数据库查询

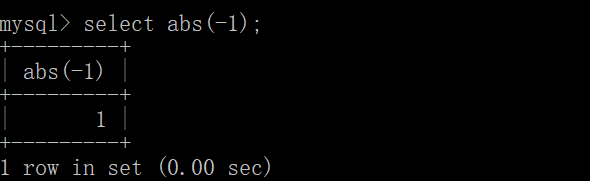
## 目标

## 内置函数

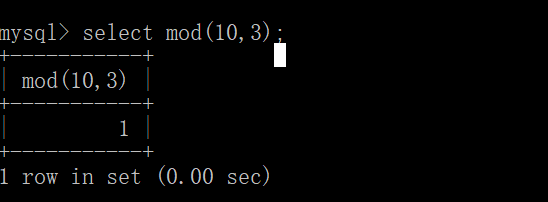
常用五大类：数字函数、字符串函数、日期时间函数、类型转换函数、流程控制

### 1.数字函数

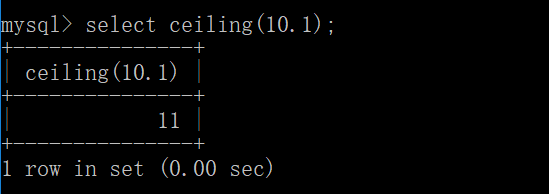
select abs(n);求绝对值



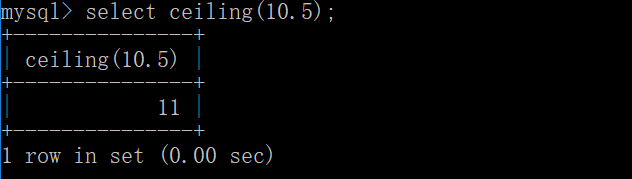
select mod(m,n); 取余



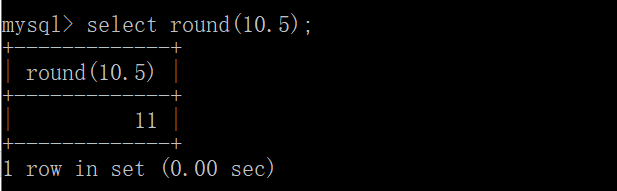
select floor(n); 向下取整



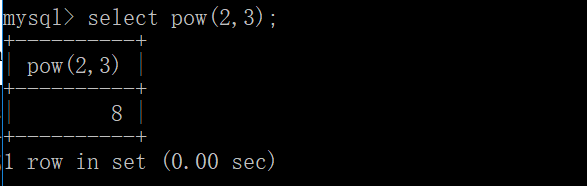
select ceiling(n);向上取整



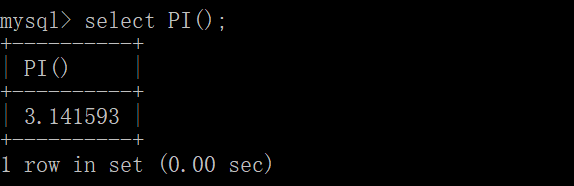
select round(n);四舍五入



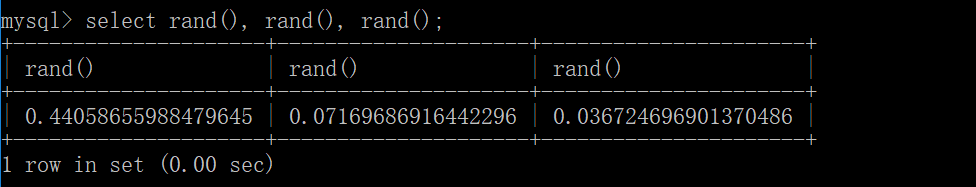
select pow(x,y);x的y次方



select PI(); 获取圆周率

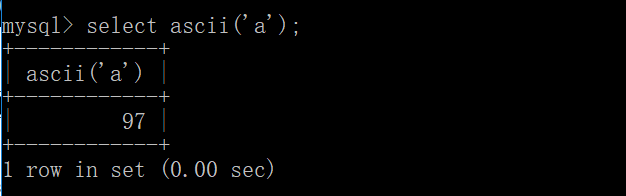


select rand(n); 随机数0-1.0 n是用来存产生的随机数

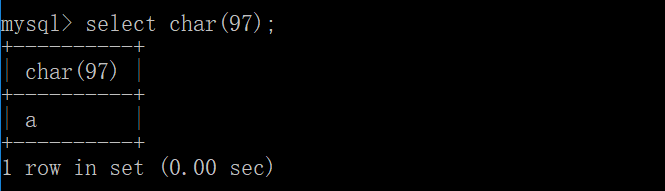


### 2.字符串函数

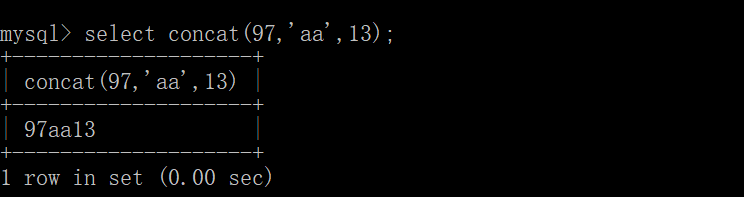
select ascii(str); 字符转ascii码



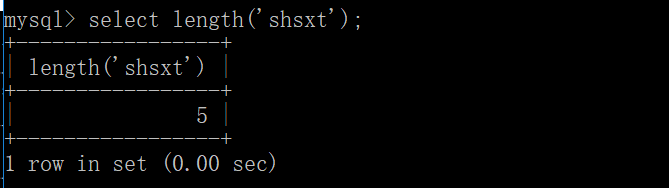
select char(n); ascii码转字符



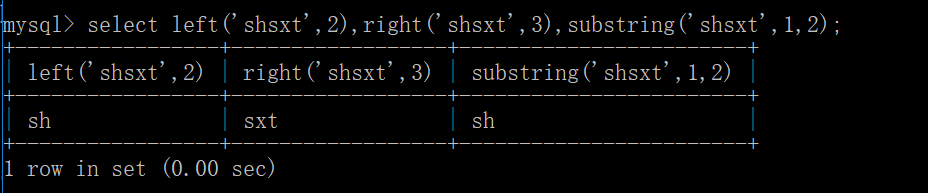
select concat(n,m,p); 会先转成字符串然后拼接



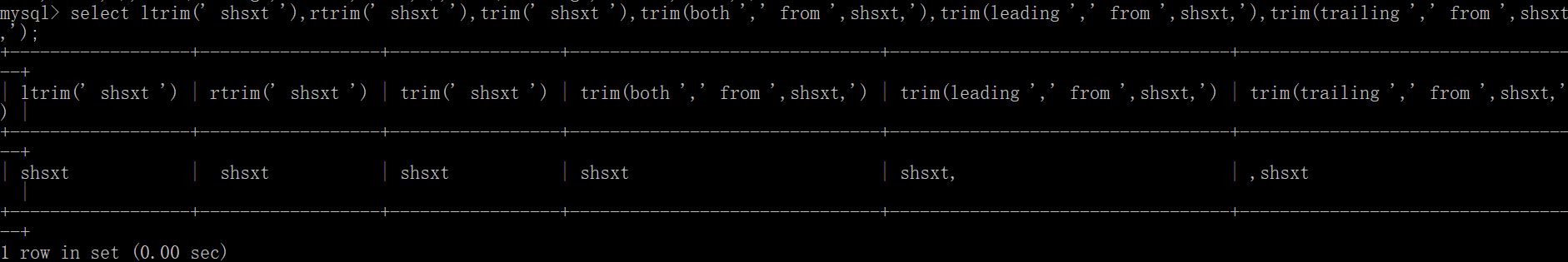
select length(str); 字符串长度



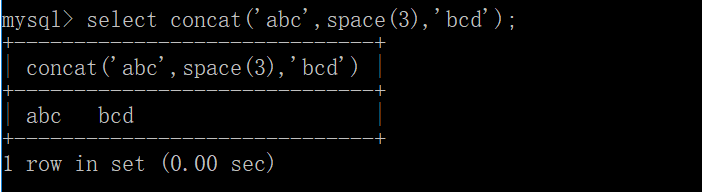
select left(str,len)/right(str,len)/substring(str,pos,len) ;字符串截取，注意：str是从下标1开始截取



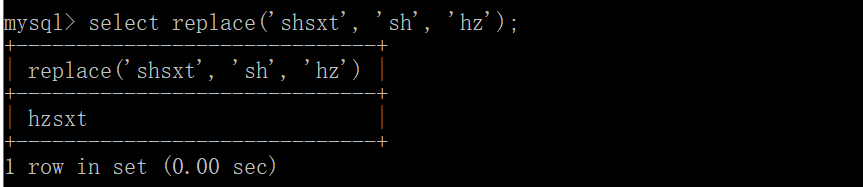
select ltrim()/rtrim()/trim()/trim(both/leading/trailing re\_str from str);两边删除特定的默认空格。



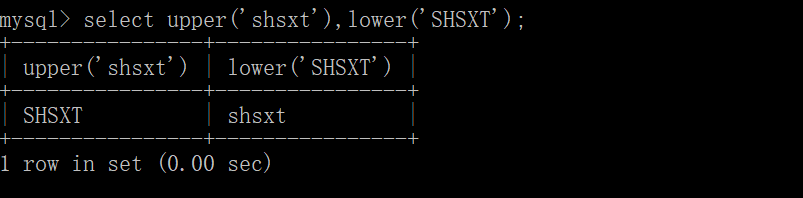
select space(n); 产生n个空格的字符



select replace(str,from\_str,to\_from); 替换

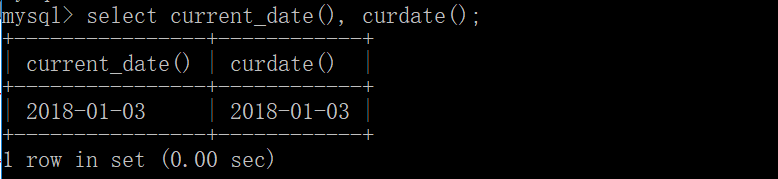


select upper(str)/lower(str);大小写切换

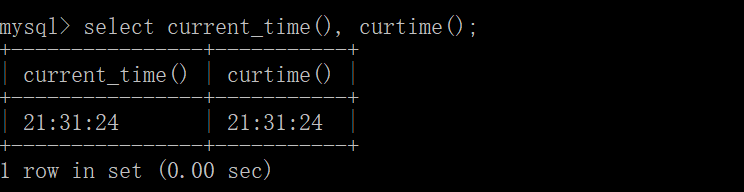


### 3.日期时间函数

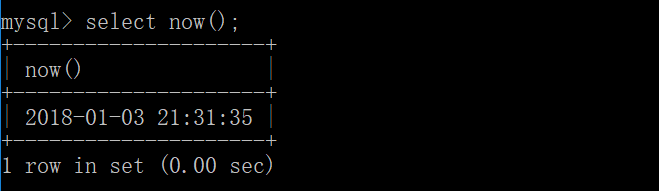
select current\_date()/curdate();获取当前日期



select current\_time()/curtime();获取当前时间

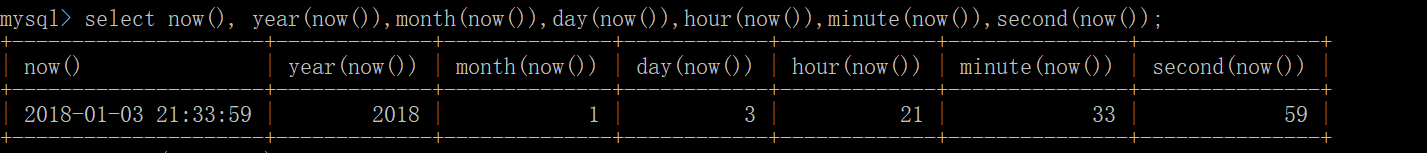


select now(); 获取当前时间日期



获取特定的值

year()/month()/day()/hour()/minute()/second()



date\_format(date,format) 格式化输出

format格式有

%y 年 17

%m 月

%d 天

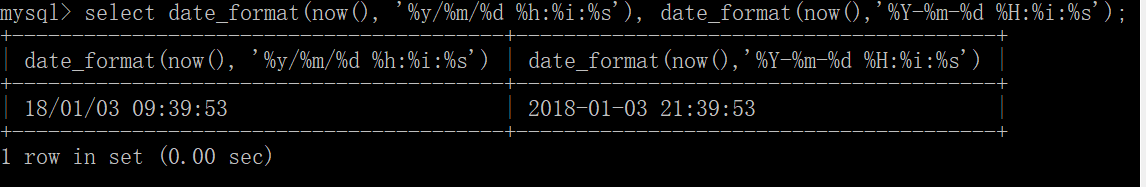
%h 小时 12制

%i 分

%s 秒

%Y 完整年 2017

%H 24制



date\_add(date, interval expr type); 给日期添加指定的时间间隔，

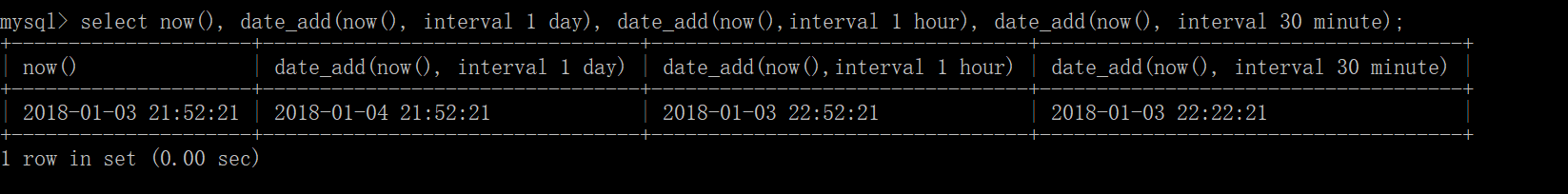
expr 表时间间隔，type参数可以是下列值：

MICROSECOND/SECOND/MINUTE/HOUR/DAY/WEEK/MONTH/QUARTER(季度)/YEAR/

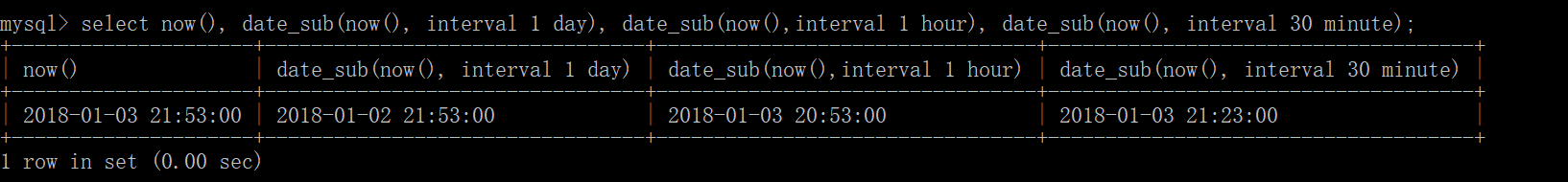
SECOND\_MICROSECOND/MINUTE\_MICROSECOND/MINUTE\_SECOND/

/HOUR\_MICROSECOND/HOUR\_SECOND/HOUR\_MINUTE/DAY\_MICROSECOND

DAY\_SECOND/DAY\_MINUTE/DAY\_HOUR/YEAR\_MONTH

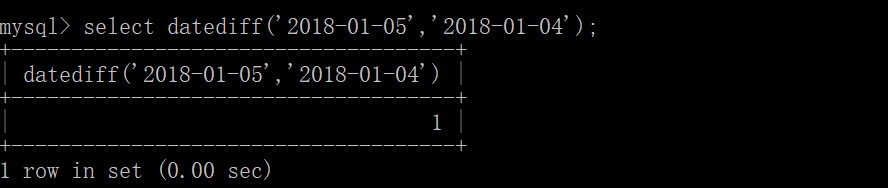


date\_sub(date, interval expr type); 从日期送去指定的时间间隔。expr 参数是您希望添加的时间间隔， type参数同DATE\_ADD()。

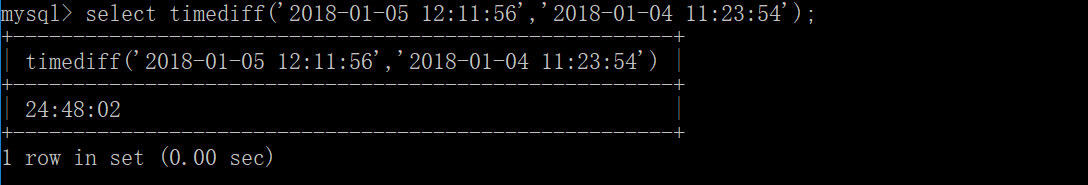


datediff (date1, date2 ); 返回两个日期之间的天数，只有值的日期部分参与计算。

是date1 – date2



timediff(time1,time2)：两个日期相减 time1 time2，返回 time 差值。



### 4.转类型函数（不重要）

cast(value as type)

convert(value,type)

type有 binary 二进制、char 字符、decimal 浮点、date 日期、time 时间、datetime 日期时间、unsigned 无符号整数、signed 有符号整数

### 5 流程控制

case 有两种：一种等值、一种值比较

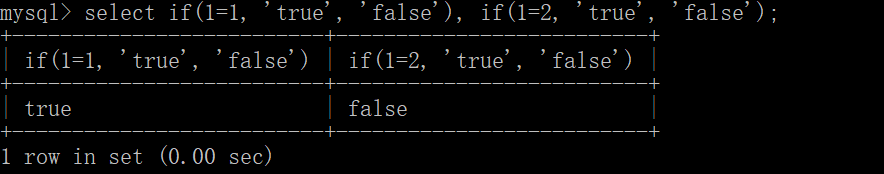
case 值 when 比较值 then 值 .... else 值 end值

|  |
| --- |
| SELECT  empno, ename,  CASE #如果  WHEN sex = 1 THEN  '男' #sex=1，则返回值'男'  WHEN sex = 0 THEN  '女' #sex=0，则返回值'女'  ELSE  '其他' #其他的返回'其他’  END as sex #结束  FROM  emp #整体理解： 在emp表中如果sex1，则返回值'男'如果sex=0，则返回值'女' 否则返回'其他’ |
|  |

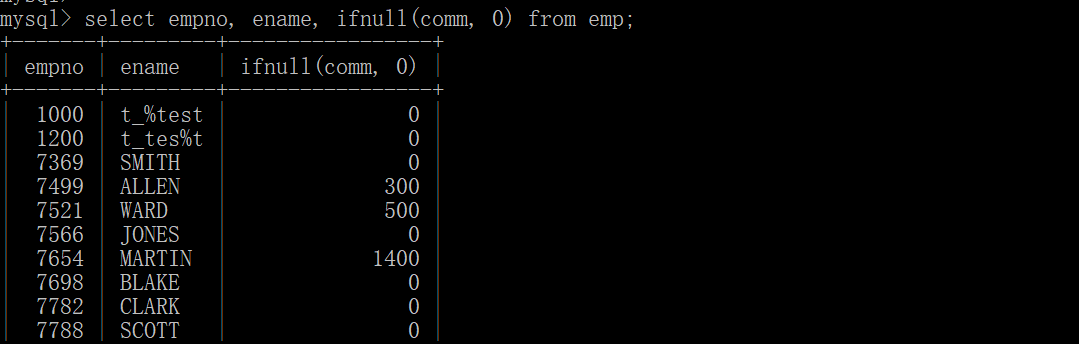
比较利用比较运算符case when 比较过程 then 值 .... else 值 end

|  |
| --- |
| SELECT  empno,ename,  CASE WHEN sal <= 1000 THEN 1  WHEN sal > 1000 AND sal <= 3000 THEN 2  WHEN sal > 3000 AND sal <= 5000 THEN 3  WHEN sal > 5000 THEN 4  ELSE NULL END sal\_class  FROM emp |
|  |

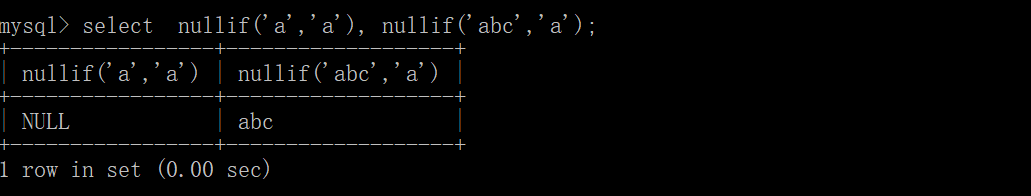
if(func, a, b) 等价于三目运算 func 为真 返回a 为假返回 b



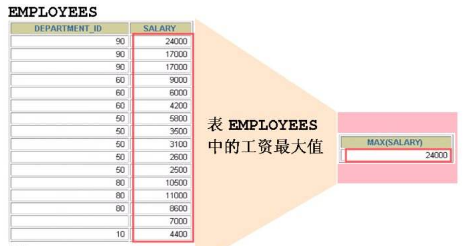
ifnull(a,b) 判断a是否为null 是返回b不是返回a



nullif(a,b) 判断a，b是否相等 是就null 不是就a



### 6、组函数



组函数|多行函数|聚合函数 即多条记录 返回一个结果。我们需要掌握如下几个组函数：avg 、sum、 min、 max、 count

1)、count :统计记录数 count() -->\* 或一个列名

2)、max/min: 最大值/最小值

3)、sum：求和

4)、avg:平均值

注意:

1、组函数仅在选择列表和having子句中有效

2、出现组函数，select 只能有组函数或分组字段

说明：

 a）组信息 与单条记录不能同时查询

 b）组函数 不能用在 where中，能使用的地方 select having

 c）null 不参与运算

1)、count :统计记录数 count() -->\* 或一个列名

|  |
| --- |
| #1、count统计所有的员工数  #1)、\*  #2)、主键  #3)、推荐  select ename,1 from emp;  select count(1) from emp where 1=1;  #2、null不参与运算  #存在佣金的员工数  #不推荐/不需要  select count(comm) from emp where comm is not null;  #推荐。comm自己处理null问题  select count(comm) from emp;  #统计 部门编号30的员工数  select count(1) from emp where deptno=30;  #统计数量过程中 ，可能处理重复  #统计存在员工的部门数量  select count(distinct(deptno)) from emp;  #统计10和20部门一共有多少人  select count(1) from emp where deptno in(10,20); |

2)、max/min: 最大值/最小值

|  |
| --- |
| #查询所有员工的 最高薪水 ，最低薪水，员工总数 #>组信息  select max(sal) maxSal , min(sal) minSal , count(1) from emp;  #查询 最高薪水的员工名称 及薪水  #组信息 与单条记录不能同时查询  select max(sal), ename, sal from emp; #错误  select ename, sal from emp where sal=(select max(sal) from emp ); |

3)、sum：求和

|  |
| --- |
| #查询10部门的所有员工的工资总和  select sum(sal) from emp where deptno=10; |

4)、avg:平均值

|  |
| --- |
| # 查询工资低于平均工资的员工编号，姓名及工资  select empno, ename,sal from emp where sal<(select avg(sal)from emp);  #查看高于本部门平均薪水员工姓名  select \* from emp e1 where sal>(select avg(sal) from emp e2 where e1.deptno=e2.deptno ); |

### 7、分组

#### 分组: group by , 将符合条件的记录 进一步的分组



#### 过滤组：having，过滤组信息 ，表达式同where 一致



#### 现在的结构如下(重点)

|  |
| --- |
| select distinct \* | 字段 | 表达式 | 函数 as 别名  from 表 表别名  where 过滤行记录条件  group by 分组字段列表  having 过滤组  order by 字段列表 asc | desc |

解析步骤

1)、from 2)、where 3)、group 4)、having 5)、select 6)、order by

group by:分组

1)、select 出现分组函数,就不能使用 非分组信息，可以使用group by字段

2)、group by字段可以不出现select中 ，反之select 除组函数外的，其他字段必 须出现在group by 中

过滤组 having :

where :过滤行记录，不能使用组函数， having:过滤组可以使用组函数

#### 例子

|  |
| --- |
| #按 部门 查询 平均工资  select avg(sal),deptno from emp group by deptno;  #按 部门岗位 查询 平均工资  select avg(sal),deptno,job from emp group by deptno,job; |
| #按部门查询平均工资，且平均工资大于2000的部门编号  #1、先分组 后过滤 (不推荐)  select \*  from (select deptno, avg(sal) avsal from emp where 1 = 1 group by deptno) temp  where avsal > 2000;  #2、过滤组 ,分组同时 过滤  select avg(sal), deptno from emp group by deptno having avg(sal)>2000; |
| #查询最低平均工资的部门编号  select deptno,avg(sal) avg from emp group by deptno HAVING  avg=(  select min(asal) from (  select avg(sal) asal from emp group by deptno) t); |

## 多表查询

多表查询是指基于两张表或者是两个视图以及以上的查询操作。在实际开发工作往往单表操作不能满足业务需求。

#### 多表查询的笛卡尔积现象

**笛卡尔积**是指在数学中，两个集合X和Y的笛卡尓积（Cartesian product），又称直积，表示为X×Y，第一个对象是X的成员而第二个对象是Y的所有可能有序对的其中一个成员。

假设集合A={a, b}，集合B={0, 1, 2}，则两个集合的笛卡尔积为{(a, 0), (a, 1), (a, 2), (b, 0), (b, 1), (b, 2)}。

##### 等值连接

在笛卡尔积基础上取条件列相同的值。

**问题1.** 显示雇员名，雇员工资及所在部门的名字

**SQL语句：**SELECT e.ename, e.sal, d.dname FROM emp e, dept d;

结果分析: 查询结果计数 emp的记录数\*dept表的记录数;

原因：产生的笛卡尔积效应，这种查询方式就是**交叉连接**

**解决方法：** 增加两张的限制条件

SELECT e.ename, e.sal, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno;

这种连接方式叫**内连接**

**问题2.** 显示部门号为10的部门名、员工名和工资

**SQL语句：** SELECT d.dname, e.ename, e.sal FROM emp e, dept d WHERE e.deptno = d.deptno and e.deptno = 10;

**问题3：**显示雇员名，雇员工资及所在部门的名字，并按部门排序

select e.ename, e.sal, d.dname ,e.deptno from emp e, dept d where e.deptno = d.deptno order by e.deptno ;

##### 非等值连接(>、<、!=、<>、between and)

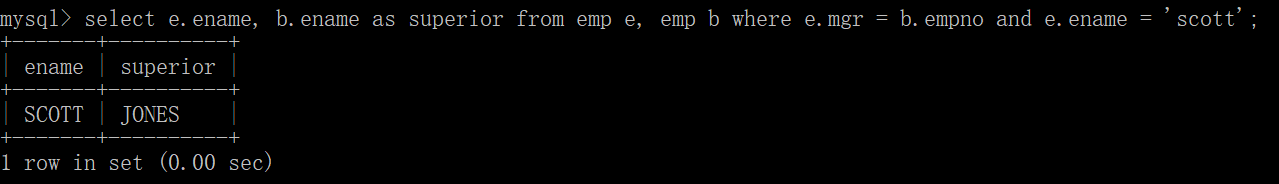
|  |
| --- |
| #查询员工姓名，工资及等级  #900 属于哪个等级  select grade from salgrade where 900 >losal and 900<hisal;  select grade from salgrade where 900 between losal and hisal;  #查询员工姓名，工资及等级  # 数据源：emp e, salgrade s  # 字段：ename, grade, sal  # 条件：sal between losal and hisal  select ename, grade, sal from salgrade s, emp e where sal between losal and hisal; |

#### 自连接

所谓自链接，是指在同一张表操作查询。是一种特殊的等值连接（来源于同一张表）

**问题1：**显示'SCOTT' 的上级领导是谁

select e.ename, b.ename from emp e, emp b where e.mgr = b.empno and e.ename = 'scott';



#### 子查询

子查询是指嵌入在其它sql语句中的select语句,也叫嵌套查询

##### 单行子查询

单行子查询是指只返回一行数据的子查询语句

|  |  |
| --- | --- |
| *-- 如何显示与SMITH同一部门的所有员工?*  */\*\**  *分析*  *1.查询 SMITH 所在的部门编号*  *2.根据编号查询出来 部门所有的人员*  *\*/*  select \* from emp e where e.deptno = (select d.deptno from emp d where d.ename = 'SMITH'); |  |

##### 多行子查询

返回多行子查询的数据

比如：*如何查询和部门10的工作相同的雇员的名字、岗位、工资、部门号*

|  |
| --- |
| */\*\**  *分析 1.查询部门编号为10的员工工作种类 select distinct e.job from emp e where e.deptno = 10;*  *2. 根据工作种类 查询 job相同的员工*  *\*/*  select s.ename, s.job, s.sal, s.deptno  from emp s  where s.job in (select distinct e.job from emp e where e.deptno = 10); |

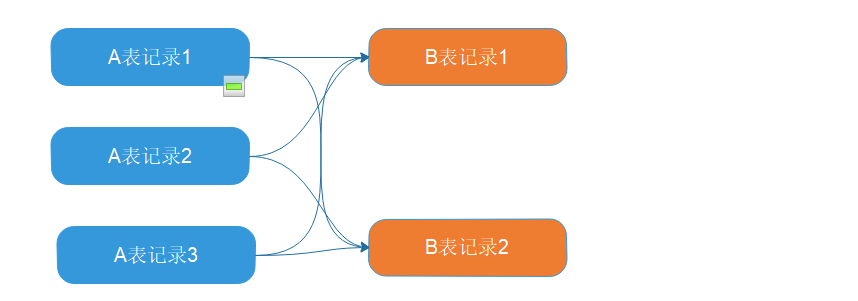
#### 连接查询方式

MySQL中的连接方式除了上面提到的交叉连接、内连接外，还有外链接包括：左连接、右连接以及联合查询。

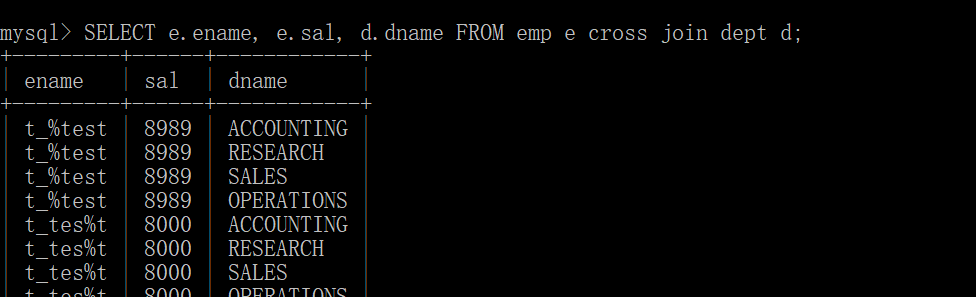
##### 交叉连接 cross join 了解

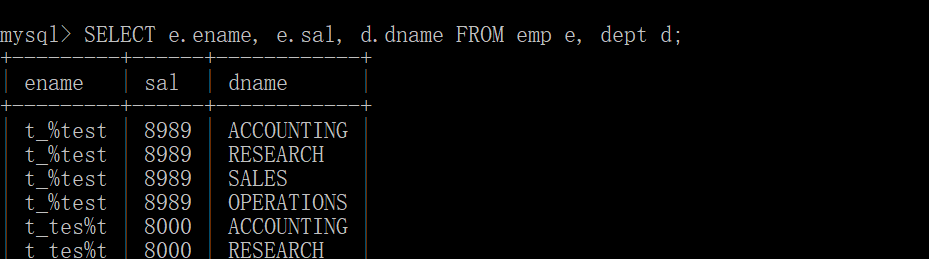
就是刚刚讲到等值连接时，如果不对两张表进行限制时，得到的结果是线性代数中的笛卡尔积，这种情况就是交叉连接。

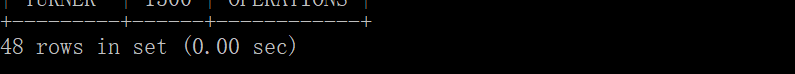
语法：select 字段… from table1 cross join table2;或者是select 字段 from table1, table2;



**Demo**







##### 内连接 inner join 重要

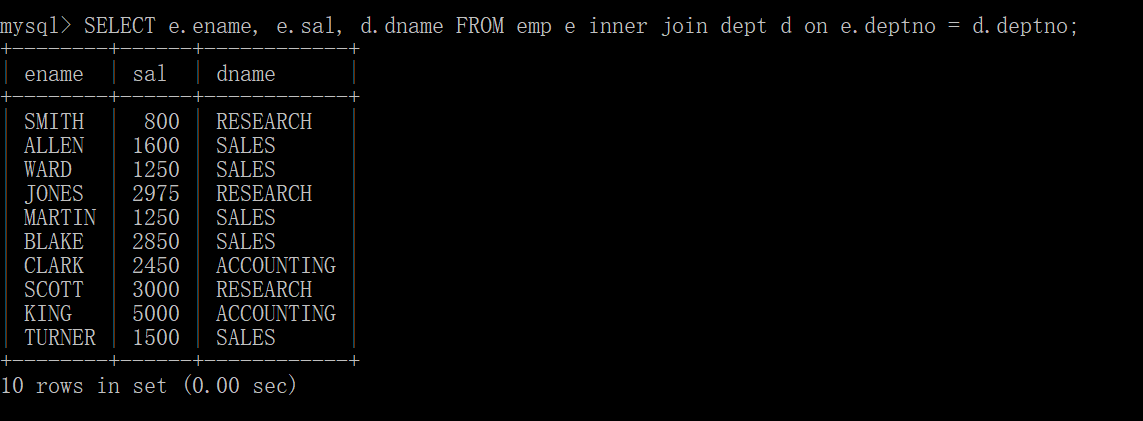
“内连接”理解成"两张表中同时符合某种条件的数据记录的组合"。内连接就包含了等值连接、非等值连接和自连接

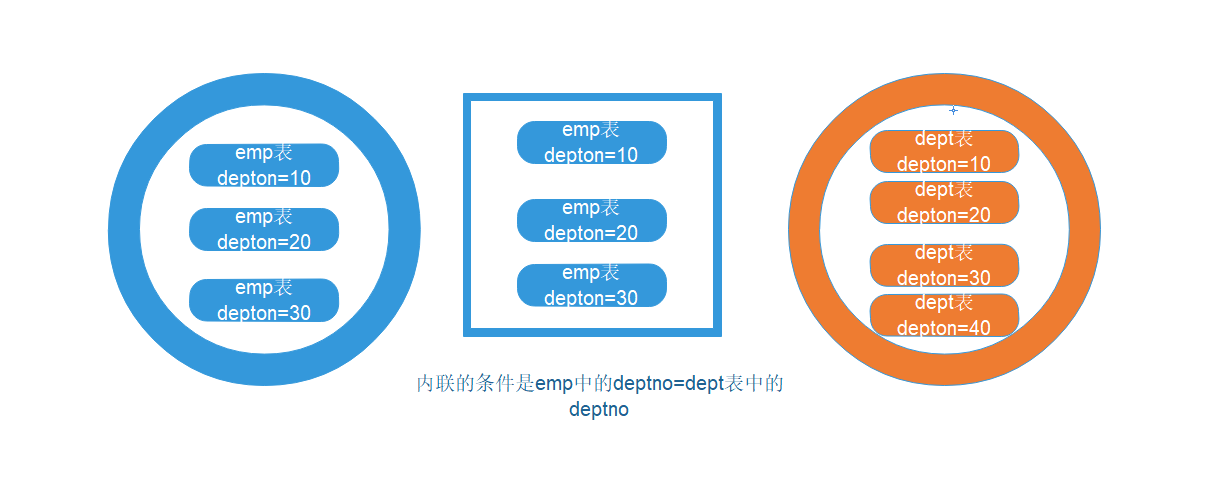
**语法：**

1、select 字段… from table1, table2 where table1.id = table2.table1ID (不推荐)

2、select 字段… from table1 inner join table2 on table1.id = table2.tableID (推荐，同时inner可以省略)

**DEMO:select e.name, e.sal, d.name from emp e inner join dept d on e.deptno = d.deptno;**





##### 外链接 outer join 重要

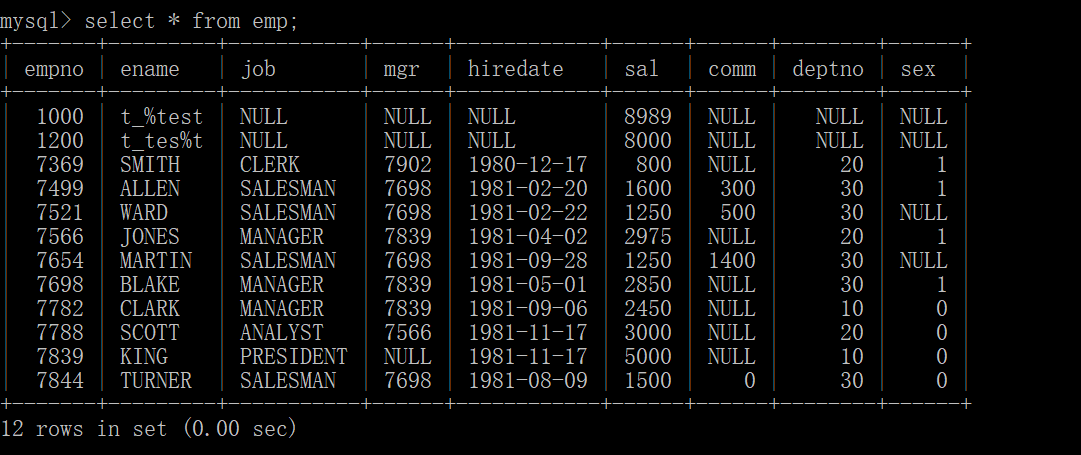
外连接包括两种一种叫左外连接(left outer join)或左连接(left join)，另外一种就是右外连接(right outer join)或右连接(right join)，一般情况下会把外省略。

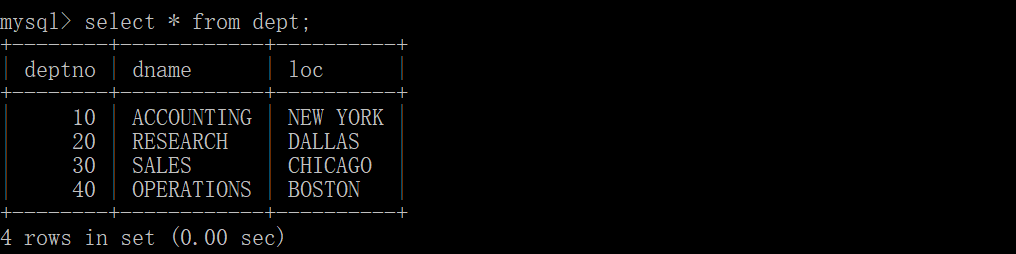
###### 左连接（left outer join或left join）

**语法：**

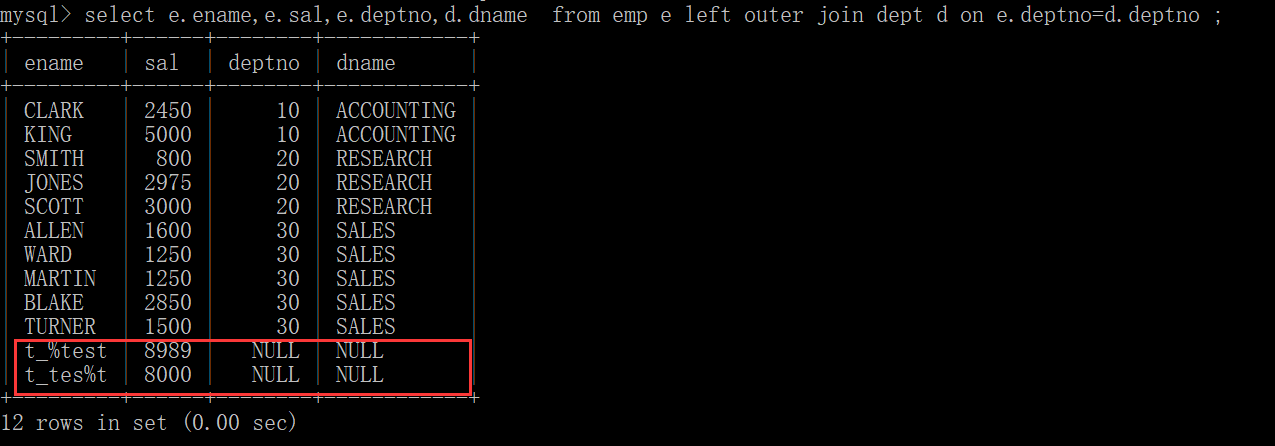
select 字段… from table1 left outer join table2 on table1.id = table2.tableID (推荐，同时outer可以省略)

**DEMO:**

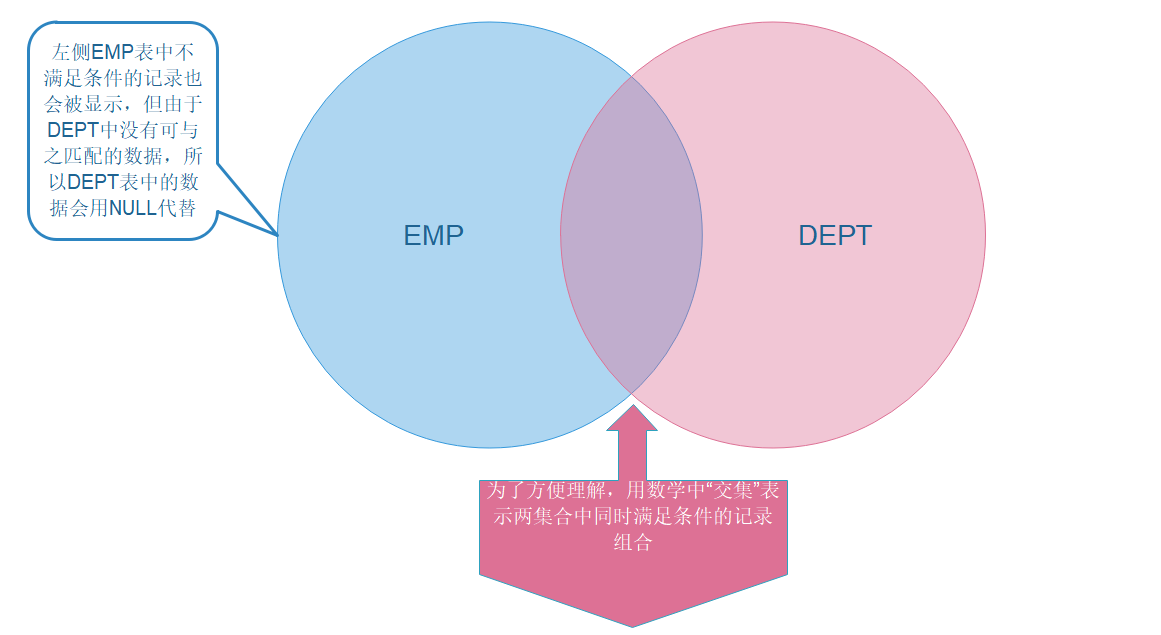




**SQL:select e.ename,e.sal,e.deptno,d.dname from emp e left outer join dept d on e.deptno=d.deptno;**



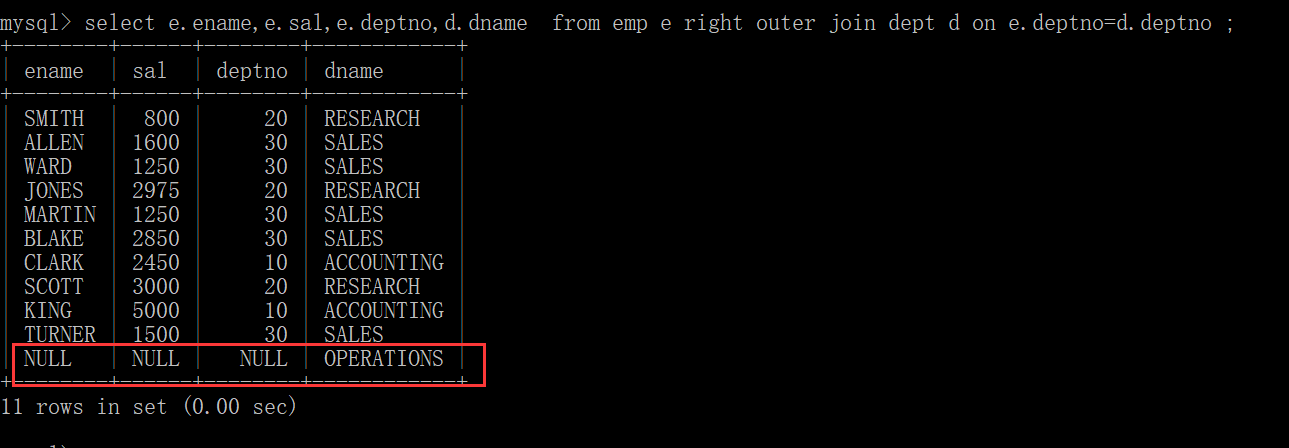
通过结果会发现：总记录数是12条，跟emp表中的记录数一致，而且emp中如果记录没有deptno的值时，这些记录会保留，但是值会为null。



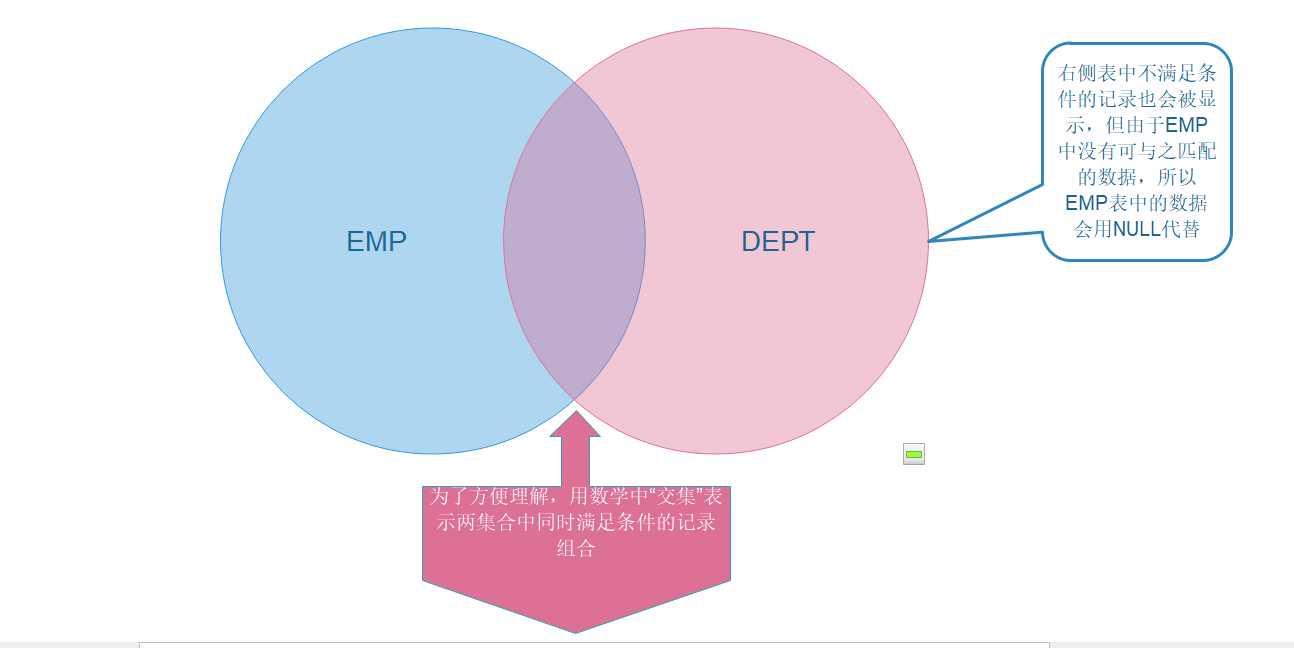
###### 右连接(right outer join 或 right join)

**语法：**select 字段… from table1 right outer join table2 on table1.id = table2.tableID (推荐，同时outer可以省略)

**DEMO：select e.ename,e.sal,e.deptno,d.dname from emp e right outer join dept d on e.deptno=d.deptno;**



通过结果会发现：总记录数是11条，是同时满足记录数的条数（即内联的条数）10条 + DEPT表中的1条。

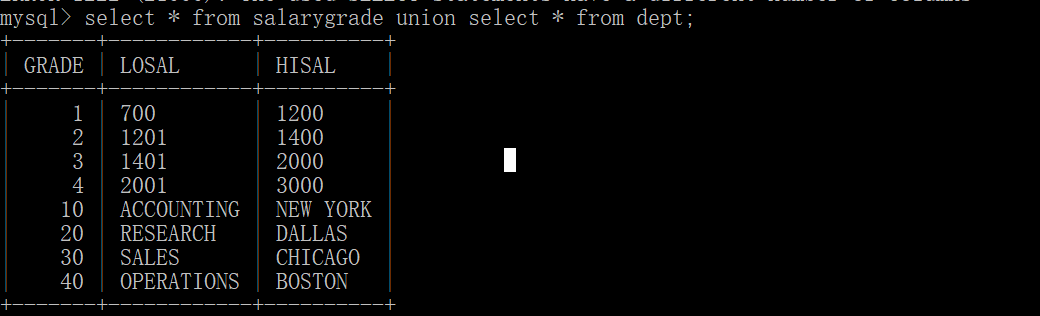


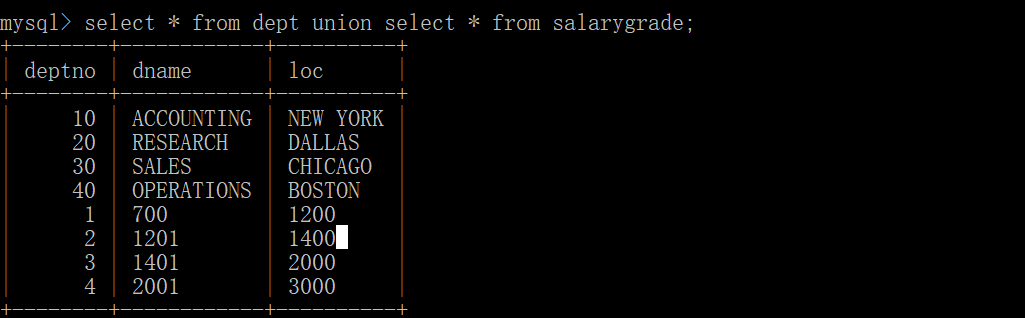
##### 联合查询 了解

**语法：**select 字段… from table1 union select 字段… from table2

而且两个表的记录数必须要一致

**DEMO：select \* from salarygrade union select \* from dept;**





如果两表记录不一致时：



## 分页(重点)

#### 为什么需要分页？

随着表数据量的不断增大，我们客户端数据的展示不可能一次性把所有的数据全部展示出来，即便全部展示，数据库把所有的数据查询出来时也需要很长时间甚至会造成数据库奔溃，因此我们只能一次查询出来多少条即可，这就是分页查询

#### 分页种类

**物理分页：**在数据库执行查询时（实现分页查询），查询需要的数据 -- 依赖数据库SQL 语句，属于后台分页。（一般推荐使用）

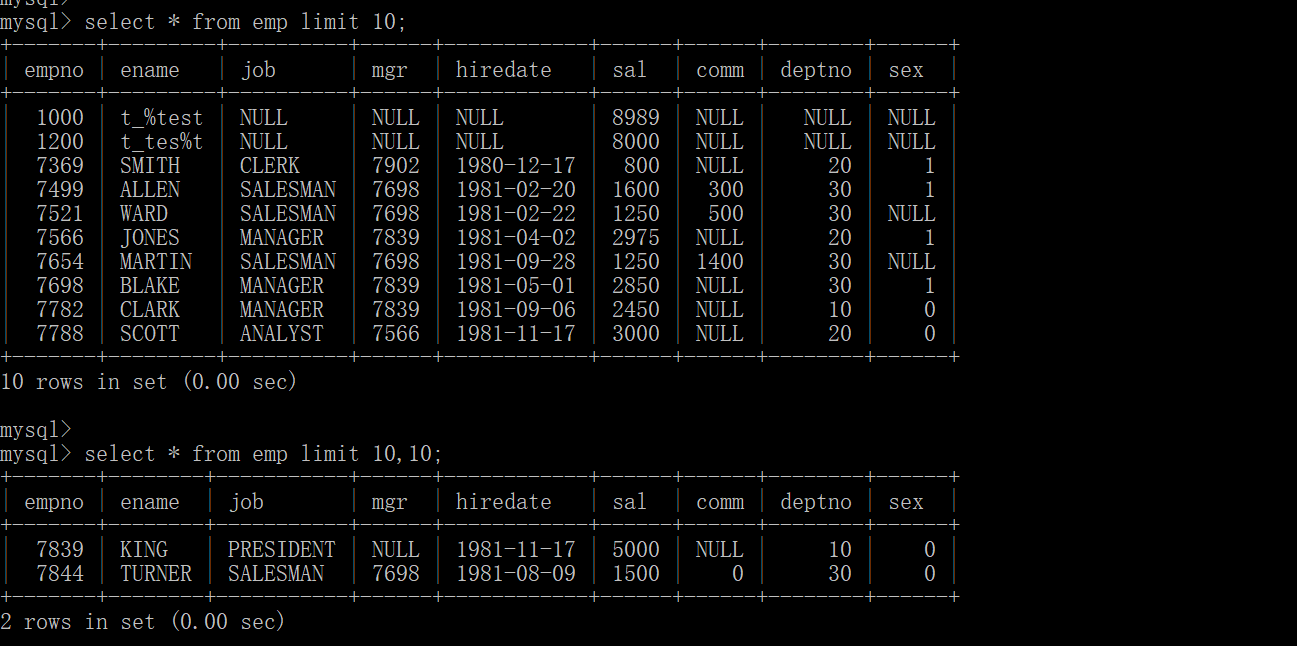
**逻辑分页：**先查询所有数据到内存，再从内存截取需要数据 -- 采用程序内部逻辑，属于前台分页。

#### MySQL分页

MySQL数据库分页查询时，使用关键字limit。

limit语法：select .... limit 开始记录索引,记录条数。

比如：select \* from emp limit 0,10; 就是分页查询从0开始查询10条记录。如果是从0开始还可以把0省略即：select \* from emp limit 10;



## 作业

1. 找出30部门的员工名称及部门名称。

2. 查询员工姓名，工资及等级

3. 列出所有员工的姓名以及其直接上级的姓名

4. 列出所有“CLERK”的姓名及其部门名称，部门的人数

5. 列出最低薪金大于1500的各种工作及此从事此工作的全部雇员人数

6. 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

7. 列出与“scott”从事相同工作的所有员工及部门名称

8. 列出薪金等于部门30中员工的薪金的所有员工的姓名和薪金

9. 列出在每个部门工作的员工数量、平均工资

10. 列出所有员工的姓名、部门名称和工资

11. 查出某个员工的上级主管，并要求出这些主管中的薪水超过3000

12. 求出部门名称中，带'S'字符的部门员工的工资总和 、部门人数。

## 视图

视图:建立在表|结果集|视图上的虚拟表，有以下作用

1、简化:select 查询语句

2、重用:封装select语句 命名

3、隐藏:内部细节

4、区分:相同数据不同查询

不是所有的用户都有创建视图的权限，需要授权。语法如下：

create or replace view 视图名 as select语句;

要求:所有列必须存在名称。

对视图的删除不会删除原有表的数据

drop view 视图名;

求部门经理人中平均薪水最低的部门名称

|  |
| --- |
| #1)、部门经理人 #>mgr  CREATE  OR REPLACE VIEW vw\_emp\_mgr AS SELECT DISTINCT  mgr  FROM  emp  WHERE  mgr IS NOT NULL;  #2)、部门经理人的薪水  CREATE  OR REPLACE VIEW vw\_emp\_mgr\_sal AS SELECT  \*  FROM  emp  WHERE  empno IN (SELECT mgr FROM vw\_emp\_mgr);  #3)、 按部门平均  CREATE  OR REPLACE VIEW vw\_emp\_sal\_group AS SELECT  deptno,  avg(sal) avgsal  FROM  vw\_emp\_mgr\_sal  GROUP BY  deptno;  #4)、过滤组  #找出最低的平均薪水  #1)、 使用第三个视图  SELECT  min(avgsal)  FROM  vw\_emp\_sal\_group;  #部门名称  SELECT  dname  FROM  vw\_emp\_sal\_group vw,  dept d  WHERE  vw.deptno = d.deptno  AND avgsal = (  SELECT  min(avgsal) minsal  FROM  vw\_emp\_sal\_group  ); |

## 索引

 索引是数据库对象之一，用于加快数据的检索，类似于书籍的索引。在数据库中索引可以减少数据库程序查询结果时需要读取的数据量，类似于在书籍中我们利用索引可以不用翻阅整本书即可找到想要的信息。

 1、索引是建立在表上的可选对象；索引的关键在于通过一组排序后的索引键来取代默认的全表扫描检索方式，从而提高检索效率

 2、索引在逻辑上和物理上都与相关的表和数据无关，当创建或者删除一个索引时，不会影响基本的表；

 3、索引一旦建立，在表上进行DML操作时（例如在执行插入、修改或者删除相关操作时），oracle会自动管理索引，索引删除，不会对表产生影响

 4、索引对用户是透明的，无论表上是否有索引，sql语句的用法不变

 5、MySQL创建主键时会自动在该列上创建索引

索引: 提高查询速度的一种手段 -->目录

1、唯一性较好字段适合建立索引

2、大数据量才有效果

3、主键|唯一: 唯一索引

create index 索引名 on表名 (字段列表...)

drop index 索引名 on 表名

create index idx\_emp on emp(sal,ename);

drop index idx\_emp on emp;

select \* from emp order by sal,ename;

## 数据库操纵语言DML（Data Manipulation Language）

数据操纵语言DML主要有三种形式：插入：INSERT、更新：UPDATE、删除：DELETE

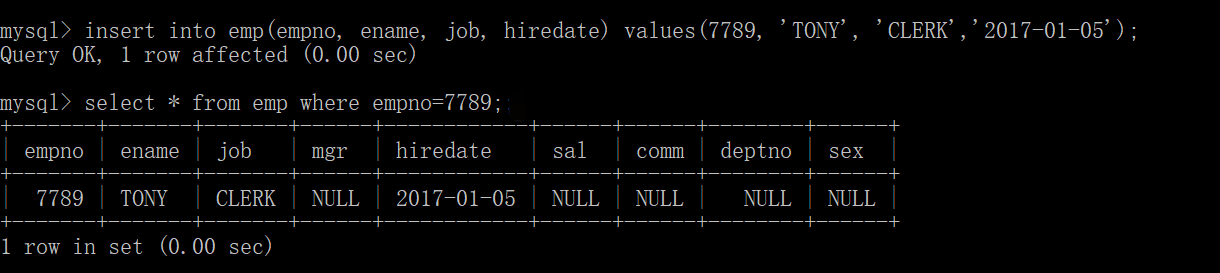
数据库操纵语言主要是对数据库中表的记录进行操作即添加、修改和删除。

### 插入：INSERT

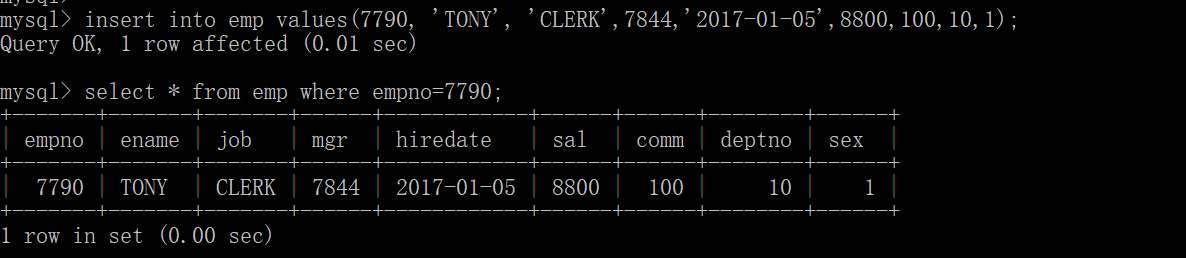
格式：**insert into table [(*column*** **[*, column...*])** **] values(value [, value....]);**

示例操作：

emp表中插入一条数据如：insert into emp(empno, ename, job, hiredate) values(7789, 'TONY', 'CLERK','2017-01-05');

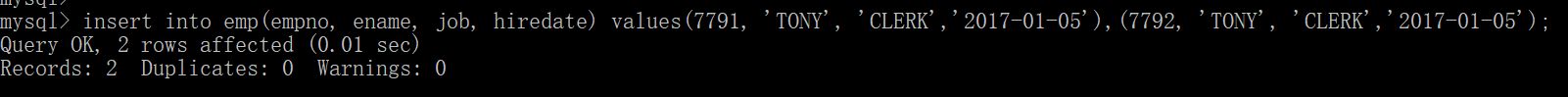


如果emp后面的括号没有的话，那values后的值必须要与字段的表中字段的顺序要一一对应，不推荐这样做。



**批量插入：在执行insert语言时，后面的字段值，可以根据()以逗号分隔进行多个添加。**

**insert into table [(*column*** **[*, column...*])** **] values(value [, value....]), (value [, value....]), (value [, value....]);**



注意事项：

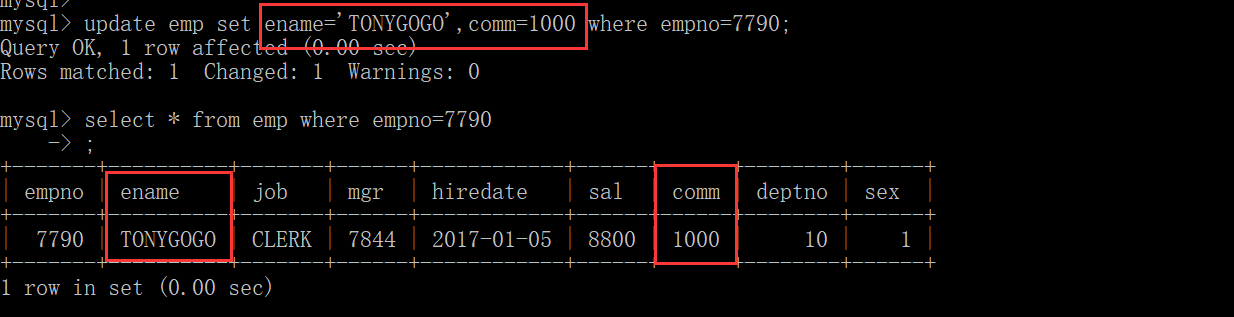
|  |
| --- |
| 1.插入的数据应与字段的数据类型相同。  2.数据的大小应在列的规定范围内，例如：不能将一个长度为80的字符串加入到长度为 40的列中。  3.在values中列出的数据位置必须与被加入的列的排列位置相对应。  4.字符和日期型数据应包含在单引号中，当然日期也可以是日期对象的数据。 |

### 更新：UPDATE

格式：update table set col\_name =exp(..) [where \_definition];

示例：update emp set ename='TONYGOGO',comm=1000 where empno=7790;

注意：如果是更新多个字段，则以逗号进行分隔



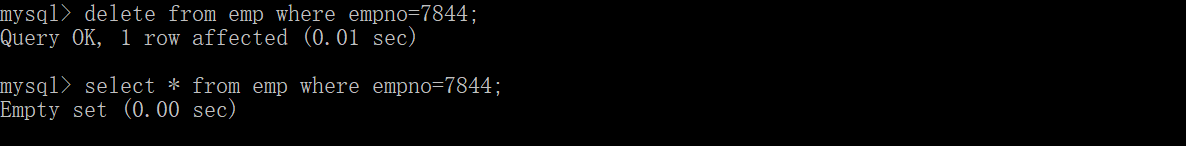
|  |
| --- |
| 1、更新所有记录  update bonus set comm=100;  2、更新带条件的记录  update bonus set sal=sal+sal\*0.15 where sal=800;  3、更新多个字段的记录，使用,隔开  update BONUS set job = null,sal=0 where ename='SCOTT';  4、从已有表更新部分数据（一条）  *--从emp 表中更新SCOTT人员的 姓名、工资、佣金*  update bonus set(ename,sal,comm)=(select ename,sal,comm from emp where ename='SCOTT')  where bonus.ename='SCOTT'; |

### 删除：DELETE

格式：delete from tbname [where\_definition];

示例：按条件删除 delete from emp where empno=7788;

注意：一般在企业中我们不会或者尽力避免直接把数据用DELETE删除，DELETE语句这种删除称为物理删除，防止删错的情况，数据不好恢复问题，而会采用逻辑删除即有一个字段标记此记录的状态比如is\_valid（0代表删除，1代表有效）。读取时只读取is\_valid为1的数据即可。



## 数据库模式定义语言DDL(Data Definition Language)

### 定义

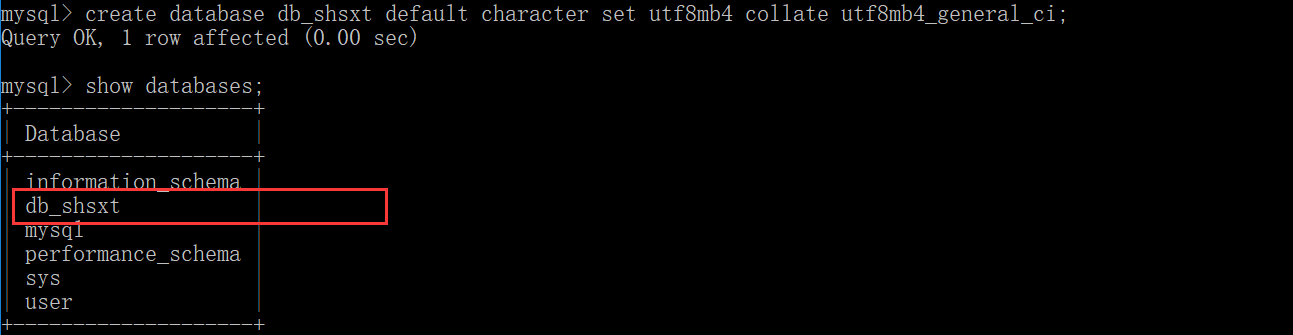
DDL: 是用于描述数据库中要存储的现实世界实体的语言, 这些定义包括结构定义、操作方法定义等。简单理解就是针对数据库、表的创建、删除、修改等操作

### 创建数据库

语法：create database **DB\_NAME** default character set utf8mb4 collate utf8mb4\_general\_ci;

DEMO：create database db\_shsxt default character set utf8mb4 collate utf8mb4\_general\_ci;

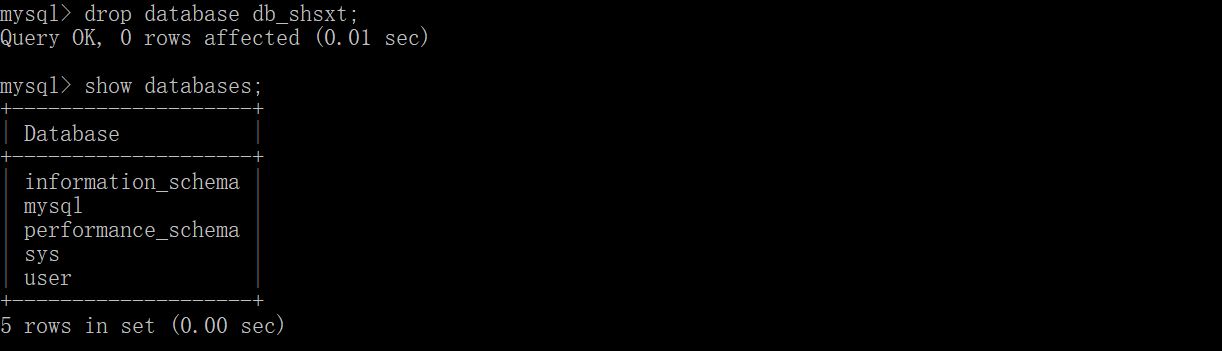
注意：utf8mb4这个字符集是mysql 5.5.3以后才会有，mb4就是most bytes 4的意思，专门用来兼容四字节的unicode。好在utf8mb4是utf8的超集，除了将编码改为utf8mb4外不需要做其他转换。当然，为了节省空间，一般情况下使用utf8也就够了。但是如果是现在的移动互联网，比如果你想要存储Emoji表情的话，就必须是utf8mb4。因为utf8的话最多只能存贮3个字节，而mb4可以存储4个字节，而emoji表情的话会有4个字节的，因此需要utf8mb4。



### 删除数据库

语法：drop database **DB\_NAME**;

DEMO：drop database **db\_shsxt**;



### 创建表

语法：create table “表名” (

字段名 类型 是否为null DEFAULT 默认值,

字段名 类型 是否为null DEFAULT 默认值,

…

PRIMARY KEY(主键)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

例如：创建一个”t\_user”表。

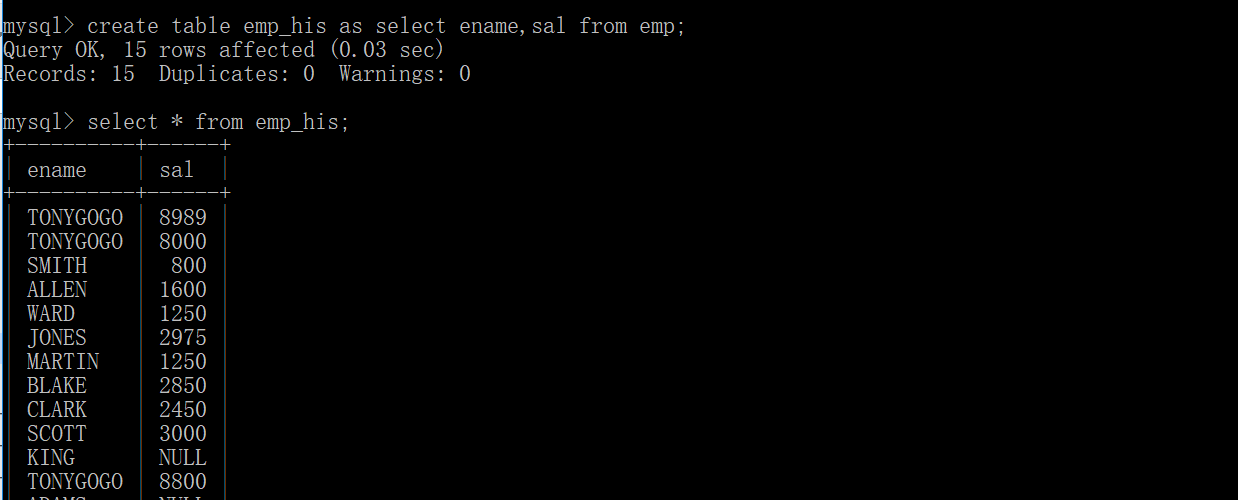
|  |
| --- |
| CREATE TABLE `t\_user` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `user\_name` varchar(9) DEFAULT NULL,  `sex` int(11) DEFAULT NULL,  `create\_time` datetime DEFAULT NULL,  `is\_valid` int(11) DEFAULT 1,  PRIMARY KEY(id)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4; |

### 复制表

create table 表名 as select 字段列表 from 已有表

比如 create table emp\_his as select ename, sale from emp

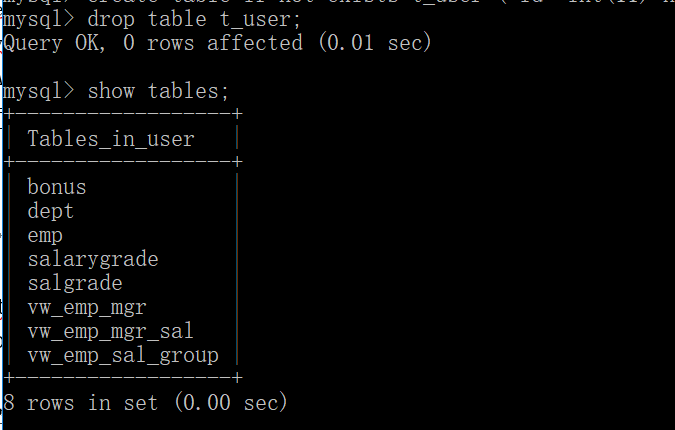
注意如果只需要拷贝表字段，则需要在后面加上where 1 != 1这种恒不等式即可



### 删除表

1、drop table table\_name

例如：drop table t\_user



2、truncate table\_name

删除表中的所有行，但表结构及其列、约束、索引等保持不变，如果主键自增的话清空数据后会从1重新开始。

### 修改表结构

|  |
| --- |
| 1、修改表名 :rename to  2、修改列名: ALTER TABLE 表名 change 原字段名 新字段名 字段类型 约束条件  3、修改类型: alter table 表名 modify 字段名称 字段类型 [完整性约束条件]  4、修改约束: 先删除 后添加  5、添加列: alter table 表名 add 字段 类型  6、删除列:alter table 表名 drop column 字段  #修改表名  rename t\_user to user;    #修改列名  alter table user change id uid int(11) not null auto\_increment;    #修改类型  alter table user modify user\_name varchar(100);    #添加列  alter table user add real\_name varchar(30);    #删除列  alter table user drop column real\_name; |

### 约束

约束是一种限制，它通过对表的行或列的数据做出限制，来确保表的数据的完整性、唯一性。

MYSQL中，常用的几种约束：

| **约束类型：** | **主键** | **默认值** | **唯一** | **外键** | **非空** |
| --- | --- | --- | --- | --- | --- |
| 关键字： | PRIMARY KEY | DEFAULT | UNIQUE | FOREIGN KEY | NOT NULL |

**1、主键(PRIMARY KEY)**是用于约束表中的一行，作为这一行的标识符，在一张表中通过主键就能准确定位到一行，因此主键十分重要。主键要求这一行的数据不能有重复且不能为空。

还有一种特殊的主键——复合主键。主键不仅可以是表中的一列，也可以由表中的两列或多列来共同标识。

**2、默认值约束(DEFAULT)**规定，当有DEFAULT约束的列，插入数据为空时该怎么办。

DEFAULT约束只会在使用INSERT语句（上一实验介绍过）时体现出来，INSERT语句中，如果被DEFAULT约束的位置没有值，那么这个位置将会被DEFAULT的值填充

**3、唯一约束(UNIQUE)**比较简单，它规定一张表中指定的一列的值必须不能有重复值，即这一列每个值都是唯一的。

当INSERT语句新插入的数据和已有数据重复的时候，如果有UNIQUE约束，则INSERT失败.

**4、外键(FOREIGN KEY**)既能确保数据完整性，也能表现表之间的关系。

一个表可以有多个外键，每个外键必须REFERENCES(参考)另一个表的主键，被外键约束的列，取值必须在它参考的列中有对应值。

在INSERT时，如果被外键约束的值没有在参考列中有对应，比如以下命令，参考列(department表的dpt\_name)中没有dpt3，则INSERT失败

**5、非空约束(NOT NULL)**,听名字就能理解，被非空约束的列，在插入值时必须非空。

比如：创建表时可以指定约束。

|  |
| --- |
| CREATE TABLE department  (  dpt\_name CHAR(20) NOT NULL,  people\_num INT(10) DEFAULT '10',  CONSTRAINT dpt\_pk PRIMARY KEY (dpt\_name)  );  CREATE TABLE employee  (  id INT(10) PRIMARY KEY,  name CHAR(20),  age INT(10),  salary INT(10) NOT NULL,  phone INT(12) NOT NULL,  in\_dpt CHAR(20) NOT NULL,  UNIQUE (phone),  CONSTRAINT emp\_fk FOREIGN KEY (in\_dpt) REFERENCES department(dpt\_name)  );    CREATE TABLE project  (  proj\_num INT(10) NOT NULL,  proj\_name CHAR(20) NOT NULL,  start\_date DATE NOT NULL,  end\_date DATE DEFAULT '2015-04-01',  of\_dpt CHAR(20) REFERENCES department(dpt\_name),  CONSTRAINT proj\_pk PRIMARY KEY (proj\_num,proj\_name)  ); |

### 序列

一般使用序列(Sequence)来处理主键字段，在MySQL中是没有序列的，但是MySQL有提供了自增长(increment)来实现类似的目的，而不能设置步长而且一张表只能由一个字段使用自增。

## 事务

### 定义及特性

事物官方解释就是: 数据库管理系统执行过程中的一个逻辑单位，由一个有限的数据库操作序列构成, 通俗理解就是要么不做要么全做完, 也就是说它是一个不可分割的总体. 这就有点类似于我们化学里面所学到的原子, 它是构成物质的基本单位. 因此人们就归纳出事物的第一个特性: **原子性（Atomicity）。**

事物在数据库中占有很重要的地位, 所以它除了原子性还有一个极其重要的特性**一致性（Consistency）**.也就是说，执行完数据库操作后，数据不会被破坏。打个比方，如果从 A 账户转账到 B 账户，不可能因为 A 账户扣了钱，而 B 账户没有加钱吧。如果出现了这类事情，大家一定会非常气愤！

当我们编写了一条 update 语句，提交到数据库的一刹那间，有可能别人也提交了一条 delete 语句到数据库中。也许我们都是对同一条记录进行操作，可以想象，如果不稍加控制，就会出大麻烦来。我们必须保证数据库操作之间是“隔离”的（线程之间有时也要做到隔离），彼此之间没有任何干扰。这就是：**隔离性（Isolation）**。要想真正的做到操作之间完全没有任何干扰是很难的，于是每天上班打酱油的数据库专家们，开始动脑筋了，“我们要制定一个规范，让各个数据库厂商都支持我们的规范！”，这个规范就是：**事务隔离级别（Transaction Isolation Level）**。能定义出这样牛逼的规范真的挺不容易的，其实就四个级别：

**1. READ\_UNCOMMITTED**

**2. READ\_COMMITTED**

**3. REPEATABLE\_READ**

**4. SERIALIZABLE**

**千万不要去翻译，那只是一个代号而已。从上往下，级别越来越高，并发性越来越差，安全性越来越高，反之则反。不同的数据库之间默认隔离级别不一致。**

**当我们执行一条 insert 语句后，数据库必须要保证有一条数据永久地存放在磁盘中，这个也算事务的一条特性， 它就是：持久性（Durability）。**

**归纳一下，以上一共提到了事务的 4 条特性，把它们的英文单词首字母合起来就是：ACID，这个就是传说中的“事务 ACID 特性”！**

**这 4 条特性，是事务管理的基石，一定要透彻理解。此外还要明确，这四个当中，谁才是老大呢？**

**其实：原子性是基础，隔离性是手段，持久性是目的，真正的老大就是一致性。数据不一致了，其他都不用谈了。所以说，这三个小弟都是跟着“一致性”这个老大混，为他全心全意服务。**

**这四个家伙当中，其实最难理解的反倒不是一致性，而是隔离性。因为它是保证一致性的重要手段，是工具，使用它不能有半点差池，否则后果很严重！因此就出来事务隔离级别了。其实，定义这四个级别就是为了解决数据在高并发下所产生的问题，那又有哪些问题呢？**

1. **Dirty Read（脏读）**
2. **Unrepeatable Read（不可重复读）**
3. **Phantom Read（幻读）**

**首先看看“脏读”，看到“脏”这个字，就是恶心、肮脏。数据怎么可能脏呢？其实也就是我们经常说的“垃圾数据”了。比如说，有两个事务，它们在并发执行（也就是竞争）。看看以下这个表格，您一定会明白我在说什么：**

|  |  |  |
| --- | --- | --- |
| 时间 | 事务 A（存款） | 事务 B（取款） |
| T1 | 开始事务 |  |
| T2 |  | 开始事务 |
| T3 |  | 查询余额（1000 元） |
| T4 |  | 取出 1000 元（余额 0 元） |
| T5 | 查询余额（0 元） |  |
| T6 |  | 撤销事务（余额恢复为 1000 元） |
| T7 | 存入 500 元（余额 500 元） |  |
| T8 | 提交事务 |  |

**余额应该为 1500 元才对！请看 T5 时间点，事务 A 此时查询余额为 0 元，这个数据就是脏数据，它是事务 B 造成的，明显事务没有进行隔离，渗过来了，乱套了。**

**那第 2 条，不可重复读又怎么解释呢？还是用类似的例子来说明：**

|  |  |  |
| --- | --- | --- |
| 时间 | 事务 A（存款） | 事务 B（取款） |
| T1 | 开始事务 |  |
| T2 |  | 开始事务 |
| T3 |  | 查询余额（1000 元） |
| T4 | 查询余额（1000 元） |  |
| T5 |  | 取出 1000 元（余额 0 元） |
| T6 |  | 提交事务 |
| T7 | 查询余额（0 元） |  |

**事务 A 其实除了查询了两次以外，其他什么事情都没有做，结果钱就从 1000 变成 0 了，这就是重复读了。可想而知，这是别人干的，不是我干的。其实这样也是合理的，毕竟事务 B 提交了事务，数据库将结果进行了持久化，所以事务 A 再次读取自然就发生了变化。**

**这种现象基本上是可以理解的，但在有些变态的场景下却是不允许的。毕竟这种现象也是事务之间没有隔离所造成的，但我们对于这种问题，似乎可以忽略。**

**最后一条，幻读。Phantom 这个单词不就是“幽灵、鬼魂”吗？其实意义就是鬼在读，不是人在读，或者说搞不清楚为什么，它就变了。用一个示例来说话吧：**

|  |  |  |
| --- | --- | --- |
| 时间 | 事务 A（统计总存款） | 事务 B（存款） |
| T1 | 开始事务 |  |
| T2 |  | 开始事务 |
| T3 | 统计总存款（10000 元） |  |
| T4 |  | 存入 100 元 |
| T5 |  | 提交事务 |
| T6 | 统计总存款（10100 元） |  |

**银行工作人员，每次统计总存款，都看到不一样的结果。不过这也确实也挺正常的，总存款增多了，肯定是这个时候有人在存钱。但是如果银行系统真的这样设计，那算是玩完了。这同样也是事务没有隔离所造成的，但对于大多数应用系统而言，这似乎也是正常的，可以理解，也是允许的。银行里那些恶心的那些系统，要求非常严密，统计的时候，甚至会将所有的其他操作给隔离开，这种隔离级别就算非常高了（估计要到 SERIALIZABLE 级别了）。**

**归纳一下，以上提到了事务并发所引起的跟读取数据有关的问题，各用一句话来描述一下：**

1. **脏读：事务 A 读取了事务 B 未提交的数据，并在这个基础上又做了其他操作。**
2. **不可重复读：事务 A 读取了事务 B 已提交的更改数据。**
3. **幻读：事务 A 读取了事务 B 已提交的新增数据。**

**第一条是坚决抵制的，后两条在大多数情况下可不作考虑。**

**这就是为什么必须要有事务隔离级别这个东西了，它就像一面墙一样，隔离不同的事务。看下面这个表格，您就清楚了不同的事务隔离级别能处理怎样的事务并发问题：**

|  |  |  |  |
| --- | --- | --- | --- |
| 事务隔离级别 | 脏读 | 不可重复读 | 幻读 |
| READ\_UNCOMMITTED | 允许 | 允许 | 允许 |
| READ\_COMMITTED | 禁止 | 允许 | 允许 |
| REPEATABLE\_READ | 禁止 | 禁止 | 允许 |
| SERIALIZABLE | 禁止 | 禁止 | 禁止 |

### MySQL事务

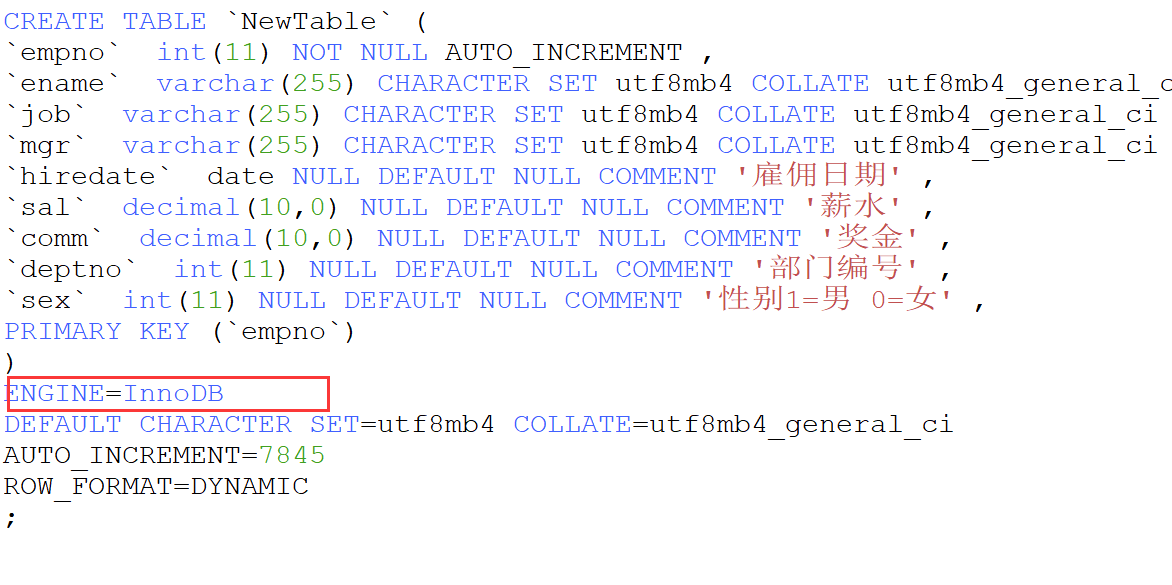
MySQL默认的事务隔离级别为REPEATABLE\_READ。

#### MySQL引擎

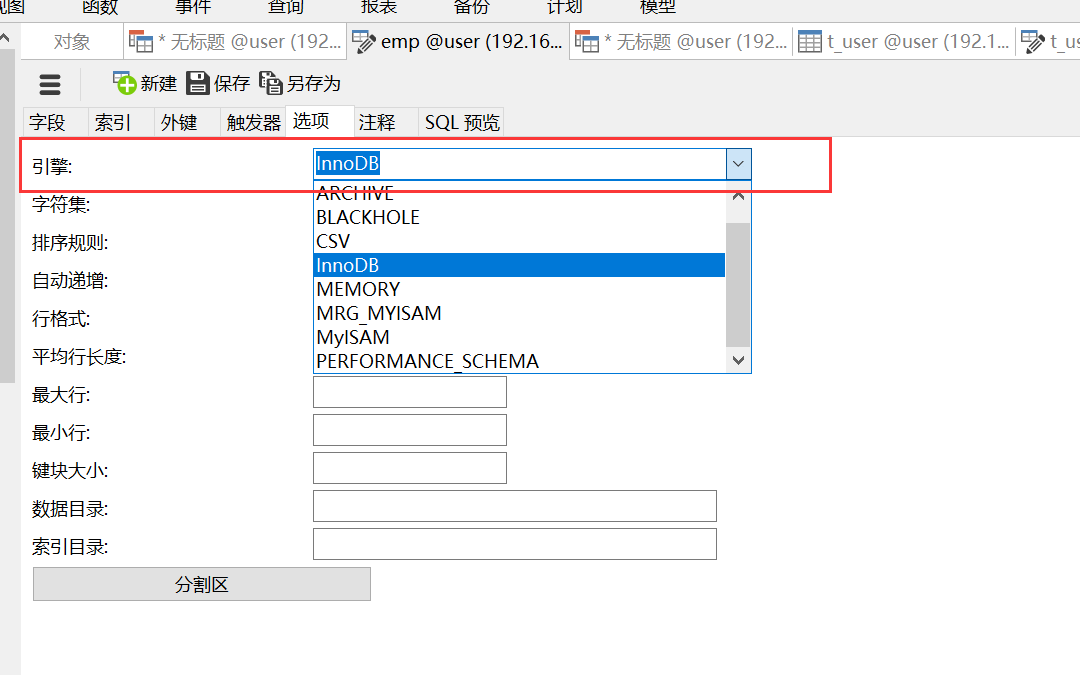
讲MySQL事务之前得先介绍一下MySQL数据库的引擎。数据库中的存储引擎其实是对使用了该引擎的表进行某种设置，数据库中的表设定了什么存储引擎，那么该表在**数据存储方式、数据更新方式、数据查询性能以及是否支持索引等方面就会有不同的“效果”**。在MySQL数据库中存在着多种引擎（不同版本的MySQL数据库支持的引擎不同），熟悉各种引擎才能在软件开发中应用引擎，从而开发出高性能的软件，MySQL数据库中的引擎有哪些呢？一般来说，MySQL有以下几种引擎：ISAM、MyISAM、HEAP（也称为MEMORY）、CSV、BLACKHOLE、ARCHIVE、PERFORMANCE\_SCHEMA、InnoDB、 Berkeley、Merge、Federated和Cluster/NDB等。现在公司用户的比较多的就两种一种是MyISAM另外一种叫InnoDB。

在创建表的时候就可以选择表的引擎。

1、使用命令创建时，在创建语句后面添加：



2、使用Navicat工具时，会在选项中选择。



#### ISAM

该引擎在读取数据方面速度很快，而且不占用大量的内存和存储资源；但是ISAM不支持事务处理、不支持外来键、不能够容错、也不支持索引。该引擎在包括MySQL 5.1及其以上版本的数据库中不再支持。

#### MyISAM

该引擎基于ISAM数据库引擎，除了提供ISAM里所没有的索引和字段管理等大量功能，MyISAM还使用一种表格锁定的机制来优化多个并发的读写操作，但是需要经常运行OPTIMIZE TABLE命令，来恢复被更新机制所浪费的空间，否则碎片也会随之增加，最终影响数据访问性能。MyISAM还有一些有用的扩展，例如用来修复数据库文件的MyISAMChk工具和用来恢复浪费空间的 MyISAMPack工具。M**yISAM强调了快速读取操作，主要用于高负载的select**，这可能也是MySQL深受Web开发的主要原因：在Web开发中进行的大量数据操作都是读取操作，所以大多数虚拟主机提供商和Internet平台提供商（Internet Presence Provider，IPP）只允许使用MyISAM格式。

MyISAM类型的表支持三种不同的存储结构：静态型、动态型、压缩型。

静态型：指定义的表列的大小是固定（即不含有：xblob、xtext、varchar等长度可变的数据类型），这样MySQL就会自动使用静态MyISAM格式。使用静态格式的表的性能比较高，因为在维护和访问以预定格式存储数据时需要的开销很低；但这种高性能是以空间为代价换来的，因为在定义的时候是固定的，所以不管列中的值有多大，都会以最大值为准，占据了整个空间。

动态型：如果列（即使只有一列）定义为动态的（xblob, xtext, varchar等数据类型），这时MyISAM就自动使用动态型，虽然动态型的表占用了比静态型表较少的空间，但带来了性能的降低，因为如果某个字段的内容发生改变则其位置很可能需要移动，这样就会导致碎片的产生，随着数据变化的增多，碎片也随之增加，数据访问性能会随之降低。

对于因碎片增加而降低数据访问性这个问题，有两种解决办法：

a、尽可能使用静态数据类型；

b、经常使用optimize table table\_name语句整理表的碎片，恢复由于表数据的更新和删除导致的空间丢失。如果存储引擎不支持 optimize table table\_name则可以转储并 重新加载数据，这样也可以减少碎片；

压缩型：如果在数据库中创建在整个生命周期内只读的表，则应该使用MyISAM的压缩型表来减少空间的占用。

#### InnoDB

该存储引擎为MySQL表提供了ACID事务支持、系统崩溃修复能力和多版本并发控制（即MVCC Multi-Version Concurrency Control）的行级锁;该引擎支持自增长列（auto\_increment）,自增长列的值不能为空，如果在使用的时候为空则自动从现有值开始增值，如果有但是比现在的还大，则直接保存这个值; 该引擎存储引擎支持外键（foreign key） ,外键所在的表称为子表而所依赖的表称为父表。该引擎在5.5后的MySQL数据库中为默认存储引擎。

#### MyISAM与InnoDB的区别（面试）

**两种类型最主要的差别就是Innodb 支持事务处理与外键和行级锁，MyISAM的查询速度会比InnoDB快。**

#### MySql中的事务

事务的实现是基于数据库的存储引擎。不同的存储引擎对事务的支持程度不一样。mysql中支持事务的存储引擎有innoDB和NDB。innoDB是mysql默认的存储引擎，默认的隔离级别是repeatable-read，并且在repeatable-read的隔离级别下更进一步，通过多版本**并发控制**（MVCC，Multiversion Concurrency Control ）解决不可重复读问题，加上间隙锁（也就是并发控制）解决幻读问题。因此innoDB的repeatable-read隔离级别其实实现了串行化级别的效果，而且保留了比较好的并发性能。

事务的隔离性是通过锁实现，而事务的原子性、一致性和持久性则是通过事务日志实现。说到事务日志，不得不说的就是redo和undo。

#### redo log

在innoDB的存储引擎中，事务日志通过重做(redo)日志和innoDB存储引擎的日志缓冲(InnoDB Log Buffer)实现。事务开启时，事务中的操作，都会先写入存储引擎的日志缓冲中，在事务提交之前，这些缓冲的日志都需要提前刷新到磁盘上持久化，这就是DBA们口中常说的“日志先行”(Write-Ahead Logging)。当事务提交之后，在Buffer Pool中映射的数据文件才会慢慢刷新到磁盘。此时如果数据库崩溃或者宕机，那么当系统重启进行恢复时，就可以根据redo log中记录的日志，把数据库恢复到崩溃前的一个状态。未完成的事务，可以继续提交，也可以选择回滚，这基于恢复的策略而定。

在系统启动的时候，就已经为redo log分配了一块连续的存储空间,以顺序追加的方式记录Redo Log,通过顺序IO来改善性能。所有的事务共享redo log的存储空间，它们的Redo Log按语句的执行顺序，依次交替的记录在一起。如下一个简单示例：

|  |
| --- |
| 记录1：<trx1, insert...>  记录2：<trx2, delete...>  记录3：<trx3, update...>  记录4：<trx1, update...>  记录5：<trx3, insert...> |

#### undo log

undo log主要为事务的回滚服务。在事务执行的过程中，除了记录redo log，还会记录一定量的undo log。undo log记录了数据在每个操作前的状态，如果事务执行过程中需要回滚，就可以根据undo log进行回滚操作。单个事务的回滚，只会回滚当前事务做的操作，并不会影响到其他的事务做的操作。

以下是undo+redo事务的简化过程

假设有2个数值，分别为A和B,值为1，2

|  |
| --- |
| 1. start transaction;  2. 记录 A=1 到undo log;  3. update A = 3；  4. 记录 A=3 到redo log；  5. 记录 B=2 到undo log；  6. update B = 4；  7. 记录B = 4 到redo log；  8. 将redo log刷新到磁盘  9. commit |

在1-8的任意一步系统宕机，事务未提交，该事务就不会对磁盘上的数据做任何影响。如果在8-9之间宕机，恢复之后可以选择回滚，也可以选择继续完成事务提交，因为此时redo log已经持久化。若在9之后系统宕机，内存映射中变更的数据还来不及刷回磁盘，那么系统恢复之后，可以根据redo log把数据刷回磁盘。

所以，**redo log其实保障的是事务的持久性和一致性，而undo log则保障了事务的原子性**。

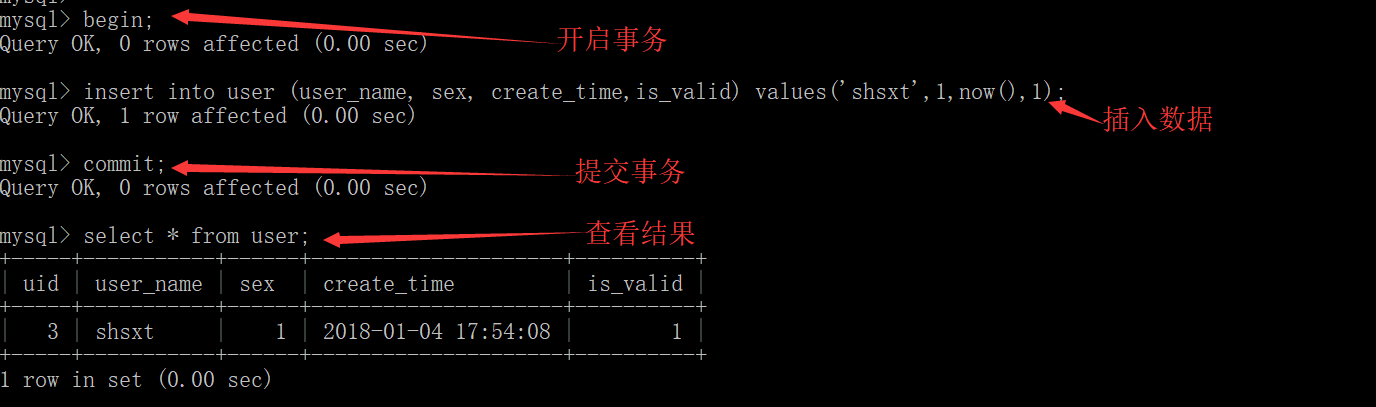
#### 命令

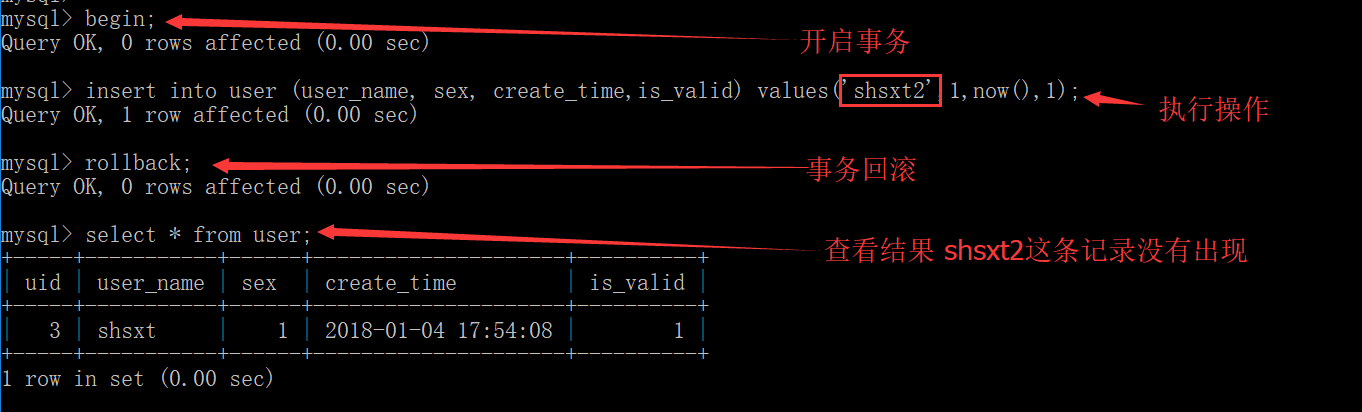
在 MySQL 命令行的默认设置下，**事务都是自动提交的**，即执行 SQL 语句后就会马上执行 COMMIT 操作。因此要显式地开启一个事务务须使用命令 BEGIN 或 START TRANSACTION，或者执行命令 SET AUTOCOMMIT=0，用来禁止使用当前会话的自动提交。

* BEGIN或START TRANSACTION；显式地开启一个事务；
* COMMIT；也可以使用COMMIT WORK，不过二者是等价的。COMMIT会提交事务，并使已对数据库进行的所有修改称为永久性的；
* ROLLBACK；有可以使用ROLLBACK WORK，不过二者是等价的。回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；
* SAVEPOINT identifier；SAVEPOINT允许在事务中创建一个保存点，一个事务中可以有多个SAVEPOINT；
* RELEASE SAVEPOINT identifier；删除一个事务的保存点，当没有指定的保存点时，执行该语句会抛出一个异常；
* ROLLBACK TO identifier；把事务回滚到标记点；
* SET TRANSACTION；用来设置事务的隔离级别。InnoDB存储引擎提供事务的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ和SERIALIZABLE。

#### DEMO

|  |
| --- |
| #显式开启事务  mysql> begin;  Query OK, 0 rows affected (0.00 sec)  #执行操作  mysql> insert into user (user\_name, sex, create\_time,is\_valid) values('shsxt',1,now(),1);  Query OK, 1 row affected (0.00 sec)  #事务提交  mysql> commit;  Query OK, 0 rows affected (0.00 sec)  #查询结果  mysql> select \* from user;  +-----+-----------+------+---------------------+----------+  | uid | user\_name | sex | create\_time | is\_valid |  +-----+-----------+------+---------------------+----------+  | 3 | shsxt | 1 | 2018-01-04 17:54:08 | 1 |  +-----+-----------+------+---------------------+----------+  #显式开启事务  mysql> begin;  Query OK, 0 rows affected (0.00 sec)  #执行插入SQL  mysql> insert into user (user\_name, sex, create\_time,is\_valid) values('shsxt2',1,now(),1);  Query OK, 1 row affected (0.00 sec)  #事务回滚  mysql> rollback;  Query OK, 0 rows affected (0.00 sec)  #查看结果发现并没有出现shsxt2这条记录  mysql> select \* from user;  +-----+-----------+------+---------------------+----------+  | uid | user\_name | sex | create\_time | is\_valid |  +-----+-----------+------+---------------------+----------+  | 3 | shsxt | 1 | 2018-01-04 17:54:08 | 1 |  +-----+-----------+------+---------------------+----------+ |





## 作业

访问开源中国网的注册网址：https://www.oschina.net/home/reg，设计一个用户注册的表。