned DEFAULT NULL,
  PRIMARY KEY (contrato,parcela)
) ENGINE=InnoDB;

CREATE TABLE datas_aula (
  id_curso tinyint(3) unsigned NOT NULL,
  ano year(4) NOT NULL,
  dt_aula date NOT NULL,
  PRIMARY KEY (id_curso,dt_aula)
) ENGINE=InnoDB;
```

11. Crie a FK entre ***edicoes*** e ***cursos***, alterando a endidade ***edicoes***.

```
ALTER TABLE edicoes
ADD CONSTRAINT edicoes_cursos_fk
  FOREIGN KEY (id_curso)
  REFERENCES cursos (id_curso);
```

12. Crie a FK entre ***datas_aula*** e ***edicoes***, alterando a endidade ***datas_aula***.

```
ALTER TABLE datas_aula
ADD CONSTRAINT datas_aula_edicoes_fk
  FOREIGN KEY (id_curso, ano)
  REFERENCES edicoes (id_curso, ano);
```

13. Crie a FK entre ***edicoes*** e ***matriculas***, alterando a endidade ***matriculas***.

```
ALTER TABLE matriculas
ADD CONSTRAINT matriculas_edicoes_fk
  FOREIGN KEY (id_curso, ano)
```

```
  REFERENCES edicoes (id_curso, ano);
```

14. Crie a FK entre ***alunos*** e ***matriculas***, alterando a endidade ***matriculas***.

```
ALTER TABLE matriculas
ADD CONSTRAINT matriculas_alunos_fk
  FOREIGN KEY (id_aluno)
  REFERENCES alunos (id_aluno);
```

15. Crie a FK entre ***matriculas*** e ***parcelas***, alterando a endidade ***parcelas***.

```
ALTER TABLE parcelas
ADD CONSTRAINT parcelas_ibfk_1
  FOREIGN KEY (contrato)
  REFERENCES matriculas (contrato);
```

16. Altere a entidade ***matriculas*** adicionando o campo "***historico***" do tipo *mediumtext*.

```
ALTER TABLE parcelas
ADD historico MEDIUMTEXT;
```

17. Altere a entidade ***matriculas*** adicionando o campo "***id_usuario***" do tipo *tinyint(3) unsigned*.

```
ALTER TABLE matriculas
ADD id_usuario tinyint(3) unsigned;
```

18. Crie a FK entre ***matriculas*** e ***usuarios***, alterando a endidade ***matriculas***.

```
ALTER TABLE matriculas
ADD CONSTRAINT matriculas_id_usuario_FK
  FOREIGN KEY (id_usuario)
  REFERENCES usuarios (id_usuario);
```

19. Crie o índice do tipo coluna, com o nome ***parcelas_dt_vcto_idx*** para a coluna ***dt_vcto*** da entidade ***parcelas***.

```
CREATE INDEX parcelas_dt_vcto_idx
ON parcelas (dt_vcto);
```

20. Crie o índice do tipo coluna, com o nome ***parcelas_dt_pgto_idx*** para a coluna ***dt_pgto*** da entidade ***parcelas***.

```
CREATE INDEX parcelas_dt_pgto_idx
ON parcelas (dt_pgto);
```

21. Crie uma view chamada ***v_parcelas_impagas*** com a segunte cláusula SQL

```
CREATE OR REPLACE VIEW v_parcelas_impagas
AS
select a.id_aluno, a.nome, p.contrato, p.parcela, p.valor, p.dt_vcto
from alunos a, matriculas m, parcelas p
where a.id_aluno=m.id_aluno
  and m.contrato=p.contrato
  and p.pago<>'S';
```

22. Verifique os dados da view executando a consulta abaixo:

```
select * from v_parcelas_impagas;
```

Exercícios de DCL

Esta seção aborda as soluções dos exercícios de *Data Control Language*.

Exercícios de DCL da página 30

1. Crie a conta de usuário **administrador** com permissão de acesso somente da máquina local, com a senha **super**.

```
create user 'administrador'@'localhost'
IDENTIFIED BY 'super';
```

2. Crie a conta de usuário **convidado** com permissão de acesso de qualquer máquina e para a máquina local, com a senha **guest**.

```
create user 'convidado'@'%'
IDENTIFIED BY 'guest';

create user 'convidado'@'localhost'
IDENTIFIED BY 'guest';
```

3. Crie a conta de usuário **funcionario** com permissão de acesso na rede **'10.1.1.%'**, com a senha **personal**.

```
create user 'funcionario'@'10.1.1.%'
IDENTIFIED BY 'personal';
```

4. Conceda a permissão de **SELECT, INSERT, UPDATE e DELETE** sobre todo os bancos de dados para a conta do usuário **administrador**.

```
grant SELECT, INSERT, UPDATE, DELETE
on *.*
to 'administrador'@'localhost';
```

5. Conceda a permissão de **SELECT, INSERT** sobre a entidade **livros** do banco de dados **escola** para a conta do usuário **convidado** do host local.

```
grant SELECT, INSERT
on escola.livros
to 'convidado'@'localhost%';
```

6. Conceda a permissão de **SELECT, INSERT, UPDATE e DELETE** sobre todo o banco de dados **escola** para a conta do usuário **funcionario**.

```
grant SELECT, INSERT, UPDATE, DELETE
on escola.*
to 'funcionario'@'10.1.1.%';
```

7. Remova a permissão de **INSERT** concedida para a conta do usuário **convidado**.

```
REVOKE INSERT on escola.livros
FROM 'convidado'@'%';
```

8. Conecte no banco de dados **escola** com o usuário **convidado** e, utilizando o *table edit*, tente inserir um registro na tabela livros. Qual o resultado que o banco de dados retorna?

```
Retorna o erro:
ERROR 1142: INSERT command denied to user 'convidado'@'localhost' for table
'livros'
```

Exercícios de DML

Esta seção aborda as soluções dos exercícios de *Data Manipulation Language*.

Exercícios de DML da página 37

Responda as questões abaixo usando instruções SQL da DML.

1. Insira na entidade **cursos** um novo curso chamado **Culinária**, com o código **6**.

```
INSERT INTO cursos
   (id_curso,nome)
VALUES
   (6,'Culinária');
```

2. Insira na entidade **edicoes** uma nova edição para o curso acima, no ano de **2013**, máximo de vagas em **10** e a data limite de matrícula em **10 de Abril.**

```
INSERT INTO edicoes
   (id_curso, ano, vagas, dt_final_matr)
VALUES
   (6,2013,10,'2013-04-10');
```

3. Insira na entidade **alunos** um novo aluno, colocando seu **nome completo**, sua **data de nascimento**, seu **sexo**, sem informar o código do aluno (*auto_increment*).

```
 INSERT INTO alunos
   (nome, dt_nasc, sexo)
 VALUES
   ('Carlos Eduardo', '1982-01-24', 'M');
```

4. Efetue uma consulta para identificar qual código o último *insert* gerou.

```
SELECT LAST_INSERT_ID();
# 1999812
```

5. Insira na entidade **matriculas** uma nova matrícula, para o aluno inserido na questão 3, na edição inserida na questão 2, com a data de matrícula na data atual e o valor de R$ 1.600,00.

```
INSERT INTO matriculas
  (id_aluno, id_curso, ano, dt_matricula, vlr_total)
VALUES
  (1999812,        6, 2013, '2013-03-10', 1600);
```

6. Atualize o número de vagas para 15 e a data limite de matrícula para 30/04, da edição inserida na questão 2.

```
UPDATE edicoes
SET vagas = 15, dt_final_matr = '2013-04-30'
WHERE id_curso = 6 AND ano = 2013;
```

7. Atualize o valor da matrícula para R$ 1.599,96.

```
UPDATE matriculas
SET vlr_total = 1599.96
WHERE
    id_curso = 6
AND id_aluno = 1999812
AND ano = 2013;
```

8. Exclua a matrícula inserida na questão 5.

```
DELETE FROM matriculas
WHERE id_curso = 6
  AND id_aluno = 1999812
  AND ano = 2013;
```

9. Exclua o aluno inserido na questão 3.

```
DELETE FROM alunos
WHERE id_aluno = 1999812;
```

10. Exclua a edição inserida na questão 2.

```
DELETE FROM edicoes
WHERE id_curso = 6
  AND ano = 2013;
```

11. Exclua o curso inserido na questão 1.

```
DELETE FROM cursos
WHERE id_curso = 6;
```

Exercícios de DQL

Exercícios de DQL - Agrupamentos da página 41

Exercícios de DQL em uma entidade:

1. Obtenha os nomes de todos os alunos.

```
SELECT nome
FROM alunos;
```

2. Obtenha o código e o nome de todos os alunos.

```
SELECT id_aluno, nome
FROM alunos;
```

3. Obtenha o nome de todas os alunos em ordem alfabética.

```
SELECT nome
FROM alunos
ORDER BY nome;
```

4. Obtenha o nome do aluno código 15.

```
SELECT nome
FROM alunos
WHERE id_aluno=15;
```

5. Obtenha o nome e a data de nascimento de todas os alunos com código maior que 2.

```
SELECT nome, dt_nasc
FROM alunos
WHERE id_aluno>2;
```

6. Obtenha o nome, o código e a matricula de todos os alunos que possuem numero de matricula.

```
SELECT nome, id_aluno, matricula
FROM alunos
WHERE matricula IS NOT NULL;
```

7. Selecione o código da disciplina, o código do curso e a carga horária de todas as disciplinas do curriculo do curso 1.

```
SELECT id_disciplina, id_curso, carga_horaria
FROM curriculos
WHERE id_curso=1;
```

8. Selecione o código dos alunos, o código do curso e a data de matrícula de todas as matrículas efetuadas entre entre o dia 1 e 30 de março de 2010.

```
SELECT id_aluno, id_curso, dt_matricula
FROM matriculas
WHERE dt_matricula BETWEEN '2010-03-01' AND '2010-03-30';
```

9. Selecione o id e a quantidade de vagas de todas as edições de curso com 45 vagas.

```
SELECT id_curso, vagas
FROM edicoes
WHERE vagas=45;
```

10. Selecione o id e a quantidade de vagas de todas as edições de curso com mais de 45 vagas.

```
SELECT id_curso, vagas
FROM edicoes
WHERE vagas>45;
```

11. Selecione o id e a quantidade de vagas de todas as edições de curso com menos de 45 vagas.

```
SELECT id_curso, vagas
FROM edicoes
```

```
WHERE vagas<45;
```

12. Selecione o id e a quantidade de vagas de todas as edições com numero de vagas entre 20 e 30 vagas.

```
SELECT e.id_curso, e.vagas
FROM edicoes e
WHERE e.vagas BETWEEN 20 AND 30;

-- OU

SELECT e.id_curso, e.vagas
FROM edicoes e
WHERE e.vagas >= 20 AND e.vagas <= 30;
```

13. Selecione o id e a quantidade de vagas de todas as edições em que o número de vagas seja diferente de 45.

```
SELECT c.id_curso, c.vagas
FROM edicoes c
WHERE c.vagas<>45;
```

Exercícios de DQL em uma entidade com agrupamentos:

14. Obtenha o maior código da tabela alunos

```
SELECT MAX(a.id_aluno) AS maior_id
FROM alunos a;
```

15. Obtenha o próximo código (maior + 1) de alunos.

```
SELECT MAX(id_aluno)+1 AS maior_id
FROM alunos;
```

16. Selecione a soma total das cargas horarias das disciplinas do currículo do curso 2;

```
SELECT AVG(valor)
FROM curriculos
WHERE id_curso=3;
```

17. Selecione a média de preço das disciplinas do currículo do curso 3.

```
SELECT AVG(valor)
FROM curriculos
WHERE id_curso=3;
```

18. Selecione o id do curso, a soma das cargas horárias e a soma dos valores das disciplinas, agrupadas pelo id do curso, de todos os currículos cadastrados.

```
SELECT id_curso,
  SUM(carga_horaria) AS total_horas,
  SUM(valor) AS preco_total
FROM curriculos
GROUP BY id_curso;
```

19. Selecione a quantidade de alunos matriculados no curso 3 (da tabela matrículas).

```
SELECT COUNT(id_aluno)
FROM matriculas
WHERE id_curso=3;
```

20. Calcule a soma das cargas horárias dos currículos, agrupado pelo curso e semestre

```
SELECT id_curso, semestre, SUM(carga_horaria) AS carga_horaria_total
FROM escola.curriculos
GROUP BY id_curso, semestre;
```

21. Busque a quantidade de alunos do sexo masculino e do sexo feminino, agrupando pela coluna sexo.

```
SELECT sexo, COUNT(id_aluno)
FROM escola.alunos
GROUP BY sexo;
```

22. Selecione o código do curso, o ano, o valor total matriculado e o valor médio matriculado, de todas as matrículas desde que tenham mais de 3 alunos matriculados no curso/ano.

```
SELECT id_curso, ano,
     SUM(vlr_total) AS Valor_Total,
     AVG(vlr_total) AS Valor_Medio
FROM escola.matriculas
GROUP BY id_curso, ano
HAVING COUNT(id_aluno)>3;
```

Exercícios de DQL – Sintaxe JOIN da página 43

Resolva os exercícios de DQL abaixo utilizando a sintaxe tradicional e a sintaxe JOIN:

1. Obtenha o nome e data de nascimento do aluno, o código do curso, o ano, a data de matrícula e o valor total de todas as matrículas efetuadas pelos alunos de código inferior a 150.

```
SELECT a.nome, a.dt_nasc, m.id_curso, m.ano, m.dt_matricula, m.vlr_total
FROM alunos a, matriculas m
WHERE a.id_aluno = m.id_aluno
  AND a.id_aluno < 150;

-- Usando JOIN
SELECT a.nome, a.dt_nasc, m.id_curso, m.ano, m.dt_matricula, m.vlr_total
FROM alunos a
  JOIN matriculas m ON a.id_aluno = m.id_aluno
WHERE a.id_aluno < 150;
```

2. Selecione o nome do curso, o ano e o número de vagas nas edições, de todos os cursos, retornando o resultado ordenado pelo ano da edição e após pelo nome dos cursos.

```
SELECT c.nome, e.ano, e.vagas
```

```
FROM cursos c, edicoes e
WHERE c.id_curso = e.id_curso
ORDER BY e.ano, c.nome;

-- Usando JOIN
SELECT c.nome, e.ano, e.vagas
FROM cursos c
  JOIN edicoes e ON c.id_curso = e.id_curso
ORDER BY e.ano, c.nome;
```

3. Liste o nome do autor e o nome dos livros que este escreveu.

```
SELECT a.nome, l.titulo
FROM autores a, livrosxautores la, livros l
WHERE a.id_autor = la.id_autor
  AND la.id_livro = l.id_livro;

-- Usando JOIN
SELECT a.nome, l.titulo
FROM autores a
  JOIN livrosxautores la, livros l
WHERE a.id_autor = la.id_autor
  AND la.id_livro = l.id_livro;
```

4. Exiba o nome do curso, o nome das disciplinas deste curso com as respectivas cargas horárias do currículo, ordenando pelo nome do curso e após pelo semestre da disciplina.

```
SELECT c.nome, d.disciplina, u.carga_horaria, u.semestre
FROM cursos c, curriculos u, disciplinas d
WHERE c.id_curso = u.id_curso
  AND u.id_disciplina = d.id_disciplina
ORDER BY c.nome, u.semestre;

-- Usando JOIN
SELECT c.nome, d.disciplina, u.carga_horaria, u.semestre
FROM cursos c
  JOIN curriculos u ON c.id_curso = u.id_curso
  JOIN disciplinas d ON u.id_disciplina = d.id_disciplina
ORDER BY c.nome, u.semestre;
```

5. Liste o nome, o id e o nome do curso de todas as pessoas e cursos, devidamente relacionados, cujo nome da pessoa termine com "A" e esteja matriculada no curso 1.

```
SELECT a.nome, a.id_aluno, c.nome, c.id_curso
FROM alunos a, matriculas m, edicoes e, cursos c
WHERE c.id_curso = e.id_curso
  AND e.id_curso = m.id_curso AND e.ano = m.ano
  AND m.id_aluno = a.id_aluno
  AND a.nome LIKE '%a'
  AND c.id_curso = 1;
```

```
-- Usando JOIN
SELECT a.nome, a.id_aluno, c.nome, c.id_curso
FROM cursos c
  JOIN edicoes e ON c.id_curso = e.id_curso
  JOIN matriculas m ON e.id_curso = m.id_curso AND e.ano = m.ano
  JOIN alunos a ON m.id_aluno = a.id_aluno
WHERE a.nome LIKE '%a'
AND c.id_curso = 1;
```

6. Liste a quantidade de pessoas matriculadas no curso 2 e que o nome contenha a letra A ('%A%').

```
SELECT COUNT(DISTINCT a.id_aluno), m.id_curso
FROM alunos a, matriculas m
WHERE a.id_aluno = m.id_aluno
  AND a.nome LIKE '%a%'
  AND m.id_curso = 2;

-- Usando JOIN
SELECT COUNT(DISTINCT a.id_aluno), m.id_curso
FROM alunos a
  JOIN matriculas m ON a.id_aluno = m.id_aluno
WHERE a.nome LIKE '%a%'
  AND m.id_curso = 2;
```

7. Selecione o código da disciplina, o código do curso, o nome da disciplina e a carga horária de todas as disciplinas do curso 1;

```
SELECT d.id_disciplina, u.id_curso, d.disciplina, u.carga_horaria
FROM disciplinas d, curriculos u
WHERE d.id_disciplina = u.id_disciplina
AND u.id_curso = 1;

-- Usando JOIN
SELECT d.id_disciplina, u.id_curso, d.disciplina, u.carga_horaria
FROM disciplinas d
  JOIN curriculos u ON d.id_disciplina = u.id_disciplina
WHERE u.id_curso = 1;
```

8. Selecione o id e o nome do curso, a soma das cargas horárias e a soma dos valores das disciplinas, agrupadas pelo id do curso.

```
SELECT c.id_curso, c.nome, SUM(u.carga_horaria)
FROM cursos c, curriculos u
WHERE c.id_curso=u.id_curso
GROUP BY c.id_curso;

-- Usando JOIN
SELECT c.id_curso, c.nome, SUM(u.carga_horaria)
FROM cursos c
```

```
  JOIN curriculos u ON c.id_curso=u.id_curso
GROUP BY c.id_curso;
```

9. Selecione o código e nome das pessoas e o código do curso de todas as matrículas efetuadas entre entre o dia 1 e 30 de março de 2008.

```
SELECT m.id_aluno, a.nome, m.id_curso
FROM matriculas m, alunos a
WHERE m.id_aluno=a.id_aluno
  AND m.dt_matricula BETWEEN '2008-03-01' AND '2008-03-30';

-- Usando JOIN
SELECT m.id_aluno, a.nome, m.id_curso
FROM matriculas m
  JOIN alunos a ON m.id_aluno=a.id_aluno
WHERE m.dt_matricula BETWEEN '2008-03-01' AND '2008-03-30';
```

10. Selecione o id, o nome e a quantidade de vagas de todos os cursos com 45 vagas.

```
SELECT c.id_curso, c.nome, e.vagas
FROM cursos c, edicoes e
WHERE c.id_curso=e.id_curso
  AND e.vagas=45;

-- Usando JOIN
SELECT c.id_curso, c.nome, e.vagas
FROM cursos c JOIN edicoes e ON c.id_curso=e.id_curso
WHERE e.vagas=45;
```

11. Selecione o id, o nome e a quantidade de vagas de todos os cursos com 45 vagas e a quantidade de matriculas para cada ano.

```
SELECT c.id_curso, c.nome, e.vagas, COUNT(m.contrato), e.ano
FROM cursos c, edicoes e, matriculas m
WHERE c.id_curso=e.id_curso
  AND m.id_curso=e.id_curso AND m.ano=e.ano
  AND e.vagas=45
GROUP BY c.id_curso, c.nome, e.vagas, e.ano;

-- Usando JOIN
SELECT c.id_curso, c.nome, e.vagas, COUNT(m.contrato), e.ano
FROM cursos c
  JOIN edicoes e ON c.id_curso=e.id_curso
  JOIN matriculas m ON m.id_curso=e.id_curso AND m.ano=e.ano
WHERE e.vagas=45
GROUP BY c.id_curso, c.nome, e.vagas, e.ano;
```

12. Selecione o id, o nome, o ano da edição e a quantidade de vagas de todos os cursos com menos de 45 vagas cujo o número de vagas seja superior à quantidade de matrículas.

```
SELECT c.id_curso, c.nome, e.ano, e.vagas, COUNT(m.contrato)
```

```
FROM cursos c, edicoes e, matriculas m
WHERE c.id_curso=e.id_curso
  AND m.id_curso=e.id_curso AND m.ano=e.ano
  AND e.vagas<45
GROUP BY c.id_curso, c.nome, e.ano, e.vagas
HAVING e.vagas>COUNT(m.contrato);

-- Usando JOIN
SELECT c.id_curso, c.nome, e.ano, e.vagas, COUNT(m.contrato)
FROM cursos c
  JOIN edicoes e ON c.id_curso=e.id_curso
  JOIN matriculas m ON m.id_curso=e.id_curso AND m.ano=e.ano
WHERE e.vagas<45
GROUP BY c.id_curso, c.nome, e.vagas, e.ano
HAVING e.vagas>COUNT(m.contrato);
```

13. Selecione o id, o nome e a quantidade de vagas de todos os cursos com numero de vagas entre 41 e 49 vagas.

```
SELECT c.id_curso, c.nome, e.vagas
FROM cursos c, edicoes e
WHERE c.id_curso=e.id_curso
  AND e.vagas BETWEEN 41 AND 49;

-- Usando JOIN
SELECT c.id_curso, c.nome, e.vagas
FROM cursos c JOIN edicoes e ON c.id_curso=e.id_curso
WHERE e.vagas BETWEEN 41 AND 49;
```

14. Selecione o id, o nome e a quantidade de vagas de todos os cursos em que o número de vagas seja diferente de 45.

```
SELECT c.id_curso, c.nome, e.vagas
FROM cursos c, edicoes e
WHERE c.id_curso=e.id_curso
  AND e.vagas <> 45;

-- Usando JOIN
SELECT c.id_curso, c.nome, e.vagas
FROM cursos c JOIN edicoes e ON c.id_curso=e.id_curso
WHERE e.vagas <> 45;
```

15. Selecione o nome do curso e a média de preço das disciplinas do curso 3.

```
SELECT c.nome, AVG(u.valor)
FROM cursos c, curriculos u
WHERE c.id_curso=u.id_curso
  AND c.id_curso=3
GROUP BY c.nome;
```

```
-- Usando JOIN
SELECT c.nome, AVG(u.valor)
FROM cursos c
  JOIN curriculos u ON c.id_curso=u.id_curso
WHERE c.id_curso=3
GROUP BY c.nome;
```

16. Selecione o nome do curso e a soma total das cargas horarias das disciplinas do curso 2.

```
SELECT c.nome, SUM(u.carga_horaria)
FROM curriculos u
  JOIN cursos c ON c.id_curso=u.id_curso
WHERE c.id_curso=2
GROUP BY c.nome;
```

Exercícios de DQL - UNION da página 51

1. Selecione o nome do curso e o ano da edição em uma linha, listando abaixo as datas de aula conforme o exemplo abaixo, do curso 1 e ano da edição em2008.

nome	ano
Administração	2008
	2008-02-25
	2008-02-26
	2008-02-27
	2008-02-28
	2008-02-29

```
SELECT c.nome, e.ano
FROM cursos c
  JOIN edicoes e ON c.id_curso=e.id_curso
WHERE c.id_curso =1
  AND e.ano=2008
UNION
SELECT '', d.dt_aula
FROM datas_aula d
WHERE d.id_curso=1 AND d.ano=2008
ORDER BY 1 DESC, 2;
```

2. Selecione o código do autor, o sobrenome e o nome concatenados em um único campo, a quantidade de empréstimos realizados para o autor e o valor ZERO num campo chamado ordem.
   Selecione o código do autor, o título do livro, a quantidade de empréstimos realizados para livro e o valor UM num campo chamado ordem.
   Una as duas consultas numa só apresentando um resultado semelhante ao abaixo.

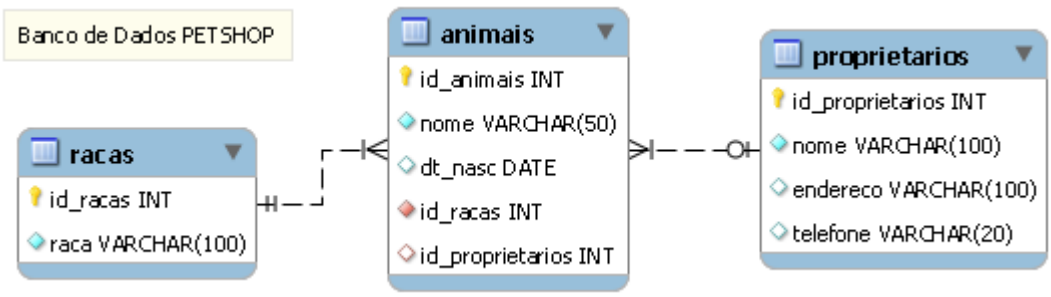
ID	Autor	Emprestimos	Ordem
1	Autor: Moraes, Fernando	4	0
1	- Corações Sujos	4	1
2	Autor: Fonseca, Rubem	2	0
2	- Agosto	2	1
3	Autor: Kahneman, Daniel	1	0
3	- Rápido e Devagar - Duas Formas de Pensar	1	1
4	Autor: Tolstoi, Leon	1	0
4	- Guerra e paz	1	1
5	Autor: Orwells, George	4	0
5	- 1984	4	1

```
SELECT a.id_autor AS ID,
       CONCAT('Autor: ',a.sobrenome,', ', a.nome) AS Autor,
       COUNT(e.id_emprestimo) AS Emprestimos,
       0 AS Ordem
FROM autoreas a
  JOIN livrosxautores la ON la.id_autor=a.id_autor
  JOIN livros l ON la.id_livro=l.id_livro
  JOIN emprestimos e ON l.id_livro=e.id_livro
GROUP BY 1,2,4
UNION
SELECT la.id_autor,
       CONCAT('- ', l.titulo),
       COUNT(e.id_emprestimo),
       1 AS ordem
FROM livrosxautores la
  JOIN livros l ON la.id_livro=l.id_livro
  JOIN emprestimos e ON l.id_livro=e.id_livro
GROUP BY 1,2,4
ORDER BY 1,ordem;
```

Apêndice - Exercícios Adicionais

Exercícios Adicionais da página 60

Exercícios de DQL em uma entidade:



1. Crie uma base de dados chamada PETSHOP, baseada no ***character set*** utf8.

```
CREATE SCHEMA IF NOT EXISTS PETSHOP DEFAULT CHARACTER SET utf8;
```

2. Crie as entidades **raças**, **proprietários** e **animais**, colocando a criação das chaves primárias e estrangeiras em instruções de ***alter table***.

```
CREATE  TABLE IF NOT EXISTS PETSHOP.racas (
  id_racas INT NOT NULL,
  raca VARCHAR(100) NOT NULL)
ENGINE = INNODB;

ALTER TABLE PETSHOP.racas
ADD CONSTRAINT racas_pk PRIMARY KEY (id_racas);

CREATE  TABLE IF NOT EXISTS PETSHOP.proprietarios (
  id_proprietarios INT NOT NULL,
  nome VARCHAR(100) NOT NULL,
  endereco VARCHAR(100),
  telefone VARCHAR(20))
ENGINE = INNODB;

ALTER TABLE PETSHOP.proprietarios
ADD CONSTRAINT proprietarios_pk PRIMARY KEY (id_proprietarios);

CREATE  TABLE IF NOT EXISTS PETSHOP.animais (
  id_animais INT NOT NULL,
  nome VARCHAR(50) NOT NULL,
  dt_nasc DATE,
  id_racas INT NOT NULL,
  id_proprietarios INT)
ENGINE = INNODB;

ALTER TABLE PETSHOP.animais
ADD CONSTRAINT animais_pk PRIMARY KEY (id_animais);

ALTER TABLE PETSHOP.animais
ADD CONSTRAINT animais_racas_fk
    FOREIGN KEY (id_racas)
    REFERENCES PETSHOP.racas (id_racas);

ALTER TABLE PETSHOP.animais
ADD CONSTRAINT animais_proprietarios_fk
    FOREIGN KEY (id_proprietarios)
    REFERENCES PETSHOP.proprietarios (id_proprietarios);
```

3. Crie as seguintes contas de usuário com as permissões listadas abaixo:

Usuário	Maquina	Senha	Bancos	Permissões
carlos	qualquer	77@368	petshop	*Insert*, *update* e *delete* em animais

maria	10.63.75.8	uhameu	qualquer	*Select* em qualquer entidade

```
CREATE USER 'carlos'@'%' IDENTIFIED BY '77@368';
CREATE USER 'maria'@'10.63.75.8' IDENTIFIED BY 'uhameu';
GRANT INSERT,UPDATE,DELETE ON petshop.animais TO 'carlos'@'%';
GRANT SELECT ON *.* TO 'maria'@'10.63.75.8';
```

4. Altere a senha de cada uma das contas para @123456.

```
SET PASSWORD FOR 'carlos'@'%' = PASSWORD('@123456');
SET PASSWORD FOR 'maria'@'10.63.75.8' = PASSWORD('@123456');
```

5. Popule com instruções de ***insert*** as entidades com os dados conforme demonstrados abaixo:

RACAS

id_racas	raca
1	Fox
2	Pug
3	Siamês
4	Himalaia

PROPRIETARIOS

id_proprietarios	nome	endereco	telefone
101	Ana	Rua A, 134	(NULL)
102	Carlos	(NULL)	32844433
103	Fabiana	(NULL)	(NULL)

ANIMAIS

id_animais	nome	dt_nasc	id_racas	id_proprietarios
11	Fifi	2010-08-13	3	101
12	Brucutu	2011-01-04	2	102
13	Joelma	2008-07-12	1	103
14	Max	2012-12-31	1	(NULL)
15	Jony	2012-07-31	4	(NULL)

```
USE petshop;

INSERT INTO racas (id_racas, raca) VALUES ('1', 'Fox');
INSERT INTO racas (id_racas, raca) VALUES ('2', 'Pug');
INSERT INTO racas (id_racas, raca) VALUES ('3', 'Siamês');
INSERT INTO racas (id_racas, raca) VALUES ('4', 'Himalaia');

INSERT INTO proprietarios (id_proprietarios, nome, endereco)
  VALUES ('101', 'Ana', 'Rua A, 134');
INSERT INTO proprietarios (id_proprietarios, nome, telefone)
  VALUES ('102', 'Carlos', '32844433');
INSERT INTO proprietarios (id_proprietarios, nome, endereco, telefone)
  VALUES ('103', 'Fabiana', NULL, NULL);

INSERT INTO animais (id_animais, nome, dt_nasc, id_racas, id_proprietarios)
  VALUES ('11', 'Fifi', '2010-08-13', '3', '101');
INSERT INTO animais (id_animais, nome, dt_nasc, id_racas, id_proprietarios)
  VALUES ('12', 'Brucutu', '2011-01-04', '2', '102');
INSERT INTO animais (id_animais, nome, dt_nasc, id_racas, id_proprietarios)
  VALUES ('13', 'Joelma', '2008-07-12', '1', '103');
```

```
INSERT INTO animais (id_animais, nome, dt_nasc, id_racas)
  VALUES ('14', 'Max', '2012-12-31', '1');
INSERT INTO animais (id_animais, nome, dt_nasc, id_racas)
  VALUES ('15', 'Jony', '2012-07-31', '4');
```

6. Atualize as seguintes informações nas entidades:

   - A Fifi nasceu em 28/03/2011

   - O nome de Jony deve ser corrigido para Jhonny

```
UPDATE ANIMAIS SET dt_nasc='2011-03-28' WHERE id_animais=11;
UPDATE ANIMAIS SET nome='Jhonny' WHERE id_animais=15;
```

7. Exclua o registro em animais com o nome de Joelma.

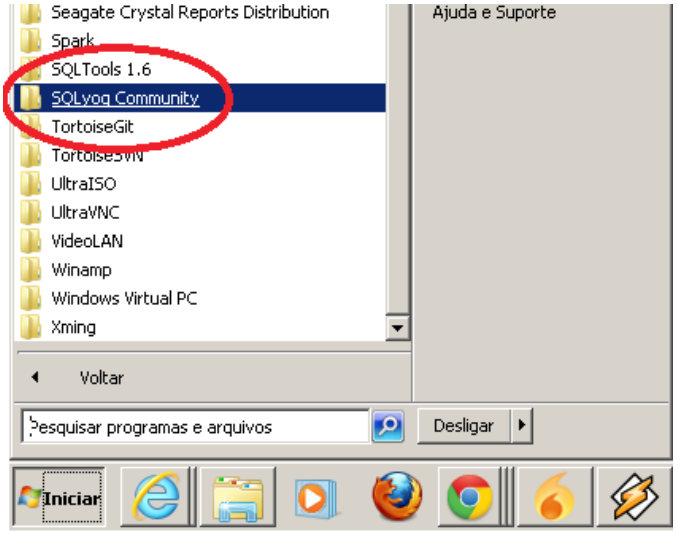
```
DELETE FROM ANIMAIS WHERE id_animais=13;
```

Usando o SQLYog Community

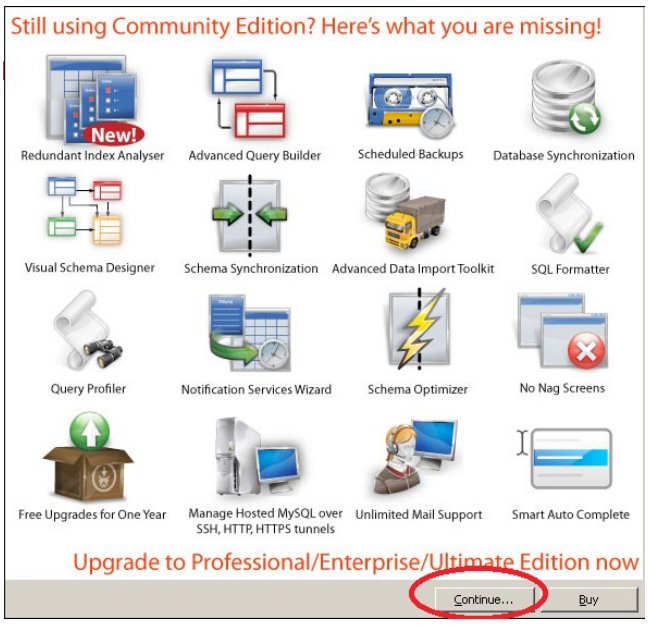
O SQLYog é uma ferramenta de acesso e trabalho no mysql que basicamente funciona para manipulação de SQL. Porém é muito útil para efetuar backup e restore de suas bases de dados. Possui uma versão community, gratuita e *open source* que pode ser obtida em http://code.google.com/p/sqlyog/downloads/list, e versões comerciais.

Efetuando *Restore* de um banco de dados

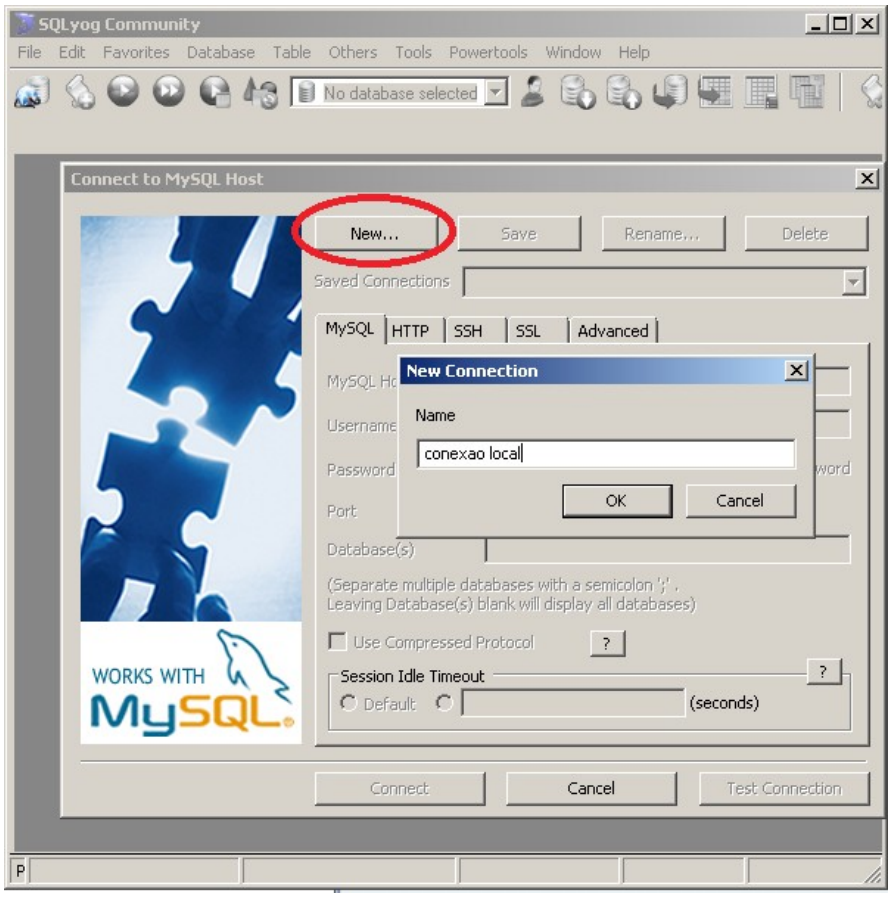
Para efetuar o restore de um database, inicie o SQLYog.



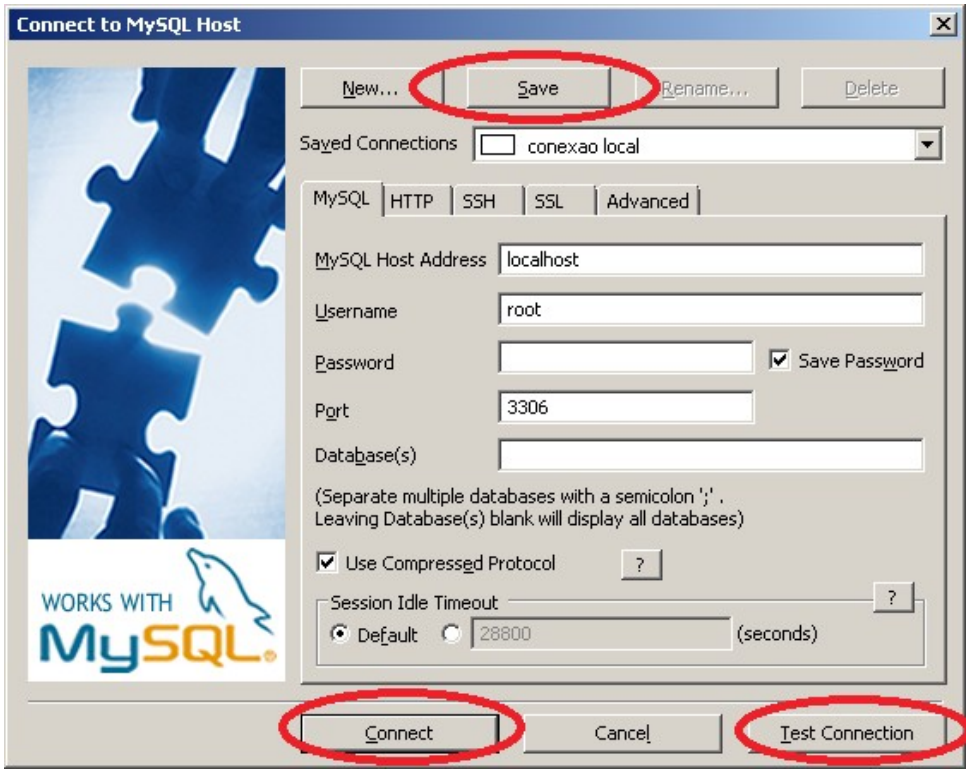
Clique em “continue”



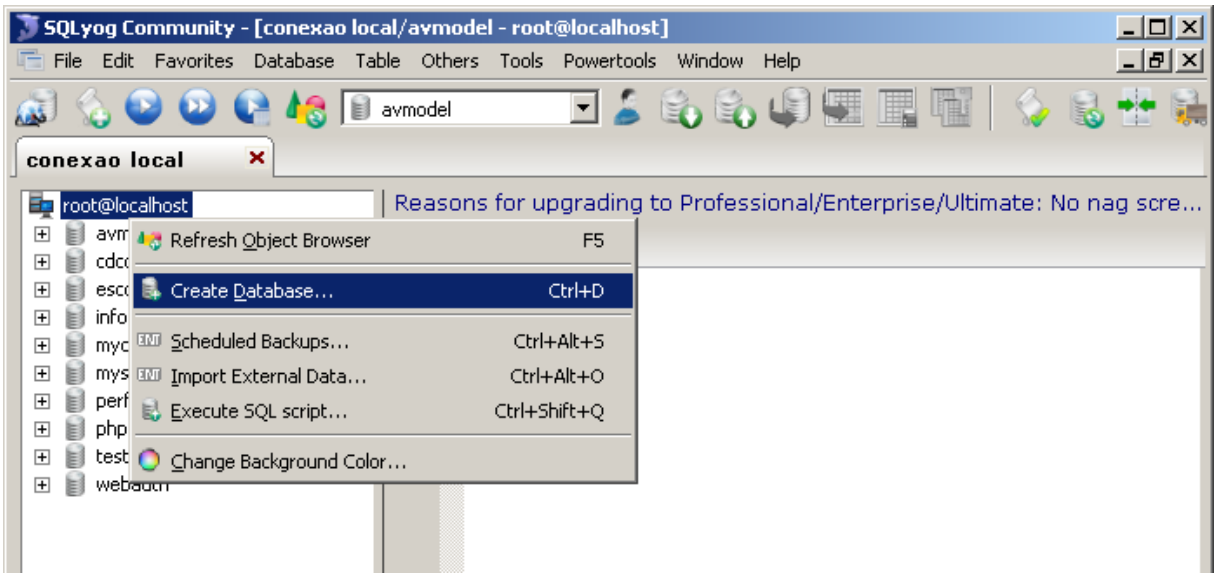
Crie uma conexão nova, se não existir



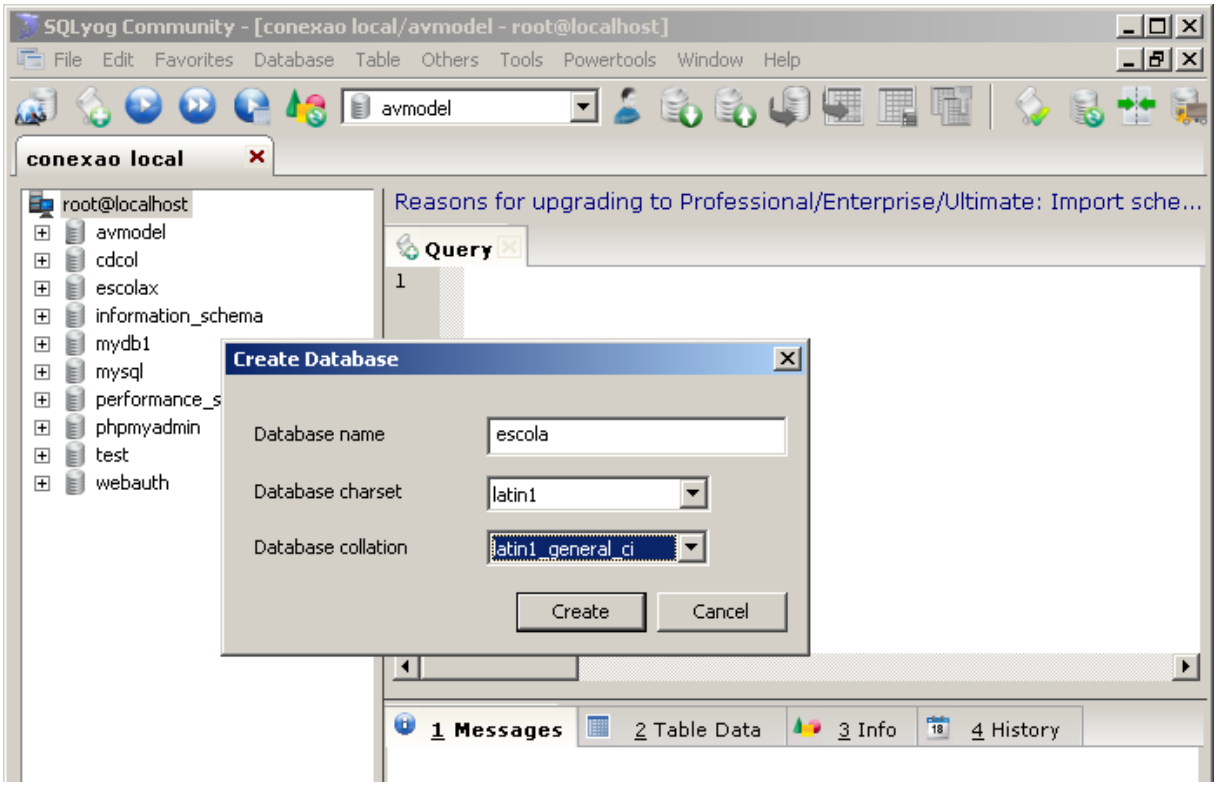
Preencha com as informações de conexão, teste a conexão, salve a conexão e conecte no servidor MySQL.



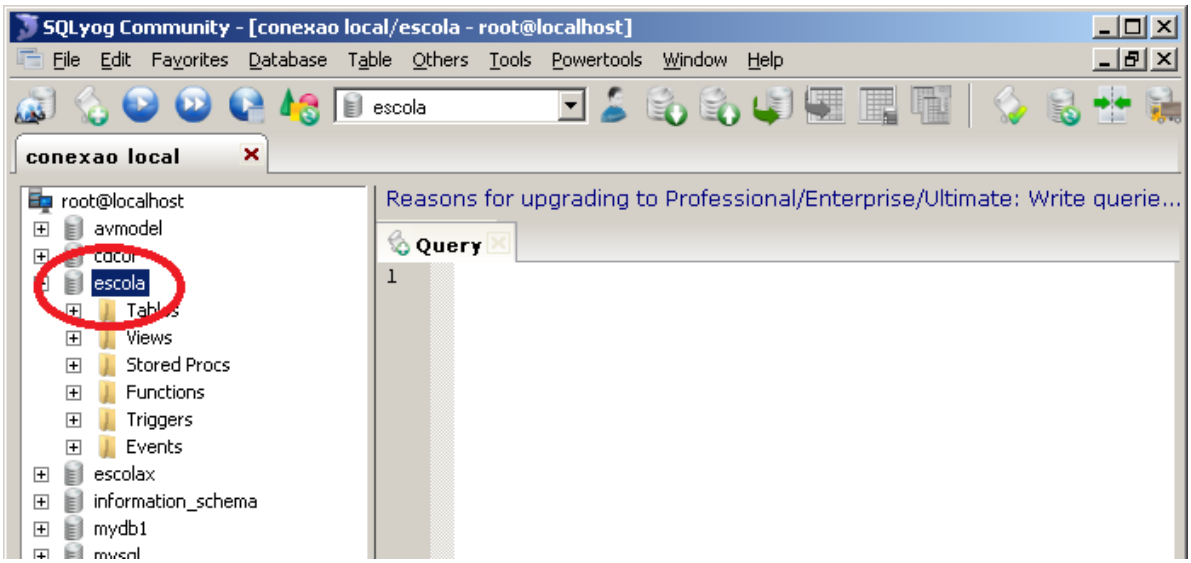
Clique sobre a raíz da árvore (root@localhost) com o botão direito do mouse e escolha "*create database*".



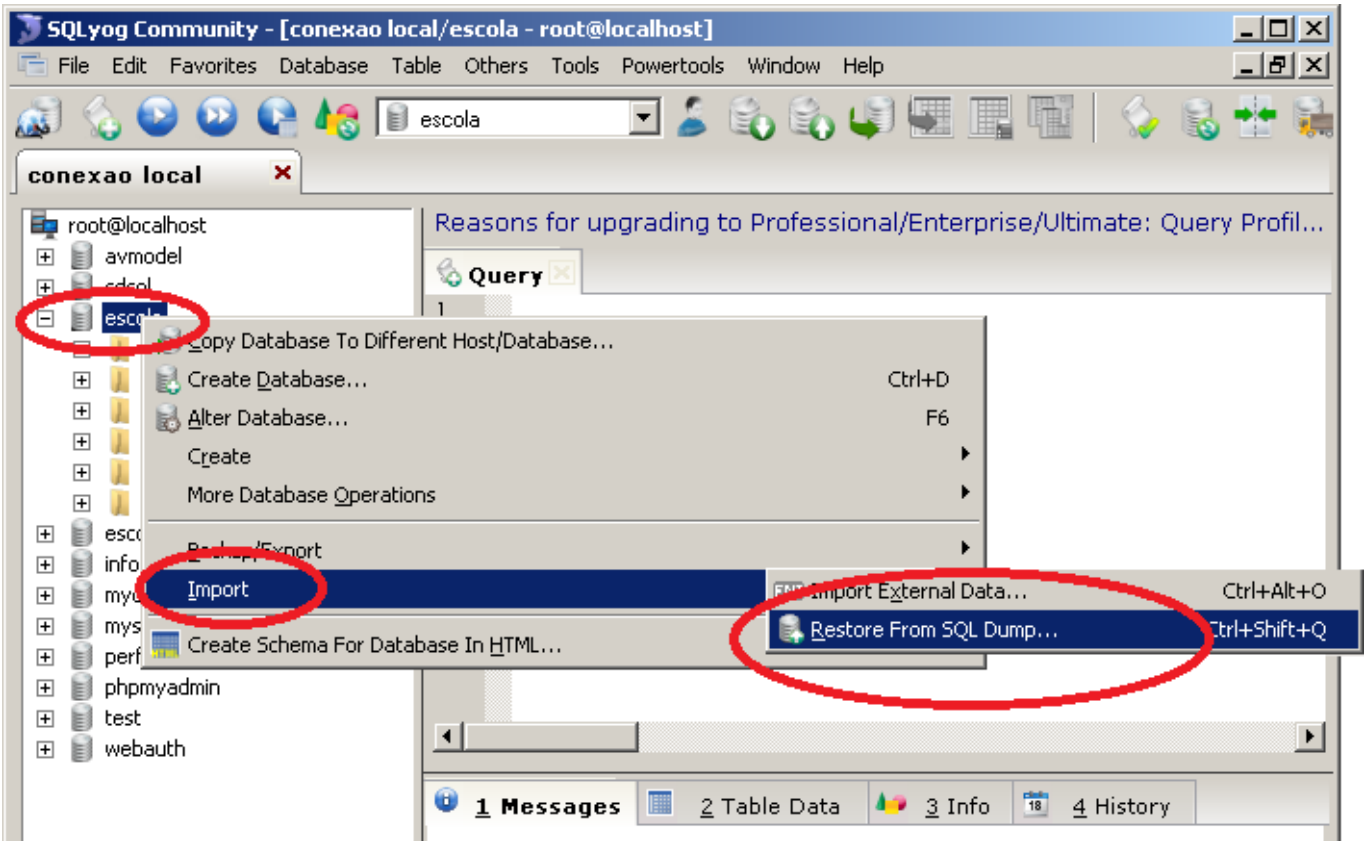
De o nome para o *database* (no exemplo **escola**), escolha o *charset* e o *collation*.



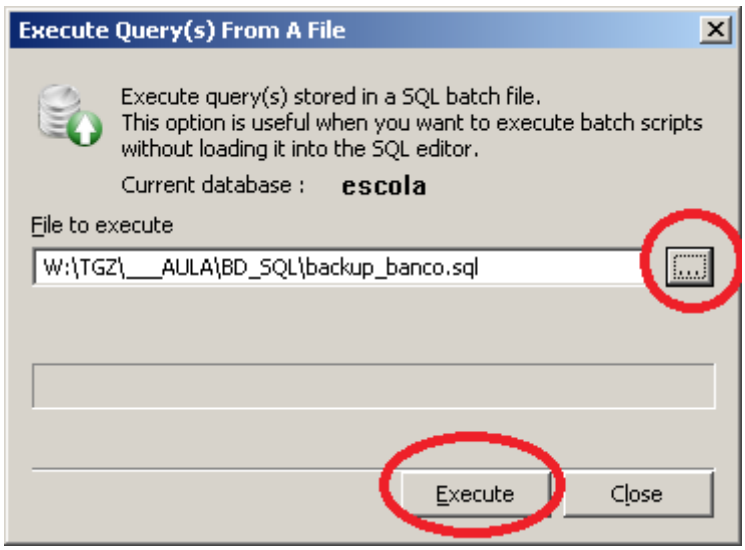
Você terá então criada uma estrutura para seu banco de dados **escola**.

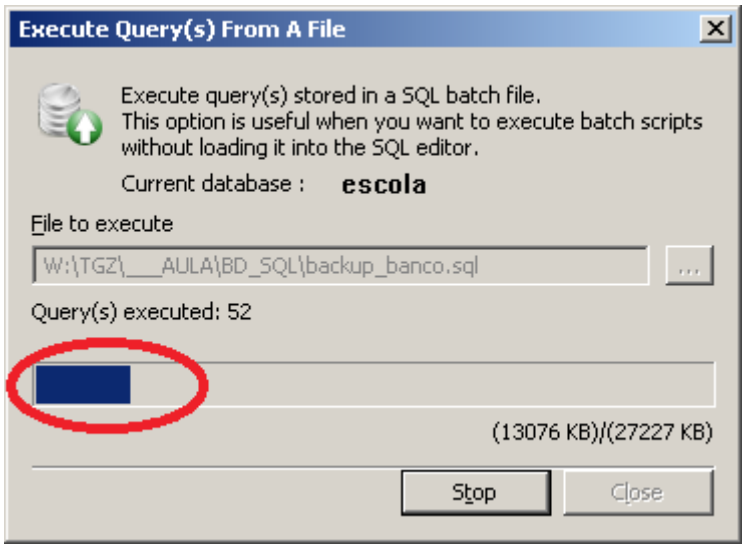


Clique com o botão direito sobre o banco de dados, escolha ***Import*** e ***Restore From SQL Dump*** (restaure de um *dump* sql).

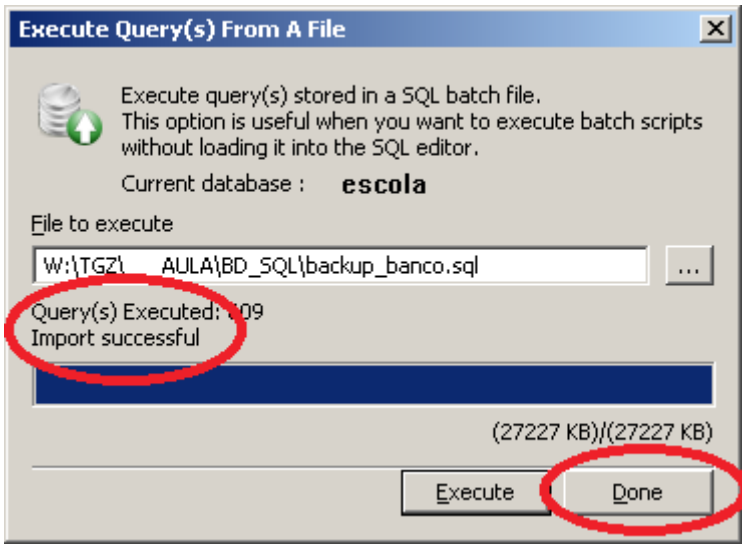


Localize o arquivo do *dump* (*backup*) pressionando o botão [**...**] e clique em execute.

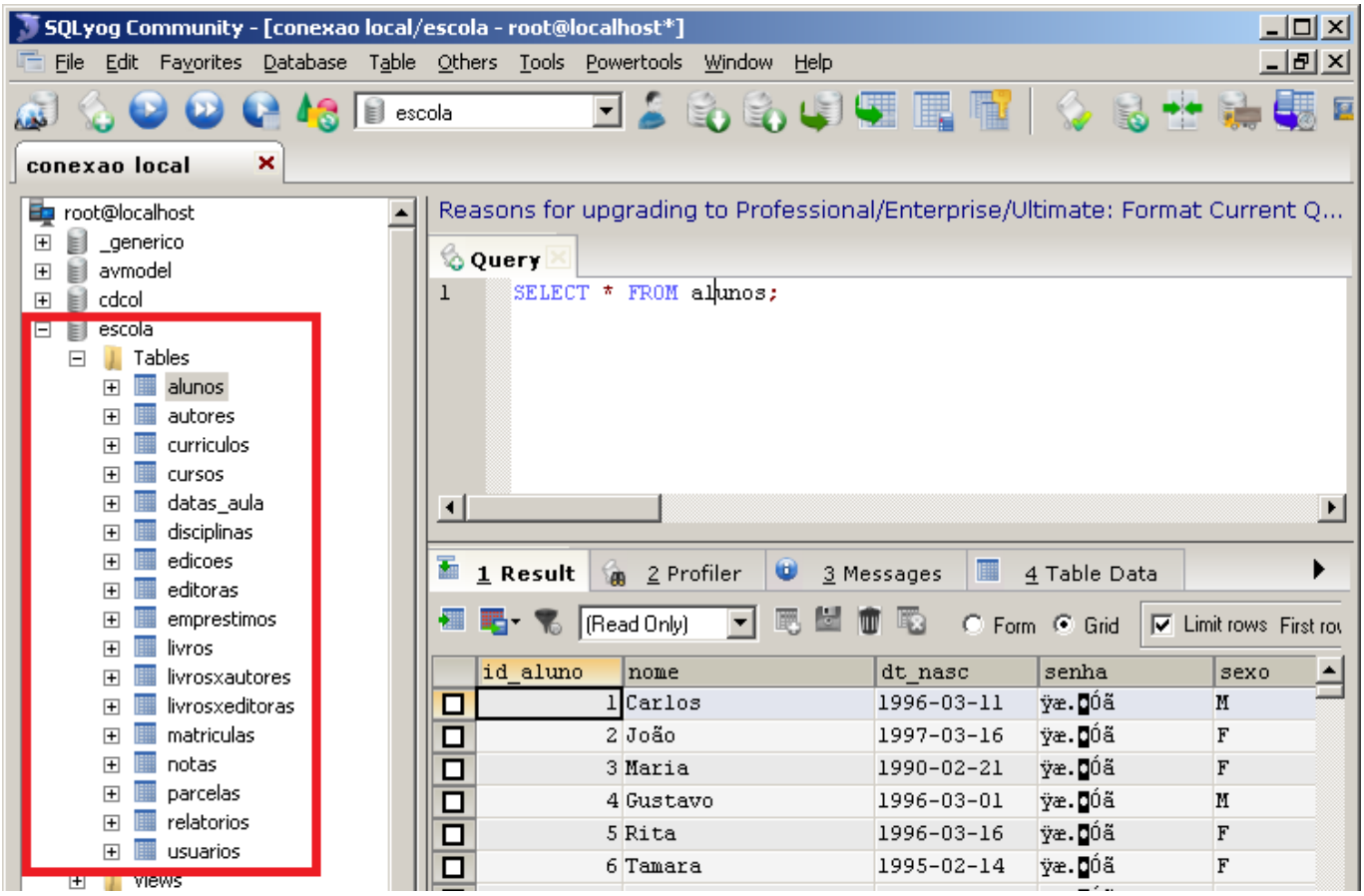




No final da importação deverá aparecer ***Import successful*** para confirmar que o *backup* foi restaurado. Pressione ***Done***.

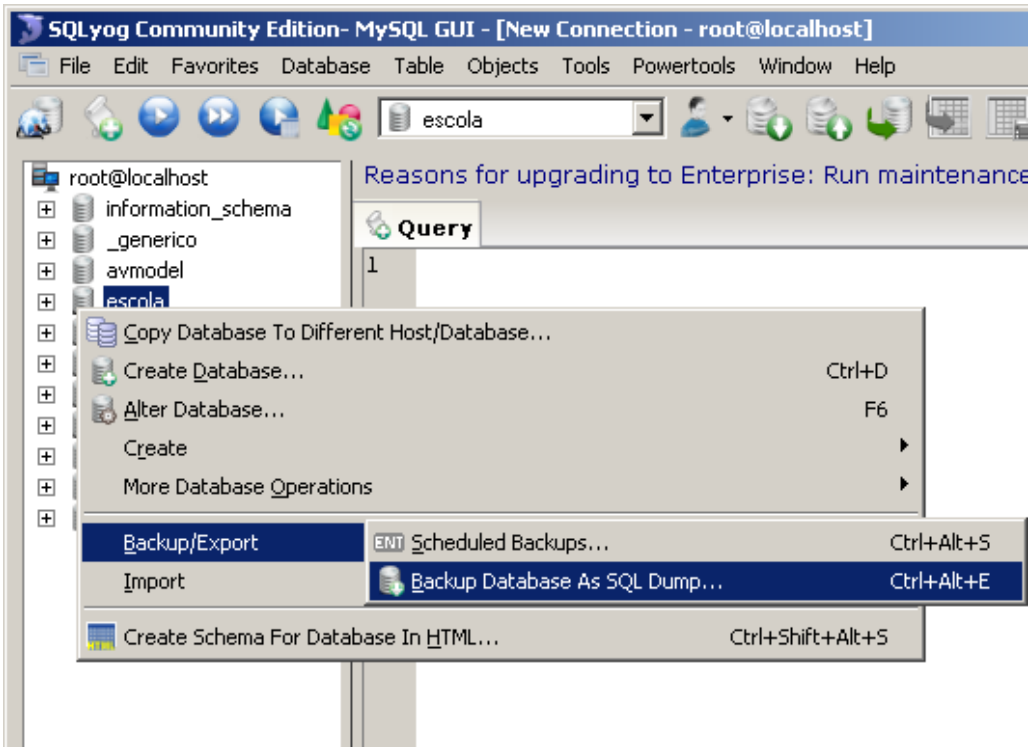


Após importado, a base de dados estará disponível com toda a estrutura e dados como a de origem.

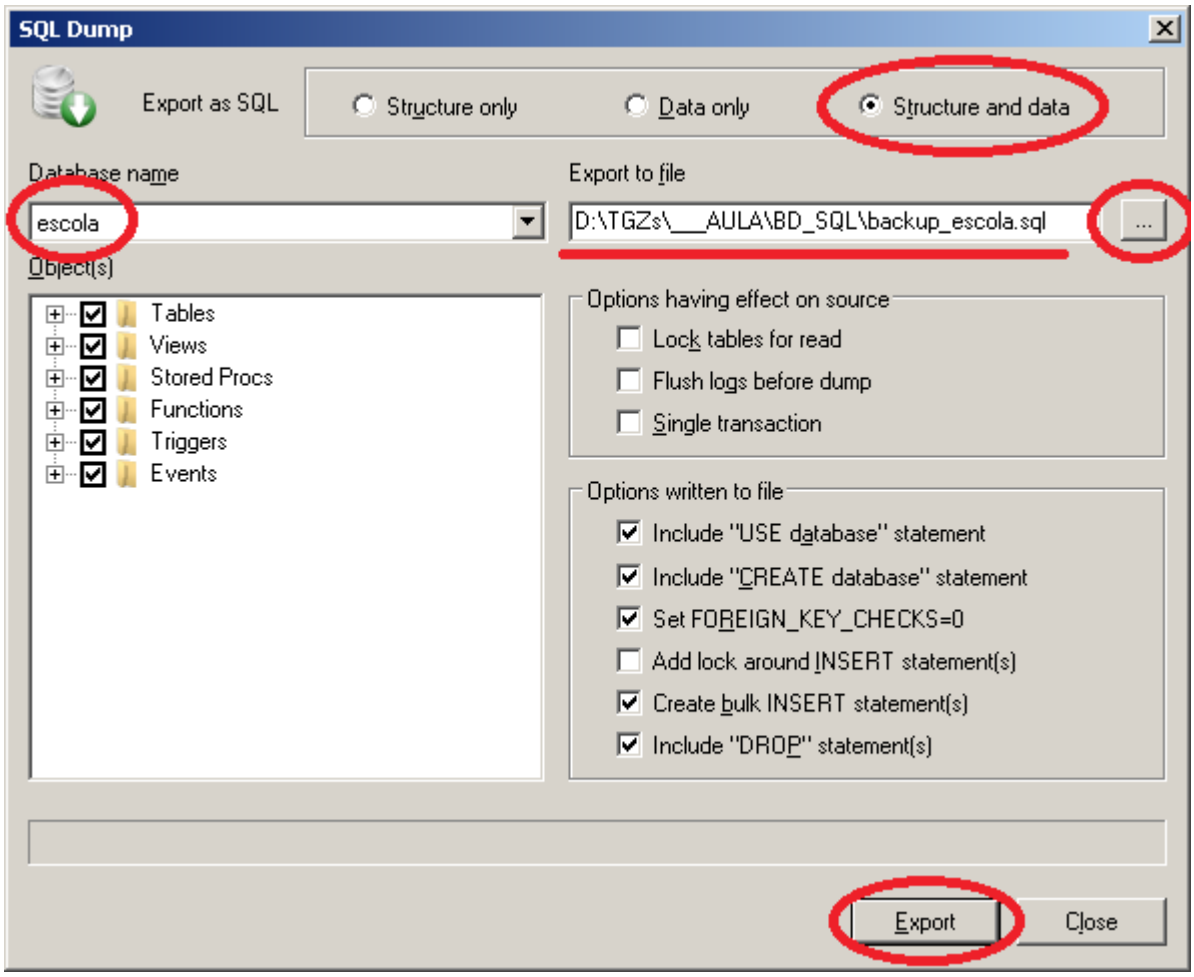


Efetuando *Backup* de um banco de dados

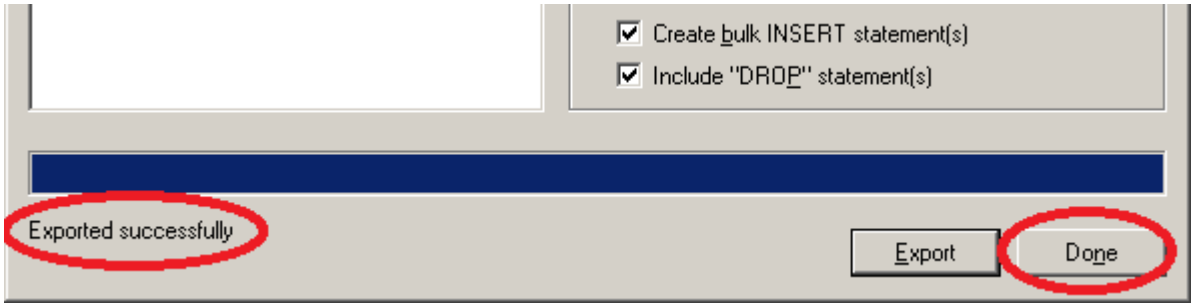
Clique com o botão direito sobre o banco de dados que deseja exportar e selecione ***Backup/Export*** – ***Backup Database As SQL Dump***.



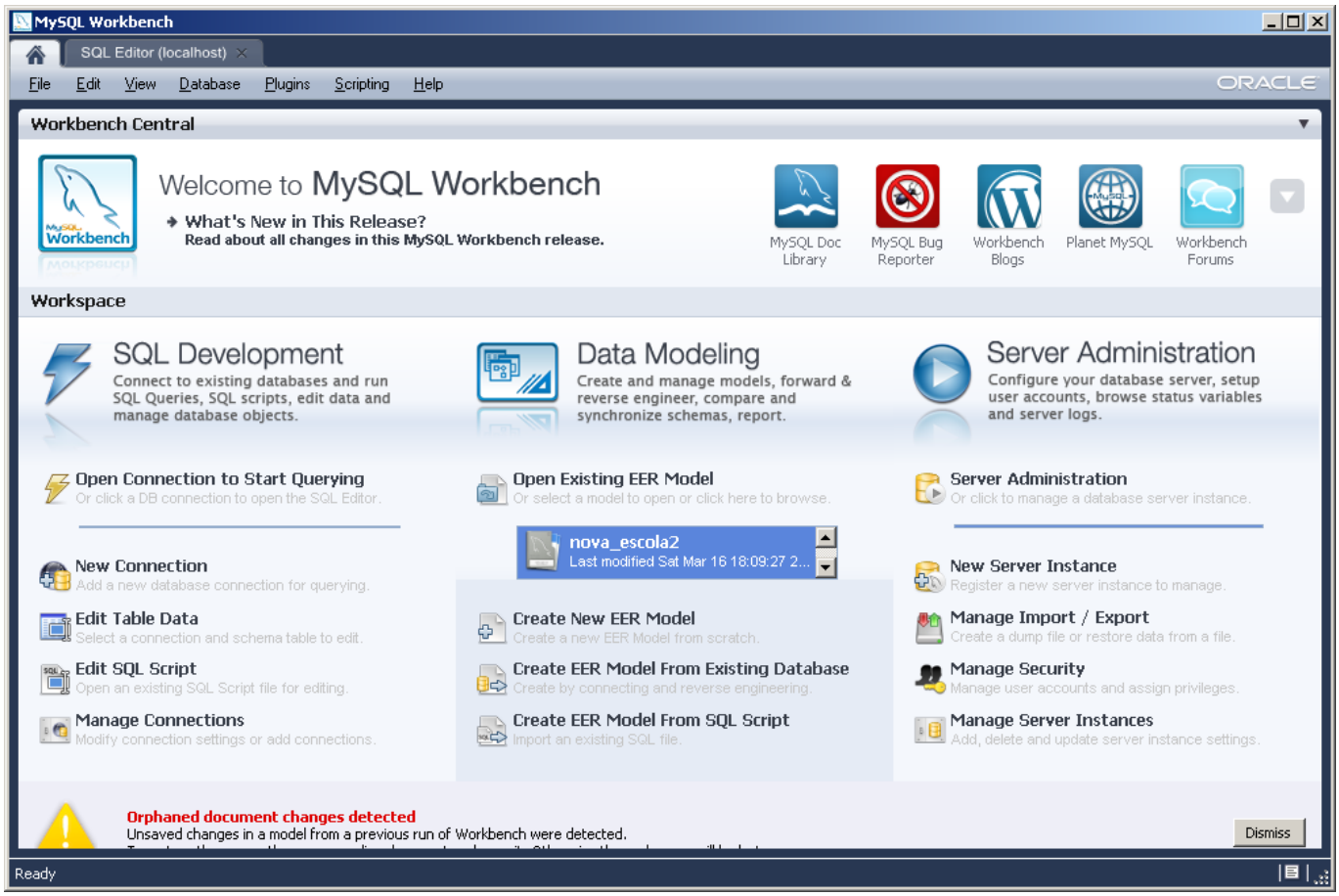
Verifique o nome da base de dados, pressione o botão [**...**] e escolha o nome do arquivo que será o *backup* (com a extensão .sql), verifique se está selecionado ***Structure and Data*** (para efetuar o *backup* da estrutura de dados e dos dados já gravados) e clique em ***Export***.

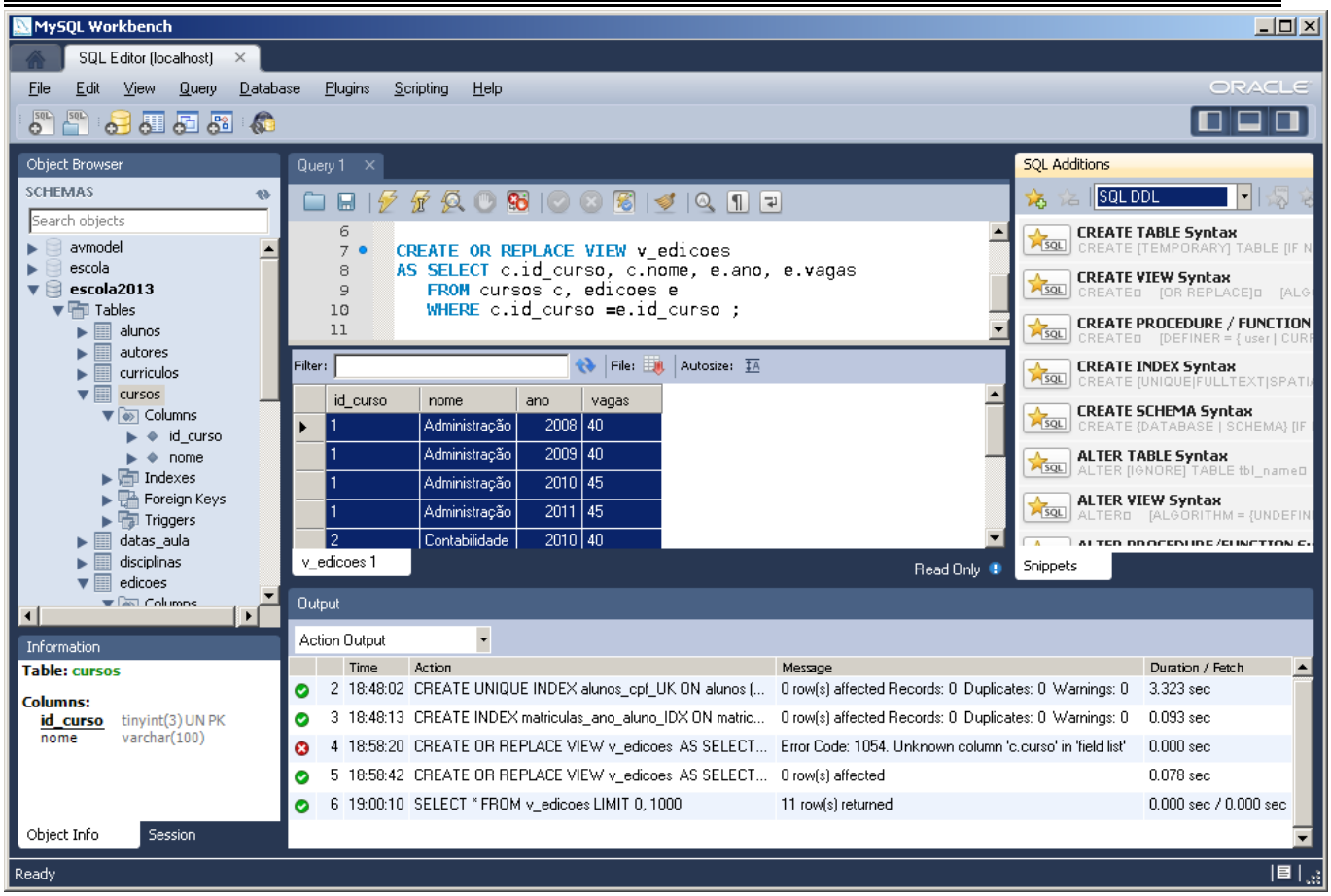


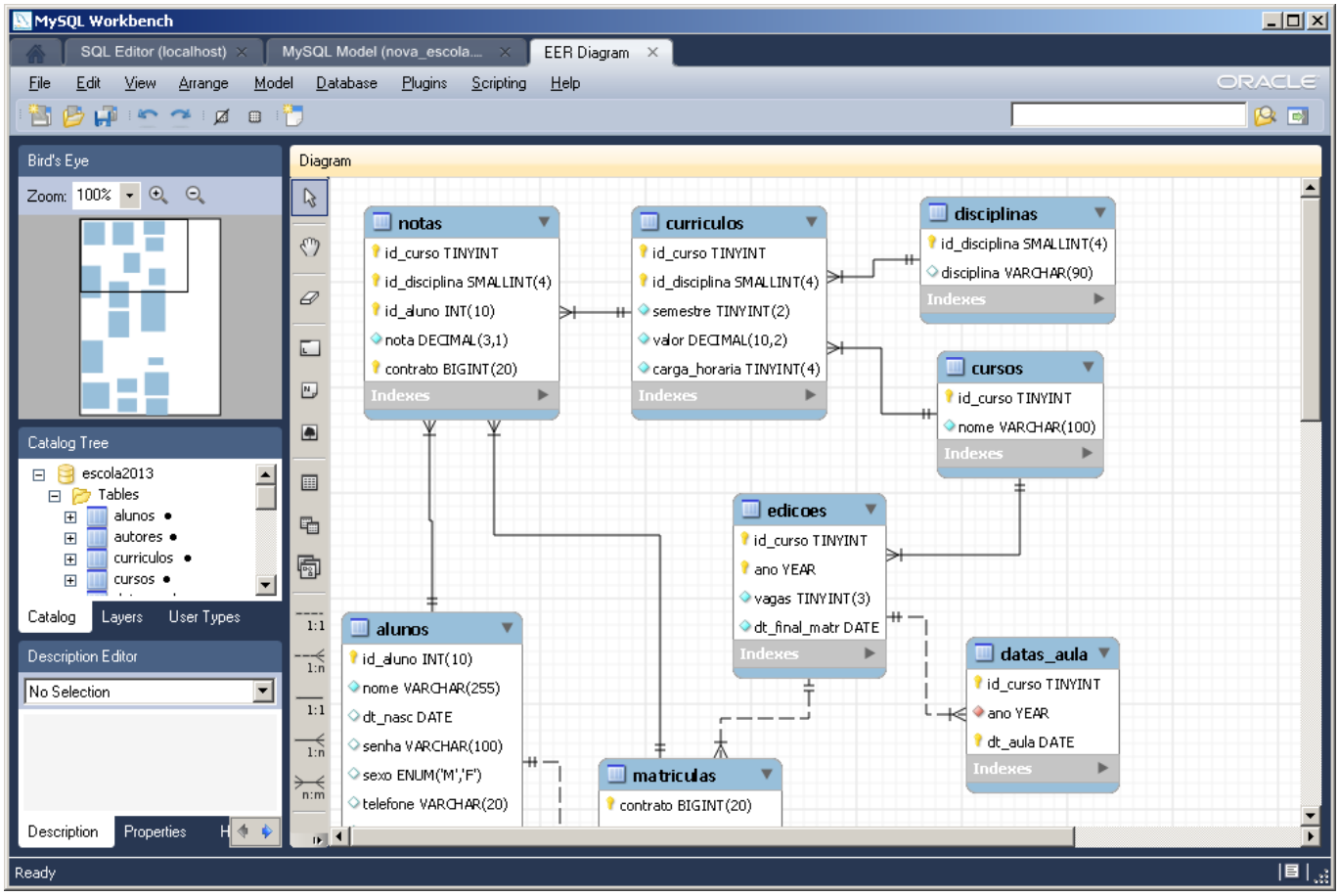
O *backup* será processado e a mensagem final deverá ser *Exported successfully*. Clique em ***Done*** para fechar.

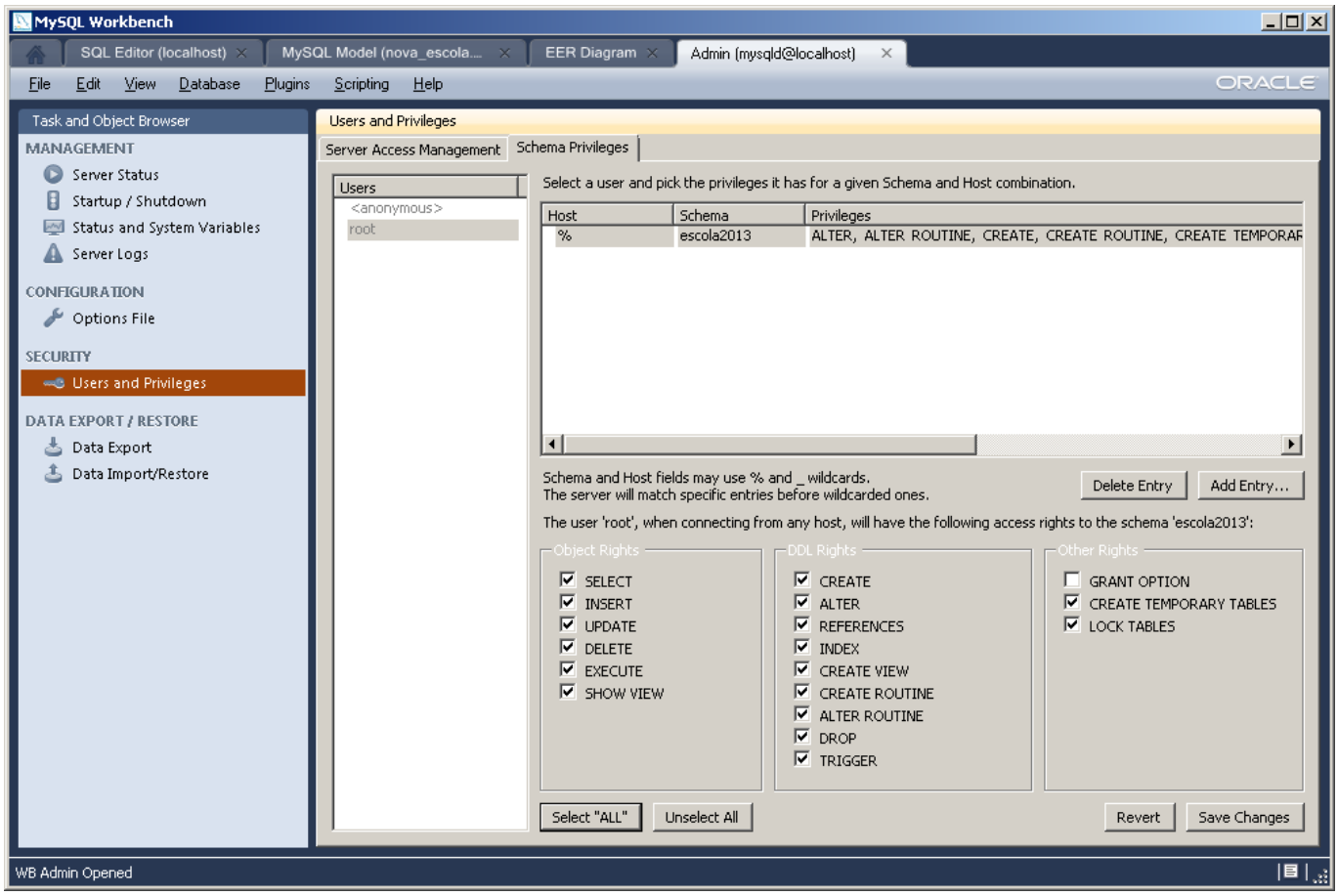


Usando o MySQL Workbench




MySQL Workbench
SQL Editor (localhost)
File Edit View Query Database Plugins Scripting Help
Object Browser
SCHEMAS
Search objects
avmodel
escola
escola2013
Tables
alunos
autores
curriculos
cursos
Columns
id_curso
nome
Indexes
Foreign Keys
Triggers
datas_aula
disciplinas
edicoes
Query 1
6
7 CREATE OR REPLACE VIEW v_edicoes
8 AS SELECT c.id_curso, c.nome, e.ano, e.vagas
9 FROM cursos c, edicoes e
10 WHERE c.id_curso =e.id_curso ;
11
Filter: File: Autosize:
id_curso nome ano vagas
1 Administração 2008 40
1 Administração 2009 40
1 Administração 2010 45
1 Administração 2011 45
2 Contabilidade 2010 40
v_edicoes 1
Read Only
SQL Additions
SQL DDL
CREATE TABLE Syntax
CREATE VIEW Syntax
CREATE PROCEDURE / FUNCTION
CREATE INDEX Syntax
CREATE SCHEMA Syntax
ALTER TABLE Syntax
ALTER VIEW Syntax
Snippets
Output
Action Output
Time Action Message Duration / Fetch
2 18:48:02 CREATE UNIQUE INDEX alunos_cpf_UK ON alunos (... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 3.323 sec
3 18:48:13 CREATE INDEX matriculas_ano_aluno_IDX ON matric... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.093 sec
4 18:58:20 CREATE OR REPLACE VIEW v_edicoes AS SELECT... Error Code: 1054. Unknown column 'c.curso' in 'field list' 0.000 sec
5 18:58:42 CREATE OR REPLACE VIEW v_edicoes AS SELECT... 0 row(s) affected 0.078 sec
6 19:00:10 SELECT * FROM v_edicoes LIMIT 0, 1000 11 row(s) returned 0.000 sec / 0.000 sec
Information
Table: cursos
Columns:
id_curso tinyint(3) UN PK
nome varchar(100)
Object Info
Session
Ready


MySQL Workbench
SQL Editor (localhost)
MySQL Model (nova_escola...
EER Diagram
File Edit View Arrange Model Database Plugins Scripting Help
Bird's Eye
Zoom: 100%
Catalog Tree
escola2013
Tables
alunos
autores
curriculos
cursos
Catalog
Layers
User Types
Description Editor
No Selection
Description
Properties
Diagram
notas
id_curso TINYINT
id_disciplina SMALLINT(4)
id_aluno INT(10)
nota DECIMAL(3,1)
contrato BIGINT(20)
Indexes
curriculos
id_curso TINYINT
id_disciplina SMALLINT(4)
semestre TINYINT(2)
valor DECIMAL(10,2)
carga_horaria TINYINT(4)
Indexes
disciplinas
id_disciplina SMALLINT(4)
disciplina VARCHAR(90)
Indexes
cursos
id_curso TINYINT
nome VARCHAR(100)
Indexes
edicoes
id_curso TINYINT
ano YEAR
vagas TINYINT(3)
dt_final_matr DATE
Indexes
alunos
id_aluno INT(10)
nome VARCHAR(255)
dt_nasc DATE
senha VARCHAR(100)
sexo ENUM('M','F')
telefone VARCHAR(20)
matriculas
contrato BIGINT(20)
datas_aula
id_curso TINYINT
ano YEAR
dt_aula DATE
Indexes
Ready


MySQL Workbench
SQL Editor (localhost)
MySQL Model (nova_escola....
EER Diagram
Admin (mysqld@localhost)
File Edit View Database Plugins Scripting Help
ORACLE
Task and Object Browser
MANAGEMENT
Server Status
Startup / Shutdown
Status and System Variables
Server Logs
CONFIGURATION
Options File
SECURITY
Users and Privileges
DATA EXPORT / RESTORE
Data Export
Data Import/Restore
Users and Privileges
Server Access Management
Schema Privileges
Users
<anonymous>
root
Select a user and pick the privileges it has for a given Schema and Host combination.
Host
Schema
Privileges
%
escola2013
ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORAR
Schema and Host fields may use % and _ wildcards.
The server will match specific entries before wildcarded ones.
Delete Entry
Add Entry...
The user 'root', when connecting from any host, will have the following access rights to the schema 'escola2013':
Object Rights
SELECT
INSERT
UPDATE
DELETE
EXECUTE
SHOW VIEW
DDL Rights
CREATE
ALTER
REFERENCES
INDEX
CREATE VIEW
CREATE ROUTINE
ALTER ROUTINE
DROP
TRIGGER
Other Rights
GRANT OPTION
CREATE TEMPORARY TABLES
LOCK TABLES
Select "ALL"
Unselect All
Revert
Save Changes
WB Admin Opened

Utilizando o XAMPP