BBDD Relacionales: SQLite

Daniel W. Tümmler Rosales

BASE DE DATOS RELACIONES

- Es una base de datos que almacena y da acceso a puntos de datos relacionados entre sí.
- El modelo relacional es una forma intuitiva y directa de representar datos sin necesidad de jerarquizarlos.
- Una base de datos relacional es un conjunto de tablas, también llamadas relaciones, formadas por filas (registros) y columnas (campos).
- Cada fila (registro) tiene una ID única, denominada clave y las columnas de la tabla contienen los atributos de los datos.
- Un ejemplo de BBDD Relacionales son las tablas de pandas.

¿Qué es SQL?

- SQL: Structured Query Language
- Es un lenguaje de programación diseñado específicamente para trabajar con bases de datos.
- Permite escribir consultas (queries) que el ordenador puede ejecutar y para retornar la información consultada.
- Permite crear y manipular bases de datos relaciones.
- Es un lenguaje de programación declarativo, lo que significa que nos vamos a enfocar en qué resultado queremos, no como hace el ordenador para conseguirlo.

¿Qué es SQLite?

- SQLite es un sistema de gestión de bases de datos relacional, contenida en una relativamente pequeña biblioteca escrita en C.
- SQLite es un proyecto de dominio público creado por D. Richard Hipp.
- Permite almacenar información en dispositivos de una forma sencilla, eficaz, potente, rápida y en equipos con pocas capacidades de hardware, como puede ser un teléfono móvil.

SQLite3: Características

- La base de datos completa se encuentra en un solo archivo.
- Puede funcionar enteramente en memoria, lo que la hace muy rápida.
- Cuenta con librerías de acceso para muchos lenguajes de programación (pysqlite3 para python).
- Soporta texto en formato UTF-8 y UTF-16, así como datos numéricos de 64 bits.
- Soporta funciones SQL definidas por el usuario (UDF).
- El código fuente es de dominio público y se encuentra muy bien documentado.
- https://www.sqlite.org/docs.html

SQL: Sintaxis
Componentes Principales

- Data Definition Language (DDL):
  - Conjunto de declaraciones que permiten al usuario definir o modificar estructuras de datos y objetos.
- Data Manipulation Language (DML):
  - Conjunto de declaraciones que permiten manipular los datos en las tablas de la base de datos.
- Data Control Language (DCL):
  - Conjunto de declaraciones que permiten al administrador del sistema gestor de base de datos, controlar el acceso a los objetos, otorgan o niegan permisos a los usuarios.
- Transaction Control Language (TCL):
  - Conjunto de declaraciones que permite administrar diferentes transacciones que ocurren dentro de una base de datos.

DATA DEFINITION LANGUAGE (DDL)

Conjunto de declaraciones que permiten al usuario definir o modificar estructuras de datos y objetos.

STATEMENTS:

- CREATE: Es usado para crear bases de datos u objetos de bases de datos, como por ejemplo tablas.
  - **CREATE** object_type object_name; # Sintaxis
  - **CREATE TABLE** table_name (column_name data_type); # Ejemplo
  - **CREATE TABLE** estudiantes (nombre CHAR); # Ejemplo
- ALTER: Es usado para alterar o modificar objetos existentes, es usado en conjunto con ADD y REMOVE.
  - **ALTER TABLE** estudiantes # Ejemplo
  - **ADD COLUMN** edad INT;

DATA DEFINITION LANGUAGE (DDL)

STATEMENTS:

- DROP: Es usado para eliminar un objeto de bases de datos.
  - **DROP** object_type object_name; # Sintaxis
  - **DROP** TABLE estudiantes; # Ejemplo
- RENAME: Permite renombrar un objeto
  - **ALTER** object_type object_name **RENAME TO** new_object_name; #Sintaxis
  - **ALTER TABLE** estudiantes **RENAME TO** estudiantes_neoland; # Ejemplo
- TRUNCATE: Vacía el contenido de un objeto sin eliminarlo.
  - **TRUNCATE** object_type object_name; # Sintaxis
  - **TRUNCATE TABLE** estudiantes_neoland; # Ejemplo
  - *** TRUNCATE NO SE PUEDE USAR.

DATA MANIPULATION LANGUAGE (DML)

Conjunto de declaraciones que permiten manipular los datos en las tablas de la base de datos.

STATEMENTS:

- SELECT: Es usado para obtener los datos de un objeto.
  - **SELECT** column_name **FROM** table_name; # Sintaxis
  - **SELECT** nombre **FROM** estudiantes_neoland; # Ejemplo
  - **SELECT** * **FROM** esrtudiantes_neoland # Ejemplo
- INSERT: Es usado para agregar o insertar datos a una tabla,es usado con INTO y VALUES.
  - **INSERT INTO** estudiantes_neoland **VALUES** (“Daniel”, 27); # Ejemplo
  - **INSERT INTO** estudiantes_neoland (nombres) **VALUES** (“Adrian”);

DATA MANIPULATION LANGUAGE (DML)

STATEMENTS:

- UPDATE: Permite modificar datos existentes en las tablas, es usado en conjunto con SET y WHERE.
  - **UPDATE** estudiantes_neoland
  - **SET** nombre = “DANIEL”
  - **WHERE** edad = 27; # Ejemplo
- DELETE: Elimina las filas o registros que cumplan con cierta condición, es usado en conjunto con WHERE.
  - **DELETE FROM** estudiantes_neoland
  - **WHERE** nombre = “DANIEL”; # Ejemplo

*Sí no se usa WHERE, entonces DELETE haría lo mismo que TRUNCATE (vaciar la tabla)

DATA MANIPULATION LANGUAGE (DML)

RESUMEN:

- **SELECT… FROM…**
- **INSERT INTO… VALUES**
- **UPDATE… SET… WHERE**
- **DELETE… FROM…. WHERE**

TRANSACTION CONTROL LANGUAGE (TCL)

Conjunto de declaraciones que permiten al administrador del sistema gestor de base de datos, controlar el acceso a los objetos, otorgan o niegan permisos a los usuarios.

STATEMENTS:

- GRANT: Concede ciertos permisos a usuarios.
  - **GRANT** type_of_permission **ON** database_name.table_name
  - **TO** “username”@”localhost”. # Sintaxis
- REVOKE: Anula los permisos a usuarios.
  - **REVOKE** type_of_permission **ON** database_name.table_name
  - **FROM** “username”@”localhost”. # Sintaxis

DATA CONTROL LANGUAGE (DCL)

Conjunto de declaraciones que permite administrar diferentes transacciones que ocurren dentro de una base de datos.

STATEMENTS:

- COMMIT: Guarda los cambios que se han realizado, está relacionado con INSERT, DELETE, UPDATE.
  - **UPDATE** estudiantes_neoland
  - **SET** nombre = “DANIEL”
  - **WHERE** edad = 27
  - **COMMIT**; # Ejemplo

DATA CONTROL LANGUAGE (DCL)

STATEMENTS:

- ROLLBACK: Permite deshacer cualquier cambio no deseado hecho después de haber ejecutado COMMIT.
  - **UPDATE** estudiantes_neoland
  - **SET** nombre = “DANIEL”
  - **WHERE** edad = 27
  - **COMMIT**;
  -
  - **ROLLBACK**; # Ejemplo

TIPOS DE DATOS EN SQL

- CHAR(x): cadenas de caracteres de tamaño “x”, todos los elementos de este tipo ocuparan espacio“x” en memoria
  - CHAR(5)
- VARCHAR(x): cadenas de caracteres de tamaño “x”, los elementos de este tipo ocuparan espacio igual o menor a “x” en memoria, dependiendo del largo de la cadena.
  - VARCHAR(10)
- ENUM: cadenas de caracteres, se utiliza para crear variables categóricas, los elementos de esta columna son dictadas al momento de su creación, no acepta ningún valor que no haya sido especificado.
  - ENUM(“M”, “F”)

    *SQLite mostrará error si intentamos agregar algún valor a esta columna que no sean “M” o “F”

TIPOS DE DATOS EN SQL

- INT o INTEGER: números enteros
- DECIMAL(p, s): número decimal de precisión p y escala s.
  - DECIMAL(5, 3): 10.543
- FLOAT(p, s): número flotante de precisión p y escala s.
- DOUBLE(p, s): igual que FLOAT pero puede almacenar el doble de tamaño.
- DATE: se usa para representar fechas en el formato YYYY-MM-DD, debe estar escrito entre comillas.
  - “11-06-2018”
- DATETIME: se usa para representar fechas y tiempo en el formato YYYY-MM-DD HH:MM:SS, debe estar escrito entre comillas.
  - “11-06-2018 13:07:48”

TIPOS DE DATOS EN SQL

- BLOB: se refiere a archivos de datos binarios, se utiliza para almacenar archivos, Binary Large OBject.
- BOOLEAN: tipo de dato True o False.

CREATE

- Crear un database:
  - **CREATE DATABASE** [IF NOT EXISTS] database_name;
    - IF NOT EXISTS es opcional
    - CREATE DATABASE neoland;
    - **CREATE DATABASE IF NOT EXISTS** neoland;
    - **** **CREATE DATABASE NO SE PUEDE USAR**
- Crear una tabla: para crear una tabla es obligatorio inicializarla al menos con una columna.
  - **CREATE TABLE** table_name
  - (column_1 data_type constraints,
  - column_2 data_type constraints,
  - …
  - column_n data_type constraints);

CONSTRAINTS

constraints: reglas específicas o límites que definimos en tablas, el objetivo de estos es trazar las relaciones existentes entre las diferentes tablas de un database.

- **PRIMARY KEY**: es un valor o combinación de valores de cada tabla que sirven para identificar de forma única los elementos de la tabla, generalmente es utilizado como un índice.
  - **PRIMARY KEY** (column_name)
- **FOREIGN KEY**: apunta a la columna de otra tabla, se utiliza para poder unirlas o relacionarlas.
  - **FOREIGN KEY** (column_name) **REFERENCE** table(primary key column)
- **UNIQUE KEY**: se utiliza para que una columna tenga siempre valores diferentes.
- **AUTOINCREMENT**: se utiliza para llenar una columna de forma ascendente, comenzando desde 0, la columan debe ser **PRIMARY KEY** o **INDEX**.

CONSTRAINTS

- NOT NULL: se utiliza para que la columna no tenga valores nulos, es decir, que siempre tenga un valor.
- ON DELETE CASCADE: constraint utilizado en conjunto con FOREIGN KEY, su objetivo es eliminar las filas de la FOREIGN KEY si estas desaparecen de PRIMARY KEY.
    - **FOREIGN KEY** (column_name) **REFERENCE** table(primary key column) **ON DELETE CASCADE**
- DEFAULT: asigna un valor particular a cada fila de esa columna.

CONSTRAINTS

```
CREATE TABLE neoland(
        estudiante_id   INT AUTOINCREMENT PRIMARY KEY,
        nombre          VARCHAR(255) NOT NULL,
        apellido        VARCHAR(255) NOT NULL,
        email           VARCHAR(255) UNIQUE,
        edad            INT,
        pais_id          INT FOREIGN KEY REFERENCES paises(pais_id) ON DELETE CASCADE
);
```

SELECT... FROM...

SELECT: permite extraer información del database.

```
SELECT
        column_1, column_2, ..., column_n
FROM
        table_name;
```

```
SELECT
        *
FROM
        table_name;
```

- Para seleccionar todas las columnas se puede usar *

WHERE

WHERE: nos permite filtrar las filas según un criterio.

```
SELECT
    *
FROM
    table_name
WHERE
    condition;
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre = “Daniel”;
```

AND & OR

Nos permite combinar 2 o más condiciones en WHERE

AND

```
SELECT
    *
FROM
    table_name
WHERE
    condition_1 AND condition_2;
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre = “Daniel” AND edad = 27;
```

OR

```
SELECT
    *
FROM
    table_name
WHERE
    condition_1 OR condition_2;
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre = “Daniel” OR nombre = “Adrian”;
```

Nota: En SQL el operador AND se aplica primero, y el operador OR de segundo.

LIKE & NOT LIKE

LIKE: retorna las filas donde se cumple un patrón.

- %: Se usa para hacer “match” de varios caracteres.
- _: Se usa para hacer “match” de un solo carácter.

LIKE

```
SELECT
    *
FROM
    table_name
WHERE
    column LIKE (pattern);
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre  LIKE (“Dan%”);
```

* En este ejemplo SQL buscará las filas donde nombre comience con “Dan” y tenga cualquier cantidad de caracteres después de “Dan”.

LIKE & NOT LIKE

LIKE

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre LIKE (“%el”); *
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre  LIKE (“%an%”); **
```

* En este ejemplo SQL buscará las filas donde nombre termine con “el” y tenga cualquier cantidad de caracteres antes de “el”.

** Aquí buscará las filas donde nombre tenga “an” en cualquier parte de la cadena de caracteres.

LIKE & NOT LIKE

LIKE

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre LIKE (“Da_”); *
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre  LIKE (“_an”); **
```

* En este ejemplo SQL buscará las filas donde nombre comience con “Da” y tenga un carácter después de “Da”.

** En este ejemplo SQL buscará las filas donde nombre termine con “an” y tenga un carácter antes de “an”.

LIKE & NOT LIKE

NOT LIKE

NOT LIKE: retorna las filas donde NO se cumpla el patrón.

IN & NOT IN

IN: retorna las filas donde se cumpla la condición.

```
SELECT
    *
FROM
    table_name
WHERE
    column_name IN (list);
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre IN (“Daniel”, “Adrian”, “Angel”);
```

IN & NOT IN

NOT IN: retorna las filas donde NO se cumpla la condición.

```
SELECT
    *
FROM
    table_name
WHERE
    column_name NOT IN (list);
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    nombre NOT IN (“Daniel”, “Adrian”, “Angel”);
```

IS NOT NULL & IS NULL

IS NOT NULL: es usado para filtrar las filas donde el valor sea NO nulo.

```
SELECT
    *
FROM
    table_name
WHERE
    column IS NOT NULL;
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    edad IS NOT NULL;
```

IS NULL: es usado para filtrar las filas donde el valor sea nulo.

COMPARADORES LOGICOS

=	EQUAL TO
>	GREATER THEN
>=	GREATER THAN OR EQUAL TO
<	LESS THAN
<=	LESS THAN OR EQUAL TO
<>, !=	NOT EQUAL, DIFFERENT FROM

SELECT DISTINCT

SELECT DISTINCT: es usado para seleccionar solo los valores únicos de una columna, funciona como .unique() de pandas.

```
SELECT DISTINCT
    column_name
FROM
    table_name;
```

```
SELECT DISTINCT
    nombre
FROM
    estudiantes_neoland;
```

BETWEEN... AND....

BETWEEN: nos ayuda a filtrar filas donde los valores estén dentro de un rango, la columna debe ser NUMERICA, funciona como between() en pandas.

```
SELECT
    *
FROM
    table_name
WHERE
    column BETWEEN a AND b;
```

```
SELECT
    *
FROM
    estudiantes_neoland
WHERE
    edad BETWEEN 25 AND 30;
```

NOT BETWEEN: nos ayuda a filtrar filas donde los valores NO estén dentro de un rango, la columna debe ser NUMERICA.

ORDER BY

```
SELECT
    column1, column2, ...
FROM
    table_name
ORDER BY
    column1, column2, ...;
```

```
SELECT
    nombre
FROM
    estudiantes_neoland
ORDER BY
    nombre DESC;
```

Se puede agregar **ASC** o **DESC** al final de **ORDER BY**, por defecto SQL usa **ASC**.