### 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)

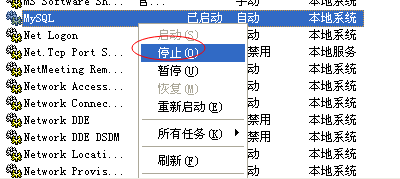
### 1.2、什么是数据库

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 。

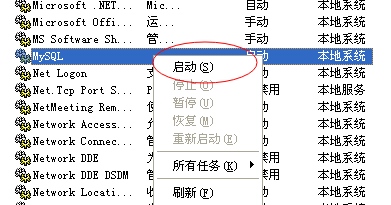
可以通过服务管理器管理 MYSQL 的服务。

通过命令调用服务管理器:services.msc

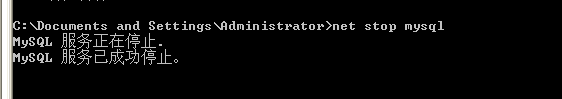
停止 MYSQL 的服务。



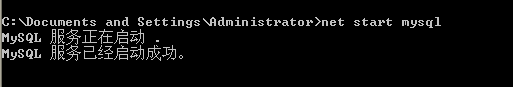
启动 MYSQL 的服务。



停止 MYSQL 的服务。



启动 MYSQL 的服务。



### 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.

### 2.2、创建数据库

1. create database 数据库名称;

create database bjpowernode;

1. 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



### 3.5、查看其他库中的表

show tables from <database name>;

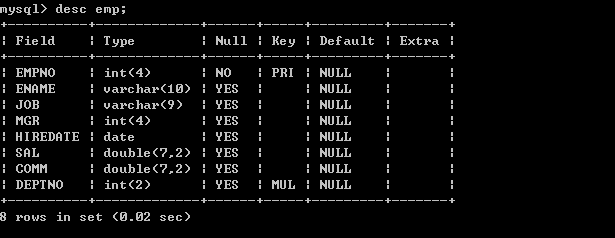
如查看exam库中的表

Show tables from exam;

### 3.6、查看表的结构

desc <table name>;

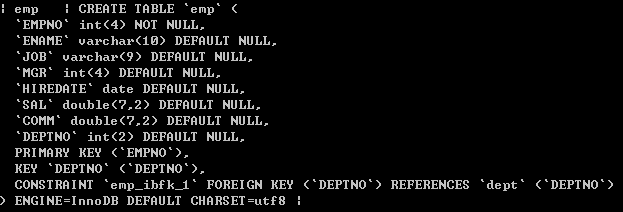
如：



### 3.7、查看表的创建语句

show create table <table name>;

如：

## 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不在这个范围中） |
| 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 |

注意：关于between … and …，它是包含最大值和最小值的

* 查询津贴为空的员工

|  |
| --- |
| select \* from emp where comm=null; |

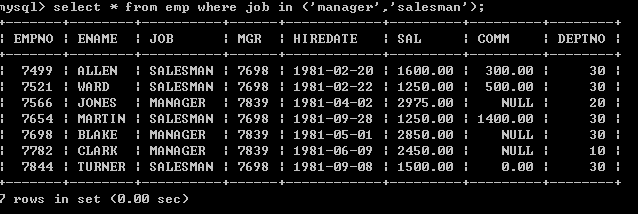


以上也无法查询出符合条件的数据，因为null类型比较特殊，必须使用 is来比较

|  |
| --- |
| select \* from emp where comm is null; |

### 5.7、表达式的优先级

### 5.8、in



### 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%'; |

* 查询姓名以N结尾的所有的员工

|  |
| --- |
| select \* from emp where ename like '%N'; |

* 查询姓名中包含O的所有的员工

|  |
| --- |
| select \* from emp where ename like '%O%'; |

* 查询姓名中第二个字符为A的所有员工

|  |
| --- |
| select \* from emp where ename like '\_A%'; |

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% |

### 6.1、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; |

* 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; |

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; |

### 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; |

* 手动指定按照薪水由大到小排序

|  |
| --- |
| select \* from emp order by sal desc; |

### 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; |

## 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

### 7.3、avg

取得某一列的平均值

* 取得平均薪水

|  |
| --- |
| select avg(sal) from emp; |

### 7.4、max

取得某个一列的最大值

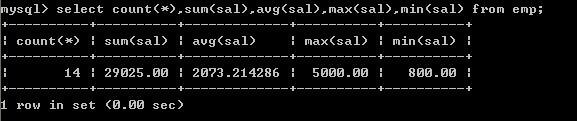
* 取得最晚入职得员工

|  |
| --- |
| select max(str\_to\_date (hiredate, '%Y-%m-%d')) from emp; |

### 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; |



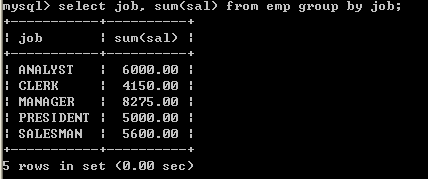
## 8、分组查询

分组查询主要涉及到两个子句，分别是：group by和having

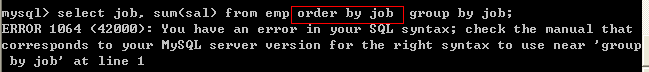
### 8.1、group by

* 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; |

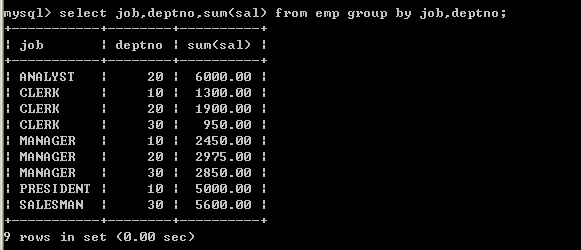


如果使用了order by，order by必须放到group by后面



* + 分组语句

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



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**。

### 8.3、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

select ename, dname from emp, dept;

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; |

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）

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代码了员工表，emp m代表了领导表，相当于员工表和部门表一样

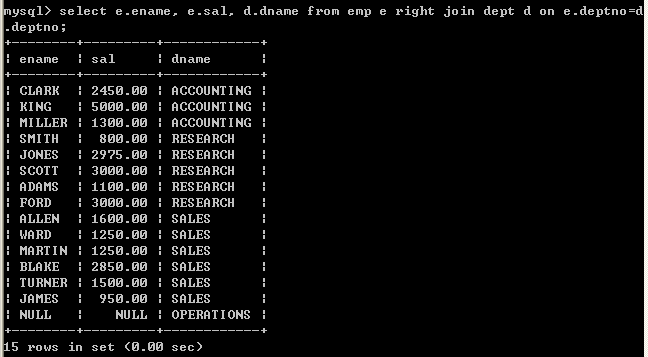
### 9.2、SQL99语法

|  |
| --- |
| 采用SQL92语法：  se（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lect 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 |



连接分类：

内链接

\* 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，

## 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); |

* 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; |

1. 取得大于平均薪水的员工

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

### 10.2、在from语句中使用子查询，可以将该子查询看做一张表

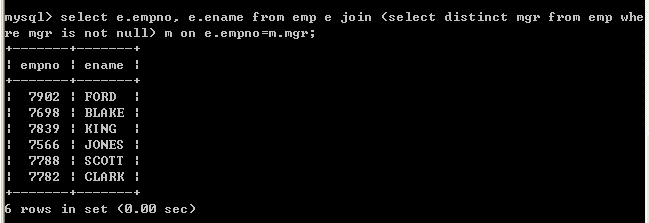
* 查询员工信息，查询哪些人是管理者，要求显示出其员工编号和员工姓名

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; |



* 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; |



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; |

### 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

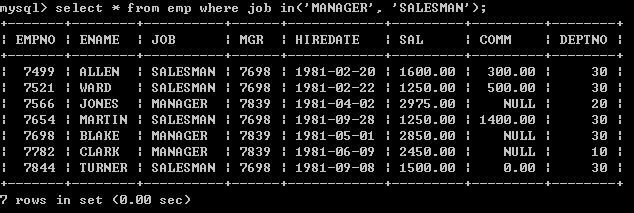
|  |
| --- |
| select e.ename, (select d.dnam from dept d where e.deptno=d.deptno) as dname from emp e; |

## 11、union

### 11.1、union可以合并集合（相加）

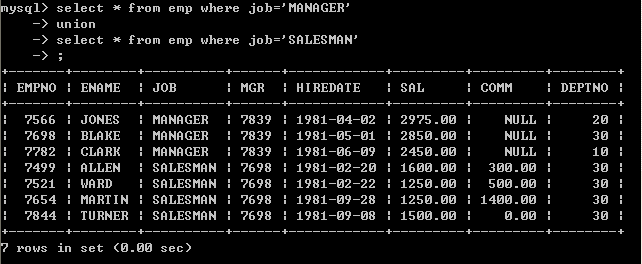
1、查询job包含MANAGER和包含SALESMAN的员工

|  |
| --- |
| select \* from emp where job in('MANAGER', 'SALESMAN'); |



2、采用union来合并

|  |
| --- |
| select \* from emp where job='MANAGER'  union  select \* from emp where job='SALESMAN' |



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; |

## 13、表

### 13.1、创建表

* 语法格式

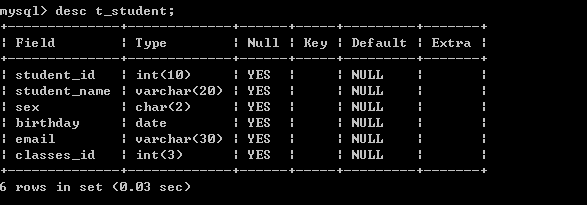
|  |
| --- |
| create table tableName(  columnName dataType(length),  ………………..  columnName dataType(length)  );  set character\_set\_results='gbk';  show variables like '%char%';  创建表的时候，表中有字段，每一个字段有：  \* 字段名  \* 字段数据类型  \* 字段长度限制  \* 字段约束 |

* 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………………… |  |

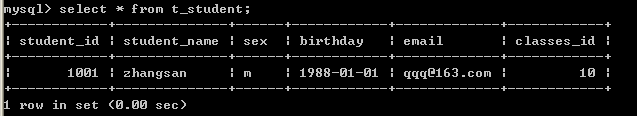
* 建立学生信息表，字段包括：学号、姓名、性别、出生日期、email、班级标识

|  |
| --- |
| create table t\_student(  student\_id int(10),  student\_name varchar(20),  sex char(2),  birthday date,  email varchar(30),  classes\_id int(3)  ) |



* 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) |



* 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) |

### 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：contatct\_tel 类型为varchar(40)

|  |
| --- |
| alter table t\_student add contact\_tel varchar(40); |

#### 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) ; |

### 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,………..) |

如何插入日期：

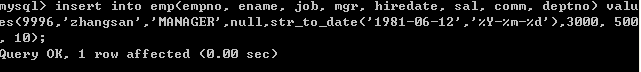
第一种方法，插入的日期格式和显示的日期格式一致

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



第二种方法，采用str\_to\_date

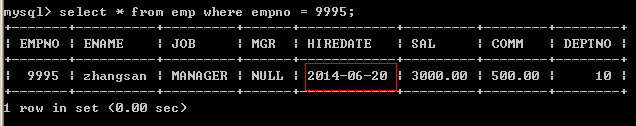
|  |
| --- |
| insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno) valu  es(9996,'zhangsan','MANAGER',null,str\_to\_date('1981-06-12','%Y-%m-%d'),3000, 500, 10); |



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); |





* 表复制

|  |
| --- |
| create table emp\_bak as select empno,ename,sal from emp; |

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; |



#### 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
  1. 非空约束，not null
  2. 唯一约束，unique
  3. 主键约束，primary key
  4. 外键约束，foreign key
  5. 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) |

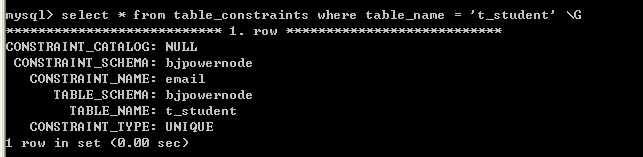
以上错误为加入的学生姓名为空。

#### 13.4.2、唯一约束，unique

* 我们可以查看一下约束

mysql> use information\_schema;

mysql> select \* from table\_constraints where table\_name = 't\_student';



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

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

我们也可以通过表级约束为约束起个名称：

CONSTRAINT p\_id PRIMARY key (student\_id)

#### 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

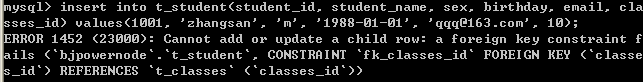
|  |
| --- |
| drop table if exists t\_classes;  create table t\_classes(  classes\_id int(3),  classes\_name varchar(40),  constraint pk\_classes\_id primary key(classes\_id)  ) |

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) |



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



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



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';    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;    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;  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;

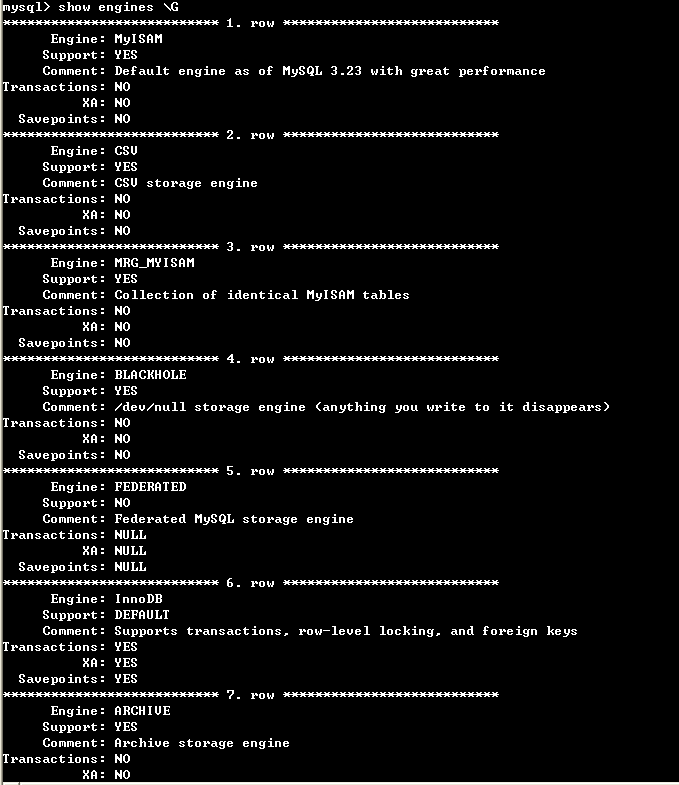
|  |
| --- |
| mysql对有些约束的修改时不支持的，所以我们可以先删除，再添加  alter table t\_student drop foreign key fk\_classes\_id;  alter table t\_student add constraint fk\_classes\_id\_1 foreign key(classes\_id) references t\_classes(classes\_id) on delete cascade;  delete from t\_classes where classes\_id = 20;    我们只删除了父表中的数据，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

### 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



* 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

张三账户

金额：10000

李四账户

金额：

张三转账给李四，必须保证张三账户减少8000，李四账户增加8000，这两项操作必须全部成功，如何保证？需要事务

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

* 1. 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）。
  2. 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。
  3. 隔离性(Isolation)
     + 一个事务不会影响其他事务的运行。
  4. 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：

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

当执行DML语句是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 |
|  |

1. 查询表中数据

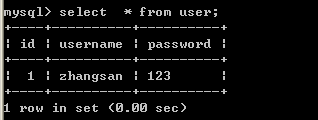


1. 开启事务START TRANSACTION;
2. 插入数据

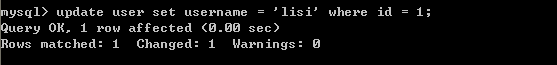
insert into user (username,password) values ('zhangsan','123');



1. 查看数据



1. 修改数据



1. 查看数据



1. 回滚事务



1. 查看数据



### 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（Phantom Read）

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 UMCOMMITTED）

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。

* + 可重复读（REPEATABLE READ）

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】

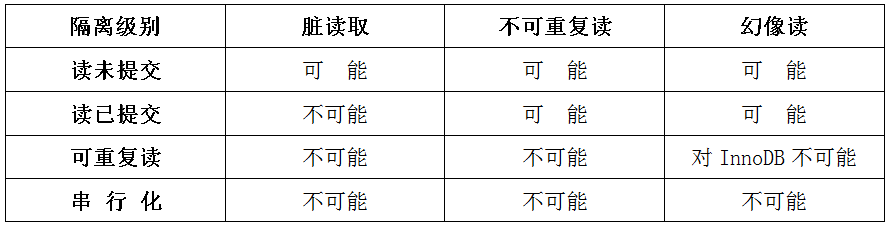
将一个事务与其他事务完全地隔离。

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



#### 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;

## 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



**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);

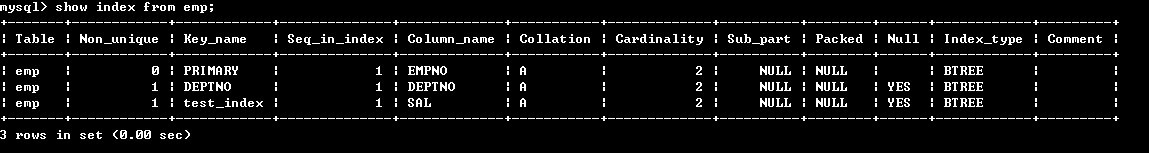
create unique index u\_ename on emp(ename);  
2、alter table 表名 add unique index 索引名 (列名);

|  |
| --- |
| create index test\_index on emp (sal); |



##### 16.2.2、查看索引

|  |
| --- |
| show index from emp; |

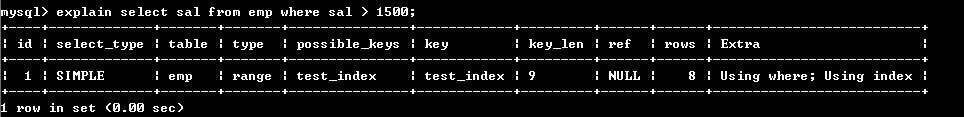


##### 16.2.3、使用索引

注意一定不可以用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



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 talbe\_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。 |

## 17、视图

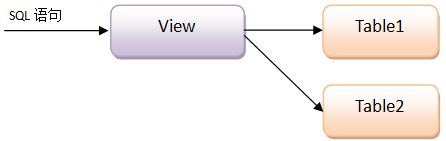
### 17.1、什么是视图

* 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.losal 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.tbname to 'username'@'login ip' identified by 'password' with grant option;   1. dbname=\*表示所有[数据库](http://www.2cto.com/database/" \t "_blank) 2. tbname=\*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

## 19.4、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。

## 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; |