Práctica de SQL y Data WareHouse KeepCoding Bootcamp I Inteligencia Artificial Full Stack Edición III

Explicación de Sentencias SQL para la Creación del Esquema de Base de Datos

Este documento proporciona una explicación técnica y detallada de las sentencias SQL utilizadas para definir un esquema de base de datos. A continuación se describe cada grupo de sentencias y su propósito.

1. Eliminación de Tablas Existentes

```
DROP TABLE IF EXISTS bootcamp;
DROP TABLE IF EXISTS student;
DROP TABLE IF EXISTS stu_bootcamp;
DROP TABLE IF EXISTS "module";
DROP TABLE IF EXISTS teacher;
DROP TABLE IF EXISTS tea_module;
```

Propósito

- Garantizar un entorno limpio al eliminar las tablas especificadas si ya existen en la base de datos lo que garantiza un entorno limpio a la hora de recrear las tablas

Detalles Técnicos

- `DROP TABLE` : Elimina tanto la estructura como los datos de la tabla.
- `IF EXIST` : Previene errores si la tabla no existe.

2. Creación de la tabla bootcamp

```
CREATE TABLE bootcamp (
    bootcamp_id SERIAL PRIMARY KEY,
    name VARCHAR(50)
);
```

Propósito

Definir la tabla `bootcamp` para almacenar información básica de los bootcamps.

Detalles Técnicos

- `SERIAL` : Genera automáticamente un valor único y autoincremental para cada registro.
- `PRIMARY KEY` : Garantiza que `bootcamp_id` sea único y no nulo.
- `VARCHAR(50)` : Define una columna de texto con una longitud máxima de 50 caracteres.

3. Creación de la Tabla

```
CREATE TABLE student (
    student_id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    surname VARCHAR(50),
    bootcamp_id INT,
    country VARCHAR(50),
    address VARCHAR(100),
    city VARCHAR(100),
    email VARCHAR(50),
    phone VARCHAR(50)
);
```

Propósito

Crear la tabla `student` para almacenar datos de estudiantes.

Detalles Técnicos

- `student_id SERIAL PRIMARY KEY` : Identificador único y autoincremental para cada estudiante.
- `bootcamp_id INT` : Columna que permite establecer una relación con la tabla `bootcamp` .
- Las columnas como `name` , `surname` , `country` , etc., están definidas con tipos

de datos adecuados para almacenar texto.

4. Restricción de Unicidad para el Campo

```
ALTER TABLE student
ADD CONSTRAINT unique_email UNIQUE (email);
```

Propósito

Asegurar que no existan valores duplicados en la columna `email` de la tabla `student`.

Detalles Técnicos

- `ALTER TABLE`: Modifica una tabla existente.
- `ADD CONSTRAINT unique_email`: Define una nueva restricción llamada `unique_email` lo que garantiza que cada valor en la columna `email` sea único.

5. Creación de la Tabla Intermedia

```
CREATE TABLE stu_bootcamp (
    stu_bootcamp_id SERIAL PRIMARY KEY,
    student_id INT,
    bootcamp_id INT,
    FOREIGN KEY (student_id) REFERENCES student(student_id),
    FOREIGN KEY (bootcamp_id) REFERENCES bootcamp(bootcamp_id)
);
```

Propósito

Implementar una relación de **muchos a muchos** entre estudiantes y bootcamps.

Detalles Técnicos

- `stu_bootcamp_id SERIAL PRIMARY KEY`: Identificador único y autoincremental para cada relación.
- `FOREIGN KEY`: Establece relaciones referenciales con las tablas `student` y `bootcamp`.

6. Creación de la Tabla

```
CREATE TABLE teacher (
    teacher_id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    surname VARCHAR(50),
    phone VARCHAR(50),
    email VARCHAR(50)
);
```

Propósito

Definir la tabla `teacher` para almacenar información de los profesores.

Detalles Técnicos

- `teacher_id SERIAL PRIMARY KEY` : Identificador único y autoincremental para cada profesor.

7. Creación de la Tabla

```
CREATE TABLE module(
    module_id SERIAL PRIMARY KEY,
    bootcamp_id INT,
    teacher_id INT,
    FOREIGN KEY (bootcamp_id) REFERENCES bootcamp(bootcamp_id)
);
```

Propósito

Definir la tabla `module` para representar los módulos impartidos en cada bootcamp.

Detalles Técnicos

- `module_id SERIAL PRIMARY KEY` : Identificador único y autoincremental para cada módulo.
- `FOREIGN KEY (bootcamp_id)` : Vincula cada módulo con un bootcamp válido.

8. Creación de la Tabla Intermedia

```
CREATE TABLE tea_module (
    tea_module_id SERIAL PRIMARY KEY,
    module_id INT,
    teacher_id INT,
    FOREIGN KEY(module_id) REFERENCES module(module_id),
    FOREIGN KEY(teacher_id) REFERENCES teacher(teacher_id)
);
```

Propósito

Implementar una relación de **muchos a muchos** entre módulos y profesores.

Detalles Técnicos

- `tea_module_id SERIAL PRIMARY KEY` : Identificador único para cada relación.
- `FOREIGN KEY` : Establece relaciones referenciales con las tablas `module` y `teacher` .

Resumen

1. **Limpieza inicial**: Se eliminan las tablas existentes para evitar conflictos al recrearlas.
2. **Definición de tablas**: Se crean las tablas `bootcamp` , `student` , `teacher` , `module` , `stu_bootcamp` y `tea_module` con sus respectivas claves primarias y columnas.
3. **Relaciones**:
    - Relaciones de **uno a muchos**: Por ejemplo, entre `module` y `bootcamp` .
    - Relaciones de **muchos a muchos**: Implementadas mediante tablas intermedias como `stu_bootcamp` y `tea_module` .
4. **Integridad referencial**: Garantizada mediante claves foráneas y restricciones de unicidad.

```
                                        Eugenio Barquin
```