一、SQL 分类：DDL、DML、DCL、DQL

- **DDL**：数据定义语言，用于定义和管理SQL数据库中的表结构和索引。
- **DML**：数据操作语言, 用于对数据库进行增、删、改、查这些操作。
- **DCL**：数据控制语言，主要用于控制不同权限的数据库用户对数据库表、视图等的访问。
- **DQL**：数据查询语言，用于从数据库获取信息，它也是大多数终端用户及应用程序员最经常使用的SQL语言。

二、DDL（Data Definition Language）：数据定义语言

1、基本操作

1操作数据库：创建库CREATE DATABASE，删除库DROP DATABASE，修改库ALTER DATABASE。

```
CREATE DATABASE TestDB;
DROP DATABASE TestDB;
ALTER DATABASE TestDB COLLATE Chinese_PRC_CI_AS;
```

2 数据类型：数字型（int，tinyint，decimal，float等）、日期和时间类型（Date，TIME等）、字符串类型（char，varchar等）。

```
CREATE TABLE NumTable (
    id INT,
    smallNum TINYINT,
    money DECIMAL(10, 2),
    size FLOAT
);
CREATE TABLE DateTable (
    date_column DATE,
    time_column TIME
);
CREATE TABLE StringTable (
    fixed_length_name CHAR(50),
    variable_length_address VARCHAR(255)
);
```

3 操作表：创建表CREATE TABLE，删除表DROP TABLE，修改表ALTER TABLE，重命名表 RENAME TABLE。

```
CREATE TABLE EmployeeTable (
    ID int,
    Name varchar(255),
    Age int
);

DROP TABLE EmployeeTable;

ALTER TABLE EmployeeTable ADD Email varchar(255);
ALTER TABLE EmployeeTable MODIFY COLUMN Age smallint;
ALTER TABLE EmployeeTable DROP COLUMN Age;

RENAME TABLE EmployeeTable TO StaffTable;
```

三、DML（Data Manipulation Language）：数据操作语言

1 插入数据：利用INSERT INTO语句添加一条或多条记录。

```
INSERT INTO employees (id, name, department_id) VALUES (1, 'Li Ming', 101);
(向employees表中插入一条新的员工记录，id为1，名字是'Li Ming'，部门id是101)

INSERT INTO employees (name, department_id) VALUES ('Wang Gang', 102);
(向employees表中插入一条新的员工记录，id将自动生成，名字是'Wang Gang'，部门id是102)

INSERT INTO employees VALUES (3, 'Zhang San', 103);
(向employees表中插入一条新的员工记录，id为3，名字是'Zhang San'，部门id是103)
```

2 修改数据：用UPDATE语句可以修改表中的数据。

```
UPDATE employees SET department_id = 201 WHERE name = 'Li Ming';
(修改名为'Li Ming'的员工的部门id为201)

UPDATE employees SET name = 'Liu Yan' WHERE id = 2;
(修改id为2的员工的名字为'Liu Yan')

UPDATE employees SET name = 'Zhu Rongji', department_id = 202 WHERE id = 3;

(修改id为3的员工的名字为'Zhu Rongji'和部门id为202)
```

3 删除数据：DELETE FROM语句用于在表中删除一条或者多条记录。

```
DELETE FROM employees WHERE id = 1;
（删除id为1的员工记录）

DELETE FROM employees WHERE name = 'Wang Gang';
（删除名字为'Wang Gang'的员工记录）

DELETE FROM employees WHERE department_id = 202;
（删除部门id为202的所有员工记录）
```

四、DCL（Data Control Language）：数据控制语言

1 创建用户：使用CREATE USER语句创建新的数据库用户。

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
（在"localhost"上创建一个名为"newuser"的新用户，密码是"password"。）
```

2 给用户授权：使用GRANT语句为用户分配访问权限。

```
GRANT SELECT, INSERT, DELETE ON database_name.table_name TO 'username'@'localhost';
（在localhost上给"username"用户赋予运database_name数据库中table_name表的SELECT，INSERT和
DELETE权限。）
```

3 撤销授权：使用REVOKE语句撤销用户的访问权限。

```
REVOKE INSERT ON database_name.table_name FROM 'username'@'localhost';
（在localhost上撤销"username"用户对database_name数据库中table_name表的INSERT权限。）
```

4 查看用户权限：使用SHOW GRANTS语句查看用户当前的权限列表。

```
SHOW GRANTS FOR 'username'@'localhost';
（显示"localhost"上"username"用户的所有权限。）
```

5 删除用户：用DROP USER语句来删除一个用户。

```
DROP USER 'username'@'localhost';
（删除"localhost"上名为"username"的用户。）
```

6 修改用户密码（以root身份）：通过ALTER USER语句可以修改用户的密码。

```
ALTER USER 'username'@'localhost' IDENTIFIED BY 'newpassword';
（修改在"localhost"上username用户的密码为"newpassword"。）
```

五、DQL（Data Query Language）：数据查询语言

1 基础查询：利用SELECT 语句可以查询数据库中的数据。

```
SELECT *
FROM Employee;
（查询Employees表中的所有记录）

SELECT FirstName, LastName
FROM Employee;
（查询Employee表中的FirstName和LastName字段所有记录）
```

2 条件查询：通过使用WHERE子句来设定查询条件。

```
SELECT *
FROM Employee
WHERE Salary > 5000;
（查询Employee表中Salary大于5000的所有记录）

SELECT FirstName, LastName
FROM Employee
WHERE Age <= 30;
（查询Employee表中年龄小于等于30的员工的firstName和lastName字段记录）

SELECT *
FROM Employee
WHERE Salary > 5000
AND Age <= 30;
（查询Employee表中Salary大于5000且年龄小于等于30的所有记录）

SELECT *
FROM Employee
WHERE Salary > 5000
OR Age <= 30;
（查询Employee表中Salary大于5000或年龄小于等于30的所有记录）
```

3 模糊查询：使用LIKE子句，配合通配符%和_进行模糊查询。

```
SELECT *
FROM Employee
WHERE FirstName LIKE 'John%';
（查询Employee表中FirstName以John开头的所有记录）

SELECT *
FROM Employee
WHERE FirstName LIKE '%John%';
（查询Employee表中FirstName包含John的所有记录）

SELECT FirstName, LastName
FROM Employee
```

```
WHERE LastName LIKE '%son_';
（查询Employee表中LastName以son+一个任意字符结束的所有记录）
```

4 字段控制查询：运用DISTINCT关键字进行去重查询。

```
SELECT DISTINCT City
FROM Employee;
（查询Employee表中，City字段去重之后的所有城市记录）

SELECT DISTINCT Age, Salary
FROM Employee
WHERE Age<50;
（查询Employee表中，Age小于50岁的员工的Age和Salary字段组合记录，并进行去重处理）
```

5 排序：用ORDER BY子句按照一个或多个列进行排序。

单列排序：

```
SELECT *
FROM Employee
ORDER BY Salary;
（按'工资'列的升序返回Employee表中的所有行。）
```

多列排序：

```
SELECT *
FROM Employee
ORDER BY Salary, Age DESC;
（首先根据'工资'列的升序对Employee表中的行进行排序，然后在工资相同的情况下，根据'Age'列的降序进行排序。）
```

6 聚合函数：包括COUNT，SUM，MAX，MIN，AVG等函数。

COUNT：

```
SELECT COUNT(*)
FROM Employee;
（返回Employee表的总行数。）
```

SUM：

```
SELECT SUM(Salary)
FROM Employee;
（返回Employee表中所有员工的薪水总和。）
```

MAX：

```
SELECT MAX(Age)
FROM Employee;
（返回Employee表中员工的最大年龄。）
```

MIN：

```
SELECT MIN(Age)
FROM Employee;
（返回Employee表中员工的最小年龄。）
```

AVG：

```
SELECT AVG(Salary)
FROM Employee;
（返回Employee表中员工的平均薪水。）
```

7 分组查询：GROUP BY子句是用于结合聚合函数，依据一个或多个列进行分组。

按单列分组：

```
SELECT Department, COUNT(*)
FROM Employee
GROUP BY Department;
（返回每个部门的员工数量。）
```

按多列分组：

```
SELECT Department, JobTitle, AVG(Salary)
FROM Employee
GROUP BY Department, JobTitle;
（返回每个部门及职位下员工的平均薪水。）
```

分组后过滤：

```
SELECT Department, JobTitle, AVG(Salary)
FROM Employee
GROUP BY Department, JobTitle
HAVING AVG(Salary) > 5000;
（返回每个部门及职位下员工的平均薪水大于5000的部分。）
```

8 LIMIT：用来限定查询结果的起始行，以及总行数。

简单LIMIT：

```
SELECT *
FROM Employee
ORDER BY Salary DESC
LIMIT 10;
（返回前10个工资最高的员工。）
```

带偏移量的LIMIT：

```
SELECT *
FROM Employee
ORDER BY Salary DESC
LIMIT 10 OFFSET 5;
（返回工资排名第6到第15的10个员工。）
```

9 多表连接查询：

内连接：INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID = Customers.CustomerID;
```

左连接：LEFT JOIN

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

右连接：RIGHT JOIN

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
RIGHT JOIN Customers
ON Orders.CustomerID = Customers.CustomerID;
```

全外连接：FULL JOIN

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

笛卡尔积：CROSS JOIN

```sql
SELECT Products.ProductName, Suppliers.SupplierName
FROM Products
CROSS JOIN Suppliers;
```