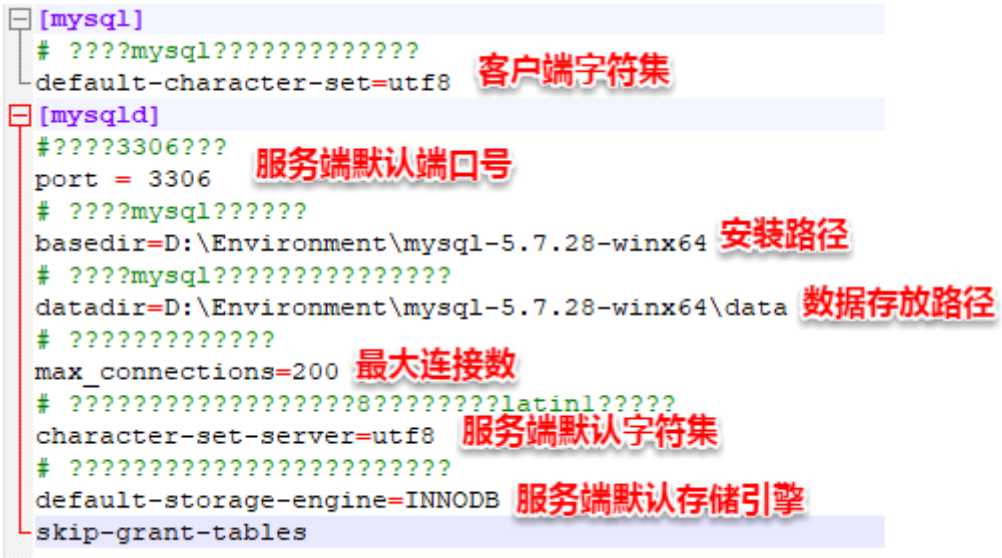
MySQL 基础

配置文件(my.ini)



MySQL服务的登录和退出 (dos 命令)

```
    基本指令：
        登入：
            mysql -h [ip地址] -P [端口号] -u [用户名] -p[密码]（注意 -p和密码之间不允许有空格）
            或
            mysql -h [IP地址] -P [端口号] -u [用户名] -p (回车)  然后再出入密码
            如果是本机，并且端口为默认的3306，则可以省略 -h、-P 的输入。

        登出：
            quit
            或
            exit
            两者都可以
```

MySQL 常见命令

```
    1. 查看当前所有的数据库
        show databases;
    2. 打开指定的库
        use [库名];
    3. 查看当前库的所有表
        show tables;
    4. 查看其它库的所有表
        show tables from [库名];
    5. 创建表
        create table [表名](
            列名 列类型,
            列名 列类型,
            ...
        );
```

```
    6. 查看表结构
        desc [表名]
    7. 查看服务器的版本
        mysql -V

```

MySQL 的语法规范

```
    1. 不区分大小写，但建议关键字大写，表名、列名小写。
    2. 每条命令用分号结尾。
    3. 每条命令根据需要，可以进行缩进或换行。
    4.  注释：
        单行注释：#注释文字
        单行注释：-- 注释文字
        多行注释：/* 注释文字 */

```

DQL 语法（select 查询）

基础查询语法（不带条件的单字段或多字段查询）

```
    -- 基础语法：
        select 查询列表 from 表名;
    -- 特点：
        -- 1. 查询列表可以是：表中的字段、常量值、表达式、函数。
        -- 2. 查询的结果是一个虚拟的表格。

    -- 例子：
        -- 1、查询表中的单个字段
            select 字段名 from 表名;

        -- 2、查询表中的多个字段
            select 字段1，字段2，字段3 from 表名;

        -- 3、查询表中的所有字段
            select * from 表名;
```

起别名（AS）

方式一：使用 AS

`SELECT last_name AS 姓,first_name AS 名 FROM employees;`

方式二：使用空格

`SELECT last_name 姓,first_name 名 FROM employees;`

> 特殊案例：查询salary，显示结果为 out put
> **因为 out 是一个关键词，所以 mysql 官方推荐我们使用双引号括起来。**
> `SELECT salary "out put" FROM employees;`

去重 (DISTINCT)

案例：查询员工表中涉及到的所有的部门编号

`SELECT DISTINCT department_id FROM employees`

拼接（concat）

案例：查询员工名和姓连接成一个字段，并显示为 姓名

`SELECT CONCAT(last_name,first_name) AS 姓名 FROM employees`

条件查询

转义符和通配符

```
/*
通配符：
    % ：代表任意的单个或多个字符
    _ ：  代表任意的单个字符

转义符：
    mysql官方默认是 ' \' ;但是如果我们想自己定义，也可以使用 ESCAPE 来指定。
*/
-- 案例：
    -- 查询员工名中第二个字符为 _ 的员工名。
        -- 方式一：
            SELECT last_name FROM employees WHERE last_name LIKE '_\_%';

        -- 方式二：（我们指定 # 为转义符）
            SELECT last_name FROM employees WHERE last_name LIKE '_#_%'
ESCAPE '#';
```

条件表达式

```
-- 简单条件运算符
    -- >、<、=、!=、<>、>=、<=

-- 案例
    -- 1、查询工资大于 12000 的员工信息
        SELECT * FROM employees WHERE salary > 12000;

    -- 2、查询部门编号不等于 90 号的员工名和部门编号
        -- 方式一：
        SELECT last_name,department_id FROM employees WHERE department_id !=
90;

        -- 方式二：（建议使用此方式）
        SELECT last_name,department_id FROM employees WHERE department_id <>
90;

-- 额外知识：
    -- 在使用 mybatis 中，编写动态sql 使用 <> 会报错，所以只能使用 != 进行判断。
```

逻辑表达式

```
-- 逻辑运算符：用于连接条件表达式
    -- && 和 and：两个条件都为true，结果为true，反之为false
    -- || 和 or：只要有一个条件为true，结果为true，反之为false
    -- ! 和 not：如果连接的条件本身为false，结果为true，反之为false
    -- mysql 推荐我们使用 and 、or 、not ；

-- 案例：
    -- 1、查询工资在 10000 到 20000 之间的员工名、工资以及奖金
        SELECT last_name,salary,commission_pct FROM employees WHERE salary >= 10000 AND salary <= 20000;

    -- 2、查询部门编号不是在 90 到 110 之间，或者工资高于 15000 的员工信息
        -- 方式一：
        SELECT * FROM employees WHERE department_id < 90 OR department_id > 110 OR salary > 15000;

        -- 方式二：
        SELECT * FROM employees WHERE NOT(department_id >= 90 AND department_id <= 110) OR salary > 15000;
```

模糊查询

like关键字

```
-- like
    -- 基本语法：
        select * from 表名 where 字段名 like '通配符和搜索内容';

    -- 特点：
        # 一般和通配符搭配使用：

    -- 案例：
        -- 1、查询员工名中包含字符 a 的员工信息
        SELECT * FROM employees WHERE last_name LIKE '%a%';

        -- 2、查询员工名中第三个字符为 n，第五个字符为 l 的员工名和工资
        SELECT last_name,salary FROM employees WHERE last_name LIKE '__n_l%';

        -- 3、查询员工名中第二个字符为 _ 的员工名（因为 _ 是一个通配符，所以我们需要使用 ' \' 进行转义）
            -- 方式一：
            SELECT last_name FROM employees WHERE last_name LIKE '_\_%';

           -- 方式二：（我们指定 # 为转义符）
            SELECT last_name FROM employees WHERE last_name LIKE '_#_%' ESCAPE '#';
```

范围查询

between and 关键字

```
-- 基本语法：
    select * from 表名 where 字段名 BETWEEN 条件1 AND 条件2;

-- 特点：
    -- 1. 使用 between and 可以提高语句的简洁度。意思是：在 条件1 和 条件2 之间。
    -- 2. 包含条件值 ，等同于 >= 、<=。
    -- 3. 注意两个条件值的顺序，弄反了不会报错但是会查不出数据。

-- 案例：
    -- 1、查询员工编号在 100 到 120 之间的员工信息
        -- 基础运算符方式：
           SELECT * from employees WHERE employee_id >= 100 AND employee_id
<= 120;

        -- 使用 between and 优化
            SELECT * from employees WHERE employee_id BETWEEN 100 AND 120;
```

in 关键字

```
-- 基本语法
    select * from 表名 where 字段名 in(条件1，条件2，...)；

-- 特点：
    -- 1. 使用 in 提高语句简洁度。意思是：包含 条件1，条件2，条件3，... 的数据
    -- 2. in 列表的值类型必须一致或兼容（可相互转换）。如：'123' 和 123 。
    --3. in 的列表不允许是空的。

-- 案例：
    -- 1、查询员工的工种编号是 IT_PROG、AD_VP、AD_PRES 中的一个员工名和工种编号
        -- 逻辑运算符方式
            SELECT * FROM employees WHERE job_id = 'IT_PROG' OR job_id =
'AD_VP' OR job_id = 'AD_PRES';

        -- 使用关键字 in
            SELECT * FROM employees WHERE job_id
IN('IT_PROG','AD_VP','AD_PRES');
```

is null 或 is not null 关键字

```
-- 基本语法：
    select * from 表名 where 字段名 is null;
    select * from 表名 where 字段名 is not null;

-- 特点：
    -- 因为 = 或 <> 都不能用于判断 null 值，所以我们可以使用 is null 或 is not null
来判断。

-- 案例：
    -- 1、查询奖金率为空的员工名和奖金率。
        SELECT last_name,commission_pct from employees WHERE commission_pct
IS NULL;
```

安全等于

```
-- 基本语法：
    SELECT * from employees where 字段名 <=> 条件;

-- 特点：
    -- 可以用于判断 null 值，也可以用于判断普通值。缺点是可读性较低。

-- 案例：
    -- 1、查询奖金率为空的员工名和奖金率
        SELECT last_name,commission_pct from employees WHERE commission_pct
<=> NULL;

    -- 2、查询工资为 12000 的员工信息
        SELECT last_name,salary from employees WHERE salary <=> 12000;
```

排序查询

```
-- 基本语法
    select * from 表名 【where 筛选条件】order by 字段名【asc/desc】 #不写默认是
asc（升序）

-- 特点：
    -- 1. ASC 代表的是升序，DESC 代表的是降序。如果不写，默认是 ASC 。
    -- 2. ORDER BY 中可以支持单个字段、多个字段、表达式、函数、别名。
    -- 3. ORDER BY 一般是放在查询语句的最后面，LIMIT 除外。

-- 案例：
    -- 1、查询员工信息，要求工资从高到低排序
        SELECT * FROM employees ORDER BY salary DESC;

    -- 2、查询部门编号 >= 90 的员工信息，按入职时间的先后进行排序
        SELECT * FROM employees WHERE department_id >= 90 ORDER BY hiredate;
#不写默认是 ASC

    -- 3、按年薪的高低显示员工的信息和年薪【按表达式结果高到低排序】
        -- 方式一：
            SELECT *,salary*12*(1+IFNULL(commission_pct,0)) 年薪 FROM
employees ORDER BY salary*12*(1+IFNULL(commission_pct,0)) DESC;

        -- 方式二：
            SELECT *,salary*12*(1+IFNULL(commission_pct,0)) 年薪 FROM
employees ORDER BY 年薪 DESC;

    -- 4、按姓名的长度显示员工的姓名和工资【按函数结果排序】
        -- 方式一：
            SELECT LENGTH(last_name) 字节长度,last_name,salary FROM employees
ORDER BY LENGTH(last_name) DESC

        -- 方式二：
            SELECT LENGTH(last_name) 字节长度,last_name,salary FROM employees
ORDER BY 字节长度 DESC

    -- 5、查询员工信息，要求先按工资排序，再按员工编号降序【按多个字段排序】
            SELECT * FROM employees ORDER BY salary ASC ,employee_id DESC;
```

分组查询

```sql
-- 基本语法：
    select 分组函数，列（要求出现在group by的后面）
    from 表
--     【where 筛选条件】
    group by 分组的列表
--     【having 子句（对分组的结果集进一步的筛选）】
--     【order by 子句】

-- 特点：
    -- 查询列要求是分组函数和 group by 后出现的字段。
    -- HAVING 子句一定要跟在 group by 子句后面
    -- 使用 HAVING 子句原因是，WHERE 关键字无法与聚合函数一起使用。HAVING 子句可以让我们筛选分组后的各组数据。
    -- 分组查询中的筛选条件分为两类
        -- 分组前筛选
            -- 数据源：原始表里存在的
            -- 位置：group by子句的前面
            -- 关键字：where
        -- 分组后筛选
            -- 数据源：分组后的结果集
            -- 位置：group by子句的后面
            -- 关键字：having
    -- 分组函数做条件肯定是放在 HAVING 子句中
    -- 能用分组前筛选的，就优先考虑使用分组前筛选。简而言之就是，能用 where 完成的就不用having。
    -- group by子句支持单个字段分组，多个字段分组（字段之间用逗号隔开），也可以使用表达式分组
    -- 同时也可以添加排序（排序放在整个分组查询的最后）

-- 案 例
    -- 查询每个工种的最高工资
        SELECT MAX(salary),job_id FROM employees GROUP BY job_id;

    -- 查询每个位置上的部门个数
        SELECT COUNT(*),location_id FROM departments GROUP BY location_id;

    -- 查询邮箱中包含a字符的，每个部门的平均工资
        SELECT AVG(salary),department_id FROM employees WHERE email LIKE '%a%' GROUP BY department_id;

    -- 查询有奖金的每个领导手下员工的最高工资
        SELECT MAX(salary),manager_id FROM employees WHERE commission_pct IS NOT NULL GROUP BY manager_id;

    -- 查询每个部门的员工个数
        SELECT COUNT(*),department_id FROM employees GROUP BY department_id;

    -- 按员工姓名的长度分组，查询每一组的员工个数，筛选员工个数>5的有哪些
        SELECT COUNT(*),LENGTH(last_name) FROM employees GROUP BY LENGTH(last_name) HAVING COUNT(*)>5;

    -- 查询每个部门的员工个数，并且对结果进行筛选，员工个数>2的才显示（HAVING 子句）
        SELECT COUNT(*),department_id FROM employees GROUP BY department_id HAVING COUNT(*)>2;

```

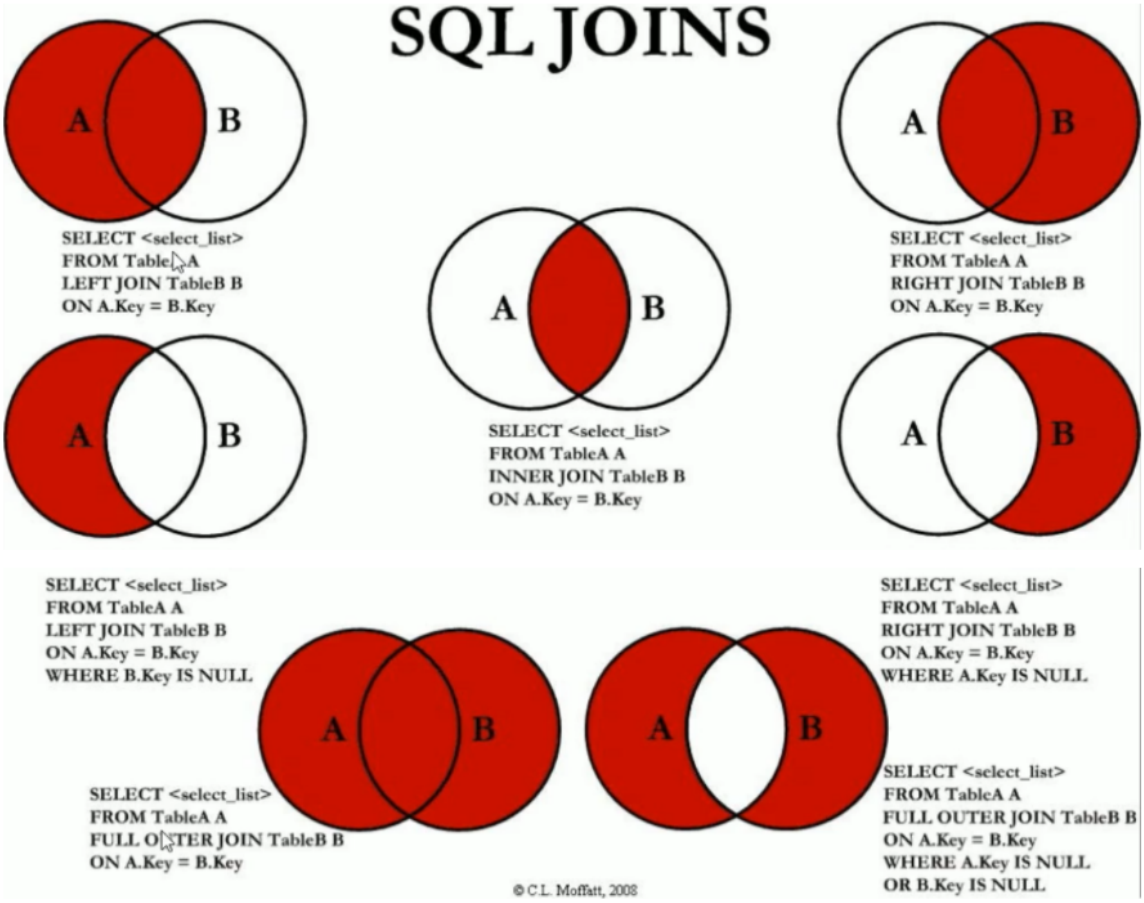
```sql
    -- 查询每个工种有奖金的员工的最高工资 > 12000 的工种编号和最高工资
        -- 方式一：使用 and
            SELECT MAX(salary),job_id FROM employees WHERE commission_pct IS NOT null AND salary>12000 GROUP BY job_id;

        -- 方式二：使用 HAVING 子句
            SELECT MAX(salary) 工资,job_id FROM employees WHERE commission_pct IS NOT null GROUP BY job_id HAVING 工资>12000

    -- 查询领导编号>102的每个领导手下的最低工资>5000的领导编号是哪个，以及其最低工资
        SELECT MIN(salary) 最低工资,manager_id FROM employees WHERE manager_id > 102 GROUP BY manager_id HAVING 最低工资>5000;

    -- 查询每个部门每个工种的员工的平均工资（多字段分组）
        SELECT AVG(salary),department_id,job_id FROM employees GROUP BY department_id,job_id;

    -- 查询每个部门每个工种的员工的平均工资，并且按平均工资的高低显示（倒序）
        SELECT AVG(salary),department_id,job_id FROM employees GROUP BY department_id,job_id ORDER BY AVG(salary) DESC

```

连接查询

```
又称多表查询，当查询的字段来自于多个表时，就会用到连接查询

注意：笛卡尔积现象的发生
    笛卡尔积：表1 有m行，表2 有n行，结果=m*n 行
    发生原因：没有有效的连接条件
    避免：一定要注意添加有效的连接条件


分类：
    按年代分类：
        sql92 标准：仅仅支持内连接，多表用逗号分隔
        sql99 标准【推荐】：支持内连接+外连接（左外和右外）+交叉连接，多表使用 【连接类型】join 连接。

    按功能分类：
        内连接：
            等值连接
            非等值连接
            自连接
        外连接：
            左外连接
            右外连接
            全外连接
        交叉连接

```

七种JOIN

七种JOIN



内连接（等值连接、非等值连接、自连接）

```
-- 其实内连接就跟我们前面写的那些查询语句差不多，只不过是多表

-- 内连接：只有条件的交叉连接，根据某个条件筛选出符合条件的记录，不符合条件的记录不会出现在结果集中，即取出两张表中匹配到的数据，匹配不到的不保留

-- 等值连接
    -- 1、多表等值连接的结果为多表的交集部分
    -- 2、n表连接，至少需要n-1个连接条件
    -- 3、多表的顺序没有要求
    -- 4、一般需要为表起别名
    -- 5、可以搭配前面介绍的所有子句使用，比如排序、分组、筛选

-- 案 例
    -- 查询有奖金的员工名、部门名
        -- sql 92：
            SELECT last_name,department_name,commission_pct FROM employees e,departments d WHERE e.department_id = d.department_id AND e.commission_pct IS NOT NULL;

        -- sql 99：
            SELECT last_name,department_name FROM `employees` e JOIN departments d ON e.department_id = d.department_id AND e.commission_pct IS NOT NULL;


    -- 查询城市名中第二个字符为 o 的部门名和城市名
```

```sql
        -- sql 92:
            SELECT department_name,city FROM departments d,locations l WHERE d.location_id = l.location_id AND city LIKE '_o%';

        -- sql 99:
            SELECT department_name,city FROM departments d JOIN locations l ON d.location_id = l.location_id WHERE city LIKE '_o%';


    -- 查询每个城市的部门个数，并且个数大于3的（分组+筛选）
        -- sql 92:
            SELECT COUNT(*) 个数, city FROM departments d,locations l WHERE d.location_id = l.location_id GROUP BY city HAVING 个数 > 3;

        -- sql 99:
            SELECT COUNT(*) 个数, city FROM departments d JOIN locations l on d.location_id = l.location_id GROUP BY city HAVING 个数 > 3;


    -- 查询有奖金的每个部门的部门名和部门的领导编号和该部门的最低工资（多条分组）
        -- sql 92:
            SELECT d.department_name,d.manager_id,MIN(salary) FROM departments d,employees e WHERE d.department_id = e.department_id AND commission_pct IS NOT NULL GROUP BY d.department_name,d.manager_id;

        -- sql 99:
            SELECT d.department_name,d.manager_id,MIN(salary) FROM departments d JOIN employees e on d.department_id = e.department_id WHERE commission_pct IS NOT NULL GROUP BY d.department_name,d.manager_id;


    -- 查询员工名、部门名和所在的城市（多表连接）
        -- sql 92:
            SELECT last_name,department_name,city FROM employees e,departments d,locations l WHERE e.department_id = d.department_id AND d.location_id = l.location_id;

        -- sql 99:
            SELECT last_name,department_name,city FROM employees e JOIN departments d  ON e.department_id = d.department_id JOIN locations l ON d.location_id = l.location_id;


-- 等值连接和非等值连接的区别：连接条件为=则为等值连接，反之为非等值连接

-- 等值连接和自然连接的区别：
    -- 等值连接中不要求相等属性值的属性名相同，而自然连接要求相等属性值的属性名必须相同，即两关系只有在同名属性才能进行自然连接。

    -- 等值连接不会将重复属性去掉，而自然连接去掉重复属性，也可以说，自然连接是去掉重复列的等值连接

```

外连接（左外连接、右外连接）

```
-- 外连接：取出连接表中匹配到的数据，匹配不到的也会保留，其值为NULL。

-- 特点：
    -- 1、外连接的查询结果为主表中的所有记录，如果从表中有和它匹配的，则显示匹配的值，如果从表中没有相匹配的值，则显示为NULL

    -- 2、左外连接，left join 左边的是主表，右外连接，right join 右边的是主表

-- 案 例
    -- 查询显示员工名以及其部门名称（左连接）
        SELECT e.last_name,d.department_name FROM `employees` e LEFT JOIN departments d ON e.department_id = d.department_id;

    -- 查询显示员工名以及其部门名称（右连接）
        SELECT e.last_name,d.department_name FROM `employees` e RIGHT JOIN departments d ON e.department_id = d.department_id;

```

全外连接

```
-- 在MySQL中，只提供了内连接，左外连接与右外连接。如果想要实现全外连接的效果，就需要使用UNION

-- UNION 操作符用于合并两个或多个 SELECT 语句的结果集。

-- 全外连接：左表和右表都不做限制，所有的记录都显示，两表不足的地方用null 填充（等同于左连接+右连接）

-- 示 例：
    -- 连接员工表和部门表，显示员工信息以及部门名称
        SELECT e.last_name,d.department_name FROM `employees` e LEFT JOIN departments d on e.department_id = d.department_id
        UNION
        SELECT e.last_name,d.department_name FROM employees e RIGHT JOIN departments d on e.department_id = d.department_id;

```

交叉连接

```
-- 交叉连接（笛卡儿积）：将两个表的所有数据全部对应显示一次，简单来说就是笛卡儿积。

-- 了解即可，正式工作如果写了，就赶紧准备提桶跑路。

-- 示 例：
    -- 连接员工表和部门表，显示员工信息以及部门名称
        SELECT * from employees CROSS JOIN departments;

-- PS:工作中用完就可以提桶跑路了。
```

子查询

```
含义：
    出现在其他语句中的 select 语句，称为子查询或内查询
    内部嵌套其他select语句的查询，称为外查询或外查询

分类：
     按子查询出现的位置：
        select 后面
        from 后面
        where 或 having 后面
        exists后面（相关子查询）

    按结果集的行列数不同：
        标量子查询（结果集只有一行一列）
        列子查询（结果集只有一列多行）
        行子查询（结果集有一行多列）
        表子查询（结果集一般为多行多列）

支持的子查询类型：
    select 后面：仅仅支持标量子查询
    from 后面：支持表子查询
    where 或 having 后面：标量子查询，列子查询，行子查询
    exists 后面：仅支持表子查询
```

select 后面的子查询使用

```
    -- select 后面：仅仅支持标量子查询（结果集只有一行一列）

-- 案例：
    -- 查询每个部门的员工个数
        SELECT d.*,(SELECT COUNT(*) FROMemployees e WHERE e.department_id =
d.department_id) 个数 FROM departments d;


    -- 查询员工号=102的部门名
        SELECT (
            SELECT department_name FROM departments d
            INNER JOIN employees e
            on d.department_id = e.department_id
            WHERE e.employee_id = 102
        ) 部门名;

```

exists 后面的子查询使用（也称作 相关子查询）

```sql
-- 特 点：
    -- 返回的结果是：1 或 0
    -- 对于子查询的类型无限制，可以是标量子查询，也可以是列子查询或是行子查询

-- 案 例：
    -- 查询有员工的部门名
        SELECT department_name FROM departments d WHERE EXISTS (
            SELECT * FROM employees e WHERE d.department_id =
e.department_id
        );

```

from 后面的子查询使用

```sql
-- 查询每个部门的平均工资和工资等级
    SELECT ag_dep.*,g.grade_level
    FROM (
        SELECT AVG(salary) ag,department_id
        FROM employees
        GROUP BY department_id
    ) ag_dep
    INNER JOIN job_grades g
    on ag_dep.ag BETWEEN lowest_sal AND highest_sal

```

where 后面的子查询使用

标量子查询

```sql

-- 特 点：
    -- 1、子查询放在条件的右侧
    -- 2、标量子查询，一般搭配着单行操作符使用：>, <, >=, <=, =, <>
    -- 3、子查询的执行优先于主查询执行。
    -- 4、在标量子查询中，如果该子查询的结果不是单行单列，那么则会报错。
    -- 5、子查询要编写在 （）里面


-- 案 例：
    -- 查询salary > Abel 的员工信息
        SELECT * FROM employees WHERE  salary > (
            SELECT salary FROM employees WHERE last_name = 'Abel');

    -- 返回 job_id 与 141 号员工相同，salary 比 143 号员工多的员工姓名，job_id 和工
资
        SELECT last_name,job_id,salary FROM employees WHERE job_id =
            (SELECT job_id FROM employees WHERE employee_id = 141)
        AND salary >
            (SELECT salary FROM employees WHERE employee_id = 143);

    -- 返回公司工资最少的员工的名字，job_id和工资
        SELECT last_name,job_id,salary FROM employees WHERE salary =
            (SELECT MIN(salary) from employees);

    -- 查询最低工资大于50号部门最低工资的部门id和其最低工资


```

```sql
        SELECT MIN(salary) 工资,department_id FROM employees GROUP BY department_id
        HAVING MIN(salary) >  (SELECT MIN(salary) FROM employees WHERE department_id = 50);
```

列子查询

```sql

-- 特 点：
    -- 1、 列子查询也称作多行子查询，因为返回的结果是 一列多行 的数据
    -- 2、使用列子查询，要搭配 多行比较操作符 来使用：
        -- IN / NOT IN ：等于列表中的任意一个
        -- ANY / SOME ：和子查询返回的某一个值比较
        -- ALL ：和子查询返回的所有值进行比较


-- 案 例 ：
    -- 返回 location_id 是 1400 或 1700 的部门中的所有员工姓名
        SELECT last_name,department_id FROM employees WHERE department_id in
            (SELECT DISTINCT department_id FROM departments WHERE location_id in (1400,1700));

    -- 返回其他部门中比 job_id 为 'IT_PROG' 部门里任一工资低的员工的员工号、姓名、job_id 以及 salary
        SELECT employee_id,last_name,job_id,salary FROM employees WHERE salary > ANY(
            SELECT DISTINCT salary FROM employees WHERE job_id = 'IT_PROG'
        ) AND job_id <> 'IT_PROG';

    -- 返回其他部门中比 job_id 为 'IT_PROG' 部门所有工资都低的员工的员工号、姓名、job_id 以及 salary
         SELECT employee_id,last_name,job_id,salary FROM employees WHERE salary < ALL(
            SELECT DISTINCT salary FROM employees WHERE job_id = 'IT_PROG'
        ) AND job_id <> 'IT_PROG';

```

行子查询（仅作了解）

```sql

-- 特 点：
    -- 行子查询。返回的结果集有可能是一行多列，也有可能是多行多列
    -- 使用行子查询，则条件必须一致，都是统一 等于 ，大于，或是小于。


-- 案 例：(由于行子查询比较难懂，所以用标量子查询来作为演示)
    -- 查询员工编号最小并且工资最高的员工信息
        -- 标量子查询写法：
           SELECT * FROM employees WHERE employee_id = (
                SELECT MIN(employee_id) FROM employees
            ) AND salary = (
                SELECT MAX(salary) FROM employees
            );

        -- 行子查询写法：
            SELECT * FROM employees WHERE (employee_id,salary) = (
                SELECT MIN(employee_id),MAX(salary) FROM employees
```

```
            );
```

分页查询

```
-- 当要显示的数据，无法在一页显示时，需要使用分页查询来分割显示的数据

-- 特 点：
    -- limit 语句放在查询语句的最后
    -- 分页起始值的计算公式：select 查询列表 from 表名 limit (page-1)*size , size
        -- page：要显示的页数
        -- size：每页的条数

--语 法：
        select 字段...
        from 表名
        【连接查询
        where 筛选条件
        group by 分组字段
        having 分组后的筛选条件
        order by 排序字段 正序/倒序】
        limit offset , size

        offset：要显示的数据的起始索引（起始索引从 0 开始）
        size：要显示的条目个数

-- 案 例：
    -- 查询前五条员工信息
        SELECT * FROM employees LIMIT 0,5;

    -- 查询第11条~第25条员工信息
        SELECT * FROM employees LIMIT 10,15;

    -- 有奖金的员工信息，并且工资较高的前10名显示
        SELECT * FROM employees WHERE commission_pct IS NOT NULL ORDER BY
salary DESC LIMIT 0,10;

```

联合查询（union）

```
-- union（联合、合并）：将多条查询语句的结果合并成一个结果

-- 特 点：
    -- 要求多条查询语句的查询列数是一致的！
    -- 要求多条查询语句的查询的每一列的类型和顺序最好一致
    -- union关键字默认去重，如果使用 union all 可以包含重复项


-- 语 法：
    查询语句1
    union
    查询语句2
    union
    。。。


-- 案 例：
```

```sql
    -- 查询部门编号>90 或 邮箱包含 a 的员工信息
        -- 不使用联合查询的方式：
            SELECT * FROM employees WHERE email LIKE '%a%' OR department_id > 90;

        -- 联合查询
            SELECT * FROM employees WHERE email LIKE '%a%'
            UNION
            SELECT * FROM employees WHERE department_id > 90;


```

函数

```
概念：
    类似于 java 的方法。

好处：
    1. 隐藏了实现细节
    2. 提高代码的重用性

调用：
    select 函数名(实参列表) 【from 表】;

分类：
    1、单行函数
        如：concat、length、ifnull 等
    2、分组函数，一般用作统计，又称为统计函数、聚合函数或者组函数。

```

系统函数

单行函数

字符函数

拼接函数(CONCAT)

```sql
 /*
    在java中，如果我们想拼接字符串，可以直接使用 + 号，但是MySQL中的 + 号只有一个作用，那就是运算。
    而如果我们想拼接字段名，可以使用 CONCAT() 函数。
 */
 -- 基础语法：
    select CONCAT(字段名1，字段名2，...) from 表名;

-- 案例：
    -- 查询显示员工的姓名，要求姓和名要合并一起显示,中间以空格隔开
        SELECT CONCAT(last_name,' ',first_name) 姓名 FROM employees;
```

判空函数（IFNULL）

```
-- 在查询数据时，其中有一个字段的值有可能存在 null 值，而我们想给予其一个默认值，可以使用判空函数。

-- 基础语法：
    select IFNULL(字段名，默认值) from 表名；

-- 案例：
    -- 查询员工名、奖金率和员工编号，如果奖金率为空则显示为 0，否则正常显示奖金率。
        SELECT last_name,IFNULL(commission_pct,0) commission_pct, employee_id FROM employees;
```

长度函数（LENGTH）

```
-- 如果我们需要计算一个字符的长度，那么我们可以使用函数：LENGTH（字符）

-- 基础语法：
    select LENGTH(字符串);

-- 案例：
    -- 查询员工名字的长度
        SELECT last_name,LENGTH(last_name) 字节长度 FROM employees;
```

大小写转换函数（UPPER、LOWER）

```
-- UPPER 可以将选定的数据转换为大写
-- LOWER 可以将选定的数据转换为小写

-- 基础语法：
    SELECT UPPER(实参); / SELECT LOWER(实参);

-- 案例：
    -- 将姓变大写，名变小写，然后拼接，中间以空格隔开
    SELECT CONCAT(UPPER(last_name),' ',LOWER(first_name)) 姓名 FROM employees;
```

截取函数（SUBSTR）

```
-- 根据条件截取指定的字符串
/*
要注意！和 java 不同，索引是从1开始！！！

截取函数一共有四种方式：

SUBSTR(str,pos,len) ：截取从指定索引处指定字符长度的字符

SUBSTR(str FROM pos FOR len)：截取从指定索引处指定字符长度的字符

SUBSTR(str,pos)：  截取从指定索引处后面所有的字符

SUBSTR(str FROM pos)：  截取从指定索引处后面所有的字符
*/

-- 案例：
    -- 查询员工名字，第一个字母和第二个字母之间使用下划线隔开
        -- 方式一：
```

```
            SELECT CONCAT(SUBSTR(first_name,1,1),'_',SUBSTR(first_name,2)) 员工名 FROM employees;

        -- 方式二：
            SELECT CONCAT(SUBSTR(first_name FROM 1 FOR 1),'_',SUBSTR(first_name FROM 2)) 员工名 FROM employees;

```

计算起始索引函数（INSTR）

```
-- 用于查找指定字符在字符串中的起始下标

-- 基本语法
    select instr(字符串，字符);
    -- 例子：
        SELECT INSTR('老王找到了起夜级李姐','李姐') AS 下标; #可直接复制运行

-- 案例：
    -- 查找员工编号为100，他的姓里面，下划线所在的起始索引
        SELECT last_name,INSTR(last_name,'_') AS 起始索引 FROM employees WHERE employee_id = 100;
```

过滤函数

```
-- 过滤指定的字符串

-- 基本语法
    -- 完整格式：
        -- TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)

    -- 简化格式：
        -- TRIM([remstr FROM] str)

-- 返回字符串 str ， 其中所有remstr前缀和/或后缀都已被删除。若分类符BOTH、LEADING或TRAILING中没有一个是给定的,则假设为BOTH。remstr为可选项，在未指定情况下，可删除空格。
    -- BOTH：过滤字符串头部和尾部
    -- LEADING：过滤字符串的头部
    -- TRAILING：过滤字符串的尾部

-- 案例：
    -- 将字符串 '    bar    ' 前后的空格去除掉
        SELECT TRIM('    bar    ');
        -- 结果： 'bar'

    -- 将字符串 ' xxxbarxxx ' 前后的 ' x ' 去除掉
        SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -- 结果： 'bar'
    或
        SELECT TRIM('x' FROM 'xxxbarxxx');
        -- 结果： 'bar'

    -- 将字符串 ' xxxbar ' 前面的 ' x ' 去除掉
        SELECT TRIM(LEADING 'x' FROM 'xxxbar');
        -- 结果： 'bar'

    -- 将字符串 ' barxxx ' 后面的 ' x ' 去除掉
        SELECT TRIM(TRAILING 'x' FROM 'barxxx');
```

```
        -- 结果： 'bar'

    -- 将字符串 ' barxxyz '  后面的 ' xyz ' 去除掉
        SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -- 结果： 'barx'

-- 在这里还有两个扩展的过滤函数，一般比较少用，因为只能去除空格，不能像 TRIM 一样可以去除指定字符
    -- 去除左空格函数
        SELECT LTRIM('     barbar');
        -- 结果： 'barbar'

    -- 去除右空格函数
        SELECT RTRIM('barbar     ');
        -- 结果： 'barbar'

```

填充函数（LPAD、RPAD）

```
-- 用指定的字符实现填充指定长度

-- 种类
    -- 左填充：LPAD(str,len,padstr)
        select LPAD();

    -- 右填充：RPAD(str,len,padstr)
        select RPAD();

-- 案例：
    -- 使用 ' * ' 给字符串 '星星' 的左边填充，使其长度达到 10 。
        select LPAD('星星',10,'*');
            -- 结果：********星星

    -- 使用 ' * ' 给字符串 '星星' 的左边填充，使其长度达到 10 。
        select RPAD('星星',10,'*');
            -- 结果：星星********

# 注意不要搞混，虽然在mysql中utf-8下，一个中文字母等于3个字节，但是填充函数里填写的长度是字符长度，不是字节长度。
```

替换函数（replace）

```
-- 将指定的字符串或字符替换成我们想要替换的数据

-- 基本语法
    select REPLACE(str,from_str,to_str);

-- 案例：
    -- 将员工编号为100，姓里面的下划线替换成 # 号
        SELECT last_name,REPLACE(last_name,'_','#') 替换 FROM employees WHERE employee_id = 100;
        -- 结果：K_ing > K#ing
```

数字函数

四舍五入函数（round）

```
-- 四舍五入：ROUND(X)

-- 保留 D 位小数，四舍五入：ROUND(X,D)

-- 案例：
    -- 将数值 1.65 四舍五入
        SELECT ROUND(1.65);
            -- 结果：2



    -- 将数值 1.65789 四舍五入，并保留两位小数
        SELECT ROUND(1.65789,2);
            -- 结果：1.66

```

向上取整函数（ceil）

```
-- 向上取整，返回大于等于该参数的最小整数

-- 基本语法：
    select ceil();

-- 案例：
    -- 向上取整 1.00
        SELECT CEIL(1.00);
            -- 结果：1

    -- 向上取整 1.02
        SELECT CEIL(1.02);
            -- 结果：2
```

向下取整函数（floor）

```
-- 向下取整，返回小于等于该参数的最大整数

-- 基本语法：
    select floor();

-- 案例：
    -- 向下取整 9.99
        SELECT FLOOR(9.99);
            -- 结果：9
```

截断函数（TRUNCATE）

```sql
-- 指定截断后留下几位小数（不会进行四舍五入）

-- 基本语法：
    SELECT TRUNCATE（数值，保留位数）;

-- 案例：
    -- 该 1.65789 数值保留两位小数，不进行四舍五入
        SELECT TRUNCATE(1.65789,2);
            -- 结果：1.65
```

取余函数（mod）

```sql
-- 效果和直接使用运算符 % 一样

-- 取余运算：a - a / b * b
    -- 如 mod（10，3），结果就是 10 - 10 / 3 * 3 --> 1

-- 基本语法：
    SELECT MOD(10,3);

-- 案例：
    -- 计算 10 % 3
        SELECT MOD(10,3);
            -- 结果：1
```

日期函数

返回当前系统日期 + 时间（now）

```sql
-- 用于返回当前系统日期 + 时间

-- 基本语法：
    SELECT NOW();

```

返回当前系统日期（CURDATE）

```sql
-- 该函数只返回系统当前日期，不返回时间

-- 基本语法：
    SELECT CURDATE();
```

返回当前系统时间（CURTIME）

```sql
-- 该函数只返回系统当前时间，不返回日期

-- 基本语法：
    SELECT CURTIME();
```

获取指定部分，年、月、日、时分秒等等

```sql
-- 年（year）
    select YEAR();

-- 月（month）
    select MONTH();

-- 日（day）
    SELECT DAY(NOW());

-- 还有 hour、minute、second 等等。因为这类型函数非常得多，具体就不一一展示了，有兴趣可以自行去查询


```

日期转换函数（StrToDate、DateFormat）

```sql
-- 将日期格式的字符转换成指定格式的日期
    select STR_TO_DATE(str,format);
    -- 例：
        SELECT STR_TO_DATE('9-13-1999','%m-%d-%Y')

-- 将日期转换成字符
    select DATE_FORMAT(date,format);
    -- 例：
        SELECT DATE_FORMAT('1999-9-13','%Y年%m月%d日')

-- 案例：
    -- 查询入职日期为 1992--4-3 的员工信息
        SELECT * from employees WHERE hiredate = STR_TO_DATE('1992--4-3','%Y--%c-%d');

    -- 查询有奖金的员工名和入职日期（xx月/xx日 xx年）
        SELECT last_name,DATE_FORMAT(hiredate,'%m月/%d日 %Y年') 入职日期 FROM `employees` WHERE commission_pct IS NOT null;
```

序号	格式符	功能
1	%Y	四位的年份
2	%y	2位的年份
3	%m	月份(01,02,...,11,12)
4	%c	月份(1,2,...,11,12)
5	%d	日(01,02,...)
6	%H	小时(24小时制)
7	%h	小时(12小时制)
8	%i	分钟(00,01,...,59)
9	%s	秒(00,01,...,59)

分组函数（又称聚合函数或统计函数、组函数）

求和函数和求平均值函数（sum & avg）

```
-- sum：将参数的值全部相加起来
    -- 基本语法
        select SUM([DISTINCT] expr) from 表名;

    -- 案 例
        -- 显示所有员工的工资总和
            SELECT SUM(salary) 总和 FROM employees;

        -- 显示所有员工的工资总和（去重）
            SELECT SUM(DISTINCT salary) 总和 FROM employees;

-- avg：将参数的所有值进行相加，然后除以个数得到平均值
    -- 基本语法
        SELECT AVG([DISTINCT] expr) FROM 表名

    -- 案 例
        -- 显示员工的工资平均值
            SELECT AVG(salary) 平均值 FROM employees;

        -- 显示员工的工资平均值（去重）
            SELECT AVG(DISTINCT salary) 平均值 FROM employees;

-- sum 和 avg 一般用于处理数值类型的数据

```

最大值和最小值（max & min）

```
-- max：用于返回参数中的最大值
    -- 基本语法
        SELECT MAX([DISTINCT] expr) FROM 表名;

    -- 案 例
        -- 查询返回工资最高的员工工资
            SELECT MAX(salary) 最高工资 FROM employees;

        -- 查询返回工资最高的员工工资（去重）
            SELECT MAX(DISTINCT salary) 最高工资 FROM employees;


-- min：用于返回参数中的最小值
    -- 基本语法
        select MIN([DISTINCT] expr) from 表名;

    -- 案 例
        -- 查询返回工资最低的员工工资
            SELECT MIN(salary) 最低工资 FROM employees;

        -- 查询返回工资最低的员工工资（去重）
            SELECT MIN(DISTINCT salary) 最低工资 FROM employees;

-- max 和 min 函数可以处理任何类型的参数。
```

计算个数函数（count）

```
-- count：用于计算参数的个数
    -- 基本语法
        select COUNT(DISTINCT expr,[expr...]) from 表名;

    -- 特点：
        -- count（*）和 count（1）都是用来统计行数，这两者的使用也比较多
        -- 在 MYISAM 存储引擎下，count（*）效率高一些
        -- 在 INNODB 存储引擎下，count（1）和 count（*）效率一样，但在没有主键时，count（1）效率高一些。

    -- 案 例
        -- 查询有工资的员工个数
            SELECT COUNT(salary) FROM employees;

        -- 查询有工资的员工个数（去重）
            SELECT COUNT(DISTINCT salary) FROM employees;
```

流程控制函数（if、case）

```
-- if 函数：等同于 if else 的效果，但其实更类似于 三元表达式

-- 基本语法
    SELECT IF(判断条件,true时的返回结果,false时的返回结果);

-- 案例：
    -- 查询显示员工的名字和奖金率，如果奖金率为 null ，则显示为 0
        SELECT last_name,commission_pct,IF(commission_pct IS NULL,0,commission_pct) 奖金率 FROM employees;


-- case函数：等同于 switch case 的效果

-- case函数有两种使用方式：
--- 使用方式一：switch case
/*
 java 中
    switch(变量或表达式){
        case 常量1:语句1；break；
        ...
        default：语句n；break；
    }

mysql 中
    case 要判断的字段或表达式
    when 常量1 then 要显示的值1或语句1；
    when 常量2 then 要显示的值2或语句2；
    ...
    else 要显示的 值n 或 语句n；
    end

由此可以发现，MySQL中 case 对应 switch，when ... then ...；对应 case，else 对应 default，最后要使用 end 结束整个流程。
else 是可以被省略的，如同 java 中 default 可以被省略一样。
而且，then 后面要显示的是值而不是语句的时候，不需要加分号，否则会报错.
*/
```

```
-- 案 例：
    /* 查询员工的工资，要求：
           部门号 = 50，显示的工资为 1.1 倍
           部门号 = 80，显示的工资为 1.2 倍
           其他部门，显示的工资为原工资
    */
        SELECT salary 原工资,department_id,
        CASE department_id
            WHEN 50 THEN salary*1.1
            WHEN 80 THEN salary*1.2
            ELSE salary
        END AS 新工资 FROM employeesl;

-- 使用方式二：if else
/*
java 中：
    if（条件 1）{
        语句 1；
    }else if（条件 2）{
        语句 2；
    }
    ...
    else{
        语句 n；
    }

mysql 中
    case
    when 条件1 then 要显示的值1或语句1
    when 条件2 then 要显示的值2或语句2
    ...
    else 要显示的值n 或 语句n；
    end

在这种使用方式下，case 后面不需要跟条件，而是在 when 的后面写条件。
同样，else 可以被省略 ，以及 then 后面要显示的是值而不是语句的时候，不需要加分号，否则会报错.
*/

-- 案 例：
    /* 查询员工的工资的情况
           如果工资 > 20000，显示A级别
           如果工资 > 15000，显示B级别
           如果工资 > 10000，显示C级别
           除此之外，显示D级别
    */
    SELECT last_name,salary 原工资,
    CASE
        WHEN salary>20000 THEN 'A'
        WHEN salary>15000 THEN 'B'
        WHEN salary>10000 THEN 'C'
        ELSE 'D'
    END 新工资 FROM employees;
```

自定义函数

```
创建语法：
    CREATE FUNCTION 函数名(参数列表) RETURNS 返回类型
    BEGIN
            函数体;
    END;

调用语法：
     SELECT 函数名(参数列表);

查看函数语法：
    SHOW CREATE FUNCTION 函数名;

删除函数语法：
    DROP FUNCTION 函数名;

注意：
    1、参数列表包含两部分：参数名 参数类型
    2、函数体：肯定会有 return 语句，如果没有会报错。如果 return 语句没有放在函数体的最后也不报错，但不建议。
    3、函数体如果仅仅只有一句SQL，则可以省略begin end。
    4、使用 delimiter 语句设置结束标记

案例：
--  无参有返回
        返回男生的个数
        CREATE FUNCTION myf1() RETURNS INT
        BEGIN
            DECLARE c INT DEFAULT 0;
            SELECT COUNT(*) INTO c FROM boys;
            RETURN c;
        END;

        SELECT myf1();

--  有参有返回
  1、根据员工名，返回它的工资
        CREATE FUNCTION myf2(empName VARCHAR(20)) RETURNS DOUBLE
        BEGIN
            SET @sal=0;
            SELECT salary INTO @sal FROM employees
            WHERE last_name = empName;

            RETURN @sal;
        END;

        SELECT myf2("Kochhar");

  2、根据部门名，返回该部门的平均工资
        CREATE FUNCTION myf3(depName VARCHAR(20)) RETURNS DOUBLE
        BEGIN
            DECLARE sal DOUBLE;
            SELECT AVG(salary) INTO sal FROM employees e
            JOIN departments d on e.department_id = d.department_id
            WHERE d.department_name = depName;

```

```
            RETURN sal;
        END;

        SELECT myf3("IT");

```

DML语言(表操作)

```
    DML操作主要有三种：
        插入：insert
        修改：update
        删除：delete
```

插入操作

 0ad953ed338cb77196b10a8f020f8606.png

插入方式一

```
-- 语 法：
    insert into 表名（列名1，...）values（值1，...）;

-- 注意点：
    -- 1、插入的值的类型要与列的类型一致或兼容
        INSERT INTO beauty(id,name,sex,borndate,phone,photo,boyfriend_id)
VALUES(13,'李姐','女','1990-4-23','18888888888',NULL,2);

    -- 2、不可以为 null 的列必须插入值。可以为 null 的列则可以选择不插入值
        INSERT INTO beauty(id,`name`,sex,phone) VALUES(13,'李
姐','女','18888888888');

    -- 3、列的顺序可以调换
        INSERT INTO beauty(`name`,sex,phone,id,) VALUES('李
姐','女','18888888888',13);

    -- 4、列数和值的个数必须一致
        INSERT INTO beauty(id,`name`,sex,phone) VALUES(13,'李
姐','女','18888888888');

    -- 5、可以省略列名，默认所有列，而且列的顺序和表中列的顺序一致
        INSERT INTO beauty VALUES(13,'李姐','女','1990-4-
23','18888888888',NULL,2);

```

插入方式二

```
-- 语 法：
    Insert into 表名 set 列名=值，列名=值，...

-- 示 例：
    Insert into beauty set id=19,name='白兰',phone='999'

```

两种插入方式的差别

```
-- 两种插入方式的差别：
    -- 1、方式一支持单次插入多行，方式二不支持
        INSERT INTO beauty(id,`name`,sex,phone) VALUES(13,'李1','女','18888888888'),(14,'李2','女','18888888888'),(15,'李3','女','18888888888');

    -- 2、方式一支持子查询，方式二不支持
        INSERT INTO beauty(id,`name`,sex,phone) select id,boyname,'123456' from boys where id < 3;

```

修改操作

```
-- 修改单表的记录
        -- 语 法：
            update 表名
            set 列=新值，列=新值，...
            where 筛选条件;

        --案 例：
            -- 修改 beauty 表中姓唐的女神的电话为 13899888899
                UPDATE beauty SET phone = '13899888899' WHERE `name` LIKE '唐%';


-- 修改多表的记录
        -- 语 法：
            -- sql 92 语法：
                update 表1 别名，表2 别名
                set 列=值，...
                where 连接条件
                and 筛选条件;



            -- sql 99 语法：
                update 表1 别名
                inner | left | right join 表2 别名
                on 连接条件
                set 列=值，...
                where 筛选条件;


        -- 案 例：(主要介绍 sql99 语法)
            -- 修改张无忌的女朋友的手机号为114
                UPDATE boys bo INNER JOIN beauty b ON bo.id = b.boyfriend_id
                SET b.phone = '114' WHERE bo.boyName = '张无忌';

            -- 修改没有男朋友的女神的男朋友编号都为 2 号
                UPDATE boys bo RIGHT JOIN beauty b on bo.id = b.boyfriend_id
                SET b.boyfriend_id = 2 WHERE bo.id IS NULL;

```

删除操作

```sql
-- 方式一：delete
    -- 1、单表的删除
        -- 语 法：
            delete from 表名 where 筛选条件

        -- 案 例：
            -- 删除手机号以 9 结尾的女神信息
                DELETE FROM beauty WHERE phone LIKE '%9';

    -- 2、多表的删除
        -- 语 法：
            -- sql 92语法：
                delete 别名
                from 表1 别名，表2 别名
                where 连接条件
                and 筛选条件;


            -- sql 99语法：
                delete 表1的别名，表2的别名
                from 表1 别名
                inner | left | right join 表2 别名 on 连接条件
                where 筛选条件;

        -- 案 例：
            -- 删除张无忌的女朋友的信息
                DELETE b FROM beauty b INNER JOIN boys bo on b.boyfriend_id = bo.id WHERE bo.boyName = '张无忌';

            -- 删除黄晓明的信息以及他女朋友的信息
                DELETE b,bo FROM beauty b INNER JOIN boys bo ON b.boyfriend_id = bo.id WHERE bo.boyName='黄晓明';


-- 方式二：truncate
    -- 语 法：
        truncate table 表名;

    -- 特 点：
        -- truncate 不允许添加筛选条件。
        -- truncate 一般用于清空表所有数据。（又名删表跑路）


    -- 案 例：
        -- 将boys 表信息清空
            TRUNCATE TABLE boys;


-- 方式一 与 方式二的区别：
    -- delete 可以加 where 条件，truncate 不能加

    -- truncate 删除，效率高一丢丢

    -- 假如要删除的表中有自增长列，如果用 delete 删除后，再插入数据，自增长列的值从断点开始，而 truncate 删除后，再插入数据，自增长列的值从 1 开始。
```

```
    -- truncate 删除没有返回值，delete 删除有返回值。

    -- truncate 删除不能回滚，delete 删除可以回滚.
```

DDL语言

```
    DDL 语言简单来说就是对库和表的管理

    一、库的管理
        创建： create
        修改： alter
        删除： drop
        展示： show
        选择： use
    二、表的管理
        创建： create
        修改： alter
        删除： drop
        展示： show
```

数据库的管理（库的CRUD）

```
-- 库的管理
    --创建： create
        -- 语 法：
            create database [ if not exists ] 库名;

    -- 修改： alter
        -- 语 法：
            alter database 库名 修改条件 set 修改后的值

    -- 删除： drop
        -- 语 法：
            drop [ if exists ] databases;

    -- 展示： show
        -- 语 法：
            show databases;

    -- 选择： use
        -- 语 法：
            use 库名;

-- 示 例：
    -- 查询有多少个库
        show databases;

    -- 更改库的字符集(gbk)
        ALTER DATABASE 库名 CHARACTER SET gbk;

    -- 新建库
        CREATE DATABASE IF NOT EXISTS 库名;

    -- 删除库
```

```
        DROP DATABASE [ IF EXISTS ] 库名;

    -- 选择库
        USE 库名;
```

表的管理（表的CRUD）

```
-- 表的管理
    --创建：
        -- 语 法：
            -- 新增列：ADD
                ALTER TABLE 表名 ADD COLUMN 列名 类型;

            -- 新增表：CREATE
                CREATE TABLE [ IF NOT  EXISTS ] 表名（列名1 类型，列名2 类型，...）;


    -- 修改： modify
        -- 语 法：
            -- modify：用于修改列名
                ALTER TABLE 表名 MODIFY COLUMN 列名 类型;

            -- change：用于修改列的类型
                ALTER TABLE 表名 CHANGE COLUMN 列名 类型;


    -- 删除： drop
        -- 语 法：
            -- 删除列：
                ALTER TABLE 表名 DROP COLUMN 列名;

            -- 删除表：
                DROP TABLE [ IF EXISTS ] 表名;



    -- 查看所有表： show
        -- 语 法：
            show tables;


    -- 查看表结构：DESC
        -- 语 法：
            DESC 表名;


-- 示 例：
    -- 查询所有表
        show tables;

    -- 创建表 Book
        CREATE TABLE book{
            id INT,
            bName VARCHAR,
            price DOUBLE,
            authorId INT,
```

```
            publishDate DATETIME
        };

    -- admin 表中新增一个列 ，列名 remark ，类型为 text；
        ALTER TABLE admin ADD COLUMN remark TEXT;

    -- 修改 admin 表中 remark 列的类型为 VARCHAR
        ALTER TABLE admin MODIFY COLUMN remark VARCHAR(20);

    -- 修改 admin 表中 remark 列名为 nick_name;
        ALTER TABLE admin CHANGE COLUMN remark nick_name VARCHAR(20);

    -- 删除 nick_name 列；
        ALTER TABLE admin DROP COLUMN nick_name;

    -- 修改表名;
        ALTER TABLE admin RENAME TO one_admin;

```

表的复制（扩展了解）

```
-- 由于表的复制只是一个扩展知识，所以直接上 示例

-- 示 例：
    -- 1、仅仅复制表的结构
            CREATE TABLE copy LIKE admin;

    -- 2、复制表的结构 + 数据
            CREATE TABLE copy2 select * from admin;

    -- 3、只复制部分数据
            CREATE TABLE copy3 select id,username from admin where id = 1;

    -- 4、仅仅复制部分字段，无数据（提示，传入永远不可能成立的条件）
            CREATE TABLE copy4 select id,username from admin where id = 0;

```

数据类型

数值型

整型

```
分类：（仅仅是范围的区别）
    Tinyint：1 个字节，范围 -128 ~ 127 / 0 ~ 255

    Smallint：2 个字节，范围 -32768 ~ 32767 / 0 ~ 65535

    Mediumint：3 个字节，范围 -8388608 ~ 8388607 / 0 ~ 1577215

    int：4 个字节，范围 -2147483648 ~ 2147483647 /  0 ~ 4294967295

    Bigint：8 个字节，范围 -9223372036854775808 ~ 9223372036854775807 / 0 ~
18446744073709551615


```

```
特点：
    1、如果不设置是无符号还是有符合，默认是有符号（可接受复数值）。要设置成无符号，则在字段类型后面追加 unsigned 关键字。

    2、如果插入的数值超过了范围，那么就会报 out of range 异常，并且插入临界值（如 tinyint，大于 255的值则会插入255，小于 0 的则会插入 0）
    PS：在新版本的 mysql ，如果超过了范围就不再会插入了。

    3、如果不设置长度，那么则会使用默认长度。
    PS：长度代表了显示的值的最大位数，如果不够会则会用 0 在左边填充，但是必须设置使用 zerofill 关键字。

```

浮点型

```
分类：（这里的范围就不写了，反正很大）
    浮点型：
        float（M,D）：4 个字节

        double（M,D）：8 个字节

    定点型：
        DEC（M,D）：DECIMAL 的缩写。

        DECIMAL（M,D）：最大取值范围与 double 相同，给定 decimal 的有效取值范围由 M 和 D 决定。


特点：
    1、M 代表整数部位 + 小数部位的总体长度，D 代表小数部位的长度。如果超过范围，则插入临界值。（5.7之后会直接报错）

    2、M 和 D 都是可以省略的，如果 decimal，则 M 默认为10，D 默认为 0。如果是 float 和 double，则会根据插入的数值的精度来决定精度。

    3、定点型的精确度较高，在金融里算是标准类型，用于存储金额等数据。

```

字符型

```
短文本：
    分 类：
        char(M) ：范围在 0 ~ 255 之间的整数
        varchar（M）：范围在 0 ~ 65535 之间的整数

    特 点：
        char ：固定长度，默认长度是1，比较耗费空间，但是效率高
        varchar ：可变长度的字符 ，默认长度是 255 ，节省空间，但是效率较低



长文本（富文本）：
    分类：
        text ：
        blob（较大的二进制）：

```

```
```

日期型

```
分 类 ：
    date ：占用4个字节，范围 1000-01-01 ~ 9999-12-31。只保存日期
    datetime ：占用8个字节，范围 1000-01-01 00:00:00 ~ 9999-12-31 23:59:59。保存日期 + 时间
    timestamp ：占用4个字节，范围 19700101080001 ~ 2038年的某个时刻。保存日期 + 时间
    time ：占用3个字节，范围 -838:59:59 ~ 838:59:59。只保存时间
    year ：占用 1 个字节，范围 1901 - 2155。只保存年


datetime 与 timestamp的区别 ：
    时间范围：timestamp的范围要比 datetime 范围小。
    实际时区：timestamp 和实际时区有关，更能反映实际的日期，而 datetime 则只能反映出插入时的当地时区。
    timestamp 的属性受 Mysql 版本和 SQLMode 的影响很大，而 datetime 不会。

```

约束

常见约束

```
    含义：一种限制，用于限制表中的数据，为了保证表中的数据的准确和可靠性

    六大约束：
        NOT NULL：非空，用于保证该字段的值不能为空

        DEFAULT：默认，用于保证该字段有默认值

        PRIMARY KEY：主键，用于保证该字段的值具有唯一性，并且非空（默认生成索引）

        UNIQUE：唯一，用于保证该字段的值具有唯一性，可以为空（默认生成索引）

        CHECK：检查约束【mysql5.8以后才支持】

        FOREIGN KEY：外键，用于限制两个表的关系，保证该字段的值必须来自于主表的关联列的值。
                                    在从表添加外键约束，用于引用主表中某列的值。

```

主键约束、唯一约束和外键约束特点

```
    主键和唯一的区别：
        相同点：
            都保证了唯一性。
        不同点：
            主键约束的字段不允许为空，唯一约束的字段允许为空；
            主键约束一个表里只允许存在一个，唯一约束可以存在多个。

    外键约束：
        1、要求在从表设置外键关系


```

```
        2、从表的外键列的类型和主表的关联列的类型要求一致或兼容，但两者的字段叫什么并不要求一致。
        3、主表的关联列必须是一个KEY（一般是主键或唯一）
        4、插入数据时，先插入主表，再插入从表
        5、删除数据时，先删除从表，在删除主表
```

列级约束

```
-- 语法：
    -- 直接在字段名和类型后面追加 约束类型即可。

-- 只支持：默认，非空，主键，唯一


CREATE TABLE stuinfo (
        -- 主键约束
        id INT PRIMARY KEY,
        -- 非空约束
        stuName VARCHAR(20) NOT NULL,
        -- 检查约束（此处仅为展示，并不支持 检查约束）
        gender CHAR(1) CHECK(gender='男' or gender ='女'),
        --唯一约束
        seat INT UNIQUE,
        -- 默认约束
        age INT DEFAULT 19,
        -- 外键约束（此处仅为展示，并不支持 外键约束）
        majorId INT REFERENCES major(id)
)

CREATE TABLE major(
                id INT PRIMARY KEY,
                majorName VARCHAR(20)
)

```

表级约束

```
-- 语法：
    -- 在所有字段的最下面：【constraint 约束名】 约束类型（字段名）

-- 写【 constraint 约束名】
    CREATE TABLE stuinfo (
            id INT,
            stuName VARCHAR(20),
            gender CHAR(1),
            seat INT,
            age INT,
            majorId INT,
            -- 主键约束
            CONSTRAINT pk PRIMARY KEY(id),
            -- 唯一约束
            CONSTRAINT uq UNIQUE(seat),
            -- 检查约束（此处仅为展示，并不支持 检查约束）
            CONSTRAINT ck CHECK(gender='男' or gender ='女'),
            -- 外键约束

```

```
            CONSTRAINT fk_stuinfo_major FOREIGN KEY (majorid) REFERENCES
major(id)
    )

-- 不写【 constraint 约束名】
    CREATE TABLE stuinfo (
            id INT,
            stuName VARCHAR(20),
            gender CHAR(1),
            seat INT,
            age INT,
            majorId INT,
            -- 主键约束
            PRIMARY KEY(id),
            -- 唯一约束
            UNIQUE(seat),
            -- 检查约束（此处仅为展示，并不支持 检查约束）
            CHECK(gender='男' or gender ='女'),
            -- 外键约束
            FOREIGN KEY (majorid) REFERENCES major(id)
    )


    CREATE TABLE major(
                    id INT PRIMARY KEY,
                    majorName VARCHAR(20)
    )

```

修改表时添加约束

```
DROP TABLE if EXISTS stuinfo;
CREATE TABLE stuinfo(
    id INT,
    stuname VARCHAR(20),
    gender CHAR(1),
    seat INT,
    majorid INT
)

/*
1、添加列级约束
    ALTER TABLE 表名 MODIFY COLUMN 字段名 字段类型 约束类型;

2、添加表级约束
    ALTER TABLE 表名 ADD 【constraint 约束名】约束类型（字段名） 【外键的引用】;
*/

-- 1、添加非空约束
ALTER TABLE stuinfo MODIFY COLUMN stuname VARCHAR(20) NOT NULL;

-- 2、添加默认约束
ALTER TABLE stuinfo MODIFY COLUMN age INT DEFAULT 18;

-- 3、添加主键
-- 列级约束
ALTER TABLE stuinfo MODIFY COLUMN id INT PRIMARY KEY;
```

```sql

-- 表级约束
ALTER TABLE stuinfo ADD PRIMARY KEY(id);

-- 4、添加唯一
-- 列级约束
ALTER TABLE stuinfo MODIFY COLUMN seat INT UNIQUE;

-- 表级约束
ALTER TABLE stuinfo ADD UNIQUE(seat);

-- 5、添加外键
ALTER TABLE stuinfo ADD CONSTRAINT fk_stuinfo_major FOREIGN KEY(majorid)
REFERENCES major(id);
```

修改表时删除约束

```sql
-- 1、删除非空约束
ALTER TABLE stuinfo MODIFY COLUMN stuname VARCHAR(20) NULL;

-- 2、删除默认约束
ALTER TABLE stuinfo MODIFY COLUMN age INT;

-- 3、删除主键
ALTER TABLE stuinfo DROP PRIMARY KEY;

-- 4、删除唯一
ALTER TABLE stuinfo DROP INDEX seat;

-- 5、删除外键
ALTER TABLE stuinfo DROP FOREIGN KEY majorid;

```

列级约束与表级约束区别

```
    列级约束与表级约束区别大致为三个点：位置、支持的约束类型、是否可以起约束名
        位置：
            列级约束：在列的后面
            表级约束：在所有列的下面

        支持的约束类型：
            列级约束：语法都支持，但外键没有效果
            表级约束：默认和非空都不支持，其他支持

        是否可以起约束名：
            列级约束：不可以
            表级约束：可以（主键没有效果）

```

外键的级联删除和级联置空（不建议使用）

添加外键的语句：

```
        ALTER TABLE stuinfo ADD CONSTRAINT fk_stuinfo_major FOREIGN KEY(majorid) REFERENCES major(id);
```

为何需要级联删除或是级联置空？

我们知道，一旦添加了外键约束，那么我们是不能直接删除主表的数据，必须先删除从表数据，才能去删除主表数据。但是现在如果我们就想直接操作主表删除数据，那么就要使用级联删除或是级联置空。

添加级联删除：

非常简单，只需要在添加外键的语句后面加上 ON DELETE CASCADE；

```
        ALTER TABLE stuinfo ADD CONSTRAINT fk_stuinfo_major FOREIGN KEY(majorid) REFERENCES major(id) ON DELETE CASCADE;
```

添加级联置空：

在添加外键的语句后面加上 ON DELETE SET NULL；

```
        ALTER TABLE stuinfo ADD CONSTRAINT fk_stuinfo_major FOREIGN KEY(majorid) REFERENCES major(id) ON DELETE SET NULL;
```

TCL（数据库事务）

事务介绍

事务：

事务由单独单元的一个或多个SQL语句组成，在这个单元中，每个MySQL语句是相互依赖的。而整个单独单元作为一个不可分割的整体，如果单元中某条SQL语句一旦执行失败或产生错误，整个单元将会回滚。所有受到影响的数据将返回到事务开始以前的状态；如果单元中的所有SQL语句均执行成功，则事务被顺利执行。

说人话：

事务就是要做的多件事情，要不全部做成功，要不全部不做。

事务的四个特性

事务的 ACID 属性：

1、原子性

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

2、一致性

事务必须使数据库从一个一致性状态变换到另外一个一致性状态。

3、隔离性

事务的隔离性是指一个事务的执行不能被其他食物干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能相互干扰。

4、持久性

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。

事务的创建

```
隐式事务：
    事务没有明显的开启和结束的标记（比如 insert 、update、delete 语句）

显式事务：
    事务具有明显的开启和结束的标记（必须先将自动提交功能禁用：set autocommit=0；）

事务的过程：
    1、开启事务
        set autocommit=0；
        start transaction；（可写可不写）

    2、编写SQL语句
        语句1；
        语句2；
        ...

    3、结束事务
        commit；提交事务
        rollback；回滚事务

```

事务造成的并发问题（脏读等）

```
为什么会出现并发问题？
    对于同时运行的多个事务，当这些事务访问数据库中相同的数据时，如果没有采取必要的隔离机制，就会导致各种并发问题。

```

脏读

```
    脏读：
        脏读就是指 T1 正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，T2 也访问这个数据，然后使用了这个数据。这时 T1 因为某些原因回滚了这些数据，那么是不是意味着，T2 刚才使用的这些数据，是无效的。

    例：现在有 A、B、两个人，每个人账户上都有 1000 块钱。
        现在 A 正准备往 B 的账户上转账 500 ，但是这个数据还没有提交，这时 B 去查账户发现自己的余额变成了 1500 ，B 很开心，于是去高级酒店大吃一顿。但是 A 正打算提交数据的时候，银行 ATM 机出现故障，钱没转成功，B 吃完大餐结账的时候一刷卡，人直接傻了，账户只有一千块。于是 B 开始了在酒店洗碗还钱的生活...

```

不可重复读

```
    不可重复读：
        是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。

    例：现在 A 的账户上有 1000 块钱。
        A 看中了一辆太空飞船，准备去银行提款，路上打开手机看了一眼自己的余额是 1000，虽然有点肉痛，但是他还是毅然决然的要买这辆太空飞船。来到 ATM 机，掏出钱包，拿出银行卡放入、输入密码，一套动作行云流水，一气呵成，正当他准备输入金额提款，突然发现自己的余额变成了 500 。他人傻了，这时手机短信提示音响起，他拿起一看，原来今天是还花呗的日子，自己设定了自动换钱，所以时间一到，直接就自动扣款了。A 心想看来今天飞船是买不了了，还是回家吃泡面吧。

```

幻读

```
    幻读：
        是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

    例：
        程序员某一天去消费，花了2千元，然后他的妻子去查看他今天的消费记录（全表扫描FTS，妻子事务开启），看到确实是花了2千元，就在这个时候，程序员花了1万买了一部电脑，即新增INSERT了一条消费记录，并提交。当妻子打印程序员的消费记录清单时（妻子事务提交），发现花了1.2万元，似乎出现了幻觉，这就是幻读。

```

幻读与不可重复读的区别

```
幻读与不可重复读的概念很相像，但是也是有本质上的区别的。

不可重复的的重点是对同一数据的修改：
        同样的条件，读取过的数据，再次读取出来就发现值不一样了。

幻读的重点在于新增或者删除：
        同样的条件，第 1 次和第 2 次读出来的记录数不一样。
```

事务的隔离级别

查询及设置隔离级别

```sql
-- 每启动一个 mysql 程序，就会获得一个单独的数据库连接，
-- 每个数据库连接都有一个全局变量 @@tx_isolation，表示当前的事务隔离级别。

-- 查看当前的隔离级别：
    SELECT @@tx_isolation;
    -- MySQL8.0：
        SELECT @@transaction_isolation;



-- 设置当前 mysql 连接的隔离级别：
    set transaction isolation level read committed;

-- 设置数据库系统的全局的隔离级别：
```

```
    set global transaction isolation level read committed;
```

数据库事务的隔离性

```
    数据库事务的隔离性：
            数据库系统必须具有隔离并发允许各个事务的能力，使它们不会相互影响，避免各种并发问题。
```

隔离级别

```
    隔离级别：
        一个事务与其他事务隔离的程度称为隔离级别
        数据库规定了多种事务隔离级别，不同隔离级别对应不同的干扰程度，隔离级别越高，数据一致性就越好，但并发性也会相对变弱。

    MySQL 的四种隔离级别：
        读未提交（Read uncommitted）：
            最低级别的隔离。
            允许一个事务读取到其他事务并未提交的变更。
            脏读、不可重复读和幻读的问题都会出现。

        读取已提交（Read committed，Oracle、PostgreSQL、SQL Server默认模式）：
            只允许一个事务读取到其他事务已经提交的变更。
            可以避免脏读，但是不可重复读和幻读依旧存在。

        可重复读（Repeatable read，MySQL默认模式）：
            确保事务可以多次从一个字段中读取相同的值。
            在这个事务的持续期间，禁止其他事务对这个字段进行更新。
            可以避免脏读和不可重复读的发生，但是无法防止幻读。

        串行化(Serializable，SQLite默认模式）：
            最高级别的隔离。
            在该级别下，所有事务都是串行化执行，简单来说可以理解原子性，事务A不操作完，事务B就无法操作。
            完全消除脏读、不可重复读和幻读的发生，但是非常的损耗数据库性能，以及效率低下。
```

回滚点

```sql
-- 在一个事务中，有可能我们只想回滚一部分操作，这个时候就可以设置回滚点

-- 回滚点 savepoint 的使用
    set autocommit=0;
    START TRANSACTION;
    DELETE FROM account WHERE id=25;
    SAVEPOINT a; # 设置保存点
    DELETE FROM account WHERE id=28;
    ROLLBACK TO a; # 回滚到保存点
    -- 最终结果：id为 25 的被删除，而 28 的数据则被回滚。

```

delete 和 truncate 在事务使用时的区别

```
 -- 总结：delete 可以回滚，truncate 不能回滚

-- 演示 delete：
    SET autocommit = 0;
    START TRANSACTION;
    DELETE FROM account;
    ROLLBACK;

-- 演示 truncate
    SET autocommit = 0;
    START TRANSACTION;
    TRUNCATE TABLE account;
    ROLLBACK;

```

视图

```
什么是视图：
        MySQL 从5.0.1版本开始提供视图功能。一种虚拟存在的表，同真实表一样，视图也由列和行构成，但视图并不实际存在于数据库中。行和列的数据来自于定义视图的查询中所使用的表，并且还是在使用视图时动态生成的。

    简单来说就是：视图是一种虚拟存在的表，是一个逻辑表，本身并不包含数据。作为一个select语句保存在数据字典中的。
```

创建视图

```
 -- 语法
    create view 视图名 as 查询语句;

-- 案例
    -- 查询邮箱中包含 a 字符的员工名、部门名和工种信息
        CREATE VIEW myv1
        AS
        SELECT last_name,department_name,job_title FROM employees e
        JOIN departments d on e.department_id = d.department_id
        JOIN jobs j ON j.job_id = e.job_id;

    -- 创建视图查看每个部门的平均工资
        CREATE VIEW myv2
        AS
        SELECT AVG(salary) ag,department_id
        FROM employees
        GROUP BY department_id;

    -- 查看平均工资最低的部门信息
        SELECT * FROM myv2 ORDER BY ag LIMIT 1;

    -- 查询平均工资最低的部门名和工资
        SELECT d.* FROM departments d WHERE
        d.department_id = (
        SELECT department_id FROM myv2 ORDER BY ag LIMIT 1;)
```

修改视图

```
-- 修改视图有两种方式：
    -- 方式一：
        create or replace view 视图名
        as
        查询语句;
    -- 例：
        CREATE OR REPLACE VIEW myv3
        AS
        SELECT AVG(salary),job_id
        FROM employees
        GROUP BY job_id;

    -- 方式二：
        alter view 视图名
        as
        查询语句;
    -- 例：
        ALTER VIEW myv3
        AS
        SELECT * FROM employees;
```

删除视图和查看视图

```
-- 删除视图
    -- 语法：
        DROP VIEW 视图名;
    -- 例：
        DROP VIEW myv1;

-- 查看视图
    -- 语法：
        DESC 视图名;
    -- 例：
        DESC myv1;
```

视图和表的对比

	创建语法的关键字	是否实际占用物理空间	使用
视图	create view	只是保存了sql逻辑	增删改查，一般不能增删改
表	create table	保存了数据	增删改查

变量

系统变量

```
系统变量分为两类：
        全局变量
        会话变量

说明：
```

```
    系统变量由系统提供，不是用户定义，当服务器启动时就会创建并赋予默认值来供我们使用。根据作用域的不同来区分。

作用域：
    全局变量：顾名思义就是全局有效
    会话变量：仅仅针对于当前会话（连接）有效

语法：
    1、查看所有的系统变量
        全局变量：SHOW GLOBAL VARIABLES;
        会话变量：SHOW SESSION VARIABLES;

    2、查看满足条件的部分系统变量
        全局变量：SHOW GLOBAL VARIABLES LIKE '% {查找条件} %'
        会话变量：SHOW SESSION VARIABLES LIKE '% {查找条件} %'

    3、查看指定的某个系统变量的值
        全局变量：SELECT GLOBAL 系统变量名;
        会话变量：SELECT SESSION 系统变量名;

    4、为某个系统变量赋值
        方式一：
            全局变量：SET GOLBAL 系统变量名 = 值;
            会话变量：SET SESSION 系统变量名 = 值;

        方式二：
            全局变量：SET @@GOLBAL 系统变量名 = 值;
            会话变量：SET @@SESSION 系统变量名 = 值;

注意：如果是全局级别，则加 GOLBAL ，如果是会话级别，则需要加上 SESSION，如果不写，则默认SESSION;
```

自定义变量

```
自定义变量分为两类：
        用户变量
        局部变量

作用域：
    用户变量：针对于当前会话（连接）有效，和会话变量的作用域一样
    局部变量：局部变量比较特殊，仅仅在定义它的 begin end 块中有效

说明：
    变量是用户自定义的

语法：
    1、声明并初始化
        用户变量:
            SET @用户变量名=值；或
            SET @用户变量名:=值；或
            SELECT @用户变量名:=值；

        局部变量：
            DECLARE 变量名 类型;
            DECLARE 变量名 类型 DEFAULT 值;


```

```
    2、赋值（更新变量的值）
        用户变量：
            方式一：与声明的语法相同
                SET @用户变量名=值；或
                SET @用户变量名:=值；或
                SELECT @用户变量名:=值；

            方式二：使用 SELECT INTO
                SELECT 字段 INTO 变量名 FROM 表；

        局部变量：
            SET 局部变量名=值；或
            SET 局部变量名:=值；或
            SELECT @局部变量名:=值；

            方式二：使用 SELECT INTO
                SELECT 字段 INTO 局部变量名 FROM 表；

    3、查看变量的值
        用户变量：
            SELECT @用户变量名；

        局部变量名：
            SELECT 局部变量名；
```

存储过程（了解）

```
存储过程是什么？
    存储过程可以说是一个记录集吧，它是由一些SQL语句组成的代码块，这些SQL语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。

存储过程的好处：
    1. 由于数据库执行动作时，是先编译后执行的。然而存储过程是一个编译过的代码块，所以执行效率要比SQL语句高。
    2. 一个存储过程在程序在网络中交互时可以替代大堆的SQL语句，所以也能降低网络的通信量，提高通信速率。
    3. 通过存储过程能够使没有权限的用户在控制之下间接地存取数据库，从而确保数据的安全。
```

存储过程的参数列表

```
注意：
    1、参数列表包含三部分：
        参数模式    参数名    参数类型
        举例：
        IN stuname VARCHAR(20)

        参数模式：
            IN：该参数可以作为输入，也就是该参数需要调用方法入值。
            OUT：该参数可以作为输出，也就是该参数可以作为返回值。
            INOUT：该参数既可以作为输入又可以作为输出，也就是该参数既需要传入值，又可以返回值。

    2、如果存储过程体仅仅只有一句话，BEGIN END 可以省略。
    3、存储过程体中的每条SQL语句的结尾要求必须加分号。
```

```
    4、如果要自定义结束符，则使用 DELIMITER 重新设置。
        语法：
            DELIMITER 结束标记
        举例：
            DELIMITER $
    5、存储过程一旦创建成功里面的SQL语句就不能修改，所以如果要修改，那就只能删除后重新创建。
```

存储过程的创建

```
创建语法：
    CREATE PROCEDURE 存储过程名 (参数列表)
    BEGIN
            存储过程体（一组合法的SQL语句）
    END 结束标记

调用语法：
    CALL 存储过程名（实参列表）；

举例：
    1、创建一个空参列表的存储过程并调用
        DELIMITER $
        CREATE PROCEDURE myp1()
        BEGIN
            INSERT INTO boys(boyName,userCP)
            VALUES('老王',800),('李姐',100);
        END $
        CALL myp1();

-- 创建带 in 模式参数的存储过程
    2、创建存储过程实现，根据女神名，查询对应的男神信息
        CREATE PROCEDURE myp2 (IN beautyName VARCHAR ( 20 ))
        BEGIN
            SELECT bo.*
            FROM boys bo
            RIGHT JOIN beauty b ON bo.id = b.boyfriend_id
            WHERE b.`name` = beautyName;
        END;

        #调用
        CALL myp2('赵敏')

    3、
        CREATE PROCEDURE myp3 (IN id INT(11), IN boyname VARCHAR(20))
        BEGIN
            DECLARE result INT DEFAULT 0;#声明并初始化

            SELECT COUNT(*) INTO result #赋值
            FROM boys
            WHERE boys.id = id
            AND boys.boyName = boyname;

            SELECT IF(result > 0,'成功','失败'); #使用
        END;

        CALL myp3(16,'李姐');

```

```sql
-- 创建带 out 模式的存储过程
    1、根据女神名，返回对应的男神名
        CREATE PROCEDURE myp4(IN beautyName VARCHAR(20),OUT boyName VARCHAR(20))
        BEGIN
            SELECT bo.boyName INTO boyname
            FROM boys bo
            INNER JOIN beauty b ON bo.id = b.boyfriend_id
            WHERE b.`name` = beautyName;
        END;

        CALL myp4('小昭',@bName);

        SELECT @bName;

    2、根据女神名，返回对应的男神名和男神魅力值
        CREATE PROCEDURE myp5(IN beautyName VARCHAR(20),OUT boyName VARCHAR(20),OUT userCP INT)
        BEGIN
            SELECT bo.boyName ,bo.userCP INTO boyName,userCP
            FROM boys bo
            INNER JOIN beauty b ON bo.id = b.boyfriend_id
            WHERE b.`name` = beautyName;
        END;

        CALL myp5('小昭',@bName,@usercp);

        SELECT @bName,@usercp;

-- 创建带 inout 模式参数的存储过程
    1、传入 a 和 b 两个值，最终 a 和 b 都翻倍并返回
        CREATE PROCEDURE myp6 (INOUT a INT,INOUT b INT)
        BEGIN
            SET a=a*2;
            SET b=b*2;
        END;
        -- 创建两个赋值的变量用于传入
        SET @n=2;
        SET @m=2;
        CALL myp6(@n,@m);
        SELECT @n,@m;
```

存储过程的删除

```
语法：
    drop procedure 存储过程名

例如：
    DROP PROCEDURE myp1();

注意：删除只能一个一个的删，不能批量删除！
        错误示范 ：
            DROP PROCEDURE myp1(), myp2();
```

查看存储过程的结构

```
语法：
    SHOW CREATE PROCEDURE 存储过程名;

例如：
    SHOW CREATE PROCEDURE myp2;
```

流程控制结构

```
流程控制结构分为以下几种
        顺序结构：程序从上往下依次执行
        分支结构：程序从两条或多条路径中选择一条去执行
        循环结构：程序在满足一定条件的基础上，重复执行一段代码
```

分支结构

CASE 结构

```
情况1：类似于 java 中的 switch 语句，一般用于实现等值判断
语法：
        CASE 变量 | 表达式 | 字段
        WHEN 要判断的值 THEN 返回的值1或语句1
        WHEN 要判断的值 THEN 返回的值2或语句2；
        ...
        ELSE 要返回的值n或语句n
        END CASE;

情况2：类似于 java 中的多重 IF 语句，一般用于实现区间判断
语法：
    CASE 变量 | 表达式 | 字段
    WHEN 要判断的条件1 THEN 返回的值1或语句1；
    WHEN 要判断的条件2 THEN 返回的值2或语句2；
    ...
    ELSE 要返回的值n或语句n；
    END CASE;

特点：
  1、可以作为表达式，嵌套在其他语句中使用，可以放在任何地方，BEGIN END 中或 BEGIN END
的外面
  2、可以作为独立的语句去使用，只能放在 BEGIN END 中。
  3、如果 WHEN 中的值满足或条件成立，则执行对应的 THEN 后面的语句，并且结束 CASE。
        如果都不满足，则执行 ELSE 中的语句或值。
  4、ELSE 可以省略，如果 ELSE 省略了，并且所有 WHEN 条件都不满足，则返回 NULL

```

IF 结构

```
    注意！是 IF 结构，不是 IF 函数

功能：
    实现多重分支

语法：
    if 条件1 then 语句1;
```

```
    elseif 条件2 then 语句2;
    ...
    【else 语句n;】
    end if;
应用在 begin end 中

案例:
    根据传入的数据,来显示等级,比如传入的成绩:90-100,返回A,80-90,返回B,60-80,返回C,否则,返回D。
    CREATE FUNCTION test_if(score INT) RETURNS CHAR
    BEGIN
        IF score>=90 AND score<=100 THEN RETURN 'A';
        ELSEIF score>=80 THEN RETURN 'B';
        ELSEIF score>=60 THEN RETURN 'C';
        ELSE RETURN 'D';
        END IF;
    END;
```

循环结构

```
分类:
    while、loop、repeat

循环控制:
    iterate 类似于 continue,结束本次循环进入下一次
    leave 类似于 break,结束。

```

while 结构

```
#语法:
    【标签:】while 循环条件 do
            循环体;
     end while【标签】;

#案例:
  # 批量插入,根据次数插入到 boy 表中,如果次数>20则停止;
        DROP PROCEDURE test_while1;
        CREATE PROCEDURE test_while1(IN insertCount INT)
        BEGIN
                DECLARE i INT DEFAULT 1;
                a:WHILE i<=insertCount DO
                        IF i>=20 THEN LEAVE a;
                        END IF;
                        INSERT INTO boy(username,age) VALUES(CONCAT('xiaohua',i),'1');
                        SET i=i+1;
                END WHILE a;
        END;

  # 批量插入,根据次数插入到 boy 表中,只插入偶数次
        DROP PROCEDURE test_while1;
        CREATE PROCEDURE test_while1(IN insertCount INT)
        BEGIN
                DECLARE i INT DEFAULT 1 ;
                a:WHILE i<=insertCount DO
```

```
                        IF MOD(i,2)==0 THEN iterate a;
                        INSERT INTO boy(username,age)
VALUES(CONCAT('xiaohua',i),'1');
                        END IF;
                        SET i=i+1;
                END WHILE a;
        END;
```

loop 结构

```
语法：
    【标签:】loop
            循环体；
     end loop【标签】;

```

repeat 结构

```
语法：
    【标签:】repeat
            循环体；
     until 结束循环的条件
     end repeat【标签】;

```

循环结构特点总结

```
特点：
    while：先判断后执行
    loop：没有条件的死循环
    repeat：先执行后判断
```