**MySQL数据库学习**

# MySQL介绍

MySQL是一种关系数据库，该类型数据库是建立在关系模型基础上的，借助于集合代数等数学概念和方法来处理数据库中的数据。现实世界中的各种实体以及实体之间的各种联系均用关系模型来表示。关系模型由关系数据结构、关系操作集合、关系完整性约束三部分组成。简单说，关系型数据库是由多张能互相联接的二维行列表格组成的数据库。

MySQL的发展历史：

1、1996年，Monty Widenius开发并发布了MySQL 1.0

2、1999～2000年，MySQL AB公司成立了（Monty Widenius也作为创始人之一），发布了MySQL 3.23

3、2000年，MySQL 公布了自己的源代码，并采用GPL（GNU General Public License）许可协议，正式进入开源世界

4、2003年3月，MySQL 4.0正式发布

5、2004年10月MySQL 4.1发布

6、2005年10月发布里程碑的一个版本 5.0

7、2008年1月16号 Sun公司收购MySQL

8、2009年4月20，Oracle收购Sun公司

9、2010年04月22 发布MySQL 5.5

* 关系型数据库是以关系模型为基础的数据库，一个关系就是一张二维表；
* 数据是存放在表中的，数据库是一组数据表的集合；
* 数据表由行和列组成；
* 一行数据表示一条数据记录；
* 一列数据表示具有相同域的值，具有相同的属性类型、属性名，列又叫字段；

# MySQL安装

1. 下载链接：<https://dev.mysql.com/downloads/mysql/> ，下载对应的版本，以mysql-5.7.19-winx64版本为例
2. 解压到指定的目录：F:\mysql-5.7.19-winx64
3. 将my.ini拷贝到F:\mysql-5.7.19-winx64 目录

记住：修改my.ini中对应 basedir 和 datadir 为你自己的目录

1. 将MySQL添加为服务
2. 以管理员身份运行 cmd 窗口
3. 在cmd窗口中切换到F:\mysql-5.7.19-winx64\bin目录
4. 执行mysql的初始化操作： mysqld --initialize

初始化成功的话，会在根目录创建一个 data 文件夹

1. 运行 mysqld --install MySQL，会显示 Service successfully installed. 服务名不区分大小写
2. 在cmd窗口中输入 net start MySQL 启动服务

下面是备用命令：

1. 在windows的 运行 中输入 services.msc，查看服务情况
2. 在cmd窗口中输入 net stop MySQL 停止服务
3. 在cmd 窗口中输入 mysqld -remove mysql ，删除服务

如下图：

官方手册地址：<https://dev.mysql.com/doc/refman/5.7/en/>

MySQL在cmd 中的常用命令（命令需要以 ; 结束）：

1. mysql -u root -p 连接数据库，接着出现 Enter password： 输入密码后回车（如密码为空 直接回车），需要在MySQL的bin目录下
2. mysql5.7有初始的默认密码，在data目录下的：类似DESKTOP-IT44UNF.err文件（\*.err）中，搜索root@localhost: ，会有类似下面的：

root@localhost：f7qaJUw4d-=% 这样的一行，后面的 f7qaJUw4d-=% 就是密码

进入mysql> 之后， 输入

set password = password('qwe123');

就可以修改密码了，其中 qwe123 就是你设置的新密码

1. show databases; 显示所有数据库
2. use db\_name; 使用指定数据库
3. show tables; 显示当前数据库下的所有表
4. \c 表示取消当前已经输入了的命令，得到一个全新的命令行
5. \h 显示所有命令
6. \q 退出
7. exit 退出
8. -> 表示命令没有结束，输入 ; 按回车 执行命令

# MySQL Workbench可视化工具

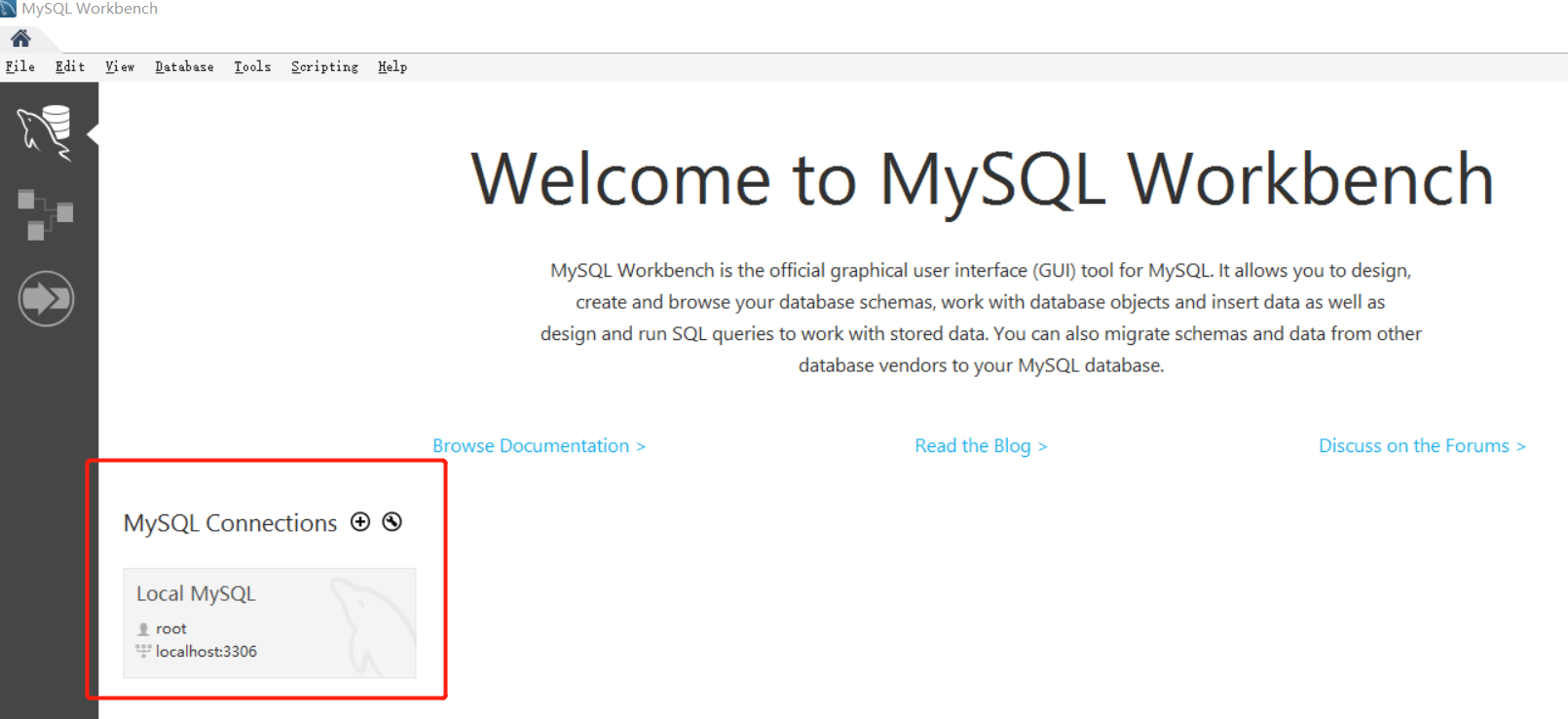
常用的MySQL可视化工具有：MySQL Workbench、Navicat、MySQL-Front等，推荐使用官方的MySQL Workbench

下载地址：<https://dev.mysql.com/downloads/workbench/> ，下载对应的版本，安装

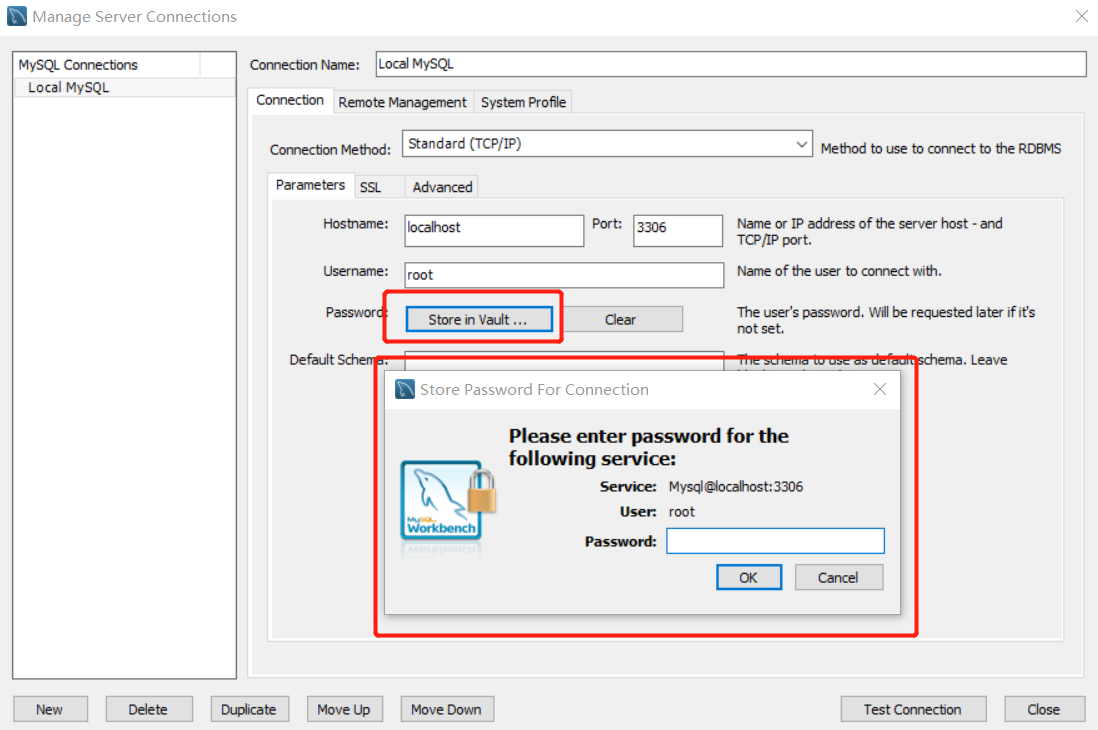
在线手册地址：<https://dev.mysql.com/doc/workbench/en/wb-launching-windows.html>

## 连接数据库

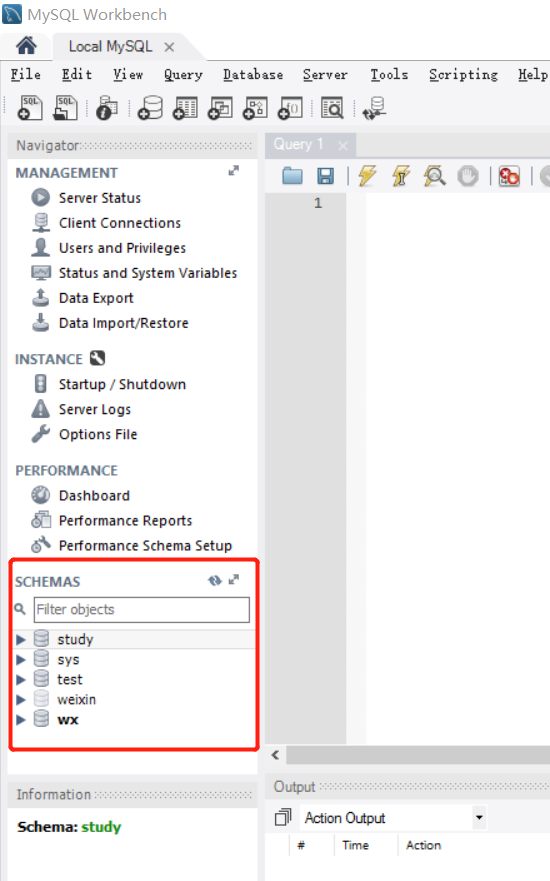
运行软件后，创建或修改一个连接，如下图：



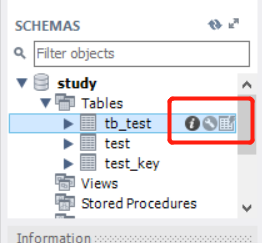
点击上图红色框中的 + 号或者 扳手符号，出现下图，配置ip，port，用户和密码



## 操作schemas（databases）



## 操作table



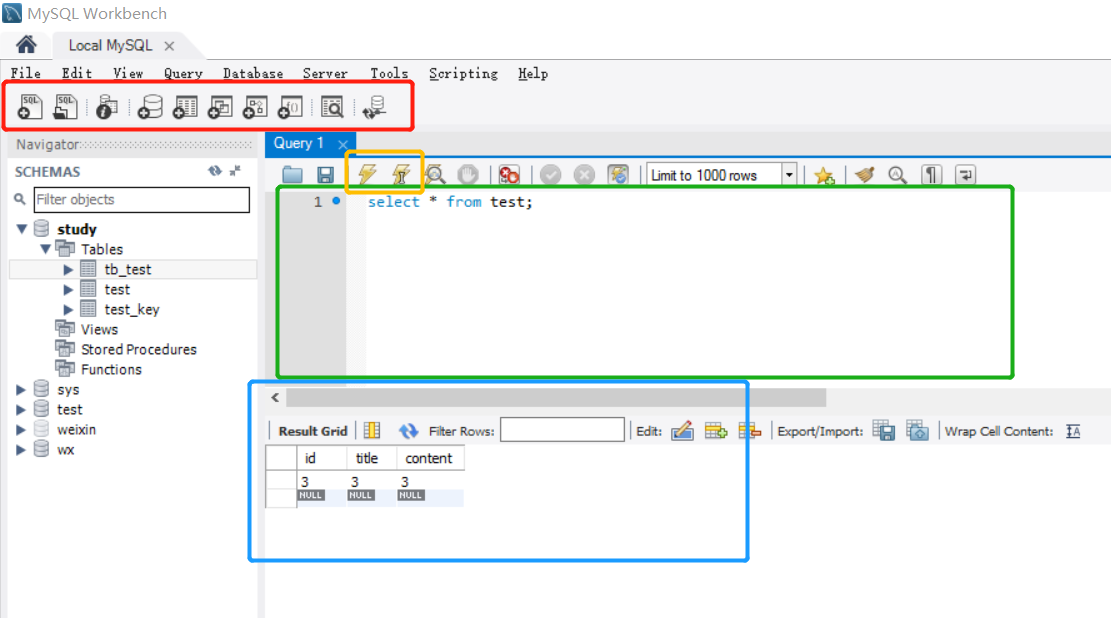
## 执行SQL语句

红框中是快捷菜单按钮

绿框中是编写 SQL 命令的文本框

黄框中是执行SQL 命令的按钮

蓝框中是执行 SQL 命令的结果



# 数据类型

## 数值类型

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **字节** | **最小值** | **最大值** |
|  |  | **(带符号的/无符号的)** | **(带符号的/无符号的)** |
| TINYINT | 1 | -128 | 127 |
|  |  | 0 | 255 |
| SMALLINT | 2 | -32768 | 32767 |
|  |  | 0 | 65535 |
| MEDIUMINT | 3 | -8388608 | 8388607 |
|  |  | 0 | 16777215 |
| INT | 4 | -2147483648 | 2147483647 |
|  |  | 0 | 4294967295 |
| BIGINT | 8 | -9223372036854775808 | 9223372036854775807 |
|  |  | 0 | 18446744073709551615 |
| FLOAT [(M, D)] | M<24，4字节，53>=M>=25，8字节 | -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38 | |
| DOUBLE [(M, D) | 8 | -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308 | |
| DECIMAL [(M [, D])] | M 个字节，如果M<D，则D+2个字节 | M最大65，D最大30 | |

M是整体长度，D是小数点后的长度，M包含D

## 日期时间

|  |  |
| --- | --- |
| **类型** | **“零”值** |
| DATETIME | '0000-00-00 00:00:00'  2018-02-26 15:10:38 |
| DATE | '0000-00-00'  2018-02-26 |
| TIMESTAMP | 00000000000000，时间戳 |
| TIME | '00:00:00'  15:10:38 |
| YEAR | '0000'  2018 |

## 字符串类型

|  |  |
| --- | --- |
| **类型** | **备注** |
| Char(n) | 定长字符串，n 是 0 到 255 |
| Varchar(n) | 可变长字符串，n 是 0 到 65535 |
| Binary(n) | 定长二进制，n 是 0 到 255 |
| Varbinary(n) | 可变长二进制，n 是 0 到 65535 |
| blob | 大对象，二进制，存图片，视频，音频，最大65535个字节 |
| text | 大对象，字符，文件，65535个字符 |
| enum | 枚举，enum(value1,value2,……valueN) 最多65535个选项，单选 |
| set | 集合，set(value1,value2,……valueN) 最多64个选项，多选 |

PS： char(5) 存入数据："ab"，那么在ab的右边以3个空格填充为："ab "，占5字节

char(5)，编码是utf8，存入“你好啊"，占11个字节

varchar(225) 如果存入数据"ab"，那么只占2个字节，加1个长度字节，共3个字节

char的查询速度远高于varchar，但是浪费空间

# SQL命令

## 数据库操作

### 创建数据库

语法：

|  |
| --- |
| /\*注释  { } 中的参数参数必写  | 是可选择，这里就是DATABASE 和 SCHEMA 2选1  [ ] 中的参数可写可不写  \*/  CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name  [create\_specification] ...  create\_specification:  [DEFAULT] CHARACTER SET [=] charset\_name  | [DEFAULT] COLLATE [=] collation\_name |

案例：

|  |
| --- |
| /\*注释  创建数据库  \*/  create database IF NOT EXISTS mydb  CHARACTER SET = utf8  COLLATE = utf8\_bin;  /\* 使用或者说切换到指定的数据库 \*/  use db\_name; |

### 删除数据库

语法：

|  |
| --- |
| DROP {DATABASE | SCHEMA} [IF EXISTS] db\_name |

案例：

|  |
| --- |
| /\*注释  删除数据库  \*/  drop database if EXISTS mydb; |

## 表操作

### 创建表

语法：

|  |
| --- |
| CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name  (create\_definition,...)  [table\_options]  [partition\_options]  CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name  [(create\_definition,...)]  [table\_options]  [partition\_options]  [IGNORE | REPLACE]  [AS] query\_expression  CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name  { LIKE old\_tbl\_name | (LIKE old\_tbl\_name) }  create\_definition:  col\_name column\_definition  | [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...)  [index\_option] ...  | {INDEX|KEY} [index\_name] [index\_type] (index\_col\_name,...)  [index\_option] ...  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]  [index\_name] [index\_type] (index\_col\_name,...)  [index\_option] ...  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index\_name] (index\_col\_name,...)  [index\_option] ...  | [CONSTRAINT [symbol]] FOREIGN KEY  [index\_name] (index\_col\_name,...) reference\_definition  | CHECK (expr)  column\_definition:  data\_type [NOT NULL | NULL] [DEFAULT default\_value]  [AUTO\_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]  [COMMENT 'string']  [COLUMN\_FORMAT {FIXED|DYNAMIC|DEFAULT}]  [STORAGE {DISK|MEMORY|DEFAULT}]  [reference\_definition]  | data\_type [GENERATED ALWAYS] AS (expression)  [VIRTUAL | STORED] [NOT NULL | NULL]  [UNIQUE [KEY]] [[PRIMARY] KEY]  [COMMENT 'string']  data\_type:  BIT[(length)]  | TINYINT[(length)] [UNSIGNED] [ZEROFILL]  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]  | INT[(length)] [UNSIGNED] [ZEROFILL]  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]  | DATE  | TIME[(fsp)]  | TIMESTAMP[(fsp)]  | DATETIME[(fsp)]  | YEAR  | CHAR[(length)]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | VARCHAR(length)  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | BINARY[(length)]  | VARBINARY(length)  | TINYBLOB  | BLOB[(length)]  | MEDIUMBLOB  | LONGBLOB  | TINYTEXT  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | TEXT[(length)]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | MEDIUMTEXT  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | LONGTEXT  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | ENUM(value1,value2,value3,...)  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | SET(value1,value2,value3,...)  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | JSON  | spatial\_type  index\_col\_name:  col\_name [(length)] [ASC | DESC]  index\_type:  USING {BTREE | HASH}  index\_option:  KEY\_BLOCK\_SIZE [=] value  | index\_type  | WITH PARSER parser\_name  | COMMENT 'string'  reference\_definition:  REFERENCES tbl\_name (index\_col\_name,...)  [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]  [ON DELETE reference\_option]  [ON UPDATE reference\_option]  reference\_option:  RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT  table\_options:  table\_option [[,] table\_option] ...  table\_option:  AUTO\_INCREMENT [=] value  | AVG\_ROW\_LENGTH [=] value  | [DEFAULT] CHARACTER SET [=] charset\_name  | CHECKSUM [=] {0 | 1}  | [DEFAULT] COLLATE [=] collation\_name  | COMMENT [=] 'string'  | COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}  | CONNECTION [=] 'connect\_string'  | {DATA|INDEX} DIRECTORY [=] 'absolute path to directory'  | DELAY\_KEY\_WRITE [=] {0 | 1}  | ENCRYPTION [=] {'Y' | 'N'}  | ENGINE [=] engine\_name  | INSERT\_METHOD [=] { NO | FIRST | LAST }  | KEY\_BLOCK\_SIZE [=] value  | MAX\_ROWS [=] value  | MIN\_ROWS [=] value  | PACK\_KEYS [=] {0 | 1 | DEFAULT}  | PASSWORD [=] 'string'  | ROW\_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}  | STATS\_AUTO\_RECALC [=] {DEFAULT|0|1}  | STATS\_PERSISTENT [=] {DEFAULT|0|1}  | STATS\_SAMPLE\_PAGES [=] value  | TABLESPACE tablespace\_name [STORAGE {DISK|MEMORY|DEFAULT}]  | UNION [=] (tbl\_name[,tbl\_name]...)  partition\_options:  PARTITION BY  { [LINEAR] HASH(expr)  | [LINEAR] KEY [ALGORITHM={1|2}] (column\_list)  | RANGE{(expr) | COLUMNS(column\_list)}  | LIST{(expr) | COLUMNS(column\_list)} }  [PARTITIONS num]  [SUBPARTITION BY  { [LINEAR] HASH(expr)  | [LINEAR] KEY [ALGORITHM={1|2}] (column\_list) }  [SUBPARTITIONS num]  ]  [(partition\_definition [, partition\_definition] ...)]  partition\_definition:  PARTITION partition\_name  [VALUES  {LESS THAN {(expr | value\_list) | MAXVALUE}  |  IN (value\_list)}]  [[STORAGE] ENGINE [=] engine\_name]  [COMMENT [=] 'string' ]  [DATA DIRECTORY [=] 'data\_dir']  [INDEX DIRECTORY [=] 'index\_dir']  [MAX\_ROWS [=] max\_number\_of\_rows]  [MIN\_ROWS [=] min\_number\_of\_rows]  [TABLESPACE [=] tablespace\_name]  [(subpartition\_definition [, subpartition\_definition] ...)]  subpartition\_definition:  SUBPARTITION logical\_name  [[STORAGE] ENGINE [=] engine\_name]  [COMMENT [=] 'string' ]  [DATA DIRECTORY [=] 'data\_dir']  [INDEX DIRECTORY [=] 'index\_dir']  [MAX\_ROWS [=] max\_number\_of\_rows]  [MIN\_ROWS [=] min\_number\_of\_rows]  [TABLESPACE [=] tablespace\_name]  query\_expression:  SELECT ... (Some valid select or union statement) |

#### 简单建表

|  |
| --- |
| /\*注释  创建表  \*/  CREATE TABLE t\_student (  stuNo int,  stuName varchar(20),  stuAge int,  stuSex char(2),  grade varchar(10),  subjectName varchar(10),  score double,  phone varchar(11)  ); |

#### 建表-主键

|  |
| --- |
| /\*注释  创建表，使用主键  \*/  CREATE TABLE t\_student (  stuNo int primary key,  stuName varchar(20),  stuAge int,  stuSex char(2),  grade varchar(10),  subjectName varchar(10),  score double,  phone varchar(11)  );  或：  CREATE TABLE t\_student (  stuNo int,  stuName varchar(20),  stuAge int,  stuSex char(2),  grade varchar(10),  subjectName varchar(10),  score double,  phone varchar(11),  primary key(stuNo)  ); |

#### 建表-主键自增

|  |
| --- |
| /\*注释  创建表，使用主键，自增  \*/  CREATE TABLE t\_student (  stuNo int primary key auto\_increment ,  stuName varchar(20),  stuAge int,  stuSex char(2),  grade varchar(10),  subjectName varchar(10),  score double,  phone varchar(11)  );  # 以下是一个复杂的建表，但是不建议大家这么建立  CREATE TABLE `t\_student` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `stuNo` int(11) DEFAULT NULL,  `stuName` varchar(20) COLLATE utf8mb4\_unicode\_ci NOT NULL,  `stuAge` tinyint(3) unsigned DEFAULT NULL,  `grade` varchar(10) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,  `subjectName` varchar(45) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,  `score` double DEFAULT NULL,  `phone` varchar(11) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,  `zero\_test` int(4) unsigned zerofill DEFAULT '0000',  PRIMARY KEY (`id`),  UNIQUE KEY `stuNo\_UNIQUE` (`stuNo`),  UNIQUE KEY ` grade\_UNIQUE` (`grade`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci |

### 查看表结构

语法：

|  |
| --- |
| DESCRIBE tbl\_name;  或  DESC tbl\_name; |

案例：

|  |
| --- |
| /\*  查看表结构  \*/  DESC t\_student; |

### 修改表

语法：

|  |
| --- |
| ALTER TABLE tbl\_name  [alter\_specification [, alter\_specification] ...]  [partition\_options]  alter\_specification:  table\_options  | ADD [COLUMN] col\_name column\_definition  [FIRST | AFTER col\_name]  | ADD [COLUMN] (col\_name column\_definition,...)  | ADD {INDEX|KEY} [index\_name]  [index\_type] (index\_col\_name,...) [index\_option] ...  | ADD [CONSTRAINT [symbol]] PRIMARY KEY  [index\_type] (index\_col\_name,...) [index\_option] ...  | ADD [CONSTRAINT [symbol]]  UNIQUE [INDEX|KEY] [index\_name]  [index\_type] (index\_col\_name,...) [index\_option] ...  | ADD FULLTEXT [INDEX|KEY] [index\_name]  (index\_col\_name,...) [index\_option] ...  | ADD SPATIAL [INDEX|KEY] [index\_name]  (index\_col\_name,...) [index\_option] ...  | ADD [CONSTRAINT [symbol]]  FOREIGN KEY [index\_name] (index\_col\_name,...)  reference\_definition  | ALGORITHM [=] {DEFAULT|INPLACE|COPY}  | ALTER [COLUMN] col\_name {SET DEFAULT literal | DROP DEFAULT}  | CHANGE [COLUMN] old\_col\_name new\_col\_name column\_definition  [FIRST|AFTER col\_name]  | [DEFAULT] CHARACTER SET [=] charset\_name [COLLATE [=] collation\_name]  | CONVERT TO CHARACTER SET charset\_name [COLLATE collation\_name]  | {DISABLE|ENABLE} KEYS  | {DISCARD|IMPORT} TABLESPACE  | DROP [COLUMN] col\_name  | DROP {INDEX|KEY} index\_name  | DROP PRIMARY KEY  | DROP FOREIGN KEY fk\_symbol  | FORCE  | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}  | MODIFY [COLUMN] col\_name column\_definition  [FIRST | AFTER col\_name]  | ORDER BY col\_name [, col\_name] ...  | RENAME {INDEX|KEY} old\_index\_name TO new\_index\_name  | RENAME [TO|AS] new\_tbl\_name  | {WITHOUT|WITH} VALIDATION  | ADD PARTITION (partition\_definition)  | DROP PARTITION partition\_names  | DISCARD PARTITION {partition\_names | ALL} TABLESPACE  | IMPORT PARTITION {partition\_names | ALL} TABLESPACE  | TRUNCATE PARTITION {partition\_names | ALL}  | COALESCE PARTITION number  | REORGANIZE PARTITION partition\_names INTO (partition\_definitions)  | EXCHANGE PARTITION partition\_name WITH TABLE tbl\_name [{WITH|WITHOUT} VALIDATION]  | ANALYZE PARTITION {partition\_names | ALL}  | CHECK PARTITION {partition\_names | ALL}  | OPTIMIZE PARTITION {partition\_names | ALL}  | REBUILD PARTITION {partition\_names | ALL}  | REPAIR PARTITION {partition\_names | ALL}  | REMOVE PARTITIONING  | UPGRADE PARTITIONING  index\_col\_name:  col\_name [(length)] [ASC | DESC]  index\_type:  USING {BTREE | HASH}  index\_option:  KEY\_BLOCK\_SIZE [=] value  | index\_type  | WITH PARSER parser\_name  | COMMENT 'string'  table\_options:  table\_option [[,] table\_option] ...  table\_option:  AUTO\_INCREMENT [=] value  | AVG\_ROW\_LENGTH [=] value  | [DEFAULT] CHARACTER SET [=] charset\_name  | CHECKSUM [=] {0 | 1}  | [DEFAULT] COLLATE [=] collation\_name  | COMMENT [=] 'string'  | COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}  | CONNECTION [=] 'connect\_string'  | {DATA|INDEX} DIRECTORY [=] 'absolute path to directory'  | DELAY\_KEY\_WRITE [=] {0 | 1}  | ENCRYPTION [=] {'Y' | 'N'}  | ENGINE [=] engine\_name  | INSERT\_METHOD [=] { NO | FIRST | LAST }  | KEY\_BLOCK\_SIZE [=] value  | MAX\_ROWS [=] value  | MIN\_ROWS [=] value  | PACK\_KEYS [=] {0 | 1 | DEFAULT}  | PASSWORD [=] 'string'  | ROW\_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}  | STATS\_AUTO\_RECALC [=] {DEFAULT|0|1}  | STATS\_PERSISTENT [=] {DEFAULT|0|1}  | STATS\_SAMPLE\_PAGES [=] value  | TABLESPACE tablespace\_name [STORAGE {DISK|MEMORY|DEFAULT}]  | UNION [=] (tbl\_name[,tbl\_name]...)  partition\_options:  (see CREATE TABLE options) |

#### 修改表名

案例：

|  |
| --- |
| /\*注释  修改表名  \*/  alter table t\_student  rename to test\_student;  alter table test\_student  rename to t\_student; |

#### 增加列

案例：

|  |
| --- |
| /\*注释  表中增加列  \*/  alter table t\_student  add column status int;  alter table t\_student  add column col\_add int(4) default 1; |

#### 修改列

案例：

|  |
| --- |
| /\*注释  表中修改列的名称和类型  \*/  alter table t\_student  change column col\_add u\_status char(4);  alter table t\_student  change column col\_add col\_two char(4) not null default '1'; |

#### 删除列

案例：

|  |
| --- |
| /\*注释  删除表中的某列  \*/  alter table t\_student  drop column u\_status; |

### 删除表

语法：

|  |
| --- |
| DROP TABLE [IF EXISTS] tbl\_name |

案例：

|  |
| --- |
| /\*注释  删除表  \*/  drop table if EXISTS t\_student; |

## 表中数据的操作

### 增加数据

语法：

|  |
| --- |
| INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]  [INTO] tbl\_name  [PARTITION (partition\_name [, partition\_name] ...)]  [(col\_name [, col\_name] ...)]  {VALUES | VALUE} (value\_list) [, (value\_list)] ...  [ON DUPLICATE KEY UPDATE assignment\_list]  INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]  [INTO] tbl\_name  [PARTITION (partition\_name [, partition\_name] ...)]  SET assignment\_list  [ON DUPLICATE KEY UPDATE assignment\_list]  INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE]  [INTO] tbl\_name  [PARTITION (partition\_name [, partition\_name] ...)]  [(col\_name [, col\_name] ...)]  SELECT ...  [ON DUPLICATE KEY UPDATE assignment\_list]  value:  {expr | DEFAULT}  value\_list:  value [, value] ...  assignment:  col\_name = value  assignment\_list:  assignment [, assignment] ... |

案例1：

|  |
| --- |
| /\*注释  增加数据  \*/  insert into t\_student(stuNo, stuName, stuAge,stuSex,grade,subjectName,score,phone)  values(1001,'张三',22,'男','一年级','java',86,'13502344345'); |

案例2：

|  |
| --- |
| /\*注释  批量增加数据  \*/  insert into t\_student(stuNo,stuName,stuAge,stuSex,grade,subjectName,score,phone)  values(1002,'李四',23,'男','二年级','python',87,'13815463587'),  (1003,'王五',24,'女','一年级','java',96,'13702344345'); |

案例3：

|  |
| --- |
| create table t\_result(  stuName varchar(20),  subjectName varchar(10),  score int  );  /\*  从一张表中将多条记录批量插入到另一张表中  \*/  insert into t\_result(stuName,subjectName,score)  select stuName,subjectName,score from t\_student; |

### 查询数据

语法：

|  |
| --- |
| SELECT  [ALL | DISTINCT | DISTINCTROW ]  [HIGH\_PRIORITY]  [STRAIGHT\_JOIN]  [SQL\_SMALL\_RESULT] [SQL\_BIG\_RESULT] [SQL\_BUFFER\_RESULT]  [SQL\_CACHE | SQL\_NO\_CACHE] [SQL\_CALC\_FOUND\_ROWS]  select\_expr [, select\_expr ...]  [FROM table\_references  [PARTITION partition\_list]  [WHERE where\_condition]  [GROUP BY {col\_name | expr | position}  [ASC | DESC], ... [WITH ROLLUP]]  [HAVING where\_condition]  [ORDER BY {col\_name | expr | position}  [ASC | DESC], ...]  [LIMIT {[offset,] row\_count | row\_count OFFSET offset}]  [PROCEDURE procedure\_name(argument\_list)]  [INTO OUTFILE 'file\_name'  [CHARACTER SET charset\_name]  export\_options  | INTO DUMPFILE 'file\_name'  | INTO var\_name [, var\_name]]  [FOR UPDATE | LOCK IN SHARE MODE]] |

案例：

|  |
| --- |
| /\*  查看表中的所有数据  \*表示所有列  \*/  select \* from t\_student;  select DISTINCT stuAge from t\_student;  select \* from t\_student where stuName='李四';  select stuName, subjectName, sum(score) as avg\_score from t\_student group by stuName, subjectName order by avg\_score desc; |

### 删除数据

DELETE命令：

语法：

|  |
| --- |
| DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM tbl\_name  [PARTITION (partition\_name [, partition\_name] ...)]  [WHERE where\_condition]  [ORDER BY ...]  [LIMIT row\_count] |

案例：

|  |
| --- |
| /\*\*  删除数据 使用delete命令  delete后如果不加where条件 那么将会删除表中所有数据  \*/  delete from t\_student where stuNo=1001; |

TRUNCATE命令：

语法：

|  |
| --- |
| TRUNCATE [TABLE] tbl\_name |

案例：

|  |
| --- |
| /\*注释  类似 delete from t\_student;  区别：   1. truncate其实是删除表后，再重新建立表 2. 速度非常快 3. 不会写入mysql日志 4. 不能回滚 5. 会重置自增字段为初始值 6. 不会触发删除触发器   \*/  truncate table t\_student; |

### 修改数据

语法：

|  |
| --- |
| UPDATE [LOW\_PRIORITY] [IGNORE] table\_reference  SET assignment\_list  [WHERE where\_condition]  [ORDER BY ...]  [LIMIT row\_count]  value:  {expr | DEFAULT}  assignment:  col\_name = value  assignment\_list:  assignment [, assignment] ... |

案例：

|  |
| --- |
| /\*\*  修改数据 使用update命令  update后跟表名  set跟要修改的列名及修改后的值  where跟条件，如果没有where条件，那么表中的所有数据都将被修改  特别注意where后的条件，如果有多个条件使用逻辑运算符连接 and or !  \*/  update t\_student set score=86 where stuNo=1002; |

## 约束

### 主键约束

主键约束primary key，其实只是一个声明，内部是使用了唯一性约束、非空约束

#### 增加主键

案例1：

|  |
| --- |
| /\*注释  建立表时设置主键  \*/  create table t\_user(  id int primary key,  name varchar(20),  age int  ); |

案例2：

|  |
| --- |
| /\*注释  增加主键  \*/  alter table t\_user  add primary key(id); |

案例3：

|  |
| --- |
| /\*注释  修改列为主键列  \*/  alter table t\_user  change column id id int primary key; |

#### 删除主键

案例1：

|  |
| --- |
| /\*注释  删除表中的主键，该命令只是把 列 的主键属性删除了，对 列 本身没有影响  \*/  alter table t\_user  drop primary key;  /\*注释  如果主键 id 是自增字段，会删除失败  自增字段必须建立在唯一索引列上  以下代码，修改id列为普通的int 类型 的主键  \*/  alter table t\_user  change column id id int ; |

案例2：

|  |
| --- |
| /\*注释  删除主键  \*/  drop index `primary` on t\_user\_n; |

### 唯一约束

唯一性约束unique，不能有重复值，但是可以为null

只有唯一性约束才能设置自增

#### 增加唯一约束

案例1：

|  |
| --- |
| /\*注释  建立表时设置唯一约束  \*/  create table t\_user\_u(  id int primary key,  name varchar(20) unique,  age int,  sex int  ); |

案例2：

|  |
| --- |
| /\*注释  增加唯一约束  \*/  alter table t\_user\_u  add unique (age);  /\*注释  增加唯一约束，完整写法  \*/  alter table t\_user\_u  add unique index age\_unique (age asc); |

案例3：

|  |
| --- |
| /\*注释  修改列为唯一约束列  \*/  alter table t\_user\_u  change column sex sex int unique; |

#### 删除唯一约束

案例1：

|  |
| --- |
| /\*注释  删除唯一约束  \*/  alter table t\_user\_u  drop index sex\_unique; |

案例2：

|  |
| --- |
| /\*注释  删除唯一约束  \*/  drop index `age\_unique` on t\_user\_u; |

### 默认值约束

默认值约束 default，当增加数据的时候没有填写，则保存为默认值，没有设置时，默认值为null

#### 增加默认值

案例1：

|  |
| --- |
| /\*注释  建立表时设置 默认值  \*/  create table t\_user\_d(  id int primary key auto\_increment,  name varchar(20) default 'test',  age int,  sex int  );  insert into t\_user\_d(age, sex) values(21, 1);  insert into t\_user\_d(name, age, sex) values('no\_default', 22, 2); |

案例2：

|  |
| --- |
| /\*注释  增加默认值  \*/  alter table t\_user\_d  alter age set default 12; |

案例3：

|  |
| --- |
| /\*注释  修改列来增加默认值  \*/  alter table t\_user\_d  change column sex sex int default 2; |

#### 删除默认值

案例：

|  |
| --- |
| /\*注释  删除或修改默认值  \*/  alter table t\_user\_d  change column sex sex int default null; |

### 非空约束

不约束的话 not null，字段默认是允许为null的，添加约束后，字段值不允许为null，但是可以配合默认值约束后，不会出现null，除非自己手动插入null值

#### 增加非空约束

案例1：

|  |
| --- |
| /\*注释  建立表时设置非空  \*/  create table t\_user\_n(  id int primary key auto\_increment,  name varchar(20) default 'test',  age int not null,  sex int  );  insert into t\_user\_n(name, sex) values('test', 1); #会失败  insert into t\_user\_n(age, sex) values(18, 1); #成功 |

案例2：

|  |
| --- |
| /\*注释  修改列来设置  \*/  alter table t\_user\_n  change column sex sex int not null; |

#### 删除非空约束

案例：

|  |
| --- |
| /\*注释  删除或修改非空  \*/  alter table t\_user\_n  change column sex sex int null; |

### 外键约束

外键约束foreign key

外键必须与参照列的数据类型必须相同（数值型要求长度和无符号都相同，字符串要求类型相同，长度可以不同），参照列必须是主键或唯一约束的列

只有引擎INNODB的数据库引擎支持外键

外键检查4个不同的规则：

Restrict：主表update/delete记录时，子表如果有关联记录，提示错误，操作失败

Cascade：主表update/delete记录时，子表如果有关联记录，同步级联update/delete

Set Null：主表update/delete记录时，子表如果有关联记录，设置为 null

No action：同restrict

#### 增加外键

案例1：

|  |
| --- |
| /\*注释  创建表时建立外键  \*/  create table t\_user(  id int primary key auto\_increment,  name varchar(20),  age int  );  CREATE TABLE `t\_grade` (  `id` INT NOT NULL AUTO\_INCREMENT,  `user\_id` INT NULL,  `name` VARCHAR(45) NULL,  PRIMARY KEY (`id`),  CONSTRAINT `fk\_t\_user`  FOREIGN KEY (`user\_id`)  REFERENCES `t\_user` (`id`)  ON DELETE CASCADE  ON UPDATE CASCADE); |

案例2：

|  |
| --- |
| /\*注释  修改表，增加外键  不能命名重复的外键  自动增加一个同名index  \*/  create table t\_user\_fk(  id int primary key auto\_increment,  user\_id int,  name varchar(20),  age int  );  ALTER TABLE `t\_user\_fk`  ADD CONSTRAINT `fk1\_t\_user`  FOREIGN KEY (`user\_id`)  REFERENCES `t\_user` (`id`)  ON DELETE NO ACTION  ON UPDATE NO ACTION; |

#### 删除外键

案例：

|  |
| --- |
| /\*注释  删除外键  连索引一起删除  \*/  ALTER TABLE `t\_user\_fk`  DROP FOREIGN KEY `fk1\_t\_user`;  ALTER TABLE 't\_user\_fk`  DROP INDEX `fk1\_t\_user` ; |

## 运算符

|  |
| --- |
| /\*注释  创建测试表  \*/  create table t\_operator(  id int primary key auto\_increment,  a int,  b int,  c varchar(45),  d varchar(45)  );  insert into t\_operator values(1, 100, 30, '123456', 'abcdef');  insert into t\_operator values(2, 97, 15, '1111', 'aaaa');  insert into t\_operator values(3, 82, 19, '2222', 'bbbb'); |

### 算术运算符

“+”，“-”，“\*”，“/”（有小数点）和“div（整除）

|  |
| --- |
| /\*注释  测试算术运算符  \*/  select a, b, a+b as s from t\_operator;  select a, b, a-b from t\_operator;  select a, b, a\*b from t\_operator;  select a, b, a/b from t\_operator;  select a, b, a div b from t\_operator; |

### 比较运算符

“>”，“>=”，“=”，“<”，“<=”，“<>”，”!=”

|  |
| --- |
| /\*注释  测试比较运算符  \*/  select 1<>1; /\*0\*/  select 1!=1; /\*0\*/  select 1=1; /\*1\*/ |

### 逻辑运算符

“and”，“&&”

“or”，“||”

“not”，“!”

“xor”：a xor b 等于 (a AND (NOT b)) OR ((NOT a) and b)，即a和b 这两个条件必须一真一假

true ,false,null 使用 is 和is not

|  |
| --- |
| /\*注释  测试逻辑运算符  \*/  SELECT 10 IS TRUE;  SELECT 'string' IS NOT NULL;  SELECT NOT 10;  SELECT NOT NULL;  SELECT ! (1+1); |
|  |

### 位运算符

“|”，“^”，“&”

|  |
| --- |
| /\*注释  测试位运算符  \*/  select 1 | 1;  select 1^2; /\* 异或 \*/  select 1&0; |

## 函数

### 数学函数

* Abs(x) 绝对值
* CEILING(X)/ CEIL(X) 向上取整
* FLOOR(X) 向下取整
* MOD(x,y) 取余 %
* PI() 圆周率
* POW(X,Y) x的y次方
* RAND() RAND(N) 返回一个随机浮点值 v ，范围在 0 到1 之间 (即, 其范围为 0 ≤ v ≤ 1.0)。若已指定一个整数参数 N ，则它被用作种子值，用来产生重复序列，同一个N产生的值永远一样
* ROUND(X) ROUND(X,D) 返回参数X, 其值接近于最近似的整数。在有两个参数的情况下，返回 X ，其值保留到小数点后D位，而第D位的保留方式为四舍五入。若要接保留X值小数点左边的D 位，可将 D 设为负值。
* TRUNCATE(X,D) 返回被舍去至小数点后D位的数字X。若D 的值为 0, 则结果不带有小数点或不带有小数部分。可以将D设为负数,若要截去(归零) X小数点左起第D位开始后面所有低位的值

|  |
| --- |
| /\*注释  测试数学函数  \*/  select abs(-1);/\*1\*/  select CEIL(13.1);/\*14\*/  select FLOOR(13.1);/\*13\*/  select mod(15,9);/\*6\*/  select 15%9;/\*6\*/  select pi();  select rand();/\*产生随机数\*/  select round(5.545,2);/\*四舍五入\*/  select truncate(5.545,2);/\*5.54\*/ |

.

### 字符串函数

* ASCII(str) 返回字母ascII码值
* BIN(x) 返回二进制
* BIT\_LENGTH(str) 返回值为二进制的字符串str 长度。
* LENGTH(STR) 返回字符串长度
* CONCAT(str1,str2,...) 字符串连接

|  |
| --- |
| /\*注释  测试字符串函数  \*/  SELECT ASCII('2'); /\*50\*/  select bin(20);/\*10100\*/  select BIT\_LENGTH(20);/\*16\*/  select length('hello world');/\*11\*/  select CONCAT('this',' is',' sunny');/\*this is sunny\*/  select LOWER('ADV');/\*转为小写\*/  select upper (adv);/\*转为大写\*/  select LTRIM(' ad');/\*去掉左边的空格\*/ |

.

### 日期和时间函数

|  |
| --- |
| /\*注释  测试日期和时间函数  \*/  select now();/\*返回当前日期时间\*/  select CURDATE();/\*返回当前日期\*/  select CURRENT\_DATE();/\*返回当前日期\*/  select CURTIME();/\*返回当前时间\*/  select current\_time();/\*返回当前时间\*/  select CURRENT\_TIMESTAMP();  select DATEDIFF(now(), '20180101'); /\*计算2个日期相差的天数\*/  select TIMESTAMPDIFF(YEAR, '2013-01-01', now());/\*计算2个日期相差的年数，YEAR有 SECOND、MINUTE、 HOUR、 DAY、 WEEK、 MONTH、 QUARTER或 YEAR\*/  select TIMESTAMPADD(YEAR, 1, now()); /\*在当前日期上增加1年\*/  /\*  ADDDATE(date,INTERVAL expr type) ADDDATE(expr,days)  \*/  SELECT DATE\_ADD('1998-01-02', INTERVAL 30 MONTH);  select day('2010-10-12');/\*返回指定日期的天数1-31\*/  select month('2010-10-12');/\*返回指定日期的月份1-12\*/  select week('2010-10-12');/\*返回指定日期的周数1-52\*/  select weekday('2017-5-8');/\*返回指定日期的周几0-6 周日是6\*/ |

.

### 加密函数

|  |
| --- |
| /\*注释  测试加密函数  \*/  select md5('root');/\*md5加密\*/  select SHA('root');  select PASSWORD('root');  select ENCODE('123','root');/\* 123 编码 'root'是秘钥\*/  select DECODE('XMx','root');/\* XMx 解码 'root'是秘钥\*/ |

.

### 流程函数

CASE value WHEN [compare-value] THEN result [WHEN [compare-value] THEN result ...] [ELSE result] END

|  |
| --- |
| /\*  Case when then end  \*/  select case 3 when 1 then 'a' when 2 then 'b' else 'c' end; |

.

IF(expr1,expr2,expr3)

如果 expr1 是TRUE (expr1 <> 0 and expr1 <> NULL)，则 IF()的返回值为expr2; 否则返回值则为 expr3。

|  |
| --- |
| select if(2>1,2,1);/\*2\*/ |

.

IFNULL(expr1,expr2)

假如expr1 不为 NULL，则 IFNULL() 的返回值为 expr1; 否则其返回值为 expr2。

|  |
| --- |
| select ifnull('1','a'); |

.

NULLIF(expr1,expr2)

如果expr1 = expr2 成立，那么返回值为NULL，否则返回值为 expr1。

|  |
| --- |
| select nullif(1,2);  select nullif(2,2); |

.

### 聚合函数

max，min，count，avg，sum

# 查询语句

## 基础查询

|  |
| --- |
| /\*  select 的语法  select \* from tbl\_name  FROM 子句  where 子句  group by  HAVING  order by  limit  \*/  /\*  查询所有  \*/  select \* from t\_student;  /\*  查询指定列  \*/  select name,age from t\_student;  /\*  为列设置别名  \*/  select name '姓名',age '年龄' from t\_student;  select name as '姓名',age as '年龄' from t\_student;  /\*  为表指定别名  \*/  select s.name as '姓名',s.age as '年龄' from t\_student s;  /\*  条件查询  查询年龄大于21的所有学生  \*/  select \* from t\_student where age>21;  /\*  多个条件可以使用and 或者or进行条件的连接  查询年龄大于21并且是男生学员  \*/  select \* from t\_student where age>21 and sex='男';  /\*  模糊查询 使用like  %表示匹配0个或者多个字符  \_表示匹配1个字符  当like后没有模式匹配符时 是完全匹配  查询姓张的所有学生  \*/  select \* from t\_student where name like '张三';  /\*  正则表达式匹配  “[abc]”：匹配“a”、“b”或“c”。为了命名字符的范围，使用一个“-”。  “[a-z]”：匹配任何字母  “[0-9]”：匹配任何数字  “ \* ”：匹配零个或多个在它前面的字符。例如，“x\*”匹配任何数量的“x”字符，“[0-9]\*”匹配任何数量的数字，而“.\*”匹配任何数量的任何字符。  “{n}”：重复n次操作符  "^”：开头标记  "$”：结尾标记  a?：匹配0个或1个a字符  de|abc：匹配序列de或abc  \*/  select \* from t\_student where name regexp '^张.\*';  /\*  is null 查询  查询没有电话的所有学生信息  \*/  select \* from t\_student where phone is null;  /\*  BETWEEN and  查询年龄在21到32之间学生信息包括21和32  \*/  select \* from t\_student where age BETWEEN 21 and 32;  /\*  DISTINCT 去重  查询学生年龄有哪些  \*/  select distinct age from t\_student;  /\*  排序 order by  查询学生信息 并且按年龄由小到大排序  asc;表示升序  desc 表示降序  \*/  select \* from t\_student order by age asc;  select \* from t\_student order by age desc;  /\*  限制结果数量 limit x,y x表示开始下标(从0开始) y表示个数  分页  查询年龄最大的2个学生信息  \*/  select \* from t\_student order by age desc LIMIT 0,2;  /\*  聚合函数  count()计数  max()最大值  min()最小值  avg()平均值  sum()求和  统计学生人数，及学生的最大年龄，最小年龄 平均年龄，总年龄  \*/  select count(\*) as '总人数',max(age),min(age),avg(age),sum(age) from t\_student;  /\*  分组查询 group by，分组查询，查询列 只能是分组列和聚合列  查询男生人数和女生人数  \*/  select sex,count(\*) from t\_student  group by sex;/\*按性别分组\*/  /\*  分组后条件查询 HAVING  查询性别大于2的人数的性别及人数  \*/  select sex,count(\*) from t\_student  group by sex  having count(\*)>2; |
|  |

.

## 连接查询

连接查询指查询的结果需要从多张表中获取。连接查询也叫多表查询。

### 内连接

|  |
| --- |
| /\*  内连接  笛卡尔积(a,b)\*(c,d) ac,ad,bc,bd  直接从多张表中获取数据的结果是多张表的笛卡尔积  查询学生姓名和年级  内连接获取数据，多张表之间应该是有关系(外键)的。  \*/  select s.name '姓名',g.name '年级' from t\_student s,t\_grade g where s.gradeId=g.id;  /\*  获取张三的java成绩  分析：数据从哪些表中获取  t\_student,t\_subject,t\_result  \*/  select score from t\_student stu,t\_subject sub,t\_result r  where stu.name='张三' and sub.name='java' and stu.id=r.stuId and sub.id=r.subjectId;  /\*  使用inner join来完成内连接  查询学生姓名和年级  \*/  select s.name,g.name from t\_student s inner join t\_grade g on s.gradeId=g.id;  /\*  获取张三的java成绩  \*/  select score from t\_student stu inner join t\_result r inner join t\_subject sub  on stu.id = r.stuId and r.subjectId=sub.id  where stu.name='张三' and sub.name='java'; |

### 外连接

外连接也是多表之间数据的查询。外连接是以一张表为主来进行查询的连接。外连接分为左外连接 left join和右外连接 right join

|  |
| --- |
| select e.name,d.name from t\_emp e left join t\_dept d on e.deptId=d.id; |

### 内连接

inner join

|  |
| --- |
| select e.name,d.name from t\_emp e inner join t\_dept d on e.deptId=d.id; |

## 子连接

一个查询结果作为另外一个查询的条件或者组成部分的查询称为子查询。

|  |
| --- |
| /\*  查询 年级id为1,2的所有学生信息  \*/  select \* from t\_student where gradeId=1 or gradeId=2;  /\*  使用in来查询 in指在某个范围中  \*/  select \* from t\_student where gradeId in (1,2);  特征：  select \* from t\_student where stuNo in (select stuNo from t\_grade);   1. in 只能作用于的单个字段 2. 后面查询出来的结果，也必须是只有一个字段的结果集 3. 执行顺序是：   1、先把 小括号 中的结果集查询出来  2、遍历 in 前面的记录集，进行匹配，看前面结果集的指定字段stuNo是否在小括号的结果集中  3、如果在结果集中，那么就返回，如果不在结果集中，那么就不会返回该条记录  exists：  select s.\* from t\_student as s where exists(select \* from t\_grade as g where s.stuNo=g.stuNo);   1. exists前面是没有字段的！！！！ 2. 小括号是必须使用exists前面的表的字段进行判断 s.stuNo=g.stuNo 3. 小括号内的结果集，字段数无关紧要 4. 执行顺序： 5. 计算小括号前的结果集并且进行遍历 6. 把前面结果集的字段放入到后年的额小括号内进行查询，如果能够查询到记录，那么就算成功匹配 7. 匹配成功则返回该条记录，匹配不成功，则不返回该条记录   /\*  查询年级名称为一年级和二年级的学生信息  分析：  要获取学生信息 那么从t\_student表中获取数据  t\_student 只有年级的编号  需要根据年级的名称去获取年级的编号（t\_grade）  \*/  select id from t\_grade where name='一年级' or name='二年级';  select \* from t\_student where gradeId in (  select id from t\_grade where name='一年级' or name='二年级'  );  /\*  查询java成绩大于90分的学生信息  分析：  根据科目名称获取科目编号  根据科目编号及成绩获取学生编号  根据学生编号获取学生信息  \*/  select id from t\_subject where name='java';  select stuId from t\_result where subjectId=(select id from t\_subject where name='java';) and score>90;  select \* from t\_student  where id in (  select stuId from t\_result  where subjectId=(  select id from t\_subject  where name='java')  and score>90);  /\*  子查询通过in来查询的是一个集合  还可以通过比较运算符来查询  要求比较运算符中的子查询只有一个结果  查询一年级的所有学生信息  \*/  select \* from t\_student where gradeId=(select id from t\_grade where name='一年级');  /\*  子查询的效率要高于连接查询  \*/  /\*  使用外连接来查询  查询所有学生的java考试成绩  \*/  select id from t\_subject where name='java';  select \* from t\_result where subjectId=(select id from t\_subject where name='java')  select s.name,r.score from t\_student s  left outer join  (select \* from t\_result where subjectId=(select id from t\_subject where name='java')) r  on s.id=r.stuId;  /\*  统计每个部门的人数  \*/  select deptId,count(\*) cnt from t\_emp  group by deptId;  select d.name,e.cnt from t\_dept d  left join  (select deptId,count(\*) cnt from t\_emp  group by deptId) e  on d.id=e.deptId;  /\*  查询有两门课程考试成绩在80分以上的学生信息  \*/  select \* from t\_result where score>80;  select r.stuId stuId from  (select \* from t\_result where score>80) r  group by stuId  HAVING count(\*)>=2;  select \* from t\_student where id in(  select r.stuId stuId from  (select \* from t\_result where score>80) r  group by stuId  HAVING count(\*)>=2  )  /\*  EXISTS 子查询  如果有考试成绩为100的，那么取出所有学生信息  \*/  select \* from t\_student  where exists(select \* from t\_result where score=100);  /\*  ALL 表示每一个  所有考试成绩都在80分以上的学生姓名  \*/  select name from t\_student s  where 80<all(select score from t\_result r where s.id=r.stuId)  /\*  假如只查询有3个结果 a,b,c  all 相当于a and b and c  \*/  /\*  any 表示任意一个  考试成绩有80分以上的学生姓名  \*/  select name from t\_student s  where 80<=any(select score from t\_result r where s.id=r.stuId) |
|  |

## 联合查询

使用union和union all查询

|  |
| --- |
| /\*  union 查询将多个查询结果组合在一起，并且去重  union all 查询将多个查询结果组合在一起，但是不去重  查询编号为1的年级和编号为2的年级的学生姓名  联合的多个结果集的字段数量要一致  \*/  select name from t\_student where gradeId=1  UNION  select name from t\_student where gradeId=2;  select name from t\_student where gradeId=1  UNION all  select name from t\_student where gradeId=1 or gradeId=2; |
|  |

## 行转列

|  |
| --- |
| create table t\_score(  name varchar(20),  subject varchar(20),  score int  );  insert into t\_score(name,subject,score)  values('张三','java',80),  ('张三','mysql',84),  ('李四','java',78),  ('李四','mysql',90),  ('王五','java',60),  ('王五','mysql',88);  select \* from t\_score;  /\*\*  name java mysql  张三 80 84  李四 78 90  王五 60 88  \*/  select name,  max(case subject when 'java' then score else 0 end) 'java',  max(case subject when 'mysql' then score else 0 end) 'mysql'  from t\_score  group by name; |

## 列转行

|  |
| --- |
| create table t\_score1(  name varchar(20),  java int,  mysql int  );  insert into t\_score1  select name,  max(case subject when 'java' then score else 0 end) 'java',  max(case subject when 'mysql' then score else 0 end) 'mysql'  from t\_score  group by name;  select \* from t\_score1;  select name,'java' subject,java score from t\_score1  UNION  select name,'mysql' subject,mysql score from t\_score1  order by name |

# 事务

* 在处理复杂的应用时，有很多sql语句需要同时执行。要求保证同时成功或者同时失败。在mysql中InnoDB引擎支持事务操作。在mysql默认情况每执行一条sql，该sql就是一个事务操作。事务具有ACID4个特性
* 原子性(Atomicity):每个事务是一个不可再分的独立单元
* 一致性(Consistency):在执行事务前和执行后数据总和保持一致
* 隔离性(Isolation):多个事务之间不影响
* 持久性(Durability):执行事务后，数据持久的写入数据库。
* 在mysql中事务可以通过begin来显示声明表示事务的开启
* 通过commit来表示提交事务
* 通过rollback来表示回滚事务

事务的隔离级别：

1. 读未提交（read-uncommitted）：就是常说的脏读，A事务执行的操作，即使还未commit，此时另一个事务B就可以查看到A事务的操作
2. 不可重复读（read-committed）
3. 可重复读（repeatable-read）
4. 串行化（serializable）

设置命令：

set session transaction isolation level repeatable read;

也可以在配置文件中配置：

transaction-isolation = REPEATABLE-READ

|  |
| --- |
| START TRANSACTION  [transaction\_characteristic [, transaction\_characteristic] ...]  transaction\_characteristic:  WITH CONSISTENT SNAPSHOT  | READ WRITE  | READ ONLY  BEGIN [WORK]  COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]  ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]  SET autocommit = {0 | 1}  drop table t\_user;  create table t\_user(  name varchar(20) primary key,  money int unsigned  );  /\*  查看自动提交状态  \*/  show variables like 'autocommit';  set autocommit = 0;  /\*  在默认情况下 一条sql语句就是一个事务  insert是一个事务  \*/  insert into t\_user(name,money)  values('张三',500),('李四',100);  select \* from t\_user;  /\*  BEGIN  # 不要在 workbench 中演示 set autocommit = 0 ， 这个工具会脏读！！！  到 cmd 中演示  \*/  start transaction;  update t\_user set money=money+300 where name='李四';  update t\_user set money=money-300 where name='张三';  commit; |
|  |

# 视图

视图是一张虚拟表，通过从多张表中获取数据动态生成。

|  |
| --- |
| /\*  建立视图  \*/  create view v\_studentInfo  AS  select s.name as '学生姓名',g.name '年级'  from t\_student s,t\_grade g  where s.gradeId=g.id;  /\* 查询视图和表一样 \*/  select \* from v\_studentInfo; |

# 索引

索引的目的是为了快速找到目标数据。

在数据库，如果数据量比较大时，查询效率比较低时可以考虑建立索引。

索引的建立应该选择不经常改变的字段。

|  |
| --- |
| /\*  union 查询将多个查询结果组合在一起，并且去重  /\*  对已有表建立索引  \*/  alter table t\_student  add index idx\_name(name);  create unique index idx\_name  on t\_student(name);  /\*  在创建表时可以指定索引  \*/  drop table t\_tt;  create table t\_tt(  id int,  name varchar(20),  unique index idx\_name (name)  );  /\*  删除索引  \*/  alter table t\_student  drop index idx\_name;  主要有5种常用不同的索引  主键索引、唯一索引、普通索引、全文索引、聚合索引  primary key(c1)  unique index index\_name (c1)  index index\_name (c1)  fulltext index index\_name(c1) : 仅仅在 myisam 引擎使用  index index\_name(c1, c2, c3) ：  从左作用，  where c1=1 and c2=1 and c3=1  where c1 = 1 and c2 =1  where c1 = 1  不起作用的： where c2 =1 and c3=1 |

# 存储程序

存储程序指的一组存储和执行在数据库服务器端的程序。存储程序总是在服务器的进程或者线程的内存中执行的。

存储程序分为：

* 存储过程：有输入输出参数，可以执行一组sql指令。
* 存储函数：有一个返回值，可以对sql进行有效扩展。
* 触发器：指事件响应，比如执行insert语句后执行另外一个动作。

## 存储过程

存储过程是一组存放在数据库服务器端的程序。可以有效的提高执行效率和安全性。

语法：

|  |
| --- |
| CREATE  [DEFINER = { user | CURRENT\_USER }]  PROCEDURE sp\_name ([proc\_parameter[,...]])  [characteristic ...] routine\_body  proc\_parameter:  [ IN | OUT | INOUT ] param\_name type  func\_parameter:  param\_name type  type:  Any valid MySQL data type  characteristic:  COMMENT 'string'  | LANGUAGE SQL  | [NOT] DETERMINISTIC  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }  | SQL SECURITY { DEFINER | INVOKER }  routine\_body:  Valid SQL routine statement |

示例：

|  |
| --- |
| -- 创建存储过程  delimiter //  create procedure hello()  begin  select 'hello procedure';  end//  delimiter ;  -- 调用存储过程  call hello();  -- 为存储过程指定参数 需要指定参数名以及参数类型  delimiter //  create procedure findStudentById(id int)  BEGIN  select id\*10 as id;  end//  delimiter ;  call findStudentById(3);  /\*  参数类型：  In 输入参数，默认  Inout 输入输出参数  Out 输出参数  输出参数的使用  \*/  -- 统计学生人数  delimiter //  create PROCEDURE totalCount(out cnt int)  BEGIN  -- 通过into 为输出参数赋值  select 12 into cnt;  end//  delimiter ;  -- @cnt 定义了一个用户变量  call totalCount(@cnt);  select @cnt '人数';  /\*  变量的使用，  \*/  delimiter //  create PROCEDURE vari\_demo()  BEGIN  -- 变量的声明使用declare  declare username VARCHAR(20);  -- 也可以在声明变量时 使用default为变量设置默认值  declare tt VARCHAR(20) DEFAULT 'ss';  -- 可以使用set为变量赋值  set username='root';  select username;  end//  delimiter ;  call vari\_demo();  -- 控制语句  delimiter //  create PROCEDURE discount(in price double,out finalPrice double)  BEGIN  if price >500 then  -- 赋值一定要使用set  set finalPrice=price\*0.8;  elseif price>300 then  set finalPrice=price\*0.9;  else  set finalPrice = price\*0.95;  end if;end//  delimiter ;  call discount(500,@finalPrice);  select @finalPrice;  -- 使用while来计算1....10的和  delimiter //  create PROCEDURE my\_sum(n int,out m\_sum int)  BEGIN  DECLARE i int;  set i=0;  set m\_sum=0;  -- while 和do中间 加条件  while i<=n DO  set m\_sum=m\_sum+i;  set i=i+1;  end while;  end//  delimiter ;  call my\_sum(10,@m\_sum);  select @m\_sum;  -- 使用loop来计算  delimiter //  create PROCEDURE my\_sum1(n int,out m\_sum1 int)  BEGIN  DECLARE i int;  set i=0;  set m\_sum1=0;  -- loop需加标记 方便leave时指定  lip:loop  set m\_sum1=m\_sum1+i;  set i=i+1;  if i>n THEN  leave lip;  end if;  end loop lip;  end//  delimiter ;  call my\_sum1(10,@m\_sum1);  select @m\_sum1; |
| --- 删除存储过程  drop procedure `P\_TestException`;  DELIMITER //  CREATE PROCEDURE `P\_TestException`(id1 int, id2 int)  BEGIN  declare \_err int default 0;  -- 定义，出现这个3种异常情况，代码块不会异常退出，继续执行，但是设置 -err的值为 1  declare continue handler for sqlexception, sqlwarning, not found set \_err=1;    start transaction;  set autocommit = 0;    insert into t\_student(id, stuName, stuAge) values(id1, '杨八', 33);  insert into t\_student(id, stuName, stuAge) values(id2, '杨九', 34);    if \_err=1 then  rollback;  else  commit;  end if;    set autocommit = 1;  select case when \_err = 1 then '出错了' else '执行成功' end;  END //  call P\_TestException(33,33); |

## 存储函数

存储函数和存储过程几乎一致，区别在于存储函数只有输入参数，存储函数有返回值

|  |
| --- |
| -- 自定义函数  delimiter //  create function studentCount()  RETURNS INT  DETERMINISTIC  BEGIN  declare cnt int;  select count(\*) into cnt from t\_student;  return cnt;  end//  delimiter ;  select studentCount();  drop function studentCount; |

## 触发器

语法

|  |
| --- |
| CREATE  [DEFINER = { user | CURRENT\_USER }]  TRIGGER trigger\_name  trigger\_time trigger\_event  ON tbl\_name FOR EACH ROW  [trigger\_order]  trigger\_body  trigger\_time: { BEFORE | AFTER }  trigger\_event: { INSERT | UPDATE | DELETE }  trigger\_order: { FOLLOWS | PRECEDES } other\_trigger\_name |

示例：

|  |
| --- |
| create table t1(  name VARCHAR(20)  );  create table t2(  name varchar(20)  )  delimiter //  create TRIGGER add\_t1\_t2  -- after insert , before insert ,after update, before update  after INSERT  on t1 FOR EACH ROW  BEGIN  insert t2(name) values('tt');  end//  delimiter ;  insert into t1(name) values('t1');  select \* from t1;  select \* from t2; |

# SQL优化

* 禁止使用select \*；
* 避免使用in,not in,or,having 使用exists和not exists替代，分内外表的大小和索引
* 避免使用union 使用union all来替代
* 避免使用like ‘%xx%’'%xx';
* 尽量少用join
* 尽量提前筛选条件
* 尽量使用索引来提高查询效率，不能滥用
* 尽量避免在where条件中进行运算
* 尽量批量处理数据
* 尽量避免使用 is null,<>,!=
* 少用排序
* 性能遇到瓶颈时处理方法：
* 建立索引
* 建立表分区
* 建立分区表
* 读写分离-分库
* 横向扩展集群，分布式

# 备份还原

备份：

|  |
| --- |
| Mysqldump -uroot -p123456 db\_name > d:/t.msql  /\* 不带密码 \*/  Mysqldump -uroot -p db\_name > d:/t.msql |

还原：执行备份生成的sql文件，就可以还原数据库了