

目录

内容.....	6
1、数据库概述及数据准备.....	6
1.1、SQL 概述.....	6
1.2、什么是数据库.....	6
1.3、MySql 概述.....	6
1.4、MySql 的安装.....	6
1.5、表.....	15
1.6、SQL 的分类.....	16
1.7、导入演示数据.....	16
1.8、表结构描述.....	16
2、常用命令.....	17
2.1、查看 mysql 版本.....	17
2.2、创建数据库.....	17
2.3、查询当前使用的数据库.....	18
2.4、终止一条语句.....	18
2.5、退出 mysql.....	18
3、查看“演示数据”的表结构.....	18
3.1、查看和指定现有的数据库.....	18
3.2、指定当前缺省数据库.....	18
3.3、查看当前使用的库.....	19
3.4、查看当前库中的表.....	19
3.5、查看其他库中的表.....	19
3.6、查看表的结构.....	19
3.7、查看表的创建语句.....	20
4、简单的查询.....	20
4.1、查询一个字段.....	20
4.2、查询多个字段.....	21
4.3、查询全部字段.....	22
4.4、计算员工的年薪.....	22
4.5、将查询出来的字段显示为中文.....	23

5、条件查询.....	23
5.1、等号操作符.....	24
5.2、 \neq 操作符.....	25
5.3、between ... and ...操作符.....	26
5.4、is null.....	27
5.5、and.....	27
5.6、or.....	28
5.7、表达式的优先级.....	28
5.8、in.....	29
5.9、not.....	29
5.10、like.....	31
6、排序数据.....	32
6.1、单一字段排序.....	32
6.2、手动指定排序顺序.....	33
6.3、多个字段排序.....	34
6.4、使用字段的位置来排序.....	35
7、分组函数/聚合函数/多行处理函数.....	35
7.1、count.....	36
7.2、sum.....	37
7.3、avg.....	38
7.4、max.....	38
7.5、min.....	38
7.6、组合聚合函数.....	39
8、分组查询.....	39
8.1、group by.....	39
8.2、having.....	41
8.3、select 语句总结.....	42
9、连接查询.....	42
9.1、SQL92 语法.....	42
9.2、SQL99 语法.....	46
10、子查询.....	47
10.1、在 where 语句中使用子查询，也就是在 where 语句中加入 select 语句.....	48

10.2、在 from 语句中使用子查询，可以将该子查询看做一张表.....	49
10.3、在 select 语句中使用子查询.....	50
11、union.....	51
11.1、union 可以合并集合（相加）	51
12、limit 的使用.....	51
12.1、取得前 5 条数据.....	52
12.2、从第二条开始取两条数据.....	52
12.3、取得薪水最高的前 5 名.....	52
13、表.....	53
13.1、创建表.....	53
13.2、增加/删除/修改表结构.....	55
13.3、添加、修改和删除.....	57
13.4、创建表加入约束.....	61
13.5、t_student 和 t_classes 完整示例.....	67
14、存储引擎（了解）	67
14.1、存储引擎的使用.....	67
14.2、常用的存储引擎.....	69
14.3、选择合适的存储引擎.....	70
15、事务.....	70
15.1、概述.....	70
15.2、事务的提交与回滚演示.....	71
15.3、自动提交模式.....	72
15.4、事务的隔离级别.....	73
16、索引.....	77
17、视图.....	79
17.1、什么是视图.....	79
17.2、创建视图.....	80
17.3、修改视图.....	80
17.4、删除视图.....	80
18、DBA 命令（了解）	80
18.1、新建用户.....	80
18.2、授权.....	81

18.3、回收权限.....	81
18.4、导出导入.....	82
19、数据库设计的三范式.....	82
19.1、第一范式.....	82
19.2、第二范式.....	83
19.3、第三范式.....	84
19.4、三范式总结.....	84
、作业.....	86
1、取得每个部门最高薪水的人员名称.....	86
2、哪些人的薪水在部门的平均薪水之上.....	86
3、取得部门中（所有人的）平均的薪水等级，如下：	86
4、不准用组函数（Max），取得最高薪水（给出两种解决方案）	87
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）	87
6、取得平均薪水最高的部门的部门名称.....	87
7、求平均薪水的等级最低的部门的部门名称.....	88
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.....	88
9、取得薪水最高的前五名员工.....	88
10、取得薪水最高的第六到第十名员工.....	89
11、取得最后入职的 5 名员工.....	89
12、取得每个薪水等级有多少员工.....	89
13、面试题.....	90
14、列出所有员工及领导的姓名.....	91
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.....	92
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..	92
17、列出至少有 5 个员工的所有部门.....	93
18、列出薪金比"SMITH"多的所有员工信息.....	93
19、列出所有"CLERK"(办事员)的姓名及其部门名称,部门的人数.....	94
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....	94
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.....	94
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.....	95

23、列出与"SCOTT"从事相同工作的所有员工及部门名称.....	95
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.....	95
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.....	95
26、列出在每个部门工作的员工数量,平均工资和平均服务期限.....	96
27、列出所有员工的姓名、部门名称和工资。	96
28、列出所有部门的详细信息和人数.....	96
29、列出各种工作的最低工资及从事此工作的雇员姓名.....	97
30、列出各个部门的 MANAGER(领导)的最低薪金.....	97
31、列出所有员工的年工资,按年薪从低到高排序.....	97
32、求出员工领导的薪水超过 3000 的员工名称与领导名称.....	98
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.....	98
34、给任职日期超过 30 年的员工加薪 10%.....	98

内容

1、数据库概述及数据准备

1.1、SQL 概述

SQL，一般发音为 sequel，SQL 的全称 Structured Query Language)，SQL 用来和数据库打交道，完成和数据库的通信，SQL 是一套标准。但是每一个数据库都有自己的特性别的数据库没有，当使用这个数据库特性相关的功能，这时 SQL 语句可能就不是标准了。(90%以上的 SQL 都是通用的)

1.2、什么是数据库

数据库，通常是一个或一组文件，保存了一些符合特定规格的数据，数据库对应的英语单词是 DataBase, 简称 :DB, 数据库软件称为数据库管理系统（DBMS），全称为 DataBase Management System，如：Oracle、SQL Server、MySQL、Sybase、informix、DB2、interbase、PostgreSql。

1.3、MySQL 概述

MySQL 最初是由“MySQL AB”公司开发的一套关系型数据库管理系统（RDBMS-Relational Database Management System）。

MySQL 不仅是最流行的开源数据库，而且是业界成长最快的数据库，每天有超过 7 万次的下载量，其应用范围从大型企业到专有的嵌入应用系统。

MySQL AB 是由两个瑞典人和一个芬兰人：David Axmark、Allan Larsson 和 Michael “Monty” Widenius 在瑞典创办的。

在 2008 年初，Sun Microsystems 收购了 MySQL AB 公司。在 2009 年，Oracle 收购了 Sun 公司，使 MySQL 并入 Oracle 的数据库产品线。

1.4、MySQL 的安装

打开下载的 mysql 安装文件 mysql-essential-5.0.22-win32.msi，

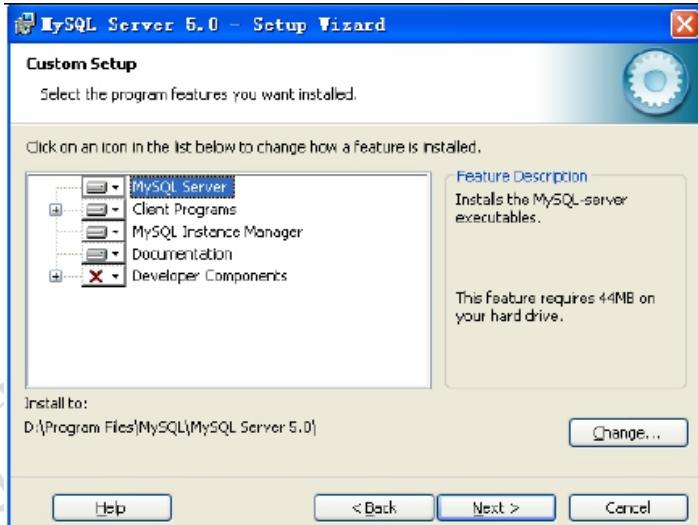
双击运行，出现如下界面



按“Next”继续



选择安装类型，有“Typical（默认）”、“Complete（完全）”、“Custom（用户自定义）”三个选项，我们选择“Custom”，有更多的选项，也方便熟悉安装过程

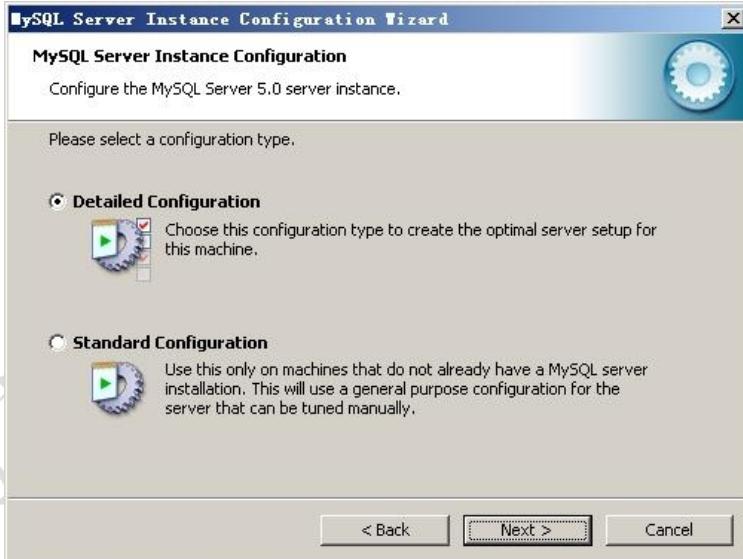


上一步选择了 Custom 安装，这里将设定 MySQL 的组件包和安装路径，设定好之后，单击 Next 继续安装。



现在软件安装完成了，出现上面的界面，将“Configure the MySQL Server now”前面的勾打上，点“Finish”结束软件的安装并启动 mysql 配置向导。

mysql 配置向导启动界面，按“Next”继续。



选择配置方式，“Detailed Configuration（手动精确配置）”、“Standard Configuration（标准配置）”，我们选择“Detailed Configuration”，方便熟悉配置过程。



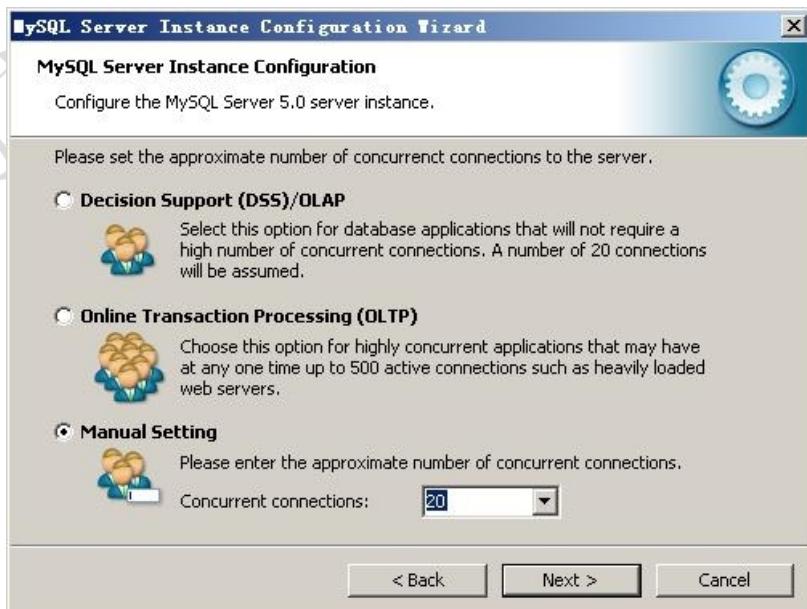
选择服务器类型，“Developer Machine（开发测试类，mysql 占用很少资源）”、“Server Machine(服务器类型，mysql 占用较多资源)”、“Dedicated MySQL Server Machine（专门的数据库服务器，mysql 占用所有可用资源）”，大家根据自己的类型选择了，一般选“Server Machine”，不会太少，也不会占满。



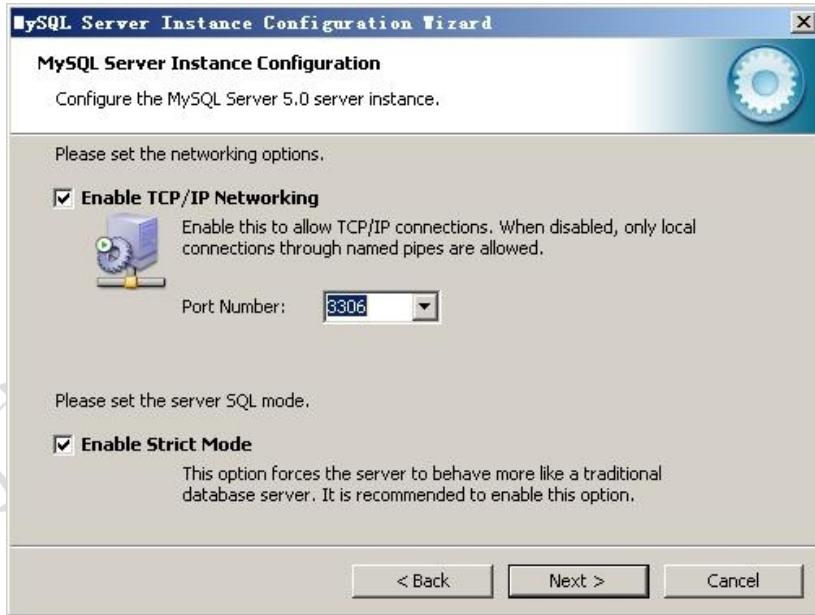
选择 mysql 数据库的大致用途，“Multifunctional Database（通用多功能型，能很好的支持 InnoDB 与 MyISAM 存储引擎）”、“Transactional Database Only（服务器类型，专注于事务处理，一般）”、“Non-Transactional Database Only（非事务处理型，较简单，主要做一些监控、记数用，对 MyISAM 数据类型的支持仅限于 non-transactional），随自己的用途而选择了，我这里选择“Multifunctional Database”，按“Next”继续。



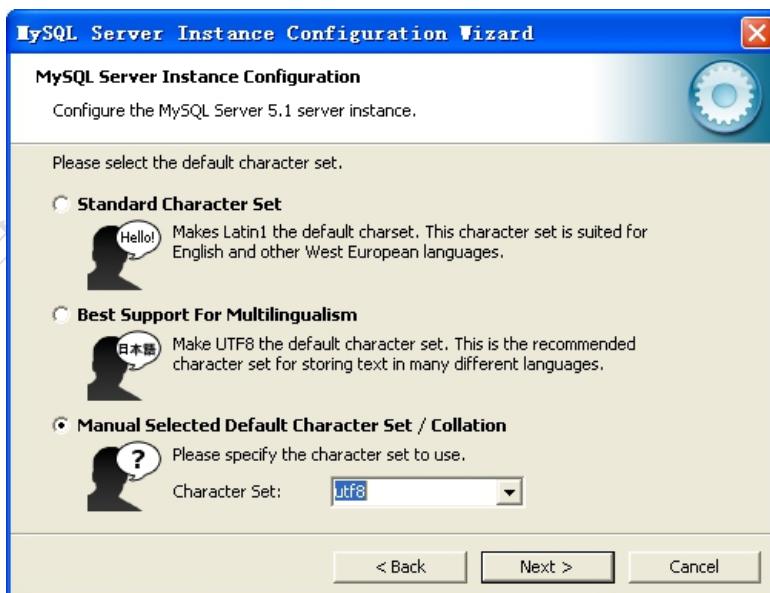
对 InnoDB Tablespace 进行配置，就是为 InnoDB 数据库文件选择一个存储空间，如果修改了，要记住位置，重装的时候要选择一样的地方，否则可能会造成数据库损坏，当然，对数据库做个备份就没问题了，这里不详述。我这里没有修改，使用用默认位置，直接按“Next”继续。



选择您的网站的一般 mysql 访问量，同时连接的数目，“Decision Support (DSS)/OLAP（20 个左右）”、“Online Transaction Processing(OLTP)（500 个左右）”、“Manual Setting（手动设置，自己输一个数）”，我这里选“Decision Support (DSS)/OLAP”，按“Next”继续



是否启用 TCP/IP 连接，设定端口，如果不启用，就只能在自己的机器上访问 mysql 数据库了，我这里启用，把前面的勾打上，Port Number: 3306，在这个页面上，您还可以选择“启用标准模式”（Enable Strict Mode），按“Next”继续。



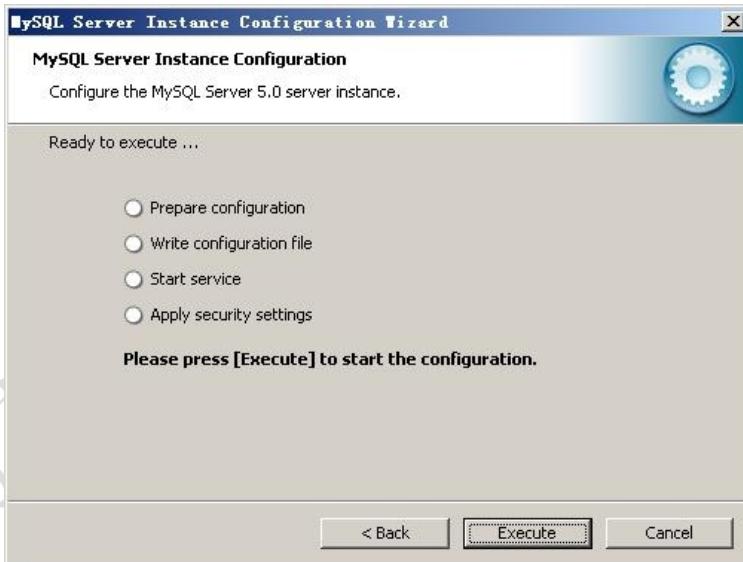
这个比较重要，就是对 mysql 默认数据库语言编码进行设置，第一个是西文编码，我们要设置的是 utf8 编码，按“Next”继续。



选择是否将 mysql 安装为 windows 服务，还可以指定 Service Name（服务标识名称），是否将 mysql 的 bin 目录加入到 Windows PATH（加入后，就可以直接使用 bin 下的文件，而不用指出目录名，比如连接，“mysql.exe -uusername -ppassword;”就可以了，不用指出 mysql.exe 的完整地址，很方便），我这里全部打上了勾，Service Name 不变。按“Next”继续。



设置完毕，按“Next”继续。



确认设置无误，如果有误，按“Back”返回检查。按“Execute”使设置生效。



设置完毕，按“Finish”结束 mysql 的安装与配置

可以通过服务管理器管理 MYSQL 的服务。

通过命令调用服务管理器:services.msc

停止 MYSQL 的服务。



启动 MySQL 的服务。



也可以在 DOS 中直接通过命令行的形式进行控制。

停止 MySQL 的服务。

```
C:\Documents and Settings\Administrator>net stop mysql
MySQL 服务正在停止。
MySQL 服务已成功停止。
```

启动 MySQL 的服务。

```
C:\Documents and Settings\Administrator>net start mysql
MySQL 服务正在启动。
MySQL 服务已经启动成功。
```

1.5、表

表(table)是一种**结构化的文件**, 可以用来存储特定类型的数据, 如: 学生信息, 课程信息, 都可以放到表中。另外表都有特定的名称, 而且不能重复。表中具有几个概念: **列、行、主键**。列叫做字段(Column), 行叫做表中的记录, 每一个字段都有:**字段名称/字段数据类型/字段约束/字段长度**

学生信息表

学号 (主键)	姓名	性别	年龄
00001	张三	男	20

00002	李四	女	20
-------	----	---	----

1.6、SQL 的分类

数据查询语言(DQL-Data Query Language)

代表关键字:select

数据操纵语言(DML-Data Manipulation Language)

代表关键字:insert,delete,update

数据定义语言(DDL-Data Definition Language)

代表关键字:create ,drop,alter,

事务控制语言(TCL-Transactional Control Language)

代表关键字:commit ,rollback;

数据控制语言(DCL-Data Control Language)

代表关键字:grant,revoke.

1.7、导入演示数据

使用 MySQL 命令行客户端来装载数据库。

1) 连接 MySql

```
C:\Documents and Settings\Administrator>mysql -uroot -proot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.51-community MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

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

mysql>
```

2) 创建 “bjpowernode”数据库

```
mysql> create database bjpowernode;
```

3) 选择数据库

```
mysql> use bjpowernode
```

4) 导入数据

```
mysql>source D:\bjpowernode.sql
```

5) 删除数据库(这里不要做!)

```
mysql> drop database bjpowernode;
```

1.8、表结构描述

表名称: dept

描述: 部门信息表

英文学段名称	中文描述	类型
DEPTNO	部门编号	INT(2)

DNAME	部门名称	VARCHAR(14)
LOC	位置	VARCHAR(13)

表名称: emp

描述: 员工信息表

英文字段名称	中文描述	类型
EMPNO	员工编号	INT (4)
ENAME	员工姓名	VARCHAR(10)
JOB	工作岗位	VARCHAR(9)
MGR	上级领导	INT (4)
HIREDATE	入职日期	DATE
SAL	薪水	DOUBLE(7,2)
COMM	津贴	DOUBLE (7,2)
DEPTNO	部门编号	INT(2)

注: DEPTNO 字段是外键, DEPTNO 的值来源于 dept 表的主键, 起到了约束的作用

表名称: salgrade

描述: 薪水等级信息表

英文字段名称	中文描述	类型
GRADE	等级	INT
LOSAL	最低薪水	INT
HISAL	最高薪水	INT

2、常用命令

2.1、查看 mysql 版本

- MySQL 程序选项具有以下两种通用形式:
 - 长选项, 由单词之前加两个减号组成
 - 短选项, 由单个字母之前加一个减号组成

```
C:\Users\Administrator>mysql --version
```

```
mysql Ver 14.14 Distrib 5.5.36, for Win32 (x86)
```

```
C:\Users\Administrator>mysql -V
```

```
mysql Ver 14.14 Distrib 5.5.36, for Win32 (x86)
```

2.2、创建数据库

- create database 数据库名称;
- ```
create database bjpowernode;
```

## 2. use 数据库名称

```
use bjpowernode;
```

在数据库中建立表，因此创建表的时候必须要先选择数据库。

## 2.3、查询当前使用的数据库

```
select database();
```

查询数据库版本也可以使用

```
select version();
```

## 2.4、终止一条语句

如果想要终止一条正在编写的语句，可键入\c。

## 2.5、退出 mysql

可使用\q、QUIT 或 EXIT:

如：

```
mysql> \q (ctrl+c)
```

# 3、查看“演示数据”的表结构

## 3.1、查看和指定现有的数据库

```
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| bjpowernode |
| exam |
| examdata |
| mysql |
| oe |
| onlineexam |
| test |
+--------------------+
8 rows in set (0.00 sec)
```

## 3.2、指定当前缺省数据库

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

### 3.3、查看当前使用的库

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

### 3.4、查看当前库中的表

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

### 3.5、查看其他库中的表

show tables from <database name>;  
如查看 exam 库中的表

```
mysql> show tables from exam;
+-----+
| Tables_in_exam |
+-----+
| admin
| examrecords
| item
| user |
+-----+
4 rows in set (0.00 sec)
```

### 3.6、查看表的结构

desc <table name>;  
如:

```
mysql> desc emp;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
EMPNO	int(4)	NO	PRI	NULL	
ENAME	varchar(10)	YES		NULL	
JOB	varchar(9)	YES		NULL	
MGR	int(4)	YES		NULL	
HIREDATE	date	YES		NULL	
SAL	double(7,2)	YES		NULL	
COMM	double(7,2)	YES		NULL	
DEPTNO	int(2)	YES	MUL	NULL	
+-----+-----+-----+-----+-----+-----+
8 rows in set (0.02 sec)
```

### 3.7、查看表的创建语句

show create table <table name>;

如：

```
mysql> show create table emp;
```

```
+-----+-----+-----+-----+-----+-----+
| emp | CREATE TABLE `emp` (
 `EMPNO` int(4) NOT NULL,
 `ENAME` varchar(10) DEFAULT NULL,
 `JOB` varchar(9) DEFAULT NULL,
 `MGR` int(4) DEFAULT NULL,
 `HIREDATE` date DEFAULT NULL,
 `SAL` double(7,2) DEFAULT NULL,
 `COMM` double(7,2) DEFAULT NULL,
 `DEPTNO` int(2) DEFAULT NULL,
 PRIMARY KEY (`EMPNO`),
 KEY `DEPTNO` (`DEPTNO`),
 CONSTRAINT `emp_ibfk_1` FOREIGN KEY (`DEPTNO`) REFERENCES `dept` (`DEPTNO`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ;
```

## 4、简单的查询

### 4.1、查询一个字段

- 查询员工姓名

```
select ename from emp;
```

```
mysql> select 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)
```

Select 语句后面跟的是字段名称, select 是关键字, select 和字段名称之间采用空格隔开, from 表示将要查询的表, 它和字段之间采用空格隔开

## 4.2、查询多个字段

- 查询员工的编号和姓名

```
select empno, ename from emp;
```

```
mysql> select empno, ename from emp;
+-----+-----+
| empno | ename |
+-----+-----+
7369	SMITH
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES
7902	FORD
7934	MILLER
+-----+-----+
14 rows in set (0.00 sec)
```

查询多个字段, select 中的字段采用逗号间隔即可, 最后一个字段, 也就是在 from 前面的字段不能使用逗号了。

## 4.3、查询全部字段

可以将所有的字段放到 select 语句的后面，这种方案不方便，但是比较清楚，我们可以采用如下便捷的方式查询全部字段

```
select * from emp;
```

```
mysql> select * from emp;
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7369	SMITH	CLERK	7902	0000-00-00	800.00	NULL	20
7499	ALLEN	SALESMAN	7698	0000-00-00	1600.00	300.00	30
7521	WARD	SALESMAN	7698	0000-00-00	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981-02-04	2975.00	NULL	20
7654	MARTIN	SALESMAN	7698	0000-00-00	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981-01-05	2850.00	NULL	30
7782	CLARK	MANAGER	7839	1981-09-06	2450.00	NULL	10
7788	SCOTT	ANALYST	7566	0000-00-00	3000.00	NULL	20
7839	KING	PRESIDENT	NULL	0000-00-00	5000.00	NULL	10
7844	TURNER	SALESMAN	7698	1981-08-09	1500.00	0.00	30
7876	ADAMS	CLERK	7788	0000-00-00	1100.00	NULL	20
7900	JAMES	CLERK	7698	1981-03-12	950.00	NULL	30
7902	FORD	ANALYST	7566	1981-03-12	3000.00	NULL	20
7934	MILLER	CLERK	7782	0000-00-00	1300.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

采用 select \* from emp，虽然简单，但是\*号不是很明确，建议查询全部字段将相关字段写到 select 语句的后面，在以后 java 连接数据库的时候，是需要在 java 程序中编写 SQL 语句的，这个时候编写的 SQL 语句不建议使用 select \* 这种形式，建议写明字段，这样可读性强。

## 4.4、计算员工的年薪

- 列出员工的编号，姓名和年薪

```
select empno, ename, sal*12 from emp;
```

```
mysql> select empno, ename, sal*12 from emp;
+-----+-----+-----+
| empno | ename | sal*12 |
+-----+-----+-----+
7369	SMITH	9600.00
7499	ALLEN	19200.00
7521	WARD	15000.00
7566	JONES	35700.00
7654	MARTIN	15000.00
7698	BLAKE	34200.00
7782	CLARK	29400.00
7788	SCOTT	36000.00
7839	KING	60000.00
7844	TURNER	18000.00
7876	ADAMS	13200.00
7900	JAMES	11400.00
7902	FORD	36000.00
7934	MILLER	15600.00
+-----+-----+-----+
14 rows in set (0.06 sec)
```

在 select 语句中可以使用运算符，以上存在一些问题，年薪的字段名称不太明确

## 4.5、将查询出来的字段显示为中文

```
select empno as '员工编号', ename as '员工姓名', sal*12 as '年薪' from emp;
```

注意：字符串必须添加单引号 | 双引号

| 员工编号 | 员工姓名   | 年薪       |
|------|--------|----------|
| 7369 | SMITH  | 9600.00  |
| 7499 | ALLEN  | 19200.00 |
| 7521 | WARD   | 15000.00 |
| 7566 | JONES  | 35700.00 |
| 7654 | MARTIN | 15000.00 |
| 7698 | BLAKE  | 34200.00 |
| 7782 | CLARK  | 29400.00 |
| 7788 | SCOTT  | 36000.00 |
| 7839 | KING   | 60000.00 |
| 7844 | TURNER | 18000.00 |
| 7876 | ADAMS  | 13200.00 |
| 7900 | JAMES  | 11400.00 |
| 7902 | FORD   | 36000.00 |
| 7934 | MILLER | 15600.00 |

14 rows in set (0.00 sec)

可以采用 as 关键字重命名表字段，其实 as 也可以省略，如：

```
select empno "员工编号", ename "员工姓名", sal*12 "年薪" from emp;
```

## 5、条件查询

条件查询需要用到 where 语句，where 必须放到 from 语句表的后面

支持如下运算符

| 运算符                  | 说明                            |
|----------------------|-------------------------------|
| =                    | 等于                            |
| <>或!=                | 不等于                           |
| <                    | 小于                            |
| <=                   | 小于等于                          |
| >                    | 大于                            |
| >=                   | 大于等于                          |
| between ... and .... | 两个值之间, 等同于 >= and <=          |
| is null              | 为 null (is not null 不为空)      |
| and                  | 并且                            |
| or                   | 或者                            |
| in                   | 包含, 相当于多个 or (not in 不在这个范围内) |
| not                  | not 可以取非, 主要用在 is 或 in 中      |
| like                 | like 称为模糊查询, 支持%或下划线匹配        |

%匹配任意个字符  
下划线，一个下划线只匹配一个字符

## 5.1、等号操作符

- 查询薪水为 5000 的员工

```
select empno, ename, sal from emp where sal=5000;
```

```
mysql> select empno, ename, sal from emp where sal=5000;
+-----+-----+-----+
| empno | ename | sal |
+-----+-----+-----+
| 7839 | KING | 5000.00 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

- 查询 job 为 MANAGER 的员工

```
select empno, ename from emp where job=manager;
```

```
mysql> select empno, ename from emp where job=manager;
ERROR 1054 (42S22): Unknown column 'manager' in 'where clause'
mysql>
```

以上查询出现错误，因为 job 为字符串，所以出现了以上错误

```
select empno, ename from emp where job="manager";
```

```
mysql> select empno, ename from emp where job="manager";
+-----+-----+
| empno | ename |
+-----+-----+
7566	JONES
7698	BLAKE
7782	CLARK
+-----+-----+
3 rows in set (0.00 sec)
```

```
select empno, ename from emp where job='manager';
```

```
mysql> select empno, ename from emp where job='manager';
+-----+-----+
| empno | ename |
+-----+-----+
7566	JONES
7698	BLAKE
7782	CLARK
+-----+-----+
3 rows in set (0.00 sec)
```

也可以使用单引号

```
select empno, ename from emp where job='MANAGER';
```

```
mysql> select empno, ename from emp where job='MANAGER';
+-----+-----+
| empno | ename |
+-----+-----+
7566	JONES
7698	BLAKE
7782	CLARK
+-----+-----+
3 rows in set (0.00 sec)
```

以上输出正确，Mysql 默认情况下大小写是不敏感的。

注意：

MySQL 在 windows 下是不区分大小写的，将 script 文件导入 MySQL 后表名也会自动转化为小写，结果再想要将数据库导出放到 linux 服务器中使用时就出错了。因为在 linux 下表名区分大小写而找不到表，查了很多都是说在 linux 下更改 MySQL 的设置使其也不区分大小写，但是有没有办法反过来让 windows 下大小写敏感呢。其实方法是一样的，相应的更改 windows 中 MySQL 的设置就行了。

具体操作：

在 MySQL 的配置文件 my.ini 中增加一行：

```
lower_case_table_names = 0
```

其中 0：区分大小写，1：不区分大小写

MySQL 在 Linux 下数据库名、表名、列名、别名大小写规则是这样的：

- 1、数据库名与表名是严格区分大小写的；
- 2、表的别名是严格区分大小写的；
- 3、列名与列的别名在所有的情况下均是忽略大小写的；
- 4、变量名也是严格区分大小写的； MySQL 在 Windows 下都不区分大小写

## 5.2、<>操作符

- 查询薪水不等于 5000 的员工

```
select empno, ename, sal from emp where sal <> 5000;
```

```
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
7844	TURNER	1500.00
7876	ADAMS	1100.00
7900	JAMES	950.00
7902	FORD	3000.00
7934	MILLER	1300.00
+-----+-----+-----+
13 rows in set (0.00 sec)
```

一下写法等同于以上写法，建议使用第一种写法

```
select empno, ename, sal from emp where sal != 5000;
```

数值也可以采用单引号引起，如一下语句是正确的(不建议这么写):

```
select empno, ename, sal from emp where sal <> '5000';
```

- 查询工作岗位不等于 MANAGER 的员工

```
select empno, ename from emp where job <> 'MANAGER';
```

### 5.3、between ... and ...操作符

- 查询薪水为 1600 到 3000 的员工(第一种方式，采用 $\geq$ 和 $\leq$ )

```
select empno, ename, sal from emp where sal >= 1600 and sal <= 3000;
```

```
mysql> select empno, ename, sal from emp where sal >= 1600 and sal <= 3000;
+-----+-----+-----+
| empno | ename | sal |
+-----+-----+-----+
7499	ALLEN	1600.00
7566	JONES	2975.00
7698	BLAKE	2850.00
7782	CLARK	2450.00
7788	SCOTT	3000.00
7902	FORD	3000.00
+-----+-----+-----+
6 rows in set (0.00 sec)
```

- 查询薪水为 1600 到 3000 的员工(第一种方式，采用 between ... and ...)

```
select empno, ename, sal from emp where sal between 1600 and 3000;
```

```
mysql> select empno, ename, sal from emp where sal between 1600 and 3000;
+-----+-----+-----+
| empno | ename | sal |
+-----+-----+-----+
7499	ALLEN	1600.00
7566	JONES	2975.00
7698	BLAKE	2850.00
7782	CLARK	2450.00
7788	SCOTT	3000.00
7902	FORD	3000.00
+-----+-----+-----+
6 rows in set (0.05 sec)
```

关于 `between ... and ...`, 它是包含最大值和最小值的

## 5.4、is null

- Null 为空, 但不是空串, 为 null 可以设置这个字段不填值, 如果查询为 null 的字段, 采用 `is null`
- 查询津贴为空的员工

```
select * from emp where comm=null;
```

```
mysql> select * from emp where comm=null;
Empty set (0.00 sec)
```

以上也无法查询出符合条件的数据, 因为 null 类型比较特殊, 必须使用 `is` 来比较

```
select * from emp where comm is null;
```

```
mysql> select * from emp where comm is null;
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
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
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
+-----+-----+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

以上查询正确

## 5.5、and

`and` 表示并且的含义, 表示所有的条件必须满足

- 工作岗位为 MANAGER, 薪水大于 2500 的员工

```
select * from emp where job='MANAGER' and sal > 2500;
```

```
mysql> select * from emp where job='MANAGER' and sal > 2500;
+-----+-----+-----+-----+-----+-----+-----+
| 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 |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

## 5.6、or

or, 只要满足条件即可, 相当于包含

- 查询出 job 为 manager 或者 job 为 salesman 的员工

```
select * from emp where job='MANAGER' or job='SALESMAN';
```

```
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)
```

## 5.7、表达式的优先级

- 查询薪水大于 1800, 并且部门代码为 20 或 30 的员工 (错误的写法)

```
select * from emp where sal > 1800 and deptno = 20 or deptno = 30;
```

```
mysql> select * from emp where sal > 1800 and deptno = 20 or deptno = 30;
+-----+-----+-----+-----+-----+-----+-----+
| 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
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
+-----+-----+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)
```

以上输出不是预期结果, 薪水小于 1800 的数据也被查询上来了, 原因是表达式的优先级导致的, 首先过滤 `sal > 1800 and deptno = 20`, 然后再将 `deptno = 30` 员工合并过来, 所以是不对的

- 查询薪水大于 1800，并且部门代码为 20 或 30 的（正确的写法）

```
select * from emp where sal > 1800 and (deptno = 20 or deptno = 30);
```

```
mysql> select * from emp where sal > 1800 and (deptno = 20 or deptno = 30);
+-----+-----+-----+-----+-----+-----+-----+
| 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
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

关于运算符的问题：不用记，没有把握尽量采用括号

## 5.8、in

in 表示包含的意思，完全可以采用 or 来表示，采用 in 会更简洁一些

- 查询出 job 为 manager 或者 job 为 salesman 的员工

```
select * from emp where job in ('manager','salesman');
```

```
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)
```

- 查询出薪水包含 1600 和薪水包含 3000 的员工

```
select * from emp where sal in(1600, 3000);
```

```
mysql> select * from emp where sal in(1600, 3000);
+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

## 5.9、not

- 查询出薪水不包含 1600 和薪水不包含 3000 的员工（第一种写法）

```
select * from emp where sal <> 1600 and sal <> 3000;
```

```
mysql> select * from emp where sal <> 1600 and sal <> 3000;
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
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
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
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

- 查询出薪水不包含 1600 和薪水不包含 3000 的员工（第二种写法）

```
select * from emp where not (sal = 1600 or sal = 3000);
```

```
mysql> select * from emp where not (sal = 1600 or sal = 3000);
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
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
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
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

- 查询出薪水不包含 1600 和薪水不包含 3000 的员工（第三种写法）

```
select * from emp where sal not in (1600, 3000);
```

```
mysql> select * from emp where sal not in (1600, 3000);
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
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
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
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

- 查询出津贴不为 null 的所有员工

```
select * from emp where comm is not null;
```

```
mysql> select * from emp where comm is not null;
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

## 5.10、like

- Like 可以实现模糊查询，like 支持%和下划线匹配
- 查询姓名以 M 开头所有的员工

```
select * from emp where ename like 'M%';
```

```
mysql> select * from emp where ename like 'M%';
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00 | 1400.00 | 30 |
| 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300.00 | NULL | 10 |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

- 查询姓名以 N 结尾的所有的员工

```
select * from emp where ename like '%N';
```

```
mysql> select * from emp where ename like '%N';
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600.00 | 300.00 | 30 |
| 7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00 | 1400.00 | 30 |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

- 查询姓名中包含 O 的所有的员工

```
select * from emp where ename like '%O%';
```

```
mysql> select * from emp where ename like '%O%';
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

- 查询姓名中第二个字符为 A 的所有员工

```
select * from emp where ename like '_A%';
```

```
mysql> select * from emp where ename like '_A%';
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Like 中%和下划线的差别?

%匹配任意字符出现的个数

下划线只匹配一个字符

Like 中的表达式必须放到单引号中|双引号中, 以下写法是错误的:

```
select * from emp where ename like _A%
```

## 6、排序数据

### 6.1、单一字段排序

排序采用 **order by** 子句, **order by** 后面跟上排序字段, 排序字段可以放多个, 多个采用逗号间隔, **order by** 默认采用升序, 如果存在 **where** 子句那么 **order by** 必须放到 **where** 语句的后面

- 按照薪水由小到大排序(系统默认由小到大)

```
select * from emp order by sal;
```

```
mysql> select * from emp order by sal;
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	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
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	NULL	10
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL	20
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

- 取得 job 为 MANAGER 的员工, 按照薪水由小到大排序(系统默认由小到大)

```
select * from emp where job='MANAGER' order by sal;
```

```
mysql> select * from emp where job='MANAGER' order by sal;
+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	NULL	10
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL	20
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

如果包含 where 语句 order by 必须放到 where 后面，如果没有 where 语句 order by 放到表的后面

以下写法是错误的：

```
select * from emp order by sal where job='MANAGER';
```

- 按照多个字段排序，如：首先按照 job 排序，再按照 sal 排序

```
select * from emp order by job,sal;
```

```
mysql> select * from emp order by job,sal;
+-----+-----+-----+-----+-----+-----+-----+
| 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
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	NULL	20
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	NULL	10
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL	20
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	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
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

## 6.2、手动指定排序顺序

- 手动指定按照薪水由小到大排序

```
select * from emp order by sal asc;
```

```
mysql> select * from emp order by sal asc;
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	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
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	NULL	10
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL	20
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

- 手动指定按照薪水由大到小排序

```
select * from emp order by sal desc;
```

```
mysql> select * from emp order by sal desc;
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
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
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

## 6.3、多个字段排序

- 按照 job 和薪水倒序

```
select * from emp order by job desc, sal desc;
```

```
mysql> select * from emp order by job desc, sal desc;
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	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
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
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
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

如果采用多个字段排序，如果根据第一个字段排序重复了，会根据第二个字段排序

## 6.4、使用字段的位置来排序

- 按照薪水升序

```
select * from emp order by 6;
```

```
mysql> select * from emp order by 6;
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	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
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	NULL	10
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL	20
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

不建议使用此种方式，采用数字含义不明确，程序不健壮

## 7、分组函数/聚合函数/多行处理函数

|       |       |
|-------|-------|
| count | 取得记录数 |
| sum   | 求和    |
| avg   | 取平均   |
| max   | 取最大的数 |

min

取最小的数

注意：分组函数自动忽略空值，不需要手动的加 where 条件排除空值。

select count(\*) from emp where xxx;                   符合条件的所有记录总数。

select count(comm) from emp;                   comm 这个字段中不为空的元素总数。

注意：分组函数不能直接使用在 where 关键字后面。

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

ERROR 1111 (HY000): Invalid use of group function

## 7.1、count

- 取得所有的员工数

```
select count(*) from emp;
```

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

Count(\*)表示取得所有记录，忽

略 null，为 null 的值也会取得

- 取得津贴不为 null 员工数

```
select count(comm) from emp;
```

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

采用 count(字段名称)，不会取得为 null 的记录

- 取得工作岗位的个数

```
select count(distinct job) from emp;
```

```
mysql> select count(distinct(job)) from emp;
+-----+
| count(distinct(job)) |
+-----+
| 5 |
+-----+
1 row in set (0.06 sec)
```

## 7.2、sum

- Sum 可以取得某一个列的和, null 会被忽略
- 取得薪水的合计

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

```
mysql> select sum(sal) from emp;
+-----+
| sum(sal) |
+-----+
| 29025.00 |
+-----+
1 row in set (0.00 sec)
```

- 取得津贴的合计

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

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

null 会被忽略

- 取得薪水的合计 (sal+comm)

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

```
mysql> select sum(sal+comm) from emp;
+-----+
| sum(sal+comm) |
+-----+
| 7800.00 |
+-----+
1 row in set (0.00 sec)
```

从以上结果来看，不正确，原因在于 comm 字段有 null 值，所以无法计算，sum 会忽略掉，正确的做法是将 comm 字段转换成 0

```
select sum(sal+IFNULL(comm, 0)) from emp;
```

```
mysql> select sum(sal+IFNULL(comm, 0)) from emp;
+-----+
| sum(sal+IFNULL(comm, 0)) |
+-----+
| 31225.00 |
+-----+
1 row in set (0.05 sec)
```

## 7.3、avg

取得某一列的平均值

- 取得平均薪水

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

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

## 7.4、max

取得某个一列的最大值

- 取得最高薪水

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

```
mysql> select max(sal) from emp;
+-----+
| max(sal) |
+-----+
| 5000.00 |
+-----+
1 row in set (0.05 sec)
```

- 取得最晚入职得员工

```
select max(str_to_date(hiredate, '%Y-%m-%d')) from emp;
```

```
mysql> select max(str_to_date(hiredate, '%Y-%m-%d')) from emp;
+-----+
| max(str_to_date(hiredate, '%Y-%m-%d')) |
+-----+
| 1987-05-23 |
+-----+
1 row in set (0.00 sec)
```

## 7.5、min

取得某个一列的最小值

- 取得最低薪水

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

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

- 取得最早入职得员工 (可以不使用 str\_to\_date 转换)

```
select min(str_to_date(hiredate, '%Y-%m-%d')) from emp;
```

```
mysql> select min(str_to_date(hiredate, '%Y-%m-%d')) from emp;
+-----+
| min(str_to_date(hiredate, '%Y-%m-%d')) |
+-----+
| 1980-12-17 |
+-----+
1 row in set (0.00 sec)
```

## 7.6、组合聚合函数

可以将这些聚合函数都放到 select 中一起使用

```
select count(*),sum(sal),avg(sal),max(sal),min(sal) from emp;
```

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

## 8、分组查询

分组查询主要涉及到两个子句，分别是：group by 和 having

### 8.1、group by

- 取得每个工作岗位的工资合计，要求显示岗位名称和工资合计

```
select job, sum(sal) from emp group by job;
```

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

如果使用了 order by, order by 必须放到 group by 后面

```
mysql> select job, sum(sal) from emp order by job group by job;
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 'group
by job' at line 1
```

- 按照工作岗位和部门编码分组，取得的工资合计

- 原始数据

```
mysql> select * from emp;
+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+-----+
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL	20
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	NULL	10
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
7876	ADAMS	CLERK	7788	1987-05-23	1100.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	30
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

- 分组语句

```
select job,deptno,sum(sal) from emp group by job,deptno;
```

```
mysql> select job,deptno,sum(sal) from emp group by job,deptno;
+-----+-----+-----+
| job | deptno | sum(sal) |
+-----+-----+-----+
ANALYST	20	6000.00
CLERK	10	1300.00
CLERK	20	1900.00
CLERK	30	950.00
MANAGER	10	2450.00
MANAGER	20	2975.00
MANAGER	30	2850.00
PRESIDENT	10	5000.00
SALESMAN	30	5600.00
+-----+-----+-----+
9 rows in set (0.00 sec)
```

```
mysql> select empno,deptno,avg(sal) from emp group by deptno;
+-----+-----+-----+
| empno | deptno | avg(sal) |
+-----+-----+-----+
7782	10	2916.666667
7369	20	2175.000000
7499	30	1566.666667
+-----+-----+-----+
```

以上 SQL 语句在 Oracle 数据库中无法执行，执行报错。

以上 SQL 语句在 Mysql 数据库中可以执行，但是执行结果矛盾。

在 SQL 语句中若有 group by 语句，那么在 select 语句后面只能跟**分组函数+参与分组的字段**。

## 8.2、having

如果想对分组数据再进行过滤需要使用 having 子句

取得每个岗位的平均工资大于 2000

```
select job, avg(sal) from emp group by job having avg(sal) >2000;
```

```
mysql> select job, avg(sal) from emp group by job having avg(sal) >2000;
+-----+-----+
| job | avg(sal) |
+-----+-----+
ANALYST	3000.000000
MANAGER	2758.333333
PRESIDENT	5000.000000
+-----+-----+
3 rows in set (0.00 sec)
```

分组函数的执行顺序：

根据条件查询数据

分组

采用 having 过滤，取得正确的数据

## 8.3、select 语句总结

一个完整的 select 语句格式如下

```
select 字段
from 表名
where
group by
having(就是为了过滤分组后的数据而存在的—不可以单独的出现)
order by
```

以上语句的执行顺序

1. 首先执行 where 语句过滤原始数据
2. 执行 group by 进行分组
3. 执行 having 对分组数据进行操作
4. 执行 select 选出数据
5. 执行 order by 排序

原则：能在 where 中过滤的数据，尽量在 where 中过滤，效率较高。having 的过滤是专门对分组之后的数据进行过滤的。

## 9、连接查询

### 9.1、SQL92 语法

连接查询：也可以叫跨表查询，需要关联多个表进行查询

- 显示每个员工信息，并显示所属的部门名称

```
select ename, dname from emp, dept;
```

```
SQL> select ename, dname from emp, dept;
```

| ENAME  | DNAME      |
|--------|------------|
| SMITH  | ACCOUNTING |
| ALLEN  | ACCOUNTING |
| WARD   | ACCOUNTING |
| JONES  | ACCOUNTING |
| MARTIN | ACCOUNTING |
| BLAKE  | ACCOUNTING |
| CLARK  | ACCOUNTING |
| SCOTT  | ACCOUNTING |
| KING   | ACCOUNTING |

|        |            |
|--------|------------|
| TURNER | ACCOUNTING |
| ADAMS  | ACCOUNTING |
| JAMES  | ACCOUNTING |
| FORD   | ACCOUNTING |
| MILLER | ACCOUNTING |
| SMITH  | RESEARCH   |
| ALLEN  | RESEARCH   |
| WARD   | RESEARCH   |
| JONES  | RESEARCH   |
| MARTIN | RESEARCH   |
| BLAKE  | RESEARCH   |
| CLARK  | RESEARCH   |
| SCOTT  | RESEARCH   |
| KING   | RESEARCH   |
| TURNER | RESEARCH   |
| ADAMS  | RESEARCH   |
| JAMES  | RESEARCH   |
| FORD   | RESEARCH   |
| MILLER | RESEARCH   |
| SMITH  | SALES      |
| ALLEN  | SALES      |
| WARD   | SALES      |
| JONES  | SALES      |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| CLARK  | SALES      |
| SCOTT  | SALES      |
| KING   | SALES      |
| TURNER | SALES      |
| ADAMS  | SALES      |
| JAMES  | SALES      |
| FORD   | SALES      |
| MILLER | SALES      |
| SMITH  | OPERATIONS |
| ALLEN  | OPERATIONS |
| WARD   | OPERATIONS |
| JONES  | OPERATIONS |
| MARTIN | OPERATIONS |
| BLAKE  | OPERATIONS |
| CLARK  | OPERATIONS |
| SCOTT  | OPERATIONS |
| KING   | OPERATIONS |
| TURNER | OPERATIONS |
| ADAMS  | OPERATIONS |

|        |            |
|--------|------------|
| JAMES  | OPERATIONS |
| FORD   | OPERATIONS |
| MILLER | OPERATIONS |

已选择 56 行。

以上输出，不正确，输出了 56 条数据，其实就是两个表记录的成绩，这种情况我们称为：

“笛卡儿乘积”，出现错误的原因是：没有指定连接条件

指定连接条件

```
select emp.ename, dept.dname from emp, dept where emp.deptno=dept.deptno;
```

也可以使用别名

```
select e.ename, d.dname from emp e, dept d where e.deptno=d.deptno;
```

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

以上结果输出正确，因为加入了正确的连接条件

以上查询也称为 “**内连接**”，只查询相等的数据（连接条件相等的数据）

- 取得员工和所属的领导的姓名

```
select e.ename, m.ename from emp e, emp m where e.mgr=m.empno;
```

```
SQL> select * from emp; (普通员工)
```

| EMPNO | ENAME  | JOB       | MGR  | HIREDATE    | SAL  | COMM | DEPTNO |
|-------|--------|-----------|------|-------------|------|------|--------|
| 7369  | SMITH  | CLERK     | 7902 | 17-12 月 -80 | 800  | 20   |        |
| 7499  | ALLEN  | SALESMAN  | 7698 | 20-2 月 -81  | 1600 | 300  | 30     |
| 7521  | WARD   | SALESMAN  | 7698 | 22-2 月 -81  | 1250 | 500  | 30     |
| 7566  | JONES  | MANAGER   | 7839 | 02-4 月 -81  | 2975 |      | 20     |
| 7654  | MARTIN | SALESMAN  | 7698 | 28-9 月 -81  | 1250 | 1400 | 30     |
| 7698  | BLAKE  | MANAGER   | 7839 | 01-5 月 -81  | 2850 |      | 30     |
| 7782  | CLARK  | MANAGER   | 7839 | 09-6 月 -81  | 2450 |      | 10     |
| 7788  | SCOTT  | ANALYST   | 7566 | 19-4 月 -87  | 3000 |      | 20     |
| 7839  | KING   | PRESIDENT |      | 17-11 月 -81 | 5000 |      | 10     |

|             |          |                  |      |   |    |
|-------------|----------|------------------|------|---|----|
| 7844 TURNER | SALESMAN | 7698 08-9 月 -81  | 1500 | 0 | 30 |
| 7876 ADAMS  | CLERK    | 7788 23-5 月 -87  | 1100 |   | 20 |
| 7900 JAMES  | CLERK    | 7698 03-12 月 -81 | 950  |   | 30 |
| 7902 FORD   | ANALYST  | 7566 03-12 月 -81 | 3000 |   | 20 |
| 7934 MILLER | CLERK    | 7782 23-1 月 -82  | 1300 |   | 10 |

已选择 14 行。

SQL> select \* from emp; (管理者)

| EMPNO       | ENAME     | JOB | MGR              | HIREDATE    | SAL  | COMM | DEPTNO |
|-------------|-----------|-----|------------------|-------------|------|------|--------|
| 7369 SMITH  | CLERK     |     | 7902 17-12 月 -80 |             | 800  |      | 20     |
| 7499 ALLEN  | SALESMAN  |     | 7698 20-2 月 -81  |             | 1600 | 300  | 30     |
| 7521 WARD   | SALESMAN  |     | 7698 22-2 月 -81  |             | 1250 | 500  | 30     |
| 7566 JONES  | MANAGER   |     | 7839 02-4 月 -81  |             | 2975 |      | 20     |
| 7654 MARTIN | SALESMAN  |     | 7698 28-9 月 -81  |             | 1250 | 1400 | 30     |
| 7698 BLAKE  | MANAGER   |     | 7839 01-5 月 -81  |             | 2850 |      | 30     |
| 7782 CLARK  | MANAGER   |     | 7839 09-6 月 -81  |             | 2450 |      | 10     |
| 7788 SCOTT  | ANALYST   |     | 7566 19-4 月 -87  |             | 3000 |      | 20     |
| 7839 KING   | PRESIDENT |     |                  | 17-11 月 -81 | 5000 |      | 10     |
| 7844 TURNER | SALESMAN  |     | 7698 08-9 月 -81  |             | 1500 | 0    | 30     |
| 7876 ADAMS  | CLERK     |     | 7788 23-5 月 -87  |             | 1100 |      | 20     |
| 7900 JAMES  | CLERK     |     | 7698 03-12 月 -81 |             | 950  |      | 30     |
| 7902 FORD   | ANALYST   |     | 7566 03-12 月 -81 |             | 3000 |      | 20     |
| 7934 MILLER | CLERK     |     | 7782 23-1 月 -82  |             | 1300 |      | 10     |

已选择 14 行。

SQL> select e.ename, m.ename from emp e, emp m where e.mgr=m.empno;

| ENAME  | ENAME |
|--------|-------|
| 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 |

已选择 13 行。

以上称为“自连接”，只有一张表连接，具体的查询方法，把一张表看作两张表即可，如以上示例：第一个表 emp e 代表了员工表，emp m 代表了领导表，相当于员工表和部门表一样

## 9.2、SQL99 语法

- （内连接）显示薪水大于 2000 的员工信息，并显示所属的部门名称

采用 SQL92 语法：

```
select e.ename, e.sal, d.dname from emp e, dept d where e.deptno=d.deptno and e.sal > 2000;
```

采用 SQL99 语法：

```
select e.ename, e.sal, d.dname from emp e join dept d on e.deptno=d.deptno where e.sal>2000;
```

或

```
select e.ename, e.sal, d.dname from emp e inner join dept d on e.deptno=d.deptno where e.sal>2000;
```

在实际中一般不加 **inner** 关键字

Sql92 语法和 sql99 语法的区别：99 语法可以做到表的连接和查询条件分离，特别是多个表进行连接的时候，会比 sql92 更清晰

- （外连接）显示员工信息，并显示所属的部门名称，如果某一个部门没有员工，那么该部门也必须显示出来

右连接：

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

左连接：

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

以上两个查询效果相同

```
mysql> select e.ename, e.sal, d.dname from emp e right join dept d on e.deptno=d.deptno;
+-----+-----+-----+
| ename | sal | dname |
+-----+-----+-----+
CLARK	2450.00	ACCOUNTING
KING	5000.00	ACCOUNTING
MILLER	1300.00	ACCOUNTING
SMITH	800.00	RESEARCH
JONES	2975.00	RESEARCH
SCOTT	3000.00	RESEARCH
ADAMS	1100.00	RESEARCH
FORD	3000.00	RESEARCH
ALLEN	1600.00	SALES
WARD	1250.00	SALES
MARTIN	1250.00	SALES
BLAKE	2850.00	SALES
TURNER	1500.00	SALES
JAMES	950.00	SALES
NULL	NULL	OPERATIONS
+-----+-----+-----+
15 rows in set (0.00 sec)
```

连接分类:

内连接

- \* 表1 inner join 表2 on 关联条件
- \* 做连接查询的时候一定要写上关联条件
- \* inner 可以省略

外连接

\*左外连接

- \* 表1 left outer join 表2 on 关联条件
  - \* 做连接查询的时候一定要写上关联条件
  - \* outer 可以省略
- \*右外连接
- \* 表1 right outer join 表2 on 关联条件
  - \* 做连接查询的时候一定要写上关联条件
  - \* outer 可以省略

\*左外连接（左连接）和右外连接（右连接）的区别:

\*左连接以左面的表为准和右边的表比较，和左表相等的不相等都会显示出来，右表符合条件的显示,不符合条件的不显示

\*右连接恰恰相反，以上左连接和右连接也可以加入 outer 关键字，但一般不建议这种写法，如：

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

左连接能完成的功能右连接一定可以完成

```
mysql> select e.ename, e.sal, d.dname from emp e right outer join dept d on e.deptno=d.deptno;
+-----+-----+-----+
| ename | sal | dname |
+-----+-----+-----+
CLARK	2450.00	ACCOUNTING
KING	5000.00	ACCOUNTING
MILLER	1300.00	ACCOUNTING
SMITH	800.00	RESEARCH
JONES	2975.00	RESEARCH
SCOTT	3000.00	RESEARCH
ADAMS	1100.00	RESEARCH
FORD	3000.00	RESEARCH
ALLEN	1600.00	SALES
WARD	1250.00	SALES
MARTIN	1250.00	SALES
BLAKE	2850.00	SALES
TURNER	1500.00	SALES
JAMES	950.00	SALES
NULL	NULL	OPERATIONS
+-----+-----+-----+
15 rows in set <0.00 sec>
```

## 10、子查询

子查询就是嵌套的 select 语句，可以理解为子查询是一张表

## 10.1、在 **where** 语句中使用子查询，也就是在 **where** 语句中加入 **select** 语句

- 查询员工信息，查询哪些人是管理者，要求显示出其员工编号和员工姓名  
实现思路：

- 1、首先取得管理者的编号，去除重复的

```
select distinct mgr from emp where mgr is not null;
```

**distinct 去除重复行**

- 2、查询员工编号包含管理者编号的

```
select empno, ename from emp where empno in(select mgr from emp where mgr is not null);
```

```
mysql> select empno, ename from emp where empno in(select mgr from emp where mgr is not null);
+-----+-----+
| empno | ename |
+-----+-----+
7566	JONES
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7902	FORD
+-----+
6 rows in set <0.03 sec>
```

- 查询哪些人的薪水高于员工的平均薪水，需要显示员工编号，员工姓名，薪水  
实现思路

- 1、取得平均薪水

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

- 2、取得大于平均薪水的员工

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

```
mysql> select empno, ename, sal from emp where sal > (select avg(sal) from emp);
+-----+-----+-----+
| empno | ename | sal |
+-----+-----+-----+
7566	JONES	2975.00
7698	BLAKE	2850.00
7782	CLARK	2450.00
7788	SCOTT	3000.00
7839	KING	5000.00
7902	FORD	3000.00
+-----+
6 rows in set <0.00 sec>
```

## 10.2、在 from 语句中使用子查询，可以将该子查询看做一张表

- 查询员工信息，查询哪些人是管理者，要求显示出其员工编号和员工姓名  
首先取得管理者的编号，去除重复的

```
select distinct mgr from emp where mgr is not null;
```

将以上查询作为一张表，放到 from 语句的后面

使用 92 语法：

```
select e.empno, e.ename from emp e, (select distinct mgr from emp where mgr is not null) m
where e.empno=m.mgr;
```

使用 99 语法：

```
select e.empno, e.ename from emp e join (select distinct mgr from emp where mgr is not null) m
on e.empno=m.mgr;
```

```
mysql> select e.empno, e.ename from emp e join (select distinct mgr from emp where mgr is not null) m on e.empno=m.mgr;
+-----+-----+
| empno | ename |
+-----+-----+
7902	FORD
7698	BLAKE
7839	KING
7566	JONES
7788	SCOTT
7782	CLARK
+-----+-----+
6 rows in set (0.00 sec)
```

- 查询各个部门的平均薪水所属等级，需要显示部门编号，平均薪水，等级编号  
实现思路

1、首先取得各个部门的平均薪水

```
select deptno, avg(sal) avg_sal from emp group by deptno;
```

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

2、将部门的平均薪水作为一张表与薪水等级表建立连接，取得等级

```
select deptno,avg(sal) avg_sal from emp group by deptno;
```

```
select * from salgrade;
```

```
select a.deptno,a.avg_sal,g.grade from (select deptno,avg(sal) avg_sal from emp group by deptno) a join salgrade g on a.avg_sal between g.losal and hisal;
```

```

mysql> select deptno,avg(sal) avg_sal from emp group by deptno;
+-----+-----+
| deptno | avg_sal |
+-----+-----+
10	2916.666667
20	2175.000000
30	1566.666667
+-----+-----+
3 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)

mysql> select a.deptno,a.avg_sal,g.grade from (select deptno,avg(sal) avg_sal from emp group by deptno) a join salgrade g on a.avg_sal between g.losal and hisal;
+-----+-----+-----+
| deptno | avg_sal | grade |
+-----+-----+-----+
30	1566.666667	3
10	2916.666667	4
20	2175.000000	4
+-----+-----+-----+
3 rows in set (0.00 sec)

```

平均薪水表

薪水等级表

### 10.3、在 select 语句中使用子查询

- 查询员工信息，并显示出员工所属的部门名称

第一种做法，将员工表和部门表连接

```
select e.ename, d.dname from emp e, dept d where e.deptno=d.deptno;
```

第二种做法，在 select 语句中再次嵌套 select 语句完成部分名称的查询

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

```

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

```

## 11、union

### 11.1、union 可以合并集合（相加）

1、查询 job 包含 MANAGER 和包含 SALESMAN 的员工

```
select * from emp where job in('MANAGER', 'SALESMAN');
```

```
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)
```

2、采用 union 来合并

```
select * from emp where job='MANAGER'
```

```
union
```

```
select * from emp where job='SALESMAN'
```

```
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 (0.00 sec)
```

合并结果集的时候，需要查询字段对应个数相同。在 Oracle 中更严格，不但要求个数相同，而且还要求类型对应相同。

## 12、limit 的使用

mySql 提供了 limit，主要用于提取前几条或者中间某几行数据

```
select * from table limit m,n
```

其中 m 是指记录开始的 index，从 0 开始，表示第一条记录

n 是指从第 m+1 条开始，取 n 条。

```
select * from tablename limit 2,4
```

即取出第 3 条至第 6 条，4 条记录

## 12.1、取得前 5 条数据

```
select * from emp limit 5;
```

```
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 emp limit 5;
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

## 12.2、从第二条开始取两条数据

```
select * from emp limit 1,2;
```

```
mysql> select * from emp limit 1,2;
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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 |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

## 12.3、取得薪水最高的前 5 名

```
select * from emp e order by e.sal desc limit 5;
```

```
mysql> select * from emp e order by e.sal desc limit 5;
+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM |
| | | | | | | |
+-----+-----+-----+-----+-----+-----+-----+
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL
7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL
7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL
+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

## 13、表

### 13.1、创建表

- 语句格式

```
create table tableName(
 columnName dataType(length),

 columnName dataType(length)
);
set character_set_results='gbk';

show variables like '%char%';
```

创建表的时候，表中有字段，每一个字段有：

- \* 字段名
- \* 字段数据类型
- \* 字段长度限制
- \* 字段约束

- MySql 常用数据类型

| 类型                  | 描述                       |
|---------------------|--------------------------|
| Char(长度)            | 定长字符串，存储空间大小固定，适合作为主键或外键 |
| Varchar(长度)         | 变长字符串，存储空间等于实际数据空间       |
| double(有效数字位数, 小数位) | 数值型                      |
| Float(有效数字位数, 小数位)  | 数值型                      |
| Int( 长度)            | 整型                       |
| bigint(长度)          | 长整型                      |
| Date                | 日期型 年月日                  |
| DateTime            | 日期型 年月日 时分秒 毫秒           |

|          |                                |
|----------|--------------------------------|
| time     | 日期型 时分秒                        |
| BLOB     | Binary Large OBject (二进制大对象)   |
| CLOB     | Character Large OBject (字符大对象) |
| 其它 ..... |                                |

- 建立学生信息表，字段包括：学号、姓名、性别、出生日期、email、班级标识

```
create table t_student(
 student_id int(10),
 student_name varchar(20),
 sex char(2),
 birthday date,
 email varchar(30),
 classes_id int(3)
)
```

```
mysql> desc t_student;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
student_id	int(10)	YES		NULL	
student_name	varchar(20)	YES		NULL	
sex	char(2)	YES		NULL	
birthday	date	YES		NULL	
email	varchar(30)	YES		NULL	
classes_id	int(3)	YES		NULL	
+-----+-----+-----+-----+-----+
6 rows in set (0.03 sec)
```

- 向 t\_student 表中加入数据, (必须使用客户端软件, 我们的 cmd 默认是 GBK 编码, 数据中设置的编码是 UTF-8)

```
insert into t_student(student_id, student_name, sex, birthday, email, classes_id) values(1001,
'zhangsan', 'm', '1988-01-01', 'qqq@163.com', 10)
```

```
mysql> select * from t_student;
+-----+-----+-----+-----+-----+
| student_id | student_name | sex | birthday | email | classes_id |
+-----+-----+-----+-----+-----+
| 1001 | zhangsan | m | 1988-01-01 | qqq@163.com | 10 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

- 向 t\_student 表中加入数据 (使用默认值)

```
drop table if exists t_student;
create table t_student(
 student_id int(10),
 student_name varchar(20),
 sex char(2) default 'm',
 birthday date,
 email varchar(30),
 classes_id int(3)
)
```

```
insert into t_student(student_id, student_name, birthday, email, classes_id)
values
(1002, 'zhangsan', '1988-01-01', 'qqq@163.com', 10)
```

```
mysql> select * from t_student;
+-----+-----+-----+-----+-----+
| student_id | student_name | sex | birthday | email | classes_id |
+-----+-----+-----+-----+-----+
| 1002 | zhangsan | m | 1988-01-01 | qqq@163.com | 10 |
+-----+-----+-----+-----+-----+
1 row in set (0.02 sec)
```

## 13.2、增加/删除/修改表结构

采用 alter table 来增加/删除/修改表结构，不影响表中的数据

### 13.2.1、添加字段

如：需求发生改变，需要向 t\_student 中加入联系电话字段，字段名称为：contact\_tel 类型为 varchar(40)

```
alter table t_student add contact_tel varchar(40);
```

```
mysql> alter table t_student add(contact_tel varchar(40));
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t_student;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
student_id	int<10>	NO	PRI	0	
student_name	varchar<50>	NO		NULL	
sex	char<2>	NO		NULL	
birthday	date	NO		NULL	
email	varchar<30>	YES	UNI	NULL	
classes_id	int<3>	NO	MUL	NULL	
contact_tel	varchar<40>	YES		NULL	
+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

### 13.2.2、修改字段

如：student\_name 无法满足需求，长度需要更改为 100

```
alter table t_student modify student_name varchar(100);
```

```
mysql> alter table t_student modify student_name varchar(100) ;
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t_student;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
student_id	int(10)	NO	PRI	0	
student_name	varchar(100)	YES		NULL	
sex	char(2)	NO		NULL	
birthday	date	NO		NULL	
email	varchar(30)	YES	UNI	NULL	
classes_id	int(3)	NO	MUL	NULL	
contact_tel	varchar(40)	YES		NULL	
+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)
```

如 sex 字段名称感觉不好，想用 gender 那么就需要更爱列的名称

```
mysql> alter table t_student change sex gender char(2) not null;
Query OK, 0 rows affected (0.38 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t_student;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
student_id	int(10)	NO	PRI	0	
student_name	varchar(100)	YES		NULL	
gender	char(2)	NO		NULL	
birthday	date	NO		NULL	
email	varchar(30)	YES	UNI	NULL	
classes_id	int(3)	NO	MUL	NULL	
contact_tel	varchar(40)	YES		NULL	
+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)
```

### 13.2.3、删除字段

如：删除联系电话字段

```
alter table t_student drop contact_tel;
```

```
mysql> alter table t_student drop contact_tel;
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t_student;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
student_id	int(10)	NO	PRI	0	
student_name	varchar(100)	YES		NULL	
gender	char(2)	NO		NULL	
birthday	date	NO		NULL	
email	varchar(30)	YES	UNI	NULL	
classes_id	int(3)	NO	MUL	NULL	
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

## 13.3、添加、修改和删除

### 13.3.1、insert

添加、修改和删出都属于 DML，主要包含的语句：insert、update、delete

- Insert 语法格式

```
Insert into 表名(字段,....) values(值,.....)
```

- 省略字段的插入

```
insert into emp values(9999,'zhangsan','MANAGER', null, null,3000, 500, 10);
```

```
mysql> insert into emp values(9999,'zhangsan','MANAGER', null, null,3000, 500, 10);
Query OK, 1 row affected (0.00 sec)
```

```

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
9999	zhangsan	MANAGER	NULL	NULL	3000.00	500.00	10
+-----+-----+-----+-----+-----+-----+-----+-----+
15 rows in set (0.00 sec)

```

不建议使用此种方式，因为当数据库表中的字段位置发生改变的时候会影响到 insert 语句

- 指定字段的插入(建议使用此种方式)

```

insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
values(9999,'zhangsan','MANAGER',null,null,3000,500,10);

```

```

mysql> insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno) values(9999,'zhangsan','MANAGER',null,null,3000,500,10);
ERROR 1062 (23000): Duplicate entry '9999' for key 'PRIMARY'

```

出现了主键重复的错误，主键表示了记录的唯一性，不能重复

```

mysql> insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno) values(9998,'zhangsan','MANAGER',null,null,3000,500,10);
Query OK, 1 row affected (0.00 sec)

```

如何插入日期：

第一种方法，插入的日期格式和显示的日期格式一致

```

insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)
values(9997,'zhangsan','MANAGER',null,'1981-06-12',3000,500,10);

```

```
mysql> insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno) values(9997,'zhangsan','MANAGER', null, '1981-06-12',3000, 500, 10);
Query OK, 1 row affected (0.01 sec)
```

第二种方法，采用 str\_to\_date

```
insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno) values(9996,'zhangsan','MANAGER',null,str_to_date('1981-06-12','%Y-%m-%d'),3000, 500, 10);
mysql> insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno) values(9996,'zhangsan','MANAGER',null,str_to_date('1981-06-12','%Y-%m-%d'),3000, 500, 10);
Query OK, 1 row affected (0.00 sec)
```

第三种方法，添加系统日期 (now())

```
insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno) values(9995,'zhangsan','MANAGER',null,now(),3000, 500, 10);
mysql> insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno) values(9995,'zhangsan','MANAGER',null,now(),3000, 500, 10);
Query OK, 1 row affected, 1 warning (0.01 sec)
```

```
mysql> select * from emp where empno = 9995;
+-----+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
+-----+-----+-----+-----+-----+-----+-----+
| 9995 | zhangsan | MANAGER | NULL | 2014-06-20 | 3000.00 | 500.00 | 10 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

## ● 表复制

```
create table emp_bak as select empno,ename,sal from emp;
```

```
mysql> create table emp_0128 as select * from emp;
Query OK, 20 rows affected (0.06 sec)
Records: 20 Duplicates: 0 Warnings: 0

mysql> desc emp_0128;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
EMPNO	int<4>	NO		NULL	
ENAME	varchar<10>	YES		NULL	
JOB	varchar<9>	YES		NULL	
MGR	int<4>	YES		NULL	
HIREDATE	date	YES		NULL	
SAL	double<7,2>	YES		NULL	
COMM	double<7,2>	YES		NULL	
DEPTNO	int<2>	YES		NULL	
+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```

```
mysql> select * from emp_0128;
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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
9994	zhangsan	MANAGER	NULL	2014-06-20	3000.00	500.00	10
9995	zhangsan	MANAGER	NULL	2014-06-20	3000.00	500.00	10
9996	zhangsan	MANAGER	NULL	1981-06-12	3000.00	500.00	10
9997	zhangsan	MANAGER	NULL	1981-06-12	3000.00	500.00	10

```

以上方式，会自动创建表，将符合查询条件的数据自动复制到创建的表中

- 如何将查询的数据直接放到已经存在的表中，可以使用条件

```
insert into emp_bak select * from emp where sal=3000;
```

```
mysql> insert into emp_0128 select * from emp where sal=3000;
Query OK, 8 rows affected (0.00 sec)
Records: 8 Duplicates: 0 Warnings: 0
```

### 13.3.2、update

可以修改数据，可以根据条件修改数据

- 语法格式：

```
update 表名 set 字段名称 1=需要修改的值 1, 字段名称 2=需要修改的值 2 where
```

- 将 job 为 manager 的员工的工资上涨 10%

```
update emp set sal=sal+sal*0.1 where job='MANAGER';
```

### 13.3.3、delete

可以删除数据，可以根据条件删除数据

- 语法格式：

```
Delete from 表名 where
```

- 删除津贴为 500 的员工

```
delete from emp where comm=500;
```

- 删除津贴为 null 的员工

```
delete from emp where comm is null;
```

### 13.4、创建表加入约束

- 常见的约束
  - a) 非空约束, not null
  - b) 唯一约束, unique
  - c) 主键约束, primary key
  - d) 外键约束, foreign key
  - e) 自定义检查约束, check (不建议使用) (在 mysql 中现在还不支持)

#### 13.4.1、非空约束, not null

非空约束，针对某个字段设置其值不能为空，如：学生的姓名不能为空

```
drop table if exists t_student;
create table t_student(
 student_id int(10),
 student_name varchar(20) not null,
 sex char(2) default 'm',
 birthday date,
 email varchar(30),
 classes_id int(3)
)

insert into t_student(student_id, birthday, email, classes_id)
values
```

```
(1002, '1988-01-01', 'qqq@163.com', 10)
```

```
mysql> insert into t_student(student_id, birthday, email, classes_id)
-> values
-> <1002, '1988-01-01', 'qqq@163.com', 10>
-> ;
ERROR 1364 (HY000): Field 'student_name' doesn't have a default value
```

以上错误为加入的学生姓名为空。

### 13.4.2、唯一约束, unique

唯一性约束, 它可以使某个字段的值不能重复, 如: email 不能重复:

```
drop table if exists t_student;
create table t_student(
 student_id int(10),
 student_name varchar(20) not null,
 sex char(2) default 'm',
 birthday date,
 email varchar(30) unique,
 classes_id int(3)
)
insert into t_student(student_id, student_name , sex, birthday, email, classes_id)
values
(1001,'zhangsan','m', '1988-01-01', 'qqq@163.com', 10)
```

```
mysql> insert into t_student<student_id,student_name,sex, birthday, email, class
es_id>
-> values
-> <1002,'zhangsan','m', '1988-01-01', 'qqq@163.com', 10>;
ERROR 1062 (23000): Duplicate entry 'qqq@163.com' for key 'email'
```

以上插入了重复的 email, 所以出现了“违反唯一约束错误”, 所以 unique 起作用了  
同样可以为唯一约束起个约束名

- 我们可以查看一下约束

```
mysql> use information_schema;
```

```
mysql> select * from table_constraints where table_name = 't_student';
```

```
mysql> select * from table_constraints where table_name = 't_student' \G
***** 1. row *****
CONSTRAINT_CATALOG: NULL
CONSTRAINT_SCHEMA: bjpowernode
CONSTRAINT_NAME: email
TABLE_SCHEMA: bjpowernode
TABLE_NAME: t_student
CONSTRAINT_TYPE: UNIQUE
1 row in set (0.00 sec)
```

关于约束名称可以到 table\_constraints 中查询  
以上约束的名称我们也可以自定义。

```
drop table if exists t_student;
create table t_student(
 student_id int(10),
 student_name varchar(20) not null,
 sex char(2) default 'm',
 birthday date,
```

```

email varchar(30) ,
classes_id int(3) ,
constraint email_unique unique(email)/*表级约束*/
)

```

### 13.4.3、主键约束， primary key

每个表应该具有主键，主键可以标识记录的唯一性，主键分为单一主键和复合（联合）主键，单一主键是由一个字段构成的，复合（联合）主键是由多个字段构成的

```

drop table if exists t_student;
create table t_student()
student_id int(10) primary key,/*列级约束*/
student_name varchar(20) not null,
sex char(2) default 'm',
birthday date,
email varchar(30) ,
classes_id int(3)
)
insert into t_student(student_id, student_name , sex, birthday, email, classes_id)
values
(1001,'zhangsan','m', '1988-01-01', 'qqq@163.com', 10)

```

向以上表中加入学号为 1001 的两条记录，出现如下错误，因为加入了主键约束

```

mysql> select * from t_student;
+-----+-----+-----+-----+-----+-----+
| student_id | student_name | sex | birthday | email | classes_id |
+-----+-----+-----+-----+-----+-----+
| 1001 | zhangsan | m | 1988-01-01 | qqq@163.com | 10 |
+-----+-----+-----+-----+-----+
1 row in set <0.00 sec>

mysql> insert into t_student(student_id, student_name , sex, birthday, email, classes_id)
-> values
-> <1001,'zhangsan','m', '1988-01-01', 'qqq@163.com', 10>;
ERROR 1062 (23000): Duplicate entry '1001' for key 'PRIMARY'

```

我们也可以通过表级约束为约束起个名称：

```

drop table if exists t_student;
create table t_student(
student_id int(10),
student_name varchar(20) not null,
sex char(2) default 'm',
birthday date,
email varchar(30) ,
classes_id int(3),
CONSTRAINT p_id PRIMARY key (student_id)
)

```

```
insert into t_student(student_id, student_name, sex, birthday, email, classes_id)
values
(1001,'zhangsan','m', '1988-01-01', 'qqq@163.com', 10)
```

### 13.4.4、外键约束, foreign key

外键主要是维护表之间的关系的，主要是为了保证参照完整性，如果表中的某个字段为外键字段，那么该字段的值必须来源于参照的表的主键，如：emp 中的 deptno 值必须来源于 dept 表中的 deptno 字段值。

建立学生和班级表之间的连接

首先建立班级表 t\_classes

```
drop table if exists t_classes;
create table t_classes(
 classes_id int(3),
 classes_name varchar(40),
 constraint pk_classes_id primary key(classes_id)
)
```

在 t\_student 中加入外键约束

```
drop table if exists t_student;
create table t_student(
 student_id int(10),
 student_name varchar(20),
 sex char(2),
 birthday date,
 email varchar(30),
 classes_id int(3),
 constraint student_id_pk primary key(student_id),
 constraint fk_classes_id foreign key(classes_id) references t_classes(classes_id)
)
```

向 t\_student 中加入数据

```
insert into t_student(student_id, student_name, sex, birthday, email, classes_id) values(1001,
'zhangsan', 'm', '1988-01-01', 'qqq@163.com', 10)
```

```
mysql> insert into t_student(student_id, student_name, sex, birthday, email, classes_id) values(1001, 'zhangsan', 'm', '1988-01-01', 'qqq@163.com', 10);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails ('bjpowernode'.t_student', CONSTRAINT 'fk_classes_id' FOREIGN KEY ('classes_id') REFERENCES 't_classes' ('classes_id'))
```

出现错误，因为在班级表中不存在班级编号为 10 班级，外键约束起到了作用

存在外键的表就是子表，参照的表就是父表，所以存在一个父子关系，也就是主从关系，主表就是班级表，从表就是学生表

```
mysql> insert into t_student(student_id, student_name, sex, birthday, email, classes_id) values(1001, 'zhangsan', 'm', '1988-01-01', 'qqq@163.com', null);
Query OK, 1 row affected (0.00 sec)
```

以上成功的插入了学生信息，当时 classes\_id 没有值，这样会影响参照完整性，所以我们建

议将外键字段设置为非空

```
drop table if exists t_student;
create table t_student(
 student_id int(10),
 student_name varchar(20),
 sex char(2),
 birthday date,
 email varchar(30),
 classes_id int (3) not null,
 constraint student_id_pk primary key(student_id),
 constraint fk_classes_id foreign key(classes_id) references t_classes(classes_id)
)
insert into t_student(student_id, student_name, sex, birthday, email, cla
sses_id) values(1001, 'zhangsan', 'm', '1988-01-01', 'qqq@163.com', null);
```

再次插入班级编号为 null 的数据

```
mysql> insert into t_student(student_id, student_name, sex, birthday, email, cla
sses_id) values(1001, 'zhangsan', 'm', '1988-01-01', 'qqq@163.com', null);
ERROR 1048 (23000): Column 'classes_id' cannot be null
```

添加数据到班级表，添加数据到学生表，删除班级数据，将会出现如下错误：

```
insert into t_classes (classes_id,classes_name) values (10,'366');

insert into t_student(
 student_id, student_name, sex, birthday, email, classes_id
) values(
 1001, 'zhangsan', 'm', '1988-01-01', 'qqq@163.com', 10
)
```

```
mysql> update t_classes set classes_id = 20 where classes_name = '366';
```

```
mysql> update t_classes set classes_id = 20 where classes_name = '366';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails ('bjpowernode'.'t_student', CONSTRAINT 'fk_classes_id' FOREIGN KEY ('cl
asses_id') REFERENCES 't_classes' ('classes_id'))
```

因为子表（t\_student）存在一个外键 classes\_id，它参照了父表（t\_classes）中的主键，所以先删除子表中的引用记录，再修改父表中的数据。

我们也可以采取以下措施 级联更新。

```
mysql> delete from t_classes where classes_id = 10;
```

```
mysql> delete from t_classes where classes_id = 10;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails ('bjpowernode'.'t_student', CONSTRAINT 'fk_classes_id' FOREIGN KEY ('cl
asses_id') REFERENCES 't_classes' ('classes_id'))
```

因为子表（t\_student）存在一个外键 classes\_id，它参照了父表（t\_classes）中的主键，所以先删除父表，那么将会影响子表的参照完整性，所以正确的做法是，先删除子表中的数据，再删除父表中的数据，采用 drop table 也不行，必须先 drop 子表，再 drop 父表

我们也可以采取以下措施 级联删除。

## 13.4.5、级联更新与级联删除

### 13.4.5.1、on update cascade;

mysql 对有些约束的修改比较麻烦，所以我们可以先删除，再添加

```
alter table t_student drop foreign key fk_classes_id;

alter table t_student add constraint fk_classes_id_1 foreign key(classes_id) references
t_classes(classes_id) on update cascade;
```

```
mysql> update t_classes set classes_id = 20 where classes_name = '366';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select *from t_classes;
+-----+-----+
| classes_id | classes_name |
+-----+-----+
| 20 | 366 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select *from t_student;
+-----+-----+-----+-----+-----+-----+
| student_id | student_name | sex | birthday | email | classes_id |
+-----+-----+-----+-----+-----+-----+
| 1001 | zhangsan | m | 1988-01-01 | qqq@163.com | 20 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

我们只修改了父表中的数据，但是子表中的数据也会跟着变动。

### 13.4.5.2、on delete cascade;

mysql 对有些约束的修改时不支持的，所以我们可以先删除，再添加

```
alter table t_student drop foreign key fk_classes_id;

alter table t_student add constraint fk_classes_id_1 foreign key(classes_id) references
t_classes(classes_id) on delete cascade;
delete from t_classes where classes_id = 20;
```

```
mysql> delete from t_classes where classes_id = 20;
Query OK, 1 row affected (0.00 sec)

mysql> select *from t_student;
Empty set (0.01 sec)

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

我们只删除了父表中的数据，但是子表也会中的数据也会删除。

## 13.5、t\_student 和 t\_classes 完整示例

```
drop table if exists t_classes;
create table t_classes(
 classes_id int (3),
 classes_name varchar(30) not null,
 constraint pk_classes_id primary key(classes_id)
)

drop table if exists t_student;
create table t_student(
 student_id int(10),
 student_name varchar(50) not null,
 sex char(2) not null,
 birthday date not null,
 email varchar(30) unique,
 classes_id int (3) not null,
 constraint pk_student_id primary key(student_id),
 constraint fk_classes_id foreign key(classes_id) references t_classes(classes_id)
)
```

## 14、存储引擎（了解）

### 14.1、存储引擎的使用

- 数据库中的各表均被（在创建表时）指定的存储引擎来处理。
- 服务器可用的引擎依赖于以下因素：
  - MySQL 的版本

- 服务器在开发时如何被配置
- 启动选项
- 为了解当前服务器中有哪些存储引擎可用，可使用 SHOW ENGINES 语句：

```
mysql> SHOW ENGINES\G
```

```
mysql> show engines \G
***** 1. row *****
 Engine: MyISAM
 Support: YES
 Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
 XA: NO
Savepoints: NO
***** 2. row *****
 Engine: CSU
 Support: YES
 Comment: CSU storage engine
Transactions: NO
 XA: NO
Savepoints: NO
***** 3. row *****
 Engine: MRG_MYISAM
 Support: YES
 Comment: Collection of identical MyISAM tables
Transactions: NO
 XA: NO
Savepoints: NO
***** 4. row *****
 Engine: BLACKHOLE
 Support: YES
 Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
 XA: NO
Savepoints: NO
***** 5. row *****
 Engine: FEDERATED
 Support: NO
 Comment: Federated MySQL storage engine
Transactions: NULL
 XA: NULL
Savepoints: NULL
***** 6. row *****
 Engine: InnoDB
 Support: DEFAULT
 Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
 XA: YES
Savepoints: YES
***** 7. row *****
 Engine: ARCHIVE
 Support: YES
 Comment: Archive storage engine
Transactions: NO
 XA: NO
```

- 在创建表时，可使用 ENGINE 选项为 CREATE TABLE 语句显式指定存储引擎。  

```
CREATE TABLE TABLENAME (NO INT) ENGINE = MyISAM;
```
- 如果在创建表时没有显式指定存储引擎，则该表使用当前默认的存储引擎
- 默认的存储引擎可在 my.ini 配置文件中使用 default-storage-engine 选项指定。
- 现有表的存储引擎可使用 ALTER TABLE 语句来改变：  

```
ALTER TABLE TABLENAME ENGINE = INNODB;
```

- 为确定某表所使用的存储引擎,可以使用 SHOW CREATE TABLE 或 SHOW TABLE STATUS 语句:

```
mysql> SHOW CREATE TABLE emp\G
mysql> SHOW TABLE STATUS LIKE 'emp' \G
```

## 14.2、常用的存储引擎

### 14.2.1、MyISAM 存储引擎

- MyISAM 存储引擎是 MySQL 最常用的引擎。
- 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)
  - 灵活的 AUTO\_INCREMENT 字段处理
  - 可被转换为压缩、只读表来节省空间

### 14.2.2、InnoDB 存储引擎

- InnoDB 存储引擎是 MySQL 的缺省引擎。
- 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
  - 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

### 14.2.3、MEMORY 存储引擎

- 使用 MEMORY 存储引擎的表, 其数据存储在内存中, 且行的长度固定, 这两个特点使得 MEMORY 存储引擎非常快。

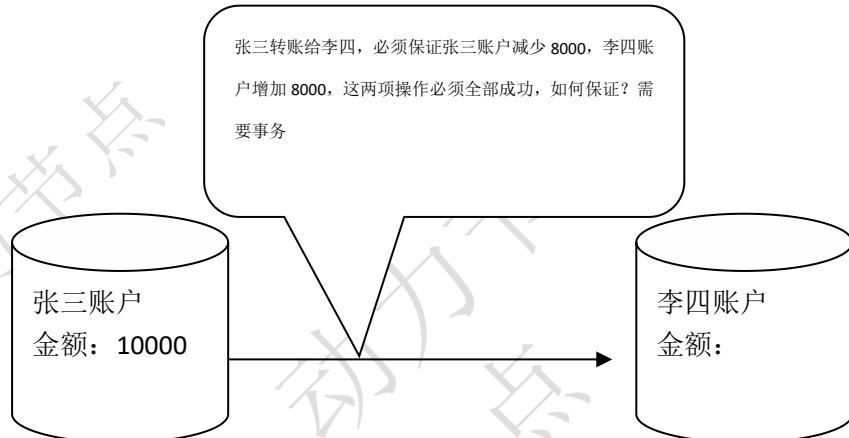
- MEMORY 存储引擎管理的表具有下列特征：
  - 在数据库目录内，每个表均以.frm 格式的文件表示。
  - 表数据及索引被存储在内存中。
  - 表级锁机制。
  - 不能包含 TEXT 或 BLOB 字段。
- MEMORY 存储引擎以前被称为 HEAP 引擎。

### 14.3、选择合适的存储引擎

- MyISAM 表最适合于大量的数据读而少量数据更新的混合操作。MyISAM 表的另一种适用情形是使用压缩的只读表。
- 如果查询中包含较多的数据更新操作，应使用 InnoDB。其行级锁机制和多版本的支持为数据读取和更新的混合操作提供了良好的并发机制。
- 可使用 MEMORY 存储引擎来存储非永久需要的数据，或者是能够从基于磁盘的表中重新生成的数据。

## 15、事务

### 15.1、概述



事务可以保证多个操作原子性，要么全成功，要么全失败。对于数据库来说事务保证批量的 DML 要么全成功，要么全失败。事务具有四个特征 ACID

- a) 原子性 (Atomicity)
  - 整个事务中的所有操作，必须作为一个单元全部完成（或全部取消）。
- b) 一致性 (Consistency)
  - 在事务开始之前与结束之后，数据库都保持一致状态。
- c) 隔离性(Isolation)
  - 一个事务不会影响其他事务的运行。

## d) 持久性(Durability)

- 在事务完成以后，该事务对数据库所作的更改将持久地保存在数据库之中，并不会被回滚。

事务中存在一些概念：

- 事务 (Transaction): 一批操作 (一组 DML)
- 开启事务 (Start Transaction)
- 回滚事务 (rollback)
- 提交事务 (commit)
- SET AUTOCOMMIT: 禁用或启用事务的自动提交模式

当执行 DML 语句时其实就是开启一个事务

关于事务的回滚需要注意：只能回滚 insert、delete 和 update 语句，不能回滚 select (回滚 select 没有任何意义)，对于 create、drop、alter 这些无法回滚。

事务只对 DML 有效果。

注意： rollback，或者 commit 后事务就结束了。

## 15.2、事务的提交与回滚演示

1) 创建表

```
create table user(
 id int (11) primary key not null auto_increment ,
 username varchar(30),
 password varchar(30)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

2) 查询表中数据

```
mysql> select * from user;
Empty set <0.00 sec>
```

3) 开启事务 START TRANSACTION;

4) 插入数据

```
insert into user (username,password) values ('zhangsan','123');
```

```
mysql> insert into user (username,password) values ('zhangsan','123');
Query OK, 1 row affected <0.00 sec>
```

5) 查看数据

```
mysql> select * from user;
+----+-----+-----+
| id | username | password |
+----+-----+-----+
| 1 | zhangsan | 123 |
+----+-----+-----+
1 row in set (0.00 sec)
```

6) 修改数据

```
mysql> update user set username = 'lisi' where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

7) 查看数据

```
mysql> select * from user;
+----+-----+-----+
| id | username | password |
+----+-----+-----+
| 1 | lisi | 123 |
+----+-----+-----+
1 row in set (0.00 sec)
```

8) 回滚事务

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

9) 查看数据

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

### 15.3、自动提交模式

- 自动提交模式用于决定新事务如何及何时启动。
- 启用自动提交模式:
  - 如果自动提交模式被启用，则单条 DML 语句将缺省地开始一个新的事务。
  - 如果该语句执行成功，事务将自动提交，并永久地保存该语句的执行结果。
  - 如果语句执行失败，事务将自动回滚，并取消该语句的结果。
  - 在自动提交模式下，仍可使用 START TRANSACTION 语句来显式地启动事务。这时，一个事务仍可包含多条语句，直到这些语句被统一提交或回滚。
- 禁用自动提交模式:
  - 如果禁用自动提交，事务可以跨越多条语句。

- 在这种情况下，事务可以用 COMMIT 和 ROLLBACK 语句来显式地提交或回滚。
- 自动提交模式可以通过服务器变量 AUTOCOMMIT 来控制。
- 例如：

```
mysql> SET AUTOCOMMIT = OFF;
mysql> SET AUTOCOMMIT = ON;
或
mysql> SET SESSION AUTOCOMMIT = OFF;
mysql> SET SESSION AUTOCOMMIT = ON;
show variables like '%auto%'; -- 查看变量状态
```

## 15.4、事务的隔离级别

### 15.4.1、隔离级别

- 事务的隔离级别决定了事务之间可见的级别。
- 当多个客户端并发地访问同一个表时，可能出现下面的一致性问题：
  - 脏读取（Dirty Read）

一个事务开始读取了某行数据，但是另外一个事务已经更新了此数据但没有能够及时提交，这就出现了脏读取。

- 不可重复读（Non-repeatable Read）  
在同一个事务中，同一个读操作对同一个数据的前后两次读取产生了不同的结果，这就是不可重复读。

幻像读是指在同一个事务中以前没有的行，由于其他事务的提交而出现的新行。

### 15.4.2、四个隔离级别

- InnoDB 实现了四个隔离级别，用以控制事务所做的修改，并将修改通告至其它并发的事务：
  - 读未提交（READ UCOMMITTED）

允许一个事务可以看到其他事务未提交的修改。

- 读已提交（READ COMMITTED）  
允许一个事务只能看到其他事务已经提交的修改，未提交的修改是不可见的。

确保如果在一个事务中执行两次相同的 SELECT 语句，都能得到相同的结果，不管其他事务是否提交这些修改。（银行总账）  
该隔离级别为 InnoDB 的缺省设置。

## - 串行化 (SERIALIZABLE) 【序列化】

将一个事务与其他事务完全地隔离。

例:A 可以开启事物,B 也可以开启事物

A 在事物中执行 DML 语句时,未提交

B 不以执行 DML,DQL 语句

### 15.4.3、隔离级别与一致性问题的关系

| 隔离级别 | 脏读取 | 不可重复读 | 幻像读          |
|------|-----|-------|--------------|
| 读未提交 | 可能  | 可能    | 可能           |
| 读已提交 | 不可能 | 可能    | 可能           |
| 可重复读 | 不可能 | 不可能   | 对 InnoDB 不可能 |
| 串行化  | 不可能 | 不可能   | 不可能          |

### 15.4.4、设置服务器缺省隔离级别

#### 通过修改配置文件设置

- 可以在 my.ini 文件中使用 transaction-isolation 选项来设置服务器的缺省事务隔离级别。
- 该选项值可以是：
  - READ-UNCOMMITTED
  - READ-COMMITTED
  - REPEATABLE-READ
  - SERIALIZABLE

- 例如：

```
[mysqld]
transaction-isolation = READ-COMMITTED
```

## 通过命令动态设置隔离级别

- 隔离级别也可以在运行的服务器中动态设置，应使用 **SET TRANSACTION ISOLATION LEVEL** 语句。
- 其语法模式为：  
**SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL <isolation-level>**  
其中的<isolation-level>可以是：
  - READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE
- 例如： **SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;**

### 15.4.5、隔离级别的作用范围

- 事务隔离级别的作用范围分为两种：
  - 全局级：对所有的会话有效
  - 会话级：只对当前的会话有效
- 例如，设置会话级隔离级别为 **READ COMMITTED**：  
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
或：  
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;  
• 设置全局级隔离级别为 **READ COMMITTED**：  
mysql> SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED;

### 15.4.6、查看隔离级别

- 服务器变量 **tx\_isolation**（包括会话级和全局级两个变量）中保存着当前的会话隔离级别。
- 为了查看当前隔离级别，可访问 **tx\_isolation** 变量：
  - 查看会话级的当前隔离级别：

```
mysql> SELECT @@tx_isolation;
或：
```

```
mysql> SELECT @@session.tx_isolation;
```

- 查看全局级的当前隔离级别:

```
mysql> SELECT @@global.tx_isolation;
```

### 15.4.7、并发事务与隔离级别示例

#### read uncommitted(未提交读) -- 脏读(Dirty Read):

| 会话一                                                         | 会话二                    |
|-------------------------------------------------------------|------------------------|
| mysql> prompt s1>                                           | mysql> use bjpowernode |
| s1>use bjpowernode                                          | mysql> prompt s2>      |
| s1>create table tx (                                        |                        |
| id int(11),                                                 |                        |
| num int (10)                                                |                        |
| );                                                          |                        |
| s1>set global transaction isolation level read uncommitted; |                        |
| s1>start transaction;                                       |                        |
|                                                             | s2>start transaction;  |
| s1>insert into tx values (1,10);                            |                        |
|                                                             | s2>select * from tx;   |
| s1>rollback;                                                |                        |
|                                                             | s2>select * from tx;   |

#### read committed(已提交读)

| 会话一                                                        | 会话二                   |
|------------------------------------------------------------|-----------------------|
| s1> set global transaction isolation level read committed; |                       |
| s1>start transaction;                                      |                       |
|                                                            | s2>start transaction; |
| s1>insert into tx values (1,10);                           |                       |
| s1>select * from tx;                                       |                       |
|                                                            | s2>select * from tx;  |
| s1>commit;                                                 |                       |
|                                                            | s2>select * from tx;  |

## repeatable read(可重复读)

| 会话一                                                         | 会话二                   |
|-------------------------------------------------------------|-----------------------|
| s1> set global transaction isolation level repeatable read; |                       |
| s1>start transaction;                                       | s2>start transaction; |
| s1>select * from tx;                                        |                       |
| s1>insert into tx values (1,10);                            |                       |
|                                                             | s2>select * from tx;  |
| s1>commit;                                                  |                       |
|                                                             | s2>select * from tx;  |

# 16、索引

## 16.1、索引原理

索引被用来快速找出在一个列上用一特定值的行。没有索引，MySQL 不得不首先以第一条记录开始，然后读完整个表直到它找出相关的行。表越大，花费时间越多。对于一个有序字段，可以运用二分查找（Binary Search），这就是为什么性能能得到本质上的提高。

MYISAM 和 INNODB 都是用 B+Tree 作为索引结构

（主键，unique 都会默认的添加索引）

## 16.2、索引的应用

### 16.2.1、创建索引

如果未使用索引，我们查询 工资大于 1500 的会执行全表扫描

```
mysql> explain select sal from emp where sal > 1500;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | emp | ALL | NULL | NULL | NULL | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.02 sec)
```

什么时候需要给字段添加索引：

- 表中该字段中的数据量庞大
- 经常被检索，经常出现在 where 子句中的字段
- 经常被 DML 操作的字段不建议添加索引

索引等同于一本书的目录

主键会自动添加索引，所以尽量根据主键查询效率较高。

如经常根据 `sal` 进行查询，并且遇到了性能瓶颈，首先查看程序是否存算法问题，再考虑对 `sal` 建立索引，建立索引如下：

```
1、create unique index 索引名 on 表名(列名);
 create unique index u_ename on emp(ename);
2、alter table 表名 add unique index 索引名 (列名);
create index test_index on emp (sal);
```

```
mysql> create index test_index on emp (sal);
Query OK, 14 rows affected (0.02 sec)
Records: 14 Duplicates: 0 Warnings: 0
```

### 16.2.2、查看索引

```
show index from emp;
```

```
mysql> show index from emp;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
emp	0	PRIMARY	1	EMPNO	A	2	NULL	NULL	YES	BTREE	
emp	1	DEPTNO	1	DEPTNO	A	2	NULL	NULL	YES	BTREE	
emp	1	test_index	1	SAL	A	2	NULL	NULL	YES	BTREE	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

### 16.2.3、使用索引

注意一定不可以使用 `select * ...` 可以看到 `type!=all` 了，说明使用了索引

```
explain select sal from emp where sal > 1500;
```

条件中的 `sal` 使用了索引

```
mysql> explain select sal from emp where sal > 1500;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | emp | range | test_index | test_index | 9 | NULL | 8 | Using where; Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

如下图：假如我们要查找 `sal` 大于 1500 的所有行，那么可以扫描索引，索引时排序的，结果得出 7 行，我们知道不会再有匹配的记录，可以退出了。

如果查找一个值，它在索引表中某个中间点以前不会出现，那么也有找到其第一个匹配索引项的定位算法，而不用进行表的顺序扫描（如二分查找法）。

这样，可以快速定位到第一个匹配的值，以节省大量搜索时间。数据库利用了各种各样的快速定位索引值的技术，通常这些技术都属于 DBA 的工作。

## 16.2.4、删除索引

```
DROP INDEX index_name ON table_name

ALTER TABLE table_name DROP INDEX index_name

ALTER TABLE table_name DROP PRIMARY KEY
```

其中，前两条语句是等价的，删除掉 table\_name 中的索引 index\_name。

第 3 条语句只在删除 PRIMARY KEY 索引时使用，因为一个表只可能有一个 PRIMARY KEY 索引，

```
mysql> ALTER TABLE EMP DROP INDEX test_index;
```

删除后就不再使用索引了，查询会执行全表扫描。

```
mysql> explain select sal from emp where sal > 1500;
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | emp | ALL | NULL | NULL | NULL | NULL | 16 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

## 17、视图

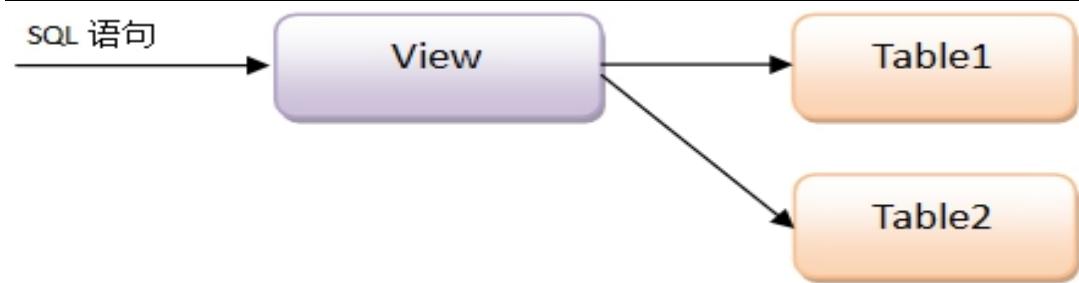
### 17.1、什么是视图

- 视图是一种根据查询（也就是 SELECT 表达式）定义的数据库对象，用于获取想要看到和使用的局部数据。
- 视图有时也被成为“虚拟表”。
- 视图可以被用来从常规表（称为“基表”）或其他视图中查询数据。
- 相对于从基表中直接获取数据，视图有以下好处：
  - 访问数据变得简单
  - 可被用来对不同用户显示不同的表的内容

用来协助适配表的结构以适应前端现有的应用程序

视图作用：

- 提高检索效率
- 隐藏表的实现细节【面向视图检索】



## 17.2、创建视图

如下示例：查询员工的姓名，部门，工资入职信息等信息。

```
select ename,dname,sal,hiredate,e.deptno from emp e,dept d where e.deptno
= e.deptno and e.deptno = 10;
```

为什么使用视图？因为需求决定以上语句需要在多个地方使用，如果频繁的拷贝以上代码，会给维护带来成本，视图可以解决这个问题

```
create view v_dept_emp as select ename,dname,sal,hiredate,e.deptno from emp e,dept d where
e.deptno
= e.deptno and e.deptno = 10;
```

```
create view v_dept_avg_sal_grade as select a.deptno, a.avg_sal, b.grade
from (select deptno, avg(sal) avg_sal from emp group by deptno) a, salgrade b
where a.avg_sal between b.loosal and b.hisal; /*注意 mysql 不支持子查询创建视图*/
```

## 17.3、修改视图

```
alter view v_dept_emp as select ename,dname,sal,hiredate,e.deptno from e
mp e,dept d where e.deptno = 20;
```

## 17.4、删除视图

```
drop view if exists v_dept_emp;
```

## 18、DBA 命令（了解）

### 18.1、新建用户

```
CREATE USER username IDENTIFIED BY 'password';
```

说明:username——你将创建的用户名, password——该用户的登陆密码,密码可以为空,如果为空则该用户可以不需要密

码登陆服务器.

例如:

```
create user p361 identified by '123';
```

--可以登录但是只可以看见一个库 information\_schema

## 18.2、授权

### 命令详解

```
mysql> grant all privileges on dbname.tablename to 'username'@'login ip' identified by 'password' with grant option;
```

- 1) dbname=\*表示所有数据库
- 2) tablename=\*表示所有表
- 3) login ip=%表示任何 ip
- 4) password 为空, 表示不需要密码即可登录
- 5) with grant option; 表示该用户还可以授权给其他用户

#### ● 细粒度授权

首先以 root 用户进入 mysql, 然后键入命令: grant select,insert,update,delete on \*.\* to p361 @localhost Identified by "123"; 如果希望该用户能够在任何机器上登陆 mysql, 则将 localhost 改为 "%"。

#### ● 粗粒度授权

我们测试用户一般使用该命令授权,

```
GRANT ALL PRIVILEGES ON *.* TO 'p361'@'%' Identified by "123";
```

注意:用以上命令授权的用户不能给其它用户授权,如果想让该用户可以授权,用以下命令:

```
GRANT ALL PRIVILEGES ON *.* TO 'p361'@'%' Identified by "123" WITH GRANT OPTION;
```

#### privileges 包括:

- 1) alter: 修改数据库的表
- 2) create: 创建新的数据库或表
- 3) delete: 删除表数据
- 4) drop: 删除数据库/表
- 5) index: 创建/删除索引
- 6) insert: 添加表数据
- 7) select: 查询表数据
- 8) update: 更新表数据
- 9) all: 允许任何操作
- 10) usage: 只允许登录

## 18.3、回收权限

### 命令详解

```
revoke privileges on dbname[.tbname] from username;
revoke all privileges on *.* from p361;

use mysql
select * from user
进入 mysql 库中
修改密码;
update user set password = password('qwe') where user = 'p646';
刷新权限;
flush privileges
```

## 18.4、导出导入

### 18.4.1、导出

#### 18.4.1.1、导出整个数据库

在 windows 的 dos 命令窗口中执行: mysqldump bjpowernode>D:\bjpowernode.sql -uroot -p123

#### 18.4.1.2、导出指定库下的指定表

在 windows 的 dos 命令窗口中执行: mysqldump bjpowernode emp> D:\ bjpowernode.sql -uroot -p123

### 18.4.2、导入

登录 MYSQL 数据库管理系统之后执行: source D:\ bjpowernode.sql

## 19、数据库设计的三范式

### 19.1、第一范式

数据库表中不能出现重复记录，每个字段是原子性的不能再分  
不符合第一范式的示例

| 学生编号 | 学生姓名 | 联系方式                     |
|------|------|--------------------------|
| 1001 | 张三   | zs@gmail.com,13599999999 |
| 1002 | 李四   | ls@gmail.com,13699999999 |
| 1001 | 王五   | ww@163.net,13488888888   |

存在问题：

- 最后一条记录和第一条重复（不唯一，没有主键）
- 联系方式字段可以再分，不是原子性的

| 学生编号(pk) | 学生姓名 | email        | 联系电话        |
|----------|------|--------------|-------------|
| 1001     | 张三   | zs@gmail.com | 13599999999 |
| 1002     | 李四   | ls@gmail.com | 13699999999 |
| 1003     | 王五   | ww@163.net   | 13488888888 |

关于第一范式，每一行必须唯一，也就是每个表必须有主键，这是我们数据库设计的最基本要求，主要通常采用数值型或定长字符串表示，关于列不可再分，应该根据具体的情况来决定。如联系方式，为了开发上的便利可能就采用一个字段了。

## 19.2、第二范式

第二范式是建立在第一范式基础上的，另外要求所有非主键字段完全依赖主键，不能产生部分依赖

示例：

| 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  |

确定主键：

| 学生编号(PK) | 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  |

以上虽然确定了主键，但此表会出现大量的冗余，主要涉及到的冗余字段为“学生姓名”和“教师姓名”，出现冗余的原因在于，学生姓名部分依赖了主键的一个字段学生编号，而没有依赖教师编号，而教师姓名部门依赖了主键的一个字段教师编号，这就是第二范式部分依赖。

解决方案如下：

学生信息表

| 学生编号 (PK) | 学生姓名 |
|-----------|------|
| 1001      | 张三   |
| 1002      | 李四   |
| 1003      | 王五   |

教师信息表

| 教师编号 (PK) | 教师姓名 |
|-----------|------|
| 001       | 王老师  |

|     |     |
|-----|-----|
| 002 | 赵老师 |
|-----|-----|

教师和学生的关系表

| 学生编号(PK) fk→学生表的学生编号 | 教师编号(PK) fk→教师表的教师编号 |
|----------------------|----------------------|
| 1001                 | 001                  |
| 1002                 | 002                  |
| 1003                 | 001                  |
| 1001                 | 002                  |

如果一个表是单一主键，那么它就复合第二范式，部分依赖和主键有关系

以上是一种典型的“多对多”的设计

### 19.3、第三范式

建立在第二范式基础上的，非主键字段不能传递依赖于主键字段。(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      | 六    | 03   | 一年三班 |

从上表可以看出，班级名称字段存在冗余，因为班级名称字段没有直接依赖于主键，班级名称字段依赖于班级编号，班级编号依赖于学生编号，那么这就是传递依赖，解决的办法是将冗余字段单独拿出来建立表，如：

学生信息表

| 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      | 六    | 03        |

班级信息表

| 班级编号 (PK) | 班级名称 |
|-----------|------|
| 01        | 一年一班 |
| 02        | 一年二班 |
| 03        | 一年三班 |

以上设计是一种典型的一对多的设计，一存储在一张表中，多存储在一张表中，在多的那张表中添加外键指向一的一方的主键

### 19.4、三范式总结

第一范式：有主键，具有原子性，字段不可分割

第二范式：完全依赖，没有部分依赖

第三范式：没有传递依赖

数据库设计尽量遵循三范式，但是还是根据实际情况进行取舍，有时可能会拿冗余换速度，最终用目的要满足客户需求。

一对一设计，有两种设计方案：

第一种设计方案：主键共享

第二种设计方案：外键唯一

## 、作业

### 1、取得每个部门最高薪水的人员名称

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

### 2、哪些人的薪水在部门的平均薪水之上

```
+-----+-----+
| 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，如下：

```
+-----+
| deptno | avg(grade) |
+-----+
10	3.6667
20	2.8000
30	2.5000
+-----+
3 rows in set (0.05 sec)
```

#### 4、不准用组函数（Max），取得最高薪水

```
+-----+
| sal |
+-----+
| 5000.00 |
+-----+
1 row in set (0.00 sec)
```

#### 5、取得平均薪水最高的部门的部门编号

```
+-----+
| 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

```
+-----+
| 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

```
+-----+
| 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) 最高薪水还要高的领导人姓名

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

## 9、取得薪水最高的前五名员工

```
+-----+
| 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

```
+-----+-----+
| 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

```
+-----+-----+
| ename | hiredate |
+-----+-----+
ADAMS	1987-05-23
SCOTT	1987-04-19
MILLER	1982-01-23
FORD	1981-12-03
JAMES	1981-12-03
+-----+-----+
5 rows in set (0.00 sec)
```

## 12、取得每个薪水等级有多少员工

```
+-----+-----+
| grade | count(*) |
+-----+-----+
1	3
2	3
3	2
4	5
5	1
+-----+-----+
5 rows in set (0.00 sec)
```

## 13、面试题

有 3 个表 S(学生表), C (课程表), SC (学生选课表)

S (SNO, SNAME) 代表 (学号, 姓名)

C (CNO, CNAME, CTEACHER) 代表 (课号, 课名, 教师)

SC (SNO, CNO, SCGRADE) 代表 (学号, 课号, 成绩)

问题:

1, 找出没选过“黎明”老师的所有学生姓名。

2, 列出 2 门以上 (含 2 门) 不及格学生姓名及平均成绩。

3, 即学过 1 号课程又学过 2 号课所有学生的姓名。

请用标准 SQL 语言写出答案, 方言也行 (请说明是使用什么方言)。

```
CREATE TABLE SC
(
 SNO VARCHAR(200),
 CNO VARCHAR(200),
 SCGRADE VARCHAR(200)
);

CREATE TABLE S
(
 SNO VARCHAR(200),
 SNAME VARCHAR(200)
);

CREATE TABLE C
(
 CNO VARCHAR(200),
 CNAME VARCHAR(200),
 CTEACHER VARCHAR(200)
);

INSERT INTO C (CNO, CNAME, CTEACHER) VALUES ('1', '语文', '张');
INSERT INTO C (CNO, CNAME, CTEACHER) VALUES ('2', '政治', '王');
INSERT INTO C (CNO, CNAME, CTEACHER) VALUES ('3', '英语', '李');
INSERT INTO C (CNO, CNAME, CTEACHER) VALUES ('4', '数学', '赵');
INSERT INTO C (CNO, CNAME, CTEACHER) VALUES ('5', '物理', '黎明');
commit;

INSERT INTO S (SNO, SNAME) VALUES ('1', '学生 1');
INSERT INTO S (SNO, SNAME) VALUES ('2', '学生 2');
INSERT INTO S (SNO, SNAME) VALUES ('3', '学生 3');
INSERT INTO S (SNO, SNAME) VALUES ('4', '学生 4');
```

```
commit;
```

```
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('1', '1', '40');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('1', '2', '30');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('1', '3', '20');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('1', '4', '80');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('1', '5', '60');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('2', '1', '60');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('2', '2', '60');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('2', '3', '60');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('2', '4', '60');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('2', '5', '40');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('3', '1', '60');
INSERT INTO SC (SNO, CNO, SCGRADE) VALUES ('3', '3', '80');
commit;
```

问题 1. 找出没选过“黎明”老师的所有学生姓名。

即:

| sname |
|-------|
| 学生3   |
| 学生4   |

问题 2: 列出 2 门以上(含 2 门)不及格学生姓名及平均成绩。

|  |
|--|
|  |
|--|

问题 3: 即学过 1 号课程又学过 2 号课所有学生的姓名。

|  |
|--|
|  |
|--|

## 14、列出所有员工及领导的姓名

|  |
|--|
|  |
|--|

```
+-----+
| ename | ifnull(m.ename, '没有上级') |
+-----+
SMITH	FORD
ALLEN	BLAKE
WARD	BLAKE
JONES	KING
MARTIN	BLAKE
BLAKE	KING
CLARK	KING
SCOTT	JONES
KING	没有上级
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

```
+-----+
| empno | ename_e | 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.

```
mysql> select d.dname,e.* from emp e right join dept d on e.deptno = d.deptno;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| dname | empno | ename | job | mgr | hiredate | sal | comm | deptno |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
ACCOUNTING	7782	CLARK	MANAGER	7839	1981-06-09	2450.00	NULL	10
ACCOUNTING	7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
ACCOUNTING	7934	MILLER	CLERK	7782	1982-01-23	1300.00	NULL	10
RESEARCH	7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
RESEARCH	7566	JONES	MANAGER	7839	1981-04-02	2975.00	NULL	20
RESEARCH	7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20
RESEARCH	7876	ADAMS	CLERK	7788	1987-05-23	1100.00	NULL	20
RESEARCH	7902	FORD	ANALYST	7566	1981-12-03	3000.00	NULL	20
SALES	7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
SALES	7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30
SALES	7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30
SALES	7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
SALES	7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
SALES	7900	JAMES	CLERK	7698	1981-12-03	950.00	NULL	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

```
+-----+-----+
| 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.

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 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.

```
+-----+-----+
| ename | dname | cc |
+-----+-----+
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.

```
+-----+-----+
| job | count(e.empno) |
+-----+-----+
ANALYST	2
MANAGER	3
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.

```
+-----+
| ename |
+-----+
| ALLEN |
| WARD |
| MARTIN |
| BLAKE |
| TURNER |
| JAMES |
+-----+
6 rows in set (0.05 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.

| 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     | 5    |
| FORD  | RESEARCH   | JONES | 4    |

## 23、列出与"SCOTT"从事相同工作的所有员工及部门名称.

```
+-----+-----+
| 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.

```
+-----+-----+
| ename | sal | dname |
+-----+-----+
JONES	2975.00	RESEARCH
SCOTT	3000.00	RESEARCH
KING	5000.00	ACCOUNTING
FORD	3000.00	RESEARCH
+-----+-----+
4 rows in set (0.02 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.

```
+-----+-----+
| ename | sal | dname |
+-----+-----+
JONES	2975.00	RESEARCH
SCOTT	3000.00	RESEARCH
KING	5000.00	ACCOUNTING
FORD	3000.00	RESEARCH
+-----+-----+
4 rows in set (0.02 sec)
```

## 26、列出在每个部门工作的员工数量,平均工资和平均服务期限。

| ACCOUNTING | 3 | 2916.67 | 33 |
|------------|---|---------|----|
| RESEARCH   | 5 | 2175.00 | 31 |
| SALES      | 6 | 1566.67 | 33 |

## 27、列出所有员工的姓名、部门名称和工资。

| ename  | dname      | sal     |
|--------|------------|---------|
| CLARK  | ACCOUNTING | 2450.00 |
| KING   | ACCOUNTING | 5000.00 |
| MILLER | ACCOUNTING | 1300.00 |
| SMITH  | RESEARCH   | 800.00  |
| JONES  | RESEARCH   | 2975.00 |
| SCOTT  | RESEARCH   | 3000.00 |
| ADAMS  | RESEARCH   | 1100.00 |
| FORD   | RESEARCH   | 3000.00 |
| ALLEN  | SALES      | 1600.00 |
| WARD   | SALES      | 1250.00 |
| MARTIN | SALES      | 1250.00 |
| BLAKE  | SALES      | 2850.00 |
| TURNER | SALES      | 1500.00 |
| JAMES  | SALES      | 950.00  |

14 rows in set (0.00 sec)

## 28、列出所有部门的详细信息和人数

| ACCOUNTING | 3 | 2916.67 | 33 |
|------------|---|---------|----|
| RESEARCH   | 5 | 2175.00 | 31 |
| SALES      | 6 | 1566.67 | 33 |

```
+-----+-----+-----+-----+
| DEPTNO | DNAME | LOC | 人数 |
+-----+-----+-----+-----+
10	ACCOUNTING	NEW YORK	3
20	RESEARCH	DALLAS	5
30	SALES	CHICAGO	6
40	OPERATIONS	BOSTON	0
+-----+-----+-----+-----+
4 rows in set <0.05 sec>
```

## 29、列出各种工作的最低工资及从事此工作的雇员姓名

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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

```
+-----+-----+
| deptno | min<sal> |
+-----+-----+
10	2450.00
20	2975.00
30	2850.00
+-----+-----+
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

```
+-----+-----+
| ename | income |
+-----+-----+
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
FORD	36000.00
SCOTT	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

```
+-----+-----+
| ename | ename |
+-----+-----+
JONES	KING
BLAKE	KING
CLARK	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.

| 部门         | 工资合计  | 人数 |
|------------|-------|----|
| OPERATIONS | 0     | 0  |
| RESEARCH   | 10875 | 5  |
| SALES      | 9400  | 6  |

### 34、给任职日期超过 30 年的员工加薪 10%.

```
+-----+
| |
+-----+
```

动力节点 动力节点 动力节点  
动力节点 动力节点 动力节点  
动力节点 动力节点 动力节点