数据库系统 期末速通教程

3. MySQL速通

参考: https://www.bilibili.com/video/BV1Kr4y1i7ru/

3.1 MySQL的启动、关闭与客户端连接

3.1.1 启动与关闭

```
net start mysql80  # 启动
net stop mysql80  # 关闭
```

3.1.2 客户端连接

```
mysql [-h 127.0.0.1] [-P 3306] -u root -p
```

输入上述命令行后输入密码即可.

3.2.SQL

3.2.1 SQL 的分类

[**结构化查询语言**, **Structural Query Language**, **SQL**]

(1) 分类:

分类	全称	缩写	说明
数据定义语言	Data Definition Language	DDL	定义 DB 对象(数据库、表、字段)
数据操作语言	Data Manipulation Language	DML	对数据表中的数据进行增删改
数据查询语言	Data Query Language	DQL	查询 DB 中表的记录
数据控制语言	Data Control Language	DCL	创建DB用户、控制DB的访问权限

(2) 语句不区分大小写.

3.2.2 DDL

3.2.2.1 DB 的查询、创建、删除、使用

```
SHOW DATABASES;  # 查询所有 DB
SELECT DATABBASE();  # 查询当前在哪个 DB 中

CREATE DATABASE [IF NOT EXISTS] DB名 [DEFAULT CHARSET 字符集] [COLLATE 排序规则];
    # 创建 DB , 若要创建的 DB 已存在则报错, 可加 "IF NOT EXISTS" 避免
    # 指定字符集为 utf8mb4 即可以每个数据 4 Bytes 存储数据

DROP DATABASE [IF EXISTS] DB名;  # 删除 DB
USE DB名;  # 切换到指定 DB , 以使用该 DB
```

3.2.2.2 数据表的查询

进入对应 DB 后, 可用如下命令:

```
SHOW TABLES;  # 查询当前 DB 的所有表
DESC 表名;  # 查询表结构
SHOW CREATE TABLE;  # 查询指定表的建表语句

CREATE TABLE 表名(
    字段1 字段1类型 [COMMENT 字段1注释],
    字段2 字段1类型 [COMMENT 字段1注释],
    ...
    字段n 字段n类型 [COMMENT 字段n注释]  # 此处无逗号
) [COMMENT 表注释];  # 创建表
```

[**例**] 在DB "myDB" 中创建下面的数据表 "tb_user" 并:

① 验证是否创建成功.

② 查询表结构.

③ 查询表的建表语句.

id	name	age	gender
1	AA	28	男
2	BB	68	男
3	CC	32	男

[**解**]

```
USE myDB;

CREATE TABLE tb_user(
    id int COMMENT '编号',
    name varchar(50) COMMENT '姓名',
    gender char(1) COMMENT '性别'
) COMMENT '用户表';

SHOW TABLES;

DESC tb_user;  # 查看表 tb_user 的表结构

SHOW CREATE TABLE tb_user;
```

3.2.2.3 数据类型

3.2.2.3.1 数值

类型	描述	大小 / Byte(s)
TINYINT	小整数	1
SMALLINT	大整数	2
MEDIUMINT	大整数	3
INT 或 INTERGER	大整数	4
BIGINT	极大整数	8
FLOAT	单精度浮点数	4
DOUBLE	双精度浮点数	8
DECIMAL	小数值(精确定点数)	/

可在整型后加 "UNSIGNED" 表示无符号整数.

3.2.2.3.2 字符串(文本)

字符串类型使用时需指定长度, 超过指定的长度会报错.

类型	描述	备注
CHAR	定长字符串	长度不足用空格补足, 也占固定空间, 适用于性别等长度基本确定的字符串
VARCHAR	变长字符串	按实际长度分配空间, 适用于用户名等长度不确定的字符串
TINYTEXT	短文本字符串	/
TEXT	长文本数据	/
MEDIUMTEXT	中等长文本数据	/
LONGTEXT	极大文本数据	/

3.2.2.3.3 日期时间

类型	描述	格式	大小
DATE	日期值	YYYY-MM-DD	3
TIME	时间值或持续时间	HH:MM:SS	3
YAER	年份值	YYYY	1
DATETIME	混合日期和时间值	YYYY-MM-DD HH:MM:SS	8
TIMESTAMP	混合日期和时间值、时间戳	YYYY-MM-DD HH:MM:SS	4

3.2.2.4 数据表的修改

3.2.2.4.1 添加字段

```
ALTER TABLE 表名 ADD 字段名 类型(长度) [COMMENT 注释] [约束];  # 添加字段
```

[**例**] 给 "EMP" 表添加一个 "昵称" 字段 "NICKNAME" , 类型为 VARCHAR(20) .

```
ALTER TABLE EMP ADD NICKNAME VARCHAR(20) COMMENT '昵称';
```

3.2.2.4.2 修改字段

```
ALTER TABLE 表名 MODIFY 字段名 新类型(长度);  # 修改数据类型

ALTER TABLE 表名 CHANGE 旧字段名 新字段名 类型(长度) [COMMENT 注释] [约束];  # 修改字段名和字段类型
```

[**例**] 将 "EMP" 表的 "昵称" 字段 "NICKNAME" 修改为 "用户名" 字段 "USERNAME" , 类型为 VARCHAR(30) .

```
ALTER TABLE EMP CHANGE NICKNAME USERNAME VARCHAR(30) COMMENT '用户名';
```

3.2.2.4.3 删除字段

```
ALTER TABLE 表名 DROP 字段名;  # 删除字段
```

[**例**] 删除 "EMP" 表中的字段 "USERNAME" .

```
ALTER TABLE EMP DROP USERNAME;
```

3.2.2.4.4 修改表名

```
ALTER TABLE 表名 RENAME TO 新表名;  # 修改表名
```

[**例**] 将 "EMP" 表的表名修改为 "EMPLOYEE" .

```
ALTER TABLE EMP RENAME TO EMPLOYEE;
```

3.2.2.4.5 删除表

```
DROP TABLE [IF EXISTS] 表名;  # 删除表

TRUNCATE TABLE 表名;  # 删除指定表, 并重创建该表
```

3.2.3 DML

3.2.3.1 添加数据

```
INSERT INTO 表名(字段名1, 字段名2, ...) VALUES (值1, 值2, ...);  # 给指定字段添加数据

INSERT INTO 表名 VALUES (值1, 值2);  # 给全部字段添加数据

# 批量添加数据
INSERT INTO 表名(字段名1, 字段名2, ...) VALUES (值1, 值2, ...), (值1, 值2, ...), ...;
INSERT INTO 表名 VALUES (值1, 值2, ...), (值1, 值2, ...), ...;
```

[**注1**] 插入数据时, 指定的字段顺序与值的顺序一一对应.

[**注2**] 字符串和日期型数据应包含在引号中.

[**注3**] 插入的数据大小应在字段可表示的范围内.

3.2.3.2 修改数据

```
UPDATE 表名 SET 字段名1 = 值1, 字段名2 = 值2, ... [WHERE 条件];  # 修改数据
```

[**注**] 修改语句无条件时会修改整张表的数据.

3.2.3.3 删除数据

```
DELETE FROM 表名 [WHERE 条件];  # 删除数据
```

[**注1**] 删除语句无条件时会删除整张表的数据.

[**注2**] 删除语句不能删除某字段的值. 若需删除某字段的值, 可用 `UPDATE` 语句将该字段的值置为 NULL .

3.2.4 DQL 单表查询

3.2.4.1 基础查询

DQL的语法格式:

(1) 基本查询: `SELECT` 、 `FROM` .

(2) 条件查询: `WHERE` .

(3) 聚合函数: `COUNT` 、 `MAX` 、 `MIN` 、 `AVG` 、 `SUM` .

(4) 分组查询: `GROUP BY` .

(5) 排序查询: `ORDER BY` .

(6) 分页查询: `LIMIT` .

编写顺序:

```
SELECT
    字段列表
FROM
    表名列表
WHERE
    条件列表
GROUP BY
    分组字段列表
HAVING
    分组条件列表
ORDER BY
    排序字段列表
LIMIT
    分页参数
```

执行顺序:

```
FROM
    表名列表
WHERE
    条件列表
GROUP BY
    分组字段列表
HAVING
    分组条件列表
SELECT
    字段列表
ORDER BY
    排序字段列表
LIMIT
    分页参数
```

基础查询语法:

```
SELECT 字段1, 字段2, ... FROM 表名;  # 查询指定字段
SELECT * FROM 表名;  # 查询所有字段

SELECT 字段1 [AS 别名1], 字段2 [AS 别名2], ... FROM 表名;  # 为字段设置别名, AS 可省略

SELECT DISTINCT 字段列表 FROM 表名;  # 去除重复记录
```

3.2.4.2 条件查询

```
SELECT 字段列表 FROM 表名 WHERE 条件列表;  # 条件查询
```

条件:

(1) 单条件

比较运算符	功能
>	大于
>=	大于等于
<	小于
<=	小于等于
=	等于
<> 或 !=	不等于
BETWEEN 最小值 AND 最大值	在某范围内的值(含最小值、最大值)
IN(...)	任一在列表中的值
LIKE 占位符	模糊匹配('_' 匹配单个字符, '%' 匹配多个字符)
IS NULL	是 NULL

(2) 多条件

逻辑运算符	功能
AND 或 &&	且
OR 或 ||	或
NOT 或 !	非, 否

[注] 注意以下的等价性:

① $=$ SOME 等价于 IN .

② $<>$ SOME 不等价于 NOT IN .

③ $<>$ ALL 等级与 NOT IN .

[**例**] 在 "emp" 表中进行如下查询.

(1) 查询年龄不超过 20 的员工信息.

```sql
SELECT * FROM emp WHERE age <= 20;
```

(2) 查询无身份证号的员工信息.

```sql
SELECT * FROM emp WHERE idcard IS NULL;
```

(3) 查询有身份证号的员工信息.

```sql
SELECT * FROM emp WHERE idcard IS NOT NULL;
```

(4) 查询年龄在 $[15, 20]$ 范围内的员工信息.

```sql
SELECT * FROM emp WHERE age >= 15 AND age <= 20;
```

或

```sql
SELECT * FROM emp WHERE age BETWEEN 15 AND 20;
```

(5) 查询年龄是 18 或 20 或 40 的员工信息.

```sql
SELECT * FROM emp WHERE age = 18 OR age = 20 or age = 40;
```

或

```sql
SELECT * FROM emp WHERE age IN(18, 20, 40);
```

(6) 查询姓名是两个字的员工信息.

'_' 匹配恰一个字符.

```sql
SELECT * FROM emp WHERE name LIKE '__';
```

(7) 查询身份证号最后一位是 'X' 的员工信息.

'\%' 匹配零个或多个字符.

```sql
SELECT * FROM emp WHERE idcard LIKE '%X';
```

3.2.4.3 聚合函数

聚合函数将表的每一列作为整体, 进行纵向计算.

常用的聚合函数:

聚合函数	功能
COUNT	统计数量
MAX	最大值
MIN	最小值
AVG	平均值
SUM	求和

[**注**] NULL 值不参与聚合函数的运算.

聚合函数语法:

```
SELECT 聚合函数(字段列表) FROM 表名;
```

[**例**] 在 "emp" 表中进行如下查询.

(1) 统计员工数.

```
SELECT COUNT(*) FROM emp;
```

(2) 统计员工的平均年龄.

```
SELECT AVG(age) FROM age;
```

(3) 统计西安地区的员工的年龄之和.

```
SELECT SUM(age) FROM emp WHERE workspace = '西安';
```

3.2.4.4 分组查询

```
SELECT 字段列表 FROM 表名 [WHERE 分组前过滤条件] GROUP BY 分组字段名 [HAVING 分组后过滤条件];
```

[**注1**] `WHERE` 和 `HAVING` 的区别:

① 执行时机不同: `WHERE` 在分组前过滤, 不满足条件的不参与分组; `HAVING` 在分组后过滤.

② 判断条件不同: `WHERE` 不可用聚合函数作为条件, 但 `HAVING` 可以.

[**注2**] 执行顺序: `WHERE` 、聚合函数、 `HAVING` .

[**注3**] 分组后, 查询字段一般为聚合函数和分组字段, 查询其他字段无意义.

[**例1**] 在 "emp" 表中进行如下查询.

(1) 按性别分组, 统计男性、女性员工的数量.

```sql
SELECT gender, COUNT(*) FROM emp GROUP BY gender;
```

(2) 按性别分组, 统计男性、女性员工的平均年龄.

```sql
SELECT gender, AVG(age) FROM emp GROUP BY gender;
```

(3) 对年龄 < 45 的员工, 按工作地址分组, 统计员工数量 ≥ 3 的工作地址.

```sql
SELECT work_address, COUNT(*) address_count  # 省略 AS
FROM emp
WHERE age < 45
GROUP BY work_address
HAVING address_count >= 3;
```

[**例2**] 在学生表 "students" 、课程表 "cources" 、修课表 "enrollment" 中进行如下查询.

(1) 查询每位学生所有课程的平均分.

```sql
SELECT AVG(grade) FROM enrollment GROUP BY sid;
```

(2) 查询所有课程的平均分 > 80 的学生的信息.

```sql
SELECT AVG(grade) avgGrade FROM enrollment
GROUP BY sid
HAVING avgGrade > 80;
```

3.2.4.5 排序查询

排序语句支持多关键字排序.

```sql
SELECT 字段列表 FROM 表名 ORDER BY 字段1 排序方式1, 字段2 排序方式2, ...;
```

排序方式:

(1) `ASC` : 非降序(默认).

(2) `DESC` : 非升序.

[**例**] 在 "emp" 表中进行如下查询.

(1) 根据年龄将员工非降序排列.

```
SELECT * FROM emp ORDER BY age ASC;
```

或

```
SELECT * FROM emp ORDER BY age;
```

(2) 根据年龄将员工非升序排列.

```
SELECT * FROM emp ORDER BY age DESC;
```

(3) 根据年龄将员工非降序排列, 年龄相同的员工按入职时间非升序排列.

```
SELECT * FROM emp ORDER BY age ASC, entrydate DESC;
```

3.2.5 DCL

[**例3.2.5.1**]

```
GRANT select, insert ON stock TO smith;
GRANT all ON product TO fox;
GRANT update(stock_code, stock_price) ON stock TO brown;
GRANT select ON branch To public;
```

[**例3.2.5.2**]

```
REVOKE select, insert ON stock FROM smith;
REVOKE all ON product FROM fox;
REVOKE update(stock_code, stock_price) ON stock FROM brown;
REVOKE select ON branch FROM public;
```

3.3 ?

3.4 约束

3.4.1 约束的分类

约束作用在表的字段上, 可在创建、修改表时添加约束.

一个字段可添加多个约束, 多个约束间用空格分隔.

约束	描述	关键字
非空约束	限制字段数据不能为 NULL	NOT NULL
唯一约束	保证该字段的所有数据不重复, 可用在身份证号、用户名等	UNIQUE
主键约束	主键是一行数据的唯一标识, 要求非空且唯一	PRIMARY KEY
默认约束	保存数据时, 若未指定该字段的值, 则用默认值	DEFAULT
检查约束	保证字段值满足某条件	CHECK
外键约束	建立两张表间的连接, 保证数据的一致性和完整性	FOREIGN KEY
自增	整型数据自动增长, 可用在 id 等	AUTO_INCREMENT

3.4.2 外键约束

删除外键后键值还在, 只是删除了关联.

```
CREATE TABLE 表名(
    字段名 数据类型,
    ...
    [CONSTRAINT] [外键名称] FOREIGN KEY(外键字段名) REFERENCES 主表(主表列名)
);  # 在建表时添加外键

ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY(外键字段名) REFERENCES 主表(主表列名);  # 添加外键
ALTER TABLE 表名 DROP FOREIGN KEY 外键名称;  # 删除外键
```

[**例1**] 包包由设计师设计, 则设计师表 "designers" 与包包表 "bags" 的间存在外键约束.

```
-- 设计师表
CREATE TABLE Designers (
  id INT(3) NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  daily DECIMAL(3, 2) NOT NULL,

  PRIMARY KEY(id)
) ENGINE = InnoDB;

-- 包包表
CREATE TABLE Bags (
  id INT(3) NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  type VARCHAR(10) NOT NULL,
  color VARCHAR(10) NOT NULL,
  did INT(3) NOT NULL,
  manufacturer VARCHAR(20) NOT NULL,
  available BOOLEAN DEFAULT TRUE,

  PRIMARY KEY(id),
  FOREIGN KEY(did) REFERENCES Designers(id)
) ENGINE = InnoDB;
```

[**例2**]

(1) 建主键为 (ssn, cno) 的表 $\text{sc}(\text{ssn}, \text{cno}, \text{grade})$, 其中 ssn 是外键, 关联学生表 "students" 的主键 ssn ; cno 是外键, 关联课程表 "cources" 的主键 cno .

```sql
CREATE TABLE sc (
  ssn NUMBER,
  cno NUMBER,
  grade NUMBER,
  PRIMARY KEY(ssn, cno),
  FOREIGN KEY(ssn) REFERENCES students(ssn),
  FOREIGN KEY(cno) REFERENCES cources(cno)
) ENGINE = InnoDB;
```

(2) 建主键为 (ssn, cno) 的表 $\text{sc}(\text{ssn}, \text{cno}, \text{grade}, \text{takeCourseDate})$, 其中 ssn 是外键, 关联学生表 "students" 的主键 ssn ; cno 是外键, 关联课程表 "cources" 的主键 cno . 检查 grade 是否 > 0 . 设置 takeCourseDate 的默认值为当前日期.

```sql
CREATE TABLE sc (
  ssn NUMBER,
  cno NUMBER,
  grade NUMBER CHECK(grade > 0),
  takeCourseDate DATE DEFAULT GETDATE(),
  PRIMARY KEY(ssn, cno),
  FOREIGN KEY(ssn) REFERENCES students(ssn),
  FOREIGN KEY(cno) REFERENCES cources(cno)
) ENGINE = InnoDB;
```

3.4.3 DQL多表查询

3.4.3.1 多表关系

[**一对多 / 多对一**]

(1) 案例: 部门与员工的关系, 即一个部门对应多个员工, 一个员工对应一个部门.

(2) 实现: 在多的一方建立外键, 指向另一方的主键.

[**多对多**]

(1) 案例: 学生与课程的关系, 即一个学生可选修多门课程, 一门课程可供多个学生选修.

(2) 实现: 建立中间表, 其至少包含两外键, 分别关联两表的主键.

[**一对一**] 一对一关系多用于单表拆分, 即将一张表的基础字段放在一张表中, 其他详情字段放在另一张表中, 提高效率.

(1) 案例: 用户与用户详情的关系.

(2) 实现: 在任一方加入外键, 关联另一方的主键, 并设置该外键为 `UNIQUE` .

3.4.3.2 多表查询概述

多表查询实际是求多张表的 Cartesian 积. 为消除冗余的 Cartesian 积, 应将表间的关系作为过滤条件.

[**例**] 查询 "emp" 表中的员工在 "dept" 表中对应的部门.

```
SELECT * FROM emp, dept WHERE emp.dept_id = dept.id;
```

[**多表查询的分类**]

(1) 连接查询:

① 内连接: 查询集合A、B的交集的数据.

② 外连接:

(i) 左外连接: 查询左表的所有数据和两表交集部分的数据.

(ii) 右外连接: 查询右表的所有数据和两表交集部分的数据.

③ 自连接: 当前表与自身的连接查询, 需用表别名.

(2) 子查询.

3.4.3.3 内连接

```
SELECT 字段列表 FROM 表1, 表2, ... WHERE 条件 ...;  # 隐式内连接

SELECT 字段列表 FROM 表1 [INNER] JOIN 表2 ON 连接条件 ... ;  # 显式内连接
```

[**例**] 在 "emp" 表和 "dept" 表中进行如下查询.

(1) 查询每个员工的姓名和关联的部门的名称, 用隐式内连接实现.

```
SELECT emp.name, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id;

# 用表别名
SELECT e.name, d.name
FROM emp e, dept d
WHERE e.dept_id = d.id;
```

(2) 查询每个员工的姓名和关联的部门的名称, 用显式内连接实现.

```
SELECT e.name, d.name
FROM emp e
INNER JOIN dept d  # INNER 可省略
ON e.dept_id = d.id;
```

3.4.3.4 外连接

```
SELECT 字段列表 FROM 表1 LEFT [OUTER] JOIN 表2 ON 条件 ...;  # 左外连接

SELECT 字段列表 FROM 表1 RIGHT [OUTER] JOIN 表2 ON 条件 ...;  # 右外连接
```

[**例**] 在 "emp" 表和 "dept" 表中进行如下查询.

(1) 查询 "emp" 表中的所有数据和对应的部门信息.

```
SELECT e.*, d.name
FROM emp e
LEFT OUTER JOIN dept d  # OUTER 可省略
ON e.dept_id = d.id;
```

(2) 查询 "dept" 表中的所有数据和对应的员工信息.

```
SELECT e.*, d.name
FROM emp e
RIGHT OUTER JOIN dept d  # OUTER 可省略
ON e.dept_id = d.id;
```

3.4.3.5 自连接

```
SELECT 字段列表 FROM 表名 别名A JOIN 表名 别名B ON 条件 ...;
```

[**注**] 自连接可是内连接也可是外连接.

[**例1**] 在 "emp" 表中进行如下查询.

(1) 查询员工及其领导的名字.

```
SELECT a.name, b.name
FROM emp a, emp b
WHERE a.manager_id = b.id;
```

(2) 查询所有员工及其领导的名字, 若某员工无领导, 也需输出.

```
SELECT a.name '员工', b.name '领导'
FROM emp a
LEFT JOIN emp b
ON a.manager_id = b.id;
```

[**例2**] 在 "emp" 表中进行查询: 工资比自己的上司高的员工的姓名和工资.

```sql
SELECT worker.name, worker.salary
FROM emp worker, emp manager
WHERE worker.manager_id = manager.id AND worker.salary > manager.salary;
```

3.4.3.6 联合查询

联合查询可合并多次查询的结果, 形成一个新的查询结果集.

```sql
SELECT 字段列表 FROM 表A ...
UNION [ALL]
SELECT 字段列表 FROM 表B ...;
```

[**注1**] 联合查询时, 多张表的列数需相等, 字段类型需相同.

[**注2**] `UNION ALL` 只合并所有查询的结果, 而 `UNION` 会对合并后的结果去重.

[**注3**] 类似的集合运算操作符还有: 交集 `INTERSECT` 、差集 `MINUS` .

[**例**] 在 "emp" 表和 "dept" 表中进行如下查询: 查询薪资 < 5000 的员工和年龄 > 50 的员工.

```sql
# 只合并不去重
SELECT * FROM emp WHERE salary < 5000
UNION ALL
SELECT * FROM emp WHERE age > 50;

# 合并并去重
SELECT * FROM emp WHERE salary < 5000
UNION
SELECT * FROM emp WHERE age > 50;
```

3.4.3.7 子查询(嵌套查询)

3.4.3.7.1 子查询概述

[**子查询**] SQL语句中嵌套 `SELECT` 语句称为**嵌套查询**或**子查询**.

(1) 子查询外部的语句可是 `INSERT` 、`UPDATE` 、`DELETE` 、`SELECT` 等.

```sql
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

(2) 按子查询的结果分类:

① 标量子查询: 子查询的结果是一个值.

② 列子查询: 子查询结果为一列.

③ 行子查询: 子查询结果为一行.

④ 表子查询: 子查询结果为多行多列的表.

(3) 按子查询的位置分类:

① `WHERE` 之后.

② `FROM` 之后.

③ `SELECT` 之后.

3.4.3.7.2 标量子查询

标量子查询返回的结果是单个值, 如数字、字符串、日期等.

常用操作符: `=` 、 `<>` 、 `>` 、 `>=` 、 `<` 、 `<=` .

[**例**] 在 "emp" 表和 "dept" 表中进行如下查询.

(1) 查询 "销售部" 的所有员工信息.

先查询 "销售部" 的部门ID, 再查询部门ID为该ID的员工信息.

```
SELECT * FROM emp
WHERE dept_id = (SELECT id FROM dept WHERE name = '销售部');
```

(2) 查询在 "东方白" 之后入职的员工信息.

```
SELECT * FROM emp
WHERE entry_date > (SELECT entry_date FROM emp WHERE name = '东方白');
```

3.4.3.7.3 列子查询

列子查询返回的结果是一列(可能多行).

常用操作符: `IN` 、 `NOT IN` 、 `ANY` 、 `SOME` 、 `ALL` .

操作符	功能
IN	在指定的集合内
NOT IN	不在指定的集合内
ANY	子查询返回的列表中, 有任一满足即可
SOME	同 ANY
ALL	子查询返回的列表中的所有值都需满足

[**例**] 在 "emp" 表和 "dept" 表中进行如下查询.

(1) 查询 "销售部" 和 "市场部" 的所有员工的信息.

```
SELECT * FROM emp
WHERE dept_id IN (SELECT id FROM dept WHERE name = '销售部' or name = '市场部');
```

(2) 查询比 "财务部" 的所有员工工资都高的员工的信息.

① 查询所有 "财务部" 的员工的工资.

```
SELECT salary FROM emp
WHERE dept_id = (SELECT id FROM dept WHERE name = '财务部');
```

② 查询比 "财务部" 的所有员工工资都高的员工的信息.

```
SELECT * FROM emp
WHERE salary > ALL (SELECT salary FROM emp
    WHERE dept_id = (SELECT id FROM dept WHERE name = '财务部')
  );
```

3.4.3.7.4 行子查询

行子查询返回的结果是一行(可能多列).

常用操作符: `=` 、 `<>` 、 `IN` 、 `NOT IN` .

[**例**] 在 "emp" 表和 "dept" 表中进行如下查询: 查询与 "张无忌" 的薪资和领导都相同的员工信息.

```
SELECT * FROM emp
WHERE (salary, manager_id) = (
    SELECT salary, manager_id FROM emp WHERE name = '张无忌'
  );
```

3.4.3.7.5 表子查询

表子查询返回的结果是多行多列的表格, 结果常作为中间表格再进行查询.

常用操作符: `IN` .

[**例**] 在 "emp" 表和 "dept" 表中进行如下查询.

(1) 查询与 "鹿仗客" 或 "宋远桥" 的职位和薪资相同的员工信息.

```
SELECT * FROM emp
WHERE (job, salary) IN (SELECT job, salary FROM emp WHERE name = '鹿仗客' or name = '宋远
桥');
```

(2) 查询入职日期在 "2006-01-01" 后的员工信息及其部门信息.

① 查询入职日期在 "2006-01-01" 后的员工信息.

```
SELECT * FROM emp WHERE entry_date > '2006-01-01';
```

② 注意员工无部门也需输出, 故用左外连接.

```sql
SELECT e.*, d.* FROM
(SELECT * FROM emp WHERE entry_date > '2006-01-01') e
LEFT JOIN dept d ON e.dept_id = d.id;
```

3.4.3.8 除法

set()	EXISTS()	NOT EXISTS()
非空	TRUE	FALSE
空	FALSE	TRUE

[**例1**] 在学生表 "students" 和修课表 "enrollment" 中进行如下查询:

(1) 查询至少修了一门课的学生的信息.

```sql
-- 子查询
SELECT * FROM students WHERE id IN
  (SELECT sid FROM enrollment);

-- 除法
SELECT * FROM students s WHERE EXISTS
  (SELECT * FROM enrollment e WHERE e.sid = s.id);
```

(2) 查询没有修编号 "21003001" 的课程的学生的信息.

```sql
-- 子查询
SELECT * FROM students WHERE id NOT IN
  (SELECT sid FROM enrollment WHERE cid = '21003001');

-- 除法
SELECT * FROM students WHERE NOT EXISTS
  (SELECT * FROM enrollment WHERE id = sid AND cid = '21003001');
```

[**例2**] 在学生表 "students" 、课程表 "courses" 和修课表 "enrollment" 中查询修过所有课程的学生的信息.

```sql
SELECT * FROM students s WHERE NOT EXISTS
  (SELECT * FROM cources c WHERE NOT EXISTS
    (SELECT * FROM enrollment WHERE sid = s.id AND cid = c.id));
```

3.5 存储对象

3.5.1 视图

3.5.1.1 视图的创建、查询、修改、删除

[**视图**]

(1) 视图(View)是一种虚拟的表, 其中的数据不在DB中实际存在, 行和列的数据来自定义视图的查询中使用的表, 且在使用视图时动态生成. 视图只保存查询的SQL逻辑, 不保存查询结果.

(2) 创建.

```
CREATE [OR REPLACE] VIEW 视图名[(列名列表)] AS SELECT语句 [WITH [CASCADED | LOCAL] CHECK OPTION];  # 创建视图
```

(3) 查询: 同查询数据表.

```
SHOW CREATE VIEW 视图名;  # 查询建视图语句

SELECT * FROM 视图名;  # 查看视图数据
```

(4) 修改.

```
CREATE [OR REPLACE] VIEW 视图名[(列名列表)] AS SELECT语句 [WITH [CASCADED | LOCAL] CHECK OPTION];
```

或

```
ALTER VIEW 视图名[(列名列表)] AS SELECT语句 [WITH [CASCADED | LOCAL] CHECK OPTION];
```

(5) 删除.

```
DROP VIEW [IF EXISTS] 视图名 [, 视图名] ... ;
```

[**例**] 在 "student" 表中进行如下查询:

(1) 创建一个展示 "student" 表中 $id \leq 10$ 的学生的 id 和 name 的视图 "stu_v1" .

```
CREATE OR REPLACE VIEW stu_v1 AS SELECT id, name FROM student WHERE id <= 10;
```

(2) 查询视图 "stu_v1" 中的建视图语句和数据.

```
SHOW CREATE VIEW stu_v1;

SELECT * FROM stu_v1;
```

(3) 查询视图 "stu_v1" 中 $id < 3$ 的学生信息.

```
SELECT * FROM stu_v1 WHERE id < 3;
```

(4) 插入一个数据 (6, 'Tom') , 数据会被添加到 "student" 表中, 查询视图 "stu_v1" 时可显示.

```
INSERT INTO stu_v1 VALUES(6, 'Tom');
```

插入一个数据 (30, 'Tom') , 数据会被添加到 "student" 表中, 查询视图 "stu_v1" 时不显示, 因为视图要求 $id \leq 10$.

```
INSERT INTO stu_v1 VALUES(30, 'Tom');
```

若要在插入数据 (30, 'Tom') 时报错, 可在建视图时加入 `cascaded` 或 `local` 检查选项.

```
CREATE OR REPLACE VIEW stu_v1 AS SELECT id, name FROM student WHERE id <= 10 WITH CASCADED
CHECK OPTION;
```

(5) 将视图 "stu_v1" 修改为展示 "student" 表中 $id \leq 10$ 的学生的 id 、name 和 no 的视图.

```
CREATE OR REPLACE VIEW stu_v1 AS SELECT id, name, no FROM student WHERE id <= 10;
```

或

```
ALTER VIEW stu_v1 AS SELECT id, name, no FROM student WHERE id <= 10;
```

(6) 删除视图 "stu_v1" .

```
DROP VIEW IF EXISTS stu_v1;
```

3.5.1.2 检查选项

使用检查选项 `WITH ... CHECK OPTION` 创建视图时, MySQL会通过视图检查正在更改的每个行是否符合视图定义.

MySQL允许基于另一视图创建视图, 它还会检查依赖视图中的规则以保持一致性.

为确定检查范围, MySQL提供了两个检查选项: `cascaded` 和 `local` , 默认为 `cascaded` .

3.5.1.2.1 cascaded

[例]

(1) 创建一个展示 "student" 表中 $id \leq 20$ 的学生的 id 和 name 的视图 "stu_v1" , 无检查选项.

```
CREATE OR REPLACE VIEW stu_v1 AS SELECT id, name FROM student WHERE id <= 20;
```

(2) 向视图 "stu_v1" 中插入数据.

① 插入数据 (5, 'Tom') , 成功.

```
INSERT INTO stu_v1 VALUES(5, 'Tom');
```

② 插入数据 (25, 'Tom') , 成功, 因为无检查选项.

```
INSERT INTO stu_v1 VALUES(25, 'Tom');
```

(3) 创建一个展示视图 "stu_v1" 中 $id \geq 10$ 的学生的 id 和 name 的视图 "stu_v2" , 使用检查选项 `cascaded` .

```
CREATE OR REPLACE VIEW stu_v2 AS SELECT id, name FROM stu_v1 WHERE id >= 10 WITH CASCADED
CHECK OPTION;
```

(4) 向视图 "stu_v2" 中插入数据.

① 插入数据 (7, 'Tom') , 报错, 因为不满足视图 "stu_v2" 的检查条件 $id \geq 10$.

```
INSERT INTO stu_v2 VALUES(7, 'Tom');
```

② 插入数据 (26, 'Tom') , 报错, 因为不满足视图 "stu_v1" 的检查条件 $id \leq 20$.

```
INSERT INTO stu_v2 VALUES(26, 'Tom');
```

③ 插入数据 (15, 'Tom') , 成功, 因为同时满足视图 "stu_v1" 的检查条件 $id \leq 20$ 和视图 "stu_v2" 的检查条件 $id \geq 10$.

```
INSERT INTO stu_v2 VALUES(15, 'Tom');
```

(5) 创建一个展示视图 "stu_v2" 中 $id \leq 15$ 的学生的 id 和 name 的视图 "stu_v3" , 无检查选项.

```
CREATE OR REPLACE VIEW stu_v3 AS SELECT id, name FROM stu_v2 WHERE id <= 15;
```

(6) 向视图 "stu_v3" 中插入数据.

① 插入数据 (11, 'Tom') , 成功, 因为同时满足视图 "stu_v2" 的检查条件 $id \geq 10$ 和视图 "stu_v1" 的检查条件 $id \leq 20$.

```
INSERT INTO stu_v3 VALUES(11, 'Tom');
```

② 插入数据 (17, 'Tom') , 成功, 因为同时满足视图 "stu_v2" 的检查条件 $id \geq 10$ 和视图 "stu_v1" 的检查条件 $id \leq 20$. 视图 "stu_v3" 无检查条件, 故无需满足 $id \geq 15$.

```
INSERT INTO stu_v3 VALUES(17, 'Tom');
```

③ 插入数据 (28, 'Tom') , 报错, 因为不满足视图 "stu_v1" 的检查条件 $id \leq 20$.

```
INSERT INTO stu_v3 VALUES(28, 'Tom');
```

3.5.1.2.2 local

[例]

(1) 创建一个展示 "student" 表中 $id \leq 15$ 的学生的 id 和 name 的视图 "stu_v4" , 无检查选项.

```
CREATE OR REPLACE VIEW stu_v4 AS SELECT id, name FROM student WHERE id <= 15;
```

(2) 向视图 "stu_v4" 中插入数据.

① 插入数据 (5, 'Tom') , 成功.

```
INSERT INTO stu_v4 VALUES(5, 'Tom');
```

② 插入数据 (16, 'Tom') , 成功, 因为无检查选项.

```
INSERT INTO stu_v4 VALUES(25, 'Tom');
```

(3) 创建一个展示视图 "stu_v4" 中 $id \geq 10$ 的学生的 id 和 name 的视图 "stu_v5" , 使用检查选项 `local` .

```
CREATE OR REPLACE VIEW stu_v5 AS SELECT id, name FROM stu_v4 WHERE id >= 10 WITH LOCAL
CHECK OPTION;
```

(4) 向视图 "stu_v5" 中插入数据.

① 插入数据 (13, 'Tom') , 成功, 因为满足视图 "stu_v5" 的检查条件 $id \geq 10$.

```
INSERT INTO stu_v5 VALUES(13, 'Tom');
```

② 插入数据 (17, 'Tom') , 成功, 因为满足视图 "stu_v5" 的检查条件 $id \geq 10$. 视图 "stu_v4" 无检查条件, 故无需满足 $id \leq 15$.

```
INSERT INTO stu_v5 VALUES(17, 'Tom');
```

(5) 创建一个展示视图 "stu_v5" 中 $id < 20$ 的学生的 id 和 name 的视图 "stu_v6" , 无检查选项.

```
CREATE OR REPLACE VIEW stu_v6 AS SELECT id, name FROM stu_v5 WHERE id < 20;
```

(6) 向视图 "stu_v6" 中插入数据.

① 插入数据 (14, 'Tom') , 成功, 因为视图 "stu_v6" 无检查条件, 故无需满足 $id < 20$; 满足视图 "stu_v5" 的检查条件 $id \geq 10$; 视图 "stu_v4" 无检查条件, 故无需满足 $id \leq 15$.

```
INSERT INTO stu_v3 VALUES(14, 'Tom');
```

(7) 创建一个展示 "student" 表中 $id \leq 15$ 的学生的 id 和 name 的视图 "stu_v7" , 使用检查选项 `local` .

```
CREATE OR REPLACE VIEW stu_v7 AS SELECT id, name FROM student WHERE id <= 15 WITH LOCAL
CHECK OPTION;
```

(8) 创建一个展示视图 "stu_v7" 中 $id \geq 10$ 的学生的 id 和 name 的视图 "stu_v8" , 使用检查选项 `local` .

```
CREATE OR REPLACE VIEW stu_v8 AS SELECT id, name FROM stu_v7 WHERE id >= 10 WITH LOCAL
CHECK OPTION;
```

(9) 向视图 "stu_v8" 中插入数据 (18, 'Tom') , 报错, 因为满足视图 "stu_v8" 的检查条件 $id \geq 10$, 不满足视图 "stu_v7" 的检查条件 $id \leq 15$.

```
INSERT INTO stu_v8 VALUES(18, 'Tom');
```

3.5.1.3 视图的更新

视图的行与基础表中的行一一对应才可更新. 若视图包含下列项之一, 则不可更新:

① 聚合函数或窗口函数, 如 `SUM` 、 `MIN` 、 `MAX` 、 `COUNT` .

② `DISTINCT` .

③ `GROUP BY` .

④ `HAVING` .

⑤ `UNION` 或 `UNION ALL` .

[**例**]

(1) 创建一个统计表 "student" 中学生数量的视图 "stu_v_count" .

```
CREATE VIEW stu_v_count AS SELECT COUNT(*) FROM student;
```

(2) 无法更新视图 "stu_v_count" , 如插入数据 10 .

```
INSERT INTO stu_v_count VALUES(10);  # 报错
```

3.5.1.4 视图的作用

[**视图的优缺点**]

(1) 优点:

① 简化查询语句: 常被使用的查询可定义为视图, 用户不必每次查询都指定全部条件.

② 安全: DB 可授权, 但不能授权到特定行和列上. 视图可让用户只能查询和修改他们可见的数据.

③ 建立物理数据独立性: 帮助用户屏蔽真实表结构变化带来的影响.

④ 提高编程效率.

⑤ 包含表中的最新数据.

⑥ 所需的存储空间小.

(2) 缺点:

① 每次引用视图时都需花费处理时间.

② 可能不被直接更新.

3.5.2 其它存储对象

[**存储函数**, **Stored Function**] 返回计算结果, 可用于 SQL 表达式.

[**存储过程**, **Stored Procedure**] 可用于执行任务和计算.

[**触发器**, **Trigger**] 自动检测某表的改变.

[**事件**, **Event**] 按时间安排执行.

[**游标**, **Cursor**] 用于遍历结果集, 每次返回一行.
