第 4 章 MySQL 锁机制

第 4 章 MySQL 锁机制

1、概述

1.1、锁的定义

> 锁的定义

1. 锁是计算机协调**多个进程或线程并发访问某一资源的机制**。
2. 在数据库中，除传统的计算资源（如CPU、RAM、I/O等）的争用以外，数据也是一种供许多用户共享的资源。
3. 如何保证数据并发访问的一致性、有效性是所有数据库必须解决的一个问题，锁冲突也是影响数据库并发访问性能的一个重要因素。
4. 从这个角度来说，锁对数据库而言显得尤其重要，也更加复杂。

1.2、锁的分类

> 锁的分类

1. 从数据操作的类型（读、写）分
    1. **读锁**（共享锁）：针对同一份数据，多个读操作可以同时进行而不会互相影响
    2. **写锁**（排它锁）：当前写操作没有完成前，它会阻断其他写锁和读锁。
2. 从对数据操作的颗粒度
    1. 表锁
    2. 行锁

2、表锁

> 表锁的特点

偏向MyISAM存储引擎，开销小，加锁快，无死锁，锁定粒度大，**发生锁冲突的概率最高，并发最低**

2.1、表锁案例

> 表锁案例分析

创建表

- 建表 SQL：**引擎选择 myisam**

```
create table mylock (
    id int not null primary key auto_increment,
    name varchar(20) default ''
) engine myisam;

insert into mylock(name) values('a');
insert into mylock(name) values('b');
insert into mylock(name) values('c');
insert into mylock(name) values('d');
insert into mylock(name) values('e');

select * from mylock;
```

- mylock 表中的测试数据

```
mysql> select * from mylock;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
|  4 | d    |
|  5 | e    |
+----+------+
5 rows in set (0.00 sec)
```

> 手动加锁和释放锁

- 查看当前数据库中表的上锁情况：`show open tables;`，0 表示未上锁

```
mysql> show open tables;
+--------------------+-----------------------------------------------------+--------+-------------+
| Database           | Table                                               | In_use | Name_locked |
+--------------------+-----------------------------------------------------+--------+-------------+
| performance_schema | events_waits_history                                |      0 |           0 |
| performance_schema | events_waits_summary_global_by_event_name           |      0 |           0 |
| performance_schema | setup_timers                                        |      0 |           0 |
| performance_schema | events_waits_history_long                           |      0 |           0 |
| performance_schema | events_statements_summary_by_digest                 |      0 |           0 |
| performance_schema | mutex_instances                                     |      0 |           0 |
| performance_schema | events_waits_summary_by_instance                    |      0 |           0 |
| performance_schema | events_stages_history                               |      0 |           0 |
| mysql              | db                                                  |      0 |           0 |
| performance_schema | events_waits_summary_by_host_by_event_name          |      0 |           0 |
| mysql              | user                                                |      0 |           0 |
| mysql              | columns_priv                                        |      0 |           0 |
| performance_schema | events_statements_history_long                      |      0 |           0 |
| performance_schema | performance_timers                                  |      0 |           0 |
| performance_schema | file_instances                                      |      0 |           0 |
| performance_schema | events_stages_summary_by_user_by_event_name         |      0 |           0 |
| performance_schema | events_stages_history_long                          |      0 |           0 |
| performance_schema | setup_actors                                        |      0 |           0 |
| performance_schema | cond_instances                                      |      0 |           0 |
| mysql              | proxies_priv                                        |      0 |           0 |
| performance_schema | socket_summary_by_instance                          |      0 |           0 |
| performance_schema | events_statements_current                           |      0 |           0 |
| mysql              | event                                               |      0 |           0 |
| performance_schema | session_connect_attrs                               |      0 |           0 |
| mysql              | plugin                                              |      0 |           0 |
| performance_schema | threads                                             |      0 |           0 |
| mysql              | time_zone_transition_type                           |      0 |           0 |
| mysql              | time_zone_name                                      |      0 |           0 |
| performance_schema | file_summary_by_event_name                          |      0 |           0 |
| performance_schema | events_waits_summary_by_user_by_event_name          |      0 |           0 |
| performance_schema | socket_summary_by_event_name                        |      0 |           0 |
| performance_schema | users                                               |      0 |           0 |
| mysql              | servers                                             |      0 |           0 |
| performance_schema | events_waits_summary_by_account_by_event_name       |      0 |           0 |
| db01               | tbl_emp                                             |      0 |           0 |
| performance_schema | events_statements_summary_by_host_by_event_name     |      0 |           0 |
| db01               | tblA                                                |      0 |           0 |
| performance_schema | table_io_waits_summary_by_index_usage               |      0 |           0 |
| performance_schema | events_waits_current                                |      0 |           0 |
| db01               | user                                                |      0 |           0 |
| mysql              | procs_priv                                          |      0 |           0 |
| performance_schema | events_statements_summary_by_thread_by_event_name   |      0 |           0 |
| db01               | emp                                                 |      0 |           0 |
| db01               | tbl_user                                            |      0 |           0 |
| db01               | test03                                              |      0 |           0 |
| mysql              | slow_log                                            |      0 |           0 |
| performance_schema | file_summary_by_instance                            |      0 |           0 |
| db01               | article                                             |      0 |           0 |
| performance_schema | objects_summary_global_by_type                      |      0 |           0 |
| db01               | phone                                               |      0 |           0 |
| performance_schema | events_waits_summary_by_thread_by_event_name        |      0 |           0 |
| performance_schema | setup_consumers                                     |      0 |           0 |
| performance_schema | socket_instances                                    |      0 |           0 |
| performance_schema | rwlock_instances                                    |      0 |           0 |
| db01               | tbl_dept                                            |      0 |           0 |
| performance_schema | events_statements_summary_by_user_by_event_name     |      0 |           0 |
| db01               | staffs                                              |      0 |           0 |
| db01               | class                                               |      0 |           0 |
| mysql              | general_log                                         |      0 |           0 |
| performance_schema | events_stages_summary_global_by_event_name          |      0 |           0 |
| performance_schema | events_stages_summary_by_account_by_event_name      |      0 |           0 |
| performance_schema | events_statements_summary_by_account_by_event_name  |      0 |           0 |
| performance_schema | table_lock_waits_summary_by_table                   |      0 |           0 |
| performance_schema | hosts                                               |      0 |           0 |
| performance_schema | setup_objects                                       |      0 |           0 |
| performance_schema | events_stages_current                               |      0 |           0 |
| mysql              | time_zone                                           |      0 |           0 |
| mysql              | tables_priv                                         |      0 |           0 |
| performance_schema | table_io_waits_summary_by_table                     |      0 |           0 |
| mysql              | time_zone_leap_second                               |      0 |           0 |
| db01               | book                                                |      0 |           0 |
| performance_schema | session_account_connect_attrs                       |      0 |           0 |
| db01               | mylock                                              |      0 |           0 |
| mysql              | func                                                |      0 |           0 |
| performance_schema | events_statements_summary_global_by_event_name      |      0 |           0 |
| performance_schema | events_statements_history                           |      0 |           0 |
| performance_schema | accounts                                            |      0 |           0 |
| mysql              | time_zone_transition                                |      0 |           0 |
| db01               | dept                                                |      0 |           0 |
| performance_schema | events_stages_summary_by_host_by_event_name         |      0 |           0 |
| performance_schema | events_stages_summary_by_thread_by_event_name       |      0 |           0 |
| mysql              | proc                                                |      0 |           0 |
| performance_schema | setup_instruments                                   |      0 |           0 |
| performance_schema | host_cache                                          |      0 |           0 |
+--------------------+-----------------------------------------------------+--------+-------------+
84 rows in set (0.00 sec)
```

- 添加锁

```
lock table 表名1 read(write), 表名2 read(write), ...;
```

- 释放表锁

```
unlock tables;
```

2.1.1、读锁示例

> 读锁示例

- 在 session 1 会话中，给 mylock 表加个读锁

```
mysql> lock table mylock read;
Query OK, 0 rows affected (0.00 sec)
```

- 在 session1 会话中能不能读取 mylock 表：可以读

```
################# session1 中的操作 #################

mysql> select * from mylock;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
|  4 | d    |
|  5 | e    |
+----+------+
5 rows in set (0.00 sec)
```

- 在 session1 会话中能不能读取 book 表：并不行。。。

```
################# session1 中的操作 #################

mysql> select * from book;
ERROR 1100 (HY000): Table 'book' was not locked with LOCK TABLES
```

- 在 session2 会话中能不能读取 mylock 表：可以读

```
################# session2 中的操作 #################

mysql> select * from mylock;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
|  4 | d    |
|  5 | e    |
+----+------+
5 rows in set (0.00 sec)
```

- 在 session1 会话中能不能修改 mylock 表：并不行。。。

```
################# session1 中的操作 #################

mysql> update mylock set name='a2' where id=1;
ERROR 1099 (HY000): Table 'mylock' was locked with a READ lock and can't be updated
```

- 在 session2 会话中能不能修改 mylock 表：阻塞，一旦 mylock 表锁释放，则会执行修改操作

```
################# session2 中的操作 #################

mysql> update mylock set name='a2' where id=1;
# 在这里阻塞着呢~~~
```

结论

1. 当前 session 和其他 session 均可以读取加了读锁的表
2. 当前 session 不能读取其他表，并且不能修改加了读锁的表
3. 其他 session 想要修改加了读锁的表，必须等待其读锁释放

2.1.2、写锁示例

> 写锁示例

- 在 session 1 会话中，给 mylock 表加个写锁

```
mysql> lock table mylock write;
Query OK, 0 rows affected (0.00 sec)
```

- 在 session1 会话中能不能读取 mylock 表：阔以

```
################ session1 中的操作 #################

mysql> select * from mylock;
+----+------+
| id | name |
+----+------+
|  1 | a2   |
|  2 | b    |
|  3 | c    |
|  4 | d    |
|  5 | e    |
+----+------+
5 rows in set (0.00 sec)
```

- 在 session1 会话中能不能读取 book 表：不阔以

```
################ session1 中的操作 #################

mysql> select * from book;
ERROR 1100 (HY000): Table 'book' was not locked with LOCK TABLES
```

- 在 session1 会话中能不能修改 mylock 表：当然可以啦，加写锁就是为了修改呀

```
################ session1 中的操作 #################

mysql> update mylock set name='a2' where id=1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

- 在 session2 会话中能不能读取 mylock 表：

```
################ session2 中的操作 #################

mysql> select * from mylock;
# 在这里阻塞着呢~~~
```

结论

1. 当前 session 可以读取和修改加了写锁的表
2. 当前 session 不能读取其他表
3. 其他 session 想要读取加了写锁的表，必须等待其读锁释放

> 案例结论

1. MyISAM在执行查询语句（SELECT）前，会自动给涉及的所有表加**读锁**，在执行增删改操作前，会自动给涉及的表加**写锁**。
2. MySQL的表级锁有两种模式：
    1. 表共享读锁（Table Read Lock）
    2. 表独占写锁（Table Write Lock）

锁类型	可否兼容	读锁	写锁
读锁	是	是	否
写锁	是	否	否

结论：结合上表，所以对MyISAM表进行操作，会有以下情况：

1. 对MyISAM表的读操作（加读锁），不会阻塞其他进程对同一表的读请求，但会阻塞对同一表的写请求。只有当读锁释放后，才会执行其它进程的写操作。
2. 对MyISAM表的写操作（加写锁），会阻塞其他进程对同一表的读和写操作，只有当写锁释放后，才会执行其它进程的读写操作
3. 简而言之，就是读锁会阻塞写，但是不会堵塞读。而写锁则会把读和写都堵塞。

2.2、表锁分析

> 表锁分析

- 查看哪些表被锁了，0 表示未锁，1 表示被锁

```
show open tables;
```

【如何分析表锁定】可以通过检查table_locks_waited和table_locks_immediate状态变量来分析系统上的表锁定，通过 `show status like 'table%';` 命令查看

1. Table_locks_immediate：产生表级锁定的次数，表示可以立即获取锁的查询次数，每立即获取锁值加1；
2. Table_locks_waited：出现表级锁定争用而发生等待的次数（不能立即获取锁的次数，每等待一次锁值加1），此值高则说明存在着较严重的表级锁争用情况；

```
mysql> show status like 'table%';
+----------------------------+--------+
| Variable_name              | Value  |
+----------------------------+--------+
| Table_locks_immediate      | 500440 |
| Table_locks_waited         | 1      |
| Table_open_cache_hits      | 500070 |
| Table_open_cache_misses    | 5      |
| Table_open_cache_overflows | 0      |
+----------------------------+--------+
5 rows in set (0.00 sec)
```

- 此外，Myisam的读写锁调度是写优先，这也是myisam不适合做写为主表的引擎。因为写锁后，其他线程不能做任何操作，**大量的更新会使查询很难得到锁**，从而造成永远阻塞

3、行锁

> 行锁的特点

1. **偏向InnoDB存储引擎**，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**。
2. InnoDB与MyISAM的最大不同有两点：**一是支持事务（TRANSACTION）；二是采用了行级锁**

3.1、事务复习

> 行锁支持事务，复习下老知识

事务（Transation）及其ACID属性

事务是由一组SQL语句组成的逻辑处理单元，事务具有以下4个属性，通常简称为事务的ACID属性。

2. **原子性**（Atomicity）：事务是一个原子操作单元，其对数据的修改，要么全都执行，要么全都不执行。
3. **一致性**（Consistent）：在事务开始和完成时，数据都必须保持一致状态。这意味着所有相关的数据规则都必须应用于事务的修改，以保持数据的完整性；事务结束时，所有的内部数都必须是正确的。
4. **隔离性**（Isolation）：数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的“独立”环境执行。这意味着事务处理过程中的中间状态对外部是不可见的，反之亦然。
5. **持久性**（Durability）：事务院成之后，它对于数据的修改是永久性的，即使出现系统故障也能够保持。

并发事务处理带来的问题

1. **更新丢失**（Lost Update）：
    1. 当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，由于每个事务都不知道其他事务的存在，就会发生丢失更新问题——最后的更新覆盖了由其他事务所做的更
    2. 例如，两个程序员修改同一java文件。每程序员独立地更改其副本，然后保存更改后的副本，这样就覆盖了原始文档。最后保存其更改副本的编辑人员覆盖前一个程序员所做的
    3. 如果在一个程序员完成并提交事务之前，另一个程序员不能访问同一文件，则可避免此问题。
2. **脏读**（Dirty Reads）：
    1. 一个事务正在对一条记录做修改，在这个事务完成并提交前，这条记录的数据就处于不一致状态；这时，另一个事务也来读取同一条记录，如果不加控制，第二个事务读取了这就会产生未提交的数据依赖关系。这种现象被形象地叫做”脏读”。
    2. 一句话：事务A读取到了事务B已修改但尚未提交的的数据，还在这个数据基础上做了操作。此时，如果B事务回滚，A读取的数据无效，不符合一致性要求。
3. **不可重复读**（Non-Repeatable Reads）：
    1. 一个事务在读取某些数据后的某个时间，再次读取以前读过的数据，却发现其读出的数据已经发生了改变、或某些记录已经被删除了！这种现象就叫做“不可重复读”。
    2. 一句话：事务A读取到了事务B已经提交的修改数据，不符合隔离性
4. **幻读**（Phantom Reads）
    1. 一个事务按相同的查询条件重新读取以前检索过的数据，却发现其他事务插入了满足其查询条件的新数据，这种现象就称为“幻读一句话：事务A读取到了事务B体提交的新增数
    2. 多说一句：幻读和脏读有点类似，脏读是事务B里面修改了数据，幻读是事务B里面新增了数据。

事物的隔离级别

1. 脏读”、“不可重复读”和“幻读”，其实都是数据库读一致性问题，必须由数据库提供一定的事务隔离机制来解决。

2. 数据库的事务隔离越严格，并发副作用越小，但付出的代价也就越大，因为事务隔离实质上就是使事务在一定程度上“串行化”进行，这显然与“并发”是矛盾的。

3. 同时，不同的应用对读一致性和事务隔离程度的要求也是不同的，比如许多应用对“不可重复读”和“幻读”并不敏感，可能更关心数据并发访问的能力。

4. 查看当前数据库的事务隔离级别： `show variables like 'tx_isolation';` mysql 默认是可重复读

读数据一致性及允许的并发副作用 隔离级别	读数据一致性	脏读	不可重复读	幻读
未提交读（Read uncommitted）	**最低级别，只能保证不读取物理上损坏的数据**	**是**	**是**	**是**
已提交度（Read committed）	**语句级**	**否**	**是**	**是**
可重复读（Repeatable read）	**事务级**	**否**	**否**	**是**
可序列化（Serializable）	**最高级别，事务级**	**否**	**否**	**否**

Oneby's Blog

3.2、行锁案例

> 行锁案例分析

创建表

- 建表 SQL

```
CREATE TABLE test_innodb_lock (a INT(11),b VARCHAR(16))ENGINE=INNODB;

INSERT INTO test_innodb_lock VALUES(1,'b2');
INSERT INTO test_innodb_lock VALUES(3,'3');
INSERT INTO test_innodb_lock VALUES(4, '4000');
INSERT INTO test_innodb_lock VALUES(5,'5000');
INSERT INTO test_innodb_lock VALUES(6, '6000');
INSERT INTO test_innodb_lock VALUES(7,'7000');
INSERT INTO test_innodb_lock VALUES(8, '8000');
INSERT INTO test_innodb_lock VALUES(9,'9000');
INSERT INTO test_innodb_lock VALUES(1,'b1');

CREATE INDEX test_innodb_a_ind ON test_innodb_lock(a);
CREATE INDEX test_innodb_lock_b_ind ON test_innodb_lock(b);
```

- test_innodb_lock 表中的测试数据

```
mysql> select * from test_innodb_lock;
+------+------+
| a    | b    |
+------+------+
|    1 | b2   |
|    3 | 3    |
|    4 | 4000 |
|    5 | 5000 |
|    6 | 6000 |
|    7 | 7000 |
|    8 | 8000 |
|    9 | 9000 |
|    1 | b1   |
+------+------+
9 rows in set (0.00 sec)
```

- test_innodb_lock 表中的索引

```
mysql> SHOW INDEX FROM test_innodb_lock;
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+---------
| Table           | Non_unique | Key_name              | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_ty
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+---------
| test_innodb_lock |          1 | test_innodb_a_ind     |            1 | a           | A         |           9 |     NULL | NULL   | YES  | BTREE
| test_innodb_lock |          1 | test_innodb_lock_b_ind |            1 | b           | A         |           9 |     NULL | NULL   | YES  | BTREE
+-----------------+------------+-----------------------+--------------+-------------+-----------+-------------+----------+--------+------+---------
2 rows in set (0.00 sec)
```

> 操作同一行数据

- session1 开启事务，修改 test_innodb_lock 中的数据

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='4001' where a=4;

```

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

- session2 开启事务，修改 test_innodb_lock 中同一行数据，将导致 session2 发生阻塞，一旦 session1 提交事务，session2 将执行更新操作

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='4002' where a=4;
# 在这儿阻塞着呢~~~

# 时间太长，会报超时错误哦
mysql> update test_innodb_lock set b='4001' where a=4;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

> 操作不同行数据

- session1 开启事务，修改 test_innodb_lock 中的数据

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='4001' where a=4;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

- session2 开启事务，修改 test_innodb_lock 中不同行的数据
- 由于采用行锁，session2 和 session1 互不干涉，所以 session2 中的修改操作没有阻塞

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='9001' where a=9;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

> 无索引导致行锁升级为表锁

- session1 开启事务，修改 test_innodb_lock 中的数据，varchar 不用 ' '，导致系统自动转换类型，导致索引失效

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set a=44 where b=4000;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

- session2 开启事务，修改 test_innodb_lock 中不同行的数据
- 由于发生了自动类型转换，索引失效，导致行锁变为表锁

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='9001' where a=9;
# 在这儿阻塞着呢~~~
```

3.3、间隙锁

> 什么是间隙锁

1. 当我们用范围条件而不是相等条件检索数据，并请求共享或排他锁时，InnoDB会给符合条件的已有数据记录的索引项加锁；对于键值在条件范围内但并不存在的记录，叫做“间隙（G
2. InnoDB也会对这个“间隙”加锁，这种锁机制是所谓的间隙锁（Next-Key锁）

> 间隙锁的危害

1. 因为Query执行过程中通过过范围查找的话，他会锁定整个范围内所有的索引键值，即使这个键值并不存在。
2. 间隙锁有一个比较致命的弱点，就是当锁定一个范围键值之后，即使某些不存在的键值也会被无辜的锁定，而造成在锁定的时候无法插入锁定键值范围内的任何数据。在某些场景下这

间隙锁示例

- test_innodb_lock 表中的数据

```
mysql> select * from test_innodb_lock;
+------+------+
| a    | b    |
+------+------+
|    1 | b2   |
|    3 | 3    |
|    4 | 4000 |
|    5 | 5000 |
|    6 | 6000 |
|    7 | 7000 |
|    8 | 8000 |
|    9 | 9000 |
|    1 | b1   |
+------+------+
9 rows in set (0.00 sec)
```

- session1 开启事务，执行修改 a > 1 and a < 6 的数据，这会导致 mysql 将 a = 2 的数据行锁住（虽然表中并没有这行数据）

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='Heygo' where a>1 and a<6;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

- session2 开启事务，修改 test_innodb_lock 中不同行的数据，也会导致阻塞，直至 session1 提交事务

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='9001' where a=9;
# 在这儿阻塞着呢~~~
```

3.4、手动行锁

> 如何锁定一行

- `select xxx ... for update` 锁定某一行后，其它的操作会被阻塞，直到锁定行的会话提交
- session1 开启事务，手动执行 for update 锁定指定行，待执行完指定操作时再将数据提交

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from test_innodb_lock  where a=8 for update;
+------+------+
| a    | b    |
+------+------+
|    8 | 8000 |
+------+------+
1 row in set (0.00 sec)
```

- session2 开启事务，修改 session1 中被锁定的行，会导致阻塞，直至 session1 提交事务

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update test_innodb_lock set b='XXX' where a=8;
# 在这儿阻塞着呢~~~
```

3.5、行锁分析

> 案例结论

1. Innodb存储引擎由于实现了行级锁定，虽然在锁定机制的实现方面所带来的性能损耗可能比表级锁定会要更高一些，但是在整体并发处理能力方面要远远优于MyISAM的表级锁定的。
2. **当系统并发量较高的时候，Innodb的整体性能和MyISAM相比就会有比较明显的优势了。**
3. 但是，Innodb的行级锁定同样也有其脆弱的一面，当我们使用不当的时候（索引失效，导致行锁变表锁），可能会让Innodb的整体性能表现不仅不能比MyISAM高，甚至可能会更差。

> 行锁分析

如何分析行锁定

- 通过检查InnoDB_row_lock状态变量来分析系统上的行锁的争夺情况

```
show status like 'innodb_row_lock%';
```

```
mysql> show status like 'innodb_row_lock%';
+-------------------------------+--------+
| Variable_name                 | Value  |
+-------------------------------+--------+
| Innodb_row_lock_current_waits | 0      |
| Innodb_row_lock_time          | 212969 |
| Innodb_row_lock_time_avg      | 42593  |
| Innodb_row_lock_time_max      | 51034  |
| Innodb_row_lock_waits         | 5      |
+-------------------------------+--------+
5 rows in set (0.00 sec)
```

对各个状态量的说明如下：

1. Innodb_row_lock_current_waits：当前正在等待锁定的数量；
2. Innodb_row_lock_time：从系统启动到现在锁定总时间长度；
3. Innodb_row_lock_time_avg：每次等待所花平均时间；
4. Innodb_row_lock_time_max：从系统启动到现在等待最常的一次所花的时间；
5. Innodb_row_lock_waits：系统启动后到现在总共等待的次数；

对于这5个状态变量，比较重要的主要是

1. Innodb_row_lock_time_avg（等待平均时长）
2. Innodb_row_lock_waits（等待总次数）
3. Innodb_row_lock_time（等待总时长）

尤其是当等待次数很高，而且每次等待时长也不小的时候，我们就需要分析系统中为什么会有如此多的等待，然后根据分析结果着手指定优化计划。

3.6、行锁优化

> **优化建议**

1. 尽可能让所有数据检索都通过索引来完成，避免无索引行锁升级为表锁
2. 合理设计索引，尽量缩小锁的范围
3. 尽可能较少检索条件，避免间隙锁
4. 尽量控制事务大小，减少锁定资源量和时间长度
5. 尽可能低级别事务隔离

4、页锁

1. 开销和加锁时间界于表锁和行锁之间：会出现死锁；
2. 锁定粒度界于表锁和行锁之间，并发度一般。
3. 了解即可