# MODELAGEM E DESENVOLVIMENTO DE BANCO DE DADOS

Pedro Henrique Chagas Freitas

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

# Cláusulas e operadores como complementos para os comandos da linguagem SQL

## Objetivos de aprendizagem

- Identificar as cláusulas e os operadores.
- Exemplificar as cláusulas e os operadores.
- Implementar as cláusulas e os operadores.

## Introdução

As cláusulas e os operadores utilizados como complemento para os comandos da linguagem SQL (*Structured Query Language*), que é uma linguagem inspirada em álgebra relacional, são utilizados para metrificar e afunilar as interações realizadas entre o SGBD e o banco de dados. Logo, as cláusulas e os operadores são apresentados ao longo deste capítulo de forma holística, com uma abordagem das principais características que vinculam a sua utilização com a linguagem SQL.

Neste capítulo, você vai estudar a modelagem e o desenvolvimento de banco de dados a partir da utilização de cláusulas e operadores que são empregados para complementar os comandos da linguagem SQL, observando as estruturas que compõem a linguagem SQL, bem como exemplos de sua utilização e implementação em bancos de dados.

## Conceituando a SQL com cláusulas e operadores

A linguagem SQL (*Structured Query Language*) é uma poderosa linguagem para interação do Sistema Gerenciador de Banco de Dados (SGBD) e o banco de dados. Essas interações podem consistir em alterações nos dados, inserções, exclusões, consultas, etc.

Por sua vez, cada instrução SQL tem, em si mesma, uma complexidade única. Assim, uma consulta, por exemplo, não é semelhante em cenários distintos. Vamos ver um caso? Uma consulta pode retornar um valor previamente estabelecido em uma coluna de uma tabela, mas esse retorno pode ocorrer por ordem alfabética, por categorização, por inserção, entre outras formas.

O surgimento dessa necessidade de apreciação diante dos diferentes tipos de interações fez com que a linguagem SQL começasse a adotar cláusulas e operadores com o objetivo de criar filtros, métricas, especificações, restrições na interação com o banco de dados. Logo, a partir das cláusulas e dos operadores, as interações entre os Sistemas Gerenciadores de Bancos de Dados (SGBDs) e os bancos de dados se tornaram mais dinâmicas, aumentando diretamente, também, a complexidade dessa interação.

Obviamente, não é possível exaurir todas as cláusulas e os operadores do SQL, tendo em vista a própria robustez da linguagem e a constante mudança e evolução do SQL. Entretanto, é possível verificar que tanto as cláusulas quanto os operadores vão representar restrições no tratamento das instruções, sendo que as cláusulas costumam vir primeiro e, caso seja necessário, operadores são adicionados para refinar mais a instrução SQL.

Inicialmente, trabalharemos com a cláusula WHERE, que significa "onde" e é responsável por adicionar filtros, por exemplo, no comando UPDATE, SELECT ou DELETE. Temos aqui, então, três divisões para entender:

1. Cláusula;
2. Comando;
3. Operador.

Assim, um comando possui uma ou mais cláusulas, sendo que uma cláusula pode possuir operadores. Por exemplo: AND e OR são operadores da cláusula WHERE. Veja o seguinte caso:

```
SELECT *FROM tb_funcionarios WHERE sexo = 'M';
```

**SELECT** é o comando SQL que restringe a consulta da tabela funcionários na cláusula 'onde' = WHERE sexo = 'M' = Masculino. Isso significa que restringimos a nossa consulta (SELECT) na tb_funcionarios, onde (WHERE) sexo equivale ao masculino. Se não utilizássemos a cláusula WHERE, teríamos uma consulta que retornaria todos os valores da **tb_funcionarios** pelo fato de utilizarmos o **SELECT * FROM tb_funcionarios**. Se adicionássemos o operador AND ao nosso exemplo com a cláusula WHERE, teríamos:

```
SELECT * FROM tb_funcionarios WHERE sexo = 'M' AND salario >500;
```

Temos, então, retorno por meio do operador AND somente dos funcionários do sexo 'Masculino' que tenham salário maior que 500. Vamos supor que alguém teve aumento de salário. A partir disso, faríamos, então:

```
UPDATE tb_funcionarios SET salario = salario * 1.5 WHERE id = 2;
```

Nesse caso, o nosso id = 2 representa um funcionário chamado 'João'; não utilizamos 'João' porque poderíamos ter vários 'João'; todavia, o nosso 'ID' não se repete, por isso usamos o 'ID'.

Temos, então, uma atualização no salário de João, representado aqui pelo ID = 2.

Assim, podemos concluir que as cláusulas e os operadores são utilizados para apurar, restringir ou afunilar a manipulação dos dados no banco de dados, o que é uma das principais razões pelas quais o SQL é tão difundido na interação com os bancos de dados.

Outros operadores muito utilizados são os operadores de 'busca' que começam ou terminam com um determinado valor. Exemplo:

```
SELECT * FROM tb_funcionarios WHERE nome LIKE '%a%';
```

Nesse caso, estamos utilizando o operador LIKE para criar uma restrição na cláusula WHERE, onde somente retornaremos elementos da coluna nome que tenham a letra a. O símbolo % é uma máscara que, nesse caso, descreve que o que está antes ou após 'a' não tem relevância. Logo, não importa como o nome começa ou termina, somente importa que tenha a letra 'a'. Vamos supor que queremos todos os nomes que comecem com a letra 'a'; nesse caso, utilizaríamos somente:

```
SELECT * FROM tb_funcionarios WHERE nome LIKE 'a%';
```

Se fosse somente os que terminam com a letra 'a':

```
SELECT * FROM tb_funcionarios WHERE nome LIKE '%a';
```

Isso nos demonstra que o operador combinado com a cláusula, no caso o operador 'LIKE' combinado com a cláusula 'WHERE', gera uma consulta mais personalizada ao banco de dados e, obviamente, não é realizada somente

uma consulta, em um determinado período, ao banco de dados. Isso significa que várias consultas complexas ao mesmo tempo podem onerar o tempo de retorno do banco de dados.

**Saiba mais**

O inverso também pode ser utilizado: se quisermos todos os nomes que não começam com a letra C, utilizaríamos o comando **NOT LIKE**:

```
SELECT * FROM tb_funcionarios WHERE nome NOT LIKE 'c%;
```

## Exemplificando a utilização de operadores

Podemos permitir, no nosso código SQL, uma consulta que faça comparação, isto é, que apresente mais de um parâmetro para realização do retorno. Para isso, temos o operador BETWEEN, por exemplo:

```
SELECT * FROM tb_funcionarios WHERE salario BETWEEN 5000 AND
10000;
```

Note, então, que estamos aumentando a complexidade de consulta, utilizando uma cláusula WHERE com dois operadores, 'BETWEEN' e 'AND', para restringir que os valores retornados da tabela funcionários na coluna salário devem ser 'entre' = BETWEEN 5.000 'e' = AND 10.000 (Figura 1).

**tb_funcionarios**

| ID | Nome | Salario |
|---|---|---|
| 556 | Vitor | 5458 |
| 847 | Caio | 5689 |
| 962 | Rafael | 6784 |
| 998 | Otávio | 8522 |

**Figura 1.** Tabela funcionários.

Nesse caso, é possível demonstrar que cláusulas podem ser vinculadas a mais de um operador, tendo o isolamento de campo a partir de dois comandos (BETWEEN) e (AND).

Temos, também, o SOUNDEX, que é um operador para busca fonética. No caso, todo nome tem um código de pronúncia, por exemplo, Luis com (S) e Luiz com (Z), para efetuar uma busca pelo código fonético. Vamos supor que estamos buscando um nome que sabemos pronunciar, mas que não sabemos como é escrito — Luis pode ser com (S) ou com (Z). Se utilizamos, por exemplo:

```
SELECT * FROM tb_funcionarios;
```

Temos um retorno de todos os funcionários e seria inviável procurar um por um. Entretanto, quando não sabemos exatamente a codificação do dado — no caso, se Luis será escrito com (S) ou com (Z) — poderíamos implementar várias consultas até chegar ao nosso dado, mas nossa ideia, aqui, não é auditar as consultas até chegar ao dado esperado. Logo, faríamos uma busca fonética:

```
SELECT * FROM tb_funcionarios WHERE SOUNDEX (nome) = SOUNDEX
('Luis');
```

Nesse caso, retornaríamos vários Luis, seja Luis com 'z' ou com 's'. Supondo que fosse Michael Jackson, teríamos, então (Figura 2):

```
SELECT * FROM tb_funcionarios WHERE SOUNDEX (nome) = SOUNDEX
('Michael');
```

**tb_funcionarios**

| Nome |
|---|
| Michael Jackson |
| Michael Jordan |
| Mikael Jakson |
| Micael Jacson |
| Maicael Jordan |

**Figura 2.** Exemplo SOUNDEX.

## Fique atento

Existem diversos comandos que, combinados com cláusulas, podem onerar e dificultar as consultas realizadas em bancos de dados. Diante disso, é importante verificar o custo/benefício de realizar, por exemplo, buscas fonéticas em grandes bases de dados. Assim, imagine a base de dados de uma operadora aérea em que temos diversos nomes com fonética parecida e a partir da qual desejamos descobrir um determinado dado, isto é um nome.

Dada a dimensão da base de dados em questão, quanto mais complexa a consulta, ou seja, quanto mais cláusulas e operadores a consulta tiver, maior a tendência de que demore um tempo desproporcional para o retorno da consulta.

Logo, é essencial que se verifique a complexidade da consulta para se avaliar se ela deve ser realizada — existem, por exemplo, consultas menores, que podem ir afunilando até chegarem ao resultado de consulta esperado (Figura 3). É importante analisar essa questão, porque diversas consultas, atualizações, inserções são realizadas em um banco de dados ao mesmo tempo. No caso de uma companhia aérea, consultas complexas poderiam derrubar a inserção de dados, o que dificultaria a entrega de recurso do banco de dados.

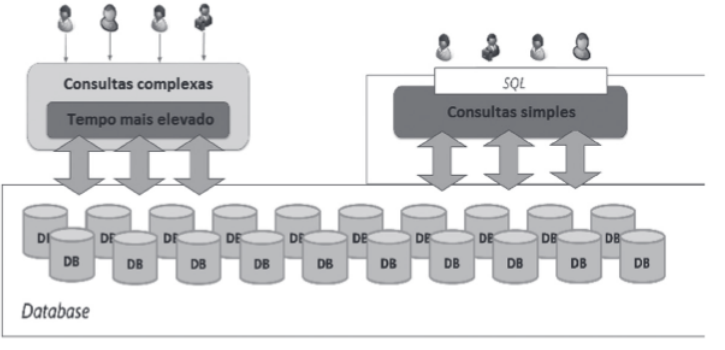


**Figura 3.** A complexidade das consultas.

### Fique atento

**As cláusulas mais comuns são:**

- **FROM:** utilizada para informar a fonte do dado. No SELECT, especificamente, o FROM é uma cláusula obrigatória, tendo em vista que se consulta algo em algum lugar; logo, o FROM precisa ser utilizado.
- **SELECT:** existe muita divergência sobre o SELECT, porque alguns autores atribuem-no como uma instrução (comando) DML, outros dizem que o SELECT é o único comando DQL, já outros afirmam que o SELECT é uma cláusula obrigatória para uma consulta SQL. De toda forma, o SELECT é fundamental para a realização de uma consulta e pode ser entendido, também, como uma cláusula.
- **WHERE:** não é obrigatória. Diferentemente do FROM, que configura uma restrição dos dados retornados, WHERE designa um local de restrição para uma transação com um dado, por exemplo: uma coluna, um index, um campo, etc.
- **GROUP BY:** responsável por agrupar os dados a partir dos campos informados por meio de funções aritméticas de ordenação. Logo, existem várias formas de ordenar os dados, ou seja, várias maneiras de utilizar o GROUP BY.
- **HAVING:** é utilizada com o GROUP BY e restringe os dados recuperados a partir de testes nos campos em que a agrupação (GROUP BY) foi realizada.
- **ORDER BY:** talvez seja a cláusula mais utilizada depois do FROM, tendo em vista que o ORDER BY ordena uma consulta à base de dados — por exemplo, por *strings* ou números. Existem diversas maneiras de ordenar uma consulta, o que depende do tipo de resultado que é esperado.

### Saiba mais

Enquanto as cláusulas visam representar uma restrição, os operadores visam representar uma condição de restrição. Durante uma instrução SQL, essa separação é puramente conceitual, tendo em vista que a composição das instruções com cláusulas + operadores serve para dar mais precisão à realização da instrução pretendida. Assim sendo, não se preocupe em separar os conceitos, mas em entender as principais finalidades e utilizações das cláusulas e dos operadores.

## Implementação de cláusulas e operadores como complemento da linguagem SQL

Nessa implementação, utilizaremos cláusulas e operadores com instruções SQL para implementar filtros com datas e, também, para implementar ordenações.

Fazendo um SELECT na tabela: tb_funcionarios, desejamos fazer comparações com datas. Assim, temos:

```
SELECT * FROM tb_funcionarios WHERE cadastro > '2016-01-01';
```

Estamos fazendo manipulação da data com ano, mês e dia. Vamos fazer, então, uma atualização em uma data de admissão:

```
UPDATE tb_funcionarios SET admissao = CURRENT_DATE ( ) WHERE
id = 1;
```

A função CURRENT_DATE é responsável por consultar a data atual diretamente na base de dados. Temos, então, uma atualização para a data atual onde o id do funcionário é igual a 1.

Agora, imaginemos que temos um processo seletivo e o funcionário entrou em um período de treinamento para validação da sua admissão de 30 dias após seu ingresso.

```
UPDATE tb_funcionarios SET admissao = DATE_ADD (CURRENT_DATE
( ), INTERVAL 30 DAY) WHERE id = 2;
```

Nesse caso, estamos passando dois parâmetros com DATE_ADD: CURRENT_DATE e INTERVAL 30 DAY vinculados, onde o **id = 2;**, sendo que o INVERVAL 30 DAY serve para somar um intervalo de 30 dias à frente da data atual.

Agora, vamos verificar informações sobre uma data, realizando a diferença entre a data final e inicial, analisando, por exemplo, a diferença em dias de duas determinadas datas, com o objetivo de constatar, então, quantos dias faltam para o fim do período de treinamento para admissão. Teríamos, assim:

```
SELECT DATADIFF (admissao, CURRENT DATA ( ) ) FROM tbm_funcio-
narios WHERE id = 2;
```

Teremos, então:

tb_funcionarios

| ID | DATADIFF (admissao, CURRENT_DATE ()) |
|---|---|
| 2 | 30 |

**Figura 4.** Exemplo de consulta: data de admissão.

Para mudar a descrição do campo DATEDIFF para 'Diferença de dias', utilizamos:

```
SELECT DATADIFF (admissao, CURRENT_DATE ( )) AS 'Diferença de dias'
FROM tb_funcionarios WHERE id = 2;
```

Logo, teremos:

| ID | Diferença de dias |
|---|---|
| 2 | 30 |

**Figura 5.** Exemplo de consulta: diferença de dias.

Caso desejássemos verificar, por exemplo, quem foi contratado no mês de maio, isto é, mês 5, faríamos:

```
SELECT * FROM tb_funcionarios WHERE MONTH (admissao) = 5;
```

Nesse caso, estamos utilizando operadores para definir que o mês de retorno de admissão é o mês 5. Teríamos, então:

| ID | Nome | salario | adminissão | sexo | cadastro |
|---|---|---|---|---|---|
| 2 | João | 2.600 | 2018.05.28 | M | 2018.05.28 15: 40: 27 |
| 5 | Diego | 4.500 | 2018.05.28 | M | 2018.05.28 16: 31: 03 |
| 8 | Victor | 5.300 | 2018.05.28 | M | 2018.05.28 16: 35: 10 |
| 17 | Carlos | 7.000 | 2018.05.28 | M | 2018.05.28 17: 45: 07 |
| 36 | Daniele | 9.200 | 2018.05.28 | F | 2018.05.28 18: 03: 35 |

**Figura 6.** Exemplo de consulta por mês.

## Leituras recomendadas

ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados*. 6. ed. São Paulo: Pearson, 2010.

HEUSER, C. A. *Projeto de banco de dados*. 6. ed. Porto Alegre: Bookman, 2010. (Série Livros Didáticos Informática UFRGS, 4).

KORTH, H. F.; SILBERSHATZ, A.; SUDARSHAN, S. *Sistemas de banco de dados*. 6. ed. Rio de Janeiro: Campus, 2012.

RAMAKRISHNAN, R. *Sistemas de Gerenciamento de Bancos de Dados*. 3. ed. Porto Alegre: Penso, 2009.

SETZER, V. W. *Banco de dados*: conceitos, modelos, gerenciadores, projeto lógico, projeto físico. 3. ed. São Paulo: Blucher, 2002.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS