

---

## 第三章 MySQL 数据库的使用

### 一、MySQL 简介

#### 1 什么是 MySQL

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 公司。MySQL 是一种关系型数据库管理系统，关系型数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

#### 2 MySQL 特点

MySQL 是开源的，所以你不需要支付额外的费用。

MySQL 支持大型系统的数据库。可以处理拥有上千万条记录的大型数据库。

MySQL 使用标准的 SQL 数据语言形式。

MySQL 可以运行于多个系统上，并且支持多种语言。这些编程语言包括 C、C++、Python、Java、Perl、PHP、Eiffel、Ruby 和 Tcl 等。

MySQL 存储数据量较大，32 位系统表文件最大可支持 4GB，64 位系统支持最大的表文件为 8TB。

MySQL 是可以定制的，采用了 GPL 协议，你可以修改源码来开发自己的 MySQL 系统。

### 二、MySQL 的安装与卸载

#### 1 MySQL 版本说明

MySQL Community Server 社区版本，开源免费，但不提供官方技术支持。

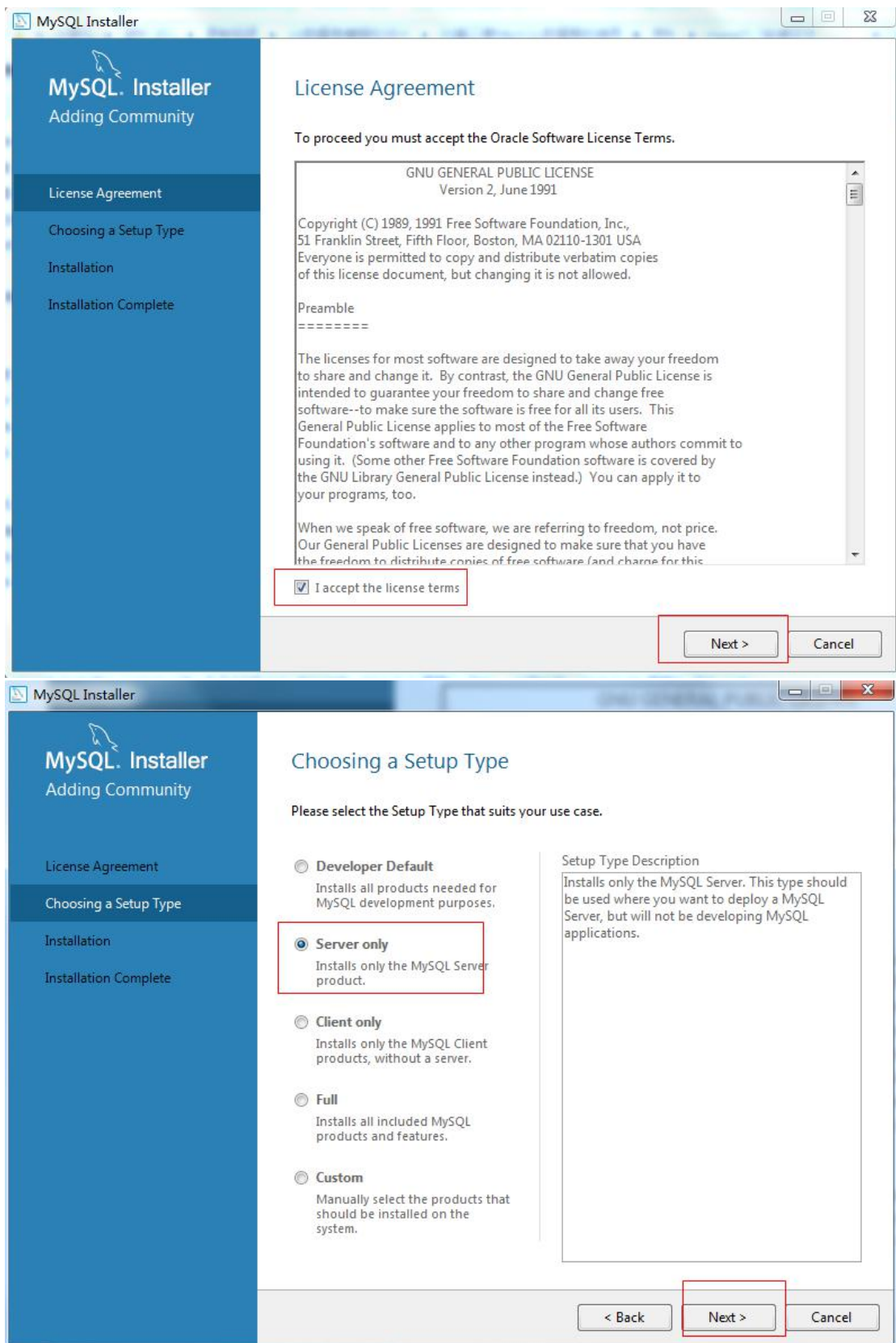
MySQL Enterprise Edition 企业版本，需付费，可以试用 30 天。

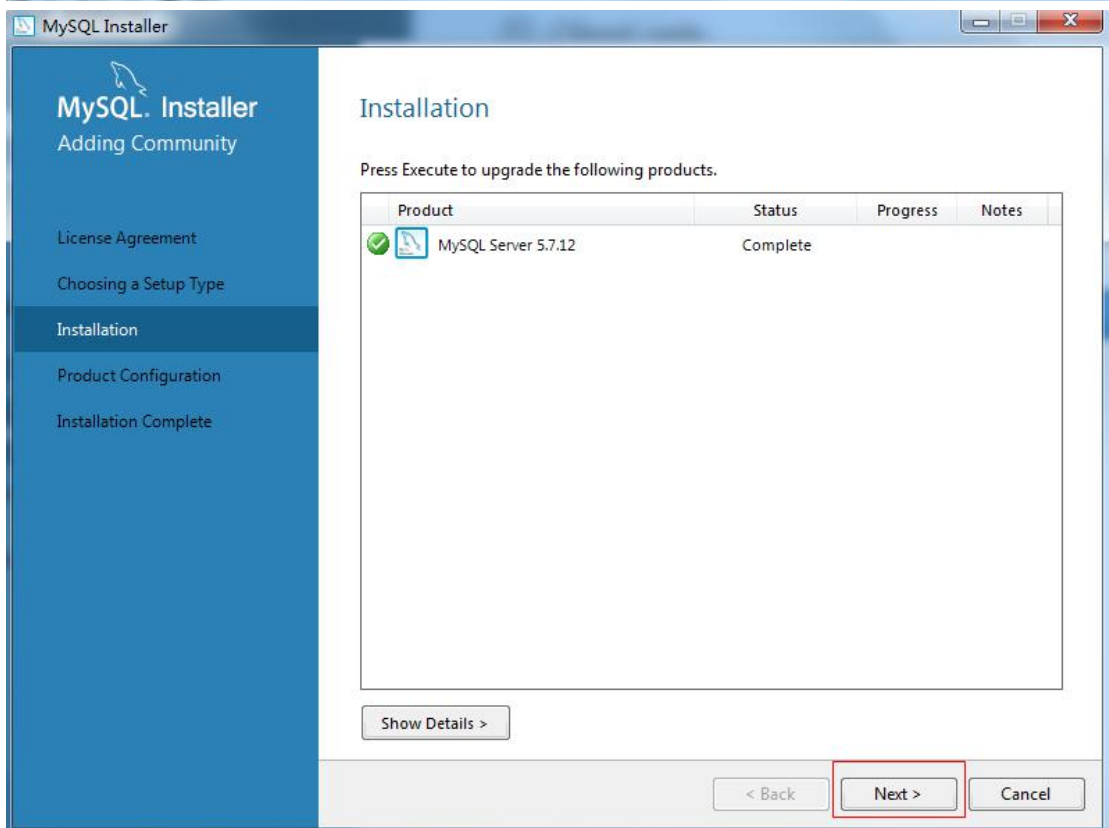
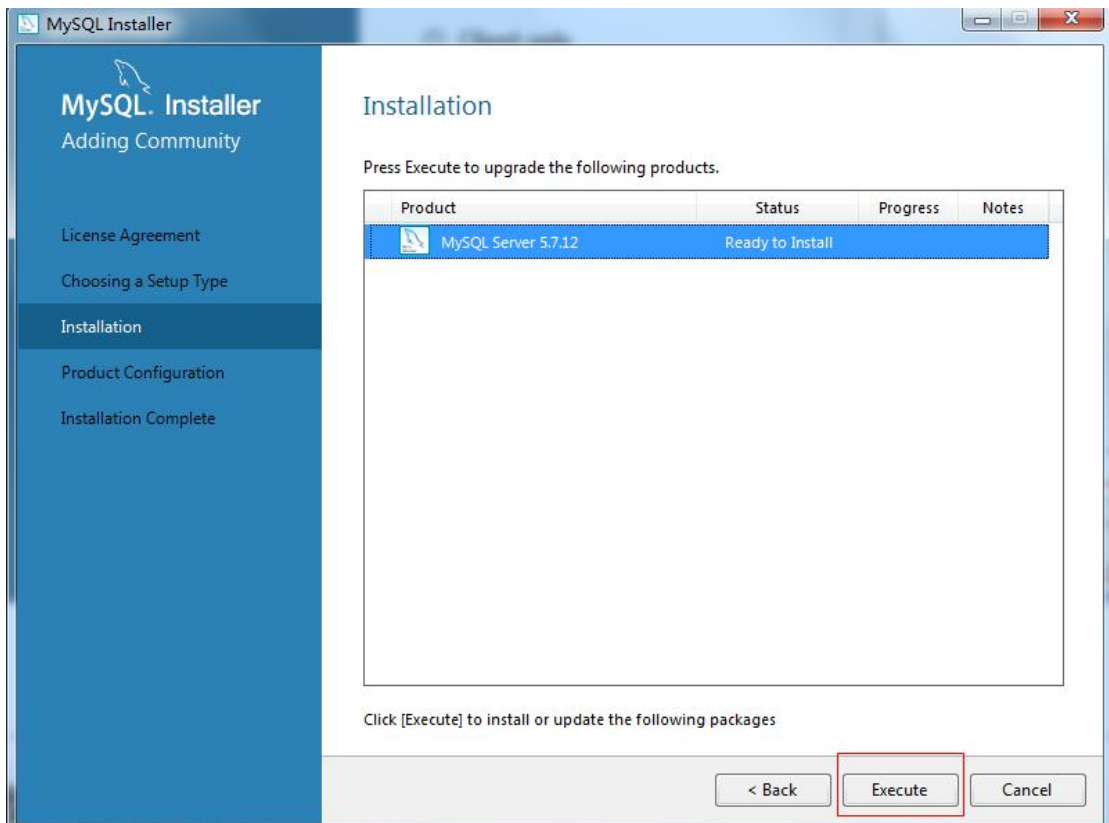
MySQL Cluster 集群版，开源免费。可将几个 MySQL Server 封装成一个 Server。

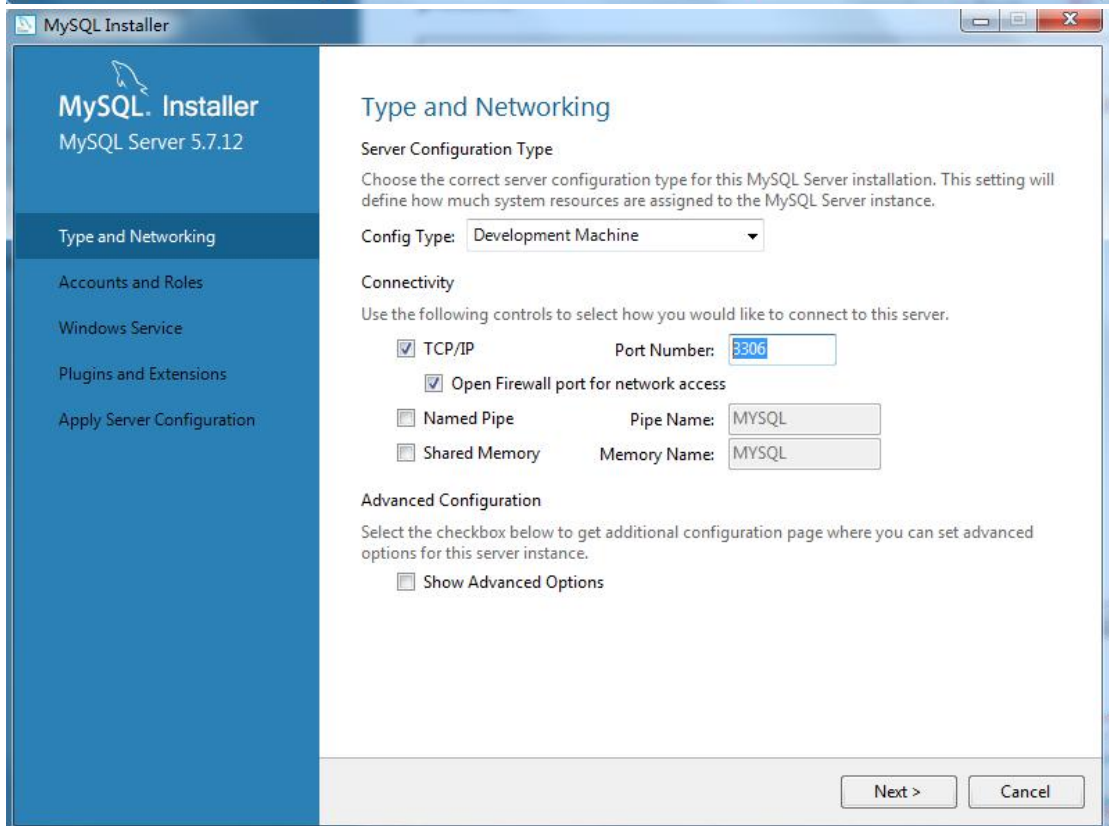
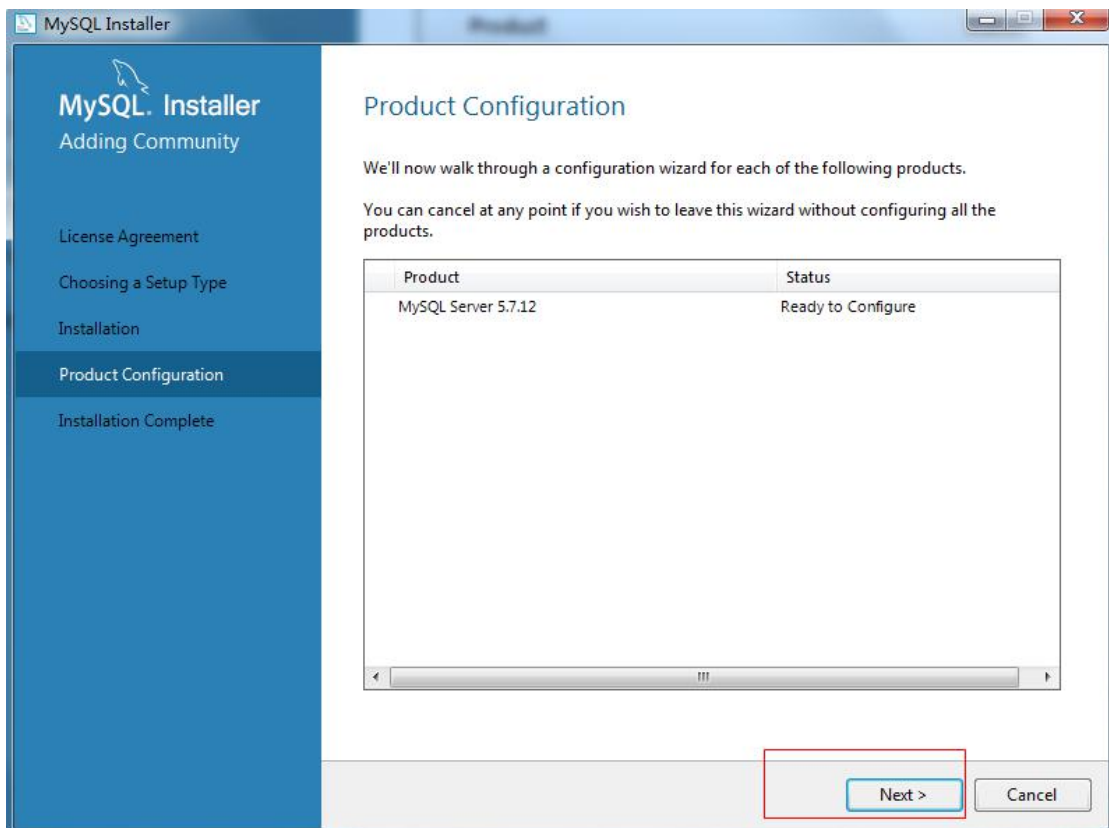
MySQL Cluster CGE 高级集群版，需付费。

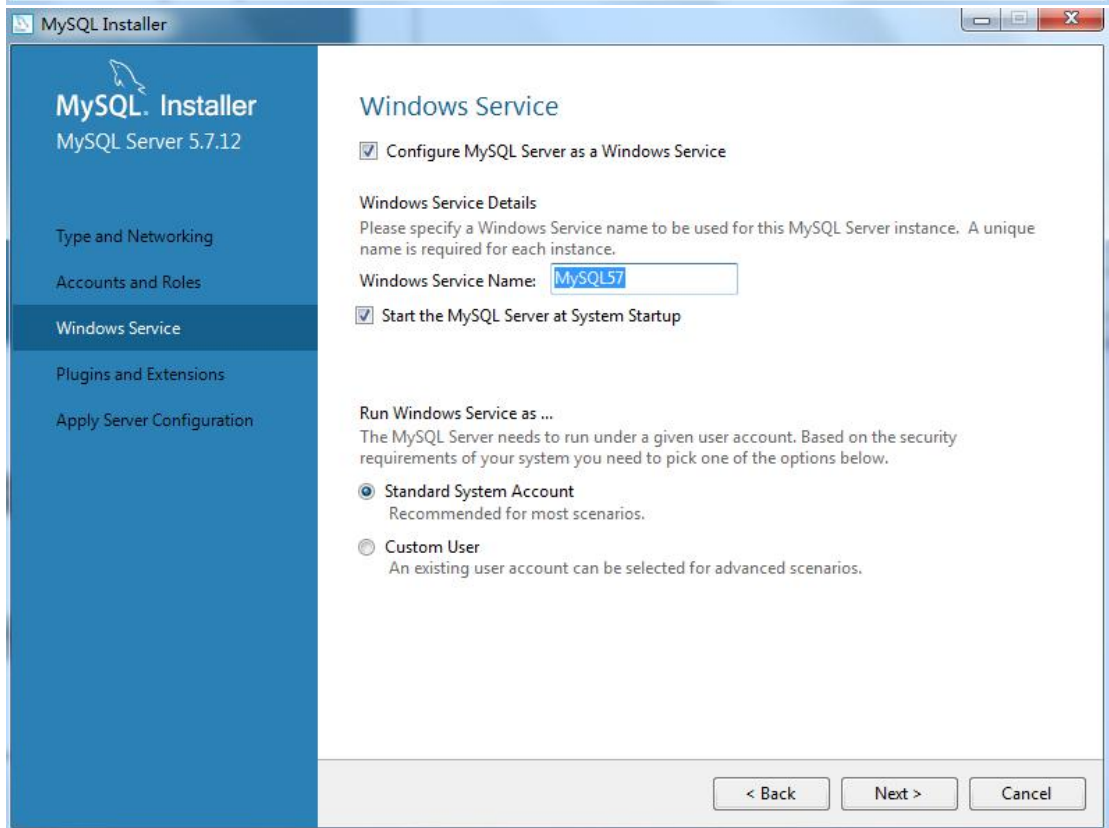
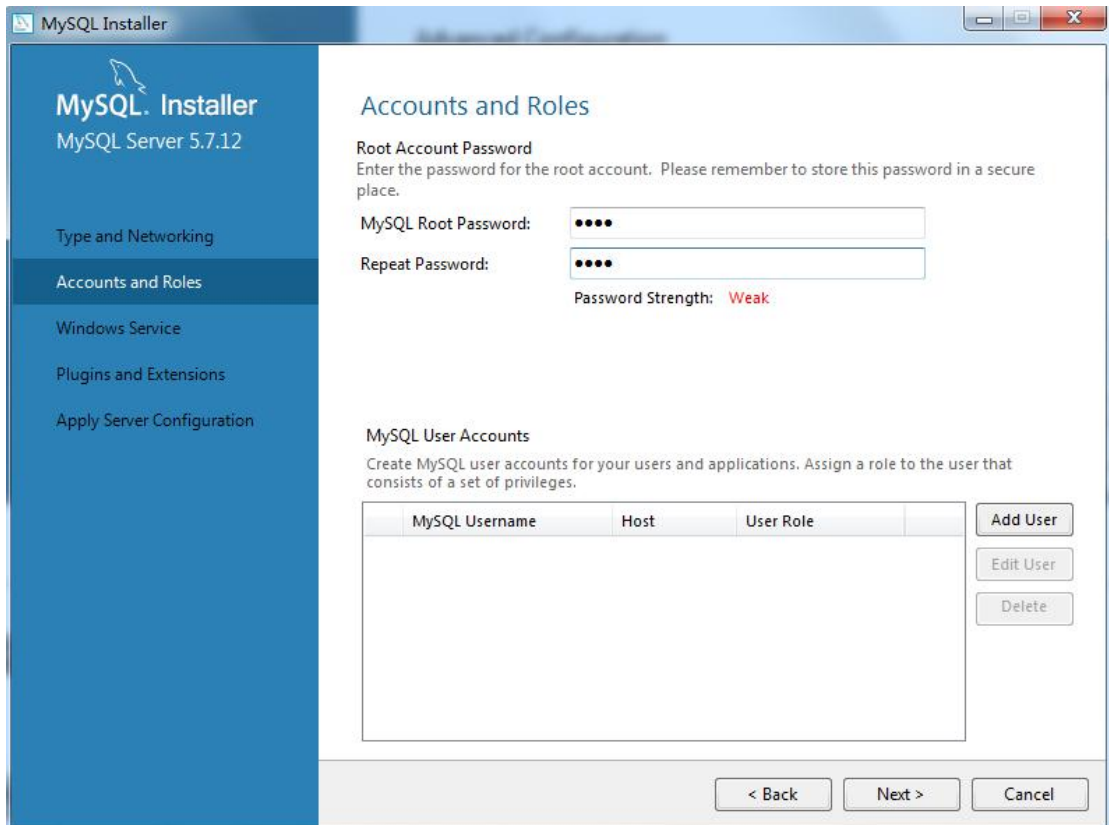
本套视频使用版本 MySQL5.7 社区版

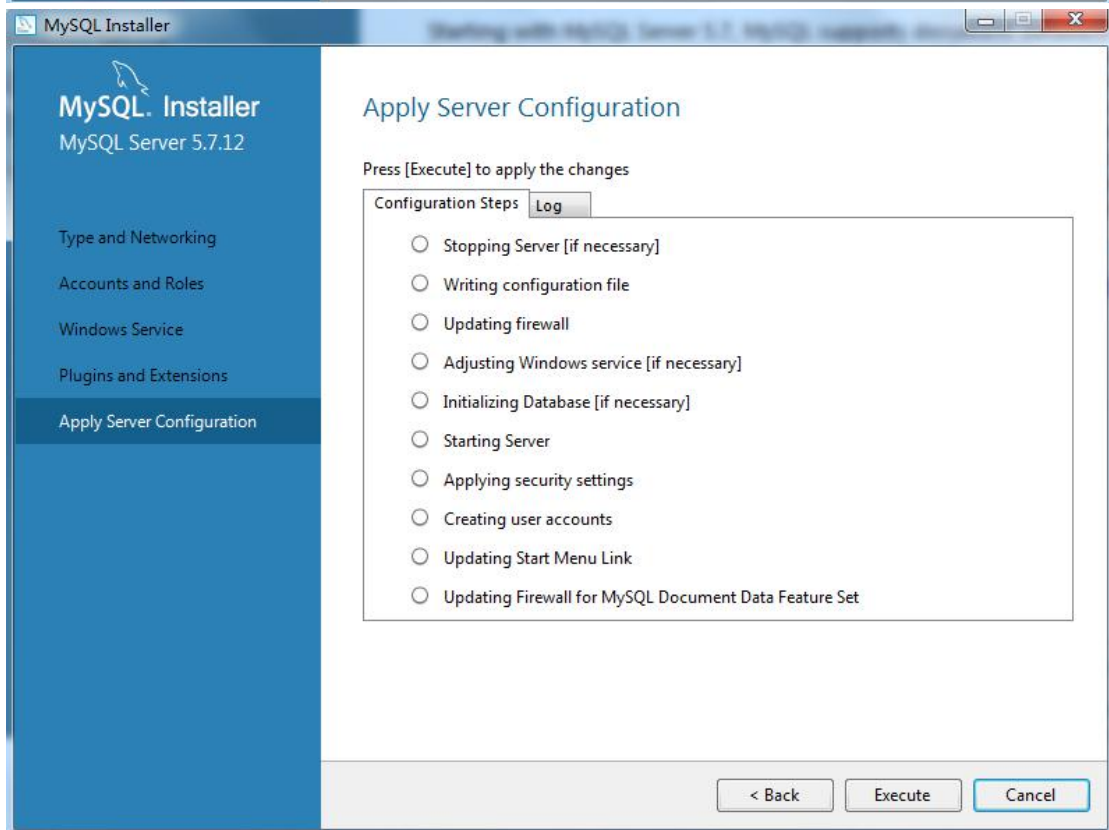
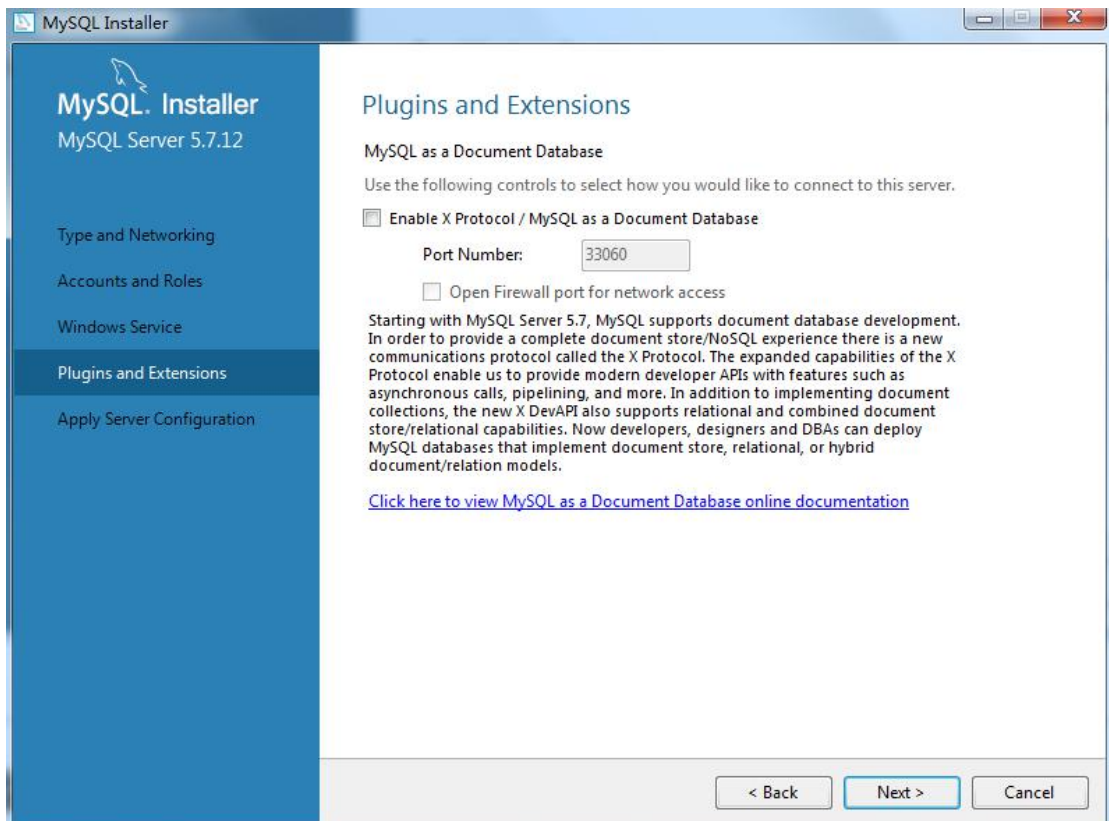
## 2 安装 MySQL



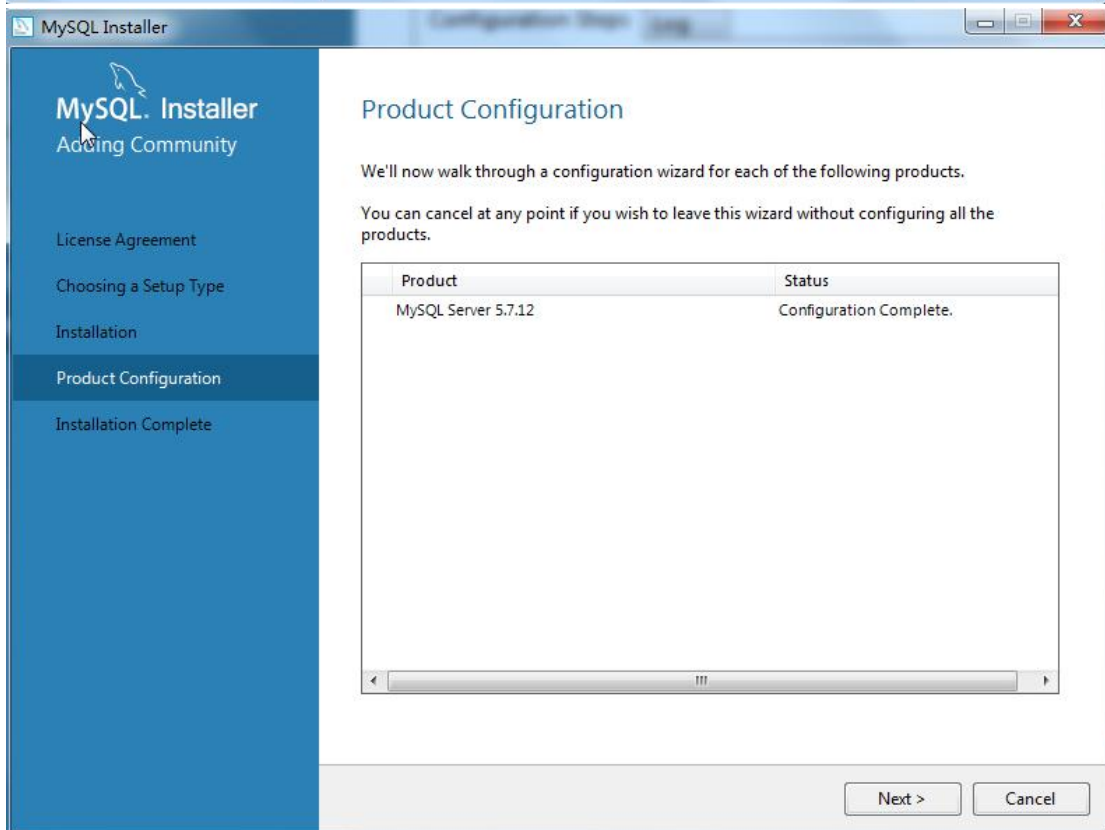
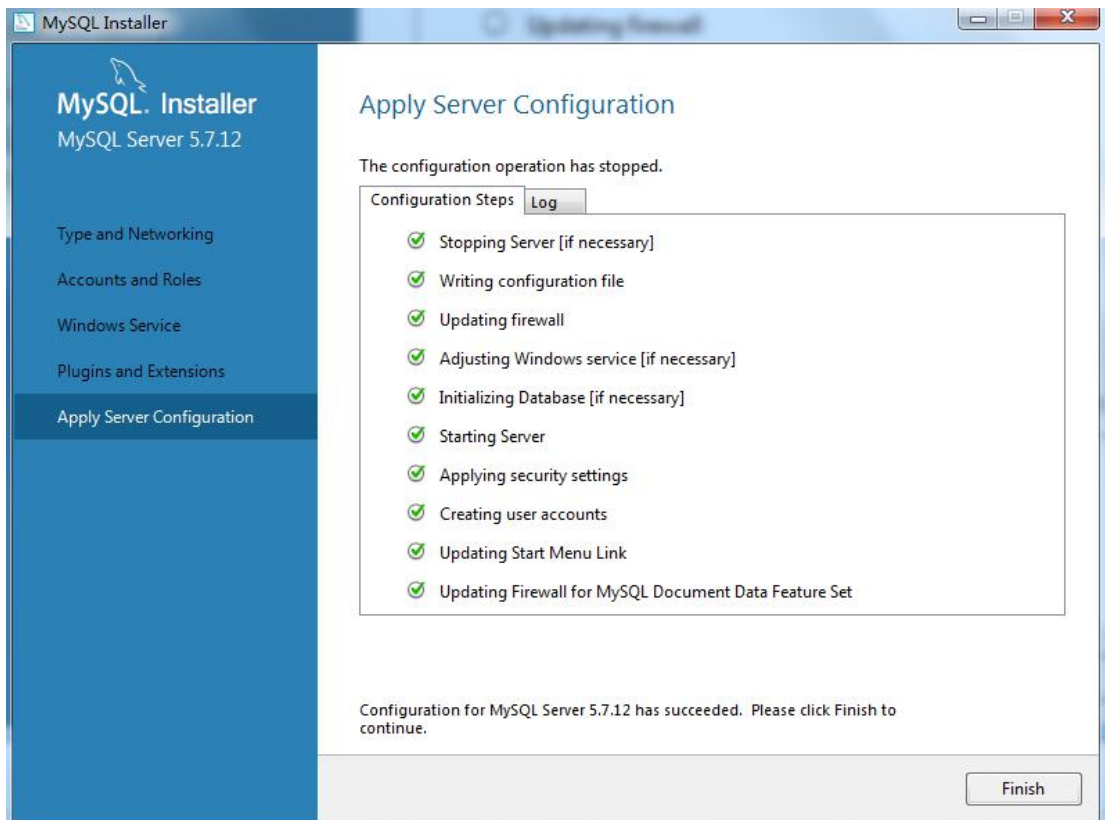


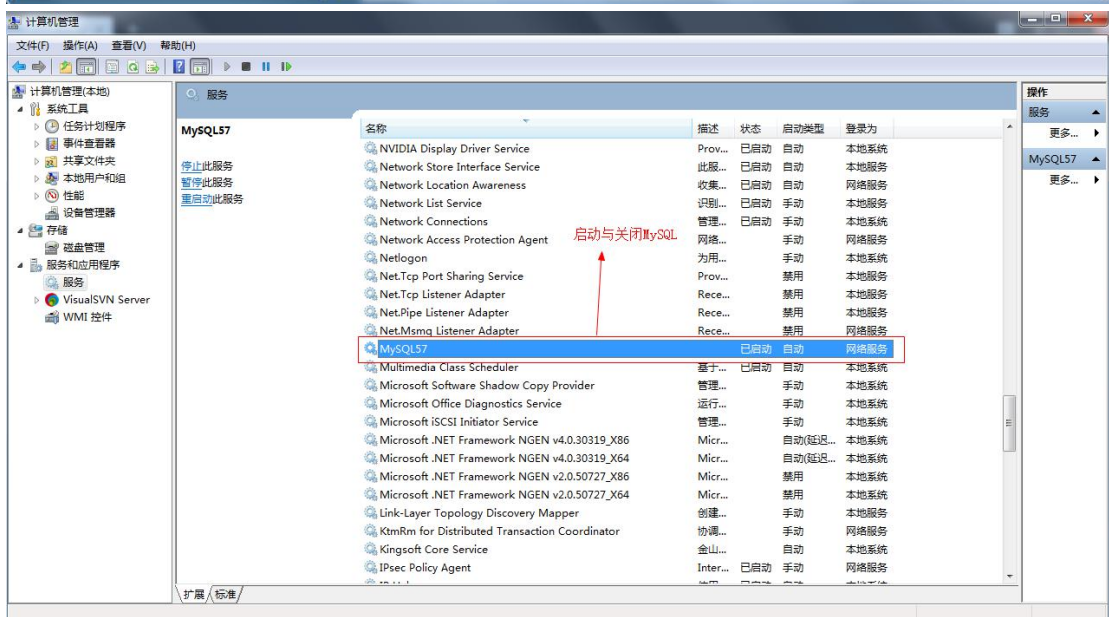
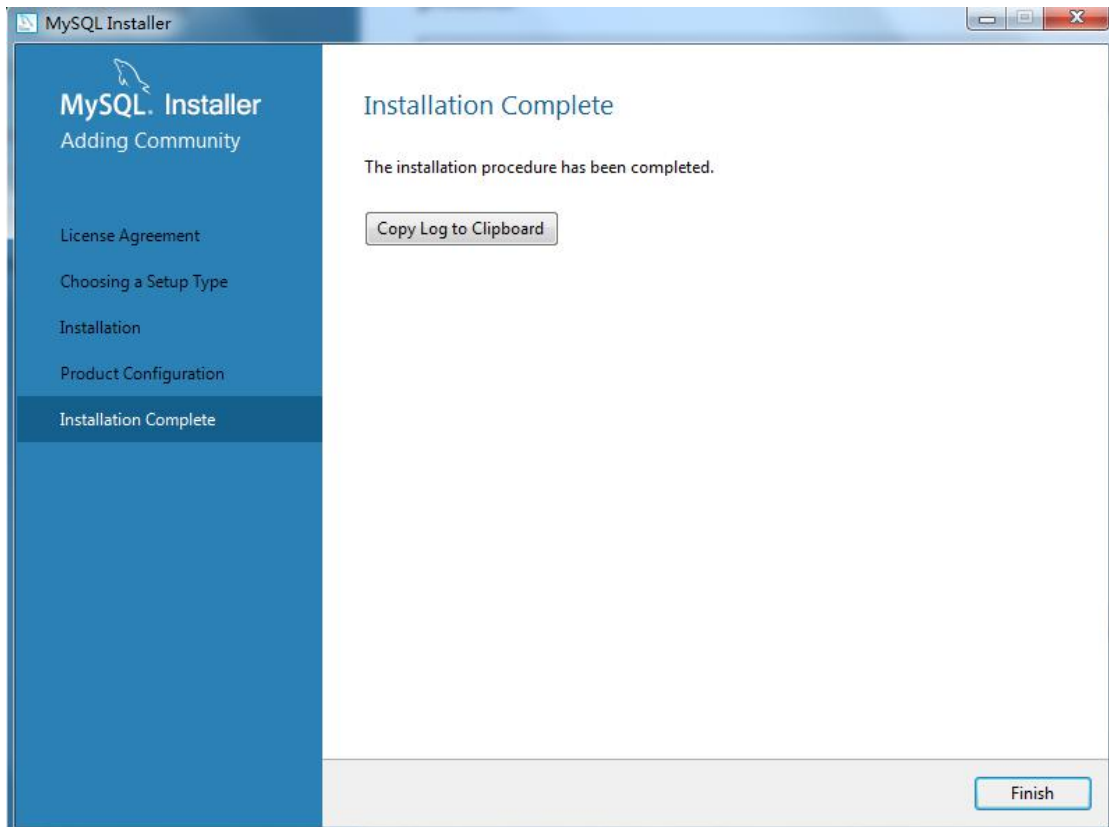












### 3 卸载 MySQL

#### 3.1 停止 MySQL 服务

开始-->所有应用-->Windows 管理工具-->服务，将 MySQL 服务停止。



---

## 3.2 卸载 mysql server

控制面板-->所有控制面板项-->程序和功能，将 mysql server 卸载掉。

## 3.3 MySQL 安装目录

将 MySQL 安装目录下的 MySQL 文件夹删除（C:\Program Files (x86)\MySQL 或者 C:\Program Files \MySQL）

## 3.4 删除注册表中信息

运行“regedit”文件，打开注册表，删除如下文件夹：

HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Services\eventlog\Application\MySQL 文件夹

HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet002\Services\eventlog\Application\MySQL 文件夹。

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\eventlog\Application\MySQL 的文件夹。

如果没有相应的文件夹，就不用删除了。

## 3.5 删除 MySQL 数据库目录

删除 C 盘下的“C:\ProgramData\MySQL”文件夹，如果删除不了则用 360 粉碎掉即可。

该 programData 文件默认是隐藏的，设置显示后即可见，或者直接复制 C:\ProgramData 到地址栏回车即可进入。将整个 MySQL 文件夹删除掉。

## 3.6 删除服务中的 MySQL

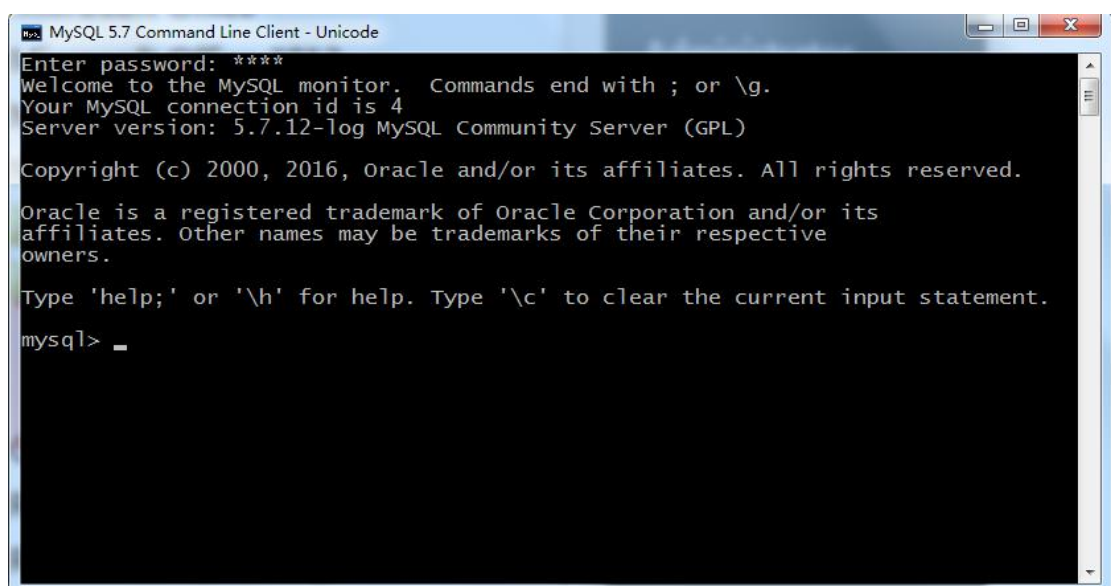
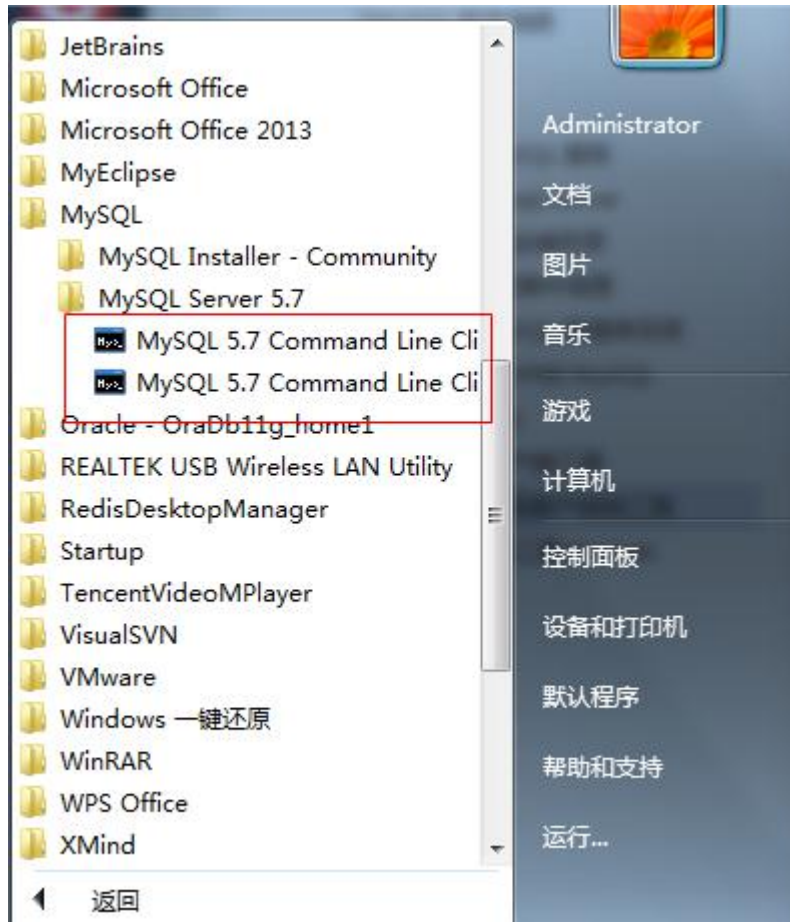
如果已经将 MySQL 卸载，但通过“开始-->所有应用-->Windows 管理工具-->服务”查看到 MySQL 服务仍然残留在系统服务里，可以在 CMD 里输入一条命令就可以将服务删除：  
sc delete mysql57 //这里的 mysql 是你要删除的服务名。

## 3.7 重启系统

重启系统安装 MySQL。

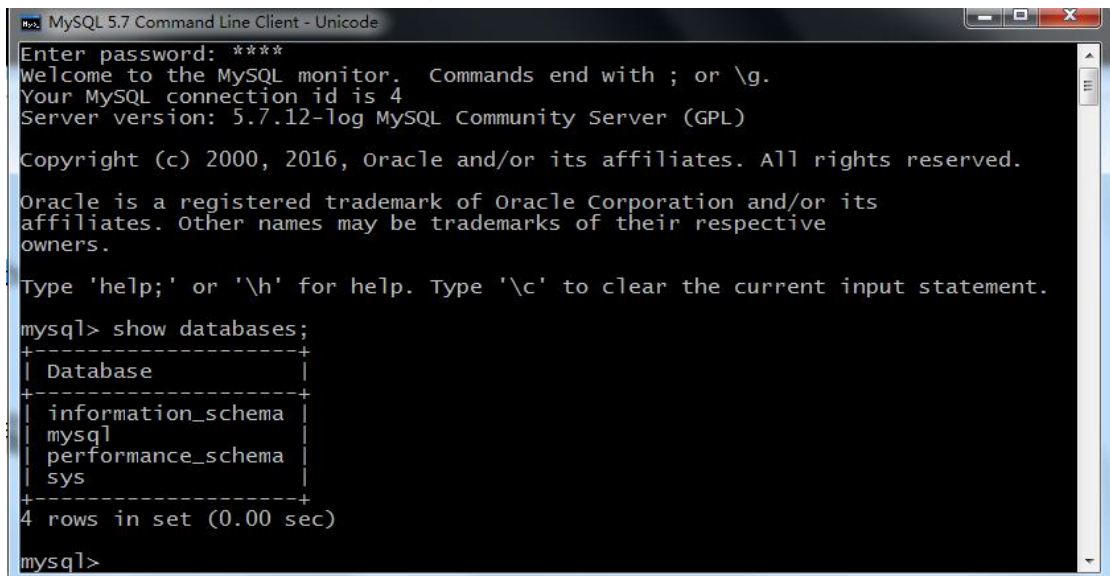
### 三、 MySQL 的客户端工具

#### 1 MySQL 自带的客户端工具



## 1.1 示例

可通过 `show databases` 命令查看 MySQL 中的库。



```
MySQL 5.7 Command Line Client - Unicode
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.12-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

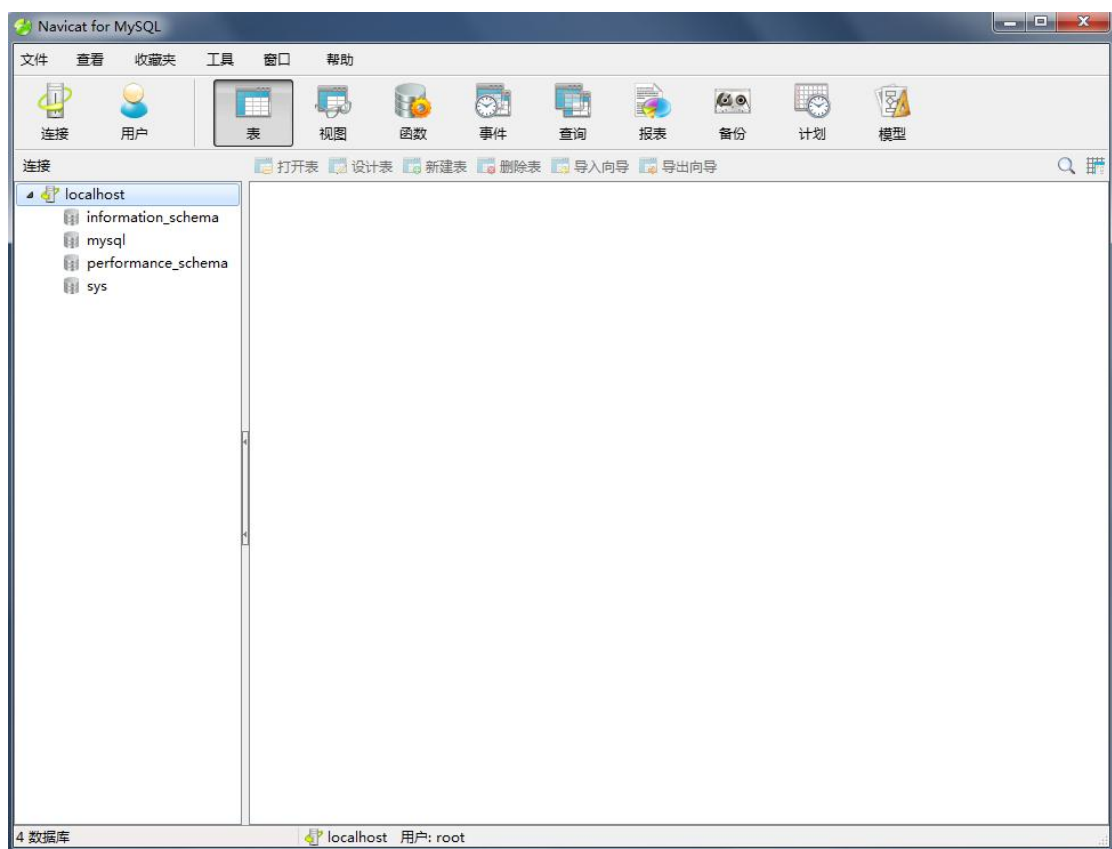
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| performance_schema |
| sys       |
+-----+
4 rows in set (0.00 sec)

mysql>
```

## 2 第三方客户端工具 Navicat



---

## 四、 MySQL 与 Oracle 的区别

### 1 实例区别

MySQL 是轻量级数据库，开源免费。Oracle 是收费的而且价格非常高。  
MySQL 一个实例可以操作多个库，而 Oracle 一个实例只能对应一个库。  
MySQL 安装完后 300M 而 Oracle 有 3G 左右。

### 2 操作区别

主键： MySQL 一般使用自动增长类型，而 Oracle 则需要使用序列对象。

单引号的处理： MySQL 里可以用双引号包起字符串，ORACLE 里只可以用单引号包起字符串。

分页的 SQL 语句： MYSQL 用 LIMIT，而 Oracle 需要使用内建视图和 rownum 伪列。

事务处理： MySQL 默认是自动提交，而 Oracle 默认不自动提交，需要用户 CTL 语言进行事务提交。

## 五、 操作 MySQL

### 1 创建与删除数据库

#### 1.1 创建数据库

##### 1.1.1 使用命令创建数据库

CREATE DATABASE 数据库名 DEFAULT CHARACTER SET 字符编码;

###### 1.1.1.1 示例

创建一个 test 的数据库，并查看该数据库，以及该数据库的编码。

创建数据库：

```
create database test default character set utf8;
```

查看数据库：

```
show databases;
```

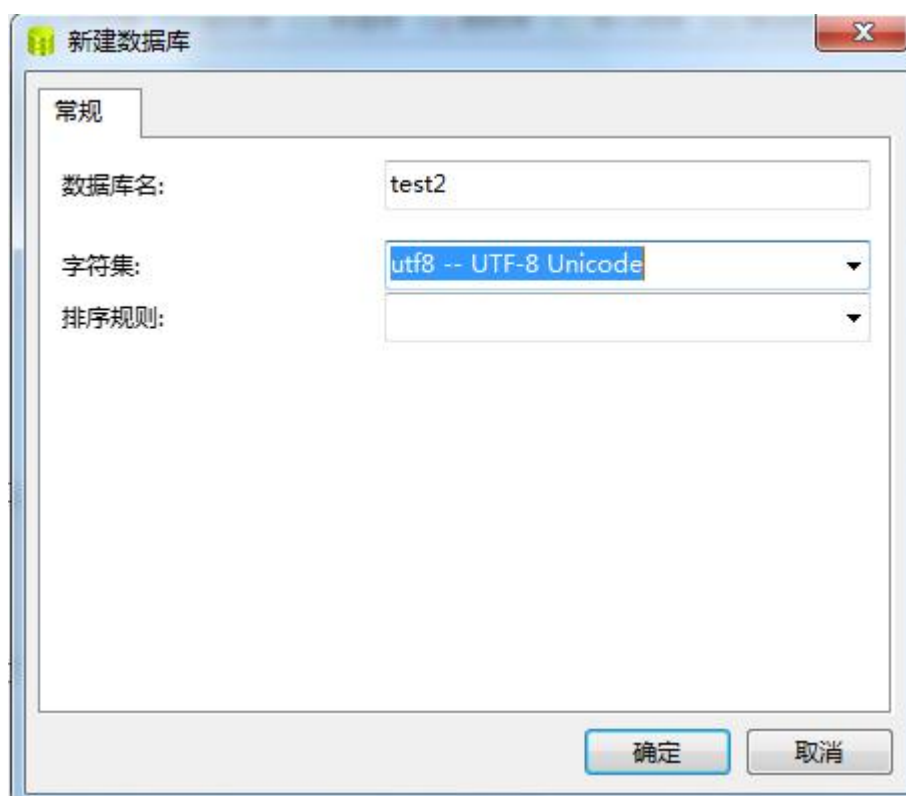
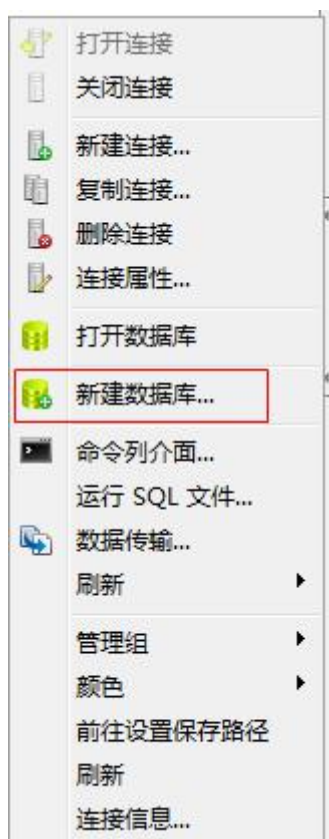
查看数据库编码：

```
select schema_name,default_character_set_name from information_schema.schemata where schema_name = 'test';
```

##### 1.1.2 使用 Navicat 创建数据库

###### 1.1.2.1 示例

创建一个 test2 的数据库



---

## 1.2 删除数据库

### 1.2.1 使用命令删除数据库

Drop database 数据库名称

#### 1.2.1.1 示例

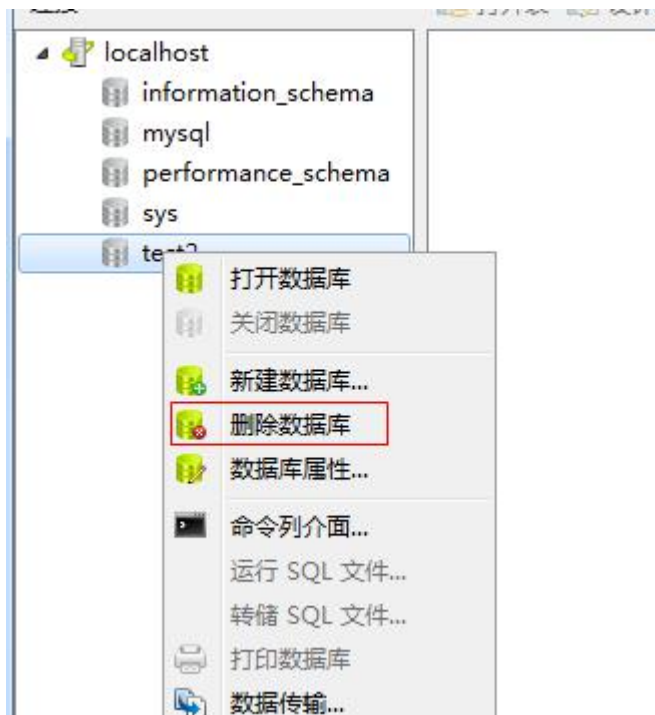
删除 test 数据库

```
drop database test;
```

### 1.2.2 使用 Navicat 工具删除数据库

#### 1.2.2.1 示例

删除 test2 数据库



## 2 选择数据库

需要在哪个库中创建表需要先选择该数据库。

Use 需要选择的库的名称。

### 2.1 示例一

创建一个名称为 bjsxt 的数据库，编码为 utf8。

```
create database bjsxt default character set = utf8;
```



## 2.2 示例二

选择该数据库

```
use bjsxt;
```

## 3 MySQL 中的数据类型

### 3.1 数值类型

MySQL 支持所有标准 SQL 数值数据类型。

作为 SQL 标准的扩展，MySQL 也支持整数类型 TINYINT、MEDIUMINT 和 BIGINT。

MySQL数据类型	含义（有符号）
tinyint(m)	1个字节 范围(-128~127)
smallint(m)	2个字节 范围(-32768~32767)
mediumint(m)	3个字节 范围(-8388608~8388607)
int(m)	4个字节 范围(-2147483648~2147483647)
bigint(m)	8个字节 范围(+/-9.22*10的18次方)

数值类型中的长度 **m** 是指显示长度，并不表示存储长度，只有字段指定 **zerofill** 时有用  
例如：int(3)，如果实际值是 2，如果列指定了 **zerofill**，查询结果就是 002，左边用 0 来  
填充

### 3.2 浮点型

MySQL数据类型	含义
float(m,d)	单精度浮点型 8位精度(4字节) m总个数，d小数位
double(m,d)	双精度浮点型 16位精度(8字节) m总个数，d小数位

### 3.3 字符串型

MySQL数据类型	含义
char(n)	固定长度，最多255个字符
varchar(n)	可变长度，最多65535个字符
tinytext	可变长度，最多255个字符
text	可变长度，最多65535个字符
mediumtext	可变长度，最多2的24次方-1个字符
longtext	可变长度，最多2的32次方-1个字符

#### 3.3.1 char 和 varchar

1)char(n) 若存入字符数小于 n，则以空格补于其后，查询之时再将空格去掉。所以 char 类型存储的字符串末尾不能有空格，varchar 不限于此。

2)char 类型的字符串检索速度要比 varchar 类型的快。

#### 3.3.2 varchar 和 text

1)varchar 可指定 n，text 不能指定，内部存储 varchar 是存入的实际字符数 +1 个字节 (n<=255) 或 2 个字节(n>255)，text 是实际字符数 +2 个字节。

2)text 类型不能有默认值。

3)varchar 可直接创建索引，text 创建索引要指定前多少个字符。varchar 查询速度快于 text，在都创建索引的情况下，text 的索引似乎不起作用。

### 3.4 日期类型

MySQL数据类型	含义
date	日期 '2008-12-2'
time	时间 '12:25:36'
datetime	日期时间 '2008-12-2 22:06:44'
timestamp	自动存储记录修改时间

### 3.5 二进制数据(BLOB)

1)BLOB 和 TEXT 存储方式不同，TEXT 以文本方式存储，英文存储区分大小写，而 Blob 是以二进制方式存储，不分大小写。

2)BLOB 存储的数据只能整体读出。

3)TEXT 可以指定字符集，BLOB 不用指定字符集。

## 4 创建表与删除表

### 4.1 创建表

#### 4.1.1 通过 DDL 语句创建表

##### 4.1.1.1 示例

创建一个 employees 表包含雇员 ID，雇员名字，雇员薪水。

```
create table employees(employee_id int,last_name varchar(30),salary float  
(8,2));
```

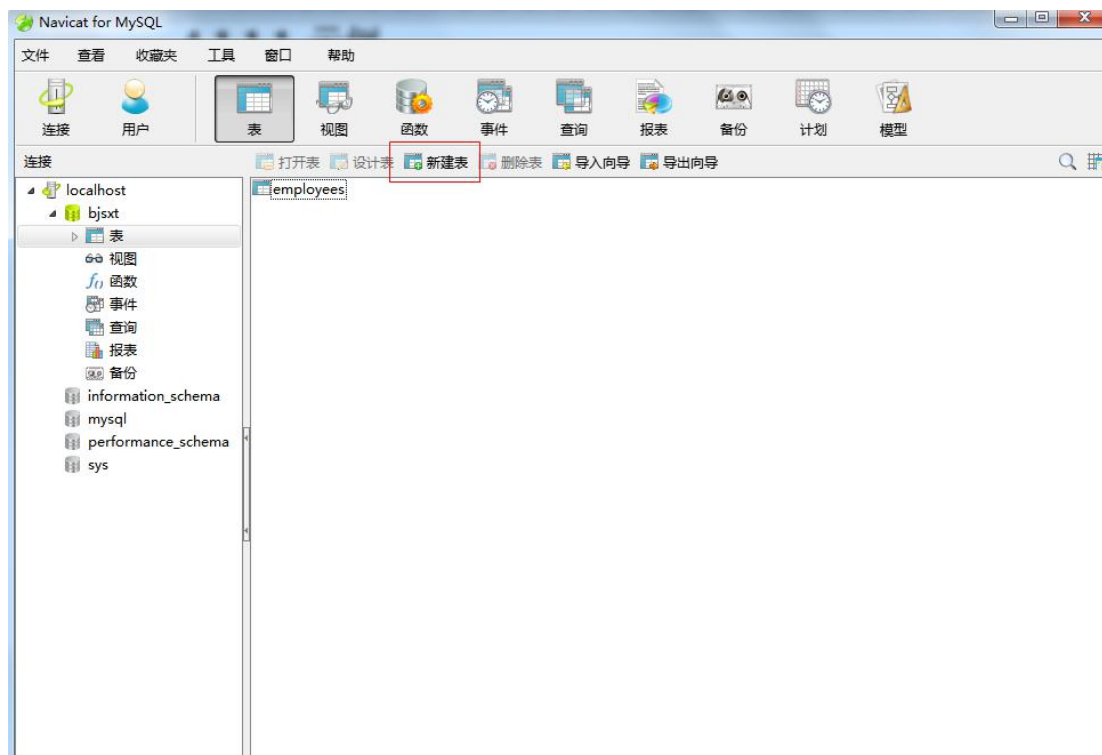
查看已创建的表

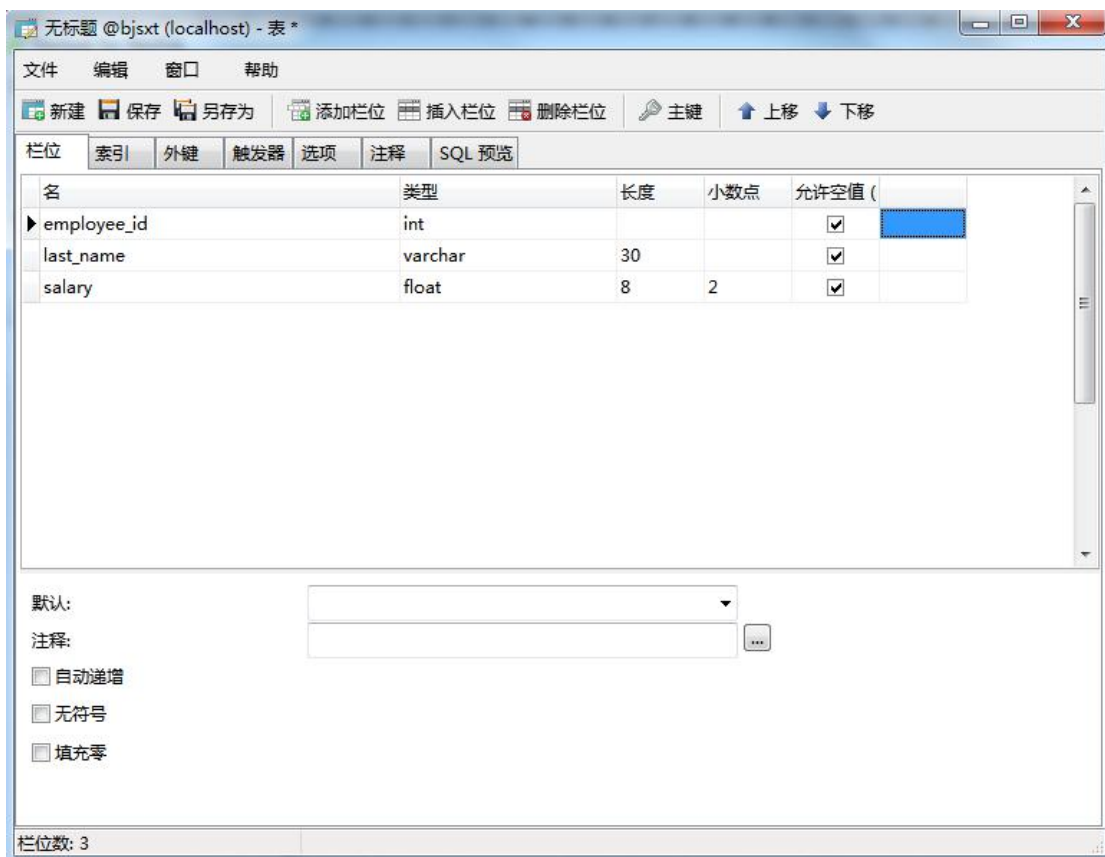
```
Show tables;
```

#### 4.1.2 通过 Navicat 工具创建表

##### 4.1.2.1 示例

创建一个 employees2 表包含雇员 ID，雇员名字，雇员薪水。





## 4.2 删除表

### 4.2.1 通过 DDL 语句删除表

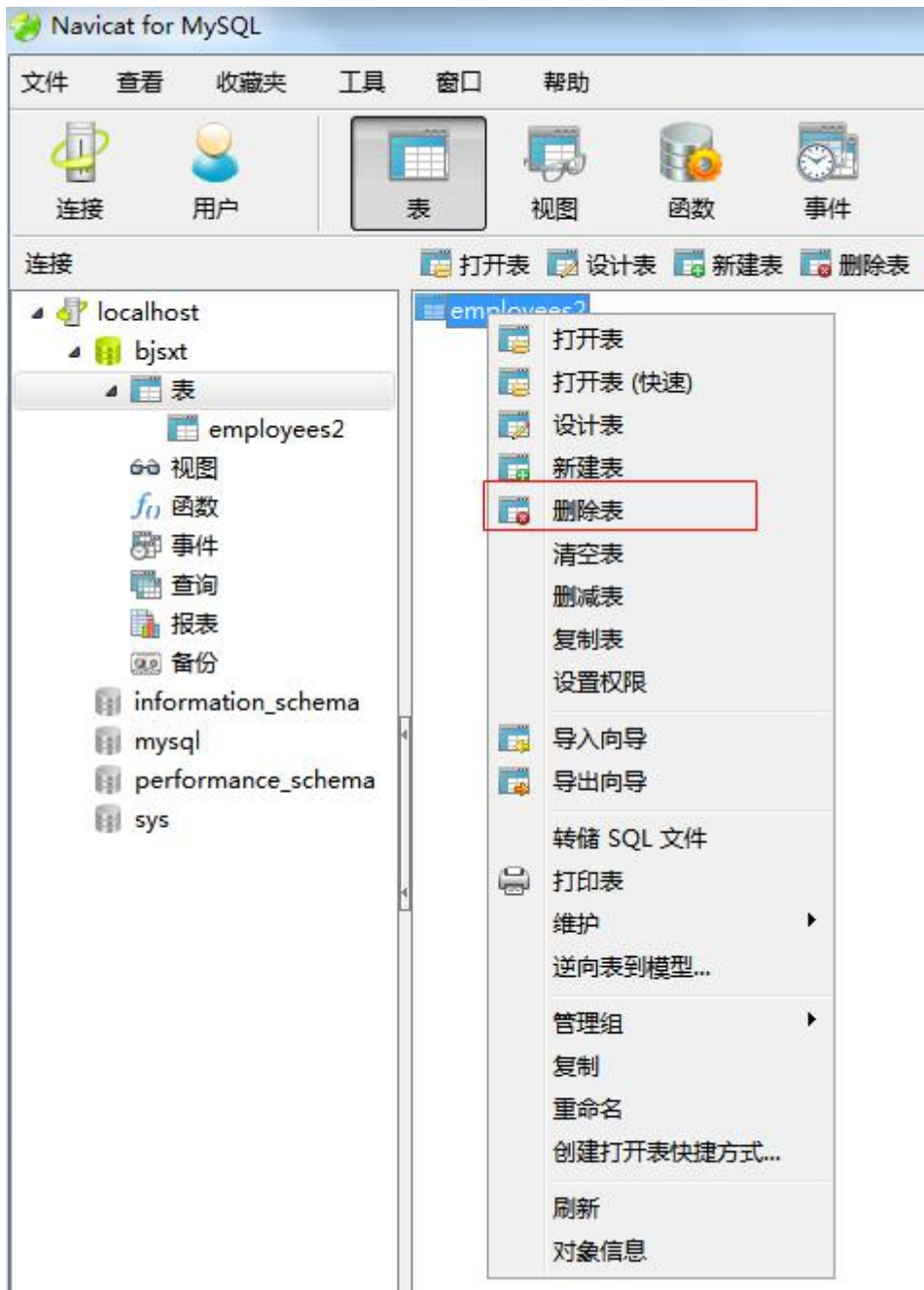
#### 4.2.1.1 示例

删除 employees 表  
`drop table employees;`

### 4.2.2 通过 Navicat 工具删除表

#### 4.2.2.1 示例

删除 employees2 表



## 5 修改表

### 5.1 使用 DDL 语句修改表名

ALTER TABLE 旧表名 RENAME 新表名

#### 5.1.1 示例一

创建一个 employees 表包含雇员 ID，雇员名字，雇员薪水。

```
create table employees(employee_id int,last_name varchar(30),salary float
(8,2));
```

---

### 5.1.2 示例二

将 employees 表名修改为 emp。

```
alter table employees rename emp;
```

## 5.2 使用 DDL 修改列名

ALTER TABLE 表名 CHANGE COLUMN 旧列名 新列名 类型

### 5.2.1 示例

将 emp 表中的 last\_name 修改为 name

```
alter table emp change column last_name name varchar(30);
```

## 5.3 使用 DDL 语句修改列类型

ALTER TABLE 表名 MODIFY 列名 新类型

### 5.3.1 示例

将 emp 表中的 name 的长度指定为 40

```
alter table emp modify name varchar(40);
```

## 5.4 使用 DDL 语句添加列

ALTER TABLE 表名 ADD COLUMN 新列名 类型

### 5.4.1 示例

在 emp 表中添加一个新的列为 commission\_pct

```
alter table emp add column commission_pct float(4,2);
```

## 5.5 使用 DDL 语句删除列

ALTER TABLE 表名 DROP COLUMN 列名

### 5.5.1 示例

删除 emp 表中的 commission\_pct

```
alter table emp drop column commission_pct;
```



---

## 6 MySQL 中的约束

### 6.1 约束类型

- 非空约束(not null)
- 唯一性约束(unique)
- 主键约束(primary key) PK
- 外键约束(foreign key) FK
- 检查约束(目前 MySQL 不支持、Oracle 支持)

### 6.2 创建表时添加约束

查询表中的约束信息

SHOW KEYS FROM 表名

#### 6.2.1 示例一

创建 departments 表包含 department\_id 该列为主键且自动增长，department\_name 列不允许重复，location\_id 列不允许含有空值。

```
create table departments(department_id int primary key auto_increment, department_name varchar(30) unique, location_id int not null);
```

#### 6.2.2 示例二

创建 employees 表包含 employees\_id 该列为主键且自动增长，last\_name 列不允许含有空值，email 列不允许有重复不允许含有空值，dept\_id 为外键参照 departments 表的主键。

```
create table employees(employees_id int primary key auto_increment, last_name varchar(30) not null, email varchar(40) not null unique, dept_id int, constraint emp_fk foreign key(dept_id) references departments(department_id));
```

### 6.3 修改表实现约束的添加与删除

#### 6.3.1 主键约束

##### 6.3.1.1 添加主键约束

ALTER TABLE 表名 ADD PRIMARY KEY(列名)

##### 6.3.1.1.1 示例

将 emp 表中的 employee\_id 修改为主键且自动增长

添加主键：alter table emp add primary key(employee\_id);

添加自动增长：alter table emp modify employee\_id int auto\_increment;

---

### 6.3.1.2 删除主键约束

ALTER TABLE 表名 DROP PRIMARY KEY

注意：删除主键时，如果主键列具备自动增长能力，需要先去掉自动增长，然后在删除主键。

#### 6.3.1.2.1 示例

删除 employee\_id 的主键约束。

去掉自动增长：alter table emp modify employee\_id int;

删除主键：alter table emp drop primary key;

### 6.3.2 非空约束

#### 6.3.2.1 添加非空约束

ALTER TABLE 表名 MODIFY 列名 类型 NOT NULL

##### 6.3.2.1.1 示例

向 emp 表中的 salary 添加非空约束。

alter table emp modify salary float(8,2) not null;

#### 6.3.2.2 删除非空约束

ALTER TABLE 表名 MODIFY 列名 类型 NULL

##### 6.3.2.2.1 示例

删除 salary 的非空约束。

alter table emp modify salary float(8,2) null;

### 6.3.3 唯一约束

#### 6.3.3.1 添加唯一约束

ALTER TABLE 表名 ADD CONSTRAINT 约束名 UNIQUE(列名)

##### 6.3.3.1.1 示例

向 emp 表中的 name 添加唯一约束。

alter table emp add constraint emp\_uk unique(name);

#### 6.3.3.2 删除唯一约束

ALTER TABLE 表名 DROP KEY 约束名

---

#### 6.3.3.2.1 示例

删除 name 的唯一约束。

```
alter table emp drop key emp_uk;
```

### 6.3.4 外键约束

#### 6.3.4.1 添加外键约束

ALTER TABLE 表名 ADD CONSTRAINT 约束名 FOREIGN KEY(列名)  
REFERENCES 参照的表名(参照的列名)

##### 6.3.4.1.1 示例一

修改 emp 表，添加 dept\_id 列。

```
alter table emp add column dept_id int;
```

##### 6.3.4.1.2 示例二

向 emp 表中的 dept\_id 列添加外键约束。

```
alter table emp add constraint e_fk foreign key(dept_id) references departments(department_id);
```

#### 6.3.4.2 删除外键约束

删除外键：

```
ALTER TABLE 表名 DROP FOREIGN KEY 约束名
```

删除外键索引(索引名与约束名相同)：

```
ALTER TABLE 表名 DROP INDEX 索引名
```

##### 6.3.4.2.1 示例

删除 dept\_id 的外键约束。

删除外键：alter table emp drop foreign key e\_fk;

删除索引：alter table emp drop index e\_fk;

## 7 MySQL 中的 DML 操作

### 7.1 添加数据(INSERT)

#### 7.1.1 插入数据

##### 7.1.1.1 选择插入

INSERT INTO 表名(列名 1, 列名 2, 列名 3.....) VALUES(值 1, 值 2, 值 3.....)

---

#### 7.1.1.1.1 示例

向 departments 表中添加一条数据，部门名称为 market，工作地点 ID 为 1。

```
insert into departments(department_name,location_id) values("market",1);
```

#### 7.1.1.2 完全插入

INSERT INTO 表名 VALUES(值 1, 值 2, 值 3.....)

如果主键是自动增长，需要使用 default 或者 null 或者 0 占位。

##### 7.1.1.2.1 示例一

向 departments 表中添加一条数据，部门名称为 development，工作地点 ID 为 2。使用 default 占位。

```
insert into departments values(default,"development",2);
```

##### 7.1.1.2.2 示例二

向 departments 表中添加一条数据，部门名称为 human，工作地点 ID 为 3。使用 null 占位。

```
insert into departments values(null,"human",3);
```

##### 7.1.1.2.3 示例三

向 departments 表中添加一条数据，部门名称为 teaching，工作地点 ID 为 4。使用 0 占位。

```
insert into departments values(0,"teaching",4);
```

### 7.1.2 自动增长(auto\_increment)

MySQL 中的自动增长类型要求：

- 一个表中只能有一个列为自动增长。
- 自动增长的列的类型必须是整数类型。
- 自动增长只能添加到具备主键约束与唯一性约束的列上。
- 删除主键约束或唯一性约束，如果该列拥有自动增长能力，则需要先去掉自动增长然后在删除约束。

#### 7.1.2.1 示例

创建一个 emp2 表。包含 id 该列为主键，包含 name，包含 seq\_num 要求该列为具备唯一性约束，该列的值自动增长。

```
create table emp2(id int primary key ,name varchar(30),seq_num int unique  
auto_increment);
```

### 7.1.3 默认值处理

在 MySQL 中可以使用 DEFAULT 为字段设定一个默认值。如果在插入数据时并未指

---

定该列的值，那么 MySQL 会将默认值添加到该列中。

### 7.1.3.1 创建表时指定列的默认值

#### 7.1.3.1.1 示例

创建 emp3 表，该表包含 emp\_id 主键且自动增长，包含 name，包含 address 该列默认值为“未知”。

```
create table emp3(emp_id int primary key auto_increment,name varchar(30),
address varchar(50) default 'Unknown');
```

### 7.1.3.2 修改表添加列的默认值

#### 7.1.3.2.1 示例

修改 emp3 表，添加 job\_id 该列默认值为 0。

```
alter table emp3 add column job_id int default 0;
```

### 7.1.3.3 插入数据时的默认值处理

如果在插入数据时并未指定该列的值，那么 MySQL 会将默认值添加到该列中。如果是完全项插入需要使用 default 来占位。

#### 7.1.3.3.1 示例

向 emp3 表中添加数据，要求 address 列与 job\_id 列使用默认值作为该列的值。

```
insert into emp3(name) values("admin");
insert into emp3 values(default,"oldlu",default,default);
```

## 7.2 更新数据(UPDATE)

UPDATE 表名 SET 列名=值, 列名=值 WHERE 条件

### 7.2.1mysql 的 update 的特点

- 更新的表不能在 set 和 where 中用于子查询;
- update 后面可以做任意的查询

#### 7.2.1.1 示例一

更新 emp3 表中的 id 为 1 的数据，添加 address 为 BeiJing。

```
update emp3 e set e.address = "BeiJing" where emp_id = 1;
```

#### 7.2.1.2 示例二

方式一：更新 emp3 中 id 为 2 的数据，将地址修改为与 id 为 1 用户的地址相同

```
Oracle:update emp3 e set e.address = (select address from emp3 where emp_id = 1)
where e.emp_id = 2;
```

---

MySQL: update emp3 e ,(select address from emp3 where emp\_id = 1)t set e.address = t.address where e.emp\_id = 2;

方式二：更新 emp3 中 id 为 2 的数据，将地址修改为与 id 为 1 用户的地址相同

update emp3 e set e.address = (select t1.address from (select emp\_id, address from emp3)t1 where t1.emp\_id = 1 ) where e.emp\_id = 2;

## 7.3 删除数据(DELETE)

### 7.3.1 使用 DELETE 子句

DELETE FROM 表名 WHERE 条件

#### 7.3.1.1 示例

删除 emp3 表中 emp\_id 为 1 的雇员信息。

delete from emp3 where emp\_id = 1

### 7.3.2 使用 TRUNCATE 清空表

TRUNCATE TABLE 表名

#### 7.3.2.1 示例

删除 emp3 表中的所有数据

truncate table emp3;

### 7.3.3 DELETE 与 TRUNCATE 区别

- truncate 是整体删除（速度较快）， delete 是逐条删除（速度较慢）；
- truncate 不写服务器 log, delete 写服务器 log, 也就是 truncate 效率比 delete 高的原因；
- truncate 是会重置自增值，相当于自增列会被置为初始值，又重新从 1 开始记录，而不是接着原来的值。而 delete 删除以后，自增值仍然会继续累加。

## 8 MySQL 中的事务处理

在 MySQL 中，默认情况下，事务是自动提交的，也就是说，只要执行一条 DML 语句就开启了事物，并且提交了事务

### 8.1 关闭 MySQL 的事务自动提交

START TRANSACTION

DML....

COMMIT|ROLLBACK



---

### 8.1.1 示例

向 emp3 表中添加一条数据，要求手动提交事务。

```
mysql> start transaction;
```

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

```
mysql> insert into emp3 values(default,"oldlu",default,default);
```

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

```
mysql> commit;
```

```
Query OK, 0 rows affected (0.01 sec)
```

## 六、 MySQL 查询数据

### 1 MySQL 的基本查询

#### 1.1 MySQL 的列选择

SELECT \* | 投影列 FROM 表名

##### 1.1.1 示例

查询 departments 表中的所有数据

```
select * from departments;
```

#### 1.2 MySQL 的行选择

SELECT \* | 投影列 FROM 表名 WHERE 选择条件

##### 1.2.1 示例

查询 departments 表中部门 ID 为 4 的部门名称与工作地点 ID。

```
select department_name,location_id from departments where department_id =4;
```

#### 1.3 SELECT 语句中的算术表达式

+ : 加法运算

- : 减法运算

\* : 乘法运算

/ : 除法运算，返回商

% : 求余运算，返回余数

---

### 1.3.1 示例一

修改 employees 表添加 salary。

```
alter table employees add column salary float(9,2);
```

### 1.3.2 示例二

查询雇员的年薪。

```
select employees_id,last_name,email,12*salary from employees;
```

### 1.3.3 示例三

计算 employees 表中的员工全年薪水加 100 以后的薪水是多少？

```
select employees_id,last_name,email,12*salary+100 from employees;
```

## 1.4 MySQL 中定义空值

包含空值的算术表达式计算结果为空。

### 1.4.1 示例

在 employees 中添加 commission\_pct，计算年薪包含佣金。

```
alter table employees add column commission_pct float(5,2);
```

```
select 12*salary*commission_pct from employees;
```

## 1.5 MySQL 中的列别名

SELECT 列名 AS 列别名 FROM 表名 WHERE 条件

### 1.5.1 示例

查询 employees 表将雇员 last\_name 列名改为 name。

```
select last_name as name from employees;
```

## 1.6 MySQL 中的连字符

MySQL 中并不支持||作为连字符，需要使用 concat 函数。在参数数量上与 oracle 的 concat 函数有区别。

### 1.6.1 示例

查询雇员表中的所有数据，将所有数据连接到一起，每列值中通过#分割。

```
select concat(employees_id,'#',last_name,'#',email,'#',salary,'#',commission_pct) from employees;
```

---

## 1.7 MySQL 中去除重复

在 SELECT 语句中用 DISTINCT 关键字除去相同的行。

### 1.7.1 示例

查询 employees 表，显示唯一的部门 ID。

```
select distinct dept_id from employees;
```

## 2 约束和排序数据

### 2.1 MySQL 中的比较条件

#### 2.1.1 比较运算符

- 等于=
- 大于>
- 大于等于>=
- 小于<
- 小于等于<=
- 不等于!=或<>

##### 2.1.1.1 示例一

查询 employees 表，员工薪水大于等于 3000 的员工的姓名与薪水。

```
select * from employees where salary >=3000;
```

##### 2.1.1.2 示例二

查询 employees 表，员工薪水不等于 5000 的员工的姓名与薪水。

```
select * from employees where salary <>5000;
```

#### 2.1.2 模糊查询

- like
- %表示任意多个任意字符
- \_表示一个任意字符

##### 2.1.2.1 示例

查询 employees 中雇员名字第二个字母是 e 的雇员信息。

```
select * from employees where last_name like '_e%'
```

---

## 2.1.3 逻辑运算符

- and
- or
- not

### 2.1.3.1 示例一

查询 employees 表中雇员薪水是 5000 的并且名字中含有 d 的雇员信息

```
select * from employees where salary = 5000 and last_name like '%e%'
```

### 2.1.3.2 示例二

查询 employees 表中雇员名字中不包含 u 的雇员信息

```
select * from employees where last_name not like '%u%'
```

## 2.1.4 范围查询

- between ... and
- in 表示在一个非连续的范围

### 2.1.4.1 示例一

查询 employees 表，薪水在 3000-8000 之间的雇员信息

```
select * from employees where salary between 3000 and 8000
```

### 2.1.4.2 示例二

查询 employees 表，找出薪水是 5000,6000,8000 的雇员信息

```
select * from employees where salary in(5000,6000,8000)
```

## 2.1.5 空值判断

- 判断空 is null
- 判断非空 is not null

### 2.1.5.1 示例一

找出 employees 表中那些没有佣金的雇员

```
select * from employees where commission_pct is null;
```

### 2.1.5.2 示例二

找出 employees 表中那些有佣金的雇员

```
select * from employees where commission_pct is not null;
```

---

## 2.2 使用 ORDER BY 排序

- 用 ORDER BY 子句排序
- ASC: 升序排序, 默认
- DESC: 降序排序

### 2.2.1.1 示例一

查询 employees 表中的所有雇员, 薪水按升序排序。

```
select * from employees order by salary
```

### 2.2.1.2 示例二

查询 employees 表中的所有雇员, 雇员名字按降序排序。

```
select * from employees order by last_name desc
```

## 3 MySQL 中常见的单行函数

### 3.1 大小写控制函数

LOWER(str) 转换大小写混合的字符串为小写字符串

UPPER(str) 转换大小写混合的字符串为大写字符串。

### 3.2 字符处理

CONCAT(str1,str2,...) 将 str1、str2 等字符串连接起来

SUBSTR(str,pos,len) 从 str 的第 pos 位 (范围: 1~str.length) 开始, 截取长度为 len 的字符串

LENGTH(str) 获取 str 的长度

INSTR(str,substr) 获取 substr 在 str 中的位置

LPAD(str,len,padstr)/RPAD(str,len,padstr)

TRIM(str) 从 str 中删除开头和结尾的空格 (不会处理字符串中间含有的空格)

LTRIM(str) 从 str 中删除左侧开头的空格

RTRIM(str) 从 str 中删除右侧结尾的空格

REPLACE(str,from\_str,to\_str) 将 str 中的 from\_str 替换为 to\_str (会替换掉所有符合 from\_str 的字符串)

### 3.3 数字函数

ROUND(arg1,arg2): 四舍五入指定小数的值。

ROUND(arg1): 四舍五入保留整数。

TRUNC(arg1,arg2): 截断指定小数的值, 不做四舍五入处理。

MOD(arg1,arg2): 取余。

---

### 3.4 日期函数

**SYSDATE()** 或者 **NOW()** 返回当前系统时间，格式为 YYYY-MM-DD hh-mm-ss

**CURDATE()** 返回系统当前日期，不返回时间

**CURTIME()** 返回当前系统中的时间，不返回日期

**DAYOFMONTH(date)** 计算日期 d 是本月的第几天

**DAYOFWEEK(date)** 日期 d 今天是星期几，1 星期日，2 星期一，以此类推

**DAYOFYEAR(date)** 返回指定年份的天数

**DAYNAME(date)** 返回 date 日期是星期几

**LAST\_DAY(date)** 返回 date 日期当月的最后一天

### 3.5 转换函数

**DATE\_FORMAT(date,format)** 将日期转换成字符串（类似 oracle 中的 to\_char()）

**STR\_TO\_DATE(str,format)** 将字符串转换成日期（类似 oracle 中的 to\_date()）



格式	描述
%a	缩写星期名
%b	缩写月名
%c	月, 数值
%D	带有英文前缀的月中的天
%d	月的天, 数值(00-31)
%e	月的天, 数值(0-31)
%f	微秒
%H	小时 (00-23)
%h	小时 (01-12)
%I	小时 (01-12)
%i	分钟, 数值(00-59)
%j	年的天 (001-366)
%k	小时 (0-23)
%l	小时 (1-12)
%M	月名
%m	月, 数值(00-12)
%p	AM 或 PM
%r	时间, 12-小时 ( hh:mm:ss AM 或 PM )
%S	秒(00-59)
%s	秒(00-59)
%T	时间, 24-小时 (hh:mm:ss)
%U	周 (00-53) 星期日是一周的第一天
%u	周 (00-53) 星期一是一周的第一天
%V	周 (01-53) 星期日是一周的第一天, 与 %X 使用
%v	周 (01-53) 星期一是一周的第一天, 与 %x 使用
%W	星期名
%w	周的天 ( 0=星期日, 6=星期六 )
%X	年, 其中的星期日是周的第一天, 4 位, 与 %V 使用
%x	年, 其中的星期一是周的第一天, 4 位, 与 %v 使用
%Y	年, 4 位
%y	年, 2 位

### 3.6 示例一

向 employees 表中添加 hire\_date 列 类型为 date 类型  
alter table employees add column hire\_date date

### 3.7 示例二

向 employees 表中添加一条数据, 名字: King , email: king@sxt.cn, 部门 ID: 1,

---

薪水: 9000, 入职时间: 2018 年 5 月 1 日, 佣金: 0.6  
insert into employees values(default,'King','king@sxt.cn',1,9000,0.6,STR\_TO\_DATE('2018 年 5 月 1 日','%Y 年%m 月%d 日'))

### 3.8 示例三

查询 employees 表中雇员名字为 King 的雇员的入职日期, 要求显示格式为 yyyy 年 MM 月 dd 日。

```
select DATE_FORMAT(hire_date,'%Y 年%m 月%d 日') from employees where last_name = 'King'
```

### 3.9 通用函数

**IFNULL(expr1,expr2)** 判断 expr1 是否为 null, 如果为 null, 则用 expr2 来代替 null (类似 oracle 的 NVL()函数)

**NULLIF(expr1,expr2)** 判断 expr1 和 expr2 是否相等, 如果相等则返回 null, 如果不相等则返回 expr1

**IF(expr1,expr2,expr3)** 判断 expr1 是否为真(是否不为 null), 如果为真, 则使用 expr2 替代 expr1; 如果为假, 则使用 expr3 替代 expr1 (类似 oracle 的 NVL2()函数)

**COALESCE(value,...)** 判断 value 的值是否为 null, 如果不为 null, 则返回 value; 如果为 null, 则判断下一个 value 是否为 null.....直至出现不为 null 的 value 并返回或者返回最后一个为 null 的 value

**CASE WHEN THEN ELSE END** 条件函数

## 4 多表连接查询

### 4.1 等值连接

#### 4.1.1 示例

查询雇员 King 所在的部门名称

```
select d.department_name from employees e,departments d where e.dept_id = d.department_id and e.last_name = 'King'
```

### 4.2 非等值连接

#### 4.2.1 示例一

创建 sal\_level 表, 包含 lowest\_sal, highest\_sal, level。

```
create table sal_level(lowest_sal int,highest_sal int ,level varchar(30))
```

---

### 4.2.2 示例二

插入数据

1000 2999 A

2000 4999 B

5000 7999 C

8000 12000 D

insert into sal\_level values(8000,12000,'D')

### 4.2.3 示例三

查询所有雇员的薪水级别。

select e.last\_name,s.level from employees e ,sal\_level s where e.salary between s.lowest\_sal and highest\_sal;

## 4.3 自连接

### 4.3.1 示例一

修改 employees 表，添加 manager\_id 列

ALTER table employees add COLUMN manager\_id int

### 4.3.2 示例二

修改数据 Oldlu 是 kevin 与 King 的经理

Taylor 是 Fox 的经理

### 4.3.3 示例三

查询每个雇员的经理的名字以及雇员的名字。

select emp.last\_name,man.last\_name from employees emp ,employees man where emp.manager\_id = man.employees\_id

## 5 外连接(OUTER JOIN)

### 5.1 左外连接(LEFT OUTER JOIN)

#### 5.1.1 示例一

向 employees 表中添加一条数据，名字：Lee，email：[lee@sxt.cn](mailto:lee@sxt.cn)，入职时间为今天。他没有薪水，没有经理，没有佣金。

insert into employees(last\_name,email,hire\_date) values('Lee','lee@sxt.cn',SYSDATE())

---

### 5.1.2 示例二

查询所有雇员的名字以及他们的部门名称，包含那些没有部门的雇员。

```
select e.last_name,d.department_name from employees e LEFT OUTER JOIN departments d  
on e.dept_id = d.department_id
```

## 5.2 右外连接(RIGHT OUTER JOIN)

### 5.2.1 示例一

向 departments 表中添加一条数据，部门名称为 Testing，工作地点 ID 为 5。

```
insert into departments values(default,'Testing',5)
```

### 5.2.2 示例二

查询所有雇员的名字以及他们的部门名称，包含那些没有雇员的部门。

```
select e.last_name,d.department_name from employees e right OUTER join departments d on  
e.dept_id = d.department_id;
```

## 5.3 全外链接

注意：MySQL 中不支持 FULL OUTER JOIN 连接

可以使用 union 实现全完连接。

### 5.3.1 UNION

可以将两个查询结果集合并，返回的行都是唯一的，如同对整个结果集使用了 DISTINCT。

### 5.3.2 UNION ALL

只是简单的将两个结果合并后就返回。这样，如果返回的两个结果集中有重复的数据，那么返回的结果集就会包含重复的数据了。

### 5.3.3 语法结构

```
SELECT 投影列 FROM 表名 LEFT OUTER JOIN 表名 ON 连接条件 UNION  
SELECT 投影列 FROM 表名 RIGHT OUTER JOIN 表名 ON 连接条件
```

#### 5.3.3.1 示例

查询所有雇员的名字以及他们的部门名称，包含那些没有雇员的部门以及没有部门的雇员。

```
(select e.last_name,d.department_name from employees e LEFT OUTER JOIN departments
```

---

```
d on e.dept_id = d.department_id) UNION (select e1.last_name,d1.department_name from employees e1 RIGHT OUTER JOIN departments d1 on d1.department_id = e1.dept_id)
```

## 6 SQL99 标准中的查询

MySQL5.7 支持 SQL99 标准。

### 6.1 SQL99 中的交叉连接(CROSS JOIN)

#### 6.1.1 示例

使用交叉连接查询 employees 表与 departments 表

```
select * from employees cross join departments
```

### 6.2 SQL99 中的自然连接(NATURAL JOIN)

#### 6.2.1 示例一

修改 employees 表中的 dept\_id 列将该列的名称修改为 department\_id

```
alter table employees change column dept_id department_id int
```

#### 6.2.2 示例二

使用自然连接查询所有有部门的雇员的名字以及部门名称。

```
select e.last_name,d.department_name from employees e natural join departments d
```

### 6.3 SQL99 中的内连接(INNER JOIN)

#### 6.3.1 示例

查询雇员名字为 OldLu 的雇员 ID，薪水与部门名称。

```
select e.employees_id,e.salary,d.department_name from employees e inner JOIN departments d on e.department_id = d.department_id where e.last_name = 'Oldlu';
```

## 7 聚合函数

### 7.1 AVG(arg)函数

对分组数据做平均值运算。

arg:参数类型只能是数字类型。

---

## 7.2 SUM(arg)函数

对分组数据求和。

arg:参数类型只能是数字类型。

## 7.3 MIN(arg)函数

求分组中最小数据。

arg:参数类型可以是字符、数字、日期。

## 7.4 MAX(arg)函数

求分组中最大数据。

arg:参数类型可以是字符、数字、日期。

## 7.5 COUNT 函数

返回一个表中的行数。

COUNT 函数有三种格式：

- COUNT(\*)
- COUNT(expr)
- COUNT(DISTINCT expr)

## 8 数据组(GROUP BY)

### 8.1 创建数据组

#### 8.1.1 示例

计算每个部门的平均薪水

```
select avg(e.salary) from employees e group by e.department_id
```

### 8.2 约束分组结果(HAVING)

显示那些最高薪水大于 5000 的部门的部门号和最高薪水。

```
select e.department_id,max(e.salary) from employees e group by e.department_id HAVING  
MAX(e.salary) > 5000
```

## 9 子查询

可以将子查询放在许多的 SQL 子句中，包括：

- WHERE 子句

- 
- HAVING 子句
  - FROM 子句

## 9.1 使用子查询的原则

- 子查询放在圆括号中。
- 将子查询放在比较条件的右边。
- 在单行子查询中用单行运算符，在多行子查询中用多行运算符。

### 9.1.1 示例

谁的薪水比 Oldlu 高

```
select em.last_name,em.salary from employees em where em.salary > (select e.salary from employees e where e.last_name = 'Oldlu')
```

## 9.2 单行子查询

运算符	含义
=	等于
>	大于
>=	大于或等于
<	小于
<=	小于或等于
<>	不等于

### 9.2.1 示例

查询 Oldlu 的同事，但是不包含他自己。

```
select empl.last_name from employees empl where empl.department_id = (select e.department_id from employees e where e.last_name = 'Oldlu') and empl.last_name <> 'Oldlu'
```

## 9.3 多行子查询

操作	含义
IN	等于列表中的任何成员
ANY	比较子查询返回的每个值
ALL	比较子查询返回的全部值

示例

查找各部门收入为部门最低的那些雇员。显示他们的名字，薪水以及部门 ID。

```
select em.last_name,em.salary,em.department_id from employees em where em.salary  
in(select min(e.salary) from employees e group by e.department_id)
```

## 10 MySQL 中的正则表达式

- MySQL 中允许使用正则表达式定义字符串的搜索条件，性能要高于 like。
- MySQL 中的正则表达式可以对整数类型或者字符类型检索。
- 使用 REGEXP 关键字表示正则匹配。
- 默认忽略大小写，如果要区分大小写，使用 BINARY 关键字

### 10.1 正则表达式的模式及其含义

模式	什么模式匹配
^	字符串的开始
\$	字符串的结尾
.	任何单个字符
[...]	在方括号内的任何字符列表
[^...]	非列在方括号内的任何字符
p1 p2 p3	交替匹配任何模式P1，P2或P3
*	零个或多个前面的元素
+	前面的元素的一个或多个实例
{n}	前面的元素的n个实例
{m, n}	m到n个实例前面的元素



---

## 10.2“^”符号

^在正则表达式中表示开始

### 10.2.1 语法

查询以 x 开头的数据(忽略大小写)

```
SELECT 列名 FROM 表名 WHERE 列名 REGEXP '^x';
```

### 10.2.2 示例

查询雇员表中名字是以 k 开头的雇员名字与薪水。

```
select last_name,salary from employees where last_name REGEXP binary '^K'
```

## 10.3“\$”符号

### 10.3.1 语法

查询以 x 结尾的数据(忽略大小写)

```
SELECT 列名 FROM 表名 WHERE 列名 REGEXP 'x$';
```

### 10.3.2 示例

查询雇员表中名字是以 n 结尾的雇员名字与薪水。

```
select last_name,salary from employees where last_name REGEXP binary 'n$'
```

## 10.4“.”符号

### 10.4.1 语法

英文的点，它匹配任何一个字符，包括回车、换行等。

```
SELECT 列名 FROM 表名 WHERE 列名 REGEXP 'x.';
```

### 10.4.2 示例

查询雇员表中名字含有 o 的雇员的姓名与薪水。

```
select last_name,salary from employees where last_name REGEXP 'o.'
```

## 10.5“\*”符号

### 10.5.1 语法

“\*”：星号匹配 0 个或多个字符，在它之前必须有内容。

---

## 10.6“+”符号

### 10.6.1 语法

“+”：加号匹配 1 个或多个字符，在它之前也必须要有内容。

SELECT 列名 FROM 表名 WHERE 列名 REGEXP 'x+';-匹配大于 1 个的任意字符

## 10.7“?”符号

### 10.7.1 语法

“?”：问号匹配 0 次或 1 次。

SELECT 列名 FROM 表名 WHERE 列名 REGEXP 'x?';-匹配 0 个或 1 个字符

## 10.8“|”符号

### 10.8.1 语法

“|”：表示或者含义

SELECT 列名 FROM 表名 WHERE 列名 REGEXP 'abc|bcd';;-匹配包含 abc 或 bcd

### 10.8.2 示例

查询雇员表中名字含有 ke 或者 lu 的雇员的名字与薪水。

select last\_name,salary from employees where last\_name REGEXP 'ke|lu'

## 10.9“[a-z]”

### 10.9.1 语法

“[a-z]”：字符范围

“^[...]”：以什么字符开头的

“[^...]”：匹配不包含在[]的字符

SELECT 列名 FROM 表名 WHERE 列名 REGEXP '[a-z]';-匹配内容包含 a-z 范围的数

据

### 10.9.2 示例一

查询雇员表中名字包含 x、y、z 字符的雇员的名字和薪水。

select last\_name,salary from employees where last\_name REGEXP '[x-z]'

---

### 10.9.3 示例二

查询雇员名字是 t、f 开头的雇员名字与薪水。

```
select last_name,salary from employees where last_name REGEXP '^[t|f]'
```

### 10.9.4 示例三

查询雇员的名字与薪水，不包含 oldlu。

```
select last_name,salary from employees where last_name REGEXP '[^oldlu]'
```

## 10.10 “{n}”

### 10.10.1 语法

“{n}”：固定次数。

```
select * from student where name REGEXP 's{2}';--匹配以 s 连续出现 2 次的所有数据
```

### 10.10.2 示例一

查询雇员名字含有连续两个 e 的雇员的姓名与薪水

```
select last_name,salary from employees where last_name REGEXP 'e{2}'
```

### 10.10.3 示例二

查询名字中含有两个 o 的雇员的名字与薪水。

```
select last_name,salary from employees where last_name REGEXP 'o.{2}'
```

## 10.11 “{n,m}”

### 10.11.1 语法

“{n,m}”：范围次数。

```
select * from student where name REGEXP '^s{2,5}';--匹配以 s 开头且重复 2 到 5 次的所有数据
```

### 10.11.2 示例

查询雇员名字中包含 1 个或者两个 o 的雇员姓名与薪水。

```
select last_name,salary from employees where last_name REGEXP 'o.{1,2}'
```

---

## 七、 MySQL 中的其他对象

### 1 索引

MySQL 索引的建立对于 MySQL 的高效运行是很重要的，索引可以大大提高 MySQL 的检索速度。

#### 1.1 MySQL 中的索引类型

- 普通索引
- 唯一索引
- 主键索引
- 组合索引
- 全文索引

#### 1.2 普通索引

是最基本的索引，它没有任何限制。

在创建索引时，可以指定索引长度。length 为可选参数，表示索引的长度，只有字符串类型的字段才能指定索引长度，如果是 BLOB 和 TEXT 类型，必须指定 length。

创建索引时需要注意：

如果指定单列索引长度，length 必须小于这个字段所允许的最大字符个数。

查询索引：SHOW INDEX FROM table\_name

##### 1.2.1 直接创建索引

```
CREATE INDEX index_name ON table(column(length))
```

###### 1.2.1.1 示例

为 emp3 表中的 name 创建一个索引，索引名为 emp3\_name\_index  
create index emp3\_name\_index ON emp3(name)

##### 1.2.2 修改表添加索引

```
ALTER TABLE table_name ADD INDEX index_name (column(length))
```

###### 1.2.2.1 示例

修改 emp3 表，为 addrees 列添加索引，索引名为 emp3\_address\_index  
alter table emp3 add index emp3\_address\_index(address)

---

### 1.2.3 创建表时指定索引列

```
CREATE TABLE `table` (  
    COLUMN TYPE ,  
    PRIMARY KEY (`id`),  
    INDEX index_name (column(length))  
)
```

#### 1.2.3.1 示例

创建 emp4 表，包含 emp\_id,name,address 列，同时为 name 列创建索引。索引名为 emp4\_name\_index

```
create table emp4(emp_id int primary key auto_increment,name varchar(30),address  
varchar(50),index emp4_name_index(name))
```

### 1.2.4 删除索引

```
DROP INDEX index_name ON table
```

#### 1.2.4.1 示例

删除 emp3 表中索引名为 emp3\_address\_index 的索引

```
drop index emp3_address_index on emp3
```

## 1.3 唯一索引

唯一索引与普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。它有以下几种创建方式：

### 1.3.1 创建唯一索引

```
CREATE UNIQUE INDEX indexName ON table(column(length))
```

#### 1.3.1.1 示例

为 emp 表中的 name 创建一个唯一索引，索引名为 emp\_name\_index

```
create unique index emp_name_index on emp(name)
```

### 1.3.2 修改表添加唯一索引

```
ALTER TABLE table_name ADD UNIQUE indexName (column(length))
```

#### 1.3.2.1 示例

修改 emp 表，为 salary 列添加唯一索引，索引名为 emp\_salary\_index

---

```
alter table emp add unique emp_salary_index(salary)
```

### 1.3.3 创建表时指定唯一索引

```
CREATE TABLE `table` (  
    COLUMN TYPE ,  
    PRIMARY KEY (`id`),  
    UNIQUE index_name (column(length))  
)
```

#### 1.3.3.1 示例

创建 emp5 表，包含 emp\_id,name,address 列，同时为 name 列创建唯一索引。索引名为 emp5\_name\_index

```
create table emp5(emp_id int primary key ,name varchar(30),address varchar(30),unique  
emp5_name_index(name))
```

## 1.4 主键索引

主键索引是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值。一般是在建表的时候同时创建主键索引。

### 1.4.1 修改表添加主键索引

```
ALTER TABLE 表名 ADD PRIMARY KEY(列名)
```

#### 1.4.1.1 示例

修改 emp 表为 employee\_id 添加主键索引

```
alter table emp add primary key(employee_id)
```

### 1.4.2 创建表时指定主键索引

```
CREATE TABLE `table` (  
    COLUMN TYPE ,  
    PRIMARY KEY(column)  
)
```

#### 1.4.2.1 示例

创建 emp6 表，包含 emp\_id,name,address 列，同时为 emp\_id 列创建主键索引

```
create table emp6(employee_id int primary key auto_increment,name varchar(20),address  
varchar(50))
```

---

## 1.5 组合索引

组合索引是指使用多个字段创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用(最左前缀原则)。

### 1.5.1 最左前缀原则

就是最左优先。

如：我们使用表中的 name, address, salary 创建组合索引，那么想要组合索引生效，我们只能使用如下组合：

name/address/salary

name/address

name/

如果使用 address/salary 或者是 salary 则索引不会生效。

### 1.5.2 修改添加组合索引

```
ALTER TABLE table_name ADD INDEX index_name (column(length),column(length))
```

#### 1.5.2.1 示例

修改 emp6 表，为 name, address 列创建组合索引

```
alter table emp6 add index emp6_index_n_a(name,address)
```

### 1.5.3 创建表时创建组合索引

```
CREATE TABLE `table` (  
    COLUMN TYPE ,  
    INDEX index_name (column(length),column(length))  
)
```

#### 1.5.3.1 示例

创建 emp7 表，包含 emp\_id,name,address 列，同时为 name,address 列创建组合索引。

```
create table emp7(emp_id int primary key auto_increment ,name varchar(20),address  
varchar(30),index emp7_index_n_a(name,address))
```

## 1.6 全文索引

全文索引(FULLTEXT INDEX)主要用来查找文本中的关键字，而不是直接与索引中的值相比较。FULLTEXT 索引跟其它索引大不相同，它更像是一个搜索引擎，而不是简单的 where 语句的参数匹配。FULLTEXT 索引配合 match against 操作使用，而不是一般的 where 语句加 like。

---

全文索引可以从 CHAR、VARCHAR 或 TEXT 列中作为 CREATE TABLE 语句的一部分被创建，或是随后使用 ALTER TABLE 添加。不过切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法。

### 1.6.1 修改添加全文索引

```
ALTER TABLE table_name ADD FULLTEXT index_content(content)
```

#### 1.6.1.1 示例一

修改 emp7 表添加 content 列类型为 TEXT

```
alter table emp7 add COLUMN content text;
```

#### 1.6.1.2 示例二

修改 emp7 表，为 content 列创建全文索引

```
alter table emp7 add fulltext emp7_content_fullindex(content)
```

### 1.6.2 创建表时创建全文索引

```
CREATE TABLE `table` (  
    COLUMN TYPE ,  
    FULLTEXT index_name (column)  
)
```

#### 1.6.2.1 示例

创建 emp8 表包含 emp\_id 列，content 列该列类型为 text，并为该列添加名为 emp8\_content\_fulltext 的全文索引

```
create table emp8(emp_id int primary key auto_increment,content text,fulltext  
emp8_content_fullindex(content))
```

### 1.6.3 删除全文索引

```
DROP INDEX index_name ON table  
ALTER TABLE table_name DROP INDEX index_name
```

#### 1.6.3.1 示例

删除 emp8 表中名为 emp8\_content\_full 的索引。

```
drop index emp8_content_fullindex on emp8
```

## 1.7 使用全文索引

全文索引的使用与其他索引不同。在查询语句中需要使用 match(column) against('content') 来检索数据。



---

## 1.7.1全文解析器

全文索引中基本单位是”词”。分词，全文索引是以词为基础的，MySQL 默认的分词是所有非字母和数字的特殊符号都是分词符。在检索数据时我们给定的检索条件也是词。

MySQL 中默认的全文解析器不支持中文分词。如果数据含有中文需要更换全文解析器 NGRAM。

## 1.7.2使用全文索引

SELECT 投影列 FROM 表名 WHERE MATCH(全文索引列名) AGAINST(‘搜索内容’)

### 1.7.2.1 示例一

修改 emp8 表，为 content 列创建名为 emp8\_content\_full 的全文索引

```
alter table emp8 add fulltext emp8_content_full(content)
```

### 1.7.2.2 示例二

向 emp8 表中插入一条数据 content 的值为”hello,bjsxt”

```
insert into emp8 values(default , 'hello bjsxt')
```

### 1.7.2.3 示例三

查询 emp8 表中内容包含 bjsxt 的数据。

```
select * from emp8 where match(content) AGAINST('bjsxt')
```

## 1.7.3更换全文解析器

在创建全文索引时可以指定 ngram 解析器

```
ALTER TABLE table_name ADD FULLTEXT index_content(content) WITH PARSER NGRAM
```

### 1.7.3.1 示例一

删除 emp8 表中的 emp8\_content\_full 全文索引

```
drop index emp8_content_full on emp8
```

### 1.7.3.2 示例二

修改 emp8 表，为 content 列添加名称为 emp8\_content\_full 的全文索引，并指定 ngram 全文解析器。

```
alter table emp8 add fulltext emp8_content_full(content) with parser ngram
```

### 1.7.3.3 示例三

向 emp8 表中添加一条数据 content 值为”你好，北京尚学堂”

```
insert into emp8 values(default , '你好，北京尚学堂')
```

---

#### 1.7.3.4 示例四

查询 emp8 表中内容包含”北京尚学堂”的数据

```
select * from emp8 where match(content) AGAINST('北京尚学堂')
```

## 2 MySQL 中的用户管理

MySQL 是一个多用户的数据库系统，按权限，用户可以分为两种：root 用户，超级管理员，和由 root 用户创建的普通用户。

### 2.1 MySQL 创建用户

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

### 2.2 查看用户

```
SELECT USER,NOST FROM USER(该表位于 mysql 库中)
```

#### 2.2.1 示例

创建一个 u\_sxt 的用户，并查看创建是否成功。

```
select user,host from mysql.user
```

```
create user u_sxt IDENTIFIED by 'sxt'
```

### 2.3 分配权限

新用户创建完后是无法登陆的，需要分配权限。

```
GRANT 权限 ON 数据库.表 TO 用户名@登录主机 IDENTIFIED BY "密码"
```

```
GRANT ALL PRIVILEGES ON *.* TO 'username'@'localhost' IDENTIFIED BY 'password'
```

登陆主机：

%            匹配所有主机

localhost    localhost 不会被解析成 IP 地址，直接通过 UNIXsocket 连接

127.0.0.1    会通过 TCP/IP 协议连接，并且只能在本机访问；

::1            ::1 就是兼容支持 ipv6 的，表示同 ipv4 的 127.0.0.1

#### 2.3.1 权限列表

ALTER: 修改表和索引。

CREATE: 创建数据库和表。

DELETE: 删除表中已有的记录。

DROP: 删除数据库和表。

INDEX: 创建或删除索引。

INSERT: 向表中插入新行。

SELECT: 检索表中的记录。

UPDATE: 修改现存表记录。

FILE: 读或写服务器上的文件。  
PROCESS: 查看服务器中执行的线程信息或杀死线程。  
RELOAD: 重载授权表或清空日志、主机缓存或表缓存。  
SHUTDOWN: 关闭服务器。  
ALL: 所有权限, ALL PRIVILEGES 同义词。  
USAGE: 特殊的 "无权限" 权限

### 2.3.2 示例

为 u\_sxt 用户分配只能查询 bjsxt 库中的 employees 表, 并且只能在本机登陆的权限。  
grant select ON bjsxt.employees to 'u\_sxt'@'localhost' IDENTIFIED by 'sxt'

## 2.4 刷新权限

每当调整权限后, 通常需要执行以下语句刷新权限  
FLUSH PRIVILEGES

## 2.5 删除用户

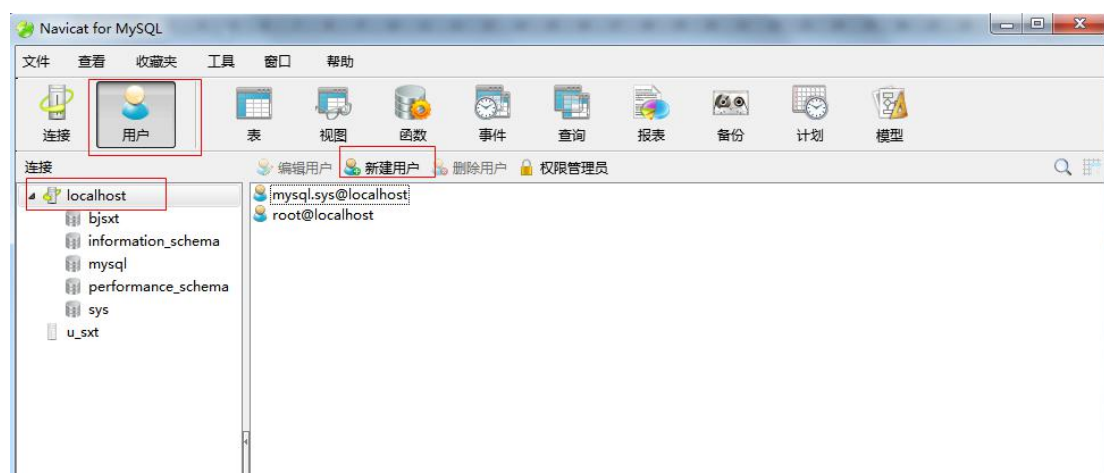
DROP USER username@localhost

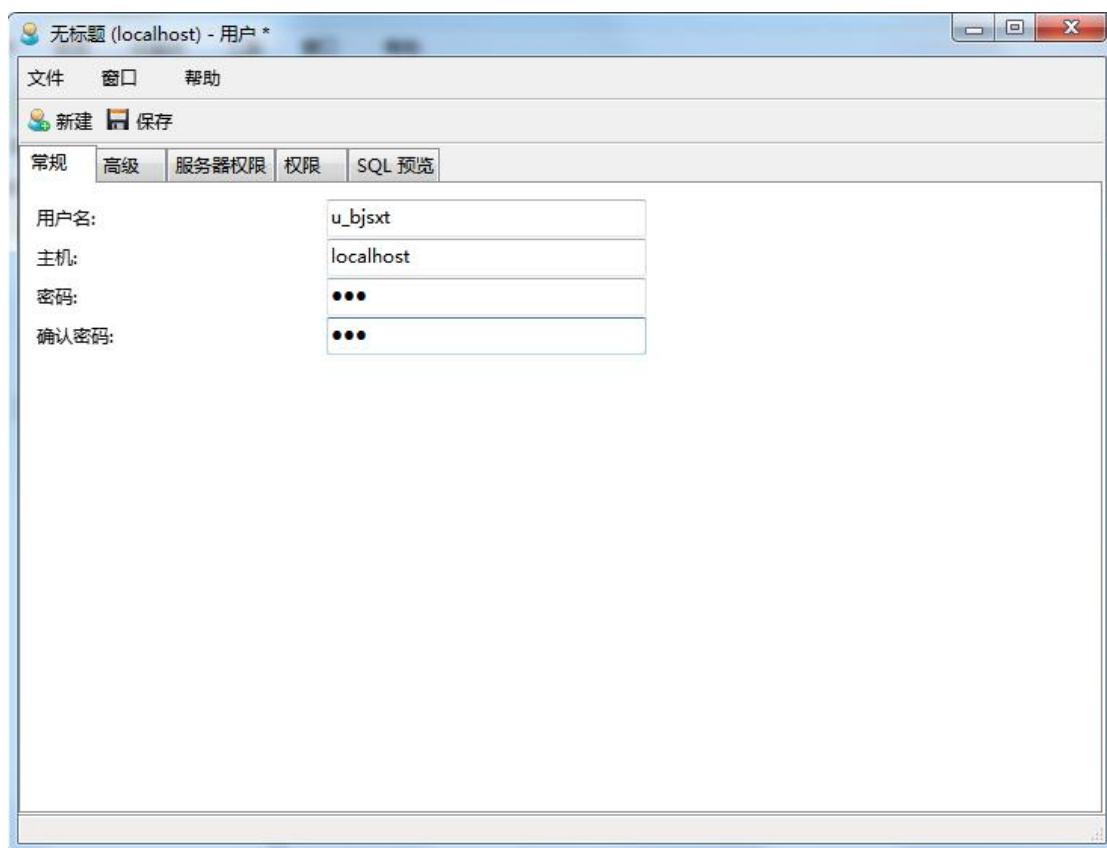
### 2.5.1 示例

删除 u\_sxt 用户  
drop user 'u\_sxt'@'localhost'

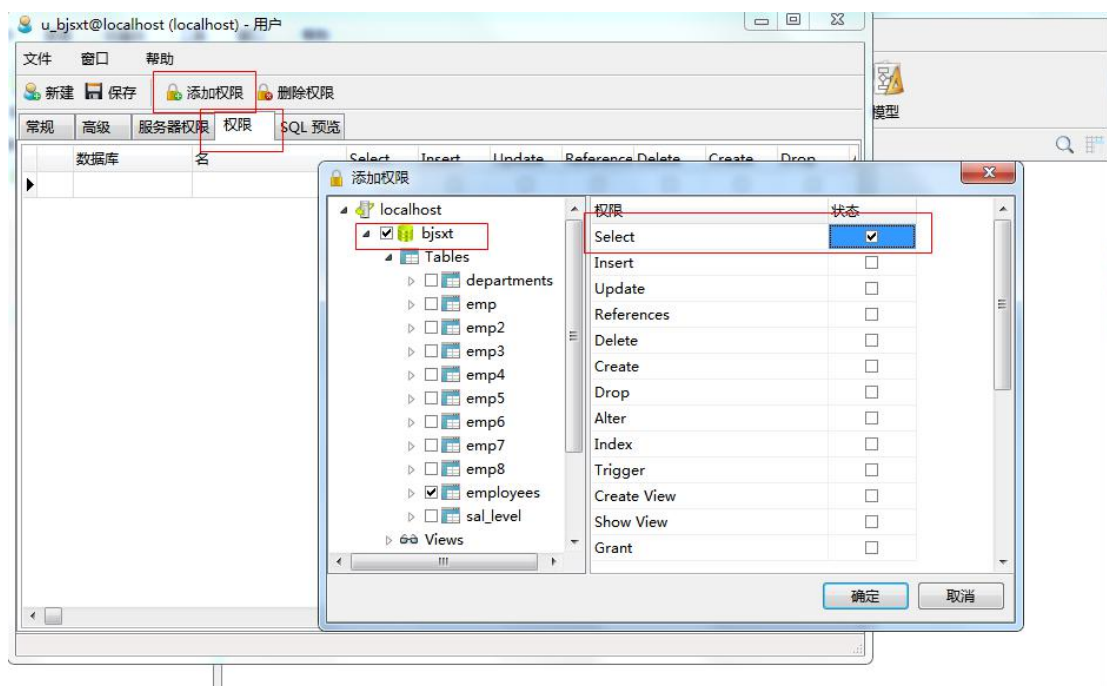
## 3 通过 Navicat 工具管理用户

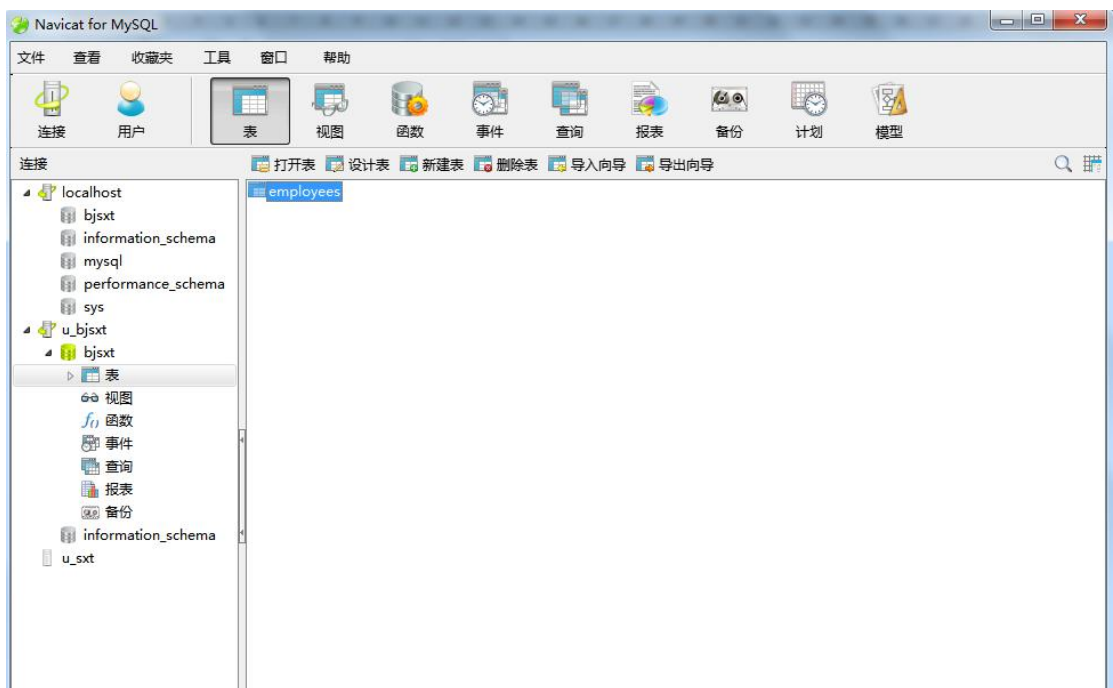
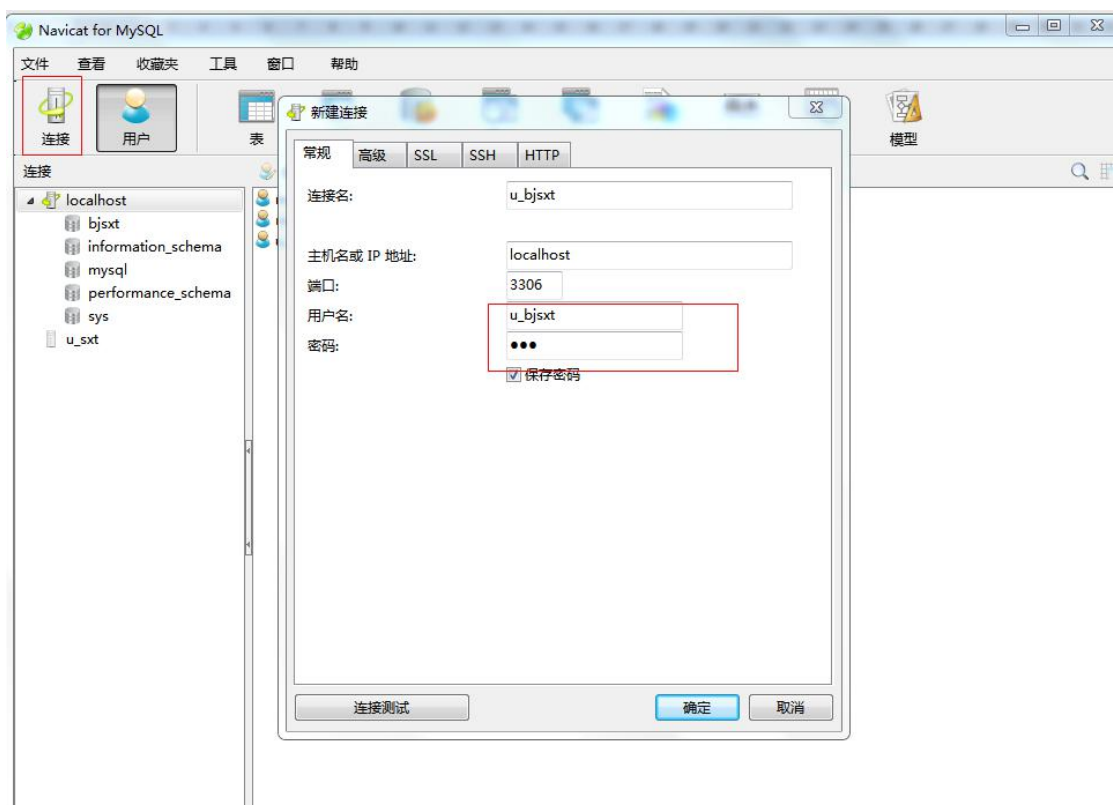
### 3.1 创建用户



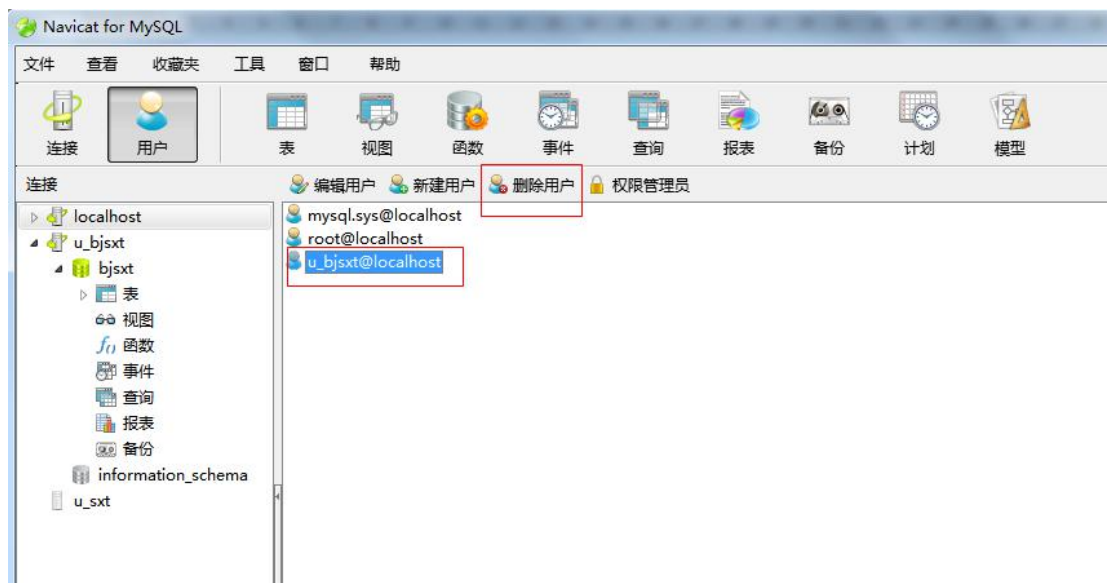


### 3.2 分配权限





### 3.3 删除用户



## 八、 MySQL 分页查询

MySQL 分页查询原则

- 在 MySQL 数据库中使用 LIMIT 子句进行分页查询。
- MySQL 分页中开始位置为 0。
- 分页子句在查询语句的最后侧。

### 1 LIMIT 子句

#### 1.1 语法格式

SELECT 投影列 FROM 表名 WHERE 条件 ORDER BY LIMIT 开始位置, 查询数量。

##### 1.1.1 示例

查询雇员表中所有数据按 id 排序，实现分页查询，每次返回两条结果。

```
select * from employees order by employees_id limit 0,2
```

### 2 LIMIT OFFSET 子句

#### 2.1 语法格式

SELECT 投影列 FROM 表名 WHERE 条件 ORDER BY LIMIT 查询数量 OFFSET 开始位置。

---

### 2.1.1 示例

查询雇员表中所有数据按 id 排序，使用 LIMIT OFFSET 实现分页查询，每次返回两条结果。

```
select * from employees order by employees_id limit 2 offset 4
```

## 九、MySQL 中的执行计划

### 1 MySQL 执行计划

在 MySQL 中可以通过 explain 关键字模拟优化器执行 SQL 语句，从而知道 MySQL 是如何处理 SQL 语句的。

### 2 MySQL 整个查询执行过程

- 客户端向 MySQL 服务器发送一条查询请求
- 服务器首先检查查询缓存，如果命中缓存，则立刻返回存储在缓存中的结果。否则进入下一阶段
- 服务器进行 SQL 解析、预处理、再由优化器生成对应的执行计划
- MySQL 根据执行计划，调用存储引擎的 API 来执行查询
- 将结果返回给客户端，同时缓存查询结果

### 3 启动执行计划

EXPLAIN SELECT 投影列 FROM 表名 WHERE 条件

### 4 EXPLAIN 列的解释

#### 4.1 ID

查询执行顺序：

id 值相同时表示从上向下执行

id 值相同被视为一组

如果是子查询，id 值会递增，id 值越高，优先级越高

#### 4.2 select\_type

simple:表示查询中不包含子查询或者 union

primary:当查询中包含任何复杂的子部分，最外层的查询被标记成 primary

derived:在 from 的列表中包含的子查询被标记成 derived

subquery:在 select 或 where 列表中包含子查询，则子查询被标记成 subquery

union:两个 select 查询时前一个标记为 PRIMARY，后一个标记为 UNION。union 出现在 from 从句子查询中，外层 select 标记为 PRIMARY，union 中第一个查询为 DERIVED，

---

第二个子查询标记为 UNION

unionresult: 从 union 表获取结果的 select 被标记成 union result 。

## 4.3 table

显示这一行的数据是关于哪张表的。

## 4.4 type

这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为 system、const、eq\_reg、ref、range、index 和 ALL。

system: 表中只有一行数据。属于 const 的特例。如果物理表中就一行数据为 ALL

const : 查询结果最多有一个匹配行。因为只有一行，所以可以被视为常量。const 查询速度非常快，因为只读一次。一般情况下把主键或唯一索引作为唯一条件的查询都是 const

eq\_ref: 查询时查询外键表全部数据。且只能查询主键列或关联列。且外键表中外键列中数据不能有重复数据，且这些数据都必须在主键表中有对应数据（主键表中数据可以有有没有用到的）

ref: 相比 eq\_ref，不对外键列有强制要求，里面的数据可以重复，只要出现重复的数据取值就是 ref。也可能是索引查询。

range: 把这个列当作条件只检索其中一个范围。常见 where 从句中出现 between、<、in 等。主要应用在具有索引的列中

index: 这个连接类型对前面的表中的每一个记录联合进行完全扫描（比 ALL 更好，因为索引一般小于表数据）。

ALL: 这个连接类型对于前面的每一个记录联合进行完全扫描，这一般比较糟糕，应该尽量避免。

## 4.5 possible\_keys

查询条件字段涉及到的索引，可能没有使用。

## 4.6 Key

实际使用的索引。如果为 NULL，则没有使用索引。

## 4.7 key\_len

表示索引中使用的字节数，查询中使用的索引的长度（最大可能长度），并非实际使用长度，理论上长度越短越好。key\_len 是根据表定义计算而得的，不是通过表内检索出的。

## 4.8 ref

显示索引的哪一列被使用了，如果可能的话，是一个常量 const。



## 4.9rows

根据表统计信息及索引选用情况,大致估算出找到所需的记录所需要读取的行数。

## 4.10Fitered

显示了通过条件过滤出的行数的百分比估计值。

## 4.11extra

MYSQL 如何解析查询的额外信息。

Distinct:MySQL 发现第 1 个匹配行后,停止为当前的行组合搜索更多的行。

Not exists:MySQL 能够对查询进行 LEFT JOIN 优化,发现 1 个匹配 LEFT JOIN 标准的行后,不再为前面的的行组合在该表内检查更多的行。

range checked for each record (index map: #):MySQL 没有发现好的可以使用的索引,但发现如果来自前面的表的列值已知,可能部分索引可以使用。

Using filesort:MySQL 需要额外的一次传递,以找出如何按排序顺序检索行。

Using index:从只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的列信息。

Using temporary:为了解决查询,MySQL 需要创建一个临时表来容纳结果。

Using where:WHERE 子句用于限制哪一个行匹配下一个表或发送到客户。

Using sort\_union(...), Using union(...), Using intersect(...): 这些函数说明如何为 index\_merge 联接类型合并索引扫描。

Using index for group-by:类似于访问表的 Using index 方式,Using index for group-by 表示 MySQL 发现了一个索引,可以用来查询 GROUP BY 或 DISTINCT 查询的所有列,而不要额外搜索硬盘访问实际的表。

## 十、 MySQL 数据库存储引擎介绍

### 1 查看 MySQL 数据库中的数据库存储引擎

#### 1.1查看数据库引擎

SHOW ENGINES

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, row-level locking	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory; no data loss	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine; events not supported	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO

---

## 2 MySQL 数据库引擎介绍

### 2.1 ISAM(Indexed Sequential Access Method)

ISAM 是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM 执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM 的两个主要不足之处在于，它不支持事务处理，也不能够容错。如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把 ISAM 用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MYSQL 能够支持这样的备份应用程序。

注意：使用 ISAM 时必须经常备份所有实时数据。

### 2.2 MyISAM

MyISAM 是 MySQL 的 ISAM 扩展格式和缺省的数据库引擎。除了提供 ISAM 里所没有的索引和字段管理的大量功能，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，其代价是你需要经常运行 OPTIMIZE TABLE 命令，来恢复被更新机制所浪费的空间。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。MYISAM 强调了快速读取操作，这可能就是为什么 MySQL 受到了 WEB 开发如此青睐的主要原因：在 WEB 开发中你所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 平台提供商只允许使用 MYISAM 格式。MyISAM 格式的一个重要缺陷就是不能在表损坏后恢复数据。

注意：MyISAM 引擎使用时必须经常使用 Optimize Table 命令清理空间；必须经常备份所有实时数据。工具有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。

如果使用该数据库引擎，会生成三个文件：

- .frm:表结构信息
- .MYD:数据文件
- .MYI:表的索引信息

### 2.3 InnoDB

InnoDB 数据库引擎都是造就 MySQL 灵活性的技术的直接产品，这项技术就是 MYSQL++ API。在使用 MYSQL 的时候，你所面对的每一个挑战几乎都源于 ISAM 和 MyISAM 数据库引擎不支持事务处理（transaction process）也不支持外键。尽管要比 ISAM 和 MyISAM 引擎慢很多，但是 InnoDB 包括了对事务处理和外来键的支持，这两点都是前两个引擎所没有的。如前所述，如果你的设计需要这些特性中的一者或者两者，那你就要被迫使用后两个引擎中的一个了。

MySQL 官方对 InnoDB 是这样解释的：InnoDB 给 MySQL 提供了具有提交、回滚和崩溃恢复能力的事务安全（ACID 兼容）存储引擎。InnoDB 锁定在行级并且也在 SELECT 语句提供一个 Oracle 风格一致的非锁定读，这些特色增加了多用户部署和性能。没有在 InnoDB 中扩大锁定的需要，因为在 InnoDB 中行级锁定适合非常小的空间。InnoDB 也支持 FOREIGN

---

KEY 强制。在 SQL 查询中，你可以自由地将 InnoDB 类型的表与其它 MySQL 的表的类型混合起来，甚至在同一个查询中也可以混合。

InnoDB 是为处理巨大数据量时的最大性能设计，它的 CPU 效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。

InnoDB 存储引擎被完全与 MySQL 服务器整合，InnoDB 存储引擎为主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB 存储它的表&索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这与 MyISAM 表不同，比如在 MyISAM 表中每个表被存在分离的文件中。InnoDB 表可以是任何尺寸，即使在文件尺寸被限制为 2GB 的操作系统上。

### innodb 与 myisam 区别

1. InnoDB 支持事务，MyISAM 不支持，对于 InnoDB 每一条 SQL 语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条 SQL 语言放在 begin 和 commit 之间，组成一个事务；

2. InnoDB 支持外键，而 MyISAM 不支持。对一个包含外键的 InnoDB 表转为 MYISAM 会失败；

3. InnoDB 是聚集索引，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此，主键不应该过大，因为主键太大，其他索引也都会很大。而 MyISAM 是非聚集索引，数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。

4. InnoDB 不保存表的具体行数，执行 select count(\*) from table 时需要全表扫描。而 MyISAM 用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；

5. InnoDB 不支持全文索引，而 MyISAM 支持全文索引，查询效率上 MyISAM 要高；(在 MySQL5.7 版本中已经支持全文索引)

如何选择：

1. 是否要支持事务，如果要请选择 innodb，如果不需要可以考虑 MyISAM
2. 如果表中绝大多数都只是读查询，可以考虑 MyISAM，如果既有读写也挺频繁，请使用 InnoDB。
3. 系统奔溃后，MyISAM 恢复起来更困难，能否接受；
4. MySQL5.5 版本开始 InnoDB 已经成为 Mysql 的默认引擎(之前是 MyISAM)，说明其优势是有目共睹的，如果你不知道用什么，那就用 InnoDB，至少不会差。

## 3 修改数据库级引擎

修改 MySQL 的 my.ini 配置文件

C:\ProgramData\MySQL\MySQL Server 5.7

default-storage-engine=数据库引擎名称

重启 MySQL

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	YES	Supports transactions, ro	YES	YES	YES
MRG_MYISAM	YES	Collection of identical My	NO	NO	NO
MEMORY	YES	Hash based, stored in m	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine	NO	NO	NO
MyISAM	DEFAULT	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO

只读 查询时间: 0.000s 第 1 条记录 (共 9 条)

## 4 修改表级存储引擎

ALTER TABLE tableName engine=InnoDB

查询表的存储引擎

show create table table\_name;