SQL Day 1: SQL Foundations

1. Introduction to SQL

What is SQL?

SQL (Structured Query Language) is a programming language used to manage and manipulate relational databases. It allows users to create, read, update, and delete (CRUD) data stored in a structured format.

Importance of SQL in the Data World

- SQL is widely used in software development, data analysis, and database administration.
- It helps businesses make informed decisions by retrieving and analyzing data efficiently.
- Essential for data professionals working in various industries.

Real-Life Applications of SQL

- **E-commerce**: Storing customer details, orders, and product catalogs.
- **Banking**: Managing customer accounts, transactions, and loan data.
- **Analytics**: Querying large datasets for reporting and business insights.

2. Relational Database Basics

Tables, Rows, and Columns

A database consists of tables, where:

- **Tables** store data in a structured format.
- **Rows (records)** represent individual data entries.
- **Columns** define data attributes (e.g., Name, Age, Email).

Data Types

SQL provides various data types to store different kinds of values. Some of the most commonly used data types include:

- **INT**: Used to store integer values. It does not accept decimal numbers.
  - Example: age INT (stores values like 10, 25, 100)
- **VARCHAR(n)**: Used to store variable-length character strings, where n defines the maximum length.
  - Example: name VARCHAR(50) (stores values like "Alice", "John Doe")
- **CHAR(n)**: Stores fixed-length character strings. If the string length is shorter than n, it is padded with spaces.
  - Example: code CHAR(5) (if storing "A1", it will be "A1 ")
- **TEXT**: Used for large text data, such as descriptions and notes.
  - Example: description TEXT
- **DATE**: Stores only date values in the format YYYY-MM-DD.
  - Example: birthdate DATE (stores values like '2024-02-21')
- **DATETIME**: Stores both date and time values.
  - Example: created_at DATETIME (stores values like '2024-02-21 14:30:00')
- **DECIMAL(p, s)**: Stores fixed-point decimal numbers, where p is the total number of digits, and s is the number of digits after the decimal point.
  - Example: price DECIMAL(10,2) (stores values like 9999.99)
- **BOOLEAN**: Stores either TRUE or FALSE (1 or 0).
  - Example: is_active BOOLEAN

Constraints

Constraints are rules enforced on table columns to maintain data integrity and reliability. Some of the important constraints include:

- **Primary Key**: Ensures that each record in a table is uniquely identified. It cannot have NULL values.
  - Example:
  - CREATE TABLE students (
  -     id INT PRIMARY KEY,
  -     name VARCHAR(50)
  -);
- **Foreign Key**: Establishes a relationship between two tables by referring to the primary key of another table.
  - Example:
  - CREATE TABLE enrollments (
  -     student_id INT,
  -     course_id INT,
  -     FOREIGN KEY (student_id) REFERENCES students(id),
  -     FOREIGN KEY (course_id) REFERENCES courses(course_id)
  -);
- **Unique**: Ensures that all values in a column are distinct (no duplicates allowed).
  - Example:
  - CREATE TABLE users (
  -     email VARCHAR(100) UNIQUE
  -);

- **Not Null**: Ensures that a column cannot have NULL values.
    - Example:
    - CREATE TABLE employees (
    -     emp_id INT PRIMARY KEY,
    -     name VARCHAR(50) NOT NULL
    -);
- **Default**: Assigns a default value to a column if no value is provided during insertion.
    - Example:
    - CREATE TABLE orders (
    -     order_id INT PRIMARY KEY,
    -     status VARCHAR(20) DEFAULT 'Pending'
    -);
- **Check**: Restricts the values that can be inserted into a column based on a specific condition.
    - Example:
    - CREATE TABLE products (
    -     id INT PRIMARY KEY,
    -     price DECIMAL(10,2) CHECK (price > 0)
    -);

Relationships

- **One-to-One**: One record in Table A maps to one record in Table B.
- **One-to-Many**: One record in Table A maps to multiple records in Table B.

- **Many-to-Many**: Multiple records in Table A map to multiple records in Table B via a linking table.

Example: Understanding Table Relationships

```
CREATE TABLE students (

    id INT PRIMARY KEY,

    name VARCHAR(50),

    age INT

);


CREATE TABLE courses (

    course_id INT PRIMARY KEY,

    course_name VARCHAR(100)

);


CREATE TABLE enrollments (

    student_id INT,

    course_id INT,

    FOREIGN KEY (student_id) REFERENCES students(id),

    FOREIGN KEY (course_id) REFERENCES courses(course_id)

);
```

3. Setting Up SQL Environment

Installing SQL Databases

- **MySQL**: Download from https://www.mysql.com

- **PostgreSQL**: Download from https://www.postgresql.org

Connecting to a Database

Using command-line tools:

```
mysql -u root -p

psql -U postgres
```

Using GUI tools:

- **MySQL Workbench** (for MySQL)
- **pgAdmin** (for PostgreSQL)

Creating a Database and Table

MySQL Example:

```
CREATE DATABASE school;

USE school;


CREATE TABLE students (

    id INT PRIMARY KEY,

    name VARCHAR(50),

    age INT,

    grade VARCHAR(10)

);
```

PostgreSQL Example:

```
CREATE DATABASE school;

\c school;
```

```
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    age INT,
    grade VARCHAR(10)
);
```

This setup ensures you have a strong foundation in SQL and relational databases, setting the stage for more advanced queries and operations.