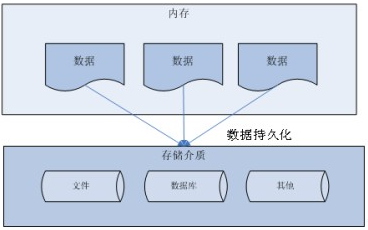
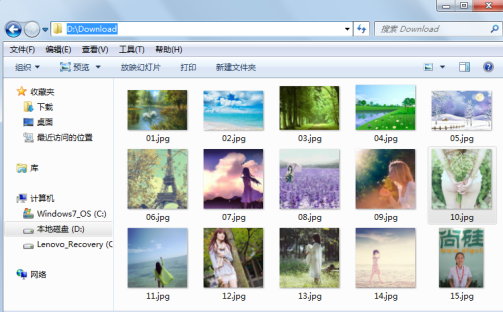
# 数据库基础

## 1、为什么要使用数据库

**持久化**(persistence)：**把数据保存到可掉电式存储设备中以供之后使用**。持久化的大多数时候是将内存中的数据存储在数据库中，当然也可以存储在磁盘文件、XML数据文件中。

**方便管理数据（例如：快速的检索等）**

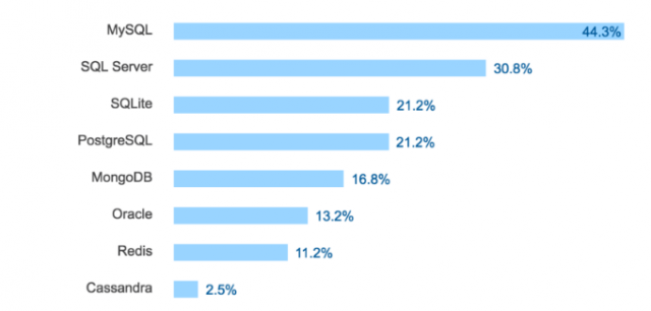
 

## 2、什么是数据库

**DB：数据库（Database）**即存储数据的“仓库”。它保存了一系列有组织的数据。

**DBMS：数据库管理系统（Database Management System）**：是一种操纵和管理数据库的大型软件，例如建立、使用和维护数据库。

目前互联网上常见的数据库管理软件有Sybase、DB2、Oracle、MySQL、Access、Visual Foxpro（面向对象型）、MS SQL Server、Informix、PostgreSQL（最符合SQL标准，但是性能差）这几种。以下是2017年StackOverflow 对各数据库受欢迎程度进行调查后的统计结果：



**MySQL是一种开放源代码的关系型数据库管理系统**，开发者为瑞典MySQL AB公司。在2008年1月16号被Sun公司收购。而2009年,SUN又被Oracle收购.目前 MySQL被广泛地应用在Internet上的中小型网站中。由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，使得很多互联网公司选择了MySQL作为网站数据库（Facebook, Twitter, YouTube，阿里的蚂蚁金服，去哪儿，魅族，百度外卖，腾讯）。

阿里巴巴/蚂蚁金服主要使用两种关系数据库：OceanBase和MySQL。数据规模：MySQL单台机器TB级，OceanBase单个集群从几个TB到几百个TB皆有。

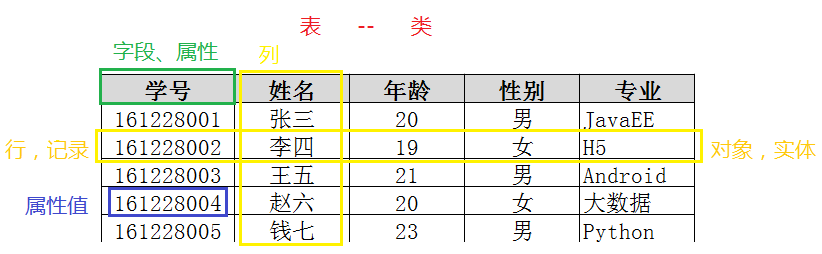
去哪儿：MySQL，Redis，HBase

腾讯社交网络主要使用深度定制MySQL数据库+自研NoSQL，规模万台以上服务器，千万级qps。

百度外卖目前线上主要使用Mysql、redis等数据库。MySQL 数据数百TB级，redis 数据几TB级。

目前魅族OLTP场景主要使用的是MySQL，缓存服务使用的是Redis。数据库实例近1000，数据大小100T+, redis实例1000+。

**关系型数据库：**关系数据库的表采用二维表格来存储数据，是一种按行与列排列的具有相关信息的逻辑组，它类似于Excle工作表。一个数据库可以包含任意多个数据表。**表中的一行即为一条记录**。**数据表中的每一列称为一个字段**，表是由其包含的各种字段定义的，每个字段描述了它所含有的数据的意义，数据表的设计实际上就是对字段的设计。创建数据表时，为每个字段分配一个数据类型，定义它们的数据长度和其他属性。行和列的交叉位置表示某个属性值，如“数据库原理”就是课程名称的属性值。



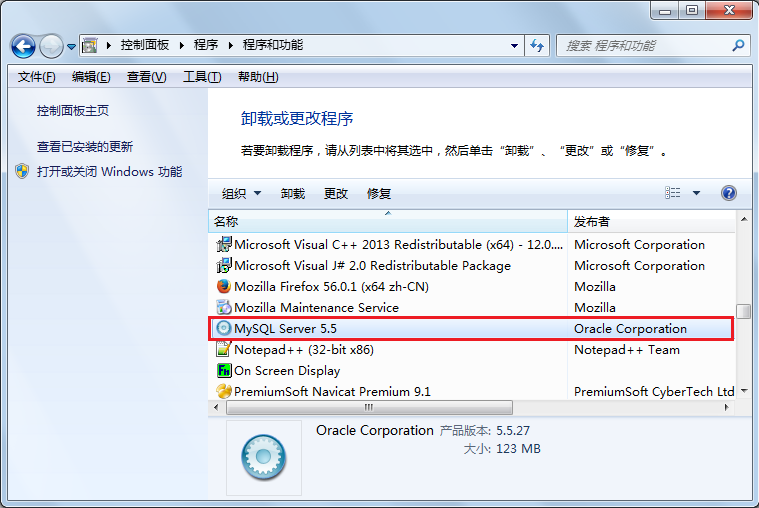
**SQL：**结构化查询语言(Structured Query Language)。

# 二、windows版MySQL软件的安装与卸载

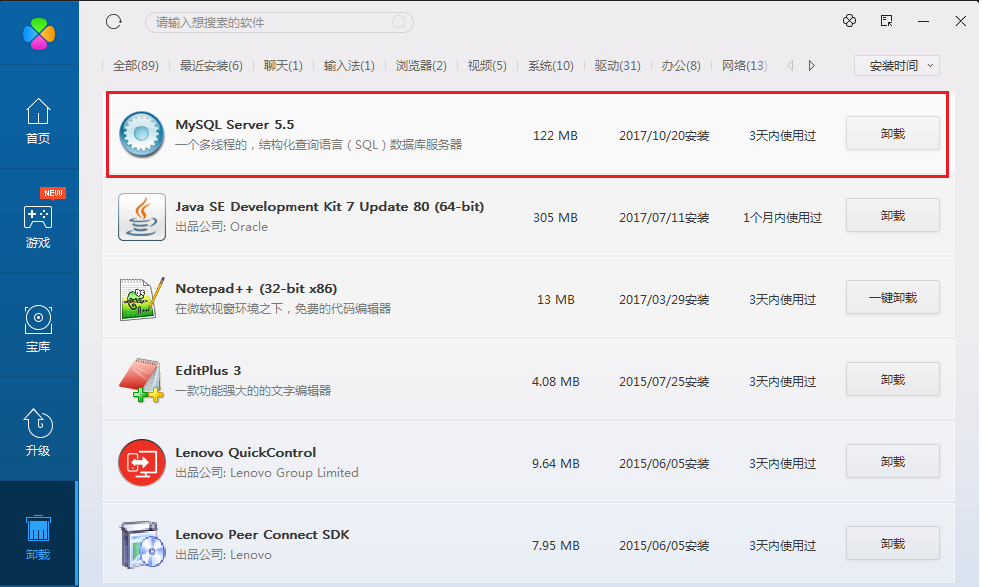
## 1、卸载

### 1、软件的卸载

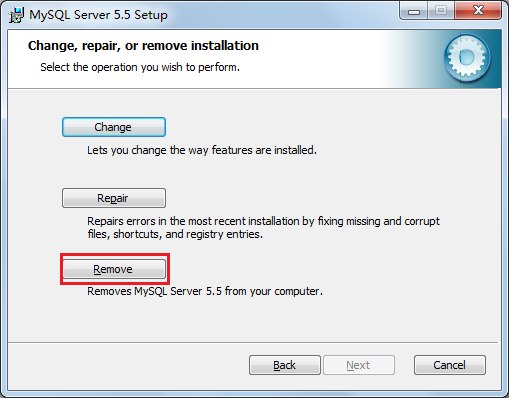
#### 方式一：通过控制面板



#### 方式二：通过电脑管家等软件卸载



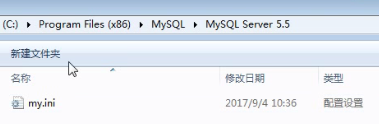
#### 方式三：通过安装包中提供的卸载功能卸载



### 2、清理残余文件

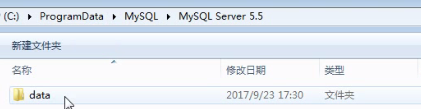
如果再次安装不成功，可以卸载后对残余文件进行清理后再安装

#### （1）清除安装残余文件



#### （2）清除数据残余文件

请在卸载前做好数据备份



### 3、清理注册表

如何打开注册表编辑器：在系统的搜索框中输入regedit

如果前两步做了，再次安装还是失败，那么可以清理注册表

1：HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Services\Eventlog\Application\MySQL服务 目录删除

2：HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Services\MySQL服务 目录删除

3：HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet002\Services\Eventlog\Application\MySQL服务 目录删除

4：HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet002\Services\MySQL服务 目录删除

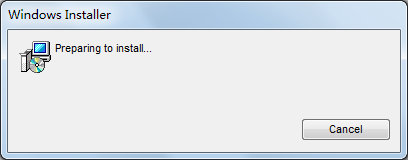
5：HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\MySQL服务目录删除

6：HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\MySQL服务删除

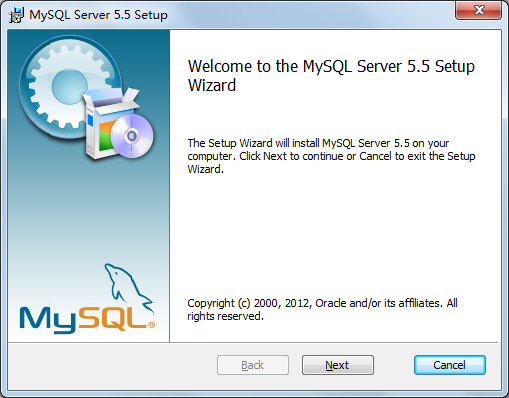
注册表中的ControlSet001,ControlSet002,不一定是001和002,可能是ControlSet005、006之类

## 2、安装

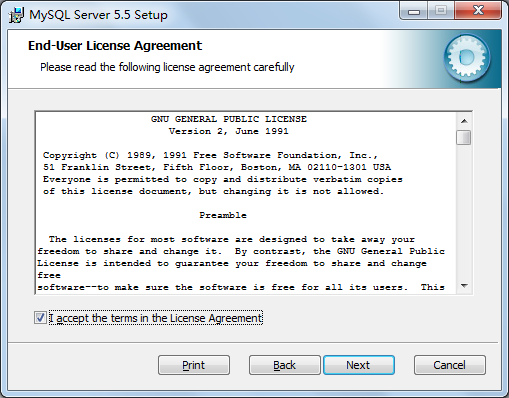
### （1）准备安装



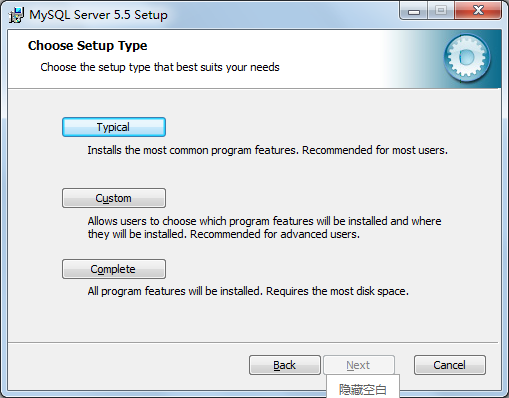
### （2）欢迎安装



### （3）准许协议



### （4）选择安装模式

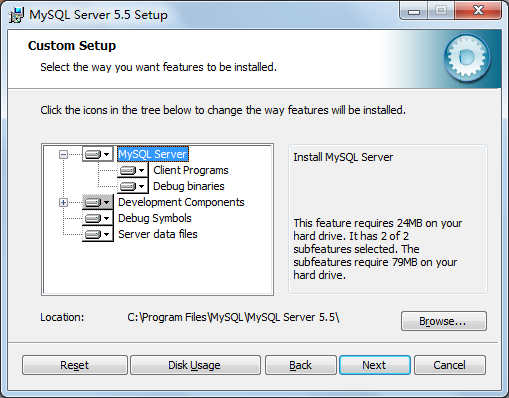


Typical：表示一般常用的组件都会被安装，默认情况下安装到”C:\Program Files\MySQL\MySQL Server 5.5\”下。

Complete：表示会安装所有的组件。此套件会占用比较大的磁盘空间。

Custom：表示用户可以选择要安装的组件，可以更改默认按照的路径。这种按照类型最灵活，适用于高级用户。

### （5）选择安装组件及安装路径



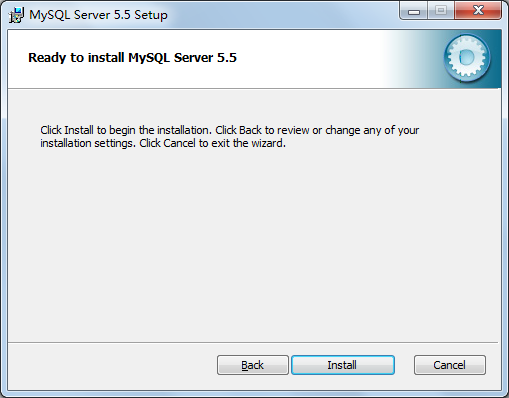
这里可以选择安装哪些部分，主要是这里可以设置两个路径：

MySQL Server的应用软件的安装路径，默认在“C:\**Program Files**\MySQL\MySQL Server 5.5\”

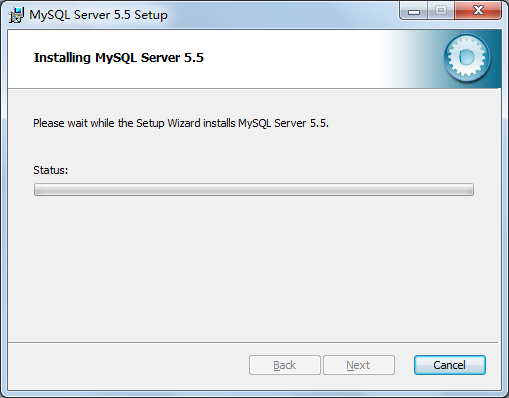
Server data files的数据存储的目录路径，默认在“C:\**ProgramData**\MySQL\MySQL Server 5.5\”

建议把数据存储的目录路径修改一下，以防系统崩溃或重装系统时数据保留。

### （6）开始安装

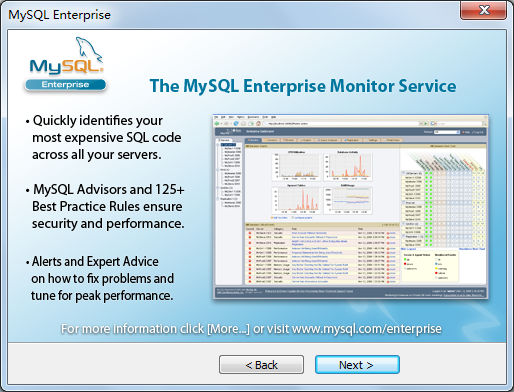


安装进度

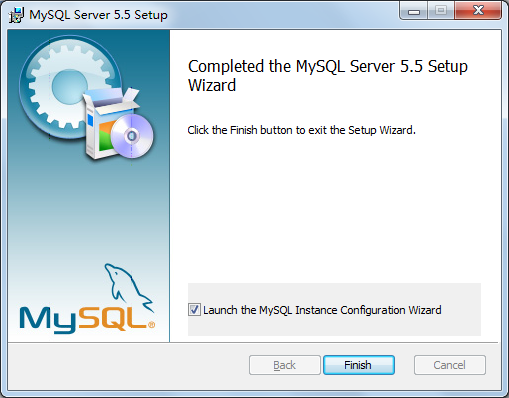


系统会显示MySQL Enterprise版（企业版）的一些功能介绍界面，可以单击“Next”继续。





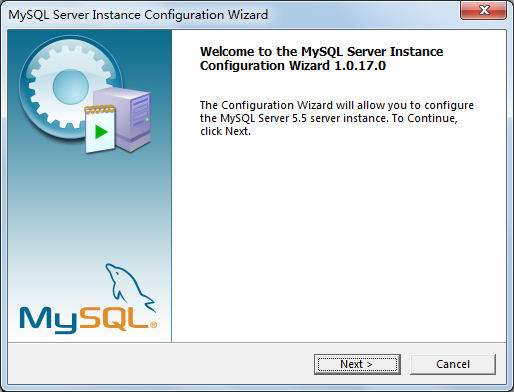
### （7）安装完成



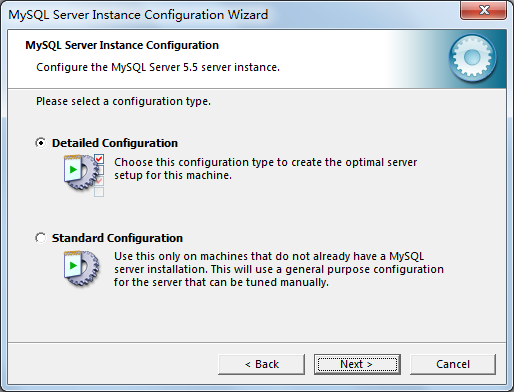
单击“Finish”按钮完成安装过程。如果想马上配置数据库连接，选择“Launch the MySQL Instance Configuration Wizard”复选框。如果现在没有配置，以后想要配置或重新配置都可以在“MySQL Server”的安装目录的bin目录下（例如：D:\ProgramFiles\MySQL5.5\MySQL Server 5.5\bin）找到“MySQLInstanceConfig.exe”打开“MySQL Instance Configuration Wizard”向导。

## 3、MySQL的配置

### （1）准备开始

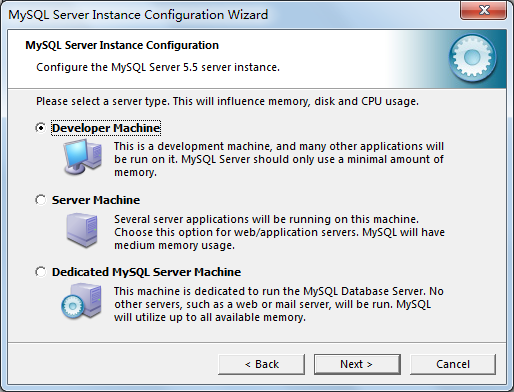


### （2）选择配置类型



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

### （3）选择MySQL的应用模式

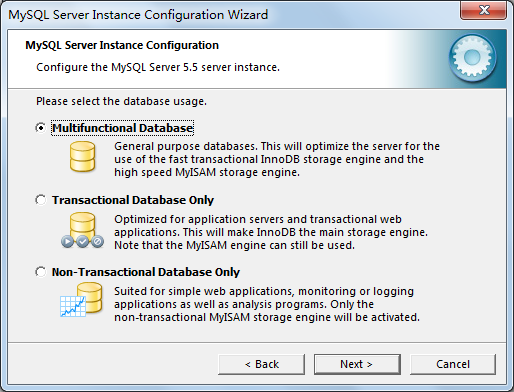


Develop Machine（开发机），使用最小数量的内存

Server Machine（服务器），使用中等大小的内存

Dedicated MySQL Server Machine（专用服务器），使用当前可用的最大内存。

### （4）选择数据库用途选择界面



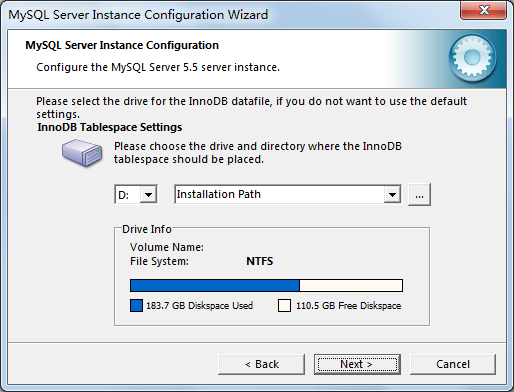
选择mysql数据库的大致用途：

“Multifunctional Database（通用多功能型，好）”：此选项对事务性存储引擎（InnoDB）和非事务性（MyISAM）存储引擎的存取速度都很快。

“Transactional Database Only（服务器类型，专注于事务处理，一般）”：此选项主要优化了事务性存储引擎（InnoDB），但是非事务性（MyISAM）存储引擎也能用。

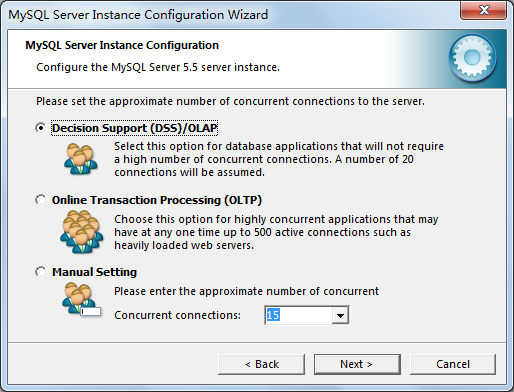
“Non-Transactional Database Only（非事务处理型，较简单）主要做一些监控、记数用，对MyISAM数据类型的支持仅限于non-transactional，注意事务性存储引擎（InnoDB）不能用。

### （5）配置InnoDB数据文件目录



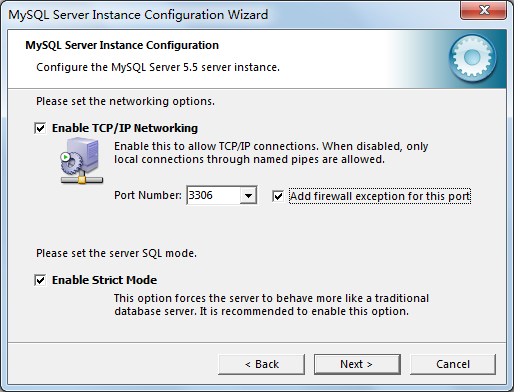
InnoDB的数据文件会在数据库第一次启动的时候创建，默认会创建在MySQL的安装目录下。用户可以根据实际的空间状况进行路径的选择。

### （6）并发连接设置



选择您的网站的一般mysql 访问量，同时连接的数目，“Decision Support(DSS)/OLAP（决策支持系统，20个左右）”、“Online Transaction Processing(OLTP)（在线事务系统，500个左右）”、“Manual Setting（手动设置，自己输一个数）”

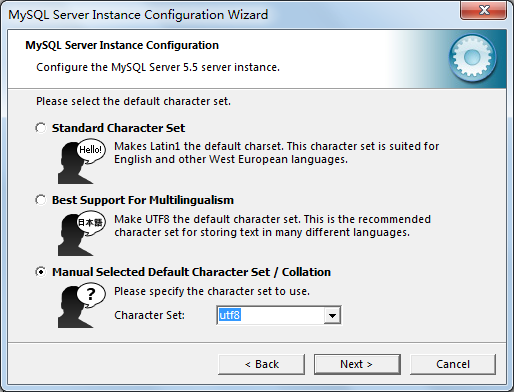
### （7）网络选项设置



是否启用TCP/IP连接，设定端口，如果不启用，就只能在自己的机器上访问mysql 数据库了，我这里启用，把前面的勾打上，Port Number：3306，还有一个关于防火墙的设置“Add firewall exception ……”需要选中，将MYSQL服务的监听端口加为windows防火墙例外，避免防火墙阻断。

在这个页面上，您还可以选择“启用标准模式”（Enable Strict Mode），这样MySQL就不会允许细小的语法错误。尽量使用标准模式，因为它可以降低**有害数据**进入数据库的可能性。

### （8）选择字符集



注意：如果要用原来数据库的数据，最好能确定原来数据库用的是什么编码，如果这里设置的编码和原来数据库数据的编码不一致，在使用的时候可能会出现乱码。

这个比较重要，就是对mysql默认数据库语言编码进行设置，第一个是西文编码，第二个是多字节的通用utf8编码，第三个，手工选择字符集。

提示：

如果安装时选择了字符集和“utf8”，通过命令行客户端来操作数据库时，有时候会出现乱码，

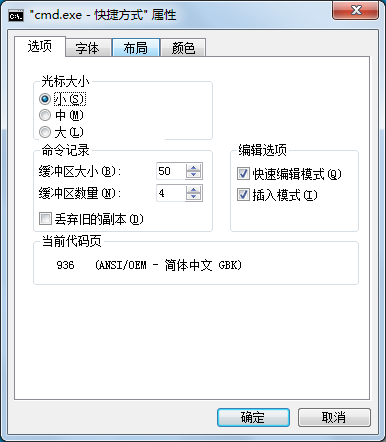
这是因为“命令行客户端”默认是GBK字符集，因此客户端与服务器端就出现了不一致的情况，会出现乱码。

可以在客户端执行：

mysql> set names gbk;

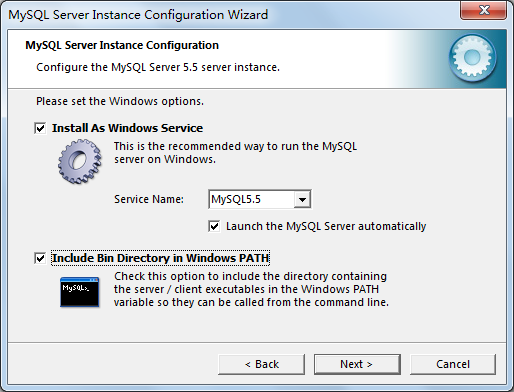
可以通过以下命令查看：

mysql> show variables like 'character\_set\_%';



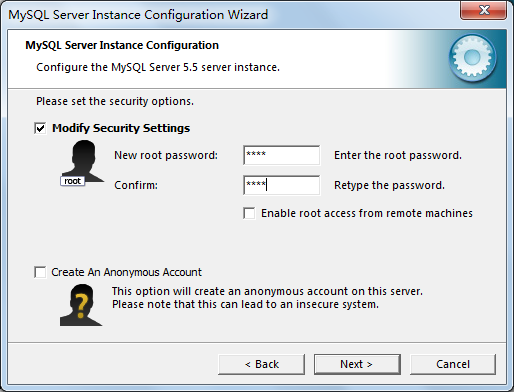
对于客户端和服务器的交互操作，MySQL提供了3个不同的参数：character\_set\_client、character\_set\_connection、character\_set\_results，分别代表客户端、连接和返回结果的字符集。通常情况下，这3个字符集应该是相同的，才能确保用户写入的数据可以正确的读出和写入。“set names xxx;”命令可以同时修改这3个参数的值，但是需要每次连接都重新设置。

### （9）安全选择



选择是否将mysql 安装为windows服务，还可以指定Service Name（服务标识名称，例如我这里取名为“MySQL5.5”），是否将mysql的bin目录加入到Windows PATH环境变量中（加入后，就可以直接使用bin下的命令）”，我这里全部打上了勾。

### （10）设置密码



这一步询问是否要修改默认root 用户（超级管理）的密码（默认为空），“New root password”如果要修改，就在此填入新密码，“Confirm（再输一遍）”内再填一次，防止输错。（如果是重装，并且之前已经设置了密码，在这里更改密码可能会出错，请留空，并将“Modify Security Settings”前面的勾去掉，安装配置完成后另行修改密码）

“Enable root access from remotemachines（是否允许root 用户在其它的机器或使用IP地址登陆，如果要安全，就不要勾上，如果要方便，就勾上它）”。如果没有勾选，默认只支持localhost和127.0.0.1连接。

最后“Create An Anonymous Account（新建一个匿名用户，匿名用户可以连接数据库，不能操作数据，包括查询，如果要有操作数据的权限需要单独分配）”，一般就不用勾了

### （11）准备执行界面



### （12）完成



# 三、MySQL的使用

## 1、启动和停止服务

**MySQL软件的服务器端必须先启动，客户端才可以连接和使用使用数据库**。

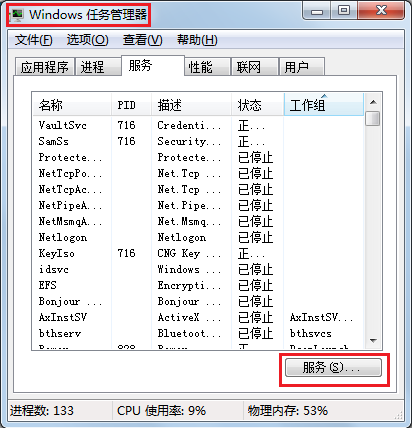
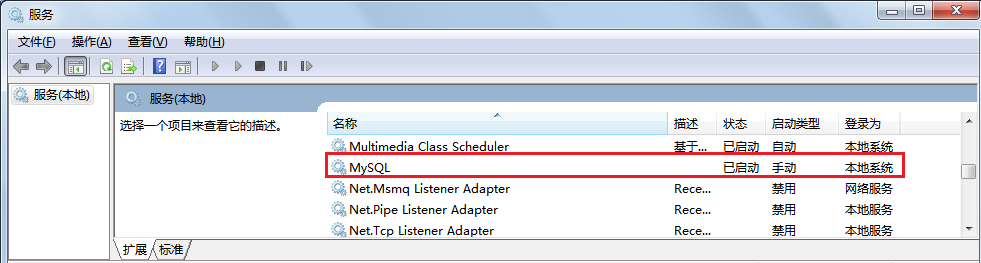
启动服务的方式：

### ~~方式一：图形化方式~~

“我的电脑/计算机”-->右键-->“管理”-->“服务”-->启动和关闭MySQL

“开始菜单”-->“控制面板”-->“管理工具”-->“服务”-->启动和关闭MySQL

“任务管理器”-->“服务”-->启动和关闭MySQL

### \*\*方式二：命令行\*\*

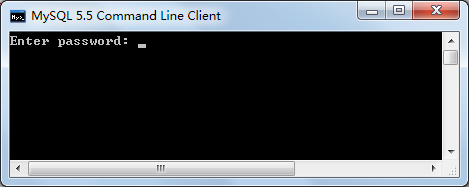
net start MySQL服务名

net stop MySQL服务名

## 2、客户端登录

### ~~方式一：MySQL自带客户端~~

~~“开始菜单”-->MySQL-->MySQL Server 5.5 --> MySQL 5.5 Command Line Client~~



仅限于root用户

### \*\*方式二：命令行\*\*

**mysql -h 主机名 -P 端口号 -u 用户名 -p密码**

例如：mysql -h localhost -P 3306 -u root -proot

注意：

（1**）-p与密码之间不能有空格**，其他参数名与参数值之间可以有空格也可以没有空格

mysql -hlocalhost -P3306 -uroot -proot

（2）密码建议在下一行输入

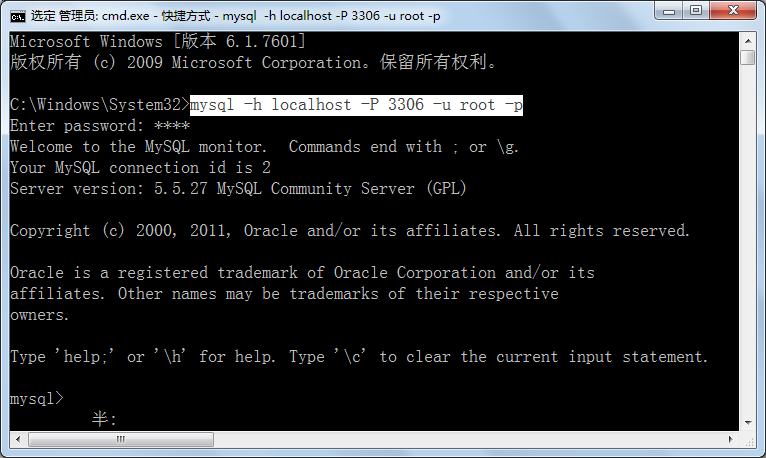
mysql -h localhost -P 3306 -u root -p

Enter password:\*\*\*\*

（3）如果是连本机：-hlocalhost就可以省略，如果端口号没有修改：-P3306也可以省略

简写成：mysql -u root -p

Enter password:\*\*\*\*

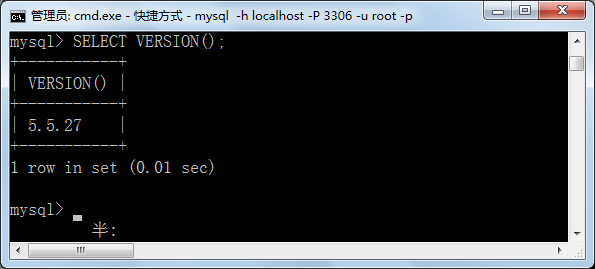


连接成功后，有关于MySQL Server服务版本的信息，还有第几次连接的id标识。

也可以在命令行通过以下方式获取MySQL Server服务版本的信息



或**登录**后，通过以下方式查看当前版本信息：

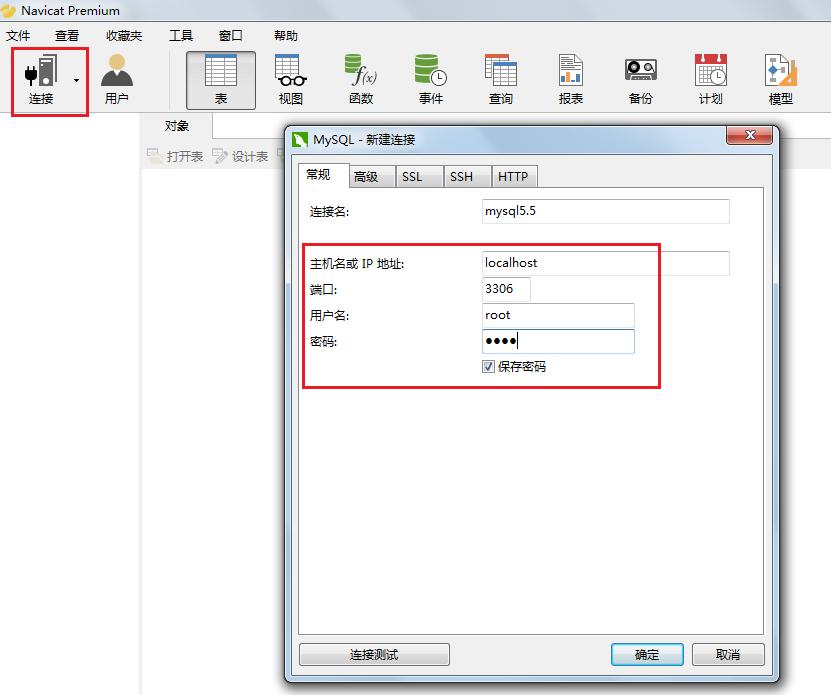


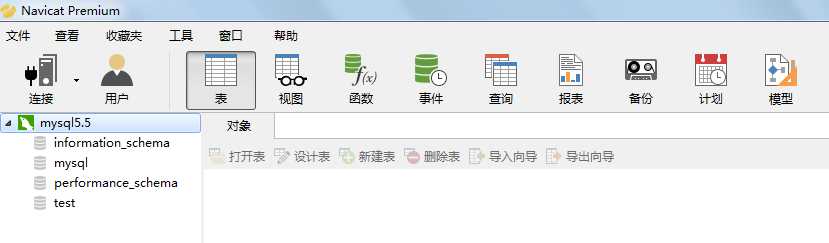
### 方式三：可视化工具

例如：Navicat Preminum，SQLyogEnt等工具

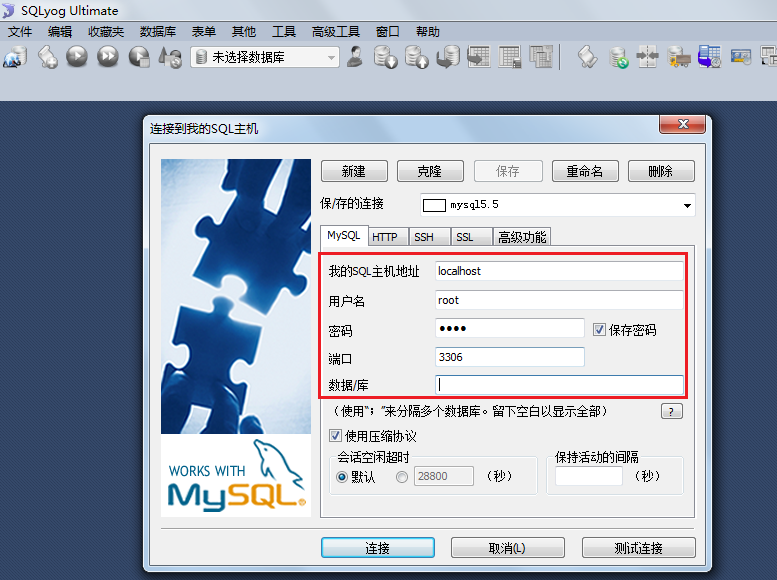
还有其他工具：mysqlfront,phpMyAdmin

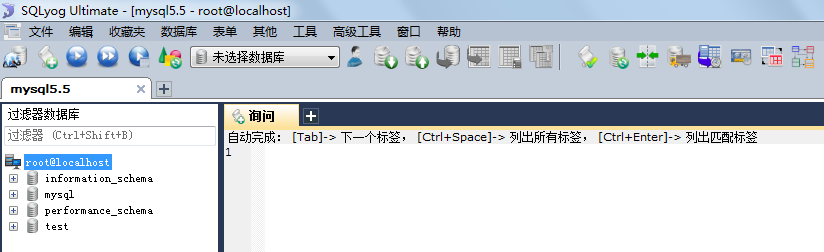
#### （1）Navicat Preminum





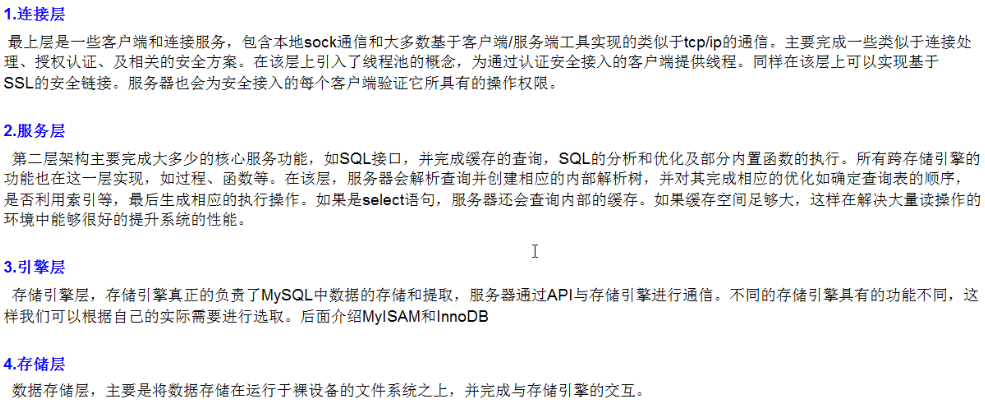
#### （2）SQLyog

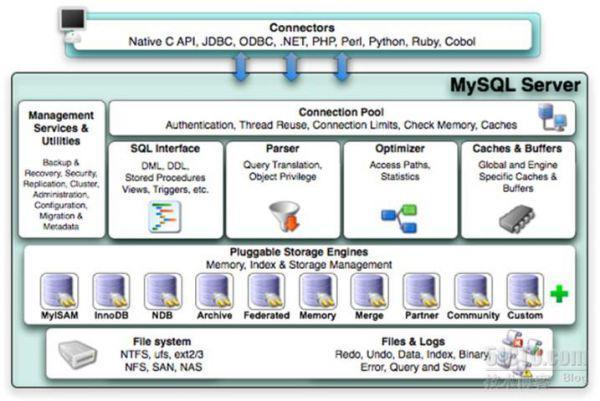




# 四、Mysql的逻辑架构与存储引擎

## 1、MySQL的逻辑架构





## 2、逻辑架构

MySQL最重要、最与众不同的特性是它的**存储引擎架构**，这种架构的设计将查询处理（Query Processing）及其他系统任务（Server Task）和数据的存储/提取相分离。这种处理和存储分离的设计可以在使用时根据性能、特性，以及其他需求来选择数据存储的方式。

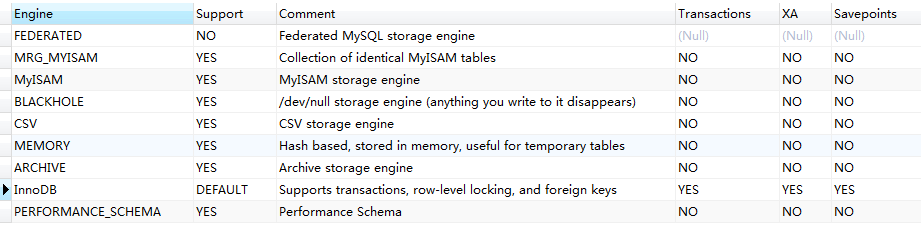
MySQL中同一个数据库，不同的表格可以选择不同的存储引擎。



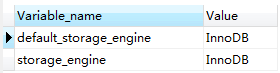
* MyISAM不支持事务、也不支持外键，其优势是访问的速度快，对事务完整性没有要求或者以SELECT、INSERT为主的应用。每个MyISAM在磁盘上存储成三个文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。.frm文件存储表定义。数据文件的扩展名为.MYD (MYData)。索引文件的扩展名是.MYI (MYIndex)。
* InnoDB存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全。但是对比MyISAM的存储引擎，InnoDB写的处理效率差一些，并且会占用更多的磁盘空间以保存数据和索引。InnoDB：所有的表都保存在同一个数据文件中，InnoDB表的大小只受限于操作系统文件的大小限制。Myisam只缓存索引，不缓存真实数据；Innodb不仅缓存索引还要缓存真实数据，对内存要求较高，而且内存大小对性能有决定性的影响。
* MEMORY存储引擎使用存在于内存中的内容来创建表。MEMORY类型的表访问非常的快，因为它的数据是放在内存中的，并且默认使用HASH索引，但是一旦服务关闭，表中的数据就会丢失。主要用于那些内容变化不频繁的代码表或者作为统计操作的中间结果表。

## 3、查看存储引擎

查看当前mysql数据库管理软件支持的存储引擎： SHOW ENGINES;

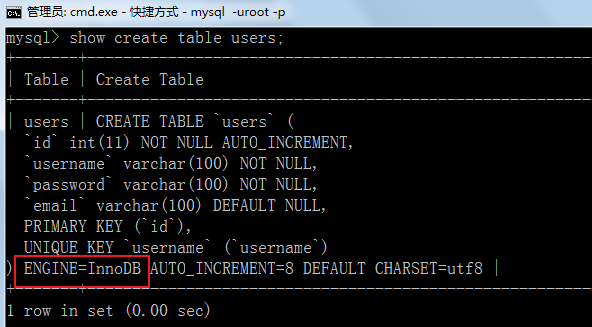


查看默认存储引擎和当前选择的存储引擎：SHOW VARIABLES LIKE '%storage\_engine%';



创建新表时如果不指定存储引擎，那么系统就会使用默认存储引擎，MySQL5.5之前的默认存储引擎是MyISAM，5.5之后改为了InnoDB。

查看已经创建的表格的存储引擎： SHOW CREATE TABLE 表名称;



# 五、简单的SQL示例

## 1、示例小demo

|  |  |
| --- | --- |
| 查看当前的MySQL服务器中有哪些数据库 | mysql> **SHOW DATABASES;**  +--------------------+  | Database |  +--------------------+  | information\_schema |  | mysql |  | performance\_schema |  | test |  +--------------------+  4 rows in set (0.05 sec) |
| 使用test数据库 | mysql> **USE test;**  Database changed |
| 创建表格 | mysql> **CREATE TABLE t\_stu(**  -> **sid INT,**  -> **sname VARCHAR(100),**  -> **gender CHAR**  -> **);**  Query OK, 0 rows affected (0.14 sec) |
| 查看表结构 | mysql> **DESC t\_stu;**  +--------+--------------+------+-----+---------+-------+  | Field | Type | Null | Key | Default | Extra |  +--------+--------------+------+-----+---------+-------+  | sid | int(11) | YES | | NULL | |  | sname | varchar(100) | YES | | NULL | |  | gender | char(1) | YES | | NULL | |  +--------+--------------+------+-----+---------+-------+  3 rows in set (0.03 sec) |
| 插入记录 | mysql> INSERT INTO t\_stu VALUES(1,'张三','男');  Query OK, 1 row affected (0.06 sec)  mysql> INSERT INTO t\_stu VALUES(2,'李四','男');  Query OK, 1 row affected (0.09 sec)  mysql> INSERT INTO t\_stu VALUES(3,'王五','男');  Query OK, 1 row affected (0.06 sec) |
| 查看记录 | mysql> SELECT \* FROM t\_stu;  +------+-------+--------+  | sid | sname | gender |  +------+-------+--------+  | 1 | 张三 | 男 |  | 2 | 李四 | 男 |  | 3 | 王五 | 男 |  +------+-------+--------+  3 rows in set (0.00 sec) |
| 修改记录 | mysql> UPDATE t\_stu SET sname = '张三丰' WHERE sid = 1;  Query OK, 1 row affected (0.08 sec)  Rows matched: 1 Changed: 1 Warnings: 0  mysql> SELECT \* FROM t\_stu;  +------+--------+--------+  | sid | sname | gender |  +------+--------+--------+  | 1 | 张三丰 | 男 |  | 2 | 李四 | 男 |  | 3 | 王五 | 男 |  +------+--------+--------+  3 rows in set (0.00 sec) |
| 删除记录 | mysql> DELETE FROM t\_stu WHERE sid = 1;  Query OK, 1 row affected (0.08 sec)  mysql> SELECT \* FROM t\_stu;  +------+-------+--------+  | sid | sname | gender |  +------+-------+--------+  | 2 | 李四 | 男 |  | 3 | 王五 | 男 |  +------+-------+--------+  2 rows in set (0.00 sec) |

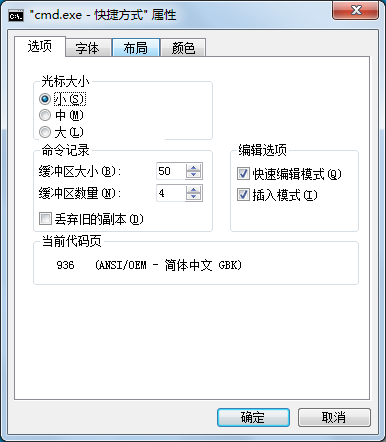
## 2、错误ERROR ：没有选择数据库就操作表格和数据

|  |
| --- |
| ERROR 1046 (3D000): No database selected |
| 解决方案一：就是使用“USE 数据库名;”语句，这样接下来的语句就默认针对这个数据库进行操作 |
| 解决方案二：就是所有的表对象前面都加上“数据库.” |

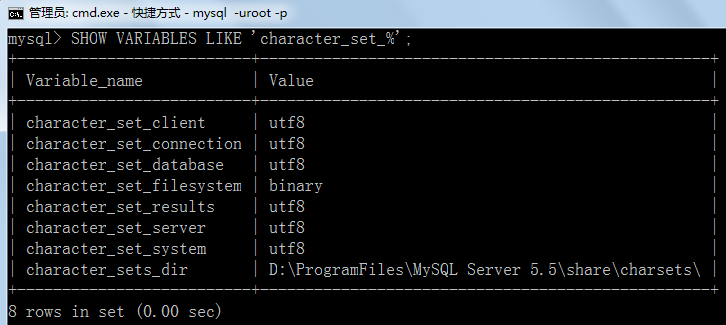
## 3、命令行客户端的字符集问题

|  |
| --- |
| mysql> INSERT INTO t\_stu VALUES(1,'张三','男');  ERROR 1366 (HY000): Incorrect string value: '\xD5\xC5\xC8\xFD' for column 'sname' at row 1 |

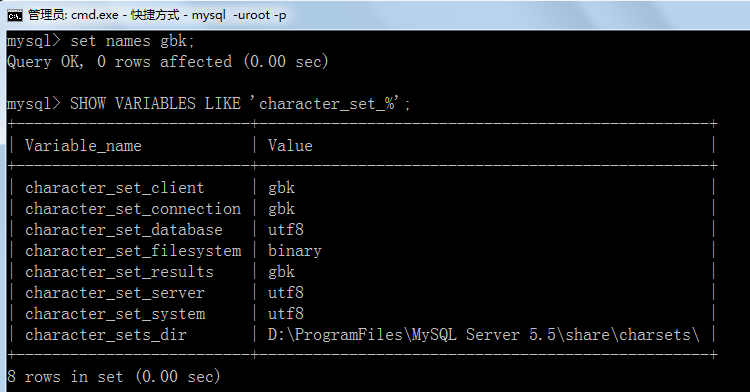
原因：服务器端认为你的客户端的字符集是utf-8，而实际上你的客户端的字符集是GBK



查看所有字符集：SHOW VARIABLES LIKE 'character\_set\_%';



解决方案，设置当前连接的客户端字符集“SET NAMES GBK;”



## 4、查看字符集和校对规则（了解，参考）

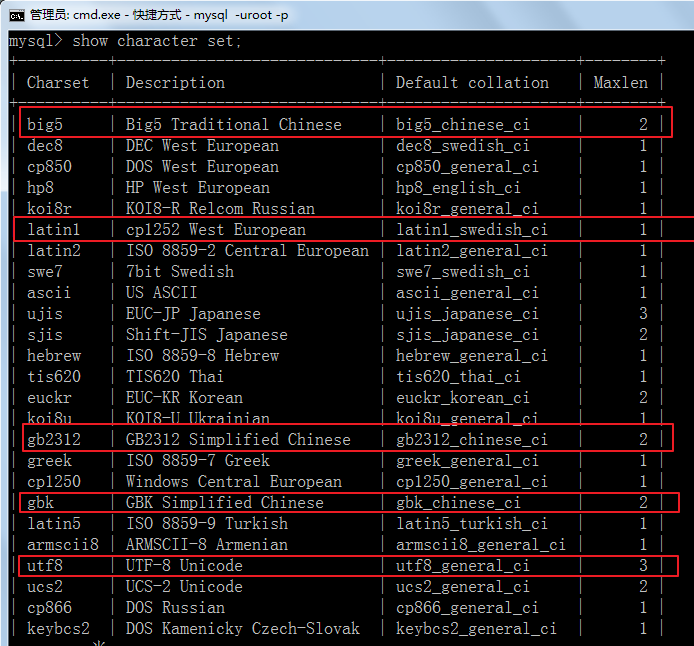
**关于SQL的关键字和函数名等不区分大小写，但是对于数据值是否区分大小写，和字符集与校对规则有关。**

\_ci（大小写不敏感），\_cs（大小写敏感），\_bin（二元，即比较是基于字符编码的值而与language无关）

**字符集：表中数据的编码方式。**

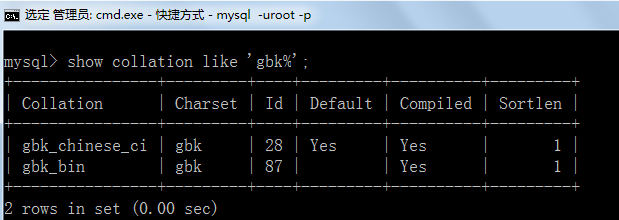
**校对规则：不懂，反正就是以\_ci结尾为大小写不敏感**

### （1）查看所有字符集和校对规则

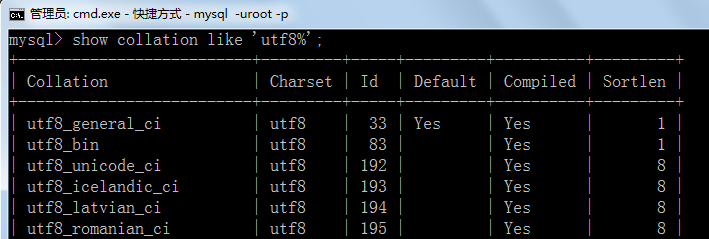


### （2）查看GBK和UTF-8字符集的校对规则

show collation like 'gbk%';



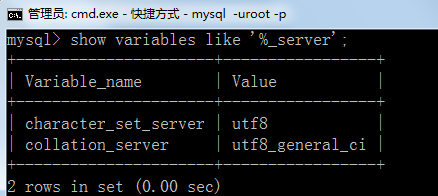
show collation like 'utf8%';



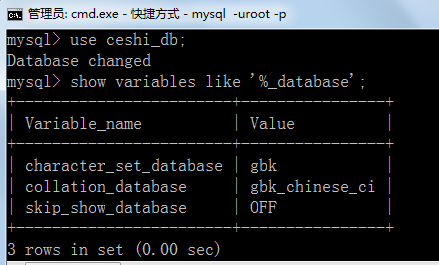
utf8\_unicode\_ci和utf8\_general\_ci对中、英文来说没有实质的差别。  
utf8\_general\_ci 校对速度快，但准确度稍差。  
utf8\_unicode\_ci 准确度高，但校对速度稍慢。

如果你的应用有德语、法语或者俄语，请一定使用utf8\_unicode\_ci。一般用utf8\_general\_ci就够了。

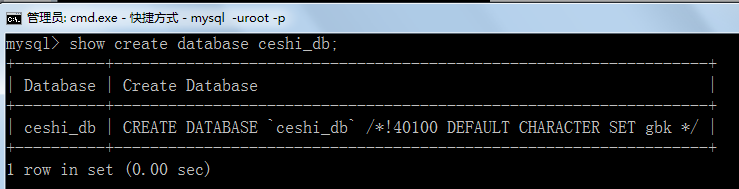
### （3）查看服务器的字符集和校对规则



### （4）查看和修改某个数据库的字符集和校对规则



或

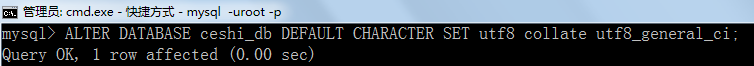


**修改数据库的字符集和校对规则：**

ALTER DATABASE 数据库名称 DEFAULT CHARACTER SET 字符集名称 【COLLATE 校对规则名称】;

例如：

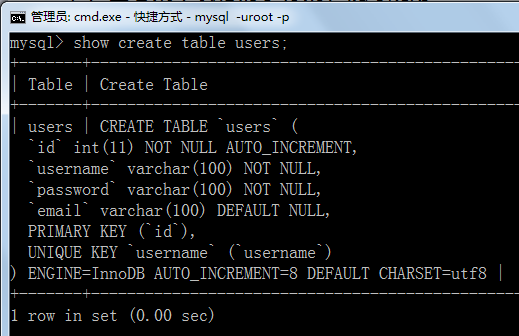
ALTER DATABASE ceshi\_db DEFAULT CHARACTER SET utf8 collate utf8\_general\_ci;



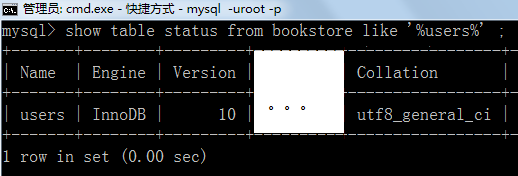
注意：修改了数据库的默认字符集和校对规则后，原来已经创建的表格的字符集和校对规则并不会改变，如果需要，那么需要单独修改。

### （5）查看某个表格的字符集和校对规则

查看字符集：show create table users;



如果要查看校对规则：show table status from bookstore like '%users%' ;



修改某个表格的字符集和校对规则：

修改表的默认字符集：

ALTER TABLE 表名称 DEFAULT CHARACTER SET 字符集名称 【COLLATE 校对规则名称】;

把表默认的字符集和所有字符列（CHAR,VARCHAR,TEXT）改为新的字符集：

ALTER TABLE 表名称 CONVERT TO CHARACTER SET 字符集名称 【COLLATE 校对规则名称】;

例如：ALTER TABLE ceshi\_table DEFAULT CHARACTER SET gbk collate gbk\_chinese\_ci;

# 六、MySQL的数据类型

常用的数据类型有：

* 整型（xxxint）
* 位类型(bit)
* 浮点型（float和double、real）
* 定点数（decimal,numeric）
* 日期时间类型（date,time,datetime,year）
* 字符串（char,varchar,xxxtext）
* 二进制数据（xxxBlob、xxbinary）
* 枚举（enum）
* 集合（set）

## 1、整数（xxxint）

|  |  |  |  |
| --- | --- | --- | --- |
| **整数类型** | **字节** | **有符号** | **无符号** |
| TINYINT (byte) | 1 | -128-127 | **0-255** |
| SMALLINT (short) | 2 | -32768-32767 | **0-65535** |
| MEDIUMINT | 3 | -8388608-8388607 | 0-1677215 |
| INT、INTEGER(int) | 4 | -2147483648-2147483647 | 0-4294967295 |
| BIGINT (long) | 8 | -9223372036854775808~  9223372036854775807 | 0-18446744073709551615 |

整数列的可选属性有三个：

* M: 宽度 (在需要0填充的时候才有意义，否则不需要指定)
* zerofill: 使用0填充 (如果某列是zerofill，那么默认就是无符号)，如果指定了zerofill只是表示不够M位时，用0在左边填充，如果超过M位，只要不超过数据存储范围即可
* unsigned: 无符号类型(非负)

**注意：**

在 int(M) 中，M 的值跟 int(M) 所占多少存储空间并无任何关系。 int(3)、int(4)、int(8) 在磁盘上都是占用 4 bytes 的存储空间。

默认是无符号；要用有符号需要在类型后面声明unsigned

每个int类型可以不设置长度，有默认长度

Create table tab(

T1 int,

T2 int unsigned

T3 int(7)

T4 int(7) zerofill);

## 2、浮点型

小数分为浮点型和定点型

浮点型：float(M,D)和double(M,D)

定点型：decimal(M,D)

特点：

1. M表示精度： 整数+小数的长度
2. D表示标度： 小数的长度

float单精度值使用4个字节，double 双精度值使用8个字节

* MySQL允许使用非标准语法（其他数据库未必支持，因此如果设计到数据迁移，则最好不要这么用）：
* FLOAT(M,D)或DOUBLE(M,D)。这里，(M,D)表示该值一共显示M位，其中D表示小数点后几位，M和D又称为精度和标度。例如，定义为FLOAT(5,2)的一个列可以显示为-999.99-999.99。M取值范围为0~255。D取值范围为0~30，同时必须<=M。
* 存储时，整数部分超出了范围（如上面的例子中，添加数值为1000.01），MySql就会报错，不允许存这样的值。如果小数点部分若超出范围，就分以下情况：若四舍五入后，整数部分没有超出范围，则只警告，但能成功操作并四舍五入删除多余的小数位后保存，例如在FLOAT(5,2)列内插入999.009，近似结果是999.01。若四舍五入后，整数部分超出范围，则MySql报错，并拒绝处理。如999.995和-999.995都会报错。
* 说明：小数类型，也可以加unsigned，但是不会改变数据范围，例如：float(3,2) unsigned仍然只能表示0-9.99的范围。
* float和double在不指定精度时，默认会按照实际的精度（由实际的硬件和操作系统决定）来显示
* REAL就是DOUBLE ，如果SQL服务器模式包括REAL\_AS\_FLOAT选项，REAL是FLOAT的同义词而不是DOUBLE的同义词。

注意：在编程中，如果用到浮点数，要特别注意误差问题，因为**浮点数是不准确的，所以我们要避免使用“=”来判断两个数是否相等。**如果希望保证值比较准确，推荐使用定点数数据类型。

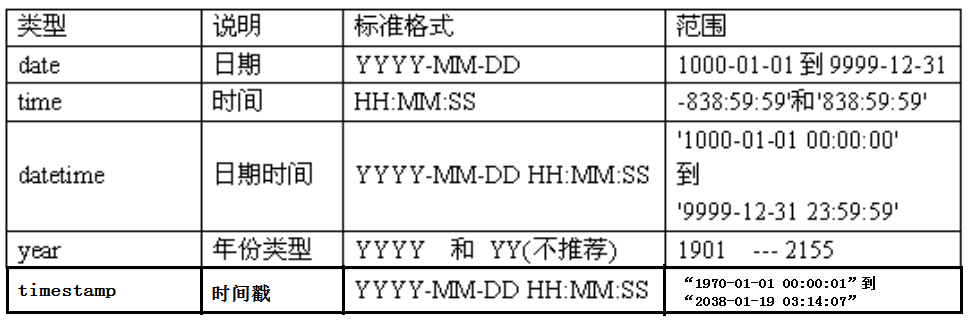
## 3、定点型

* DECIMAL在MySQL内部以字符串形式存放，**比浮点数更精确。**定点类型占M+2个字节
* DECIMAL(M,D)与浮点型一样处理规则。M的取值范围为0~65，D的取值范围为0~30，而且必须<=M，超出范围会报错。
* DECIMAL如果指定精度时，默认的整数位是10，默认的小数位为0。
* NUMERIC等价于DECIMAL。

#类型的选择：

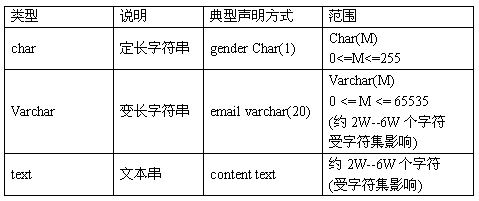
1.所选择的类型越简单越好，能保存数值的类型越小越好

## 4、日期时间类型



* 对于year类型，可以输入4位数，例如2018，也可以输入两位数，例如18，如果输入的是两位，“00-68”表示2000-2069年，“70-99”表示1970-1999年。
* 'YYYY-MM-DD HH:MM:SS'或'YY-MM-DD HH:MM:SS'，'YYYY-MM-DD'或'YY-MM-DD'格式的字符串。允许“不严格”语法：任何标点符都可以用做日期部分或时间部分之间的间割符。例如，'98-12-31 11:30:45'、'98.12.31 11+30+45'、'98/12/31 11\*30\*45'和'98@12@31 11^30^45'是等价的。
* 'YYYYMMDD'或'YYMMDD'格式的没有间割符的字符串，假定字符串对于日期类型是有意义的。例如，'19970523'和'970523'被解释为 '1997-05-23'，但'971332'是不合法的(它有一个没有意义的月和日部分)，将变为'0000-00-00'。
* 对于包括日期部分间割符的字符串值，如果日和月的值小于10，不需要指定两位数。'1979-6-9'与'1979-06-09'是相同的。同样，对于包括时间部分间割符的字符串值，如果时、分和秒的值小于10，不需要指定两位数。'1979-10-30 1:2:3'与'1979-10-30 01:02:03'相同。
* 数字值应为6、8、12或者14位长。如果一个数值是8或14位长，则假定为YYYYMMDD或YYYYMMDDHHMMSS格式，前4位数表示年。如果数字 是6或12位长，则假定为YYMMDD或YYMMDDHHMMSS格式，前2位数表示年。其它数字被解释为仿佛用零填充到了最近的长度。
* 一般存注册时间、商品发布时间等，不建议使用datetime存储，而是使用时间戳，因为datetime虽然直观，但不便于计算。而且timestamp还有一个重要特点，就是和时区有关。还有如果插入NULL，会自动设置为当前系统时间。

## 5、字符串型



Char和varchar保存短文本

Text保存长文本

Blob 较大的二进制

### char,varchar,text区别\*\*\*

* char是一种固定长度的类型，varchar则是一种可变长度的类型，它们的区别是：
* char如果不指定(M)则表示长度默认是1个字符。varchar必须指定(M)。
* char(M)类型的数据列里，每个值都占用M个**字符**，如果某个长度小于M，MySQL就会在它的右边用空格字符补足（在检索操作中那些填补出来的空格字符将被去掉；如果存入时右边本身就带空格，检索时也会被去掉）；在varchar(M)类型的数据列里，每个值只占用刚好够用的字符再加上一个到两个用来记录其长度的字节（即总长度为L字符+1/2字字节）。
* 由于某种原因char 固定长度，所以在处理速度上要比varchar快速很多，但相对费存储空间，所以对存储不大，但在速度上有要求的可以使用char类型，反之可以用varchar类型来实例。
* text文本类型，可以存比较大的文本段，搜索速度稍慢，因此如果不是特别大的内容，建议使用char，varchar来代替。还有text类型不用加默认值，加了也没用。而且text和blob类型的数据删除后容易导致“空洞”，使得文件碎片比较多，所以频繁使用的表不建议包含text类型字段，建议单独分出去，单独用一个表。

### 哪些情况使用char或varchar更好

* 一，存储很短的信息，比如门牌号码101，201……这样很短的信息应该用char，因为varchar还要占个byte用于存储信息长度，本来打算节约存储的现在得不偿失。
* 二，固定长度的。比如使用uuid作为主键，那用char应该更合适。因为他固定长度，varchar动态根据长度的特性就消失了，而且还要占个长度信息。
* 三，十分频繁改变的column。因为varchar每次存储都要有额外的计算，得到长度等工作，如果一个非常频繁改变的，那就要有很多的精力用于计算，而这些对于char来说是不需要的。
* MyISAM和MEMORY存储引擎中无论使用char还是varchar其实都是作为char类型处理的。
* 其他像InnoDB存储引擎，建议使用varchar类型，因为对于InnoDB数据表，内部的行存储格式并没有区分固定长度和可变长度列（所有数据行都使用指向数据列值的头指针），而且主要影响性能的因素是数据行使用的存储总量，由于char平均占用的空间多于varchar，所以除了简短并且固定长度的，其他考虑varchar。

M指的是最大的字符个数

Char 固定字符个数的字符串 耗费空间 效率高

Varchar 可变长度的字符串 节省空间 效率低

## 6、位类型（了解）

BIT数据类型可用来保存位字段值。BIT(M)类型允许存储M位值。M范围为1~64，默认为1。

BIT其实就是存入二进制的值，类似010110。如果存入一个BIT类型的值，位数少于M值，则左补0。如果存入一个BIT类型的值，位数多于M值，MySQL的操作取决于此时有效的SQL模式：如果模式未设置，MySQL将值裁剪到范围的相应端点，并保存裁减好的值。如果模式设置为traditional(“严格模式”)，超出范围的值将被拒绝并提示错误，并且根据SQL标准插入会失败。

对于位字段，直接使用SELECT命令将不会看到结果，可以用bin()或hex()函数进行读取。

## 7、二进制值类型（了解）

包括：xxxBLOB和xxxBINARY

BINARY和VARBINARY类型类似于CHAR和VARCHAR类型，但是不同的是，它们存储的不是字符字符串，而是二进制串。所以它们没有字符集，并且排序和比较基于列值字节的数值值。当保存BINARY(M)值时，在它们右边填充0x00(零字节)值以达到指定长度。取值时不删除尾部的字节。比较时所有字节很重要（因为空格和0x00是不同的，0x00<空格），包括ORDER BY和DISTINCT操作。比如插入'a '会变成'a \0'。

BLOB是一个二进制大对象，可以容纳可变数量的数据。有4种BLOB类型：TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB。它们只是可容纳值的最大长度不同。分别与四种TEXT类型：TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT对应有相同的最大长度和存储需求。在TEXT或BLOB列的存储或检索过程中，不存在大小写转换。BLOB和TEXT列不能有默认值。BLOB或TEXT对象的最大大小由其类型确定，但在客户端和服务器之间实际可以传递的最大值由可用内存数量和通信缓存区大小确定。你可以通过更改max\_allowed\_packet变量的值更改消息缓存区的大小，但必须同时修改服务器和客户端程序。

## 8、枚举（ENUM）（了解）

MySql中的ENUM是一个字符串对象，其值来自表创建时在列规定中显式枚举的一列值：

* 可以插入空字符串""和NULL（如果运行NULL的话）。
* 如果你将一个非法值插入ENUM(也就是说，允许的值列之外的字符串)，如果是严格模式，将不能插入，如果是非严格模式，将选用第一个元素代替，并警告。
* ENUM最多可以有65,535个成员，需要2个字节存储。
* 当创建表时，ENUM成员值的尾部空格将自动被删除。
* 例如：enum(‘M’,’F’)

值的索引规则如下：

* 来自列规定的允许的值列中的值从1开始编号。
* 空字符串错误值的索引值是0。
* NULL值的索引是NULL。

Create table tab(

C1 enum(‘a’,’b’,’c’)

);

插入的时候，只能插入这三个值，差别的不报错但显示为空

## 9、集合（SET）（了解）

SET和ENUM类型非常类似，也是一个字符串对象，里面包含0~64个成员。

SET和ENUM存储上有所不同，SET是根据成员的个数决定存储的字节数。

SET和ENUM最主要的区别在于SET类型一次可以选择多个成员，而ENUM则只能选择一个。

例如：set(‘a’,’b’,’c’,’d’)

## 10、特殊的NULL值

Null特征：

（1）所有的类型的值都可以是null，包括int、float等数据类型

（2）空字符串””，不等于null，0也不等于null，false也不等于null

（3）任何运算符,判断符碰到NULL,都得NULL

（4）NULL的判断只能用is null,is not null

（5）NULL 影响查询速度,一般避免使值为NULL

面试：

为什么建表时,加not null default '' 或 default 0

答:不想让表中出现null值.

为什么不想要的null的值

答:（1）不好比较,null是一种特殊值,比较时,只能用专门的is null 和 is not null来比较.

碰到运算符,一律返回null

（2）效率不高,影响提高索引效果.

因此,我们往往,在建表时 not null default '' 或 default 0

# 七、SQL

SQL：Structure Query Language结构化查询语言，它是使用关系模型的数据库应用语言，由IBM上世纪70年代开发出来。后由美国国家标准局（ANSI）开始着手制定SQL标准，先后有SQL-86，SQL-89，SQL-92，SQL-99等标准。

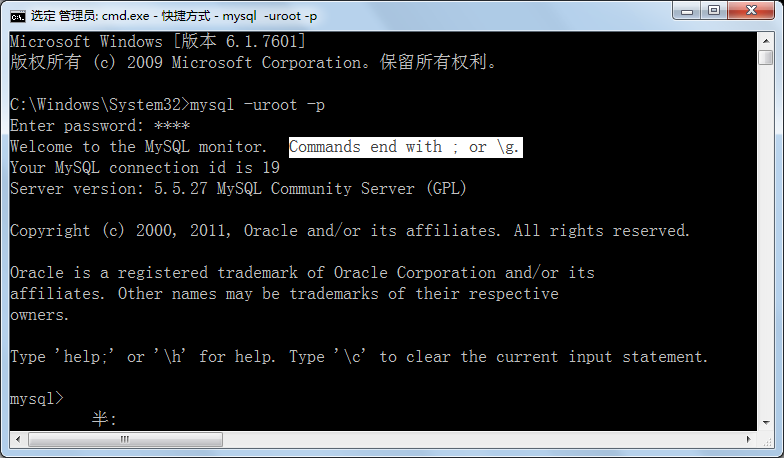
## 1、SQL的语言规范

* **mysql对于SQL语句不区分大小写，SQL语句关键字尽量大写 SHOW DATABASES;**
  + **至于表中的数据是否区分大小写和校对规则有关**
* 值，**除了数值型，字符串型和日期时间类型使用单引号（’’）**
  + **SELECT \* from employee where gender=’男’**
* **别名，尽量使用双引号（“”）**，而且不建议省略as
* 所有标点符号使用英文状态下的半角输入方式
* 必须保证所有(),单引号，双引号是成对结束的
* 可以使用（1）#单行注释 （2）--空格单行注释 （3）/\* 多行注释 \*/

命名规则：

* 数据库、表名不得超过30个字符，变量名限制为29个
* 必须只能包含 A–Z, a–z, 0–9, \_共63个字符
* 不能在对象名的字符间留空格
* 必须不能和用户定义的其他对象重名
* 必须保证你的字段没有和保留字、数据库系统或常用方法冲突
* 保持字段名和类型的一致性,在命名字段并为其指定数据类型的时候一定要保证一致性。假如数据类型在一个表里是整数,那在另一个表里可就别变成字符型了

在命令行中的要求：



说明：一个语句可以分开多行编写，以;或\g结束

## 3、SQL分类

SQL的分类：

* DDL（Data Definition Languages)：数据定义语言，这些语句定义了不同的数据段、数据库、表、列、索引等数据库对象。

主要的语句关键字包括create、drop、alter等。

* DML（Data Manipulation Language）：数据操作语句，**用于添加、删除、更新和查询数据库记录**，并检查数据完整性。

主要的语句关键字包括insert、delete、update、select等。

* DCL（Data Control Language）：数据控制语句，用于控制不同数据段直接的许可和访问级别的语句。这些语句定义了数据库、表、字段、用户的访问**权限和安全级别**。

主要的语句关键字包括grant、revoke等。

# 八、DDL

## 8.1 操作Database

注意：database不能改名。

一些可视化工具可以改名，它是建新库,把所有表复制到新库,再删旧库完成的。

### 1、创建数据库

**create database [if not exits] 数据库名 [charset 字符集];**

如果不指定字符集，则按照安装mysql服务时选择的默认字符集。

### 2、查看有哪些数据库

**show databases;**

提示：当前用户有权限查看的

### 3、删除数据库

**drop database 数据库名;**

### 4、选择数据库

**use 数据库名;**

### 5、查看当前正在使用哪个数据库

select database();

注意：要操作表格和数据之前必须先说明是对哪个数据库进行操作，否则就要对所有对象加上“数据库名.”。

### 6、修改库的字符集

alter database books character set gbk;

## 8.2 表结构的操作TABLE

修改表的存储引擎

alter table teacher engine innodb;

### 1、查看当前数据库的所有表格

**show tables;**  #前面必须有use 数据库名语句，否则报错ERROR 1046 (3D000): No database selected

**show tables from 数据库名;**

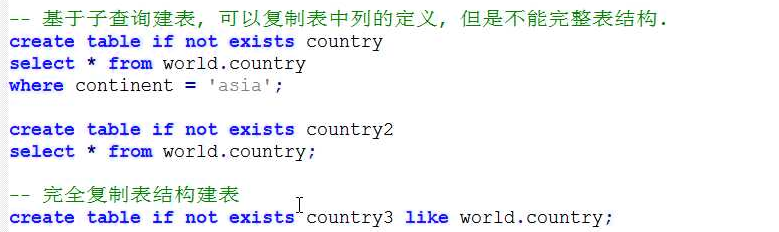
### 2、创建表

#### （1）基础版

|  |  |
| --- | --- |
| CREATE TABLE 表名称(  字段名1 数据类型1,  字段名2 数据类型2,  字段名3 数据类型3  ); | CREATE TABLE t\_stu(  sid INT,  sname VARCHAR(100),  gender CHAR  ); |

#### （2）详细版

|  |  |
| --- | --- |
| CREATE TABLE 表名称(  字段名1 数据类型1 主键 自增长,  字段名2 数据类型2 非空 默认值,  字段名3 数据类型3  )ENGINE=当前表格的引擎 AUTO\_INCREMENT=自增长的起始值 DEFAULT CHARSET=表数据的默认字符集; | CREATE TABLE t\_stu(  sid INT PRIMARY KEY AUTO\_INCREMENT,  sname VARCHAR(100) NOT NULL,  gender CHAR NOT NULL DEFAULT '男'  )ENGINE=INNODB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8; |



### 3、查看表结构

**desc 表名称;**

查看表的定义：SHOW CREATE TABLE 表名;

|  |
| --- |
|  |

### 4、删除表

drop table if exists 表名称;

注意：

数据和结构都被删除

所有正在运行的相关事务被提交

所有相关索引被删除

**DROP TABLE 语句不能回滚**

### 修改表

修改表的列相关：

Alter table 表名

add | drop | modify | change

修改表名

Alter table 表名 rename to 新表名

#### （1）修改表名

alter table 表名 rename 新表名;

rename table 表名 rename to 新表名;

#### （2）增加列

alter table 表名 add 【column】 列名 数据类型 【default 默认值】【not null】; #默认在最后

alter table 表名 add 【column】 列名 数据类型 【default 默认值】【not null】 after 某一列;

alter table 表名 add 【column】 列名 数据类型 【default 默认值】【not null】 first;

#### （3）删除列

alter table 表名 **drop** 【column】 列名;

#### （4）修改列类型/非空

alter table 表名 **modify** 【column】 列名 数据类型【default 默认值】【not null】;

alter table 表名 modify 【column】 列名 数据类型【default 默认值】【not null】 after 某一列;

alter table 表名 modify 【column】 列名 数据类型【default 默认值】【not null】 first;

#### （5）修改列名等

alter table 表名 **change** 【column】 列名 新列名 数据类型【default 默认值】【not null】;

### 约束【重点】

约束的含义：对字段中的值的限制，保证表中的数据的准确性，可靠性

约束的分类：

* **键约束：主键约束（非空+唯一）、外键约束、唯一键约束（值唯一不重复）**
* **Not NULL约束：字段值不能为空**
* **Check约束：检查约束（mysql不支持）**
* **Default约束：保证字段有默认值**
* **自增约束**

**按照约束的创建方式分类：**

**表级约束：在创建表的时候，所有的字段后面设置约束**

**列级约束：在创建表的时候，字段后面指定约束**

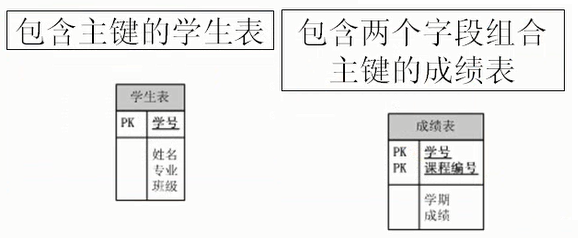
**额外说明：外键约束只能通过表级约束的方式设置；表级约束不支持非空和默认**

#### （1）查看某个表的约束和索引

|  |
| --- |
| 查看约束： |
| SELECT \* FROM **information\_schema.table\_constraints** WHERE table\_name = '表名称'; |
| **SHOW CREATE TABLE 表名;** |
| **查看索引：** |
| **SHOW INDEX FROM 表名称;** |

#### （2）主键约束PRIMARY KEY PK

主键：Primary key，简称PK，数据库主键作用保证实体的完整性



主键约束的特点：

（1）可以是一个列或多列的组合：

（2）主键约束 = **唯一约束+非空约束** 如果是多列组合的主键约束，那么这些列都不允许为空值，并且组合的值不允许重复。

（3）每个表有且最多只允许一个主键约束。

（4）MySQL的主键名总是PRIMARY，就算自己命名了主键约束名也没用。

（5）当创建主键约束时，**MySQL默认在对应的列上建立主键索引。删除主键时，也会直接删除主键索引。**

如何建立主键？

1. 建表时设置主键，有两种方式
2. 修改表时添加主键 DDL ALTER table XX ADD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **建表时指定主键** | | | | |
| 主键约束方式1：  **列级约束** | 主键约束方式2  **表级约束** | 复合主键  复合主键只能使用这种方式 | | |
| CREATE TABLE t\_stu(  sid INT PRIMARY KEY,  sname VARCHAR(100),  gender CHAR  );  **这种方法不能设置组合主键** | CREATE TABLE t\_course(  cid INT ,  cname VARCHAR(100),  decription VARCHAR(200),  PRIMARY KEY(cid)  ); | CREATE TABLE t\_stu\_course(  sid INT,  cid INT,  score DOUBLE(4,1),  PRIMARY KEY(sid,cid)  ); | | |
| **建表后添加主键约束** | | | | |
| alter table 表名称 add 【constraint 约束名】 primary key (字段名); | | | |
| alter table 表名称 add 【constraint 约束名】 primary key (字段名1，字段名2); | | | |
| ALTER TABLE t\_stu ADD PRIMARY KEY(sid); | ALTER TABLE t\_course ADD PRIMARY KEY(cid); | | ALTER TABLE t\_stu\_course ADD PRIMARY KEY(sid,cid); | |

如何删除主键和对应的索引？

|  |
| --- |
| 删除主键约束，不需要指定主键名，因为一个表只有一个主键，删除主键约束后，非空还存在  alter table表名称drop primary key; |

#### 唯一键Unique key，简称UK，

特点

* 同一个表可以有多个唯一约束。
* 唯一约束可以是某一个列的值唯一，**也可以多个列组合值的唯一。**
* MySQL会给唯一约束的列上默认创建一个**唯一索引**。
* 删除唯一键只能通过删除唯一索引的方式删除，删除时需要指定唯一索引名，唯一索引名就是唯一约束名一样。如果创建唯一约束时未指定名称，如果是单列，就默认和列名相同，如果是组合列，那么默认和()中排在第一个的列名相同。也可以自定义唯一性约束名。

如何建立唯一性约束？

|  |  |  |
| --- | --- | --- |
| 在某个列后面直接加唯一性约束 | 单独指定表的唯一性约束 | 组合列唯一性约束 |
| CREATE TABLE t\_course(  cid INT PRIMARY KEY,  cname VARCHAR(100) UNIQUE,  description VARCHAR(200)  ); | CREATE TABLE t\_stu(  sid INT PRIMARY KEY,  sname VARCHAR(100),  card\_id CHAR(18),  ~~CONSTRAINT uk\_card\_id~~ UNIQUE ~~KEY~~(card\_id)  );  #其中CONSTRAINT uk\_cname和KEY可以省略 | CREATE TABLE t\_stu\_course(  id INT PRIMARY KEY,  sid INT,  cid INT,  score DOUBLE(4,1),  CONSTRAINT uk\_sid\_cid UNIQUE KEY(sid,cid)  );  #其中CONSTRAINT uk\_sid\_cid和KEY可以省略 |
| 建表后增加唯一性约束 | alter table表名称 add 【constraint 约束名】 unique 【key】 (字段名);  alter table表名称 add 【constraint 约束名】 unique 【key】 (字段名1，字段名2); | |
| ALTER TABLE t\_course ADD CONSTRAINT uk\_cname UNIQUE KEY(cname);  #其中CONSTRAINT uk\_cname和KEY可以省略 | ALTER TABLE t\_stu ADD CONSTRAINT uk\_card\_id UNIQUE KEY(card\_id);  #其中CONSTRAINT uk\_cname和KEY可以省略 | ALTER TABLE t\_stu\_course ADD CONSTRAINT uk\_sid\_cid UNIQUE KEY(sid,cid);  #其中CONSTRAINT uk\_cname和KEY可以省略 |

如何删除唯一性约束和索引？

|  |
| --- |
| ALTER TABLE 表名称 DROP INDEX 唯一性约束名;  #注意：如果忘记名称，可以通过“show index from 表名称;”查看 |

主键和唯一键的区别：

（1）主键是非空，唯一键允许空

（2）主键一个表只能一个，唯一键可以有多个

#### （4）外键：Foreign key，简称FK

外键的作用：

外键是构建于两个字段之间的参照关系。可以是一个表的两个字段，也可以是两个表的两个字段。

外键的特点：

**（1）外键约束的名称：**在创建外键约束时，如果不指定外键约束名称，默认名不是列名，而是自动产生一个外键名（例如 student\_ibfk\_1;）。可以指定外键约束名。

**（2）外键约束有索引：**当创建外键约束时，系统默认会在所在的列上建立对应的**普通索引**。但是索引名是列名，不是外键的约束名。

1. 删除外键时，关于外键列上的普通索引需要单独手动删除。
2. 外键是在从表上建立，而且主表必须要先存在。
3. 从表的外键列，在主表中引用的只能是键列（主键，唯一键，外键）。
4. 从表的外键列与主表被参照的列名字可以不相同，但是数据类型必须一样
5. 一个表可以建立多个外键约束
6. 从表的外键值必须"在主表中能找到"或者为空，从而约束了从表的外键列的值的添加和修改。[主表键值决定了从表外键列值的范围，不可随意添加或者修改]
7. 当主表的记录被从表参照时，主表中被参考记录的删除和更新也会受到限制。[主表键值被参考，不能随意的更改或者删除，这样从表的外键值就对不上了]

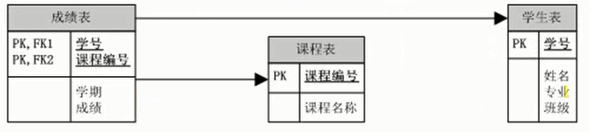
外键约束的五个约束等级

1. **Cascade：** 级联； 主动权在主表上，如果主表的被参考字段做了修改或删除，从表相应的也要修改或删除【删除一整行即一条记录】。
2. **SET NULL:** 级联置空；主动权在主表上，如果主表的被参考字段做了修改或删除，从表相应的需要将外键列中引用了的置为null 注意从表外键字段得允许为空才行
3. **No action:**
4. **Restric:** 和No action是同一种约束登记。 当主表的记录被从表参照时，主表的记录将不允许删除，如果要删除数据，需要先删除从表中依赖该记录的数据，然后才可以删除主表的数据。
5. **Set default：**mysql的InnoDB不支持。

默认情况下，主表和从表是严格依赖关系RESTRICT

对于外键约束,最好是采用: ON UPDATE CASCADE ON DELETE RESTRICT 的方式

如果要删除表，需要先删除从表，才能删除主表。建表时，先建主表，再建从表。



**如何建立外键约束？**

|  |  |
| --- | --- |
| 创建表时指定外键 | CREATE TABLE t\_department(  did INT PRIMARY KEY,  dname VARCHAR(100) NOT NULL UNIQUE,  description VARCHAR(200) NOT NULL  ); 主表  CREATE TABLE t\_employee(  eid INT PRIMARY KEY,  ename VARCHAR(100) NOT NULL,  dept\_id INT,  【CONSTRAINT fk\_emp\_dept\_did】  FOREIGN KEY(dept\_id) REFERENCES t\_department(did)  【 ON UPDATE CASCADE ON DELETE RESTRICT】  );从表  #其中CONSTRAINT fk\_emp\_dept\_did可以省略 这是起别名写的  #ON UPDATE CASCADE ON DELETE RESTRICT如果省略表示都是RESTRICT  **ON UPDATE CASCADE** **ON DELETE RESTRICT 用于控制两个列之间的约束关系，这里：修改（更新）时，是CASCADE的约束等级，删除时，是RESTRICT等级的约束** |
| 一个表可以有多个外键，而且主表和从表可以是一张表 | CREATE TABLE t\_emp(  eid INT PRIMARY KEY,  ename VARCHAR(100) NOT NULL,  manager\_id INT,  dept\_id INT,  CONSTRAINT fk\_emp\_dept\_did **FOREIGN KEY(dept\_id) REFERENCES t\_department(did) ON UPDATE CASCADE ON DELETE RESTRICT,**  CONSTRAINT fk\_emp\_mid\_eid **FOREIGN KEY(manager\_id) REFERENCES t\_emp(eid) ON UPDATE CASCADE ON DELETE RESTRICT**  );  #其中CONSTRAINT fk\_emp\_dept\_did可以省略  #ON UPDATE CASCADE ON DELETE RESTRICT如果省略表示都是RESTRICT |
| 建表后创建外键 | **alter table表名称 add 【constraint 约束名】 foreign key (从表字段名) references 主表名(主表被参照字段名);**  ALTER TABLE t\_emp ADD CONSTRAINT fk\_emp\_dept\_did FOREIGN KEY(dept\_id) REFERENCES t\_department(did) ON UPDATE CASCADE ON DELETE RESTRICT;  #其中CONSTRAINT fk\_emp\_dept\_did可以省略  #ON UPDATE CASCADE ON DELETE RESTRICT如果省略表示都是RESTRICT |

如何删除外键约束？

|  |
| --- |
| ALTER TABLE 表名称 DROP FOREIGN KEY 外键约束名;  ALTER TABLE t\_emp DROP FOREIGN KEY fk\_emp\_dept\_did; |
| **查看约束名**  SELECT \* FROM **information\_schema.table\_constraints** WHERE table\_name = '表名称';  **information\_schema.table\_constraints是系统库中记录约束的表** |

**如何删除外键列上的索引？需要单独删除**

|  |
| --- |
| ALTER TABLE 表名称 DROP INDEX 外键列索引名;  ALTER TABLE t\_emp DROP INDEX dept\_id; |
| 查看索引名  show index from 表名称; |

#### （5）非空约束

* NOT NULL 一个非空约束，只能规定某**1个**字段不能为空
* 非空约束不能存入null值
* 一个表中可以有很多个非空约束

|  |
| --- |
| CREATE TABLE t\_stu(  sid INT PRIMARY KEY,  sname VARCHAR(100) NOT NULL  ); |

如果某列已经创建好，那么可以修改列语句修改：

例如：原来非空，修改为允许空

|  |
| --- |
| ALTER TABLE ceshi\_table MODIFY des VARCHAR(20); |

例如：原来允许空，修改为非空 空改为非空 就去掉not null即可。

|  |
| --- |
| ALTER TABLE ceshi\_table MODIFY des VARCHAR(20) NOT NULL; des是字段名 |

#### （6）检查约束

注意： **MySQL不支持check约束，但可以使用check约束，而没有任何效果；**

例如：age tinyint check(age >20) 或 sex char(2) check(sex in(‘男’,’女’))

|  |
| --- |
| CREATE TABLE t\_stu(  sid INT PRIMARY KEY,  sname VARCHAR(100) NOT NULL,  gender CHAR NOT NULL **CHECK(gender IN('男','女'))**  ); |

#### （7）Default缺省约束

default：默认值，在插入数据时某列如果没指定其他的值,那么会将默认值添加到新记录。

|  |
| --- |
| CREATE TABLE t\_stu(  sid INT PRIMARY KEY,  sname VARCHAR(100) NOT NULL,  gender CHAR NOT NULL CHECK(gender IN('男','女'))  ); |

如果某列已经创建好，那么可以修改列语句修改：

例如：原来有默认值，去除默认值

|  |
| --- |
| ALTER TABLE ceshi\_table MODIFY des VARCHAR(20); |

例如：原来没有默认值，增加默认值

|  |
| --- |
| **ALTER TABLE ceshi\_table MODIFY des VARCHAR(20) DEFAULT '略';** |

#### （8）自增列：AUTO\_INCREMENT

自增长列又叫做标识列，可以不用手动插入值，系统提供默认序列值

（1）创建表时，指定标识列

|  |
| --- |
| CREATE TABLE t\_stu(  sid INT PRIMARY KEY **AUTO\_INCREMENT**,  sname VARCHAR(100) NOT NULL,  gender CHAR NOT NULL DEFAULT '男',  birthday DATE,  address VARCHAR(200)  ); |

1. **自增长auto\_increment的特点：**

[1]整数类型的字段才可以设置自增长。

[2]一个表最多只能有一个自增长列

[3]自增长列必须非空

[4]自增长列必须是键列（主键、外键、唯一键）。

[5]InnoDB表的自动增长列可以手动插入，手动插入按照插入的值来，但是插入的值如果是null或者0，则实际插入的将是自动增长后的值。

|  |
| --- |
|  |

如果自增长键中间不连续，插入一个中间值，该行数据会插入中间而不是插在后面。

取消自增

|  |
| --- |
| ALTER table 表名 modify 字段名 数据类型； |

添加自增

|  |
| --- |
| ALTER table 表名 modify 字段名 数据类型 auto\_increment； |

### 7、索引：INDEX

**索引**：索引是对数据库表中**一列或多列的值进行排序的一种结构**。**索引是一个单独的、物理的数据库结构**，它是某个表中一列或若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。由此可知，**索引是要消耗数据库空间的**。而**约束是一种逻辑概念。 用某一列或者多列来作为索引**



例如：一本字典，如何快速找到某个字，可以给字典加目录，对数据库来说**,索引的作用即是给"数据"加目录。**

设有N条随机记录,不用索引,平均查找N/2次,那么用了索引之后呢。如果是btree(二叉树)索引，，如果是hash(哈希)索引，时间复杂度是1。

**索引好处**：加快了查询速度(select )

**索引坏处**:降低了增,删,改的速度(update/delete/insert)，增大了表的文件大小(索引文件甚至可能比数据文件还大)

MySQL提供多种索引类型供选择：

* 普通索引：
* 唯一性索引：
* 主键索引：只有一个主键索引
* 全文索引：MySQL5.X版本只有MyISAM存储引擎支持FULLTEXT，并且只限于CHAR、VARCHAR和TEXT类型的列上创建。

MySQL的索引方法：

* HASH
* BTREE

MySQL中多数索引都以BTREE的形式保存。

**索引的使用原则：**

（1）不过度索引

（2）索引条件列(where后面最频繁的条件比较适宜索引)

（3）索引散列值,过于集中的值不要索引，例如:给性别"男","女"加索引,意义不大

CREATE INDEX 索引名 ON 表名称 (column\_name,[column\_name...]); 最左边的列最关键

alter table 表名称 drop index 索引名;

# 九、DML

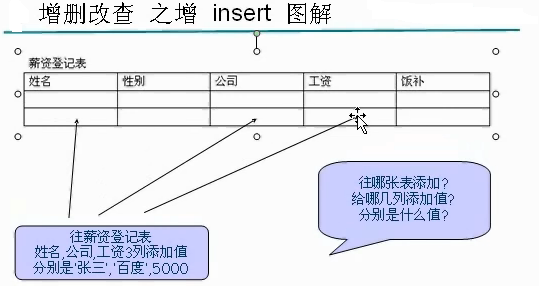
数据操纵语言（DML）DML用于增删改查，包括如下SQL语句：

**INSERT：添加数据到数据库中**

**UPDATE：修改数据库中的数据**

**DELETE：删除数据库中的数据**

## 9.1 插入数据



### 1、语法

|  |
| --- |
| INSERT INTO 表名称 VALUES(值1，值2，......); |
| INSERT INTO 表名称 VALUES(值1，值2，......),(值1，值2，......),...; 插入多行 |
| INSERT INTO 表名称 (字段1，字段2，......) VALUES(值1，值2，......); |
| INSERT INTO 表名称 (字段1，字段2，......) VALUES(值1，值2，......),(值1，值2，......),.....; |

### 2、说明

1、值列表(值1，值2，......)的顺序、个数与字段列表(字段1，字段2，......) 中字段的顺序、个数一致

（1）如果个数少了就报Column count doesn’t match value count

（2）如果VALUES前面的()中没有列出字段，那么默认就是为表中的所有字段赋值，那么个数与顺序与表结构中字段定义的一致

**2、关于自增长列、默认值列、允许为NULL列的赋值**

（1）**如果字段列表列出了字段名，那么值列表中就要为其赋值，哪怕它是自增长列、有默认值列、可以为NULL值的列。**

* **InnoDB表的自动增长列可以手动插入合适的值；但是插入的值如果是NULL或者0，则实际插入的将是自动增长后的值；**
* **如果列声明了“默认约束”那么对应的位置可以赋值具体的值；也可以使用“DEFAULT”，表示使用默认值；**
* **如果列允许了NULL值，那么可以为对应的字段可以赋值为具体值也可以赋值为NULL**

（2）**对于没有列出的字段，像自增列就自动赋值，像默认值列就自动赋默认值，像允许NULL的列就自动赋NULL值，但是非空列又没有提供默认值会自动赋值为对应数据类型的默认值，例如字符串赋值为空字符串，int赋值为0；**

3、VALUES也可以写成VALUE，但是VALUES是标准写法

4、可以同时插入多行

5、如果插入从表的数据，要注意查看主表参照字段的值是否存在

6、值的位置可以是常量值、表达式、函数

### 3、示例

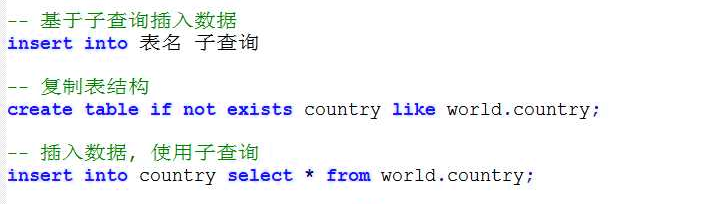
#### 练习（一）

|  |  |
| --- | --- |
| CREATE TABLE t\_stu(  sid INT PRIMARY KEY AUTO\_INCREMENT,  sname VARCHAR(100) NOT NULL,  gender CHAR NOT NULL DEFAULT '男',  card\_id CHAR(18) NOT NULL UNIQUE,  birthday DATE,  address VARCHAR(200)  ); |  |
| INSERT INTO t\_stu VALUES(1,'张三',DEFAULT,'123456789012345678','1989-09-09',NULL);  INSERT INTO t\_stu VALUES(2,'李四','女','123456789012345677','1988-09-09','尚硅谷');  INSERT INTO t\_stu VALUES(0,'王五','男','123456789012345676','1987-09-09','尚硅谷');  INSERT INTO t\_stu VALUES(NULL,'赵六','男','123456789012345675','1987-09-09','尚硅谷'); | |
| INSERT INTO t\_stu VALUES  (NULL,'冰冰','女','123456789012345674','1988-09-09','尚硅谷'),  (NULL,'小丽','女','123456789012345673','1988-09-09','尚硅谷'); | |
| INSERT INTO t\_stu (sname,card\_id,birthday)  VALUES('小薇','123456199012045672',STR\_TO\_DATE(SUBSTRING(card\_id,7,8),'%Y%m%d')); | |
| INSERT INTO t\_stu (sname,card\_id,birthday)VALUES  ('小红','123456789012345671','1990-09-09'),  ('小紫','123456789012345670','1990-09-09'); | |

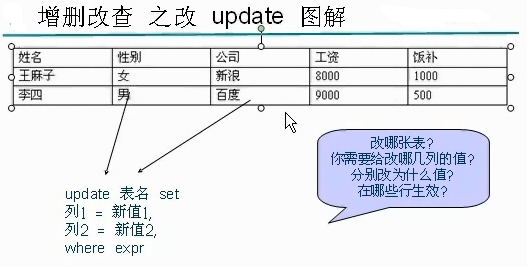
#### 练习（二）

|  |  |
| --- | --- |
| CREATE TABLE t\_department(  did INT PRIMARY KEY AUTO\_INCREMENT,  dname VARCHAR(100) NOT NULL,  description VARCHAR(200),  manager\_id INT  );  INSERT INTO t\_department(dname,description)  VALUES('教学部','技术培训'),  ('咨询部','课程咨询服务'); |  |
| CREATE TABLE `t\_job` (  `job\_id` INT(11) PRIMARY KEY AUTO\_INCREMENT,  `job\_name` VARCHAR(100) DEFAULT NULL,  `description` VARCHAR(200) DEFAULT NULL  );  INSERT INTO t\_job VALUES  (NULL,'JavaSE讲师','Java基础'),  (NULL,'Web讲师','Web基础'),  (NULL,'JavaEE框架','框架讲解'),  (NULL,'课程顾问','课程咨询'); |
| CREATE TABLE t\_employee(  eid INT PRIMARY KEY AUTO\_INCREMENT,  ename VARCHAR(100) NOT NULL,  gender CHAR NOT NULL DEFAULT '男',  card\_id CHAR(18) UNIQUE,  tel CHAR(11),  job\_id INT,  `mid` INT,  birthday DATE,  hiredate DATE,  address VARCHAR(100),  dept\_id INT,  FOREIGN KEY (dept\_id) REFERENCES t\_department(did),  FOREIGN KEY (job\_id) REFERENCES t\_job(job\_id)  );  INSERT INTO `t\_employee`(`eid`,`ename`,`gender`,`card\_id`,`tel`,`job\_id`,`mid`,`birthday`,`hiredate`,`address`,`dept\_id`)  VALUES (1,'孙红雷','男','123456789012345678','12345678901',1,NULL,'1990-01-01','2015-01-01','白庙',1),  (2,'张亮','男','123456789012345677','12345678902',2,NULL,'1990-01-02','2015-01-02','天通苑北',1),  (3,'鹿晗','男','123456789012345676','12345678903',3,NULL,'1990-01-03','2015-01-03','北苑',1),  (4,'邓超','男','123456789012345675','12345678904',2,NULL,'1990-01-04','2015-01-04','和谐家园',1),  (5,'孙俪','女','123456789012345674','12345678905',3,NULL,'1990-01-05','2015-01-05','霍营',1),  (6,'Angelababy','女','123456789012345673','12345678906',4,NULL,'1990-01-06','2015-01-06','回龙观',2); |
| CREATE TABLE t\_salary(  eid INT PRIMARY KEY,  basic\_salary DECIMAL(10,2),  performance\_salary DECIMAL(10,2),  commission\_pct DECIMAL(10,2),  deduct\_wages DECIMAL(10,2),  FOREIGN KEY (eid) REFERENCES t\_employee(eid)  );  INSERT INTO `t\_salary`(`eid`,`basic\_salary`,`performance\_salary`,`commission\_pct`,`deduct\_wages`)  VALUES (1,'12000.00','6000.00','0.40','0.00'),  (2,'9000.00','5000.00','0.20',NULL),  (3,'11000.00','8000.00',NULL,NULL),  (4,'13000.00','5000.00',NULL,NULL),  (5,'8000.00','8000.00','0.30',NULL),  (6,'15000.00','6000.00',NULL,NULL); |

基于子查询插入数据：



## 9.2 更新数据



### 1、语法

|  |
| --- |
| UPDATE 表名称 **SET** 字段名1 = 值1, 字段名2=值2,...... 【WHERE 条件】; |
| UPDATE 表1，表2,...... SET  **表1.字段名1 = 值1, 表1.字段名2=值2,表2.字段1 = 值1, 表2.字段2=值2......** 【WHERE 条件】; |

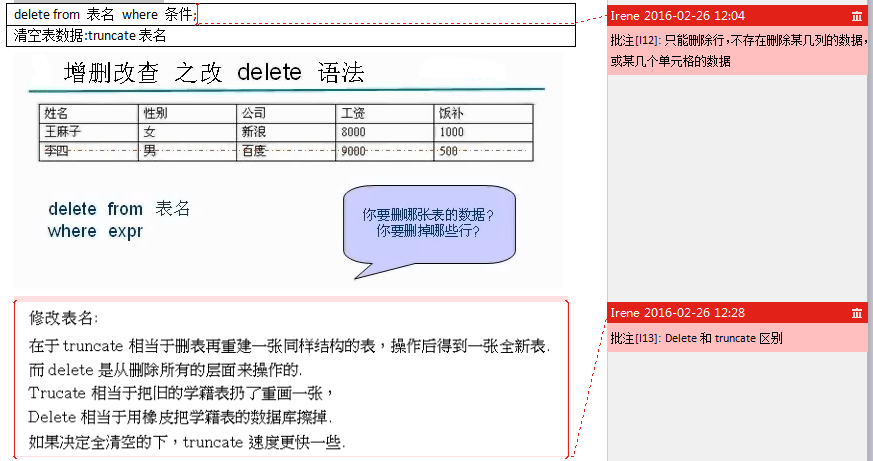
### 2、说明

1. **如果不写where条件，会修改所有行**
2. **值可以是常量值、表达式、函数**
3. **可以同时更新多张表**
4. **如果修改从表外键字段的数据，要注意查看主表参照字段的值是否存在**
5. 如果修改主表的被参考的字段的值，要注意查看从表的外键是否有依赖该值，如果有
6. 如果外键是on update RESTRICT或on update NO ACTION，那么要先处理从表的数据，才能修改
7. 如果外键是on update SET NULL 或 on update CASCADE，那么直接修改，从表的外键字段会自动处理

### 3、示例

|  |
| --- |
| #修改所有人的基本工资，涨薪5%  UPDATE t\_salary  SET basic\_salary = basic\_salary \* 1.05; |
| #修改"孙俪"的手机号码为"13709098765"，生日为"1982-09-26"  UPDATE t\_employee SET tel = '13709098765',birthday = '1982-09-26'  WHERE ename = '孙俪'; |
| #修改"邓超"的入职日期为今天  UPDATE t\_employee SET hiredate = CURDATE() WHERE ename ='邓超'; |
| #修改"咨询部"的主管id为6  UPDATE t\_department SET manager\_id =6 WHERE did = 2; |
| #修改"教学部"的主管id为1  UPDATE t\_department SET manager\_id =1 WHERE did = 1; |
| #修改"教学部"的主管id为"孙红雷"的编号  UPDATE t\_department,t\_employee  SET t\_department.manager\_id =t\_employee.eid  WHERE t\_department.`dname` = '教学部'  AND t\_department.`did` = t\_employee.`dept\_id`  AND t\_employee.ename = '孙红雷'; |
| #修改所有员工的领导编号为该员工所在部门的主管编号  UPDATE t\_employee,t\_department  SET t\_employee.mid = t\_department.manager\_id  WHERE t\_employee.dept\_id = t\_department.did; |
| #修改教学部的主管编号，以及该部门所有员工的领导编号为"邓超"的编号  UPDATE t\_department,t\_employee  SET t\_department.manager\_id =t\_employee.eid  WHERE t\_department.`dname` = '教学部'  AND t\_department.`did` = t\_employee.`dept\_id`  AND t\_employee.ename = '邓超';    UPDATE t\_employee,t\_department  SET t\_employee.mid = t\_department.manager\_id  WHERE t\_employee.dept\_id = t\_department.did  AND t\_department.`dname` = '教学部'; |

## 9.3 删除数据



### 1、语法

|  |
| --- |
| delete from 表名 【where 条件】; |
| delete 表1，表2,....... from 表1，表2，...... 【where 条件】; |

### 2、说明

**1、如果不加where条件，表示删除整张表的数据，表结构保留。**

delete from 表名;

删除整张表的数据还可以使用truncate 表名;

**区别：**

**truncate相当于删除表再重建一张同名结构的表，操作后得到一张全新表，而delete是在原有表中一行一行的删除数据可以回滚。如果决定清空一张表的数据，truncate速度更快一些。**

**TRUNCATE语句不能回滚**

2、如果删除主表的记录，要注意查看从表的外键是否有依赖该行的值，如果有

**（1）如果外键是on delete RESTRICT或on delete NO ACTION，那么要先处理从表的数据，才能删除**

**（2）如果外键是on delete SET NULL 或 on delete CASCADE，那么删除时从表的对应记录也会被置空或跟着删除**

3、可以一次删除多个表的数据

例如：两个表没有建立外键，但逻辑上有外键关系，也可以通过删除多个表的数据来实现级联删除

### 3、示例

|  |
| --- |
| #删除学号为9的学生信息  DELETE FROM t\_stu WHERE sid = 9; |
| #注意：前提是没有外键或外键是on delete cascade  #删除所有“教学部”的员工信息和薪资信息和“教学部”部门信息  DELETE t\_employee,t\_department,t\_salary  FROM t\_employee,t\_department,t\_salary  WHERE t\_department.`dname` ='教学部'  AND t\_employee.`dept\_id`=t\_department.`did`  AND t\_employee.`eid` = t\_salary.eid; |

## 9.4查询



### 1、语法

|  |
| --- |
| SELECT 查询列表  FROM 表名或视图列表  【WHERE 条件表达式】  【GROUP BY 字段名 【HAVING 条件表达式】】  【ORDER BY 字段 【ASC|DESC】】  【LIMIT m,n】; |

例如：

|  |
| --- |
| #查询表中的所有行所有列  #使用\*表示，查询所有字段，即查询所有行  select \* from t\_stu;  #查询部分字段，查询部分列  select sname,major from t\_stu;  #查询所有列，部分行  select \* from t\_stu where major = 'JavaEE';  #查询部分行，部分列  select sname,major from t\_stu where major = 'JavaEE'; |

说明：

1. 如果SELECT后面是\*，那么表示查询所有字段
2. SELECT后面的查询列表，可以是表中的字段，常量值，表达式，函数
3. **查询的结果是一个虚拟的表**
4. select语句，可以包含5种子句：依次是where、 group by、having、 order by、limit必须照这个顺序。

### 2、别名AS

语法：AS 别名

说明：

（1）可以给字段取别名、可以给表名取别名

（2）AS 可以省略

（3）**如果给字段取别名，如果别名中包含特殊符号，例如“空格”等，建议给别名加上双引号或单引号**

**（4）如果是给表名取别名，那么不能加双引号或单引号，也不能有特殊符号，例如“空格”等**

（5）建议别名简短，见名知意

示例：

|  |
| --- |
| UPDATE t\_department AS d,t\_employee AS e  SET d.manager\_id =e.eid  WHERE d.dname = '教学部'  AND d.did = e.`dept\_id`  AND e.ename = '孙红雷'; |
| #查询员工姓名以及手机号码  SELECT ename AS '员工姓名',tel AS '手机号码'  FROM t\_employee; |

### 3、去重DISTINCT

|  |
| --- |
| #查询员工表的部门编号  SELECT DISTINCT dept\_id FROM t\_employee; 在字段前面加distinct  #统计员工表中员工有几个部门  SELECT COUNT(DISTINCT dept\_id) FROM t\_employee; |

### 4、`着重号`

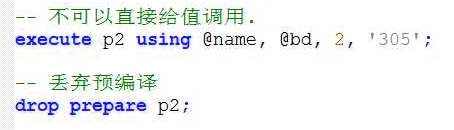
例如：select `name` from t\_stu;

可以给字段或表名加着重号

如果字段名或表名与关键字一样更要加着重号了

# 预编译





**？只能代替sql中值得部分，不能代替字段**

# 十、MySQL的运算符

（1）算术运算符：+ - \* /（除也可以写成div，div取整) %（取模可以写成mod）

（2）比较运算符：= > >= < <= !=（不等于还可以写成<>） <=>（安全等于）

（3）逻辑运算符：&&（逻辑与也可以写成and） ||（逻辑或也可以写成or） not（逻辑非） xor（逻辑异或）

（4）范围：表达式 between ... and ... （也可以写成 表达式>=... and 表达式 <=...）

表达式 not between ... and ...（也可以写成 表达式<... || 表达式 >...）

（5）集合：in (值，值，值...) not in(值，值，值...)

（6）模糊查询：LIKE NOT LIKE，通配符：%表示0-n个字符，\_下划线代表一个字符

（7）位运算符：&（按位与） |（按位或）^（按位异或）~（按位取反）>>（右移）<<（左移）

（8）NULL值判断，is null 或 is not null，如果使用null=null，null<>null,null=0,null<>0,null=false等都不对

不过xxx is null 可以使用xxx <=> null ，xxx is not null 可以写成 not xxx <=> null

结论：所有的运算符遇到NULL结果都是NULL，除了<=>

## 1、算术运算符

|  |
| --- |
| #一、算术运算符  #+,-,\*,/(div),%(mod)  #筛选出eid是偶数的员工  SELECT \* FROM t\_employee WHERE eid % 2 = 0;  SELECT \* FROM t\_employee WHERE eid MOD 2 = 0;  #查看每天的基本工资值，每个月按22天算  SELECT eid,basic\_salary/12 AS "日薪"  FROM t\_salary;  #div也表示除，但是只保留整数部分  SELECT eid,basic\_salary DIV 12 AS "日薪"  FROM t\_salary;  #关于+，在Java中，+的左右两边如果有字符串，那么表示字符串的拼接,但是在MySQL中+只表示数值相加，  #如果遇到非数值类型，先尝试转成数值，如果转失败，就按0计算  SELECT eid+ename FROM t\_employee;  SELECT eid+birthday FROM t\_employee;  #MySQL中字符串拼接要使用字符串函数实现  SELECT CONCAT(eid,":",ename) AS result FROM t\_employee; |

## 2、比较运算符

|  |
| --- |
| #二、比较运算符  #=，>, <，>=, <=， !=(不等于<>)，<=>（安全等于）  #查询basic\_salary!=10000  SELECT eid,basic\_salary FROM t\_salary WHERE basic\_salary != 10000;  SELECT eid,basic\_salary FROM t\_salary WHERE basic\_salary <> 10000;  #查询basic\_salary=10000，注意在Java中比较是==  SELECT eid,basic\_salary FROM t\_salary WHERE basic\_salary = 10000;  #查询commission\_pct等于0.40  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct = 0.40;  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct <=> 0.40;  #查询commission\_pct等于NULL  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct IS NULL;  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct <=> NULL;  #查询commission\_pct不等于NULL  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct IS NOT NULL;  SELECT eid,commission\_pct FROM t\_salary WHERE NOT commission\_pct <=> NULL; |

## 3、逻辑运算符

|  |
| --- |
| #三、逻辑运算符  #与&&，或||，非!  #与 AND，或 OR ，非 NOT，异或 XOR  #查询性别男，并且在90以前出生的员工  SELECT \* FROM t\_employee WHERE gender='男' AND birthday<'1990-01-01';  #查询职位编号job\_id是1或2的员工  SELECT \* FROM t\_employee WHERE job\_id =1 OR job\_id = 2;  #查询基本薪资是在9000-12000之间的员工编号和基本薪资  SELECT eid,basic\_salary FROM t\_salary WHERE basic\_salary >=9000 AND basic\_salary<=12000;  #查询基本薪资不在9000-12000之间的员工编号和基本薪资  SELECT eid,basic\_salary FROM t\_salary WHERE NOT (basic\_salary >=9000 AND basic\_salary<=12000);  SELECT eid,basic\_salary FROM t\_salary WHERE basic\_salary <9000 OR basic\_salary>12000; |

## 4、范围和集合

|  |
| --- |
| #四、表示区间范围和集合范围  #between ... and ... 和 not between ... and ...  #in(集合) 和 not in(...)  #查询基本薪资是在9000-12000之间的员工编号和基本薪资  SELECT eid,basic\_salary FROM t\_salary WHERE basic\_salary BETWEEN 9000 AND 12000;  #查询eid是1,3,5的基本工资  SELECT eid,basic\_salary FROM t\_salary WHERE eid IN (1,3,5); |

## 5、模糊查询

|  |
| --- |
| #五、模糊查询  #like 和 通配符 一起使用  #like \_ 匹配单个字符  #like % 匹配任意个字符  #查询名字中有'冰'字的员工信息  SELECT \* FROM t\_employee WHERE ename LIKE '%冰%';  #查询姓李的员工信息  SELECT \* FROM t\_employee WHERE ename LIKE '李%';  #查询姓李，名字就一个字的员工信息  SELECT \* FROM t\_employee WHERE ename LIKE '李\_';  #查询李冰冰的信息  SELECT \* FROM t\_employee WHERE ename LIKE '李冰冰'; |
| 通常情况，可以使用FIND\_IN\_SET()函数或LIKE操作符搜索SET值：  mysql> SELECT \* FROM tbl\_name WHERE FIND\_IN\_SET('value',set\_col)>0;  mysql> SELECT \* FROM tbl\_name WHERE set\_col LIKE '%value%';  第1个语句找出SET\_col包含value set成员的行。第2个类似，但有所不同：它在其它地方找出set\_col包含value的行，甚至是在另一个SET成员的子字符串中。  下面的语句也是合法的：  mysql> SELECT \* FROM tbl\_name WHERE set\_col & 1;  mysql> SELECT \* FROM tbl\_name WHERE set\_col = 'val1,val2';  第1个语句寻找包含第1个set成员的值。第2个语句寻找一个确切匹配的值。应注意第2类的比较。将set值与'val1,val2'比较返回的结果与同'val2,val1'比较返回的结果不同。指定值时的顺序应与在列定义中所列的顺序相同。  如果想要为SET列确定所有可能的值，使用SHOW COLUMNS FROM tbl\_name LIKE set\_col并解析输出中第2列的SET定义。  有什么实际应用呢？  比如我们设定用户的权限控制，一个用户可能会有多种权限，我们使用所有权限创建一个SET类型的字段，我们不需要用一系列int来定义各种权限了，直接使用一个SET字段即可：  /\*  用户权限permission表  \*/  create table user\_permission(  id int UNSIGNED not null auto\_increment,  user\_id int not null ,  permission set('阅读','评论','发帖') not null,  primary key(id),  unique (user\_id)  );  desc user\_permission;  insert into user\_permission values (0,1,'阅读'),(0,2,'阅读'),(0,3,'阅读,评论');  insert into user\_permission values (0,4,'阅读,评论,发帖');  select \*,permission+0 from user\_permission;  select permission from user\_permission where user\_id=1;  select \* from user\_permission where permission & 10;  SELECT \* FROM user\_permission WHERE FIND\_IN\_SET('评论',permission)>0; |

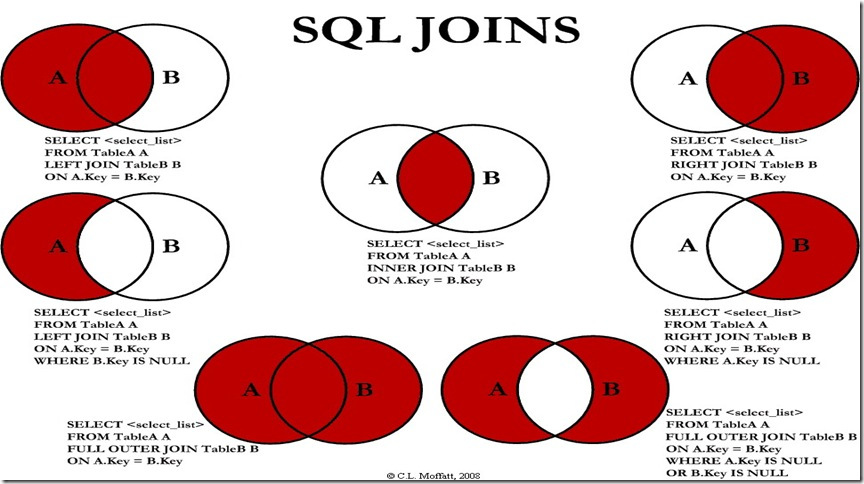
## 6、NULL值判断与计算处理

|  |
| --- |
| #NULL值判断与处理  #查询奖金百分比不为空的员工编号  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct IS NOT NULL;  #查询奖金百分比为空的员工编号  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct IS NULL;  #关于null值计算  #所有运算符遇到null都是null  #计算实际的薪资： basic\_salary + salary \* 奖金百分比  #函数：IFNULL(表达式,用什么值代替)  SELECT eid,basic\_salary + performance\_salary \*(1+ commission\_pct) FROM t\_salary;#错误的  SELECT eid,basic\_salary + performance\_salary \*(1+ IFNULL(commission\_pct,0)) FROM t\_salary;  #<=>安全等于  #查询奖金百分比为空的员工编号  SELECT eid,commission\_pct FROM t\_salary WHERE commission\_pct <=> NULL; |

## 7、位运算符

|  |
| --- |
| #七、位运算符  #>> << & | ~ ^（异或）  SELECT 2^3,2&3,2|3,2>>3,2<<3,~3; |

# 十一、关联查询，联合查询【重点】



作用:从2张或多张表中,取出有关联的数据.

关联查询一共有几种情况：

* 内连接：INNER JOIN 、CROSS JOIN
* 外连接：左外连接（LEFT OUTER JOIN）、右外连接（RIGHT OUTER JOIN）、**全外连接（FULL OUTER JOIN）**
* 自连接：当table1和table2本质上是同一张表，只是用取别名的方式虚拟成两张表以代表不同的意义

说明：

1. 连接 n个表,至少需要 n-1个连接条件。 例如：连接三个表，至少需要两个连接条件。
2. 当两个关联查询的表如果有字段名字相同，并且要查询中涉及该关联字段，那么需要使用表名前缀加以区分
3. 当如果表名比较长时，可以给表取别名，简化SQL语句

|  |  |
| --- | --- |
| 关联条件 |  |
| 表名前缀 | **否则报Column 'eid' in field list is ambiguous 说明字段名模糊不清** |
| 表别名 | **起别名在FROM 后的表名的后面，当起了别名后，其他位置出现表名时，都要用别名替换。** |

## 笛卡尔积

定义：将两（或多）个表的所有行进行组合，**连接后的行数为两（或多）个表的乘积数.**

在MySQL中如下情况会出现笛卡尔积，**因为缺少关联条件或者关联条件不准确**

**M\*N条**

注：外连接必须写关联条件，否则报语法错误

|  |
| --- |
| #笛卡尔积  #查询员工姓名和所在部门名称  SELECT ename,dname FROM t\_employee,t\_department;  SELECT ename,dname FROM t\_employee INNER JOIN t\_department;  SELECT ename,dname FROM t\_employee CROSS JOIN t\_department;  SELECT ename,dname FROM t\_employee JOIN t\_department; |

## 关联条件

表连接的约束条件可以有三种方式：WHERE, ON, USING

* WHERE：适用于所有关联查询
* **ON**：只能和JOIN一起使用，只能写关联条件。虽然关联条件可以并到WHERE中和其他条件一起写，但分开写可读性更好。
* USING()：只能和JOIN一起使用，而且要求**两个**关联字段在关联表中名称一致，而且只能表示关联字段值相等

|  |
| --- |
| #关联条件  Case1：把关联条件写在where后面  SELECT ename,dname FROM t\_employee,t\_department **WHERE** t\_employee.dept\_id=t\_department.did;  Case2:把关联条件写在on后面，只能和JOIN一起使用  SELECT ename,dname FROM t\_employee INNER JOIN t\_department ON t\_employee.dept\_id=t\_department.did;  SELECT ename,dname FROM t\_employee CROSS JOIN t\_department ON t\_employee.dept\_id=t\_department.did;  SELECT ename,dname FROM t\_employee JOIN t\_department ON t\_employee.dept\_id=t\_department.did;  Case3: 把关联字段写在using()中，只能和JOIN一起使用  #而且两个表中的关联字段必须名称相同，而且只能表示=  #查询员工姓名与基本工资  SELECT ename,basic\_salary FROM t\_employee INNER JOIN t\_salary USING(eid);  **#n张表关联，需要n-1个关联条件**  #查询员工姓名，基本工资，部门名称  SELECT ename,basic\_salary,dname FROM t\_employee,t\_department,t\_salary  WHERE t\_employee.dept\_id=t\_department.did AND t\_employee.eid=t\_salary.eid;  SELECT ename,basic\_salary,dname FROM t\_employee **INNER JOIN** t\_department **INNER JOIN** t\_salary  **ON** t\_employee.dept\_id=t\_department.did **AND** t\_employee.eid=t\_salary.eid; |

## 内连接（INNER JOIN）

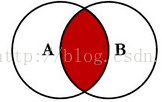
有两种，显式的和隐式的，返回连接表中符合连接条件和查询条件的数据行

**隐式：**SELECT [cols\_list] from 表1,表2 where [condition]

**显式：**SELECT [cols\_list] from 表1 INNER JOIN 表2 ON [关联条件] where [其他筛选条件]

SELECT [cols\_list] from 表1 CROSS JOIN 表2 ON [关联条件] where [其他筛选条件]

SELECT [cols\_list] from 表1 JOIN 表2 ON [关联条件] where [其他筛选条件]



|  |
| --- |
| #内连接  #查询员工姓名和所在部门名称 四种写法  SELECT ename,dname FROM t\_employee,t\_department WHERE t\_employee.dept\_id=t\_department.did;  SELECT ename,dname FROM t\_employee **INNER JOIN** t\_department **ON** t\_employee.dept\_id=t\_department.did;  SELECT ename,dname FROM t\_employee **CROSS JOIN** t\_department **ON** t\_employee.dept\_id=t\_department.did;  SELECT ename,dname FROM t\_employee **JOIN** t\_department **ON** t\_employee.dept\_id=t\_department.did;  #查询员工姓名，基本工资，部门名称  SELECT ename,basic\_salary,dname FROM t\_employee,t\_department,t\_salary  WHERE t\_employee.dept\_id=t\_department.did AND t\_employee.eid=t\_salary.eid;  SELECT ename,basic\_salary,dname FROM t\_employee **INNER JOIN** t\_department **INNER JOIN** t\_salary  ON t\_employee.dept\_id=t\_department.did AND t\_employee.eid=t\_salary.eid; |
|  |

## 外连接（OUTER JOIN）

外连接分为：

左外连接（LEFT OUTER JOIN），简称左连接（LEFT JOIN）

右外连接（RIGHT OUTER JOIN），简称右连接（RIGHT JOIN）

全外连接（FULL OUTER JOIN），简称全连接（FULL JOIN）。

### 左外连接（LEFT JOIN）

|  |  |
| --- | --- |
| A | A - A∩B |
| 返回左表中的所有行，**如果左表中的行在右表中没有匹配行，则结果中右表中的列返回空值。** | 返回左边中行在右表中没有匹配行的记录 |
| #左连接  #查询所有部门信息以及该部门员工信息  SELECT did,dname,eid,ename  FROM t\_department **LEFT OUTER JOIN** t\_employee  ON t\_department.did = t\_employee.dept\_id; | #查询部门信息，仅保留没有员工的部门信息  SELECT did,dname,eid,ename  FROM t\_department **LEFT OUTER JOIN** t\_employee  ON t\_department.did = t\_employee.dept\_id  WHERE t\_employee.dept\_id IS NULL;  #“从表外键列”是NULL |
|  |  |
| #查询所有员工信息，以及员工的部门信息  SELECT eid,ename,did,dname  FROM t\_employee LEFT OUTER JOIN t\_department  ON t\_employee.dept\_id = t\_department.did ; | #查询员工信息，仅保留没有分配部门的员工  SELECT eid,ename,did,dname  FROM t\_employee LEFT OUTER JOIN t\_department  ON t\_employee.dept\_id = t\_department.did  WHERE t\_employee.dept\_id IS NULL;  #“从表外键列”是NULL |
|  |  |

### 右外连接（RIGHT JOIN）

|  |  |
| --- | --- |
| B | B-A∩B |
| 恰与左连接相反，返回右表中的所有行，如果右表中行在左表中没有匹配行，则结果中左表中的列返回空值。 | 返回右表中在左表没有匹配行的记录 |
| #查询所有部门信息以及该部门员工信息  SELECT did,dname,eid,ename  FROM t\_employee RIGHT OUTER JOIN t\_department  ON t\_department.did = t\_employee.dept\_id; | #查询部门信息，仅保留没有员工的部门信息  SELECT did,dname,eid,ename  FROM t\_employee RIGHT OUTER JOIN t\_department  ON t\_department.did = t\_employee.dept\_id  WHERE t\_employee.dept\_id IS NULL;  #“从表外键列”是NULL |
|  |  |
| #查询所有员工信息，以及员工的部门信息  SELECT eid,ename,did,dname  FROM t\_department RIGHT OUTER JOIN t\_employee  ON t\_employee.dept\_id = t\_department.did ; | #查询员工信息，仅保留没有分配部门的员工  SELECT eid,ename,did,dname  FROM t\_department RIGHT OUTER JOIN t\_employee  ON t\_employee.dept\_id = t\_department.did  WHERE t\_employee.dept\_id IS NULL;  #“从表外键列”是NULL |
|  |  |

### 外连接（FULL JOIN）

mysql不支持FULL JOIN，但是可以用 left join **union** right join代替

|  |  |
| --- | --- |
| A∪B  A UNION B | A∪B - A∩B  A-A∩B UNION B-A∩B |
| #查询所有部门信息和员工信息  SELECT did,dname,eid,ename  FROM t\_department **LEFT OUTER JOIN** t\_employee  ON t\_department.did = t\_employee.dept\_id  **UNION**  SELECT did,dname,eid,ename  FROM t\_department **RIGHT OUTER JOIN** t\_employee  ON t\_department.did = t\_employee.dept\_id;  UNION前后的两个SELECT语句中，除了连接不一样，其他全一样。 | #查询所有没有员工的部门和没有分配部门的员工  SELECT did,dname,eid,ename  FROM t\_department **LEFT OUTER JOIN** t\_employee  ON t\_department.did = t\_employee.dept\_id  WHERE t\_employee.dept\_id IS NULL  UNION  SELECT did,dname,eid,ename  FROM t\_employee **LEFT OUTER JOIN** t\_department  ON t\_department.did = t\_employee.dept\_id  WHERE t\_employee.dept\_id IS NULL;  WHERE中的IS NULL用从表的，因为主表的主键不能为null |
|  |  |

## 自连接

当table1和table2本质上是同一张表，只是用取别名的方式虚拟成两张表以代表不同的意义。然后两个表再进行内连接，外连接等查询 要查询的信息都在一张表中

|  |
| --- |
| #自连接    所要查询的字段 员工的姓名和领导的姓名都在ename列中 重复了 此时就需要用自连接      #查询员工姓名以及领导姓名，仅显示有领导的员工  SELECT emp.ename,mgr.ename  FROM t\_employee AS emp, t\_employee AS mgr  WHERE emp.mid = mgr.eid;  #查询员工姓名以及领导姓名，仅显示有领导的员工  SELECT emp.ename,mgr.ename  FROM t\_employee AS emp INNER JOIN t\_employee AS mgr  ON emp.mid = mgr.eid;  #查询所有员工姓名及其领导姓名  SELECT emp.ename,mgr.ename  FROM t\_employee AS emp LEFT JOIN t\_employee AS mgr  ON emp.mid = mgr.eid; |

# select的5个子句【重点】

SELECT 字段名

FROM 表名、视图

WHERE 后面跟条件 选取哪几行 如果是所有数据，就不用加where

GROUP BY 字段

HAVING

ORDER BY

LIMIT

## 1、where条件查询

从原表中的记录中进行筛选，根据字段名和值选取符合的行。

## 2、group by 分组查询

很多情况下，用户都需要进行一些汇总操作，比如统计整个公司的人数或者统计每一个部门的人数等。

聚合函数【又叫做分组函数】 字段必须为数值类型

* AVG(【DISTINCT】 expr) 返回expr的平均值
* COUNT（【DISTINCT】 expr）返回expr的非NULL值的数目
* MIN(【DISTINCT】 expr)返回expr的最小值
* MAX(【DISTINCT】 expr)返回expr的最大值
* SUM（【DISTINCT】 expr）返回expr的总和

|  |
| --- |
| #聚合函数  #AVG(【DISTINCT】 expr) 返回expr的平均值  SELECT AVG(basic\_salary) FROM t\_salary;  #COUNT（【DISTINCT】 expr）返回expr的非NULL值的数目  #统计员工总人数  SELECT COUNT(\*) FROM t\_employee;#count(\*)统计的是记录数  #统计员工表的员工所在部门数  SELECT COUNT(dept\_id) FROM t\_employee;#统计的是非NULL值  SELECT COUNT(DISTINCT dept\_id) FROM t\_employee;#统计的是非NULL值，并且去重  #MIN(【DISTINCT】 expr)返回expr的最小值  #查询最低基本工资值  SELECT MIN(basic\_salary) FROM t\_salary;  #MAX(【DISTINCT】 expr)返回expr的最大值  #查询最高基本工资值  SELECT MAX(basic\_salary) FROM t\_salary;  #查询最高基本工资与最低基本工资的差值  SELECT MAX(basic\_salary)-MIN(basic\_salary) FROM t\_salary;  #SUM（【DISTINCT】 expr）返回expr的总和  #查询基本工资总和  SELECT SUM(basic\_salary) FROM t\_salary; |

### group by + 聚合函数

|  |
| --- |
| #group by + 聚合函数  #统计**每个部门**的人数  SELECT **dept\_id**,COUNT(\*) FROM t\_employee  GROUP BY **dept\_id**;  如果写 SELECT ename**,dept\_id**,COUNT(\*) FROM t\_employee  GROUP BY **dept\_id**;  这样写不会报错，但是语义不规范  #统计每个部门的平均基本工资  SELECT emp.dept\_id,AVG(s.basic\_salary )  FROM t\_employee AS emp,t\_salary AS s  WHERE emp.eid = s.eid  GROUP BY emp.dept\_id;  #统计每个部门的年龄最大者  SELECT dept\_id,MIN(birthday) FROM t\_employee GROUP BY dept\_id;  #统计每个部门基本工资最高者  SELECT emp.dept\_id,MAX(s.basic\_salary )  FROM t\_employee AS emp,t\_salary AS s  WHERE emp.eid = s.eid  GROUP BY emp.dept\_id;  #统计每个部门基本工资之和  SELECT **emp.dept\_id**,SUM(s.basic\_salary )  FROM t\_employee AS emp,t\_salary AS s  WHERE emp.eid = s.eid  GROUP BY **emp.dept\_id;**  #统计每一个部门 男女最高工资 |

#### 注意：

**用count(\*),count(1),谁好呢?**

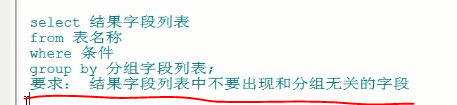
**其实,对于myisam引擎的表,没有区别的.**

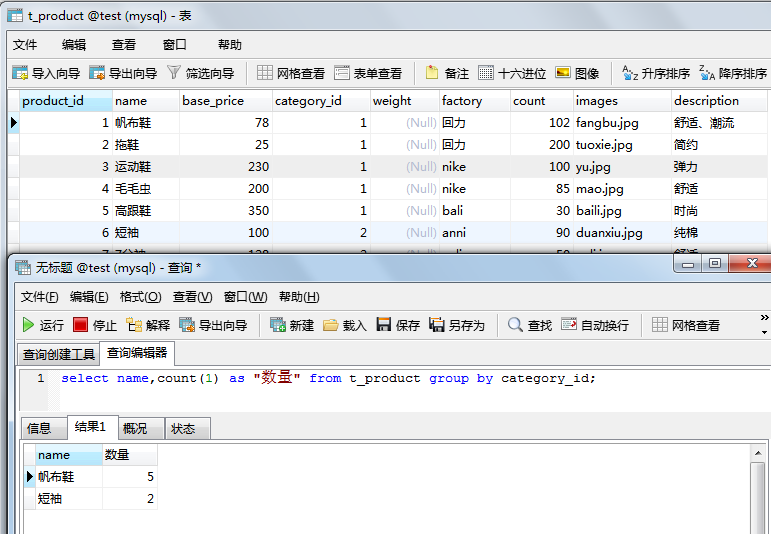
**这种引擎内部有一计数器在维护着行数.**

**Innodb的表,用count(\*)直接读行数,效率很低,因为innodb真的要去数一遍.**

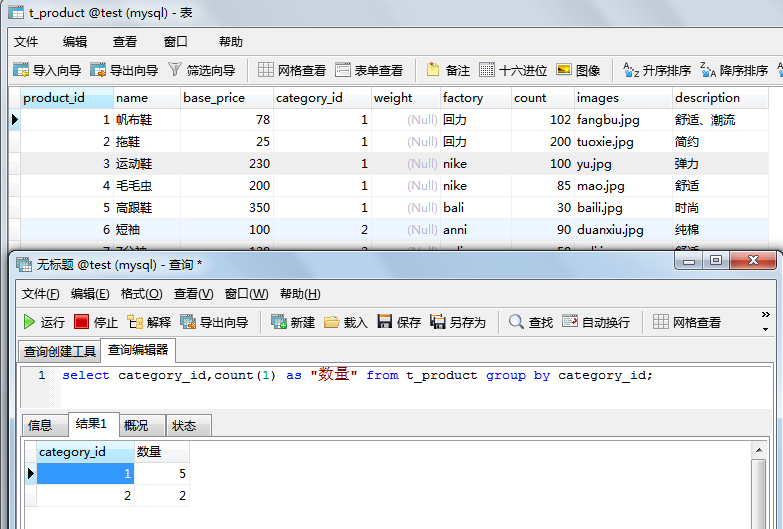
关于mysql的group by的特殊：

注意：在SELECT 列表中所有未包含在组函数中的列都应该是包含在 GROUP BY 子句中的，换句话说，**SELECT列表中最好不要出现GROUP BY子句中没有的列。**





对于标准语句来说，这个语句是错误的，但是mysql可以这么干，出于可移植性和规范性,不推荐这么写。



## having 筛选

Having是对统计结果再进行筛选

having与where类似,可筛选数据

having与where不同点

* **where针对表中的列发挥作用,查询数据；having针对查询结果中的列发挥作用,**筛选数据
* **where后面不能写分组函数**，而**having后面可以使用分组函数**
* **having只用于group by分组统计语句**

|  |
| --- |
| #按照部门统计员工人数，仅显示部门人数少于3人的  SELECT dept\_id,COUNT(\*) AS c  FROM t\_employee  WHERE dept\_id IS NOT NULL  GROUP BY dept\_id  **HAVING c <3;** |
| #查询每个部门的平均工资，并且仅显示平均工资高于10000  SELECT emp.dept\_id,AVG(s.basic\_salary ) AS avg\_salary  FROM t\_employee AS emp,t\_salary AS s  WHERE emp.eid = s.eid AND dept\_id IS NOT NULL  GROUP BY emp.dept\_id  HAVING avg\_salary >10000; |

## 4、order by 排序

* 按一个或多个字段对查询结果进行排序

用法：order by col1,col2,col3...

说明：先按col1排序如果col1相同就按照col2排序，依次类推

col1，col2，col3可以是select后面的字段也可以不是

* 默认是升序，也可以在字段后面加**asc显示说明是升序，desc为降序**

例如：order by click\_count desc;

如果两个字段排序不一样，例如：

order by 字段1 asc ,字段2 desc;

* order by 后面除了跟1个或多个字段，还可以写表达式，函数，别名等

|  |
| --- |
| #排序  #查询员工基本工资，按照基本工资升序排列，如果工资相同，按照eid升序排列  SELECT t\_employee.eid,basic\_salary FROM t\_employee INNER JOIN t\_salary  ON t\_employee.eid = t\_salary.eid  ORDER BY basic\_salary,eid;  #查询员工基本工资，按照基本工资降序排列，如果工资相同，按照eid排列  SELECT t\_employee.eid,basic\_salary FROM t\_employee INNER JOIN t\_salary  ON t\_employee.eid = t\_salary.eid  ORDER BY basic\_salary DESC,eid;  #统计每个部门的平均基本工资，并按照平均工资降序排列  SELECT emp.dept\_id,AVG(s.basic\_salary)  FROM t\_employee AS emp,t\_salary AS s  WHERE emp.eid = s.eid  GROUP BY emp.dept\_id  ORDER BY AVG(s.basic\_salary) DESC; |

## 5、limit 分页

limit m,n

**m表示从下标为m的记录开始查询，第一条记录下标为0，n表示取出n条出来，**如果从m开始不够n条了，就有几条取几条。m=(page-1)\*n，（page页码，n表示每页显示的条数）

如果第一页limit 0,n 1 2 3 4 5...n

如果第二页limit n,n

依次类推，得出公式limit (page-1)\*n , n

比如每页十个

第一页：0 10 第二页：10 10 第三页：20 10

每页n个

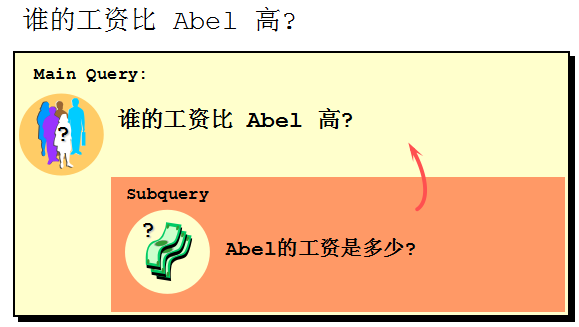
第一页：0 n 第二页：n n 第三页：2n n

|  |
| --- |
| #分页  #查询员工信息，每页显示5条，第二页  SELECT \* FROM t\_employee LIMIT 5,5;  #统计每个部门的平均基本工资，并显示前三名  SELECT emp.dept\_id,AVG(s.basic\_salary)  FROM t\_employee AS emp,t\_salary AS s  WHERE emp.eid = s.eid  GROUP BY emp.dept\_id  ORDER BY AVG(s.basic\_salary) DESC  LIMIT 0,3; |

# 十四、子查询

某些情况下，当进行一个查询时，需要的条件或数据要用另外一个 select 语句的结果，这个时候，就要用到子查询。

例如：



子查询的作用： 为主查询提供数据。 授限制性。

**为了给主查询（外部查询）提供数据而首先执行的查询（内部查询）被叫做子查询。**

**一般根据子查询的嵌入位置分为，where型子查询,from型子查询，exists型子查询。**

## 1、where型子查询

where型子查询即把**内层sql语句查询的结果作为外层sql查询的条件.**

* 子查询要包含在括号内。
* 建议将子查询放在比较条件的右侧。
* 单行操作符对应单行子查询，多行操作符对应多行子查询。
  + 单行操作符 右边子查询必须返回的是**单个值**，单行比较运算符（=,>,>=,<,<=,<>）
  + 多行操作符 右边子查询可以返回多行，但**必须是单列**，**ALL, ANY,IN 其中，ALL和ANY运算符必须与单行比较运算符（=,>,>=,<,<=,<>）结合使用**

|  |
| --- |
| IN：等于任何一个  **ALL：和子查询返回的所有值比较。例如：sal>ALL(1,2,3)等价于sal>1 && sal>2 && sal>3，即大于所有。**  **ANY：和子查询返回的任意一个值比较。例如：sal>ANY(1,2,3)等价于sal>1 or sal>2 or sal>3，即大于任意一个就可以。**  　　EXISTS：判断子查询是否有返回结果（不关心具体行数和内容），如果返回则为TRUE，否则为FALSE。 |

|  |
| --- |
| #where型子查询实例  #查询比**“孙红雷”的工资高**的员工编号  SELECT \* FROM t\_salary  WHERE basic\_salary > **(SELECT basic\_salary FROM t\_employee INNER JOIN t\_salary ON t\_employee.eid=t\_salary.eid WHERE t\_employee.ename='孙红雷')**; **子查询：查询孙红雷的工资**  #查询和孙红雷，李晨在同一个部门的员工 **子查询：孙红雷和李晨的部门**  SELECT \* FROM t\_employee  WHERE dept\_id **IN (SELECT dept\_id FROM t\_employee WHERE ename='孙红雷' OR ename = '李晨')**;  SELECT \* FROM t\_employee  WHERE dept\_id **= ANY**(SELECT dept\_id FROM t\_employee WHERE ename='孙红雷' OR ename = '李晨');  #查询全公司工资最高的员工编号，基本工资  SELECT eid,basic\_salary FROM t\_salary  WHERE basic\_salary **=** (SELECT MAX(basic\_salary) FROM t\_salary);  SELECT eid,basic\_salary FROM t\_salary  WHERE basic\_salary **>= ALL**(SELECT basic\_salary FROM t\_salary); |
|  |

## 2、from型子查询

from型子查询即把内层sql语句查询的结果作为临时表供外层sql语句再次查询.

|  |
| --- |
| #from型  #查询每个部门的编号、名称和人数  部门编号和名称都在部门表中  部门编号也在员工表中，部门的人数需要由员工表来统计    #找出比部门平均工资高的员工编号，基本工资  SELECT t\_employee.eid,basic\_salary  FROM t\_salary INNER JOIN t\_employee INNER JOIN (  **SELECT emp.dept\_id AS did,AVG(s.basic\_salary) AS avg\_salary**  **FROM t\_employee AS emp,t\_salary AS s**  **WHERE emp.eid = s.eid**  **GROUP BY emp.dept\_id**) AS temp  ON t\_salary.eid = t\_employee.eid AND t\_employee.dept\_id = temp.did  WHERE t\_salary.basic\_salary > temp.avg\_salary; |
|  |

## 3、exists型子查询

|  |
| --- |
| #exists型  #查询部门信息，该部门必须有员工  SELECT \* FROM t\_department  WHERE **EXISTS (SELECT \* FROM t\_employee WHERE t\_employee.dept\_id = t\_department.did);** |

## 4、复制表子查询（了解）

### （1）复制表

（1）拷贝表结构

CREATE TABLE newadmin LIKE admin;

（2）拷贝表结构和数据（但约束与索引除外）

CREATE TABLE newadmin AS ( SELECT \* FROM admin ) ;

（3）拷贝表结构+数据

CREATE TABLE newadmin LIKE admin;

INSERT INTO newadmin SELECT \* FROM admin;

（4）跨数据库拷贝表

CREATE TABLE newadmin LIKE shop.admin;

CREATE TABLE newshop.newadmin LIKE shop.admin;

（5）拷贝一个表中其中的一些字段（指定新名），其中一些数据

CREATE TABLE newadmin AS

(

SELECT id, username AS uname, password AS pass FROM admin WHERE id<10

) ;

（6）在创建表的同时定义表中的字段信息。

create table tt

(

eid int primary key auto\_increment

)

as

(

select employee\_id as eid,first\_name,last\_name,email from employees

);

### （2）复制数据

在 INSERT 语句中加入子查询。

不必书写 VALUES 子句。

子查询中的值列表应与 INSERT 子句中的列名对应。

INSERT INTO emp2

SELECT \* FROM employees WHERE department\_id = 90;

或

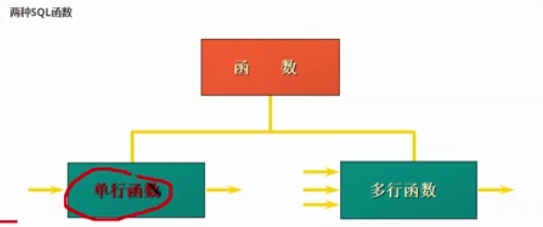
INSERT INTO sales\_reps(id, name, salary, commission\_pct)

SELECT employee\_id, last\_name, salary, commission\_pct

FROM employees

WHERE job\_id LIKE '%REP%';

# 单行函数



一行返回一个结果

多行函数是多行返回一个结果

**该章中的函数都是指单行函数：**

特点：

单行函数可以嵌套

参数可以是**一个字段**或者**一个表达式**或者**一个值**

MySQL数据库提供了很多函数包括：

* 数学函数；
* 字符串函数；
* 日期和时间函数；
* 条件判断函数；流程控制函数；
* 系统信息函数；
* 加密函数；
* 格式化函数；

## 1、数学函数

|  |  |
| --- | --- |
| ABS（x） | 返回x的绝对值 |
| CEIL（x） | 返回大于x的最小整数值 |
| FLOOR（x） | 返回大于x的最大整数值 |
| MOD(x,y) | 返回x/y的模 |
| RAND(x) | 返回0~1的随机值 |
| **ROUND(x,y)** | **返回参数x的四舍五入的有y位的小数的值** |
| TRUNCATE(x,y) | 返回数字x截断为y位小数的结果 |
| SQRT(x) | 返回x的平方根 |
| POW(x,y) | 返回x的y次方 |

## 2、字符串函数

|  |  |
| --- | --- |
| CONCAT(S1,S2,......,Sn) | 连接S1,S2,......,Sn为一个字符串 |
| CONCAT\_WS(s, S1,S2,......,Sn) | 同CONCAT(s1,s2,...)函数，但是每个字符串之间要加上s s是自定义的分隔符 |
| **CHAR\_LENGTH(s)** | 返回字符串s的**字符数** |
| **LENGTH(s)** | 返回字符串s的**字节数**，和字符集有关 |
| INSERT(str, index , len, instr) | 将字符串str从第index位置开始，len个字符长的子串替换为字符串instr |
| UPPER(s) 或 UCASE(s) | 将字符串s的所有字母转成大写字母 |
| LOWER(s) 或LCASE(s) | 将字符串s的所有字母转成小写字母 |
| LEFT(s,n) | 返回字符串s最左边的n个字符 |
| RIGHT(s,n) | 返回字符串s最右边的n个字符 |
| LPAD(str, len, pad) | 用字符串pad对str最左边进行填充，直到str的长度为len个字符 |
| RPAD(str ,len, pad) | 用字符串pad对str最右边进行填充，直到str的长度为len个字符 |
| LTRIM(s) | 去掉字符串s左侧的空格 |
| RTRIM(s) | 去掉字符串s右侧的空格 |
| TRIM(s) | 去掉字符串s开始与结尾的空格 |
| TRIM(【BOTH 】s1 FROM s) | 去掉字符串s开始与结尾的s1 |
| TRIM(【LEADING】s1 FROM s) | 去掉字符串s开始处的s1 |
| TRIM(【TRAILING】s1 FROM s) | 去掉字符串s结尾处的s1 |
| REPEAT(str, n) | 返回str重复n次的结果 |
| REPLACE（str, a, b） | 用字符串b替换字符串str中所有出现的字符串a |
| STRCMP(s1,s2) | 比较字符串s1,s2 |
| SUBSTRING(s,index,len) | 返回从字符串s的index位置其len个字符 |

## 3、日期时间函数

|  |  |
| --- | --- |
| CURDATE() 或 CURRENT\_DATE() | 返回当前日期 |
| CURTIME() 或 CURRENT\_TIME() | 返回当前时间 |
| NOW()  SYSDATE()  CURRENT\_TIMESTAMP()  LOCALTIME()  LOCALTIMESTAMP() | 返回当前系统日期时间 |
| YEAR(date)  MONTH(date)  DAY(date)  HOUR(time)  MINUTE(time)  SECOND(time) | 返回具体的时间值 |
| WEEK(date)  WEEKOFYEAR(date) | 返回一年中的第几周 |
| DAYOFWEEK() | 返回周几，注意：周日是1，周一是2，。。。周六是7 |
| WEEKDAY(date) | 返回周几，注意，周1是0，周2是1，。。。周日是6 |
| DAYNAME(date) | 返回星期：MONDAY,TUESDAY.....SUNDAY |
| MONTHNAME(date) | 返回月份：January，。。。。。 |
| DATEDIFF(date1,date2)  TIMEDIFF(time1, time2) | 返回date1 - date2的日期间隔  返回time1 - time2的时间间隔 |
| DATE\_ADD(datetime, INTERVALE expr type) | 返回与给定日期时间相差INTERVAL时间段的日期时间 |
| DATE\_FORMAT(datetime ,fmt) | 按照字符串fmt格式化日期datetime值 |
| STR\_TO\_DATE(str, fmt) | 按照字符串fmt对str进行解析，解析为一个日期 |

* （1）DATE\_ADD(datetime,INTERVAL expr type)

|  |  |
| --- | --- |
| SELECT DATE\_ADD(NOW(), INTERVAL 1 YEAR);  SELECT DATE\_ADD(NOW(), INTERVAL -1 YEAR); #可以是负数  SELECT DATE\_ADD(NOW(), INTERVAL '1\_1' YEAR\_MONTH); #需要单引号 | |
| 表达式类型 | YEAR\_MONTH |
| YEAR | DAY\_HOUR |
| MONTH | DAY\_MINUTE |
| DAY | DAY\_SECOND |
| HOUR | HOUR\_MINUTE |
| MINUTE | HOUR\_SECOND |
| SECOND | MINUTE\_SECOND |

* （2）DATE\_FORMAT(datetime ,fmt)和STR\_TO\_DATE(str, fmt)

|  |  |  |  |
| --- | --- | --- | --- |
| 格式符 | 说明 | 格式符 | 说明 |
| %Y | 4位数字表示年份 | %y | 表示两位数字表示年份 |
| %M | 月名表示月份（January,....） | %m | 两位数字表示月份（01,02,03。。。） |
| %b | 缩写的月名（Jan.，Feb.，....） | %c | 数字表示月份（1,2,3,...） |
| %D | 英文后缀表示月中的天数（1st,2nd,3rd,...） | %d | 两位数字表示月中的天数(01,02...) |
| %e | 数字形式表示月中的天数（1,2,3,4,5.....） |  |  |
| %H | 两位数字表示小数，24小时制（01,02..） | %h和%I | 两位数字表示小时，12小时制（01,02..） |
| %k | 数字形式的小时，24小时制(1,2,3) | %l | 数字形式表示小时，12小时制（1,2,3,4....） |
| %i | 两位数字表示分钟（00,01,02） | %S和%s | 两位数字表示秒(00,01,02...) |
| %W | 一周中的星期名称（Sunday...） | %a | 一周中的星期缩写（Sun.，Mon.,Tues.，..） |
| %w | 以数字表示周中的天数(0=Sunday,1=Monday....) |  |  |
| %j | 以3位数字表示年中的天数(001,002...) | %U | 以数字表示年中的第几周，（1,2,3。。）其中Sunday为周中第一天 |
| %u | 以数字表示年中的第几周，（1,2,3。。）其中Monday为周中第一天 |  |  |
| %T | 24小时制 | %r | 12小时制 |
| %p | AM或PM | %% | 表示% |

## 4、流程函数

|  |  |
| --- | --- |
| IF(value,t ,f) | 如果value是真，返回t，否则返回f |
| **IFNULL(value1, value2)** | **如果value1不为空，返回value1，否则返回value2** |
| CASE  WHEN 条件1 THEN result1  WHEN 条件2 THEN result2  ....  [ELSE resultn]  END | 相当于Java的if...else if... |
| CASE expr  WHEN 常量值1 THEN 值1  WHEN 常量值1 THEN 值1  ....  [ELSE 值n]  END | 相当于Java的switch |
|  | SELECT ename ,CASE  WHEN salary>=15000 THEN '高薪'  WHEN salary>=10000 THEN '潜力股'  WHEN salary>=8000 THEN '屌丝'  ELSE '草根'  END  FROM t\_employee; |
|  | SELECT oid,`status`, CASE `status`  WHEN 1 THEN '未付款'  WHEN 2 THEN '已付款'  WHEN 3 THEN '已发货'  WHEN 4 THEN '确认收货'  ELSE '无效订单'  END  FROM t\_order; |

## 5、其他函数

|  |  |
| --- | --- |
| 函数 | 描述 |
| database() | 返回当前数据库名 |
| version() | 返回当前数据库版本 |
| user() | 返回当前登录用户名 |
| password(str) | 返回字符串str的加密版本，41位长的字符串 |
| md5(str) | 返回字符串str的md5值，也是一种加密方式 |

## 分组函数

Select **COUNT(X)** from 表名； X是\*或者一个数都是 查记录数。

X是字段名时，统计该字段的非NULL值

SUM(): 求和

MAX():最大值

MIN()

AVG():平均值

# 十六、事务

思考：我去银行给朋友汇款,我卡上有1000元,朋友卡上500元,我给朋友转账50元(无手续费),如果,我的钱刚扣,而朋友的钱又没加时,网线断了,怎么办?

|  |
| --- |
| **（1）事务的定义：**一个或者一组sql语句组成一个执行单元，这个执行单元要么全部执行，要么全部不执行。  **（2）事务的组成：**   1. **一个或多个DML 语句** 2. 一个 DDL(Data Definition Language – 数据定义语言) 语句 3. 一个 DCL(Data Control Language – 数据控制语言) 语句     **事务：**是一些列增删改查（DML）组成的。  DDL和DCL自身一条语句就是一个事务，如果事务中有DCL或者DDL则会自动终结事务。 |

### 1、事务的ACID特性

* **原子性(Atomicity)：**原子意为最小的粒子，或者说不能再分的事物。数据库事务的不可再分的原则即为原子性。

组成事务的所有SQL必须：要么全部执行，要么全部取消（就像上面的银行例子）。

* **一致性(Consistency)：**几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致。
* **隔离性(Isolation)：**事务的执行不受其他事务的干扰，当数据库被多个客户端并发访问时，隔离它们的操作，防止出现：**脏读**、幻读、不可重复读。
* **持久性(Durability)：**对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失。

### 2、事务的使用

|  |
| --- |
| 使用事务的注意事项：   1. **只有 Innodb引擎才支持事务** 2. 和Oracle一样，DDL语句是不能回滚rollback的，并且部分的DDL语句会造成隐式的提交，因此最好事务中不要涉及DDL语句。 |
| **隐式事务和显示事务**  （1）show variable ‘autocommit’ 可以查看到mysql中默认情况下是自动提交的  （2）隐式事务就是自动提交，每次执行一个 SQL 语句时，如果执行成功，就会向数据库自动提交，而不能回滚。  insert ,update,delete都是自动提交  （3）显式事务的两种使用方式：  **[1]一次性事务的使用方式：**start transaction；然后写SQL，最终一旦rollback或commit就结束当次事务，之后的操作又自动提交。  **[2] 基于当前会话事务：**在建立当前会话（连接）之后，需要取消Mysql中自动提交事务的设置，进行手动提交事务，就需要设置set autocommit = false;或set autocommit = 0;那样的话每一句SQL都需要手动commit提交才会真正生效。rollback或commit之前的所有操作都视为一个事务，之后的操作视为另一个事务，还需要手动提交或回滚。**此方法只作用于当前连接。** |

|  |
| --- |
| **手动处理事务的两种方式：**  **方式1：基于当前会话的事务使用**  #开启手动处理事务模式  #set autocommit = false;  然后在本次连接中，所有的SQL执行都需要手动提交。否则不生效。  **方式2：一次性事务的使用（推荐）**  #开始事务1  start transaction;  事务1 SQL  commit;或者rollback;  #开始事务2  start transaction;  事务2 SQL  commit;或者rollback;  #剩下的都是自动提交的SQL  ……自动提交的SQL  ------------------------------------------------------------------------------------------------------------------------------------------------------------------  #查看当前表的数据  select \* from t\_stu\_detail;  #删除整张表的数据  delete from t\_stu\_detail;  #查询该表数据，发现显示删除后的结果  select \* from t\_stu\_detail;  #回滚  rollback  #查看当前表的数据，发现又回来了  select \* from t\_stu\_detail;  #删除整张表的数据  delete from t\_stu\_detail;  #提交事务  commit;  #查看当前表的数据，发现真删除了  select \* from t\_stu\_detail; |
| #插入一条记录  INSERT INTO t\_stu\_detail VALUES  (1, '123456789012345678', '1990-01-21', '12345678901', 'a@163.com', '北七家');  #保存还原点1  savepoint point1;  #插入一条记录  INSERT INTO t\_stu\_detail VALUES  (2, '123456789012345677', '1990-02-21', '12345678902', 'b@163.com', '北七家');  #保存还原点2  savepoint point2;  #查看当前效果  select \* from t\_stu\_detail;  #回滚到某个还原点  rollback to point1;  #提交事务  commit; |
| **#2.delete和truncate在事务使用时的区别**  **#delete演示**  Set autocommit = 0;  Start transaction;  Delete from account;  Rollback;  **#回滚，数据还在**  **#truncate演示**  Set autocommit = 0;  Start transaction;  truncate t\_stu\_detail;  Rollback;  #回滚，表结构还在，数据没了 |
| **#修改表结构**  alter table t\_stu\_detail add description varchar(50);  #回滚，对于修改表结构的语句无法回滚  rollback; |
| **#3. Savepoint的使用**  Set autocommit = 0;  Start transaction;  Delete from account where id = 25;  Savepoint a; **#设置保存点**  Delete from account where Id = 29;  Rollback to a;**#回滚到设置的保存点处**  **Savepoint只能搭配rollback使用，savepoint之前的sql无法回滚，savepoint后到rollbcak前的sql可以回滚** |

### 事务的隔离级别

|  |
| --- |
| 事务的隔离性要考虑多个线程操作同一个资源造成的多线程并发安全问题。  加锁可以非常完美的保证隔离性，但是这会造成数据库性能的大大下降。   * 如果两个事务并发的修改——必须隔离开 * 如果两个事务并发的查询——完全不用隔离 * 如果一个事务修改，另一个事务查询——脏读、不可重复读、虚读 |

多事务操作的**并发问题**: （问题的本质就是线程安全问题，共享数据的问题）

|  |
| --- |
| * **脏读: 一个事务读取到另一个事务还未提交的数据；**   对于两个事务 T1, T2； T1 读取到了已经被 T2 更新但还没有被提交的字段。之后, 若 T2 回滚, T1读取的内容就是临时且无效的.   * **不可重复读: 一个事务读取到另一个事务的更新并提交的数据**   对于两个事务 T1, T2,；T1 读取了一个字段, 然后 T2 **进行了更新并提交**了该字段。 之后, T1再次读取同一个字段, 值就不同了.   * **幻读: 一个事务读取到另一个事务插入并提交的数据**   对于两个事务 T1, T2, T1 从一个表中读取了一个字段, 然后 T2 在该表中**插入、删除**了一些新的行，但是还没有提交. 之后, 如果 T1 再次读取同一个表, 就会**多出、少了**几行  总结：  脏读就是说事务A读取到事务B还未提交的增删改  不可重复读和幻读是说事务A多次查询同一个结果，由于B事务进行了增删改的提交，导致A多次查询结果不同  不可重复读和幻读其实都可以归纳为不可重复读，只不过幻读是插入或者删除数据 |

为了解决上面的三个问题，数据库提出四大隔离级别

数据库事务的隔离性:隔离并发事务的能力，从而避免各种并发问题。隔离级别越高, 数据一致性就越好, 但并发性越弱。

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别\并发问题** | **脏读** | **不可重复读** | **幻读** |
| read uncommitted | **√** | **√** | **√** |
| Read committed | **×** | **√** | **√** |
| Repeatable read | **×** | **×** | **√** |
| serializable | **√** | **√** | **√** |
| **Oracle 支持的 2 种事务隔离级别：READ COMMITED和SERIALIZABLE.**  **Oracle 默认的事务隔离级别为: READ COMMITED 2**  **Mysql 支持 4 中事务隔离级别.**  **Mysql 默认的事务隔离级别为: REPEATABLE-READ 3** | | | |

|  |
| --- |
| 每启动一个 mysql 程序, 就会获得一个单独的数据库连接. 每个数据库连接都有一个变量 @@tx\_isolation, 表示当前的事务隔离级别.   * 查看当前的隔离级别: SELECT @@tx\_isolation; * 查看全局的隔离级别：select @@global.tx\_isolation; * 设置当前 mySQL 连接的隔离级别: set tx\_isolation ='repeatable-read'; * 设置数据库系统的全局的隔离级别: set global tx\_isolation ='read-committed'; |

|  |  |
| --- | --- |
| 隔离级别 | 描述 |
| READ-UNCOMMITTED | 允许事务读取其他事务未提交的数据，脏读、不可重复读、幻读的问题都会出现 |
| READ-COMMITTED | 只允许事务读取其他事务已经提交的数据，可以避免脏读，但是不可重复读、幻读的问题仍然会出现 |
| REPEATABLE-READ | 确保事务可以多次从一个字段中读取相同的值，好比在事务开启时对现有的数据进行了拍照，其他事务对数据的修改，不管事务是否提交，我这里读取的是拍照下来的数据，可以避免脏读和不可重复读，但幻读的问题仍然存在。  注意：INNODB使用了MVCC (Multiversion Concurrency Control)，即多版本并发控制技术防止幻读。真正的像拍照一样，其他事务新插入或删除的记录也看不出来。 |
| SERIALIZABLE | 确保事务可以从一个表中读取相同的行，在这个事务持续期间，禁止其他事务对该表执行插入、更新、删除操作，所有并发问题都可以避免，但是性能十分低下。 |

### 4、示例演示（数据库框架中讲解）

#### （1）脏读

脏读：进入餐厅发现“梦中情人”旁边座位已经有“帅哥”坐那儿了，正郁闷，打完饭，发现那个位置是空着的，又欣喜若狂，其实刚刚那个“帅哥”只是临时过去打个招呼。

|  |  |
| --- | --- |
| **读未提交级别脏读演示** | |
| 客户端A却读取到B未提交的脏数据 | 客户端B还未提交 |
| **设置隔离级别为读已提交** | |
| 客户端A读取不到B未提交的脏数据 | 客户端B还未提交 |

#### （2）不可重复读【更新并提交】

不可重复读：在图书馆门口，发现自己占的位置旁边有位“美女”，等刷完卡，兴冲冲的走到那儿，发现已经变成一“如花”了。

|  |  |
| --- | --- |
| A客户端在同一个事务中，前后两次读取同一条记录，值不同 | B客户端提交了新修改的数据 |
| A客户端在同一个事务中，前后两次读取同一条记录，值相同 | B客户端提交了新修改的数据 |

#### （3）幻读

大学考前画重点，老师说“第一章 xxxxxx概念”，你赶紧找，“天啊，在哪儿啊”，等你画完，就听老师说：“第四章xxxxx”，中间那些你都没听到。

|  |  |
| --- | --- |
| A客户端在同一个事务中，对同一个表的查询记录数不相同 | B客户端事务删除了数据，并提交 |
| A客户端在同一个事务中，对同一个表的查询记录数不相同 | B客户端事务添加了数据，并提交 |

#### （4）序列化

|  |  |
| --- | --- |
| A客户端的事务级别是序列化，A客户端开启了事务，且正在查看temp表 | B客户端想要多temp进行增、删、改操作不被允许  如果A客户端迟迟不结束事务，B客户端将会报  ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction |
|  |  |
|  |  |
|  | 但是B客户端对A客户端未涉及的表不受影响  **表级锁** |

# 十七、存储引擎

1.Mysql中的数据用不同的存储技术存储在文件或者内存中，存储就是就是存储引擎。

2.通过show egines； 查看mysql支持的存储引擎

3.mysql中用的最多的存储引擎有：innodb,myisam.memory等；只有innodb支持事务

4.myisam是5.5版本之前的默认的存储引擎

# 十八、视图

视图的含义：虚拟的表，和普通的表一样使用

Mysql5.1版本出现的新特性；通过表动态生成的数据

视图就是虚拟的结果集，是临时性的，是从普通表中查询出来的结果集。**视图只保存了sql逻辑**，并不保存查询结果，因此当表中的数据发生变化，视图中的数据也会有可能变化

应用场景：

* 1. 多个地方用到同样的sql查询结果
  2. 该查询结果使用的sql语句较复杂

相当于用视图封装一个sql，方便下次使用

|  |
| --- |
| **#1.创建视图**  语法：  Create view 视图名  As  查询语句；  #2.使用演示  Create view my\_v1  as  Select name,majorname  From stuinfo s  Inner join major m on s.’majorid’ = m.’id’;  Select \* from v1 where name like ‘张%’； |

# 十九、变量

系统变量：

全局变量

会话变量

自定义变量：

用户变量

局部变量

|  |
| --- |
| **#1.系统变量**  变量由系统提供，不是用户定义的，数据mysql服务器层面   1. 查看所有的系统变量   show global variables ;  show session variables;(等价于show variables)   1. 查看满足条件的部分系统变量   Show global|session variables like ‘%char%’;   1. 查看指定的某个系统变量的值   Select @@系统变量名；  Select @@global.系统变量名；  Select @@session.系统变量名   1. 为某个系统变量赋值   Set global|session 系统变量名 = 值；  Set @@global|session.系统变量名 = 值；  注意：  如果是全局级别，则要声明global  如果是会话级别，可以加session,也可以不写，默认为session的 |

|  |
| --- |
| 全局变量作用域为当前mysql服务，服务器重启后又恢复默认  默认是根据mysql服务器的配置文件来的  会话变量的作用域仅仅针对于当前会话 |
| **#2自定义变量**  使用步骤： 声明，赋值，使用  **#1.用户变量**  作用域：**当前会话有效**  #2.声明并初始化  Set @用户变量名=值；  Set @用户变量名:=值；  Select @用户变量名:=值；  案例：  Set @name=’john’;  Set @name=100;#变量是弱类型，想怎么赋值就怎么赋值  #3.赋值（更新用户变量的值）  两种方式：  方式1 set  Set @用户变量名=值；  Set @用户变量名:=值；  Select @用户变量名:=值；  方式2 select into  Select 字段 into @变量名  From 表；  #4.使用  Select @变量名  #5.用户变量可以写在任何地方  Begin end里面外面都可以  **#1.局部变量**  作用域：**仅仅在定义它的begin end中有效**。  应用在 begin end中的第一句话  使用：严格三步走：  **#1.声明**  Declare 变量名 类型；  Declare 变量名 类型 default 值；  **#2.赋值**  赋值要满足声明的类型  两种方式：  方式1 set  Set 局部变量名=值；  Set 局部变量名:=值；  Select 局部变量名:=值；  方式2 select into  Select 字段 into局部变量名  From 表；  #3.使用  Select 局部变量名；  #案例  Set @m=1;  Set @n=2;  Set @sum=@m+@n;  Select @sum; |

# 十九、存储过程和函数

存储过程和函数类似于java中的方法

好处：提高代码的重用性，简化操作

存储过程：一组预先编译好的SQL语句的集合，理解成批处理语句

存储过程两方面优势：

1. 封装好SQL，方便调用
2. 编译好了，再次使用不用经过编译了
3. 减少了和Mysql服务器的连接次数，提高效率；一次存储过程只需要一次连接

|  |
| --- |
| 存储过程的语法  #1.创建存储过程  Create procedure 存储过程名字(参数列表)  Begin  存储过程体（一组合法的SQL语句）  End  注意：  1.参数列表包含三个部分： 参数模式 参数名 参数类型  比如: in name varchar(20)  2.参数模式：  In : 该参数可以作为输入，需要传入值  Out：该参数可以作为输出，就是存储过程的返回值  Inout：即做输入也做输出  3.如果存储过程中只有一句sql，那么begin end可以省略  4.存储过程体中的每条SQL语句的结尾都必须加分号  5.存储过程的结尾可以使用delimiter设置结尾标记符。  #存储过程的调用  Call 存储过程名（参数）结束标记；  #1.空参列表  #案例：插入到admin表中五条记录  Delimiter $ #设置存储过程结束标记符  Create procedure myp1()  Begin  Insert into admin(username)  values(‘john1’,’0000’), (‘john2’,’0000’), (‘john3’,’0000’), (‘john4’,’0000’), (‘john5’,’0000’),  End $  调用：call myp1() $;  #2.带in模式参数的存储过程  #案例1：创建存储过程实现 根据女神名，查询对应的男神信息  Create producer myp2(in beautyName varchar(20))  Begin  Select bo.\* from boys bo  Right join beauty b on bo.id = b.boyfriend\_id  Where b.name=beautyName;  End $  调用 call myp2(‘刘岩’)；  #案例2：查询表中有没有该用户和密码  Create producer myp3(in username varchar(20),in password varchar(20))  Begin  Declare result int default 0; #声明局部变量并初始化  Select count(\*) into result #给变量赋值  From admin  Where admin.username = username #字段名和参数名冲突，用表名区别  And admin.password = password;  Select if(result>0,’存在’,’不存在’);#使用变量  End $  调用： call myp3(‘张飞’，‘8888’)$ |

# 二十、权限与安全

数据库的权限和数据库的安全是息息相关的，不当的权限设置可能会导致各种各样的安全隐患，操作系统的某些设置也会对MySQL的安全造成影响。

## 17.1 权限系统的工作原理

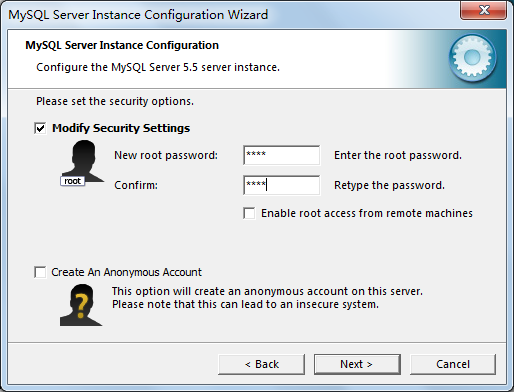
MySQL的权限系统通过下面两个阶段进行认证：

* 对连接到数据库的用户进行身份认证，以此来判断此用户是否属于合法的用户，合法的用户通过认证，不合法的用户拒绝连接
* 对通过认证的合法的用户则赋予相应的权限，用户可以在这些权限范围内对数据库做相应的操作

### 1、身份认证

对于身份认证，MySQL是通过IP地址和用户名联合进行确认的，也就是说，同样的一个用户名如果来自不同的IP地址，则MySQL将其视为不同的用户。

例如MySQL安装后默认创建的用户root@localhost表示用户root只能从本地(localhost)进行连接才可以通过认证，此用户从其他任何主机对数据库进行的连接都将被拒绝，除非安装时选择了（Enable root access from remote machines），那创建的就是root@%用户，就表示可以从任意主机通过root用户进行连接。



### 2、权限表的存取

在权限存取的两个过程中，系统会用到”mysql”数据库中user和db、host这三个最重要的权限表。

user表中的权限是针对所有数据库的，db表存储了某个用户对一个数据库的权限，host表中存储了某个主机对数据库的操作权限，配合db表对给定主机上数据库级操作权限做更细致的控制；但是很少用，新版本已经取消了host表。

当用户进行连接时，权限表的存取过程有一些两个阶段：

* 先从user表中的host、user和password这3个字段中判断连接的IP、用户名和密码是否存在与表中，如果存在，则通过身份验证，否则拒绝连接。
* 如果通过身份验证，则按照一些权限表的顺序得到数据库权限：user->db->tables\_priv->columns\_priv。在这几个权限表中，权限范围依次递减，全局权限覆盖局部权限。
  + 当用户通过权限认证，进行权限分配是时先检查全局权限表user，如果user中对应权限为Y，则此用户对所有数据库的权限都为Y，将不再检查db、tables\_priv和columns\_priv；如果为N，则到db表中检查此用户对应的具体数据库，并得到db中为Y的权限，如果db中相应权限为N，则检查tables\_priv中此数据库对应的具体表，取得表中为Y的权限，如果tables\_priv中相应的权限为N，则检查columns\_priv中此表对应的具体列，取得列中为Y的权限。

#### 用户表user

user表有39个字段。这些字段可以分为4类：

* 用户列：host，user，password三个字段
* 安全列：ssl\_type、ssl\_cipher、x509\_issuer、x509\_subject
  + ssl用于加密；x509标准可以用来标识用户。普通的发行版都没有加密功能。可以使用SHOW VARIABLES LIKE 'have\_openssl'语句来查看是否具有ssl功能。如果取值为DISABLED，那么则没有ssl加密功能。
* 资源控制列：max\_questions（每小时可以允许执行多少次查询）、max\_updates（每小时可以允许执行多少次更新）、max\_connections（每小时可以建立多少连接）、max\_user\_connections（单个用户可以同时具有的连接数）
  + 默认值为0，表示无限制。
* 权限列：
  + 这些字段的值只有Y和N。Y表示该权限可以用到所有数据库上；N表示该权限不能用到所有数据库上；通常，可以使用GRANT语句Wie用户赋予一些权限，也可以通过Update语句更新user表的方式来设置权限；不过，修改user表之后，一定要执行一下FLUSH PRIVILEGES

|  |  |
| --- | --- |
| 列 | 说明 |
| Select\_priv | 确定用户是否可以通过SELECT命令选择数据 |
| Insert\_priv | 确定用户是否可以通过INSERT命令插入数据 |
| Update\_priv | 确定用户是否可以通过UPDATE命令修改现有数据 |
| Delete\_priv | 确定用户是否可以通过DELETE命令删除现有数据 |
| Create\_priv | 确定用户是否可以创建新的数据库和表 |
| Drop\_priv | 确定用户是否可以删除现有数据库和表 |
| Reload\_priv | 确定用户是否可以执行刷新和重新加载MySQL所用各种内部缓存的特定命令，包括日志、权限、主机、查询和表  执行  flush-hosts,flush-logs,flush-privileges,flush-status,flush-tables,flush-threads,refresh,reload等命令的而全新 |
| Shutdown\_priv | 确定用户是否可以关闭MySQL服务器。在将此权限提供给root账户之外的任何用户时，都应当非常谨慎 |
| Process\_priv | 确定用户是否可以通过SHOW PROCESSLIST命令查看其他用户的进程 |
| File\_priv | 确定用户是否可以执行SELECT INTO OUTFILE和LOAD DATA INFILE命令，查看服务器主机上的文件 |
| Grant\_priv | 确定用户是否可以将已经授予给该用户自己的权限再授予其他用户 |
| References\_priv | 目前只是某些未来功能的占位符；现在没有作用 |
| Index\_priv | 确定用户是否可以创建和删除表索引 |
| Alter\_priv | 确定用户是否可以重命名和修改表结构 |
| Show\_db\_priv | 确定用户是否可以查看服务器上所有数据库的名字，包括用户拥有足够访问权限的数据库 |
| Super\_priv | 确定用户是否可以执行某些强大的管理功能，例如通过KILL命令删除用户进程，使用SET GLOBAL修改全局MySQL变量，执行关于复制和日志的各种命令 |
| Create\_tmp\_table\_priv | 确定用户是否可以创建临时表 |
| Lock\_tables\_priv | 确定用户是否可以使用LOCK TABLES命令阻止对表的访问/修改 |
| Execute\_priv | 确定用户是否可以执行存储过程 |
| Repl\_slave\_priv | 确定用户是否可以读取用于维护复制数据库环境的二进制日志文件。此用户位于主系统中，有利于主机和客户机之间的通信 |
| Repl\_client\_priv | 确定用户是否可以确定复制从服务器和主服务器的位置 |
| Create\_view\_priv | 确定用户是否可以创建视图 |
| Show\_view\_priv | 确定用户是否可以查看视图或了解视图如何执行 |
| Create\_routine\_priv | 确定用户是否可以更改或放弃存储过程和函数 |
| Alter\_routine\_priv | 确定用户是否可以修改或删除存储函数及函数 |
| Create\_user\_priv | 确定用户是否可以执行CREATE USER命令，这个命令用于创建新的MySQL账户 |
| Event\_priv | 确定用户能否创建、修改和删除事件 |
| Trigger\_priv | 确定用户能否创建和删除触发器 |

其他几个db、host、tables\_priv、columns\_priv权限表类似，可以通过desc查看字段

## 17.2 账号管理

### 1、创建或更改账号

#### 语句命令方式

有两种方法：

1. 使用GRANT语法创建（推荐）
2. 直接insert，update权限表

|  |
| --- |
| GRANT 权限类型列表  on object\_type {表名称|\*|\*.\*|db\_name.\*}  to user [identified by password ‘密码’][,user [identified by password ‘密码’] ....  with grant option;  其中：object\_type = TABLE | FUNCTION | PROCEDURE |

示例：

例1：创建用户admin，权限为可以在所有数据库上执行所有权限，但只能从本地进行连接

|  |
| --- |
| GRANT ALL PRIVILEGES ON \*.\* TO admin@localhost; |

可以发现除了Grant\_priv权限外，所有权限在user表里面都是Y.

例2：在例1基础上，增加对admin的grant权限

|  |
| --- |
| GRANT ALL PRIVILEGES ON \*.\* TO admin@localhost WITH GRANT OPTION; |

例3：在例2基础上，设置密码为“123”

|  |
| --- |
| GRANT ALL PRIVILEGES ON \*.\* TO admin@localhost IDENTIFIED BY '123' WITH GRANT OPTION; |

例4：创建新用户chai，可以从任何IP进行连接，权限为对test数据库的所有表进行SELECT、UPDATE、INSERT、DELETE操作，初始密码为“123”

|  |
| --- |
| GRANT SELECT,INSERT,UPDATE,DELETE ON test.\* TO 'chai'@'%' IDENTIFIED BY '123'; |

发现此例，user表中权限都是N，db表中增加的记录权限则都是Y。

注意：

* mysql数据库的user表中user的值为空，表示所有用户都可以连接（此处不能用\*，\*表示用户名为\*的用户了）
* mysql数据库的user表中host的值为\*或空，表示所有外部IP都可以连接，但是不包括本地服务器localhost，因此如果要包括本地服务器，必须单独为localhost赋予权限。如果host的值为%，表示所有IP，包括本地服务器localhost。

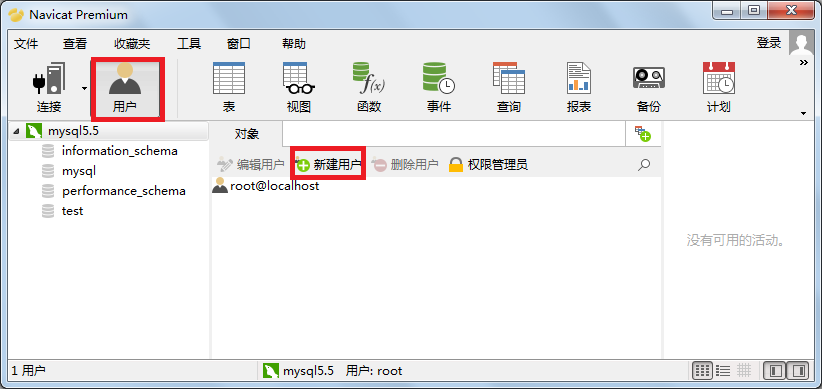
|  |  |  |
| --- | --- | --- |
| Host值 | User值 | 被条目匹配的连接 |
| ‘thomas.loc.gov’ | ‘fred’ | fred，从thomas.loc.gov连接 |
| ‘thomas.loc.gov’ | ‘’ | 任何用户，从thomas.loc.gov连接 |
| ‘%’ | ‘fred’ | fred，从任何主机连接 |
| ‘%’ | ‘’ | 任何用户，从任何主机连接 |
| ‘%.loc.gov’ | ‘fred’ | fred，从在loc.gov域的任何主机连接 |
| ‘x.y.%’ | ‘fred’ | fred，从在x.y.net，x.y.com等主机连 |
| ‘144.155.166.177’ | ‘fred’ | fred，从144.155.166.177的IP地址连接 |
| ‘144.155.166.%’ | ‘fred’ | fred，从‘144.155.166.\*的C类子网的任何主机连接 |

如果有多个匹配，服务器必须选择使用哪个条目，按照下述原则来解决：

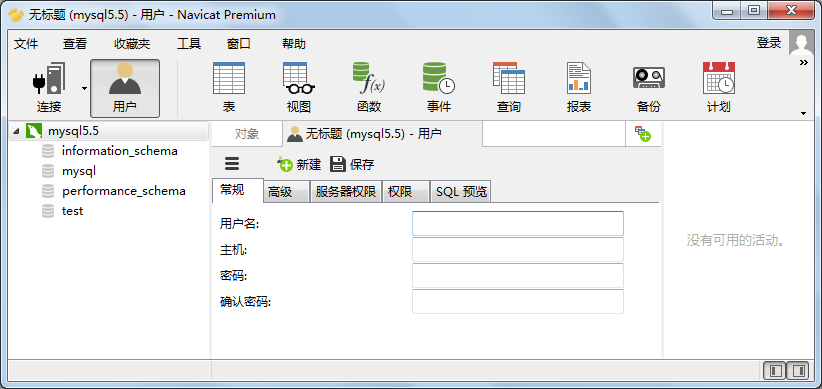
* 服务器在启动时读入user表后进行排序
  + 首先以最具体的host值排序，%排后面，如果host相同，以最具体的user值排序，空User值排后面。
* 服务器使用与客户端和用户名匹配的第一行

#### navicat图形化界面

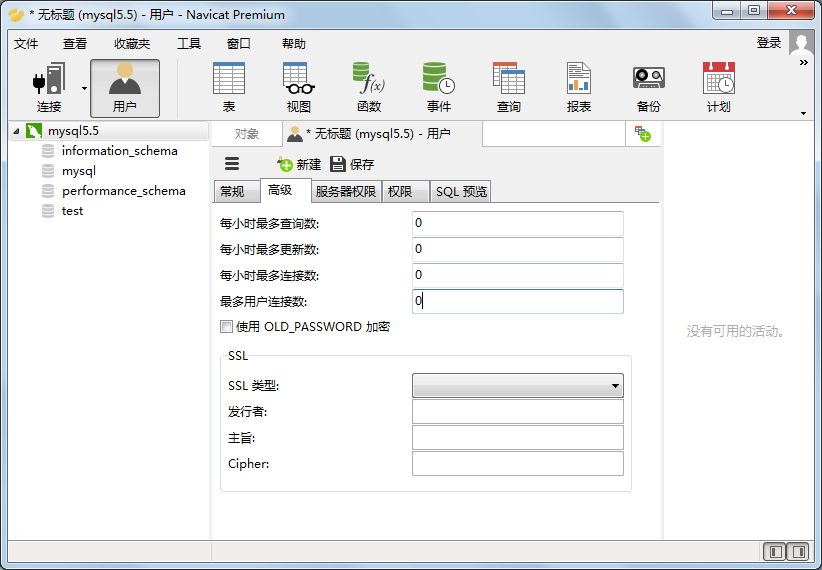
1、新建用户



2、填写用户名、主机和密码

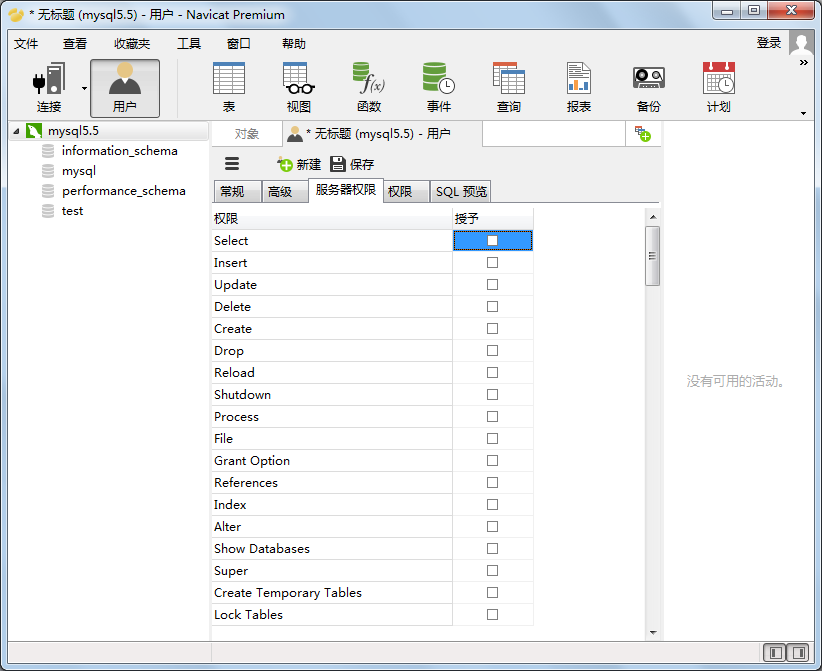


3、配use表的安全列和资源权限列的信息

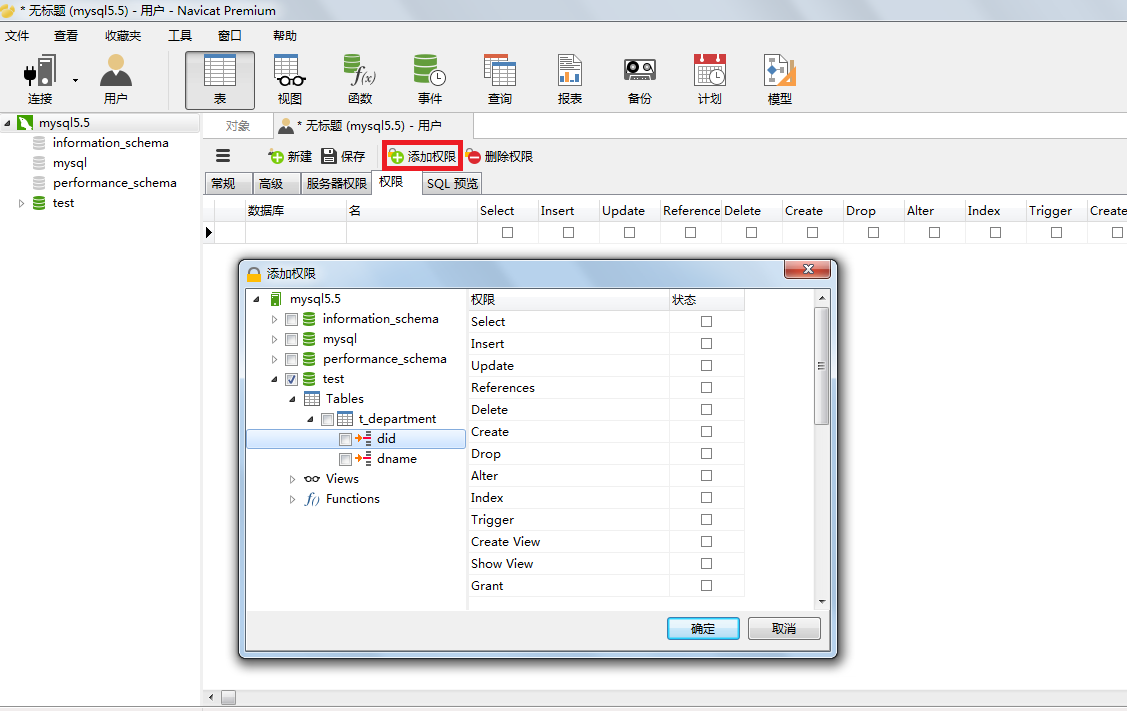


0表示不限制

4、选择user表的权限，这是针对所有数据库的全局权限

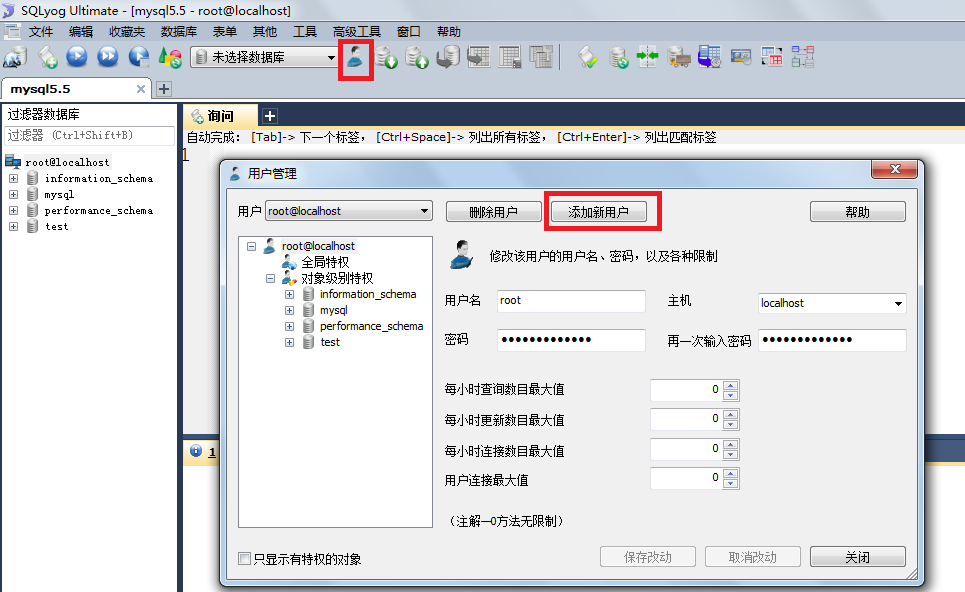


5、也可以设置单独某个数据库或某个表，或某个列的权限

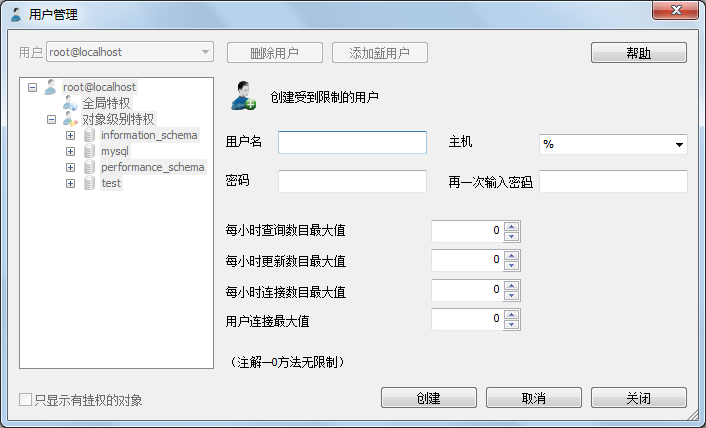


SQLyog图形化界面

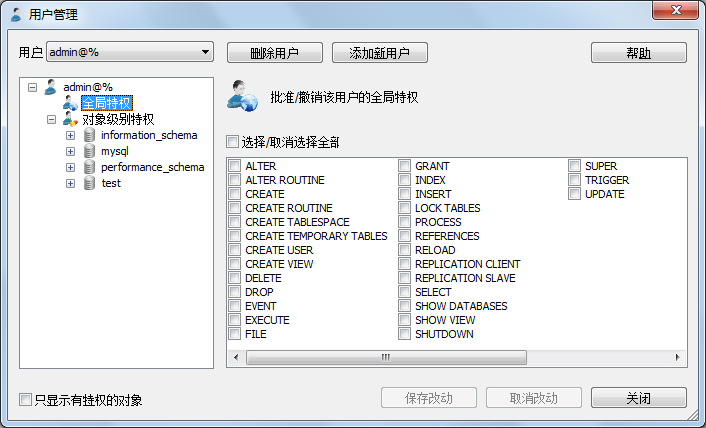
1、新建用户



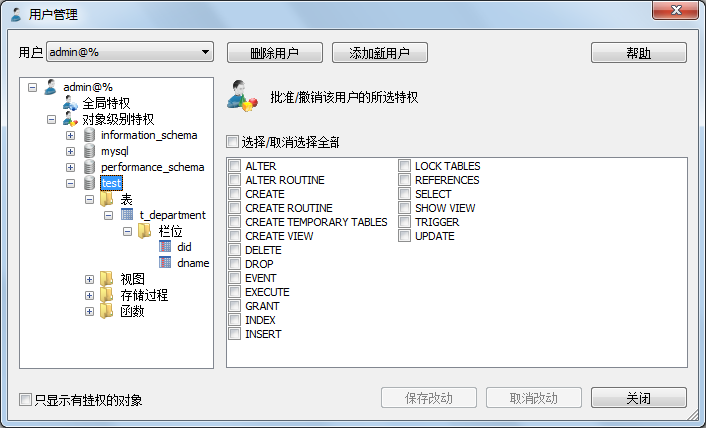
2、填写用户名密码和资源权限等



3、全局权限



4、某个库或某个表等局部权限



### 2、查看账户权限

show grants for user@host;

### 3、删除用户

drop user 用户名;

# 经典问题

1、在命令行出现乱码问题

按照数据库时选择utf8, 而我们在windows下窗口是GBK的,因此,需要在命令行客户端声明字符集.

set names gbk;是为了告诉服务器,客户端用的GBK编码,防止乱码.

mysql> set names gbk;

Query OK, 0 rows affected (0.00 sec)

可以查看字符集

mysql> show variables like 'character\_set\_%';

2、退出当前错误语句

语句打错以后应该退出本语句,再继续打新语句.

也可以打\c,快速退出本语句.

3、如何破解数据库的密码？安全模式登录

1:通过任务管理器或者服务管理,关掉mysqld(服务进程)

2:通过命令行+特殊参数开启mysqld

mysqld --skip-grant-tables

3:此时,mysqld服务进程已经打开,并且,不需要权限检查.

4:mysql -uroot 无密码登陆服务器.

5: 修改权限表

A: use mysql;

B: update user set Password = password('123456') where User = 'root';

C: flush privileges;

6:通过任务管理器,关掉mysqld服务进程.

7:再次通过服务管理,打mysql服务。

8:即可用修改后的新密码登陆.

1. 转义：mysql中转义不是/ 而是’