第六章：MySQL基础

第1节：数据库简介

1.1 理解

就是一款用于持久化保存数据的软件
保存数据的技术：
    1. 变量 2. 对象 3. 数组 4. 集合 将数据保存到内存 易失性
    5. IO 可以持久化保存 操作性差
数据库既可以持久化保存数据 又非常灵活的操作数据

1.2 特点

1. 可以持久化保存数据
2. 保存数据是有组织 有结构
3. 数据库是以表的形式 保存数据的
4. 提供SQL语句 用于操作里面的数据

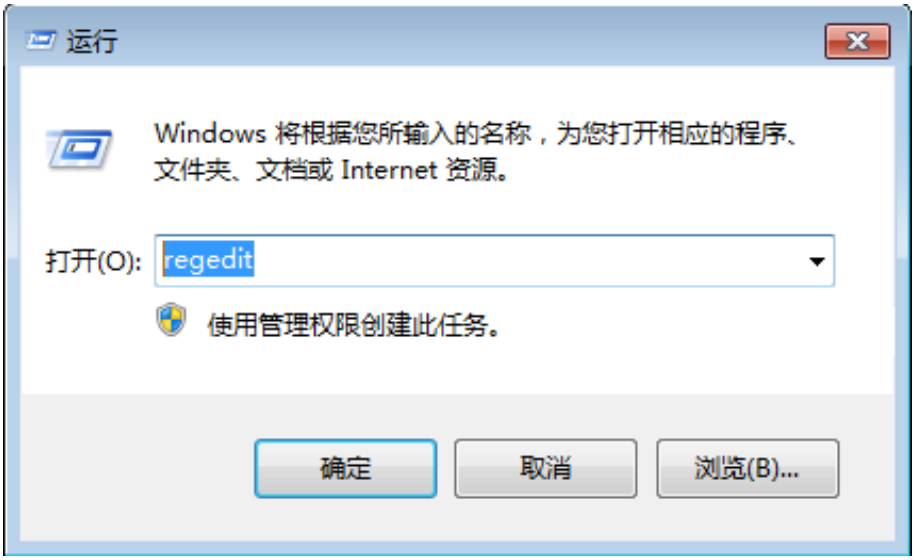
1.3 分类

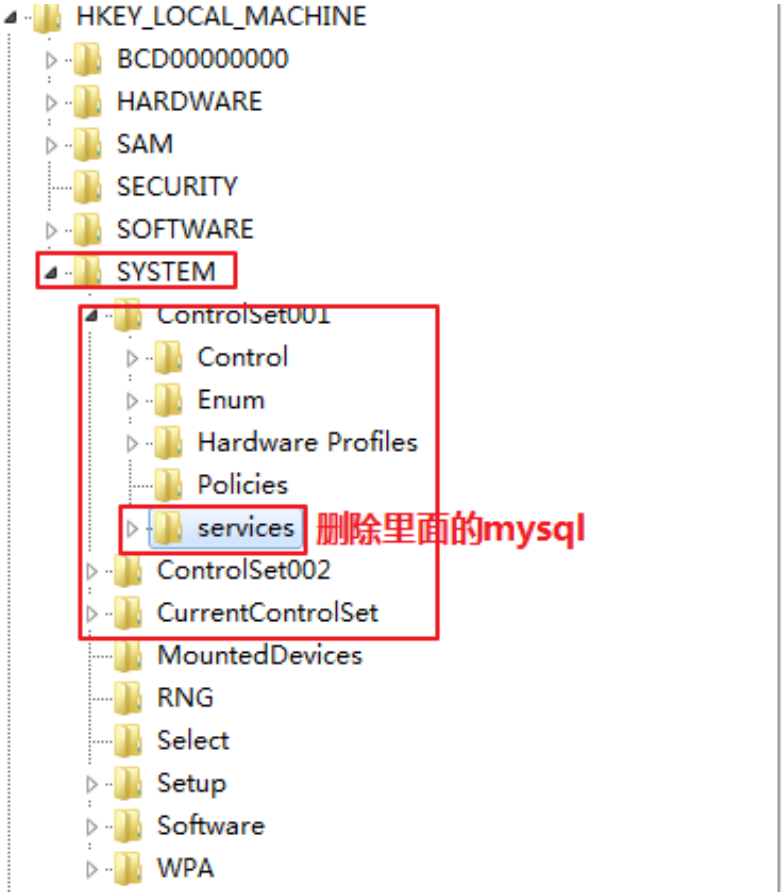
1. 关系型数据库
    Oracle、MySQL、SQLServer.....
2. 非关系型数据库
    redis、MongoDB、HBase

数据库类型	特性	优点	缺点
关系型数据库 SQLite、Oracle、mysql	1、关系型数据库，是指采用了关系模型来组织数据的数据库； 2、关系型数据库的最大特点就是事务的一致性； 3、简单来说，关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。	1、容易理解：二维表结构是非常贴近逻辑世界一个概念，关系模型相对网状、层次等其他模型来说更容易理解； 2、使用方便：通用的SQL语言使得操作关系型数据库非常方便； 3、易于维护：丰富的完整性(实体完整性、参照完整性和用户定义的完整性)大大减低了数据冗余和数据不一致的概率； 4、支持SQL，可用于复杂的查询。	1、为了维护一致性所付出的巨大代价就是其读写性能比较差； 2、固定的表结构； 3、高并发读写需求； 4、海量数据的高效率读写；
非关系型数据库 MongoDb、redis、HBase	1、使用键值对存储数据； 2、分布式； 3、一般不支持ACID特性； 4、非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。	1、无需经过sql层的解析，读写性能很高； 2、基于键值对，数据没有耦合性，容易扩展； 3、存储数据的格式：nosql的存储格式是key,value形式、文档形式、图片形式等等，文档形式、图片形式等等，而关系型数据库则只支持基础类型。	1、不提供sql支持，学习和使用成本较高； 2、无事务处理，附加功能bi和报表等支持也不好；

1.4 安装和卸载

```
1. 安装 mysql-installer-community-8.0.28.0.msi  参考《MySQL8安装教程.pdf》

2. 卸载 MySQL
    ① 控制面板中 先卸载软件
    ② 删除安装的目录中 MySQL文件夹
      C:\Program Files\MySQL
         C:\ProgramData\MySQL
    ③ 删除注册表  win+r
    ④ 关机 开机重新安装
3. 测试
      doc命令行中 输入：mysql -uroot -p密码
      mysql -hIP地址 -uroot -p密码 远程连接别人的数据库

       net stop 服务的名字; 停止服务
       net start 服务的名字; 启动服务
4. 安装数据库客户端
         SQLyog 或者 Navicat
```





第2节：SQL命令

2.1 SQL分类

```
数据库存储结构
    一个数据库系统 ----> 保存很多个数据库
    一个库中 ----> 保存很多数据表
    表中  ----> 表头(字段/列名) 一条条数据

分类:
    DDL 数据定义语言 创建/删除库  创建、删除和修改表
    DML 数据操作语言 数据库表中数据的增删改操作
    DQL 数据查询语言 数据库表中数据的查询
    DCL 数据控制语言 数据库用户权限的操作
    事务
```

2.2 DDL

```
库操作
  1. show databases;  查看数据库系统中所有的库
  2. create database if not exists 库名; 创建一个新的数据库
  3. drop database if exists 库名; 删除一个数据库
    4. show create database 库名; 查看创建数据库语句
    5. use 库名;  选中库
表操作
    1. show tables; 查看库中的所有表
    2. create table if not exists 表名(列名 类型,列名 类型,.......)
        完整的语句:
         create table if not exists 表名(
             列名1 类型 【约束】,
             列名2 类型 【约束】,
                .... ...  ...
           )

    3. desc 表名; 查看表结构
    4. drop table if exists 表名; 删除表
    5. 修改表
        alter table 表名 rename to 新表名; #修改表名
        alter table 表名 change column 列名 新列名 类型; #修改列名
        alter table 表名 modify column 列名 新类型; #修改列的类型
        alter table 表名 add column 新列名 类型; # 添加新列
        alter table 表名 drop column 列名; # 删除一列
```

```
约束的操作
#4. 约束
# 4.1 添加约束
# 1. 添加主键约束
ALTER TABLE users ADD PRIMARY KEY(id); # --- 表级添加
ALTER TABLE users MODIFY COLUMN id INT PRIMARY KEY; # --- 列级添加
```

```
# 2. 添加 not null
ALTER TABLE users MODIFY COLUMN NAME VARCHAR(16) NOT NULL;

# 3. 添加 default
ALTER TABLE users MODIFY COLUMN age CHAR(4) DEFAULT 18;

# 4. 添加 unique
ALTER TABLE users ADD UNIQUE(cardno); # --- 表级添加

ALTER TABLE users MODIFY COLUMN cardno VARCHAR(20) UNIQUE; # --- 列级添加


# 5. 添加 外键 foreign key
ALTER TABLE users ADD CONSTRAINT fk_u_c FOREIGN KEY(c_id) REFERENCES class(cid);

# 4.2 删除约束
# 1. 删除主键约束
ALTER TABLE users DROP PRIMARY KEY;
ALTER TABLE users MODIFY COLUMN id INT NULL;


# 2. 删除 not null
ALTER TABLE users MODIFY COLUMN NAME VARCHAR(16) NULL;


# 3. 删除默认约束
ALTER TABLE users MODIFY COLUMN age CHAR(4);

# 4. 删除 唯一约束
ALTER TABLE users DROP INDEX cardno;

# 5. 删除外键
ALTER TABLE users DROP FOREIGN KEY fk_u_c;
ALTER TABLE users DROP INDEX fk_u_c;
```

2.3 DML

```
数据操作语言 Data Mainpulation Language 增删改
    1. insert
      格式:
      单插: insert into 表名(列名,列名,...) values(值1,值2,...)
              多插: insert into 表名(列名,列名,...) values(值1,值2,...),(值1,值2,...)
         注意:
               ① 插入值为整型或者浮点型 不用加引号 如果是字符型或者 日期类型需要加引号(单双都可以)

               ② 如果所有列的值都要插入 表名后的列名可以省略 否则不能省略
```

③ 列名的顺序可以颠倒 但是values的值需要和列名一一对应

④ 如果列的约束是 可以为空或者有默认值 可以不用添加值

⑤ 插入的值一定要符合列的类型和约束

2. delete 删除数据

格式：

```
delete from 表名 where 条件;
delete from 表名; 全部删除
truncate table 表名; 全部删除 推荐使用
```

3. update 更新数据

格式：

```
update 表名 set 列名=新值,列名=新值,... where 条件;
```

注意：

1. where条件一定要加 否则所有数据都会被修改
2. 一般主键作为条件 效率高

【面试题】delete全删和truncate全删的区别?

1. DELETE 删除后面可以加where条件 truncate不可以
2. TRUNCATE 删除整个表 再新建一个表 delete是一条数据一条数据的删 truncate删除效率高
3. delete支持回滚 可以回退 truncate不支持回滚
4. delete删除不会删除自增的记录，truncate 会自增的记录从新从1开始

MySQL中常见的类型

整型：tinyint/smallint/int/bigint

浮点型：float(n,m) double(n,m) decimal(n,m) 准确的浮点型

n：代表 整数位和小数位总和

m：代表小数位的个数

字符型：

varchar(n) 变长字符串 效率相对低 n保存字符的最大个数 必写

char(n) 定长字符串 效率相对高 n保存字符的最大个数 可选 默认1

日期类型：

date 日期

time 时间

datetime 日期+时间 8字节 不受时区和服务器的版本影响

取值范围：1900-1-1~ 9999-12-31

timestamp 日期+时间 4字节 受时区和服务器的版本影响

取值范围：1970-1-1~2038-12-31

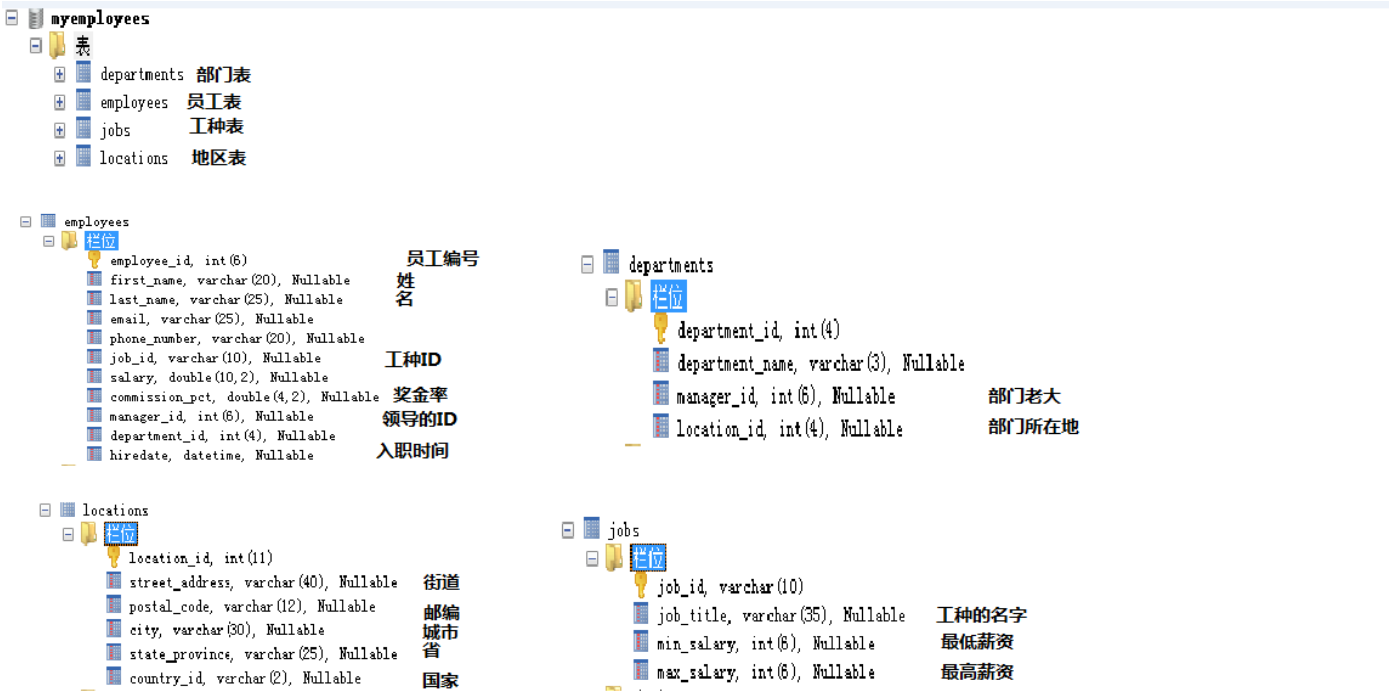
MySQL五大约束

1. 理解
   额外的限定数据库表中数据的值，保证数据的完整性(准确性和完整性)
2. 分类
   ① primary key 主键约束 保证数据不能重复 不能为null 一个表对应一个主键约束

   ② not null 非空约束 保证数据不能为null
   ③ default 默认约束 保证数据不写也有默认值
   ④ unique 唯一约束 保证数据不重复 可以为null 并且可以有多个
   ⑤ foreign key 外键约束 建立表与表之前的关系
   要求：
   1. 一个表(从表)的任意列与另一个表(主表)的主键列建立外键关系
   2. 要先创建主表 再创建从表
   3. 从表的外键字段和主表的主键字段类型一致
   4. 从表的外键的取值 一定要来源于主表的主键的值

表级约束：primary key / unique/ foreign key
列级约束：not null / default / primary key / unique

```
约束的使用
MySQL五大约束
    1. 理解
        额外的限定数据库表中数据的值，保证数据的完整性(准确性和完整性)
    2. 分类
        ① primary key 主键约束  保证数据不能重复 不能为null 一个表对应一个主键约束

        ② not null 非空约束  保证数据不能为null
        ③ default  默认约束  保证数据不写也有默认值
        ④ unique 唯一约束  保证数据不重复 可以为null 并且可以有多个
            ⑤ foreign key  外键约束 建立表与表之前的关系
               要求：
                    1. 一个表(从表)的列 与另一个表(主表)的主键列建立外键关系
                    2. 要先有主表 才能有从表
                    3. 外键字段和主键字段类型一致
                    4. 外键的取值 一定要来源于主键的值
*/
CREATE TABLE IF NOT EXISTS class(
        cid INT PRIMARY KEY,
        cname VARCHAR(12) NOT NULL
);

CREATE TABLE IF NOT EXISTS student(
        sid INT PRIMARY KEY,
        sname VARCHAR(16) NOT NULL,
        sex CHAR(4) DEFAULT '男',
        cardno VARCHAR(20) UNIQUE,
        class_id INT,
        CONSTRAINT fk_stu_cl FOREIGN KEY(class_id) REFERENCES class(cid)
);
DESC student;
```



2.4 DQL

数据查询语言：提供了各种情况的数据库表数据的查询 关键字：`select`

分类：

1. 基本查询
2. 条件查询
3. 排序查询
4. 分组查询
5. 分页查询
6. 多表查询
7. 子查询

2.4.1 基本查询

格式：

```
select 查询列表 from 表名;
```

特点：

1. 查询的结果是一张虚拟表
2. 查询的列表中可以是常量、变量、函数、字段名(列名)、表达式或者以上组合

   相当于：`System.out.print()`

代码：

```
# 1. 常量
SELECT 100;

# 2. 查表达式
SELECT 100+99;
SELECT 100+'abc';
```

```
    SELECT 100+'11abc';
    SELECT 100+'abc11';
    SELECT 100+'a11bc';

    #3. 函数
    SELECT VERSION();  -- 查询当前数据库的版本
    SELECT USER();  -- 当前用户

    # 4. 字段名
    SELECT last_name FROM employees;

    SELECT first_name,last_name FROM employees;

    # 5. 以上组合
    SELECT USER(),10+9, last_name FROM employees;

    # 6. 通配符* 查询所有的数据
    SELECT * FROM employees;
```

2.4.2 条件查询

```
格式:
    select 查询列表 from 表名 where 条件;
特点:
    条件的形式:
      1. 关系表达式: > < >= <= = <>或者!=
      2. 逻辑表达式: and or not【&& || !】
      3. 模糊查询: like / between...and... / in / is null
代码:
    # 1. 关系表达式: > < >= <= = <>或者!=
    # 案例1: 查询薪资大于5000的员工信息
    SELECT *
    FROM employees
    WHERE salary > 5000;

    # 案例2: 查询薪资不等于6000的员工的名和薪资
    SELECT last_name,salary
    FROM employees
    WHERE salary <> 6000;

    # 2. 逻辑表达式: and or not【&& || !】
    # 案例3: 查询员工薪资大于5000 , 并且小于10000 的员工的名和薪资
    SELECT last_name,salary
    FROM employees
    WHERE salary > 5000 AND salary < 10000;


    # 案例4: 查询员工薪资 小于5000 或者 大于10000的员工信息
    SELECT *
```

```
FROM employees
WHERE salary < 5000 OR salary > 10000;

# 案例5：查询员工的job_id不是"IT_PROG"的员工信息
SELECT *
FROM employees
WHERE NOT(job_id='IT_PROG');

#3. 模糊查询：like / between...and... / in / is null
/*
    % 代表任意个数任意字符
    _ 代表任意一个字符
    \ 可以转移特殊字符

*/

# 案例6：查询员工名中第一个字符是e的员工信息
SELECT *
FROM employees
WHERE last_name LIKE 'e%';

# 案例7：查询员工名中第二个字符是e 第五个字符是a的员工信息
SELECT *
FROM employees
WHERE last_name LIKE '_e__a%';

# 案例8：查询员工名中第二个字符是_的员工信息
SELECT *
FROM employees
WHERE last_name LIKE '_\_%';

# between...and...
/*
    1. 等价于 逻辑表达式的结果
    2. between后接小值 and后接大值
    3. 既包含between后的值 也包含and后的值
*/
# 案例9：查询薪资在10000-17000之前的员工的薪资和名
SELECT salary, last_name
FROM employees
WHERE salary BETWEEN 10000 AND 17000;

# in
# 案例10：查询薪资为 11000 12000 17000 14000的员工的薪资和名
SELECT salary, last_name
FROM employees
WHERE salary = 11000 OR salary = 12000;

SELECT salary, last_name
```

```
    FROM employees
    WHERE salary IN(11000,12000,17000,14000, 20000);

    # 案例11：查询薪资不为 11000 12000 17000 14000的员工的薪资和名
    SELECT salary, last_name
    FROM employees
    WHERE salary NOT IN(11000,12000,17000,14000, 20000);

    # is null
    # 案例12：查询没有奖金的员工的信息
    SELECT *
    FROM employees
    WHERE `commission_pct` IS NULL;

    # 案例13：查询没有奖金的员工的信息
    SELECT *
    FROM employees
    WHERE `commission_pct` IS NOT NULL;
```

2.4.3 排序查询

```
格式：
    select 查询列表
    from 表名
    where 条件
    order by 排序列表 asc|desc;
特点：
    1. asc 默认升序  desc降序
    2. 排序列表：单个字段、多个字段、函数、表达式、别名或者以上组合
代码：
    # 1. 单字段：可以是数值类型也可以是字符型
    # 案例1：查询员工信息  并且按照薪资升序
    SELECT *
    FROM employees
    ORDER BY salary ASC;

    # 案例2：查询员工信息  并且按照薪资降序
    SELECT *
    FROM employees
    ORDER BY salary DESC;

    # 2. 多字段：先按照order by后第一个字段排序 如果相同再按第二个字段排序
    # 案例3：查询员工信息  并且按照名降序    z-a由大到小

    SELECT *
    FROM employees
    ORDER BY last_name DESC;
```

```
# 案例4：查询员工信息  先按照薪资降序 如果薪资相同按照名降序
SELECT *
FROM employees
ORDER BY salary DESC, last_name DESC;

# 函数
# 案例5：查询员工信息 按照名的长度降序
SELECT *
FROM employees
ORDER BY LENGTH(last_name) DESC;

# 表达式
# IFNULL(如果不为null取值,如果为null的取值)
# 案例6：查询员工信息 按照年薪降序


SELECT *,salary*12*(1+IFNULL(`commission_pct`,0)) AS 年薪
FROM employees
ORDER BY salary*12*(1+IFNULL(`commission_pct`,0)) DESC

# 别名
SELECT *,salary*12*(1+IFNULL(`commission_pct`,0)) AS 年薪
FROM employees
ORDER BY 年薪 DESC
```

2.4.4 常见函数

```
常见函数
  类似Java中的方法，mysql中封装好的一些函数，调用获取到结果
  分类：
      单行函数：调用函数时，传递参数 返回一个结果
      分组函数(多行)：调用函数时,传递一组参数 返回一个结果
  单行函数：
     1. 字符串操作的函数
    length(); 获取字符长度
    upper(); 将字符转为大写
    lower(); 将字符转为小写
    substr(); 截取字符
    trim(); 去除字符两侧空格
    lpad(); 左填充
    rpad(); 右填充
    concat(); 拼接字符串
     2. 数学函数
    ceil();  向上取整
    floor(); 向下取整
    rand(); 随机数
    abs(); 取绝对值
    mod(); 取余
```

```
    round(); 四舍五入
    truncate(); 截断

     3. 日期函数
    now(); 获取当前的日期+时间
    curdate(); 获取当前日期
    curtime(); 获取当前的时间
    date_format(要格式化的日期,格式化规则); 格式化日期和时间

     4. 流程控制函数
    ① if();
    ② case
        when 条件 then 取值
        when 条件 then 取值
        ....
        else 取值
    end
  分组函数:
    sum(); 求和
    avg(); 平均值
    max(); 最大值
    min(); 最小值
    count(); 求个数
  特点:
    1. sum/avg只支持数值类型 max/min/count支持所有类型
    2. 自动的将null值 忽略掉
```

2.4.5 分组查询

```
格式:
    select 查询列表 ----> ④
    from 表名    ----> ①
    where 分组前筛选 ----> ②
    group by 分组列表 ----> ③
    having 分组后筛选 ----> ④
    order by 排序列表 asc|desc ----> ⑤
特点:
    1. 查询的结果只能查分组函数的结果和分组字段的值
    2. 分组前筛选where条件 和 分组后筛选having
代码:
    # 案例1: 查询每个部门的平均薪资
    SELECT AVG(salary), department_id, last_name
    FROM employees
    GROUP BY department_id;

    # 案例2: 查询每个部门的最高薪资
    SELECT MAX(salary), department_id
    FROM employees
    GROUP BY department_id ;
```

```
    # 1. 添加where分组前筛选
    # 案例3：查询所有没有奖金的员工所属每个部门的平均薪资
    SELECT AVG(salary), department_id
    FROM employees
    WHERE `commission_pct` IS NULL
    GROUP BY department_id;

    # 2. 添加分组后筛选

    # 案例4：查询所有没有奖金的员工所属每个部门的平均薪资大于5000的 部门id和平均薪资
     SELECT AVG(salary), department_id
     FROM employees
     WHERE `commission_pct` IS NULL
     GROUP BY department_id
     HAVING AVG(salary) > 5000;

     # 3. 添加排序
     # 案例5：查询所有没有奖金的员工所属每个部门的平均薪资大于5000的 部门id和平均薪资，并且按照平均薪资降序
     SELECT AVG(salary), department_id
     FROM employees
     WHERE `commission_pct` IS NULL
     GROUP BY department_id
     HAVING AVG(salary) > 5000
     ORDER BY AVG(salary) DESC;

     # 4. 多个字段分组
     # 案例6：查询每个部门每个工种的 平均薪资
     SELECT AVG(salary), department_id, job_id
     FROM employees
     GROUP BY department_id,job_id;
```

2.4.6 分页查询

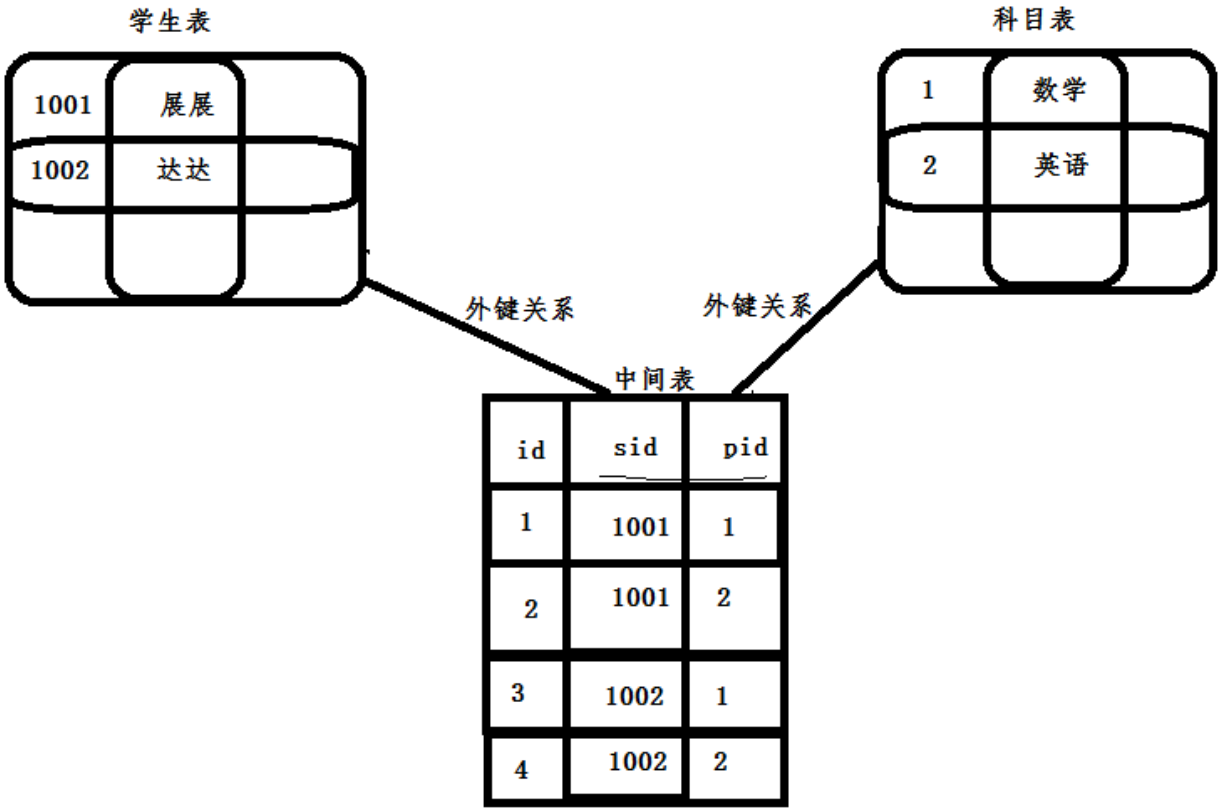
```
格式：
    select 查询列表
    from 表名
    where 条件
    group by 分组字段
    having 分组后筛选
    order by 排序列表 asc|desc
    limit 偏移量,一页查询的个数;
特点：
    1. 偏移量指跳过的个数 与页码的关系(n-1)*size
    2. 一页查询的个数
    3. limit 关键字 放在sql语句的最后面
代码：
    # 案例1：查询员工前10条数据 第一页
```

```
SELECT *
FROM employees
LIMIT 0,10;

# 案例2：查询员工11-20条数据 第二页
SELECT *
FROM employees
LIMIT 10,10;

# 案例3：查询员工21-30条数据 第三页
SELECT *
FROM employees
LIMIT 20,10;

# 案例4：查询员工数据 第n页

SELECT *
FROM employees
LIMIT (n-1)*10,10;
```

2.4.7 多表查询

2.4.7.1 表之间的关系

1. 一对一 如：丈夫与妻子 用户和身份证
   特点： 一个表中的数据 只能对应一次 另一个表的一条数据
   建立：一个表的主键字段与另一个表的主键字段建立外键关系
2. 一对多 和 多对一 ：员工与部门 、学生与班级
   特点：一个表的列对应的数据 可以 对应另一个的字段的多个数据
   建立：一个表的普通字段 与另一个表的主键字段建立外键关系
3. 多对多：如：学生和所选科目
   特点：一个表的字段多个值 对应 另一个表的字段的多个数据
   建立：通过建立中间表 中间表中建立两个字段分别与两个表建立外键关系



2.4.7.2 多表查询

1. 理解
   查询的数据来源于多个表，此时需要多表查询
2. 分类
   sql92语法
   隐式内连接查询
   格式：

```
select 查询列表
from 表1 别名1, 表2 别名2 , ....
where 表1.字段名 = 表2.字段名 and ....
and 分组前筛选条件
group by 分组字段
having 分组后筛选
order by 排序列表 asc|desc
limit 偏移量,一页个数
```

   特点：
   1. 连接条件一定要写 否则会出现笛卡尔乘积现象
   2. 根据连接条件格式
      ① 等值连接：连接条件是等号
      ② 非等值连接：连接添加不是等号
      ③ 自连接：自己连接自己
   3. from后多表的顺序没有要求
   4. n张表连接 连接条件至少n-1个

```sql
# ① 等值连接：连接条件是等号
# 案例1：查询员工的名字和其对应部门的名字
SELECT e.last_name, d.`department_name`
FROM employees e, departments d
WHERE e.`department_id` = d.`department_id`;

# 添加条件
# 案例2：查询没有奖金的员工的名字和其对应部门的名字
SELECT e.`last_name`, d.`department_name`,`commission_pct`
FROM employees e, departments d
WHERE e.`department_id` = d.`department_id`
AND `commission_pct` IS NULL;

# 添加分组
# 案例3：查询每个部门的员工个数（查询部门的名称和员工个数）

SELECT d.`department_name`, COUNT(*)
FROM departments d, employees e
WHERE d.`department_id` = e.`department_id`
GROUP BY d.`department_name`;

# 添加分组后筛选
# 案例4：查询哪些部门的员工个数>2（查询部门的名称和员工个数）
SELECT d.`department_name`, COUNT(*)
FROM departments d, employees e
WHERE d.`department_id` = e.`department_id`
GROUP BY d.`department_name`
HAVING COUNT(*) > 2;
```

```sql
# 添加排序
# 案例5：查询哪些部门的员工个数>2,并且按照个数降序（查询部门的名称和员工个数）
SELECT d.`department_name`, COUNT(*)
FROM departments d, employees e
WHERE d.`department_id` = e.`department_id`
GROUP BY d.`department_name`
HAVING COUNT(*) > 2
ORDER BY COUNT(*) DESC;


# ② 非等值连接：连接添加不是等号
# 案例6：查询每个员工的名字，薪资和所属等级
SELECT e.`last_name`,e.`salary`,s.`grade`
FROM employees e, sal_grade s
WHERE e.`salary` BETWEEN s.`min_salary` AND s.`max_salary`;

# 三张表
# 案例7：查询员工的名字 对应部门名字和对应工种名

SELECT e.`last_name`, d.`department_name`, j.`job_title`
FROM employees e, departments d, jobs j
WHERE e.`job_id` = j.`job_id` AND e.`department_id` = d.`department_id`;

# ③ 自连接
# 案例8：查询每个员工的名字和其对应领导的名字，薪资，邮箱
SELECT e.`last_name`,e.`employee_id`, e.`manager_id`, m.`employee_id`, m.`last_name`, m.`salary`, m.`email`
FROM employees e, employees m
WHERE e.`manager id` = m.`employee id`;
```

sql99语法

显示内连接查询

格式：

```
select 查询列表
from 表1 别名1
inner join 表2 别名2
on 表1.字段名 = 表2.字段名
where 分组前筛选
group by 分组字段
having 分组后筛选
order by 排序列表 asc|desc
```

limit 偏移量,一个的个数

特点:

1. 将连接条件和筛选条件的分离，提高sql可读性
2. 表的顺序没有要求
3. 连接条件的格式
   ① 等值连接
   ② 非等值连接
   ③ 自连接

```
# 案例1：查询员工名和对应的部门名
SELECT e.`last_name`, d.`department_name`
FROM employees e
INNER JOIN departments d
ON e.`department_id` = d.`department_id`;

# 添加where条件
# 案例2：查询名字中包含a的员工的名字和对应的部门名
SELECT e.`last_name`, d.`department_name`
FROM employees e
INNER JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE e.`last_name` LIKE '%a%';

# 添加分组
# 案例3：查询部门名和对应的员工个数
SELECT d.`department_name`, COUNT(*)
FROM departments d
INNER JOIN employees e
ON d.`department_id` = e.`department_id`
GROUP BY d.`department_name`;
```

```
# 添加分组后筛选
# 案例4：查询部门名和对应的员工个数，筛选中个数小于5的
SELECT d.`department_name`, COUNT(*)
FROM departments d
INNER JOIN employees e
ON d.`department_id` = e.`department_id`
GROUP BY d.`department_name`
HAVING COUNT(*) < 5;

# 添加排序
# 案例5：查询部门名和对应的员工个数，筛选中个数小于5的 按照个数降序
SELECT d.`department_name`, COUNT(*)
FROM departments d
INNER JOIN employees e
ON d.`department_id` = e.`department_id`
GROUP BY d.`department_name`
HAVING COUNT(*) < 5
ORDER BY COUNT(*) DESC
LIMIT 0,3;
```

```
# 非等值查询
# 案例6：查询员工的名、薪资和薪资等级
SELECT e.`last_name`, e.`salary`, s.`grade`
FROM employees e
INNER JOIN sal_grade s
ON e.`salary` BETWEEN s.`min_salary` AND s.`max_salary`;


# 三个表
# 案例7:查询员工的名 所属部门名以及对应的工种名
SELECT e.`last_name`, d.`department_name`, j.`job_title`
FROM employees e
INNER JOIN departments d
ON e.`department_id` = d.`department_id`
INNER JOIN jobs j
ON e.`job_id` = j.`job_id`;

# 自连接
# 案例8：查询每个员工的名字 和其对应领导的名字、薪资
SELECT e.`last_name`,e.`manager_id`,m.`employee_id`, m.`last_name`,m.`salary`
FROM employees e
INNER JOIN employees m
ON e.`manager_id` = m.`employee_id`;
```

sql99语法

外连接查询

语法：

```
select 查询列表
from 表1 别名1
left|right outer join 表2 别名2
on 连接条件
where 筛选条件
group by 分组字段
having 分组后筛选
order by 排序列表 asc|desc
limit 偏移量,一页的个数;
```

特点：

1. left join以左表为主表，将主表中所有数据全部查询出来，
   以右表为副表，副表中与主表有对应关系的数据查询出来，没有的显示为null
2. right join以右表为主表，将主表中所有数据全部查询出来，
   以左表为副表，副表中与主表有对应关系的数据查询出来，没有的显示为null

3. 多表之间的顺序 不能调换，否则会影响结果

```
# 案例1：查询哪个员工没有部门
SELECT e.*, d.*
FROM employees e
LEFT OUTER JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE d.`department_id` IS NULL;

# 案例2：查询哪些部门没有员工
SELECT e.*, d.*
FROM employees e
RIGHT OUTER JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE e.`employee_id` IS NULL;
```

2.4.8 子查询

1. 理解

一个查询语句中嵌套了另一个完整的查询语句，被嵌套在里面的查询语句 称为子查询或内查询，外面的查询语句称为主查询或外查询。

2. 作用

主要为主查询提供条件的，主查询需要用到另个查询语句的结果

3. 分类

select 后面

from 后面

where 后面 *

4. 特点：

① 子查询写在小括号中

② 子查询优先于主查询执行，主查询会用到子查询的结果

③ 一般子查询放在where之后 搭配操作符使用

单行子查询：子查询结果只有一个 搭配的操作符：> < = >= <= <>

多行子查询：子查询的结果有多个 搭配的操作符：in / all任意 / any 任一

```
# 单行子查询
# 案例1：查询比Ernst高员工的信息
SELECT salary
FROM employees
WHERE last_name = 'Ernst';

SELECT *
FROM employees
WHERE salary > 6000;
# 子查询
SELECT *
FROM employees
WHERE salary > (
    SELECT salary
    FROM employees
    WHERE last_name = 'Ernst'
);

# 多行子查询
# 案例2:查询location_id为1400或1700 的部门中员工的信息

# ① 查询location_id为1400或者1700的部门
SELECT department_id
FROM departments
WHERE location_id IN(1400,1700);

# ② 将①为条件查询的部门 下的员工信息
SELECT *
FROM employees
WHERE department_id IN(
        SELECT department_id
        FROM departments
        WHERE location_id IN(1400,1700)
);
```

```sql
# 案例3：查询比job_id='IT_PROG'工种中任意员工工资低的其他工种的员工信息
# DISTINCT去重
# ① 查job_id='IT_PRG' 员工的薪资
SELECT DISTINCT salary
FROM employees
WHERE job_id='IT_PROG';

# ②

SELECT *
FROM employees
WHERE job_id <> 'IT_PROG'
AND salary < ALL(
        SELECT DISTINCT salary
        FROM employees
        WHERE job_id='IT_PROG'
);
# 等价于
SELECT *
FROM employees
WHERE job_id <> 'IT_PROG'
AND salary < (
        SELECT MIN(salary)
        FROM employees
        WHERE job_id='IT_PROG'
);
```

```sql
# 案例4：查询比job_id='IT_PROG'工种中任一员工工资低的其他工种的员工信息 (4200,4800,6000)

SELECT *
FROM employees
WHERE job_id <> 'IT_PROG'
AND salary < (
        SELECT MAX(salary)
        FROM employees
        WHERE job_id='IT_PROG'
);

# 等价于

SELECT *
FROM employees
WHERE job_id <> 'IT_PROG'
AND salary < ANY(
        SELECT DISTINCT salary
        FROM employees
        WHERE job_id='IT_PROG'
);
```

2.5 DCL

1. 理解

   数据控制语言，处理连接数据库用户的权限

2. 常见操作

```sql
# 1. 查看当前数据库系统中的用户
SELECT * FROM mysql.user;

# 2. 创建新的用户
CREATE USER 'zhanzhan'@'%' IDENTIFIED BY '123456';

# 3. 分配权限
GRANT SELECT ON 0713db.users TO 'zhanzhan'@'%';

GRANT ALL ON 0713db.* TO 'zhanzhan'@'%';


# 4. 取消权限
```

```
    REVOKE ALL ON 0713db.* FROM 'zhanzhan'@'%';
    REVOKE SELECT ON 0713db.users FROM 'zhanzhan'@'%';


    # 5. 查用户权限
    SHOW GRANTS FOR 'zhanzhan'@'%';
    SHOW GRANTS FOR 'root'@'localhost';

    # 6. 删除用户
    DROP USER 'zhanzhan'@'%';

3. 注意:
    ① 由于MySQL8加密方式和navicat不同 需要执行: alter user 'zhanzhan'@'%' identified with mysql_native_password by '123456';

    ② MySQL8以下的版本不用
```

2.6 事务

```
1. 理解
    完成某个功能的整个过程称为一个事务。如果中间出现了问题 整个事务失败 返回到开始事务前的状态

2. 使用 默认一个sql语句为一个事务
    # 1. 查看当前数据库事务的提交方式 默认自动开启事务
  SHOW VARIABLES LIKE 'autocommit';

    # 2. 改变事务的提交方式 为手动
  SET autocommit = 0;

  # 3. 开启事务
  START TRANSACTION;

  # 4. 指定事务的执行语句

  # 5. 事务执行语句成功 提交事务
  COMMIT;

  # 5. 事务执行语句失败 回滚事务
  ROLLBACK;

3. 事务的特性
    ① 原子性 不可在分割的
    ② 一致性 数据的一致性
    ③ 持久性 事务一经提交 数据的变化就是持久性的
    ④ 隔离性 多用户访问数据时 每一个用户都会开启一个事务 之前需要隔离
      隔离级别 低 ---> 高  隔离级别越低  执行效率越高
        1. read uncommitted 读未提交
          引发问题: 脏读
          解决: 提高隔离级别 read committed
```

2. read committed 读已提交
   引发问题：不可重复读
   解决：提高隔离级别 repeatable read

3. repeatable read 可重复读
   引发问题：幻读
   解决：提高隔离级别Serializable

4. Serializable 串行化