Resumo do Capítulo: SQL como Ferramenta para Trabalhar com Dados

Bancos de Dados e Tabelas

O **banco de dados** é um local onde se armazenam os dados de forma estruturada. **Entidades** são grupos de objetos que compartilham características comuns. **Objetos** são instâncias individuais de entidades.

Vamos estudar bancos de dados relacionais, onde as *entidades* são tabelas e as *linhas* a tabela são seus *objetos*.

Para trabalhar com bancos de dados, você precisa de um **SGBD** ou sistema de gerenciamento de banco de dados. Este é um conjunto de programas que permite criar um banco de dados, preenchê-lo com as novas tabelas, editar e exibir o conteúdo das tabelas existentes. Em nosso curso usaremos o **PostgreSQL**, um dos SGBDs mais populares.

Tabelas

A **tabela** é um conjunto de linhas e colunas. As colunas são chamadas de **campos**. Elas contêm as características do objeto. Cada campo tem um nome exclusivo e um tipo de dados específico. As linhas da tabela são chamadas de **tuplas** ou **registros**. Cada linha contém informação sobre um objeto específico. Uma **célula** é uma unidade onde se cruzam uma linha e uma coluna.

As **chaves primárias** são usadas para fornecer a cada linha um identificador **unívoco**. Algumas tabelas usam vários campos ao mesmo tempo como suas chaves primárias; nesses casos, elas são chamadas de **chaves primárias compostas**.

Sua primeira instrução SQL

SQL é uma linguagem de computação projetada para gerenciar dados em bancos de dados relacionais. A sintaxe SQL é diferente do Python. Aqui estão suas características básicas:

1) O início de um comentário de uma linha é marcado com dois hífens: `--`

```
-- um comentário de linha única em SQL
```

2) Um comentário de várias linhas é colocado entre `/*` e `*/`:

```
/* um comentário de várias linhas
tem
várias
linhas */
```

3) Os comandos são escritos em letras maiúsculas:

```
SELECT, WHERE, FROM
```

4) Cada instrução (ou consulta) termina com um ponto e vírgula `;` :

```
SELECT
  *
FROM
  table_name;
-- A instrução que solicita todos os dados da tabela termina com ";"
SELECT
  *
FROM
  table_name
WHERE
  column_name IN (1,7,9);
-- A instrução que seleciona dados por condição também termina com ";"
```

5) Quebras de linha após cada palavra-chave:

```
SELECT
  column_1,
```

```
  column_2,
  column_3,
  column_4
FROM
  table_name
WHERE
  column_1 = value_1 AND
  column_2 = value_2 AND
  column_4 = value_3;
```

Para selecionar os dados de tabelas, você precisa escrever uma **instrução** ou **consulta**. A **instrução** é uma requisição escrita de acordo com a sintaxe SQL. Sua instrução deve especificar quais dados selecionar e como processá-los.

O operador **SELECT** faz a seleção que você precisa. As instruções SELECT são assim:

```
SELECT
  column_1,
  column_2,
  column_3 ...
FROM
  table_name;
--Selecionar as colunas da tabela
```

Temos duas palavras-chave em nossa instrução: **SELECT** e **FROM**. SELECT especifica as colunas necessárias da tabela do banco de dados.
Para selecionar todas as colunas da tabela, adicione o símbolo `*` ao operador `SELECT`.
FROM especifica a tabela da qual os dados deveriam ser obtidos.

Fatias de Dados em SQL

O início da condição usada para selecionar os dados é marcado com o comando **WHERE**. A condição é avaliada em cada linha da tabela. Em condições, os operadores de comparação são usados:

```
SELECT
  column_1,
  column_2 --selecionando os nomes de colunas
FROM
  table_name --especificando a tabela
```

```
WHERE
  condition; --definindo a condição de seleção de linha
```

A ordem dos operadores é estritamente definida:

1) `SELECT`

2) `FROM`

3) `WHERE`

Assim como em Python, o SQL usa os operadores lógicos `AND`, `OR`, `NOT`. Eles permitem que você faça uma seleção com várias condições:

```
SELECT
  *
FROM
  table_name
WHERE
  condition_1 AND condition_2;
--Seleciona as linhas em que ambas as condições são verdadeiras
SELECT
  *
FROM
  table_name
WHERE
  condition_1 OR condition_2;
--Seleciona as linhas em que uma ou ambas as condições são verdadeiras
SELECT
  *
FROM
  table_name
WHERE
  condition_1 AND NOT condition_2;
--Seleciona as linhas em que condition_1 é verdadeira e condition_2 é falsa
```

Se você precisar fazer uma seleção de linhas para as quais os valores de um campo estejam dentro de um determinado intervalo, use a instrução **BETWEEN**. `BETWEEN` inclui os limites inicial e final na seleção resultante:

```
SELECT
  *
FROM
  table_name
WHERE
  field_1 BETWEEN value_1 AND value_2;
-- Selecionando linhas em que o valor do campo_1 está entre o valor_1 e o valor_2 (inclusive)
```

Se você precisar fazer uma seleção de linhas para as quais os valores de um campo correspondem aos de uma lista, use o operador **IN**. Indique a lista de valores após `IN`:

```
SELECT
  *
FROM
  table_name
WHERE
  column_name IN ('value_1','value_2','value_3');
```

Se os valores forem números, eles serão separados por vírgulas: `IN (3,7,9)`. Se forem strings, são colocadas entre aspas simples e, novamente, separadas por vírgulas: `IN ('value_1','value_2','value_3')`. A data e a hora são indicadas da seguinte forma: `IN ('yyyy-mm-dd','yyyy-mm-dd')`

Funções de agregação

Assim como Python, SQL possui funções específicas para calcular o número total de linhas, somas, médias e valores mínimos e máximos. São chamados de **funções de agregação**. Eles coletam, ou *agregam*, todos os objetos dentro de um grupo para calcular um único valor de resumo.

Aqui está a sintaxe de uma instrução com uma função de agregação:

```
SELECT
  AGGREGATE_FUNCTION(field) AS here_you_are
--here_you_are - nome da coluna onde a saída da função será armazenada
FROM
  table;
```

Quando você chama uma função de agregação, ela dá à coluna um nome complicado. Para evitar isso, use o comando **AS** e digite um nome novo e mais simples.

A função **COUNT** retorna o número de linhas em uma tabela:

```
SELECT
  COUNT(*) AS cnt
```

```
FROM
  table
```

O número de linhas pode ser calculado de várias maneiras, dependendo da tarefa:

- **COUNT(*)** retorna o número total de linhas da tabela
- **COUNT(coluna)** retorna o número de linhas em uma `column`
- **COUNT(DISTINCT coluna)** retorna o número de linhas *únicas* em uma `column`

A função **SUM(coluna)** retorna a soma dos valores em uma `column`. Ela ignora os valores ausentes.

O **AVG (coluna)** retorna o valor médio de `column`.

O menor e o maior valor podem ser encontrados com as funções **MIN** e **MAX**.

Convertendo Tipos de Dados

Algumas funções de agregação só podem ser usadas com valores numéricos.

Os dados em um campo podem parecer números, mas na verdade são armazenados como strings no banco de dados. Isso acontece com frequência, mais comumente devido a erros no design dos bancos de dados.

Podemos usar uma instrução **CAST** para converter o tipo de dados de valores em uma coluna:

```
CAST (column_name AS data_type)
```

`column_name` é o campo cujo tipo de dados deve ser convertido. `data_type` é o tipo desejado. Também podemos escrever isso:

```
column_name :: data_type
```

Tipos de dados numéricos

integer: um tipo inteiro semelhante a *int* em Python. No PostgreSQL os números inteiros variam de -2147483648 a 2147483647.

real: um número de ponto flutuante, como *float* em Python. A precisão do número para o tipo *real* é de até 6 casas decimais.

Tipos de dados string

'Practicum': este é um exemplo de um valor do tipo string. Nas instruções SQL, ele vai entre as aspas simples.

varchar(n): uma cadeia de caracteres de comprimento variável, em que **n** é o número máximo de caracteres.

text: uma string de qualquer comprimento. Este tipo é como o tipo string do Python.

Data e hora

Datas e horas vão entre as aspas simples.

timestamp: uma data e hora. Análogo ao datetime `datetime` pandas. Esse formato é usado com mais frequência para armazenar eventos que ocorrem várias vezes ao dia, como diários de usuários de sites.

date: uma data

Lógico

boolean — é um tipo de dados lógico. Ele pode ter três valores no PostgreSQL: **TRUE**, **FALSE**, e **NULL** (desconhecido).