MySql

一、安装及设置

- 端口号 3306
- 登录 mysql -uroot -p密码
- 怎么卸载mysql？（压缩包安装）
  卸载服务：先将mysql服务中止，进入cmd，输入sc delete mysql，进行删除删除MySql文件；
  删除环境：将环境变量中mysql路径删除
  可以将注册表中的MySQL删除
  重启电脑

1.1 DB、DBMS、SQL

- DB: DataBase（数据库，数据库实际上在硬盘上以文件的形式存在）
- DBMS: DataBase Management System（数据库管理系统，常见的有：MySQL Oracle DB2 Sybase SqlServer...）
- SQL: 结构化查询语言，是一门标准通用的语言。标准的sql适合于所有的数据库产品。
  SQL语句在执行的时候，实际上内部也会先进行编译，然后再执行sql。（sql语句的编译由DBMS完成。）
- DBMS负责执行sql语句，通过执行sql语句来操作DB当中的数据。DBMS -(执行)-> SQL -(操作)-> DB

1.2 表table

- 表：table是数据库的基本组成单元，
- 一个表包括行和列：行：被称为数据/记录(data) 列：被称为字段(column)
- 每一个字段应该包括哪些属性？字段名、数据类型、相关的约束/字段长度。

1.3 SQL分类

- DQL（数据查询语言）：查询语句，凡是select语句都是DQL。----Data Query Language
- DML（数据操作语言）：insert delete update，对表当中的数据进行增删改。---Data Manipulation language
- DDL（数据定义语言）：create drop alter，对表结构的增删改。---Data Definition Language
- TCL（事务控制语言）：commit提交事务，rollback回滚事务。----Transactional Control Language

- DCL（数据控制语言）：grant授权、revoke撤销权限等。----Data Control Language

1.4 导入数据

- 第一步：登录mysql数据库管理系统 dos命令窗口：mysql -uroot -p密码
- 第二步：查看有哪些数据库show databases;(这个不是SQL语句，属于MySQL的命令。)
- 第三步：创建属于我们自己的数据库---create database bjpowernode; (这个不是SQL语句，是MySQL的命令。)
- 第四步：使用bjpowernode数据---use bjpowernode; (这个不是SQL语句，属于MySQL的命令。)
- 第五步：查看当前使用的数据库中有哪些表----show tables; (这个不是SQL语句，属于MySQL的命令。)
- 第六步：初始化数据----mysql> source D:\Program\mysql\mysql-8.0.25-winx64\sourse\bjpowernode.sql
  将文件直接拖进去即可
- bjpowernode.sql，这个文件以sql结尾，这样的文件被称为“sql脚本”。什么是sql脚本呢？当一个文件的扩展名是.sql，并且该文件中编写了大量的sql语句，我们称这样的文件为sql脚本。
- 注意：直接使用source命令可以执行sql脚本。sql脚本中的数据量太大的时候，无法打开，请使用source命令完成初始化。
- 删除数据库：drop database bjpowernode;
- 查看表结构：desc 表名;

```
mysql> show tables ;
    +-----------------------+
    | Tables_in_bjpowernode |
    +-----------------------+
    | dept                  |   (部门表)
    | emp                   |   (员工表)
    | salgrade              |   (工资等级表)
    +-----------------------+
查看表结构：desc 表名;
    mysql> desc dept;
    +--------+-------------+------+-----+---------+-------+
    | Field  | Type        | Null | Key | Default | Extra |
    +--------+-------------+------+-----+---------+-------+
    | DEPTNO | int(2)      | NO   | PRI | NULL    |       |      部门编号
    | DNAME  | varchar(14) | YES  |     | NULL    |       |      部门名称
    | LOC    | varchar(13) | YES  |     | NULL    |       |      部门位置
    +--------+-------------+------+-----+---------+-------+

```

```
    mysql> desc emp;
    +----------+-------------+------+-----+---------+-------+
    | Field    | Type        | Null | Key | Default | Extra |
    +----------+-------------+------+-----+---------+-------+
    | EMPNO    | int(4)      | NO   | PRI | NULL    |       |   员工编号
    | ENAME    | varchar(10) | YES  |     | NULL    |       |   员工姓名
    | JOB      | varchar(9)  | YES  |     | NULL    |       |   工作岗位
    | MGR      | int(4)      | YES  |     | NULL    |       |   上级领导编号
    | HIREDATE | date        | YES  |     | NULL    |       |   入职日期
    | SAL      | double(7,2) | YES  |     | NULL    |       |   月薪
    | COMM     | double(7,2) | YES  |     | NULL    |       |   补助/津贴
    | DEPTNO   | int(2)      | YES  |     | NULL    |       |   部门编号
    +----------+-------------+------+-----+---------+-------+

    mysql> desc salgrade;
    +-------+---------+------+-----+---------+-------+
    | Field | Type    | Null | Key | Default | Extra |
    +-------+---------+------+-----+---------+-------+
    | GRADE | int(11) | YES  |     | NULL    |       |      等级
    | LOSAL | int(11) | YES  |     | NULL    |       |      最低薪资
    | HISAL | int(11) | YES  |     | NULL    |       |      最高薪资
    +-------+---------+------+-----+---------+-------+
 表中的数据
mysql> select * from emp;
+-------+--------+-----------+------+------------+---------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   | SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+---------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |  800.00 |    NULL |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 | 1600.00 |  300.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 | 1250.00 |  500.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 | 2975.00 |    NULL |     20 |
```

```
|  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 | 1250.00 |
1400.00 |     30 |
|  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 | 2850.00 |
 NULL |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 | 2450.00 |
 NULL |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 | 3000.00 |
 NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 1981-11-17 | 5000.00 |
 NULL |     10 |
|  7844 | TURNER | SALESMAN  | 7698 | 1981-09-08 | 1500.00 |
 0.00 |     30 |
|  7876 | ADAMS  | CLERK     | 7788 | 1987-05-23 | 1100.00 |
 NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 1981-12-03 |  950.00 |
 NULL |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 | 3000.00 |
 NULL |     20 |
|  7934 | MILLER | CLERK     | 7782 | 1982-01-23 | 1300.00 |
 NULL |     10 |
+-------+--------+-----------+------+------------+---------+------
---+--------+

mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+

mysql> select * from salgrade;
+-------+-------+-------+
| GRADE | LOSAL | HISAL |
+-------+-------+-------+
|     1 |   700 |  1200 |
|     2 |  1201 |  1400 |
|     3 |  1401 |  2000 |
|     4 |  2001 |  3000 |
|     5 |  3001 |  9999 |
+-------+-------+-------+
```

二、常见命令

- 查看mysql版本----mysql -V　　select version();
- 查看当前使用的是哪个数据库-----select database();

- 结束一条语句----\c 命令,
- 退出mysql------exit 或者\q
- 查看其他库中的表----show tables from 库名;
- 查看创建表的语句----show create table emp;

```
mysql> select database(); 查看当前使用的是哪个数据库
+-------------+
| database()  |
+-------------+
| bjpowernode |
+-------------+

mysql> select version(); 查看mysql的版本号。
+-----------+
| version() |
+-----------+
| 5.5.36    |
+-----------+
```

三、查询语句(DQL)

- 字符串必须添加单引号或双引号，最好用单引号

```
总结一个完整的DQL语句怎么写？

    select          5
        ..
    from            1
        ..
    where           2
        ..
    group by        3
        ..
    having          4(就是为了过滤分组后的数据而存在的,不可以单独的出现,只能和group by连用)
        ..
    order by        6
        ..
    limit           7
        ...;
    原则：能在 where 中过滤的数据，尽量在 where 中过滤，效率较高。having 的过滤是专门 对分组之后的数据进行过滤的。

    select e.ENAME as '姓名',e.HIREDATE as '入职日期',e.SAL AS '月薪',s.GRADE '等级',d.DNAME as '部门名称',d.LOC as '部门位置'
    from emp e
    join
        dept d
```

```
    on e.DEPTNO=d.DEPTNO
    join
        salgrade s
    on e.SAL between s.LOSAL and s.HISAL
    order by '入职日期';
```

3.1 简单查询

- 语法格式：select 字段名1,字段名2,字段名3,.... from 表名;
  提示：1、任何一条sql语句以“;”结尾。
  2、sql语句不区分大小写。
- 查询员工的年薪？（字段可以参与数学运算。）
- 给查询结果的列重命名----select ename,sal * 12 as yearsal from emp;
- 别名中有中文----select ename,sal * 12 as '年薪' from emp;
  字符串最好使用单引号
- as关键字可以省略----select empno,ename,sal * 12 yearsal from emp;
- 查询所有字段-----select * from emp; // 实际开发中不建议使用*，效率较低。

```
查询员工的年薪？（字段可以参与数学运算。）
        select ename,sal * 12 from emp;
        +--------+----------+
        | ename  | sal * 12 |
        +--------+----------+
        | SMITH  |  9600.00 |
        | ALLEN  | 19200.00 |
        | WARD   | 15000.00 |
        | JONES  | 35700.00 |
        | MARTIN | 15000.00 |
        | BLAKE  | 34200.00 |
        | CLARK  | 29400.00 |
        | SCOTT  | 36000.00 |
        | KING   | 60000.00 |
        | TURNER | 18000.00 |
        | ADAMS  | 13200.00 |
        | JAMES  | 11400.00 |
        | FORD   | 36000.00 |
        | MILLER | 15600.00 |
        +--------+----------+

    给查询结果的列重命名？
        select ename,sal * 12 as yearsal from emp;

    别名中有中文？
        select ename,sal * 12 as 年薪 from emp; // 错误
        select ename,sal * 12 as '年薪' from emp;
        +--------+----------+
        | ename  | 年薪       |
        +--------+----------+
```

```
        | SMITH  |  9600.00 |
        | ALLEN  | 19200.00 |
        | WARD   | 15000.00 |
        | JONES  | 35700.00 |
        | MARTIN | 15000.00 |
        | BLAKE  | 34200.00 |
        | CLARK  | 29400.00 |
        | SCOTT  | 36000.00 |
        | KING   | 60000.00 |
        | TURNER | 18000.00 |
        | ADAMS  | 13200.00 |
        | JAMES  | 11400.00 |
        | FORD   | 36000.00 |
        | MILLER | 15600.00 |
        +--------+----------+

    注意：标准sql语句中要求字符串使用单引号括起来。虽然mysql支持双引号，尽量别用。

    as关键字可以省略？
        mysql> select empno,ename,sal * 12 yearsal from emp;
        +-------+--------+----------+
        | empno | ename  | yearsal  |
        +-------+--------+----------+
        |  7369 | SMITH  |  9600.00 |
        |  7499 | ALLEN  | 19200.00 |
        |  7521 | WARD   | 15000.00 |
        |  7566 | JONES  | 35700.00 |
        |  7654 | MARTIN | 15000.00 |
        |  7698 | BLAKE  | 34200.00 |
        |  7782 | CLARK  | 29400.00 |
        |  7788 | SCOTT  | 36000.00 |
        |  7839 | KING   | 60000.00 |
        |  7844 | TURNER | 18000.00 |
        |  7876 | ADAMS  | 13200.00 |
        |  7900 | JAMES  | 11400.00 |
        |  7902 | FORD   | 36000.00 |
        |  7934 | MILLER | 15600.00 |
        +-------+--------+----------+

    查询所有字段？
        select * from emp; // 实际开发中不建议使用*，效率较低。
```

3.2 条件查询

- 需要用where语句，where必须放在from语句表后面

- 语法格式：
  select 字段, 字段... from 表名 where 条件;
  执行顺序：先from，然后where，最后select
- 不等于：<> 或 !=
  两个值之间：between...and..（闭区间，必须左小右大，等同于>=and<=）
  并且： and
  或者：or
- 在数据库当中NULL不是一个值，代表什么也没有，为空。空不是一个值，不能用等号衡量。必须使用 is null或者is not null
- in等同于or，包含，相当于多个or，注意in后面不是区间，表示确定值
  找出工作岗位是MANAGER和SALESMAN的员工

  select ename,job from emp where job = 'SALESMAN' or job = 'MANAGER';
  select ename,job from emp where job in('SALESMAN', 'MANAGER');
- 模糊查询like： 在模糊查询当中，必须掌握两个特殊的符号，一个是%，一个是_
  %代表任意多个字符，_代表任意1个字符。

```
例如：找出名字中第二个字母是A的：select ename from emp where ename like '_A%';
找出名字中有下划线的:select name from t_user where name like '%\_%';
```

```
    查询SMITH的工资？
        select sal from emp where ename = 'SMITH'; // 字符串使用单引号括起来。
        +--------+
        | sal    |
        +--------+
        | 800.00 |
        +--------
    找出工资不等于3000的？
        select ename,sal from emp where sal <> 3000;
        select ename,sal from emp where sal != 3000;

    找出工资在1100和3000之间的员工，包括1100和3000？
        select ename,sal from emp where sal >= 1100 and sal <= 3000;

        select ename,sal from emp where sal between 1100 and 3000; // between...and...是闭区间 [1100 ~ 3000]

        select ename,sal from emp where sal between 3000 and 1100; // 查询不到任何数据

        between and除了可以使用在数字方面之外，还可以使用在字符串方面。
        select ename from emp where ename between 'A' and 'C';
```

```
        +-------+
        | ename |
        +-------+
        | ALLEN |
        | BLAKE |
        | ADAMS |
        +-------+
        select ename from emp where ename between 'A' and 'D'; //左闭右开。

        找出哪些人津贴为NULL?
            select ename,sal,comm from emp where comm is null;
            +--------+---------+------+
            | ename  | sal     | comm |
            +--------+---------+------+
            | SMITH  |  800.00 | NULL |
            | JONES  | 2975.00 | NULL |
            | BLAKE  | 2850.00 | NULL |
            | CLARK  | 2450.00 | NULL |
            | SCOTT  | 3000.00 | NULL |
            | KING   | 5000.00 | NULL |
            | ADAMS  | 1100.00 | NULL |
            | JAMES  |  950.00 | NULL |
            | FORD   | 3000.00 | NULL |
            | MILLER | 1300.00 | NULL |
            +--------+---------+------+
            select ename,sal,comm from emp where comm = null; //不能比较
            Empty set (0.00 sec)

        找出哪些人没有津贴?
            select ename,sal,comm from emp where comm is null or comm = 0;
            +--------+---------+------+
            | ename  | sal     | comm |
            +--------+---------+------+
            | SMITH  |  800.00 | NULL |
            | JONES  | 2975.00 | NULL |
            | BLAKE  | 2850.00 | NULL |
            | CLARK  | 2450.00 | NULL |
            | SCOTT  | 3000.00 | NULL |
            | KING   | 5000.00 | NULL |
            | TURNER | 1500.00 | 0.00 |
            | ADAMS  | 1100.00 | NULL |
            | JAMES  |  950.00 | NULL |
            | FORD   | 3000.00 | NULL |
            | MILLER | 1300.00 | NULL |
            +--------+---------+------+
```

找出工作岗位是MANAGER和SALESMAN的员工？

```
select ename,job from emp where job = 'MANAGER' or job = 'SALESMAN';
+--------+----------+
| ename  | job      |
+--------+----------+
| ALLEN  | SALESMAN |
| WARD   | SALESMAN |
| JONES  | MANAGER  |
| MARTIN | SALESMAN |
| BLAKE  | MANAGER  |
| CLARK  | MANAGER  |
| TURNER | SALESMAN |
+--------+----------+
```

and和or联合起来用：找出薪资大于1000的并且部门编号是20或30部门的员工。

```
select ename,sal,deptno from emp where sal > 1000 and deptno = 20 or deptno = 30; // 错误的
select ename,sal,deptno from emp where sal > 1000 and (deptno = 20 or deptno = 30); // 正确的。
```

注意：当运算符的优先级不确定的时候加小括号。

in等同于or：找出工作岗位是MANAGER和SALESMAN的员工？

```
select ename,job from emp where job = 'SALESMAN' or job = 'MANAGER';
select ename,job from emp where job in('SALESMAN', 'MANAGER');

select ename,job from emp where sal in(800, 5000); // in后面的值不是区间，是具体的值。
+-------+-----------+
| ename | job       |
+-------+-----------+
| SMITH | CLERK     |
| KING  | PRESIDENT |
+-------+-----------+
```

not in: 不在这几个值当中。

```
select ename,job from emp where sal not in(800, 5000);
```

模糊查询like ？（在模糊查询当中，必须掌握两个特殊的符号，一个是%，一个是_）%代表任意多个字符，_代表任意1个字符。

找出名字中第二个字母是A的？

```
select ename from emp where ename like '_A%';
+--------+
| ename  |
```

```
                +--------+
                | WARD   |
                | MARTIN |
                | JAMES  |
                +--------+
            找出名字中有下划线的？
                select name from t_user where name like '%\_%';
                +---------+
                | name    |
                +---------+
                | WANG_WU |
                +---------+

            找出名字中最后一个字母是T的？
                select ename from emp where ename like '%T';
                +-------+
                | ename |
                +-------+
                | SCOTT |
                +-------+
```

3.3 排序

- 排序采用 order by 子句，order by 后面跟上排序字段，排序字段可以放多个，多个采用逗号 间隔，
- order by 默认采用升序,asc表示升序，desc表示降序，如果存在 where 子句那么 order by 必须放到 where 语句的后 面
- select ename , sal from emp order by sal; // 升序
  select ename , sal from emp order by sal asc; // 升序
  select ename , sal from emp order by sal desc; // 降序。
- 多个字段排序，如果根据第一个字段排序重复了，会根据第二个字段排序。越靠前的字段越能起到主导作用。
- 注：先执行from再执行where再执行select,order by是最后执行的。

```
按照工资的降序排列，当工资相同的时候再按照名字的升序排列。
    select ename,sal from emp order by sal desc , ename asc;

找出工作岗位是SALESMAN的员工，并且要求按照薪资的降序排列。
    select ename,job,sal from emp where job = 'SALESMAN'order by
sal desc;

    注：先执行from再执行where再执行select,order by是最后执行的。
```

3.4 分组函数

- count 计数
  sum 求和
  avg 平均值
  max 最大值
  min 最小值
- 所有的分组函数都是对“某一组”数据进行操作的。
- 分组函数还有另一个名字：多行处理函数。多行处理函数的特点：输入多行，最终输出的结果是1行。
- 重点：分组函数自动忽略NULL，对某组数据进行计数时，是不算null在内的。
  例如：select sum(comm) from emp where comm is not null; // 不需要额外添加这个过滤条件。sum函数自动忽略NULL
- 重点：所有数据库都是这样规定的，只要有NULL参与的运算结果一定是NULL。
- 找出工资高于平均工资的员工:select ename,sal from emp where sal > avg(sal); // 报错
  原因：SQL语句当中有一个语法规则，分组函数不可直接使用在where子句当中。因为group by是在where执行之后才会执行的,而分组函数是在group by之后执行。

```
    找出工资总和？
        select sum(sal) from emp;
    找出最高工资？
        select max(sal) from emp;
    找出最低工资？
        select min(sal) from emp;
    找出平均工资？
        select avg(sal) from emp;
    找出总人数？
        select count(*) from emp;
        select count(ename) from emp;


        select sum(comm) from emp;
        +-----------+
        | sum(comm) |
        +-----------+
        |   2200.00 |
        +-----------+
        select sum(comm) from emp where comm is not null; // 不需要额外添加这个过滤条件。sum函数自动忽略NULL。

        找出工资高于平均工资的员工？
            select ename,sal from emp where sal > avg(sal);
//ERROR 1111 (HY000): Invalid use of group function
            思考以上的错误信息：无效的使用了分组函数？
            原因：SQL语句当中有一个语法规则，分组函数不可直接使用在where子句当中。why????
            因为group by是在where执行之后才会执行的。
```

```
                select          5
                    ..
                from            1
                    ..
                where           2
                    ..
                group by        3
                    ..
                having          4
                    ..
                order by        6
                    ..

    找出工资高于平均工资的员工?
        第一步：找出平均工资
            select avg(sal) from emp;
            +-------------+
            | avg(sal)    |
            +-------------+
            | 2073.214286 |
            +-------------+
        第二步：找出高于平均工资的员工
            select ename,sal from emp where sal > 2073.214286;
            +-------+---------+
            | ename | sal     |
            +-------+---------+
            | JONES | 2975.00 |
            | BLAKE | 2850.00 |
            | CLARK | 2450.00 |
            | SCOTT | 3000.00 |
            | KING  | 5000.00 |
            | FORD  | 3000.00 |
            +-------+---------+

            select ename,sal from emp where sal > (select avg(sal)
from emp);
```

- count(*)和count(具体的某个字段)，他们有什么区别?
- count(*):不是统计某个字段中数据的个数，而是统计总记录条数。（和某个字段无关）
- count(comm): 表示统计comm字段中不为NULL的数据总数量。

```
联合使用：
        select count(*),sum(sal),avg(sal),max(sal),min(sal) from
emp;

        +---------+---------+-------------+---------+---------+
        | count(*) | sum(sal) | avg(sal)    | max(sal) | min(sal) |
        +---------+---------+-------------+---------+---------+
        |       14 | 29025.00 | 2073.214286 |  5000.00 |   800.00 |
        +---------+---------+-------------+---------+---------+
```

3.5 单行处理函数

- 单行处理函数:输入一行，输出一行。
- 重点：所有数据库都是这样规定的，只要有NULL参与的运算结果一定是NULL。
- ifnull------空处理函数:
  ifnull(可能为NULL的数据,被当做什么处理)： 属于单行处理函数。

```
    计算每个员工的年薪？
        select ename,(sal+comm)*12 as yearsal from emp; //null
        使用
        select ename,(sal+ifnull(comm,0))*12 as yearsal from emp;

        select ename,ifnull(comm,0) as comm from emp;
        +--------+---------+
        | ename  | comm    |
        +--------+---------+
        | SMITH  |    0.00 |
        | ALLEN  |  300.00 |
        | WARD   |  500.00 |
        | JONES  |    0.00 |
        | MARTIN | 1400.00 |
        | BLAKE  |    0.00 |
        | CLARK  |    0.00 |
        | SCOTT  |    0.00 |
        | KING   |    0.00 |
        | TURNER |    0.00 |
        | ADAMS  |    0.00 |
        | JAMES  |    0.00 |
        | FORD   |    0.00 |
        | MILLER |    0.00 |
        +--------+---------+
```

3.6 group by和having

- group by： 按照某个字段或者某些字段进行分组。
- having : having是对分组之后的数据进行再次过滤。

- 注意：分组函数一般都会和group by联合使用，这也是为什么它被称为分组函数的原因。
- 并且任何一个分组函数（count sum avg max min）都是在group by语句执行结束之后才会执行的。
- 当一条sql语句没有group by的话，整张表的数据会自成一组。
- 记住一个规则：当一条语句中有group by的话，select后面只能跟分组函数和参与分组的字段。
- 例如：找出每个部门不同工作岗位的最高薪资：
  select deptno,job,max(sal) from emp group by deptno,job;
- 例如：找出每个部门的平均薪资，要求显示薪资大于2000的数据
  select deptno,avg(sal) from emp group by deptno having avg(sal) > 2000;
- 注：优先使用where对数据进行过滤，无法使用where再考虑用having，并且having只能用在group by后面

```
案例：找出每个工作岗位的最高薪资。
    select max(sal),job from emp group by job;

    +---------+-----------+
    | max(sal) | job       |
    +---------+-----------+
    |  3000.00 | ANALYST   |
    |  1300.00 | CLERK     |
    |  2975.00 | MANAGER   |
    |  5000.00 | PRESIDENT |
    |  1600.00 | SALESMAN  |
    +---------+-----------+

    select ename,max(sal),job from emp group by job;
    以上在mysql当中，查询结果是有的，但是结果没有意义，在oracle数据库当中会报错。语法错误。
因为ename没有参与分组，无法查询
    记住一个规则：当一条语句中有group by的话，select后面只能跟分组函数和参与分组的字段。

    找出每个部门的最高薪资，要求显示薪资大于2900的数据。

        第一步：找出每个部门的最高薪资
        select max(sal),deptno from emp group by deptno;
        +---------+--------+
        | max(sal) | deptno |
        +---------+--------+
        |  5000.00 |     10 |
        |  3000.00 |     20 |
        |  2850.00 |     30 |
        +---------+--------+

        第二步：找出薪资大于2900
```

```
        select max(sal),deptno from emp group by deptno having max(sal) > 2900; // 这种方式效率低。
        +----------+--------+
        | max(sal) | deptno |
        +----------+--------+
        |  5000.00 |     10 |
        |  3000.00 |     20 |
        +----------+--------+

        select max(sal),deptno from emp where sal > 2900 group by deptno;  // 效率较高，建议能够使用where过滤的尽量使用where。
        +----------+--------+
        | max(sal) | deptno |
        +----------+--------+
        |  5000.00 |     10 |
        |  3000.00 |     20 |
        +----------+--------+

    找出每个部门的平均薪资，要求显示薪资大于2000的数据。

    第一步：找出每个部门的平均薪资
    select deptno,avg(sal) from emp group by deptno;
    +--------+-------------+
    | deptno | avg(sal)    |
    +--------+-------------+
    |     10 | 2916.666667 |
    |     20 | 2175.000000 |
    |     30 | 1566.666667 |
    +--------+-------------+

    第二步：要求显示薪资大于2000的数据
    select deptno,avg(sal) from emp group by deptno having avg(sal) > 2000;
    +--------+-------------+
    | deptno | avg(sal)    |
    +--------+-------------+
    |     10 | 2916.666667 |
    |     20 | 2175.000000 |
    +--------+-------------+

    where后面不能使用分组函数：
        select deptno,avg(sal) from emp where avg(sal) > 2000 group by deptno;  // 错误了。
        这种情况只能使用having过滤。
```

3.7 去重

- distinct关键字去除重复记录。
- 记住：distinct只能出现在所有字段的最前面。

- select distinct deptno,job from emp; -----这个表示deptno,job这两个字段联合去重。

```
mysql> select ename,distinct job from emp;
以上的sql语句是错误的。
记住：distinct只能出现在所有字段的最前面。

mysql> select distinct deptno,job from emp;
+--------+-----------+
| deptno | job       |
+--------+-----------+
|     20 | CLERK     |
|     30 | SALESMAN  |
|     20 | MANAGER   |
|     30 | MANAGER   |
|     10 | MANAGER   |
|     20 | ANALYST   |
|     10 | PRESIDENT |
|     30 | CLERK     |
|     10 | CLERK     |
+--------+-----------+

案例：统计岗位的数量？
select count(distinct job) from emp;
+----------------------+
| count(distinct job) |
+----------------------+
|                    5 |
+----------------------+
```

3.8 连接查询

- 连接查询的分类:
  根据语法出现的年代来划分的话，包括：
  SQL92（一些老的DBA可能还在使用这种语法。DBA：DataBase Administrator，数据库管理员）
  SQL99（比较新的语法）
- 根据表的连接方式来划分，包括：
  - 内连接：
    - 等值连接
    - 非等值连接
    - 自连接
  - 外连接：
    - 左外连接（左连接）
    - 右外连接（右连接）
  - 全连接（这个不讲，很少用！）

- 笛卡尔积现象：当两张表进行连接查询的时候，没有任何条件进行限制，最终的查询结果条数是两张表记录条数的乘积。避免笛卡尔积现象？-----加条件进行过滤。
  注：避免了笛卡尔积现象，会减少记录的匹配次数吗？---不会，次数还是56次。只不过显示的是有效记录。

```
在表的连接查询方面有一种现象被称为：笛卡尔积现象（笛卡尔乘积现象）

案例：找出每一个员工的部门名称，要求显示员工名和部门名。（笛卡尔乘积现象）
select e.ename,d.dname from emp e,dept d;

案例：找出每一个员工的部门名称，要求显示员工名和部门名。
select  e.ename,d.dname from emp e , dept d where e.deptno = d.deptno; //SQL92，以后不用。
```

3.8.1 内连接

- 假设A和B表进行连接，使用内连接的话，凡是A表和B表能够匹配上的记录查询出来，这就是内连接。AB两张表没有主副之分，两张表是平等的。AB匹配不上的就不查了
- 内连接之等值连接：最大特点是：条件是等量关系。
- 语法：select 字段 from 表名 A inner join 表名B on 连接条件 where ...
- inner可以省略的，带着inner目的是可读性好一些。

```
案例：查询每个员工的部门名称，要求显示员工名和部门名。
SQL99：
    select e.ename,d.dname from emp e join dept d on e.deptno = d.deptno;

    // inner可以省略的，带着inner目的是可读性好一些。
    select
        e.ename,d.dname
    from
        emp e
    inner join
        dept d
    on
        e.deptno = d.deptno;
```

- 内连接之非等值连接：最大的特点是：连接条件中的关系是非等量关系。

```
案例：找出每个员工的工资等级，要求显示员工名、工资、工资等级。
mysql> select ename,sal from emp; e
+--------+---------+
| ename  | sal     |
+--------+---------+
| SMITH  |  800.00 |
```

```
| ALLEN  | 1600.00 |
| WARD   | 1250.00 |
| JONES  | 2975.00 |
| MARTIN | 1250.00 |
| BLAKE  | 2850.00 |
| CLARK  | 2450.00 |
| SCOTT  | 3000.00 |
| KING   | 5000.00 |
| TURNER | 1500.00 |
| ADAMS  | 1100.00 |
| JAMES  |  950.00 |
| FORD   | 3000.00 |
| MILLER | 1300.00 |
+--------+---------+

mysql> select * from salgrade; s
+-------+-------+-------+
| GRADE | LOSAL | HISAL |
+-------+-------+-------+
|     1 |   700 |  1200 |
|     2 |  1201 |  1400 |
|     3 |  1401 |  2000 |
|     4 |  2001 |  3000 |
|     5 |  3001 |  9999 |
+-------+-------+-------+

select
    e.ename,e.sal,s.grade
from
    emp e
join
    salgrade s
on
    e.sal between s.losal and s.hisal;

// inner可以省略
select
    e.ename,e.sal,s.grade
from
    emp e
inner join
    salgrade s
on
    e.sal between s.losal and s.hisal;
+--------+---------+-------+
| ename  | sal     | grade |
+--------+---------+-------+
| SMITH  |  800.00 |     1 |
| ALLEN  | 1600.00 |     3 |
```

```
| WARD   | 1250.00 |      2 |
| JONES  | 2975.00 |      4 |
| MARTIN | 1250.00 |      2 |
| BLAKE  | 2850.00 |      4 |
| CLARK  | 2450.00 |      4 |
| SCOTT  | 3000.00 |      4 |
| KING   | 5000.00 |      5 |
| TURNER | 1500.00 |      3 |
| ADAMS  | 1100.00 |      1 |
| JAMES  |  950.00 |      1 |
| FORD   | 3000.00 |      4 |
| MILLER | 1300.00 |      2 |
+--------+---------+--------+
```

- 自连接：最大的特点是：一张表看做两张表。自己连接自己。

```
案例：找出每个员工的上级领导，要求显示员工名和对应的领导名。

员工的领导编号 = 领导的员工编号

select
    a.ename as '员工名',b.ename as '领导名'
from
    emp a
inner join
    emp b
on
    a.mgr = b.empno;
```

3.8.2 外连接

- 外连接：假设A和B表进行连接，使用外连接的话，AB两张表中有一张表是主表，一张表是副表，主要查询主表中的数据，捎带着查询副表，当副表中的数据没有和主表中的数据匹配上，副表自动模拟出NULL与之匹配。
- 内连接的话A表与B表匹配不上就不查了。
- 外连接的分类：左外连接（左连接）：表示左边的这张表是主表。
  右外连接（右连接）：表示右边的这张表是主表。
- 左连接有右连接的写法，右连接也会有对应的左连接的写法。
- 外连接最重要的特点是：主表的数据无条件的全部查询出来。

```
案例：找出每个员工的上级领导？（所有员工必须全部查询出来。）
emp a 员工表
+-------+--------+------+
| empno | ename  | mgr  |
+-------+--------+------+
|  7369 | SMITH  | 7902 |
|  7499 | ALLEN  | 7698 |
```

```
|  7521 | WARD   | 7698 |
|  7566 | JONES  | 7839 |
|  7654 | MARTIN | 7698 |
|  7698 | BLAKE  | 7839 |
|  7782 | CLARK  | 7839 |
|  7788 | SCOTT  | 7566 |
|  7839 | KING   | NULL |
|  7844 | TURNER | 7698 |
|  7876 | ADAMS  | 7788 |
|  7900 | JAMES  | 7698 |
|  7902 | FORD   | 7566 |
|  7934 | MILLER | 7782 |
+-------+--------+------+

使用内连接，最高领导查不出来

外连接：（左外连接/左连接）
select
    a.ename '员工', b.ename '领导'
from
    emp a
left join
    emp b
on
    a.mgr = b.empno;

// outer是可以省略的。
select
    a.ename '员工', b.ename '领导'
from
    emp a
left outer join
    emp b
on
    a.mgr = b.empno;

外连接：（右外连接/右连接）
select
    a.ename '员工', b.ename '领导'
from
    emp b
right join
    emp a
on
    a.mgr = b.empno;


案例：找出哪个部门没有员工？
select
```

```
    d.*
from
    emp e
right join
    dept d
on
    e.deptno = d.deptno
where
    e.empno is null;

找出每一个员工的部门名称、工资等级、以及上级领导。
    select
        e.ename '员工',d.dname,s.grade,e1.ename '领导'
    from
        emp e
    join
        dept d
    on
        e.deptno = d.deptno
    join
        salgrade s
    on
        e.sal between s.losal and s.hisal
    left join
        emp e1
    on
        e.mgr = e1.empno;
```

3.9 子查询

- select语句当中嵌套select语句，被嵌套的select语句是子查询。
- 子查询可以出现在哪里？
  select
  ..(select).
  from
  ..(select).
  where
  ..(select).
- where子句中使用子查询

```
案例：找出高于平均薪资的员工信息。
select * from emp where sal > avg(sal); //错误的写法，where后面不能直接使用分组函数。
正确写法：
    select * from emp where sal > (select avg(sal) from emp);
```

- from后面嵌套子查询

```
案例：找出每个部门平均薪水的等级。

第一步：找出每个部门平均薪水（按照部门编号分组，求sal的平均值）
select deptno,avg(sal) as avgsal from emp group by deptno;
+--------+-------------+
| deptno | avgsal      |
+--------+-------------+
|     10 | 2916.666667 |
|     20 | 2175.000000 |
|     30 | 1566.666667 |
+--------+-------------+
第二步：将以上的查询结果当做临时表t，让t表和salgrade s表连接，条件是：t.avgsal between s.losal and s.hisal

select
    t.*,s.grade
from
    (select deptno,avg(sal) as avgsal from emp group by deptno) t
join
    salgrade s
on
    t.avgsal between s.losal and s.hisal;
+--------+-------------+-------+
| deptno | avgsal      | grade |
+--------+-------------+-------+
|     30 | 1566.666667 |     3 |
|     10 | 2916.666667 |     4 |
|     20 | 2175.000000 |     4 |
+--------+-------------+-------+
```

- 在select后面嵌套子查询。

```
案例：找出每个员工所在的部门名称，要求显示员工名和部门名。

select
    e.ename,d.dname
from
    emp e
join
    dept d
on
    e.deptno = d.deptno;
嵌套：
select
    e.ename,(select d.dname from dept d where e.deptno = d.deptno) as dname
from
    emp e;
```

3.10 union

- union :可以将查询结果集相加
- 两张不相干的表中的数据拼接在一起显示
  select ename from emp
  union
  select dname from dept;
- 注意：两个查询必须列数一致才能拼接

```
案例：找出工作岗位是SALESMAN和MANAGER的员工？
第一种：select ename,job from emp where job = 'MANAGER' or job = 'SALESMAN';
第二种：select ename,job from emp where job in('MANAGER','SALESMAN');
第三种：union
select ename,job from emp where job = 'MANAGER'
union
select ename,job from emp where job = 'SALESMAN';
+--------+----------+
| ename  | job      |
+--------+----------+
| JONES  | MANAGER  |
| BLAKE  | MANAGER  |
| CLARK  | MANAGER  |
| ALLEN  | SALESMAN |
| WARD   | SALESMAN |
| MARTIN | SALESMAN |
| TURNER | SALESMAN |
+--------+----------+
```

3.11 limit

- limit是mysql特有的，其他数据库中没有，不通用。（Oracle中有一个相同的机制，叫做rownum）
- 作用：limit取结果集中的部分数据
- 语法机制：limit startIndex, length
  startIndex表示起始位置，从0开始，0表示第一条数据，可以省略。
  length表示取几个
- limit是sql语句最后执行的一个环节

```
案例：找出工资排名在第4到第9名的员工？
    select ename,sal from emp order by sal desc limit 3,6;
```

- 通用的标准分页sql
- 每页显示pageSize条记录：
  第pageNo页：(pageNo - 1) * pageSize, pageSize

- pageSize-----是每页显示多少条记录　pageNo-----显示第几页

四、表

4.1 创建表

- create table 表名(
  　字段名1 数据类型,
  　字段名2 数据类型,
  　....
  );
- 创建表的时候，表中有字段，每一个字段有：* 字段名 * 字段数据类型 * 字段长度限制 * 字段约束
- 表名在数据库当中一般建议以：t_ 或者 tbl_开始。
- char和varchar怎么选择?
  在实际的开发中，当某个字段中的数据长度不发生改变的时候，是定长的，例如：性别、生日等都是采用char。当一个字段的数据长度不确定，例如：简介、姓名等都是采用varchar。
- 删除表：drop table if exists t_student; // 当这个表存在的话删除。

```
关于MySQL当中字段的数据类型？以下只说常见的
        int             整数型(java中的int)
        bigint          长整型(java中的long)
        float           浮点型(java中的float double)
        char            定长字符串(String)
        varchar         可变长字符串(StringBuffer/StringBuilder)----最大存255个字符
        date            日期类型 （对应Java中的java.sql.Date类型）
        BLOB            二进制大对象（存储图片、视频等流媒体信息） Binary Large OBject （对应java中的Object）
        CLOB            字符大对象（存储较大文本，比如，可以存储4G的字符串。） Character Large OBject（对应java中的Object）
        ......
    创建学生表：
        学生信息包括：
            学号、姓名、性别、班级编号、生日
            学号：bigint
            姓名：varchar
            性别：char
            班级编号：int
            生日：char

        create table t_student(
            no bigint,
            name varchar(255),
            sex char(1) default 1,    ----指定默认值
            classno varchar(255),
```

```
            birth char(10)
        );

```

4.2 插入数据insert

- 语法格式：insert into 表名(字段名1,字段名2,字段名3,....) values(值1,值2,值3,....)
- 要求：字段的数量和值的数量相同，并且数据类型要对应相同（前后对上就行）。
- 注意：当一条insert语句执行成功之后，表格当中必然会多一行记录。即使多的这一行记录当中某些字段是NULL，后期也没有办法在执行insert语句插入数据了，只能使用update进行更新。
- 字段可以省略不写，但是后面的value对数量和顺序都有要求。
- 可以一次插入多行数据；

```
    insert into t_student(no,name,sex,classno,birth) values(1,'zhangsan','1','gaosan1ban', '1950-10-12');

    insert into t_student(name) values('wangwu'); // 除name字段之外，剩下的所有字段自动插入NULL（除非指定默认值）。
    insert into t_student(no) values(3);
    mysql> select * from t_student;
    +------+----------+------+------------+------------+
    | no   | name     | sex  | classno    | birth      |
    +------+----------+------+------------+------------+
    |    1 | zhangsan | 1    | gaosan1ban | 1950-10-12 |
    | NULL | wangwu   | 1    | NULL       | NULL       |
    |    3 | NULL     | 1    | NULL       | NULL       |
    +------+----------+------+------------+------------+

    // 字段可以省略不写，但是后面的value对数量和顺序都有要求。
    insert into t_student values(1,'jack','0','gaosan2ban');
    ERROR 1136 (21S01): Column count doesn't match value count at row 1

    // 一次插入多行数据
    insert into t_student
        (no,name,sex,classno,birth)
    values
        (3,'rose','1','gaosi2ban','1952-12-14'),(4,'laotie','1','gaosi2ban','1955-12-14');
```

4.3 表的复制

- 语法：create table 表名 as select语句;--------将查询结果当做表创建出来。
- 将查询结果插入到一张表中： insert into dept1 select * from dept;
  对表结构有要求，列要对应上

4.4 修改数据update

- 语法：update 表名 set 字段名1=值1,字段名2=值2... where 条件;
- 注意：没有条件整张表数据全部更新。
  中间是逗号隔开

```
    案例：将部门10的LOC修改为SHANGHAI，将部门名称修改为RENSHIBU
    update dept1 set loc = 'SHANGHAI', dname = 'RENSHIBU' where
deptno = 10;
    +--------+------------+----------+
    | DEPTNO | DNAME      | LOC      |
    +--------+------------+----------+
    |     10 | RENSHIBU   | SHANGHAI |
    |     20 | RESEARCH   | DALLAS   |
    |     30 | SALES      | CHICAGO  |
    |     40 | OPERATIONS | BOSTON   |
    |     10 | RENSHIBU   | SHANGHAI |
    |     20 | RESEARCH   | DALLAS   |
    |     30 | SALES      | CHICAGO  |
    |     40 | OPERATIONS | BOSTON   |
    +--------+------------+----------+

    更新所有记录
        update dept1 set loc = 'x', dname = 'y';
        +--------+-------+------+
        | DEPTNO | DNAME | LOC  |
        +--------+-------+------+
        |     10 | y     | x    |
        |     20 | y     | x    |
        |     30 | y     | x    |
        |     40 | y     | x    |
        |     10 | y     | x    |
        |     20 | y     | x    |
        |     30 | y     | x    |
        |     40 | y     | x    |
        +--------+-------+------+
```

4.5 删除数据delete

- 语法：delete from 表名 where 条件;
- 注意：没有条件全部删除。
- delete语句，实际没有释放数据的物理空间，可以用回滚找回数据，但对于大数据量来说效率较慢
- 增删改查有一个术语：CRUD操作
  Create（增） Retrieve（检索） Update（修改） Delete（删除）

```
    删除10部门数据？
        delete from dept1 where deptno = 10;
    删除所有记录？
        delete from dept1;

    怎么删除大表中的数据？（重点）---效率较高
        truncate table 表名; // 表被截断，不可回滚。永久丢失。
    删除表？
        drop table 表名; // 这个通用。
        drop table if exists 表名; // oracle不支持这种写法。
```

4.6 表结构的修改

- 依赖工具
- create drop alter，对表结构的增删改

4.7 约束(Constraint)

- 在创建表的时候，可以给表的字段添加相应的约束，添加约束的目的是为了保证表中数据的合法性、有效性、完整性。
- 常见的约束：
- 非空约束(not null)：约束的字段不能为NULL
  唯一约束(unique)：约束的字段不能重复
  主键约束(primary key)：约束的字段既不能为NULL，也不能重复（简称PK)
  外键约束(foreign key)：...（简称FK)
  检查约束(check)：注意Oracle数据库有check约束，但是mysql没有，目前mysql不支持该约束。
- 非空约束 not null
- 注意：not null约束只有列级约束。没有表级约束。只能加到字段后面

```
    create table t_user(
        id int,
        username varchar(255) not null,
        password varchar(255)
    );
    insert into t_user(id,password) values(1,'123');
    ERROR 1364 (HY000): Field 'username' doesn't have a default value

    insert into t_user(id,username,password) values(1,'lisi','123');
```

- 唯一约束(unique)-----唯一约束修饰的字段具有唯一性，不能重复。但可以为NULL，多个NULL不算重复。
- username varchar(255) unique // 列级约束，列后面直接加

- unique(usercode,username) // 多个字段联合起来添加1个约束unique 【表级约束】

```
    案例：给某一列添加unique
        drop table if exists t_user;
        create table t_user(
            id int,
            username varchar(255) unique  // 列级约束
        );
        insert into t_user values(1,'zhangsan');
        insert into t_user values(2,'zhangsan');
        ERROR 1062 (23000): Duplicate entry 'zhangsan' for key 'username'

    * 案例：给两个列或者多个列添加unique
        drop table if exists t_user;
        create table t_user(
            id int,
            usercode varchar(255),
            username varchar(255),
            unique(usercode,username) // 多个字段联合起来添加1个约束 unique 【表级约束】
        );
```

- 主键约束(primary key)-----主键的特点：不能为NULL，也不能重复。
- 一张表的主键约束只能有1个。（必须记住）
- 主键约束 : primary key
  主键字段 : id字段添加primary key之后，id叫做主键字段
  主键值 : id字段中的每一个值都是主键值。
- 表的设计三范式中有要求，第一范式就要求任何一张表都应该有主键。
- 主键作用：主键值是这行记录在这张表当中的唯一标识。（就像一个人的身份证号码一样。）
- 主键的分类：
  根据主键字段的字段数量来划分：
  单一主键（推荐的，常用的。）
  复合主键(多个字段联合起来添加一个主键约束)（复合主键不建议使用，因为复合主键违背三范式。）
  根据主键性质来划分：
  自然主键：主键值最好就是一个和业务没有任何关系的自然数。（这种方式是推荐的）
  业务主键：主键值和系统的业务挂钩，例如：拿着银行卡的卡号做主键，拿着身份证号码作为主键。（不推荐用）
- mysql提供主键值自增：
  create table t_user(
  id int primary key auto_increment, // id字段自动维护一个自增的数字，从1开始，以1递增。

```
    username varchar(255)
);
```

```
怎么给一张表添加主键约束呢？
    drop table if exists t_user;
    create table t_user(
        id int primary key,    // 列级约束
        username varchar(255),
        email varchar(255)
    );

* 使用表级约束方式定义主键：
    drop table if exists t_user;
    create table t_user(
        id int,
        username varchar(255),
        primary key(id)
    );

    以下内容是演示以下复合主键，不需要掌握：
        drop table if exists t_user;
        create table t_user(
            id int,
            username varchar(255),
            password varchar(255),
            primary key(id,username)
        );
        insert .......

    提示:Oracle当中也提供了一个自增机制，叫做：序列（sequence）对象。
```

- 外键约束(foreign key)
- 关于外键约束的相关术语：
  外键约束: foreign key
  外键字段：添加有外键约束的字段
  外键值：外键字段中的每一个值。
- 先删子再删父，先创父再创子
- 语法：foreign key (需要添加外键约束的字段) references 父表名(父表中被关联的字段)
  例如：foreign key(classno) references t_class(cno)
- 注意：添加外键约束之后，该字段下的值必须是父表中的一部分，不能新加
- 外键值可以为NULL。
- 外键字段引用其他表的某个字段的时候，被引用的字段必须是主键吗？
  被引用的字段不一定是主键，但至少具有unique约束。

```
业务背景：
        请设计数据库表，用来维护学生和班级的信息？
```

```
        两张表（班级表和学生表）
            t_class 班级表
            cno(pk)      cname
            ------------------------------------------------------
            101      北京大兴区经济技术开发区亦庄二中高三1班
            102      北京大兴区经济技术开发区亦庄二中高三2班

            t_student 学生表
            sno(pk)      sname                classno(该字段添加外键约束fk)
            ------------------------------------------------------
            1                zs1              101
            2                zs2              101
            3                zs3              102
            4                zs4              102
            5                zs5              102

    * 将以上表的建表语句写出来：
        t_student中的classno字段引用t_class表中的cno字段，此时t_student表叫做子表。t_class表叫做父表。

        顺序要求：
            删除数据的时候，先删除子表，再删除父表。
            添加数据的时候，先添加父表，在添加子表。
            创建表的时候，先创建父表，再创建子表。
            删除表的时候，先删除子表，在删除父表。

        drop table if exists t_student;
        drop table if exists t_class;

        create table t_class(
            cno int,
            cname varchar(255),
            primary key(cno)
        );

        create table t_student(
            sno int,
            sname varchar(255),
            classno int,
            primary key(sno),
            foreign key(classno) references t_class(cno)
        );

        insert into t_class values(101,'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
```

```
        insert into t_class
values(102,'yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy');

        insert into t_student values(1,'zs1',101);
        insert into t_student values(2,'zs2',101);
        insert into t_student values(3,'zs3',102);
        insert into t_student values(4,'zs4',102);
        insert into t_student values(5,'zs5',102);
        insert into t_student values(6,'zs6',102);
        select * from t_class;
        select * from t_student;

        insert into t_student values(7,'lisi',103);
        ERROR 1452 (23000): Cannot add or update a child row: a
foreign key constraint fails (`bjpowernode`.INT `t_student_ibfk_1`
FOREIGN KEY (`classno`) REFERENCES `t_class` (`cno`))
```

五、存储引擎(了解)

- 表的不同存储方式，不同的存储引擎导致底层的表在存储数据时采用了不同的方式去存储。
- mysql默认使用的存储引擎是InnoDB方式。默认采用的字符集是UTF8

```
5.1、完整的建表语句
        CREATE TABLE `t_x` (
          `id` int(11) DEFAULT NULL
        ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

        注意：在MySQL当中，凡是标识符是可以使用飘号括起来的。最好别用，不通
用。
        建表的时候可以指定存储引擎，也可以指定字符集。

5.2、什么是存储引擎呢？
        存储引擎这个名字只有在mysql中存在。（Oracle中有对应的机制，但是不
叫做存储引擎。Oracle中没有特殊的名字，就是“表的存储方式”）
        mysql支持很多存储引擎，每一个存储引擎都对应了一种不同的存储方式。每
一个存储引擎都有自己的优缺点，需要在合适的时机选择合适的存储引擎。

5.3、查看当前mysql支持的存储引擎？
        show engines \G
```

5.1 常见的存储引擎

- MyISAM存储引擎
- InnoDB存储引擎
- MEMORY存储引擎

```
        Engine: MyISAM
        Support: YES
        Comment: MyISAM storage engine
        Transactions: NO
        XA: NO
        Savepoints: NO

            MyISAM这种存储引擎不支持事务。
            MyISAM是mysql最常用的存储引擎，但是这种引擎不是默认的。
            MyISAM采用三个文件组织一张表：
                xxx.frm（存储格式的文件）---存储表结构的定义
                xxx.MYD（存储表中数据的文件）---存储表行的内容
                xxx.MYI（存储表中索引的文件）----存储表上索引
            优点：可被压缩，节省存储空间。并且可以转换为只读表，提高检索效率。
            缺点：不支持事务。

--------------------------------------------------------------------------

        Engine: InnoDB
        Support: DEFAULT
        Comment: Supports transactions, row-level locking, and foreign keys
        Transactions: YES
        XA: YES
        Savepoints: YES

            优点：支持事务、行级锁、外键等。这种存储引擎数据的安全得到保障。

            表的结构存储在xxx.frm文件中
            数据存储在tablespace这样的表空间中（逻辑概念），无法被压缩，无法转换成只读。

            这种InnoDB存储引擎在MySQL数据库崩溃之后提供自动恢复机制。
            InnoDB支持级联删除和级联更新。

--------------------------------------------------------------------------------

        Engine: MEMORY
        Support: YES
        Comment: Hash based, stored in memory, useful for temporary tables
        Transactions: NO
        XA: NO
        Savepoints: NO

```

```
43              缺点：不支持事务。数据容易丢失。因为所有数据和索引都是存储在内存当中的，断电就没了。
44              优点：查询速度最快。
45
46              不能包含 TEXT 或 BLOB 字段
47              在数据库目录内，每个表均以.frm 格式的文件表示。
48              表级锁机制。
49              以前叫做HEPA引擎。
```

六、事务(Transaction)

- 一个事务是一个完整的业务逻辑单元，不可再分。为了保证数据的完整性，安全性，事务必须保证多条DML语句同时成功或者同时失败。
- 和事务相关的语句只有：DML语句。（insert delete update）
  因为它们这三个语句都是和数据库表当中的“数据”相关的。事务的存在是为了保证数据的完整性，安全性。
- 事务包括四大特性：ACID
  A: 原子性：事务是最小的工作单元，不可再分。
  C: 一致性：事务必须保证多条DML语句同时成功或者同时失败。
  I：隔离性：事务A与事务B之间具有隔离。
  D：持久性：持久性说的是最终数据必须持久化到硬盘文件中，事务才算成功的结束。
- 开启事务之后，对数据做的增删改操作会先保存在历史操作中，不会对数据库中文件做出改动，只有提交事务(commit)之后，才会对文件做出修改，并清空历史操作。回滚事务会清空历史操作，但不会对文件做出修改。
- 注意：rollback，或者 commit 后事务就结束了。
- 关于事务的回滚需要注意：只能回滚 insert、delete 和 update 语句，不能回滚 select（回滚 select 没有任何意义），对于 create、drop、alter 这些无法回滚

```
1  关于事务之间的隔离性
2      事务隔离性存在隔离级别，理论上隔离级别包括4个：
3          第一级别：读未提交（read uncommitted）
4              对方事务还没有提交，我们当前事务可以读取到对方未提交的数据。读未提交存在脏读（Dirty Read）现象：表示读到了脏的数据（不稳定）。
5
6          第二级别：读已提交（read committed）
7              对方事务提交之后的数据我方可以读取到。
8              这种隔离级别解决了：脏读现象没有了。
9              读已提交存在的问题是：不可重复读（就是每次读到的都是更改后的数据）。
10
11         第三级别：可重复读（repeatable read）
12             这种隔离级别解决了：不可重复读问题。
13             （两个事务隔离，另一个事务对数据的增删改我方无法得知，我方读取到的仍然是最开始的那张表，可重复读。）
14             这种隔离级别存在的问题是：读取到的数据是幻象。
```

```
        一个事务按相同的查询条件重新读取以前检索过的数据，却发现其他事务插入了满足其查询条件的新数据，这种现象就称为幻读。幻读和不可重复读都是读取了另一条已经提交的事务（这点就脏读不同），所不同的是不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）。

    第四级别：序列化读/串行化读（serializable）
        解决了所有问题。效率低。需要事务排队。

        oracle数据库默认的隔离级别是：读已提交。
        mysql数据库默认的隔离级别是：可重复读。
```

- 不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）。
- 不可重复读重点在于update和delete，而幻读的重点在于insert。避免不可重复读需要锁行（某一行在select操作时，不允许update与delete）就行，避免幻读则需要锁表。
- 幻读不能通过行锁来避免，需要Serializable隔离级别 ，读用读锁，写用写锁，读锁和写锁互斥，这么做可以有效的避免幻读、不可重复读、脏读等问题，但会极大的降低数据库的并发能力。所以说不可重复读和幻读最大的区别，就在于如何通过锁机制来解决他们产生的问题。

6.1 提交与回滚

- mysql事务默认情况下是自动提交的。----只要执行任意一条DML语句则提交一次。
- 关闭自动提交：start transaction;
  然后执行DML语句，最后用commit提交或者rollback回滚，标志着此次事务结束

```
    准备表：
        drop table if exists t_user;
        create table t_user(
            id int primary key auto_increment,
            username varchar(255)
        );
        insert into t_user(username) values('zs');
-------------------------------------------------------------------
    演示：使用start transaction;关闭自动提交机制。
        mysql> start transaction;
               Query OK, 0 rows affected (0.00 sec)

        mysql> insert into t_user(username) values('lisi');
               Query OK, 1 row affected (0.00 sec)

        mysql> insert into t_user(username) values('wangwu');
               Query OK, 1 row affected (0.00 sec)

```

```
            mysql> select * from t_user;
                    +----+----------+
                    | id | username |
                    +----+----------+
                    |  1 | zs       |
                    |  2 | lisi     |
                    |  3 | wangwu   |
                    +----+----------+
                    3 rows in set (0.00 sec)

            mysql> rollback;
                    Query OK, 0 rows affected (0.02 sec)

            mysql> select * from t_user;      //回滚之后，文件数据并未修改，只是清空了操作记录
                    +----+----------+
                    | id | username |
                    +----+----------+
                    |  1 | zs       |
                    +----+----------+
                    1 row in set (0.00 sec)
-------------------------------------------------------------------
--
            mysql> start transaction;
                    Query OK, 0 rows affected (0.00 sec)

            mysql> insert into t_user(username) values('wangwu');
                    Query OK, 1 row affected (0.00 sec)

            mysql> insert into t_user(username) values('rose');
                    Query OK, 1 row affected (0.00 sec)

            mysql> insert into t_user(username) values('jack');
                    Query OK, 1 row affected (0.00 sec)

            mysql> select * from t_user;
                    +----+----------+
                    | id | username |
                    +----+----------+
                    |  1 | zs       |
                    |  4 | wangwu   |
                    |  5 | rose     |
                    |  6 | jack     |
                    +----+----------+
                    4 rows in set (0.00 sec)

            mysql> commit;
                    Query OK, 0 rows affected (0.04 sec)

```

```
mysql> select * from t_user;    //提交之后，文件数据被修改
        +----+----------+
        | id | username |
        +----+----------+
        |  1 | zs       |
        |  4 | wangwu   |
        |  5 | rose     |
        |  6 | jack     |
        +----+----------+
        4 rows in set (0.00 sec)

mysql> rollback;       //提交之后，再次回滚数据也不会丢失
        Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
        +----+----------+
        | id | username |
        +----+----------+
        |  1 | zs       |
        |  4 | wangwu   |
        |  5 | rose     |
        |  6 | jack     |
        +----+----------+
        4 rows in set (0.00 sec)
```

6.2 事务隔离级别

- 事务隔离级别的作用范围分为两种： – 全局级：对所有的会话有效 ---------- GLOBAL
  – 会话级：只对当前的会话有效---------- SESSION
- 通过命令动态设置隔离级别
  语法：SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
- 查看当前隔离级别，可访问 transaction_isolation 变量：
  – 查看会话级的当前隔离级别：SELECT @@transaction_isolation; 或者 SELECT @@session.transaction_isolation;
  – 查看全局级的当前隔离级别：SELECT @@global.transaction_isolation;

```
演示两个事务，假如隔离级别
        演示第1级别：读未提交
                set global transaction isolation level read uncommitted;
        演示第2级别：读已提交
                set global transaction isolation level read committed;
        演示第3级别：可重复读
                set global transaction isolation level repeatable read;

        * mysql远程登录：mysql -h192.168.151.18 -uroot -p444
```

七、索引

- 索引就相当于一本书的目录，通过目录可以快速的找到对应的资源。在数据库方面，查询一张表的时候有两种检索方式：第一种方式：全表扫描
  第二种方式：根据索引检索（效率很高）
- 添加索引是给某一个字段，或者说某些字段添加索引。
- 什么时候考虑给字段添加索引？（满足什么条件）
  ---数据量庞大。（根据客户的需求，根据线上的环境）
  ---该字段很少的DML操作。（因为字段进行修改操作，索引也需要维护）
  ---该字段经常出现在where子句中。（经常根据哪个字段查询）
- 注意：主键和具有unique约束的字段自动会添加索引。根据主键查询效率较高。尽量根据主键检索。
- explain SQL语句 -------查看sql语句的执行计划：
  例如：explain select ename,sal from emp where sal = 5000;

```
    索引为什么可以提高检索效率呢？-----其实最根本的原理是缩小了扫描的范围。

    索引虽然可以提高检索效率，但是不能随意的添加索引，因为索引也是数据库当中的对象，也需要数据库不断的维护。是有维护成本的。比如，表中的数据经常被修改这样就不适合添加索引，因为数据一旦修改，索引需要重新排序，进行维护。
```

7.1 添加索引

- 创建索引对象：create index 索引名称 on 表名(字段名);
  例如：给薪资sal字段添加索引：create index emp_sal_index on emp(sal);
- 删除索引对象：drop index 索引名称 on 表名;
- 索引底层采用的数据结构是：B + Tree
- 索引的实现原理：通过B Tree缩小扫描范围，底层索引进行了排序，分区，索引会携带数据在表中的“物理地址”，最终通过索引检索到数据之后，获取到关联的物理地址，通过物理地址定位表中的数据，效率是最高的。
  例如：select ename from emp where ename = 'SMITH';通过索引转换为：
  select ename from emp where 物理地址 = 0x3;

```
    4.8、索引的分类：
        单一索引：给单个字段添加索引
        复合索引：给多个字段联合起来添加1个索引
        主键索引：主键上会自动添加索引
        唯一索引：有unique约束的字段上会自动添加索引
        ....

    4.9、索引什么时候失效？
        select ename from emp where ename like '%A%';
        模糊查询的时候，第一个通配符使用的是%，这个时候索引是失效的。
```

八、视图(view)

- 什么是视图？-----站在不同的角度去看到数据。（同一张表的数据，通过不同的角度去看待）。
- 视图有时也被成为“虚拟表”。
- 创建视图----create view myview as select empno,ename from emp;
  删除视图----drop view myview;
- 注意：只有DQL语句(select语句)才能以视图对象的方式创建出来。
- 对视图进行增删改查，会影响到原表数据。（通过视图影响原表数据的，不是直接操作的原表）可以对视图进行CRUD操作。
- 视图的作用：视图可以隐藏表的实现细节。保密级别较高的系统，数据库只对外提供相关的视图，java程序员只对视图对象进行CRUD。

```
8.1、面向视图操作
        create table emp_bak as select * from emp;
        create view myview1 as select empno,ename,sal from emp_bak;
        update myview1 set ename='hehe',sal=1 where empno = 7369;
// 通过视图修改原表数据。
        delete from myview1 where empno = 7369;    // 通过视图删除原表数据。
```

九、DBA命令

```
9.1、将数据库当中的数据导出
        在windows的dos命令窗口中执行：（导出整个库）-----不要登录进去执行
            mysqldump bjpowernode>D:\bjpowernode.sql -uroot -p333
-----  >表示导出到哪个位置

        在windows的dos命令窗口中执行：（导出指定数据库当中的指定表）
            mysqldump bjpowernode emp>D:\bjpowernode.sql -uroot -
p123

9.2、导入数据
        create database bjpowernode;
        use bjpowernode;
        source D:\bjpowernode.sql
```

十、数据库设计三范式

- 什么是设计范式：设计表的依据。按照这个三范式设计的表不会出现数据冗余。
- 第一范式：任何一张表都应该有主键，并且每一个字段原子性不可再分。
- 第二范式：建立在第一范式的基础之上，所有非主键字段完全依赖主键，不能产生部分依赖。
- 第三范式：建立在第二范式的基础之上，所有非主键字段直接依赖主键，不能产生传递依赖。
- 提醒：在实际的开发中，以满足客户的需求为主，有的时候会拿冗余换执行速度。表的联查执行速度会降低。
- 多对多？三张表，关系表两个外键。
- 一对多？两张表，多的表加外键。------多的那个字段加外键
- 一对一怎么设计？
  一对一设计有两种方案：主键共享。
  一对一设计有两种方案：外键唯一。

```
   多对多？三张表，关系表两个外键。
        t_student学生表
        sno(pk)      sname
        --------------------
        1                张三
        2                李四
        3                王五

        t_teacher 讲师表
        tno(pk)      tname
        ----------------------
        1                王老师
        2                张老师
        3                李老师
```

```
            t_student_teacher_relation 学生讲师关系表
            id(pk)      sno(fk)     tno(fk)
            ----------------------------------
            1               1               3
            2               1               1
            3               2               2
            4               2               3
            5               3               1
            6               3               3


    一对多？两张表，多的表加外键。
            班级t_class
            cno(pk)         cname
            -------------------------
            1                   班级1
            2                   班级2

            学生t_student
            sno(pk)         sname               classno(fk)
            ------------------------------------------
            101             张1             1
            102             张2             1
            103             张3             2
            104             张4             2
            105             张5             2


    一对一怎么设计？

        一对一设计有两种方案：主键共享
            t_user_login  用户登录表
            id(pk)      username            password
            ------------------------------------
            1               zs                  123
            2               ls                  456

            t_user_detail 用户详细信息表
            id(pk+fk)   realname            tel         ....
            ------------------------------------------------
            1               张三              1111111111
            2               李四              1111415621

        一对一设计有两种方案：外键唯一。
            t_user_login  用户登录表
            id(pk)      username            password
            ------------------------------------
            1               zs                  123
            2               ls                  456
```

```
65
66              t_user_detail 用户详细信息表
67              id(pk)      realname          tel
    userid(fk+unique)....
68              ----------------------------------------------------
    -----
69              1                张三               1111111111       2
70              2                李四               1111415621       1
71
```

作业

- 1、取得每个部门最高薪水的人员名称

```
1  第一步：取得每个部门最高薪水(按照部门编号分组，找出每一组最大值)
2  mysql> select deptno,max(sal) as maxsal from emp group by deptno;
3  +--------+---------+
4  | deptno | maxsal  |
5  +--------+---------+
6  |     10 | 5000.00 |
7  |     20 | 3000.00 |
8  |     30 | 2850.00 |
9  +--------+---------+
10 第二步：将查询结果当做临时表t，t和emp表连接，条件：t.deptno = e.deptno
   and t.maxsal = e.sal
11 select
12     e.ename, t.*
13 from
14     emp e
15 join
16     (select deptno,max(sal) as maxsal from emp group by deptno) t
17 on
18     t.deptno = e.deptno and t.maxsal = e.sal;
```

- 2、哪些人的薪水在部门的平均薪水之上

```
1  第一步：先找平均薪水
2      select deptno,avg(sal) as avgsal from emp group by deptno;
3  第二步：
4  select
5      e.ename,e.sal,t.*
6  from
7      emp e
8  join
9      (select deptno,avg(sal) as avgsal from emp group by deptno) t
10 on
11     e.deptno=t.deptno and e.sal > t.avgsal;
```

- 3、

```
1
```