

第一章、认识MySQL

目标:

- 1、数据库简介
- 2、MySQL环境搭建
- 3、客户端连接MySQL服务
- 4、图形化客户端连接MySQL

一、数据库简介

1、为什么需要数据库

持久化数据：把数据保存到可掉电式存储设备中，以供之后使用

2、基本概念

- a、数据 (Data)：广义的理解数据，它包含很多的种类，如文字、图形、图像、声音以及学生的档案记录等，这些都是数据
- b、数据库 (DataBase)：是按照数据结构来组织、存储和管理数据的仓库，是一个长期存储在计算机内的有组织、可共享统一管理的数据集合。
- c、数据库管理系统 (DBMS)：实现对数据库资源进行组织，管理和存取的系统软件。如mysql, sqlserver, oracle, 达梦等
- d、数据库系统：为方便终端用户，需要使用定制的、更为简洁的应用程序 (Application Program) 来利用数据库，这些应用程序称为数据库应用程序

3、常见的数据库模型

- a、层次型
- b、网络型
- c、关系型

4、国内外数据库

- a、国外：oracle、sqlserver、db2、mysql、sybase
- b、国内：TiDB、DM、Oceanbase、GBase

二、MySQL环境搭建

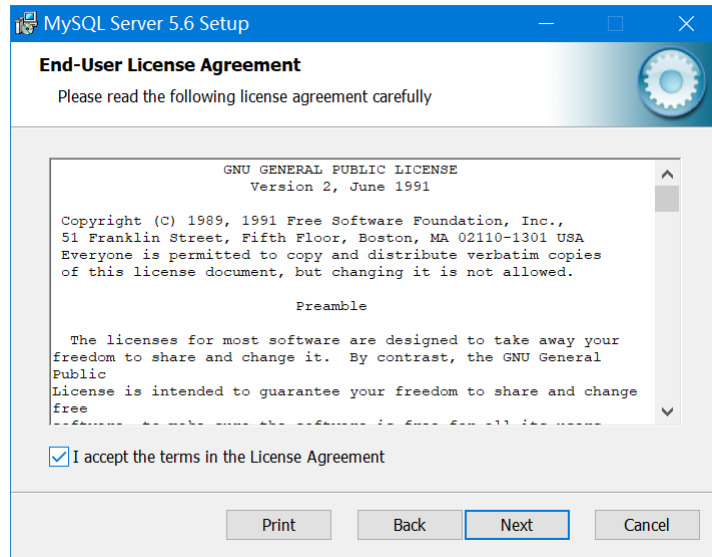
以下步骤使用MySQL5.6版本

1) 安装MySQL

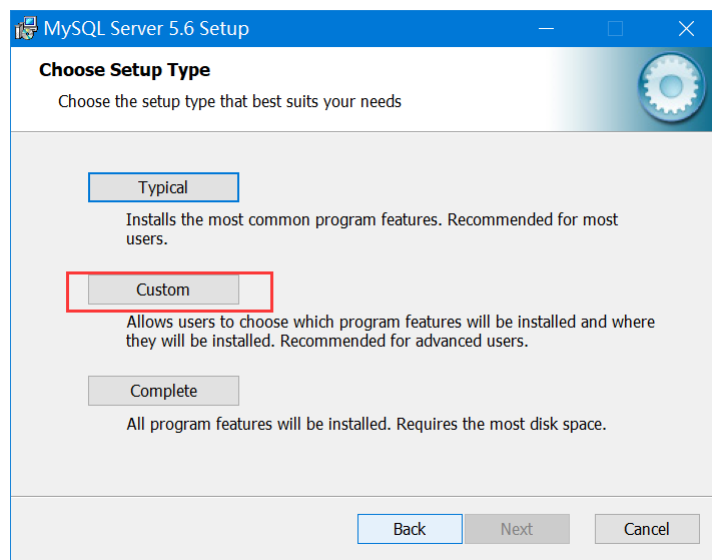
注意：以msi结尾的文件是可以直接双击安装；以zip结尾的文件需解压后，使用命令安装。

- 1、双击下发的文件“mysql-5.6.5-win32.msi”，或在官网下载安装msi的安装文件。

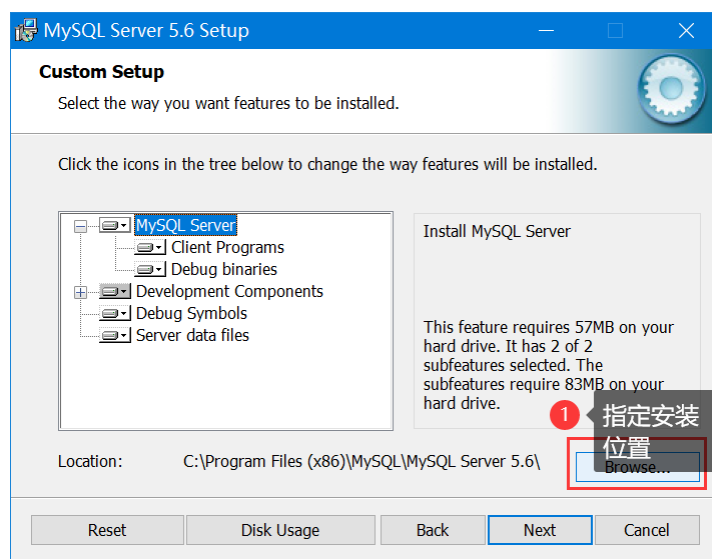
2、勾选"I accept ..."

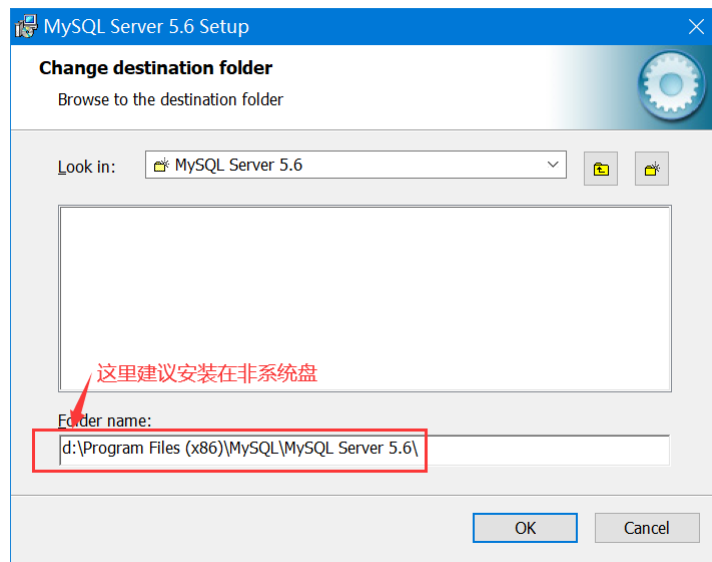


3、单击"custom"



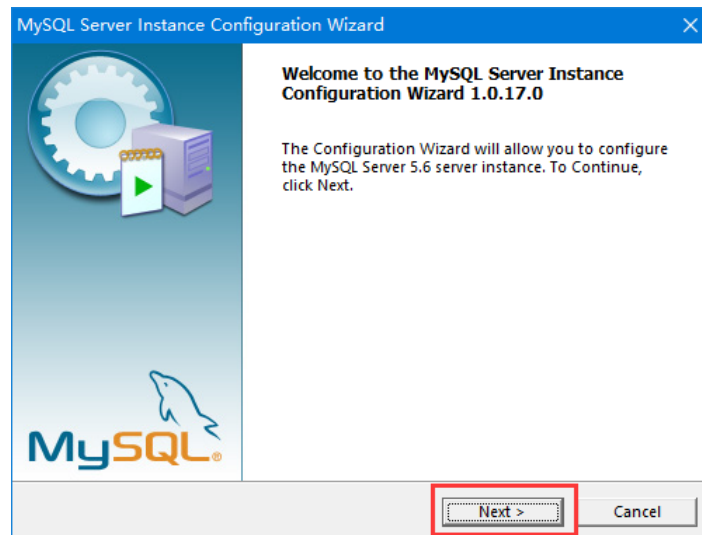
4、指定安装位置



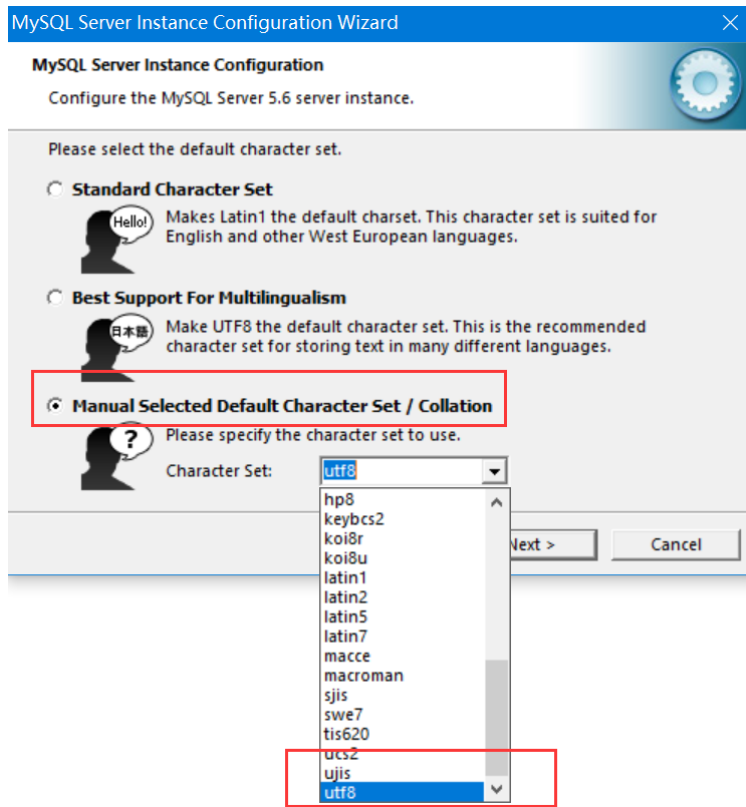


5、依次单击“Finish”，“Install”，“Finish”

6、安装完成后，会进入MySQL服务器的配置界面，单击“Next”

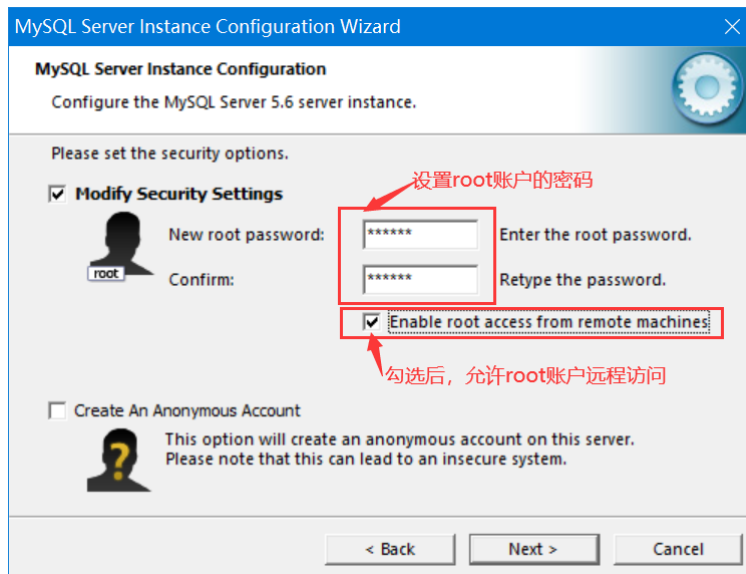


7、依次采用默认配置，单击“Next”，在以下界面注意选择第三个选项，该界面用于配置默认的**字符编码**

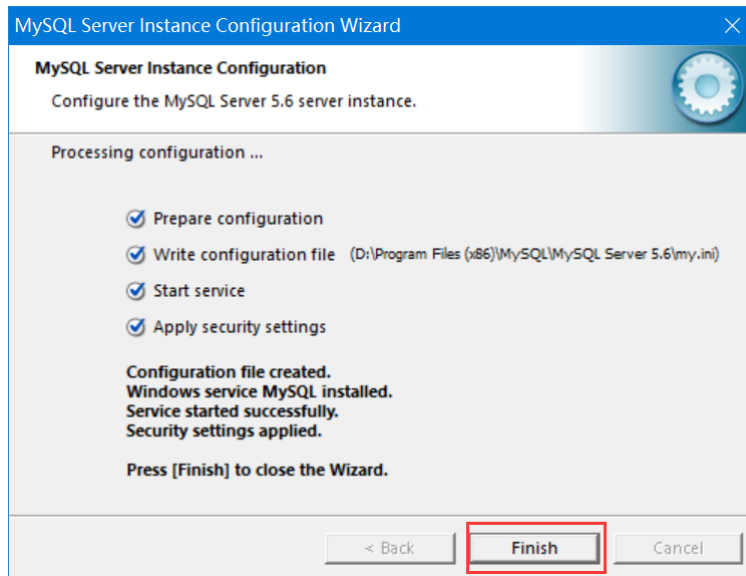


8、设置MySQL的服务名称，保持默认即可。勾选导入bin目录的路径。

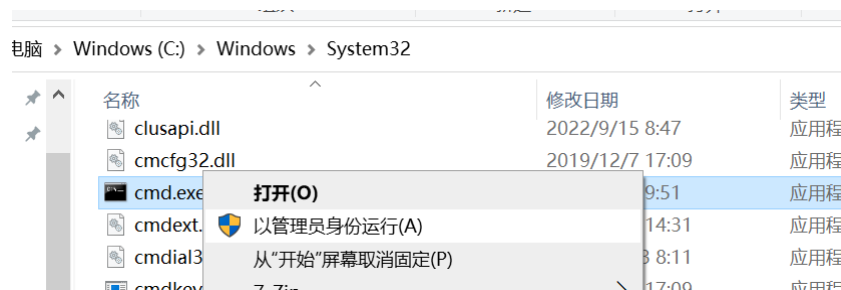
9、设置root账号的密码，并允许远程访问



10、单击“Execute”执行配置，成功后如下图，单击Finish完成配置



11、以管理员身份打开命令行提示符工具，启动或停止mysql服务。该"cmd.exe"工具"C:\Windows\System32"目录下

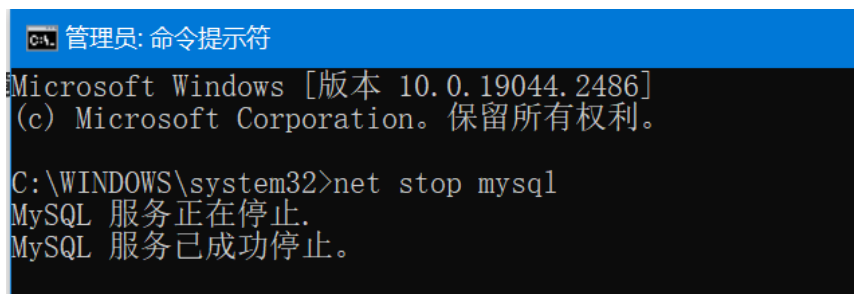


启动和停止的命令：



2) 卸载MySQL步骤

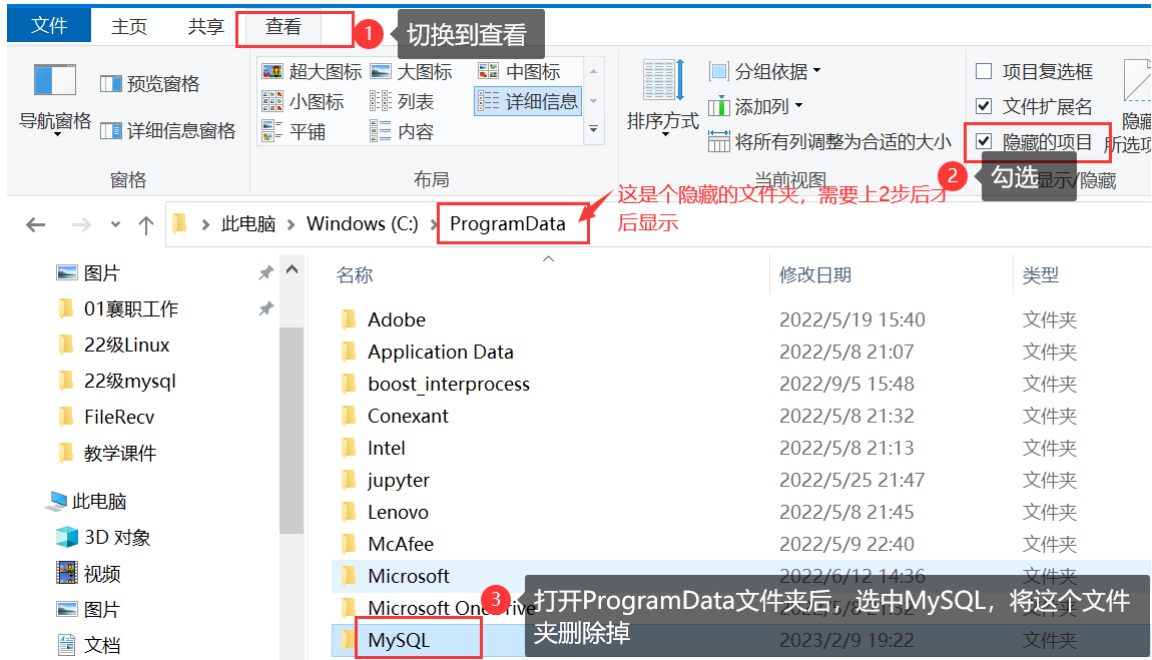
- 1、以管理员身份打开“命令行提示符”工具。
- 2、在“命令行提示符”中输入命令，停止后台MySQL服务



- 3、输入命令“sc delete mysql”卸载MySQL

```
C:\WINDOWS\system32>sc delete mysql
[SC] DeleteService 成功
```

4、删除"C:\ProgramData\MySQL"文件夹，注意ProgramData是隐藏文件夹



5、清空注册表

按下win+r，在运行中输入"regedit"，按以下顺序找到MySQL节点，删除该节点

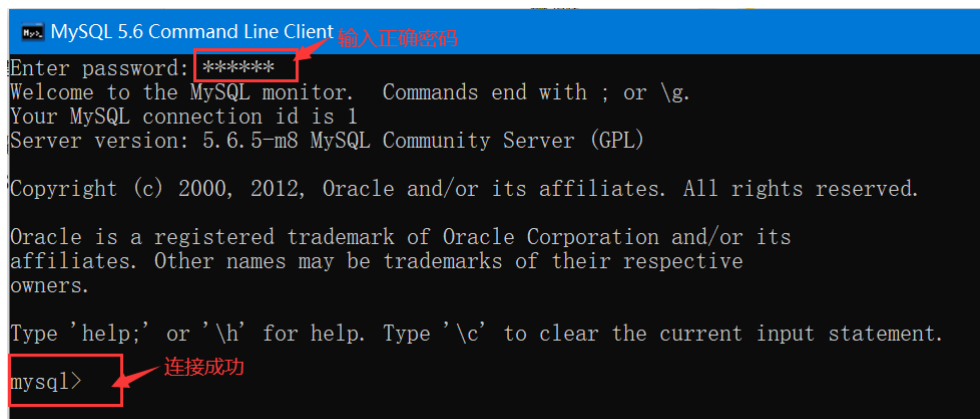
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\EventLog\Application\MySQL

三、客户端连接MySQL服务

1、按windows键，在“开始”菜单中找到“MySQL”文件夹，单击“MySQL5.6 Command Line Client”



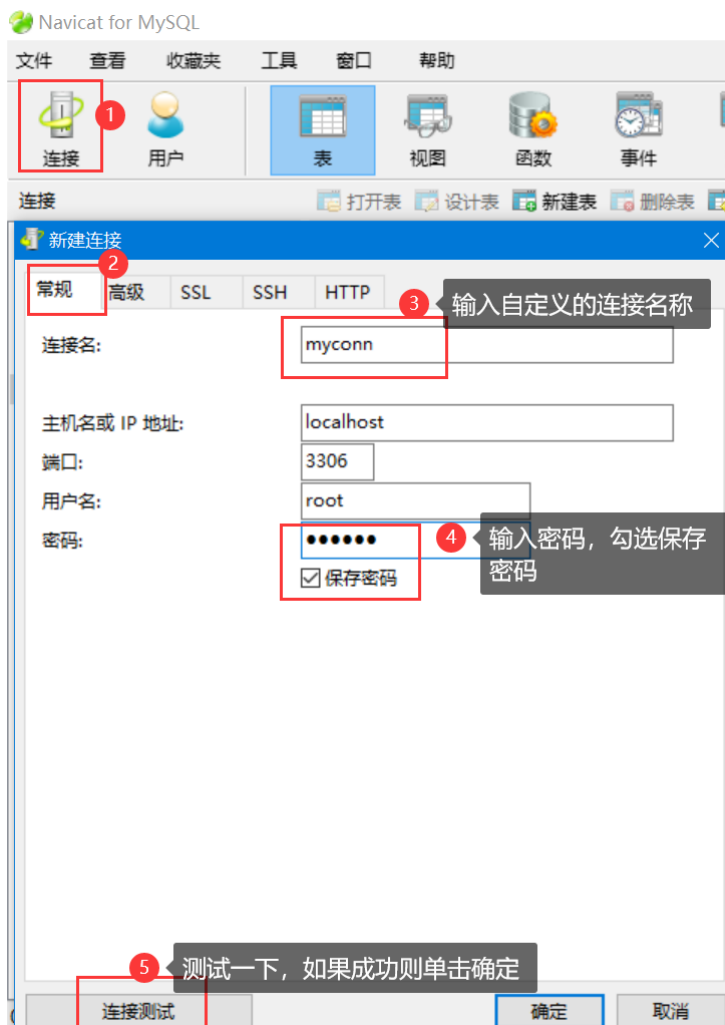
输入正确密码后，出现下列界面，则说明连接MySQL服务器成功



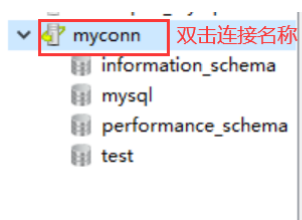
注意：如果mysql的服务未开启，输入密码后，客户端会闪退，这时需要启动服务，参考上面步骤

四、图形化工具连接MySQL服务

1、使用Navicat for MySQL 连接MySQL



2、双击连接“myconn”，打开连接



二、创建和管理MySQL数据库与表

SQL: Structured Query Language的简称，结构化查询语言

SQL包含4个部分：

DDL：数据定义语言，包括create、drop、alter等

DML：数据操作语言，包括insert、update、delete等

DQL：数据查询语言，包括select语句

DCL：数据控制语言，包括grant、revoke、commit、rollback等

1、查看所有数据库

a、查看数据库：`show databases;`

b、系统数据库：

information_schema：提供了访问数据库元数据的方式。保存了其他数据库的信息，如数据库名，数据表，列的数据类型或访问权限等

mysql：是MySQL的核心数据库，主要负责存储数据库的用户、权限设置、关键字等

performance_schema：主要用于收集数据库服务器性能参数

2、创建数据库

1) 使用语句创建

语法：

```
create database [if not exists] 数据库名
[[default] character set 字符集名]
[[default] collate 排序规则名];
```

例：创建学生数据库

```
create database studb
default character set utf8
default collate utf8_general_ci;
```

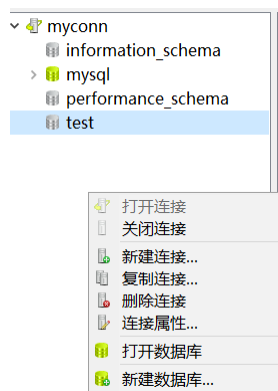
补充：

查看某个数据库的创建信息

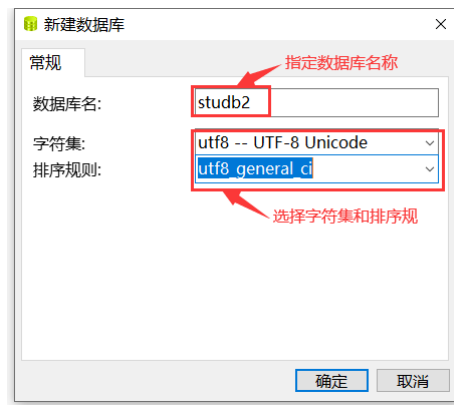
`show create database 数据库名;`

2) 使用图形工具创建

a、在navicat中空白外右键，选择“新建数据库”



b、在弹出的窗口中，输入新数据库名称，指定字符集和排序规则



c、在左侧窗口空白处右键，“刷新”，如果数据库的颜色是灰色，则是未打开状态，双击打开后呈绿色

3、管理数据库

1) 使用语句修改数据库字符集

语法:

```
alter database 数据库名  
[[default] character set 字符集名]  
[[default] collate 排序规则名];
```

例: 修改mydb的字符集和排序规则

```
-- 创建数据库, 采用默认的字符集和排序规则  
create database mydb;  
-- 修改mydb的字符集为utf8、排序规则为utf8_general_ci  
alter database mydb  
default character set utf8  
default collate utf8_general_ci;
```

2) 使用图形工具修改字符集

步骤:

- 保证数据库是打开状态 (数据库如果是打开状态, 则图标呈绿色)
- 选中某个数据库, 右键->“数据库属性”
- 在弹出的对话框中设置数据库的字符集和排序规则

3) 使用语句删除数据库

注意: 删除数据库后, 该数据库的所有文件都会删除, 数据将全部丢失, 所以需慎用。

语法:

```
drop database [if exists] 数据库名;
```

if exist: 如果数据库存在则执行删除

例: 删除mydb数据库

```
drop database if exists mydb;
```

4) 使用图形工具删除数据库

步骤:

- 选中某个数据库, 右键->"删除数据库"->确定

4、数据库导出/导入


1) 将数据库导出为sql文件

步骤:

- a、保证数据库是打开状态
- b、选中所需导出的数据库, 右键弹出对话框, 选择"转储 sql 文件 ..."
- c、选择转储的 SQL 文件的保存位置
- d、输入 SQL 文件名和保存类型 (默认文件名与导出的数据库同名, 保存类型为 sql 类型)

2) 将sql文件导入为数据库

步骤:

- a、创建一个空数据库, 双击数据库, 确保是打开状态
- b、选中步骤1创建的数据库, 右键弹出对话框, 选择"运行 sql 文件 ..."
- c、单击  浏览并选择需要运行的 SQL 的文件
- d、点击"开始", 如果出现"executed successfully"字样, 则表明sql 文件中的数据成功恢复至数据库中

5、数据表及数据类型

1) 数据表

数据表: 是**数据存储的基本单位**, 数据表被定义为**列的集合**, 数据在表中是按照行和列的格式来存储的

2) 常用数据类型

类型分类	类型	取值范围或描述	示例
整数类型	int	-2147483648 ~ 2147483647	150
小数类型	decimal	decimal(5,2) 表示小数的取值范围是-999.99 ~ 999.99	892.35
变长字符串类型	varchar	varchar(20) 按实际字符数量分配空间	湖北省襄阳市襄城区
日期类型	date	格式: YYYY-MM-DD (年-月-日) 取值范围: 1000-01-01 ~ 9999-12-31	1999-03-24
时间类型	time	格式: HH:MM:SS (时:分钟:秒) 取值范围: -838:59:59 ~ 838:59:59 注意: 时间不限当天	08:30:00
日期时间类型	datetime	格式: YYYY-MM-DD HH:MM:SS 取值范围: 1000-01-01 00:00:00 ~ 9999-12-31 23:59:59	2015-05-12 14:35:00

6、DDL语句创建表

完整语法:

```
create table 表名(
  列名1 数据类型 [约束1] [约束2],
  列名1 数据类型 [约束1] [约束2]
  ..
);
```

约束说明如下:

primary key : 主键约束, 保证表中任意二条记录不完全相同。一个表中只能有1个主键, 主键列不能有Null值。主键允许创建在1个或多个列上。

auto_increment : 值自动增长。只能创建在整数类型, 并且是主键的列。该列的值由mysql自动生成

unique : 唯一约束, 与主键作用相同。一个表中可以有任意个唯一约束, 唯一约束允许null值。

not null : 非空约束, 要求必须填入数据

default : 默认值, 该列未输入数据时, 使用默认的数据

foreign key : 主外键约束, 约束某列的值必须来自于另一表的列

例: 创建班级表和学生表

```
create table classes(
  cid int primary key auto_increment, -- cid为主键, 并且值自动增长
  cname varchar(20) not null,
  major varchar(10) default '大数据' -- 设置默认值为大数据
);

create table student(
```

```
sid int primary key auto_increment,  
stuno varchar(20) not null unique, -- 学号非空, 并且唯一  
name varchar(20) not null,  
address varchar(50),  
age int,  
cid int,  
foreign key(cid) references classes(cid) -- 指定cid是外键, 引用classes表的cid  
);
```

7、修改表结构

1) 修改表通用语法

```
alter table 表名 修改子句
```

2) 新增字段的修改子句

```
add 列名 数据类型 [约束]
```

例: 学生表中新增email列

```
alter table student add email varchar(30);
```

3) 更改列名的修改子句

```
change 旧列名 新列名 数据类型 [约束]
```

例: 将学生表的name列更改为stuname

```
alter table student change name stuname varchar(20);
```

4) 更改数据类型的修改子句

```
modify 列名 新数据类型 [约束]
```

例: 将学生表的phone列数据类型改为char(11)

```
alter table student modify phone char(11);
```

5) 删除列的修改子句

```
drop column 列名
```

例: 删除学生表中的email列

```
alter table student drop column email;
```

6) 添加约束的修改子句

```
add constraint 约束名 约束类型(列)
```

例: 为班级表的班级名称添加唯一约束

```
alter table classes add constraint uni_cname unique(cname)
```

例: 为学生表的cid添加外键约束, 引用班级表的id列

```
alter table student add constraint fk_cid foreign key(cid) references classes(cid);
```

7) 删除约束的修改子句

`drop` 约束类型 约束名;

例: 删除学生表cid的外键约束

```
alter table student drop foreign key fk_cid;
```

例: 删除班级表的班级名称的唯一约束

```
alter table student drop index uni_cname
```

8) 更改表名

`alter table` 旧表名 `rename to` 新表名 或者 `rename table` 旧表名 `to` 新表名;

例: 将student表更改为stu

```
alter table student rename to stu;
```

```
rename table student to stu;
```

三、使用DML语句操纵数据

数据操作语言, 主要包含insert、delete、update语句

1、insert语句

语法:

```
insert [into] 表名[(列名列表)] values(值列表)
```

例:

```
insert into classes values(null,'大数据1903')
```

注意:

- 1、表名后如无指定列名列表, 则表示向所有列插入数据, 值列表中数据与列一一对应
- 2、使用自动增长值, 则值列表中使用null占位, 使用default默认值, 则值列表中使用default占位
- 3、表名后可以指定部分列名, 则值列表中只给部分列数据, 但非空的列必须给值
- 4、指定值列表时, 字符串和日期类型数据需要用引号

例:

```
insert into student(sid,stuno,stuname,age) values(null,'19000078','张三',21)
```

插入多行:

```
insert into student(sid,stuno,stuname,age) values
```

```
(null,'19000075','张三',18),
```

```
(null,'19000076','李四',19),
```

```
(null,'19000077','王五',20)
```

2、update语句

语法:

```
update 表名 set 列1=值1 [,列2=值2,...] [where 条件表达式];
```

例: 将学号为19000077的学生姓名更改为赵六

```
update student set stuname='赵六' where stuno='19000077'
```

例: 将所有人的年龄在原来基础上增加1

```
update student set age=age+1
```

说明:

1、where 表达式指定了表中的哪些记录需要修改, 若省略了 where 子句, 则表示修改表中的所有记录。

2、set 子句指定了要修改的字段以及该字段修改后的值。

3、delete语句

语法:

```
delete from 表名 [where 条件表达式];
```

例:

```
delete from student where stuno='19000077'
```

四、使用DQL查询数据

1、select完整语法

```
select 列名列表|*      -- 指定显示的列, 多个列名用逗号分隔。显示所有列用 '*' 5
from 表名              -- 指定要查询的表或视图, 可以多个, 用逗号分隔 1
where 条件表达式      -- 指定筛选的条件 2
group by 列名         -- 指定分组规则 3
having 条件表达式     -- 指定对分组的结果继续筛选的条件 4
order by 列名 asc|desc -- 指定对查询结果进行排序 6
limit 偏移量,记录数  -- 指定略过多少条记录及显示的记录数 7
```

2、列名列表的用法

1) 查询所有列

```
select * from 表名;
```

2) 查询部分列

```
select 列名列表 from 表名;
```

例：查询学生的学号和姓名

```
select stuno,stuname from student;
```

注意：多个列名用逗号分隔，最后一个列名与from之间没有逗号

3) 列使用别名显示

```
select 列1 [as] 别名 [,列n as 别名n] from 表名;
```

例：查询学生的学号及姓名，显示中文列名

```
select stuno as 学号,stuname as 姓名 from student;
```

4) 去重

```
select distinct 列名 from 表名;
```

例：显示所有专业信息，去掉重复值

```
select distinct major from classes;
```

5) 在列中使用算术运算符

例：查询每个班级在加10人后的人数

```
select *,nums+10 as 加后人数 from classes;
```

3、where子句

where子句用于条件筛选，可使用and或or指定一个或多个条件。

运算符	含义
=	等于
<=>	安全等于, 可以比较 null
<> 或 !=	不等于
>=	大于等于
<=	小于等于
>	大于
<	小于
IS NULL	判断一个值是否为 null
IS NOT NULL	判断一个值是否不为 null

运算符	含义
AND 或者 &&	逻辑与。当所有操作数均为非零值且不为 null 时, 返回值为 1; 当一个或多个操作数为 0 时, 返回值为 0;
OR 或者	逻辑或 当所有操作数不为 null 且任意一个操作数为非零时, 返回值为 1, 否则为 0; 当有操作数为 null, 且存在另一个操作数为非零时, 返回值为 1, 否则为 null; 当所有操作数均为 null 时, 返回值为 null
NOT 或者 !	逻辑非。当操作数为 0 时, 返回值为 1; 当操作数为非零时, 返回值为 0; 当操作数为 null 时, 返回值为 null

1) 单条件查询

例: 查询超过40人的班级信息

```
select * from classes where nums>40;
```

2) 多条件查询

例: 查询来自河南的学生中, 电话未登记的学生

```
select * from student where phone is null and address='河南';
```

3) 模糊查询

a、like 与通配符 '*', '_' 结合使用

语法: `select 列名 from 表名 where 列 like '模糊值';`

例: 查询电话号码以17开头的学生姓名及班级编号

```
select stuname,cid from student where phone like '17%';
```

例: 查询名字以c开头, 四个字母的学生信息


```
select * from student where stuname like 'c___';
```

b、in 运算符

语法: `select 列名 from 表名 where 列 in (值列表);`

例: 查询班级编号为1, 3, 4的学生信息

```
select * from student where cid in(1,3,4);
```

c、not in 运算符

d、between ..and..

语法: `select 列名 from 表名 where 列 between 值1 and 值2;`

注意:

- 值1小于值2
- 查询结果中包含值1和值2

例: 查询班级人数在35到40人之间的班级信息

```
select * from classes where nums between 35 and 40;
```

4、定制查询结果

1) 排序 order by子句

语法: `select 列名 from 表名 order by 列1[,列2] [asc|desc];`

说明: 升序是asc, 默认是升序, 所以asc可以省略; 降序使用desc

例: 查询所有学生信息, 按班级编号排序显示

```
select * from student order by cid;
```

例: 查询所有学生信息, 按班级编号升序和生日排序降序显示

```
select * from student order by cid,birthday desc;
```

2) 使用limit限制查询记录数

语法: `SELECT 列名 FROM 表名 LIMIT [偏移量,] 记录数;`

例: 查询所有学生信息, 显示前5条记录

```
select * from student limit 5;
```

例: 查询所有学生信息, 显示从第6条开始的3条记录

```
select * from student limit 5,3;
```

五、聚合函数和分组查询

1、聚合函数

1) sum: 求某列值的总和

语法:

```
select sum(列名) from 表名;
```

2) max: 返回某列的最大值

3) min: 返回某列的最小值

4) avg: 返回某列的平均值

5) count: 统计记录行数

语法:

```
select count(*) from 表名;    -- 返回该表的行数, null值也参与统计
select count(列) from 表名;  -- 返回某列的非null行数, null值不参与统计
```

2、group by分组

1) group by作用

根据一定规则将数据集划分成若干小组

2) 语法

```
select 聚合函数(列名)      -- group by通常与max,min,count,sum,avg结合使用, 即分组后对每
进行统计
from 表名
group by 列名列表         -- 按某列值的不同来分组
[having 条件表达式]      -- 分组统计后, 再次条件筛选
```

注意: 在select后的列, 要么是聚合列, 要么是分组依据列, 否则查询结果无意义。

例: 统计班级表中不同省份的人数

```
select address,count(*) from student group by address
```

例: 统计登记电话的学生中不同省份的人数, 按人数多少排序

```
select address,count(*) 人数
from student
where phone is not null
group by address
order by 人数
```

- group by 用于分组
- order by 用于排序

3) 多字段分组

group by 子句的分组依据可以是一个列名列表，分组优先级从左至右，即先按第一个列进行分组，然后在第一个列值相同的记录中，再根据第二个列的值进行分组，依次类推。

例：统计不同班级不同省份的学生人数

```
select cid,address,count(*) from student group by cid,address
```

4) having 子句

having 子句用于分组统计后，对统计的数据进行条件筛选。

例：统计不同班级不同省份的学生人数，仅显示人数超过2人的记录

```
select cid,address,count(*) 人数
from student
group by cid,address
having 人数>2
```

注意：where用于原表数据的条件筛选，having用于分组统计后的条件筛选

例：统计大数据的3个班中各个省份的学生人数及平均分,仅显示学生人数大于2人的记录，按平均分降序显示

```
select address,count(*) 人数,avg(score) 平均分
from student
where cid in(1,2,3)
group by address
having 人数>2
order by 平均分 desc
```

六、MySQL函数

1、数学函数和控制流函数

1) 数学函数

a、ceil(num) 对num向上取整

如：select ceil(23.1) 返回24

b、floor(num) 对num向下取整

如：select floor(23.7) 返回23

c、round(num) 对num四舍五入取整

如：select round(23.7) 返回24

d、round(num,d) 对num保留d位小数

如：select round(23.75645, 2) 返回23.76

e、truncate(num,d) 对num进行截取操作

如: select truncate(23.75645, 2) 返回23.75

2) 控制流函数

a、if(表达式,值1,值2)

判断表达式的条件, 如果为true则返回值1, 否则为值2。在mysql中, 非0的值或非null的值作为条件时, 也为true。

例:

```
SELECT if(20>10, '大于', '小于')    -- 返回大于
```

b、ifnull(值1,值2)

如果值1不是null, 则返回值1, 否则值2

例:

```
SELECT IFNULL(null, '未分配')    -- 返回未分配
```

2、字符串函数

作用: 用于处理字符串数据。

- 1) char_length(str): 计算str的字符个数, 返回数值
- 2) concat(s1,s2,...): 连接多个字符串, 返回字符串
- 3) insert(str,pos,len,newstr): 替换字符串, 在str中pos位置开始,将len长度的内容替换为newstr。返回字符串
- 4) replace(str,oldstr,newstr): 替换字符串, 在str中查找oldstr,将其替换为newstr。返回字符串
- 5) substr/substring(str,pos,len): 截取字符串, 在str中从pos开始截取len长度字符串。返回字符串。
- 6) locate(s1,str): 查找字符串, 返回子字符串s1在字符串str中第一次出现的位置, 未找到返回0。返回数值

3、日期函数

作用: 用来处理日期和时间的值

- 1) curdate(): 返回系统的日期
- 2) curtime(): 返回系统的时间
- 3) now(): 返回系统日期和时间
- 4) dayofweek(date): 返回date是一周的哪一天, 返回值为1~7。1为周日。
- 5) year(date)/month(date)/day(date): 返回date所对应的年或月或日

6) `date_add(date,interval 数值 type)` : 返回在date基础上添加时间间隔。type表示时间间隔的类型。adddate函数与date_add函数功能及用法相同。

常用type类型取值如下:

type值	描述说明
YEAR	对日期中年份部分添加时间间隔
MONTH	对日期中月份部分添加时间间隔
DAY	对日期中天的部分添加时间间隔
HOUR	对日期中小时部分添加时间间隔
MINUTE	对日期中分的部分添加时间间隔

例: 查询'2022-03-11'增长2天后的日期

```
select date_add('2022-03-11',INTERVAL 2 DAY) -- 返回'2022-03-13'
```

7) `datediff(date1,date2)` : 返回两个日期之间相隔的天数。

8) `date_format(date,格式字符串)`: 返回按格式字符串指定的格式显示的date值

常用格式说明符如下:

格式说明符	描述说明
%y / %Y	2位的年份 / 4位的年份
%m	月份的数字表现形式
%d	月中某一天的数字表现形式
%H	24时制显示的小时
%i	数字形式表现的分钟数
%s	数字形式表现的秒数
%T / %r	24时制的时间(hh:mm:ss) / 12时制的时间, 后跟AM或PM

4、系统信息和加密

1) 系统信息函数

a、`user()` : 获取当前登录用户及服务器地址。

b、`database()` : 获取当前正在使用的数据库。

c、`LAST_INSERT_ID()` : 获取最后一个自动生成的ID值。

注意:

- 如果向一个表插入多条数据, 则函数返回第一条数据产生的ID值。
- 函数结果与表无关, 向表1插入数据后, 再向表2插入数据, 则该函数返回的结果是表2中的ID值。

2) 加密函数

password(str): 返回对字符串str进行加密后的字符串。不可逆。

md5(str): 对str计算出一个md5 128比特校验和, 返回32位十六进制的字符串。

encode(str,pass_str): 将pass_str作为密钥对str进行加密。可使用decode解密

decode(crypt,pass_str): 将pass_str作为密钥对加密后的密文crypt进行解密。

七、连接查询

当查询的数据来自于多个表时, 就要考虑使用连接查询。

1、交叉连接

语法:

```
select 列名列表 from 表1 cross join 表2 [where 表1.列名=表2.列名]
```

注意: 当使用交叉连接后, 会产生迪卡尔乘积, 返回的记录数是2个表的记录数之积。一般不用交叉连接

2、内连接

内连接: 返回2个表相匹配的数据

语法:

```
select 列名列表 from 表1 inner join 表2 on 表1.列名=表2.列名 [where 条件] 或  
select 列名列表 from 表1,表2 where 表1.列名=表2.列名 [and 其他条件] -- 推荐写法
```

例: 查询学生学号, 姓名及班级名称

```
select stuno,stuname,classname from student s inner join classes c on  
s.cid=c.cid;  
select stuno,stuname,classname from student s ,classes c where s.cid=c.cid;
```

例: 查询学号为'Xx2201001'的学生姓名、班级编号和班级名称

```
select s.cid,stuname,classname from student s inner join classes c on  
s.cid=c.cid where stuno='xx2201001';  
select s.cid,stuname,classname from student s,classes c where s.cid=c.cid and  
stuno='xx2201001';
```

3、外连接

1) 分类:

左外连接: 显示左表所有记录及右表相匹配的记录

右外连接: 显示右表所有记录及左表相匹配的记录

全外连接：显示所有连接表中的所有记录

2) 左外和右外语法：

```
select 列名列表 from 表1 left join 表2 on 表1.列名=表2.列名 [where 条件] -- 左外连接
select 列名列表 from 表1 right join 表2 on 表1.列名=表2.列名 [where 条件] -- 右外连接
```

例：显示所有班级信息及学生信息

分析：如果使用内连接查询，这时只会显示部分班级信息，因为在学生表中仅有部分班级编号。这里需要使用左外或右外连接

```
select * from classes c left join student s on c.cid=s.cid; -- 左外连接
select * from student s right join classes c on c.cid=s.cid; -- 右外连接
```

3) 全连接语法

```
select 列名列表 from 表1
union
select 列名列表 from 表2
```

注意：在使用全连接时，2个查询的表中列要相同

例：显示所有学生及所有班级信息

```
select * from classes c left join student s on c.cid=s.cid
union
select * from classes c right join student s on c.cid=s.cid;
```

4、子查询

子查询是一个嵌套在select、insert、update、delete语句或其他子查询中的查询语句，任何允许使用表达式的地方均可使用子查询。

实质：一个select子句的查询结果作为另一个子句的输入值。

1) 单行子查询：查询结果仅返回单行单列的数据，将该数据做为另一查询语句的输入值

单行子查询的结果可以使用=,<,>,>=,<=来比较

例：查询大数据2203班所有的学生信息

思路：先从班级表中查询2203班的编号，再将编号做为学生表的条件，查询对应的学生信息

```
select * from student where cid=(select cid from classes where classname='大数据2203')
```

例：查询分数大于全班平均分的学生信息

思路：先求全班平均分，再将平均分做为查询其他学生信息的条件

```
select * from student where score>(select avg(score) from student)
```

例：查询“张鹏飞”所有的订购的商品编号同，商品数量及商品金额

思路：先求张鹏飞的客户编号，再查询该客户编号在orders订单表中对应的订单编号，再将订单编号做为orderdetail订单详细表的查询条件

```
SELECT goodsid,quantity,money from ordersdetail where ordersid=
(SELECT ordersid from orders where customerid=
(SELECT customerid from customer where customername='张鹏飞'));
```

2) 多行子查询：子查询结果返回多行单列的数据

多行子查询的结果可以用 in,not in,all,any 运算符来比较

例：查询大数据技术的学生信息

```
SLECT * from student WHERE cid in
(SELECT cid from classes where major='大数据技术')
```

例：查询比所有河南学生分数高的学生信息

```
SELECT * from student where score>all
(SELECT score from student WHERE address='河南')
```

例：查询比任一河南学生分数高的学生信息

```
SELECT * from student where score>ANY
(SELECT score from student WHERE address='河南')
```

3) 在其他语句中使用子查询

a、在update语句中使用子查询

例：将班级表的人数登记有误，统计学生表每个班的人数更新到班级表

将统计每个班人数的子查询做为一个虚拟表与classes表连接，将虚拟表的数据更新到班级表

```
update classes c,
(SELECT cid,count(*) number from student GROUP BY cid) b
set c.nums=b.number where c.cid=b.cid
```

b、在select语句中使用子查询

例：查询班级信息及每个班级的最高分

```
SELECT c.cid,classname,nums,major,tmp.max from classes c left JOIN
(select cid,max(score) max FROM student GROUP BY cid) tmp
on c.cid=tmp.cid
```

八、索引和视图

1、索引的概念和作用

1) 索引：是一种单独的、物理的对数据库表中**一列或多列的值进行排序的一种存储结构**

2) 作用：加快对数据的检索

3) 分类：

- 普通索引
- 唯一索引：具有唯一约束的列，即为唯一索引
- 主键索引：主键列，即为主键索引
- 全文索引
- 空间索引

缺点：维护需要时间，占磁盘空间。

2、索引的创建和使用

1) 创建索引

语法一：

```
CREATE INDEX 索引名 ON table_name (字段名[长度],...[ASC|DESC])
```

语法二：

```
ALTER TABLE 表名 ADD INDEX[索引名](字段名(长度),...[ASC|DESC])
```

2) 查看索引

```
SHOW INDEX from 表名;
```

3) 删除索引

语法一：

```
Drop index 索引名 on 表名
```

语法二：

```
alter table 表名 drop index 索引名
```

3、视图的概念和作用

1) 视图：是一种虚拟的表，结构形式和表一样，可以进行查询，修改，删除或更新操作。

2) 优点：

- 定制用户数据
- 简化数据操作
- 提高数据的安全性
- 重用sql语句

4、视图的创建和使用

1) 创建语法

```
create [or replace] view 视图名[(列名列表)]
as
查询语句;
```

- or replace : 当视图存在, 则修改该视图

例: 创建视图, 视图中仅显示学生的学号,姓名和分数

```
create view view_student
as
select stuno 学号,stuname 姓名,score 分数 from student;
```

使用视图: 可以查询视图, 修改或删除视图

```
select * from view_student;
update view_student set 姓名='朱一龙' where 学号='xx2201002';
delete from view_student where 学号='xy2101022';
```

注意: 在对视图进行删除或修改的时候, 影响的是视图引用的原表的数据。

2) 查看视图的详细信息

```
DESCRIBE 视图名;
SHOW CREATE VIEW 视图名;
```

3) 修改视图

```
alter view 视图名
as
查询语句;
```

4) 删除视图

```
drop view 视图名;
```

九、变量、存储过程和触发器

1、变量

1) 变量分类:

局部变量: 仅在begin和end中使用, 如函数或存储过程

用户变量: 用户定义的变量, 作用域为当前连接

会话变量: 服务器为每个连接的客户端维护一系列会话变量。

全局变量: 整个服务器运行时有效

2) 局部变量

局部变量定义:

```
declare 变量名 变量类型 [default 值];
```

局部变量赋值:

```
set 变量名=值;  
select 列 into 变量 from 表名;
```

3) 用户变量

用户变量定义并赋值:

```
set @变量名=值;  
select @变量名:=值;  
select 列 into @变量名 from 表名;
```

例: 声明变量保存jack的学号

```
select @sno:=stuno from student where stuname='jack';  
select @sno;
```

注意: 用户变量名以@开头

4) 会话变量

显示会话变量:

```
show session variables [like '%关键字%']
```

会话变量赋值:

```
set @@session.变量名称=值;  
set session 变量名=值;  
set 变量名=值
```

例: 修改自动递增步长的会话变量为2

```
set auto_increment_increment=2
```

5) 全局变量

显示全局变量:

```
show global variables;
```

全局变量赋值:

```
set @@global.变量名=值;  
set global 变量名=值;
```

例: 修改最大连接数的全局变量为1000

```
set global max_connections=1000;
```

注意：会话变量和全局变量由**系统创建**

2、函数

函数用于实现某种特定的功能，前面学习了字符串函数，数学函数，日期函数等。mysql也允许自定义函数。

定义函数语法：

```
create function 函数名称([参数列表]) RETURNS 返回值类型  
    函数体  
    return 具体值
```

调用函数语法：

```
select 函数名([参数]);
```

例：定义函数，用于得到学生对应的学号

```
-- 定义函数  
CREATE FUNCTION getStuno(name varchar(10)) RETURNS VARCHAR(20) -- 定义函数，有一个参数name，返回类型是varchar  
BEGIN  
    DECLARE stunoo VARCHAR(20);  
    SELECT stuno INTO stunoo FROM student WHERE stuname=name;  
    RETURN stunoo; -- 返回的具体值  
end;  
-- 调用函数  
select getstuno('jack');
```

例：编写从1累加到100的函数

```
CREATE FUNCTION getsum() RETURNS int -- 声明返回值的类型  
BEGIN -- 函数开始  
    DECLARE a int default 1;  
    declare sum int default 0;  
    WHILE a<=100 DO -- while循环开始  
        set sum=sum+a;  
        set a=a+1;  
    end while; -- 循环结束  
    RETURN sum; -- 返回的具体值  
END; -- 结束函数  
  
-- 调用函数  
select getsum();
```

3、存储过程

1) 存储过程概念

一组为了完成特定功能的SQL语句集，在第一次使用经过编译后，再次调用就不需要重复编译，因此执行效率比较高

2) 存储过程优点

- 重复使用：存储过程可以重复使用
- 提高效率：存储过程在第一次使用的时候会编译，一次编译后不用再次编译，提高执行效率。
- 减少网络流量：调用的时候只需要传递存储过程的名称和参数。
- 安全性：参数化的存储过程可以防止sql注入。

3) 存储过程语法：

```
create procedure 存储过程名([[in |out| inout]参数名 数据类型)
begin
    过程体;
end;
```

存储过程设置参数时，在参数名前还可以指定参数的来源及用途，区别如下。

- IN：表示输入参数，即参数是在调用存储过程时传入到存储过程里面使用，传入的数据可以是直接数据（如5），也可以是保存数据的变量。
- OUT：表示输出参数，初始值为NULL，它是将存储过程中的值保存到OUT指定的参数中，返回给调用者。
- INOUT：表示输入输出参数，即参数在调用时传入到存储过程，同时在存储过程中操作之后，又可将数据返回为调用者
- 未指定参数用途，则默认是输入参数(in)

调用语法：

```
CALL 存储过程名称([实参列表]);
```

语法说明：

- **实参列表**传递的参数需要与创建存储过程的**形参相对应**。
- 当形参被指定为**IN**时，则实参值可以为变量或是直接数据；
- 当形参被指定为**OUT**或**INOUT**时，调用存储过程传递的参数必须是一个变量，用于接收返回给调用者的数据

a、无参存储过程

例：编写存储过程，用于查询学生学号及班级名称

```
drop PROCEDURE if EXISTS getinfo1; -- 如果存在则删除getinfo1
CREATE PROCEDURE getinfo1()
BEGIN
    SELECT stuno,classname from classes c,student s
    where c.cid=s.cid;
END;

-- 调用无参存储过程
CALL getinfo1();
```

b、输入参数的存储过程

例：-定义存储过程，用于根据班级名称查询学生信息

```
drop PROCEDURE if EXISTS getinfobyname;
CREATE PROCEDURE getinfobyname(cname VARCHAR(20))
BEGIN
    SELECT c.classname,s.* FROM student s,classes c
    WHERE c.cid=s.cid AND c.classname=cname;
end;

-- 调用输入参数的存储过程
CALL getinfobyname('大数据2202')
```

例：根据参数显示分页内容

```
DROP PROCEDURE if EXISTS pagingByIndexAndSize;
CREATE PROCEDURE pagingByIndexAndSize(page int,size int)
BEGIN
    DECLARE skip int;
    set skip=(page-1)*size;
    -- limit中偏移量的计算公式(page-1)*size
    SELECT * from student LIMIT skip,size;
end

-- 调用分页存储过程
CALL pagingByIndexAndSize(3,5);-- 显示学生第3页的5条记录
```

c、输入输出参数的存储过程

例：查看某个职称最高的工资，得到工资，用于后续计算

```
drop PROCEDURE if EXISTS proc_querySalaryByRankName;
create PROCEDURE proc_querySalaryByRankName(rankname VARCHAR(20),out salary INT)
BEGIN
SELECT max(e.salary) INTO salary from employee e,rank r where e.rankid=r.rankid
and r.RankName=rankname;
END;

-- 调用输入输出存储过程
set @salary=0; -- 声明保存输出参数的变量
CALL proc_querySalaryByRankName('工程师',@salary);
SELECT @salary;
```

4) 存储过程与函数的区别

- 返回值上的不同：函数有且只有一个返回值，存储过程可以返回多个结果集。
- 参数的不同：函数的参数只是输入函数，而存储过程的参数类型有三种：in, out, inout。

十、流程控制语句、事务和触发器

1、流程控制语句

1) 条件判断语句

a、IF语句：一般在函数或存储过程中使用

语法：

```
if 条件1 then 语句1;
[elseif 条件2 then 语句2;]
[else 语句n;]
end if;
```

例：创建存储过程，用于更新指定班级名的人数，如果低于40人则+5人，如果大于40人则+3人

```
CREATE PROCEDURE updateClassNum(cname VARCHAR(20))
BEGIN
    DECLARE num INT;
    SELECT nums into num from classes where classname=cname;
    if num>40 THEN -- 判断该班级的人数是否大于40人
        update classes set nums=nums+5 WHERE classname=cname;
    ELSE
        update classes set nums=nums+3 WHERE classname=cname;
    end if;-- 结束if语句，注意加;号
END

-- 调用存储过程，传入数据
CALL updateClassNum('大数据2202')
```

b、Case语句

语法1：

```
case 变量
when 值1 then 语句1
when 值2 then 语句2
else 语句n
end case;
```

语句2：

```
case
when 条件1 then 语句1
when 条件2 then 语句2
else 语句n
end case;
```

例：显示学生信息及分数层次，如果>90显示优秀，如果>80良好，如果>60 中等，<60不及格

```
SELECT *,
CASE
WHEN score>90 THEN '优秀'
WHEN score>80 THEN '良好'
WHEN score>60 THEN '中等'
else '不及格'
end '层次'
from student
```

注意：上例中的case在select语句中使用，则end后可以不跟case，同时then后的语句没有;号。

2) 循环语句

循环语句仅用于begin..end语句块内。如函数和存储过程

a、while循环

语法：

```
while 条件 do
    循环体;
end while;
```

b、repeat循环

语法：

```
repeat          -- 直接执行循环体
    循环体;
until 条件      -- 循环退出条件，与while条件相反
end repeat;
```

c、loop循环

语法：

```
标识:loop
    循环体;          -- 循环体内可用'leave 标识'跳出循环
end loop;
```

2、游标

1) 概念：一种能从多条数据记录的结果集中每次提取一条记录的机制。

2) 游标的使用

a、声明游标 `declare 游标名 cursor for 表名或查询结果;`

b、打开游标 `open 游标名;`

c、读取记录 `fetch 游标名 into 变量1[,变量2,...];` 每条记录有几个列，则需要有几个变量

d、关闭游标 `close 游标名;`

例：读取班级名称，利用游标将所有班级名称整合成一个完整的字符串


```

CREATE PROCEDURE getClassname()
BEGIN
    DECLARE cname VARCHAR(20);      -- 保存每一条记录中的班级名
    DECLARE result VARCHAR(200) default ''; -- 保存最终结果的字符串
    DECLARE error int default 0;    -- 是否还有结果的标识
    DECLARE youbiao CURSOR FOR SELECT classname from classes; -- 声明游标, 遍历20条
    班级名称
    DECLARE CONTINUE HANDLER for not found set error=1; -- 声明事件处理程序, 当找不到
    数据时, 让error变量为1
    OPEN youbiao; -- 打开游标
    lp:LOOP
        FETCH youbiao into cname; -- 执行一次fetch 只能读一条
        set result=CONCAT(result,cname,','); -- 将班级名拼接到result中
        if error=1 THEN -- 当error为1时, 表示已没有数据
            leave lp; -- 离开loop循环
        end if;
    end loop;
    CLOSE youbiao; -- 关闭游标
    SELECT result; -- 查询结果
end;

```

3、事务

- 1) 作用: 事务保证数据库中数据的一致性和可恢复性
- 2) 概念: 一个由用户所定义的完整的工作单元。一个事务内的所有语句作为一个整体来执行
- 3) 四大特性:
 - 原子性: 事务中的操作语句是一个不可分割的整体, 要么全部执行, 要么全部不执行。
 - 一致性: 事务执行前后, 数据库数据的状态保持一致, 以保持所有数据的完整性。
 - 隔离性: 事务A所作的修改与其它并发事务所作的修改隔离。
 - 持久性: 一个事务成功完成之后, 它对数据库所作的改变是永久性的。
- 4) 事务使用:

开启事务: `start transaction;`

提交事务: `commit;`

回滚事务: `rollback;`

例: 使用存储过程向学生表添加1条信息, 则班级表人数也需更新。将insert和update语句做为事务整体提交。

```

CREATE PROCEDURE proc_tran()
BEGIN
    DECLARE error int default 0; -- 声明变量, 用于标识异常
    -- 声明事件处理程序, 当有异常或警告发生, 修改标识变量为1;
    DECLARE CONTINUE HANDLER FOR SQLWARNING, SQLEXCEPTION set error=1;
    start transaction; -- 开启事务
    INSERT into student values(null, 'xx2002220', 'aaa', null, null, null, 87, 2);
    UPDATE classes SET nums=nums+1 where cid=2;
    IF error=0 then
        COMMIT; -- 当没有异常则提交事务
    ELSE

```

```
ROLLBACK;    -- 有异常回滚事务，所有语句都不执行
END if;
end
-- 调用存储过程
CALL proc_tran();
```

4、触发器

1) 概念：是与表的事件insert、update、delete相关的一种特殊的存储过程。触发器的执行是由事件触发的。

2) 语法：

```
-- CREATE TRIGGER 触发器名 触发时间 触发事件
CREATE TRIGGER 触发器名 AFTER|BEFORE INSERT|UPDATE|DELETE
ON 表名 FOR EACH ROW
BEGIN
触发器内容;
END;
```

- AFTER|BEFORE：表示触发的时间，是在操作之前还是操作之后发生。
- INSERT|UPDATE|DELETE：触发事件的语句类型。如果是insert，表示向指定表插入数据时，该触发器被触发。
- 触发器内容：触发器被触发后，执行的sql语句。

注意：在触发器中使用new表示表的新数据。old表示表的旧数据。

例：当删除班级表数据时，将被删除的数据记录下来。

思路：

先创建一个日志表，记录操作班级表的用户，时间和其他列的数据。

再在班级表上创建delete触发器，当删除班级表数据时，触发器自动向日志表中记录被删除的数据。

```
DROP TABLE if EXISTS mylogs;
CREATE TABLE mylogs(
    id int auto_increment PRIMARY key,
    ope_type VARCHAR(50),
    ope_user VARCHAR(50),
    ope_date datetime,
    classname VARCHAR(50),
    nums int,
    major VARCHAR(50)
);

DROP TRIGGER if EXISTS tri_classes;-- 如果tri_classes触发器存在则删除
-- 在班级表中创建触发器，当删除数据之后，自动向日志表中保存被删除的数据及操作信息
CREATE TRIGGER tri_classes AFTER Delete ON classes for EACH ROW
BEGIN
INSERT INTO mylogs
VALUES(null,'delete',user(),now(),old.classname,old.nums,old.major);
end;
```

十一、用户权限

1、用户管理

1) 查看所有用户

用户信息存储在mysql数据库的user表中

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

2) 使用create语句创建用户

语法:

```
create user 用户名@主机 IDENTIFIED by [password] 密码;
```

主机: localhost表示本机, 其他主机需要指定主机ip地址。

密码: 是该用户登录的密码, password用于对密码加密。

3) 使用insert语句创建用户

向mysql数据库中的user表插入一条数据。要使用insert插入用户, 需对user表有insert权限。

```
INSERT
mysql.`user`(host,user,authentication_string,ssl_cipher,x509_issuer,x509_subject
)
VALUES('localhost','admin2',PASSWORD('123456'),'','','');
```

4) 删除用户

语法:

```
drop user '用户名'@'主机名';    -- drop语句删除用户
delete from mysql.user where user='用户名' and host='主机名';    -- delete语句从user
表中删除用户
```

2、权限管理

1) 授予权限语法:

```
grant 权限列表 [列名列表] on 数据库.表名 to '用户名'@'主机名';
```

参数说明:

- 权限列表: 表示用户的权限, 用逗号分隔。all privileges 表示所有权限;
- 数据库.表名: 表示用户权限作用在哪个数据库的哪个表, 例: test.student、test.*、*.*

常用权限列表:

权限	作用范围	作用
all	服务器	所有权限
select	表、列、视图	选择列
insert	表、列	插入行
update	表、列	更新行
delete	表	删除行
create	数据库、表、索引	创建
drop	数据库、表、视图	删除
alter	表	修改表结构

2) 查看用户权限

```
SHOW GRANTS for '用户名'@'主机';
```

3) 收回权限

```
revoke 权限列表 [列名列表] on 数据库名.表名 from '用户名'@'主机名';
```

3、windows下对数据库进行备份和还原

1) 使用mysqldump命令备份

注意：必须进入mysql安装路径的bin目录再执行命令。

```
C:\Users\Administrator>cd C:\Program Files\MySQL\MySQL Server 5.7\bin
C:\Program Files\MySQL\MySQL Server 5.7\bin>mysqldump -u root -p test>d:\demo.sql
Enter password: *****
```

进入mysql安装的bin目录
将test数据库备份到d盘
文件名是demo.sql

```
cd C:\Program Files\MySQL\MySQL Server 5.7\bin
mysqldump -u root -p test>d:\demo.sql
```

2) 使用mysql命令对数据还原

```
C:\Program Files (x86)\MySQL\MySQL Server 5.6\bin>mysql -u root -p school < d:\demo.sql
```