Resumen del capítulo: SQL como herramienta para trabajar con datos

Bases de datos y tablas

Una **base de datos** es un lugar donde se almacenan datos estructurados. Las entidades son grupos de **objetos** que comparten características comunes. Los objetos son instancias individuales de entidades.

Vamos a estudiar bases de datos relacionales, donde las entidades son tablas y las filas de las tablas son sus *objetos*.

Para trabajar con bases de datos, necesitas un **DBMS** o sistema de administración de bases de datos. Este es un conjunto de programas que te permite crear una base de datos, llenarla con tablas nuevas y editar y mostrar el contenido de las tablas existentes. En nuestro curso utilizaremos PostgreSQL, uno de los DBMS más populares.

Tablas

Una **tabla** es un conjunto de filas y columnas. Las columnas se llaman campos. Dichos campos contienen las características del objeto. Cada campo tiene un nombre único y un tipo de datos específico. Las filas de la tabla se denominan tuplas o registros. Cada fila contiene información sobre un objeto en particular. Una celda es una unidad donde se cruzan una fila y una columna.

Las claves primarias se utilizan para dar a cada fila un identificador único. Algunas tablas utilizan varios campos a la vez como claves principales; en tales casos, se denominan claves primarias compuestas.

Tu primera instrucción SQL

SQL es un lenguaje informático diseñado para gestionar datos en bases de datos relacionales. La sintaxis de SQL es diferente a la de Python. A continuación se describen sus características básicas:

1) El comienzo de un comentario de una sola línea se marca con dos guiones: `--`

```
-- un comentario de una línea en SQL
```

2) Un comentario de varias líneas se coloca entre `/*` y `*/`:

```
/* un comentario de varias líneas
tiene
varias
líneas */
```

3) Los comandos se escriben en mayúsculas:

```
SELECT, WHERE, FROM
```

4) Cada declaración (o consulta) termina con un punto y coma `;`:

```
SELECT
  *
FROM
  table_name;
-- Una declaración que solicita todos los datos de la tabla termina con ";"
SELECT
  *
FROM
  table_name
WHERE
  column_name IN (1,7,9);
-- Una declaración que selecciona datos por condición también termina con ";"
```

5) Saltos de línea después de cada palabra clave:

```
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 seleccionar datos de tablas, debes escribir una declaración o consulta. Una declaración es una solicitud escrita de acuerdo con la sintaxis SQL. Tu instrucción debe especificar qué datos seleccionar y cómo procesarlos.

El operador **SELECT** toma la selección que necesitas. Las instrucciones SELECT se ven así:

```
SELECT
  column_1,
  column_2,
  column_3 ...
FROM
  table_name;
-- Seleccionar columnas de la tabla
```

Tenemos dos palabras clave en nuestra declaración: SELECT y FROM. SELECT especifica las columnas necesarias de la tabla de la base de datos. PPara seleccionar todas las columnas de la tabla, agrega el símbolo `*` al operador `SELECT`. FROM especifica la tabla de la que se deben tomar los datos.

Data Slices en SQL

El comienzo de la condición utilizada para seleccionar datos se marca con el comando **WHERE**. La condición se evalúa en cada fila de la tabla. En condiciones, se utilizan operadores de comparación:

```
SELECT
  column_1,
  column_2 -- seleccionar nombres de columna
FROM
  table_name -- especificar la tabla
WHERE
  condition; -- definir la condición de selección de fila
```

El orden de los operadores está definido estrictamente:

1) `SELECT`

2) `FROM`

3) `WHERE`

Al igual que Python, SQL utiliza los operadores lógicos `AND`, `OR` y `NOT`. Estos permiten hacer una selección con varias condiciones:

```
SELECT
  *
FROM
  table_name
WHERE
  condition_1 AND condition_2;
-- Selecciona las filas donde ambas condiciones sean verdaderas
SELECT
  *
FROM
  table_name
WHERE
  condition_1 OR condition_2;
-- Selecciona las filas donde una o ambas condiciones sean verdaderas
SELECT
  *
FROM
  table_name
WHERE
  condition_1 AND NOT condition_2;
-- Selecciona las filas donde condition_1 es verdadera y condition_2 es falsa
```

Si necesitas hacer una selección de filas para las cuales los valores de un campo se encuentran dentro de un cierto rango, usa la instrucción **BETWEEN**. `BETWEEN` incluye los límites inicial y final en la selección resultante:

```
SELECT
  *
FROM
  table_name
WHERE
  field_1 BETWEEN value_1 AND value_2;
-- Seleccionar filas donde el valor de field_1 está entre value_1 y value_2 (incluyente)
```

Si necesitas hacer una selección de filas para las cuales los valores de un campo coinciden con los de una lista, usa el operador **IN.** Indica la lista de valores después de

`IN`:

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

Si los valores son números, se separan entre sí por comas: `IN (3,7,9)`. Si son cadenas, se ponen entre comillas simples y, de nuevo, se separan por comas: `IN ('value_1','value_2','value_3')`. La fecha y la hora se indican de la siguiente manera: `IN ('yyyy-mm-dd','yyyy-mm-dd')`

Funciones de agregación

Al igual que Python, SQL tiene funciones específicas para calcular el número total de filas, sumas, promedios y valores mínimos y máximos. Estas se conocen como funciones de agregación. Recopilan o agregan todos los objetos dentro de un grupo para calcular un único valor de resumen.

Aquí está la sintaxis de una instrucción con una función de agregación:

```
SELECT
  AGGREGATE_FUNCTION(field) AS here_you_are
-- nombre de la columna donde se almacenará la salida de la función
FROM
  table;
```

Cuando llamas a una función de agregación, esta le da a la columna un nombre difícil de manejar. Para evitar esto, usa el comando **AS** y escribe uno nuevo y más simple.

La función **COUNT** devuelve el número de filas en una tabla:

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

El número de filas se puede calcular de varias maneras dependiendo de la tarea:

- **COUNT(*)** devuelve el número total de filas de la tabla
- **COUNT(column)** devuelve el número de filas en una columna `column`
- **COUNT(DISTINCT column)** devuelve el número de filas únicas en una columna `column`

La función **SUM(column)** devuelve la suma de los valores de una `column`. Ignora los valores ausentes.

El **AVG (column)** devuelve el valor promedio de los valores de `column`.

Los valores más pequeños y más grandes se pueden encontrar utilizando las funciones **MIN** y **MAX**.

Conversión de tipos de datos

Algunas funciones de agregación solo se pueden usar con valores numéricos.

Los datos de un campo pueden parecer números, pero en realidad se almacenan como cadenas en la base de datos. Esto sucede a menudo y se debe sobre todo a errores en el diseño de bases de datos.

Podemos usar una instrucción **CAST** para convertir el tipo de datos de valores en una columna:

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

`column_name` es el campo cuyo tipo de datos se va a convertir. `data_type` es el tipo deseado. También podemos escribir lo siguiente:

```
column_name :: data_type
```

Tipos de datos numéricos

número entero: un tipo de número entero similar a int en Python. En PostgreSQL los números enteros van desde -2147483648 hasta 2147483647.

real: un número de coma flotante, como float en Python. La precisión numérica para el tipo real es de hasta 6 puntos decimales.

Tipos de datos de cadena

'Practicum': este es un ejemplo de un valor de tipo cadena. En las instrucciones SQL va entre comillas simples.

varchar(n): una cadena de caracteres de longitud variable, donde n es el número máximo de caracteres.

texto: una cadena de cualquier longitud. Este tipo es igual que el tipo de string en Python.

Fecha y hora

Las fechas y horas van entre comillas simples.

timestamp: una fecha y hora. Análogo a `datetime` en pandas. Este formato se usa con mayor frecuencia para almacenar eventos que ocurren varias veces al día, como registros de usuarios de sitios web.

date: una fecha

Lógico

booleano — un tipo de datos lógicos. Puede tener tres valores en PostgreSQL: **TRUE**, **FALSE** y **NULL** (desconocido).