MySQL基础知识

# MySQL数据库基础

## mysql命令行程序使用

基本使用：mysql –u root –h localhost –p密码(p和密码之间没有空格)

输入语句含义：

* ；或者\g：代表语句结束
* \c：表示清除多行输入
* \G：表示垂直显示结果
* quit或者exit退出命令行使用程序。

简化连接方法：

方式一：在~/my.cnf中加入配置

|  |
| --- |
| [client] #默认为localhost  host=server\_host  user=root  password=123 |

方式二：利用shell别名

## 服务器的SQL模式

### Mysql SQL Mode简介

通常来说MySQL服务器能够工作在不同的SQL模式下，并能针对不同的客户端以不同的方式应用这些模式。这样，应用程序就能对服务器操作进行量身定制以满足自己的需求。这类模式定义了MySQL应支持的SQL语法，以及应该在数据上执行何种确认检查。这样，就能在众多不同的环境下、与其他数据库服务器一起更容易地使用MySQL。可以使用"--sql-mode="modes""选项，通过启动mysqld来设置默认的SQL模式。而从MySQL 4.1开始，也能在启动之后，使用SET [SESSION|GLOBAL] sql\_mode='modes'语句，通过设置sql\_  
mode变量更改其模式。（设置全局的sql模式需要有SUPER权限）

在linux下安装完mysql后，其默认的sql-mode值是空，在这种情形下mysql执行的是一种不严格的检查，例如日期字段可以插入'0000-00-00 00:00:00'这样的值，还有如果要插入的字段长度超过列定义的长度，那么mysql不会终止操作，而是会自动截断后面的字符继续插入操作，如下例：

|  |
| --- |
| mysql> create table t5 (c1 char(3));  mysql> insert into t5 values('abcd');  mysql> select \* from t5;  +------+  | c1 |  +------+  | abc |  +------+  1 row in set (0.00 sec) |

我们发现插入的字符被自动截断了，但是如果我们本意希望如果长度超过限制就报错，那么我们可以设置sql\_mode为STRICT\_TRANS\_TABLES，如下：

|  |
| --- |
| mysql> set session sql\_mode='STRICT\_TRANS\_TABLES'  再执行同样的操作，mysql就会告诉我们插入的值太长，操作被终止，如下：  mysql> insert into t5 values('abcd');  ERROR 1406 (22001): Data too long for column 'c1' at row 1 |

经常使用的sql\_mode值：

|  |  |
| --- | --- |
| **Sql\_mode值** | **描述** |
| ANSI | 更改语法和行为，使其更符合标准SQL。（双引号识别为一个标识符引用字符()，||识别为字符串连接运算符） |
| STRICT\_TRANS\_TABLES | 如果不能将给定的值插入到事务表中，则放弃该语句。对于非事务表，如果值出现在单行语句或多行语句的第1行，则放弃该语句。本节后面给出了更详细的描述。（严格对待接收到的坏数据） |
| TRADITIONAL | Make MySQL的行为象"传统"SQL数据库系统。该模式的简单描述是当在列中插入不正确的值时"给出错误而不是警告"。注释：一旦发现错误立即放弃INSERT/UPDATE。如果你使用非事务存储引擎，这种方式不是你想要的，因为出现错误前进行的数据更改不会"滚动"，结果是更新"只进行了一部分"。 |

**说明**：如果把sql\_mode的值设置成后面的两个值（也就是我们说的严格模式），那么当在列中插入或更新不正确的值时，mysql将会给出错误，并且放弃insert/update操作。**在我们的一般应用中建议使用这两种模式**，而不是使用默认的空或ANSI模式。但是需要注意的问题是，如果数据库运行在严格模式下，并且你的存储引擎不支持事务，那么有数据不一致的风险存在，比如一组sql中有两个dml语句，如果后面的一个出现了问题，但是前面的已经操作成功，那么mysql并不能回滚前面的操作。因此说设置sql\_mode需要应用人员权衡各种得失，从而得到一个合适的选择。

### SQL Mode与数据效验

SQL Mode 还可以实现对数据效验和转移等功能如：

* 效验日期数据合法性.
* 在INSERT或UPDATE过程中，如果被零除(或MOD(X，0))，则产生错误
* 将'"'视为识别符引号('`'引号字符)
* 禁用反斜线字符('\')做为字符串内的退出字符。启用NO\_BACKSLASH\_  
  ESCAPES模式，反斜线则成为普通字符。
* 将||视为字符串连接操作符（＋）(同CONCAT())，而不视为OR。

## MySQL标识符和命名规则

标识符里的合法字符。不加引号的标识符必须由系统字符集（utf8）中的字母和数字字符，再加上"\_"和"$"字符构成。标识符的第一个字符可以是允许用在标识符里的任何一种字符，包括数字。不过，不加引号的标识符不允许完全由数字字符构成，因为那会使它与数值难以区分。MySQL允许标识符以数字字符开头的做法在种类繁多的数据库系统中是不常见的。

标识符可以用反引号字符（" `"）括起来（加以界定），这意味着允许使用任意字符，只有取值为0或255的单字节例外。

内建函数的名字一般来说都不是保留字，可以不加引号地用作标识符。不过，如果启用了IGNORE\_SPACE SQL模式，函数名将被视为保留字，还想使用它们作为标识符就必须给它们加上引号。

注意：从MySQL 5.1.6版开始，把SQL语句里的标识符映射为目录名和文件名的机制经过了修改，使得早期版本里的部分非法字符也可以用在标识符里。具体地说，只需给标识符加上引号，在其中使用路径名字符（"/"或"\"）和"."字符就是合法的了。启用ANSI\_QUOTES还有额外效果--字符串值必须用单引号写出。如果使用了双引号，MySQL服务器将把该值解释为标识符而不是字符串。

## SQL大小写规则

SQL关键字和函数名：关键字和函数名不区分字母的大小写。

数据库、数据表和视图的名字：在服务器主机上，MySQL数据库和数据表用底层文件系统中的目录和文件表示。因而数据库和数据表名字的默认字母大小写情况将取决于服务器主机上的操作系统在文件名方面的规定。Windows文件名不区分字母的大小写，所以运行在Windows主机上的MySQL服务器也就不区分数据库和数据表名字的字母大小写。运行在Unix主机上的MySQL服务器往往要区分数据库和数据表名字的字母大小写，因为Unix文件系统是区分字母大小写的。Mac OS X平台上的HFS+文件系统中的名字是个例外，不区分字母的大小写。

存储程序的名字。存储函数、存储过程和事件的名字**不区分字母**的大小写。**触发器的名字要区分字母的大小写**，这一点和标准的SQL行为是不一样的。

数据列和索引名字：数据列和索引名字在MySQL环境里不区分字母的大小写。

假名的名字。在默认的情况下，**数据表假名区分字母的大小写**。可以使用任意的字母大小写组合（大写、小写或大小写混用）来给出一个假名，但如果需要在同一条语句里多次用到同一个假名，就必须让它们保持同样的字母大小写组合。如果lower\_case\_table\_names变量是非零值，数据表假名将不区分字母的大小写。

字符串值：字符串值是否区分字母大小写，这取决于它是二进制还是非二进制，而非二进制字符串还要取决于字符集的排序方式。

## mysql注释

|  |
| --- |
| （1） # 到该行结束 # 这个注释直到该行结束 mysql> SELECT 1+1;  （2）-- 到该行结束 -- 这个注释直到该行结束  （3）/\* 行中间或多个行 \*/  mysql> SELECT 1/\* 这是一个在行中间的注释 \*/ + 1;  mysql> SELECT 1+  /\* 这是一个  多行注释的形式  \*/  1;  自MySQL3.23 版以来，可在C 风格的注释中"隐藏" MySQL特有的关键字，注释以"/ \* !"而不是以" / \*"起头。MySQL查看这种特殊类型注释的内部并使用这些关键字，但其他数据库服务器将这些关键字作为注释的一部分忽略。  select /\*!50712 1,\*/ 2;、、版本大于5.7.12则会显示1。 |

## 字符集支持

MySQL字符集特性：

* 一个给定的字符集可以有一种或者多种排序规则
* 支持Unicode的字符集有：utf8和ucs2字符集，它们包括及基本多文种平面字符（BMP）；以及 utf16，utf32和utf8mb4字符集，他们包括BMP字符和补充字符。

### 指定字符集

CHARACTER SET charset 或者CHARSET charset

COLLATE collation

|  |
| --- |
| **--创建数据库指定**  CREATE DATABASE db\_name CHARSET charset COLLATE collation  **--创建表设置字符集和排序规则**  CREATE TABLE tb\_name (…) CHARSET charset COLLATE collation  **--列中设置**  c CHAR(10) CHARACTER SET utf8 |

### 确定可用字符集和当前设置

|  |
| --- |
| show character set;  show collation;  **show** **variables** **like** 'character\\_set\\_%';  Variable\_name;Value  character\_set\_client;utf8mb4//客户端向服务器发送sql的字符集  character\_set\_connection;utf8mb4// 客户端过来的字符串处理该语句  character\_set\_database;utf8//数据库默认字符集  character\_set\_filesystem;binary//文件系统字符集，使用load data语句  character\_set\_results;utf8mb4//服务器向客户端返回结果字符集  character\_set\_server;utf8//服务器默认字符集  character\_set\_system;utf8//存储标识符字符集  **show** **variables** **like** 'collation\\_%';  Variable\_name;Value  collation\_connection;utf8mb4\_general\_ci  collation\_database;utf8\_general\_ci//数据库默认排序规则  collation\_server;utf8\_general\_ci//服务器默认排序  注意：\_ci表示排序规则不区分大小写,\_cs表示排序规则区分大小写,\_bin表示二进制排序规则。影响的操作符：比较操作符，排序（order by min max）,分组（group by distinct）  set names utf8 等效于：  set character\_set\_client=utf8mb4  set character\_set\_connection=utf8mb4  set character\_set\_results=utf8mb4 |

### Unicode支持

Unicode的目标是提供一种统一的字符编码系统，让所有语言的字符集都可以以一种统一的方式进行表示。

Utf8与ucs2这两种类型的Unicode字符集，都只包括BMP里定义的字符，最大只有65536个字符。他们都不包括BMP之外的补充字符。

Ucs2字符集与Unicode的ucs-2编码方案相对应。它使用2个字节来表示一个字符，并且最高有效字节优先

Utf8字符集采用了一种长度可变的格式。使用1到3个字节来表示一个字符。它与UTF-8编码格式对应。

从MySQL5.5.3开始其他的Unicode字符集都包括了BMP之外的哪些补充字符：utf16和utf32类似于ucs2,只是它们增加了补充字符的支持。Utf8mb4字符集包含了所有的utf8字符（其中每个字符占用1到3个字节。另外，也包含了补充字符，其中每个字条占用4个字节）

# mysql的SQL语句

## 了解SQL

|  |
| --- |
| **数据库**：保存有组织的数据的容器。  **DBMS数据库管理系统**：数据库是通过DBMS创建和操纵的容器。常见的  DBMS有sqlserver，mysql，oracle。  **表**：一种结构化得文件，可以用来存储某种特定类型的数据。对应实体类。  表名具有唯一性。  **模式**：关于数据库和表的布局及特性的信息。  **列和数据结构**：表中的一个字段。对应实体中的属性。表中的每个列都有相应  的数据类型。数据类型用于限制列存储的数据。  **行**：表中的一行记录。  **主键**：唯一标识表中行的列或者列组。特性：唯一性，非null。  **SQL**：结构化查询语言，是一种专门用来数据库通信的语言。是一种通用的操作关系  型数据库语言。但是不同的数据库的SQL语法存在方言。 |

## 数据定义语言(DDL)

数据定义语言(DDL)，例如：CREATE、DROP、ALTER等语句。

### 数据库的选择创建和更改

|  |
| --- |
| **#create 创建数据库**  create if not exists database firstDB;  **#show 查看所有数据库**  show databases;  show create database firstDB;  **#alter 修改数据库,alter 命令修改数据库编码**  ALTER DATABASE firstDB CHARACTER SET UTF8;  **#查看当前使用的数据库**  select database();  **#drop 删除数据库**  drop database if exists firstDB; |

### 表的创建删除，索引和更改

|  |
| --- |
| **#表基本创建语句**  CREATE [TEMPORARY] TABLE [IF NOT EXISTS] `question\_subquestion` (  `id` INT(11) NOT NULL AUTO\_INCREMENT,  `question\_id` BIGINT(20) NOT NULL,  `sub\_question\_id` BIGINT(20) NOT NULL, KEY(question\_id), PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=1;  注意：临时表被在会话结束后自动删除，最好用完显示删除临时表，释放资源；临时表名和永久表名重复时候，永久表会被隐藏起来，直到临时表删除才会出现。  **#根据其他表或者查询结果创建表**  CREATE TABLE new\_tb LIKE tb;  INSERT INTO new\_tb SELECT \* FROM tb;  #CREATE TABLE SELECT …用于保存数据，  **#删除表**  DROP TABLE IF EXISTS tb\_name;  TRUNCATE tb\_name;  **#查看表状态**show table status [from db\_name] {like 'pattern' | where}     * Name:表名称 * Engine:表的存储引擎 * Version:版本 * Row\_format:行格式。对于MyISAM引擎，这可能是Dynamic，Fixed或Compressed。动态行的行长度可变，例如Varchar或Blob类型字段。固定行是指行长度不变，例如Char和Integer类型字段。 * Rows:表中的行数。对于非事务性表，这个值是精确的，对于事务性引擎，这个值通常是估算的。 * Avg\_row\_length:平均每行包括的字节数 * Data\_length:整个表的数据量(单位：字节) * Max\_data\_length:表可以容纳的最大数据量 * Index\_length:索引占用磁盘的空间大小 * Data\_free:对于MyISAM引擎，标识已分配，但现在未使用的空间，并且包含了已被删除行的空间。 * Auto\_increment:下一个Auto\_increment的值 * Create\_time:表的创建时间 * Update\_time:表的最近更新时间 * Check\_time:使用 check table 或myisamchk工具检查表的最近时间 * Collation:表的默认字符集和字符排序规则 * Checksum:如果启用，则对整个表的内容计算时的校验和 * Create\_options:指表创建时的其他所有选项 * Comment:包含了其他额外信息，对于MyISAM引擎，包含了注释徐标新，如果表使用的是innodb引擎 ，将现实表的剩余空间。 * 如果是一个视图，注释里面包含了VIEW字样。 |
| **#添加索引**  ALTER TABLE tb\_name ADD INDEX [idx\_name] (列…)  ALTER TABLE tb\_name ADD UNIQUE [idx\_name] (列…)  ALTER TABLE tb\_name ADD PRIMARY KEY (列…)  ALTER TABLE tb\_name ADD FULLTEXT [idx\_name] (列…)  CREATE INDEX idx\_name ON tb\_name(列…)  CREATE UNIQUE INDEX idx\_name ON tb\_name(列…)  CREATE FULLTEXT INDEX idx\_name ON tb\_name(列…)    CREATE TABLE tb\_name(  ......  INDEX [idx\_name](列…)  UNIQUE [idx\_name](列…)  PRIMARY KEY(列…)  FULLTEXT index\_name(列…)  )  **#删除索引**  DROP INDEX index\_name ON tb\_name;  ALTER TABLE tb\_name DROP INDEX idx\_name;  ALTER TABLE tb\_name DROP PRIMARY KEY ;  #查看索引    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Table: area （表名）  Non\_unique: 0 （索引是否允许包含重复值）  Key\_name: PRIMARY （索引名）  Seq\_in\_index: 1 （列在索引的序号）  Column\_name: id （索引列名）  Collation: A （列在索引的排序，A升序，D降序，null不排序）  Cardinality: 3339 （索引唯一值数量）  Sub\_part: NULL （前缀索引，则输出前缀长度）  Packed: NULL （键的压缩方式，null表示未压缩）  Null: （列是否包含null值，是则为yes,否则为空白）  Index\_type: BTREE （索引实现方式）  Comment: （索引内部注释）  Index\_comment: （定义索引的注释）  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| **#表的修改**  #更改列数据类型，CHANGE可以修改列名称  **ALTER** **TABLE** tb\_name CHANGE **COLUMN** col\_name col\_name **列属性**;  **ALTER** **TABLE** tb\_name MODIFY col\_name **列属性**;  #添加列，可以同时添加多个属性  **ALTER** **TABLE** tb\_name  **ADD** **COLUMN** col\_name 列属性{**INT**(5) **NULL** AFTER `seq\_num`}  #删除列  ALTER TABLE tb\_name DROP COLUMN 列名;  #更改存储引擎  create table new\_tb like old\_tb;  alter table new\_tb engine=InnoDB;  insert into new\_tb select \* from old\_tb;  #更改auto\_increament  ALTER TABLE tb\_name auto\_increament=…  #重命名表名  RENAME TABLE tb\_name TO new\_tb\_name;#可以多个表进行重命名  ALTER TABLE tb\_name RENAME TO new\_tb\_name;  rename table t1 to tmp,t2 to t1 ,tmp to t2; |

### 外键和引用完整性

MySQL的InnoDB引擎支持外键，它可以保证外键关系里不会出现不匹配的情况（外键中可以为null），保证了数据库的参照完成性：insert子表的时候，外键必须在父表中。更新父表的时候，能够级联更新子表中的数据。删除父表中的数据时候，能够练级删除子表数据。外键语法如下：

|  |
| --- |
| [CONSTRAINT constraint\_name]  FOREIGN KEY [fk\_name] (index\_col)  REFERENCES tb\_parent (index\_col)  [ON DELETE action]  [ON UPDATE action]  [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]  说明：  CONSTRAINT指定外键约束名字  FOREIGN KEY：指定子表索引列。  REFERENCES：指定父表关联索引列  ON DELETE：指定删除父表时候，子表的行为。   * ON DELETE ON ACTION 和ON DELETE RESTRICT：默认行为，表示禁止方式。当子表中存在父表中的关联索引列，,则不允许进行删除父表关联索引列的行为 * ON DELETE CASCADE：级联方式，父表中删除和更新关联索引列的效果会蔓延到子表。 * ON DELETE SET NULL：级联方式，父表中删除和更新关联索引列。子表会将父表中不存在的关联索引字段设置为null.   ON UPDATE: 指定更新父表时候，子表的行为。   * ON UPDATE ON ACTION 和ON UPDATE RESTRICT：默认行为，表示禁止方式。当子表中存在父表中关联的对象，则不允许进行更新父表关联索引列的行为 * ON UPDATE CASCADE：级联方式，父表中更新关联索引列的效果会蔓延到子表。 * ON UPDATE SET NULL：级联方式，父表中更新关联索引列。子表会将父表中不存在的关联索引字段设置为null.   添加外键约束：ALTER TABLE `child\_table` ADD CONSTRAINT `FK\_child\_parent ` FOREIGN KEY (`t\_id`) REFERENCES `parent\_table` (`id`) ON UPDATE CASCADE ON DELETE CASCADE;  **注意**：父子表关联列必须建立索引（子表没有建立索引的时候，MySQL会自动给该字段建立索引），列必须兼容（相同大小，符号数） |

注意：导入数据的时候，我们可以通过set foreign\_key\_checks=0来忽略外键检测，然后load data … ,最后set foreign\_key\_checks=1

### 使用FULLTEXT搜索

全文搜索类型有三种：

* 自然语言搜索(默认)：MySQL 会把搜索出来字符串解析成一系列的单词，然后搜索出包含这些单词的哪些行。
* 布尔模式搜索：在搜索字符串里那些单词可以包含修饰符，用以表明某些特定要求。那些单词在里面，哪些不在里面。
* 查询扩展搜索

全文搜索这种索引具有的特点：

* 全文索引基于FULLTEXT
* 全文索引将会忽略掉哪些常见次（出现一半以上的单词）
* 有些内建的常用单词，如the,after等。
* 太短的单词也会被忽略
* 全文搜索对单词的定义：字母，数字，撇号，下划线构成的字符序列。
* 可以为单个列或者多个列创建FULLTEXT索引

|  |
| --- |
| **#自然语言FULLTEXT搜索**  select \* from tb\_name where match(col\_name) against (' 搜索字符串');  注意：输出是按照相关程度递减排序的，相关度是非负浮点数，其中0表示毫不相关。可以通过select match(col\_name) against (' 搜索字符串') from tb\_name来查看。 |
| **#布尔模式全文搜索**，可以获取对多单词搜索的更多控制。需要在AGAINST()函数后面加上 IN BOOLEAN MODE。布尔模式全文搜索有以下特点：   * 50%不起作用 * 查询结果不再按相关度排列输出 * 搜索要求所有的单词必须按某种特定的顺序出现 * 可以对为包含在fulltext索引的列进行搜索，只是更慢一点   select \* from tb\_name where match(col\_name) against ( 限定符 ' 搜索字符串' IN BOOLEAN MODE) ;  +,-：表示该单词必须出现或者不出现  <,>：表示减少或者增加该单词对相关度计算值的贡献  ~：是给定的单词在相关度计计算里的贡献值变换正负号。  \*：通配符，任意字符。  "短语"：匹配短语，要求顺序一致。 |
| **#全文搜索引擎变量**，配置单词具有的最小长度和最大长度  ft\_min\_word\_len=3  ft\_min\_word\_len=84 |
| # MySQL 5.7开始，MySQL内置了ngram全文检索插件，用来支持中文分词，并且对MyISAM和InnoDB引擎有效。在使用中文检索分词插件ngram之前，先得在MySQL配置文件里面设置他的分词大小，比如，  [mysqld]  ngram\_token\_size=2  **CREATE** **TABLE** articles (  id INTUNSIGNED **AUTO\_INCREMENT** **NOT** **NULL** **PRIMARY** **KEY**,  titleVARCHAR(200),  body **TEXT**, ***FULLTEXT (title,body) WITH PARSER ngram***  ) **ENGINE**=InnoDBCHARACTER **SET** utf8mb4;  **#自然语言检索**  **SELECT** **COUNT**(\*) **FROM** articles **WHERE** **MATCH** (title,body) **AGAINST** ('数据库' **IN** **NATURAL** **LANGUAGE** **MODE**);  **SELECT** id, **MATCH** (title,body) **AGAINST** ('数据库' **IN** **NATURAL** **LANGUAGE** **MODE**) **AS** score **FROM** articles; *-- 匹配率*  #布尔模式  **SELECT** \* **FROM** articles **WHERE** **MATCH** (title,body) **AGAINST** ('+数据库 +管理' **IN** BOOLEAN **MODE**); |

## 数据操作语言(DML)

数据操作语言(DML)，例如：INSERT（插入）、UPDATE（修改）、DELETE（删除）语句。

### 插入数据

|  |
| --- |
| INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]  [INTO] tbl\_name [(col\_name,...)]  VALUES ({expr | DEFAULT},...),(...),...  [ ON DUPLICATE KEY UPDATE col\_name=expr, ... ]  或：  INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]  [INTO] tbl\_name  SET col\_name={expr | DEFAULT}, ...  [ ON DUPLICATE KEY UPDATE col\_name=expr, ... ]  或：  INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE]  [INTO] tbl\_name [(col\_name,...)]  SELECT ...  [ ON DUPLICATE KEY UPDATE col\_name=expr, ... ]  **注意**：replace语句作用与语法和insert相同。不同之处在于：如果将插入的行导致唯一性索引出现重复的键值，MySQL会先删除旧行然后再插入新行。 |
| insert into tb\_name values(value1,value2,…);#一次性指定全部列  insert into tb\_name(col1,col2,…) values(value1,value2,…);#指定列赋值  insert into tb\_name set col1=val1,col2=val2,…;#指定列赋值  insert into tb\_name [col1,col2,…] select [[col1,col2,…] from other\_tb;  例如：  **INSERT** **INTO** user (name,phone) **VALUES** ('hello ', '18868812342 ');  **REPLACE** **INTO** `demo` **VALUES** (1,'Nfjaklsdjjkl', 'goog', 3, '2016-10-12 16:34:04', '2016-12-13 10:32:10');  **INSERT** **INTO** TABLE (a,c) **VALUES** (1,3) **ON DUPLICATE KEY** **UPDATE** c=c+1; |

### 更新和删除数据

|  |
| --- |
| #更新操作  UPDATE [LOW\_PRIORITY] [IGNORE] tbl\_name或者tb\_refs  SET col\_name1=expr1 [, col\_name2=expr2 ...]  [WHERE where\_definition]  [ORDER BY ...]  [LIMIT row\_count]  说明：  **LOW\_PRIORITY**：延迟到没有客户端读取该表位置，适合MyISAM或MEMORY  **IGNORE**：忽略严格模式的错误。  **ORDER** BY：执行顺序  **LIMIT**：更新数量 |
| 举例：  UPDATE student\_kp\_status a  JOIN (  SELECT id, CASE WHEN tg\_score<=0 THEN NULL WHEN tg\_score>0 AND tg\_score<65 THEN 'U' WHEN tg\_score>=65 AND tg\_score<70 THEN 'A' WHEN tg\_score>=70 AND tg\_score<80 THEN 'B' WHEN tg\_score>=80 AND tg\_score<90 THEN 'C' WHEN tg\_score>=90 AND tg\_score<95 THEN 'D' WHEN tg\_score>=95 THEN 'E' END AS tg\_rank  FROM student\_kp\_status) b ON a.id=b.id SET a.tg\_rank=b.tg\_rank; |
| **#删除操作**  #*单表删除*  DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM tbl\_name  [WHERE where\_definition]  [ORDER BY ...]  [LIMIT row\_count]  #*多表删除*  DELETE [LOW\_PRIORITY] [QUICK] [IGNORE]  tbl\_name[.\*] [, tbl\_name[.\*] ...]  FROM table\_references  [WHERE where\_definition]  说明：  LOW\_PRIORITY：DELETE的执行被延迟，直到没有其它客户端读取本表时再执行。  IGNORE：会使 MySQL忽略所有的错误。（在分析阶段遇到的错误会以常规方式处理。）由于使用本选项而被忽略的错误会作为警告返回。 |
| 举例：#删除重复数据，并保留较小的id  DELETE s1  FROM school s1  JOIN (  SELECT NAME  ,area\_id  ,LEVEL  ,min(id) AS id  FROM school  GROUP BY NAME  ,area\_id  ,LEVEL  HAVING COUNT(\*) > 1  ) s2 ON (  s1.NAME = s2.NAME  AND s1.area\_id = s2.area\_id  AND s1.LEVEL = s2.LEVEL  )  WHERE s1.id > s2.id; |

## 数据查询语言(DQL)

数据查询语言(DQL)，例如：SELECT语句。

### 单表查询

|  |
| --- |
| select 属性列表  from 表名和视图列表  [where 条件表达式]  [group by 属性名[having 条件表达式]]  [order by 属性名[asc|desc]]  [limit <offset>,row count] |
| #1 SELECT子句（选择列）  选择列：可以检索多个列，不同列使用逗号隔开（,），使用\*检索所有的列。  使用完全限定名：products.prod\_name，似的表命名唯一  别名的使用：product.prod\_name AS name//可以简化SQL的书写，消去不符合规则字符，其中AS可以省略  DISTINCT关键字：检索指定列唯一显示（应用于给定所有列）  LIMIT关键字：LIMIT 开始行（从0开始）,行数不够时候，返回只能返回的行数；只有一个参数的时候，参数为显示的行数。  注意：SELECT选择的列中可以使用计算字段（支持+，-，\*，/），拼接字段(可以使用mysql函数)。在列中直接处理字段比程序要快的多 |
| #2 WHERE子句（过滤行）  条件操作符：= <> != < <= > >=  BETWEEN…AND…：可以应用数值和日期类型。包含开始和结束值  空值检查：IS NULL/IS NOT NULL  逻辑操作符：AND(优先级高于OR,使用OR的括起来) OR(或) ,NOT(MYSQL支持NOT对IN/BETWEEN/EXISTS子句取反)  集合运算:其实，IN效果和OR是一样的。IN的操作符一般比OR更快（in/not in）  LIKE通配符过滤：%(任意字符出现任意次),\_(匹配单个字符)，不匹配NULL  'S%E'：S开头，E结尾,"%SE%"：包含SE  REGEXP正则表达式匹配：prod\_name REGEXP '[123] ton'  .：表示任意字符  |：为正则表达式or匹配，匹配其中的一个  […]:匹配几个字符中的一个字符，其中可以使用^表示非的意思  -：范围匹配，[1-9]  匹配特殊字符：\\f(换页)\\n(换行)\\r(回车)\\t(制表)\\v(纵向制表)\\\(\)  以\\开始，如\\(,\\),\\.等  匹配字符类：  [:alnum:]：任意字母和数字 同[a-zA-Z0-9]  [:alpha:]：任意字符[a-zA-Z]  [:lower:]：任意小写字母[a-z]  [:upper:]：任意大写字母[A-Z]  [:digit:]：任意数字[0-9]  [:xdigit:]：任意十六进制数字[a-fA-F0-9]  [:blank:]：空格和制表 [ \\t]  [:space:]：包括空格在内的任意空白字符  [:cntrl:] ：ASCII控制字符0-31和127  [:graph:]：除空格外的所有打印字符  [:print:]：任意可打印字符  [:punct:]：不在[:alnum:]和[:cntrl:]的所有字符  匹配多个实例：  \*：(0个或者多个)  +：(1个或者多个)  ?：(0个或者一个)  {n}：指定数目匹配  {n, }：不少于指定数目的匹配  {n,m}：匹配数目的范围（m不超过255）  定位符：  ^：文本开始  $：文本结束  [[:<:]]：单词开始  [[:>:]]：单词结尾  注意：正则表达式REGEXP区分大小写可以使用REGEXP BINARY |
| #3 FROM子句  From后面接的是表，具体看连接查询。 |
| #4 ORDER BY子句  ORDER BY 列1 ASC(默认)/DESC, 列2 ASC(默认)/DESC,…  说明：先按列1排序，列1相同的时候按列2排序。使用ORDER BY和LIMIT  可以实现排名。 |
| #5GROUP BY子句  集函数: 运行在行组上，计算和返回单行单列数据。聚集函数会忽略列值为NULL的行。count(\*)不会忽略NULL的行。聚集不同的值：聚集函数(DISTINCT 列名)。不能是计算或者表达式或\*。可以在函数中写条件   * AVG(某列)：返回某列的平均值 * MAX(某列)：返回某列的最大值 * MIN(某列)：返回某列的最小值 * SUM(某列)：返回某列值之和 * COUNT(某列)：返回某列的行数。   GROUP BY 子句可以包含任意数目的列，数据只能在分组上进行汇总。  GROUP BY 子句中列出的都必须是有效的表达式或者列，不能是汇总函数  GROUP BY子句使用的表达式必须是SELECT中有的，不能使用别名。  SELECT子句中只能包含GROUP BY中给出的列  分组中具有NULL，则NULL将作为一个分组返回  使用ROLLUP对分组进行汇总，注意，当使用ROLLUP时，不能同时使用ORDER BY子句进行结果排序，即ROLLUP和ORDER BY是互相排斥的。  SELECT vend\_id,COUNT(\*) AS num\_prods  FROM products  GROUP BY vend\_id WITH ROLLUP  HAVING过滤分组：HAVING支持所有的WHERE操作符。WHERE是行过滤。  注意：GROUP BY 分组数据确实是以分组的顺序输出的。一般使用GROUPBY子句的时候，应该给出ORDER BY子句。这是保证数据正确排序的唯一的方法。 |
| SQL语句的执行顺序：from（表）--->where(筛选行)--->group by(分组)--->select(选择列)--->order by(排序) |

### 多表查询

|  |
| --- |
| #1连接多表查询  #笛卡尔乘积  FROM t1 ,t2 或者FROM t1 [INNER|CROSS]JOIN t2（行数:t1Xt2,列数:t1+t2）  #内连接 ,需要在笛卡尔乘积上面加上关联关系。ON | WHERE 条件 | USING(col\_name)  FROM t1,t2 WHERE t1.id=t2.id  FROM t1 JOIN t2 ON t1.id=t2.id  FROM t1 JOIN t2 USING(id)  #左外连接和右外连接：左边【右边】没有匹配的行也会检索出来，而连接的右表【左表】是一个假行，其中每一列都包含NULL值。  举例：查询包含在t1中而不在t2中的记录  SELECT \* FROM t1 JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL; |
| #2子查询多表查询  #带关系比较运算符的子查询（<>,=,>,>=,<,<=）,where中需要使用聚合函数  SELECT \* FROM president WHERE birth=(SELECT MIN(binth) FROM president)  #IN和NOT IN子查询，子查询返回多行时候使用。IN和NOT IN 实际就是=ANY 和<>ALL的同义词。  #ALL,ANY和SOME子查询，SOME和ANY是同义词。<=ALL相当于取最小，>=ALL表示取最大，<>ALL表示不在其中；=ANY表示在其中，>=ANY表示取最大，<=ALL相当于取最小。  #相关子查询  #FROM子查询：用在FROM子句的子查询，使用的时候必须加上别名 |
| #3 union多表查询  SELECT col\_1,col\_2,… FROM t1 UNION SELECT col\_1,col\_2,… FROM t2;  #列名和数据类型：各个SELECT中列名和数据类型可以不一致，MySQL会对其进行相应的强制转换，但是列的个数必须一致。  #重复行处理：UNION DISTINCT是UNION的同义词。尽量不要使用，会影响性能。  #ORDER BY 和LIMIT处理：必须把每个SELECT用括号括起来，并且使用第一个SELECT语句的列名。  (SELECT col\_1,col\_2,… FROM t1) UNION (SELECT col\_1,col\_2,… FROM t2) ORDER BY col\_1 LIMIT 3; |

## 事务控制语句（TCL）

事务控制语句（TCL），例如：COMMIT、ROLLBACK等语句。

事务的ACID特性：

* Atomic(原子性)：构成事务的所有语句是一个独立的逻辑单元。
* Consistent(一致性)：数据库在事务的执行前后都必须是一致的。
* Isolated(隔离性)：事务之间不应该互相影响，并发执行和单独执行效果一样。通过锁来实现。
* Durable(持久性)：当事务完成时，其影响永久被记录到数据库。

### 利用事务保证语句安全执行

默认情况下，MySQL的运行模式是自动提交，即每条语句所做的修改会立刻提交到数据库，并永久保留起来。事务相关的语句有：

|  |
| --- |
| #常用事务语句  start transaction | begin(begin会和复合语句begin…end冲突)  …  savepoint identifier  …  rollback to [savepoint] identifier  rollback |commit  #设置自动提交  set auto\_commit=0  …  set auto\_commit=1 |
| #隐式提交的SQL语句    注意：truncate也是默认提交的。 |

### InnoDB存储引擎事务隔离级别

|  |
| --- |
| **#事务并发产生的问题**  脏读（Dirty Read）：读取未提交的数据。  不可重复读（Nonrepeatable Read）：一个事务在两次执行同一个select产生不一样的结果。（另一个事务在两次select期间执行更新操作）。  幻读（Phantom Read）：一个事务两次执行同一个select出现新增行。（另一个事务在两次select期间插入了数据。） |
| **#** InnoDB存储引擎事务隔离级别  **READ UNCOMMITTED(未提交)** 即使没有提交，对其它事务也可见。未提交的数据会引起脏读(Dirty Read)。 测试流程： 1、A设置read-uncommitted, start transaction 2、B执行start transaction，修改一条记录， 3、A查询记录，得到了以为正确的记录 4、B回滚。 问题：A读到了B没有提交的记录，也就是脏读。   **READ COMMITTED(提交读)** 一个从开始直到提交之前所做的任何修改对其它事务都是不可见的。两次同样的查询可能会得到不一样的结果，称为不可重复读(nonrepeatable read) 测试流程： 1、A设置read-committed, start transaction 2、B执行start transaction，修改一条记录，查询记录，记录已经修改成功 3、A查询记录，结果还是老的记录 4、B提交事务 5、A再次查询记录，结果是新的记录。 问题：两次查询结果不一致，也就是不可重复读问题。  **REPEATABLE READ(可重复读)-MySQL默认的事务隔离级别** 保证了在同一事务中多次读取结果是一致的。但会引起另外一个幻读问题，当某个事务在读取某个范围记录时，另外一个事务在该范围插入和新记录，当之前事务再次读取该范围记录时会产生幻行。 测试流程： 1、A设置repeatable-read, start transaction，查询记录，结果是老的记录 2、B执行start transaction，修改一条记录，查询记录，记录已经修改成功 3、A查询记录，结果还是老的记录 4、B提交事务 5、A再次查询记录，结果还是老的记录。 问题：可以重复读，A在事务过程中，即使B修改了数据，并且commit，A读取的还是老的数据。即可重复读。 注意：这里可能会存在一个新的问题，A在事务过程中，B增加一条记录，并提交，导致A的两次读取不一致，会多一条记录，也就是幻影读。InnoDB通过多版本并发控制(MVCC)解决了幻读问题。  **SERIALIZABLE(可串行化)** 强制事务串行执行，但可能导致大量超时和锁争问题。对于某个事务正在查看的行，只有等到该事物完成才能被其他事务修改。 测试流程： 1、A设置serializable, start transaction，查询记录，结果是老的记录 2、B执行start transaction，修改一条记录，B卡在这里，要等待A完成才行。 3、A查询记录，结果还是老的记录，A提交。 4、B的修改操作才进行下去。 注意：B在等待过程中，会出现lock超时  #MySQL事务隔离级别总结： |
| #设置InnoDB的隔离级别方法  SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE} |

## 获取数据库元数据

### 使用show语句获取元数据

|  |
| --- |
| #查看服务器状态，复制(主从服务器)状态，表状态，进程状态  show global status;  show master status;  show slave status;  show table status;  show full processlist;  #查看主从，数据库，系统变量，权限，字符集，表,列,索引，触发器，函数，存储过程，事件，插件,引擎，警告，错误等信息  show binary logs; #显示当前二进制日志文件的名称和大小  #显示二进制日志里的事件  show binlog events IN 'file\_name' [FROM position] [LIMIT [skip\_count,]show\_count] ;  #在主服务器上查看从服务器信息  show slave hosts;  #查看从服务器中继日志内容  show relaylog events;  show databases;  show variables like '…' ;  show grants for [account];  show CHARACTER SET ;  show collation;  show tables;  show open tables;  show columns from tb\_name from db\_name like '…' ;  show index from tb\_name from db\_name ;  show triggers;  show function status;  show procedure status;  show events from db\_name like '' ;  show engines;  show plugins;  show warnings;  show errors;  #查看创建语句  show create database/event/funcation/procedure/table/trigger/view name |

### 使用INFORMATION\_SCHEMA获取元数据

|  |
| --- |
| SCHEMATA,TABLES,VIEW,ROUTNES,TRIGGERS,EVENTS,PARAMETERS,PARTITIONS,COLUMNS：查看与数据库，表，视图，存储过程，触发器，事件，表分区，列信息  FILES；用于存储表空间数目的那些文件有关的信息  TABLE\_CONSTRAINS,KEY\_COLUMN\_USAGE：与具有约束条件的条有列有关  STATISTICS：与表索引特性有关的信息  REFERENTIAL\_CONSTRAINTS：与外键有关的信息  CHARACTER\_SETS,COLLATIONS,COLLATION\_CHARACTER\_SET\_APPLICABILITY：与支持的字符集，排序，以及他们映射关系表。  ENGINES,PLUGINS:存储引擎和服务器插件有关的信息  USER\_PRIVILEGES,SCHEMA\_PRIVILEGES, TABLE\_PRIVILEGES, COLUMN\_PRIVILEGES:权限分配有关的表  GLOBAL\_VARIABLES,SESSION\_VARIABLES,GLOBAL\_STATUS,SESSION\_STATUS:  全局和会话的系统变量值和状态变量值  PROCESSLIST:与服务器内线程有关的信息。 |