**一、概述**

MySQL有三种锁的级别：页级、表级、行级。

MyISAM和MEMORY存储引擎采用的是表级锁（table-level locking）

BDB存储引擎采用的是页面锁（page-levellocking），但也支持表级锁

InnoDB存储引擎既支持行级锁（row-level locking），也支持表级锁，但默认情况下是采用行级锁

页锁 - 页级的典型代表引擎为BDB。

表锁 - 典型代表引擎为MyISAM,MEMORY以及很久以前的ISAM。

行锁 - 典型代表引擎为INNODB。

MySQL这3种锁的特性可大致归纳如下：

表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高,并发度最低。

行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。

页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

MyISAM使用的是 flock 类的函数，直接就是对整个文件进行锁定（叫做文件锁定），InnoDB使用的是 fcntl 类的函数，可以对文件中局部数据进行锁定（叫做行锁定），所以区别就是在这里。

**语法：**

**表级锁（MyISAM、Innodb）：**

LOCK TABLES tbl\_name [[AS] alias] lock\_type [, tbl\_name [[AS] alias] lock\_type] ...

lock\_type:

READ [LOCAL] | [LOW\_PRIORITY] WRITE

UNLOCK TABLES

**全局锁：**

FLUSH TABLES WITH READ LOCK

执行了命令之后所有库所有表都被锁定只读。一般都是用在数据库联机备份，这个时候数据库的写操作将被阻塞，读操作顺利进行。

UNLOCK TABLES

**行级锁（Innodb）：**

**共享锁(S LOCK)**

用法：SELECT ...LOCK IN SHARE MODE;

Mysql会对查询结果中的每行都加共享锁，当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请共享锁，否则会被阻塞。其他线程也可以读取使用了共享锁的表，而且这些线程读取的是同一个版本的数据。

**排他锁(X LOCK)**

用法：SELECT ... FOR UPDATE;

**二、怎么查看和解除死锁：**

查看进程列表信息， 找到死锁进程Id，然后 kill　processNo 关闭这个进程

show processlist; 只列出前100条

如果想全列出请使用 show full processlist;

查看正在锁的事务

mysql> select \* from information\_schema.innodb\_locks;

Empty set (0.00 sec)

查看等待锁的事务

mysql> select \* from information\_schema.innodb\_lock\_waits;

Empty set (0.00 sec)

通过下面的语句查看 innodb 状态

show innodb status;

show engine innodb status; （mysql 5.5之后）

show status like '%lock%';

**三、测试**

**（1）测试1 （加表写锁，其他会话不能对表执行读写操作，本会话可以执行读写操作）**

启动一个mysql 会话1，对表tbContainer设置写锁

mysql> lock tables tbContainer write;

Query OK, 0 rows affected (0.00 sec)

启动另一个mysql会话2，对表tbContainer查询操作，会发现查询操作被挂起

mysql> select \* from tbContainer;

在会话1中解锁表tbContainer （**注意：解锁需要在同一个mysql会话， 在其他mysql会话执行解锁没有作用**）

mysql> unlock tables;

Query OK, 0 rows affected (0.00 sec)

会话2刚才挂起的查询操作显示查询出结果。

mysql> select \* from tbContainer;

+----+-----------+-------------+

| Id | Ip | Description |

+----+-----------+-------------+

| 2 | 170.0.0.2 | NULL |

| 4 | 170.0.0.3 | NULL |

| 5 | 170.0.0.4 | NULL |

| 6 | 170.0.0.6 | NULL |

+----+-----------+-------------+

4 rows in set (1 min 33.75 sec)

**（2）测试2 （加表读锁，其他会话只能执行读操作不能执行写操作，本会话可以执行读操作不能执行写操作）**

启动一个mysql会话1，对表tbContainer设置读锁

mysql> lock tables tbContainer read;

Query OK, 0 rows affected (0.00 sec)

启动另一个mysql会话2，对表tbContainer查询操作，可以查询到结果。

mysql> select \* from tbContainer;

+----+-----------+-------------+

| Id | Ip | Description |

+----+-----------+-------------+

| 2 | 170.0.0.2 | NULL |

| 4 | 170.0.0.3 | NULL |

| 5 | 170.0.0.4 | NULL |

| 6 | 170.0.0.6 | NULL |

+----+-----------+-------------+

4 rows in set (0.00 sec)

在mysql会话2，对表tbContainer写操作会被挂起

mysql> insert into tbContainer values(7, "170.0.0.7", null);

在mysql会话1，对表tbContainer写操作会提示信息

mysql> insert into tbContainer values(7, "170.0.0.7", null);

ERROR 1099 (HY000): Table 'tbContainer' was locked with a READ lock and can't be updated

在mysql会话1中解锁表tbContainer

mysql> unlock tables;

Query OK, 0 rows affected (0.00 sec)

**四、MyISAM锁机制**

MyISAN引擎使用表锁

**读锁**

用法：LOCK TABLE table\_name [ AS alias\_name ] READ

释放锁使用UNLOCK tables.可以为表使用别名，如果一旦使用别名在使用的时候也必须采用别名。成功申请读锁的前提是当前没有线程对该表使用写锁，否则该语句会被阻塞。申请读锁成功后，其他线程也可以对该表进行读操作，但不允许有线程对其进行写操作，就算是当前线程也不允许。当锁住了A表之后，就只能对A表进行读操作，对其他表进行读操作会出现错误（tablename was not locked with LOCK TABLES）

**写锁**

用法： LOCK TABLE table\_name [AS alias\_name] [ LOW\_PRIORITY ] WRITE

同样也可以使用别名，与读锁不同的是，写锁中可以指定锁的优先级。LOW\_PRIORITY是一种比读锁更低优先级的锁,当多个线程同时申请多种锁（LOW\_PRIORITY,READ,WRITE）时，LOW\_PRIORITY的优先级最低。读锁申请成功的前提是没有线程对表加读锁和其他写锁，否则会被阻塞。

表级锁在MyISAM和innoDB中都有用到，创建锁的开销小，出现死锁概率很小，由于锁定的是整张表，所以并发度低。当需要频繁对大部分数据做 GROUP BY 操作或者需要频繁扫描整个表时，推荐使用表级锁。

**五、Innodb锁机制**

InnoDB同时支持行锁和表锁，默认采用行锁。

InnoDB的行锁实现的特点：只有通过索引条件检索操作记录，InnoDB才会使用行级锁，否则，InnoDB将会使用表锁。

InnoDB行级锁并不是直接锁记录，而是锁索引。如果一条SQL语句用到了主键索引，mysql会锁住主键索引；如果一条语句操作了非主键索引，mysql会先锁住非主键索引，再锁定主键索引。

为了允许行锁和表锁共存，实现多粒度锁机制，InnoDB还有两种内部使用的意向锁（意向共享锁和意向排他锁）。这两种意向锁都是表锁。意向锁是InnoDB自动加的，不需要用户干预。

所以Innodb有6种锁：4个对外的使用的，2个内部使用。

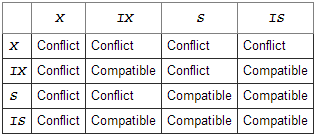
在行锁机制中有4个锁在起作用， 如下：

**共享锁(S):** 允许一个事务去读一行,阻止其他事务获得相同数据集的排他锁.

**排他锁(X):** 允许获得排他锁的事务更新数据,阻止其他事务获取相同数据集的各项读锁和排他写锁.

**意向共享锁(IS):** 事务打算给数据行加共享锁,事务在取得一个数据行的共享锁之前必须先取得该表的IS锁.

**意向排他锁(IX):** 事务打算给数据行加排他锁,事务在取得一个数据行的排他锁之前必须先取得该表的IX锁.



如果一个事务请求的锁模式与当前的锁兼容，InnoDB就将请求的锁授予该事务；反之，如果两者不兼容，该事务就要等待锁释放。

行级锁是Mysql中锁定粒度最细的一种锁，能大大减少数据库操作的冲突，由于其粒度小，加锁的开销最大。行级锁分为共享锁和排他锁。

**共享锁(S LOCK)**

用法：SELECT ...LOCK IN SHARE MODE;

Mysql会对查询结果中的每行都加共享锁，当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请共享锁，否则会被阻塞。其他线程也可以读取使用了共享锁的表，而且这些线程读取的是同一个版本的数据。

**排他锁(X LOCK)**

用法：SELECT ... FOR UPDATE;

Mysql会对查询结果中的每行都加排他锁，当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请排他锁，否则会被阻塞。

行级锁都是基于索引的，如果一条SQL语句用不到索引是不会使用行级锁的，会使用表级锁。行级锁的缺点是：由于需要请求大量的锁资源，所以速度慢，内存消耗大。

**意向锁（Intention Lock）介绍**

因为表锁覆盖了行锁的数据，所以表锁和行锁也会产生冲突。如：

（1）T1 BEGIN

（2）T1 给 Table1 加X锁，修改表结构

（3）T2 BEGIN

（4）T2 给 Table1 的一行记录加S或X锁（事务被阻塞，等待加锁成功）

T1要操作整个表，锁住了整个表。那么T2就不能再对Table1的单条记录加X或S锁，去读取或修这条记录。为了方便检测表级锁和行级锁之间的冲突，就引入了意向锁。

意向锁分为意向读锁（IS）和意向写锁（IX）

意向锁是表级锁，在给一行记录加锁前，首先要给该表加意向锁。也就是要同时加表意向锁和行锁。

意向锁之间不会冲突, 因为意向锁仅仅代表准备对某行记录进行操作。

采用了意向锁后，上面的例子就变成了：

（1）T1 BEGIN

（2）T1 给 Table1 加X锁，修改表结构

（3）T2 BEGIN

（4）T2 给 Table1 加IX锁（事务被阻塞，等待加锁成功，然后在给某一行记录加X锁。）

**锁的范围：**

inoodb除了上面锁概念还包括锁的范围概念,如下：

index record lock：

索引记录锁，InnoDB行级锁是通过给索引的索引项加锁来实现的，而不是对记录实例本身加锁。

如果表没有定义索引，InnoDB创建一个隐藏的聚簇索引，然后用它来实现记录加锁。

如果where检索固定索引： index record lock 只锁这个固定索引；

如果where检索范围索引： index record lock 锁这个范围索引；

gap lock：

间隙锁，不包括边界值。比如当前表索引字段值有 3、10、20， 间隙包括（-oo, 3）, (3, 10), (10, +oo)

sql语句where检索唯一索引的固定索引值，gap lock 是不起作用的。

间隙锁是可以显示关闭的，如果你将事务的隔离级别设为READ COMMITTED，或者打开innodb\_locks\_unsafe\_for\_binlog系统变量，间隙锁就会关闭。

在这种情况下，查找或扫描索引仅会进行外键约束检查和重复键值检查。

此外，READ COMMITTED隔离级别和关闭nodb\_locks\_unsafe\_for\_binlog还有另外一个负作用：MySQL会释放掉不匹配Where条件的记录锁。例如，对于UPDATE语句，

InnoDB只能进行“半一致性(semi\_consistent)读”，所以，它会返回最新提交事务所做改变，从而产生不可重复读和幻像问题。

next-key lock：

包括index record lock+index record lock之前的gap lock。比如当前表索引字段值有 3、10、20， next-key lock 包括（-oo, 3], (3, 10], (10, +oo)

例如：

（1）id列是唯一索引，SELECT \* FROM child WHERE id = 100; 只会对id=100的加索引记录锁(index-record lock)，而不会加之前的间隙索引。

如果上面查询是 SELECT \* FROM child WHERE id>100; 会对id>100的所有加索引记录锁(index-record lock)，也会加之前的间隙索引。

（2）id列是非唯一索引，则语句会加相应记录锁和之前的间隙锁。

**重复索引的问题：**

如果检索的索引值重复率太高，MYSQL 会无视了你的索引，它觉得与其行锁，还不如直接表锁（意向锁），毕竟它觉得表锁所花的代价比行锁来的小。即便你使用了 force index 强制索引，结果还是一样，永远都是表锁。详见<http://blog.csdn.net/ldb2741/article/details/5325161>（文中给出的例子，5条以上重复值会造成锁表的情况，但是本人在mysql 5.0.51a-log中测试，10条重复值才会出现锁表的情况）；另外经过测试，组合索引在重复率高的情况也会锁表，但是使用组合主键的一部分进行检索时，没有这样的问题（如i，j为组合主键，使用where i=100 进行检索，虽说有很多重复记录，但是使用的依然行锁）。

**申请锁规则：**

（1）select ：一致性读，不使用锁

select操作读取的是数据库快照（使用copy on write机制），同一事务内，多次读取的结果一致。

（2）update/delete： 使用exclusive next-key lock

被锁住的记录及记录前的间隙不能进行update、insert、select…lock in share mode、select … for update 等操作；注意使用唯一索引检索一条唯一的记录时，不会锁住记录前的间隙（参见next-key lock说明）。

（3）insert： 使用exclusive index-record lock

insert操作，只要多个事务不会在同一索引区间的同一个位置插入记录，它们就不用互相等待。例如，索引记录的值为4和7，两个独立的事务分别插入5和6，尽管它们都持有4—7之间的间隙锁，但是它们不会相互阻塞。这可以提高事务的并发性。

（4）select … for update：与update/delete操作使用的锁相同

使用场景: 如一个简单的给用户发送cdkey的系统，cdkey存于表t中，多个进程从表t中取cekey发给用户，如果不加锁的话，有可能会将一条cdkey发给多个用户(或者部分进程取cdkey失败)；这种场景下，可以使用select … for update对读取的cdkey记录加锁，在锁定期间将读取的cdkey状态置为“已使用”，这样其他进程就不会取到这条cdkey了。

（5）select ... from ... lock in share mode：使用 shared next-key lock

加上了这个锁之后，别的事务不能对索引记录和索引记录之前的区间进行修改和插入，也不能执行select…for update,但是可以进行select 和select ... from ... lock in share mode操作。使用场景：两张表:父亲t\_parent和孩子t\_child ,在t\_child中为parent A增加一个孩子A1时，若不进行加锁，则可能插入A1时，t\_parent中的A已经其他人删掉了，为了避免这种情况，可以使用select ... from ... lock in share mode将A锁住。

**六、mysql锁系统参数**

mysql> show variables like '%lock%';

+-----------------------------------------+------------+

| Variable\_name | Value |

+-----------------------------------------+------------+

| innodb\_api\_disable\_rowlock | OFF |

| innodb\_autoinc\_lock\_mode | 1 |

| innodb\_lock\_wait\_timeout | 50 | <---- Innodb行级锁超时时间

| innodb\_locks\_unsafe\_for\_binlog | OFF |

| innodb\_old\_blocks\_pct | 37 |

| innodb\_old\_blocks\_time | 1000 |

| innodb\_print\_all\_deadlocks | OFF |

| innodb\_table\_locks | ON |

| key\_cache\_block\_size | 1024 |

| lock\_wait\_timeout | 31536000 | <---- 表级锁超时时间

| locked\_in\_memory | OFF |

| max\_write\_lock\_count | 4294967295 |

+-----------------------------------------+------------+

21 rows in set (0.00 sec)

如果 session 锁等待超过 innodb\_lock\_wait\_timeout 设置的超时时间，在等待超时后会报错， 如下：

mysql> select \* from t for update;

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

**死锁测试1：**

**（1）在一个sql会话A执行如下：**

mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;

Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);

Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \* FROM t WHERE i = 1 LOCK IN SHARE MODE;

+------+

| i |

+------+

| 1 |

+------+

1 row in set (0.10 sec)

**（2）在另一个sql会话B执行如下：**

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1; <---- 由于会话A已经在事务中获取S锁， 会话B想获取X锁，被阻塞

**（3）在会话A执行如下：**

mysql> DELETE FROM t WHERE i = 1;

Query OK, 1 row affected (0.00 sec)

由于执行delete是获取X锁， 但是会话B之前已经尝试获取X锁，并且被阻塞中， 所以会话A也无法获取到X锁， 出现死锁。

结果Innodb为其中一个会话产生错误信息并且释放它等待的锁。

**（4）会话B原来还在挂着... 后面报错并且被innodb解锁**

mysql> DELETE FROM t WHERE i = 1;

ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction

上面这个例子参考：<http://dev.mysql.com/doc/refman/5.0/en/innodb-lock-modes.html>

**死锁测试2：**

在mysql会话A执行如下：

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql>

mysql> SELECT \* FROM t WHERE i = 1 for update;

+------+

| i |

+------+

| 1 |

+------+

1 row in set (0.01 sec)

mysql> delete from t where i=1; <----- 在本事务中多次获取X锁不会死锁挂死

Query OK, 1 row affected (0.00 sec)

mysql> select \* from t;

Empty set (0.00 sec)

**死锁测试3：**

在mysql会话A执行如下：

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \* FROM t WHERE i = 1 LOCK IN SHARE MODE;

+------+

| i |

+------+

| 1 |

+------+

1 row in set (0.00 sec)

mysql>

mysql> DELETE FROM t WHERE i = 1; <----- 在本事务中不会进行死锁检测

Query OK, 1 row affected (0.00 sec)

mysql> select \* from t;

Empty set (0.00 sec)

mysql> commit;

Query OK, 0 rows affected (0.01 sec)

**死锁测试4：**

在mysql会话A执行如下：

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql>

mysql> SELECT \* FROM t WHERE i = 1 LOCK IN SHARE MODE;

+------+

| i |

+------+

| 1 |

+------+

1 row in set (0.00 sec)

在mysql会话B执行如下：

mysql> SELECT \* FROM t WHERE i = 1 LOCK IN SHARE MODE; <----- 其他mysql会话获取S锁查询没有问题

+------+

| i |

+------+

| 1 |

+------+

1 row in set (0.00 sec)

在mysql会话A继续执行如下：

mysql> SELECT \* FROM t WHERE i = 1 for update; <------ 会话A 获取X锁

+------+

| i |

+------+

| 1 |

+------+

1 row in set (0.00 sec)

在mysql会话B执行如下：

mysql> SELECT \* FROM t WHERE i = 1 LOCK IN SHARE MODE; <------ 会话B 获取S锁挂住

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

在mysql会话A继续执行如下：

mysql> SELECT \* FROM t WHERE i = 1 LOCK IN SHARE MODE; <------ 会话A再次获取S锁挂住

+------+

| i |

+------+

| 1 |

+------+

1 row in set (0.00 sec)

在mysql会话B执行如下：

mysql> SELECT \* FROM t WHERE i = 1 LOCK IN SHARE MODE; <------ 会话B 获取S锁依然挂住

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

**七、参考**

<http://dev.mysql.com/doc/refman/5.0/en/innodb-lock-modes.html>

<http://dev.mysql.com/doc/refman/5.0/en/innodb-record-level-locks.html>

<http://dev.mysql.com/doc/refman/5.1/en/innodb-locks-set.html>