Introduction to SQL with MySQL/MariaDB

History of SQL

SQL (Structured Query Language) is a standardized database management language first developed in the 1970s by IBM. It was designed to manipulate and retrieve data stored in relational database management systems. The first prototype, called SEQUEL (Structured English Query Language), was developed for the relational database management system project called System R at IBM. Over time, the language was standardized by ANSI and ISO, leading to its widespread adoption in the industry.

List of Some SQL DBMS

SQL-based database management systems (DBMS) are numerous and varied, each with specific features and advantages. Here are some of the main SQL-based DBMS:

1. **Oracle Database**: One of the most robust and widely used relational database management systems, ideal for large enterprises with vast amounts of data and complex transactions.
2. **Microsoft SQL Server**: This DBMS is very popular in Windows environments and offers easy integration with other Microsoft products. It is known for its ease of use and good performance.
3. **MySQL**: Owned by Oracle Corporation, MySQL is very popular in the web world for its ease of use and free availability. It is often used for web applications and websites.
4. **PostgreSQL**: Known for its standards compliance and extensibility, PostgreSQL is a highly respected and powerful object-relational database management system.
5. **IBM Db2**: Db2 offers advanced data management and analytics capabilities for enterprises, with options available for both cloud and on-premises deployments.
6. **SQLite**: Very lightweight, SQLite is a database solution often used for mobile applications and small desktop applications. It is unique in that it is integrated directly into an application.
7. **MariaDB**: Created by the original developers of MySQL, MariaDB is often considered a direct replacement, offering more features, better performance, and greater openness.

8. **SAP HANA**: This is an in-memory platform that allows the processing of large amounts of data in real-time. SAP HANA is particularly well-suited for business environments using other SAP software.

Each of these DBMS has its own characteristics that may be better suited to certain applications or environments than others.

MySQL and MariaDB

MySQL is one of the most popular open-source relational database management systems. It was created by Michael Widenius ("Monty") and David Axmark in the 1990s. MySQL is famous for its speed, robustness, and ease of use. It is widely used for web applications and has become an integral part of the LAMP (Linux, Apache, MySQL, PHP/Python/Perl) technology stack.

MariaDB is open source and was forked from MySQL in 2009 by the same developers who created MySQL, after MySQL was acquired by Oracle Corporation. MariaDB is designed to be compatible with MySQL, while offering new features, improved performance, and replaced functionalities that were not available in the free versions of MySQL.

General Operation

SQL works by parsing and executing instructions written in the SQL language. These instructions can perform various tasks, such as creating tables, updating data, retrieving data, and managing transactions. SQL uses statements to define data structures, queries to retrieve data, and commands to manage data and transactions.

MySQL Table Engines

MySQL uses "storage engines" to manage how data is stored, managed, and retrieved. Each storage engine has its own properties, optimizations, and specific uses. Here are some of the most common engines:

- **MyISAM**: This was the default engine before MySQL 5.5. It is known for its speed in read operations but does not support transactions or crash recovery.
- **InnoDB**: This is the default storage engine for MySQL since version 5.5. InnoDB supports transactions, crash recovery, and row-level locking. It is designed to maximize data robustness and integrity.

- **Memory**: This engine stores data in memory, offering very fast access. However, data is lost when the database is shut down, as it is not stored persistently.
- **Archive**: Used for storing large amounts of data that do not require frequent modification. It is optimized for fast insertions and data compression.

MySQL table engines allow users to choose the best way to store and manage their data based on their specific needs, making this system extremely flexible and powerful for a wide range of applications.

SQL Documentation: Quotes and Syntax

Introduction

In SQL, the use of single quotes ('), double quotes ("), and backticks (`) varies depending on the database management system (DBMS) used. These characters are used to identify database elements, such as table and column names, and to delimit strings.

Single Quotes (')

- **Usage:** Used to delimit strings.
- **Example:**

```
SELECT * FROM utilisateurs WHERE nom = 'Dupont';
```

Double Quotes ("")

- **Usage:** According to the SQL standard, double quotes are used to identify identifiers (table names, column names, etc.) that do not follow normal naming conventions or contain special characters.
- **Example:**

```
SELECT "nomColonne" FROM "maTable" WHERE "nomColonne" = 'valeur';
```

- **Note:** Not all DBMS follow this convention. For example, MySQL uses double quotes as single quotes to delimit strings unless the ANSI_QUOTES mode is enabled.

Backticks (`)

- **Usage:** Specifically used in MySQL and some other DBMS to surround identifiers.
- **Example:**

```
SELECT `nomColonne` FROM `maTable` WHERE `nomColonne` = 'valeur';
```

- **Note:** This ensures that reserved words can be used as table or column names without causing errors.

General Syntax Rules

1. **Case Sensitivity:** SQL is not case-sensitive for keywords, but the handling of identifiers can be case-sensitive depending on the DBMS and the underlying operating system configuration.
2. **Comments:**
   - Single-line comments: Use `-- comment` or `# comment` (specific to MySQL).
   - Multi-line comments: Use `/* comment */`.
3. **Statement Termination:** SQL statements should end with a semicolon (`;`), although some tools and environments may tolerate statements without a semicolon at the end of the query.
4. **Naming Identifiers:**
   - Should not start with a digit but can be numeric if properly delimited.
   - Should not contain spaces. Use underscores `_` as separators.
   - Avoid using SQL reserved words unless they are surrounded by appropriate quotes.

SQL in Practice

1. Creating the Database and Tables

```
-- Drops the database if it exists.
DROP DATABASE IF EXISTS restaurant;
```

```
-- Creates the database.
CREATE DATABASE restaurant;
```

```sql
-- Selects the database.
USE restaurant;
```

```sql
-- Creates the client table.
CREATE TABLE client (
    id_client INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    nom VARCHAR(50),
    prenom VARCHAR(50),
    email VARCHAR(100)
);
```

```sql
-- Creates the order table.
CREATE TABLE commande (
    id_commande INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    id_client INT,
    date_commande DATE,
    total DECIMAL(10, 2),
    FOREIGN KEY (id_client) REFERENCES client(id_client)
);
```

```sql
-- Creates the category table.
CREATE TABLE categorie (
    id_categorie INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    nom_categorie VARCHAR(50)
);
```

```sql
-- Creates the dish table.
CREATE TABLE plat (
    id_plat INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    nom_plat VARCHAR(50),
    id_categorie INT,
    prix DECIMAL(10, 2),
    FOREIGN KEY (id_categorie) REFERENCES categorie(id_categorie)
);
```

```sql
-- Data for the `client` table
INSERT INTO client (nom, prenom, email) VALUES
('Martin', 'Lucie', 'lucie.martin@example.com'),
('Bernard', 'Julien', 'julien.bernard@example.com'),
('Kuong', 'Émilie', 'emilie.kuong@example.com'),
('Petit', 'Sophie', 'sophie.petit@example.com'),
('Robert', 'Christophe', 'christophe.robert@example.com');
```

```sql
-- Data for the `categorie` table
INSERT INTO categorie (nom_categorie) VALUES
('Entrée'),
('Plat principal'),
('Dessert'),
('Boisson');
```

```sql
-- Data for the `plat` table
INSERT INTO plat (nom_plat, id_categorie, prix) VALUES
('Salade Niçoise',        (SELECT categorie.id_categorie FROM categorie where
categorie.nom_categorie = 'Entrée'),        12.50),
('Steak Frites',        (SELECT categorie.id_categorie FROM categorie where
categorie.nom_categorie = 'Plat principal'),18.90),
('Mousse au Chocolat',    (SELECT categorie.id_categorie FROM categorie where
categorie.nom_categorie = 'Dessert'),        6.50),
('Poulet Basquaise',    (SELECT categorie.id_categorie FROM categorie where
categorie.nom_categorie = 'Plat principal'),16.50),
('Tarte Tatin',            (SELECT categorie.id_categorie FROM categorie where
categorie.nom_categorie = 'Dessert'),        8.00),
('Limonade Maison',        (SELECT categorie.id_categorie FROM categorie where
categorie.nom_categorie = 'Boisson'),        3.75);
```

```sql
-- Data for the `commande` table
INSERT INTO commande (id_client, date_commande, total) VALUES
((SELECT client.id_client FROM client where client.email = 'lucie.martin@example.com'),
'2023-09-01',45.90),
((SELECT client.id_client FROM client where client.email = 'julien.bernard@example.com'),
'2023-09-02',24.25),
((SELECT client.id_client FROM client where client.email = 'emilie.kuong@example.com'),
'2023-09-03',19.75),
((SELECT client.id_client FROM client where client.email = 'sophie.petit@example.com'),
'2023-09-04',34.75),
((SELECT client.id_client FROM client where client.email =
'christophe.robert@example.com'),'2023-09-05',22.50);
```

2. SQL Queries

```sql
-- 1. Selects all columns from the client table.
SELECT * FROM client;
```

```sql
-- 2. Selects the name and email of clients whose name starts with "A".
SELECT nom, email FROM client WHERE nom LIKE 'A%';
```

```sql
-- 3. Selects the total number of clients.
SELECT COUNT(*) AS nombre_clients FROM client;
```

```sql
-- 4. Selects the cheapest dish.
SELECT * FROM plat ORDER BY prix LIMIT 1;
```

```sql
-- 5. Selects dishes priced above 10 euros in a specific category.
SELECT * FROM plat WHERE prix > 10 AND id_categorie = (SELECT id_categorie FROM categorie WHERE
nom_categorie = 'plat principal');
```

```sql
-- 6. Selects orders placed by a given client (ID 1).
SELECT * FROM commande WHERE id_client = 1;
```

```sql
-- 7. Selects the total amount of all orders.
SELECT SUM(total) AS montant_total FROM commande;
```

```sql
-- 8. Updates the name of a client (ID 1).
UPDATE client SET nom = 'Nouveau Nom' WHERE id_client = 1;
```

```sql
-- 9. Inserts a new dish into the dish table.
INSERT INTO plat (nom_plat, prix, id_categorie) VALUES ('Nouveau plat', 15.99, (SELECT
id_categorie FROM categorie WHERE nom_categorie = 'Entrée'));
```

```sql
-- 10. Deletes a dish from the dish table (ID 1).
DELETE FROM plat WHERE id_plat = 1;
```

```sql
-- 11. Selects dishes in descending order of price.
SELECT * FROM plat ORDER BY prix DESC;
```

```sql
-- 12. Selects clients who have placed an order with an amount greater than 50 euros.
SELECT DISTINCT c.* FROM client c INNER JOIN commande cm ON c.id_client = cm.id_client WHERE
cm.total > 50;
```

```sql
-- 13. Selects dishes with their respective categories.
SELECT p.nom_plat AS plat, c.nom_categorie AS categorie FROM plat p INNER JOIN categorie c ON
p.id_categorie = c.id_categorie;
```

```sql
-- 14. Selects clients who have never placed an order.
SELECT * FROM client WHERE id_client NOT IN (SELECT DISTINCT id_client FROM commande);
```

```sql
-- 15. Selects dishes and their respective number of orders, sorted by number of orders in
descending order.
SELECT p.nom_plat AS plat, COUNT(cm.id_commande) AS nombre_commandes FROM plat p LEFT JOIN
commande cm ON p.id_plat = cm.id_plat GROUP BY p.id_plat ORDER BY nombre_commandes DESC;
```