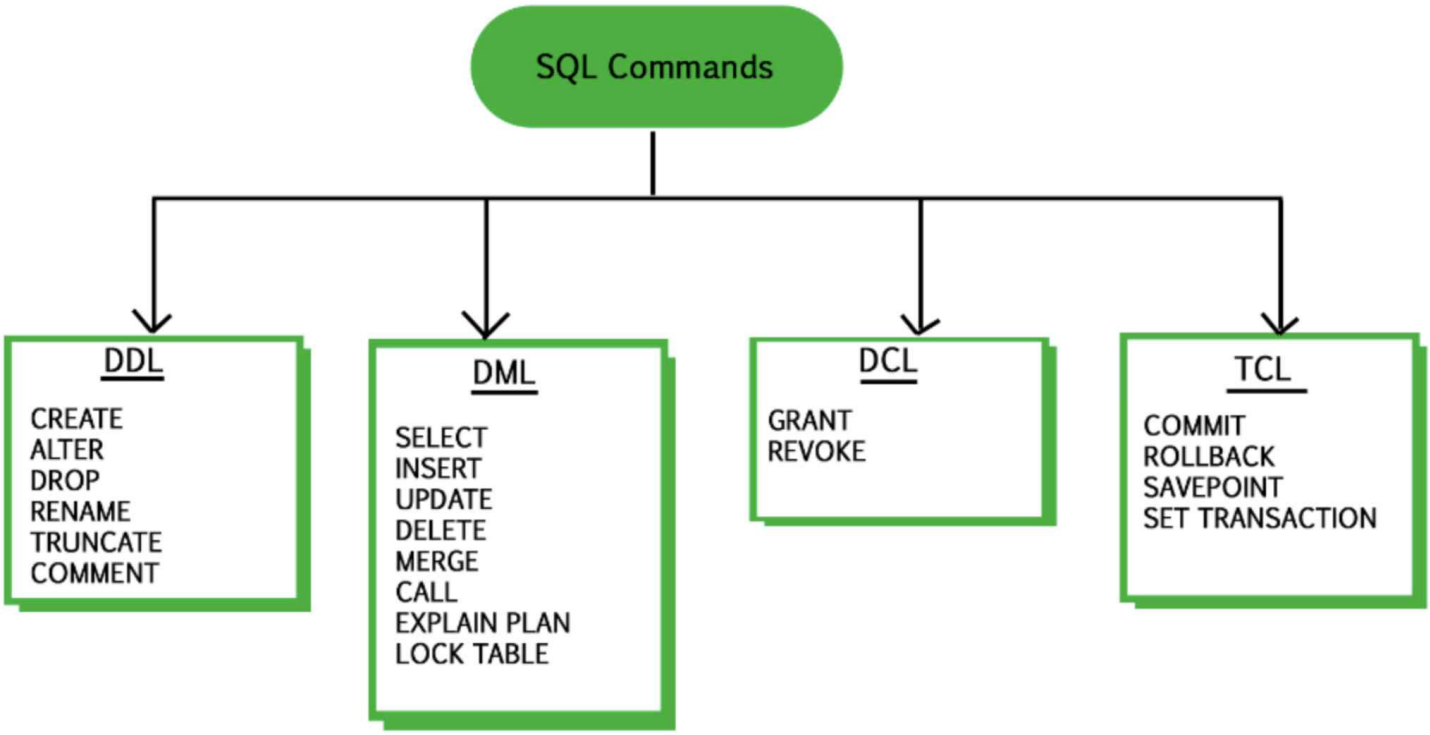
2-Data Definition Language (DDL)

Author: Vincent Lau

Learning Objectives

- Understand Data Types in Column Definition
- Understand the ways to create tables, modify tables & columns

Introduction



Data Definition Language (DDL) is a subset of SQL (Structured Query Language) that is used to define and manage the structure of a database and its objects. DDL statements are used to create, modify, and delete database objects such as tables, indexes, views, and more. DDL is one of the core components of a database management system (DBMS) and is used to define the schema of a database.

Let's start with data types in column definitions.

Column Definition

MySQL provides a variety of data types to accommodate different types of data, including numbers, strings, dates, and more. Each data type has specific characteristics and storage requirements. Here's an introduction to some common data types in MySQL:

Numeric Data Types

1. **INT or INTEGER**: Represents whole numbers. The size can vary (e.g., INT, TINYINT, BIGINT) based on the range of values it can hold.
2. **DECIMAL or NUMERIC**: Used for exact numeric values with decimal points. You can specify precision and scale (total number of digits and digits after the decimal point).
3. **FLOAT**: Used for approximate numeric values with floating-point precision.
    - The `FLOAT` data type is used to store single-precision floating-point numbers.
    - It requires 4 bytes of storage.
    - It provides about 7 decimal places of precision.
    - It can represent a wide range of values, both large and small.

4. **DOUBLE**: Same as FLOAT. DOUBLE has higher precision than FLOAT.
    - The `DOUBLE` data type is used to store double-precision floating-point numbers.
    - It requires 8 bytes of storage.
    - It provides about 15 decimal places of precision.
    - It can also represent a wide range of values with higher precision compared to `FLOAT`.

String Data Types

1. VARCHAR: Variable-length character string. Requires storage for the actual data length plus some overhead.
2. CHAR: Fixed-length character string. Pads the data with spaces to the defined length.
3. TEXT: Used for longer character data, like large amounts of text.

Date and Time Data Types

date '2001-04-25'
time '09:30:00'
timestamp '2001-04-25 10:29:01.45'

1. **DATE**: Stores a date value (year, month & day).
2. TIME: Stores a time value (hour, minute, second).
3. **DATETIME**: Stores both date and time values.
4. **TIMESTAMP**: Stores a timestamp value that represents a specific point in time.

Boolean Data Type

1. BOOLEAN, BOOL, TINYINT(1): Represents true/false or 0/1 values.

Binary Data Types

1. BINARY: Fixed-length binary data.
2. VARBINARY: Variable-length binary data.
3. **BLOB**: Used for storing large binary data.

Other Data Types

1. ENUM: Represents a predefined set of values.
2. SET: Represents a set of possible values.

3. JSON: Stores JSON-formatted data.

Example of Defining a Column with Data Type

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    birthdate DATE,
    gpa DECIMAL(3, 2),
    is_active BOOLEAN
);
```

In this example, the `students` table has columns with various data types such as `INT`, `VARCHAR`, `DATE`, `DECIMAL`, and `BOOLEAN`.

Choosing the appropriate data type is important for efficient storage and data manipulation. You should choose data types based on the nature of the data you're storing and the operations you'll perform on that data. Incorrect data type choices can lead to wasted storage space or performance issues. Always consider the specific needs of your application when selecting data types for your database schema.

Table Definition

Creating a Table

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT
);
```

Adding a Column

```
ALTER TABLE employees ADD email VARCHAR(100);
```

Modifying a Column

```sql
ALTER TABLE employees MODIFY COLUMN first_name VARCHAR(60);
```

Dropping a Column

```sql
ALTER TABLE employees DROP COLUMN email;
```

Creating a Primary Key

```sql
ALTER TABLE employees ADD PRIMARY KEY (employee_id);
```

Creating a Foreign Key

```sql
ALTER TABLE orders ADD FOREIGN KEY (customer_id) REFERENCES customers(customer_id);
-- ALTER TABLE orders ADD CONSTRAINT fk_order FOREIGN KEY (customer_id) REFERENCES customers(customer_id);
```

Creating a Unique Constraint

```sql
ALTER TABLE products ADD CONSTRAINT uc_product_code UNIQUE (product_code);
```

Creating a Check Constraint

```sql
ALTER TABLE employees ADD CONSTRAINT chk_salary CHECK (salary > 0);
```

Creating an Index

- For detail, refer to the later chapter.

```sql
CREATE INDEX idx_department ON employees (department_id);
```

Creating a View

- For detail, refer to the later chapter.

```sql
CREATE VIEW employee_details
AS
SELECT employee_id, CONCAT(first_name, ' ', last_name) AS full_name
FROM employees;
```

Creating a Table Trigger

- For detail, refer to the later chapter.

```sql
CREATE TRIGGER update_salary
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    -- Trigger logic here
END;
```

Creating a Stored Procedure

- For detail, refer to the later chapter.

```sql
DELIMITER //

CREATE PROCEDURE get_employee_count()
BEGIN
    SELECT COUNT(*) FROM employees;
END;
//

DELIMITER ;
```

Creating a Database

```sql
CREATE DATABASE company_db;
```

Dropping a Database

```
1 DROP DATABASE company_db;
```

DDL statements allow you to define and manage the structure of your database. They are used to create tables, indexes, views, triggers, and other database objects. Keep in mind that DDL statements are often powerful and can have a significant impact on your database schema, so use them carefully and ensure you have appropriate backup and testing processes in place before making changes to your production databases.