链滴

MySQL 技术内幕 | 锁篇

作者：douniwan

原文链接：https://ld246.com/article/1582314859728

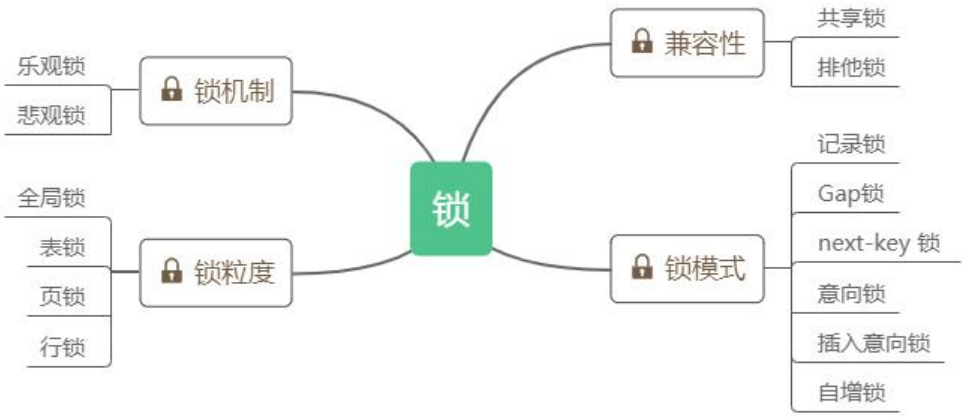
来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

事务相关内容：MySQL技术内幕 | 事务篇

锁的类型

在MySQL中锁按不同标准划分有如下类型：



1.乐观锁与悲观锁是**两种并发控制的思想**，可用于**解决丢失更新**问题：

- 乐观锁会“乐观地”假定大概率不会发生并发更新冲突，访问、处理数据过程中不加锁，只在更新据时再根据版本号或时间戳判断是否有冲突，有则处理，无则提交事务。
- 悲观锁会“悲观地”假定大概率会发生并发更新冲突，访问、处理数据前就加排他锁，在整个数据理过程中锁定数据，事务提交或回滚后才释放锁。

2.MySQL支持多种锁粒度：

- 全局锁就是对整个数据库实例加锁，所有存储引擎都可以用命令实现。

原文链接：MySQL 技术内幕 | 锁篇

- 表级锁定的主要是MyISAM，MEMORY，CSV等一些非事务性存储引擎。
- 使用页级锁定的主要是BerkeleyDB存储引擎。
- 行级锁定的主要是InnoDB存储引擎。

InnoDB**支持多种锁粒度,默认使用行锁，锁粒度最小**，锁冲突发生的概率最低，支持的并发度也最高但系统消耗成本也相对较高。

3.共享锁与排他锁是**InnoDB**实现的两种标准的**行锁**。

- 共享锁（S锁）:共享 (S) 用于不更改或不更新数据的操作（只读操作），如 SELECT 语句。

如果事务T对数据A加上共享锁后，则其他事务只能对A再加共享锁，不能加排他锁。获准共享锁的事只能读数据，不能修改数据。

- 排他锁（X锁）：用于数据修改操作，例如 INSERT、UPDATE 或 DELETE。确保不会同时同一资进行多重更新。

如果事务T对数据A加上排他锁后，则其他事务不能再对A加任任何类型的封锁。获准排他锁的事务既读数据，又能修改数据。

4.InnoDB有三种锁算法——**记录锁(Record Locks)**、**间隙锁(Gap Locks)**、还有结合了记录锁与间锁的**临键锁(Next-key Locks)**，InnoDB对于行的查询加锁是使用的是next-key locking这种算法，定程度上解决了幻读问题。(**这三种都是行级锁**)

5.**意向锁(Intention Locks)** 存在的意义在于，使得行锁和表锁能够共存。

- 意向锁是表级别的锁，用来说明事务稍后会对表中的数据行加哪种类型的锁(共享锁或排他锁)。
- 当一个事务对表加了意向排他锁时，另外一个事务在加锁前就会通过该表的意向排他锁知道前面已有事务在对该表进行独占操作，从而等待。

6.**插入意向锁(Insert Intention Locks)** 是在数据行插入之前通过插入操作设置的间隙锁定类型。

- 如果多个事务插入到相同的索引间隙中，如果它们不在间隙中的相同位置插入，则无需等待其他事。例如：在4和7的索引间隙之间两个事务分别插入5和6，则两个事务不会发冲突阻塞。

7.**自增锁(Auto-inc Locks)** 是事务插入到有自增列的表中而获得的一种特殊的表级锁。如果一个事正在向表中插入值，那么任何其他事务都必须等待，保证第一个事务插入的行是连续的自增值。

全局锁

实现原理

让整个库处于只读状态，之后其他线程的以下语句会被阻塞：数据更新语句（数据的增删改）、数据义语句（包括建表、修改表结构等）和更新类事务的提交语句。

MySQL 提供了一个加全局读锁的方法(FTWRL)，命令是Flush tables with read lock 。

使用场景

全局锁的典型使用场景是，做全库逻辑备份（mysqldump）。重新做主从时候，也就是把整库每个都 select 出来存成文本。

有一种做法，是通过 FTWRL 确保不会有其他线程对数据库做更新，然后对整个库做备份。注意，在份过程中整个库完全处于只读状态。

为什么需要全局读锁(FTWRL)

官方自带的逻辑备份工具是 mysqldump。当 mysqldump 使用参数--single-transaction的时候，数据之前就会启动一个事务，来确保拿到一致性快照视图。而由于 MVCC 的支持，这个过程中数据可以正常更新的。为什么还需要 FTWRL 呢?

一致性读是好，但前提是引擎要支持这个隔离级别。比如，对于 MyISAM 这种不支持事务的引擎，果备份过程中有更新，总是只能取到最新的数据，那么就破坏了备份的一致性。这时，我们就需要使 FTWRL 命令了。

所以，single-transaction 方法只适用于所有的表使用事务引擎的库。如果有的表使用了不支持事务引擎，那么备份就只能通过 FTWRL 方法。这往往是 DBA 要求业务开发人员使用 InnoDB 替代 MyISM 的原因之一。

为什么不是set global readonly=true?

既然要全库只读，为什么不使用 set global readonly=true 的方式呢？确实 readonly 方式也可以让库进入只读状态，但还是建议使用用 FTWRL 方式，主要有几个原因：

- 在有些系统中，readonly 的值会被用来做其他逻辑，比如用来判断一个库是主库还是备库。因此修改 global 变量的方式影响面更大，我不建议你使用。
- 在异常处理机制上有差异。如果执行FTWRL 命令之后由于客户端发生异常断开，那么 MySQL 会动释放这个全局锁，整个库回到可以正常更新的状态。而将整个库设置为 readonly 之后，如果客户发生异常，则数据库就会一直保持 readonly 状态，这样会导致整个库长时间处于不可写状态，风险高。
- readonly 对super用户权限无效

全局锁的危害性

如果你在主库上备份，那么在备份期间都不能执行更新，业务基本上就能停止。如果你在从库上备份那么备份期间从库不能执行主库同步过来的binlog，会导致主从延迟。

表级别锁

MySQL 里面**表级别**的锁有两种：一种是**表锁**，一种是**元数据锁（meta data lock，MDL)**

实现原理

当前线程对表进行操作的时候进行加读/写锁，从而限制**别的线程的读/写**，也限定**当前线程的操作对象**。

表锁

MyISAM存储引擎使用的锁机制（MyISAM支持表锁，不支持行锁。）

加/解锁

- 加锁：lock table xxx read/write
- 解锁：unlock tables（客户端断开的时候也会自动释放）

当会话将表加上锁后，在锁释放之前，会话只能访问这些加锁的表（**即限定了当前线程的操作对象**）

加锁规则

- 读锁：
  - 持有读锁的会话可以读表，但不能写表
  - 允许多个会话同时持有读锁，其他会话就算没有给表加读锁，也是可以读表的，但是不能写表申请写锁时会阻塞
- 写锁：
  - 持有写锁的会话既可以读表，也可以写表
  - 只有持有写锁的会话才可以访问该表，其他会话访问该表会被阻塞，申请读锁或写锁时都会阻塞

释放规则

- unlock tables释放
- 会话在持有表锁的情况下执行lock table语句，会释放掉之前持有的锁
- 会话在持有表锁的情况下执行start transaction或者begin开启一个新事务，会释放掉之前持有的锁
- 会话连接断开，会释放掉之前持有的锁

MDL锁(metadata lock)

MDL 不需要显式使用，在访问一个表的时候会被自动加上。MDL 的作用是，保证读写的正确性，假一个查询正在遍历一个表中的数据，而执行期间另一个线程对这个表结构做变更，删了一列，那么查线程拿到的结果跟表结构对不上，这显然是有问题的。所以MySQL在5.5引入了MDL来进行控制。

加/解锁

- 加锁：在语句执行开始时申请
- 解锁：语句结束后并不会马上释放，而会等到整个事务提交后再释放

加锁规则

- 读锁：对一个表做增删改查操作的时候，加 MDL读锁
- 写锁：当要对表做结构变更操作的时候，加 MDL 写锁

读锁之间不互斥，因此你可以有多个线程同时对一张表增删改查。

读写锁之间、写锁之间是互斥的，用来保证变更表结构操作的安全性。因此，如果有两个线程要同时一个表加字段，其中一个要等另一个执行完才能开始执行

查看表级锁争用情况

使用命令show status like 'table%';

```
mysql> show status like 'table%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Table_locks_immediate      | 99    |
| Table_locks_waited         | 0     |
| Table_open_cache_hits      | 11    |
| Table_open_cache_misses    | 2     |
| Table_open_cache_overflows | 0     |
+----------------------------+-------+
5 rows in set (0.05 sec)
```

Table_locks_immediate：产生表级锁定的次数；

Table_locks_waited：出现表级锁定争用而发生等待的次数；

如果Table_locks_waited状态值比较高，那么说明系统中表级锁定争用现象比较严重，就需要进一步析为什么会有较多的锁定资源争用了。

优先级

MyISAM存储引擎默认是**写优先级大于读优先级**。即使是写请求后到，写锁也会插到读锁请求之前。

但是，有时像修改文章点击数 操作是不那么重要的，我们希望的是读更快，此时我们可以这样：

UPDATE LOW_PRIORITY article SET click_num=666 WHERE id = 888

LOW_PRIORITY使得系统认为update操作优化级比读操作低，如果同时出现读操作和上面的更新操，则优先执行读操作

MySQL提供了几个语句调节符，允许你修改它的调度策略：

- LOW_PRIORITY关键字应用于：DELETE、INSERT、LOAD DATA、REPLACE和UPDATE。
- HIGH_PRIORITY关键字应用于：SELECT、INSERT语句。

* delayed(延迟)关键字应用于：INSERT、REPLACE语句。

更多内容参考:MySQL的MDL

注意事项

- InnoDB加表锁

在用 LOCK TABLES对InnoDB表加锁时要注意，要将AUTOCOMMIT设为0，否则MySQL不会给表锁；事务结束前，不要用UNLOCK TABLES释放表锁，因为UNLOCK TABLES会隐含地提交事务；COMMIT或ROLLBACK并不能释放用LOCK TABLES加的表级锁，必须用UNLOCK TABLES释放表锁。

行锁

每次锁定的是一行数据，这很好理解，比如事务 A 更新了一行，而这时候事务 B 也要更新同一行，必须等事务 A 的操作完成后才能进行更新。行级锁定不是MySQL自己实现的锁定方式，而是由其他储引擎自己所实现的

实现原理

1.InnoDB行锁是通过给索引上的**索引项**加锁来实现的。所以，**只有通过索引**条件**检索数据**，InnoDB**使用行级锁**，否则，InnoDB将使用表锁。

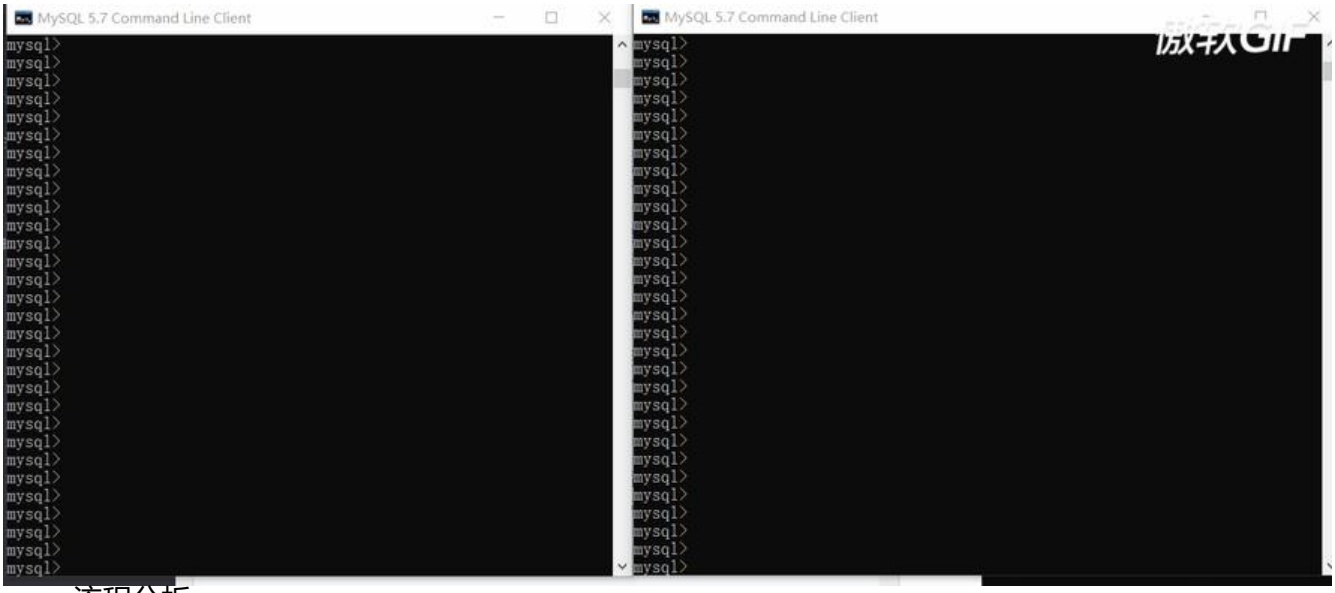
2.由于MySQL的行锁是针对索引加的锁，不是针对记录加的锁，所以虽然是访问不同行的记录，但是果是**使用相同的索引键**，是**会出现锁冲突**的。应用设计的时候要注意这一点。

- 这里需要重点解释的是，这条原理中指的这句 **使用相同的索引键**中的**索引**，**不是唯一索引**(唯一引也满足不了这个场景)。知道这个前提后，改这句话修改为**使用相同的索引键,并且查询的键值一样**在这种情况下才会产生冲突。
- 代码示例如下。
  - 现在建有如下表：

id是辅助索引，主键索引为系统创建的隐式索引，并插入图中数据：

```
mysql> create table tt (id int(11) not null,age int(11) not null) ENGINE=InnoDB;
Query OK, 0 rows affected (0.06 sec)

mysql> alter table tt add index id(id);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> insert into tt select 1,20;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> insert into tt select 1,21;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> insert into tt select 2,21;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> insert into tt select 2,20;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

  - 操作动画如下：可以先看看流程推测一下为什么右边的事务中的 id =1查询要等左边的事务commit后，才能查出来。



  - 流程分析

- 首先左边事务中执行 **id=1and age =20**的查询（加上了排他锁）
    - 然后右边事务中先执行 **id=2 and age =21**的查询，加上排他锁，这时可以正常查询。
    - 接着右边事务中再执行 **id=1 and age =21**的查询，加上排他锁，这时就不能查询出，一处于等待中。
    - 然后左边事务commit后，右边事务 **id=1 and age =21**的查询返回结果。

之所以会出现这个结果就是因为左边事务的查询** id=1**用到索引，而事务右边也用到了id这个索引查询，而**id=2**可以正常执行，但是** id=1**就不可以了，这就是上面原理2所表述的那种情况。而左边事务提交后（释放锁），右边事务返回结果更加验证了这一原理。

3.当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，另外，不论是使用主键索、唯一索引或普通索引，InnoDB都会使用行锁来对数据加锁。

4.即便在条件中使用了索引字段，但是否使用索引来检索数据是由MySQL通过判断不同执行计划的代来决定的，如果MySQL认为全表扫描效率更高，比如对一些很小的表，它就不会使用索引，这种情下InnoDB将使用表锁，而不是行锁。因此，在分析锁冲突时，别忘了检查SQL的执行计划，以确认否真正使用了索引

行锁类型

InnoDB的行级锁定分为两种类型：**共享锁**和**排他锁**，而在锁定机制的实现过程中为了让行级锁定和级锁定**共存**，InnoDB也同样使用了**意向锁（表级锁定）** 的概念，也就有了**意向共享锁**和**意向排他锁**两种。

意向锁的作用就是当一个事务在需要获取资源锁定的时候，如果遇到自己需要的资源已经被排他锁占的时候，该事务可以需要锁定行的表上面添加一个合适的意向锁。如果自己需要一个共享锁，那么就表上面添加一个意向共享锁。而如果自己需要的是某行（或者某些行）上面添加一个排他锁的话，则在表上面添加一个意向排他锁。

意向共享锁可以同时并存多个，但是意向排他锁同时只能有一个存在。所以，可以说InnoDB的锁定式实际上可以分为四种：

- 共享锁（S）
- 排他锁（X）
- 意向共享锁（IS）
- 意向排他锁（IX）

锁的兼容性

	共享锁（S）	**排他锁（X**	**意向共享锁（IS**	**向排他锁（IX）**
共享锁（S）	兼容	冲突	兼容	突
排他锁（X）	冲突	冲突	兼容	突
意向共享锁（IS）	兼容	冲突	冲突	容
意向排他锁（IX）	冲突	冲突	兼容	

容

加锁

隐式加锁

- InnoDB自动加意向锁。
- 对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加 排他锁（X)。
- 对于普通SELECT语句，InnoDB不会加任何锁。

显式加锁

- 共享锁（S）： SELECT * FROM table_name WHERE ... LOCK IN SHARE MODE
- 排他锁（X)： SELECT * FROM table_name WHERE ... FOR UPDATE

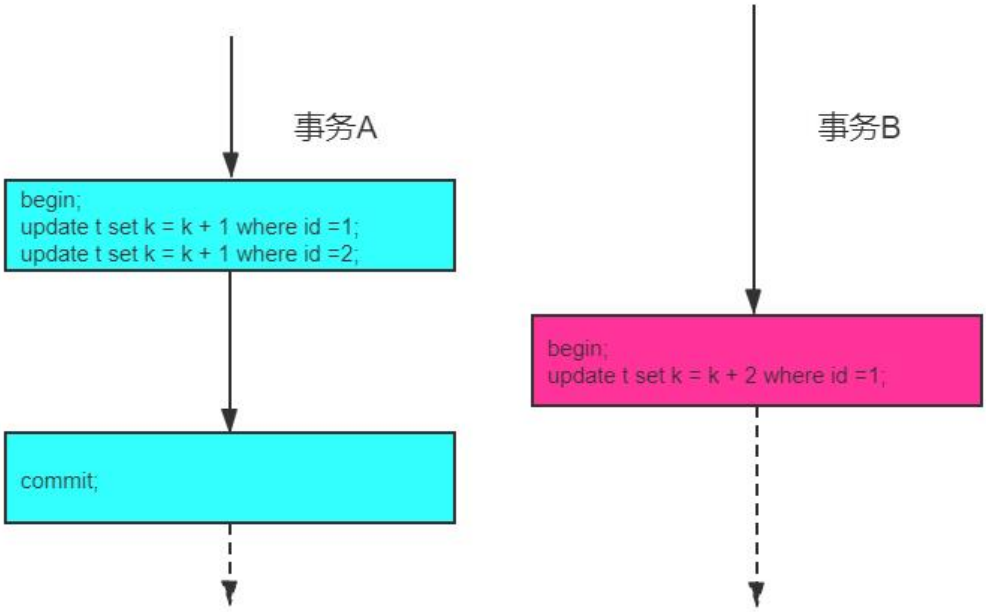
用SELECT ... IN SHARE MODE获得共享锁，主要用在需要数据依存关系时来确认某行记录是否存在并确保没有人对这个记录进行UPDATE或者DELETE操作。但是如果当前事务也需要对该记录进行更操作，则很有可能造成死锁，对于锁定行记录后需要进行更新操作的应用，应该使用SELECT... FOR UDATE方式获得排他锁。

释放锁

在 InnoDB 事务中，行锁是在需要的时候才加上的，但并不是不需要了就立刻释放，而是要等到事务束时才释放。

两阶段锁协议

先举个例子，假设有一个表 t，主键是 id，其中一个字段是 k，在下面的操作中，事务 B 的 update 句执行时，会是什么现象呢？



原文链接：MySQL 技术内幕 | 锁篇

这个问题的结论取决于事务 A 执行完前两条语句后，持有哪些锁，以及在什么时候释放。实际上，事 A 持有两个记录的行锁，都是在 commit 的时候才释放的，所以事务 B 的 update 就会被阻塞，直事务 A 执行 commit 之后，事务 B 才能被继续执行。

也就是说，在 InnoDB 事务中，行锁是在需要的时候才加上的，但并不是不需要了就立刻释放，需要事务结束时才释放，这就是**两阶段锁协议，分为加锁阶段和解锁阶段，所有的 lock 操作都在 unlock 操作之后。**

两阶段锁协议的运用

知道两阶段锁协议后，我们就可以，来避免一些并发操作中的锁。如果你的事务中需要锁多个行，要最可能造成锁冲突、最可能影响并发度的锁尽量往后放。

举个例子：

假设你负责实现一个电影票在线交易业务，顾客 A 要在影院 B 购买电影票。我们简化一点，这个业需要涉及到以下操作：

1. 从顾客 A 账户余额中扣除电影票价；
2. 给影院 B 的账户余额增加这张电影票价；
3. 记录一条交易日志。

也就是说，要完成这个交易，我们需要 update 两条记录，并 insert 一条记录。当然，为了保证交易原子性，我们要把这三个操作放在一个事务中。那么，你会怎样安排这三个语句在事务中的顺序呢？

分析解决：

试想如果同时有另外一个顾客 C 要在影院 B 买票，那么这两个事务冲突的部分就是语句 2了。因为它要更新同一个影院账户的余额，需要修改同一行数据。根据两阶段锁协议，不论你怎样安排语句顺序所有的操作需要的行锁都是在事务提交的时候才释放的。所以，如果你把语句 2 安排在最后，比如按 3、1、2 这样的顺序，那么影院账户余额这一行的锁时间就最少。这就最大程度地减少了事务之间的等待，提升了并发度。

查看行级锁争用情况

使用命令：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          | 0     |
| Innodb_row_lock_time_avg      | 0     |
| Innodb_row_lock_time_max      | 0     |
| Innodb_row_lock_waits         | 0     |
+-------------------------------+-------+
5 rows in set (0.01 sec)
```

Innodb_row_lock_current_waits：当前正在等待锁的数量；

Innodb_row_lock_time：从系统启动到现在锁定总时间长度；

Innodb_row_lock_time_avg：每次等待所花平均时间；

Innodb_row_lock_time_max：从系统启动到现在等待最长的一次所花的时间长度；

Innodb_row_lock_waits：系统启动到现在总共等待的次数；

对于这5个状态变量，比较重要的是：

Innodb_row_lock_time_avg，Innodb_row_lock_waits，Innodb_row_lock_time。

尤其是当等待次数很高，而且每次等待时长也很大的时候，我们就要分析系统中为什么有这么多的等，然后根据分析结果来制定优化。

利用InnoDB Monitors分析锁争用的原因

设置监视器：create table InnoDB_monitor(a INT) engine=InnoDB;

查看： show engine InnoDB status;

停止查看：drop table InnoDB_monitor;

分析详情参考：InnoDB Monitor

三种行级锁算法

InnoDB的三种行级锁算法包括记录锁(Record Lock)，间隙锁(Gap Lock)，临键锁(Next-key Locks)因为都是行级锁，所以他们都是对索引加锁，只不过加锁的范围不同。

记录锁(Record Lock)

顾名思义，记录锁就是为**某行**记录加锁，它封锁该行的索引记录。他的语法规则跟排他锁无差别。也是ELECT * FROM table_name WHERE ... FOR UPDATE，那么如何去区别是记录锁还是排他锁呢？既两者语法一样，那么会不是where中的查询键值的类型不一样，从而导致变成不一样的锁了？如果你想到这理，那你已经成功一半了。事实上在记录锁中需要保证：

- where后面的条件列为 **为主键列或唯一索引列**
- 查询语句必须为 精准匹配（=），不能为 >、<、like等

以上两点有任意一点不满足都会退化成**临键锁**(见后文介绍)。

---TBD---

本文参考

极客时间·MySQL实战45讲

一分钟深入Mysql的意向锁——《深究Mysql锁》

五分钟了解Mysql的行级锁——《深究Mysql锁》