1. Introduction to SQL

SQL (Structured Query Language) is the cornerstone of managing and interacting with relational databases. It's a standardized language that allows users to perform various operations on the data stored in a database, such as querying, updating, and managing the database structure. SQL is essential for developers, data analysts, and database administrators who work with relational databases.

History of SQL

Origin and Development

- **Early 1970s**: SQL was developed at IBM by Raymond Boyce and Donald Chamberlin. Initially named **SEQUEL** (Structured English Query Language), it was designed to manipulate and retrieve data stored in IBM's first relational database, System R.
- **Renaming**: Due to trademark issues, SEQUEL was renamed **SQL**.
- **User-Friendly Syntax**: The language was designed with a syntax that is close to natural English, making it easier for users to learn and apply compared to previous query languages.

Standardization

- **1986**: The American National Standards Institute (**ANSI**) standardized SQL, making it a universally recognized language for database management.
- **1987**: The International Organization for Standardization (**ISO**) adopted SQL as a standard, further solidifying its position in the industry.
- **Evolution**: SQL has evolved over the years with various versions introducing new features such as support for XML, triggers, and procedural extensions like PL/SQL (Procedural Language/SQL) and T-SQL (Transact-SQL).

Importance of SQL in Database Management

- **Data Manipulation**: SQL allows efficient querying, updating, and deletion of data.
- **Cross-Platform Consistency**: Being a standardized language, SQL ensures that data manipulation and retrieval are consistent across different database management systems (DBMS) like MySQL, PostgreSQL, Oracle, and SQL Server.
- **Scalability**: SQL can handle large datasets, making it suitable for both small applications and large enterprise systems.

What is SQL?

Definition and Purpose

- **SQL** is a programming language specifically designed for managing and manipulating relational databases. It allows users to define the structure of the data (using **DDL** - Data Definition Language), manipulate the data itself (using **DML** - Data Manipulation Language), control access to the data (using **DCL** - Data Control Language), and manage transactions (using **TCL** - Transaction Control Language).

Overview of Relational Databases

- **Relational Database**: A collection of data organized into tables (also known as relations). Each table consists of rows (records) and columns (attributes).
- **Relationships**: Tables can be related to each other through **keys** (Primary Key, Foreign Key), enabling complex queries that retrieve data from multiple tables.
- **Normalization**: A process used in relational databases to reduce redundancy and improve data integrity by organizing data into related tables.

SQL Syntax

Basic SQL Syntax Rules

- **Case-Insensitive**: SQL keywords can be written in uppercase or lowercase, but the standard convention is to write them in uppercase for readability.
- **Semicolon (`;`)**: SQL statements usually end with a semicolon, especially in environments where multiple statements are executed sequentially.
- **Comments**:
  - Single-line comments: `-- This is a comment`
  - Multi-line comments: `/* This is a multi-line comment */`

Structure of SQL Statements

A basic SQL query typically follows this structure:

```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

- **SELECT**: Specifies the columns of data to retrieve.
- **FROM**: Specifies the table from which to retrieve the data.
- **WHERE**: Filters the results based on a specific condition.

Example:

```sql
SELECT ProductName, Price
FROM Products
WHERE Price > 500;
```

This query retrieves the names and prices of products that cost more than $500 from the `Products` table.

Writing Your First SQL Query

Let's write a simple SQL query step by step:

1. **Scenario**: Retrieve the names and cities of customers who are located in the USA from the `Customers` table.
2. **Query**:

   ```sql
   SELECT CustomerName, City
   FROM Customers
   WHERE Country = 'USA';
   ```

3. **Explanation**:
   - **SELECT CustomerName, City**: Specifies the columns to be retrieved.
   - **FROM Customers**: Indicates the table where the data is stored.
   - **WHERE Country = 'USA'**: Filters the results to include only customers from the USA.

Additional Example Tables

To better understand how SQL works with relational databases, consider the following example tables:

Products Table

A table that stores information about products.

ProductID	ProductName	SupplierID	CategoryID	Price
1	Laptop	1	2	800
2	Smartphone	2	1	600
3	Tablet	3	2	300

- **Columns**:
  - `ProductID`: Unique identifier for each product.
  - `ProductName`: Name of the product.
  - `SupplierID`: ID of the supplier providing the product.
  - `CategoryID`: ID of the category to which the product belongs.
  - `Price`: Price of the product.

Sample Query:

```sql
SELECT ProductName, Price
FROM Products
WHERE CategoryID = 2;
```

This query retrieves the names and prices of products in category 2 (e.g., Electronics).

Orders Table

A table that stores information about customer orders.

OrderID	CustomerID	OrderDate	TotalAmount
101	1	2024-08-10	1500
102	2	2024-08-11	2000
103	1	2024-08-12	500

- **Columns**:
  - `OrderID`: Unique identifier for each order.
  - `CustomerID`: ID of the customer who placed the order.
  - `OrderDate`: Date when the order was placed.
  - `TotalAmount`: Total amount for the order.

Sample Query:

```sql
SELECT OrderID, TotalAmount
FROM Orders
WHERE CustomerID = 1;
```

This query retrieves the order IDs and total amounts for orders placed by customer 1.

Visual Representation of Relationships

Here's how these two tables (Products and Orders) could be related in a relational database:

- **Products Table**: Stores product details.
- **Orders Table**: Stores order details and may reference products through a foreign key (though not explicitly shown in this example).

This relationship allows for complex queries that retrieve data from both tables, such as finding all orders that include a specific product.

Example Query Using JOIN:

```sql
SELECT Orders.OrderID, Products.ProductName, Orders.TotalAmount
FROM Orders
JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
JOIN Products ON OrderDetails.ProductID = Products.ProductID
WHERE Products.ProductName = 'Laptop';
```

This query retrieves all orders that include the product 'Laptop', showing the order ID, product name, and total amount.
