1 DML

**DELETE**

**DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM tbl\_name**

**[WHERE where\_condition]**

**[ORDER BY ...]**

**[LIMIT row\_count]**

**#当一个表中有10行，表的ID字段属性为AUTO\_INCREMENT时，删除第四行， 新增一行的ID为11，这是因为，存在LAST\_INSERT\_ID()这个函数，新增行为该函 数值加一，DELETE 不会重置该函数。**

**TRUNCATE**

**TRUNCATE tb\_name**

**#清空表，并重置AUTO\_INCREMENT计数器**

**INSERT** 分三种插入方式

第一种：

**INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]**

**[INTO] tbl\_name [(col\_name,...)]**

**{VALUES | VALUE} ({expr | DEFAULT},...),(...),...**

**[ ON DUPLICATE KEY UPDATE**

**col\_name=expr**

**[, col\_name=expr] ... ]**

**#一次可以插入多行**

第二种：

**INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]**

**[INTO] tbl\_name**

**SET col\_name={expr | DEFAULT}, ...**

**[ ON DUPLICATE KEY UPDATE**

**col\_name=expr**

**[, col\_name=expr] ... ]**

**#一次只能插入一行**

第三种：

**INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE]**

**[INTO] tbl\_name [(col\_name,...)]**

**SELECT ...**

**[ ON DUPLICATE KEY UPDATE**

**col\_name=expr**

**[, col\_name=expr] ... ]**

REPLACE 与INSERT用法相同

**UPDATE**

**UPDATE [LOW\_PRIORITY] [IGNORE] table\_reference**

**SET col\_name1={expr1|DEFAULT} [, col\_name2={expr2|DEFAULT}] ...**

**[WHERE where\_condition]**

**[ORDER BY ...]**

**[LIMIT row\_count]**

MySQL用户密码修改：

1、# mysqladmin -u USERNAME -h HOSTNAME password 'NEW\_PASS' -p

2、mysql> SET PASSWORD FOR 'USERNAME'@'HOST'=PASSWORD('new\_pass');

3、mysql> UPDATE mysql.user SET PASSWORD=PASSWORD('new\_pass') WHERE CONDITION;

2 并发控制

锁：最简单的并发控制机制

读锁：共享锁

写锁：独占锁

表锁：MYSQL服务器支持表级别的锁，更高级的锁需要由存储引擎支持。

页锁：

行锁：

**LOCK TABLES tb\_name lock\_type;**

**UNLOCK TABLES;**解除所有文件的锁。

3 事务

ACID测试

原子性Atomicity：要么同时完成，要么同时完不成

一致性Consistency：状态和一致

隔离性Isolation：两个事务之间相互隔离，存在**隔离级别**，当两个事务都用一张 表时，可能会产生冲突，MySQL会提供许多策略进行事务调度，尽量规避冲突。

READ-UNCOMMITTED：读未提交，事务1改动，事务2立刻能看见

READ-COMMITTED：读提交，事务1提交后，事务2才能看见

REPATABLE-READ：可重读，到事务1提交前，第一次看见什么样，第二次也 是这样

SERIABLITABLE：串行

**SHOW GLOBAL VARIABLES LIKE ‘tx\_isolation’;**查看当前隔离级别

持久性Durability：事务完成后，即使服务器宕机，数据仍不会改变。

事务日志：为了使事务能有以上机制，引入了事务日志的概念。

Redo log:重做日志文件，也就是下图中的Log File文件，每个数据库至少要有两个 日志组，每个日志组至少要有一个日志文件，循环使用，日志组1满了后使用日 志组2，然后将1的数据写入磁盘，并清空1.

Undo log:撤销日志，存储在数据库内部或者表空间中。

用户空间内存

内核空间内存

Data File

LOG File

假设为了完成一个事务需要四个步骤（每个步骤都是一个命令），每个步骤都先在内存中完成，然后写入Log File（**Log File文件写入的是操作步骤本身，而不是数据，并且事务日志记录在磁盘上是顺序IO，也就是连续的block**），然后在内存中执行下一个步骤，再写入Log File文件，最后根据Log File的步骤将改变的数据写入Data File文件。

当事务提交完成（也就是在内存中完成了四个步骤，已经写入了Log File），这时服务器宕机了，重启服务器后，服务器会根据Log File的记录的步骤重新将数据写入Data File

当在内存中4个步骤仅仅完成两个，这时服务器宕机了，重启服务器后，服务器会根据undo log文件所记录的每条步骤执行前的状态，将日志文件恢复到之前的状态中。