Renata Miyagusku

Curso Prático de SQL

Guia de referência completo para usar a linguagem SQL nos bancos de dados:

MS SQL Server

Oracle

PostgreSQL

MySQL

Diretor Editorial
Luis Matos

Coordenação Editorial
Renata Miyagusku

Assistência Editorial
Carolina Evangelista

Projeto Gráfico
Daniele Fátima

Preparação dos Originais
Jeferson Ferreira

Revisão
Shirley Figueiredo Ayres

Diagramação
Fabiana Pedrozo e Stephanie Lin

Capa
Daniel Brito

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

M685c Miyagusku, Renata.

Curso Prático de SQL / Renata Miyagusku. – São Paulo : Digerati Books, 2008.
96 p.

ISBN 978-85-60480-85-2

1. MySQL (Linguagem de programação de computadores). 2. Bancos de dados. I. Título.

CDD 005.13

Universo dos Livros Editora Ltda.
Rua Tito, 1.609
CEP 05051-001 • São Paulo/SP
Telefone: (11) 3648-9090 • Fax: (11) 3648-9083
www.universodoslivros.com.br
e-mail: editor@universodoslivros.com.br

Conselho Administrativo: Alessandro Gerardi, Alessio Fon Melozo, Luis Afonso G. Neira, Luis Matos e William Nakamura.

Sumário

Capítulo 1

Definição de banco de dados e de linguagem SQL

Conceitos relacionados a banco de dados

A seguir, vejamos os principais conceitos relacionados a bancos de dados:

- **Campo**: unidade que armazena uma informação. Pode ser um nome, um código, um telefone etc. Cada campo possui um determinado tipo (caractere, numérico, data etc.) e só aceita informações desse tipo definido. Alguns campos também podem ficar vazios;
- **Registro**: conjunto de campos com informações relacionadas, isto é, pertencentes a um mesmo assunto, pessoa ou empresa, por exemplo. Um registro contendo código, nome, endereço e telefone pode ser referente a um cliente, do mesmo modo que um registro com CNPJ, razão social, nome fantasia, endereço e telefone armazena os dados de uma empresa;
- **Tabela**: conjunto de registros com a mesma estrutura (os mesmos campos). Considerando a estrutura do registro descrita no conceito anterior, uma tabela de clientes seria formada por vários registros de clientes com os mesmos campos de código, nome, endereço e telefone;
- **Banco de dados**: consiste em um conjunto de tabelas que podem ou não se relacionar entre si, e, em geral, são utilizadas por um ou vários sistemas de computadores que acessam essas tabelas tanto para incluírem novas informações quanto para consultá-las;
- **Sistema Gerenciador de Banco de Dados (SGBD)**: consiste em um conjunto de aplicativos, recursos e funcionalidades que organizam e gerenciam todos os componentes de um banco de dados (conjunto de tabelas), para que ele, além de armazenar as informações, permita a consulta e manipulação dos dados de forma eficiente e confiável;
- **Chave primária**: pode ser definida como um ou mais campos cuja função é a de identificar um registro. Esse campo contém uma informação única e não permite que se repita na tabela a qual pertence. Uma chave primária é simples quando for definida por apenas um campo, e composta quando formada por mais de um campo;

- **Relacionamento 1-1**: ocorre quando um valor de um campo de uma tabela só pode ser referenciado em outra tabela uma única vez, e vice-versa, criando uma referência única;
- **Relacionamento 1-N**: ocorre quando um valor de um campo de uma tabela pode ser referenciado várias vezes em outra tabela, ou seja, vários registros de uma tabela podem possuí-lo como chave estrangeira;
- **Relacionamento N-N**: ocorre quando vários valores de um campo de uma tabela (no caso, a chave primária) podem se repetir várias vezes em um campo de outra tabela (como chave estrangeira).

Conceitos relacionados à linguagem SQL

A linguagem SQL envolve comandos não só de consulta a dados, mas também para manipulação e definição de regras e operações que conservam a integridade e consistência dos dados, além de permitir a implementação de procedimentos (*stored procedures*), funções (*functions*) e gatilhos (*triggers*), aplicações desenvolvidas em outras linguagens. A seguir, uma breve descrição de cada uma elas:

- *Stored procedure*: trata-se de um procedimento que pode ou não receber parâmetros de entrada e retornar dados de saída (quando o banco de dados permitir parâmetros In/Out), assim como modificar tabelas do banco de dados por meio de inclusões, alterações e exclusões contidas em seu código. Um exemplo seria um procedimento chamado `Cadastra _ Aluno` receber como parâmetros o nome, o endereço, o telefone e a sala em que o aluno está matriculado. Essas informações serão recebidas pelo código do procedimento e gravadas na tabela de alunos;

- *Function*: uma função consiste em uma rotina que, ao ser chamada, executa seu código e retorna um resultado por meio da própria chamada. As funções podem ser implementadas e existem diversas, próprias para bancos de dados. Um exemplo seria uma função `Date()` retornar, pela chamada de seu nome, a data corrente do sistema, que deve ser atribuída à uma variável;

• *Trigger*: é chamado de "gatilho", pois é "disparado" em função de eventos que ocorrem nas tabelas do banco de dados. Não precisa ser chamado via código. Um exemplo seria criarmos um *trigger* relacionado a toda inclusão de informação em uma determinada tabela. Nesse *trigger* podemos incluir um código-fonte que valida alguma informação na própria tabela, ou em outro local, antes do dado ser incluído. Esse recurso é bastante útil para manter a consistência e integridade das informações.

Capítulo 2

Conhecendo cada banco de dados

Neste capítulo, conheceremos algumas características de cada um dos bancos de dados que serão, aqui, trabalhados, incluindo seus recursos, funcionalidades e as diferenças em relação a tipos de dados, sintaxe e funções. Como este livro é focado na linguagem SQL, e não em administração de dados, ou em uma plataforma específica de utilização, informações mais complexas relacionadas à instalação e configuração dos bancos de dados podem ser obtidas por meio de fontes especializadas no assunto.

MS SQL Server

Este é o banco de dados relacional da Microsoft. Possui integração com ferramentas e linguagens como o Framework.NET (para o desenvolvimento de aplicações em VB.NET e C#, por exemplo), mas não é multiplataforma, como o PostgreSQL e o Oracle, ou seja, roda somente na plataforma Microsoft.

Entre os principais recursos do SQL Server, podemos destacar os serviços de replicação para processamento distribuído de dados, ou armazenamento secundário, além dos recursos de notificação relacionados a atualizações e outras informações personalizadas, que podem ser feitos por meio de dispositivos conectados ou móveis. Também possui recursos de ETL (extração, transformação e carregamento) para integração de dados, análise on-line (OLAP) com armazenamento multidimensional para dados complexos, soluções para criação e gerenciamento de relatórios (tanto em papel quanto interativos), integração com o Microsoft Visual Studio 2005, entre outros.

Entre as ferramentas do SQL Server, temos:

- **MS SQL Enterprise Manager**: permite a configuração e o gerenciamento do banco de dados;
- **Query Analyzer**: utilizado para execução de *queries* (consultas) às informações do banco de dados;
- **Profile**: exibe um histórico das ações efetuadas no banco de dados, inclusive das que estão sendo executadas no momento.

A última versão do SQL Server a ser lançada é a 2008, sucedendo a 2005. Essa nova versão destaca-se por ir além de um banco de dados relacional, na medida em que poderá ler, carregar e armazenar diversos tipos de dados, desde documentos a arquivos XML. Também fornecerá uma infra-estrutura para implementar e fornecer

BI (*Business Intelligence*) para toda a empresa que o utiliza, promovendo uma completa integração. Além disso, a integração do SQL Server 2008 com o Framework.NET 3.0 promete um desenvolvimento de aplicações muito mais ágil. O site oficial do SQL Server é http://www.microsoft.com/brasil/servidores/sql/default.mspx.

Oracle

O Oracle é um banco de dados baseado em arquitetura Cliente/Servidor, ou seja, os dados são requisitados no lado cliente e processados no servidor, que, por sua vez, só retorna o resultado do processamento.

Essa divisão de tarefas facilita o processamento entre dois sistemas, diminuindo o tráfego na rede e a sobrecarga de trabalho. Os dados enviados pelo lado cliente são armazenados em variáveis ou parâmetros enviados ao servidor. O Oracle recebe esses dados e chama o processamento solicitado (pode ser uma *stored procedure* ou *function*, por exemplo). Depois, o servidor só envia ao cliente um retorno, informando se a tarefa foi concluída com sucesso ou não.

As principais ferramentas do Oracle são:

SQL*PLUS

É o ambiente que permite executar procedimentos e comandos individuais do SQL, sendo composto por um editor de *queries*, no qual é possível salvar as linhas de comando em arquivos .sql, além de executá-las.

Oracle Navigator

Consiste em uma interface gráfica que permite a visualização e manipulação de todos os componentes do banco de dados. Por meio do Navigator, o desenvolvedor pode criar, alterar e excluir tabelas, procedimentos, *functions*, *triggers* etc. É muito útil para a organização e manutenção do banco de dados.

SQL Developer

O Oracle SQL Developer é uma ferramenta gráfica gratuita para desenvolvimento de aplicações relacionadas ao banco de dados.

Com ele, você pode visualizar os objetos do banco de dados, executar *queries* e *scripts* SQL e editar/debugar enunciados em PL/SQL. Também é possível executar relatórios já existentes, do mesmo modo que podemos criar e salvar novos.

O SQL Developer pode ser utilizado para versões do Oracle a partir da 9.2.0.1 e roda em plataforma Windows, Linux e Mac OSX.

Oracle Application Express

O Oracle Application Express (Oracle APEX), antigamente chamado de HTML DB, é uma ferramenta de desenvolvimento de aplicações Web para Oracle. Com um browser e um pouco de conhecimento em programação é possível desenvolver aplicações profissionais rápidas e seguras.

Oracle TimesTen In-Memory Database

O Oracle TimesTen In-Memory Database é um banco de dados relacional, por memória, que dá autonomia às aplicações, por meio de respostas instantâneas, e resultados rápidos requeridos por empresas que trabalham com sistemas de tempo real e indústrias de telecom, mercados de capitais e defesa. Localizado na camada de aplicação como cachê ou banco de dados embutido, o Oracle TimesTen In-Memory Database opera em bases de dados ajustadas por inteiro na memória física utilizando as interfaces padrão do Oracle.

Oracle Berkeley DB Product Family

O Oracle Berkeley DB Family consiste na linha de aplicativos *open source* que provê alta performance para desenvolver casos de uso que não requerem SQL.

SQL*Loader

SQL*Loader é um ultra-rápido utilitário de carregamento de arquivos externos para as tabelas do banco de dados. O SQL*Loader aceita o *input* de dados em vários formatos, com ou sem filtros, e pode carregar dados em múltiplos bancos de dados Oracle durante uma única sessão de carregamento.

O SQL*Loader provê três métodos de carregamento de dados: **Conventional Path Load**, **Direct Path Load** e **External Table Load**.

PL/SQL

Sigla para *Procedural Language/SQL*, é a linguagem SQL do Oracle, composta por todos os comandos SQL padrão, além de possuir funções e comandos próprios do Oracle (deve-se tomar cuidado ao migrar esses códigos para outro banco de dados, pois existem diferenças na sintaxe).

A última versão do banco de dados é a 11g, lançada em julho de 2007. O site oficial da Oracle é www.oracle.com.

PostgreSQL

O PostgreSQL é um SGBD (Sistema Gerenciador de Banco de Dados) relacional de código aberto e gratuito, que surgiu a partir do antigo Ingres, da Universidade de Berkeley. Inicialmente, o PostgreSQL só rodava em plataforma Unix, mas, atualmente, já existe suporte para plataforma Windows. O site oficial do fabricante é http://www.postgresql.org/ e o download do SGBD pode ser feito via browser pelo endereço http://www.postgresql.org/ftp/, ou via FTP pelo endereço ftp://ftp.postgresql.org/pub/. O banco de dados encontra-se atualmente na versão 8.2.5, mas a versão beta 8.3 já está disponível para download e testes.

Para acesso ao banco de dados, uma boa opção é o PgAdmin (www.pgadmin.org), que consiste em uma poderosa ferramenta para administração e desenvolvimento, disponível para plataforma Linux e Microsoft.

MySQL

O MySQL pode ser considerado o mais popular SGBD SQL *open source*. Consiste em um banco de dados relacional. Funciona em plataforma Windows, Linux, FreeBSD, BSDI, Solaris, Mac OSX, SunOS, SGI etc. É compatível com drivers ODBC, JDBC, .NET, entre outros. O MySQL consiste apenas no SGBD e seu site oficial é o www.mysql.com.

As principais ferramentas do MySQL são:

MySQL Migration Toolkit

Consiste em um *framework* que permite migrar de um banco de dados proprietário para o MySQL. Com uma interface de assistente, o MySQL Migration Toolkit implementa uma metodologia que orienta o usuário nos passos necessários para efetuar uma migração completa.

MySQL Administrator

O MySQL Administrator é uma ferramenta de administração com interface gráfica para o gerenciamento e monitoramento dos dados. Verifica, também, como o banco de dados está sendo operado.

MySQL Query Browser

O MySQL Query Browser é uma ferramenta com interface gráfica para implementar, executar e otimizar *queries* em SQL no MySQL.

MySQL Workbench

O MySQL Workbench é uma ferramenta de modelagem de dados para MySQL e pode ser utilizada para desenhar e criar novos bancos de dados, documentar os já existentes e efetuar migrações para o MySQL. Além disso, outra opção é o MySQL Control Center (MySQLCC), que provê interface gráfica para gerenciamento de usuários, administração do servidor e acesso e manipulação aos dados.

Capítulo 3

Utilizando o SQL

Com base nas teorias apresentadas no primeiro capítulo, definiremos um pequeno banco de dados que será manipulado pela linguagem SQL. Esse banco de dados será simples e possuirá as seguintes tabelas e relacionamentos, conforme mostra a figura a seguir:

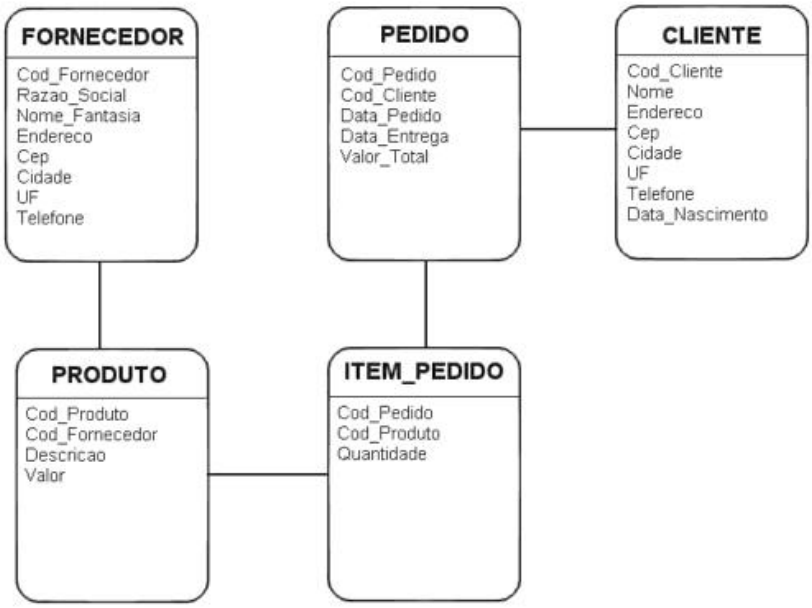

Figura 3.1: Modelo Entidade-Relacionamento.

A partir desse modelo, apresentaremos a sintaxe básica do SQL, válida para todos os bancos de dados, além das diferenças existentes entre eles. Com base nesses conceitos, criaremos algumas tabelas que serão utilizadas em exemplos específicos.

Nota sobre a sintaxe dos comandos SQL: as sintaxes apresentadas ao longo deste livro possuem marcas que não são escritas nas *queries*, mas que significam opções e instruções opcionais. Por exemplo:

```
ALTER TABLE <tabela>
[ADD/DROP/ALTER/RENAME] COLUMN <campo> [<tipo>]
[SET/DROP] [NOT NULL];
```

No exemplo, que altera a estrutura de uma tabela, temos:

- `[ADD/DROP/ALTER/RENAME]`: quando o par de colchetes apresenta opções, indica que você deve utilizar apenas uma delas;
- `[NOT NULL]`: já nesse caso, quando só há uma instrução entre colchetes, indica que ela é opcional e pode ser omitida.

Para retirar, por exemplo, a propriedade `NOT NULL` de um campo, a *query* seria escrita da seguinte forma:

```
ALTER TABLE tabela
ALTER COLUMN coluna DROP NOT NULL;
```

Em primeiro lugar, especificaremos os dados das cinco tabelas a que nos referimos. Os nomes dos campos não precisam ter as iniciais escritas em maiúsculas, porém, vamos adotar essa formatação para diferenciar campo de código SQL. Os tipos são definidos com o tamanho do campo entre parênteses. A definição em SQL será feita posteriormente para cada banco de dados, visto que cada um trabalha com nomenclaturas de tipos diferentes:

Campo	Tipo
Cod_Fornecedor	Numérico(15)
Razao_Social	Caractere(60)
Nome_Fantasia	Caractere(60)
Endereço	Caractere(60)
CEP	Numérico(8)
Cidade	Caractere(35)
UF	Caractere(2)
Telefone	Numérico(10)

Tabela 3.1: Fornecedor.

Campo	Tipo
Cod_Produto	Numérico(15)
Cod_Fornecedor	Numérico(15)

Descricao	Caractere(60)
Valor	Numérico(10,2)

Tabela 3.2: Produto.

Campo	Tipo
Cod_Pedido	Numérico(15)
Cod_Cliente	Numérico(15)
Data_Pedido	Data
Data_Entrega	Data
Valor_Total	Numérico(10,2)

Tabela 3.3: Pedido.

Campo	Tipo
Cod_Pedido	Numérico(15)
Cod_Produto	Numérico(15)
Quantidade	Numérico(5)

Tabela 3.4: Item_Pedido.

Campo	Tipo
Cod_Cliente	Numérico(15)
Nome	Caractere(60)
Endereço	Caractere(60)
CEP	Numérico(8)
Cidade	Caractere(35)
UF	Caractere(2)
Telefone	Numérico(10)
Data_Nascimento	Data

Tabela 3.5: Cliente.

Mas, para criar as tabelas correspondentes às entidades descritas, é preciso definir a nomenclatura correta dos tipos de campos que armazenarão tais informações. No tópico a seguir, conheceremos os tipos de dados utilizados em cada banco de dados.

Tipos de dados

As informações que armazenamos em um banco de dados podem se referir a códigos, nomes, datas e valores. Cada uma dessas informações possui um tipo específico e, para que sejam armazenadas adequadamente, os campos responsáveis devem suportar esses tipos de dados.

Cada banco de dados possui nomenclatura própria para definir campos, assim como a capacidade de armazenamento. A seguir, listaremos os tipos de dados suportados por cada um dos bancos de dados, tornando, assim, mais fácil a tarefa de migrar o código de um sistema para outro, considerando a compatibilidade e correspondência dos tipos de dados.

MS SQL Server

A seguir, uma tabela com os tipos de dados definidos no MS SQL Server:

Tipo de dado	Descrição
`BIGINT`	Até 2^63-1 (9,223,372,036,854,775,807).
`BINARY, VARBINARY`	Até 8.000 bytes.
`BIT`	0, 1 ou nulo.
`CHAR`	Até 8.000 posições.
`DATETIME`	1 de Janeiro de 1753 até 31 de Dezembro de 9999.
`DECIMAL(M,D)`	M números antes da vírgula (precisão de até 38 dígitos) e D casas decimais (17 posições).
`FLOAT`	Até 53 dígitos, sendo 15 de precisão.
`IMAGE`	Até 2.147.483.647 bytes.
`INT, INTEGER`	Até 2^31-1 (2,147,483,647).

MONEY	-922,337,203,685,477.5808 a 922,337,203,685,477.5807.
NCHAR	Até 4.000 posições (formato Unicode).
NTEXT	Até 1.073.741.823 caracteres (formato Unicode).
NVARCHAR	Até 4.000 posições (formato Unicode).
NUMERIC	Precisão de até 38 dígitos e até 17 casas decimais.
REAL	Até 53 dígitos, sendo 15 de precisão.
SMALL DATETIME	1 de Janeiro de 1900 até 6 de Junho de 2079.
SMALL MONEY	Até 214,748.3647.
SMALLINT	Até 2^15-1 (32,767).
TEXT	Até 2.147.483.647 caracteres.
TIMESTAMP	8 bytes.
TINYINT	0 a 255.
VARCHAR	Até 8.000 posições.

Tabela 3.6.

Oracle

Agora, conheceremos os tipos de dados do Oracle:

Tipo de dado	Descrição
CHAR	Strings de caracteres de tamanho pré-definido. Caso uma variável do tipo CHAR for declarada com dez posições, ela ocupará este espaço mesmo que seja atribuído apenas um caractere a ela.
VARCHAR2	Dados alfanuméricos com comprimento de -32.768 a 32.767 bytes. Uma variável do tipo VARCHAR, independente do tamanho que é declarada, ocupará apenas o espaço da informação, que pode ter até o tamanho da variável.
NUMBER	Valores numéricos de até 38 dígitos e ponto flutuante.
DATE	Valores em formato de data (ano, mês e dia).
BOOLEAN	Valores TRUE e FALSE.
LONG	Valores numéricos longos e inteiros.
FLOAT	Valores numéricos com ponto flutuante.

ROWID	Identificador de linha.
RAW	Armazena imagens.
LONG RAW	Armazena dados binários longos do tipo RAW, de tamanho variável. Esse tipo não é interpretado pelo PL/SQL.
NCHAR	String de caracteres no padrão Unicode com comprimento fixo. Suporta 32.767 bytes e o tamanho mínimo padrão é de um byte.

Tabela 3.7.

PostgreSQL

Os tipos de dados aceitos pelo PostgreSQL são os seguintes:

Tipo de dado	Descrição
SMALLINT	-32.768 até 32.767.
INTEGER	-2.147.483.648 até 2.147.483.647.
BIGINT	-9.223.372.036.854.775.808 até 9.223.372.036.854.775.807.
DECIMAL	Numérico ilimitado.
NUMERIC	Numérico ilimitado.
REAL	Numérico com seis casas decimais.
DOUBLE PRECISION	Numérico com 15 casas decimais.
SERIAL	1 até 2.147.483.647.
BIGSERIAL	1 até 9.223.372.036.854.775.807.
MONEY	-21474836.48 a 21474836.47.
CHARACTER VARYING(N), VARCHAR(N)	Comprimento variável com tamanho limitado.
CHARACTER(N), CHAR(N)	Comprimento fixo com brancos à direita.
TEXT	Comprimento variável com tamanho ilimitado.
BYTEA	String binária de comprimento variável.
NAME	Descrição.
TIMESTAMP [(P)] [WITHOUT TIME ZONE]	Data e hora.

TIMESTAMP [(P)] WITH TIME ZONE	Data e hora com *time zone*.
INTERVAL [(P)]	Intervalo de tempo.
DATE	Data.
TIME [(P)] [WITHOUT TIME ZONE]	Hora.
TIME [(P)] WITH TIME ZONE	Hora com *time zone*.

Tabela 3.8.

MySQL

A seguir, a listagem completa dos tipos de dados utilizados no MySQL:

Tipo de dado	Descrição
BIGINT	Números inteiros entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807.
BLOB, TEXT	Blocos longos de texto de até 65.536 caracteres.
CHAR(n)	N bytes.
DATE	Ano-Mês-Dia.
DATETIME	Ano-Mês-Dia hh:mm:ss.
DECIMAL(M,D)	M números antes da vírgula e D casas decimais.
DOUBLE, DOUBLE PRECISION	Valor numérico com vírgula, podendo ter até 308 posições.
ENUM('valor1','valor2',...)	1 ou mais bytes dependendo do valor escolhido. A lista pode conter até 65.535 valores distintos.
FLOAT	Valor numérico com vírgula, podendo ter até 38 posições.
INT, INTEGER	Números inteiros entre -2.147.483.548 e 2.147.483.547.
LONGBLOB, LONGTEXT	Blocos longos de textos de até 4.294.967.295 caracteres.
MEDIUMBLOB, MEDIUMTEXT	Blocos longos de textos de até 16.444.216 caracteres.
MEDIUMINT	Números inteiros entre -8.388.608 e 8.388.607.

NUMERIC(M,D)	M números antes da vírgula e D casas decimais.
REAL	Valor numérico com vírgula, podendo ter até 308 posições.
SET('valor1','valor2',...)	1, 2, 3, 4 ou 8 bytes, dependendo do valor escolhido. A lista pode ter até 64 valores diferentes.
SMALLINT	Números inteiros entre -32.758 e 32.757.
TIME	Hora (hh:mm:ss).
TIMESTAMP	Data e hora.
TINYBLOB, TINYTEXT	Valores BLOB ou TEXT com até 256 caracteres.
TINYINT	Números inteiros entre -128 a 127.
VARCHAR(n)	Seqüência de caracteres de comprimento variável, podendo chegar até 255 caracteres.
YEAR	Dois ou quatro dígitos para valores de anos.

Tabela 3.9.

Símbolos e operadores

Além dos tipos de dados, é importante saber quais operadores e símbolos são aceitos nos bancos de dados, para evitar erros em cálculos, ou, até mesmo, de execução de *queries* em caso de incompatibilidade.

Este tópico apresenta tais informações para cada banco de dados.

MS SQL Server

O MS SQL Server trabalha com os seguintes operadores:

Operador	Descrição	Exemplo	Resultado
+	Adição.	2 + 3	5
-	Subtração.	2 - 3	-1
*	Multiplicação.	2 * 3	6
/	Divisão.	4 / 2	2
%	Módulo (retorna o resto inteiro de uma divisão).	12 % 5	2

Tabela 3.10: Operadores aritméticos.

Operador	Descrição
=	Igual a.
>	Maior que.
<	Menor que.
>=	Maior ou igual que.
<=	Menor ou igual que.
<>	Diferente de.
!=	Diferente de.
!<	Não menor que.
!>	Não maior que.

Tabela 3.11: Operadores relacionais.

Operador	Descrição
AND ou &	E binário para dois operandos.
OR ou |	OU binário para dois operandos.
^	OU exclusivo para dois operandos.

Tabela 3.12: Operadores lógicos/binários.

Oracle

O Oracle trabalha com os seguintes operadores e símbolos:

Operador	Descrição	Exemplo	Resultado
+	Adição.	2 + 3	5
-	Subtração.	2 - 3	-1
*	Multiplicação.	2 * 3	6
/	Divisão.	4 / 2	2
**	Exponenciação.	2 ** 3	8

Tabela 3.13: Operadores aritméticos.

Operador	Descrição
=	Igual a.
<>, ^= ou !=	Diferente de.
>	Maior que.
<	Menor que.
>=	Maior ou igual que.
<=	Menor ou igual que.

Tabela 3.14: Operadores relacionais.

Operador	Descrição
AND	E
OR	OU
NOT	Negação

Tabela 3.15: Operadores lógicos/binários.

Símbolo	Descrição
( )	Delimitadores de lista. Exemplo: `Cor in ('Azul', 'Vermelho', 'Amarelo')`.
;	Final de declaração. Exemplo: `COMMIT WORK;`.
.	Separador de item. Exemplo: `Cliente.Cod _ Cliente`.
'	Delimitador de *string*. Exemplo: `'Texto'`.
:=	Atribuição de valor. Exemplo: `variavel : = 'Texto'`.
| |	Concatenação. Exemplo: `'Codigo Cliente: '| | Cliente. Cod _ Cliente`.
--	Comentário de uma linha. Exemplo: `Begin -- Inicio da execução`.
/* e */	Delimitadores de comentários abrangendo várias linhas (início e fim de comentário). Exemplo: `/*Comentário com mais de uma linha. */`.

Tabela 3.16: Outros símbolos.

PostgreSQL

O PostgreSQL trabalha com os seguintes operadores:

Operador	Descrição	Exemplo	Resultado
+	Adição.	2 + 3	5
-	Subtração.	2 - 3	-1
*	Multiplicação.	2 * 3	6
/	Divisão.	4 / 2	2
%	Módulo (resto da divisão).	5 % 4	1
^	Exponenciação.	2.0 ^ 3.0	8
|/	Raiz quadrada.	|/ 25.0	5
||/	Raiz cúbica.	||/ 27.0	3
!	Fatorial.	5 !	120
!!	Fatorial (operador pré-fixado).	!! 5	120
@	Valor absoluto.	@ -5.0	5

Tabela 3.17: Operadores aritméticos.

Operador	Descrição
=	Igual a.
<> ou !=	Diferente de.
>	Maior que.
<	Menor que.
>=	Maior ou igual a.
<=	Menor ou igual a.

Tabela 3.18: Operadores relacionais.

Operador	Descrição
`&` ou `AND`	E
`|` ou `OR`	OU
`~` ou `NOT`	Negação
`#`	OU exclusivo
`<<`	Move um bit à esquerda
`>>`	Move um bit à direita

Tabela 3.19: Operadores lógicos/binários.

MySQL

O MySQL trabalha com os seguintes operadores:

Operador	Descrição	Exemplo	Resultado
`+`	Adição.	`2 + 3`	`5`
`-`	Subtração.	`2 - 3`	`-1`
`*`	Multiplicação.	`2 * 3`	`6`
`/`	Divisão.	`4 / 2`	`2`
`MOD` ou `%`	Módulo (retorna o resto inteiro de uma divisão).	`12 MOD 5`	`2`

Tabela 3.20: Operadores aritméticos.

Operador	Descrição
`=`	Igual a.
`<>` ou `!=`	Diferente de.
`>`	Maior que.
`<`	Menor que.
`>=`	Maior ou igual a.
`<=`	Menor ou igual a.

Tabela 3.21: Operadores relacionais.

Operador	Descrição
AND ou &&	E
OR ou ||	OU
NOT ou !	Negação
XOR	OU exclusivo

Tabela 3.22: Operadores lógicos/binários.

Manipulação de tabelas

Apresentaremos, neste tópico, a sintaxe para manipulação de tabelas, incluindo criação, alteração, exclusão e acesso às informações nelas contidas.

CREATE TABLE

Observe:

```
CREATE TABLE <tabela> (
     <campo1> <tipo>,
     <campo2> <tipo>,
     .
     .
     .
)
```

A partir dessa sintaxe básica, criaremos as cinco tabelas definidas anteriormente. Os tipos de dados devem ser definidos de acordo com o banco de dados a ser utilizado. Por exemplo, um campo numérico poderá ser `Int`, `Integer`, `Number` ou `Numeric`. A seguir, o código está definido no padrão MS SQL Server. Caso você utilize outro banco de dados, consulte a referência de tipos de dados fornecida anteriormente:

```
CREATE TABLE Fornecedor (
Cod_Fornecedor Numeric(15),
Razao_Social Varchar(60),
Nome_Fantasia Varchar(60),
Endereco Varchar(60),
CEP Numeric(8),
```

```
Cidade Varchar(35),
UF Char(2),
Telefone Numeric(10)
);

CREATE TABLE Cliente (
Cod_Cliente Numeric(15),
Nome Varchar(60),
Endereco Varchar(60),
CEP Numeric(8),
Cidade Varchar(35),
UF Char(2),
Telefone Numeric(10),
Data_Nascimento Date
);

CREATE TABLE Produto (
Cod_Produto Numeric(15),
Cod_Fornecedor Numeric(15),
Descricao Varchar(60),
Valor Numeric(10,2)
);

CREATE TABLE Pedido (
Cod_Pedido Numeric(15),
Cod_Cliente Numeric(15),
Data_Pedido Date,
Data_Entrega Date,
Valor_Total Numeric(10,2)
);

CREATE TABLE Item_Pedido (
Cod_Pedido Numeric(15),
Cod_Produto Numeric(15),
Quantidade Numeric(5)
);
```

Para definir quais campos serão a chave primária da tabela, existem diferenças de sintaxe entre os bancos de dados:

- MS SQL Server e PostgreSQL:

```
CREATE TABLE Fornecedor (
Cod_Fornecedor Numeric(15) PRIMARY KEY,
Razao_Social Varchar(60),
Nome_Fantasia Varchar(60),
Endereco Varchar(60),
```

```
CEP Numeric(8),
Cidade Varchar(35),
UF Char(2),
Telefone Numeric(10)
);
```

- Oracle:

```
CREATE TABLE Fornecedor (
Cod _ Fornecedor Number(15) PRIMARY KEY,
Razao _ Social Varchar2(60),
Nome _ Fantasia Varchar2(60),
Endereco Varchar2(60),
CEP Number(8),
Cidade Varchar2(35),
UF Char(2),
Telefone Number(10)
);
```

- MySQL:

```
CREATE TABLE Fornecedor (
Cod _ Fornecedor Numeric(15),
Razao _ Social Varchar(60),
Nome _ Fantasia Varchar(60),
Endereco Varchar(60),
CEP Numeric(8),
Cidade Varchar(35),
UF Char(2),
Telefone Numeric(10),
PRIMARY KEY (Cod _ Fornecedor)
);
```

Da mesma forma, para definirmos chave estrangeira, temos:

- MS SQL Server e PostgreSQL:

```
CREATE TABLE Produto (
Cod _ Produto Numeric(15) PRIMARY KEY,
Cod _ Fornecedor Numeric(15) REFERENCES Fornecedor(Cod _ For-
necedor),
Descricao Varchar(60),
Valor Numeric(10,2)
);
```

- Oracle:

```
CREATE TABLE Produto (
Cod_Produto Number(15) PRIMARY KEY,
Cod_Fornecedor Number(15) REFERENCES Fornecedor(Cod_Fornecedor),
Descricao Varchar2(60),
Valor Number(10,2)
);
```

- MySQL:

```
CREATE TABLE Produto (
Cod_Produto Numeric(15),
Cod_Fornecedor Numeric(15),
Descricao Varchar(60),
Valor Numeric(10,2)
PRIMARY KEY (Cod_Produto),
FOREIGN KEY (Cod_Fornecedor) REFERENCES Fornecedor(Cod_Fornecedor)
);
```

ALTER TABLE

O comando `ALTER TABLE` serve para alterar propriedades da tabela, bem como os registros e campos que a compõem. Os atributos de *primary* e *foreign key*, por exemplo, poderiam ser definidos na alteração de tabela. Para exemplificar o uso, definiremos os campos de uma tabela como `NOT NULL`, isto é, de preenchimento obrigatório (não podem ficar nulos). A sintaxe básica é a seguinte:

MS SQL Server

```
ALTER TABLE <tabela>
{ [ ALTER COLUMN <nome coluna>
    { <novo tipo da coluna> [ ( precisão [ , escala ] ) ]
        [ NULL | NOT NULL ]
        | {ADD | DROP } ROWGUIDCOL }
    ]
    | ADD
        { [ <definição de coluna> ]
         | <nome coluna> AS <expressão que define valor da coluna>
        } [ ,...n ]
    | [ WITH CHECK | WITH NOCHECK ] ADD
```

```
        { <constraint da tabela> } [ ,...n ]
    | DROP
        { [ CONSTRAINT ] <nome constraint>
            | COLUMN <coluna> } [ ,...n ]
    | { [ WITH CHECK | WITH NOCHECK ] CHECK | NOCHECK }
CONSTRAINT
        { ALL | <nome constraint> [ ,...n ] }
    | { ENABLE | DISABLE } TRIGGER
        { ALL | <nome trigger> [ ,...n ] }
}
```

Vejamos algumas opções:

- `ADD`: adiciona nova coluna/campo da tabela;
- `DROP`: exclui uma coluna/campo da tabela;
- `CHANGE`: altera o nome da coluna/campo;
- `MODIFY`: altera os atributos de um coluna/campo, como, tipo, se é nulo ou não, se é chave primária ou estrangeira etc;
- `[,...n]`: indica que mais campos podem ser alterados simultaneamente, bastando, para isso, separá-los por vírgulas.

Oracle

```
ALTER TABLE <tabela>
[ADD/DROP/CHANGE/MODIFY] (<campo> [<tipo>])
[,...n];
```

Vejamos algumas opções:

- `ADD`: adiciona nova coluna/campo da tabela;
- `DROP`: exclui uma coluna/campo da tabela;
- `CHANGE`: altera o nome da coluna/campo;
- `MODIFY`: altera os atributos de uma coluna/campo, como, tipo, se é nulo ou não, se é chave primária ou estrangeira etc.

Observação: a diferença entre o MS SQL Server e o Oracle está em colocar o campo a ser alterado entre parênteses, após definir qual operação será feita com ele.

PostgreSQL

```
ALTER TABLE <tabela>
[ADD/DROP/ALTER/RENAME] COLUMN <campo> [<tipo>][SET/DROP][NOT
NULL]
[,...n];
```

- `ADD`: adiciona nova coluna/campo na tabela;
- `DROP`: exclui uma coluna/campo da tabela;
- `ALTER`: altera as propriedades de uma coluna/campo da tabela;
- `RENAME`: renomeia uma coluna/campo da tabela.

A seguir, vamos definir um campo como `Not Null`:

```
ALTER TABLE Item _ Pedido
ALTER COLUMN Quantidade SET NOT NULL;
```

MySQL

```
ALTER [ONLINE | OFFLINE] [IGNORE] TABLE <nome tabela>
      | ADD [COLUMN] <coluna> <tipo> [FIRST | AFTER <colu-
na> ]
  | ADD {INDEX|KEY} [<nome índice>] [<tipo índice>] (<coluna
do índice>,...)
  | ADD [CONSTRAINT [<símbolo>]]
           PRIMARY KEY [<tipo índice>] (<coluna do índi-
ce>,...)
  | ADD [CONSTRAINT [<símbolo>]]
        UNIQUE [INDEX|KEY] [<nome índice>] [<tipo índice>]
(<coluna do índice>,...)
  | ADD [CONSTRAINT [<símbolo>]]
           FOREIGN KEY [<nome índice>] (<coluna do índi-
ce>,...)
        <definição da referência>
  | ALTER [COLUMN] <coluna> {SET DEFAULT literal | DROP DE-
FAULT}
   | CHANGE [COLUMN] <nome antigo> <novo nome> <definição
coluna>
        [FIRST|AFTER <coluna>]
  | MODIFY [COLUMN] <coluna> <definição coluna> [FIRST | AF-
TER <coluna>]
  | DROP [COLUMN] <coluna>
  | DROP PRIMARY KEY
  | DROP {INDEX|KEY} <nome índice>
  | DROP FOREIGN KEY <símbolo foreign key>
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] <novo nome tabela>
  | ORDER BY <coluna> [, <coluna>] ...
  | CONVERT TO CHARACTER SET <nome charset> [COLLATE <nome
intercalação>]
   | [DEFAULT] CHARACTER SET <nome charset> [COLLATE <nome
intercalação>]
```

DROP TABLE

A sintaxe de exclusão de tabelas é:

```
DROP TABLE <tabela>;
```

Lembre-se de que, caso tenha de excluir uma tabela, verifique se sua chave primária é uma chave estrangeira em outra tabela. Para evitar problemas com integridade referencial, apague, primeiro, os registros que contenham essa referência para, só depois, apagar a tabela. Isso também pode ser feito definindo *constraints* que apagam esses "registros filhos" automaticamente, quando se deleta a "tabela pai".

CREATE INDEX

Os índices são parte importante na definição e criação de um banco de dados, pois por meio deles as informações são consultadas mais rapidamente nas tabelas, e isso significa sistemas e sites com boas performances.

A partir deste tópico, definiremos as sintaxes para manipulação de índices nos quatro bancos de dados, iniciando com o `CREATE INDEX`, que cria índices para tabelas.

MS SQL Server

```
CREATE [UNIQUE] [CLUSTERED|NONCLUSTERED] INDEX <nome índice>
    ON [database_name.[schema_name].|schema_name.]
        <tabela> ( <coluna> [ASC|DESC] [,...n] )
    [ INCLUDE ( <coluna> [ ,...n ] ) ]
    [ WITH (  PAD_INDEX  = {ON|OFF}
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = {ON|OFF}
  | IGNORE_DUP_KEY = {ON|OFF}
  | STATISTICS_NORECOMPUTE = {ON|OFF}
  | DROP_EXISTING = {ON|OFF}
  | ONLINE = {ON|OFF}
  | ALLOW_ROW_LOCKS = {ON|OFF}
  | ALLOW_PAGE_LOCKS = {ON|OFF} )
  ]
 [ ; ]
```

Oracle

```
CREATE [UNIQUE|BITMAP] INDEX [schema.]<nome índice>
   ON [<nome base dados>.]<tabela> [tbl_alias]
      (<coluna> [ASC | DESC])
[LOCAL STORE IN (<tablespace>)]
[NOSORT|SORT]
[REVERSE]
[COMPRESS <int>]
[NOCOMPRESS]
[COMPUTE STATISTICS]
[[NO]LOGGING]
[ONLINE]
[TABLESPACE {<tablespace>|DEFAULT}]
[PCTFREE <int>]
[PCTUSED <int>]
[INITRANS <int>]
[MAXTRANS <int>]
;
```

Veja um exemplo:

```
CREATE UNIQUE INDEX meu_indice ON meu_schema.
tabela(coluna);
```

PostgreSQL

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] name ON table [
USING method ]
    ( { column | ( expression ) } [ opclass ] [, ...] )
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace ]
    [ WHERE predicate ]
```

Veja um exemplo:

```
CREATE UNIQUE INDEX title_idx ON films (title);
```

MySQL

```
CREATE [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL] INDEX in-
dex_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option ...]

index_col_name:
```

```
    col _ name [(length)] [ASC | DESC]

index _ type:
    USING {BTREE | HASH | RTREE}

index _ option:
    KEY _ BLOCK _ SIZE value
  | index _ type
  | WITH PARSER parser _ name
  | COMMENT 'string'
```

ALTER INDEX

O ALTER INDEX é utilizado para alterar atributos dos índices.

MS SQL Server

```
    ALTER INDEX {index | ALL} ON object DISABLE [;]
        ALTER INDEX {index | ALL} ON object REORGANIZE
[PARTITION = partition _ number ]           [WITH ( LOB _ COM-
PACTION = {ON | OFF} ) ] [;]

      ALTER INDEX {index | ALL} ON object REBUILD          [
[WITH ( rebuild _ index _ option [ ,...n ] ) ]             |
[PARTITION = partition _ number                  [ WITH ( sin-
gle _ ptn _ rebuild _ index _ option                    [ ,...n
] )                ]            ]          ] [;]
       ALTER INDEX {index | ALL} ON object SET ( set _ in-
dex _ option [ ,...n ] )
```

Oracle

```
ALTER INDEX [<schema>.]<nome índice>  [ENABLE]
                                      [DISABLE]
 [COALESCE]
   [UNUSABLE]
 [RENAME TO <novo nome do índice>]
 [[NO]LOGGING]
 [PCTFREE <int>]
 [PCTUSED <int>]
 [INITRANS <int>]
 [MAXTRANS <int>];
```

Veja um exemplo:

```
ALTER INDEX indice01 RENAME TO indice02;
```

PostgreSQL

```
ALTER INDEX name SET TABLESPACE tablespace_name
ALTER INDEX name SET ( storage_parameter = value [, ... ]
)
ALTER INDEX name RESET ( storage_parameter [, ... ] )
```

MySQL

O MySQL não possui alteração de conteúdo de um índice, sendo possível, apenas, torná-lo ativo ou inativo. Para alterar o conteúdo de um índice, é necessário apagá-lo por meio do `DROP INDEX` e recriá-lo com o `CREATE INDEX`.

A sintaxe para a alteração do status de um índice é:

```
ALTER INDEX <nome índice> [ ACTIVE | INACTIVE ];
```

Veja um exemplo:

```
ALTER INDEX meu_indice INACTIVE;
```

DROP INDEX

Para excluir um índice, utilizamos o `DROP INDEX`, que possui parâmetros diferentes para cada banco de dados:

MS SQL Server

```
DROP INDEX index ON [database_name.[schema_name].
|schema_name.]<tabela>
     [WITH ([MAXDOP = max_degree_of_parallelism]
         [ONLINE = { ON | OFF }
       [MOVE TO { <partição ou schema> (<coluna>)
  ["default"] [ ,...n ] ) ];
```

Oracle

```
DROP INDEX [schema.]index [FORCE];
```

PostgreSQL

```
DROP INDEX [ IF EXISTS ] name [,...] [ CASCADE | RESTRICT ];
```

MySQL

```
DROP [ONLINE|OFFLINE] INDEX <nome índice> ON <tabela>;
```

INSERT INTO

Este comando permite a inclusão de informações em uma tabela por meio da atribuição de dados aos campos dela:

```
INSERT INTO <tabela> (campo1, campo2, ...)
VALUES (valor1, valor2, ...);
```

Agora, vamos inserir dados nas tabelas previamente criadas. Será dado o exemplo para a inclusão de um registro, e você poderá incluir outros, tomando o cuidado de seguir as orientações indicadas.

Incluiremos um registro para cada tabela. Como as tabelas Fornecedores e Clientes não possuem chaves estrangeiras (registros de outras tabelas) e possuem referência em outras tabelas (como produtos, pedidos e itens de pedido), estes serão os primeiros registros a serem criados:

```
INSERT INTO Fornecedor (
Cod_Fornecedor,
Razao_Social,
Nome_Fantasia,
Endereco,
Cep,
Cidade,
Uf,
Telefone
)
VALUES (
1,
'Empresa A e Cia Ltda',
'Empresa A',
'Rua X, 100',
10001001,
'São Paulo',
'SP',
1155550000
);

INSERT INTO Cliente (
Cod_Cliente,
Nome,
Endereco,
Cep,
Cidade,
Uf,
```

```
Telefone,
Data _ Nascimento
)
VALUES(
2,
'Jose da Silva',
'Rua Y, 500',
01010001,
'São Paulo',
'SP',
1155550101,
to _ date('01/01/1970','DD/MM/YYYY')
);
```

Agora, criaremos um produto (que pertence ao fornecedor "Empresa A" e, com isso, é possível gerar um pedido para o cliente "José da Silva" desse produto.

```
INSERT INTO Produto (
Cod _ Produto,
Cod _ Fornecedor,
Descricao,
Valor
(
VALUES(
3,
1,
'Produto 1',
1000
);

INSERT INTO Pedido (
Cod _ Pedido,
Cod _ Cliente,
Data _ Pedido,
Data _ Entrega,
Valor _ Total
)
VALUES(
4,
2,
GETDATE,
10000
);

CREATE TABLE Item _ Pedido (
```

```
Cod _ Pedido,
Cod _ Produto,
Quantidade
)
VALUES(
4,
3,
10
);
```

SELECT

O comando `SELECT` é um dos principais da linguagem SQL e retorna as informações das tabelas de um banco de dados, bem como os dados de um sistema.

```
SELECT * FROM <tabela>;
```

Se efetuarmos esse comando para a tabela Produto, teremos:

```
SELECT * FROM Produto;
COD _ PRODUTO  | COD _ FORNECEDOR  | DESCRICAO | VALOR
-------------+------------------+-----------+-------
          3   |                1  | Produto 1 |  1000
```

SELECT INTO

O comando `SELECT INTO` copia as informações de uma tabela para outra, conforme a sintaxe seguinte:

```
SELECT <campo1>, <campo2>, <campo3>
INTO <tabela2>
FROM <tabela1>;
```

Ou:

```
SELECT *
INTO <tabela2>
FROM <tabela1>;
```

ALIASES

Um comando `ALIAS` pode ser considerado um "apelido", com o qual uma tabela será referenciada em uma *query*. Esse apelido pode ser utilizado para distinguir campos entre tabelas, e até para seleções que utilizam a mesma tabela mais de uma vez (recursivas):

```
SELECT <alias1>.<campo>, <alias2>.<campo>
FROM <tabela1> AS <alias1>, <tabela2> AS <alias2>;
```

Como exemplo, atribuiremos `aliases` para as tabelas Produto e Fornecedor:

```
SELECT A.Descricao, B.Nome _ Fantasia
FROM Produto AS A, Fornecedor AS B
WHERE A.Cod _ Fornecedor = B.Cod _ Fornecedor;

DESCRICAO | NOME _ FANTASIA
----------+----------------------------------------
Produto 1 | Empresa A
```

SELECT DISTINCT

O `DISTINCT` é utilizado com o `SELECT` para retornar um conjunto de dados relacionados com entradas distintas para uma determinada coluna de uma tabela:

```
SELECT DISTINCT <campo1>, <campo2>
FROM <tabela>;
```

WHERE

O comando `WHERE` serve para especificar condições/critérios de seleção, funcionando como um filtro para retornar apenas as informações desejadas e específicas de uma tabela:

```
SELECT * FROM <tabela>
WHERE <campo> = <valor>;
```

Por meio desse exemplo, selecionaremos alguns registros de duas tabelas, relacionando-as entre si. No caso, o `SELECT` retornará os dados do `Produto 1`, mas, ao invés de retornar o código do fornecedor, trará o nome fantasia:

```
SELECT Prod.Cod _ Produto,
       Forn.Nome _ Fantasia,
       Prod.Descricao,
       Prod.Valor
FROM Produto Prod, Fornecedor Forn
WHERE Prod.Cod _ Fornecedor = Forn.Cod _ Fornecedor;

COD _ PRODUTO | NOME _ FANTASIA  | DESCRICAO | VALOR
------------+-----------------+-----------+--------
          3   | Empresa A       | Produto 1 |  1000
```

LIKE

A cláusula `LIKE` é usada como condição no comando `WHERE` para especificar um critério baseado em um conjunto de caracteres que pode ocorrer em um campo (que possui esse conjunto de caracteres como parte do seu conteúdo):

```
SELECT * FROM <tabela>
WHERE <campo> LIKE %<valor>%;
```

Utilizando as tabelas do nosso modelo, temos:

```
SELECT *
FROM Produto
WHERE Descricao LIKE "%Produto%";

COD _ PRODUTO | COD _ FORNECEDOR  | DESCRICAO | VALOR
------------+------------------+-----------+--------
           3  | Produto 1        |          1 |  1000
```

Na *query* anterior, selecionamos todos os produtos cuja descrição conteve a *string* `Produto`, independente de sua posição no conteúdo do campo `DESCRICAO`. Se fossemos selecionar apenas as descrições que começam com `Produto`, bastaria utilizar o `%` no final da *string* `Produto%`. O mesmo aconteceria para a palavra `Produto` no fim da descrição `%Produto`. O símbolo `%` representa o restante do conteúdo do campo.

UPDATE

O `UPDATE` é utilizado para atualizar/alterar os dados contidos em uma tabela, de acordo com as cláusulas definidas no `WHERE`:

```
UPDATE <tabela>
SET <campo> = <valor>
WHERE <condição>;
```

Como exemplo prático, vamos atualizar o valor de `Produto 1`:

```
UPDATE Produto
SET Valor = 1200
WHERE Cod _ Produto = 3;
```

No caso apresentado, o valor de `Produto 1`, que era de R$ 10,00 (1000, sendo duas casas decimais), passa a ser de R$ 12,00 (1200, duas casas decimais).

DELETE

O `DELETE` é utilizado para excluir registros de uma tabela, de acordo com as cláusulas definidas no `WHERE`:

```
DELETE FROM <tabela>
WHERE <condição>;
```

Como exemplo, excluiremos os itens do quarto pedido:

```
DELETE FROM Item _ Pedido
WHERE Cod _ Pedido = 4;
```

ORDER BY

A instrução `ORDER BY` ordena um conjunto de dados retornados por um `SELECT` em função de um determinado campo especificado na *query*:

```
SELECT <campo1>, <campo2>
FROM <tabela>
ORDER BY <campo1>;
```

No exemplo a seguir, selecionaremos os produtos ordenados por descrição:

```
SELECT Cod _ Produto, Cod _ Fornecedor, Descricao, Valor
FROM Produto
ORDER BY Descricao;
```

OR e AND

As palavras-chave `OR` e `AND` são utilizadas para combinar as cláusulas presentes no `WHERE`, permitindo retornar os dados baseados nessa combinação de condições:

```
SELECT <campo1>
FROM <tabela>
WHERE <condição1> AND <condição2>; → ambas verdadeiras.
```

Ou:

```
SELECT <campo1>
FROM <tabela>
WHERE <condição1> OR <condição2>; → uma ou outra verdadeira.
```

Como exemplo, selecionaremos produtos (com o nome do fornecedor) com valores superiores a R$ 10,00:

```
SELECT Prod.Cod _ Produto,
       Forn.Nome _ Fantasia,
       Prod.Descricao,
       Prod.Valor
FROM Produto Prod, Fornecedor Forn
WHERE Prod.Cod _ Fornecedor = Forn.Cod _ Fornecedor AND
Prod.Valor > 1000;
```

IN

A cláusula `IN` permite que se especifiquem dados finitos como critério de busca em uma condição `WHERE`:

```
SELECT * FROM <tabela>
WHERE <campo1> IN (<valor1>, <valor2>, <valor3>);
```

A seguir, um exemplo que retornará apenas os clientes cujos códigos sejam 1, 2 ou 3:

```
SELECT * FROM Cliente
WHERE Cod_Cliente IN (1, 2, 3);
```

BETWEEN

O conjunto de palavras-chave BETWEEN/AND define um intervalo de dados delimitados por dois valores:

```
SELECT *
FROM <tabela>
WHERE <campo> BETWEEN <valor1> AND <valor2>;
```

Neste exemplo, selecionamos os pedidos cujos valores totais estejam entre R$ 100,00 e R$ 200,00:

```
SELECT *
FROM Pedido
WHERE Valor_Total BETWEEN 10000 AND 20000;
```

COUNT

A função de agrupamento COUNT é utilizada para contar o número de linhas retornadas por um SELECT em uma tabela, de acordo com as condições especificadas (ou não) na cláusula WHERE:

```
SELECT COUNT(1)
FROM <tabela>;
```

Como exemplo, vamos contar quantos itens possui um pedido:

```
SELECT COUNT(1) AS Qtde
FROM Item_Pedidos
WHERE Cod_Pedido = 4;

Qtde
-----
1
```

MAX

A função de agrupamento `MAX` retorna o valor mais alto (máximo) de uma determinada coluna:

```
SELECT MAX(<campo>)
FROM <tabela>;
```

MIN

A função de agrupamento `MIN` retorna o valor mais baixo (mínimo) de uma determinada coluna:

```
SELECT MIN(<campo>)
FROM <tabela>;
```

AVG

A função de agrupamento `AVG` retorna o valor médio entre todos os valores de um determinado campo de uma tabela:

```
SELECT AVG(<campo>)
FROM <tabela>;
```

SUM

A função de agrupamento `SUM` efetua a somatória dos valores de uma determinada coluna de uma tabela:

```
SELECT SUM(<campo>)
FROM <tabela>;
```

GROUP BY

A instrução `GROUP BY` é utilizada em funções de agrupamento, como a `SUM`, para fornecer meios de agrupar resultados em função de determinadas colunas:

```
SELECT SUM(<campo1>), <campo2>
FROM <tabela>
GROUP BY <campo2>;
```

HAVING

A cláusula `HAVING` é usada para restringir condicionalmente o retorno de uma instrução SQL por meio de uma função agregada utilizada na lista de colunas do `SELECT`:

```
SELECT <campo1>, SUM(<campo2>)
FROM <tabela>
GROUP BY <campo1>
HAVING SUM(<campo2>) > 100;
```

JOIN

A cláusula `JOIN` é utilizada sempre que é necessário selecionar dados de duas ou mais tabelas:

```
SELECT A.<campo1>, B.<campo2>, SUM(B.<campo3>) AS Soma
FROM <tabela1> AS A JOIN <tabela2> AS B
ON A.<campo> = B.<campo>
GROUP BY A.<campo1>, B.<campo2>;
```

UNION

A cláusula `UNION` é utilizada sempre que é preciso retornar o resultado de vários `SELECT` em um único conjunto de registros, desde que estes `SELECT` contenham as mesmas colunas:

```
SELECT A.<campo> AS Campo
FROM <tabela1> AS A
UNION
SELECT B.<campo> AS Campo
FROM <tabela2> AS B;
```

No exemplo apresentado, os resultados dos `SELECT` das tabelas `<tabela1>` e `<tabela2>` estarão unidos em um único resultado, visto que ambos possuem a coluna `Campo`.

INTERSECT

Diferente da cláusula `UNION`, a `INTERSECT` faz a intersecção de dois ou mais `SELECT`, isto é, retorna apenas os resultados comuns às consultas efetuadas:

```
SELECT A.<campo> AS Campo
FROM <tabela1> AS A
INTERSECT
SELECT B.<campo> AS Campo
FROM <tabela2> AS B;
```

Estruturas de controle

MS SQL Server

A seguir, a sintaxe das principais estruturas de controle do MS SQL Server. Lembre-se, sempre, de que a sintaxe entre colchetes ([...]) é opcional; os colchetes não devem ser digitados.

IF

A instrução `IF` permite executar comandos de acordo com determinadas condições:

```
IF <expressão booleana>
     { <enunciado, comandos, instruções> }
[ ELSE
     { <enunciado, comandos, instruções> } ]
```

WHILE

```
WHILE <expressão booleana>
     { <comando> }
     [ BREAK ]
     { <comando2> }
     [ CONTINUE ]
     { <comando3> }
```

TRY ... CATCH

```
BEGIN TRY
     {<enunciado>}
END TRY
BEGIN CATCH
          [{<enunciado>}]
END CATCH
[;]
```

CASE

```
CASE <expressão>
     WHEN <expressão> THEN <resultado da expressão>
     [ ...n ]
```

```
     [
    ELSE <expressão>
     ]
END
```

Ou:

```
CASE
     WHEN <expressão> THEN <resultado da expressão>
     [ ...n ]
     [
    ELSE <expressão>
     ]
END
```

Oracle

As estruturas de controle servem tanto para prover mecanismos de decisão como para controlar a execução de um código. A seguir, veremos a sintaxe das principais estruturas de controle do Oracle.

IF

- IF-THEN:

```
IF  (<expressão>) THEN
  <comando>;
END IF;
```

- IF-THEN-ELSE:

```
IF (<expressão1>) THEN
  <comando1>;
ELSE
  <comando1>;
END IF;
```

- IF-THEN-ELSIF-ELSE:

```
IF (<expressão1>) THEN
  <comando1>;
ELSIF <expressão2> THEN
  <comando2>
ELSE
  <comando3>;
END IF;
```

LOOP

O LOOP, assim como o WHILE e o FOR, são estruturas de repetição que permitem a execução de um mesmo comando várias vezes, sem a necessidade de reescrevê-los ou de fixar quantas vezes uma operação deve ser repetida:

```
<expressão1>;
LOOP
  <comando1>;
  EXIT WHEN <expressão2>;
END LOOP;
```

WHILE

```
<expressão1>;
WHILE (expressão1) LOOP
  <comando1>;
END LOOP;
```

FOR

```
FOR <variável> in <início>..<fim> LOOP
  <comando>;
END LOOP;
```

PostgreSQL

A seguir, as estruturas de controle do PostgreSQL.

RETURN

A instrução RETURN, como o próprio nome diz, retorna um valor que pode ser, por exemplo, o resultado de uma função. Veja, a seguir, o conteúdo de uma função (sem sua declaração), que consiste em retornar a soma de dois valores:

```
v_soma:= v_valor1 + v_valor2;
Return v_soma;
```

IF

```
IF ... THEN

IF ... THEN ... ELSE

IF ... THEN ... ELSE IF

IF ... THEN ... ELSIF ... THEN ... ELSE
```

Vejamos um exemplo:

```
IF (<expressão ou condição>) THEN
   <comandos>;
ELSIF (<expressão ou condição>) THEN
     <comandos>;
ELSE
     <comandos>;
END IF;
```

LOOP

```
LOOP
    <instruções>;
END LOOP;
```

Vejamos um exemplo:

```
LOOP
IF v_count < 100 THEN
             EXIT; -- abandona o loop
     ELSE
       v_count:= v_count + 1;
    END IF;
END LOOP;
```

WHILE

```
WHILE <expressão> LOOP
    <instruções>;
END LOOP;
```

Vejamos um exemplo:

```
WHILE v_count < 100 LOOP
    v_count:= v_count + 1;
END LOOP;
```

FOR

```
FOR <variável ou campo> IN [REVERSE] <valor1> .. <valor2>
LOOP
    <instruções>;
END LOOP;
```

Vejamos um exemplo:

```
FOR v_numero IN 1..10 LOOP
```

```
    v_numero:= v_numero + 1;
    RAISE NOTICE ''numero: %'', v_numero;
END LOOP;
```

EXIT

O comando `EXIT` abandona o *loop* (ou qualquer outra estrutura de repetição) em que ele foi declarado:

```
EXIT [<nível do loop>] [WHEN <expressão>];
```

O `<nível do loop>` é opcional e refere-se ao ponto no qual a execução deve continuar, fora do *loop* em que estava. Se ele não for especificado, por padrão a execução das instruções continuará a partir da primeira linha fora do *loop*. Vejamos um exemplo:

```
v_count
LOOP
     <instruções>
.
.
.
    EXIT WHEN v_count > 100;
END LOOP;
```

MySQL

As estruturas de controle do MySQL são as seguintes.

IF

```
IF <condição1> THEN <lista de instruções ou operações>
     [ELSEIF <condição2> THEN <lista de instruções ou ope-
rações>
    [ELSE <lista de instruções ou operações>]
END IF
```

LOOP

```
[<rótulo>:]
LOOP
    <lista de instruções ou operações>
END LOOP [<rótulo>]
```

REPEAT

```
[<rótulo>:]
REPEAT
```

```
    <lista de instruções ou comandos>
UNTIL <condição para sair do repeat>
END REPEAT [<rótulo>]
```

Vejamos um exemplo:

```
SET @x = 0;
REPEAT SET @x = @x + 1;
UNTIL @x > p1 END REPEAT;
```

WHILE

```
[<rótulo>:]
WHILE <condição1> DO
    <lista de instruções>
END WHILE [<rótulo>]
```

CASE

```
CASE <valor>
WHEN [<valor para comparação>] THEN <resultado1>
[WHEN [<valor para comparação>] THEN <resultado2>]
[ELSE <resultado3>]
END

CASE
WHEN [<condição>] THEN <resultado>
[WHEN [<condição>] THEN <resultado>]
[ELSE <resultado>]
END
```

LEAVE

Este comando é utilizado para sair de uma estrutura de controle definida com algum rótulo (como visto nos exemplos anteriores, como o `LOOP`, `REPEAT` e `WHILE`):

```
LEAVE <rótulo>
```

ITERATE

`ITERATE` pode aparecer dentro das estruturas `LOOP`, `REPEAT` e `WHILE` e indica que o *loop* deve ser repetido naquele ponto:

```
ITERATE <rótulo>
```

Vejamos um exemplo:

```
rotulo1: LOOP
  SET p1 = p1 + 1;
  IF p1 < 10 THEN ITERATE rotulo1; END IF;
  LEAVE label1;
END LOOP label1;
SET @x = p1;
```

IF()

```
IF (<expressão1>,<retorno1>,<retorno2>)
```

A <expressão1>, quando verdadeira, faz o IF retornar o valor de <retorno1>. Caso contrário, temos <retorno2>. Vejamos um exemplo:

```
SELECT IF(1>10,'verdadeiro','falso')
```

IFNULL()

```
IFNULL(<retorno1>,<retorno2>)
```

Se <retorno1> não é NULL, a função volta <retorno1>. Caso contrário, devolve <retorno2>. Vejamos uns exemplos:

- `SELECT IFNULL(NULL,1);`
- `SELECT IFNULL(1,2);`

NULLIF()

```
NULLIF(<expressão1>,<expressão2>)
```

NULLIF retorna <expressão1> caso ela seja igual a <expressão2>. Do contrário, retorna NULL.

Sequences

As *sequences*, ou seqüências, são estruturas presentes apenas no Oracle, que armazenam um determinado valor numérico como se fosse um contador que gera números seqüenciais e únicos. São muito utilizadas para atribuir valores às chaves primárias, já que estas não podem se repetir.

CREATE SEQUENCE

```
CREATE SEQUENCE [<nome banco de dados>.]<nome sequence>
<opções>

Opções:
      INCREMENT BY <número inteiro>
      START WITH <número inteiro>
      MAXVALUE <número inteiro> | NOMAXVALUE
      MINVALUE <número inteiro> | NOMINVALUE
      CYCLE | NOCYCLE
      CACHE int | NOCACHE
      ORDER | NOORDER
```

ALTER SEQUENCE

```
ALTER SEQUENCE <nome banco de dados>.]<nome sequence> [op-
ções]

Opções:
      INCREMENT BY <número inteiro>
      MAXVALUE <número inteiro> | NOMAXVALUE
      MINVALUE <número inteiro> | NOMINVALUE
      CYCLE | NOCYCLE
      CACHE <número inteiro> | NOCACHE
      ORDER | NOORDER
```

DROP SEQUENCE

```
  DROP SEQUENCE [<nome banco de dados>.]<nome sequence>
```

Como utilizar *sequences*

Para acessar o valor corrente da *sequence*, digite:

```
SELECT <nome sequence>.CURRVAL FROM Dual;
```

Para acessar um novo valor (e alterar o valor corrente), digite:

```
SELECT <nome sequence>.NEXTVAL FROM Dual;
```

Observação: `Dual` é uma tabela de sistema do Oracle que possui apenas um registro com várias colunas contendo informações de sistema, como os valores das *sequences*, data de sistema, entre outras. Todas as informações solicitadas no Oracle, quando referentes a dados de sistema, devem ser chamadas dessa tabela.

Declaração de variáveis

As variáveis são parte importantíssima na implementação de programas em SQL, visto que elas recebem informações, armazenam dados temporariamente, enquanto estes são manipulados, e retornam resultados para o usuário, ou os gravam no banco de dados.

A seguir, conheceremos a sintaxe para declaração de variáveis nos quatro bancos de dados.

MS SQL Server

Declaração de variável: `DECLARE @variavel CHAR(20);`
Atribuição de valor: `SET @variavel = 'Texto';`
Seleção de variável: `SELECT @variavel;`

Oracle

Declaração de variável: `DECLARE variavel VARCHAR(20);`
Atribuição de valor: `variavel:= 'Texto';`
Seleção de variável: `SELECT variavel;`

PostgreSQL

Declaração de variável: `DECLARE variavel VARCHAR(10);`
Atribuição de valor: `variavel:= 'Texto';`
Seleção de variável: `SELECT variavel;`

MySQL

Declaração de variável: `DECLARE variavel INTEGER;`
Atribuição de valor: `SET variavel = 100;`
Seleção de variável: `SELECT variavel;`

Capítulo 4

Funções, procedimentos e *triggers*

Além dos tipos de dados, as funções também apresentam diversas diferenças entre os bancos de dados. Apresentaremos aqui, dois tipos de funções: as que já pertencem à linguagem SQL, que chamaremos de "Funções da Linguagem SQL" e as que podemos implementar, que chamaremos de "Funções de Usuário", apenas em caráter de diferenciação.

Na seqüência, apresentaremos a sintaxe básica para a criação de *triggers* em cada banco de dados.

Funções da linguagem SQL

As funções que conheceremos a seguir são nativas dos bancos de dados, isto é, não necessitam ser criadas e podem ser chamadas tanto nas *queries* de consulta ao banco de dados quanto em procedimentos, funções e *triggers*.

Funções para *strings* e caracteres

No MS SQL Server seriam as seguintes:

Função	Descrição
`ASCII(string)`	Retorna o valor da *string* em ASCII. Um exemplo seria `SELECT ASCII('t')` retorna '116'.
`CHAR(integer)`	Retorna o caractere do código ASCII correspondente ao número informado. Um exemplo seria `SELECT CHR(116)` retornar 't'.
`LEN(string)`	Retorna o comprimento de uma expressão em número de caracteres. Um exemplo seria `SELECT LEN('SQL')` retornar '3'.
`LOWER(string)`	Converte uma *string* escrita em maiúsculas para minúsculas. Um exemplo seria `SELECT LOWER(SQL SERVER)` retornar 'sql server'.
`LTRIM(string)`	Remove os espaços em branco à esquerda. Um exemplo seria `SELECT LTRIM('  SQL')` retornar 'SQL'.
`PATINDEX(string, expressão)`	Devolve a posição de uma *string* dentro de um texto. Caso não encontre, retorna zero. Um exemplo seria `SELECT PATINDEX('b','abcabc')` retornar '2'.

`REPLICATE(string, n)`	Repete n vezes um caractere especificado. Um exemplo seria `SELECT REPLICATE('abc', 2)` retornar `'abcabc'`.
`REVERSE(string)`	Retorna o inverso de uma expressão. Um exemplo seria `SELECT REVERSE('SQL')` retornar `'LQS'`.
`RTRIM (string)`	Remove os espaços em branco à direita de uma *string*. Um exemplo seria `SELECT LTRIM('SQL ')` retornar `'SQL'`.
`SPACE(integer)`	Retorna o número de espaços em branco informados no parâmetro. Um exemplo seria `SELECT SPACE(3)` retornar `'  '`.
`STUFF(string_texto, X, Y, texto_a_inserir)`	Apaga da *string* `"texto"` os Y, caracteres a partir da posição X, e os substitui por `"texto_a_inserir"`. Um exemplo seria `SELECT STUFF('abcdef', 2, 3, 'ijk')` retornar `'aijkef'`.
`SUBSTRING(string texto, posicao_inicial, tamanho)`	Retorna uma *string* com o comprimento definido em `"tamanho"` extraída da *string* `"texto"`, a partir da `"posicao_inicial"`. Um exemplo seria `SELECT SUBSTRING('abcdef',2,3)` retornar `'bcd'`.
`UPPER(string)`	Retorna *string* com letras maiúsculas. Um exemplo seria `SELECT UPPER('sql')` retorna `'SQL'`.

Tabela 4.1.

Por sua vez, no Oracle teríamos:

Função	Descrição
`ASCII(caractere)`	Retorna o código ASCII de um determinado caractere. Um exemplo seria `SELECT ASCII('t')` retornar `'116'`.
`CHR(código ASCII).`	Retorna o caractere a partir de um código ASCII. Um exemplo seria `SELECT CHR(116)` retornar `'t'`.
`CONCAT(string1, string2)`	Concatena duas *strings*. Um exemplo seria `SELECT CONCAT('ab', 'cd')` retornar `'abcd'`.
`INITCAP(string)`	Retorna *string* com iniciais maiúsculas. Um exemplo seria `SELECT INITCAP('ORACLE database')` retornar `'Oracle Database'`.

`INSTR(string1, string2 [, posição inicial [, ocorrência ] ] )`	Retorna a posição de uma *string* dentro de outra *string*, podendo ser definida qual ocorrência deve ser apontada. Um exemplo seria `SELECT INSTR('abcabc', 'a', 1, 1)` retornar `1` (primeira ocorrência). Já `INSTR('abcabc', 'a', 1, 2)` retornaria `'4'` (segunda ocorrência).
`LENGTH(string)`	Retorna o comprimento de uma *string*. Um exemplo seria `SELECT LENGTH('abc')` retornar `'3'`.
`LOWER(string)`	Converte *string* para letras minúsculas. Um exemplo seria `SELECT LOWER('ORACLE Database 123')` retornar `'oracle database 123'`.
`LPAD(string, tamanho [, string de preenchimento] )`	Esta função preenche uma *string* à esquerda com o caractere especificado até completar o tamanho definido no segundo parâmetro. Um exemplo seria `SELECT LPAD('oracle', 10, '0')` retornar `'0000oracle'`.
`LTRIM (string, caractere a retirar)`	Esta função retira o caractere especificado à esquerda. Um exemplo seria `SELECT LTRIM('000oracle', '0')` retornar `'oracle'`.
`REPLACE( string, string a ser substituída, [ string de substituição ] )`	Substitui uma *string* por outra. Se não for informada a *string* de substituição, a *string* a ser substituída é retirada. Um exemplo seria `SELECT REPLACE('222oracle', '2', '1')` retornar `'111oracle'`.
`RPAD( string, tamanho [, string de preenchimento ] )`	Esta função preenche uma *string* à direita com o caractere especificado até completar o tamanho definido no segundo parâmetro. Um exemplo seria `SELECT RPAD('oracle', 10, '0')` retorna `'oracle0000'`.
`RTRIM (string, caractere a retirar)`	Esta função retira o caractere especificado à direita. Um exemplo seria `SELECT RTRIM('000123', '0')` retornar `'123'`.
`SUBSTR( string, posição inicial, [ comprimento ] )`	Permite extrair uma *string* dentro de outra *string*. Um exemplo seria `SELECT SUBSTR ('Oracle', 3, 2)` retornar `'ac'`.
`TRANSLATE (string, string a se substituir, string de substituição)`	Esta função substitui uma seqüência de caracteres em uma *string* por outro conjunto de caracteres. Porém, essa substituição é feita um caractere por vez. Um exemplo seria `SELECT TRANSLATE('1oracle23', '123', '456')` retornar `'4oracle56'`.

`TRIM( [ leading | trailing | both [ caractere ] from ] string )`	Esta função remove todos os caracteres especificados tanto no início quanto no fim da *string*. Exemplos: – `SELECT TRIM ('  oracle  ')` retorna `'oracle'`; – `SELECT TRIM(leading '0' from '0001230')` retorna `'1230'`; – `SELECT TRIM(trailing '1' from '1Oracle1')` retorna `'1Oracle'`; – `SELECT TRIM(both '1' from '123Oracle11')` retorna `'23Oracle'`.
`UPPER(string)`	Esta função converte todas as letras de uma *string* em caixa alta. Um exemplo seria `SELECT UPPER('oracle 123')` retornar `'ORACLE 123'`.
`VSIZE(string)`	Esta função retorna o número de bytes de uma expressão. Exemplos: – `SELECT VSIZE('Oracle')` retorna `'6'`; – `SELECT VSIZE('Oracle ')` retorna `'7'`; – `SELECT VSIZE(null)` retorna `'<null>'`; – `SELECT VSIZE('')` retorna `'<null>'`; – `SELECT VSIZE(' ')` retorna `'1'`.

Tabela 4.2.

Em PostgreSQL, temos:

Função	Descrição
`ASCII(text)`	Esta função retorna o código ASCII do caractere informado. Exemplo: `SELECT ASCII ('x')` retorna `'120'`.
`BIT _ LENGTH(string)`	Esta função retorna o número de bits de uma *string*. Exemplo: `SELECT BIT _ LENGTH('jose')` retorna `'32'`.
`BTRIM(string text, trim text)`	Esta função remove uma *substring* informada de uma *string*, tanto de seu início quanto de seu fim. Exemplo: `SELECT BTRIM('xyxabcdyyx','xy')` retorna `'abcd'`.
`CHAR _ LENGTH(string)` ou `CHARACTER _ LENGTH(string)`	Esta função retorna o número de caracteres de uma *string*. Exemplo: `SELECT CHAR _ LENGTH('jose')` retorna `'4'`.
`CHR(integer)`	Esta função retorna o caractere correspondente ao código ASCII informado. Exemplo: `SELECT CHR(65)` retorna `'A'`.

`INITCAP(text)`	Esta função retorna uma palavra com inicial em maiúscula. Exemplo: `SELECT INITCAP('abc def')` retorna `'Abc Def'`.
`LENGTH(string)`	Esta função retorna o tamanho de uma *string*. Exemplo: `SELECT LENGTH('Jose')` retorna `4`.
`LOWER(string)`	Esta função retorna uma *string* toda em caixa baixa (letras minúsculas). Exemplo: `SELECT LOWER('PostgreSQL')` retorna `'postgresql'`.
`LPAD(string text, length integer [, fill text])`	Preenche uma *string* à esquerda até um determinado comprimento com a *string* indicada (o *default* é o espaço em branco). Se a *string* ultrapassa o tamanho definido, seu conteúdo será truncado à direita. Exemplo: `SELECT LPAD('ab', 5, 'xy')` retorna `'xyxab'`.
`LTRIM(string text, text text)`	Remove a *string* ou caractere localizado no começo da *string* (à esquerda). Exemplo: `SELECT LTRIM('xxxxabcd','x')` retorna `'abcd'`.
`OCTET_LENGTH(string)`	Esta função retorna o número de bytes de uma *string*. Exemplo: `SELECT OCTET_LENGTH('jose')` retorna `'4'`.
`OVERLAY(string PLACING string FROM integer [FOR integer])`	Esta função insere uma *substring* em uma *string* na posição e extensão indicados. Exemplo: `SELECT OVERLAY('Postgrexxx' PLACING 'SQL' FROM 8 FOR 3)` retorna `'PostgreSQL'`.
`POSITION(substring IN string)`	Esta função localiza a posição de uma determinada *substring* dentro de uma *string*. Exemplo: `SELECT POSITION('SQL' IN 'PostgreSQL')` retorna `'8'`.
`REPEAT(string, integer)`	Repete um texto o número de vezes definido no segundo parâmetro. Exemplo: `SELECT REPEAT('AB', 4)` retorna `'ABABABAB'`.
`REPLACE(string, string a ser substituída, string de substituição)`	Substitui uma *substring* por outra contida em uma *string*. Exemplo: `SELECT REPLACE('abcdefabcdef', 'cd', 'XX')` retorna `'abXXefabXXef'`.
`RPAD(string, comprimento [, string de preenchimento])`	Preenche uma *string* à direita até um determinado comprimento com a *string* indicada (o *default* é o espaço em branco). Se a *string* ultrapassa o tamanho definido, seu conteúdo será truncado. Exemplo: `SELECT RPAD('ab', 5, 'xy')` retorna `'abxyx'`.

`RTRIM(string, string ou caractere a ser retirado)`	Remove a *string* ou caractere localizado no fim da *string* (à direita). Exemplo: `SELECT RTRIM('abcdxxxx','x')` retorna `'abcd'`.
`SPLIT_PART(string, delimitador, coluna)`	Divide *string* nos delimitadores, retornando a *string* que está entre eles em uma coluna numérica. Exemplo: `SELECT SPLIT_PART('abc~@~def~@~ghi','~@~',1)` retorna `'def'` em uma coluna cujo nome será `'1'`.
`STRPOS(string, substring)`	Esta função localiza uma *substring* em uma *string* e retorna sua posição inicial. Exemplo: `SELECT STRPOS('high','ig')` retorna `'2'`.
`SUBSTR(string, from [, count])`	Esta função extrai uma *substring* (definida pela posição inicial e tamanho) de uma *string*. Exemplo: `SELECT SUBSTR('alphabet', 3, 2)` retorna `'ph'`.
`SUBSTRING(string [from integer] [for integer])`	Esta função extrai um trecho de uma *string* de acordo com a posição inicial e o tamanho informados. Exemplo: `SELECT SUBSTRING('PostgreSQL' from 8 for 3)` retorna `'SQL'`.
`TRIM([leading | trailing | both] [characters] from string)`	Esta função remove *strings* ou caracteres (sendo o espaço em branco o padrão, caso não seja informado) de uma *string* de seu início, fim ou ambos. Exemplo: `SELECT TRIM(both 'x' from 'xSQLxx')` retorna `'SQL'`.
`UPPER(string)`	Esta função retorna uma *string* toda em caixa alta (letras maiúsculas). Exemplo: `SELECT UPPER('sql')` retorna `'SQL'`.

Tabela 4.3.

No MySQL, temos:

Função	Descrição
`ASCII(string)`	Esta função retorna o código ASCII do caractere informado. Exemplo: `SELECT ASCII('2')` retorna `'50'`.
`BIN(n)`	Esta função retorna uma representação em *string* de um número. Exemplo: `SELECT BIN(12)` retorna `'1100'`.
`BIT_LENGTH(string)`	Esta função retorna o tamanho de uma *string* em bits. Exemplo: `SELECT BIT_LENGTH('text')` retorna `'32'`.
`CHAR_LENGTH(string)`	Esta função retorna o número de caracteres de uma *string* (incluindo espaços em branco). Exemplo: `SELECT CHAR_LENGTH('text')` retorna `'4'`.
`CHAR(N,... [USING charset_name])`	Esta função retorna os caracteres correspondentes a cada número inteiro passado (no caso, código ASCII). Exemplo: `SELECT CHAR(77,121,83,81,76)` retorna `'MySQL'`.
`CHARACTER_LENGTH(string)`	Ver `CHAR_LENGTH()`.
`CONCAT_WS(separador,string1,string2,...)`	Retorna uma série de *strings* concatenadas com separador. Exemplo: `SELECT CONCAT_WS('-','A','B','C')` retorna `'A-B-C'`.
`CONCAT(string1, string2, ...)`	Retorna uma série de strings concatenadas. Exemplo: `SELECT CONCAT('My', 'S', 'QL')` retorna `'MySQL'`.
`ELT(n, string1, string2, string3, ...)`	Esta função retorna a string no índice (posição), indicado no primeiro parâmetro. Exemplos: `SELECT ELT(1, 'ab', 'cd', 'ef', 'gh')` retorna `'ab'` e `SELECT ELT(4, 'ab', 'cd', 'ef', 'gh')` retorna `'gh'`.
`FIELD(str, string1, string2, string3, ...)`	Esta função retorna o índice (posição) do primeiro parâmetro dentro da seqüência de ítens. Exemplo: `SELECT FIELD('cd', 'ab', 'cd', 'ef', 'gh')` retorna `'2'`.

Função	Descrição
`FIND_IN_SET(string, lista de strings)`	Esta função retorna o índice (posição) do primeiro parâmetro dentro da lista subseqüente de itens. Exemplo: `SELECT FIND_IN_SET('b','a,b,c,d')` retorna '2'.
`FORMAT(n,d)`	Esta função retorna um número formatado com o número de casas decimais informado. Exemplos: `SELECT FORMAT(122.1,4)` retorna '122.1000', e `SELECT FORMAT(122.1,0)` retorna '122'.
`HEX(numérico ou string)`	Esta função retorna a representação decimal de um número ou *string*. Exemplos: `SELECT HEX(255)` retorna 'FF' e `SELECT HEX('abc')` retorna '616263'.
`INSERT(string, posição, comprimento, substring)`	Esta função insere uma *substring* em uma posição específica até o número de caracteres informado. Exemplo: `SELECT INSERT('AAAAAAAA', 3, 4, 'BBBB')` retorna 'AABBBBAA'.
`INSTR(string, substring)`	Esta função retorna a posição da primeira ocorrência de uma *substring* em uma *string*. Exemplo: `SELECT INSTR('ABCDEDE', 'DE')` retorna '4'.
`LCASE(string)`	Ver `LOWER()`.
`LEFT(string, comprimento)`	Esta função retorna os n caracteres à esquerda da *string*. Exemplo: `SELECT LEFT('abcdefgh', 5)` retorna 'abcde'.
`LENGTH(string)`	Esta função retorna o tamanho de uma *string* em bytes. Exemplo: `SELECT LENGTH('texto')` retorna '5'.
`LOCATE(substring, string[, posição])`	Esta função retorna a posição da primeira ocorrência (ou a partir da posição definida) de uma *substring* em uma *string*. Exemplos: `SELECT LOCATE('FG','ABCDE')` retorna 0 e `SELECT LOCATE('DE', 'ABCDEDE', 5)` retorna '6'.

Função	Descrição
`LOWER(string)`	Esta função retorna uma *string* toda em letras minúsculas. Exemplo: `SELECT LOWER('MYSQL')` retorna `'mysql'`.
`LPAD(string, comprimento, string de preenchimento)`	Esta função retorna uma *string* preenchida à esquerda com o caractere ou *string* definida no segundo parâmetro. Exemplos: `SELECT LPAD('ab',4,'11')` retorna `'11ab'` e `SELECT LPAD('ab',1,'11')` retorna `'a'`.
`LTRIM(string)`	Esta função remove os espaços em branco à esquerda da *string*. Exemplo: `SELECT LTRIM('  abcdef')` retorna `'abcdef'`.
`MID(string, posição, comprimento)`	Retorna uma *substring* a partir da posição e com comprimento definidos. Funciona da mesma forma que a função `SUBSTRING`.
`OCTET_LENGTH(string)`	Ver `LENGTH()`.
`POSITION(substring IN string)`	Ver `LOCATE()`.
`QUOTE(string)`	Esta função insere aspas em uma citação que será utilizada em um enunciado de uma *query* em SQL. Exemplo: `SELECT QUOTE('Don\'t!')` retorna `'Don\'t!'`.
`REPEAT(string,n)`	Esta função retorna a repetição de uma *string* n vezes. Exemplo: `SELECT REPEAT('MySQL', 3)` retorna `'MySQLMySQLMySQL'`.
`REPLACE(string, string a ser substituída, string de substituição)`	Esta função substitui as ocorrências de uma *string* por outra, definidas como parâmetro. Exemplo: `SELECT REPLACE('www.mysql.com', 'm', '-')` retorna `'www.-ysql.co-'`.
`REVERSE(string)`	Esta função inverte a ordem dos caracteres de uma *string*. Exemplo: `SELECT REVERSE('abc')` retorna `'cba'`.

Função	Descrição
`RIGHT(string, comprimento)`	Esta função retorna os n caracteres à direita da *string*. Exemplo: `SELECT RIGHT('abcdef', 4)` retorna `'cdef'`.
`RPAD(string, comprimento, string de preenchimento)`	Esta função retorna uma *string* preenchida à direita com o caractere ou *string* definido no segundo parâmetro. Exemplos: `SELECT RPAD('ab',4,'11')` retorna `'ab11'` e `SELECT RPAD('ab',1,'11')` retorna `'1'`.
`RTRIM(string)`	Esta função remove os espaços em branco à esquerda da *string*. Exemplo: `SELECT RTRIM('abcdef  ')` retorna `'abcdef'`.
`SPACE(n)`	Esta função retorna uma *string* com um número definido de espaços em branco. Exemplo: `SELECT SPACE(6)` retorna `'  '`.
`SUBSTR(string, posição), SUBSTR(string FROM posição), SUBSTR(string, posição, comprimento), SUBSTR(string FROM posição FOR comprimento)`	Estas funções retornam uma *substring* a partir de uma *string*, considerando a posição inicial e o comprimento definido. Exemplos: – `SELECT SUBSTRING('www.mysql.com',5)` retorna `'mysql.com'`; – `SELECT SUBSTRING('www.mysql.com' FROM 4)` retorna `'.mysql.com'`; – `SELECT SUBSTRING('www.mysql.com',5,6)` retorna `'mysql.'`; – `SELECT SUBSTRING('www.mysql.com',-3)` retorna `'com'`; – `SELECT SUBSTRING('www.mysql.com', -5, 3)` retorna `'l.c'`; – `SELECT SUBSTRING('www.mysql.com' FROM -4 FOR 2)` retorna `'.c '`.
`SUBSTRING(string, posição), SUBSTRING(string FROM posição), SUBSTRING(string, posição, comprimento), SUBSTRING(string FROM posição FOR comprimento)`	Ver `SUBSTR()`.

Função	Descrição
TRIM([BOTH | LEADING | TRAILING] [substring] FROM] string), TRIM([substring FROM] string)	Esta função remove espaços em branco à direita e à esquerda da *string*. Exemplos: – SELECT TRIM(' abc ') retorna 'abc'; – SELECT TRIM(LEADING 'x' FROM 'xxxabcxxx') retorna 'abcxxx'; – SELECT TRIM(BOTH 'x' FROM 'xxxabcxxx') retorna 'abc'; – SELECT TRIM(TRAILING 'xyz' FROM 'abcxxyz') retorna 'acbx'.
UCASE(string)	Ver UPPER().
UPPER(string)	Esta função converte uma *string* para caixa alta (letras maiúsculas). Exemplo: SELECT UPPER('Sql') retorna 'SQL'.

Tabela 4.4.

Funções para data

No MS SQL Server, os formatos de data são os seguintes:

Formato	Descrição
`YY, YYYY`	Ano com dois ou quatro dígitos.
`QUALQUER, QUE`	*Quarter* (1/4 do ano, um trimestre).
`MM, M`	Mês.
`DY, Y`	Dia do ano.
`DD, D`	Dia com um ou dois dígitos.
`WK, WW`	Dia da semana.
`HH`	Hora com dois dígitos.
`MI, N`	Minutos com um ou dois dígitos.
`SS, S`	Segundo com um ou dois dígitos.
`MS`	Milissegundo.

Tabela 4.5.

Agora, as funções:

Funções	Descrição
`DATEADD (parte, número, data)`	Adiciona um valor a parte de uma data.
`DATEDIFF (parte, data inicial, data final)`	Subtrai a data inicial da data final, indicando o resultado na unidade definida em `'parte'`.
`GETDATE()`	Retorna a data atual do sistema.
`DATENAME (parte, data)`	Retorna o nome da parte de uma data.
`DATEPART (parte, data)`	Retorna a parte de uma data.

Tabela 4.6.

No caso do Oracle, as partes de uma data podem ser extraídas sempre a partir do campo Sysdate da tabela Dual. Confira a seguir os formatos:

Formato	Descrição
MM	Representação numérica do mês, de 01 a 12.
MON	Nome do mês abreviado (em inglês). Exemplo: JULY é JUL.
MONTH	Nome do mês inteiro (em inglês). Exemplo: JULY.
DD	Dia do mês. Exemplo: 20.
DY	Dia da semana abreviado. Exemplo: FRIDAY é FRI.
YYYY	Ano de quatro dígitos. Exemplo: 2008.
YY	Dois últimos dígitos do ano. Exemplo: 08.
RR	Dois últimos dígitos do ano, porém "arredondado" para um ano entre 1950 e 2049. Exemplo: 06 é considerado 2006, ao invés de 1906.
AM (or PM)	Indicador de *meridian*.
HH	Hora (entre 1 e 12).
HH24	Hora do dia (entre 0 e 23).
MI	Minuto (entre 0 e 59).
SS	Segundo (entre 0 e 59).

Tabela 4.7.

Agora, as funções:

Função	Descrição
ADD_MONTHS	Adiciona meses à data informada. Exemplo: SELECT ADD_MONTHS('01-Jan-08', 3) retorna '01-Apr-08'.
CURRENT_DATE	Seleciona a data corrente da tabela Dual. Exemplo: 01-Jan-08.
CURRENT_TIMESTAMP	Seleciona a hora corrente da tabela Dual. Exemplo: 01:00.
DBTIMEZONE	Seleciona a *time zone* da tabela Dual. Exemplo: +00:00.
LAST_DAY	A função LAST_DAY retorna o último dia de um determinado mês com base em uma data. Exemplo: SELECT LAST_DAY(to_date('2008/03/10', 'yyyy/mm/dd')) retorna Mar 31, 2008.

`MONTHS_BETWEEN(date1, date2)`	Esta função retorna a quantidade de meses entre duas datas. Exemplo: `SELECT MONTHS_BETWEEN(TO_DATE ('2008/01/01', 'yyyy/mm/dd'), TO_DATE ('2008/03/01', 'yyyy/mm/dd') )` retorna '2'.
`NEXT_DAY(date, weekday)`	Esta função retorna o primeiro dia correspondente ao dia da semana definido posterior à data informada. Exemplo: `SELECT NEXT_DAY('10-Jan-08', 'MONDAY')` retorna '14-Jan-08'.
`SYSDATE`	Esta função retorna a data e hora do sistema. Exemplo: `SELECT SYSDATE FROM DUAL;`.
`SYSTIMESTAMP`	Esta função retorna a data corrente com hora, incluindo frações de segundos e *time zone* do banco de dados local.

Tabela 4.8.

Em PostgreSQL, os formatos de data são:

Formato	Descrição
`HH`	Hora do dia (01-12).
`HH12`	Hora do dia (01-12).
`HH24`	Hora do dia (00-23).
`MI`	Minuto (00-59).
`SS`	Segundo (00-59).
`MS`	Milissegundo (000-999).
`US`	Microssegundo (000000-999999).
`SSSS`	Segundos após a meia-noite (0-86399).
`AM`, `A.M.`, `PM` ou `P.M.`	Indicador de meridiano (maiúsculas).
`am`, `a.m.`, `pm` ou `p.m.`	Indicador de meridiano (minúsculas).
`Y,YYY`	Ano (quatro dígitos) com vírgula.
`YYYY`	Ano (quatro dígitos).
`YYY`	Últimos três dígitos do ano.
`YY`	Últimos dois dígitos do ano.

`Y`	Último dígito do ano.
`BC`, `B.C.`, `AD` ou `A.D.`	Indicador de era – antes e depois de Cristo (maiúsculas).
`bc` ou `b.c.` ou `ad` ou `a.d.`	Indicador de era – antes e depois de Cristo (minúsculas).
`MONTH`	Nome completo do mês em maiúsculas (com espaços em branco à direita, caso não complete nove posições).
`Month`	Nome completo do mês com inicial maiúscula (com espaços em branco à direita caso não complete nove posições).
`month`	Nome completo do mês em minúsculas (com espaços em branco à direita caso não complete nove posições).
`MON`	Nome do mês abreviado em letras maiúsculas (três caracteres).
`Mon`	Nome do mês abreviado com inicial maiúscula (três caracteres).
`mon`	Nome do mês abreviado em letras minúsculas (três caracteres).
`MM`	Número do mês (01-12).
`DAY`	Nome do dia da semana inteiro em maiúsculas (com espaços em branco à direita caso não complete nove posições). Exemplo: `SUNDAY`.
`Day`	Nome do dia da semana inteiro com inicial em maiúscula (com espaços em branco à direita caso não complete nove posições). Exemplo: `Sunday`.
`day`	Nome do dia da semana inteiro em minúsculas (com espaços em branco à direita caso não complete nove posições). Exemplo: `sunday`.
`DY`	Nome do dia da semana abreviado em maiúsculas (três caracteres). Exemplo: `SUN`.
`Dy`	Nome do dia da semana abreviado com inicial em maiúscula (três caracteres). Exemplo: `Sun`.
`dy`	Nome do dia da semana abreviado em minúsculas (três caracteres). Exemplo: `sun`.
`DDD`	Dia do ano (001-366).
`DD`	Dia do mês (01-31).
`D`	Dia da semana (1-7).

W	Semana do mês (1-5), em que a primeira semana inicia-se no primeiro dia do mês.
WW	Semana do ano (1-53), em que a primeira semana inicia-se no primeiro dia do ano.
IW	Número da semana do ano na ISO (a primeira quinta-feira do ano localiza-se na semana um).
CC	Século (dois dígitos).
J	Data Juliana (número de dias desde 1º de Janeiro de 4712 a.c.).
Q	*Quarter* (¼ do ano = trimestre)
RM	Mês em numeral romano (I-XII; I sendo Janeiro) em maiúsculas.
rm	Mês em numeral romano (I-XII; I sendo janeiro) em minúsculas.
TZ	*Time zone* em maiúsculas.
tz	*time zone* em minúsculas.

Tabela 4.9.

Agora, as funções:

Função	Descrição
`AGE(timestamp)`	Esta função subtrai uma data da data corrente. Exemplo: supondo que `timestamp` seja igual a `'2008-01-13'`, `SELECT AGE(timestamp '2008-01-01')` retorna `12 days`.
`AGE(timestamp, timestamp)`	A mesma função `AGE`, com dois parâmetros, subtrai uma data de outra. Exemplo: `SELECT AGE('2001-04-10', timestamp '1957-06-13')` retorna `'43 years 9 months 27 days'`.
`CURRENT_DATE`	Esta função retorna a data corrente. Exemplo: `SELECT CURRENT_DATE;`.
`CURRENT_TIME`	Esta função retorna a hora corrente. Exemplo: `SELECT CURRENT_TIME;`.
`CURRENT_TIMESTAMP`	Esta função retorna a data e a hora correntes. Exemplo: `SELECT CURRENT_TIMESTAMP;`.

`DATE _ PART(text, timestamp)`	Esta função extrai uma parte definida de uma data. Exemplo: `DATE _ PART('hour', timestamp '2007-02-16 20:38:40')` retorna `20`.
`DATE _ PART(text, interval)`	Esta função retorna uma parte de um intervalo. Exemplo: `SELECT DATE _ PART('month', interval '2 years 3 months')` retorna `3`.
`DATE _ TRUNC(text, timestamp)`	Esta função trunca uma data para a precisão definida. Exemplo: `SELECT DATE _ TRUNC('hour', timestamp '2007-02-16 20:38:40')` retorna `2007-02-16 20:00:00+00`.
`EXTRACT(parte FROM timestamp)`	Esta função extrai uma parte de um `timestamp`. Exemplo: `SELECT EXTRACT(hour FROM timestamp '2007-02-16 20:38:40')` retorna `20`.
`EXTRACT(field FROM interval)`	Esta função extrai uma parte de um intervalo. Exemplo: `SELECT EXTRACT(month FROM interval '2 years 3 months')` retorna `3`.
`LOCALTIMESTAMP`	Esta função retorna a data e hora correntes e locais. Exemplo: `SELECT LOCALTIMESTAMP;`.
`NOW()`	Esta função retorna a data e hora correntes (equivalente a `CURRENT _ TIMESTAMP`).
`TIMEOFDAY()`	Esta função retorna a data e hora correntes. Exemplo: `SELECT TIMEOFDAY()` retorna o formato `Wed Feb 21 17:01:13.000126 2001 EST`.

Tabela 4.10.

Em MySQL, temos os seguintes formatos de data:

Formato	Descrição
`%a`	Dia da semana abreviado (`Sun` a `Sat`).
`%b`	Nome do mês abreviado (`Jan` a `Dec`).
`%c`	Mês numérico (0 a 12).
`%D`	Dia do mês com sufixo em inglês (0th, 1st, 2nd, 3rd, 4th etc.).
`%d`	Dia do mês (00 a 31).
`%e`	Dia do mês (0 a 31).
`%f`	Microssegundos (000000 a 999999).
`%H`	Hora (00 a 23).

`%h`	Hora (01 a 12).
`%I`	Hora (01 a 12).
`%i`	Minutos (00 a 59).
`%j`	Dia do ano (001 a 366).
`%k`	Hora (0 a 23).
`%l`	Hora (1 a 12).
`%M`	Nome do mês completo.
`%m`	Mês numérico (00 a 12).
`%p`	AM ou PM.
`%r`	Horário de 0 a 12 (hh:mm:ss) seguido de AM ou PM.
`%S`	Segundos (00 a 59).
`%s`	Segundos (00 a 59).
`%T`	Horário de 0 a 23 (hh:mm:ss).
`%U`	Semana do ano (00 a 53). Domingo é o primeiro dia da semana.
`%u`	Semana do ano (00 a 53). Segunda-feira é o primeiro dia da semana.
`%V`	Semana do ano (01 a 53). Domingo é o primeiro dia da semana e é utilizado em conjunto com o `%X`.
`%v`	Semana do ano (01 a 53), em que Segunda-feira é o primeiro dia da semana e é utilizado em conjunto com o `%x`.
`%W`	Nome completo do dia da semana.
`%w`	Dia da semana (`0`, Domingo a `6`, Sábado).
`%X`	Ano numérico de quatro dígitos para semana, em que Domingo é o primeiro dia.
`%x`	Ano numérico de quatro dígitos para semana, em que Segunda-feira é o primeiro dia.
`%Y`	Ano numérico de quatro dígitos.
`%y`	Ano numérico de dois dígitos.
`%%`	Caractere "`%`" literal.

Tabela 4.11.

Agora, as funções:

Função	Descrição
`CURDATE()`	Esta função retorna a data corrente.
`CURRENT_DATE(), CURRENT_DATE`	Ver `CURDATE()`.
`CURRENT_TIME(), CURRENT_TIME`	Ver `CURTIME()`.
`CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP`	Ver `NOW()`.
`CURTIME()`	Esta função retorna a hora corrente.
`DATE_ADD(data,INTERVAL unidade)`	Esta função adiciona um determinado intervalo a uma data. Ex: `DATE_ADD('2007-12-31 23:59:59',INTERVAL 1 SECOND)` retorna '2008-01-01 00:00:00'.
`DATE_FORMAT(data,formato)`	Esta função configura uma data conforme o formato especificado. Exemplo: `SELECT DATE_FORMAT('2008-01-12 19:01:00', '%W %M %Y')` retorna 'Saturday January 2008' e `SELECT DATE_FORMAT('2008-01-12 19:01:00', '%D %y %a %d %m %b')` retorna '12th 08 Sat 12 01 Jan'.
`DATE_SUB(data,INTERVAL unidade)`	Esta função subtrai um determinado intervalo de uma data. Ver exemplo de `DATE_ADD()`.
`DATEDIFF(expressão1,expressão2)`	Esta função retorna a subtração de duas datas. Exemplos: `SELECT DATEDIFF('2008-01-12 23:59:59','2008-01-11 23:59:59')` retorna '1' e 'SELECT DATEDIFF('2008-01-01 00:00:00','2008-01-12 00:00:00') retorna '-11'.
`DAY(data)`	Ver `DAYOFMONTH()`.
`DAYNAME(data)`	Esta função retorna o dia da semana da data informada. Exemplo: `SELECT DAYNAME('2008-01-12')` retorna 'Saturday'.
`DAYOFMONTH(data)`	Esta função retorna o dia do mês. Exemplo: `SELECT DAYOFMONTH('2008-01-12')` retorna 12.

`DAYOFWEEK(data)`	Esta função retorna o índice do dia da semana correspondente à data informada (`1` é Domingo, `2` é Segunda-feira, `7` é Sábado etc.). Exemplo: `SELECT DAYOFWEEK('2008-01-12')` retorna `'7'`.
`DAYOFYEAR(data)`	Esta função retorna o dia do ano correspondente à data informada. Exemplo: `SELECT DAYOFYEAR('2008-02-03')` retorna `'34'`.
`EXTRACT(unidade FROM data)`	Esta função extrai uma parte de uma data. Exemplo: `SELECT EXTRACT(YEAR FROM '2008-01-12')` retorna `2008`.
`FROM_DAYS(n)`	Esta função converte o número de um dia em uma data equivalente. Exemplo: `SELECT FROM_DAYS(729670)` retorna `'1997-10-08'`.
`GET_FORMAT([DATE|TIME|DATETIME], ['EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'])`	Esta função retorna uma *string* de formato de data de acordo com a localização informada. Exemplos: – `SELECT GET_FORMAT(DATE,'USA')` retorna `'%m.%d.%Y'`; – `SELECT GET_FORMAT(DATETIME,'USA')` retorna `'%Y-%m-%d %H.%i.%s'`; – `SELECT GET_FORMAT(TIME,'USA')` retorna `'%h:%i:%s %p'`.
`HOUR(hora)`	Esta função extrai a hora de uma data. Exemplo: `SELECT HOUR('10:05:03')` retorna `'10'`.
`LAST_DAY(data)`	Esta função retorna o último dia do mês da data informada. Exemplos: – `SELECT LAST_DAY('2008-02-05')` retorna `'2008-02-29'`; `SELECT LAST_DAY('2007-02-05')` retorna `'2007-02-28'`; – `SELECT LAST_DAY('2008-03-32')` retorna `NULL`.
`LOCALTIME(), LOCALTIME`	Ver `NOW()`.
`LOCALTIMESTAMP, LOCALTIMESTAMP()`	Ver `NOW()`.
`MAKEDATE(year,dayofyear)`	Esta função cria uma data a partir de um ano e do dia do ano. Exemplos: – `SELECT MAKEDATE(2008,31)` retorna `'2008-01-31'`; – `SELECT MAKEDATE(2007,365)` retorna `'2007-12-31'`; – `SELECT MAKEDATE(2008,0)` retorna `'NULL'`.

`MAKETIME(hour,minute,second)`	Cria uma hora a partir dos valores informados. Exemplo: `SELECT MAKETIME(12,25,30)` retorna `'12:25:30'`.
`MICROSECOND(expressão)`	Esta função retorna os microssegundos de uma data. Exemplos: `SELECT MICROSECOND('12:00:00.12345')` retorna `12345` e `SELECT MICROSECOND('2007-12-31 23:59:59.000010')` retorna `10`.
`MINUTE(hora)`	Esta função retorna os minutos de uma hora. Exemplo: `SELECT MINUTE('08-02-03 10:05:03');` retorna `5`.
`MONTH(data)`	Esta função retorna o mês de uma data. Exemplo: `SELECT MONTH('2008-02-03')` retorna `2`.
`MONTHNAME(data)`	Esta função retorna o nome do mês de uma data. Exemplo: `SELECT MONTHNAME('2008-02-15')` retorna `'February'`.
`NOW()`	Esta função retorna a data e a hora correntes. Exemplo: `SELECT NOW();`.
`SECOND(hora)`	Esta função retorna os segundos de uma hora (0-59). Exemplo: `SELECT SECOND('10:05:03')` retorna `3`.
`SUBTIME(expressão1,expressão2)`	Esta função subtrai horas de uma data ou de outra hora. Exemplo: `SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002')` retorna `'2007-12-30 22:58:58.999997'`.
`SYSDATE()`	Ver `NOW()`.
`TIME_FORMAT(hora, formato)`	Esta função formata uma hora. Exemplo: `SELECT TIME_FORMAT('10:00:00', '%H %k %h %I %l')` retorna `'10 10 04 04 4'`.
`TIME(expressão)`	Esta função extrai a hora (com minutos e segundos) de uma expressão de data. Exemplo: `SELECT TIME('2008-12-31 01:02:03')` retorna `'01:02:03'`.
`TIMESTAMP(expressão), TIMESTAMP(expressão1,expressão2)`	Esta função, quando utilizada com apenas um argumento, retorna a expressão em formato data/hora. Com dois argumentos, retorna, também nesse formato, a soma das duas expressões de data. Exemplos: `SELECT TIMESTAMP('2003-12-31')` retorna `'2003-12-31 00:00:00'` e `SELECT TIMESTAMP('2007-12-31 12:00:00','12:00:00')` retorna `'2008-01-01 00:00:00'`.

`TIMESTAMPADD(unidade, intervalo, expressão datetime)`	Esta função adiciona um intervalo à expressão *datetime*. Exemplos: `SELECT TIMESTAMPADD(MINUTE,1,'2008-01-02')` retorna `'2008-01-02 00:01:00'` e `SELECT TIMESTAMPADD(WEEK,1,'2008-01-02')` retorna `'2008-01-09'`.
`TIMESTAMPDIFF(unidade, expressão datetime1, expressão datetime2)`	Esta função subtrai um determinado intervalo de uma expressão *datetime* em relação a outra. Exemplos: `SELECT TIMESTAMPDIFF(MONTH,'2008-02-01','2008-05-01')` retorna `3` e `SELECT TIMESTAMPDIFF(YEAR,'2008-05-01','2007-01-01')` retorna `'-1'`.
`TO_DAYS(data)`	Esta função retorna uma determinada data convertida em dias corridos. Exemplo: `SELECT TO_DAYS('1997-10-06')` retorna `'729668'`.
`WEEKDAY(data)`	Esta função retorna o índice do dia da semana relacionado à data informada (`0` é Segunda-feira, `1` é Terça-feira...). Exemplo: `SELECT WEEKDAY('2008-01-01 22:23:00')` retorna `'1'`.
`WEEKOFYEAR(data)`	Esta função retorna a semana do calendário em que se localiza a data informada. Exemplo: `SELECT WEEKOFYEAR('2008-01-31')` retorna `'5'`.
`YEAR(data)`	Esta função retorna um ano a partir de uma determinada data. Exemplo: `SELECT YEAR('08-02-03')` retorna `'2008'`.

Tabela 4.12.

Funções matemáticas

No MS SQL Server, temos:

Função	Descrição
`ABS(n)`	Retorna o valor absoluto de `n`.
`ACOS(n)`	Retorna o arco-co-seno de `n`.
`ASIN(n)`	Retorna o arco-seno de `n`.
`ATAN(n)`	Retorna o arco-tangente de `n`.
`ATN2(x,y)`	Arco-tangente de `x/y`.

`CEILING(n)`	Retorna o limite superior de `n`.
`COS(n)`	Retorna o co-seno de `n`.
`COT(n)`	Retorna a cotangente de `n`.
`DEGREES(n)`	Converte radianos para graus.
`EXP(n)`	Retorna o exponencial de `n`.
`FLOOR(n)`	Retorna o limite inferior de `n`.
`LOG(n)`	Retorna o logaritmo natural de `n`.
`LOG10(n)`	Retorna o logaritmo base 10 de `n`.
`PI()`	Retorna o valor de PI: 3,1415926535897931.
`POWER(n,p)`	Retorna o valor `n` elevado à potência `p`.
`RADIANS(n)`	Converte grau em radiano.
`RAND(expressao)`	Retorna um número aleatório entre zero e um.
`ROUND(n, p, arredonda _ ou _ trunca)`	Esta função arredonda ou trunca um número de acordo com a precisão. Caso o terceiro parâmetro não for informado, o número é arredondado. Para truncar, usa-se `1`.
`SIGN(n)`	Retorna sinal positivo, negativo ou zero do número.
`SIN(n)`	Seno do angulo especificado.
`SQRT(n)`	Raiz quadrada de `n`.
`SQUARE(n)`	Raiz quadrada de `n`.
`TAN(n)`	Tangente de `n`.

Tabela 4.13.

No Oracle, temos:

Função	Descrição
`ABS(n)`	Retorna o valor absoluto do número `n`.
`ACOS(n)`	Retorna o arco-co-seno do número `n`.
`ASIN(n)`	Retorna o arco-seno do número `n`.
`ATAN2(x,y)`	Retorna o arco-tangente de `x` e `y`.

`ATAN(n)`	Retorna o arco-tangente de `n`.
`AVG(expressão)`	Retorna o valor médio de uma expressão.
`CEIL(n)`	Retorna o limite superior de `n`.
`COS(n)`	Retorna o co-seno do número `n`.
`COSH(n)`	Retorna o co-seno hiperbólico do número `n`.
`EXP(n)`	Eleva `n` à potência `e` (2.71828183).
`FLOOR(n)`	Retorna o limite inferior de `n`.
`GREATEST(n1[, n2, n3, ....])`	Retorna o maior valor da lista de expressões.
`LEAST(n1[, n2, n3, ....])`	Retorna o menor valor da lista de expressões.
`LN(n)`	Retorna o logaritmo natural de `n`.
`LOG(b,n)`	Retorna o logaritmo de `n` na base `b`.
`MOD(m,n)`	Retorna o resto da divisão de `m` por `n`.
`POWER(m,n)`	Retorna `m` elevado à `n` potência.
`REMAINDER(m,n)`	Ver `MOD()`.
`ROUND( data [, formato ] )`	Arredonda `n` com o número de casas decimais definido.
`ROUND( n [, decimal ] )`	Arredonda `n` com o número de casas decimais definido.
`SIGN(n)`	Retorna o sinal do número `n`, sendo os retornos possíveis `-1`, `0` e `+1`.
`SIN(n)`	Retorna o seno do número `n`.
`SINH(n)`	Retorna o seno hiperbólico do número `n`.
`SQRT(n)`	Retorna a raiz quadrada de `n`.
`TAN(n)`	Retorna a tangente do número `n`.
`TANH(n)`	Retorna a tangente hiperbólica do número `n`.
`TRUNC( data, [ formato ] )`	Trunca uma data de acordo com o formato definido.
`TRUNC( n [, decimal ] )`	Trunca `n` com o número de casas decimais definido.

Tabela 4.14.

Em PostgreSQL, temos:

Função	Descrição
`ABS(x)`	Valor absoluto. Exemplo: `ABS(-20.5)` retorna '20.5'.
`ACOS(x)`	Arco-co-seno do ângulo especificado.
`ASIN(x)`	Arco-seno do ângulo especificado.
`ATAN(x)`	Arco-tangente do ângulo especificado.
`ATAN2(x,y)`	Arco-tangente de x/y.
`CBRT(dp)`	Raiz cúbica. Exemplo: `SELECT CBRT(27.0)` retorna '3'.
`CEIL(dp ou numeric)`	Limite superior. Exemplo: `SELECT CEIL(-42.8)` retorna '-42'.
`COS(x)`	Co-seno do ângulo especificado.
`COT(x)`	Cotangente do ângulo especificado.
`DEGREES(dp)`	Transforma radiano em grau. Exemplo: `SELECT DEGREES(0.5)` retorna '28.6478897565412'.
`EXP(dp ou numeric)`	Exponencial. Exemplo: `SELECT EXP(1.0)` retorna '2.71828182845905'.
`FLOOR(dp ou numeric)`	Limite inferior. Exemplo: `SELECT FLOOR(-42.8)` retorna '-43'.
`LN(n)`	Logaritmo natural ou neperiano. Exemplo: `SELECT LN(2.0)` retorna '0.693147180559945'.
`LOG(b,x)`	Logaritmo de x com base b. Exemplo: `SELECT LOG(2.0, 64.0)` retorna '6.0000000000'.
`LOG(n)`	Logaritmo com base dez. Exemplo: `SELECT LOG(100.0)` retorna '2'.
`MOD(y,x)`	Retorna o resto da divisão de y por x. Exemplo: `SELECT MOD(9,4)` retorna '1'.
`PI()`	Retorna a constante PI. Exemplo: `SELECT PI()` retorna '3.14159265358979'.
`POW(x,e)`	Eleva um número x à e potência. Exemplo: `SELECT POW(9.0, 3.0)` retorna '729'.
`RADIANS(n)`	Transforma grau em radiano. Exemplo: `SELECT RADIANS(45.0)` retorna '0.785398163397448'.

`RANDOM()`	Retorna um valor randômico entre 0.0 e 1.0.
`ROUND(n)`	Arredonda para o número inteiro mais próximo. Exemplo: `SELECT ROUND(52.7)` retorna '53'.
`ROUND(n,s)`	Arredonda o número `n` em `s` casas decimais. Exemplo: `SELECT ROUND(80.4382, 2)` retorna '80.44'.
`SIGN(n)`	Retorna um dos sinais, dependendo do argumento (`-1`, `0`, `+1`). Exemplo: `SELECT SIGN(-8.5)` retorna '-1'.
`SIN(x)`	Seno do ângulo especificado.
`SQRT(n)`	Raiz quadrada. Exemplo: `SELECT SQRT(4.0)` retorna '2'.
`TAN(x)`	Tangente do ângulo especificado.
`TRUNC(n)`	Trunca o número sem casas decimais. Exemplo: `SELECT TRUNC(50.2)` retorna '50'.
`TRUNC(n,s)`	Trunca o número `n` em `s` casas decimais. Exemplo: `SELECT TRUNC(42.4352, 3)` retorna '42.435'.

Tabela 4.15.

No MySQL, temos:

Função	Descrição
`ABS(n)`	Retorna o valor absoluto.
`ACOS(a)`	Retorna o arco-co-seno de `a`.
`ASIN(a)`	Retorna o arco-seno de `a`.
`ATAN2(x,y), ATAN(x,y)`	Retorna o arco-tangente de `x` e `y`.
`ATAN(a)`	Retorna o arco-tangente de `a`.
`CEIL(a)`	Retorna o limite superior de `a`.
`CEILING(a)`	Ver `CEIL()`.
`CONV(a,base1, base2)`	Converte `a` da `base1` para `base2`.
`COS(a)`	Retorna o co-seno de `a`.
`COT(a)`	Retorna a cotangente de `a`.
`DEGREES(a)`	Converte radiano em grau.
`EXP(x)`	Eleva a base do logaritmo à potência de `x`.

`FLOOR(a)`	Retorna o limite inferior de `a`.
`MOD(x,y)`	Retorna o resto da divisão de `x` por `y`.
`LN(x)`	Retorna o logaritmo natural ou neperiano de `x`.
`LOG10(x)`	Retorna o logaritmo de `x` na base dez.
`LOG2(x)`	Retorna o logaritmo de `x` na base dois.
`LOG(x) e LOG (x,b)`	Retorna o logaritmo natural de `x` na base `b`.
`PI()`	Retorna o valor de PI.
`POW(x,y)`	Retorna o valor de `x` elevado à potência de `y`.
`POWER(x,y)`	Ver `POW()`.
`RADIANS(a)`	Retorna o grau convertido em radianos.
`RAND()`	Retorna um número randômico com ponto flutuante.
`ROUND(n)`	Arredonda o número passado.
`SIGN(a)`	Retorna o sinal do argumento (`-1`, `0`, `+1`).
`SIN(a)`	Retorna o seno de a.
`SQRT(a)`	Retorna a raiz quadrada de a.
`TAN(a)`	Retorna a tangente de a.
`TRUNCATE(n,d)`	Trunca o número `n` em `d` casas decimais.

Tabela 4.16.

Função de conversão

As funções de conversão visam a transformar um tipo de dado em outro, podendo ou não definir a formatação desse dado.

Em MS SQL Server, elas seriam:

Função	Descrição
`BIT_TO_NUM(n1[, n2, n3, ...])`	Converte uma seqüência de bits em um número, sendo que `n1`, `n2` etc. só podem ser `0` ou `1`.
`CAST (expressão AS tipo_dado [(comprimento)])`	Converte a expressão para o tipo de dado.

`CONVERT(tipo(tamanho), expressão, estilo)`	Converte a expressão para o tipo de dado.
`STR ( expressão [ , comprimento [ , decimal ] ] )`	Converte um número em caractere.

Tabela 4.17.

Em Oracle, temos:

Função	Descrição
`CAST ( { expr | ( subquery ) | MULTISET ( subquery ) } AS type_name )`	A função `CAST` converte um tipo em outro. Exemplo: `SELECT CAST('22-Aug-2003' AS VARCHAR2(30) ) FROM DUAL;`.
`HEXTORAW(char)`	Esta função converte um valor hexadecimal em `RAW`.
`RAWTOHEX(raw)`	Esta função converte um valor `RAW` em hexadecimal, em geral, utilizado para gravar uma informação em um campo `BLOB`.
`TO_CHAR(numeric, [formato_máscara])`	Esta função converte um número ou data em *string*. Exemplo: `SELECT TO_CHAR(100,'00999')` retorna `'00100'`.
`TO_DATE(string, [formato_data])`	Esta função converte uma *string* em data de acordo com o formato definido. Exemplo: `SELECT TO_DATE('2003/07/09', 'yyyy/mm/dd')` retorna a data `'Julho, 09, 2003'`.
`TO_NUMBER(string1, [formato_máscara])`	Esta função converte uma *string* em número. Exemplo: `SELECT TO_NUMBER('1210.73', '9999.99')` retorna `'1210.73'`.
`TO_TIMESTAMP(string1, [ formato_máscara ])`	Esta função converte uma *string* em `TIMESTAMP`. Exemplo: `SELECT TO_TIMESTAMP('2007/JAN/13 10:15:00', 'YYYY/MON/DD HH:MI:SS')` retorna `'13-JAN-07 10.15.00.000000000 AM'`.
`TO_TIMESTAMP_TZ ( string1, [ format_máscara ])`	Esta função converte uma *string* em `TIMESTAMP` com *time zone*. Exemplo: `SELECT TO_TIMESTAMP_TZ('2007/12/31 10:15:00 -8:00', 'YYYY/MM/DD HH:MI:SS TZH:TZM')` retorna `'31-DEC-07 10.15.00.000000000 AM -08:00'`.

Tabela 4.18.

Em PostgreSQL, encontramos:

Função	Descrição
`TO_CHAR(numeric, text)`	Converte número em *string*. Exemplo: `TO_CHAR(10050, '999D99')` retorna `'100,50'`.
`TO_CHAR(timestamp, text)`	Converte hora em *string*. Exemplo: `TO_CHAR(current_timestamp,'HH12:MI:SS')` retorna `'22:09:00'` (no caso, a hora corrente como *string*).
`TO_CHAR(interval, text)`	Converte **interval** em *string*. Exemplo: `TO_CHAR(interval '15h 2m 12s','HH24:MI:SS')` retorna `'15:02:12'`.
`TO_CHAR(int, text)`	Converte inteiro em *string*. Exemplo: `TO_CHAR(125, '999')` retorna `'125'`.
`TO_CHAR(double precision, text)`	Converte **real/double precision** em *string*. Exemplo: `TO_CHAR(125.8, '999D9')` retorna `'125.8'`.
`TO_DATE(text, text)`	Converte *string* em data. Exemplo: `TO_DATE('05 Dec 2000', 'DD Mon YYYY')` retorna `'05 Dec 2000'`.
`TO_TIMESTAMP(text, text)`	Converte *string* em `TIMESTAMP`. Exemplo: `TO_TIMESTAMP('05 Dec 2000', 'DD Mon YYYY')` retorna `'05 Dec 2000'`.
`TO_NUMBER(text, text)`	Converte *string* em valor numérico. Exemplo: `TO_NUMBER('12,454.80', '99G999D90')` retorna `'12,454.80'`.
`TO_HEX(integer ou bigint)`	Converte um número em sua representação hexadecimal equivalente. Exemplo: `TO_HEX(2147483647)` retorna `'7fffffff'`.

Tabela 4.19.

No MySQL, encontramos:

Função	Descrição
`CONVERT_TZ(data, timezone1, timezone2)`	Esta função converte uma data de um *time zone* para outro. Exemplo: `SELECT CONVERT_TZ('2008-01-01 12:00:00', '+00:00', '+05:00')` retorna `'2008-01-01 17:00:00'`.
`SEC_TO_TIME(seconds)`	Esta função converte segundos para o formato `'HH:MM:SS'`. Exemplo: `SELECT SEC_TO_TIME(2378)` retorna `'00:39:38'`.

`STR _ TO _ DATE(string,formato da string)`	Esta função converte uma *string* em data. Exemplos: `SELECT STR _ TO _ DATE('00/00/0000', '%m/%d/%Y')` retorna `'0000-00-00'` e `SELECT STR _ TO _ DATE('04/31/2004', '%m/%d/%Y')` retorna `'2004-04-31'`.
`TIME _ TO _ SEC(time)`	Esta função converte uma hora em segundos. Exemplo: `SELECT TIME _ TO _ SEC('22:23:00')` retorna `'80580'` e `SELECT TIME _ TO _ SEC('00:39:38')` retorna `'2378'`.

Tabela 4.20.

Funções e procedimentos de usuário

Ao contrário das funções prontas dos bancos de dados, é possível implementar funções para processos específicos, utilizando, inclusive, as funções existentes abordadas anteriormente.

Neste tópico, conheceremos a sintaxe básica de cada função e o procedimento dos bancos de dados abordados, para que seja possível diferenciá-los no momento da execução ou conversão de código.

MS SQL Server

Aqui, teríamos:

```
CREATE FUNCTION [ <nome base de dados>. ] <nome função>
( [ { @<parâmetro> [ AS ] <tipo de dado do parâmetro>
     .
.
.
  ]
)
RETURNS <tipo de dado de retorno>
    [ WITH <opções> [ ,...n ] ]
    [ AS ]
    BEGIN
        <corpo da função>
        RETURN <expressão>
    END
[ ; ]
```

Vejamos um exemplo:

```
CREATE FUNCTION RetornaTotal (@cod _ pedido int)
RETURNS int
```

```
AS
RETURN
(
    SELECT valor_total AS 'VL Total'
    FROM Pedido
    WHERE Cod_Pedido = @cod_pedido
);
GO
```

Oracle

Os procedimentos (*stored procedures*) do Oracle são os mais completos dentre todos os bancos de dados abordados. Isso porque sua sintaxe permite definir parâmetros de entrada, de saída, e entrada/saída, além de manipular cursores contendo conjuntos de dados. Já as funções (*functions*) podem ter vários parâmetros de entrada, porém só retornam um tipo de dado. Vejamos as sintaxes de ambos. Em primeiro lugar, a *stored procedure*:

```
CREATE [OR REPLACE] PROCEDURE <nome do procedimento>
    [ (<parâmetro>[, <parâmetro>]) ]
IS
    [<declaração de variáveis>]
BEGIN
    <instruções>
[EXCEPTION
    <instruções a serem executadas em caso de exceção>]
END [<nome do procedimento>];
```

Agora, a função:

```
CREATE [OR REPLACE] FUNCTION <nome da função>
    [ (<parâmetro> [, <parâmetro>]) ]
    RETURN <tipo de retorno>
IS | AS
    [<declaração de variáveis>]
BEGIN
    <instruções>
[EXCEPTION
    <instruções a serem executadas em caso de exceção>]
END [<nome da função>];
```

Os parâmetros podem ser de três tipos:

- `IN`: este tipo de parâmetro pode ser referenciado tanto em procedimentos quanto em funções, sendo que seu conteúdo não pode ser alterado pelas instruções que o utilizam;
- `OUT`: este tipo de parâmetro não pode ser referenciado em procedimentos ou funções, mas seu conteúdo pode ser por eles alterado;
- `IN OUT`: este tipo de parâmetro pode ser referenciado em procedimentos e funções, e seu conteúdo pode ser alterado e retornado após a execução das instruções.

A seguir, vejamos alguns exemplos de como criar procedimentos e funções simples:

```
CREATE OR REPLACE Function RetornaCliente
   ( p_cod_cliente IN NUMBER )
   RETURN VARCHAR2
IS
    v_Nome_Cliente VARCHAR2(60);

BEGIN
SELECT nome INTO v_Nome_Cliente
FROM Cliente
WHERE Cod_Cliente = p_cod_cliente;
RETURN v_Nome_Cliente;
END;
```

Esta função poderia ser utilizada em uma *query*, como, por exemplo:

```
SELECT Cod_Pedido, RetornaCliente(Cod_Cliente) AS Nome_
Cliente
FROM Pedido;
```

Packages

Um *package* é um "pacote" com vários procedimentos e funções relacionados entre si. Não que essa relação seja obrigatória, mas um *package* é útil justamente porque possui a função de reunir, em um único espaço, rotinas referentes a uma determinada tabela, ou a um determinado processo. Os *packages* também são úteis para organizar bibliotecas de funções utilizadas por vários sistemas. Basta, para isso, referenciar o nome do *package* e da função que se deseja usar. Veja como criar *packages* e *package bodies*:

```
CREATE [OR REPLACE] PACKAGE [<schema>.]<nome package> [AU-
THID CURRENT _ USER | AUTHID DEFINER] AS package

CREATE [OR REPLACE] PACKAGE BODY [<schema>.]<nome package>
AS <nome package body>
```

Para alterar *packages* e *package bodies*, proceda da seguinte forma:

```
ALTER PACKAGE [<schema>.]<nome package> COMPILE [DEBUG] PA-
CKAGE [REUSE SETTINGS];

ALTER PACKAGE [<schema>.]<nome package> COMPILE [DEBUG] SPE-
CIFICATION [REUSE SETTINGS];

ALTER PACKAGE [<schema>.]<nome package> COMPILE [DEBUG] BODY
[REUSE SETTINGS];
```

Enfim, exclua *packages* e *package bodies*:

```
DROP PACKAGE [BODY] [<schema>.]<nome package>;
```

PostgreSQL

O PostgreSQL não trabalha com procedimentos, mas as funções podem fazer o mesmo papel. Seus parâmetros, no entanto, são apenas de entrada, e não podem ser modificados, como no Oracle:

```
CREATE [OR REPLACE] FUNCTION <nome da função>
( [ <tipo do parâmetro> [,<tipo do parâmetro>] ] )
    RETURNS <tipo do retorno>
  { LANGUAGE <linguagem>
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT |
STRICT
    | [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DE-
FINER
    | AS '<instruções>'
    | AS 'obj _ file', 'link _ symbol'
  } ...
    [ WITH ( <atributo> [,<atributo>] ) ]
```

Exemplo de uma função simples que retorna o resultado de um SELECT:

```
CREATE FUNCTION UM() RETURNS integer
```

```
    AS 'SELECT 1 AS RESULT;'
    LANGUAGE SQL;

SELECT UM() AS Numero;
  Numero
--------
       1
```

MySQL

```
CREATE
    [DEFINER = { <usuário> | CURRENT_USER }]
     PROCEDURE <nome procedimento> ([[ IN | OUT | INOUT ]
<nome parâmetro> <tipo> [,...n]])
    [LANGUAGE SQL
     | [NOT] DETERMINISTIC
     | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES
SQL DATA }
     | SQL SECURITY { DEFINER | INVOKER }
     | COMMENT 'string']
     <corpo do procedimento>


CREATE
    [DEFINER = { <usuário> | CURRENT_USER }]
     FUNCTION <nome função> ([<nome parâmetro> <tipo> [,...
n]])
    RETURNS <tipo retorno>
    [LANGUAGE SQL
     | [NOT] DETERMINISTIC
     | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES
SQL DATA }
     | SQL SECURITY { DEFINER | INVOKER }
     | COMMENT 'string']
   <corpo da função>
```

Triggers

Um gatilho pode ser especificado para disparar antes (`BEFORE`) de a operação ser realizada na tupla[1] (antes das restrições serem verificadas e o `INSERT`, `UPDATE` ou `DELETE` serem efetuados), ou após (`AFTER`) a operação ser realizada (ou seja, após as restrições serem verificadas e o `INSERT`, `UPDATE` ou `DELETE` terem sido completados).

1 Tupla é um registro.

Se o gatilho disparar antes do evento, ele pode evitar a operação para a tupla atual, ou modificar a tupla (para as operações de `INSERT` e `UPDATE`, somente). Se o gatilho disparar após o evento, todas as modificações, incluindo a última inserção, atualização ou exclusão, são "visíveis" para ele.

MS SQL Server

Vejamos a criação de um *trigger*:

```
CREATE TRIGGER [<nome schema>.]<nome trigger>
ON { <tabela> | <view> }
[ WITH [ ENCRYPTION ][ EXECUTE AS <cláusula> ] [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { <instruções> [ ; ] }
```

A seguir, a alteração de um *trigger*:

```
ALTER TRIGGER [ schema.]<nome trigger>
         ON { <tabela> | <view> }
            [WITH [ ENCRYPTION ][ EXECUTE AS <cláusula> ] ]
               {FOR | AFTER | INSTEAD OF}
                  { [INSERT] [, ] [UPDATE] [, ] [DELETE]] }
                     [NOT FOR REPLICATION]
                        AS { <comandos>  [;] [,...n] [;] }
```

Por último, a exclusão:

```
DROP TRIGGER [<schema>.]<nome trigger> [,...n] [;]
```

Oracle

Observe a criação de um *trigger*:

```
CREATE [OR REPLACE] TRIGGER <trigger _ name>
    [BEFORE|AFTER] [INSERT|DELETE|UPDATE] ON <table _ name>
     [REFERENCING [NEW AS <new _ row _ name>] [OLD AS <old _
row _ name>]]
    [FOR EACH ROW [WHEN (<trigger _ condition>)]]
    <trigger _ body>
```

Vejamos um exemplo:

```
CREATE TRIGGER trig _ pedido
    AFTER INSERT ON ITEM _ PEDIDO
    REFERENCING NEW AS newRow
    FOR EACH ROW
    WHEN (newRow.a <= 10)
    BEGIN
        INSERT INTO T5 VALUES(:newRow.b, :newRow.a);
    END trig1;
```

Para alterar, faça o seguinte:

```
ALTER TRIGGER [schema.]<nome trigger> { ENABLE
    | DISABLE
    | RENAME TO <novo nome trigger>
    | COMPILE [DEBUG] [REUSE SETTINGS]}
```

Enfim, a exclusão do *trigger*:

```
DROP TRIGGER [schema.]<nome trigger>;
```

PostgreSQL

No caso do PostgreSQL, a chamada do *trigger* é feita em uma *procedure*:

```
CREATE TRIGGER nome [ BEFORE | AFTER ] [ evento [OR ...] ]
    ON tabela FOR EACH [ ROW | STATEMENT ]
    EXECUTE PROCEDURE função ( argumentos )
```

Veja um exemplo:

```
CREATE TRIGGER trig _ pedido
    BEFORE INSERT ON ITEM _ PEDIDO FOR EACH ROW
   EXECUTE PROCEDURE atualiza _ qtde _ pedido ('cod _ pedido',
'qtde');
```

Para alterar o *trigger*:

```
    ALTER TRIGGER <nome trigger> ON <tabela>
    RENAME TO <novo nome trigger>;
```

Agora, faça a exclusão:

```
DROP TRIGGER <nome trigger> ON <tabela>;
```

Veja um exemplo:

```
DROP TRIGGER trig _ cliente ON CLIENTE;
```

MySQL

Vejamos a criação do *trigger*:

```
CREATE
    [DEFINER = { <usuário> | CURRENT _ USER }]
   TRIGGER <nome trigger> {BEFORE | AFTER} {INSERT | UPDATE
| DELETE}
    ON <nome tabela> FOR EACH ROW <instruções>;
```

Nota: no MySQL, um *trigger* é alterado apenas se o deletarmos para criá-lo novamente.

Para excluir um *trigger*, faça o seguinte:

```
DROP TRIGGER [IF EXISTS] [<schema>.]<nome da trigger>
```

Veja um exemplo:

```
DROP TRIGGER IF EXISTS trig _ pedido;
```