CONSULTAS EM BANCO DE DADOS

Aline Zanin

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Seleção em SQL

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Reconhecer a operação de seleção em álgebra relacional e em bancos de dados relacionais.
- Explicar o uso de restrições condicionais e operadores relacionais nas consultas em SQL.
- Aplicar os operadores lógicos e especiais nas consultas em SQL.

Introdução

Por meio da linguagem SQL (Structured Query Language), utilizada para interagir com banco de dados, é possível inserir, buscar, remover e alterar valores e estruturas do banco de dados. A seleção (busca) de dados em banco de dados compreende uma das ações mais importantes, uma vez que de nada adiantaria existirem valores armazenados se não houvesse sistemas que permitissem consultá-los.

Neste capítulo, aprenderemos como fazer consultas utilizando SQL, de maneira direta (que traz todos os dados do banco de dados) e com filtros empregando operadores lógicos, os quais possibilitam usar uma inteligência na busca dos dados para localizar valores específicos, e não mais todos os valores localizados no banco.

1 Álgebra relacional e operação de seleção em bancos de dados relacionais

No contexto de banco de dados, a **álgebra relacional** constitui uma linguagem de consulta procedural em que, por comandos determinados pelo usuário, o sistema realiza operações em uma base de dados para calcular o resultado desejado. As operações fundamentais na álgebra relacional são seleção, projeção, produto cartesiano, renomear, união e diferença de conjuntos.

As operações fundamentais seleção, projeção e renomear são chamadas de operações unárias, pois operam sobre uma única relação (tabela), e as demais operam um par de relações e denominam-se operações binárias (AMORIN, 2015). A álgebra relacional foi o meio encontrado para provar matematicamente que o modelo relacional funcionava com perfeição. Na terminologia formal de modelo relacional, temos os seguintes conceitos (MACORATTI, 2020):

- uma linha é chamada de tupla;
- o cabeçalho da coluna é chamado de atributo;
- tabela é chamada de relação;
- o tipo de dados que descreve os tipos de valores que podem aparecer em cada coluna nomeia-se domínio.

A abordagem relacional representa um modo de descrever o banco de dados por meio de conceitos matemáticos simples, como a teoria dos conjuntos, que faz com que os usuários vejam o banco de dados como um conjunto de tabelas bidimensionais, separadas em linhas e colunas (MACHADO, 2014). A álgebra relacional se baseia em operadores que permitem construir expressões capazes de retornar uma relação contendo tuplas/registros que satisfazem às condições impostas pelas expressões (ALVES, 2014).

A operação de seleção em álgebra relacional se caracteriza por selecionar tuplas que satisfaçam à condição de seleção, podendo envolver operadores de comparação (=, >, <, ≥, ≤ e ≠) passíveis de combinação por conjunção e disjunção (E, OU, ∧, ∨). Sua representação em relação a uma tabela é feita utilizando o nome da tabela e, entre parênteses, os campos que devem ser selecionados. A Figura 1 mostra uma operação de seleção em álgebra relacional.

cliente (nro_cli , nome_cli, end_cli, saldo, cod_vend)

nro_cli	nome_cli	end_cli	saldo	cod_vend
1	**Márcia**	Rua X	100,00	1
2	Cristina	Avenida 1	10,00	1
3	Manoel	Avenida 3	234,00	1
4	Rodrigo	Rua X	137,00	2

Figura 1. Seleção em álgebra relacional.

Para exemplificarmos a nomenclatura utilizada em álgebra relacional, utilizaremos a tabela a seguir, que representa a tabela hipótetica `pessoa` com os campos nome, telefone e *e-mail* e seus dados respectivos.

Tabela `pessoa`

NOME	TELEFONE	E-MAIL
Maria Silva	5198484848	mariasilva@gmail.com
Ana Julia Freitas	2198563636	anajuliafreitas@gmail.com
Marli Lima	5199855699	mlima@gmail.com
Ana Clara Lima	5488966589	anaclima@gmail.com
Mariana Falcão	1198569856	marifalc@gmail.com
Elisabete de Souza	2196985369	esouza@gmail.com

Aplicando a nomenclatura especificada para essa tabela, podemos dizer que:

- São exemplos de tuplas:

Maria Silva	5198484848	mariasilva@gmail.com
Ana Julia Freitas	2198563636	anajuliafreitas@gmail.com
Marli Lima	5199855699	mlima@gmail.com
Ana Clara Lima	5488966589	anaclima@gmail.com
Mariana Falcão	1198569856	marifalc@gmail.com
Elisabete de Souza	2196985369	esouza@gmail.com

- `Pessoa` é um exemplo de relação
- `Nome`, `telefone` e `e-mail` compreendem exemplos de atributos
- No exemplo, o único domínio é `varchar`, porque todos os valores na tabela são palavras, correspondendo a *strings* na maioria das linguagens de programação. Outros exemplos de domínio utilizados pelo SGBD PostgreSQL são `integer` e `boolean`.

Em todos os sistemas de informação, os recursos de dados devem ser organizados e estruturados de alguma maneira lógica, para que possam ser acessados com facilidade, processados de maneira eficiente, recuperados com rapidez e gerenciados com eficácia (OBRIEN; MARAKAS, 2013). Em bancos de dados relacionais, os valores são estruturados em tabelas, nas quais se deve inserir todos os dados a serem armazenados, como no exemplo da tabela `pessoa` para exibir as nomenclaturas utilizadas em álgebra relacional. Embora haja diversos bancos de dados relacionais e seus respectivos sistemas gerenciadores de banco de dados, neste capítulo apresentaremos apenas exemplos baseados no SBGD PostgreSQL (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2020).

Em um banco de dados, cada tabela dispõe de colunas, que representam os atributos, e linhas, os registros, as informações que foram salvas na base de dados — por exemplo, uma tabela `cidade` poderia ter as colunas, `ID`, `CEP`, `Descricao`, `UF` e diversas linhas para cadastrar cada uma das cidades desejadas. A partir dessa estrutura de tabela, para capturar os dados armazenados, faz-se necessário executar comandos de seleção.

Toda seleção de dados inicia com a palavra `SELECT`, a partir da qual se desenvolvem as configurações da busca. A operação `SELECT` (selecionar) é usada para criar um subconjunto de registros que atendam a um critério estabelecido. Por exemplo, uma operação de selecionar pode ser usada em um banco de dados de empregado para criar um subconjunto de registros que contenham todos os empregados que ganham mais de US$ 30 mil por ano e quem tenham trabalhado na companhia por mais de 3 anos (OBRIEN; MARAKAS, 2013).

Quando se deseja buscar todos os dados de determinada tabela, utilizam-se `SELECT * FROM` e o nome da tabela. O `FROM` é sempre empregado para dar nome à tabela na qual se fará a busca; já para facilitar a identificação das tabelas que fazem parte de uma consulta, podem ser dados apelidos, como `FROM pessoa p`, `FROM cidade c`.

Em uma mesma consulta, é possível utilizar dados de tabelas distintas, caso em que o nome das tabelas deve ser escrito separado por vírgulas, o que aumenta a importância de usar apelidos. No código SQL a seguir, há um exemplo dessa situação: atente-se para as informações das linhas 1 e 2, nas quais estão sendo utilizadas as cláusulas `SELECT` e `FROM`. Na linha 3, é empregada a cláusula `WHERE`, descrita na sequência para fazer a conexão entre as duas tabelas.

```
SELECT *
    FROM pessoa p, cidade c
    WHERE p.idCIdade = c.id
```

A partir desses comandos SQL básicos criados apenas com a utilização das cláusulas `SELECT` e `FROM`, já é possível visualizar os dados armazenados nas tabelas, embora tenhamos como resultado com isso todos os dados de uma tabela. Esse cenário não é favorável porque em geral tabelas em bancos de dados armazenam um grande volume de dados e informações que datam desde a criação da tabela até o dia atual. A seguir, aprenderemos como filtrar esses dados.

2 Restrições condicionais e operadores relacionais

Quando fazemos uma busca utilizando SQL sem empregar restrições, teremos como resultado a exibição de todos os dados que constam na respectiva tabela do banco de dados; assim, se houver, por exemplo, uma tabela com um milhão de registros, ao realizar um `SELECT * FROM` na tabela, teremos como resultado todos os registros.

Contudo, suponhamos que você esteja realizando uma busca entre as vendas feitas na sua empresa com o objetivo de identificar as vendas feitas no dia de ontem: não seria muito mais fácil localizá-las se o banco de dados já trouxesse apenas os valores que são do seu interesse? Obviamente, a resposta é sim. Existem diversas formas de filtrar esses dados na consulta SQL, mas o filtro mais comum e importante é o `WHERE`, já que em todas as consultas em que se desejar realizar filtros, o `WHERE` será obrigatório, a partir do qual poderão ser colocados outros filtros. Por exemplo, dada uma tabela denominada `pessoa` com os campos, `nome`, `cpf` e `nacionalidade`, se o desejo consistir em buscar apenas as pessoas cuja nacionalidade seja brasileira, será necessário fazer a seguinte busca:

```
SELECT *
FROM pessoa
WHERE nacionalidade = 'brasileira'
```

A resposta para uma cláusula `WHERE` é booleana (verdadeiro ou falso, 0 ou 1), e um valor é exibido apenas quando a resposta for verdadeira e nela se possa fazer verificações com os mais diversos tipos de dados (inteiro, *string*, booleano, caractere, etc.). É importante enfatizar que a cláusula `WHERE` pode ser utilizada não apenas em um `SELECT`, mas também em um `UPDATE` ou em um `DELETE`, outras operações possíveis em SQL.

Para realizar filtragem utilizando `WHERE`, existem operadores condicionais, responsáveis por estruturar as validações — eles são inseridos na cláusula `WHERE` em conjunto com dois valores, que podem ser variáveis ou estáticos. No código SQL exemplificado, foi utilizado o operador de igualdade (=) entre o valor variável `nacionalidade`, que representa a coluna do banco de dados, e o valor fixo `brasileira`. O Quadro 1 representa os operadores que podem ser utilizados e a sua descrição. Eles são utilizados na cláusula `WHERE`, comparando o valor de determinado atributo, posicionado à esquerda do operador, com o resultado de uma subconsulta, entre parênteses, à direita do operador (MILETTO; BERTAGNOLLI, 2014).

Quadro 1. Operadores utilizados para filtrar resultados de consultas SQL

Operador	Descrição
=	Operador de igualdade, compara um valor com outro
>	Operador maior que, verifica se o valor da esquerda é maior que o da direita
<	Operador menor que, verifica se o valor da esquerda é menor que o da direita
>=	Operador maior que, verifica se o valor da esquerda é maior que ou igual ao da direita
<=	Operador menor que, verifica se o valor da esquerda é menor que ou igual ao da direita
<> ou !=	Operadores de diferença, podem ser usados para verificar se os dois valores são diferentes

Para explorarmos os exemplos dados de forma prática, considere a tabela `cliente` exibida a seguir.

Tabela `cliente`

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Maria Julia Coutinho	mj@mymail.com	18	5	2
Paola Silveira	psilveira@email.com.br	25	0	4
Suelen Cabral	suc@xyzmail.com	36	3	1
Juliana Silveira	julianas@supermail.com.br	48	4	
Roberta Marinho	beta@imail.com	36	2	2

Iniciaremos pelo operador de igualdade (=) por meio do seguinte comando:

```
SELECT * FROM cliente WHERE idade = 18
```

Com ele, teremos, entre todos os registros da tabela, apenas os seguintes resultados selecionados, por se tratar de clientes com idade igual a 18 anos:

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Maria Julia Coutinho	mj@mymail.com	18	5	2

Contudo, se trabalharmos com o operador **diferente de** (<> ou !=) por meio do seguinte comando:

```
SELECT * FROM cliente WHERE idade <> 18
```

ou então:

```
SELECT * FROM cliente WHERE idade != 18
```

Assim, teremos, entre todos os registros da tabela, apenas os seguintes resultados selecionados:

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Paola Silveira	psilveira@email.com.br	25	0	4
Suelen Cabral	suc@xyzmail.com	36	3	1
Juliana Silveira	julianas@supermail.com.br	48	4	
Roberta Marinho	beta@imail.com	36	2	2

Esses resultados foram selecionados porque constituem os únicos registros cuja idade do(a) cliente é diferente de 18. Foram utilizados dois exemplos para esse caso porque o operador que representa **diferente de** pode ser utilizado de duas formas.

Agora, vamos explorar o operador **maior que** (>) pelo seguinte comando:

```
SELECT * FROM cliente WHERE filhos > 3
```

Com esse comando, entre todos os registros da tabela, apenas dois registros são selecionados:

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Julia Coutinho	mj@mymail.com	18	5	2
Juliana Silveira	julianas@supermail.com.br	48	4	

Este foi o único resultado selecionado porque o critério estabelecido foi número de filhos maior que três, a partir do qual apenas registros com quatro filhos ou mais seriam selecionados.

Contudo, se utilizarmos o operador **maior ou igual a** (>=) por meio do comando:

```
SELECT * FROM cliente WHERE filhos >= 3
```

Assim, serão selecionados os seguintes registros:

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Julia Coutinho	mj@mymail.com	18	5	2
Suelen Cabral	suc@xyzmail.com	36	3	1
Juliana Silveira	julianas@supermail.com.br	48	4	

Esses registros foram selecionados porque o filtro feito na busca considerou número de filhos maior ou igual a 3, obtendo-se como resultado a seleção de clientes com três ou mais filhos.

Agora, exploraremos o operador **menor que** (<) por meio do seguinte comando:

```
SELECT * FROM cliente WHERE filhos < 3
```

Com esse comando, foram selecionados os seguintes registros, todos considerando o filtro daqueles que devem ter menos de três filhos:

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Paola Silveira	psilveira@email.com.br	25	0	4
Roberta Marinho	beta@imail.com	36	2	2

No entanto, se utilizarmos o operador **menor ou igual a** (<=) pelo comando:

```
SELECT * FROM cliente WHERE filhos <= 3
```

Com esse comando, foram selecionados os seguintes registros, todos considerando o filtro daqueles que devem ter três filhos ou menos.

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Paola Silveira	psilveira@email.com.br	25	0	4
Suelen Cabral	suc@xyzmail.com	36	3	1
Roberta Marinho	beta@imail.com	36	2	2

3 Operadores lógicos e especiais

Além da cláusula `WHERE` e dos operadores condicionais descritos, há operadores lógicos (`AND`, `OR` e `NOT`) e especiais (`BETWEEN`, `IS NULL`, `LIKE`, `IN`). Os operadores lógicos são utilizados como intersecção entre duas cláusulas, por exemplo, com cláusulas A e B, o operador `AND` é utilizado para retornar valores que sejam verdadeiros tanto para A quanto para B, o operador `OR` é empregado para retornar valores verdadeiros para uma das duas opções, A ou B, e o operador `NOT` para negar uma condição. Para demonstrar a aplicação desses operadores, seguiremos utilizando a tabela `cliente`, aplicada para demonstrar o emprego dos operadores condicionais e aritméticos.

Iniciaremos com o operador `AND`, utilizado para que seja aplicada mais de uma condição e para o qual todas as condições precisam ser atendidas. Por exemplo:

```
SELECT * FROM cliente WHERE idade = 18 AND filhos = 2
```

Embora duas clientes tenham 18 anos, apenas uma tem dois filhos.

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0

Analisaremos agora o operador OR, utilizado para que sejam aplicadas mais de uma condição e para o qual uma ou outra condição precisa ser atendida. Por exemplo:

```
SELECT * FROM cliente WHERE idade = 18 OR filhos = 2
```

Agora, as duas clientes com 18 anos são selecionadas, mesmo que uma tenha 5 filhos, além de uma cliente que não tem 18 anos, mas com 2 filhos.

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Maria Julia Coutinho	mj@mymail.com	18	5	2
Roberta Marinho	beta@imail.com	36	2	2

Falaremos agora do operador BETWEEN, utilizado para testar se um valor está dentro de um intervalo. O operador BETWEEN ... AND retorna verdadeiro se a coluna é maior ou igual ao primeiro valor e menor ou igual ao segundo valor (MANNINO, 2008), como:

```
SELECT * FROM cliente WHERE animais BETWEEN 1 AND 3
```

Assim, tem-se o seguinte resultado:

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Julia Coutinho	mj@mymail.com	18	5	2
Suelen Cabral	suc@xyzmail.com	36	3	1
Roberta Marinho	beta@imail.com	36	2	2

O operador `IN`, por sua vez, é empregado para percorrer uma lista de valores e buscar dados dentro dessa lista, como:

```
SELECT * FROM cliente WHERE nome IN ('MARIA CLARA PADILHA',
'MARIA JULIA COUTINHO', 'PAOLA SILVEIRA')
```

Assim, temos como resultado:

NOME	E-MAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Maria Julia Coutinho	mj@mymail.com	18	5	2
Paola Silveira	psilveira@email.com.br	25	0	4

O operador `LIKE` retorna valores cuja composição apresenta valores similares ao especificado. Por exemplo, se usarmos o comando `SELECT * FROM usuario WHERE nome LIKE %an%`, como foi utilizado `%` no início e no fim da palavra, serão retornados valores que contenham `an` em qualquer ponto, como `André` ou `Mariana`. O operador `LIKE` pode ser utilizando também com a negação `NOT`, retornando assim valores que não contenham os caracteres especificados, além de permitir o uso de expressões regulares como padrão na busca de texto (RAMAKRISHNAN; GEHRKE, 2011). Na nossa tabela `cliente`, por exemplo:

```
SELECT * FROM cliente WHERE email LIKE '%y%'
```

retornará como resultado registros cujo `email` contenha `y` em qualquer local:

NOME	EMAIL	IDADE	FILHOS	ANIMAIS
Maria Julia Coutinho	mj@mymail.com	18	5	2
Suelen Cabral	suc@xyzmail.com	36	3	1

Agora, trataremos do operador `IS NULL`, que, ao ser usado em uma consulta, retorna apenas os registros cujo valor seja nulo, isto é, esteja vazio, não informado. O comando `IS NULL` foi criado porque não é possível comparar `NULL` com qualquer valor ou consigo mesmo por meio do operador de igualdade tradicional, já que `NULL` é considerado ausência de qualquer dado. Por exemplo:

```
SELECT * FROM cliente WHERE animais IS NULL
```

retorna o seguinte resultado:

NOME	EMAIL	IDADE	FILHOS	ANIMAIS
Juliana Silveira	julianas@supermail.com.br	48	4	

O operador `NOT` em conjunto com o operador `IN` é utilizado para verificar se determinada condição não está em uma lista. Por exemplo, realizando a operação:

```
SELECT * FROM cliente WHERE nome NOT IN ('JULIANA SILVEIRA',
'MARIA JULIA COUTINHO')
```

teremos como resultado todos os registros cujo nome não está entre os nomes especificados:

NOME	EMAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Paola Silveira	psilveira@email.com.br	25	0	4
Suelen Cabral	suc@xyzmail.com	36	3	1
Roberta Marinho	beta@imail.com	36	2	2

O operador NOT em conjunto com o operador LIKE é utilizado para verificar se determinado valor não está contido em uma palavra. Por exemplo, realizando a operação:

```
SELECT * FROM cliente WHERE nome NOT LIKE ('%RA%')
```

teremos como resultado todos os registros cujo nome não contenha em nenhum lugar a sílaba RA, sendo com isso eliminados os nomes de Paola Silvei**ra**, Suelen Cab**ra**l e Juliana Silvei**ra**.

NOME	EMAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Maria Julia Coutinho	mj@mymail.com	18	5	2
Roberta Marinho	beta@imail.com	36	2	2

O operador NOT em conjunto com o operador BETWEEN é empregado para verificar se um valor está fora de um intervalo de valores. Por exemplo, realizando a operação:

```
SELECT * FROM cliente WHERE filhos NOT BETWEEN 3 AND 5
```

teremos como resultado os seguintes registros:

NOME	EMAIL	IDADE	FILHOS	ANIMAIS
Maria Clara Padilha	mariap@xmail.com	18	2	0
Paola Silveira	psilveira@email.com.br	25	0	4
Roberta Marinho	beta@imail.com	36	2	2

Ao utilizar as formas de consulta apresentadas, você consegue otimizar o seu trabalho de análise de dados tendo resultados mais próximos do desejado do que ao empregar uma consulta aberta apenas com SELECT e FROM.

Referências

ALVES, W. P. *Banco de dados*. 1. ed. São Paulo: Érica, 2014.

AMORIN, M. *Apostila de álgebra relacional*. [2015]. Disponível em: https://ead2.iff.edu.br/pluginfile.php/142088/mod_resource/content/1/AlgebraRelacional.pdf. Acesso em: 07 maio 2020.

MACHADO, F. N. R. *Projeto e implementação de banco de dados*. 3. ed. São Paulo: Érica, 2014.

MACORATTI, J. C. *SQL*: álgebra relacional: operações fundamentais: conceitos básicos. [2020]. Disponível em: http://www.macoratti.net/13/06/sql_arcb.htm. Acesso em: 07 maio 2020.

MANNINO, M. V. Projeto, *Desenvolvimento de aplicações e administração de banco de dados*. 3. ed. Porto Alegre: AMGH, 2008.

MILETTO, E. M.; BERTAGNOLLI, S. C. *Desenvolvimento de software II*: introdução ao desenvolvimento Web com HTML, CSS, JavaScript e PHP. Porto Alegre: Bookman, 2014. (Série Tekne).

OBRIEN, J. A.; MARAKAS, G. M. *Administração de sistemas de informação*. 15. ed. Porto Alegre: AMGH, 2012.

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de gerenciamento de bancos de dados*. 3. ed. aum. Porto Alegre: AMGH, 2008.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL*: the world's most advanced open source relational database. 2020. Disponível em: https://www.postgresql.org/. Acesso em: 07 maio 2020.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS