**Introdução à Linguagem SQL**

* **INTRODUÇÃO**

A linguagem SQL (*Structured Query Language*) foi, sem sombra de dúvida, uma das principais razões do enorme sucesso alcançado pelos SGBD relacionais (SGBDR). Se não a totalidade, a grande maioria dos fornecedores de SGBD relacionais implementam algum dialeto desta linguagem. Pelo fato de ser o padrão adotado pelos SGBDR, ela **permite que esquemas de bancos de dados relacionais sejam migrados entre plataformas (quase) sem problemas**. Na prática, apenas aqueles comandos mais utilizados seguem o padrão. Por conta disto, sempre que há migração de plataformas, algum(ns) ajuste(s) deve(m) ser feito(s). A linguagem SQL é uma **linguagem de domínio específico** (DSL – *Domains Specific Language*), isto é, ela foi criada com o propósito específico de manipular objetos em bancos de dados relacionais. SQL também é uma **linguagem declarativa**. Isto significa que os “programas” (*scripts*) em SQL se preocupam em definir o “que” e não o “como”.

A linguagem SQL é dividida em sublinguagens, de acordo com o tipo de manipulação executada. A divisão mais granular envolve 5 sublinguagens:

* **DDL** (***D****ata****D****efinition****L****anguage*) – Utilizada na manipulação (criação, alteração, exclusão) de **objetos** do banco de dados (comandos CREATE, ALTER e DROP);
* **DML** (***D****ata****M****anipulation****L****anguage*) – Utilizada para manipulação do **conteúdo** de objetos do banco de dados, ou seja, os dados propriamente ditos (comandos INSERT, UPDATE e DELETE);
* **DQL**(***D****ata****Q****uery****L****anguage*) – Utilizada na **consulta** dados (comando SELECT);
* **TCL**(***T****ransaction****C****ontrol****L****anguage*) – Utilizada no controle de **transações** (comandos SET TRANSACTION, START TRANSACTION, COMMIT, ROLLBACK e SAVEPOINT);
* **DCL** (***D****ata****C****ontrol****L****anguage*) – Utilizada no controle da segurança dos dados, atribuindo permissões e privilégios de usuários (comandos GRANT, REVOKE e DENY).

*Alguns autores consideram a DML como parte da DQL e dividem a linguagem SQL em apenas quatro sublinguagens.*

A linguagem SQL **não**é um SGBDR, mas sim parte dele. Comandos SQL são enviados ao SGBDR por aplicações (programas) ou de forma interativa, através de um *front-end*. O SGBDR recebe os comandos, os envia para o mecanismo de armazenamento (*database engine*) que os interpreta eexecuta, retornando o resultado para o solicitante. Os comandos SQL possuem todos a mesma forma: começam com um **verbo**, que indica a função do comando, seguido por uma ou mais **cláusulas** que especificam um objeto (tabela, índice etc.), um componente de um objeto (coluna), uma condição ou complementam a ação a ser tomada. Por exemplo, o comando a seguir cria um novo esquema relacional chamado FINANCEIRO:

         CREATE SCHEMA FINANCEIRO DEFAULT CHARACTER SET utf8;

O comando CREATE (verbo) é seguido das cláusulas SCHEMA, que completa o significado do comando, indicando a criação de um novo esquema relacional, e DEFAULT CHARACTER SET, que especifica o conjunto de caracteres padrão. Todo comando SQL deve ser terminado com um ponto-e-vírgula.

Nem todas as cláusulas são obrigatórias. A sintaxe completa do comando INSERT, como implementado no SGBDR MySQL, é mostrado abaixo. As cláusulas entre colchetes são opcionais.

INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]  
   
    [INTO] tbl\_name  
   
    [PARTITION (partition\_name [, partition\_name] …)]  
   
    [(col\_name [, col\_name] …)]  
   
    { {VALUES | VALUE} (value\_list) [, (value\_list)] …  
   
      |  
   
      VALUES row\_constructor\_list  
   
    }  
   
    [AS row\_alias[(col\_alias [, col\_alias] …)]]  
   
    [ON DUPLICATE KEY UPDATE assignment\_list]

Em SQL, utilizam-se os termos **banco de dados**(ou **esquema**), **tabela**, **campo**(ou **coluna**), **tipo** e **registro**(ou **linha**) no lugar de **esquema relacional**, **relação**, **atributo**, **domínio** e **tupla**, respectivamente. Ao longo do texto, ambas as denominações poderão ser utilizadas.

Na apresentação dos comandos da linguagem SQL, será utilizada a seguinte notação:

Itens opcionais entre colchetes ([ ]). Exemplo: INSERT [INTO];

Itens obrigatórios com mais de uma opção: as opções são colocadas entre chaves ({ }), separadas por uma barra vertical (|). Exemplo: CREATE { DATABASE  |  SCHEMA };

Itens que podem ocorrer uma ou mais vezes indicados por três pontos (…). Exemplo: { VALUES | VALUE } (*value\_list*) [, (*value\_list*)] …

Comandos ou cláusulas SQL sempre em letra maiúscula;

Demais itens em letra maiúscula ou minúscula.

*Apesar de os comandos serem apresentados em letras maiúsculas, a linguagem SQL não diferencia letras minúsculas e maiúsculas.*

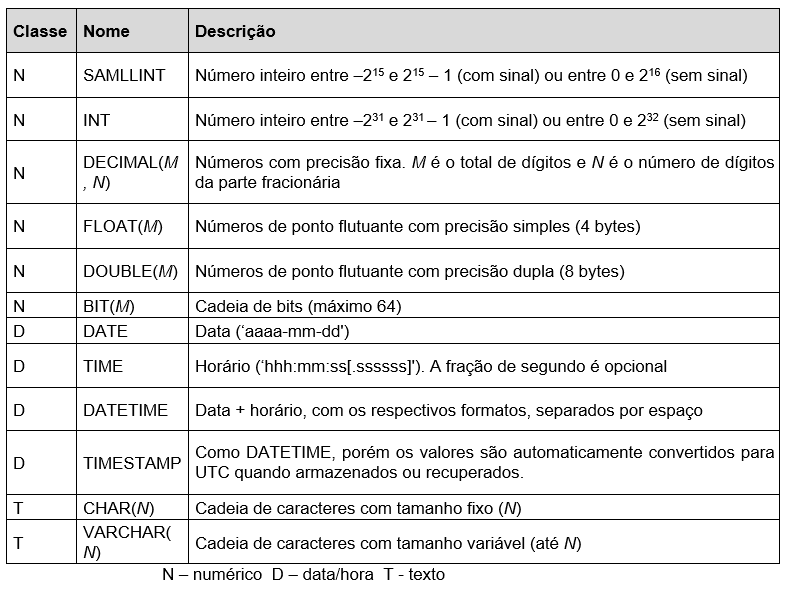
Em SQL, os valores literais (texto) devem sempre vir entre aspas (“ “) ou apóstrofos (‘ ‘).

A sintaxe e exemplos apresentados seguirão o dialeto e o comportamento do SGBDR MySQL. Serão apontadas as diferenças de comportamento e sintaxe quanto estas forem significativas.

* **TIPOS DE DADOS**

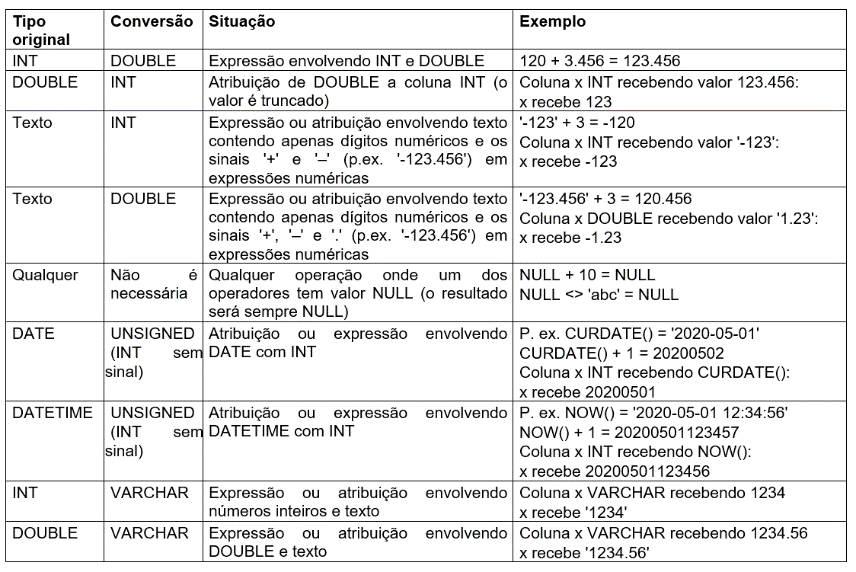
Um domínio, ou tipo, define o conjunto de valores que um atributo (coluna) pode receber. Embora haja um padrão estabelecido para a linguagem SQL, a maioria dos fornecedores inclui tipos próprios em seus dialetos. A grande maioria das aplicações comerciais utilizam um conjunto relativamente pequeno de tipos, que podem ser divididos em 3 classes: numéricos, data/hora e texto.

A Tabela apresenta os principais tipos de cada classe definidos na linguagem padrão. A princípio, todos os SGBDR suportam estes tipos.



Apesar de não fazer parte da linguagem padrão, o tipo BOOLEAN é implementado em praticamente todos os dialetos mais usados. Este é um tipo lógico, que admite apenas os valores TRUE (verdadeiro) e FALSE (falso). Com o grande crescimento do uso de conteúdos digitais (vídeos, áudios e imagens), a maioria dos dialetos implementa também a classe BLOB(***B****inary****L****arge****OB****ject* ou Grande Objeto Binário) para armazenamento destes conteúdos. Outra classe, CLOB (***C****haracter****L****arge****OB****ject* ou Grande Objeto de Texto), presente na linguagem SQL padrão, é utilizada para o armazenamento de textos grandes.

Apesar de os SGBDR tentarem, sempre que possível, compatibilizar valores atribuídos a colunas com tipos diferentes, deve-se tomar muito cuidado. Por exemplo, o MySQL faz automaticamente as seguintes compatibilizações:



*Na dúvida, utilize as funções CAST() ou CONVERT() para garantir a conversão desejada.*

* **CRIAÇÃO DE ESQUEMAS E TABELAS**

Os comandos da Linguagem de Descrição de Dados (DDL) são utilizados para criação, alteração e exclusão de objetos, isto é, esquemas, tabelas, índices etc. O primeiro passo é a criação de um esquema. A forma geral do comando utilizado para tal é:

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nome\_do\_esquema  
   
    [especificações] …  
   
   
especificações: {  
   
    [DEFAULT] CHARACTER SET [=] conjunto\_de\_caracteres  
   
  | [DEFAULT] COLLATE [=] nome\_do\_conjunto\_de\_regras\_de\_agrupamento  
   
  | DEFAULT ENCRYPTION [=] {‘Y’ | ‘N’}  
   
}

Os comandos da DDL possuem as cláusulas opcionais IF NOT EXISTS, para criação de objetos, e IF EXISTS, para alteração e exclusão, que verificam se o objeto existe antes da execução. Isto evita que o SGBDR rejeite o comando e retorne um código de erro. Em alguns dialetos, como o MySQL, as cláusulas DATABASE e SCHEMA são sinônimos. Conjunto\_de\_caracteres define como os dados de texto serão interpretados e armazenados. Nome\_do\_conjunto\_de\_regras\_de\_agrupamento define as regras para ordenação e agrupamento. É aconselhável não alterar os valores padrão para estas duas cláusulas. Por exemplo, os dois comandos a seguir são equivalentes e criam o esquema *contábil:*

         CREATE DATABASE IF NOT EXISTS contabil;  
         CREATE SCHEMA IF NOT EXISTS contabil;

Depois de criado o esquema, o passo seguinte é a criação de tabelas. A sintaxe completa do comando CREATE TABLE é muito extensa. Serão apresentadas apenas as cláusulas mais utilizadas:

CREATE TABLE [IF NOT EXISTS] nome\_tabela  
   
             (coluna1 definição\_coluna1 [, coluna2 definição\_coluna2] …  
   
[, CONSTRAINT [nome] PRIMARY KEY (coluna1 [, coluna2] …)]  
   
[, CONSTRAINT [nome] UNIQUE KEY (coluna1 [, coluna2] …)]  
   
[, CONSTRAINT [nome] FOREIGN KEY (coluna1 [, coluna2] …)]  
   
[, CONSTRAINT [nome] CHECK (expressão)]   
   
    )   
   
    [AUTO\_INCREMENT [=] valor];  
   
   
definição\_coluna:  
   
    tipo [NOT NULL | NULL] [DEFAULT {valor\_literal | (expressão)}]  
      [AUTO\_INCREMENT] [[CONSTRAINT] nome CHECK (expressão)]

Quando um aplicativo ou usuário se conecta ao SGBDR, são solicitados o login/senha e o nome de um esquema para a conexão (esquema padrão). Se não for usado qualquer qualificador, a tabela será criada no esquema definido como padrão. A criação tabelas em esquemas diferentes é possível qualificando-se a tabela: *nome\_do\_esquema.nome\_da\_tabela* (o usuário precisa ter permissão de criação de tabelas no esquema de destino). Supondo que o esquema padrão seja *contábil,* ambos os comandos a seguir cria uma tabela chamada *lancamento*(foram omitidas as demais cláusulas):

         CREATE TABLE IF NOT EXISTS lancamento …;  
         CREATE TABLE IF NOT EXISTS contabil.lancamento …;

As colunas devem ter obrigatoriamente um nome e um tipo. As demais cláusulas são detalhadas a seguir.

* [NOT NULL | NULL] informa se a coluna pode ter ou não o valor NULL. O padrão é NULL;
* [DEFAULT {valor\_literal | (expressão)}] atribui um valor padrão à coluna, caso não lhe seja atribuído qualquer valor na inclusão de linhas. Valor padrão pode ser um valor constante (*valor\_literal*) ou uma expressão;
* [AUTO\_INCREMENT] especifica que a coluna é autoincrementada, isto é, a cada novo registro inserido, o valor da coluna é automaticamente incrementado de uma unidade. Esta cláusula se aplica apenas para colunas com tipos numéricos. Só pode haver uma coluna autoincrementada por tabela;
* [AUTO\_INCREMENT [=] valor] define o valor inicial de uma coluna autoincrementada.

CREATE TABLE IF NOT EXISTS contabil.lancamento (  
   
    num INT(6) AUTO\_INCREMENT,  
   
    conta\_cred DECIMAL(10) NOT NULL,  
   
    conta\_debt DECIMAL(10) NOT NULL,  
   
    valor DECIMAL(9,2) NOT NULL,  
   
    data DATE NOT NULL,  
   
    historico VARCHAR(100) DEFAULT ‘’  
   
) AUTO\_INCREMENT = 1;

*O script acima cria a tabela lançamento no esquema contabil com 6 colunas. A coluna num é autoincrementada. Nenhuma das colunas pode ser NULL, exceto histórico, que possui valor padrão ‘’. Os valores de num começam com o valor 1.* A cláusula AUTO\_INCREMENT = 1 não é necessária, pois 1 é o valor padrão para campos autoincrementados.

Tanto a linguagem SQL padrão como seus inúmeros dialetos permitem a definição de restrições no momento de criação da tabela. Restrições são definidas pela cláusula CONSTRAINT e podem ocorrer tanto na definição de coluna (restrições de coluna) quanto fora (restrições de tabela).

 CONSTRAINT nome\_restrição tipo\_restrição atribui uma restrição de nome *nome\_restrição*e tipo *tipo\_restrição*a uma coluna ou à tabela. Os tipos de restrição são:

 PRIMARY KEY define a chave primária da tabela;

UNIQUE KEY define uma chave única. Pode haver múltiplas chaves únicas por tabela;

FOREIGN KEY define uma chave estrangeira para a tabela. Pode haver múltiplas chaves estrangeiras por tabela. A tabela e coluna(s) referenciadas (tabela-pai) existir no esquema;

CHECK (expressão) define uma restrição aos valores de uma ou mais colunas. *Expressão*é verificada a cada inclusão ou alteração de linha em uma tabela. Se *expressão* é VERDADEIRA ou NULL, a operação é concluída. Caso contrário, ela é rejeitada.

Algumas regras se aplicam à definição de restrições:

Se *nome*não for fornecido, o SGBDR criará um;

O escopo de uma restrição é o equema todo. Isto significa que não pode haver duas restrições com o mesmo nome, mesmo que em tabelas diferentes;

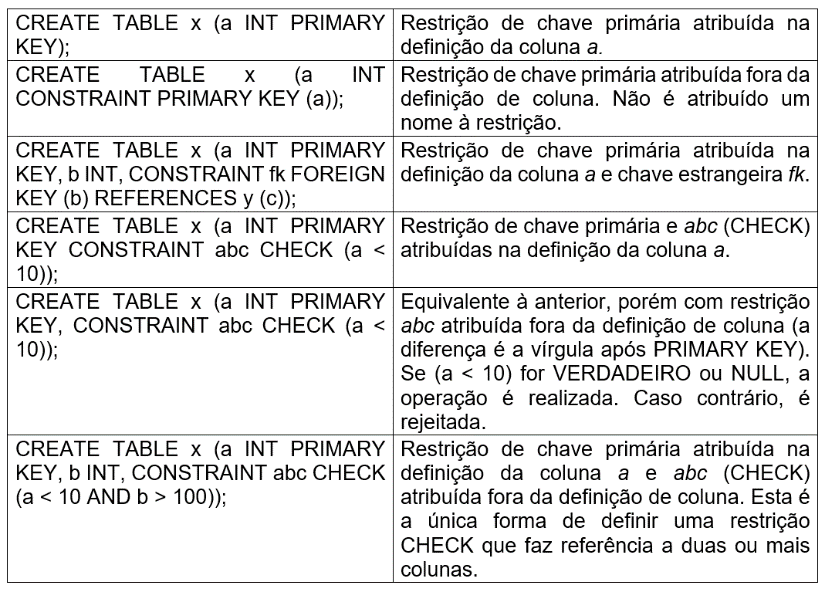
Restrições de coluna só podem fazer referência à própria coluna;

A única restrição que pode ser utilizada juntamente com a cláusula CONSTRAINT, tanto na definição de coluna quanto fora dela, é a CHECK. PRIMARY KEY e UNIQUE KEY podem ser utilizadas na definição de coluna, porém sem a cláusula CONSTRAINT;

A restrição FOREIGN KEY só pode ser atribuída fora da definição de coluna;

Se PRIMARY KEY é atribuída na definição de coluna, não pode haver outra restrição PRIMARY KEY na tabela e vice-versa. PRIMARY KEY compostas (com mais de uma coluna) devem ser definidas como restrição de tabela;

CHECK definido como restrição de tabela pode fazer referência a mais de uma coluna.



* **EXCLUSÃO DE ESQUEMAS E TABELAS**

A exclusão de objetos é feita através do comando DROP. O usuário precisa ter privilégios para usar este comando. Mas cuidado: o esquema e **todas** as suas tabelas serão excluídos. Depois de executado o comando, não há como recuperar os dados.

         DROP {DATABASE | SCHEMA} [IF EXISTS] nome\_do\_esquema;

A exclusão de tabelas é feita da mesma forma e também necessita que o usuário tenha privilégios para tal. A tabela e todos os seus registros são excluídos do esquema.

         DROP TABLE [IF EXISTS] nome\_da\_tabela;

* **ALTERAÇÃO DE TABELAS**

A alteração da estrutura de tabelas é feita pelo comando ALTER TABLE. O usuário precisa ter privilégios para usar este comando. A sintaxe completa deste comando é muito extensa. Serão apresentadas as cláusulas mais comuns.

ALTER TABLE nome\_da\_tabela  
   
[  RENAME [TO | AS] novo\_nome\_tabela]  
   
[, ADD [COLUMN] nome\_coluna definição\_coluna [FIRST | AFTER nome\_coluna] …]  
   
[, DROP [COLUMN] nome\_coluna…]  
   
[, CHANGE [COLUMN] nome\_antigo nome\_novo definição\_coluna [FIRST | AFTER nome\_coluna]]  
   
[, ALTER [COLUMN] nome\_coluna{SET DEFAULT {literal | (expr)} …)]                
   
[, ADD CONSTRAINT [nome\_chave] PRIMARY KEY (coluna1 [, coluna2] …)]  
   
[, ADD CONSTRAINT [nome\_chave] UNIQUE KEY (coluna1 [, coluna2] …)]  
   
[, ADD CONSTRAINT [nome\_chave] FOREIGN KEY (coluna1 [, coluna2] …) REFERENCES tabela\_pai  
(coluna1, coluna2, …)]  
   
[, CONSTRAINT [nome] CHECK (expressão)]   
   
[, DROP PRIMARY KEY]   
   
[, DROP FOREIGN KEY nome\_chave]  
   
[, DROP KEY nome\_chave]  
   
[, DROP CONSTRAINT [nome]]   
   
[, AUTO\_INCREMENT [=] valor];

**Troca do nome da tabela**: ALTER TABLE lancamento TO lanc;

**Inclusão de novas colunas**: a cláusula ADD COLUMN permite incluir colunas em uma tabela. Todas as regras que se aplicam ao comando CREATE TABLE, em relação a restrições, se aplicam aqui também. O comando a seguir inclui duas novas colunas, c1 e c2 em lançamento, a primeira após a coluna data e a segunda após c1. É obrigatória a definição do tipo da coluna.

ALTER TABLE lancamento ADD COLUMN c1 VARCHAR(10) AFTER historico,   
ADD COLUMN c2 VARCHAR(10) AFTER c1;

**Exclusão de colunas**: a cláusula DROP COLUMN permite excluir colunas em uma tabela. O comando a seguir exclui as colunas c1 e c2 adicionadas acima.

ALTER TABLE lancamento DROP COLUMN c1, DROP COLUMN c2;

**Alteração de nome, tipo e posição de colunas**: a cláusula CHANGE COLUMN permite alterar, de uma só vez, todos os três atributos de uma coluna. Caso você não queira alterar todos, basta repetir o valor atual. O comando abaixo muda o nome de histórico para hist, seu tipo para VARCHAR(150) e muda sua posição, colocando-o depois de data.

ALTER TABLE lancamento CHANGE COLUMN historico hist VARCHAR(150) AFTER data;

**Alteração de valor padrão de colunas:** a cláusula ALTER COLUMN permite alterar o valor padrão de uma coluna. O primeiro comando a seguir remove o valor padrão de hist vigente. O segundo, altera seu valor padrão para ‘ND’.

ALTER TABLE lancamento ALTER COLUMN hist SET DEFAULT NULL;  
ALTER TABLE lancamento ALTER COLUMN hist SET DEFAULT ‘ND’;

**Inclusão de restrições:** a cláusula ADD CONSTRAINT inclui restrições em uma tabela. Todas as regras que se aplicam ao comando CREATE TABLE se aplicam aqui também. Os comandos a seguir incluem um restrição de chave primária, com nome pk\_lancamento, formada pelo conjunto de conta\_cred e conta\_debt , uma chave estrangeira (num), com nome fk\_lancamento, que faz referência ao campo cpf da tabela emp, e uma chave única, com nome uk\_lancamento, formada pelo conjunto de valor e data, todas em lancamento. Se o nome da restrição não for fornecido, o SGBDR criará automaticamente um.

ALTER TABLE lancamento ADD CONSTRAINT pk\_lancamento PRIMARY KEY (conta\_cred, conta\_debt);  
 ALTER TABLE lancamento ADD CONSTRAINT fk\_lancamento FOREIGN KEY num REFERENCES emp(cpf);  
   
ALTER TABLE lancamento ADD CONSTRAINT uk\_lancamento UNIQUE KEY (valor, data);

*Cuidados a serem tomados caso a tabela já esteja populada:*

* Os valores da combinação dos campos da chave primária devem ser únicos;
* Os valores da combinação dos campos da chave única devem ser únicos;
* Os valores da combinação de campos da chave estrangeira devem existir na tabela referenciada;

**Exclusão de restrições:** as cláusulas DROP PRIMARY KEY, DROP FOREIGN KEY e DROP KEY permitem a exclusão de chaves primárias, estrangeiras e únicas de uma tabela independentemente de como foram criadas. DROP CONSTRAINT permite a exclusão de restrições a partir de seus nomes. Os comandos a seguir excluem as três restrições incluídas acima.

ALTER TABLE lancamento DROP PRIMARY KEY;  
ALTER TABLE lancamento DROP FOREIGN KEY fk\_lancamento;  
ALTER TABLE lancamento DROP KEY uk\_lancamento;

*Cuidados a serem tomados caso a tabela já esteja populada:*

* Não é possível excluir restrições de chave primária de tabelas que estejam sendo referenciadas por alguma chave estrangeira.
* **DICIONÁRIO DE DADOS**

Em esquemas simples, com um número reduzido de tabelas, é fácil manter na cabeça as estruturas de todas elas. À medida que o tamanho e complexidade dos esquemas aumenta, torna-se necessário recorrer a ferramentas que forneçam estas informações.

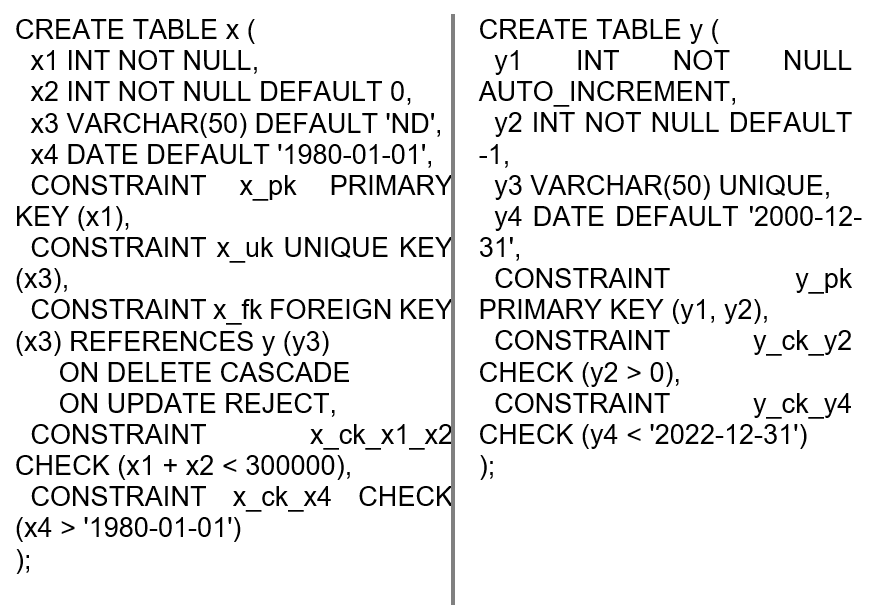
Dicionários de dados são metadados9que descrevem os objetos de um SGBDR, como por exemplo seus esquemas, tabelas, colunas de cada tabela, usuários dentre outros. A partir dos elementos do dicionário de dados, é possível conhecer e compreender tudo aquilo que compõe um SGBR.

No passado, dicionários de dados eram documentos em papel ou meio digital, mantidos à parte, e necessitavam ser atualizados sempre que havia alguma alteração nos esquemas ativos. Hoje em dia, todos os SGBDR armazenam estas informações em esquemas e tabelas próprios. O trabalho de atualização fica por conta do SGBDR e as informações são obtidas através de consultas às tabelas correspondentes.

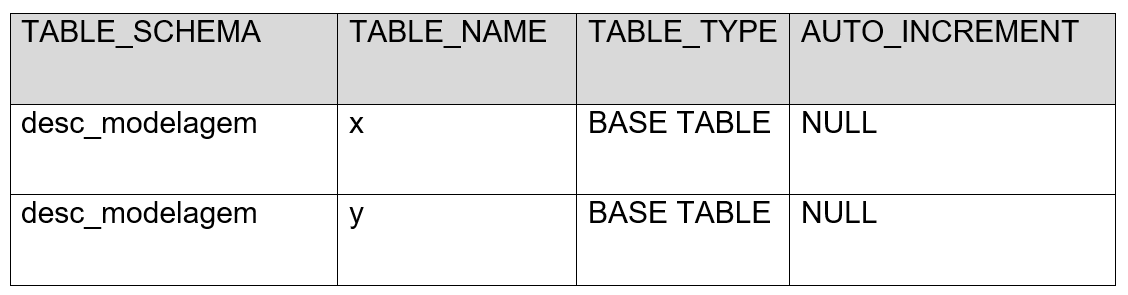
O esquema INFORMATION\_SCHEMA contém todas as tabelas utilizadas pelo SGBDR, porém não há um padrão para nomea-las. No SGBDR MySQL, por exemplo, existem quatro tabelas que contém os metadados dos esquemas: SCHEMATA, TABLES, COLUMNS e REFERENTIAL\_CONSTRAINTS.

* **SCHEMATA** relaciona todos os esquemas presentes no SGBDR e suas características;
* **TABLES**relaciona todas as tabelas presentes no SGBDR e suas características, inclusive as tabelas de INFORMATION\_SCHEMA;
* **COLUMNS** relaciona todas as colunas presentes no SGBDR e suas características;
* **REFERENTIAL\_CONSTRAINTS**relaciona todas as chaves estrangeiras presentes no SGBDR e suas características;

A partir do conteúdo das tabelas acima, é possível compreender e reproduzir todos os esquemas relacionais presentes no SGBDR. Por exemplo, considere as tabelas *X*e *Y*, cujos comandos de criação são apresentados a seguir:

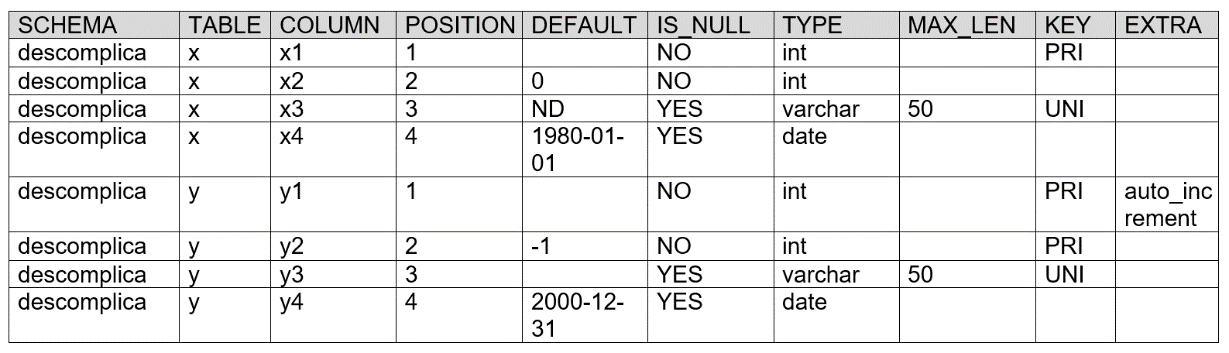


A tabela INFORMATION\_SCHEMA.TABLES apresenta o seguinte conteúdo relativo à *X*e *Y*10:



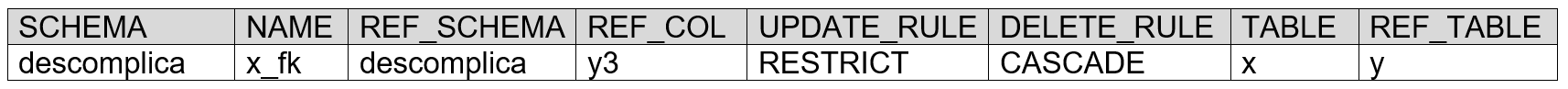
TABLE\_SCHEMA e TABLE\_NAME correspondem ao esquema a que pertence a tabela e seu nome. TABLE\_TYPE informa o tipo de tabela: BASE TABLE (tabela normal), VIEW, SYSTEM VIEW (tabelas do esquema INFORMATION\_SCHEMA). AUTO\_INCREMENT informa o próximo valor a ser usado no campo AUTO\_INCREMENT.

A tabela INFORMATION\_SCHEMA.COLUMNS apresenta o seguinte conteúdo relativo às tabelas *X*e *Y***11**:



O significado de cada coluna fica claro quando comparadas com os comandos de criação de *X* e *Y*.

A tabela INFORMATION\_SCHEMA.REFERENTIAL\_CONSTRAINTS apresenta o seguinte conteúdo relativo à *X*e *Y*12:



* **INCLUSÃO DE REGISTROS**

Considere que a tabela *lançamento* tenha sido criada a partir do comando abaixo:

CREATE TABLE IF NOT EXISTS lancamento (  
   
    num INT(6) AUTO\_INCREMENT,  
   
    conta\_cred DECIMAL(10) NOT NULL,  
   
    conta\_debt DECIMAL(10) NOT NULL,  
   
    valor DECIMAL(9,2) NOT NULL,  
   
    data DATE NOT NULL,  
   
    historico VARCHAR(100) DEFAULT ‘’,  
   
   PRIMARY KEY pk\_lancamento (num)  
   
) AUTO\_INCREMENT = 1;

O comando INSERT é utilizado para inclusão de registros em uma tabela. A sintaxe do comando e de suas cláusulas mais comuns é mostrada abaixo.

INSERT [INTO] nome\_da\_tabela  
    [(coluna1 [, coluna2] …)]  
    {VALUES | VALUE} (valor1, [, valor2]) [, (valor1, [, valor2])] …  
   
      
   
         As quatro formas mais comuns do comando INSERT são:  
   
   
         INSERT INTO lancamento (conta\_cred, num, valor, conta\_debt, data, historico)  
   
VALUES (123456, 0, 12345.89, 654321,‘2020-04-01’, ‘Referente compra de papel’);  
   
   
         INSERT INTO lancamento  
   
VALUES (0, 123456, 654321, 12345.89, ‘2020-04-01’, ‘Referente compra de papel’);  
   
   
         INSERT INTO lancamento (conta\_cred, valor, conta\_debt, data)  
   
VALUES (666666, 3456.898, 222222,‘2020-04-01’);  
   
   
         INSERT INTO lancamento  
   
VALUES (0, 333333, 444444, 100.00, ‘2020-04-02’, ‘Referente compra de café’),   
   
                  (0, 555555, 666666, 200.00, ‘2020-04-03’, ‘Referente compra de água’);

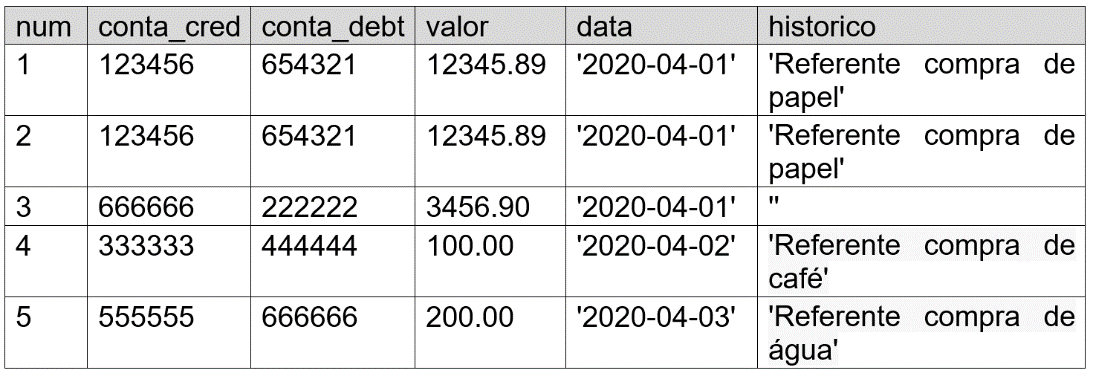
No primeiro comando, é fornecida a lista de colunas, **em qualquer ordem**, e a lista de valores, **na mesma ordem da lista de colunas**. No segundo, a lista de colunas é omitida, porém a lista de valores deve especificar valores para **todas** as colunas, inclusive as autoincrementadas (neste caso, utilize o valor 0), na mesma ordem em que aparecem na tabela. Observe que os dois primeiros comandos são equivalentes.

O terceiro caso é similar ao primeiro, porém apenas as colunas que receberão valores aparecem na lista de colunas. As demais colunas recebem seus valores padrão ou NULL. Se alguma das colunas omitidas não tiver valor padrão e for não nula, o comando será rejeitado retornando um código de erro. A única exceção são as colunas autoincrementadas, como *num*, cujo valor é determinado pelo SGBDR.

No terceiro comando, *valor* foi fornecido com 3 casas decimais embora tenha sido especificado com 2. O SGBDR automaticamente arredonda o valor até o número de casas decimais especificado. Se o valor ultrapassar o tamanho máximo de um campo, o comando é rejeitado.

É possível incluir mais de um registro em um único comando INSERT. Basta que as listas de valores sejam separadas por vírgulas, como no quarto comando.

Após a execução dos três comandos, a tabela *lançamento*ficaria como mostrado na Tabela:



*Cuidados a serem tomados:*

* Inclusão de linhas com chave primária ou chave única repetida é rejeitada;
* Inclusão de linhas com chave estrangeira inválida é rejeitada.
* Atribuição de valores que ultrpassem os limites da respectiva coluna é rejeitada. Para colunas do tipo VARCHAR, pode-se utilizar a função SUBSTR() para evitar erros.

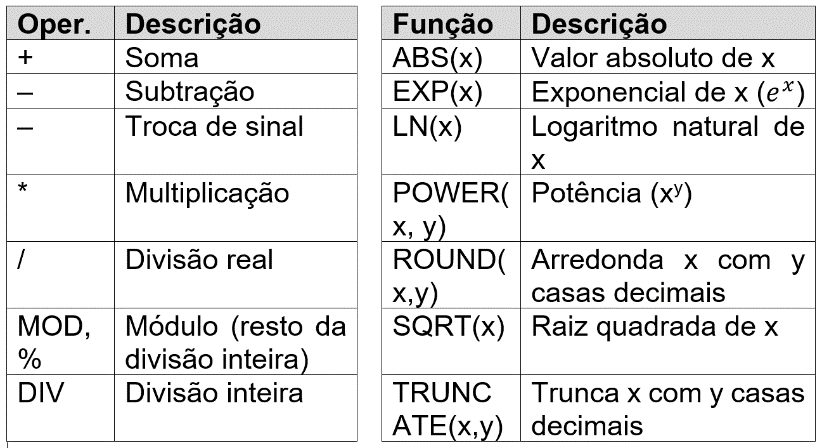
​

* **OPERADORES E FUNÇÕES SELECIONADOS**

A linguagem SQL oferece um conjunto de operadores aritméticos, lógicos, relacionais e de conjuntos que são muito úteis para seleção de registros de uma tabela.

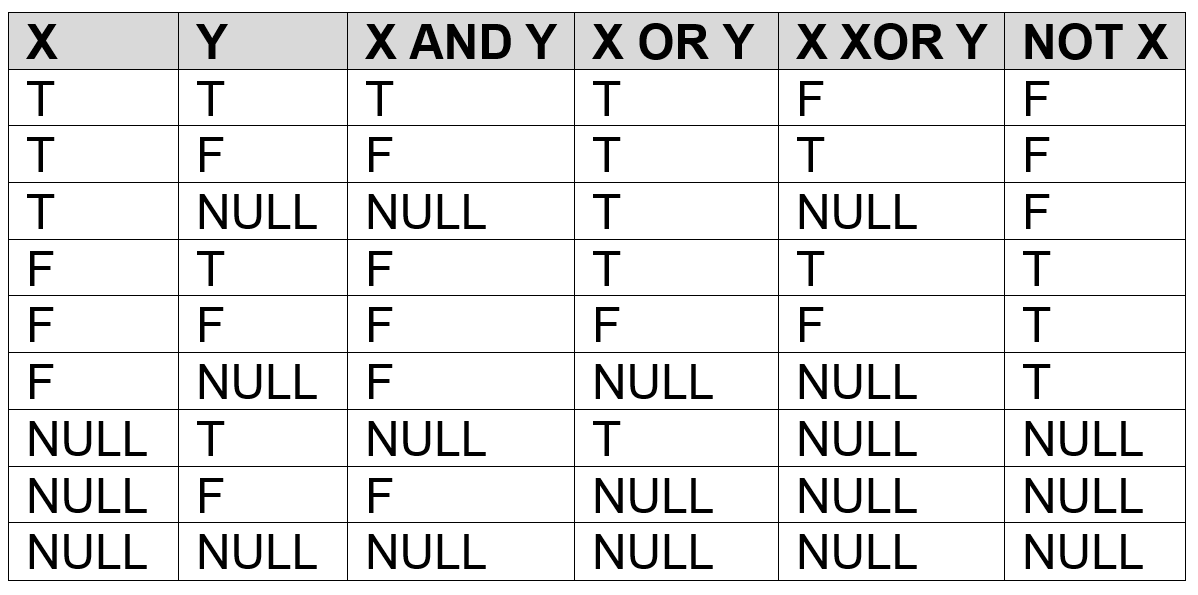
* **Operadores aritméticos**

Os operadores aritméticos são, em sua maioria, binários, isto é, possuem dois operandos. A exceção é o operador de troca de sinal (menos unário). Todas as funções aritméticas retornam o valor NULL em caso de erro.



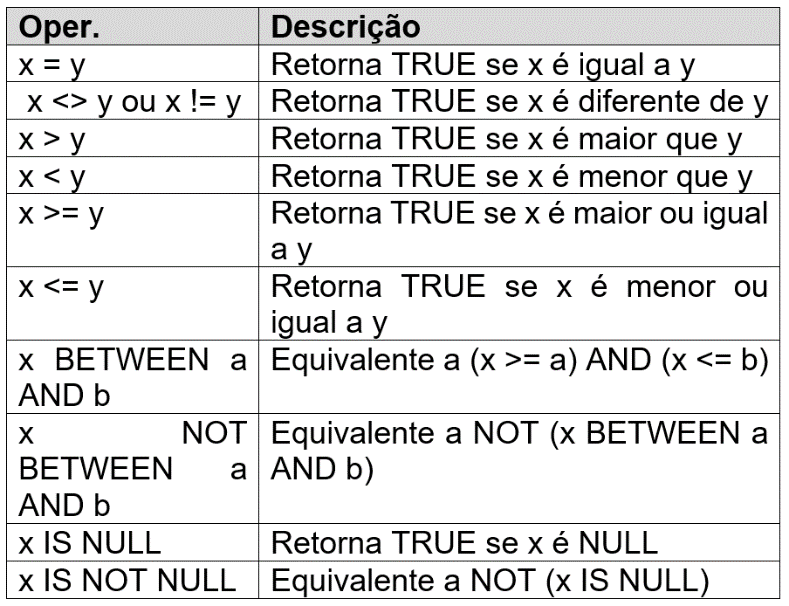
* **Operadores lógicos**

Os operadores lógicos são binários a exceção do não unário (NOT), retornando os valores TRUE, FALSE ou NULL, dependendo dos valores de seus operandos. Os operandos podem ser dos tipos lógico, numérico ou texto. Números diferentes de zero são tratados como TRUE e o valor 0, FALSE. Operandos do tipo texto são considerados sempre FALSE. A tabela-verdade a seguir resume o funcionamento dos operadores lógicos. Os operadores lógicos são: AND ou &&, OR ou ||, XOR e NOT ou !.



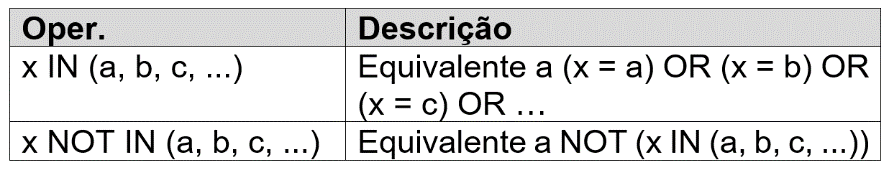
* **Operadores relacionais**

Operadores relacionais comparam dois operandos retornando sempre TRUE, FALSE ou NULL. Como regra geral, se um dos operandos tem o valor NULL, o resultado é NULL.  Embora seja possível comparar operandos de tipos diferentes, o melhor a fazer é converter o valor de um dos operandos para o tipo do outro utilizando a função CAST().



* **Operadores de conjuntos**

Os operadores de conjunto verificam se um elemento pertence a um conjunto, retornando TRUE, FALSE ou NULL.



A precedência dos operadores é a seguinte: operadores unários (! e –), operadores aritméticos multiplicativos (\*, /, DIV, %, MOD), operadores aritméticos aditivos (+ e –), operadores relacionais (=, <>, !=, <, >=, <=) e de conjunto (IN e NOT IN), operadores relacionais BETWEEN e NOT BETWEEN, operadores lógicos multiplicativos (AND e &&), operador lógico XOR, operadores lógicos aditivos (OR, ||). Expressões entre parênteses são executadas primeiro. Os operadores de um mesmo nível são executados da esquerda para a direita. Em caso de dúvida, coloque a expressão desejada entre de parênteses para forçar sua execução.

* **ALTERAÇÃO DE REGISTROS**

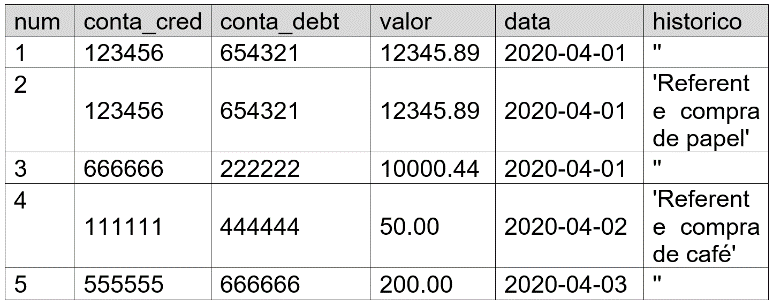
O comando UPDATE é utilizado para alterar o valor de campos de um registro. Diferentemente do INSERT, é necessário informar em que registro(s) a(s) alteração(ões) será (ão) feita(s). A cláusula WHERE é utilizada para este fim.

UPDATE nome\_da\_tabela  
    SET coluna1 = [valor1 | expressão1 | DEFAULT]    
                  [,coluna2 = [valor2 | expressão2 | DEFAULT]  ] …  
    [WHERE condição];

Este comando atribui novos valores a colunas de um registro **existente**. O valor atribuído pode ser uma constante, uma expressão (pode incluir colunas da tabela) ou o valor padrão para a coluna (DEFAULT). A cláusula SET indica que colunas modificar e com que valores. Se a cláusula WHERE for incluída, apenas o(s) registro(s) em que *condição* for verdadeira serão afetados. Caso contrário, **todos** os registros da tabela serão afetados. *Condição* pode ser simples, com apenas um termo, ou composta, com dois ou mais termos conectados pelos operadores AND, OR e NOT. Os termos de *condição*utilizam os operadores relacionais =, <> (diferente de), >, <, ≥ e ≤13. Veja os exemplos abaixo. Considere o estado inicial da tabela *lançamento* conforme mostrado na Tabela.

UPDATE lancamento SET valor = 10000.44 WHERE num = 3;  
UPDATE lancamento SET historico = DEFAULT WHERE (num = 1 OR num = 5);  
UPDATE lancamento SET valor = valor / 2, conta\_cred = 111111 WHERE num = 4;  
UPDATE lancamento SET conta\_debt = 999999 WHERE num = 10;

O primeiro comando altera *valor* apenas em um registro (*num* = 3). O segundo altera dois registros (*num =*1 e *num* = 5), atribuindo a *histórico* o seu valor padrão (‘’). O terceiro altera duas colunas do registro com *num* = 4, utilizando uma expressão para atribuir a *valor* metade de seu valor inicial e uma constante, atribuindo a *conta\_cred* o valor 111111. Finalmente o último comando não altera qualquer registro de *lançamento*, pois nenhum deles atende às condições da cláusula WHERE. Após a execução dos quatro comandos, a tabela *lançamento*ficaria como mostrado:



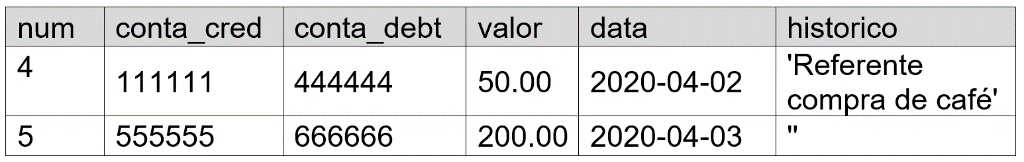
* **EXCLUSÃO DE REGISTROS**

O comando DELETE é utilizado para remover registros. Muito cuidado: a cláusula WHERE **deve** ser utilizada para selecionar os registros a serem excluídos ou **todos** os registros da tabela serão excluídos. A sintaxe mais comum de DELETE é simples:

DELETE FROM nome\_da\_tabela  
    [WHERE condição]

DELETE FROM lancamento WHERE conta\_debt <> 654321 AND conta\_cred = 123456;  
DELETE FROM lancamento WHERE data < ‘2020-04-02’;

O primeiro comando não exclui qualquer registro, pois não há registros que atendam à condição. O segundo exclui os três primeiros registros, pois todos atendem à condição. Após a execução dos dois comandos, a tabela *lançamento*ficaria como mostrado:



* **INTEGRIDADE REFERENCIAL EM SQL**

         A linguagem SQL fornece mecanismos que garantem a integridade referencial de um esquema relacional. Modificações ou exclusões de registros em tabelas referenciadas por chaves estrangeiras podem violar a integridade referencial do esquema. Duas cláusulas são utilizadas para definir, se desejado, as ações corretivas a serem tomadas. Estas cláusulas complementam a criação de chaves estrangeiras.

[CONSTRAINT [nome\_chave]] FOREIGN KEY (coluna1, coluna2, …)  
    REFERENCES tabela\_pai (coluna1, coluna2, …)  
    [ON DELETE RESTRICT | CASCADE | SET NULL | NO ACTION]  
    [ON UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION];

1.   A cláusula ON DELETE define a ação a ser tomada quando um registro na tabela-pai que esteja sendo referenciado pela chave estrangeira é excluído. As ações possíveis são:

a.   RESTRICT – não permite que o registro seja excluído da tabela-pai, retornando um código de erro;

b.   CASCADE – propaga a exclusão para a tabela-filho (os registros da tabela-filho que fazem referência ao registro excluído da tabela-pai são excluídos também);

c.    SET NULL – atribui o valor NULL à chave estrangeira dos registros da tabela- filho que fazem referência ao registro excluído da tabela-pai (a chave estrangeira da tabela-filho não pode ser NOT NULL);

d.   NO ACTION – o mesmo que RESTRICT.

2.   A cláusula ON UPDATE define a ação a ser tomada quando o campo referenciado pela chave estrangeira é alterado na tabela-pai. As ações idênticas às da cláusula ON DELETE.

A inclusão de restrições em tabelas populadas faz com que todas as suas linhas sejam verificadas. A inclusão é feita se não há qualquer violação de integridade referencial no esquema. Caso contrário, a inclusão é rejeitada. Deve-se tomar cuidado com a ordem de criação de tabelas com referências e restrições de integridade relacional. As tabelas referenciadas devem ser criadas antes da criação de tabelas que fazem referência a elas.

O exemplo a seguir ilustra o funcionamento destas duas cláusulas. Considera as tabelas EMP e DEPT, criadas e populadas a partir dos comandos abaixo (observe a ordem de criação):

CREATE TABLE IF NOT EXISTS dept (  
did INT(6) NOT NULL AUTO\_INCREMENT,   
nome VARCHAR(30) NOT NULL,  
CONSTRAINT pk\_dept PRIMARY KEY (did),  
);  
   
CREATE TABLE IF NOT EXISTS emp (  
ssn INT(6) NOT NULL AUTO\_INCREMENT,   
dept\_id INT(6) NOT NULL,  
CONSTRAINT pk\_emp PRIMARY KEY (ssn),  
CONSTRAINT fk\_dept FOREIGN KEY (dept\_id) REFERENCES dept (did)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE  
);  
   
INSERT INTO dept  
VALUES (10, ‘Financeiro’), (20, ‘Recursos Humanos’), (30, ‘Marketing’);  
   
INSERT INTO emp  
VALUES (111111, 10), (222222, 10), (333333, 20);

Agora, os comandos abaixo são aplicados na ordem em que aparecem:

1. INSERT INTO emp VALUES (444444, 40);
2. INSERT INTO dept VALUES (40, ‘Comercial’);
3. INSERT INTO emp VALUES (444444, 40);
4. UPDATE dept SET did = 15 WHERE did = 10;
5. DELETE FROM dept WHERE did = 30;
6. DELETE FROM dept WHERE did = 20;

O comando (1) tenta incluir um novo registro em EMP, porém a referência da chave estrangeira é inválida (não há em DEPT qualquer registro com *did* = 40). O comando é rejeitado.

O comando (2) inclui um novo registro em dept.

O comando (3) repete a ação do comando (1), dó que com sucesso (a referência agora existe).

O comando (4) altera o registro com *did* = 10 em DEPT. O valor de *did* passa a ser 15. **Todas** as referências existentes são alteradas de acordo, por conta da cláusula ON UPDATE CASCADE (os dois registros de EMP que fazem referência ao registro alterado têm o valor de suas chaves estrangeiras alterado).

O comando (5) exclui o registro com *ssn* = 30 em DEPT. Nenhuma ação é tomada, pois não há qualquer referência a este registro.

Finalmente, o comando (6) exclui o registro com *did* = 20 em DEPT. **Todos** os registros que fazem referência ao registro excluído são também excluídos de suas respectivas tabelas por conta do comando ON DELETE CASCADE (os dois registros de EMP que fazem referência ao registro excluído são também excluídos).

Um cuidado a ser tomado: os nomes dados às restrições não podem ser repetidos dentro de um mesmo esquema, mesmo que pertençam a tabelas distintas.