MySQL day01 课堂笔记

1.什么是数据库？什么是数据库管理系统？什么是 SQL？他们之间的关系是什么？
数据库：英文单词 Database，简称 DB；
顾名思义，存储数据的仓库，实际上就是一堆文件，这些文件中存储了具有特定格式的数据
数据库管理系统：DataBaseManagement，简称 DBMS；
数据库管理系统是专门用来管理数据库中的数据的，数据库管理系统可以对数据库当中的数据进行增删改查

常见的数据库管理系统：
MySQL、Oracle、MS SQLServer、DB2、Sybase 等

SQL：结构化查询语言
程序员需要学习 SQL 语句，程序员通过编写 SQL 语句，然后 DBMS 负责执行 SQL 语句，
最终来完成数据库中数据的增删改查

SQL 是一套标准，程序员主要学习 SQL 语句，这个 SQL 可以在 MySQL 中使用，
同时 Oracle、DB2 中都可以使用

三者之间的关系：
DBMS->执行->SQL->操作->DB

先安装数据库管理系统 MySQL，然后学习 SQL 语句，编写 SQL 语句之后，DBMS 对 SQL 语句进行执行，
最终来完成数据库的数据管理

2.安装 MySQL 数据库管理系统
第一步：先安装，选择经典版
第二步：需要进行 MySQL 数据库实例配置
注意：
端口号：端口号 port 是任何一个软件/应用都会有的，端口号是应用的唯一代表
端口号通常和 IP 地址在一块，IP 地址用来定位计算机，端口号 port 用来定位某个服务/应用
在同一台计算机上，端口号不能重复，具有唯一性
MySQL 数据库启动的时候，这个服务占有的默认端口号为 3306

字符编码方式：
设置 MySQL 数据库的字符编码方式为 uft8

3.MySQL 数据库的卸载：
第一步：双击安装包进行卸载
第二步：删除目录

4.MySQL 的服务
计算机->右键->管理->服务和应用程序->服务->MySQL
MySQL 服务默认是启动状态，只有启动了 MySQL 才能用

5.在 Windows 操作系统中，如何启动和关闭 MySQL 服务
语法：net stop 服务名称;
net start 服务名称;

6.MySQL 安装了，服务启动，怎么使用客户端登录 MySQL 数据库？
使用 bin 目录下的 mysql.exe 命令连接 MySQL 数据库服务器

本地登录(显示编写密码的形式)：

```
C:\WINDOWS\system32>mysql -uroot -p(密码)
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.24 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

本地登录(显示编写密码的形式)：

```
C:\WINDOWS\system32>mysql -uroot -p
Enter password: ************
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.24 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

7.mysql 常用命令

退出 mysql：exit 或 \q

查看 mysql 中有哪些数据库：

```
show databases;
```

注意以英文分号结尾

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
5 rows in set (0.01 sec)
```

mysql 默认自带了四个数据库 information_schema mysql performance_schema sys

选择当前要使用的数据库：

```
mysql> use test;
Database changed
```

表示正在使用一个名字叫做 test 的数据库

创建数据库：

```
mysql> create database bjpowernode;
Query OK, 1 row affected (0.01 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| bjpowernode        |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

查看某个数据库下有哪些表：

```
mysql> use bjpowernode;
Database changed
mysql> show tables;
Empty set (0.01 sec)
```

查看 mysql 数据库的版本号:

```
mysql> select version();
+-----------+
| version() |
+-----------+
| 8.0.24    |
+-----------+
1 row in set (0.00 sec)
```

查看当前正在使用的数据库：

```
mysql> select database();
+-------------+
| database()  |
+-------------+
| bjpowernode |
+-------------+
1 row in set (0.00 sec)
```

注意：mysql 不见分号不执行

终止当前命令的输入用\c

```
mysql> show
-> 
-> 
-> \c
mysql>
```

注意：以上的命令不区分大小写，都可以

8.数据库当中最基本的单元是表(table)

什么是表 table？为什么用表来存储数据

数据库中是以表格的形式表示数据的

任何一张表都有行和列：
行 row：被称为数据/记录
列 column：被称为字段
姓名字段、性别字段、年龄字段... 每个字段都有：字段名 数据类型 约束等属性
字段名可以理解，是一个普通的名字，见名知意
数据类型：字符串 数字 日期等
约束：约束也有很多，其中一个叫做唯一性约束，
在这种约束添加以后，该字段的数据就不能重复

9.关于 SQL 语句的分类：
SQL 语句有很多，最好分门别类，容易记忆
分为：
DQL:数据查询语言(凡是带有 select 关键字的都是查询语句)
select...
DML:数据操作语言(凡是对表当中的数据进行增删改的都是 DML)
insert delete update
insert 增
delete 删
update 改
这些命令主要是操作表中的数据 data
DDL:数据定义语言(凡是带有 create、drop、alter 的都是 DDL)
DDL 主要操作的是表的结构，不是表中的数据
create:增
drop:删
alter:改
这些命令增删改与 DML 不同，这个主要是对表结构进行操作
TCL:事务控制语言
包括：
事务提交：commit
事务回滚：rollback
DCL:数据控制语言
例如：授权 grant、撤销授权 revoke

10.导入提前准备好的数据
bjpowernode.sql
导入数据 source
mysql> source F:\Toolkit\Backup\bjpowernode.sql

注意：路径中不要有中文

11.关于导入的这几张表?
```
mysql> show tables;
+----------------------+
| Tables_in_bjpowernode |
+----------------------+
| dept                 |
| emp                  |
| salgrade             |
+----------------------+
3 rows in set (0.01 sec)
```

dept 部门表
emp 员工表
salgrade 工资等级表

查看表中的数据?

select * from 表名; // 统一执行这个 SQL 语句

mysql> select * from emp; // 从 emp 表查询所有数据

```
+-------+--------+-----------+------+------------+---------+---------+--------+
| 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 |
+-------+--------+-----------+------+------------+---------+---------+--------+
14 rows in set (0.00 sec)
```

```
mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)
```

```
mysql> select * from salgrade;
+-------+-------+-------+
| GRADE | LOSAL | HISAL |
+-------+-------+-------+
|     1 |   700 |  1200 |
|     2 |  1201 |  1400 |
|     3 |  1401 |  2000 |
|     4 |  2001 |  3000 |
|     5 |  3001 |  9999 |
+-------+-------+-------+
5 rows in set (0.00 sec)
```

12.不看表中的数据，只看表的结构，有一个命令

desc 表名;

describe 表名;

```
mysql> desc dept;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| DEPTNO | int         | NO   | PRI | NULL    |       | 部门编号
| DNAME  | varchar(14) | YES  |     | NULL    |       | 部门名字
| LOC    | varchar(13) | YES  |     | NULL    |       | 地理位置
+--------+-------------+------+-----+---------+-------+
```

3 rows in set (0.01 sec)

mysql> desc emp;

Field	Type	Null	Key	Default	Extra	
EMPNO	int	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	YES		NULL		上级编号
HIREDATE	date	YES		NULL		入职日期
SAL	double(7,2)	YES		NULL		工资
COMM	double(7,2)	YES		NULL		补助
DEPTNO	int	YES		NULL		部门编号

8 rows in set (0.00 sec)

mysql> desc salgrade;

Field	Type	Null	Key	Default	Extra
GRADE	int	YES		NULL		工资等级
LOSAL	int	YES		NULL		最低工资
HISAL	int	YES		NULL		最高工资

3 rows in set (0.00 sec)

13.简单查询 DQL

1.查询一个字段

select 字段名 from 表名;

注意：select 和 from 都是关键字，字段名和表名都是标识符

对于所有的 SQL 语句，都是通用的，所有的 SQL 语句都是以";"结尾

SQL 语句不区分大小写，都行

查询员工的名字

mysql> select dname from dept;

dname
ACCOUNTING
RESEARCH
SALES
OPERATIONS

4 rows in set (0.00 sec)

2.查询两个字段，或多个字段

使用逗号隔开","，查询部门编号和部门名

mysql> select deptno,dname from dept;

deptno	dname
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

4 rows in set (0.00 sec)

3.查询所有字段

第一种方式：可以把所有字段都写上

```
select a,b,s,c,d... from tablename;
```

第二种方式：可以使用*

```
select * from tablename;
```

```
mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)
```

第二种方式的缺点：效率低、可读性差

在实际开发中不建议第二种方式，建议使用第一种方式

4.查询的列起别名 as 关键字

```
mysql> select deptno,dname as deptname from dept;
+--------+------------+
| deptno | deptname   |
+--------+------------+
|     10 | ACCOUNTING |
|     20 | RESEARCH   |
|     30 | SALES      |
|     40 | OPERATIONS |
+--------+------------+
4 rows in set (0.00 sec)
```

使用 as 关键字起别名。

注意：只是将显示的查询结果列名显示为 deptname，原表列名还是 dname

记住：select 语句是永远不会进行修改操作的(因为只负责查询)

as 关键字也可以省略，换成空格

```
mysql> select deptno,dname deptname from dept;
+--------+------------+
| deptno | deptname   |
+--------+------------+
|     10 | ACCOUNTING |
|     20 | RESEARCH   |
|     30 | SALES      |
|     40 | OPERATIONS |
+--------+------------+
4 rows in set (0.00 sec)
```

假设起别名的时候，别名里面含有空格，例如 DNAME->dept name

```
mysql> select deptno,dname dept name from dept;
```

DBMS 看到这样的语句，进行 SQL 语句的编译，不符合语法，编译报错

解决办法是给本身含有空格的别名使用单/双引号括起来

```
mysql> select deptno,dname 'dept name' from dept;
+--------+------------+
| deptno | dept name  |
+--------+------------+
|     10 | ACCOUNTING |
|     20 | RESEARCH   |
|     30 | SALES      |
|     40 | OPERATIONS |
+--------+------------+
```

```
4 rows in set (0.00 sec)

mysql> select deptno,dname "dept name"    from dept;
+--------+------------+
| deptno | dept name    |
+--------+------------+
|        10 | ACCOUNTING |
|        20 | RESEARCH     |
|        30 | SALES          |
|        40 | OPERATIONS |
+--------+------------+
4 rows in set (0.00 sec)
```

注意：在所有的数据库当中，字符串统一使用单引号括起来。

单引号是标准，双引号在 Oracle 数据库中用不了，但是在 mysql 中可以使用

再次强调：数据库中字符串都是采用单引号括起来的，这是标准的。双引号是不标准的

5.计算员工的薪资 sal * 12

```
mysql> select ename,sal from emp;
+--------+---------+
| 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 |
+--------+---------+
14 rows in set (0.00 sec)
```

直接让工资乘以 12 即可，但是字段名称变成了 sal*12

```
mysql> select ename,sal*12 from emp; // 结论：字段可以使用数学表达式
+--------+----------+
| 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 |
+--------+----------+
14 rows in set (0.00 sec)
```

起别名，如果别名是中文，则使用单引号括起来，当做字符串处理

```
mysql> select ename,sal*12 '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 |
+--------+----------+
14 rows in set (0.00 sec)
```

14.条件查询 不是将表中所有数据都查出来，而是将符合条件的数据查询出来

语法格式：

```
select
    字段 1,字段 2,字段 3...
from
    表名
where
    条件;
```

条件语句：

1.=等于

```
select * from 表 where 字段名='查询的字段值';
```

注意:查询的字段值最好用''包裹

例如：查询薪资等于 800 的员工姓名和编号

```
mysql> select empno,ename from emp where sal='800';
+-------+-------+
| empno | ename |
+-------+-------+
|  7369 | SMITH |
+-------+-------+
1 row in set (0.00 sec)
```

例如：查询 SMITH 的姓名、编号和薪资

```
mysql> select empno,ename,sal from emp where ename='SMITH';
+-------+-------+--------+
| empno | ename | sal    |
+-------+-------+--------+
|  7369 | SMITH | 800.00 |
+-------+-------+--------+
1 row in set (0.00 sec)
```

2.<>或!=不等于

select * from 表 where 字段名<>'查询的字段值';
select * from 表 where 字段名!='查询的字段值';

例如：查询薪资不等于 800 的员工姓名和编号

```
mysql> select ename,empno from emp where sal != '800';
或
mysql> select ename,empno from emp where sal <> '800';
+--------+-------+
| ename  | empno |
+--------+-------+
| ALLEN  |  7499 |
| WARD   |  7521 |
| JONES  |  7566 |
| MARTIN |  7654 |
| BLAKE  |  7698 |
| CLARK  |  7782 |
| SCOTT  |  7788 |
| KING   |  7839 |
| TURNER |  7844 |
| ADAMS  |  7876 |
| JAMES  |  7900 |
| FORD   |  7902 |
| MILLER |  7934 |
+--------+-------+
13 rows in set (0.00 sec)
```

3.<小于
select * from 表 where 字段名<'查询的字段值';

例如：查询薪资小于 2000 的员工姓名和编号、薪资

```
mysql> select empno,ename,sal from emp where sal<'2000';
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|  7369 | SMITH  |  800.00 |
|  7499 | ALLEN  | 1600.00 |
|  7521 | WARD   | 1250.00 |
|  7654 | MARTIN | 1250.00 |
|  7844 | TURNER | 1500.00 |
|  7876 | ADAMS  | 1100.00 |
|  7900 | JAMES  |  950.00 |
|  7934 | MILLER | 1300.00 |
+-------+--------+---------+
8 rows in set (0.00 sec)
```

4.<=小于等于
select * from 表 where 字段名<='查询的字段值';

例如：查询薪资小于等于 5000 的员工姓名和编号、薪资

```
mysql> select empno,ename,sal from emp where sal<='5000';
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|  7369 | SMITH  |  800.00 |
|  7499 | ALLEN  | 1600.00 |
|  7521 | WARD   | 1250.00 |
|  7566 | JONES  | 2975.00 |
|  7654 | MARTIN | 1250.00 |
```

```
|   7698 | BLAKE    | 2850.00 |
|   7782 | CLARK    | 2450.00 |
|   7788 | SCOTT    | 3000.00 |
|   7839 | KING     | 5000.00 |
|   7844 | TURNER | 1500.00 |
|   7876 | ADAMS    | 1100.00 |
|   7900 | JAMES    |  950.00 |
|   7902 | FORD     | 3000.00 |
|   7934 | MILLER | 1300.00 |
+-------+--------+---------+
14 rows in set (0.00 sec)
```

5.>大于

select * from 表 where 字段名>查询的字段值';

例如：查询薪资大于 3000 的员工姓名和编号、薪资

```
mysql> select empno,ename,sal from emp where sal > '3000';
+-------+-------+---------+
| empno | ename | sal     |
+-------+-------+---------+
|   7839 | KING  | 5000.00 |
+-------+-------+---------+
1 row in set (0.00 sec)
```

6.>=大于等于

select * from 表 where 字段名>='查询的字段值';

例如：查询薪资大于等于 1000 的员工姓名和编号、薪资

```
mysql> select empno,ename,sal from emp where sal >= '1000';
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|   7499 | ALLEN  | 1600.00 |
|   7521 | WARD   | 1250.00 |
|   7566 | JONES  | 2975.00 |
|   7654 | MARTIN | 1250.00 |
|   7698 | BLAKE  | 2850.00 |
|   7782 | CLARK  | 2450.00 |
|   7788 | SCOTT  | 3000.00 |
|   7839 | KING   | 5000.00 |
|   7844 | TURNER | 1500.00 |
|   7876 | ADAMS  | 1100.00 |
|   7902 | FORD   | 3000.00 |
|   7934 | MILLER | 1300.00 |
+-------+--------+---------+
12 rows in set (0.00 sec)
```

7.between...and..两个值之间等同于>=and<= ，闭区间

select * from 表 where 字段名 between 值 1 and 值 2;

注意在查询的时候小值必须写在前面如果大值写在前面查询不到任何数据

查询薪资大于 1000 且小于 3000 的员工姓名和编号、薪资

```
mysql> select empno,ename,sal from emp where sal between '1000' and
'3000';
```

或

```
select empno,ename,sal from emp where sal>'1000' and sal<'3000';
+-------+--------+---------+
| empno | ename  | sal     |
```

```
+-------+--------+---------+
|  7499 | ALLEN  | 1600.00 |
|  7521 | WARD   | 1250.00 |
|  7566 | JONES  | 2975.00 |
|  7654 | MARTIN | 1250.00 |
|  7698 | BLAKE  | 2850.00 |
|  7782 | CLARK  | 2450.00 |
|  7788 | SCOTT  | 3000.00 |
|  7844 | TURNER | 1500.00 |
|  7876 | ADAMS  | 1100.00 |
|  7902 | FORD   | 3000.00 |
|  7934 | MILLER | 1300.00 |
+-------+--------+---------+
11 rows in set (0.00 sec)
```

注意：使用 between...and...的时候，必须遵循左小右大的闭区间

8.is null 为 null(is not null 非空)

```
SELECT * FROM 表 WHERE 字段名 IS null;
```

例如：查询哪些员工的津贴/补助为 null

```
mysql> select empno,ename,sal,comm from emp where comm is null;
+-------+--------+---------+------+
| empno | ename  | sal     | comm |
+-------+--------+---------+------+
|  7369 | SMITH  |  800.00 | NULL |
|  7566 | JONES  | 2975.00 | NULL |
|  7698 | BLAKE  | 2850.00 | NULL |
|  7782 | CLARK  | 2450.00 | NULL |
|  7788 | SCOTT  | 3000.00 | NULL |
|  7839 | KING   | 5000.00 | NULL |
|  7876 | ADAMS  | 1100.00 | NULL |
|  7900 | JAMES  |  950.00 | NULL |
|  7902 | FORD   | 3000.00 | NULL |
|  7934 | MILLER | 1300.00 | NULL |
+-------+--------+---------+------+
10 rows in set (0.00 sec)
```

注意：在数据库当中 null 不能使用等号进行衡量，需要使用 is null

因为数据库中的 null 代表什么也没有，它不是一个值，所以不能使用等号进行衡量

例如：查询哪些员工的津贴/补助不为 null

```
mysql> select empno,ename,sal,comm from emp where comm is not null;
+-------+--------+---------+---------+
| empno | ename  | sal     | comm    |
+-------+--------+---------+---------+
|  7499 | ALLEN  | 1600.00 |  300.00 |
|  7521 | WARD   | 1250.00 |  500.00 |
|  7654 | MARTIN | 1250.00 | 1400.00 |
|  7844 | TURNER | 1500.00 |    0.00 |
+-------+--------+---------+---------+
4 rows in set (0.00 sec)
```

9.and 并且

```
select * from 表 where 字段>'值 1' and 字段<'值 2';
```

例如：查询工作岗位是 MANAGER 并且工资大于 2000 的员工信息

```
mysql> select * from emp where job='MANAGER' and sal >= '2000';
```

```
+-------+-------+---------+------+------------+---------+------+-------
-+
| EMPNO | ENAME | JOB       | MGR  | HIREDATE   | SAL       | COMM |
DEPTNO |
+-------+-------+---------+------+------------+---------+------+-------
-+
|  7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL |      20 |
|  7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850.00 | NULL |      30 |
|  7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450.00 | NULL |      10 |
+-------+-------+---------+------+------------+---------+------+-------
-+
3 rows in set (0.00 sec)
```

10.or 或者

select * from 表 where 字段='值 1' or 字段='值 2';

例如：查询工作岗位是 MANAGER 或者 SALESMAN 的员工信息

```
mysql> select * from emp where job='MANAGER' or job='SALESMAN';
+-------+--------+----------+------+------------+---------+---------+--
------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   | SAL       | COMM     |
DEPTNO |
+-------+--------+----------+------+------------+---------+---------+--
------+
|  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 |
|  7844 | TURNER | SALESMAN | 7698 | 1981-09-08 | 1500.00 |     0.00 |
30 |
+-------+--------+----------+------+------------+---------+---------+--
------+
7 rows in set (0.00 sec)
```

and 和 or 同时出现的话，优先级问题？

例如：查询一下工资大于 2500 并且部门编号为 10 或者 20 部门的员工

select * from emp where sal>'2500' and deptno='10' or deptno='20';

分析以上语句的问题？

and 的优先级高于 or

以上语句会先执行 and，然后执行 or

语句的含义是：查询工资大于 2500 并且部门编号为 10 的员工信息，

或者部门编号为 20 的所以员工信息。

条件相当于 (sal>'2500' and deptno='10') or (deptno='20')

执行的结果为

```
mysql> select * from emp where sal>'2500' and deptno='10' or deptno='20';
+-------+-------+----------+------+------------+---------+------+-----
---+
| EMPNO | ENAME | JOB       | MGR  | HIREDATE   | SAL       | COMM |
DEPTNO |
```

```
+-------+-------+-----------+------+-----------+---------+------+-----
---+
| 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800.00 | NULL | 20 |
| 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL | 20
|
| 7788 | SCOTT | ANALYST | 7566 | 1987-04-19 | 3000.00 | NULL | 20 |
| 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL | 10 |
| 7876 | ADAMS | CLERK | 7788 | 1987-05-23 | 1100.00 | NULL | 20 |
| 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL | 20 |
+-------+-------+-----------+------+-----------+---------+------+-----
---+
6 rows in set (0.00 sec)
```

解决办法，加括号提高优先级

```
select * from emp where sal>'2500' and (deptno='10' or deptno='20');
```

```
mysql> select * from emp where sal>'2500' and (deptno='10' or
deptno='20');
+-------+-------+-----------+------+-----------+---------+------+-
-------+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM |
DEPTNO |
+-------+-------+-----------+------+-----------+---------+------+-
-------+
| 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975.00 | NULL |
20 |
| 7788 | SCOTT | ANALYST | 7566 | 1987-04-19 | 3000.00 | NULL |
20 |
| 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL |
10 |
| 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000.00 | NULL |
20 |
+-------+-------+-----------+------+-----------+---------+------+-
-------+
4 rows in set (0.00 sec)
```

and 和 or 同时出现，and 优先级较高。如果想让 or 的优先级先执行，需要加小括号；

以后的开发中，如果不确定优先级，就加小括号就可以了

11.in 包含相当于多个 or(not in 不在这个范围中)

```
SELECT * FROM 表 WHERE 字段 IN ('值 1','值 2',.....);
```

例如：查询工作岗位是 MANAGER 或者 SALESMAN 的员工信息

```
mysql> select * from emp where job in ('MANAGER','SALESMAN');
+-------+--------+----------+------+-----------+---------+---------+--
------+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM |
DEPTNO |
+-------+--------+----------+------+-----------+---------+---------+--
------+
| 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 |
|   7844 | TURNER | SALESMAN | 7698 | 1981-09-08 | 1500.00 |    0.00 |
30 |
+-------+--------+----------+------+------------+---------+---------+--
------+
7 rows in set (0.00 sec)
```

注意：in 不是一个区间

例如：查询薪资是 800 或者 5000 的员工信息

```
mysql> select * from emp where sal in ('800','5000');
+-------+-------+-----------+------+------------+---------+------+-----
---+
| EMPNO | ENAME | JOB       | MGR  | HIREDATE   | SAL     | COMM |
DEPTNO |
+-------+-------+-----------+------+------------+---------+------+-----
---+
|  7369 | SMITH | CLERK     | 7902 | 1980-12-17 |  800.00 | NULL |
20 |
|  7839 | KING  | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL |
10 |
+-------+-------+-----------+------+------------+---------+------+-----
---+
2 rows in set (0.00 sec)
```

not in 表示不在这几个值当中的数据

```
mysql> select * from emp where sal not in ('800','5000','3000');
+-------+--------+----------+------+------------+---------+--------
-+--------+
| EMPNO | ENAME  | JOB      | MGR  | HIREDATE   | SAL     | COMM
| DEPTNO |
+-------+--------+----------+------+------------+---------+--------
-+--------+
|  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 |
|  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 |
|  7934 | MILLER | CLERK    | 7782 | 1982-01-23 | 1300.00 |    NULL |
10 |
+-------+--------+----------+------+------------+---------+--------
-+--------+
10 rows in set (0.00 sec)
```

12.like 模糊查询
支持%或下划线匹配
　　%匹配任意个字符(%是一个特殊的符号)
　　下划线，一个下划线只能匹配一个字符(下划线也是一个特殊的字符)
比如我们要查询表中字段名为 name 字段值为张三的数据(两个 sql 语句获得的值相等)

```
select * from  表  where name like '%三%';
select * from  表  where name like '张三';
```

例如，找出名字中含有 O 的员工信息

```
mysql> select ename from emp where ename like '%O%';
+-------+
| ename |
+-------+
| JONES |
| SCOTT |
| FORD  |
+-------+
3 rows in set (0.00 sec)
```

例如，找出名字中以 T 结尾的员工信息

```
mysql> select ename from emp where ename like '%T';
+-------+
| ename |
+-------+
| SCOTT |
+-------+
1 row in set (0.00 sec)
```

例如，找出名字中以 S 开始的员工信息

```
mysql> select ename from emp where ename like 'S%';
+-------+
| ename |
+-------+
| SMITH |
| SCOTT |
+-------+
2 rows in set (0.00 sec)
```

例如，找出名字中第二个字母为 A 的员工信息

```
mysql> select ename from emp where ename like '_A%';
+--------+
| ename  |
+--------+
| WARD   |
| MARTIN |
| JAMES  |
+--------+
3 rows in set (0.00 sec)
```

例如，找出名字中第三个字母为 R 的员工信息

```
mysql> select ename from emp where ename like '__R%';
+--------+
| ename  |
+--------+
| WARD   |
| MARTIN |
| TURNER |
| FORD   |
+--------+
```

4 rows in set (0.00 sec)

例如，找出名字中第三个字母为 R 并且最后一个字母不为 R 的员工信息

```
mysql> select ename from emp where (ename like '__R%') and (ename not like '%R');
+--------+
| ename  |
+--------+
| WARD   |
| MARTIN |
| FORD   |
+--------+
3 rows in set (0.00 sec)
```

如何查询字段值含有%或_的数据信息，使用转义字符\

有一张表

```
mysql> select * from t_student;
+----------+
| name     |
+----------+
| zhangsan |
| lisi     |
| wangwu   |
| jack_son |
| jack%ma  |
+----------+
5 rows in set (0.00 sec)
```

现要将 jack_son 和 jack%ma 查找出来，则使用转义字符

```
mysql> select * from t_student where name like '%\_%';
+----------+
| name     |
+----------+
| jack_son |
+----------+
1 row in set (0.00 sec)

mysql> select * from t_student where name like '%\%%';
+---------+
| name    |
+---------+
| jack%ma |
+---------+
1 row in set (0.00 sec)
```

13.not 可以取非或不是(不在这个范围内一般用于 is 和 in)

```
SELECT * FROM 表 WHERE 字段名 NOT IN('值 1','值 2',....);
SELECT * FROM 表 WHERE 字段名 IS NOT null;
```

15.排序

1.查询所有员工的薪资，并排序，默认升序

```
mysql> select empno,ename,sal from emp order by sal;
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|  7369 | SMITH  |  800.00 |
|  7900 | JAMES  |  950.00 |
|  7876 | ADAMS  | 1100.00 |
```

```
|   7521 | WARD     | 1250.00 |
|   7654 | MARTIN | 1250.00 |
|   7934 | MILLER | 1300.00 |
|   7844 | TURNER | 1500.00 |
|   7499 | ALLEN    | 1600.00 |
|   7782 | CLARK    | 2450.00 |
|   7698 | BLAKE    | 2850.00 |
|   7566 | JONES    | 2975.00 |
|   7788 | SCOTT    | 3000.00 |
|   7902 | FORD     | 3000.00 |
|   7839 | KING     | 5000.00 |
+-------+--------+---------+
14 rows in set (0.01 sec)
```

2.怎么降序排列

指定降序：

```
mysql> select empno,ename,sal from emp order by sal desc;
+-------+--------+---------+
| empno | ename    | sal        |
+-------+--------+---------+
|   7839 | KING     | 5000.00 |
|   7788 | SCOTT    | 3000.00 |
|   7902 | FORD     | 3000.00 |
|   7566 | JONES    | 2975.00 |
|   7698 | BLAKE    | 2850.00 |
|   7782 | CLARK    | 2450.00 |
|   7499 | ALLEN    | 1600.00 |
|   7844 | TURNER | 1500.00 |
|   7934 | MILLER | 1300.00 |
|   7521 | WARD     | 1250.00 |
|   7654 | MARTIN | 1250.00 |
|   7876 | ADAMS    | 1100.00 |
|   7900 | JAMES    |   950.00 |
|   7369 | SMITH    |   800.00 |
+-------+--------+---------+
14 rows in set (0.00 sec)
```

指定升序：

```
mysql> select empno,ename,sal from emp order by sal asc;
+-------+--------+---------+
| empno | ename    | sal        |
+-------+--------+---------+
|   7369 | SMITH    |   800.00 |
|   7900 | JAMES    |   950.00 |
|   7876 | ADAMS    | 1100.00 |
|   7521 | WARD     | 1250.00 |
|   7654 | MARTIN | 1250.00 |
|   7934 | MILLER | 1300.00 |
|   7844 | TURNER | 1500.00 |
|   7499 | ALLEN    | 1600.00 |
|   7782 | CLARK    | 2450.00 |
|   7698 | BLAKE    | 2850.00 |
|   7566 | JONES    | 2975.00 |
|   7788 | SCOTT    | 3000.00 |
|   7902 | FORD     | 3000.00 |
|   7839 | KING     | 5000.00 |
+-------+--------+---------+
14 rows in set (0.00 sec)
```

2.可以两个字段排序吗？或者说按照多个字段排序

查询员工名字和薪资，要求按照薪资升序，如果薪资一样，按照名字升序拍

```
mysql> select empno,ename,sal from emp order by sal asc,ename asc;
```

// sal 在前，起主导，只有 sal 相等的时候，ename 才发挥作用，按照升序排列

```
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|  7369 | SMITH  |  800.00 |
|  7900 | JAMES  |  950.00 |
|  7876 | ADAMS  | 1100.00 |
|  7654 | MARTIN | 1250.00 |
|  7521 | WARD   | 1250.00 |
|  7934 | MILLER | 1300.00 |
|  7844 | TURNER | 1500.00 |
|  7499 | ALLEN  | 1600.00 |
|  7782 | CLARK  | 2450.00 |
|  7698 | BLAKE  | 2850.00 |
|  7566 | JONES  | 2975.00 |
|  7902 | FORD   | 3000.00 |
|  7788 | SCOTT  | 3000.00 |
|  7839 | KING   | 5000.00 |
+-------+--------+---------+
14 rows in set (0.00 sec)
```

3.了解，根据字段的位置也可以排序

```
mysql> select empno,ename,sal from emp order by 2 asc;
// 2 表示第 2 列，整个表格按照查询结果的第 2 列升序排列
// 不建议这样写，因为不健壮
// 因为列的顺序很容易发生改变，列顺序修改以后，2 就不可以了
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|  7876 | ADAMS  | 1100.00 |
|  7499 | ALLEN  | 1600.00 |
|  7698 | BLAKE  | 2850.00 |
|  7782 | CLARK  | 2450.00 |
|  7902 | FORD   | 3000.00 |
|  7900 | JAMES  |  950.00 |
|  7566 | JONES  | 2975.00 |
|  7839 | KING   | 5000.00 |
|  7654 | MARTIN | 1250.00 |
|  7934 | MILLER | 1300.00 |
|  7788 | SCOTT  | 3000.00 |
|  7369 | SMITH  |  800.00 |
|  7844 | TURNER | 1500.00 |
|  7521 | WARD   | 1250.00 |
+-------+--------+---------+
14 rows in set (0.00 sec)
```

4.综合案例：

找出工资在 1250-3000 的员工信息，要求按照薪资降序排列

```
mysql> select * from emp where sal between '1250' and '3000' order by sal
desc;
// 关键字顺序不能变
// select ... from ... where ... order by ...
+-------+--------+----------+------+------------+---------+---------+--
------+
| EMPNO | ENAME  | JOB      | MGR  | HIREDATE   | SAL     | COMM    |
DEPTNO |
+-------+--------+----------+------+------------+---------+---------+--
------+
|  7788 | SCOTT  | ANALYST  | 7566 | 1987-04-19 | 3000.00 |    NULL |
20 |
|  7902 | FORD   | ANALYST  | 7566 | 1981-12-03 | 3000.00 |    NULL |
```

```
20 |
            |   7566 | JONES   | MANAGER   | 7839 | 1981-04-02 | 2975.00 |      NULL |
20 |
            |   7698 | BLAKE   | MANAGER   | 7839 | 1981-05-01 | 2850.00 |      NULL |
30 |
            |   7782 | CLARK   | MANAGER   | 7839 | 1981-06-09 | 2450.00 |      NULL |
10 |
            |   7499 | ALLEN   | SALESMAN  | 7698 | 1981-02-20 | 1600.00 |   300.00 |
30 |
            |   7844 | TURNER  | SALESMAN  | 7698 | 1981-09-08 | 1500.00 |     0.00 |
30 |
            |   7934 | MILLER  | CLERK     | 7782 | 1982-01-23 | 1300.00 |      NULL |
10 |
            |   7521 | WARD    | SALESMAN  | 7698 | 1981-02-22 | 1250.00 |   500.00 |
30 |
            |   7654 | MARTIN  | SALESMAN  | 7698 | 1981-09-28 | 1250.00 | 1400.00 |
30 |
            +-------+--------+----------+------+------------+---------+---------+--
------+
            10 rows in set (0.00 sec)
```

以上语句的执行顺序必须掌握：

第一步：from

第二步：where

第三步：select

第四步：order by(排序总是在最后执行)

16.数据处理函数

1.数据处理函数又被称为单行处理函数

单行处理函数的特点：一个输入对应一个输出

和单行处理函数相对的是：多行处理函数(多行处理函数特点：多个输入对应一个输出)

2.常见的单行处理函数

1. lower 转换小写

例如，将员工姓名全部转换成小写

```
mysql> select lower(ename) as ename from emp;
+--------+
| ename  |
+--------+
| smith  |
| allen  |
| ward   |
| jones  |
| martin |
| blake  |
| clark  |
| scott  |
| king   |
| turner |
| adams  |
| james  |
| ford   |
| miller |
+--------+
14 rows in set (0.00 sec)
```

14 个输入，最后还是 14 个输出，这是单行处理的特点

2. upper 转换大写

例如，将 t_student 中字段为 name 的值全部改大写

```
mysql> select * from t_student;
+----------+
| name     |
+----------+
| zhangsan |
| lisi     |
| wangwu   |
| jack_son |
| jack%ma  |
+----------+
5 rows in set (0.00 sec)

mysql> select upper(name) name from t_student;
+----------+
| name     |
+----------+
| ZHANGSAN |
| LISI     |
| WANGWU   |
| JACK_SON |
| JACK%MA  |
+----------+
5 rows in set (0.00 sec)
```

3.substr 取子串(被截取的字符串,起始下标,截取的长度)

例如：所有员工的姓名取前两位输出

```
mysql> select substr(ename,1,2) ename from emp;
+-------+
| ename |
+-------+
| SM    |
| AL    |
| WA    |
| JO    |
| MA    |
| BL    |
| CL    |
| SC    |
| KI    |
| TU    |
| AD    |
| JA    |
| FO    |
| MI    |
+-------+
14 rows in set (0.00 sec)
```

注意：起始下标从 1 开始，没有 0

例如，找出员工姓名中第一个字母为 A 的员工信息

方式一：模糊查询

```
mysql> select * from emp where ename like 'A%';
+-------+-------+----------+------+------------+---------+--------+
--------+
| EMPNO | ENAME | JOB      | MGR  | HIREDATE   | SAL     | COMM
| DEPTNO |
+-------+-------+----------+------+------------+---------+--------+
--------+
|  7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 |
```

```
30 |
|   7876 | ADAMS | CLERK       | 7788 | 1987-05-23 | 1100.00 |     NULL |
20 |
+-------+-------+---------+------+------------+---------+-------+
--------+
2 rows in set (0.00 sec)
```

方式二：使用单行函数 substr 处理

```
mysql> select * from emp where substr(ename,1,1)='A';
+-------+-------+---------+------+------------+---------+-------+
--------+
| EMPNO | ENAME | JOB         | MGR  | HIREDATE   | SAL     | COMM
| DEPTNO |
+-------+-------+---------+------+------------+---------+-------+
--------+
|   7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 |
30 |
|   7876 | ADAMS | CLERK       | 7788 | 1987-05-23 | 1100.00 |     NULL |
20 |
+-------+-------+---------+------+------------+---------+-------+
--------+
2 rows in set (0.00 sec)
```

例如，将 t_student 中姓名中第一个字母变大写

```
// 使用了字符串拼接函数 concat
mysql> select concat(upper(substr(name,1,1)),substr(name,2,length(name)-1))
as name from t_student;
+----------+
| name     |
+----------+
| Zhangsan |
| Lisi     |
| Wangwu   |
| Jack_son |
| Jack%ma  |
+----------+
5 rows in set (0.00 sec)
```

4.length 取长度

例如：计算 emp 表格中员工姓名的长度

```
mysql> select length(ename) as ename from emp;
+-------+
| ename |
+-------+
|     5 |
|     5 |
|     4 |
|     5 |
|     6 |
|     5 |
|     5 |
|     5 |
|     4 |
|     6 |
|     5 |
|     5 |
|     4 |
|     6 |
+-------+
14 rows in set (0.00 sec)
```

5.trim 去空格，去除前后空格

例如：查询姓名为 KING 的员工

```
mysql> select * from emp where ename=' KING     ';
Empty set (0.00 sec)

mysql> select * from emp where ename=trim(' KING     ');
+-------+-------+-----------+------+------------+---------+------+-----
---+
| EMPNO | ENAME | JOB       | MGR  | HIREDATE   | SAL     | COMM |
DEPTNO |
+-------+-------+-----------+------+------------+---------+------+-----
---+
|  7839 | KING  | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL |
10 |
+-------+-------+-----------+------+------------+---------+------+-----
---+
1 row in set (0.00 sec)
```

6.str_to_date 将字符串转换成日期

7.date_format 格式化日期

例如：将 HIREDATE 的时间格式进行转换

```
mysql> select date_format(HIREDATE,'%b %d %Y') from emp;
+----------------------------------+
| date_format(HIREDATE,'%b %d %Y') |
+----------------------------------+
| Dec 17 1980                      |
| Feb 20 1981                      |
| Feb 22 1981                      |
| Apr 02 1981                      |
| Sep 28 1981                      |
| May 01 1981                      |
| Jun 09 1981                      |
| Apr 19 1987                      |
| Nov 17 1981                      |
| Sep 08 1981                      |
| May 23 1987                      |
| Dec 03 1981                      |
| Dec 03 1981                      |
| Jan 23 1982                      |
+----------------------------------+
14 rows in set (0.00 sec)
```

8.format 设置千分位

SELECT FORMAT(column_name,format) FROM table_name

```
mysql> select ename,sal,format(NOW(),'%Y-%m-%d') as date from emp;
+--------+---------+--------------------+
| ename  | sal     | date               |
+--------+---------+--------------------+
| SMITH  |  800.00 | 20,210,508,230,016 |
| ALLEN  | 1600.00 | 20,210,508,230,016 |
| WARD   | 1250.00 | 20,210,508,230,016 |
| JONES  | 2975.00 | 20,210,508,230,016 |
| MARTIN | 1250.00 | 20,210,508,230,016 |
| BLAKE  | 2850.00 | 20,210,508,230,016 |
| CLARK  | 2450.00 | 20,210,508,230,016 |
| SCOTT  | 3000.00 | 20,210,508,230,016 |
| KING   | 5000.00 | 20,210,508,230,016 |
| TURNER | 1500.00 | 20,210,508,230,016 |
| ADAMS  | 1100.00 | 20,210,508,230,016 |
| JAMES  |  950.00 | 20,210,508,230,016 |
| FORD   | 3000.00 | 20,210,508,230,016 |
| MILLER | 1300.00 | 20,210,508,230,016 |
```

```
+--------+---------+--------------------+
14 rows in set, 14 warnings (0.00 sec)
```

9.round 四舍五入

```
mysql> select 2100 as num from dept;
+------+
| num  |
+------+
| 2100 |
| 2100 |
| 2100 |
| 2100 |
+------+
4 rows in set (0.00 sec)
```

结论：select 后面可以跟某个表的字段名，也可以跟字面量/字面值(数据)

```
mysql> select round( '123.456',2) as result from emp;
+--------+
| result |
+--------+
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
| 123.46 |
+--------+
14 rows in set (0.00 sec)
```

例如：将员工的工资四舍五入取整

// 注意 round 函数四舍五入存在一些问题

```
mysql> select sal,round(sal,-2) from emp;
+---------+---------------+
| sal     | round(sal,-2) |
+---------+---------------+
|  800.00 |           800 |
| 1600.00 |          1600 |
| 1250.00 |          1200 |
| 2975.00 |          3000 |
| 1250.00 |          1200 |
| 2850.00 |          2800 |
| 2450.00 |          2400 |
| 3000.00 |          3000 |
| 5000.00 |          5000 |
| 1500.00 |          1500 |
| 1100.00 |          1100 |
|  950.00 |          1000 |
| 3000.00 |          3000 |
| 1300.00 |          1300 |
+---------+---------------+
14 rows in set (0.00 sec)
```

10.rand 生成随机数

例如：取 100 以内的随机数

```
mysql> select round(rand()*100) rand from emp;
+------+
| rand |
+------+
|   24 |
|   24 |
|   49 |
|   74 |
|   23 |
|   90 |
|   84 |
|   51 |
|    1 |
|   53 |
|   63 |
|   56 |
|   89 |
|   79 |
+------+
14 rows in set (0.01 sec)
```

11.ifnull 空处理函数 可以将 null 转换成一个具体值

在所有数据库当中，只要有 null 参与的数据运算，结果都为 null

例如：计算每个员工的月薪，计算公式：工资 = (月薪+补助);

```
mysql> select ename,sal+comm as salary from emp;
+--------+---------+
| ename  | salary  |
+--------+---------+
| SMITH  |    NULL |
| ALLEN  | 1900.00 |
| WARD   | 1750.00 |
| JONES  |    NULL |
| MARTIN | 2650.00 |
| BLAKE  |    NULL |
| CLARK  |    NULL |
| SCOTT  |    NULL |
| KING   |    NULL |
| TURNER | 1500.00 |
| ADAMS  |    NULL |
| JAMES  |    NULL |
| FORD   |    NULL |
| MILLER |    NULL |
+--------+---------+
14 rows in set (0.00 sec)
```

注意：null 只要参与运算，最后结果一定是 null。为了避免这个结果，需要使用 ifnull 函数处理

ifnull 的用法：ifnull(数据，要替换的值);

例如：计算每个员工的年薪，计算公式：年薪 = (月薪+补助)*12;

```
mysql> select ename,(sal+ifnull(comm,'0'))*12 from emp;
+--------+--------------------------+
| ename  | (sal+ifnull(comm,'0'))*12 |
+--------+--------------------------+
| SMITH  |                     9600 |
| ALLEN  |                    22800 |
| WARD   |                    21000 |
| JONES  |                    35700 |
```

```
| MARTIN |                          31800 |
| BLAKE  |                          34200 |
| CLARK  |                          29400 |
| SCOTT  |                          36000 |
| KING   |                          60000 |
| TURNER |                          18000 |
| ADAMS  |                          13200 |
| JAMES  |                          11400 |
| FORD   |                          36000 |
| MILLER |                          15600 |
+--------+--------------------------+
14 rows in set (0.00 sec)
```

12.case...when...then...when...then...else...end

例如：当员工的工作岗位为 MANAGER 的时候，工资上调 10%，
员工的工作岗位为 SALESMAN 的时候，工资上调 50%
(注意，不修改数据库，只是将查询结果显示的工资上调)

```
mysql> select ename,job,sal as 'original sal',case job when 'MANAGER' then sal*1.1
when 'SALESMAN' then sal*1.5 else sal end as 'increase salary'from emp ;
+--------+-----------+-------------+-----------------+
| ename  | job       | original sal | increase salary |
+--------+-----------+-------------+-----------------+
| SMITH  | CLERK     |       800.00 |          800.00 |
| ALLEN  | SALESMAN  |      1600.00 |         2400.00 |
| WARD   | SALESMAN  |      1250.00 |         1875.00 |
| JONES  | MANAGER   |      2975.00 |         3272.50 |
| MARTIN | SALESMAN  |      1250.00 |         1875.00 |
| BLAKE  | MANAGER   |      2850.00 |         3135.00 |
| CLARK  | MANAGER   |      2450.00 |         2695.00 |
| SCOTT  | ANALYST   |      3000.00 |         3000.00 |
| KING   | PRESIDENT |      5000.00 |         5000.00 |
| TURNER | SALESMAN  |      1500.00 |         2250.00 |
| ADAMS  | CLERK     |      1100.00 |         1100.00 |
| JAMES  | CLERK     |       950.00 |          950.00 |
| FORD   | ANALYST   |      3000.00 |         3000.00 |
| MILLER | CLERK     |      1300.00 |         1300.00 |
+--------+-----------+-------------+-----------------+
14 rows in set (0.00 sec)
```

17.分组函数(多行处理函数)

多行函数的特点：输入多行，最终输出一行

常用的分组函数：

1.count 计数

例如，计算有多少个员工

```
mysql> select count(ename) from emp;
+--------------+
| count(ename) |
+--------------+
|           14 |
+--------------+
1 row in set (0.00 sec)
```

2.sum 求和

例如：计算员工的薪资总和

```
mysql> select sum(sal) from emp;
+----------+
| sum(sal) |
+----------+
| 29025.00 |
+----------+
```

```
1 row in set (0.00 sec)
```

3.avg 平均值

例如：计算员工的薪资的平均值

```
mysql> select avg(sal) from emp;
+-------------+
| avg(sal)    |
+-------------+
| 2073.214286 |
+-------------+
1 row in set (0.00 sec)
```

4.max 最大值

例如：计算员工的薪资的最大值

```
mysql> select max(sal) from emp;
+----------+
| max(sal) |
+----------+
|  5000.00 |
+----------+
1 row in set (0.00 sec)
```

5.min 最小值

```
mysql> select min(sal) from emp;
+----------+
| min(sal) |
+----------+
|   800.00 |
+----------+
1 row in set (0.00 sec)
```

例如：计算员工的薪资的最小值

注意：分组函数在使用的时候必须先进行分组，然后才能用。
如果没有对数据进行分组，则整张表默认一个分组

分组函数在使用的过程中，需要注意哪些？

1.分组函数自动忽略 null，不需要提前对 null 进行处理

```
mysql> select sum(comm) from emp;
或
select sum(ifnull(comm,0)) from emp;
+-----------+
| sum(comm) |
+-----------+
|   2200.00 |
+-----------+
1 row in set (0.00 sec)

mysql> select count(comm) from emp;
+-------------+
| count(comm) |
+-------------+
|           4 |
+-------------+
1 row in set (0.00 sec)

mysql> select avg(comm) from emp;
+------------+
| avg(comm)  |
+------------+
| 550.000000 |
+------------+
1 row in set (0.00 sec)
```

2.分组函数中，count(*)和 count(具体字段)有什么区别？

```
mysql> select count(*),count(comm) from emp;
+----------+-------------+
| count(*) | count(comm) |
+----------+-------------+
|       14 |           4 |
+----------+-------------+
1 row in set (0.00 sec)
```

count(具体字段)：表示统计该字段下所有不为 null 的元素总数

count(*)：统计表当中的总行数，只要有一行数据，则 count++

3.分组函数不能直接使用在 where 子句中

例如：找出比最低工资高的员工信息

错误写法：

```
mysql> select ename,sal from emp where sal>min(sal);
ERROR 1111 (HY000): Invalid use of group function
```

分组查询：group by

4.所有的分组函数可以组合起来一起使用

```
mysql> select sum(sal),min(sal),max(sal),avg(sal),count(*) from emp;
+----------+----------+----------+-------------+----------+
| sum(sal) | min(sal) | max(sal) | avg(sal)    | count(*) |
+----------+----------+----------+-------------+----------+
| 29025.00 |   800.00 |  5000.00 | 2073.214286 |       14 |
+----------+----------+----------+-------------+----------+
1 row in set (0.00 sec)
```

19.分组查询(非常重要)

1.什么是分组查询？在实际的应用中，可能有这样的需求，需要先进行分组，然后对每一组数据进行操作

select... from... group by ...

例如：计算每个部门的工资和？

例如：计算每个工作岗位的平均薪资？

例如：计算每个工作岗位的最高薪资？

2.将之前的关键字全部组合在一起，看一下执行顺序？

select... from... where... group by... order by...

以上关键字的顺序不能颠倒，需要记忆。

执行顺序是什么？

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

为什么分组函数不能直接使用在 where 后面？

```
mysql> select ename,sal from emp where sal>min(sal); //爆错
```

因为分组函数在使用的时候，必须先分组之后才能使用。

where 执行的时候，还没有分组，所以 where 后面不能出现分组函数。

```
select sum(sal) from emp;
```

这个没有分组，为啥 sum()函数可以使用呢？

因为 select 在 group by 之后执行

3.案例

找出每个工作岗位的工资和?

实现思路：按照工作岗位分子，然后对分组求和

```
mysql> select job,sum(sal) from emp group by job;
+-----------+----------+
| job       | sum(sal) |
+-----------+----------+
| CLERK     |  4150.00 |
| SALESMAN  |  5600.00 |
| MANAGER   |  8275.00 |
| ANALYST   |  6000.00 |
| PRESIDENT |  5000.00 |
+-----------+----------+
5 rows in set (0.00 sec)
```

以上这个语句的执行顺序：

先从 emp 表中查询数据，根据 job 字段进行分组，然后对每一组的数据进行求和

重点结论：在一条 select 语句中，如果有 group by 语句的话，

select 后面只能跟：参加分组的字段，以及分组函数。

其他的字段一律不能跟。

找出每个部门的最高薪资?

实现思路：按照部门编号分组，然后查找每一组的最大值

```
mysql> select deptno,max(sal) from emp group by deptno;
+--------+----------+
| deptno | max(sal) |
+--------+----------+
|     20 |  3000.00 |
|     30 |  2850.00 |
|     10 |  5000.00 |
+--------+----------+
3 rows in set (0.00 sec)
```

找出每个部门不同岗位的最高薪资?

实现思路：利用多个字段进行分组，查找，排序

```
mysql> select deptno,job,max(sal) from emp group by deptno,job order by deptno,job;
+--------+-----------+----------+
| deptno | job       | max(sal) |
+--------+-----------+----------+
|     10 | CLERK     |  1300.00 |
|     10 | MANAGER   |  2450.00 |
|     10 | PRESIDENT |  5000.00 |
|     20 | ANALYST   |  3000.00 |
|     20 | CLERK     |  1100.00 |
|     20 | MANAGER   |  2975.00 |
|     30 | CLERK     |   950.00 |
|     30 | MANAGER   |  2850.00 |
|     30 | SALESMAN  |  1600.00 |
+--------+-----------+----------+
9 rows in set (0.00 sec)
```

找出每个部门的最高薪资，要求显示最高薪资大于 800 的员工信息?

实现思路：先把大于 800 的都找出来，然后在分组

第一步：求薪资大于 800 的员工

```
mysql> select sal from emp where sal>'800';
+---------+
| sal     |
+---------+
| 1600.00 |
```

```
| 1250.00 |
| 2975.00 |
| 1250.00 |
| 2850.00 |
| 2450.00 |
| 3000.00 |
| 5000.00 |
| 1500.00 |
| 1100.00 |
|  950.00 |
| 3000.00 |
| 1300.00 |
+---------+
13 rows in set (0.00 sec)
```

第二步：将此数据按照 deptno 分组，并求每组的最大值

```
mysql> select deptno,max(sal) from emp where sal>'800' group by deptno order by deptno;
+--------+----------+
| deptno | max(sal) |
+--------+----------+
|     10 |  5000.00 |
|     20 |  3000.00 |
|     30 |  2850.00 |
+--------+----------+
3 rows in set (0.00 sec)
```

找出每个部门的平均薪资，要求显示平均薪资高于 2500 的？

实现思路：使用 group by 结合 having 搭配使用，having 相当于在 group by 的基础上进一步过滤

```
mysql> select deptno,avg(sal) from emp group by deptno having avg(sal)>'2500';
+--------+-------------+
| deptno | avg(sal)    |
+--------+-------------+
|     10 | 2916.666667 |
+--------+-------------+
1 row in set (0.01 sec)
```

优化策略：where 和 having，优先选择使用 where，where 实在完成不了，在选择 having。

因为使用 having 的效率较低

19.大总结(单表查询)

```
select
    ...
from
    ...
where
    ...
group by
    ...
having
    ...
order by
    ...
```

以上关键字的顺序不能颠倒。

执行顺序：

1.from

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

从某张表中查询数据，
先经过 where 条件筛选出有价值的数据
对这些有价值的数据分组，分组之后可以使用 having 继续筛选
select 查询出来
最后排序输出

例如：找出每个岗位的平均薪资，显示平均薪资大于 1500 的，除 MANAGER 之外，要求按照平均薪资降序排列？

```
mysql>
    select
        job,avg(sal)
    from
        emp
    where
        job <> 'MANAGER'
    group by
        job
    having
        avg(sal)>'1500'
    order by
        avg(sal)
    desc;
    +-----------+-------------+
    | job       | avg(sal)    |
    +-----------+-------------+
    | PRESIDENT | 5000.000000 |
    | ANALYST   | 3000.000000 |
    +-----------+-------------+
    2 rows in set (0.01 sec)
```

MySQL day02 课堂笔记

1.把查询结果去除重复记录【distinct】

注意：原表数据不会被删除，只是查询结果去重

去重需要使用一个关键字：distinct

案例：员工工作岗位去重，查看实际工作岗位

```
mysql> select distinct job from emp;
+-----------+
| job       |
+-----------+
| CLERK     |
| SALESMAN  |
| MANAGER   |
| ANALYST   |
| PRESIDENT |
+-----------+
5 rows in set (0.00 sec)
```

// 这样编写是错误的，语法错误

// distinct 只能出现在所有字段的前方

```
mysql> select ename,distinct job from emp;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'distinct job from emp' at line 1
```

// distinct 出现在 job，deptno 两个字段之前，表示两个字段联合起来去重

```
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     |
+--------+-----------+
9 rows in set (0.00 sec)
```

案例：统计一下工作岗位的数量？

```
mysql> select count(distinct job) from emp;
+---------------------+
| count(distinct job) |
+---------------------+
|                   5 |
+---------------------+
1 row in set (0.00 sec)
```

2.连接查询

1.什么是连接查询？

从一张表中单独查询称为单表查询

emp 表和 dept 表联合起来查数据，从 emp 表中取员工名字，从 dept 表中取部门名字

这种跨表查询，多张表联合起来查询数据，被称为多表查询

2.连接查询的分类

根据语法的年代分类：

SQL92：1992 年的时候出现的语法

SQL99：1999 年的时候出现的语法

根据表连接的方式分类：

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

3.当两张表进行连接查询时，没有任何条件的限制，会发生什么现象？

案例：查询每个员工所在部门名称？

```
mysql> select ename,deptno from emp;
+--------+--------+
| ename  | deptno |
+--------+--------+
| SMITH  |     20 |
| ALLEN  |     30 |
| WARD   |     30 |
| JONES  |     20 |
| MARTIN |     30 |
| BLAKE  |     30 |
| CLARK  |     10 |
| SCOTT  |     20 |
| KING   |     10 |
| TURNER |     30 |
| ADAMS  |     20 |
| JAMES  |     30 |
| FORD   |     20 |
| MILLER |     10 |
+--------+--------+
14 rows in set (0.00 sec)

mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)
```

// 两表之间没有任何条件限制

```
mysql> select ename,dname from emp,dept;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | OPERATIONS |
| SMITH  | SALES      |
| SMITH  | RESEARCH   |
| SMITH  | ACCOUNTING |
```

```
| ALLEN  | OPERATIONS |
| ALLEN  | SALES      |
| ALLEN  | RESEARCH   |
| ALLEN  | ACCOUNTING |
.....
56 rows in set (0.00 sec)
14 * 4 = 56
```

// 当两张表进行连接查询，没有任何条件限制的时候，最终查询价格的条数
是两张表条数的乘积，这种现象称为：笛卡尔积现象(笛卡尔发现的一种数学现象)

4.怎么避免笛卡尔积现象？

连接时加条件，满足这个条件的记录被筛选出来

```
mysql> select ename,dname from emp,dept where emp.deptno=dept.deptno;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| JONES  | RESEARCH   |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| CLARK  | ACCOUNTING |
| SCOTT  | RESEARCH   |
| KING   | ACCOUNTING |
| TURNER | SALES      |
| ADAMS  | RESEARCH   |
| JAMES  | SALES      |
| FORD   | RESEARCH   |
| MILLER | ACCOUNTING |
+--------+------------+
14 rows in set (0.00 sec)
```

思考：最终查询的结果条数是 14，但是匹配的过程中，匹配的次数减少了吗？
答案是没有，匹配的次数还是 14*4=56 次，只不过进行了四选一，次数没有减少

// 给表起别名，很重要(SQL92 语法)

```
mysql> select e.ename,d.dname from emp e,dept d where e.deptno=d.deptno;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| JONES  | RESEARCH   |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| CLARK  | ACCOUNTING |
| SCOTT  | RESEARCH   |
| KING   | ACCOUNTING |
| TURNER | SALES      |
| ADAMS  | RESEARCH   |
| JAMES  | SALES      |
| FORD   | RESEARCH   |
| MILLER | ACCOUNTING |
+--------+------------+
14 rows in set (0.00 sec)
```

通过笛卡尔积现象得出，表的连接次数越多效率就越低，尽量避免表的连接次数

5.内连接- 等值连接

查询每个员工所在的部门名称，显示员工名和部门名？

emp e 和 dept d 表进行连接。条件是：e.deptno = d.deptno

SQL92 语法：

```
mysql> select e.ename,d.dname from emp e,dept d where e.deptno=d.deptno;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| JONES  | RESEARCH   |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| CLARK  | ACCOUNTING |
| SCOTT  | RESEARCH   |
| KING   | ACCOUNTING |
| TURNER | SALES      |
| ADAMS  | RESEARCH   |
| JAMES  | SALES      |
| FORD   | RESEARCH   |
| MILLER | ACCOUNTING |
+--------+------------+
14 rows in set (0.00 sec)
```

SQL92 的缺点：结构不清晰，表的连接条件，和后期进一步筛选的条件，都放到了 where 后面

SQL99 语法：

```
// emp e inner join dept d，inner 可以省略，带 inner 可读性更好，一眼就能看出来是内连接
// on e.deptno=d.deptno，条件是等量关系，所以被称为等值连接
mysql> select e.ename,d.dname from emp e inner join dept d on e.deptno=d.deptno;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| JONES  | RESEARCH   |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| CLARK  | ACCOUNTING |
| SCOTT  | RESEARCH   |
| KING   | ACCOUNTING |
| TURNER | SALES      |
| ADAMS  | RESEARCH   |
| JAMES  | SALES      |
| FORD   | RESEARCH   |
| MILLER | ACCOUNTING |
+--------+------------+
14 rows in set (0.00 sec)
```

SQL99 的优点：表连接的条件是独立的，连接之后，

如果还需要进一步的筛选，再往后继续 where

SQL99 语法：

```
select
    ...
```

```
from
    a
join
    b
on
    a 和 b 的连接条件
where
    筛选条件
```

6.内连接- 非等值连接

案例：找出每个员工的薪资等级，要求显示员工名、薪资、薪资等级？

// on e.sal between s.losal and s.hisal，条件不是等值连接，称为非等值连接

mysql> select e.ename,e.sal,s.grade from emp e 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 |
+--------+---------+-------+
14 rows in set (0.00 sec)
```

7.内连接- 自连接

案例：查询员工的上级领导，要求显示员工名和对应的领导名？

// e1.mgr = e2.empno，员工的领导编号 等于 领导的员工编号

mysql> select e1.ename as employee,e2.ename as employer from emp e1 join emp e2 on e1.mgr = e2.empno;

```
+----------+----------+
| employee | employer |
+----------+----------+
| SMITH    | FORD     |
| ALLEN    | BLAKE    |
| WARD     | BLAKE    |
| JONES    | KING     |
| MARTIN   | BLAKE    |
| BLAKE    | KING     |
| CLARK    | KING     |
| SCOTT    | JONES    |
| TURNER   | BLAKE    |
| ADAMS    | SCOTT    |
| JAMES    | BLAKE    |
| FORD     | JONES    |
| MILLER   | CLARK    |
+----------+----------+
13 rows in set (0.00 sec)
```

13 条记录，没有 KING

以上就是内连接中的自连接，自连接的技巧就是：一张表看做两张表

8.外连接

```
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 |
+-------+--------+-----------+------+------------+---------+---------+-----
---+
14 rows in set (0.00 sec)

mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)
```

内连接：(A 和 B 连接，两张表是平等的)

```
SELECT
    e.ename,
    d.dname
FROM
    emp e
        JOIN
    dept d
ON
    e.deptno = d.deptno;// 内连接的特点：将能够匹配上这个条件的数据查询出来
+--------+------------+
| ename  | dname      |
```

```
+--------+------------+
| SMITH  | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| JONES  | RESEARCH   |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| CLARK  | ACCOUNTING |
| SCOTT  | RESEARCH   |
| KING   | ACCOUNTING |
| TURNER | SALES      |
| ADAMS  | RESEARCH   |
| JAMES  | SALES      |
| FORD   | RESEARCH   |
| MILLER | ACCOUNTING |
+--------+------------+
14 rows in set (0.00 sec)
```

外连接(右外连接)：outer 是可以省略的，带上 outer 可读性强

```
// emp e right join dept d，
// right 代表什么：表示将 join 关键字右边的这张表看做主表，
// 主要是为了将这张表的数据全部查询出来，捎带着关联查询左边的表
// 外连接中，两张表分主次关系
mysql> select e.ename,d.dname from emp e right outer join dept d on e.deptno = d.deptno;
+--------+------------+
| ename  | dname      |
+--------+------------+
| MILLER | ACCOUNTING |
| KING   | ACCOUNTING |
| CLARK  | ACCOUNTING |
| FORD   | RESEARCH   |
| ADAMS  | RESEARCH   |
| SCOTT  | RESEARCH   |
| JONES  | RESEARCH   |
| SMITH  | RESEARCH   |
| JAMES  | SALES      |
| TURNER | SALES      |
| BLAKE  | SALES      |
| MARTIN | SALES      |
| WARD   | SALES      |
| ALLEN  | SALES      |
| NULL   | OPERATIONS |
+--------+------------+
15 rows in set (0.00 sec)
```

外连接(左外连接)：

```
mysql> select e.ename,d.dname from dept d left outer join emp e on e.deptno = d.deptno;
+--------+------------+
| ename  | dname      |
+--------+------------+
| MILLER | ACCOUNTING |
| KING   | ACCOUNTING |
| CLARK  | ACCOUNTING |
| FORD   | RESEARCH   |
| ADAMS  | RESEARCH   |
| SCOTT  | RESEARCH   |
| JONES  | RESEARCH   |
```

```
| SMITH  | RESEARCH    |
| JAMES  | SALES       |
| TURNER | SALES       |
| BLAKE  | SALES       |
| MARTIN | SALES       |
| WARD   | SALES       |
| ALLEN  | SALES       |
| NULL   | OPERATIONS  |
+--------+-------------+
15 rows in set (0.00 sec)
```

带有 right 的是右外连接，又叫做右连接；
带有 left 的是左外连接，又叫做左连接；
任何一个右连接都有左连接的写法，任何一个左连接都有右连接的写法

思考：外连接的查询结果条数一定是大于等于内连接的查询结果条数？
正确。

案例：查询每个员工的上级领导，要求显示所有员工的姓名和领导名？

```
mysql> select empno,ename,mgr from emp;
+-------+--------+------+
| 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 |
+-------+--------+------+
14 rows in set (0.00 sec)
```

```
mysql> select e1.ename '员工',e2.ename '领导' from emp e1 left join emp e2 on e1.mgr = e2.empno;
+--------+--------+
| 员工   | 领导   |
+--------+--------+
| SMITH  | FORD   |
| ALLEN  | BLAKE  |
| WARD   | BLAKE  |
| JONES  | KING   |
| MARTIN | BLAKE  |
| BLAKE  | KING   |
| CLARK  | KING   |
| SCOTT  | JONES  |
| KING   | NULL   |
| TURNER | BLAKE  |
| ADAMS  | SCOTT  |
| JAMES  | BLAKE  |
| FORD   | JONES  |
| MILLER | CLARK  |
```

```
+--------+--------+
14 rows in set (0.00 sec)
```

9.多张表如何连接？

语法：

```
select
    ...
from
    a
join
    b
on
    a 和 b 的连接条件
join
    c
on
    a 和 c 的连接条件
right join
    d
on
    a 和 d 的连接条件
```

一条 SQL 中，内连接和外连接可以混合。都可以实现。

案例：找出每个员工的部门名称，以及工资等级，

要求显示员工名、部门名、薪资和薪资等级？

```
mysql> SELECT
            e.ename,
            d.dname,
            e.sal,
            s.grade
        FROM
            emp e
            LEFT JOIN dept d ON e.deptno = d.deptno
            LEFT JOIN salgrade s ON e.sal BETWEEN s.losal
            AND s.hisal;
+--------+------------+---------+-------+
| ename  | dname      | sal     | grade |
+--------+------------+---------+-------+
| SMITH  | RESEARCH   |  800.00 |     1 |
| ALLEN  | SALES      | 1600.00 |     3 |
| WARD   | SALES      | 1250.00 |     2 |
| JONES  | RESEARCH   | 2975.00 |     4 |
| MARTIN | SALES      | 1250.00 |     2 |
| BLAKE  | SALES      | 2850.00 |     4 |
| CLARK  | ACCOUNTING | 2450.00 |     4 |
| SCOTT  | RESEARCH   | 3000.00 |     4 |
| KING   | ACCOUNTING | 5000.00 |     5 |
| TURNER | SALES      | 1500.00 |     3 |
| ADAMS  | RESEARCH   | 1100.00 |     1 |
| JAMES  | SALES      |  950.00 |     1 |
| FORD   | RESEARCH   | 3000.00 |     4 |
| MILLER | ACCOUNTING | 1300.00 |     2 |
+--------+------------+---------+-------+
14 rows in set (0.00 sec)
```

案例：找出每个员工的部门名称，以及工资等级，还有上级领导

要求显示每个员工名、领导名，部门名、薪资和薪资等级？

```
mysql>
    SELECT
```

```
        e1.ename '员工名',
        e2.ename '领导名',
        d.dname '部门名称',
        e1.sal '薪资',
        s.grade '薪资等级'
    FROM
        emp e1
        LEFT JOIN emp e2 ON e1.mgr = e2.empno
        LEFT JOIN dept d ON e1.deptno = d.deptno
        LEFT JOIN salgrade s ON e1.sal BETWEEN s.losal
        AND s.hisal;
+----------+----------+-------------+---------+-------------+
| 员工名   | 领导名   | 部门名称    | 薪资    | 薪资等级    |
+----------+----------+-------------+---------+-------------+
| SMITH    | FORD     | RESEARCH    |  800.00 |           1 |
| ALLEN    | BLAKE    | SALES       | 1600.00 |           3 |
| WARD     | BLAKE    | SALES       | 1250.00 |           2 |
| JONES    | KING     | RESEARCH    | 2975.00 |           4 |
| MARTIN   | BLAKE    | SALES       | 1250.00 |           2 |
| BLAKE    | KING     | SALES       | 2850.00 |           4 |
| CLARK    | KING     | ACCOUNTING  | 2450.00 |           4 |
| SCOTT    | JONES    | RESEARCH    | 3000.00 |           4 |
| KING     | NULL     | ACCOUNTING  | 5000.00 |           5 |
| TURNER   | BLAKE    | SALES       | 1500.00 |           3 |
| ADAMS    | SCOTT    | RESEARCH    | 1100.00 |           1 |
| JAMES    | BLAKE    | SALES       |  950.00 |           1 |
| FORD     | JONES    | RESEARCH    | 3000.00 |           4 |
| MILLER   | CLARK    | ACCOUNTING  | 1300.00 |           2 |
+----------+----------+-------------+---------+-------------+
14 rows in set (0.00 sec)
```

3.子查询

1.什么是子查询?

select 语句中嵌套 select 语句，被嵌套的 select 语句称为子查询。

2.子查询都可以出现在哪里呢?

```
select
    ..(select).
from
    ..(select).
where
    ..(select).
```

案例：找出比最低工资高的员工姓名?

实现思路：

第一步：查询最低工资是多少?

```
mysql> select min(sal) from emp;
+----------+
| min(sal) |
+----------+
|   800.00 |
+----------+
1 row in set (0.00 sec)
```

第二步：找出大于 800 的

```
mysql> select ename,sal from emp where sal>'800';
+--------+---------+
| ename  | sal     |
+--------+---------+
| 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 |
+--------+---------+
13 rows in set (0.00 sec)
```

第三步：合并前 2 条语句

```
mysql> SELECT ename, sal FROM emp WHERE sal >( SELECT min( sal ) FROM emp );
+--------+---------+
| ename   | sal       |
+--------+---------+
| 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 |
+--------+---------+
13 rows in set (0.00 sec)
```

3.from 子句中的子查询

注意：from 后面的子查询，可以将子查询的查询结果当做一张临时表。(技巧)

案例：找出每个岗位的平均工资的薪资等级？

第一步：找出每个岗位的平均工资(按照岗位分组求平均值)

```
mysql> select job,avg(sal) from   emp group by job; // t 表
+-----------+-------------+
| job          | avg(sal)      |
+-----------+-------------+
| CLERK       | 1037.500000 |
| SALESMAN   | 1400.000000 |
| MANAGER    | 2758.333333 |
| ANALYST    | 3000.000000 |
| PRESIDENT | 5000.000000 |
+-----------+-------------+
5 rows in set (0.00 sec)
```

第二步：把以上的查询结果作为真实存在的表 t

```
mysql> select * from salgrade; // s 表
+-------+-------+-------+
| GRADE | LOSAL | HISAL |
+-------+-------+-------+
|     1 |   700 |  1200 |
|     2 |  1201 |  1400 |
|     3 |  1401 |  2000 |
```

```
|      4 |  2001 |  3000 |
|      5 |  3001 |  9999 |
+-------+-------+-------+
5 rows in set (0.00 sec)
```

将 t 表和 s 表进行连接，条件是：t 表中的 avg(sal) between s.losal and s.hisal;

```
mysql>
    SELECT
        t.job,
        t.avgSal,
        s.grade
    FROM
        ( SELECT job, avg( sal ) AS avgSal FROM emp GROUP BY job ) AS t
        LEFT JOIN salgrade AS s ON t.avgSal BETWEEN s.losal
        AND s.hisal;
+-----------+-------------+-------+
| job       | avgSal      | grade |
+-----------+-------------+-------+
| CLERK     | 1037.500000 |     1 |
| SALESMAN  | 1400.000000 |     2 |
| MANAGER   | 2758.333333 |     4 |
| ANALYST   | 3000.000000 |     4 |
| PRESIDENT | 5000.000000 |     5 |
+-----------+-------------+-------+
5 rows in set (0.00 sec)
```

4.select 后面出现的子查询(这个内容不需要掌握，了解即可)

案例：找出每个员工的部门名称？要求显示员工名和部门名？

```
mysql>
    SELECT e.ename,( SELECT d.dname FROM dept d WHERE e.deptno = d.deptno ) AS dname
    FROM
        emp e;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| JONES  | RESEARCH   |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| CLARK  | ACCOUNTING |
| SCOTT  | RESEARCH   |
| KING   | ACCOUNTING |
| TURNER | SALES      |
| ADAMS  | RESEARCH   |
| JAMES  | SALES      |
| FORD   | RESEARCH   |
| MILLER | ACCOUNTING |
+--------+------------+
14 rows in set (0.00 sec)

// 错误案例
// mysql> select e.ename,(select d.dname from dept d) as dname from emp e;
// ERROR 1242 (21000): Subquery returns more than 1 row
```

注意：对于 select 后面的子查询来说，这个子查询只能一次返回 1 条结果

大于 1 条就会报错

4.union 合并查询结果集合

案例：查询工作岗位是 MANAGER 或 SALESMAN 的员工？

```
mysql> select * from emp where job in ('MANAGER','SALESMAN');
+-------+--------+----------+------+------------+---------+---------+------
--+
| EMPNO | ENAME  | JOB      | MGR  | HIREDATE   | SAL     | COMM    |
DEPTNO |
+-------+--------+----------+------+------------+---------+---------+------
--+
|  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
|
|  7844 | TURNER | SALESMAN | 7698 | 1981-09-08 | 1500.00 |    0.00 |      30 |
+-------+--------+----------+------+------------+---------+---------+------
--+
7 rows in set (0.00 sec)

mysql> select * from emp where job = 'MANAGER' or job = 'SALESMAN';
+-------+--------+----------+------+------------+---------+---------+------
--+
| EMPNO | ENAME  | JOB      | MGR  | HIREDATE   | SAL     | COMM    |
DEPTNO |
+-------+--------+----------+------+------------+---------+---------+------
--+
|  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
|
|  7844 | TURNER | SALESMAN | 7698 | 1981-09-08 | 1500.00 |    0.00 |      30 |
+-------+--------+----------+------+------------+---------+---------+------
--+
7 rows in set (0.00 sec)
```

使用 union 进行合并结果查询

```
mysql> select * from emp where job='MANAGER' union select * from emp where
job = 'SALESMAN';
+-------+--------+----------+------+------------+---------+---------+------
--+
| EMPNO | ENAME  | JOB      | MGR  | HIREDATE   | SAL     | COMM    |
DEPTNO |
+-------+--------+----------+------+------------+---------+---------+------
--+
|  7566 | JONES  | MANAGER  | 7839 | 1981-04-02 | 2975.00 |    NULL |      20
|
|  7698 | BLAKE  | MANAGER  | 7839 | 1981-05-01 | 2850.00 |    NULL |      30
|
|  7782 | CLARK  | MANAGER  | 7839 | 1981-06-09 | 2450.00 |    NULL |      10
```

```
|
        |  7499 | ALLEN  | SALESMAN | 7698 | 1981-02-20 | 1600.00 |  300.00 |      30 |
        |  7521 | WARD   | SALESMAN | 7698 | 1981-02-22 | 1250.00 |  500.00 |      30 |
        |  7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00 | 1400.00 |      30 |
        |  7844 | TURNER | SALESMAN | 7698 | 1981-09-08 | 1500.00 |    0.00 |      30 |
        +-------+--------+----------+------+------------+---------+---------+------
--+
        7 rows in set (0.01 sec)
```

union 的效率要高一些。对于表连接来说，每连接一次新表，
则匹配的次数满足笛卡尔积，成倍增加匹配次数。
但是 union 可以减少匹配次数，在减少匹配次数的情况下，
还可以完成两个结果集的拼接。

a 连接 b 连接 c
a 10 条记录
b 10 条记录
c 10 条记录
匹配次数是：10 * 10 * 10 -> 1000 次

a 连接 b 的一个结果是：10 * 10 -> 100 次
a 连接 c 的一个结果是：10 * 10 -> 100 次
使用 union 的话是：100 + 100 -> 200 次(把乘法变成加法运算)

union 在使用的时候的注意事项？

```
// 错误的：union 在进行结果拼接的时候，要求两个结果集的列数相同
mysql> select ename,job from emp where job='MANAGER' union select ename
from emp where job = 'SALESMAN';
ERROR 1222 (21000): The used SELECT statements have a different number of
columns
// MySQL 可以，Oracle 语法严格，不可以。报错
// 要求结果集合并时列和列的数据类型也要是一样
mysql> select ename,job from emp where job='MANAGER' union select ename
from emp,sal where job = 'SALESMAN';
ERROR 1146 (42S02): Table 'bjpowernode.sal' doesn't exist
```

结论：合并的两个结果集的列数相同、数据类型相同

5.limit(非常重要)

1.limit 是将查询结果集的一部分取出来，通常使用在分页查询当中
百度默认：一页显示 10 条搜索记录
分页的作用：为了提高用户体验，因为一次全部查出来，用户体验差
可以一页一页的翻。

2.limit 怎么使用？
完整用法：limit startIndex,length
startIndex 是起始下标，length 是截取长度
起始下标从 0 开始

案例：按照薪资降序，取出排名在前五的员工信息？

```
mysql> select ename,sal from emp order by sal desc limit 5;
或
mysql> select ename,sal from emp order by sal desc limit 0,5;
+-------+---------+
| ename | sal     |
+-------+---------+
| KING  | 5000.00 |
```

```
| SCOTT | 3000.00 |
| FORD  | 3000.00 |
| JONES | 2975.00 |
| BLAKE | 2850.00 |
+-------+---------+
5 rows in set (0.00 sec)
```

3.注意：MySQL 当中，limit 是在 order by 之后执行。

案例：取工资排名在 3- 5 名的员工？

```
//limit 2,3
// 其中，2 表示起始位置从 2 下标开始，就是第三条记录
// 3 表示截取的长度 3- 5 名员工，共有 3 个

mysql> select ename,sal from emp order by sal desc limit 2,3;
+-------+---------+
| ename | sal     |
+-------+---------+
| FORD  | 3000.00 |
| JONES | 2975.00 |
| BLAKE | 2850.00 |
+-------+---------+
3 rows in set (0.00 sec)
```

案例：取出工资排名在[5,9]名的员工？

```
mysql> select ename,sal from emp order by sal desc limit 4,5;
+--------+---------+
| ename  | sal     |
+--------+---------+
| BLAKE  | 2850.00 |
| CLARK  | 2450.00 |
| ALLEN  | 1600.00 |
| TURNER | 1500.00 |
| MILLER | 1300.00 |
+--------+---------+
5 rows in set (0.00 sec)
```

案例：分页

每页显示 3 条记录

第 1 页 limit 0,3; [0 1 2]

第 2 页 limit 3,3; [3 4 5]

第 3 页 limit 6,3; [6 7 8]

第 4 页 limit 9,3; [9 10 11]

每页显示 pageSize 条记录

第 pageNo 页：limit (pageNo - 1) * pageSize, pageSize;

```
public static void main(String[] void){
    int  pageNo = 5; // 页码
    int pageSize = 10; // 每页显示 pageSize 条记录

    // 计算开始索引位置
    int startIndex = (pageNo - 1) * pageSize;
    // 拼接 SQL 语句
    String sql = "select xxx from xxx order by xxx" + startIndex + "," +
pageSize;
}
```

公式：limit (pageNo - 1) * pageSize, pageSize;

4.关于 DQL 语句的大总结

```
select
    ...
from
    ...
group by
    ...
having
    ...
order by
    ...
limit
    ...
```

执行顺序?

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

6.表的创建(建表)

1.建表的语法格式：(建表属于 DDL 语句，DDL 包括：create drop alter)

```
create table 表名(字段名 1 数据类型,字段名 2 数据类型,字段名 3 数据类型,...);
create table 表名(
    字段名 1 数据类型,
    字段名 2 数据类型,
    字段名 3 数据类型
);
```

表名：建议以 t_或者 tbl_开始，可读性强，见名知意
字段名：见名知意
表名和字段名都属于标识符

2.关于 MySQL 中的数据类型

很多数据类型，我们只需要掌握一些常见的数据类型即可。

varchar
可变长度的字符串(variable char)
最长 255 个字符长度
比较智能，节省空间
会根据实际的数据长度动态分配空间

优点：节省空间
缺点：需要动态分配空间，速度慢，效率低

char
定长的字符串
最长 255 个字符长度
不管实际的数据长度是多少，分配固定长度的空间去存储数据
使用不恰当的时候，可能会导致空间的浪费

优点：不需要动态分配空间，速度快，效率高
缺点：使用不当可能会导致空间的浪费

varchar 和 char 应该如何选择？
性别字段：char。因为性别是固定长度的字符串，所以选择 char
姓名字段：varchar。每个人的名字长度不一样，所以选择 varchar

int
最长 11 位
数字中的整数型，等同于 java 中的 int。

bigint
数字中的整数型，等同于 java 中的 long。

float
单精度浮点型数据

double
双精度浮点型数据

data
短日期类型

datatime
长日期类型

clob
字符大对象
最多可以存储 4G 的字符串
比如：存储一篇文章，存储一个说明
超过 255 个字符的，都要采用字符大对象来存储
Character Large OBject:CLOB

blob
二进制大对象
专门用来存储图片、声音、视频等流媒体数据
往 BLOB 类型的字段值插入数据的时候，例如插入一张图片、一个视频等
需要使用 IO 流才行
Binary Large OBject:BLOB

案例：创建一个电影表
t_movie　电影表 (专门用来存储电影信息的表)
no(bigint):电影编号,10000,10001
name(varchar):电影名字,'哪吒','林正英之娘娘'
history(clob):故事情节,'','
playtime(data):上映日期,2019-10-11,2019-11-11
time(double):时长,2.5,1.5
image(blob):电影宣传海报,
type(char):类型,'1','2'

3.创建学生表
学号、姓名、年龄、性别、邮箱地址
```
create table t_student(
    no int,
    name varchar(32),
    sex char(1),
    age int(3),
    email varchar(255)
```

```
);
```

删除表：

```
drop table t_student; // 当这张表不存在的时候会报错
// 如果这张表存在的话，删除
drop table if exists t_student;
```

4.插入数据 insert(DML 语句)

语法格式：

```
insert into 表名(字段名 1 数据类型,字段名 2 数据类型,字段名 3 数据类型,...)
values(值 1，值 2，值 3);
```

注意：字段名和值要一一对应，数量要对应，数据类型要对应。

```
insert into t_student(no,name,sex,age,email)
values(1,'zhangsan','m',20,'zhangsan01@123.com');
insert into t_student(email,name,sex,age,no) values('lisi@123.com','lisi','f',20,2);

mysql> insert into t_student(no) values(3);
Query OK, 1 row affected (0.00 sec)

mysql> select * from t_student;
+------+----------+------+------+--------------------+
| no   | name     | sex  | age  | email              |
+------+----------+------+------+--------------------+
|    1 | zhangsan | m    |   20 | zhangsan01@123.com |
|    2 | lisi     | f    |   20 | lisi@123.com       |
|    3 | NULL     | NULL | NULL | NULL               |
+------+----------+------+------+--------------------+
3 rows in set (0.00 sec)

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

mysql> select * from t_student;
+------+----------+------+------+--------------------+
| no   | name     | sex  | age  | email              |
+------+----------+------+------+--------------------+
|    1 | zhangsan | m    |   20 | zhangsan01@123.com |
|    2 | lisi     | f    |   20 | lisi@123.com       |
|    3 | NULL     | NULL | NULL | NULL               |
| NULL | wangwu   | NULL | NULL | NULL               |
+------+----------+------+------+--------------------+
4 rows in set (0.00 sec)
```

注意：inert 语句但凡是执行成功了，那么必然会多一条记录。

没有给其他字段指定值的话，默认值为 null。

```
drop table if exists t_student;
create table t_student(
    no int,
    name varchar(32),
    sex char(1) default 'm', // 指定默认值
    age int(3),
    email varchar(255)
);

mysql> desc t_student;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
```

```
| no    | int          | YES  |     | NULL    |       |
| name  | varchar(32)  | YES  |     | NULL    |       |
| sex   | char(1)      | YES  |     | m       |       |默认值
| age   | int          | YES  |     | NULL    |       |
| email | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> insert into t_student(no) values(1);
Query OK, 1 row affected (0.01 sec)
mysql> select * from t_student;
+------+------+------+------+-------+
| no   | name | sex  | age  | email |
+------+------+------+------+-------+
|    1 | NULL | m    | NULL | NULL  |
+------+------+------+------+-------+
1 row in set (0.00 sec)
```

insert 语句中的"字段名"可以省略吗？可以

```
insert into t_student values(2); // 错误的
// 注意：前面的字段名省略的话，等于都写上了。所以值也要按照字段顺序都写上
insert into t_student values(2,'lisi','f',20,'lisi@123.com');

mysql> insert into t_student values(2,'lisi','f',20,'lisi@123.com');
Query OK, 1 row affected (0.00 sec)

mysql> select * from t_student;
+------+------+------+------+--------------+
| no   | name | sex  | age  | email        |
+------+------+------+------+--------------+
|    1 | NULL | m    | NULL | NULL         |
|    2 | lisi | f    |   20 | lisi@123.com |
+------+------+------+------+--------------+
2 rows in set (0.00 sec)
```

5.insert 插入日期

数字格式化：format

```
mysql> select ename,sal from emp;
+--------+---------+
| 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 |
+--------+---------+
14 rows in set (0.00 sec)
```

格式化数字：format(数字，'格式') 加入千分位

```
mysql> select ename,format(sal,'$999,999') as sal from emp;
+--------+-------+
| ename  | sal   |
+--------+-------+
| SMITH  | 800   |
| ALLEN  | 1,600 |
| WARD   | 1,250 |
| JONES  | 2,975 |
| MARTIN | 1,250 |
| BLAKE  | 2,850 |
| CLARK  | 2,450 |
| SCOTT  | 3,000 |
| KING   | 5,000 |
| TURNER | 1,500 |
| ADAMS  | 1,100 |
| JAMES  | 950   |
| FORD   | 3,000 |
| MILLER | 1,300 |
+--------+-------+
14 rows in set, 14 warnings (0.00 sec)
```

str_to_date:将字符串 varchar 类型转换成 date 类型
date_format：将 data 类型转换为具有一定格式的 varchar 字符串类型

```
drop table if exists t_user;
create table t_user(
    id int,
    name varchar(32),
    birth date
);

create table t_user(
id int,
name varchar(32),
birth varchar(10)
);
```

注意：数据库中的一条命名规范：
所有的标识符都是全部小写，单词和单词之间使用下划线进行衔接

```
mysql> drop table if exists t_user;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> create table t_user(
    -> id int,
    -> name varchar(32),
    -> birth date
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> select * from t_user;
Empty set (0.00 sec)

mysql> desc t_user;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int         | YES  |     | NULL    |       |
| name  | varchar(32) | YES  |     | NULL    |       |
```

```
| birth | date          | YES  |     | NULL    |       |
+-------+---------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

插入数据？

```
// 错误的，类型不匹配
// birth 数据类型是 date
// 插入的数据'01-10-1990' 是 varchar 类型
insert into t_user(id,name,birth) values(1,'zhangsan','01-10-1990'); // 1990 年 10 月 1 日
```

怎么办？可以实验 str_to_date 函数可以将字符串转换成日期类型 date？

语法格式：

```
str_to_date('字符串日期','日期格式')
```

MySQL 的日期格式：

```
%Y 年
%m 月
%d 日
%h 时
%i 分
%s 秒
```

```
insert into t_user(id,name,birth) values(1,'zhangsan',str_to_date('01-10-1990','%d-%m-%Y'));
```

```
mysql> insert into t_user(id,name,birth) values(1,'zhangsan',str_to_date('01-10-1990','%d-%m-%Y'));
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 1990-10-01 |
+------+----------+------------+
1 row in set (0.00 sec)
```

str_to_date 函数可以把字符串 varchar 转换成日期 date 类型，
通常使用在插入 insert 方面，因为插入的时候需要一个日期类型的数据，
需要通过该函数将字符串转换成 date

如果提供的日期字符串是这个格式，str_to_date 函数就不需要了，就被自动转换成对应的 date 格式了

```
%Y-%m-%d
insert into t_user(id,name,birth) values(2,'lisi','1990-10-01');
```

```
mysql> insert into t_user(id,name,birth) values(2,'lisi','1990-10-01');
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 1990-10-01 |
|    2 | lisi     | 1990-10-01 |
+------+----------+------------+
```

2 rows in set (0.00 sec)

查询的时候，可以以某个特定的日期格式展示吗？

date_format：这个函数可以将日期类型的数据转换成特定格式的字符串。

```
mysql> select * from t_user;
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 1990-10-01 |
|    2 | lisi     | 1990-10-01 |
+------+----------+------------+
2 rows in set (0.00 sec)

mysql> select id,name,date_format(birth,'%m/%d/%Y') birth  from t_user;
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 10/01/1990 |
|    2 | lisi     | 10/01/1990 |
+------+----------+------------+
2 rows in set (0.00 sec)
```

date_format 函数怎么用？

date_format(日期类型数据,'日期格式')

这个函数通常使用在查询日期方面， 设置展示的日期格式

```
mysql> select * from t_user;
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 1990-10-01 |
|    2 | lisi     | 1990-10-01 |
+------+----------+------------+
2 rows in set (0.00 sec)
```

以上的 SQL 语句实际上式进行了默认的日期格式，

自动将数据库中的 date 类型数据转换成 varchar

并且采用的格式是 MySQL 默认的日期格式：'%Y-%m-%d'

java 中的数据格式？

yyyy-MM-dd HH:mm:ss SSS

6.date 和 datetime 两个类型的区别？

date 是短日期：只包括年月日信息

datetime 是长日期：包括年月日时分秒信息

```
drop table if exists t_user;
create table t_user(
    id int,
    name varchar(32),
    birth date,
    create_time datetime
);
```

id 是整数

name 是字符串

birth 是短日期

create_time 是这条记录的创建时间：长日期类型

MySQL 短日期的默认格式：%Y-%m-%d
MySQL 长日期的默认格式：%Y-%m-%d %h:%i:%s

插入数据：

```
insert into t_user(id,name,birth,create_time) values(1,'zhangsan','1990-10-01','2020-03-18 15:49:50');
```

```
mysql> drop table if exists t_user;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> create table t_user(
    -> id int,
    -> name varchar(32),
    -> birth date,
    -> create_time datetime
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> insert into t_user(id,name,birth,create_time) values(1,'zhangsan','1990-10-01','2020-03-18 15:49:50');
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+------+----------+------------+---------------------+
| id   | name     | birth      | create_time         |
+------+----------+------------+---------------------+
|    1 | zhangsan | 1990-10-01 | 2020-03-18 15:49:50 |
+------+----------+------------+---------------------+
1 row in set (0.00 sec)
```

一次可以插入多条记录？

语法格式：insert into t_user(id,name,birth,create_time) values(),(),(),()...;

```
mysql>
insert into t_user(id,name,birth,create_time) values(1,'zhangsan','1980-10-01',now()),
    (2,'lisi','1982-08-01',now()),
    (3,'wangwu','1980-02-01',now())
    ;
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from t_user;
+------+----------+------------+---------------------+
| id   | name     | birth      | create_time         |
+------+----------+------------+---------------------+
|    1 | zhangsan | 1980-10-01 | 2021-05-11 15:29:01 |
|    2 | lisi     | 1982-08-01 | 2021-05-11 15:29:01 |
|    3 | wangwu   | 1980-02-01 | 2021-05-11 15:29:01 |
+------+----------+------------+---------------------+
3 rows in set (0.00 sec)
```

在 MySQL 中如何获取系统的当前时间？

now() 函数，并且获取的时间带有时分秒信息。是 datetime 类型的数据

```
insert into t_user(id,name,birth,create_time) values(2,'lisi','1990-11-11',now());

mysql> insert into t_user(id,name,birth,create_time) values(2,'lisi','1990-11-11',now());
```

```
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+------+----------+------------+---------------------+
| id   | name     | birth      | create_time         |
+------+----------+------------+---------------------+
|    1 | zhangsan | 1990-10-01 | 2020-03-18 15:49:50 |
|    2 | lisi     | 1990-11-11 | 2021-05-11 13:26:11 |
+------+----------+------------+---------------------+
2 rows in set (0.00 sec)
```

7.修改(update) (DML 语句)

语法格式：

update 表名 set 字段名 1=值 1，字段名 2=值 2，字段名 3=值 3... hwere 条件;

注意：没有条件限制，会导致所有数据的更新

```
// 修改 id 为 2 的那一行数据的 name 和 birth
mysql> update t_user set name='Jack',birth = '2000-10-01' where id='2';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from t_user;
+------+----------+------------+---------------------+
| id   | name     | birth      | create_time         |
+------+----------+------------+---------------------+
|    1 | zhangsan | 1990-10-01 | 2020-03-18 15:49:50 |
|    2 | Jack     | 2000-10-01 | 2021-05-11 13:26:11 |
+------+----------+------------+---------------------+
2 rows in set (0.00 sec)

// 修改 id 为 2 的那一行数据的 name、birth，将 create_time 改为当前时间 now()
mysql> update t_user set name='Jack',birth = '2000-10-01',create_time = now() where id = '2';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from t_user;
+------+----------+------------+---------------------+
| id   | name     | birth      | create_time         |
+------+----------+------------+---------------------+
|    1 | zhangsan | 1990-10-01 | 2020-03-18 15:49:50 |
|    2 | Jack     | 2000-10-01 | 2021-05-11 13:33:54 |
+------+----------+------------+---------------------+
2 rows in set (0.00 sec)

// 不加 where 限制条件，则更新所有
mysql> update t_user set name = 'abc';
Query OK, 2 rows affected (0.02 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from t_user;
+------+------+------------+---------------------+
| id   | name | birth      | create_time         |
+------+------+------------+---------------------+
|    1 | abc  | 1990-10-01 | 2020-03-18 15:49:50 |
|    2 | abc  | 2000-10-01 | 2021-05-11 13:33:54 |
+------+------+------------+---------------------+
2 rows in set (0.00 sec)
```

8.删除数据 delete(DML 语句)

语法格式：delete from 表名 where 条件;

注意：没有条件，整张表的数据都会被删除

```
// 删除 t_user 表中 id=2 的数据
mysql> delete from t_user where id='2';
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+------+------+------------+---------------------+
| id   | name | birth      | create_time         |
+------+------+------------+---------------------+
|    1 | abc  | 1990-10-01 | 2020-03-18 15:49:50 |
+------+------+------------+---------------------+
1 row in set (0.00 sec)

// 不加 where 限制条件，清空整张表格
mysql> insert into t_user(id) values(2);
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+------+------+------------+---------------------+
| id   | name | birth      | create_time         |
+------+------+------------+---------------------+
|    1 | abc  | 1990-10-01 | 2020-03-18 15:49:50 |
|    2 | NULL | NULL       | NULL                |
+------+------+------------+---------------------+
2 rows in set (0.00 sec)

mysql> delete from t_user;
Query OK, 2 rows affected (0.01 sec)

mysql> select * from t_user;
Empty set (0.00 sec)
```

增删改查：insert delete update select

insert 语法格式：

insert into 表名(字段名 1 数据类型,字段名 2 数据类型,字段名 3 数据类型,...) values(值 1，值 2，值 3);

delete 语法格式：

delete from 表名 where 条件;

update 语法格式：

update 表名 set 字段名 1=值 1，字段名 2=值 2，字段名 3=值 3... where 条件;

MySQL day03 课堂笔记

1.快速创建表？

```
mysql> show tables;
+----------------------+
| Tables_in_bjpowernode |
+----------------------+
| dept                  |
| emp                   |
| salgrade              |
| t_student             |
| t_user                |
+----------------------+
5 rows in set (0.00 sec)

// 将查询的结果复制一份
mysql> create table emp2 as (select * from emp);
Query OK, 14 rows affected, 2 warnings (0.02 sec)
Records: 14  Duplicates: 0  Warnings: 2

mysql> show tables;
+----------------------+
| Tables_in_bjpowernode |
+----------------------+
| dept                  |
| emp                   |
| emp2                  |
| salgrade              |
| t_student             |
| t_user                |
+----------------------+
6 rows in set (0.00 sec)
```

原理：

将一个查询结果当做一张表新建

可以完成表的快速复制

表创建出来，同时表中的数据也存在了

```
mysql> create table mytable as (select empno,ename from emp where job = 'MANAGER');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from mytable;
+-------+-------+
| empno | ename |
+-------+-------+
|  7566 | JONES |
|  7698 | BLAKE |
|  7782 | CLARK |
+-------+-------+
3 rows in set (0.00 sec)
```

2.将查询结果插入到一张表当中？insert 相关的

```
// 如果存在，删除一张表
mysql> drop table if exists dept_cpy;
Query OK, 0 rows affected, 1 warning (0.01 sec)

// 创建一张表。将查询结果当作一张表新建
```

```
mysql> create table dept_cpy as select * from dept;
Query OK, 4 rows affected (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 0

// 显示新建的表
mysql> show tables;
+----------------------+
| Tables_in_bjpowernode |
+----------------------+
| dept                  |
| dept_cpy              |
| emp                   |
| emp2                  |
| mytable               |
| salgrade              |
| t_student             |
| t_user                |
+----------------------+
8 rows in set (0.00 sec)

// 查看新建表的内容
mysql> select * from dept_cpy;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)

// 将查询的结果插入到新建表中，追加。该方法很少用
mysql> insert into dept_cpy select * from dept;
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from dept_cpy;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
8 rows in set (0.00 sec)
```

3.快速删除表中的数据？

```
// 删除 dept_cpy 表中的数据
mysql> delete from dept_cpy; //这种删除数据的方式比较慢
Query OK, 12 rows affected (0.01 sec)
```

delete 语句删除数据的原理：

表中的数据被删除，但这个数据在磁盘上的真实存储空间不会被释放

这种删除的缺点：删除效率比较低
这种删除的优点：支持回滚，后悔了可以在使用回滚恢复数据。rollback 命令

delete 案例：

```
mysql> select * from dept_cpy;
Empty set (0.00 sec)

// 追加数据
mysql> insert into dept_cpy select * from dept;
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

// 查看数据
mysql> select * from dept_cpy;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

// 删除数据
mysql> delete from dept_cpy;
Query OK, 4 rows affected (0.00 sec)

// 查看数据，为空
mysql> select * from dept_cpy;
Empty set (0.00 sec)

// 回滚
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

// 数据恢复
mysql> select * from dept_cpy;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)
```

truncate 删除数据的原理：
这种删除效率比较高，表被一次阶段，物理删除
这种删除的缺点：不支持回滚
这种删除的优点：删除效率比较高

truncate 案例：

```
// 查询数据
```

```
mysql> select * from dept_cpy;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)

// 删除表中数据，清空表
mysql> truncate table dept_cpy;
Query OK, 0 rows affected (0.03 sec)

// 回滚
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

// 回滚之后无法表中恢复数据
mysql> select * from dept_cpy;
Empty set (0.00 sec)
```

用法：truncate table tept_cpy;(这种操作属于 DDL 操作)

大表非常大，上一条记录？

删除的时候，使用 delete，也许需要执行 1 个小时才能删除完，效率较低

可以选择使用 truncate 删除表中数据，只需要不到 1 秒钟就可以全部删完，效率较高。

但是使用 truncate 之前，必须仔细询问客户是否真要删除，并警告删除之后无法恢复。

删除表操作？

drop table if exists 表名;

4.对表结构的增删改

什么是对表结构的修改？

添加一个字段，删除一个字段，或者修改一个字段

对表结构的修改，需要使用：alter（属于 DDL 语句）

DDL 语句包括：create drop alter

第一：在实际开发中，需求确定，表一旦设计好，很少进行表结构的修改

因为开发进行中，修改表结构，成本比较高

修改表的结构，对应的程序代码就需要进行大量的修改，成本比较高

这个责任应该由设计人员承担

第二：由于修改表结构的操作很少。如果需要修改表的结构，可以使用工具。例如 navicat

5.约束

1.什么是约束？

约束的英文单词：constraint

在创建表的时候，可以表中的字段加上一些约束，来保证这个表中的数据完整性、有效性。

2.约束包括哪些？

非空约束：not null
唯一性约束：unique
主键约束：primary key (简称 PK)
外键约束：foreign key (简称 FK)
检查约束：check (MySQL 不支持，Oracle 支持)

重点学习 4 个约束：
    not null
    unique
    primary key
    foreign key

3.非空约束：not null

非空约束 not null 约束的字段不能为 null

```
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255) not null
);

insert into t_vip(id,name) values(1,'zhangsan');
insert into t_vip(id,name) values(2,'lisi');
insert into t_vip(id) values(3);
```

xx.sql 这种文件被称为 sql 脚本文件。
sql 脚本文件中编写了大量的 sql 语句。
执行 sql 脚本文件的时候，可以使用 sql 脚本文件。
在 MySQL 当中是如何执行 sql 脚本的呢？
    source sql 脚本的绝对路径

执行 sql 脚本文件，数据库数据就有了

```
// 因为 name 不能为空，所以只给 id 添加了数据而没有给 name 添加数据就会报错
mysql> insert into t_vip(id) values(3);
ERROR 1364 (HY000): Field 'name' doesn't have a default value
```

4.唯一性约束：unique

唯一性约束 unique 约束的字段不能重复，但是可以为 null

```
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255) unique,
    email varchar(255)
);

insert into t_vip(id,name,email) values
(1,'zhangsan','zhangsan@123.com'),
(2,'lisi','lisi@123.com'),
(3,'wangwu','wangwu@123.com');

select * from t_vip;

mysql> select * from t_vip;
+------+----------+-----------------+
| id   | name     | email           |
+------+----------+-----------------+
|    1 | zhangsan | zhangsan@123.com |
```

```
|      2 | lisi     | lisi@123.com     |
|      3 | wangwu   | wangwu@123.com   |
+------+----------+-----------------+
3 rows in set (0.00 sec)

// 再插入 name 为'zhangsan'的信息，报错：'zhangsan'重复了
mysql> insert into t_vip values(4,'zhangsan','zhangsan1@qq.com');
ERROR 1062 (23000): Duplicate entry 'zhangsan' for key 't_vip.name'

// 在 unique 约束下的字段中，null 是可以重复的
mysql> insert into t_vip values(5,null,'null1@qq.com');
Query OK, 1 row affected (0.01 sec)

mysql> insert into t_vip values(6,null,'null2@qq.com');
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_vip;
+------+----------+-----------------+
| id   | name     | email           |
+------+----------+-----------------+
|      1 | zhangsan | zhangsan@123.com |
|      2 | lisi     | lisi@123.com     |
|      3 | wangwu   | wangwu@123.com   |
|      5 | NULL     | null1@qq.com     |
|      6 | NULL     | null2@qq.com     |
+------+----------+-----------------+
5 rows in set (0.00 sec)

// null 可以重复
mysql> insert into t_vip(id) values(5),(6);
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from t_vip;
+------+----------+-----------------+
| id   | name     | email           |
+------+----------+-----------------+
|      1 | zhangsan | zhangsan@123.com |
|      2 | lisi     | lisi@123.com     |
|      3 | wangwu   | wangwu@123.com   |
|      5 | NULL     | NULL             |
|      6 | NULL     | NULL             |
+------+----------+-----------------+
5 rows in set (0.00 sec)
```

name 字段虽然被唯一性 unique 约束，但是可以为 null。

新需求：name 和 email 联合起来使用唯一性约束。也就是多个字段的唯一性约束

```
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255) unique, // 约束添加在列的后面，称为列级约束
    email varchar(255) unique
);
```

这张表这样创建是不符合新需求的

这样创建表示 name 具有唯一性，email 具有唯一性，各自具有唯一性

以下这样的数据是符合新需求的。

但如果采用以上方式创建表，则会创建失败。

因为'zhangsan'和'zhangsan'重复了。

```
insert into t_vip values(1,'zhangsan','zhangsan@qq.com');
insert into t_vip values(2,'zhangsan','zhangsan@sina.com');
```

正确的创建方式为：

```
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255),
    email varchar(255),
    unique(name,email) // 约束没有添加在列的后面，称为表级约束
);

insert into t_vip values(1,'zhangsan','zhangsan@qq.com');
insert into t_vip values(2,'zhangsan','zhangsan@sina.com');
```

执行结果：

```
mysql> drop table if exists t_vip;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> create table t_vip(
    -> id int,
    -> name varchar(255),
    -> email varchar(255),
    -> unique(name,email)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> insert into t_vip values(1,'zhangsan','zhangsan@qq.com');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip values(2,'zhangsan','zhangsan@sina.com');
Query OK, 1 row affected (0.00 sec)

mysql> select * from t_vip;
+------+----------+-------------------+
| id   | name     | email             |
+------+----------+-------------------+
|    1 | zhangsan | zhangsan@qq.com   |
|    2 | zhangsan | zhangsan@sina.com |
+------+----------+-------------------+
2 rows in set (0.00 sec)

mysql> insert into t_vip values(1,'zhangsan','zhangsan@qq.com');
ERROR 1062 (23000): Duplicate entry 'zhangsan-zhangsan@qq.com' for key
't_vip.name'
```

什么时候使用表级约束？

需要给多个字段联合起来添加某一个约束时，要使用表级约束

not null 只有列级约束，没有表级约束

unique 和 not null 可以联合使用吗？

```
drop table if exists t_vip;
create table t_vip(
        id int,
        name varchar(255) unique not null
);
```

```
// unique 和 not null 可以联合使用
// 在 MySQL 中，字段被 unique 和 not null 联合约束，该字段自动变成主键
// 注意:Oracle 中不一样
mysql> drop table if exists t_vip;
Query OK, 0 rows affected (0.02 sec)

mysql> create table t_vip(
    -> id int,
    -> name varchar(255) unique not null
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> desc t_vip;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int          | YES  |     | NULL    |       |
| name  | varchar(255) | NO   | PRI | NULL    |       |
+-------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> insert into t_vip(id,name) values(1,'zhangsan');
Query OK, 1 row affected (0.02 sec)

mysql> select * from t_vip;
+------+----------+
| id   | name     |
+------+----------+
|    1 | zhangsan |
+------+----------+
1 row in set (0.00 sec)

// 再次插入 name 为 zhangsan 记录，会受到 unique 约束而报错
mysql> insert into t_vip(id,name) values(1,'zhangsan');
ERROR 1062 (23000): Duplicate entry 'zhangsan' for key 't_vip.name'

// 插入记录，name 为 null，会受到 not null 约束而报错
mysql> insert into t_vip(id) values(2);
ERROR 1364 (HY000): Field 'name' doesn't have a default value
```

5.主键约束：primary key(简称 PK)

主键约束的相关术语：

主键约束：就是一种约束

主键字段：该字段上添加了主键字段，这样的字段叫做：主键字段

主键值：主键字段中的每一个值都叫做：主键值

什么是主键？作用是什么？（非常重要）

主键值是每一行记录的唯一标识

主键值是每一行记录的身份证号

任何一张表都应该有主键，没有主键，表无效。

主键的特征：not null + unique (主键值不能为 null，同时也不能出现重复)

类似于身份证号，中国人民的身份证号不能为 null，不能出现重复。

怎么给一张表添加主键约束？

```
drop table if exists t_vip;
// 一个字段做主键，叫做：单一主键
```

```
create table t_vip(
    id int primary key, //列级主键
    name varchar(255)
);

insert into t_vip(id,name) values(1,'zhangsan');
insert into t_vip(id,name) values(2,'lisi');
insert into t_vip(id,name) values(2,'wangwu');

// 创建 t_vip 表
mysql> drop table if exists t_vip;
Query OK, 0 rows affected (0.01 sec)

// 设定 id 为主键
mysql> create table t_vip(
    -> id int primary key,
    -> name varchar(255)
    -> );
Query OK, 0 rows affected (0.02 sec)

// 插入记录
mysql> insert into t_vip(id,name) values(1,'zhangsan');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip(id,name) values(2,'lisi');
Query OK, 1 row affected (0.01 sec)

// 插入记录，但是 id 出现重复，会受到主键（unique）的约束，故报错
mysql> insert into t_vip(id,name) values(2,'wangwu');
ERROR 1062 (23000): Duplicate entry '2' for key 't_vip.PRIMARY'
mysql> select * from t_vip;
+----+----------+
| id | name     |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
+----+----------+
2 rows in set (0.00 sec)

// 插入记录，id 为 null，会受到主键（not null）的约束，故报错
mysql> insert into t_vip(name) values('wangwu');
ERROR 1364 (HY000): Field 'id' doesn't have a default value
```

可以这样添加主键吗？

```
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255),
    primary key(id)
);
insert into t_vip values(1,'shangsan');
insert into t_vip values(1,'shangsan');

// 可以通过表级约束添加主键
mysql> drop table if exists t_vip;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> create table t_vip(
    -> id int,
```

```
    -> name varchar(255),
    -> primary key(id)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into t_vip values(1,'shangsan');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip values(1,'shangsan');
ERROR 1062 (23000): Duplicate entry '1' for key 't_vip.PRIMARY'
```

表级约束是给多个字段联合起来添加约束？

```
// id 和 name 联合起来作主键，叫做复合主键
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255),
    email varchar(255),
    primary key(id,name)
);

insert into t_vip values(1,'shangsan','zhangsan@qq.com');
insert into t_vip values(1,'lisi','lisi@qq.com');
insert into t_vip(name,email) values('wangwu','wangwu@qq.com');
insert into t_vip(id,email) values(2,'wangwu@qq.com');
insert into t_vip(email) values('123@qq.com');

insert into t_vip values(1,'shangsan');

//
mysql> drop table if exists t_vip;
Query OK, 0 rows affected (0.01 sec)

mysql> create table t_vip(
    -> id int,
    -> name varchar(255),
    -> email varchar(255),
    -> primary key(id,name)
    -> );
Query OK, 0 rows affected (0.02 sec)

// 虽然 id 都是 1，但是 id 和 name 联合起来就不会重复了
mysql>
mysql> insert into t_vip values(1,'shangsan','zhangsan@qq.com');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip values(1,'lisi','lisi@qq.com');
Query OK, 1 row affected (0.00 sec)

mysql> select * from t_vip;
+----+----------+-----------------+
| id | name     | email           |
+----+----------+-----------------+
|  1 | lisi     | lisi@qq.com     |
|  1 | shangsan | zhangsan@qq.com |
+----+----------+-----------------+
2 rows in set (0.00 sec)

// id 和 name 联合起来出现重复，就会报错
mysql> insert into t_vip values(1,'lisi','lisi@qq.com');
```

ERROR 1062 (23000): Duplicate entry '1- lisi' for key 't_vip.PRIMARY'

在实际开发中，不建议使用复合主键，建议使用单一主键。
因为主键值存在的意义就是这行记录的身份证号码，只要意义达到，单一主键就可以做到
复合主键比较复杂，不建议使用

一个表中，主键约束可以加多个吗？

```
drop table if exists t_vip;
create table t_vip(
    id int primary key,
    name varchar(255) primary key,
    email varchar(255)
);

// 结论：一张表中，主键约束只能添加 1 个，主键只能有 1 个
mysql> drop table if exists t_vip;
Query OK, 0 rows affected (0.02 sec)

mysql> create table t_vip(
    -> id int primary key,
    -> name varchar(255) primary key,
    -> email varchar(255)
    -> );
ERROR 1068 (42000): Multiple primary key defined
```

主键值建议使用：
- int
- bigint
- char
- 等类型
- 不建议使用 varchar 来做主键，主键值一般都是数字，一般都是定长的

主键除了单一主键和复合主键，还可以这样进行分类？
- 自然主键：主键值是一个自然数，和业务没关系
- 业务主键：主键值和业务密切相关，例如拿银行卡号作为主键值，这就是业务主键

在实际开发中，使用业务主键多还是自然主键多？
- 答案是：自然主键。因为主键只要做到不重复就行，不需要有意义。
- 业务主键不好，因为主键一旦和业务挂钩，那么当业务发生变动的时候，可能会影响到主键值，所以业务主键不建议使用。尽量使用自然主键

在 MySQL 当中，有一种机制，可以帮助我们自动维护一个主键值？

```
drop table if exists t_vip;
create table t_vip(
    id int primary key auto_increment,
    name varchar(255)
);

insert into t_vip(name) values('zhangsan');
insert into t_vip(name) values('zhangsan');
insert into t_vip(name) values('zhangsan');
insert into t_vip(name) values('zhangsan');
insert into t_vip(name) values('zhangsan');
insert into t_vip(name) values('zhangsan');

select * from t_vip;
```

// 在主键后面增加 auto_increment 关键字

```
mysql> drop table if exists t_vip;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> create table t_vip(
    -> id int primary key auto_increment,
    -> name varchar(255)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> insert into t_vip(name) values('zhangsan');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip(name) values('zhangsan');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip(name) values('zhangsan');
Query OK, 1 row affected (0.01 sec)

mysql> insert into t_vip(name) values('zhangsan');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip(name) values('zhangsan');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_vip(name) values('zhangsan');
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> select * from t_vip;
+----+----------+
| id | name     |
+----+----------+
|  1 | zhangsan |
|  2 | zhangsan |
|  3 | zhangsan |
|  4 | zhangsan |
|  5 | zhangsan |
|  6 | zhangsan |
+----+----------+
6 rows in set (0.00 sec)
```

auto_increment 表示自增，从 1 开始，以 1 递增

6.外键约束：foreign key(简称 FK) 非常重要

外键约束涉及到的相关术语：

外键约束：foreign key 一种约束

外键字段：该字段上添加了外键约束

外键值：外键字段当中的每一个值

业务背景：请设计数据库表，来描述学生和班级的信息

第一种方案：班级和学生存储在一张表中

no(pk) name classno classname

---

1 'jack' 100 北京市大兴区亦庄镇第二中学高三 1 班
2 'lucy' 100 北京市大兴区亦庄镇第二中学高三 1 班
3 'lilei' 100 北京市大兴区亦庄镇第二中学高三 1 班
4 'hanmeimei' 100 北京市大兴区亦庄镇第二中学高三 1 班
5 'zhangsan' 101 北京市大兴区亦庄镇第二中学高三 2 班
6 'lisi' 101 北京市大兴区亦庄镇第二中学高三 2 班
7 'wangwu' 101 北京市大兴区亦庄镇第二中学高三 2 班
8 'zhaoliu' 101 北京市大兴区亦庄镇第二中学高三 2 班

分析以上方案的缺点：
数据冗余，空间浪费
这个设计是比较失败的

第二种方案：班级一张表，学生一张表

t_class 班级表
classno(pk) classname
---
100 北京市大兴区亦庄镇第二中学高三 1 班
101 北京市大兴区亦庄镇第二中学高三 2 班

t_student 学生表
no(pk) name cno(班级编号)
---
1 'jack' 100
2 'lucy' 100
3 'lilei' 100
4 'hanmeimei' 100
5 'zhangsan' 101
6 'lisi' 101
7 'wangwu' 101
8 'zhaoliu' 101

当 cno 字段没有任何约束的时候，可能会导致数据无效。可能会出现一个 102，但是 102 班级不存在
所以为了保证 cno 字段中的值都是 100 和 101，需要给 cno 字段添加外键约束
那么，cno 字段就是外键约束，cno 字段中的每一个值都是外键值

注意：
t_class 是父表，t_student 是子表
删除表的顺序？ 先删子表，再删父表
创建表的顺序？ 先建父表，再建子表
删除数据的顺序？先删子表，再删父表
插入数据的顺序？先插父表，再插子表

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

create table t_class(
    classno int primary key,
    classname varchar(255)
```

```
);

create table t_student(
    no int primary key auto_increment,
    name varchar(255),
    cno int,
    foreign key(cno) references t_class(classno)
);

insert into t_class values(100,'北京市大兴区亦庄镇第二中学高三 1 班');
insert into t_class values(101,'北京市大兴区亦庄镇第二中学高三 2 班');

insert into t_student(name,cno) values('jack',100);
insert into t_student(name,cno) values('lucy',100);
insert into t_student(name,cno) values('lilei',100);
insert into t_student(name,cno) values('hanmeimei',100);
insert into t_student(name,cno) values('zhangsan',101);
insert into t_student(name,cno) values('lisi',101);
insert into t_student(name,cno) values('wangwu',101);
insert into t_student(name,cno) values('zhaoliu',101);

select * from t_student;
select * from t_class;

mysql> select * from t_class;
+---------+-----------------------------------------------------+
| classno | classname                                           |
+---------+-----------------------------------------------------+
|     100 | 北京市大兴区亦庄镇第二中学高三 1 班                 |
|     101 | 北京市大兴区亦庄镇第二中学高三 2 班                 |
+---------+-----------------------------------------------------+

mysql> select * from t_student;
+----+-----------+------+
| NO | NAME      | cno  |
+----+-----------+------+
|  1 | jack      |  100 |
|  2 | lucy      |  100 |
|  3 | lilei     |  100 |
|  4 | hanmeimei |  100 |
|  5 | zhangsan  |  101 |
|  6 | lisi      |  101 |
|  7 | wangwu    |  101 |
|  8 | zhaoliu   |  101 |
+----+-----------+------+
8 rows in set (0.00 sec)
```

t_student 使用外键后，cno 的键值就只能在 100 和 101 之间选择。

思考一下：子表中的外键引用的父表中的某个字段，被引用的这个字段必须是主键吗？

答案是：被引用的这个字段不一定是主键，但是必须具有唯一性。至少具有 unique 约束

思考：外键值可以为 null 吗？

答案是：可以，也就是 t_student 中的 cno 可以为 null

```
mysql> insert into t_student(name) values ('xiaohong');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from t_student;
+----+-----------+------+
| NO | NAME      | cno  |
+----+-----------+------+
|  1 | jack      |  100 |
|  2 | lucy      |  100 |
|  3 | lilei     |  100 |
|  4 | hanmeimei |  100 |
|  5 | zhangsan  |  101 |
|  6 | lisi      |  101 |
|  7 | wangwu    |  101 |
|  8 | zhaoliu   |  101 |
|  9 | xiaohong  | NULL |
+----+-----------+------+
9 rows in set (0.00 sec)
```

为什么外键可以为 null？

ORACLE 里的外键也允许为空，一般用处不大但也不能说完全没用。

举个例子，公司新采购一批电脑，主键为主表.电脑 ID，

外键为子表.员工 ID，如果电脑暂时还没有归属人，员工 ID 可以默认为 NULL

6.存储引擎

1.什么是存储引擎？有什么作用？

存储引擎：MySQL 中特有的一个术语，其他数据库中没有。（Oracle 中有，但是不叫这个名字）

实际上存储引擎是一个表，存储/组织数据的方式

不同的存储引擎，表存储的方式会有不同

2.怎么给表添加/指定存储引擎？

show create table t_student;

可以在建表的时候给表指定存储引擎。

```
t_student | CREATE TABLE `t_student` (
  `NO` int NOT NULL AUTO_INCREMENT,
  `NAME` varchar(255) DEFAULT NULL,
  `cno` int DEFAULT NULL,
  PRIMARY KEY (`NO`),
  KEY `cno` (`cno`),
  CONSTRAINT `t_student_ibfk_1` FOREIGN KEY (`cno`) REFERENCES `t_class` (`classno`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8
```

在建表的时候，可以在最后小括号的')'的右边使用：

ENGINE 用来指定存储引擎

CHARSET 用来指定字符编码方式

结论：

MySQL 默认的存储引擎是：InnoDB

MySQL 默认的字符编码方式为：uft8

建表时指定存储引擎以及字符编码方式

```
drop table if exists t_product;
create table t_product(
    id int primary key,
    name varchar(255)
)engine = InnoDB default charset = gbk;
```

3.MySQL 支持哪些存储引擎？

命令：show engines \g 注意不加分号

```
mysql> show engines \g
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
9 rows in set (0.00 sec)

// 当前数据库版本
mysql> select version();
+-----------+
| version() |
+-----------+
| 8.0.24    |
+-----------+
1 row in set (0.00 sec)
```

MySQL 支持 9 大存储引擎，当前 8.0.24 支持 8 个存储引擎。版本不同，支持情况不同

4.关于 MySQL 中常用的存储引擎？

MYISAM 存储引擎？

它管理的表具有以下特征：使用三个文件表示每个表：

格式文件 - 存储表结构的定义（mytable.frm）

数据文件 - 存储表行的内容（mytable.MYD）

索引文件 - 存储表上索引（mytable.MYI） 索引，相当于一本书的目录，缩小扫描范围，提高查询效率的机制

可被转换为压缩、只读表来节省空间

提示一下：对于一张表来说，只要是主键，

或者加有 unique 约束的字段上就会自动创建索引

MyISAM 存储引擎的特点：

可被转换为压缩、只读表来节省空间

MyISAM 不支持事务，安全性很低

```
drop table if exists t1;
create table t1(
    no int primary key,
    name varchar(255),
    email varchar(255)
) engine = MyISAM default charset = utf8;

mysql> drop table if exists t1;
Query OK, 0 rows affected (0.01 sec)

mysql> create table t1(
    -> no int primary key,
    -> name varchar(255),
    -> email varchar(255)
    -> ) engine = MyISAM default charset = utf8;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> show create table t1;
+-------+-----------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------+
|                Table                |                Create                Table
|
+-------+-----------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------+
| ttt    | CREATE TABLE `t1` (
  `no` int NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`no`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb3 |
+-------+-----------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------+
1 row in set (0.00 sec)
```

InnoDB 存储引擎？

这是 MySQL 默认的存储引擎，同时也是一个重量级的存储引擎。

InnoDB 支持事务，支持数据库崩溃后的自动恢复机制。

InnoDB 最主要的特点是非常安全。

```
drop table if exists t2;
create table t2(
    no int primary key,
    name varchar(255),
    email varchar(255)
) engine = InnoDB default charset = utf8;

mysql> drop table if exists t2;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> create table t2(
    -> no int primary key,
    -> name varchar(255),
    -> email varchar(255)
    -> ) engine = InnoDB default charset = utf8;
Query OK, 0 rows affected, 1 warning (0.02 sec)

mysql> show create table t2;
```

```
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------+
|  Table  |  Create  Table
|
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------+
| t2    | CREATE TABLE `t2` (
  `no` int NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 |
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------+
1 row in set (0.00 sec)
```

它管理的表具有下列主要特征：

- 每个 InnoDB 表在数据库目录中以 .frm 格式文件表示
- InnoDB 表空间 tablespace 被用于存储表的内容（表空间是一个逻辑名称。表空间存储数据+索引）
- 提供一组用来记录事务性活动的日志文件
- 用 COMMIT 提交 、 SAVEPOINT 及 ROLLBACK 回滚，支持事务处理
- 提供全 ACID 兼容
- 在 MySQL 服务器崩溃后提供自动恢复
- 多版本（ MVCC ）和行级锁定
- 支持外键及引用的完整性 ，包括级联删除和更新

InnoDB 最大的特点就是支持事务：

以保证数据的安全。效率不是很高，并且也不能压缩，不能转换为只读，不能很好地节省存储空间

MEMORY 存储引擎?

MEMORY 存储引擎管理的表具有下列特征：

- 在数据库目录内，每个表均以 .frm 格式的文件表示。
- 表数据及索引 被存储在内存中。
- 表级锁机制。
- 不能包含 TEXT 或 BLOB 字段。

```
drop table if exists t3;
create table t3(
    no int primary key,
    name varchar(255),
    email varchar(255)
) engine = MEMORY default charset = utf8;

mysql> drop table if exists t3;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> create table t3(
    -> no int primary key,
    -> name varchar(255),
    -> email varchar(255)
    -> ) engine = MEMORY default charset = utf8;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

```
mysql> show create table t3;
+-------+-----------------------------------------------------------------
--------------------------------------------------------------------------
-----------------------------+
|  Table  |  Create  Table
|
+-------+-----------------------------------------------------------------
--------------------------------------------------------------------------
-----------------------------+
| t3    | CREATE TABLE `t3` (
  `no` int NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`no`)
) ENGINE=MEMORY DEFAULT CHARSET=utf8mb3 |
+-------+-----------------------------------------------------------------
--------------------------------------------------------------------------
-----------------------------+
1 row in set (0.00 sec)
```

MEMORY 存储引擎以前被称为 HEAP 引擎。

MEMORY 引擎优点：查询效率是最高的。不需要和硬盘交互
MEMORY 引擎缺点：不安全，关机断电之后数据消失。因为数据和索引都是在内存中

7.事务（重点：五颗星*****，必须理解，必须掌握）

1. 一个事务其实就是一个完整的业务逻辑。
   是一个最小的工作单元，不可再分。

   什么是一个完整的业务逻辑?
   假设转账，从 A 账户向 B 账户中转账 10000.
   将 A 账户的钱减少 10000 (update 语句)
   将 B 账户的钱增加 10000 (update 语句)
   这就是一个完整的业务逻辑

   以上的操作是一个最小的工作单元，要么同时成功，要么同时失败，不可再分
   这两个 update 语句要求必须是同时成功或者同时失败，这样才能保证钱是正确的

2.只有 DML 语句才会有事务一说，其他语句和事务无关
   insert
   delete
   update
   只有以上的三个语句和事务有关系，其他都没有关系

   因为 只有以上的三个语句是数据库表中数据进行增、删、改的
   只要操作一旦涉及到数据的增删改，那么就一定要考虑安全机制

   数据安全是第一位

3.假设所有的业务，只要一条 DML 语句就能完成，还有必要存在事务机制吗?
   正是因为做某件事的时候，需要多条 DML 语句共同联合起来完成才能完成。
   所以需要事务的存在。

   如果任何一件事务都能一条 DML 语句就能搞定，那么事务就没有存在的价值了。

   到底什么是事务呢?

说到底，本质上，一个事务就是多条 DML 语句同时成功或者同时失败。
事务：就是批量的 DML 语句同时成功或者同时失败。

4.事务是如何做到多条 DML 语句同时成功或者同时失败的呢？
InnoDB 存储引擎：提供一组用来记录事务性活动的日志文件
事务开启：
insert
insert
insert
insert
update
update
update
delete
delete
...
事务结束了.
在事务的执行过程中，每一条 DML 的操作都会记录到"事务活动的日志文件"中。
在事务的执行过程中，我们可以提交事务，也可以回滚事务。

提交事务？commit
清空事务性活动的日志文件，将数据的全部彻底持久化到数据库表当中。
提交事务标志着事务的结束。并且是一种全部成功的结束。
回滚事务？rollback
将之前所有的 DML 操作全部撤销，并且清空事务性活动的日志文件
回滚事务标志着事务的结束，并且是一种全部失败的结束。

5.怎么提交事务？怎么回滚事务？
提交事务：commit 语句
回滚事务：rollback 语句（回滚永远都是只能回滚到上一次的提交点）

事务对应的英语单词：transaction

测试一下，在 MySQL 中默认的事务行为是什么？
MySQL 默认情况下是支持自动提交事务的。（自动提交）
什么是自动提交？
每执行一条 DML 语句，则提交一次。
这种自动提交实际上是不符合实际的而开发习惯的，因为一个业务
通常是需要多条 DML 语句共同执行才能完成，所以为了保证数据的安全，
必须要求同时成功之后在提交，所以不能执行一条就提交一条

怎么将 MySQL 的自动提交机制关闭呢？
先执行命令：start transaction;

演示事务：
```
mysql> use bjpowernode;
Database changed
mysql> select * from dept_cpy;
Empty set (0.00 sec)
```

---------- 回滚事务案例 1----------

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into dept_cpy values(10,'abc','bj');
```

```
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_cpy values(10,'abc','bj');
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_cpy values(10,'abc','bj');
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_cpy values(10,'abc','bj');
Query OK, 1 row affected (0.00 sec)

mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     10 | abc   | bj   |
|     10 | abc   | bj   |
|     10 | abc   | bj   |
+--------+-------+------+
4 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from dept_cpy;
Empty set (0.00 sec)

--------------------------- 回滚事务案例 2----------------------------------
----------------------
// 起始状态
mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
+--------+-------+------+
1 row in set (0.00 sec)

// 关闭自动提交事务
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)

mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | def   | sx   |
|     20 | def   | sx   |
|     20 | def   | sx   |
```

```
+--------+-------+------+
4 rows in set (0.00 sec)

mysql> update dept_cpy set dname = 'salesman' where deptno = '20';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from dept_cpy;
+--------+----------+------+
| DEPTNO | DNAME    | LOC  |
+--------+----------+------+
|     10 | abc      | bj   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
+--------+----------+------+
4 rows in set (0.00 sec)

// 回滚
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

// 可以回到起始状态
mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
+--------+-------+------+
1 row in set (0.00 sec)
--------------------------- 回滚事务案例 3----------------------------------
----------------------
// 起始状态
mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | def   | sx   |
|     20 | def   | sx   |
+--------+-------+------+
3 rows in set (0.00 sec)

// 关闭自动提交事务
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

// 事务-插入操作
mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)
// 事务-插入操作
mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)
// 事务-插入操作
mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)
// 事务-修改操作
mysql> update dept_cpy set dname = 'salesman' where deptno = '20';
Query OK, 5 rows affected (0.00 sec)
```

```
Rows matched: 5  Changed: 5  Warnings: 0

mysql> select * from dept_cpy;
+--------+----------+------+
| DEPTNO | DNAME    | LOC  |
+--------+----------+------+
|     10 | abc      | bj   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
+--------+----------+------+
6 rows in set (0.00 sec)

// 事务- 删除操作
mysql> delete from dept_cpy where deptno='10';
Query OK, 1 row affected (0.00 sec)

mysql> select * from dept_cpy;
+--------+----------+------+
| DEPTNO | DNAME    | LOC  |
+--------+----------+------+
|     20 | salesman | sx   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
|     20 | salesman | sx   |
+--------+----------+------+
5 rows in set (0.00 sec)

// 满足客户需求，则提交一次 commit，事务成功。
// 否则回滚，撤销事务的全部操作，事务失败
mysql> rollback;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | def   | sx   |
|     20 | def   | sx   |
+--------+-------+------+
3 rows in set (0.00 sec)

---------------提交事务-----------------------------------------------------
-----
// 起始状态
mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
+--------+-------+------+
1 row in set (0.00 sec)

// 关闭自动提交事务
mysql> start transaction;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_cpy values(20,'def','sx');
Query OK, 1 row affected (0.00 sec)

// 提交事务
mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | def   | sx   |
|     20 | def   | sx   |
|     20 | def   | sx   |
+--------+-------+------+
4 rows in set (0.00 sec)

// 回滚
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

// 无法回滚到起始状态
mysql> select * from dept_cpy;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | def   | sx   |
|     20 | def   | sx   |
|     20 | def   | sx   |
+--------+-------+------+
4 rows in set (0.00 sec)
```

6.事务包括 4 个特性。

A:原子性

说明事务是最小的工作单元，不可再分

C:一致性

在同一个事务当中，所有的操作必须同时成功，或者同时失败

以保证数据的一致性

I:隔离性

A 事务和 B 事务之间具有一定的隔离

教室 A 和教室 B 之间有一道墙，这道墙就是隔离性

A 事务在操作一张表的时候，另一个事务 B 也在操作这个表

多线程并发操作这张表

D:持久性

事务最终结束的一个保障。

事务提交，就相当于将没有保存到硬盘上的数据保存到硬盘上

7.重点研究事务的隔离性

教室 A 和教室 B 之间有一道墙，这道墙可以很厚，也可以很薄，这就是事务的隔离级别

隔离级别越高，表示这道墙越厚。

事务和事务之间的隔离级别：

读未提交：read uncommitted(最低的隔离级别) "没有提交就读到了"

事务 A 可以读取到事务 B 未提交的数据

这种隔离级别存在的问题就是：脏读现象（Dirty read）

我们称读到了脏数据

这种隔离级别一般都是理论上的，大多数的数据库的隔离级别二档起步

读已提交：read committed "提交之后才能读到"

事务 A 只能读取到事务 B 提交的数据

这种隔离级别解决了什么问题?

解决了脏读现象

这种隔离级别存在什么问题?

不可重复读取问题

什么是不可重复读取数据?

在事务开启之后，第一次读到的数据是 3 条，当前事务还没有结束

可能第二次在读取的时候，读到的数据是 4 条，3 不等于 4，

称为不可重复读取

这种隔离级别是比较真实的数据，每一次读到的数据是绝对的真实

Oracle 数据库默认的隔离级别是：读已提交

可重复读：repeatable read "提交之后也读不到，永远读取的时候都是刚开始事务时的数据"

事务 A 开启之后，不管是多久，每次在事务 A 中读取到的数据都是一致的

即使事务 B 将数据修改，并且已经提交，事务 A 读取到的数据还是没有发生改变

这就是可重复读取

可重复读解决了什么问题?

可重复读解决了不可不可重复读的问题

可重复读存在的问题：

可能会出现幻影读

每一次读到的数据都是幻象，不够真实

早上 9 点 B 开启了事务，只要事务 B 不结束，到晚上 9 点，事务 A 读到的数据还是那样，也就是最初状态的数据

读到的是假象，不够绝对的真实

MySQL 中事务默认的隔离级别事务：可重复读

序列化/串行化：serializable(最高的隔离级别)

这是最高隔离级别，效率最低。解决了所有问题

这种隔离级别表示事务排队，不能并发

有点类似于：synchronized，线程同步

每一次读取到的数据都是最真实的，但是效率是最低的

8.验证各种隔离级别

查看当前的事务隔离级别：

执行语句：select @@transaction_isolation;

```
mysql> select @@transaction_isolation;
+-------------------------+
| @@transaction_isolation |
+-------------------------+
| REPEATABLE-READ         |
+-------------------------+
1 row in set (0.00 sec)
```

这是 MySQL 默认的事务隔离级别

设置当前事务的事务隔离等级：

```
mysql> set global transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00 sec)
```

或者

```
SET transaction_isolation = 'READ-UNCOMMITTED';
```

或

```
SET SESSION transaction_isolation = 'READ-UNCOMMITTED';
```

被测试的表是 t_user;

1.验证：read uncommitted

```
mysql> set global transaction isolation level read uncommitted;
```

事务 A	事务 B
use bjpowernode;	
	use bjpowernode;
start transaction;	
	start transaction;
select * from t_user;(空表)	
	insert into t_user values('zhangsan');
select * from t_user;(B 事务没有提交，但是可以查到 t_user 的数据)	

事务 A 的操作：

```
mysql> use bjpowernode;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
Empty set (0.00 sec)

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)
```

事务 B 的操作：

```
mysql> use bjpowernnode;
```

```
Query OK, 0 rows affected (0.00 sec)
mysql> use bjpowernode;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

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

2.验证：read committed

mysql> set transaction_isolation = 'read-committed';

事务 A                                                  事务 B
-------------------------------------------------------
use bjpowernode;
                                                        use bjpowernode;

start transaction;
                                                        start transaction;
select * from t_user;(空表)
                                                        insert into t_user
values('zhangsan');
select * from t_user;(空表)
                                                        commit;
select * from t_user;(B 事务 commit 提交后，可以查到 t_user 的数据)

事务 A 的操作：
mysql> use bjpowernode;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select @@transaction_isolation;
+-------------------------+
| @@transaction_isolation |
+-------------------------+
| READ-COMMITTED          |
+-------------------------+
1 row in set (0.00 sec)

// 初始状态 t_user 为空表
mysql> select * from t_user;
Empty set (0.00 sec)

// 事务 B 已经往 t_user 插入了记录，但是没有提交
// 由于事务隔离等级为 read committed， 此时事务 A 无法查询到 B 提交的记录
mysql> select * from t_user;
Empty set (0.00 sec)

// 事务 B 提交了记录，此时事务 A 方可查询到 B 提交的记录
mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)
```

事务 B 的操作：

```
mysql> use bjpowernode;
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
Empty set (0.00 sec)

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

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

3.验证：repeatable read

```
mysql> set global transaction isolation level repeatable read;
```

事务 A	事务 B
use bjpowernode;	
	use bjpowernode;
start transaction;	
	start transaction;
select * from t_user;(初始记录)	
	insert into t_user values('lisi');
	insert into t_user values('lisi');
	insert into t_user values('lisi');
	insert into t_user values('lisi');
	insert into t_user values('lisi');
select * from t_user;(记录不变)	
	commit;
select * from t_user;(记录不变)	
	insert into t_user values('jack');
	insert into t_user values('tom');
	commit;
select * from t_user;(记录不变)	
	insert into t_user values('wangwu');
	insert into t_user values('tom');
	commit;
select * from t_user;(记录不变)	

事务 A 的操作：

```
mysql> use bjpowernode;
Database changed
```

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

// 第一次提交的数据
mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)

// 事务 B 已经往 t_user 插入了记录，并且提交
// 但是事务 A 查询到的记录不变
mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)
```

事务 B 的操作：

```
mysql> use bjpowernode;
Database changed
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
+----------+
1 row in set (0.00 sec)

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

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

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

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

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

mysql> select * from t_user;
+----------+
| name     |
+----------+
| zhangsan |
| lisi     |
| lisi     |
| lisi     |
| lisi     |
| lisi     |
+----------+
6 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

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

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

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_user values('wangwu');
Query OK, 1 row affected (0.01 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

4.验证：serializable

mysql> set global transaction isolation level serializable;

事务 A	事务 B
use bjpowernode;	
	use bjpowernode;
start transaction;	
	start transaction;
select * from t_user;(初始记录)	
insert into t_user values('lisi');	
	select * from t_user;
commit;	
start transaction;	
	start transaction;
	insert into t_user values('wangwu');
select * from t_user;	
	commit;

事务 A 的操作：

```
mysql> use bjpowernode;
Database changed
mysql> select * from t_user;
+------+
| name |
+------+
| abc  |
+------+
1 row in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
+------+
| name |
+------+
| abc  |
+------+
1 row in set (0.00 sec)

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

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
+--------+
| name   |
+--------+
| abc    |
| lisi   |
| wangwu |
+--------+
```

3 rows in set (20.64 sec)

事务 B 的操作：

```
mysql> select @@transaction_isolation;
+-------------------------+
| @@transaction_isolation |
+-------------------------+
| SERIALIZABLE            |
+-------------------------+
1 row in set (0.00 sec)

mysql> use bjpowernode;
Database changed
mysql> select * from t_user;
+------+
| name |
+------+
| abc  |
+------+
1 row in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
+------+
| name |
+------+
| abc  |
| lisi |
+------+
2 rows in set (33.05 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

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

mysql> select * from t_user;
+--------+
| name   |
+--------+
| abc    |
| lisi   |
| wangwu |
+--------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

mysql day04 课堂笔记

1.索引

1.什么是索引？(index)

索引在数据库表的字段上添加的，是为了提高检索效率而存在的一种机制。

一张表的一个字段可以添加一个索引，当然，多个字段联合起来也可以添加索引。

索引相当于一本书的目录，是为了缩小扫描范围而存在的一种机制。

对于一本字典来说，查找某个汉字有两种方式

第一种方式：一页一页的挨着找，直到找到为止，这种查找方式属于全字典扫描。

效率较低。

第二种方式：先通过目录（索引）去定位一个大概的位置，然后直接定位到这个位置，做局域性扫描，缩小扫描的范围，快速查找。这种查找方式效率较高

t_user

id name(nameIndex) email(emailIndex) address(emailAddressIndex)

---

1 zhangsan

2 lisi

3 wangwu

4 zhaoliu

5 hanmeimei

6 jack

select * from t_user where name = 'jack';

以上的这条 SQL 语句会去那么字段上扫描，因为查询条件是：name = 'jack';

如果 name 字段上没有添加索引（目录），或者说没有给 name 字段创建索引，MySQL 会进行全扫描，会将 name 字段上的每一个值都比对一遍。效率比较低

MySQL 在查询方面主要就是两种方式：

第一种方式：全表扫描

第二种方式：根据索引检索

注意：

在实际中，汉语字典前面的目录是排序的，按照 abcdefgh...排序

为什么排序呢？因为只要排序了才会有区间查找一说。

缩小扫描范围其实就是扫描某个区间。

在 MySQL 数据库中索引也是需要排序的，并且这个索引的排序和 TreeSet 数据结构相同。TreeSet 的底层是一个自平衡的二叉树。

在 MySQL 当中索引是一个 B- Tree 数据结构

遵循左小右大原则存放。采用中序遍历方式读取数据

2.索引的实现原理？

t_user

id(PK) name 每一行记录在硬盘上都有物理存储编号

---

100 zhangsan ox1111

120 lisi ox2222

99 wangwu ox8888

88 zhaoliu ox9999

101 jack ox6666

55 lucy ox5555

130 tom ox7777

提醒 1：在任何数据库当中主键上都会自动添加索引对象，id 字段上自动有索引
因为 id 是主键。另外在 MySQL 当中，一个字段上如果有 unique 约束的话，也会
自动创建索引对象。

提醒 2：在任何数据库当中，任何一张表的任何一条记录在硬盘存储上都有
一个硬盘的物理存储编号。

提醒 3：在 MySQL 当中，索引是一个单独的对象，不同的存储引擎以不同的形式
存在，在 MyISAM 存储引擎中，索引存储在一个.MYI 文件中。在 InnoDB 存储引擎中，
索引存储在一个逻辑名称叫做 tablespace 的当中。在 MEMORY 存储引擎中，索引是被
存储在内存当中。不管索引存储在哪里，索引在 MySQL 当中都是一个数的形式存在。
（自平衡二叉树：B-Tree）

索引的实现原理：就是缩小扫描的范围，避免全表扫描

3.在 MySQL 中，主键上以及 unique 字段上，都会自动添加索引
什么条件下，我们会考虑给字段添加索引？
条件 1：数据量庞大（到底多么大算庞大，需要测试，每一个硬件环境不同）
条件 2：该字段经常出现在 where 的后面，以条件的形式存在。也就是说这个
字段总是被扫描
条件 3：该字段有很少的 DML（insert delete update）操作。（因为 DML 之后，索引需要重新排序）

建议不要随意添加索引，因为索引也是需要维护的，太多的话反而会降低系统的性能。
建议通过主键或者 unique 约束的字段去查找，这样操作效率会高一些

4.索引怎么创建？怎么删除？语法是什么？
创建索引：给 emp 表中的 ename 字段创建索引，命名为 emp_ename_index
```
mysql> create index emp_ename_index on emp(ename);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

删除索引：将 emp 表上的 emp_ename_index 索引删除
```
mysql> drop index emp_ename_index on emp;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

5.在 MySQL 中，怎么查看一个 SQL 语句是否使用了索引进行检索？
使用 explain 关键字

```
mysql> explain select * from emp where ename = 'KING';
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | emp   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   14 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
```

```
---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
扫描 14 条记录：说明没有使用索引。type=ALL

为 emp 表中的 enamel 字段添加 emp_index_ename 索引后，
mysql> explain select * from emp where ename = 'KING';
+----+-------------+-------+------------+------+----------------+-----
------------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys     | key                |
key_len | ref     | rows | filtered | Extra |
+----+-------------+-------+------------+------+----------------+-----
------------+---------+-------+------+----------+-------+
|    1 | SIMPLE           | emp      | NULL             | ref    | emp_index_ename |
emp_index_ename | 43          | const |      1 |    100.00 | NULL    |
+----+-------------+-------+------------+------+----------------+-----
------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
扫描 1 条记录：说明使用索引。type=ref

6.索引有失效的时候，什么时候索引会失效呢？

失效的第一种情况：
select * from emp where ename like '%T';

即使添加了索引，使用模糊查询时，以'%'开头
mysql> explain select * from emp where ename like '%T';
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key    | key_len | ref    |
rows | filtered | Extra           |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
|    1 | SIMPLE          | emp      | NULL           | ALL    | NULL               | NULL |
NULL       | NULL |     14 |      11.11 | Using where |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
扫描 14 条记录：说明没有使用索引。type=ALL。说明索引失效

ename 上即使添加了字段，也不会走索引，为什么？
    原因是因为模糊匹配当中以'%'开头
    尽量避免使用模糊查询的时候以'%'开头

失效的第二种情况：
使用 or 的时候会失效，如果使用 or，那么要求 or 两边的条件字段都要有
索引，才会走索引。如果其中一边有一个字段没有索引，那么另一个字段
的索引也会失效。所以这就是为什么不建议使用 or 的原因

// ename 字段有索引，job 字段没有添加索引
//  此时，使用 or 的话，索引会失效
mysql> explain select * from emp where ename = 'KING' or job = 'MANAGER';
+----+-------------+-------+------------+------+----------------+------+--
-------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys     | key    | key_len | ref    |
rows | filtered | Extra           |
+----+-------------+-------+------------+------+----------------+------+--
-------+------+------+----------+-------------+
|    1 | SIMPLE          | emp      | NULL           | ALL    | emp_index_ename | NULL |
```

```
NULL      | NULL |     14 |      16.43 | Using where |
          +----+-------------+-------+------------+------+----------------+------+--
-------+------+------+----------+-------------+
          1 row in set, 1 warning (0.00 sec)
```

扫描 14 条记录：说明没有使用索引。type=ALL。说明索引失效

失效的第三种情况：

使用复合索引的时候，没有使用左侧的列查找，索引失效。

什么是复合索引：

两个字段或者更多的字段联合起来添加一个索引，叫做复合索引

```
create index emp_job_sal_index on emp(job,sal);
mysql> create index emp_job_sal_index on emp(job,sal);
Query OK, 0 rows affected (0.08 sec)
Records: 0    Duplicates: 0    Warnings: 0

mysql> explain select * from emp where job = 'MANAGER';
+----+-------------+-------+------------+------+------------------+---
----------------+---------+-------+------+----------+-------+
| id  | select_type  | table  | partitions  | type  | possible_keys            | key
| key_len | ref     | rows | filtered | Extra |
+----+-------------+-------+------------+------+------------------+---
----------------+---------+-------+------+----------+-------+
|    1 | SIMPLE           | emp      | NULL            | ref    | emp_job_sal_index |
emp_job_sal_index | 39          | const |      3 |    100.00 | NULL    |
+----+-------------+-------+------------+------+------------------+---
----------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.01 sec)
```

扫描 3 条记录：说明使用索引。type=ref

```
explain select * from emp where sal > 1000;
mysql> explain select * from emp where sal = 5000;
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key    | key_len | ref    |
rows | filtered | Extra           |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
|   1 | SIMPLE          | emp     | NULL           | ALL    | NULL              | NULL |
NULL      | NULL |     14 |      10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

扫描 14 条记录：说明没有使用索引。type=ALL。说明索引失效

没有使用左侧的列 job 进行查找，索引失效

失效的第四种情况：

在 where 当中索引列参数参加了运算，索引失效。

给 emp 表中的 sal 字段添加索引

```
mysql> create index emp_sal_index on emp(sal);
Query OK, 0 rows affected (0.02 sec)
Records: 0    Duplicates: 0    Warnings: 0

mysql> explain select * from emp where sal = 800;
+----+-------------+-------+------------+------+---------------+------+-------
--------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key                  |
key_len | ref     | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+-------
```

```
--------+---------+-------+------+----------+-------+
|    1 | SIMPLE            | emp       | NULL              | ref    | emp_sal_index |
emp_sal_index | 9           | const |      1 |    100.00 | NULL   |
+----+-------------+-------+------------+------+---------------+-------
--------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
扫描 1 条记录：说明使用索引。type=ref

mysql> explain select * from emp where sal+1 = 800;
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref   |
rows | filtered | Extra           |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
|   1 | SIMPLE          | emp     | NULL           | ALL   | NULL              | NULL |
NULL     | NULL |    14 |    100.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
扫描 14 条记录：说明没有使用索引。type=ALL。说明索引失效
在 where 当中索引 sal 列参数参加了运算，索引失效。

失效的第五种情况：
在 where 当中索引使用了函数
explain select * from emp where lower(ename)    = 'smith';
mysql> explain select * from emp where lower(ename)    = 'smith';
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref   |
rows | filtered | Extra           |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
|   1 | SIMPLE          | emp     | NULL           | ALL   | NULL              | NULL |
NULL     | NULL |    14 |    100.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+
---------+------+------+----------+-------------+
1 row in set, 1 warning (0.01 sec)
扫描 14 条记录：说明没有使用索引。type=ALL。说明索引失效
在 where 当中,sal 索引使用了 lower 函数，索引失效

失效的第六种情况：
失效的第七种情况：
.....
```

7.索引是各种数据库进行优化的重要手段。优化的时候，优先考虑的因素就是索引。

索引在数据库当中有很多类别：

单一索引：

一个字段上添加索引

复合索引：

两个或多个字段上添加索引

主键索引：

主键上添加索引

唯一性索引：

在 unique 约束的字段上添加索引

注意：唯一性比较弱的字段上添加索引用处不大。越唯一，索引效率越高。

2.视图（view）

1.什么是视图？

view：站在不同的角度去看待同一份数据

2.怎么创建视图对象？怎么删除视图对象？

表复制：

```
mysql> create table dept2 as select * from dept;
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

dept3 表的记录：

```
mysql> select * from dept2;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)
```

创建视图对象：

```
create view dept2_view as select * from dept2;
```

删除视图对象：

```
drop view dept2_view;
```

```
mysql> create view dept2_view as select * from dept2;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> drop view dept2_view;
Query OK, 0 rows affected (0.01 sec)
```

注意：只要 DQL 语句才能以 view 的形式创建

```
create view dept2_view as 这里的语句必须是 DQL 语句(select 语句);
```

3.用视图做什么？

我们可以面向视图对象进行增删改查，对视图对象的增删改查会导致

原表被操作。（视图的特点：通过对视图对象的操作，会影响原表数据）

视图和复制表的区别：

```
// 表的复制，dept2 复制了 dept 表中的数据
create table dept2 as select * from dept;
// 创建 dept2 的视图对象 dept2_view
create view dept2_view as select * from dept2;
```

表的复制相当于是深拷贝，dept2 复制了 dept 表中的数据，修改 dept2 的记录，dept 不会改变。

创建 dept2 的视图对象 dept2_view 相当于浅拷贝，修改视图对象 dept2_view 的记录，dept2 的记录也会被修改。

```
// 创建 dept2 的视图对象 dept2_view
mysql> create view dept2_view as select * from dept2;
Query OK, 0 rows affected (0.01 sec)
```

```
// 查看 dept2_view 的记录
mysql> select * from dept2_view;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)

// 通过视图对象 dept2_view 插入数据
mysql> insert into dept2_view values(50,'BOSS','BEIJING');
Query OK, 1 row affected (0.01 sec)

mysql> select * from dept2_view;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
|     50 | BOSS       | BEIJING  |
+--------+------------+----------+
5 rows in set (0.00 sec)

// 可知原表的数据也发生了改变
mysql> select * from dept2;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
|     50 | BOSS       | BEIJING  |
+--------+------------+----------+
5 rows in set (0.00 sec)

// 利用视图对象进行删除
mysql> delete from dept2_view;
Query OK, 5 rows affected (0.01 sec)

// 原表也会删除
mysql> select * from dept2;
Empty set (0.00 sec)

create view
    emp_dept_view
as
    select
        e.ename,e.sal,d.dname
    from
        emp e
        join
        dept d
    on
        e.deptno = d.deptno;
```

```
// 创建视图对象
mysql> create view
-> emp_dept_view
-> as
-> select
-> e.ename,e.sal,d.dname
-> from
-> emp e
-> join
-> dept d
-> on
-> e.deptno = d.deptno;
Query OK, 0 rows affected (0.01 sec)

// 查询视图对象
mysql> select * from emp_dept_view;
+--------+---------+------------+
| ename  | sal     | dname      |
+--------+---------+------------+
| SMITH  |  800.00 | RESEARCH   |
| ALLEN  | 1600.00 | SALES      |
| WARD   | 1250.00 | SALES      |
| JONES  | 2975.00 | RESEARCH   |
| MARTIN | 1250.00 | SALES      |
| BLAKE  | 2850.00 | SALES      |
| CLARK  | 2450.00 | ACCOUNTING |
| SCOTT  | 3000.00 | RESEARCH   |
| KING   | 5000.00 | ACCOUNTING |
| TURNER | 1500.00 | SALES      |
| ADAMS  | 1100.00 | RESEARCH   |
| JAMES  |  950.00 | SALES      |
| FORD   | 3000.00 | RESEARCH   |
| MILLER | 1300.00 | ACCOUNTING |
+--------+---------+------------+
14 rows in set (0.00 sec)

// 面向视图更新，在 ACCOUNTING 部门工作的员工，工资修改为 1000
mysql> update emp_dept_view set sal = 1000 where dname = 'ACCOUNTING';
Query OK, 3 rows affected (0.01 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from emp_dept_view;
+--------+---------+------------+
| ename  | sal     | dname      |
+--------+---------+------------+
| SMITH  |  800.00 | RESEARCH   |
| ALLEN  | 1600.00 | SALES      |
| WARD   | 1250.00 | SALES      |
| JONES  | 2975.00 | RESEARCH   |
| MARTIN | 1250.00 | SALES      |
| BLAKE  | 2850.00 | SALES      |
| CLARK  | 1000.00 | ACCOUNTING |
| SCOTT  | 3000.00 | RESEARCH   |
| KING   | 1000.00 | ACCOUNTING |
| TURNER | 1500.00 | SALES      |
| ADAMS  | 1100.00 | RESEARCH   |
| JAMES  |  950.00 | SALES      |
| FORD   | 3000.00 | RESEARCH   |
| MILLER | 1000.00 | ACCOUNTING |
```

```
+--------+---------+------------+
14 rows in set (0.00 sec)

// 面向视图更新，原表数据被修改
// 在 emp 中，工作部门位于 ACCOUNTING 部门的工作的员工的 sal 被修改为
1000
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 | 1000.00 |    NULL |
10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 | 3000.00 |    NULL |
20 |
|  7839 | KING   | PRESIDENT | NULL | 1981-11-17 | 1000.00 |    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 | 1000.00 |    NULL |
10 |
+-------+--------+-----------+------+------------+---------+---------+-
-------+
14 rows in set (0.00 sec)

mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
4 rows in set (0.00 sec)
```

4.视图对象在实际开发在到底有什么用？“方便，简化开发，利于维护”

```
create view
    emp_dept_view
as
```

```
select
    e.ename,e.sal,d.dname
from
    emp e
    join
    dept d
on
    e.deptno = d.deptno;
```

假设有一条非常复杂的 SQL 语句，而这条 SQL 语句 需要在不同的位置上反复使用
每一次使用这个 SQL 语句的时候都需要重新编写，很长，很麻烦。
因此，可以把这条复杂的 SQL 语句以视图对象的形式新建。
在需要编写这条 SQL 语句的位置直接使用视图对象，可以大大简化开发。
并且利于后期的维护，因为修改的时候也只需要修改一个位置就行，
只需要修改视图对象所映射的 SQL 语句。

我们以后面向视图开发的时候，使用视图的时候可以像视图 table 一样。
可以对视图进行增删改查等操作。视图不是在内存当中，视图对象也是
存储在硬盘上，不会消失。

再提醒一下：
视图对应的语句只能是 DQL 语句
但是视图对象创建完成以后，可以对视图进行增删改查等操作

小插曲：
增删改查，又叫做：CRUD
CRUD 是在公司中程序员之间沟通的术语。

C:create(增)
R:retrive(查：检索)
U:update(改)
D:delete(删)

3.DBA 常用命令

新建用户：
语法格式：
create user 用户名 identified by 该用户的登录密码;

```
mysql> create user heweibin identified by 'hwb02345678';
Query OK, 0 rows affected (0.04 sec)
```

重点掌握：
数据的导入和导出（数据的备份）
其他命令了解一下即可。

数据导出？
```
mysql dump bjpowernode>D:\bjpowernode.sql - u root - p
Enter password:
```

数据导入？
注意：需要先登录的奥 MySQL 数据库的服务器上
然后创建数据库：create database bjpowernode
使用数据库：use bjpowernode;
然后初始化数据库：source D:\bjpowernode.sql

4.数据库设计三范式

1.什么是数据库设计范式?

数据库表的设计依据。

2.数据库设计范式共有 3 个

第一范式：要求任何一张表必须有主键，没有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，要求所有的非主键字段完全依赖主键，不要产生部分依赖

第三范式：建立在第二范式的基础之上，要求所有的非主键字段直接依赖主键，不要产生传递依赖

声明：三范式是面试官经常要问的，所以一定要熟记于心

设计数据库表的时候，按照以上的范式进行，可以避免表中数据的冗余，空间的浪费

3.第一范式

最核心，最重要的范式，所有表的设计都需要满足

必须有主键，并且每一个字段都是原子性不可再分

学生编号	学生姓名	联系方式
1001	张三	zs@gmail.com,1359999999
1002	李四	ls@gmail.com,13699999999
1001	王五	ww@163.net,13488888888

以上是学生表，满足第一范式吗?

不满足，第一：没有主键。第二：联系方式可以分为邮箱地址和电话

学生编号(PK)	学生姓名	邮箱地址	联系电话
1001	张三	zs@gmail.com	1359999999
1002	李四	ls@gmail.com	13699999999
1001	王五	ww@163.net	13488888888

4.第二范式：

建立在第一范式的基础之上，

要求所有的非主键字段完全依赖主键，不要产生部分依赖

学生编号	学生姓名	教师编号	教师姓名
1001	张三	001	王老师
1002	李四	002	赵老师
1003	王五	001	王老师
1001	张三	002	赵老师

以上是学生表，描述了学生和老师的关系：一个学生对应多个老师，一个老师对应多个学生

这是典型的：多对多关系。

满足第一范式吗?

不满足第一范式，没有主键。

怎么满足第一范式呢? 修改

学生编号+教师编号(PK)	学生姓名	教师姓名
1001	001	张三	王老师
1002	002	李四	赵老师

1003	001	王五	王老师
1001	002	张三	赵老师

学生编号 教师编号，两个字段联合做主键，复合主键（PK:学生编号+教师编号）

经过修改之后，以上的 表满足了第一范式。但是满足第二范式吗？

不满足第二范式，"张三"依赖 1001，"王老师"依赖 001，显然产生了部分依赖

产生部分依赖的缺点是什么？

数据冗余，空间浪费。"张三"重复，"王老师"重复

为了让以上的表满足第二范式，需要这样设计：

使用三张表来表示多对多的关系

学生表

学生编号(PK)	学生姓名
1001	张三
1002	李四
1003	王五

教师表

教师编号(PK)	教师姓名
001	王老师
002	赵老师

学生教师关系表

id(PK)	学生编号(FK)	教师编号(FK)
1	1001	001
2	1002	002
3	1003	001
4	1001	002

背口诀：

多对多怎么设计？

多对多，三张表，关系表两个外键。

5.第三范式

建立在第二范式的基础之上，

要求所有的非主键字段直接依赖主键，不要产生传递依赖

学生编号(PK)	学生姓名	班级编号	班级名称
1001	张三	01	一年一班
1002	李四	02	一年二班
1003	王五	03	一年三班
1004	赵六	03	一年三班

以上表满足的设计是描述班级和学生的关系，很显然是一对多关系

一个教室中有多个学生

分析以上表是否满足第一范式？

满足，因为有主键。

分析以上表是否满足第二范式？

满足第二范式，因为主键不是复合主键，没有产生部分依赖。主键是单一主键

分析以上表是否满足第三范式？

第三范式要求：不要产生传递依赖

一年一班依赖 01,01 依赖 1001，产生了传递依赖

不符合第三范式的要求，产生了数据的冗余

那么应该怎么设计一对多呢？

班级表：

班级编号(PK)	班级名称
01	一年一班
02	一年二班
03	一年三班

学生表：

学生编号(PK)	学生姓名	班级编号(FK)
1001	张三	01
1002	李四	02
1003	王五	03
1004	赵六	03

背口诀：

一对多，两张表，多的表加外键。

6.总结一下表的设计

一对多：

一对多，两张表，多的表加外键

多对多：

多对多，三张表，关系表加外键

一对一：

一对一放到一张表不就行了吗？为什么还要拆分多张表？

在实际的开发中，可能存在一张表字段太多，太庞大，这个时候要拆分表。

一对一怎么设计？

没有拆分表之前：一张表

t_user

id	login_name	login_pwd	real_name	email	address…
1	zhangsan	123456	张三	zhangsan@xxx.com	bj
2	lisi	123	李四	lisi@qq.com	tj
…					

这种庞大的表，建议拆分为两张表：

t_login 登录信息表

id(PK)	login_name	login_pwd
1	zhangsan	123456
2	lisi	123
…		

t_user 用户详细信息表

id(PK)	real_name	email	address…

login_id(FK+unique)

---

100 张三 zhangsan@xxx.com bj
1

200 李四 lisi@qq.com tj
2

…

口诀：一对一，外键唯一。

1.取得每个部门最高薪水的人员名称?

实现思路：

第一步：查询每个部门的最高薪资，作 b 表

```
mysql> select deptno,max(sal) as maxsal from emp group by deptno;
+--------+---------+
| deptno | maxsal  |
+--------+---------+
|     20 | 3000.00 |
|     30 | 2850.00 |
|     10 | 5000.00 |
+--------+---------+
3 rows in set (0.00 sec)
```

第二步：根据 b 表的字段，连接 emp 表(a 表)进行查询

连接条件：on a.deptno = b.deptno;

筛选条件：where a.sal = b.sal;

最终结果：

```
mysql>
        SELECT
            a.ename,
            b.deptno,
            b.sal
        FROM
            emp a
            LEFT JOIN ( SELECT deptno, max( sal ) sal FROM emp GROUP BY
deptno ) b ON a.deptno = b.deptno;
```

```
+-------+--------+---------+
| ename | deptno | sal     |
+-------+--------+---------+
| BLAKE |     30 | 2850.00 |
| SCOTT |     20 | 3000.00 |
| KING  |     10 | 5000.00 |
| FORD  |     20 | 3000.00 |
+-------+--------+---------+
4 rows in set (0.00 sec)
```

2.哪些人的薪水在部门的平均薪水之上?

实现思路：

第一步：查询每个部门的最高薪资，作 b 表

```
mysql> select deptno,avg(sal) avgsal  from emp group by deptno;
+--------+-------------+
| deptno | avgsal      |
+--------+-------------+
|     20 | 2175.000000 |
|     30 | 1566.666667 |
|     10 | 2916.666667 |
+--------+-------------+
3 rows in set (0.00 sec)
```

第二步：将 emp 表(a 表)和 b 表连接查询

连接条件：on a.deptno = b.deptno

筛选条件：where a.sal > avgsal

最终结果：

```
mysql>
```

```
SELECT
    a.ename,
    a.sal
FROM
    emp a
LEFT JOIN ( SELECT deptno, avg( sal ) avgsal FROM emp GROUP BY deptno )
b ON a.deptno = b.deptno
WHERE
    a.sal > avgsal;
+-------+---------+
| ename | sal     |
+-------+---------+
| ALLEN | 1600.00 |
| JONES | 2975.00 |
| BLAKE | 2850.00 |
| SCOTT | 3000.00 |
| KING  | 5000.00 |
| FORD  | 3000.00 |
+-------+---------+
6 rows in set (0.00 sec)
```

3.取得部门中(所有人的)平均的薪水等级？

实现思路：

第一步：查询所有员工的薪资等级，使用内连接中的非等值连接，查询结果作为 c 表

非等值连接条件：on a.sal between b.losal and b.hisal

```
mysql> select a.deptno deptno,b.grade grade from emp a left join salgrade b
on a.sal between b.losal and b.hisal;
+--------+-------+
| deptno | grade |
+--------+-------+
|     20 |     1 |
|     30 |     3 |
|     30 |     2 |
|     20 |     4 |
|     30 |     2 |
|     30 |     4 |
|     10 |     4 |
|     20 |     4 |
|     10 |     5 |
|     30 |     3 |
|     20 |     1 |
|     30 |     1 |
|     20 |     4 |
|     10 |     2 |
+--------+-------+
14 rows in set (0.00 sec)
```

第二步：将 c 表作为临时表，进行子查询，分组求均值 avg(sal)

分组条件：group by deptno;

最终结果:

```
mysql>
SELECT
    deptno,
    avg( grade )
FROM
    (
    SELECT
        a.deptno deptno,
```

```
        b.grade grade
    FROM
        emp a
        LEFT JOIN salgrade b ON a.sal BETWEEN b.losal
        AND b.hisal
    ) AS c
GROUP BY
    deptno
ORDER BY
    deptno;
+--------+------------+
| deptno | avg(grade) |
+--------+------------+
|     10 |     3.6667 |
|     20 |     2.8000 |
|     30 |     2.5000 |
+--------+------------+
3 rows in set (0.00 sec)
```

4.不准用组函数(Max)，取得最高薪水(给出两种解决方案)

第一种方案：

实现思路：将数据排序(降序)，然后取第一个数据

```
mysql>
    SELECT
        sal
    FROM
        emp
    ORDER BY
        sal DESC
    LIMIT 1;
+---------+
| sal     |
+---------+
| 5000.00 |
+---------+
1 row in set (0.00 sec)
```

第二中方案：

实现思路：

第一步：首先求出所有的非最大值，然后利用集合的差运算即可。计算所有的非最大值，要利用表自身的笛卡尔积

```
// 该运算后，sal 的数据中包含所有数据，除包含最大值的数据
mysql> select a.sal from emp a,emp b where a.sal < b.sal;
+---------+
| sal     |
+---------+
| 1300.00 |
|  950.00 |
| 1100.00 |
| 1500.00 |
| 1250.00 |
| 1250.00 |
|  800.00 |
|  950.00 |
| 1100.00 |
|  800.00 |
| 1300.00 |
|  950.00 |
| 1100.00 |
| 1500.00 |
```

| 2450.00 |
| 2850.00 |
| 1250.00 |
| 1250.00 |
| 1600.00 |
| 800.00 |
| 950.00 |
| 1100.00 |
| 800.00 |
| 1300.00 |
| 950.00 |
| 1100.00 |
| 1500.00 |
| 2450.00 |
| 1250.00 |
| 1250.00 |
| 1600.00 |
| 800.00 |
| 1300.00 |
| 950.00 |
| 1100.00 |
| 1500.00 |
| 1250.00 |
| 1250.00 |
| 1600.00 |
| 800.00 |
| 1300.00 |
| 950.00 |
| 1100.00 |
| 1500.00 |
| 2450.00 |
| 2850.00 |
| 1250.00 |
| 2975.00 |
| 1250.00 |
| 1600.00 |
| 800.00 |
| 1300.00 |
| 3000.00 |
| 950.00 |
| 1100.00 |
| 1500.00 |
| 3000.00 |
| 2450.00 |
| 2850.00 |
| 1250.00 |
| 2975.00 |
| 1250.00 |
| 1600.00 |
| 800.00 |
| 1300.00 |
| 950.00 |
| 1100.00 |
| 1250.00 |
| 1250.00 |
| 800.00 |
| 950.00 |
| 800.00 |
| 800.00 |
| 1300.00 |

```
|   950.00 |
| 1100.00 |
| 1500.00 |
| 2450.00 |
| 2850.00 |
| 1250.00 |
| 2975.00 |
| 1250.00 |
| 1600.00 |
|   800.00 |
|   950.00 |
| 1100.00 |
| 1250.00 |
| 1250.00 |
|   800.00 |
+---------+
89 rows in set (0.00 sec)
```

第二步：将最大值筛选出来。也就是不在 sal 中的数据

实现结果：

```
mysql>
    SELECT
        sal
    FROM
        emp
    WHERE
        sal NOT IN ( SELECT a.sal FROM emp a, emp b WHERE a.sal < b.sal );
+---------+
| sal     |
+---------+
| 5000.00 |
+---------+
1 row in set (0.01 sec)
```

5.取得平均薪水最高的部门的部门编号(至少给出两种解决方案)?

方案一：

1.1 将部门薪资求均值并排序(升序)，作 a 表

```
mysql> select deptno,avg(sal) avgsal from emp group by deptno order by avgsal desc;
+--------+-------------+
| deptno | avgsal      |
+--------+-------------+
|     10 | 2916.666667 |
|     20 | 2175.000000 |
|     30 | 1566.666667 |
+--------+-------------+
3 rows in set (0.00 sec)
```

1.2 使用子查询方法，将 a 表作为临时表嵌套，取第一个数据

实现结果：

```
mysql>
    SELECT
        deptno
    FROM
        ( SELECT deptno, avg( sal ) avgsal FROM emp GROUP BY deptno ORDER BY avgsal DESC ) AS a
    LIMIT 1;
    +--------+
    | deptno |
```

```
+--------+
|      10 |
+--------+
1 row in set (0.00 sec)
```

方案二：

2.1 首先将部门薪资求均值，作 a 表

```
mysql> select deptno,avg(sal) avgsal from emp group by deptno;
+--------+-------------+
| deptno | avgsal      |
+--------+-------------+
|      20 | 2175.000000 |
|      30 | 1566.666667 |
|      10 | 2916.666667 |
+--------+-------------+
3 rows in set (0.00 sec)
```

2.2 利用子查询，找出平均薪资最高的

```
// 将 a 表当做临时表，从 from 后面嵌套
mysql> select max(avgsal) from (select avg(sal) avgsal from emp group by deptno) as a;
+-------------+
| max(avgsal) |
+-------------+
| 2916.666667 |
+-------------+
1 row in set (0.00 sec)
```

2.3 利用筛选条件，将 deptno 筛选出来

```
// where a.avgsal = (select max(avgsal) from (select avg(sal) avgsal from emp group by deptno) as b);
```

实现结果：

```
mysqp>
    SELECT
        a.deptno
    FROM
        ( SELECT deptno, avg( sal ) avgsal FROM emp GROUP BY deptno ) a
    WHERE
        a.avgsal = ( SELECT max( avgsal ) FROM ( SELECT avg( sal ) avgsal FROM emp GROUP BY deptno ) AS b );
+--------+
| deptno |
+--------+
|      10 |
+--------+
1 row in set (0.00 sec)
```

6.取得平均薪水最高的部门的部门名称?

实现思路：

第一步：计算每个部门的平均工资，并按照 avgsal 排序(降序)，取第一个数据

将此表作为临时表 a;

```
mysql> select deptno,avg(sal) avgsal from emp group by deptno order by avgsal desc limit 1;
+--------+-------------+
| deptno | avgsal      |
+--------+-------------+
|      10 | 2916.666667 |
+--------+-------------+
1 row in set (0.00 sec)
```

第二步：将 a 表和 deptno 表(b 表)连接查询

连接条件：on a.deptno=b.deptno

实现结果：

```
mysql>
    SELECT
        b.dname
    FROM
        ( SELECT deptno, avg( sal ) avgsal FROM emp GROUP BY deptno ORDER
BY avgsal DESC LIMIT 1 ) a
        JOIN dept b ON a.deptno = b.deptno;
    +------------+
    | dname      |
    +------------+
    | ACCOUNTING |
    +------------+
    1 row in set (0.00 sec)
```

7.求平均薪水的等级最低的部门的部门名称?(参考题目 6，把降序改为升序即可)

实现结果：

```
mysql>
    SELECT
        b.dname
    FROM
        ( SELECT deptno, avg( sal ) avgsal FROM emp GROUP BY deptno ORDER
BY avgsal ASC LIMIT 1 ) a
        JOIN dept b ON a.deptno = b.deptno;
+-------+
| dname |
+-------+
| SALES |
+-------+
1 row in set (0.00 sec)
```

8.取得比普通员工(员工代码没有在 mgr 字段上出现的)的最高薪水还要高的领导人姓名

实现思路：

第一步：找出 mgr 字段上出现的员工编号，也就是领导人的员工编号

```
// 一定要使用去重关键字 distinct
// 还有，注意 mgr 字段值为 null 的是领导
// 要先把 mgr 值为 null 的去掉，避免计算最大值的时候出错
    mysql> select distinct mgr from emp where mgr is not null;
    +------+
    | mgr  |
    +------+
    | 7902 |
    | 7698 |
    | 7839 |
    | 7566 |
    | 7788 |
    | 7782 |
    +------+
    6 rows in set (0.00 sec)
```

第二步：找出普通员工的最高薪资

```
mysql> select max(sal) from emp where empno not in (select distinct mgr from
emp where mgr is not null);
+----------+
| max(sal) |
+----------+
|  1600.00 |
+----------+
1 row in set (0.00 sec)
```

第三步：找出比普通员工的最高薪水还要高的领导人姓名

where 条件：

1.是领导，员工编号在领导人的员工编号中

empno in (select distinct mgr from emp where mgr is not null)

2.工资大于普通员工的最高薪水

sal > (select max(sal) from emp where empno not in (select distinct mgr from emp where mgr is not null))

实现结果：

```
mysql>
SELECT
    ename,
    sal
FROM
    emp
WHERE
    empno IN ( SELECT DISTINCT mgr FROM emp WHERE mgr IS NOT NULL )
    AND sal > (
    SELECT
        max( sal )
    FROM
        emp
    WHERE
    empno NOT IN ( SELECT DISTINCT mgr FROM emp WHERE mgr IS NOT NULL ));
+-------+---------+
| ename | sal     |
+-------+---------+
| JONES | 2975.00 |
| BLAKE | 2850.00 |
| CLARK | 2450.00 |
| SCOTT | 3000.00 |
| KING  | 5000.00 |
| FORD  | 3000.00 |
+-------+---------+
6 rows in set (0.01 sec)
```

9.取得薪水最高的前五名员工?

实现思路：先将 emp 表中的数据降序排列，然后使用 limit 取前 5 个

实现结果：

```
mysql>
    SELECT
        ename,
        sal
    FROM
        emp
    ORDER BY
        sal DESC
        LIMIT 5;
+-------+---------+
| ename | sal     |
+-------+---------+
| KING  | 5000.00 |
| SCOTT | 3000.00 |
| FORD  | 3000.00 |
| JONES | 2975.00 |
| BLAKE | 2850.00 |
+-------+---------+
```

5 rows in set (0.00 sec)

10.取得薪水最高的第六到第十名员工?

实现思路：先将 emp 表中的数据降序排列，然后使用 limit 取 6-10 名员工

limit 中，startIndex=5，length=5

实现结果：

```
mysql>
    SELECT
        ename,
        sal
    FROM
        emp
    ORDER BY
        sal DESC
        LIMIT 5,
        5;
或
mysql>
SELECT
    ename,
    sal
FROM
    emp
ORDER BY
    sal DESC,
    ename ASC
    LIMIT 5,
    5;
+--------+---------+
| ename  | sal     |
+--------+---------+
| CLARK  | 2450.00 |
| ALLEN  | 1600.00 |
| TURNER | 1500.00 |
| MILLER | 1300.00 |
| MARTIN | 1250.00 |
+--------+---------+
5 rows in set (0.00 sec)
```

11.取得最后入职的 5 名员工

实现思路：将 hiredate 数据排序(降序)，然后取前五个数据

实现结果：

```
mysql>
SELECT
    ename,
    hiredate
FROM
    emp
ORDER BY
    hiredate DESC
    LIMIT 5;
+--------+------------+
| ename  | hiredate   |
+--------+------------+
| ADAMS  | 1987-05-23 |
| SCOTT  | 1987-04-19 |
| MILLER | 1982-01-23 |
```

```
| JAMES    | 1981-12-03 |
| FORD     | 1981-12-03 |
+--------+------------+
5 rows in set (0.00 sec)
```

12.取得每个薪水等级有多少员工?

实现思路:

第一步:使用连接查询,为每个员工的工资进行分级。将此表作为 a 表

```
mysql> select b.grade from emp a join salgrade b on a.sal between b.losal and b.hisal;
+-------+
| grade |
+-------+
|     1 |
|     3 |
|     2 |
|     4 |
|     2 |
|     4 |
|     4 |
|     4 |
|     5 |
|     3 |
|     1 |
|     1 |
|     4 |
|     2 |
+-------+
14 rows in set (0.00 sec)
```

第二步:将 a 表作为临时表进行子查询,并按照 grade 字段进行分组

实现结果:

```
mysql>
SELECT
    grade,
    count(*)
FROM
    ( SELECT b.grade FROM emp a JOIN salgrade b ON a.sal BETWEEN b.losal AND b.hisal ) AS a
GROUP BY
    grade;
+-------+----------+
| grade | count(*) |
+-------+----------+
|     1 |        3 |
|     3 |        2 |
|     2 |        3 |
|     4 |        5 |
|     5 |        1 |
+-------+----------+
5 rows in set (0.00 sec)
```

13.

14.列出所有员工及领导的姓名

实现思路:使用自连接查询,获得员工的 mgr 编号,找该编号对应的员工

实现结果:

```
mysql>
SELECT
```

```
    a.ename '员工',
    ifnull( b.ename, '没有领导' ) '领导'
FROM
    emp a
    LEFT JOIN emp b ON a.mgr = b.empno;
+--------+--------------+
| 员工   | 领导         |
+--------+--------------+
| SMITH  | FORD         |
| ALLEN  | BLAKE        |
| WARD   | BLAKE        |
| JONES  | KING         |
| MARTIN | BLAKE        |
| BLAKE  | KING         |
| CLARK  | KING         |
| SCOTT  | JONES        |
| KING   | 没有领导     |
| TURNER | BLAKE        |
| ADAMS  | SCOTT        |
| JAMES  | BLAKE        |
| FORD   | JONES        |
| MILLER | CLARK        |
+--------+--------------+
14 rows in set (0.00 sec)
```

15.列出受雇日期早于其直接上级的所有员工的编号,姓名,部门名称?

实现思路:

第一步:使用自连接查询,获得受雇日期早于其直接上级的所有员工的编号,姓名,部门编号

并将此表作 e 表

连接条件:a.mgr = b.empno

筛选条件:where a.hiredate < b.hiredate

```
mysql> select a.empno,a.ename,a.deptno from emp a left join emp b on a.mgr = b.empno where a.hiredate < b.hiredate;
+-------+-------+--------+
| empno | ename | deptno |
+-------+-------+--------+
|  7369 | SMITH |     20 |
|  7499 | ALLEN |     30 |
|  7521 | WARD  |     30 |
|  7566 | JONES |     20 |
|  7698 | BLAKE |     30 |
|  7782 | CLARK |     10 |
+-------+-------+--------+
6 rows in set (0.00 sec)
```

第二步:将 e 表和 dept 表(d 表)连接查询

连接条件:e.deptno = d.deptno

实现结果:

```
mysql>
SELECT
    e.empno,
    e.ename,
    d.dname
FROM
    (
    SELECT
        a.empno,
```

```
        a.ename,
        a.deptno
    FROM
        emp a
        LEFT JOIN emp b ON a.mgr = b.empno
    WHERE
        a.hiredate < b.hiredate
    ) AS e
    JOIN dept d ON e.deptno = d.deptno;
+-------+-------+------------+
| empno | ename | dname      |
+-------+-------+------------+
|  7369 | SMITH | RESEARCH   |
|  7499 | ALLEN | SALES      |
|  7521 | WARD  | SALES      |
|  7566 | JONES | RESEARCH   |
|  7698 | BLAKE | SALES      |
|  7782 | CLARK | ACCOUNTING |
+-------+-------+------------+
6 rows in set (0.00 sec)
```

16.列出部门名称和这些部门的员工信息,同时列出那些没有员工的部门?

实现思路：dept 和 emp 进行左外连接查询，要求列出那些没有员工的部门，这说明 dept 是主表

实现结果：

```
mysql>
SELECT
    d.dname,
    e.*
FROM
    dept d
    RIGHT JOIN emp e ON e.deptno = d.deptno;
+------------+-------+--------+-----------+------+------------+---------+------
---+--------+
| dname      | EMPNO | ENAME  | JOB       | MGR  | HIREDATE   | SAL     |
COMM    | DEPTNO |
+------------+-------+--------+-----------+------+------------+---------+------
---+--------+
| ACCOUNTING |  7934 | MILLER | CLERK     | 7782 | 1982-01-23 | 1300.00 |
NULL |     10 |
| ACCOUNTING |  7839 | KING   | PRESIDENT | NULL | 1981-11-17 | 5000.00 |
NULL |     10 |
| ACCOUNTING |  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 | 2450.00 |
NULL |     10 |
| RESEARCH   |  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 | 3000.00 |
NULL |     20 |
| RESEARCH   |  7876 | ADAMS  | CLERK     | 7788 | 1987-05-23 | 1100.00 |
NULL |     20 |
| RESEARCH   |  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 | 3000.00 |
NULL |     20 |
| RESEARCH   |  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 | 2975.00 |
NULL |     20 |
| RESEARCH   |  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |  800.00 |
NULL |     20 |
| SALES      |  7900 | JAMES  | CLERK     | 7698 | 1981-12-03 |  950.00 |    NULL
|     30 |
| SALES      |  7844 | TURNER | SALESMAN  | 7698 | 1981-09-08 | 1500.00 |    0.00
|     30 |
| SALES      |  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 | 2850.00 |
NULL |     30 |
```

```
| SALES      |  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 | 1250.00 | 1400.00 |
30 |
| SALES      |  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 | 1250.00 |
500.00 |      30 |
| SALES      |  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 | 1600.00 |  300.00
|      30 |
| OPERATIONS |  NULL | NULL   | NULL      | NULL | NULL       |    NULL |
NULL |    NULL |
+------------+-------+--------+-----------+------+------------+---------+------
---+--------+
15 rows in set (0.00 sec)
```

17.列出至少有 5 个员工的所有部门?

实现思路：

第一步：使用分组的方式计算每个部门有多少个员工，作 b 表

```
mysql> select deptno,count(*) from emp group by deptno;
+--------+----------+
| deptno | count(*) |
+--------+----------+
|     20 |        5 |
|     30 |        6 |
|     10 |        3 |
+--------+----------+
3 rows in set (0.00 sec)
```

第二步：将 dept 表(a 表)和 b 表等值连接查询

连接条件：on a.deptno = b.deptno

筛选条件：where b.count >= 5

实现结果：

```
mysql>
SELECT
    a.dname,
    b.count
FROM
    dept a
    JOIN ( SELECT deptno, count(*) count FROM emp GROUP BY deptno ) b ON
a.deptno = b.deptno
WHERE
    b.count >= 5;
+----------+-------+
| dname    | count |
+----------+-------+
| RESEARCH |     5 |
| SALES    |     6 |
+----------+-------+
2 rows in set (0.00 sec)
```

18.列出薪金比"SMITH"多的所有员工信息?

实现思路：

第一步：将 SMITH 的薪资查询出来，然后在使用嵌套查询

```
mysql> select sal from emp where ename = 'SMITH';
+--------+
| sal    |
+--------+
| 800.00 |
+--------+
1 row in set (0.00 sec)
```

第二步：使用 where 子查询方法

实现结果：

```
mysql>
SELECT
    *
FROM
    emp
WHERE
    sal > ( SELECT sal FROM emp WHERE ename = 'SMITH' );
+-------+--------+-----------+------+------------+---------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   | SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+---------+---------+--------+
|  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 |
+-------+--------+-----------+------+------------+---------+---------+--------+
13 rows in set (0.00 sec)
```

19.列出所有"CLERK"(办事员)的姓名及其部门名称,部门的人数?

实现思路：

第一步：统计工作岗位为"CLERK"(办事员)的员工姓名和部门编号，作 a 表

```
mysql> select ename,deptno from emp where job = 'CLERK';
+--------+--------+
| ename  | deptno |
+--------+--------+
| SMITH  |     20 |
| ADAMS  |     20 |
| JAMES  |     30 |
| MILLER |     10 |
+--------+--------+
4 rows in set (0.00 sec)
```

第二步：使用分组的方式计算每个部门有多少个员工，作 c 表

```
mysql> select deptno,count(*) count from emp group by deptno;
+--------+-------+
| deptno | count |
+--------+-------+
|     20 |     5 |
|     30 |     6 |
|     10 |     3 |
+--------+-------+
3 rows in set (0.00 sec)
```

第三步：使用连接查询，首先是将 a 表和 dept 表(b 表)连接，得到员工所在部门的名称

连接条件：a.deptno = b.deptno

再将将 a 表和 c 表连接，统计每个部门的人数

连接条件：a.deptno = c.deptno

实现结果：

```
mysql>
    SELECT
    a.ename,
```

```
        b.dname,
        c.count
    FROM
        ( SELECT ename, deptno FROM emp WHERE job = 'CLERK' ) AS a
        JOIN dept b ON a.deptno = b.deptno
        JOIN ( SELECT deptno, count(*) count FROM emp GROUP BY deptno ) AS c ON
a.deptno = c.deptno;
    +--------+------------+-------+
    | ename  | dname      | count |
    +--------+------------+-------+
    | SMITH  | RESEARCH   |     5 |
    | ADAMS  | RESEARCH   |     5 |
    | JAMES  | SALES      |     6 |
    | MILLER | ACCOUNTING |     3 |
    +--------+------------+-------+
    4 rows in set (0.00 sec)
```

20.列出最低薪金大于 1500 的各种工作及从事此工作的全部雇员人数?

实现思路：

第一步：首先查询各个工作岗位的最低工资，作 a 表

```
mysql> select job,min(sal) minsal from emp group by job;
+-----------+---------+
| job       | minsal  |
+-----------+---------+
| CLERK     |  800.00 |
| SALESMAN  | 1250.00 |
| MANAGER   | 2450.00 |
| ANALYST   | 3000.00 |
| PRESIDENT | 5000.00 |
+-----------+---------+
5 rows in set (0.00 sec)
```

第二步：查询各个工作岗位的员工人数，作 b 表

```
mysql> select job,count(*) count from emp group by job;
+-----------+-------+
| job       | count |
+-----------+-------+
| CLERK     |     4 |
| SALESMAN  |     4 |
| MANAGER   |     3 |
| ANALYST   |     2 |
| PRESIDENT |     1 |
+-----------+-------+
5 rows in set (0.00 sec)
```

第三步：将 a、b 表连接查询

连接条件：on a.job = b.job

筛选条件：where a.minsal > 1500

实现结果：

```
mysql>
SELECT
    a.job,
    b.count
FROM
    ( SELECT job, min( sal ) minsal FROM emp GROUP BY job ) AS a
    JOIN ( SELECT job, count(*) count FROM emp GROUP BY job ) AS b ON a.job =
b.job
WHERE
    a.minsal > 1500;
+-----------+-------+
| job       | count |
```

```
+-----------+-------+
| MANAGER   |     3 |
| ANALYST   |     2 |
| PRESIDENT |     1 |
+-----------+-------+
3 rows in set (0.00 sec)
```

21.列出在部门"SALES"<销售部>工作的员工的姓名,假定不知道销售部的部门编号?

实现思路:

第一步:把非销售人员的部门编号统计出来

```
mysql> select deptno from dept where dname <> 'SALES';
+--------+
| deptno |
+--------+
|     10 |
|     20 |
|     40 |
+--------+
3 rows in set (0.00 sec)
```

第二步:where 条件筛选,那么部门编号不在第一步的 deptno 表中的员工即为"SALES"<销售部>工作的员工

实现结果:

```
mysql>
SELECT
    ename
FROM
    emp
WHERE
    deptno NOT IN ( SELECT deptno FROM dept WHERE dname <> 'SALES' );
+--------+
| ename  |
+--------+
| ALLEN  |
| WARD   |
| MARTIN |
| BLAKE  |
| TURNER |
| JAMES  |
+--------+
6 rows in set (0.00 sec)
```

22.列出薪金高于公司平均薪金的所有员工,所在部门,上级领导,雇员的工资等级?

实现思路:

第一步:首先计算公司员工的平均薪资

```
mysql> SELECT avg( sal ) FROM emp;
+-------------+
| avg( sal )  |
+-------------+
| 2073.214286 |
+-------------+
1 row in set (0.00 sec)
```

第二步:使用 where 子查询方法,统计公司中高于平均值的员工信息,包括姓名、工资、部门编号

该表记为 e1 表

```
mysql> SELECT ename,sal,deptno FROM emp WHERE sal > ( SELECT avg( sal ) FROM emp );
+-------+---------+--------+
| ename | sal     | deptno |
```

```
+-------+---------+--------+
| JONES | 2975.00 |       20 |
| BLAKE | 2850.00 |       30 |
| CLARK | 2450.00 |       10 |
| SCOTT | 3000.00 |       20 |
| KING  | 5000.00 |       10 |
| FORD  | 3000.00 |       20 |
+-------+---------+--------+
6 rows in set (0.00 sec)
```

第三步：根据 e1 表中的部门编号信息，结合 dept 表(e2 表)使用连接查询，
得到 e1 表中员工所在部门

第四步：使用左外连接方法查询员工的上级领导，如果员工的上级领导不存在，
则该员工的上级领导记为 null，该查询结果表记为 e3 表

```
mysql> SELECT a.ename, b.ename ename_e FROM emp a LEFT JOIN emp b ON a.mgr = b.empno ;
+--------+---------+
| ename  | ename_e |
+--------+---------+
| SMITH  | FORD    |
| ALLEN  | BLAKE   |
| WARD   | BLAKE   |
| JONES  | KING    |
| MARTIN | BLAKE   |
| BLAKE  | KING    |
| CLARK  | KING    |
| SCOTT  | JONES   |
| KING   | NULL    |
| TURNER | BLAKE   |
| ADAMS  | SCOTT   |
| JAMES  | BLAKE   |
| FORD   | JONES   |
| MILLER | CLARK   |
+--------+---------+
14 rows in set (0.00 sec)
```

第五步：利用 e1 表中的 deptno，结合 e2 表使用连接查询，获得员工的所在部门；
利用 e1 表中的 ename，结合 e2 表使用连接查询，获得员工的上级领导；
利用 e1 表中的 sal，结合 e3 表使用连接查询，获得员工的薪资等级。

实现结果：

```
mysql>
SELECT
    e1.ename '姓名',
    e2.dname '部门名称',
    IFNULL(e3.ename_e,'无领导') '上级领导',
    e4.grade '工资等级'
FROM
    (
    SELECT
        ename,
        sal,
        deptno
    FROM
        emp
    WHERE
    sal > ( SELECT avg( sal ) FROM emp )) e1
    JOIN dept e2 ON e1.deptno = e2.deptno
    JOIN ( SELECT a.ename, b.ename ename_e FROM emp a LEFT JOIN emp b ON a.mgr = b.empno ) e3 ON e1.ename = e3.ename
    JOIN (
```

```
        SELECT DISTINCT
            a.sal,
            b.grade
        FROM
            emp a
            JOIN salgrade b
        WHERE
            a.sal BETWEEN b.losal
        AND b.hisal
        ) e4 ON e1.sal = e4.sal;
    +--------+-------------+-------------+-------------+
    | 姓名    | 部门名称      | 上级领导      | 工资等级      |
    +--------+-------------+-------------+-------------+
    | JONES  | RESEARCH    | KING        |           4 |
    | BLAKE  | SALES       | KING        |           4 |
    | CLARK  | ACCOUNTING  | KING        |           4 |
    | SCOTT  | RESEARCH    | JONES       |           4 |
    | KING   | ACCOUNTING  | 无领导       |           5 |
    | FORD   | RESEARCH    | JONES       |           4 |
    +--------+-------------+-------------+-------------+
    6 rows in set (0.00 sec)
```

23.列出与"SCOTT"从事相同工作的所有员工及部门名称?

实现思路:

第一步:首先统计一下,使用自连接方法查询和'SCOTT'具有相同工作岗位的员工信息,

记录该员工的姓名、部门编号,记该表为 e1

连接条件:on a.job = b.job

筛选条件:where a.ename = 'SCOTT' and b.ename <> 'SCOTT'

```
mysql> select b.ename,b.deptno from emp a join emp b on a.job = b.job where a.ename = 'SCOTT' and b.ename <> 'SCOTT';
+-------+--------+
| ename | deptno |
+-------+--------+
| FORD  |     20 |
+-------+--------+
1 row in set (0.00 sec)
```

第二步:结合 dept 表,根据 deptno 信息使用连接查询方法,获得 e1 表中的员工所在的部门名称

实现结果:

```
mysql>
SELECT
    e1.ename,
    e2.dname
FROM
    (
    SELECT
        b.ename,
        b.deptno
    FROM
        emp a
        JOIN emp b ON a.job = b.job
    WHERE
        a.ename = 'SCOTT'
        AND b.ename <> 'SCOTT'
    ) e1
    JOIN dept e2 ON e1.deptno = e2.deptno;
+-------+----------+
| ename | dname    |
```

```
+-------+----------+
| FORD  | RESEARCH |
+-------+----------+
1 row in set (0.00 sec)
```

24.列出薪金等于部门 30 中员工的薪金的其他员工的姓名和薪金?

实现思路:

第一步:使用连接查询方法,查找薪金等于部门 30 中员工的薪金的其他员工的姓名和部门编号

记为 e1 表

```
mysql> select ename,sal,deptno from emp where sal in (select sal from emp where deptno = '30') and deptno <> '30';
Empty set (0.00 sec)
```

第二步:根据 e1 表中的 deptno 信息,结合 dept 表(e2 表)使用连接查询,

查找 e1 表中的员工所在的部门

实现结果:

```
mysql>
SELECT
    e1.ename,
    e1.sal,
    e2.dname
FROM
    ( SELECT ename, sal, deptno FROM emp WHERE sal IN ( SELECT sal FROM emp WHERE deptno = '30' ) AND deptno <> '30' ) e1
    JOIN dept e2 ON e1.deptno = e2.deptno;
Empty set (0.00 sec)
```

25.列出薪金高于在部门 30 工作的所有员工的薪金的员工姓名和薪金.部门名称?

实现思路:

第一步:使用连接查询方法,查找薪金高于部门 30 中员工的薪金的其他员工的姓名和部门编号

也就是说在其他部门寻找薪资高于 30 部门的最高薪资的员工信息,查询这类员工的姓名、薪资和部门编号

记为 e1 表

```
mysql> select ename,sal,deptno from emp where sal > (select max(sal) from emp where deptno = '30') and deptno <> '30';
+-------+---------+--------+
| ename | sal     | deptno |
+-------+---------+--------+
| JONES | 2975.00 |     20 |
| SCOTT | 3000.00 |     20 |
| KING  | 5000.00 |     10 |
| FORD  | 3000.00 |     20 |
+-------+---------+--------+
4 rows in set (0.00 sec)
```

第三步:根据 e1 表中的 deptno 信息,结合 dept 表(e2 表)使用连接查询,

查找 e1 表中的员工所在的部门

实现结果:

```
mysql>
SELECT
    e1.ename,
    e1.sal,
    e2.dname
FROM
    (
    SELECT
```

```
        ename,
        sal,
        deptno
    FROM
        emp
    WHERE
        sal > ( SELECT max( sal ) FROM emp WHERE deptno = '30' )
        AND deptno <> '30'
    ) e1
    JOIN dept e2 ON e1.deptno = e2.deptno;
+-------+---------+------------+
| ename | sal     | dname      |
+-------+---------+------------+
| JONES | 2975.00 | RESEARCH   |
| SCOTT | 3000.00 | RESEARCH   |
| KING  | 5000.00 | ACCOUNTING |
| FORD  | 3000.00 | RESEARCH   |
+-------+---------+------------+
4 rows in set (0.00 sec)
```

26.列出在每个部门工作的员工数量,平均工资?

实现思路:

第一步:emp 表中按照 deptno 分组,计算每组的员工数量、平均工资和部门编号

记该查询结果表为 e1 表

```
mysql> select deptno,count(*) as count,avg(sal) as avgsal from emp group by deptno;
+--------+-------+-------------+
| deptno | count | avgsal      |
+--------+-------+-------------+
|     20 |     5 | 2175.000000 |
|     30 |     6 | 1566.666667 |
|     10 |     3 | 2916.666667 |
+--------+-------+-------------+
3 rows in set (0.00 sec)
```

第二步:

结合 dept 表(e2 表)使用连接查询,查询部门名称

实现结果:

```
mysql>
SELECT
    e2.dname,
    e1.count,
    e1.avgsal
FROM
    ( SELECT deptno, count(*) AS count, avg( sal ) AS avgsal FROM emp GROUP BY deptno ) e1
    JOIN dept e2 ON e1.deptno = e2.deptno;
+------------+-------+-------------+
| dname      | count | avgsal      |
+------------+-------+-------------+
| ACCOUNTING |     3 | 2916.666667 |
| RESEARCH   |     5 | 2175.000000 |
| SALES      |     6 | 1566.666667 |
+------------+-------+-------------+
3 rows in set (0.00 sec)
```

27.列出所有员工的姓名、部门名称和工资?

实现思路:

第一步:从 emp 表中获得员工姓名、薪资和部门编号

记作 e1 表

```
mysql> select ename,sal,deptno from emp;
+--------+---------+--------+
| ename  | sal     | deptno |
+--------+---------+--------+
| SMITH  |  800.00 |     20 |
| ALLEN  | 1600.00 |     30 |
| WARD   | 1250.00 |     30 |
| JONES  | 2975.00 |     20 |
| MARTIN | 1250.00 |     30 |
| BLAKE  | 2850.00 |     30 |
| CLARK  | 2450.00 |     10 |
| SCOTT  | 3000.00 |     20 |
| KING   | 5000.00 |     10 |
| TURNER | 1500.00 |     30 |
| ADAMS  | 1100.00 |     20 |
| JAMES  |  950.00 |     30 |
| FORD   | 3000.00 |     20 |
| MILLER | 1300.00 |     10 |
+--------+---------+--------+
14 rows in set (0.00 sec)
```

第二步：

利用 dept 表(e2 表)，使用连接查询方法获得每个员工的工作部门信息

连接条件：on e1.deptno = e2.deptno

实现结果：

```
mysql>
SELECT
    e1.ename,
    e2.dname,
    e1.sal
FROM
    ( SELECT ename, sal, deptno FROM emp ) e1
    JOIN dept e2 ON e1.deptno = e2.deptno;
+--------+------------+---------+
| ename  | dname      | sal     |
+--------+------------+---------+
| SMITH  | RESEARCH   |  800.00 |
| ALLEN  | SALES      | 1600.00 |
| WARD   | SALES      | 1250.00 |
| JONES  | RESEARCH   | 2975.00 |
| MARTIN | SALES      | 1250.00 |
| BLAKE  | SALES      | 2850.00 |
| CLARK  | ACCOUNTING | 2450.00 |
| SCOTT  | RESEARCH   | 3000.00 |
| KING   | ACCOUNTING | 5000.00 |
| TURNER | SALES      | 1500.00 |
| ADAMS  | RESEARCH   | 1100.00 |
| JAMES  | SALES      |  950.00 |
| FORD   | RESEARCH   | 3000.00 |
| MILLER | ACCOUNTING | 1300.00 |
+--------+------------+---------+
14 rows in set (0.00 sec)
```

28.列出所有部门的详细信息和人数?

实现思路：

第一步：按照 deptno 分组获得每个部门的部门编号和部门员工数量

将该表记为 e1 表

```
mysql> select deptno,count(*) from emp group by deptno;
+--------+----------+
```

```
| deptno | count(*) |
+--------+----------+
|     20 |        5 |
|     30 |        6 |
|     10 |        3 |
+--------+----------+
3 rows in set (0.00 sec)
```

第二步：结合 dept 表(e2 表)使用连接查询，获得部门的名称、位置信息

注意使用的是左连接方法，还有 ifnull

实现结果：

```
mysql>
SELECT
    e2.deptno '部门编号',
    e2.dname '部门名称',
    e2.loc '部门位置',
    ifnull( e1.count, 0 ) '人数'
FROM
    ( SELECT deptno, count(*) count FROM emp GROUP BY deptno ) e1
    RIGHT JOIN dept e2 ON e1.deptno = e2.deptno;
+--------+------------+----------+--------------------+
| deptno | dname      | loc      | ifnull(e1.count,0) |
+--------+------------+----------+--------------------+
|     10 | ACCOUNTING | NEW YORK |                  3 |
|     20 | RESEARCH   | DALLAS   |                  5 |
|     30 | SALES      | CHICAGO  |                  6 |
|     40 | OPERATIONS | BOSTON   |                  0 |
+--------+------------+----------+--------------------+
4 rows in set (0.00 sec)
```

29.列出各种工作的最低工资及从事此工作的雇员姓名？

实现思路：

第一步：emp 表中按照 job 分组，计算每个工作岗位的最低工资和部门编号

记该表为 e2 表

```
mysql> select job,min(sal) minsal from emp group by job;
+-----------+---------+
| job       | minsal  |
+-----------+---------+
| CLERK     |  800.00 |
| SALESMAN  | 1250.00 |
| MANAGER   | 2450.00 |
| ANALYST   | 3000.00 |
| PRESIDENT | 5000.00 |
+-----------+---------+
5 rows in set (0.00 sec)
```

第二步：利用 dept 表(e1 表)，使用连接查询的方法，获得统计数据

mysql>

实现结果：

```
mysql>
SELECT
    e1.*
FROM
    emp e1
    JOIN ( SELECT job, min( sal ) minsal FROM emp GROUP BY job ) e2 ON e1.job =
e2.job
```

```
    AND e1.sal = e2.minsal;
+-------+--------+-----------+------+------------+---------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   | SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+---------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |  800.00 |    NULL |     20 |
|  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 | 1250.00 |  500.00 |     30 |
|  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 | 1250.00 | 1400.00 |     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 |
|  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 | 3000.00 |    NULL |     20 |
+-------+--------+-----------+------+------------+---------+---------+--------+
7 rows in set (0.00 sec)
```

30.列出各个部门的 MANAGER(领导)的最低薪金?

实现思路：

第一步：筛选出领导的工号

```
mysql> select distinct mgr from emp where mgr is not null ;
+------+
| mgr  |
+------+
| 7902 |
| 7698 |
| 7839 |
| 7566 |
| 7788 |
| 7782 |
+------+
6 rows in set (0.00 sec)
```

第二步：根据

实现结果：将 emp 表按照 deptno 进行分组，然后按照计算出各个部门的 MANAGER(领导)的最低薪金

筛选条件：where empno in (select distinct mgr from emp where mgr is not null)

```
mysql>
SELECT
    deptno,
    min( sal )
FROM
    emp
WHERE
    empno IN ( SELECT DISTINCT mgr FROM emp WHERE mgr IS NOT NULL )
GROUP BY
    deptno;
+--------+----------+
| deptno | min(sal) |
+--------+----------+
|     20 |  2975.00 |
|     30 |  2850.00 |
|     10 |  2450.00 |
+--------+----------+
3 rows in set (0.00 sec)
```

31.列出所有员工的年工资,按年薪从低到高排序?

实现思路：年工资 = (月工资 sal + 津贴 comm) * 12;

实现结果：

```
mysql>
SELECT
```

```
    ename,(
    sal + ifnull( comm, 0 )) * 12 sal
FROM
    emp
ORDER BY
    sal;
+--------+----------+
| ename  | sal      |
+--------+----------+
| SMITH  |  9600.00 |
| JAMES  | 11400.00 |
| ADAMS  | 13200.00 |
| MILLER | 15600.00 |
| TURNER | 18000.00 |
| WARD   | 21000.00 |
| ALLEN  | 22800.00 |
| CLARK  | 29400.00 |
| MARTIN | 31800.00 |
| BLAKE  | 34200.00 |
| JONES  | 35700.00 |
| SCOTT  | 36000.00 |
| FORD   | 36000.00 |
| KING   | 60000.00 |
+--------+----------+
14 rows in set (0.00 sec)
```

32.求出员工领导的薪水超过 3000 的员工名称与领导名称?

实现思路：

第一步：利用自连接查询，将员工与领导的对应关系连接起来

连接条件：on a.mgr = b.empno

记此表为 e1 表

```
mysql> select a.ename,b.ename ename_e from emp a join emp b on a.mgr = b.empno;
+--------+---------+
| ename  | ename_e |
+--------+---------+
| SMITH  | FORD    |
| ALLEN  | BLAKE   |
| WARD   | BLAKE   |
| JONES  | KING    |
| MARTIN | BLAKE   |
| BLAKE  | KING    |
| CLARK  | KING    |
| SCOTT  | JONES   |
| TURNER | BLAKE   |
| ADAMS  | SCOTT   |
| JAMES  | BLAKE   |
| FORD   | JONES   |
| MILLER | CLARK   |
+--------+---------+
13 rows in set (0.00 sec)
```

第二步：结合 emp 表(e2 表)使用连接查询方法，进行查询

连接条件：on e1.ename_e = e2.ename

筛选条件：where e2.sal > 3000

实现结果：

```
mysql>
SELECT
    e1.ename,
```

```
        e1.ename_e
    FROM
        ( SELECT a.ename, b.ename ename_e FROM emp a JOIN emp b ON a.mgr = b.empno ) e1
        JOIN emp e2 ON e1.ename_e = e2.ename
    WHERE
        e2.sal > 3000;
    +-------+---------+
    | ename | ename_e |
    +-------+---------+
    | CLARK | KING    |
    | BLAKE | KING    |
    | JONES | KING    |
    +-------+---------+
    3 rows in set (0.00 sec)
```

33.求出部门名称中,带'S'字符的部门员工的工资合计、部门人数?

实现思路:

第一步:使用 emp 表,按照 deptno 进行分组,获得每个的部门的部门编号、部门人数、部门总工资

记作 e2 表

```
mysql> select deptno,count(*) count,sum(sal) sal from emp group by deptno;
+--------+-------+----------+
| deptno | count | sal      |
+--------+-------+----------+
|     20 |     5 | 10875.00 |
|     30 |     6 |  9400.00 |
|     10 |     3 |  8750.00 |
+--------+-------+----------+
3 rows in set (0.00 sec)
```

第二步:结合 dept 表(e1 表)使用连接查询,注意使用左外连接,并使用模糊查询方法筛选

连接条件:on e1.deptno = e2.deptno

筛选条件:where e1.dname like '%S%'

实现结果:

```
mysql>
SELECT
    e1.dname,
    ifnull( e2.sal, 0 ) sal,
    ifnull( e2.count, 0 ) count
FROM
    dept e1
    LEFT JOIN ( SELECT deptno, count(*) count, sum( sal ) sal FROM emp GROUP BY deptno ) e2 ON e1.deptno = e2.deptno
WHERE
    e1.dname LIKE '%S%';
+------------+----------+-------+
| dname      | sal      | count |
+------------+----------+-------+
| RESEARCH   | 10875.00 |     5 |
| SALES      |  9400.00 |     6 |
| OPERATIONS |     0.00 |     0 |
+------------+----------+-------+
3 rows in set (0.00 sec)
```

34.给任职日期超过 30 年的员工加薪 10%?

实现思路:

第一步:计算当前时间减去 emp 表中的时间大于 30 的作为筛选条件

第二步：给任职日期超过 30 年的员工加薪 10%

实现结果：

```
mysql>
SELECT
    ename,
    sal * 1.1 sal
FROM
    emp
WHERE
    (
    DATE_FORMAT( NOW(), '%Y' )- DATE_FORMAT( hiredate, '%Y' ))> 30;
+--------+---------+
| ename  | sal     |
+--------+---------+
| SMITH  |  880.00 |
| ALLEN  | 1760.00 |
| WARD   | 1375.00 |
| JONES  | 3272.50 |
| MARTIN | 1375.00 |
| BLAKE  | 3135.00 |
| CLARK  | 2695.00 |
| SCOTT  | 3300.00 |
| KING   | 5500.00 |
| TURNER | 1650.00 |
| ADAMS  | 1210.00 |
| JAMES  | 1045.00 |
| FORD   | 3300.00 |
| MILLER | 1430.00 |
+--------+---------+
14 rows in set (0.00 sec)
```