Practical 2: Database Creation and SQL Queries

Objective: Create a real-world scenario to understand SQL database creation, table design, and essential SQL commands. This exercise will help you learn how to design databases, create tables with proper relationships, and perform queries

Part 1: Database Setup

1. **Database Creation**:
   - **Database Name**: `SchoolManagementSystem`
   - create a new database in PostgreSQL or MySQL with this name.

1. Students Table

- **Fields**:
  - `student_id`: Integer (Primary Key, unique identifier for each student)
  - `first_name`: Varchar (Student's first name)
  - `last_name`: Varchar (Student's last name)
  - `age`: Integer (Student's age)
  - `gender`: Char (Gender code, e.g., 'M' or 'F')
  - `class_id`: Integer (Foreign Key, links to Classes table)

2. Classes Table

- **Fields**:
  - `class_id`: Integer (Primary Key, unique identifier for each class)
  - `class_name`: Varchar (Name of the class)
  - `teacher_id`: Integer (Foreign Key, links to Teachers table)

3. Teachers Table

- **Fields**:
  - `teacher_id`: Integer (Primary Key, unique identifier for each teacher)
  - `first_name`: Varchar (Teacher's first name)
  - `last_name`: Varchar (Teacher's last name)
  - `subject`: Varchar (Subject taught by the teacher)

4. Subjects Table

- **Fields**:
  - `subject_id`: Integer (Primary Key, unique identifier for each subject)
  - `subject_name`: Varchar (Name of the subject)
  - `class_id`: Integer (Foreign Key, links to Classes table)

5. Enrollments Table

- **Fields**:
  - `enrollment_id`: Integer (Primary Key, unique identifier for each enrollment)
  - `student_id`: Integer (Foreign Key, links to Students table)
  - `subject_id`: Integer (Foreign Key, links to Subjects table)
  - `enrollment_date`: Date (Date of enrollment)

1. **Understanding Data Types**:
   - **Integer**: Stores whole numbers (e.g., age, IDs).
   - **Text**: Stores longer text data (e.g., descriptions or notes).
   - **Varchar**: Stores variable-length strings with a max length (e.g., first and last names).
   - **Char**: Stores fixed-length strings (e.g., gender code).
   - **Date**: Stores dates without any time component (e.g., enrollment or birth dates).
   - **Timestamp**: Stores both date and time (e.g., tracking when records are created or updated).
   - **Boolean**: Stores `TRUE` or `FALSE` values (e.g., an `is_active` field to show if a record is active).
   - **Float/Decimal**: Stores numbers with decimal points for precise values (e.g., prices, scores, or financial data).
   -
2. **Primary and Foreign Keys**:
   - **Primary Key (PK)**: A unique identifier for each row in a table (e.g., student_id in the Students table). No two rows can have the same primary key, and it cannot be null.
   - **Foreign Key (FK)**: A field that links one table to another. For instance, class_id in the Students table is a foreign key linking to the Classes table, establishing a relationship between students and their classes.
3. **Insert Data**:
   - Insert at least 20 rows of sample data into each table. Use unique, realistic data for students, classes, teachers, and subjects to make the dataset meaningful.

Part 2: SQL Queries

Now that the tables and data are set up, it's time to practice querying the data.

1. **Query 1**: Select All Fields from a Table
    - Write a query to retrieve all fields from the Students table.
2. **Query 2**: Filter with WHERE
    - Retrieve the details of students who are above a certain age (e.g., 18). Use the WHERE clause.
3. **Query 3**: Ordering Results
    - Retrieve the list of classes, ordered alphabetically by class_name. Use ORDER BY.
4. **Query 4**: Grouping Data
    - Count the number of students in each class. Use the GROUP BY command with an aggregate function like COUNT or DISTINCT.
5. **Query 5**: Joining Tables
    - Retrieve the first and last names of students along with the names of the classes they are enrolled in. Use JOIN to connect the Students and Classes tables.
6. **Query 6**: Multi-Table Join with Conditions
    - Retrieve the names of students along with the subjects they are enrolled in and the teacher's name for each subject. This requires joining multiple tables: Students, Enrollments, Subjects, and Teachers.

Submission Instructions

1. **SQL Queries**:
    - Save each query with a short comment explaining the purpose and any insights from the result. Eg. Query 1 selects all fields in the student table, Query 2 ....
    - All the queries should be in one script, but you decide to make them in different scripts if that makes it simple for you.
    - Take screenshot of every query result, save it(eg. query_1_result_image)
    - And upload all to github
2. **Upload to GitHub**:
    - Push the SQL script to your GitHub repository for submission by copying and pasting it in a file in VSCODE and push to GitHub.

- Share your repo with me in this link : link to shrae repo

Facing A Problem And Want To Use Online Tools:

there are several online database editors and tools that can be used to work with PostgreSQL and MySQL databases without having to set up a local database. Here are some user-friendly options:

1. db-fiddle (for PostgreSQL and MySQL)

- **Website**: db-fiddle.com
- **Features**:
  - Supports both PostgreSQL and MySQL.
  - Allows you to create tables, insert data, and run SQL queries in a simple, browser-based editor.
- **Usage**:
  - Choose the database type (PostgreSQL or MySQL) from the options.
  - Create tables, insert sample data, and write queries.
- **Ideal for**: Practicing SQL queries and quick prototyping.

2. SQL Fiddle (for PostgreSQL and MySQL)

- **Website**: sqlfiddle.com
- **Features**:
  - Supports various database engines, including PostgreSQL and MySQL.
  - Allows creating schema, inserting data, and running queries.
- **Usage**:
  - Select the database type and create a schema, enter data, and execute queries within the interface.
- **Limitations**:
  - SQL Fiddle can sometimes be slow or unavailable, so it’s best to use it for smaller datasets and lighter operations.