MD - 1

Comandos SQL no MYSQL
Apostila 2

Fernando José Moreira de Lima

EXERCÍCIOS DE MYSQL

CRIAÇÃCO DO BD E DAS TABELAS, INSERÇÃO E CONSULTA DE REGISTROS

1. Manuseio do banco de dados

As seguintes instruções manuseiam o banco de dados:

cria o banco	create database [if not exists] <nome>
mostra os bancos existentes	show databases
coloca um BD em uso	use <nome>
exclui um BD	drop database <nome>

2. Criação de tabelas

A tabela para o cadastro de funcionários deve se chamar *cadfun*, utiliza alguns dos principais comandos da linguagem SQL e tem a seguinte estrutura:

Campo	*Tipo*	*Descrição*
CodFun**	integer	código do funcionário (não nulo)
Nome	varchar(40)	nome do funcionário (não nulo)
Depto	char (2)	departamento onde está locado o funcionário
Funcao	char (20)	função do funcionário
Salario	decimal (10, 2)	salário do funcionário

** chave primária.

Antes de criar a tabela, vamos criar um novo BD *virtual*, que conterá a maior parte dos dados que usaremos nas lições seguintes. Para isto, digite o seguinte:

```
create database virtual;
use virtual;
```

Para criar a tabela, digite o comando abaixo. Compare a sintaxe do comando digitado com a descrição da estrutura da tabela. Para efeito didático, a instrução foi digitada em várias linhas para destacar a endentação; observe a colocação das vírgulas e parênteses e que o ; (ponto e vírgula) encerra a instrução.

```
create table CadFun (
    CODFUN  integer not null primary key,
    NOME    varchar(40) not null,
    DEPTO    char( 2),
    FUNCAO  char(20),
    SALARIO  decimal(10, 2)
);
```

Para mostrar as tabelas do BD em uso, digite: *show tables;*

Para mostrar a estrutura da tabela, ou de apenas um campo, digite:

```
describe CadFun;
describe CadFun CODFUN;
```

3. Inserção de registros

A inserção de registros pode ser feita pelos comandos INSERT e pelo comando LOAD DATA. O primeiro insere diretamente os registros na tabela, após a execução de cada *Insert*; já *Load data* permite inserir de uma vez os comandos gravados em um arquivo. *Insert* tem o seguinte formato (atenção com os parênteses):

```
INSERT INTO <tabela>
(campo1, campo2, campo3…, campoN)
VALUES (valor1, valor2, valor3…, valorN);
```

Insira o primeiro registro. Observe os parênteses, as vírgulas e as aspas para os campos tipo char, bem como o ponto decimal e o ; (ponto e vírgula).

```
INSERT INTO cadfun
 (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
VALUES (12, 'CARLOS ALBERTO', '3', 'VENDEDOR', 1530.00);
```

Só por castigo, ou melhor, para treinar um pouco mais, acrescente mais dois registros:

codfun	*nome*	*depto*	*funcao*	*salario*
15	MARCOS HENRIQUE	2	GERENTE	1985.75
7	APARECIDA SILVA	3	SECRETARIA	1200.50

Quando um campo deva ser inserido com NULL, basta suprimir o nome deste campo da lista de campos. Na nossa tabela, os campos CodFun e Nome não podem ser nulos. Por exemplo, o comando a seguir insere um registro sem informar o campo FUNCAO.

```
INSERT INTO cadfun
 (CODFUN, NOME, DEPTO, SALARIO)
VALUES (44, JACIBA DA SILVA', '3', 1500.00);
```

Quando estiver inserindo todos os campos do registro e na sua ordem correta, você pode suprimir a primeira parte da instrução. Insira mais dois registros digitando o seguinte:

```
INSERT INTO cadfun VALUES (
  2, 'WILSON DE MACEDO', '3', 'PROGRAMADOR', 1050.00);
INSERT INTO cadfun VALUES (
  5, 'AUGUSTO SOUZA', '3', 'PROGRAMADOR', 1050.00);


INSERT INTO cadfun (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
    VALUES (4, ‘CARLOS BASTOS’, ‘5', 'VENDEDOR', 1530.00);
INSERT INTO cadfun (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
    VALUES (25, ‘PEDRO SILVA’, ‘3', ‘SUPERVISOR', 1599.51);
INSERT INTO cadfun (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
    VALUES (3, ‘ANA BASTOS’, ‘5', 'VENDEDORA', 1530.00);
INSERT INTO cadfun (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
    VALUES (10, ‘PAULO DA SILVA’, ‘2', ‘VENDEDOR’, 1530.00);
INSERT INTO cadfun (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
    VALUES (9, ‘SOLANGE PACHECO’, ‘5', ‘SUPERVISORA’, 1599.51);
INSERT INTO cadfun (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
    VALUES (6, ‘MARCELO SOUZA’, '3', ‘ANALISTA’, 2250.00);
INSERT INTO cadfun (CODFUN, NOME, DEPTO, FUNCAO, SALARIO)
    VALUES (1, ‘CELIA NASCIMENTO’, ‘2', ‘SECRETARIA’, 1200.50);
```

4. Consulta de registros

A consulta pode ser feita com a seguinte instrução:

SELECT [tipo] <campos> FROM <tabela> [condição];

- Tipo: pode ser ALL (todos os registros) ou DISTINCT (registros distintos).
- Campos: é uma lista dos campos, separada por vírgula; caso se use * (asterisco), deseja-se ver todos os campos.
- Condição: é dada por outras cláusulas WHERE, ORDER BY..., etc, que serão detalhadas mais à frente.

Para ver todos os funcionários cadastrados, digite: *SELECT * FROM cadfun;*

Para ver somente o nome e a função, digite: *SELECT NOME, FUNCAO FROM cadfun;*

Para ver quem trabalha em um departamento, digite:

```
SELECT NOME, CODFUN FROM cadfun
    WHERE DEPTO = '3';
```

Para mudar a ordem da consulta, ascendente e descendente, digite estes exemplos:

```
SELECT NOME, SALARIO FROM cadfun
    ORDER BY NOME;

SELECT NOME, SALARIO FROM cadfun
    ORDER BY NOME DESC;

SELECT NOME, SALARIO FROM cadfun
    ORDER BY NOME DESC;

SELECT DEPTO, NOME FROM cadfun
    ORDER BY DEPTO, NOME DESC;

SELECT NOME FROM cadfun
    WHERE DEPTO = '3'
    ORDER BY NOME;
```

5. Alteração de registros

A alteração de registros pode ser feita com a seguinte instrução:

UPDATE <tabela> SET <campo1> = <expressão1>, <campo2> = <expressão2>, ..., <campoN> = <expressãoN>
[condição];

- *Campo*: nome da coluna da tabela que se deseja atualizar. *Expressão*: valor que se deseja atribuir ao campo. A mesma instrução UPDATE pode alterar vários campos do registro.
- *Condição*: é dada pela cláusula WHERE.

Vamos alterar o salário do funcionário 7 e transferir a Ana Bastos do departamento 5 para o 3. Os SELECT mostram o estado antes e depois dos UPDATE.

```
SELECT * FROM cadfun WHERE CODFUN = 7;

UPDATE cadfun SET SALARIO = 2300.56 WHERE CODFUN = 7;

SELECT * FROM cadfun WHERE CODFUN = 7;

SELECT * FROM cadfun WHERE NOME = 'ANA BASTOS';

UPDATE cadfun SET DEPTO = '3' WHERE NOME = 'ANA BASTOS';

SELECT * FROM cadfun WHERE NOME = 'ANA BASTOS';
```

Vamos também conceder um reajuste de 10% para todos os empregados. Digite:

```
SELECT * FROM cadfun;
UPDATE cadfun SET SALARIO = SALARIO * 1.10;
SELECT * FROM cadfun;
```

6. Exclusão de registros

A exclusão de registros pode ser feita com a seguinte instrução:

```
DELETE FROM <tabela> [condição];
```

- *Condição*: é dada pela cláusula WHERE. Atenção!, pois se esta cláusula for omitida, é feita a exclusão de todos os registros da tabela.

Vamos excluir o departamento 2 inicialmente e depois o empregado Marcelo Souza. Os SELECT mostram o estado antes e depois dos DELETE.

```
SELECT * FROM cadfun;
SELECT DISTINCT DEPTO FROM cadfun;
DELETE FROM cadfun WHERE DEPTO = '2';
SELECT * FROM cadfun;
SELECT DISTINCT DEPTO FROM cadfun;
DELETE FROM cadfun WHERE NOME = 'MARCELO SOUZA';
SELECT * FROM cadfun;
```

Embora possamos excluir todos os registros de uma tabela com DELETE, a instrução abaixo faz isto com maior eficiência:

```
TRUNCATE TABLE <tabela>;
```

7. Alteração da estrutura da tabela

A alteração da estrutura da tabela pode ser feita com a seguinte instrução:

```
ALTER TABLE <tabela> <operação>;
```

- *Operação*: permite adicionar e excluir colunas de uma tabela:
  - ADD <campo> <tipo>: insere uma nova coluna <campo>, informando em seguinda seu <tipo> de dados,
  - DROP <campo>: elimina um campo da tabela.

Vamos colocar na tabela o campo ADMISSAO (data de admissão). Em seguida, vamos preencher os dados destes campos. No mySql o formato da data é: ‘aaaa-mm-dd’; em outros SGBD este formato pode ser diferente. Os SELECT mostram o estado da tabela antes e depois.

```
SELECT * FROM cadfun;
ALTER TABLE cadfun ADD ADMISSAO DATE;
SELECT * FROM cadfun;
UPDATE cadfun SET ADMISSAO = '2006-01-15' WHERE CODFUN = 2;
UPDATE cadfun SET ADMISSAO = '1999-10-21' WHERE CODFUN = 3;
UPDATE cadfun SET ADMISSAO = '2004-10-21' WHERE CODFUN = 4;
UPDATE cadfun SET ADMISSAO = '2006-04-26' WHERE CODFUN = 5;
UPDATE cadfun SET ADMISSAO = '1980-05-10' WHERE CODFUN = 7;
UPDATE cadfun SET ADMISSAO = '1999-12-15' WHERE CODFUN = 9;
UPDATE cadfun SET ADMISSAO = '2000-12-21' WHERE CODFUN = 12;
UPDATE cadfun SET ADMISSAO = '2000-10-21' WHERE CODFUN = 25;
```

```
SELECT * FROM cadfun;
```

Vamos aproveitar e testar o campo de data. O primeiro select usa uma função para extrair o mês; o segundo usa >=. Digite:

```
SELECT NOME, ADMISSAO FROM cadfun WHERE MONTH(ADMISSAO) = 12;
SELECT NOME, ADMISSAO FROM cadfun WHERE ADMISSAO >= '2000-01-01';
```

Para preparar a tabela para os próximos exercícios, vamos recriar o departamento 2, readmitir um empregado e admitir outros três. Digite:

```
INSERT INTO cadfun VALUES (
    15, 'MARCOS HENRIQUE', '2', 'GERENTE', 2184.33, '2006-05-25');
INSERT INTO cadfun VALUES (
    20, 'AUDREY TOLEDO', '2', 'SUPERVISORA', 1700.00, '2006-07-05');
INSERT INTO cadfun VALUES (
    22, 'SANDRA MANZANO', '2','ANALISTA', 2000.00, '2006-07-01');
INSERT INTO cadfun VALUES (
    24, 'MARCIO CANUTO', '2', 'PROGRAMADOR', 1200.00, '2006-07-10');
SELECT * FROM cadfun;
```

A microempresa deve ter ficado com doze empregados.

8. Criação da tabela do arquivo morto

Frequentemente precisamos guardar registros excluídos para atender consultas que abordam dados passados. No nosso caso, vamos criar uma tabela *morto* com a mesma estrutura da *CadFun* para guardar os empregados demitidos. Digite:

```
CREATE TABLE morto (
    CODFUN   INTEGER PRIMARY KEY,
    NOME     VARCHAR(40),
    DEPTO    CHAR( 2),
    FUNCAO   CHAR(20),
    SALARIO  DECIMAL(10, 2),
    ADMISSAO DATE);
SHOW TABLES;
```

Para copiar os dados de uma tabela para outra podemos usar um INSERT casado com um SELECT, no formato abaixo:

```
INSERT INTO <tabeladestino>
    SELECT <campos>
    FROM <tabelafonte>
    WHERE <condição>;
```

- Tabeladestino: tabela onde será feito o INSERT;
- Campos: lista de campos da <tabelafonte>. É importante que os <campos> estejam na mesma ordem e sejam do mesmo tipo que os da <tabeladestino>. Neste caso, não pode ser usado o * asterisco.
- Tabelafonte: a tabela que fornecerá os registros.
- Condição: a condição do SELECT.

Vamos demitir o funcionário 12. Primeiro o inserimos no morto e depois o deletamos do CadFun. Digite:

```
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO
    FROM cadfun
    WHERE CODFUN = 12;
```

```
SELECT * FROM morto;
DELETE FROM cadfun WHERE CODFUN = 12;
SELECT * FROM cadfun;
SELECT * FROM morto;
```

9. Subconsultas

Podemos basear uma query em outra. Isto permite estruturar as consultas de forma que possamos isolar partes da instrução e testá-las separadamente (o que chamamos de depuração ou debug). A uma query assim construída dá-se o nome de subquery (subconsulta). Por exemplo, vamos fazer (1) uma consulta para os nomes dos empregados ativos que tenham salário igual aos dos funcionários demitidos, e (2) uma consulta para saber os empregados cujo departamento é igual ao dos funcionários demitidos. Para isto, digite:

```
SELECT NOME FROM cadfun;
SELECT SALARIO FROM morto;
SELECT NOME FROM cadfun
    WHERE SALARIO = (SELECT SALARIO FROM morto);
SELECT * FROM cadfun
    WHERE SALARIO = (SELECT SALARIO FROM morto);
SELECT DEPTO FROM morto;
SELECT * FROM cadfun
    WHERE DEPTO = (SELECT DEPTO FROM morto);
```

10. Exercícios adicionais

- Mostre nomes e funções de todos os funcionários.
- Mostre códigos, nomes e funções de todos os funcionários.
- Mostre códigos, nomes, funções e departamentos de todos os funcionários.
- Mostre os nomes dos funcionários do departamento 5.
- Mostre nomes e departamentos de todos os funcionários que ocupam o cargo de vendedor.
- Mostre quantas funções diferentes existem no departamento 5.
- Mostre os salários dos funcionários do departamento 3 ordenados de forma descendente.
- Mostre os nomes e funções de todos os funcionários, ordenados de forma descendente pelo nome da função.
- Mostre os nomes, funções e salários de todos os funcionários, ordenados de forma ascendente pelo nome da função e de forma descendente pelo salário.
- Mostre os registros dos funcionários que tenham sido admitidos no mês de outubro de qualquer ano.
- Mostre os registros dos funcionários que tenham sido admitidos antes de 2005.
- Mostre os registros dos funcionários que tenham mais de 5 anos na empresa.
- Mostre quantas funções diferentes existem na empresa. Como existem casos em que está escrito “VENDEDORA”, “SUPERVISORA”..., faça algumas sql de update para trocar para “VENDEDOR”, “SUPERVISOR”, etc. Obs.: tem de ser feita uma sql para cada cargo cuja notação se deseja alterar.

LIÇÃO 4 – CONSULTAS COM OPERADORES

1. Operadores aritméticos

Os operadores aritméticos (soma, subtração, multiplicação, divisão e resto) estão descritos na tabela anexa "tabela de operadores e funções matemáticas". Os operadores podem ser usados dentro de um comando Select: eles criam um campo calculado dentro da consulta, mas não afetam os valores das tabelas. Verifique, após os exemplos abaixo, que a tabela permanece inalterada. Digite:

```
SELECT NOME, SALARIO FROM cadfun;
SELECT NOME, SALARIO + 100 FROM cadfun;
SELECT NOME, SALARIO FROM cadfun;
SELECT NOME, SALARIO * 1.20 FROM cadfun;
SELECT NOME, SALARIO FROM cadfun;
```

Digite também os exemplos abaixo para relembrar a calculadora do comando Select:

```
SELECT 2 + 3 * 5;
SELECT (2 + 3) * 5;
```

2. Operadores de comparação ou relacionais

Os operadores relacionais que podem ser usados no mySql constam na tabela a seguir.

Operador	*Descrição*
>	Maior do que
<	Menor do que
=	Igual a
<> ou !=	Diferente de
>=	Maior ou igual a
<=	Menor ou igual a
<=>	Igual para Null

Por exemplo, digite:

```
SELECT * FROM cadfun WHERE DEPTO = '5';
SELECT * FROM cadfun WHERE FUNCAO = 'VENDEDOR';
SELECT * FROM cadfun WHERE SALARIO <= 1700;
SELECT * FROM cadfun WHERE SALARIO > 1700 + 50;
SELECT * FROM cadfun WHERE SALARIO > (1700 + 50);
```

3. Operadores lógicos

Os operadores lógicos conhecidos na programação também podem ser usados no mySql e sua notação consta na tabela a seguir.

Operador	*Operador*	*Descrição*
AND	&&	E – conjunção
OR	||	Ou – disjunção
NOT	!	Não – negação
XOR		Ou exclusivo

Por exemplo, digite:

```
SELECT * FROM cadfun
    WHERE (DEPTO = '3') AND (FUNCAO = 'PROGRAMADOR');
SELECT * FROM cadfun
    WHERE (DEPTO = '3') OR (DEPTO = '5');
SELECT * FROM cadfun
    WHERE NOT (FUNCAO = 'VENDEDOR');
SELECT * FROM cadfun
    WHERE (DEPTO = '5') XOR (FUNCAO = 'PROGRAMADOR');
```

4. Operadores adicionais

Alguns operadores adicionais podem ser usados no mySql e sua notação e seu significado consta na tabela a seguir.

Operador	*Descrição*
IS NULL	Verifica se um campo está vazio.
BETWEEN	Verifica se um valor está dentro de um intervalo.
IN	Verifica se um valor esté em um conjunto de valores.
LIKE	Busca valores semelhantes.

Para testar estes operadores, crie um novo campo (a quantidade de filhos) nas tabelas CadFun e Morto e depois verifique seus valores. Para isto, digite:

```
ALTER TABLE cadfun ADD FILHOS SMALLINT;
ALTER TABLE morto ADD FILHOS SMALLINT;
SELECT NOME, FILHOS FROM cadfun;
SELECT CODFUN, NOME, FILHOS FROM cadfun
    WHERE FILHOS IS NULL;
SELECT CODFUN, NOME, FILHOS FROM cadfun
    WHERE NOT FILHOS IS NULL;
SELECT CODFUN, NOME, FILHOS FROM cadfun
    WHERE FILHOS IS NOT NULL;
SELECT CODFUN, NOME, FILHOS FROM cadfun
    WHERE NOT (FILHOS IS NULL);
```

Vamos alterar os valores deste campo para melhorar os exemplos. Para isto, digite:

```
UPDATE cadfun SET FILHOS = 1 WHERE CODFUN = 2;
UPDATE cadfun SET FILHOS = 3 WHERE CODFUN = 3;
UPDATE cadfun SET FILHOS = 2 WHERE CODFUN = 5;
UPDATE cadfun SET FILHOS = 1 WHERE CODFUN = 9;
UPDATE cadfun SET FILHOS = 4 WHERE CODFUN = 20;
UPDATE cadfun SET FILHOS = 3 WHERE CODFUN = 25;
SELECT CODFUN, NOME, FILHOS FROM cadfun;
SELECT CODFUN, NOME, FILHOS FROM cadfun
    WHERE FILHOS IS NULL;
SELECT CODFUN, NOME, FILHOS FROM cadfun
    WHERE NOT (FILHOS IS NULL);
```

```
SELECT * FROM cadfun
    WHERE SALARIO BETWEEN 1700 AND 2000;
SELECT * FROM cadfun
    WHERE SALARIO NOT BETWEEN 1700 AND 2000;
SELECT * FROM cadfun
    WHERE DEPTO IN ('2', '3');
SELECT * FROM cadfun
    WHERE DEPTO NOT IN ('2', '3');
```

O operador LIKE é ainda mais poderoso, pois pode trabalhar com dois caracteres curinga:

% (percentagem) significa qualquer quantidade (0, 1, 2...) de caracteres combinando, (alguns BDs usam * asterisco),

_ (sublinhado) significa apenas um caractere igual.

Estes caracteres podem ser usados no início, meio e fim do campo, permitindo inúmeras situações e combinações. Alguns exemplos:

Exemplo: Where...	*Operação*
Salario like ‘11%’	Encontra valores que começam com 11.
Salario like ‘%8%’	Encontra valores que tenham 8 em qualquer posição.
Salario like ‘_0%’	Encontra valores que tenham 0 na segunda posição.
Salario like ‘1_%_%’	Encontra valores que começam com 1 e tenham 3 caracteres de comprimento (?? no mínimo??)
Salario like ‘%6’	Encontra valor que termine com 6.
Salario like ‘_1%6’	Encontra valores que tenham 1 na segunda posição e termine com 6.

Digite os comandos a seguir para testar o operador Like:

```
SELECT * FROM cadfun WHERE NOME LIKE 'A%';
SELECT * FROM cadfun WHERE NOME LIKE '_A%';
SELECT * FROM cadfun WHERE NOME LIKE '%AN%';
SELECT * FROM cadfun WHERE SALARIO LIKE '%6';
SELECT * FROM cadfun WHERE SALARIO LIKE '_5%2';
SELECT * FROM cadfun WHERE NOME NOT LIKE '%AN%';
```

Agora digite os comandos a seguir completando de acordo com a tabela anterior, para verificar os exemplos apresentados:

SELECT NOME, SALARIO FROM cadfun WHERE SALARIO LIKE …(complete de acordo com a tabela de exemplos).

LIÇÃO 5 – FUNÇÕES

Há um resumo dos tipos de função permitidos no anexo "TABELA DE OPERADORES E FUNÇÕES MATEMÁTICAS".

1. Funções de agregação ou estatísticas

Função	*Retorna...*
AVG()	Média aritmética
COUNT()	A quantidade de registros
MAX()	O maior valor
MIN()	O menor valor
STD() ou STDDEV()	O desvio padrão
SUM()	Soma dos valores
VARIANCE()	A variância

Para testar estas funções, digite:

```
SELECT AVG(SALARIO) FROM cadfun;
SELECT AVG(SALARIO) FROM cadfun WHERE DEPTO = '3';
SELECT COUNT(*) FROM cadfun WHERE DEPTO = '3';
SELECT COUNT(*) FROM cadfun WHERE SALARIO > 2000;
SELECT COUNT(DISTINCT DEPTO) FROM cadfun;
SELECT MAX(SALARIO) FROM cadfun;
SELECT MIN(SALARIO) FROM cadfun;
SELECT STD(FILHOS) FROM cadfun;
SELECT STDDEV(FILHOS) FROM cadfun;
SELECT SUM(SALARIO) FROM cadfun;
SELECT SUM(SALARIO) FROM cadfun WHERE DEPTO = '2';
SELECT VARIANCE(FILHOS) FROM cadfun;
```

Um exercício interessante é verificar como estas funções se comportam com valores nulos. Verifique primeiro se existem ainda campos SALARIO ou FILHOS com valores nulos. Caso não exista, inclua (INSERT) dois novos registros com estes dados nulos ou altere (UPDATE) dois registros existentes para colocar estes dados nulos. Em seguida, digite novamente os comandos abaixo para testar as funções com os valores nulos.

```
SELECT SALARIO FROM cadfun;
SELECT AVG(SALARIO) FROM cadfun;
SELECT SUM(SALARIO) FROM cadfun;
SELECT COUNT(*) FROM cadfun;
SELECT COUNT(SALARIO) FROM cadfun;
SELECT MIN(SALARIO) FROM cadfun;
SELECT FILHOS FROM cadfun;
SELECT STD(FILHOS) FROM cadfun;
SELECT VARIANCE(FILHOS) FROM cadfun;
```

2. Funções de data e hora

Função	*Retorna...*
CURDATE()	Data atual
CURTIME()	Hora atual

DATEDIFF()	Diferença entre duas datas
DAY()	O dia, de uma data
HOUR()	O valor da hora, de um campo de hora
MINUTE()	Os minutos, de um campo de hora
MONTH()	O mês, de uma data
MONTHNAME()	O nome do mês, de uma data
NOW()	Data e hora atual
SECOND()	Os segundos, de um campo de hora
YEAR()	O ano, de uma data

Para testar estas funções, digite as instruções abaixo. As duas primeiras calculam o número de dias ou de anos desde a data de admissão do empregado.

```
SELECT NOME, DATEDIFF('2006-09-15', ADMISSAO) FROM cadfun;
SELECT NOME, DATEDIFF('2006-09-15', ADMISSAO) / 365 FROM cadfun;
SELECT DAY(ADMISSAO), MONTH(ADMISSAO), YEAR(ADMISSAO)
    FROM cadfun;
SELECT NOME, ADMISSAO, MONTHNAME(ADMISSAO) FROM cadfun;
SELECT HOUR(NOW()), MINUTE(NOW()), SECOND(NOW());
```

3. Funções numéricas

Função	*Retorna...*
ABS()	Valor absoluto
ACOS(), ASIN(), ATAN()	Arco do cosseno, do seno ou da tangente
COS(), SIN(),TAN()	Cosseno, seno ou tangente
DEGREES()	Converte radianos para graus
EXP()	O exponencial
LN()	O logaritmo natural
MOD()	O resto de uma divisão de inteiros
PI()	O valor de PI
POWER()	A potência de uma base elevada a um expoente
RADIANS()	Converte graus para radianos
ROUND()	Arredonda
SQRT()	Extrai a raiz quadrada

Para testar estas funções, digite as instruções abaixo. Para o caso das funções trigonométricas, observe se os valores retornados consideram os parâmetros em radianos ou em graus. Para as funções de exponenciação e logarítmicas, observe se os valores retornados consideram os parâmetros na base 10 ou na base *e* (logaritmo natual).

```
SELECT ABS(-8.238765);
SELECT ABS(-9);
SELECT MOD(5, 2);
SELECT MOD(4, 2);
SELECT ROUND(1.9);
SELECT ROUND(1.3);
SELECT SQRT(9);
SELECT SQRT(2);
SELECT PI();
SELECT PI() + 0.000000000000000000;

SELECT ACOS(-1);
```

```
SELECT ACOS(0.5);
SELECT ASIN(-1);
SELECT ASIN(0.5);
SELECT ATAN(-1);
SELECT ATAN(0.5);
SELECT COS(-1);
SELECT COS(0.5);
SELECT SIN(1);
SELECT SIN(0.5);
SELECT TAN(1);
SELECT TAN(0.5);
SELECT DEGREES(ASIN(1));
SELECT DEGREES(ASIN(0.5));
SELECT RADIANS(1);
SELECT RADIANS(0.5);

SELECT EXP(3.4);
SELECT EXP(1);
SELECT LN(3);
SELECT LN(EXP(10));
SELECT POWER(2, 3);
SELECT POWER(2, -3);
```

4. Funções de tratamento de strings

Função	*Retorna...*
ASCII()	Código ASCII da string
BIN(), OCT(), HEX()	Converte decimal para binário, octal, hexadecimal
CONCAT()	Concatena strings
LOWER(),UPPER()	Converte para minúsculas, maiúsculas
LEFT()	Parte do lado esquerdo da string
LENGTH()	Tamanho da string

Para testar estas funções, digite as instruções abaixo.

```
SELECT ASCII('A');
SELECT ASCII('B');
SELECT ASCII('AB');
SELECT BIN(10);
SELECT BIN(255);
SELECT HEX(10);
SELECT HEX(255);
SELECT OCT(10);
SELECT OCT(255);
SELECT CONCAT('My', 'SQL');
SELECT CONCAT('Augusto', ' ', 'Manzano');
SELECT LEFT('Augusto Manzano', 5);
```

```
SELECT LEFT(NOME, 9) FROM cadfun WHERE DEPTO = '2';
SELECT LENGTH('Augusto Manzano');
SELECT NOME, LENGTH(NOME) FROM cadfun WHERE DEPTO = '2';
SELECT LOWER('Augusto Manzano');
SELECT NOME, LOWER(NOME) FROM cadfun WHERE DEPTO = '2';
SELECT UPPER('Augusto Manzano');
```

LIÇÃO 6 – AGRUPAMENTOS E UNIÕES (GROUP BY e UNION)

Até agora vimos seleções com uma única tabela. Esta lição e a próxima apresentam consultas com mais de uma tabela e também novas funções de agregação e agrupamento. O *agrupamento*, feito pela cláusula GROUP BY, usa um campo básico ou mais, para permitir a agregação das demais colunas da consulta. A *união*, feita pela cláusula UNION, é obtida combinando duas ou mais consultas em apenas uma; neste caso, os campos das consultas combinadas devem ter o mesmo significado.

Para permitir enriquecer os exemplos, vamos acrescentar novos registros ao BD Virtual. Digite os seguintes comandos:

```
INSERT INTO cadfun VALUES (
    1, 'SILVIO DE MENESES', '2', 'ANALISTA', 2000.00, '2006-08-10', 1);
INSERT INTO cadfun VALUES (
    6, 'SILVIA DA SILVA', '5', 'ANALISTA', 2000.00, '2006-08-10', 3);
INSERT INTO cadfun VALUES (
    8, 'ANTONIO DA SILVA', '5', 'GERENTE', 2184.33, '2006-09-05', 2);
INSERT INTO cadfun VALUES (
    10, 'ANTONIA DE CARVALHO', '5', 'SECRETARIA', 2530.62, '2006-09-07', 4);
INSERT INTO cadfun VALUES (
    11, 'SILVANA DOS SANTOS', '4', 'VENDEDOR', 1683.00, '2006-09-07', 0);
INSERT INTO cadfun VALUES (
    12, 'ANTONIO DOS SANTOS', '4', 'PROGRAMADOR', 1155.00, '2006-10-10', 1);
INSERT INTO cadfun VALUES (
    21, 'EPAMINONDAS DA SILVA', '4', 'PROGRAMADOR', 1155.00, '2006-10-10', 2);
```

Além disto, os registros cujo campo Filhos estejam com valor NULL devem ser zerados e o campo Função deve ser escrito da mesma forma (alguns registros têm escrito Vendedora e Supervisora, enquanto outros têm escrito Vendedor e Supervisor). Digite os comandos para homogeneizar estes registros e depois verifique se algum ainda apresenta o campo Filhos com valores nulos ou Função escrita de forma diferente.

```
UPDATE cadfun SET FILHOS = 0 WHERE CODFUN = 4;
UPDATE cadfun SET FILHOS = 0 WHERE CODFUN = 7;
UPDATE cadfun SET FILHOS = 0 WHERE CODFUN = 15;
UPDATE cadfun SET FILHOS = 0 WHERE CODFUN = 22;
UPDATE cadfun SET FILHOS = 0 WHERE CODFUN = 24;
UPDATE cadfun SET FUNCAO = 'SUPERVISOR'
    WHERE FUNCAO = 'SUPERVISORA';
UPDATE cadfun SET FUNCAO = 'VENDEDOR'
```

```
    WHERE FUNCAO = 'VENDEDORA';
SELECT * FROM cadfun;
```

1. Agrupamentos

Para o agrupamento, usa-se a cláusula GROUP BY dentro da instrução SELECT. Digite os exemplos abaixo, comparando os resultados individuais com os resultados agregados obtidos. Observe no exemplo final que foi introduzida a cláusula HAVING, que permite selecionar valores em campos agregados; é, assim, diferente da cláusula WHERE, que permite a seleção de valores em colunas não agregadas.

```
SELECT DEPTO FROM cadfun;
    SELECT DISTINCT DEPTO FROM cadfun;
    SELECT DEPTO, COUNT(*) FROM cadfun GROUP BY DEPTO;
    SELECT DEPTO, SUM(SALARIO) FROM cadfun GROUP BY DEPTO;
    SELECT DEPTO, SUM(SALARIO) FROM cadfun GROUP BY DEPTO DESC;
    SELECT DEPTO, COUNT(*) FROM cadfun GROUP BY DEPTO
      ORDER BY DEPTO DESC;
SELECT FUNCAO FROM cadfun;
    SELECT DISTINCT FUNCAO FROM cadfun;
    SELECT FUNCAO, COUNT(*) FROM cadfun GROUP BY FUNCAO;
    SELECT FUNCAO, COUNT(*), SUM(SALARIO)
       FROM cadfun GROUP BY FUNCAO;
SELECT DEPTO, SUM(SALARIO) FROM cadfun
    WHERE DEPTO IN ('2', '4')
    GROUP BY DEPTO;
SELECT DEPTO, SALARIO FROM cadfun;
SELECT DEPTO, COUNT(*), SUM(SALARIO) FROM cadfun GROUP BY DEPTO;
SELECT DEPTO, AVG(SALARIO) FROM cadfun GROUP BY DEPTO;
SELECT DEPTO, MAX(SALARIO) FROM cadfun GROUP BY DEPTO;
SELECT DEPTO, MIN(SALARIO) FROM cadfun GROUP BY DEPTO;
SELECT DEPTO, SUM(SALARIO) FROM cadfun
     GROUP BY DEPTO
     HAVING SUM(SALARIO) > 8000;
```

2. Uniões

Uma união baseia-se em múltiplos comandos Select, de uma forma parecida com uma subconsulta, na forma geral:

```
SELECT   ...   UNION [ALL | DISTINCT]
SELECT   ...   [UNION [ALL | DISTINCT]
SELECT   ...   ];   onde
```

ALL | DISTINCT: representam todos os registros, ou apenas os registros distintos (eliminando duplicatas) – neste caso, DISTINCT é o default.

Antes de executar as uniões, vamos atualizar a tabela CadFun, imaginando que a empresa fez oito demissões. Neste caso, temos que inserir os demitidos na tabela Morto antes de deletá-los da CadFun. Observe ainda que as inserções em Morto estão sendo feitas a partir de uma consulta, o que também é um recurso interessante para evitar linhas de código. Lembre-se que estamos utilizando a tabela Morto porque as colunas das consultas utilizadas devem ser do mesmo tipo. Para isto, digite o seguinte:

```
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERE CODFUN = 7;
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERE CODFUN = 3;
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERE CODFUN = 9;
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERE CODFUN = 25;
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERE CODFUN = 4;
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERe CODFUN = 2;
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERE CODFUN = 5;
INSERT INTO morto
    SELECT CODFUN, NOME, DEPTO, FUNCAO, SALARIO, ADMISSAO, FILHOS
    FROM cadfun
    WHERE CODFUN = 15;
DELETE FROM cadfun WHERE CODFUN = 7;
DELETE FROM cadfun WHERE CODFUN = 3;
DELETE FROM cadfun WHERE CODFUN = 9;
DELETE FROM cadfun WHERE CODFUN = 25;
DELETE FROM cadfun WHERE CODFUN = 4;
DELETE FROM cadfun WHERE CODFUN = 2;
DELETE FROM cadfun WHERE CODFUN = 5;
DELETE FROM cadfun WHERE CODFUN = 15;
```

Confira, antes da realização da consulta Union, os registros existentes em cada uma das tabelas individualmente e depois compare com o resultado da união. Para isto, digite o seguinte:

```
SELECT * FROM cadfun;
SELECT * FROM morto;
SELECT * FROM cadfun UNION SELECT * FROM morto;
```

Para exemplificar o ALL | DISTINCT, vamos inserir um registro no Morto sem excluí-lo do CadFun. Em seguida, compare o conteúdo das duas tabelas. Digite:

```
INSERT INTO morto VALUES (
    21, 'EPAMINONDAS DA SILVA', '4','PROGRAMADOR', 1155.00, '2006-10-10', 2);
SELECT * FROM cadfun;
SELECT * FROM morto;
```

Agora, execute as uniões com o ALL | DISTINCT; lembre-se que, neste caso, o default é DISTINCT. Digite:

```
SELECT * FROM cadfun UNION SELECT * FROM morto;
SELECT * FROM cadfun UNION ALL SELECT * FROM morto;
```

Uma consulta Union pode também ser ordenada, mas, para que a ordenação funcione é necessário usar parênteses. Digite o exemplo abaixo:

```
(SELECT * FROM cadfun ORDER BY NOME)
    UNION
    (SELECT * FROM morto ORDER BY NOME)
    ORDER BY NOME;
```

.

LIÇÃO 7 – JUNÇÕES E VISÕES (JOIN e VIEW)

As junções (JOIN) são outro recurso que nos permite juntar várias tabelas em uma única consulta, além das subconsultas e uniões, vistas anteriormente. As visões (VIEW) podem ser encaradas como consultas permanentes ou como tabelas virtuais mantidas automaticamente pelo SGBD.

Até agora, trabalhamos com duas tabelas em nosso BD Virtual: CadFun e Morto. Embora estas tabelas tenham a mesma estrutura e aparentemente tenham um relacionamento 1:1, na realidade não têm relacionamento, pois um funcionário existente na tabela CadFun não existe na tabela Morto e vice-versa. Para criar junções precisamos de tabelas relacionadas. Para isto, vamos criar duas novas tabelas em nosso BD Virtual: Cliente e Venda. A tabela Cliente conterá o cadastro dos clientes e a tabela Venda (o nome não é muito apropriado) conterá o valor das duplicatas das vendas a prazo. Estas tabelas têm a seguinte estrutura:

tabela CLIENTE	
campo	*tipo**
CODCLI	char(3)**
NOME	char(40)
ENDERECO	char(50)
CIDADE	varchar(20)
ESTADO	char(2)
CEP	char(9)

tabela VENDA		
campo	*tipo**	*descrição*
DUPLIC	char(6)**	número da duplicata
VALOR	decimal(10, 2)	
VENCTO	date	
CODCLI	char(3)	

* todos os campos da tabelas são Not Null,

** chave primária.

Crie a estrutura da tabela Cliente e depois insira os seguintes registros:

```
INSERT INTO cliente VALUES (
    '250', 'BANCO BARCA S/A', 'R. VITO, 34', 'SAO SEBASTIAO',
    'CE', '62380-000');
INSERT INTO cliente VALUES (
    '820', 'MECANICA SAO PAULO', 'R. DO MACUCO, 99', 'SANTO ANTONIO',
    'ES', '29810-020');
INSERT INTO cliente VALUES (
    '170', 'POSTO BRASIL LTDA.', 'AV. IMPERIO, 85', 'GUAGIRUS',
```

```
    'BA', '42837-000');
INSERT INTO cliente VALUES (
    '340', 'TRANSP. CARGA PESADA LTDA.', 'AV. DOS AUTONOMISTAS, 1000',
    'OSASCO', 'SP', '06020-010');
INSERT INTO cliente VALUES (
    '100', 'MICROS INFORMATICA S/A', 'R. DAS PALMEIRAS, 4 - LOJA 2',
    'SAO PAULO', 'SP', '01226-010');
INSERT INTO cliente VALUES (
    '750', 'IMOBILIARIA PAVAO', 'AV. BRASIL, 105', 'RIO DO PRADO',
    'MG', '39940-111');
INSERT INTO cliente VALUES (
    '860', 'ASSIS CONTABIL S/C LTDA.', 'R. DO MONUMENTO, 550',
    'SANTO AMARO', 'BA', '44200-090');
INSERT INTO cliente VALUES (
    '230', 'SUPERMERCADO BOTAFOGO', 'R. DA LAGOA, 999', 'RIO DAS OSTRAS',
    'RJ', '28890-540');
INSERT INTO cliente VALUES (
    '150', 'MARCONDES E IRMAO LTDA.', 'R. DO ORATORIO, 66',
    'ROSARIO OESTE', 'MT', '78470-010');
INSERT INTO cliente VALUES (
    '800', 'DOCEIRA PAO DE ACUCAR', 'R. MARTINS PENA, 200', 'SANTO ANDRE',
    'SP', '09190-700');
INSERT INTO cliente VALUES (
    '990', 'METALURGICA FOGO E FERRO', 'R. ARTUR BERNARDES, 3500',
    'SANTO ANDRE', 'SP', '09193-410');
INSERT INTO cliente VALUES (
    '500', 'JOSE DANTAS E FILHOS S/C LTDA.', 'AV. IPIRANGA, 1000', 'LUCRECIA',
    'RN', '59805-010');
INSERT INTO cliente VALUES (
    '300', 'PCTEC - MICROCOMPUTADORES S/A.',
    'R. CAIS DO PORTO, ARMAZEM 3', 'SANTOS', 'SP', '11000-005');
INSERT INTO cliente VALUES (
    '550', 'ROTRAM COMERCIAL LTDA.', 'R. DAS PALMEIRAS, 45 - CJ 10',
    'SAO PAULO', 'SP', '01226-010');
SELECT * FROM cliente;
```

Para evitar o visual confuso, você pode alterar as propriedades da janela; aumente sua largura para 115. Em seguida, crie a estrutura da tabela Venda mostrada anteriormente e insira os seus registros.

```
INSERT INTO venda VALUES ( '230001', 1300.00, '2001-06-10', '340');
INSERT INTO venda VALUES ( '230099', 1000.00, '2002-02-10', '820');
INSERT INTO venda VALUES ( '997818', 3000.00, '2001-11-11', '170');
INSERT INTO venda VALUES ( '202550', 9518.55, '2002-11-21', '750');
INSERT INTO venda VALUES ( '113340', 2002.00, '2001-11-25', '230');
INSERT INTO venda VALUES ( '900450', 1200.00, '2002-09-09', '340');
INSERT INTO venda VALUES ( '202020', 2390.00, '2001-11-11', '550');
INSERT INTO venda VALUES ( '100200', 3500.00, '2002-10-11', '230');
INSERT INTO venda VALUES ( '345611', 5090.67, '2002-12-12', '550');
INSERT INTO venda VALUES ( '900855', 2356.00, '2002-10-10', '340');
```

```
SELECT * FROM venda;
```

Para que os exemplos das junções retornem uma quantidade de registros razoável, devem ser incluídos mais os registros seguintes.

100000	5060.88	1999-05-15	300
100334	2002.50	2005-12-20	990
100335	2002.50	2006-01-20	990
100336	2002.50	2006-02-20	990
101010	5060.88	2000-03-15	300
101110	3460.66	2000-04-15	300
101210	9060.55	2001-10-15	300
111999	1250.00	2005-04-15	150
112099	1000.00	2005-06-22	150
112199	1350.00	2005-08-30	150
112299	1000.00	2005-10-15	150
121212	1300.00	2002-01-15	170
121213	1000.00	2002-02-15	170
121214	1400.00	2002-03-15	170
121515	1000.00	2003-10-15	230
121616	1600.00	2003-11-15	230
155099	1800.00	2004-11-13	250
155199	1000.00	2003-10-15	250
155299	1900.00	2003-11-15	250
155999	1000.00	2004-10-14	250
210001	6000.00	1999-04-12	100
230001	1300.00	2005-09-10	550
231015	1800.00	2005-10-10	550
250013	2002.00	2002-03-10	820
400400	6000.00	2003-03-15	340
451300	1750.00	1999-05-22	800
460460	1902.50	2004-02-20	100
460560	1902.50	2004-03-22	100
460660	1902.50	2004-04-24	100
500005	5060.88	1999-05-15	990
600099	1300.00	2001-10-13	860
600199	1400.00	2002-11-15	860
600299	1500.00	2002-12-05	860
600999	1200.00	2001-09-14	860
653099	1350.88	2003-09-13	750
653199	1432.77	2004-08-15	750
653999	1255.99	2003-09-14	750
750299	1560.00	2004-12-03	800
750299	1570.00	2005-12-05	800
750300	1565.00	2004-11-05	800
752252	1675.00	2006-12-15	820
752252	1675.00	2006-11-22	820
950888	2002.50	2006-07-10	500
950889	2002.50	2006-08-10	500
950890	2002.50	2006-09-10	500
950891	2002.50	2006-10-10	500

A inserção também pode ser feita usando os comandos a seguir.

```
INSERT INTO venda VALUES (‘100000’, 5060.88, ‘1999-05-15’, ‘300’);
INSERT INTO venda VALUES (‘100334’, 2002.50, ‘2005-12-20’, ‘990’);
INSERT INTO venda VALUES (‘100335’, 2002.50, ‘2006-01-20’, ‘990’);
INSERT INTO venda VALUES (‘100336’, 2002.50, ‘2006-02-20’, ‘990’);
INSERT INTO venda VALUES (‘101010’, 5060.88, ‘2000-03-15’, ‘300’);
INSERT INTO venda VALUES (‘101110’, 3460.66, ‘2000-04-15’, ‘300’);
INSERT INTO venda VALUES (‘101210’, 9060.55, ‘2001-10-15’, ‘300’);
INSERT INTO venda VALUES (‘111999’, 1250.00, ‘2005-04-15’, ‘150’);
INSERT INTO venda VALUES (‘112099’, 1000.00, ‘2005-06-22’, ‘150’);
INSERT INTO venda VALUES (‘112199’, 1350.00, ‘2005-08-30’, ‘150’);
INSERT INTO venda VALUES (‘112299’, 1000.00, ‘2005-10-15’, ‘150’);
INSERT INTO venda VALUES (‘121212’, 1300.00, ‘2002-01-15’, ‘170’);
INSERT INTO venda VALUES (‘121213’, 1000.00, ‘2002-02-15’, ‘170’);
INSERT INTO venda VALUES (‘121214’, 1400.00, ‘2002-03-15’, ‘170’);
INSERT INTO venda VALUES (‘121515’, 1000.00, ‘2003-10-15’, ‘230’);
INSERT INTO venda VALUES (‘121616’, 1600.00, ‘2003-11-15’, ‘230’);
INSERT INTO venda VALUES (‘155099’, 1800.00, ‘2004-11-13’, ‘250’);
INSERT INTO venda VALUES (‘155199’, 1000.00, ‘2003-10-15’, ‘250’);
INSERT INTO venda VALUES (‘155299’, 1900.00, ‘2003-11-15’, ‘250’);
INSERT INTO venda VALUES (‘155999’, 1000.00, ‘2004-10-14’, ‘250’);
INSERT INTO venda VALUES (‘210001’, 6000.00, ‘1999-04-12’, ‘100’);
INSERT INTO venda VALUES (‘230001’, 1300.00, ‘2005-09-10’, ‘550’);
INSERT INTO venda VALUES (‘231015’, 1800.00, ‘2005-10-10’, ‘550’);
INSERT INTO venda VALUES (‘250013’, 2002.00, ‘2002-03-10’, ‘820’);
INSERT INTO venda VALUES (‘400400’, 6000.00, ‘2003-03-15’, ‘340’);
INSERT INTO venda VALUES (‘451300’, 1750.00, ‘1999-05-22’, ‘800’);
INSERT INTO venda VALUES (‘460460’, 1902.50, ‘2004-02-20’, ‘100’);
INSERT INTO venda VALUES (‘460560’, 1902.50, ‘2004-03-22’, ‘100’);
INSERT INTO venda VALUES (‘460660’, 1902.50, ‘2004-04-24’, ‘100’);
INSERT INTO venda VALUES (‘500005’, 5060.88, ‘1999-05-15’, ‘990’);
INSERT INTO venda VALUES (‘600099’, 1300.00, ‘2001-10-13’, ‘860’);
INSERT INTO venda VALUES (‘600199’, 1400.00, ‘2002-11-15’, ‘860’);
INSERT INTO venda VALUES (‘600299’, 1500.00, ‘2002-12-05’, ‘860’);
INSERT INTO venda VALUES (‘600999’, 1200.00, ‘2001-09-14’, ‘860’);
INSERT INTO venda VALUES (‘653099’, 1350.88, ‘2003-09-13’, ‘750’);
INSERT INTO venda VALUES (‘653199’, 1432.77, ‘2004-08-15’, ‘750’);
INSERT INTO venda VALUES (‘653999’, 1255.99, ‘2003-09-14’, ‘750’);
INSERT INTO venda VALUES (‘750299’, 1560.00, ‘2004-12-03’, ‘800’);
INSERT INTO venda VALUES (‘750299’, 1570.00, ‘2005-12-05’, ‘800’);
INSERT INTO venda VALUES (‘750300’, 1565.00, ‘2004-11-05’, ‘800’);
INSERT INTO venda VALUES (‘752252’, 1675.00, ‘2006-12-15’, ‘820’);
INSERT INTO venda VALUES (‘752252’, 1675.00, ‘2006-11-22’, ‘820’);
INSERT INTO venda VALUES (‘950888’, 2002.50, ‘2006-07-10’, ‘500’);
INSERT INTO venda VALUES (‘950889’, 2002.50, ‘2006-08-10’, ‘500’);
INSERT INTO venda VALUES (‘950890’, 2002.50, ‘2006-09-10’, ‘500’);
INSERT INTO venda VALUES (‘950891’, 2002.50, ‘2006-10-10’, ‘500’);
```

1. Junções

Para permitir as junções, criamos a chave estrangeira CODCLI na tabela Venda que criará o relacionamento com a tabela Cliente. Este relacionamento tem de ser explicitado na cláusula WHERE. As outras condições da cláusula WHERE já foram vistos em exemplos anteriores. Observe que, além do nome do campo, indicamos também o nome da tabela. Vamos aos primeiros exemplos. Digite:

```
SELECT * FROM cliente;
SELECT * FROM venda;
SELECT venda.DUPLIC, cliente.NOME FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI;
SELECT cliente.NOME, venda.DUPLIC, venda.VALOR    FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI
    ORDER BY cliente.NOME, DUPLIC;
SELECT cliente.NOME, venda.DUPLIC, venda.VALOR FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI AND cliente.NOME LIKE 'PCTEC%';
SELECT cliente.NOME, venda.VENCTO FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI AND venda.VENCTO LIKE '2002-11%'
    ORDER BY venda.VENCTO;
SELECT cliente.NOME, venda.VENCTO FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI AND venda.VENCTO LIKE '____-10%'
    ORDER BY venda.VENCTO;
```

No último exemplo o caractere sublinhado _ foi usado quatro vezes para substituir qualquer ano.

2. Junções com agregações e alias

As junções também ser agrupardas usando-se GROUP BY. O uso de alias para nomes de campos já vimos em exemplos anteriores. Digite os seguintes exemplos:

```
SELECT cliente.NOME, COUNT(*) FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI
    GROUP BY cliente.NOME;
```

Para as próxias junções insira mais alguns registros:

```
INSERT INTO venda VALUES ( '235100', 1500.00, '2005-06-12', '500');
INSERT INTO venda VALUES ( '999820', 3110.22, '2005-05-11', '170');
INSERT INTO venda VALUES ( '203052', 9008.33, '2005-08-22', '550');
INSERT INTO venda VALUES ( '223345', 2112.11, '2006-08-03', '230');
INSERT INTO venda VALUES ( '922452', 1211.98, '2006-09-03', '340');
INSERT INTO venda VALUES ( '222228', 2390.00, '2006-10-07', '170');
INSERT INTO venda VALUES ( '111211', 3535.00, '2006-11-15', '230');
INSERT INTO venda VALUES ( '347711', 5092.55, '2006-11-20', '170');
INSERT INTO venda VALUES ( '907754', 2056.90, '2005-10-30', '340');
```

Os exemplos seguintes mostram o uso do recurso do alias para colocar nomes mais explicativos nas colunas da consulta, tanto para valores de totais como para campos calculados. Digite:

```
SELECT cliente.CODCLI, cliente.NOME, COUNT(*), SUM(venda.VALOR)
    FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI
    GROUP BY cliente.NOME;
```

```
SELECT cliente.CODCLI, cliente.NOME,
    COUNT(*) AS TITULOS, SUM(venda.VALOR) AS TOTAL
    FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI
    GROUP BY cliente.NOME;
SELECT cliente.NOME AS CLIENTE, COUNT(*) AS VENCIDOS
    FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI  AND VENCTO <= '2003-12-31'
    GROUP BY cliente.NOME
    ORDER BY cliente.NOME;
SELECT cliente.NOME, venda.VALOR, venda.VALOR * 0.10 AS JUROS,
    venda.VALOR * 1.10 AS TOTAL
    FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI AND VENCTO <= '1999-12-31'
    ORDER BY cliente.NOME;
```

3. Visões

As visões VIEW têm duas aplicações práticas muito importantes:

- consultas que são feitas com muita frequência: estas consultas podem ser feitas em cima das visões e não em cima das tabelas, o que garante menor tempo de resposta;
- segurança: não permitir que alguns usuários (ou programas) tenham acesso às tabelas, mas somente a algumas colunas destas tabelas.

As visões são criadas por uma instrução CREATE como as tabelas e por uma SELECT, como as consultas; assim, são consultas ou tabelas virtuais em cima das quais se executam outras consultas. A forma geral de crição de uma VIEW é:

```
CREATE VIEW <nomeDaVisão> AS <consulta: SELECT ...>
```

O primeiro exemplo cria a Visao1 com algumas das colunas da tabela CadFun; o SELECT a seguir mostra todos os dados retornados pela Visao1. O exemplo seguinte usa o recurso junção para criar a Visao2 e mostrar seus registros.

```
CREATE VIEW visao1 AS SELECT NOME, DEPTO, SALARIO FROM cadfun;
SELECT * FROM visao1;
CREATE VIEW visao2 AS SELECT cliente.NOME AS CLIENTE,
    COUNT(*) AS VENCIDOS FROM cliente, venda
    WHERE cliente.CODCLI = venda.CODCLI AND VENCTO <= '2005-12-31'
    GROUP BY cliente.NOME ORDER BY cliente.NOME;
SELECT * FROM visao2;
```

O exemplo a seguir cria a Visao3 e executa um SELECT apenas de algumas de suas colunas, como se forsse realmente uma tabela. Em seguida é criada ainda a Visao4 a partir da Visao3. Digite:

```
CREATE VIEW visao3 AS
   SELECT cliente.CODCLI, cliente.NOME,  cliente.ENDERECO, cliente.CIDADE,
        cliente.ESTADO, cliente.CEP,  venda.DUPLIC, venda.VALOR, venda.VENCTO
     FROM cliente, venda WHERE cliente.CODCLI = venda.CODCLI
     ORDER BY cliente.NOME;
SELECT * FROM visao3;
SELECT CODCLI, NOME, DUPLIC, VALOR, VENCTO FROM visao3
    WHERE CODCLI IN ('100', '500', '990');
CREATE VIEW visao4 AS
  SELECT CODCLI, NOME, DUPLIC, VALOR, VENCTO
```

FROM visao3 ;

SELECT * FROM visao4;

Já vimos que podemos inserir dados em uma tabela a partir de uma consulta. Neste exemplo vamos ver um outro recurso interessante: poder criar uma tabela física a partir de uma visão, o que pode facilitar a exportação de dados para outros ambientes. Observe que o comando SHOW TABLES também trata uma visão como se fosse uma tabela. Digite:

CREATE TABLE dadoscli AS SELECT * FROM visao4;

SHOW TABLES;

SELECT * FROM visao4;

SELECT * FROM dadoscli;

Para eliminar uma visão usamos o DROP VIEW, de forma similar à que usams para apagar tabelas (DROP TABLE). Digite:

DROP VIEW visao1;

DROP VIEW visao2;

DROP VIEW visao3;

DROP TABLE dadoscli;

SHOW TABLES;

LIÇÃO 8 – CHAVES E GERENCIAMENTO DE USUÁRIOS

1. Manuseio de índices

Já vimos no comando CREATE TABLE a criação de chaves primárias. Chaves secundárias também podem ser criadas a qualquer tempo e, assim como as primárias, podem ser:

- únicas: não permitem duplicação do índice,
- compostas: abrangem mais de um campo.

Os comandos create e drop permite a criação e eliminação de índices:

CREATE [UNIQUE] INDEX <índice> ON <tabela (campos)>

DROP INDEX <índice> ON <tabela>, onde:

- UNIQUE: indica que o índice não permite valores duplicados no índice,
- <índice>: nome do índice: toda chave secundária tem um nome,
- tabela: nome da tabela onde está sendo criado o índice,
- (campos): lista dos campos que montarão o índice.

Vamos criar um índice para o campo Nome e depois criar um índice exclusivo para um campo de CPF. Para isto, vamos alterar a estrutura da tabela CadFun para inserir o campo CPF e atribuir valores para este campo criado. Digite:

DESCRIBE cadfun;

SHOW INDEX FROM cadfun;

CREATE INDEX indice1 ON cadfun (NOME);

DESCRIBE cadfun;

SHOW INDEX FROM cadfun;

ALTER TABLE cadfun ADD CPF CHAR(11);

SELECT * FROM cadfun;

```
CREATE UNIQUE INDEX indice2 ON cadfun (CPF);
DESCRIBE cadfun;
SHOW INDEX FROM cadfun;
UPDATE cadfun SET CPF = '10020011199' WHERE CODFUN = 1;
UPDATE cadfun SET CPF = '10020022299' WHERE CODFUN = 6;
UPDATE cadfun SET CPF = '10020033399' WHERE CODFUN = 8;
UPDATE cadfun SET CPF = '10020044499' WHERE CODFUN = 10;
UPDATE cadfun SET CPF = '10020055599' WHERE CODFUN = 11;
UPDATE cadfun SET CPF = '10020066699' WHERE CODFUN = 12;
UPDATE cadfun SET CPF = '10020077799' WHERE CODFUN = 20;
UPDATE cadfun SET CPF = '10020088899' WHERE CODFUN = 21;
UPDATE cadfun SET CPF = '10022211199' WHERE CODFUN = 22;
UPDATE cadfun SET CPF = '10022233399' WHERE CODFUN = 24;
SELECT * FROM cadfun;
```

Verifique, no select anterior, se ficou algum campo com CPF = null. Caso isto tenha acontecido, altere estes registros de forma tal que todos tenham CPF válido. A seguir, vamos testar a inserção de registros com uma chave primária duplicada e um com a chave secundária duplicada (ambos, primária e secundária, são índices UNIQUE) e de um registro sem duplicação. Observe as mensagens de erro ou de aceitação que o MySql devolve. Digite:

```
INSERT INTO cadfun VALUES (
    24, 'MARCOS INACIO', '2', 'GERENTE', 2184.33, '2006-09-25', 2, '11122233344');
INSERT INTO cadfun VALUES (
    50, 'MARCOS INACIO', '2', 'GERENTE', 2184.33, '2006-09-25',  2,  '11122233344');
INSERT INTO cadfun VALUES (
    55, 'MARIA JOSE',  '3',  'GERENTE',  2184.33,  '2006-10-25',  2,  '11122233344');
SELECT * FROM cadfun;
```

Nos próximos comandos vamos criar um índice secundário composto. Como a tabela Morto deve ter a mesma estrutura da CadFun, vamos acrescentar o CPF e criar seu índice2. Digite:

```
CREATE INDEX indice3 ON cadfun (DEPTO, FUNCAO);
DESCRIBE cadfun;
SHOW INDEX FROM cadfun;
ALTER TABLE morto ADD CPF CHAR(11);
CREATE UNIQUE INDEX indice2 ON morto (CPF);
SELECT * FROM cadfun;
SELECT * FROM morto;
DESCRIBE morto;
SHOW INDEX FROM morto;
DROP INDEX indice3 ON cadfun;
DESCRIBE cadfun;
SHOW INDEX FROM cadfun;
```

2. Gerenciamento dos direitos dos usuários

O usuário root, com o qual estamos trabalhando, tem todos os privilégios. Vamos aprender a definir privilégios de acesso a outros usuários (pois nem todos os usuáros podem ter direito de excluir uma tabela ou um BD, por exemplo). O MySql permite tratar estes privilégios em quatro níveis:

- global ou de servidor: garante o acesso a este usuário a todos os BDs de forma irrestrita em um determinado servidor. As informações dos privilégios deste nível ficam armazenados na tabela mysql.user.
- banco de dados: garante o acesso a este usuário a todas as tabelas de um determinado BD. As informações dos privilégios deste nível ficam armazenados na tabela mysql.db e mysql.host.
- tabelas: garante o acesso a este usuário apenas a uma determinada tabela de um BD. As informações dos privilégios deste nível ficam armazenados na tabela mysql.tables_priv.
- colunas: garante o acesso a este usuário apenas a uma determinada coluna de uma tabela de um BD. As informações dos privilégios deste nível ficam armazenados na tabela mysql.columns_priv.

Os comandos grant e revoke permitem ao DBA a administração destes direitos dos usuários. Grant pode criar usuários e lhes atribuir direitos; revoke retira direitos, mas não exclui usuários.

GRANT <privilégios> [colunas] ON <*.* | db.* | db.tabela>
TO <usuários> [IDENTIFIED BY [PASSWORD 'senha']]
[WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR valor |
MAX_UPDATES_PER_HOUR valor |
MAX_CONNECTIONS_PER_HOUR valor]];

REVOKE <privilégios> [colunas] ON <* | *.* | db.tabela>
FROM <usuários>; onde:

- <privilégios>: pode ser uma lista de direitos garantidos a este usuários. A tabela com a lista destes direitos está nos anexos;
- colunas: quais são as colunas da tabela que estão sendo liberadas;
- *.*: todos os BDs;
- db.*: todas as tabelas de um BD chamado bd;
- db.tabela: uma tabela chamada tabela de um BD chamado bd;
- <usuários>: uma lista (ou um) de usuários aos quais estão sendo atribuídos os direitos;
- GRANT OPTION: atribui o direito de aquele usuário passar adiante os seus próprios direitos;
- a cláusula WITH permite ainda fixar o número máximo de queries, de updates e de conexões por hora para aquele usuário.

Além disto, o DBA pode fazer comandos Insert, Update e Delete diretamente nas tabelas mysql.user, mysql.db, mysql.host, mysql.tables_priv e mysql.columns_priv. O comando delete, de deleção normal, é usado diretamente nas tabelas de controle mantidas pelo MySql para eliminar um usuário. O comando Flush privileges lê estas tabelas de controle e refresca a lista de direitos dos usuários na memória:

DELETE FROM mysql.user WHERE user = <usuário> AND host = <localhost>;

FLUSH PRIVILEGES; onde:

- mysql.user: é a tabela interna de controle de usuários mantida pelo mySql;
- user: definição do nome do usuário;
- host: nome do local onde o usuário está cadastrado.

Exemplos:

```
GRANT ALL PRIVILEGES ON *.* TO usuariol ©localhost
    IDENTIFIED BY 'abc123';
```

Cria o usuário1 no servidor localhost, com senha de acesso definida como abc123, com todos os privilégios (ALL PRIVILEGES) a todos os BDs e tabelas.

```
GRANT ALL PRIVILEGES ON *.* TO usuariol@localhost
    IDENTIFIED BY 'abcl23' WITH GRANT OPTION;
```

Idem, com a possibilidade de conceder seus privilégios a outros usuários.

```
GRANT ALL PRIVILEGES ON *.* TO usuario2@'%'
    IDENTIFIED BY 'abc123' WITH GRANT OPTION;
```

Cria o usuario2 no servidor localhost com senha de acesso = abc123 com todos os privilégios. Neste exemplo é omitido o nome do host e em seu lugar é usado o símbolo % (percentagem), um curinga que indica a criação do usuário específico em qualquer máquina do domínio host.

```
GRANT SELECT (NOME, SALÁRIO) ON virtual.cadfun TO usuario3@localhost
    IDENTIFIED BY 'abcl23';
```

Cria o usuario3 no servidor localhost com senha de acesso = abc123 e informa que esse usuário pode apenas acessar as colunas NOME e SALÁRIO da tabela cadfun do BD virtual.

```
GRANT UPDATE, INSERT ON virtual.* TO usuario4@localhost
    IDENTIFIED BY 'abcl23';
```

Cria o usuario4 no servidor localhost com senha de acesso = abc123 e informa que esse usuário pode executar os comandos UPDATE e INSERT em qualquer tabela do BD virtual.

Para verificar as tabelas do MySql que controlam estes direitos (user, db, host, tables_priv e columns_priv), vamos olhar o conteúdo do bd de controle do MySql. Digite:

```
SHOW DATABASES;
USE mySql;
SHOW TABLES;
```

A seguir, vamos ver a estrutura e os registros destas tabelas. Observe a lista dos direitos mostrados (comando DESCRIBE) pela estrutura da tabela e quais usuários estão cadastrados e seus respectivos direitos (comando SELECT). Observe e compare as chaves primárias. O único usuário cadastrado até o momento é o root. As tabelas tables_priv e columns_priv não mostram nenhum registro, pois até o momento nenhum usuário teve seus direitos restringidos. Digite:

```
DESCRIBE user;
SELECT * FROM user;
DESCRIBE host;
SELECT * FROM host;
DESCRIBE db;
SELECT * FROM db;
DESCRIBE tables_priv;
SELECT * FROM tables_priv;
DESCRIBE columns_priv;
SELECT * FROM columns_priv;
```

Volte ao bd virtual e faça a criação dos quatro usuários (1, 2, 3, 4) cujos exemplos mostramos anteriormente. Digite:

```
USE virtual;
GRANT ALL PRIVILEGES ON *.* TO usuariol@localhost
    IDENTIFIED BY 'abcl23' WITH GRANT OPTION;
```

```
GRANT ALL PRIVILEGES ON *.* TO usuario2@'%'
    IDENTIFIED BY 'abc123' WITH GRANT OPTION;
GRANT SELECT (NOME, SALÁRIO) ON virtual.cadfun TO usuario3@localhost
    IDENTIFIED BY 'abcl23';
GRANT UPDATE, INSERT ON virtual.* TO usuario4@localhost
    IDENTIFIED BY 'abcl23';
```

Volte ao bd mySql e examine novamente o conteúdo das tabelas de controle. Digite:

```
USE mySql;
SELECT * FROM user;
SELECT * FROM host;
SELECT * FROM db;
SELECT * FROM tables_priv;
SELECT * FROM columns_priv;
```

3. Teste de uso como outro usuário

Vamos agora fazer uns testes logando como outros usuários para testar os direitos atribuídos. Saia do mysql, *digite exit;*. A seguir, vamos nos logar como o usuário1 e verificar o usuário conectado. Digite:

```
mySql --user=usuario1 --password=abc123
SELECT USER();
```

Como o usuario1 possui todos os privilégios do root, vamos testar agora o usuario3, que não tem permissão total sobre a tabela CadFun. Saia do mysql, e faça novo login. Digite:

```
exit;
mySql --user=usuario3 --password=abc123
SELECT USER();
USE virtual;
SELECT * FROM cadfun;
```

O acesso total deve ter sido negado. Digite mais:

```
SELECT NOME, SALARIO FROM cadfun;
SELECT * FROM morto;
```

Esta forma de acesso do usuário deixa sua senha exposta, tanto na hora do login como posteriormente na barra de título da janela. Para evitar este inconveniente, há uma outra forma de login que oculta a password. Saia do mysql e faça novo login. Digite:

```
exit;
mySql -u usuario3 -p
```

Digite *abc123* na password solicitada. Aproveite e faça outros testes de direitos com o usuario3. Saia e faça login como o usuario4, cujos privilégios são outros. Saia e teste os privilégios do usuario2, que deve poder fazer tudo.

Uma vez testados os privilégios, vamos alterá-los. Vamos coibir o direito de update para o usuário4 e depois restaurá-lo, examinando as tabelas do bd mySql. Saia e faça login como o usuário root. Digite:

```
exit;
mySql -u root
REVOKE UPDATE ON virtual.cadfun FROM usuario4@localhost;
```

```
USE mySql;
SELECT * FROM user;
SELECT * FROM host;
SELECT * FROM db;
SELECT * FROM tables_priv;
SELECT * FROM columns_priv;
GRANT UPDATE, INSERT, SELECT ON virtual.*
      TO usuario4@localhost IDENTIFIED BY 'abc123';
SELECT * FROM user;
SELECT * FROM host;
SELECT * FROM db;
SELECT * FROM tables_priv;
SELECT * FROM columns_priv;
```

Vamos agora deletar dois usuários que não usamos, alterar a senha do usuario3, atualizar o privilégios na memória do servidor e exibi-los. Digite:

```
DELETE FROM mysql.user WHERE user='usuario1' AND host='localhost';
FLUSH PRIVILEGES;
DELETE FROM mysql.user WHERE user='usuario2' AND host='localhost';
FLUSH PRIVILEGES;
UPDATE mysql.user SET PASSWORD = PASSWORD('123456')
    WHERE user='usuario3';
FLUSH PRIVILEGES;
SHOW GRANTS FOR root@localhost;
```

4. Chaves estrangeiras e relacionamentos

Já vimos a criação e eliminação de chaves primárias e secundárias, permitindo ou não valores duplicados. Vamos agora estudar os relacionamentos e a criação de chaves estrangeiras. O bd virtual, que vimos estudando, tem as tabelas CadFun e Morto. Embora pareça que têm um relacionamento 1:1, pois têm a mesma estrtura, estas duas tabelas não têm relacionamento, pois um registro da tabela CadFun não tem correspondente na Morto e vice-versa. Vamos criar um novo bd Escola, com duas tabelas CadPro (professores) e CadDis (disciplinas), que permite criar os relacionamentos.

CadPro: cadastro dos professores:

campo	*tipo*	*descrição*
CODPROP	integer(3)	código do professor, chave primária, não nulo
NOMEPRO	varchar(40)	nome, não nulo
CPF	char(11)	não nulo, chave candidata (unique)

CadDis: cadastro das disciplinas:

campo	*tipo*	*descrição*
CODDIS	char(6)	código da disciplina, chave primária, não nulo
NOMEDIS	varchar(40)	nome, não nulo
CODPROF	integer(3)	código do professor, chave estrangeira, não nulo

Vamos logar novamente como root, criar o novo bd, as tabelas e as chaves primárias e secundária; a última instrução cria a chave estrangeira, o relacionamento e a integridade referencial. A criação de chaves estrangeiras obriga o mySql a usar um novo modo de acesso aos dados (engine), chamado InnoDB, que é assumido automaticamente. Digite:

```
CREATE DATABASE IF NOT EXISTS escola;
USE escola;
CREATE TABLE cadpro (
    CODPROP INTEGER(3)  UNSIGNED NOT NULL
        AUTO_INCREMENT PRIMARY KEY,
    NOMEPRO VARCHAR(40) NOT NULL,
    CPF     CHAR(11)    NOT NULL UNIQUE);
CREATE TABLE caddis (
    CODDIS  CHAR(6)     NOT NULL,
    NOMEDIS VARCHAR(40) NOT NULL,
    CODPROF INTEGER(3)  UNSIGNED NOT NULL);
ALTER TABLE caddis
    ADD CONSTRAINT fkCODPRO FOREIGN KEY (CODPROF)
    REFERENCES cadpro (CODPROP)
    ON UPDATE RESTRICT
    ON DELETE RESTRICT;
```

As cláusulas aqui utilizadas têm o seguite significado:

- ADD CONSTRAINT: cria a chave estrangeira e lhe dá um nome fkCODPRO. A palavra CONSTRAINT normalmente é traduzida por restrição, embora no contexto da linguagem sql signifique um pouco mais do que isto;
- REFERENCES: indica a tabela e o campo apontado pela chave estrangeira: cadpro (CODPROP)
- ON UPDATE RESTRICT: o valor de cadpro (CODPROP) não pode ser alterado se tiver correspondente em CadDis;
- ON DELETE RESTRICT: o valor de cadpro (CODPROP) não pode ser excluído se tiver correspondente em CadDis.

Vamos criar os registros das duas tabelas para poder exercitar os relacionamentos. Digite:

```
INSERT INTO cadpro VALUES (100,   'SILVANA SOUZA',  '11122233399');
INSERT INTO cadpro VALUES (110, 'JOSE PAULO SILVA', '11122244488');
INSERT INTO cadpro VALUES (120, 'RENATO DE ABREU', '11122244477');
INSERT INTO cadpro VALUES (130, 'PENELOPE DA SILVA', '11122255566');
INSERT INTO cadpro VALUES (140, 'JULIANA DE ALBUQUERQUE', '11122266655');
INSERT INTO cadpro VALUES (150, 'CARLOS MUNHOZ DA SILVA', '11122277744');

INSERT INTO caddis VALUES ('CG-100', 'MATEMATICA', 100);
INSERT INTO caddis VALUES ('CG-110', 'FISICA', 100);
INSERT INTO caddis VALUES ('CG-200', 'PORTUGUES', 110);
INSERT INTO caddis VALUES ('CG-300', 'FILOSOFIA', 110);
INSERT INTO caddis VALUES ('TI-100', 'ALGORITMOS', 120);
INSERT INTO caddis VALUES ('TI-200', 'LINGUAGEM DE PROGRAMACAO I', 120);
INSERT INTO caddis VALUES ('TI-205', 'LINGUAGEM DE PROGRAMACAO II', 130);
INSERT INTO caddis VALUES ('TI-300', 'BANCO DE DADOS', 130);
```

```
INSERT INTO caddis VALUES ('TI-400', 'SISTEMAS OPERACIONAIS', 140);
INSERT INTO caddis VALUES ('TI-500', 'PROGRAMAÇÃO WEB', 140);
INSERT INTO caddis VALUES ('CG-100', 'MATEMATICA', 150);
INSERT INTO caddis VALUES ('CG-300', 'FILOSOFIA', 150);

SELECT * FROM caddis;
SELECT * FROM cadpro;
SHOW INDEX FROM caddis;
SHOW INDEX FROM caddis;
```

Agora vamos testar os CONSTRAINTs de integridade referencial criada. Digite:

```
DELETE FROM cadpro WHERE CODPROF = 100;
SELECT * FROM cadpro;
```

Observe que, para excluir o professor 100 será necessário excluir todos os seus registros da tabela CadDis. Digite:

```
DELETE FROM caddis WHERE CODPROF = 100;
SELECT * FROM caddis;
DELETE FROM cadpro WHERE CODPROP = 100;
SELECT * FROM cadpro;
```

Os dois registros seguintes devem apresentar erro de chave duplicada (primária e secundária). Digite:

```
INSERT INTO cadpro VALUES (150, 'JOSE PAULO DE ALMEIDA', '11155566655');
INSERT INTO cadpro VALUES (300, 'MARIANA DOS SANTOS SILVA',
    '11122277744');
```

Existindo ou não o relacionamento definido (pela CONSTRAINT), pode ser feita a consulta de junção entre duas tabelas. Digite:

```
SELECT caddis.CODDIS, caddis.NOMEDIS, cadpro.NOMEPRO
    FROM caddis, cadpro
    WHERE caddis.CODPROF = cadpro.CODPROP;
SHOW INDEX FROM caddis;
SHOW INDEX FROM caddis;
```

O comando show index mostra uma tabela cujas colunas têm o seguinte significado:

table	nome da tabela
non_unique	0-único 1-aceita duplicação
key_name	nome da chave
seq_in_index	número da sequencia da coluna dentro do índice, a artir de 1
columns_name	nome da coluna
collation	A-coluna é ordenada de forma ascendente. NULL- sem ordenação
cardinality	número de valores únicos no índice ???
sub_part	número de caracteres indexados, se a coluna for parcialmente indexada; NULL: a coluna toda é indexada
null	YES-se a coluna pode ter valor NULL
index_type	tipo (método) de indexação
comment	comentários

ANEXOS

TABELA DE OPERADORES E FUNÇÕES MATEMÁTICAS:

Operador	*Operação*	*Tipo*	*Resultado*
+	Manutenção de sinal	Operador	Positivo
-	Inversão de sinal	Operador	Negativo
POWER(X,N)	Exponenciação: xn	Função	Real
SQRT(X)	Raiz quadrada de X	Função	Real
POWER	Raiz de índice qualquer:	Função	Real
DIV	Divisão com quociente	Operador	Inteiro
MOD(X.N)	Resto de divisão de X	Função	Inteiro
/	Divisão com quociente	Operador	Real
*	Multiplicação	Operador	Inteiro ou
+	Adição	Operador	Inteiro ou
•	Subtração	Operador	Inteiro ou
%	Resto de divisão	Operador	Inteiro

LISTA PARCIAL DE FUNÇÕES:

- *Agregação (estatística):* AVG, COUNT, MAX, MIN, STD, STDDEV, SUN, VARIANCE.
- *Controle de fluxo*: CASE FUNCTION, IF FUNCTION, IFNULL e NULLIF.
- *Conversão*: CAST e CONVERT.
- *Data e hora (calendário e relógio):* ADDDATE, ADDTIME, CONVERTJZ, CURDATE, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, CURTIME, DATE FUNCTION, DATE OPERATIONS, DATEDIFF, DATE_FORMAT, DAY, DAYNAME, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, EXTRACT, FROMJDAYS, FROM_UNIXTIME, GETJORMAT, HOUR, LAST_DAY, LOCALTIME, LOCALTIMESTAMP, MAKEDATE, MAKETIME, MICROSECOND, MINUTE, MONTH, MONTHNAME, NOW, PERIOD_ADD, PERIOD_DIFF, QUARTER, SECOND, SEC_TO_TIME, STR_TO_DATE, SUBDATE, SUBTIME, SYSDATE, TIME FUNCTION, TIMEDIFF, TIMESTAMP FUNCTION, TIMESTAMPADD, TIMESTAMPDIFF, TIME_FORMAT, TIME_TO_SEC, TO_DAYS, UNIX_TIMESTAMP, UTC_DATE, UTC_TIME, UTC.TIMESTAMP, WEEK, WEEKDAY, WEEKOFYEAR, YEAR e YEARWEEK.
- *Encriptação:* AES_DECRYPT, COMPRESS, DECODE, DES_DECRYPT, DES_ENCRYPT, ENGODE, ENCRYPT, MD5, OLD.PASSWORD, PASSWORD, SHA, UNCOMPRESS e UNCOMPRESSED_LENGTH.
- *Informação:* BENCHMARK, CHARSET, COERCIBILITY, COLLATION, CONNECTIONJD, CURRENT.USER, DATABASE, FOUND_ROWS, LASTJNSERTJD, ROW_COUNT, SCHEMA, SESSION.USER, SYSTEM_USER, USER e VERSION.
- *Numérica (matemática):* ABS, ACOS, ASIN, ATAN, ATAN2, CEILING, COS, COT, CRC32, DEGREES, DIV, EXP, FLOOR, LN, LOG, LOG10, LOG2, PI, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, TAN e TRUNCATE.
- *String*: ASCII, BIN, BINARY OPERATOR, BIT_LENGTH, CAST, CHAR FUNCTION, CHARACTER_LENGTH, CHAR.LENGTH, CONCAT, CONCAT_WS, CONV, ELT, EXPORT_SET, FIELD, FIND_IN_SET, FORMAT, HEX, INSERT FUNCTION, INSTR, LCASE, LEFT, LENGTH, LIKE, LOAD_FILE, LOCATE, LOWER, LPAD, LTRIM, MAKE_SET, MATCH AGAINST, MID, NOT LIKE, NOT REGEXP, OCT, OCTETLENGTH, ORD, POSITION, QUOTE, REGEXP, REPEAT FUNCTION,

REPLACE, REVERSE, RIGHT, RPAD, RTRIM, SOUNDEX, SOUNDS LIKE, SPACE, STRCMP, SUBSTRING, SUBSTRINGJNDEX, TRIM, UCASE, UNHEX e UPPER.

- *Variadas*: DEFAULT, GETJ.OCK, INET_ATON, INET_NTOA, IS_FREE_LOCK, IS_USED_LOCK, MASTER_POS-WAIT, NAME_CONST, RELEASE_LOCK, SLEEP, UUID e VALUES.

LISTA PARCIAL DE COMANDOS:

ABS	ACOS	ADDDATE	ADDTIME
AES_DECRYPT	AFTER	ALTER	ANALYZE
AND	AS	ASCII	ASIN
ATAN	ATAN2	AUTO__INCREMENT	AVG
BACKUP	BEFORE	BEGIN	BENCHMARK
BETWEEN	BIGINT	BIN	BINAR Y
BINLOG	BIT	BIT_LENGTH	BLOB
BOOLEAN	BY	BYTE	CACHE
CALL	CASE	CAST	CEILING
CHANCE	CHAR	CHAR_LENGTH	CHARACTER
CHARACTER_LENGTH	CHARSET	CHECK	CHECKSUM
CLIENT	CLOSE	COALESCE	COERCIBILITY
COLLATION	COLUMNS	COMMAND	COMPRESS
CONCAT	CONCAT_WS	CONNECTION_ID	CONSTRAINT
CONV	CONVERT	CONVERT_TZ	COS
COT	COUNT	CRC32	CREATE
CURDATE	CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP
CURRENT_USER	CURTIME	DATA	DATABASE
DATABASES	DATE	DATE_FORMAT	DATEDIFF
DATETIME	DAY	DAYNAME	DAYOFMONTH
DAYOFWEEK	DAYOFYEAR	DEALLOCATE	DEC
DECIMAL	DECLARE	DECODE	DEFAULT
DEGREES	DELAYED	DELETE	DELIMITER
DES_ENCRYPT	DESCRIBE	DIV	DO
DOUBLE	DROP	DUAL	ELSE
ELSEIF	ELT	ENGODE	ENCRYPT
END	ENGINE	ENGINES	ENUM
ERRORS	EVENTS	ECUTE	EXP
EXPLAIN	EXPORT_SET	EXTRACT	FETCH
FIELD	FILE	FIND_IN_SET	FLOAT
FLOOR	FLUSH	FORMAT	FOUND_ROWS
FROM	FROM_DAYS	FROM_UNIXTIME	FUNCTION
GEOMETRY	GET_FORMAT	GET_LOCK	GLOBAL
GRANT	GRANTS	GREATEST	GROUP
HANDLER	HELP	HEX	HIERARCHY
HOSTS	HOUR	IF	IFNULL
IN	INDEX	INET_ATON	INET_NTOA
INFILE	INNODB	INSERT	INSTR
INT	INTEGER	INTERVAL	INTO
IS	IS_FREE_LOCK	IS_USED_LOCK	ISNULL
ISOLATION	ITERATE	JOIN	KILL
LAST_DAY	LAST_INSERT_ID	LCASE	LEAST
L EA VÊ	LEFT	LENGTH	LIKE
LN	LOAD	LOAD_FILE	LOCAL
LOCALTIME	LOCALTIMESTAMP	LOCATE	LOCK
LOG	LOG10	LOG2	LOGS
LONGBLOB	LONGTEXT	LOOP	LOWER

LPAD	LTRIM	MAKE_SET	MAKEDATE
MAKETIME	MASTER	MASTER_POS_WAIT	MATCH AGAINST
MAX	MD5	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MERGE	MICROSECOND	MID
MIN	MINUTE	MONTH	MONTHNAME
MUTEX	NAME_CONST	NEW	NOT
NOW PERIOD_ADD	NULL	NULLIF	OCT
OCTETLENGTH	OLD	OLD_PASSWORD	ON
OPEN	OPERATIONS	OPERATOR	OPTIMIZE
OPTION	OR	ORD	ORDER
PASSWORD	PERIOD_DIFF	PI	POSITION
POWER	PRECISION	PREPARE	PRIVILEGES
PROCEDURE	PROCESS	PROCESSLIST	PURGE
QUARTER	QUERY	QUOTE	RADIANS
RAND	REFERENCES	REGEXP	RELEASE_LOCK
RELOAD	RENAME	REPAIR	REPEAT
REPLACE	REPLICATION	RESET	RESTORE
REVERSE	REVOKE	RIGHT	ROUND
ROW_COUNT	RPAD	RTRIM	SAVEPOINT
SCHEMA	SEC_TO_TIME	SECOND	SELECT
SESSION_USER	SET	SHA	SHOW
SHUTDOWN	SIGN	SIN	SLAVE
SLAVE	SLEEP	SMALLINT	SOUNDEX
SOUNDS	SPACE	SPATIAL	SQL_LOG_BIN
SQRT	START	STATEMENT	STATUS
STD	STDDEV	STOP	STR_TO_DATE
STRCMP	SUBDATE	SUBSTRING	SUBSTRING_INDEX
SUBTIME	SUN	SYSDATE	SYSTEM_USER
TABLE	TABLES	TAN	TEMPORARY
TEXT	THEN	TIME	TIME_FORMAT
TIME_TO_SEC	TIMEDIFF	TIMESTAMP	TIMESTAMP
TIMESTAMPADD	TIMESTAMPDIFF	TINYBLOB	TINYINT
TINYTEXT	TO	TO_DAYS	TRANSACTION
TRIGGER	TRIGGERS	TRIM	TRUNCATE
TYPE	UCAS E	UNCOMPRESS	UNHEX
UNION	UNIQUE	UNIX__TIMESTAMP	UNTIL
UPDATE	UPDATE	UPPER	USAGE
USER	UTC_DATE	UTC_TIME	UTCJTIMESTAMP
UUID	VALUES	VARBINARY	VARCHAR
VARIABLES	VARIANCE	VERSION	VIEW
WARNINGS	WEEK	WEEKDAY	WEEKOFYEAR
WHEN	WHERE	WHILE	XOR
XOR	YEAR	YEARWEEK	

TIPOS DE DADOS (relação parcial):

- AUTOINCREMENT - usado para gerar um valor único sequencial para um novo registro.
- TINYINT[(tamanho)] [UNSIGNED] [ZEROFILL] - quando precisar de valores inteiros pequenos na faixa de -128 até 127. O parâmetro tamanho é opcional e estabelece o tamanho máximo do valor a ser exibido no monitor, podendo ser um valor máximo 255. O parâmetro UNSIGNED, quando usado, estabelece que o valor definido será positivo, ou seja, podem ser utilizados valores na faixa de O a 255. O parâmetro ZEROFILL define o preenchimento com valor zero caso o campo (coluna) permaneça em branco.

- SMALLINT[(tamanho)] [UNSIGNED] [ZEROFILL] - quando for necessário usar valores inteiros pequenos na faixa de valores de -32.768 até 32.767. O parâmetro tamanho é opcional estabelece o tamanho máximo do valor a ser exibido no monitor de vídeo, podendo ser um valor máximo 255. O parâmetro UNSIGNED, quando usado, estabelece que o valor definido será positivo, ou seja, podem ser utilizados valores na faixa de O a 65.535. O parâmetro ZEROFILL estabelece o preenchimento com valor zero caso o campo (coluna) fique em branco.
- INTEGER[(tamanho)] [UNSIGNED] [ZEROFILL] ou INT[(tamanho)] [UNSIGNED] [ZEROFILL] - utilizado quando houver a necessidade de usar valores inteiros longos na faixa de valores de -2.147.483.648 até 2.147.483.647 O parâmetro tamanho é opcional e permite estabelecer o tamanho máximo do valor a ser exibido no monitor de vídeo, podendo ser um valor máximo 255. O parâmetro UNSIGNED, quando usado, estabelece que o valor definido será positivo, ou seja, podem ser utilizados valores na faixa de O a 4.294.967.295. O parâmetro ZEROFILL o preenchimento com valor zero caso o campo (coluna) fique em branco.
- BIGINT[(tamanho)] [UNSIGNED] [ZEROFILL] - utilize esse tipo de dado quando usar valores inteiros grandes na faixa de valores de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. O parâmetro tamanho é opcional e estabelece o tamanho máximo do valor a ser exibido no monitor de vídeo, podendo ser um valor máximo 255. O parâmetro UNSIGNED, quando usado, estabelece que o valor definido será positivo, ou seja, podem ser utilizados valores na faixa de O a 18.446.744.073.709.551.615. O parâmetro ZEROFILL estabelece a definição do preenchimento com valor zero caso o campo (coluna) seja deixado em branco.
- FLOAT[(tamanho, decimal)] [UNSIGNED] [ZEROFILL] - quando usar valores com ponto flutuante pequenos (precisão simples) na faixa de valores de -3.402823466E+38 a -1.175494351E-38, O, e entre 1.175494351E-38 e 3.402823466E+38. O parâmetro tamanho é opcional e permite estabelecer o tamanho máximo do valor a ser exibido no monitor de vídeo, podendo ser um valor máximo 255. O parâmetro decimal determina o número de casas decimais. O parâmetro UNSIGNED, quando usado, estabelece que o valor definido será positivo. O parâmetro ZEROFILL define o preenchimento com valor zero caso o campo (coluna) seja deixado em branco.
- DOUBLE [(tamanho, decimal)] [UNSIGNED] [ZEROFILL] - quando for necessário usar valores com tamanho normal (dupla precisão) na faixa de valores -1.7976931348623157E+308 e -2.2250738585072014E-308, O e entre 2.2250738585072014E-308 e 1.7976931348623157E+308. O parâmetro decimal determina o número de casas decimais. O parâmetro UNSIGNED, quando usado, estabelece que o valor definido será positivo. O parâmetro ZEROFILL estabelece a definição do preenchimento com valor zero caso o campo (coluna) seja deixado em branco.
- DECIMAL[(tamanho [, decimal])] [UNSIGNED] [ZEROFILL] - quando for preciso usar valores com ponto flutuante como se fossem uma sequência de caracteres do tipo CHAR. O parâmetro decimal determina o número de casas decimais. O parâmetro UNSIGNED, quando usado, estabelece que o valor definido será positivo. O parâmetro ZEROFILL estabelece a definição do preenchimento com valor zero caso o campo (coluna) seja deixado em branco.
- DATE - utilizado com uma data de calendário no formato AAAA-MM-DD (formato ANSI) dentro do intervalo de tempo entre '1000-01-01' e '9999-12-31'.
- TIME - utiliza-se esse tipo de dado quando houver necessidade de usar uma informação relacionada a um determinado horário de relógio no intervalo de tempo entre '-838:59:59' e '838:59:59'. Os valores de hora do MySQL são mostrados no formato 'HH:MM:SS', no entanto é possível atribuir valores de colunas (campos) usando strings ou números.
- CHAR(tamanho) ou CHARACTER(tamanho) - usado com sequências de caracteres de tamanho fixo que estejam limitadas a 255 caracteres de comprimento. O parâmetro tamanho

determina o valor máximo em caracteres que pode conter a sequência. Esse tipo de dado, quando definido, preenche o campo com espaços em branco até completar o total de caracteres definidos, quando a totalidade do tamanho do campo não é preenchida.

- VARCHAR(tamanho) - quando precisar de sequências de caracteres de tamanho variável que estejam limitadas a 255 caracteres de comprimento. A diferença entre esse tipo e o CHAR é que, neste caso, os espaços em branco excedentes do lado direito da sequência de caracteres não utilizados são automaticamente desprezados. O parâmetro tamanho determina o valor máximo da sequência em caracteres.
- ENUM(lvalor1',lvalor2',..., ValorN1) - usado com uma lista de valores do tipo string, em que um dos valores pode ser selecionado. O campo (coluna) do tipo ENUM pode possuir até no máximo 65535 valores diferentes.

TIPOS DE DIREITOS (PRIVILÉGIOS) DOS COMANDOS GRANT E REVOKE

Privilégio	*Descrição*
ALL[PRIVILEGES]	Configura todos os privilégios simples, exceto GRANT OPTION
ALTER	Permite uso de ALTER TABLE
CREATE	Permite uso de CREATE TABLE
CREATE TEMPORARY TABLES	Permite uso de CREATE TEMPORARY TABLE
DELETE	Permite uso de DELETE
DROP	Permite uso de DROP TABLE
ECUTE	Permite que o usuário execute stored procedures
FILE	Permite uso de SELECT / INTO OUTFILE e LOAD DATA INFILE
INDEX	Permite uso de CREATE INDEX e DROP INDEX
INSERT	Permite uso de INSERT
LOCK TABLES	Permite uso de LOCK TABLES em tabelas que têm privilégio SELECT
PROCESS	Permite o uso de SHOW FULL PROCESSLIST
REFERENCES	Para uso futuro, não está implementado
RELOAD	Permite o uso de FLUSH
REPLICATION CLIENT	Permite ao usuário obter a localização do master ou slave
REPLICATION SLAVE	Necessário para a replicação slave (leitura dos eventos do log binário do master)
SELECT	Permite o uso de SELECT
SHOW DATABASES	Permite exibir todos os banco de dados
SHUTDOWN	Permite uso de mysqladmin shutdown
SUPER	Permite executar (uma vez) mesmo se max_connections tiverem sido alcançados e executa o comando CHANGE MASTER, KILL thread, mysqladmin debug, PURGE MASTER LOGS e SET GLOBAL
UPDATE	Permite uso de UPDATE
USAGE	Sinónimo para "sem privilégios"
GRANT OPTION	Permite ao usuário repassar os seus privilégios