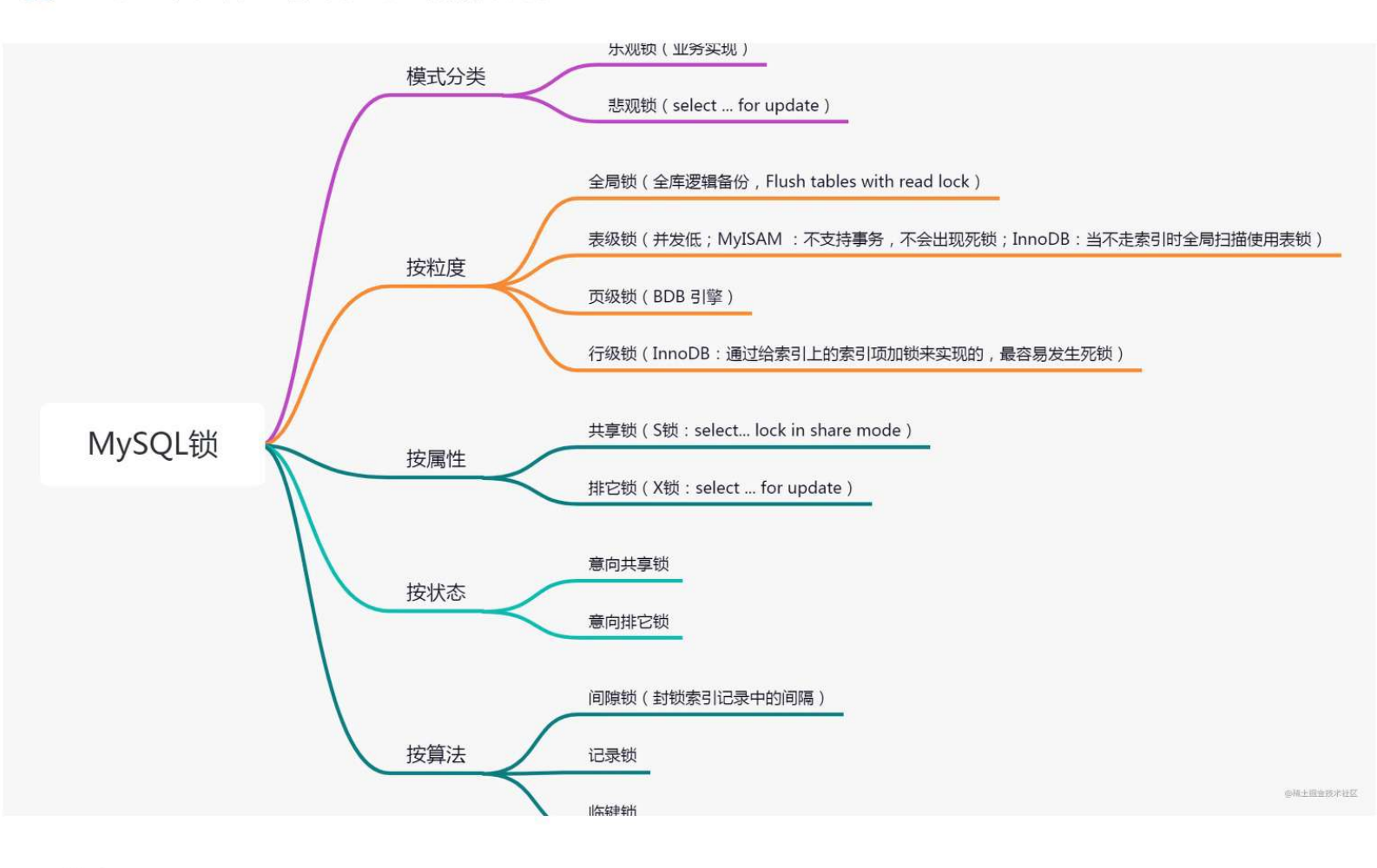
史上最全MySQL各种锁详解

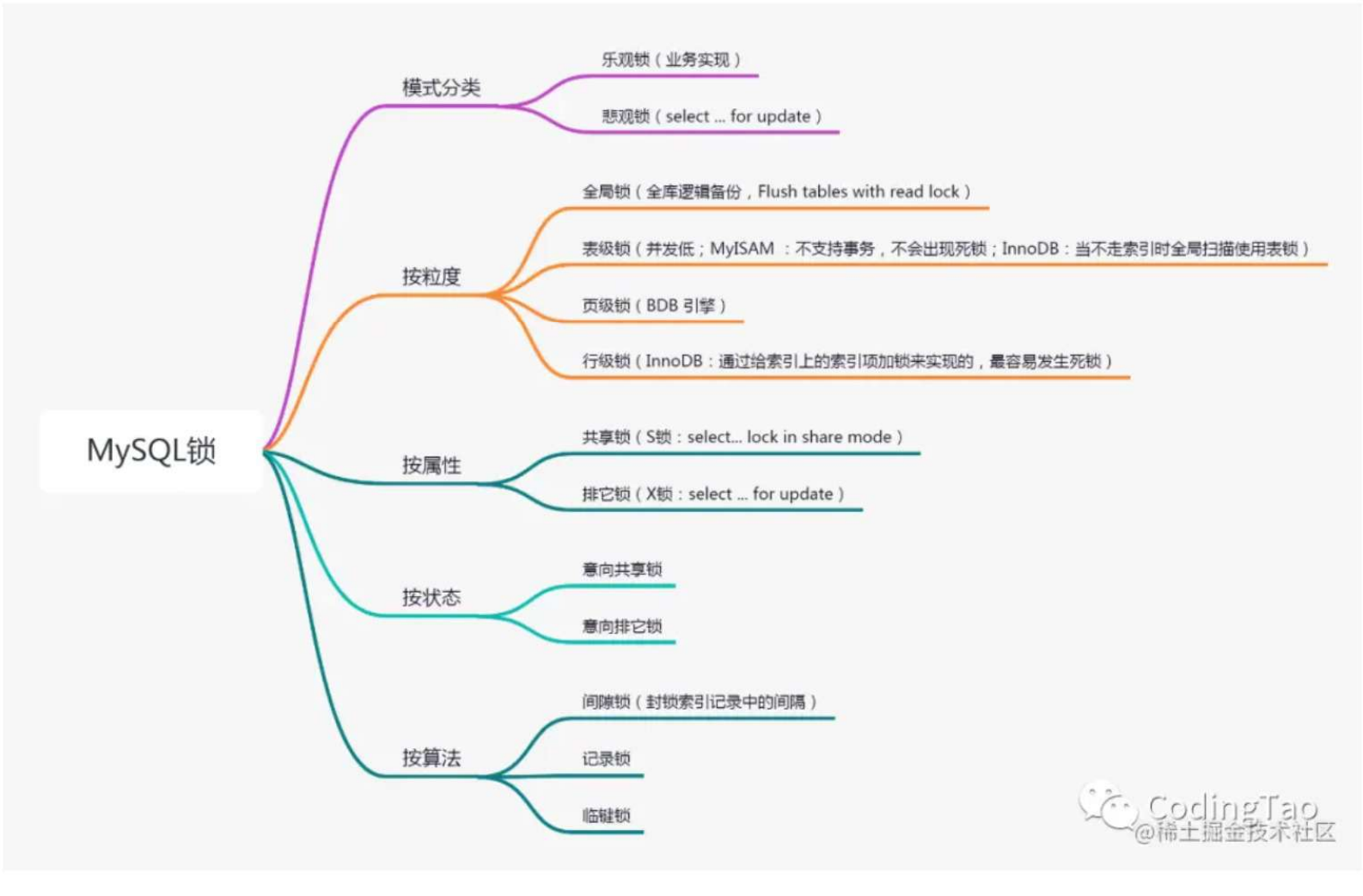
CodingTao LV.3
2021年02月22日 00:23 · 阅读 6809



一、前言

锁是计算机在执行多线程或线程时用于并发访问同一共享资源时的同步机制，MySQL中的锁是在服务器层或者存储引擎层实现的，保证了数据访问的一致性与有效性。

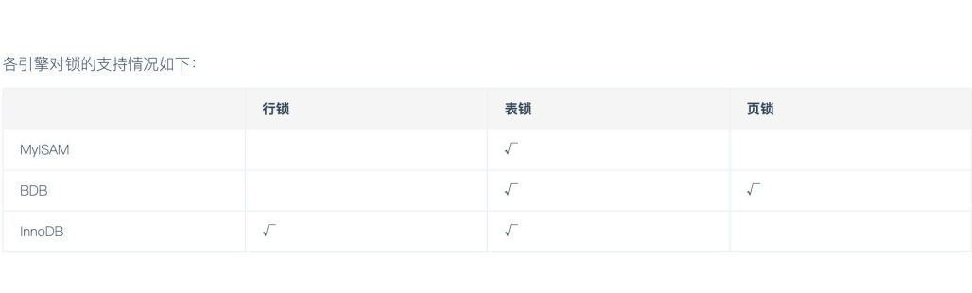
MySQL锁可以按模式分类为：乐观锁与悲观锁。按粒度分可以分为全局锁、表级锁、页级锁、行级锁。按属性可以分为：共享锁、排它锁。按状态分为：意向共享锁、意向排它锁。按算法分为：间隙锁、临键锁、记录锁。



下面将会按照上图进行一一讲解。

二、全局锁、表级锁、页级锁、行级锁

1. 全局锁



各引擎对锁的支持情况如下：

	行锁	表锁	页锁
MyISAM		√	
BDB		√	√
InnoDB	√	√	

(1) 概念

全局锁就是对整个数据库实例加锁。

(2) 应用场景

全库逻辑备份（mysqldump）

(3) 实现方式

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

当你需要让整个库处于只读状态的时候，可以使用这个命令，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）、数据定义语句（包括建表、修改表结构等）和更新类事务的提交语句。

风险点：

如果在主库上备份，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。

解决办法：

mysqldump使用参数--single-transaction，启动一个事务，确保拿到一致性视图。而由于MVCC的支持，这个过程中数据是可以正常更新的。

2. 表级锁

(1) 概念

当前操作的整张表加锁，最常使用的 MyISAM 与 InnoDB 都支持表级锁定。

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)。

(2) 实现方式

表锁：lock tables ... read/write；

例如lock tables t1 read, t2 write; 命令，则其他线程写 t1、读写 t2 的语句都会被阻塞。同时，线程 A 在执行 unlock tables 之前，也只能执行读 t1、读写 t2 的操作。连写 t1 都不允许，自然也不能在unlock tables 之前访问其他表。

元数据锁：MDL 不需要显式使用，在访问一个表的时候会被自动加上，在 MySQL 5.5 版本中引入了 MDL，当对一个表做增删改查操作的时候，加 MDL读锁；当要对表做结构变更操作的时候，加 MDL 写锁。

(3) 风险点

所谓元数据，就是表示数据本身或关系的数据，这些数据会有点奇怪，一般来说，只要不是我们存储到数据库里的数据，大多都可以理解为元数据。描述数据库的任何数据—作为数据库内容的对立面—是元数据。因此，列名、数据库名、用户名、版本名以及从SHOW语句得到的结果中的大部分字符串是元数据

参考于：www.cnblogs.com/keme/p/1106...

给一个表加字段，或者修改字段，或者加索引，需要扫描全表的数据。在对大表操作的时候，肯定会特别小心，以免对线上服务造成影响。而实际上，即使是小表，操作不慎也会出问题。

1. sessionA:

begin;

select* from t limit 1;

2. sessionB:

select* from t limit 1;

3. sessionC:

altertable t add f int;

#会mdl锁住

4. sessionD:

select* from t limit 1;

我们可以看到 session A 先启动，这时候会对表 t 加一个 MDL 读锁。由于session B 需要的也是 MDL 读锁，因此可以正常执行。

之后 session C 会被 blocked，是因为 session A 的 MDL 读锁还没有释放，而 sessionC 需要MDL 写锁，因此只能被阻塞。

如果只有 session C 自己被阻塞还没什么关系，但是之后所有要在表 t 上新申请 MDL 读锁的请求也会被 session C 阻塞。前面说了，`所有对表的增删改查操作都需要先申请MDL 读锁，而这时读锁没有释放，对表alter ，产生了mdl写锁，把表t锁住了，这时候就对表t完全不可读写了` 。

如果某个表上的查询语句频繁，而且客户端有重试机制，也就是说超时后会再起一个新session 再请求的话，这个库的线程很快就会爆满。

事务中的 MDL 锁，在语句执行开始时申请，但是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。

`**注 ：一般行锁都有锁超时时间。但是MDL锁没有超时时间的限制，只要事务没有提交就会一直锁注。**`

(4) 解决办法

首先我们要解决长事务，事务不提交，就会一直占着 MDL 锁。在 MySQL 的information_schema 库的 innodb_trx 表中，你可以查到当前执行中的事务。如果你要做 DDL 变更的表刚好有长事务在执行，要考虑先暂停 DDL，或者 kill 掉这个长事务。这也是为什么需要在低峰期做ddl 变更。

3. 页级锁

(1) 概念

页级锁是 MySQL 中锁定粒度介于行级锁和表级锁中间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。因此，采取了折衷的页级锁，一次锁定相邻的一组记录。BDB 引擎支持页级锁。

InnoDB 与MyISAM不支持页级锁

4.行级锁

(1) 概念

行级锁是粒度最低的锁，发生锁冲突的概率也最低、并发度最高。但是加锁慢、开销大，容易发生死锁现象。

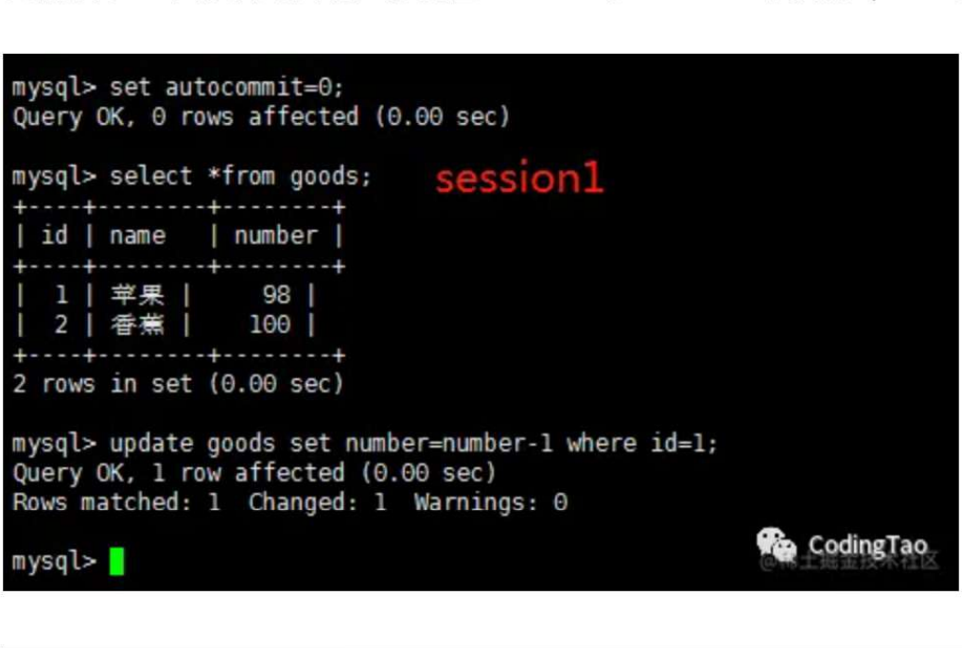
MySQL中只有InnoDB支持行级锁，行级锁分为共享锁和排他锁。

(2) 实现方式

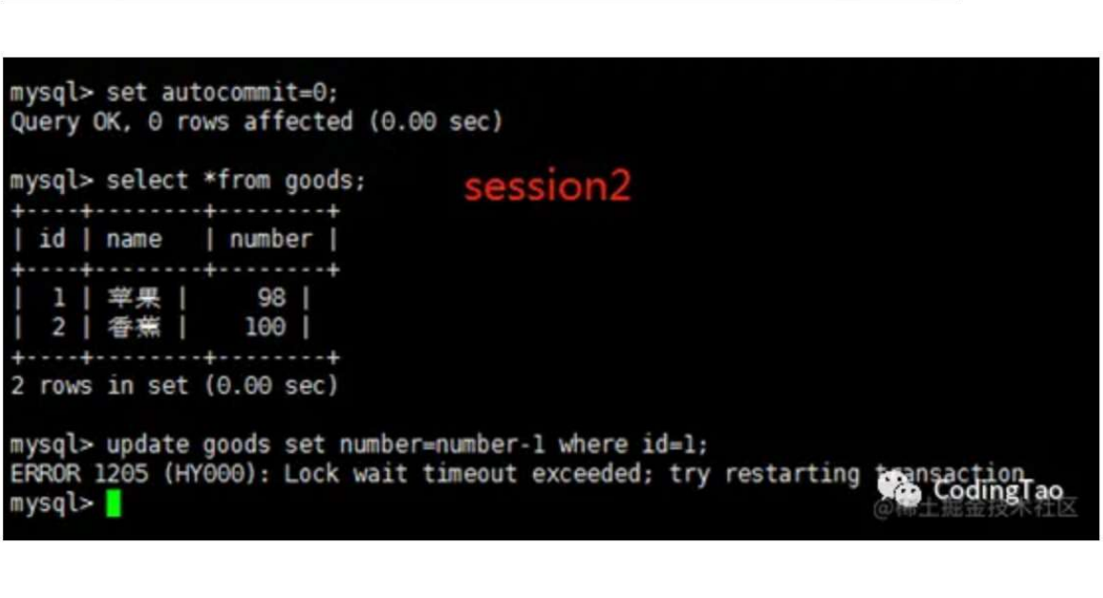
在MySQL中，行级锁并不是直接锁记录，而是锁索引。索引分为主键索引和非主键索引两种，如果一条sql 语句操作了主键索引，MySQL就会锁定这条主键索引；如果一条语句操作了非主键索引，MySQL会先锁定该非主键索引，再锁定相关的主键索引。在UPDATE、DELETE操作时，MySQL不仅锁定WHERE条件扫描过的所有索引记录，而且会锁定相邻的键值，即所谓的next-key locking。

(3) 实战

我们演示一下行锁的表现，分别在session1和session2中执行update操作，看会不会被锁定。



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

mysql> select *from goods;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     98 |
|  2 | 香蕉   |    100 |
+----+--------+--------+
2 rows in set (0.00 sec)

mysql> update goods set number=number-1 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

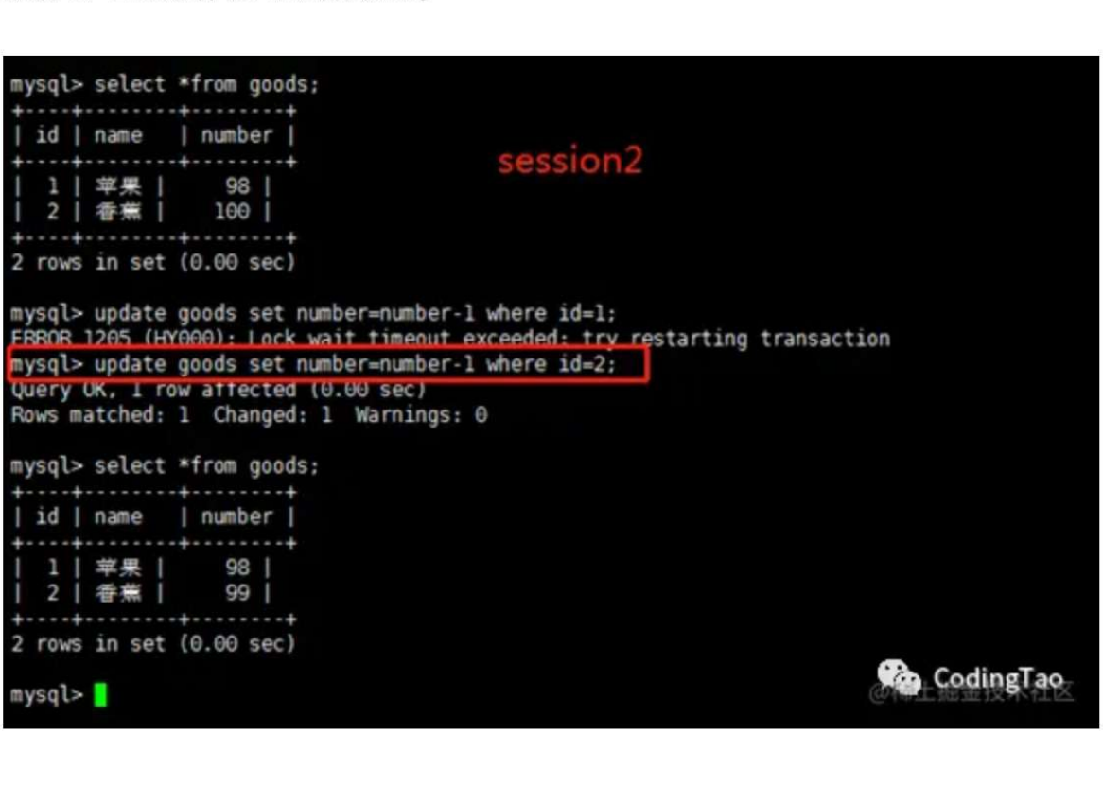


```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from goods;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     98 |
|  2 | 香蕉   |    100 |
+----+--------+--------+
2 rows in set (0.00 sec)

mysql> update goods set number=number-1 where id=1;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

可以看到由于session1迟迟未提交事务，session2在等待session1释放锁时出现了超过锁定超时的警告了。

那么如果session2执行id=2的操作会不会成功呢?

执行id=2的操作是可以成功的。



```
mysql> select *from goods;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     98 |
|  2 | 香蕉   |    100 |
+----+--------+--------+
2 rows in set (0.00 sec)

mysql> update goods set number=number-1 where id=1;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql> update goods set number=number-1 where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select *from goods;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     98 |
|  2 | 香蕉   |     99 |
+----+--------+--------+
2 rows in set (0.00 sec)

mysql>
```

三、乐观锁和悲观锁

1. 乐观锁

(1) 概念

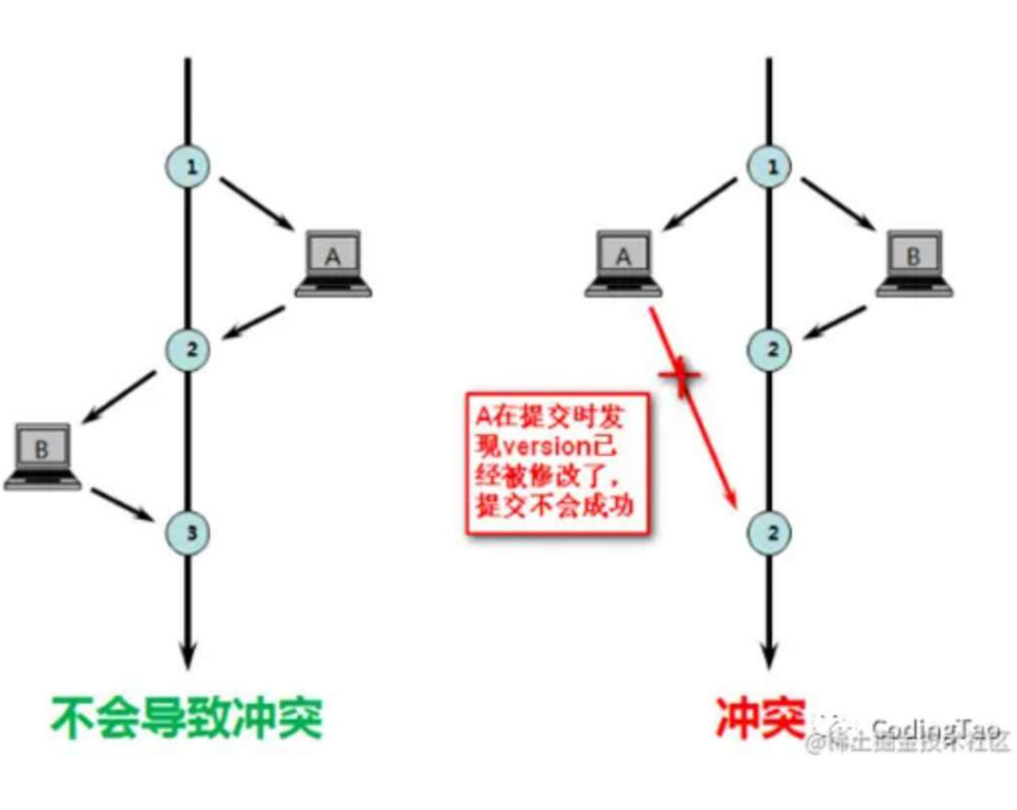
乐观锁是相对悲观锁而言的，乐观锁假设数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则返回给用户错误的信息，让用户决定如何去做。

(2) 应用场景

适用于读多写少，因为如果出现大量的写操作，写冲突的可能性就会增大，业务层需要不断重试，会大大降低系统性能。

(3) 实现方式

一般使用数据版本（Version）记录机制实现，在数据库表中增加一个数字类型的"version"字段来实现。当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加一。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的version值进行比对，如果数据库表当前版本号与第一次取出来的version值相等，则予以更新，否则认为是过期数据。



(4) 实战

订单order表中id,status,version分别代表订单ID，订单状态，版本号。

1.查询订单信息

select id,status,versionfrom order where id=#{id};

2.用户支付成功

3.修改订单状态

update set status=支付成功,version=version+1 where id=#{id} and version=#{ version};

2. 悲观锁

(1) 概念

悲观锁，正如其名，具有强烈的独占和排他特性，每次去拿数据的时候都认为别人会修改，对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。

(2) 应用场景

适用于并发量不大、写入操作比较频繁、数据一致性比较高的场景。

(3) 实现方式

在MySQL中使用悲观锁，必须关闭MySQL的自动提交，set autocommit=0。共享锁和排它锁是悲观锁的不同的实现，它俩都属于悲观锁的范畴。

(4) 实战

商品goods表中id,name,number分别代表商品ID，商品名称，商品库存。

1.开启事务并关闭自动提交

setautocommit=0;

2.查询商品信息

selectid,name,number from goods where id=1 for update;

3.用户下单，生成订单

4.修改商品库存

updateset number= number-1 where id=1;

5.提交事务

commit;

说明：select...for update是MySQL提供的实现悲观锁的方式，属于排它锁，在goods表中，id为1的那条数据就被当前事务锁定了，其它的要执行select id,name,number from goods where id=1for update;的事务必须等本次事务提交之后才能执行。这样我们可以保证当前的数据不会被其它事务修改。

注意：此时MySQL InnoDB默认行级锁。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，会使用表级锁把整张表锁住。

四、共享锁和排它锁

1. 共享锁

(1) 概念

共享锁，又称之为读锁，简称S锁，当事务A对数据加上读锁后，其他事务只能对该数据加读锁，不能做任何修改操作，也就是不能添加写锁。只有当事务A上的读锁被释放后，其他事务才能对其添加写锁。

(2) 应用场景

共享锁主要是为了支持并发的读取数据而出现的，读取数据时，不允许其他事务对当前数据进行修改操作，从而避免"不可重读"的问题的出现。

适合于两张表存在关系时的写操作，拿mysql官方文档的例子来说，一个表是child表，一个是parent表，假设child表的某一列child_id映射到parent表的c_child_id列，那么从业务角度讲，此时我直接insert一条child_id=100记录到child表是存在风险的，因为刚insert的时候可能在parent表里删除了这条c_child_id=100的记录，那么业务数据就存在不一致的风险。正确的方法是再插入时执行select * from parent where c_child_id=100 lock in share mode,锁定了parent表的这条记录，然后执行insert into child(child_id)values (100)就不会存在这种问题了。

(3) 实现方式

select ...lock in share mode

(4) 实战

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from goods where id=1 lock in share mode;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     97 |
+----+--------+--------+
1 row in set (0.00 sec)
```

```
mysql> select *from goods;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     97 |
|  2 | 香蕉   |     99 |
+----+--------+--------+
2 rows in set (0.00 sec)

mysql> update goods set number=number-1 where id=1;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

session1持有共享锁，未提交。session2的查询不受影响，但是update操作会被一直阻塞，直到超时。

2. 排它锁

(1) 概念

排它锁，又称之为写锁，简称X锁，当事务对数据加上写锁后，其他事务既不能对该数据添加读写，也不能对该数据添加写锁，写锁与其他锁都是互斥的。只有当前数据写锁被释放后，其他事务才能对其添加写锁或者是读锁。

MySQL InnoDB引擎默认update,delete,insert都会自动给涉及到的数据加上排他锁，select语句默认不会加任何锁类型。

(2) 应用场景

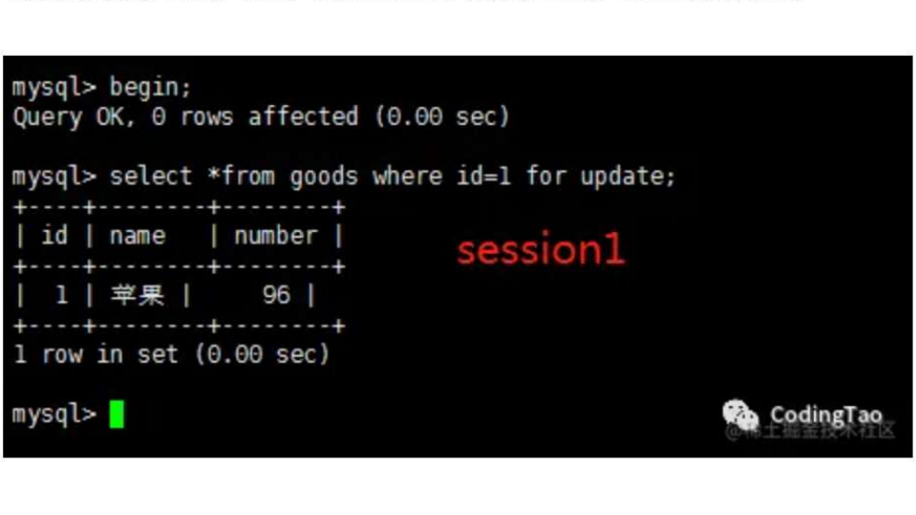
写锁主要是为了解决在修改数据时，不允许其他事务对当前数据进行修改和读取操作，从而可以有效避免"脏读"问题的产生。

(3) 实现方式

select ...for update

(4) 实战

session1排它锁查询。session2也做排它锁查询会被阻塞。



```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from goods where id=1 for update;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     96 |
+----+--------+--------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from goods where id=1 for update;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

五、意向共享锁和意向排它锁

1. 概念

意向锁是表锁，为了协调行锁和表锁的关系，支持多粒度（表锁与行锁）的锁并存。

2. 作用

当有事务A有行锁时，MySQL会自动为该表添加意向锁，事务B如果想申请整个表的写锁，那么不需要遍历每一行判断是否存在行锁，而直接判断是否存在意向锁，增强性能。

3. 意向锁的兼容互斥性

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

4. 实战注意：这里的排它 / 共享锁指的都是表锁！！！意向锁不会与行级的共享 / 排它锁互斥！！！

session1获取了某一行的排他锁，并未提交：

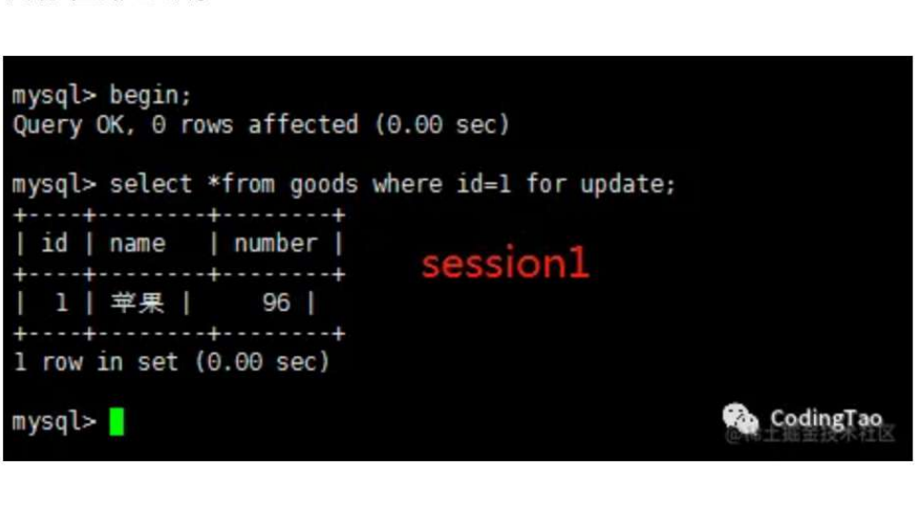
select*from goods where id=1 for update;

此时 goods 表存在两把锁：goods 表上的意向排它锁与 id 为 1 的数据行上的排它锁。

session2 想要获取 goods 表的共享锁：

LOCK TABLES goods READ;

此时session2 检测session1 持有goods 表的意向排他锁，就可以得知session1必然持有该表中某些数据行的排他锁，那么session2 对 goods 表的加锁请求就会被排斥（阻塞），而无需去检测表中的每一行数据是否存在排它锁。



```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from goods where id=1 for update;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     96 |
+----+--------+--------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> LOCK TABLES goods READ;
```

六、间隙锁、临键锁、记录锁

- **概念**

记录锁、间隙锁、临键锁都是排它锁，而记录锁的使用方法跟排它锁介绍一致。

- **记录锁**

记录锁是封锁记录，记录锁也叫行锁，例如：

select *from goods where **`id`=**1 for update;

它会在 id=1 的记录上加上记录锁，以阻止其他事务插入，更新，删除 id=1 这一行。

- **间隙锁**

间隙锁基于非唯一索引，它锁定一段范围内的索引记录。使用间隙锁锁住的是一个区间，而不仅仅是这个区间中的每一条数据。

select* from goods where id between 1 and 10 for update;

即所有在（1，10）区间内的记录行都会被锁住，所有id 为 2、3、4、5、6、7、8、9 的数据行的插入会被阻塞，但是 1和 10 两条记录行并不会被锁住。

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from goods where id between 1 AND 10 for update;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     96 |
|  2 | 香蕉   |     99 |
+----+--------+--------+
2 rows in set (0.00 sec)

mysql>
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into goods(id,name,number) values(3,'苹果',100);
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

- **临键锁**

临键锁，是记录锁与间隙锁的组合，它的封锁范围，既包含索引记录，又包含索引区间，是一个左开右闭区间。临键锁的主要目的，也是为了避免幻读(Phantom Read)。如果把事务的隔离级别降级为RC，临键锁则也会失效。

每个数据行上的非唯一索引列上都会存在一把临键锁，当某个事务持有该数据行的临键锁时，会锁住一段左开右闭区间的数据。需要强调的一点是，InnoDB 中行级锁是基于索引实现的，临键锁只与非唯一索引列有关，在唯一索引列（包括主键列）上不存在临键锁。

```
mysql> select *from goods;
+----+--------+--------+
| id | name   | number |
+----+--------+--------+
|  1 | 苹果   |     96 |
|  2 | 香蕉   |     99 |
+----+--------+--------+
2 rows in set (0.00 sec)

mysql> update goods SET name = '苹果2' WHERE number = 96;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

goods表中隐藏的临键锁有：(-∞, 96],(96, 99],(99, +∞]

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into goods(id,name,number) VALUES(100, '苹果3', 97);
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

session1 在对 number 为 96 的列进行 update 操作的同时，也获取了(-∞, 96],(96, 99]这两个区间内的临键锁。

最终我们就可以得知，在根据非唯一索引对记录行进行 UPDATE \ FOR UPDATE \LOCK IN SHARE MODE 操作时，InnoDB 会获取该记录行的临键锁，公式为：左gap lock + record lock + 右gap lock。

即session1在执行了上述的 SQL 后，最终被锁住的记录区间为 (-∞, 99)。

【MySQL】MySQL中的锁机制

✔ MySQL锁机制起步

锁是计算机用以协调多个进程间并发访问同一共享资源的一种机制。MySQL中为了保证数据访问的一致性与有效性等功能，实现了锁机制，MySQL中的锁是在服务器层或者存储引擎层实现的。

✔ 行锁与表锁

首先我们来了解行锁与表锁的基本概念，从名字中我们就可以了解：表锁就是对整张表进行加锁，而行锁则是锁定某行、某几行数据或者行之间的间隙。

表锁是MySQL Server实现的，所以任何引擎都可以支持表锁

各引擎对锁的支持情况如下：

	行锁	表锁	页锁
MyISAM		√	
BDB		√	√
InnoDB	√	√	

行锁

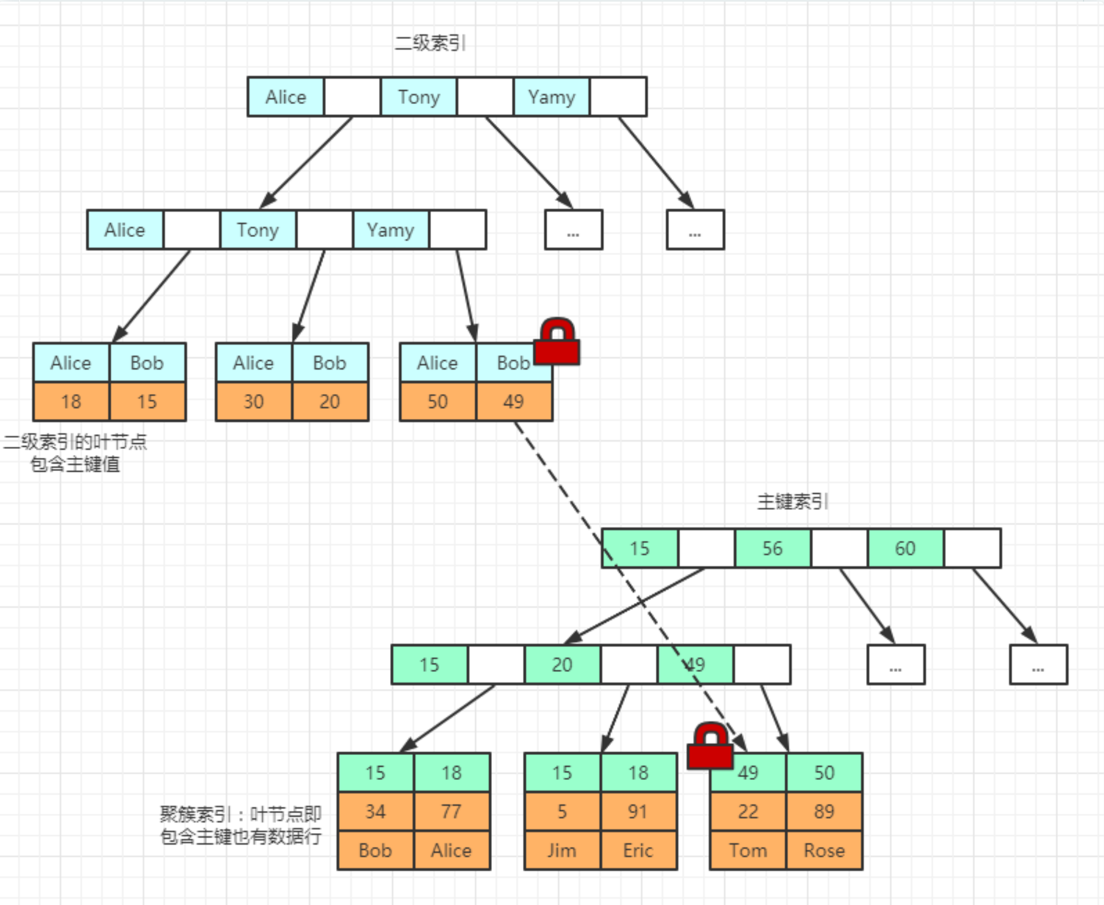
> A record lock is a lock on an index record. Record locks always lock index records, even if a table is defined with no indexes. For such cases, InnoDB creates a hidden clustered index and uses this index for record locking.

上文出自MySQL的官方文档，从这里我们可以看出行锁是作用在索引上的，哪怕你在建表的时候没有定义一个索引，InnoDB也会创建一个聚簇索引并将其作为锁作用的索引。所以MySQL中实现行锁的前提是索引

这里还是讲一下InnoDB中的聚簇索引。每一个InnoDB表都需要一个聚簇索引，有且只有一个。如果你为该表表定义一个主键，那么MySQL将使用主键作为聚簇索引；如果你不为定义一个主键，那么MySQL将会把第一个唯一索引（而且要求NOT NULL）作为聚簇索引；如果上诉两种情况都GG，那么MySQL将自动创建一个名字为 `GEN_CLUST_INDEX` 的隐藏聚簇索引。

因为是聚簇索引，所以B+树上的叶子节点都存储了数据行，那么如果现在是二级索引呢？InnoDB中的二级索引的叶节点存储的是主键值（或者说聚簇索引的值），所以通过二级索引查询数据时，还需要将对应的主键去聚簇索引中再次进行查询。

关于索引的问题就到这，我们用一张直观的图来表示行锁：



接下来以两条SQL的执行为例，讲解一下InnoDB对于单行数据的加锁原理：

```
update user set age = 10 where id = 49;
update user set age = 10 where name = 'Tom';
```

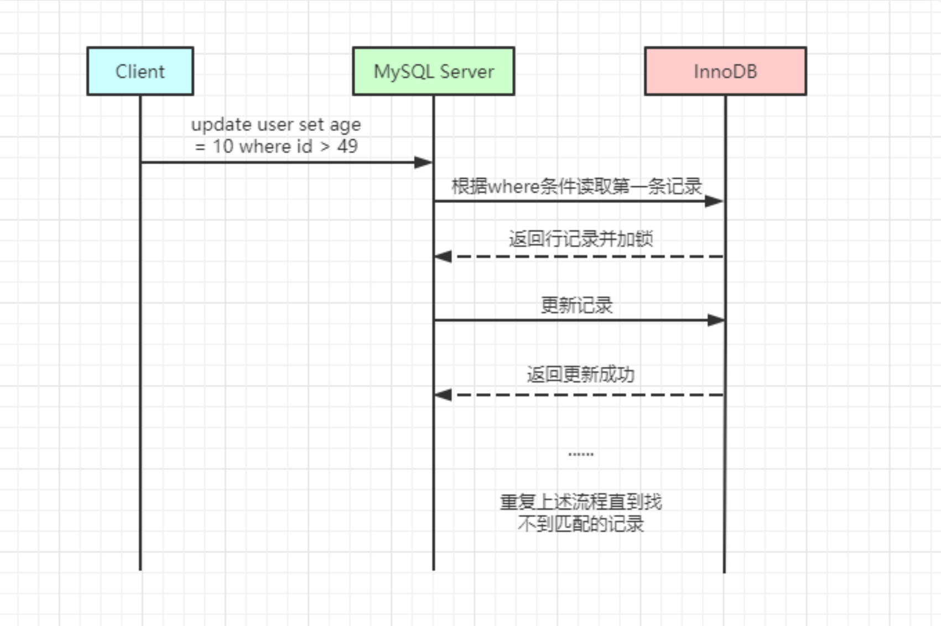
第一条SQL使用主键查询，只需要在 id = 49 这个主键索引上加上锁。第二条 SQL 使用二级索引来查询，那么首先在 name = Tom 这个索引上加写锁，然后由于使用 InnoDB 二级索引还需再次根据主键索引查询，所以还需要在 id = 49 这个主键索引上加锁。

也就是说使用主键索引需要加一把锁，使用二级索引需要在二级索引和主键索引上各加一把锁。

根据索引对单行数据进行更新的加锁原理了解了，那如果更新操作涉及多个行呢，比如下面 SQL 的执行场景。

```
update user set age = 10 where id > 49;
```

上述 SQL 的执行过程如下图所示。MySQL Server 会根据 WHERE 条件读取第一条满足条件的记录，然后 InnoDB 引擎会将第一条记录返回并加锁，接着 MySQL Server 发起更新改行记录的 UPDATE 请求，更新这条记录。一条记录操作完成，再读取下一条记录，直至没有匹配的记录为止。

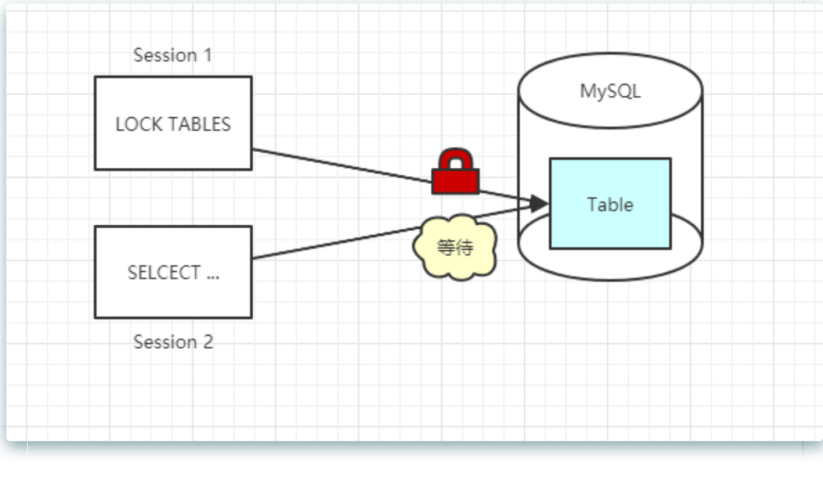


表锁

上面我们讲解行锁的时候，操作语句中的条件判断列都是有建立索引的，那么如果现在的判断列不存在索引呢？InnoDB既支持行锁，也支持表锁，当没有查询列没有索引时，InnoDB就不会去搞什么行锁了，毕竟行锁一定要有索引，所以它现在搞表锁，把整张表给锁住了。那么具体啥是表锁？还有其他什么情况下也会进行锁表呢？

表锁使用的是一次性锁技术，也就是说，在会话开始的地方使用 lock 命令将后续需要用到的表都加上锁，在表释放前，只能访问这些加锁的表，不能访问其他表，直到最后通过 unlock tables 释放所有表锁。

除了使用 unlock tables 显示释放锁之外，会话持有其他表锁时执行lock table 语句会释放会话之前持有的锁；会话持有其他表锁时执行 start transaction 或者 begin 开启事务时，也会释放之前持有的锁。



表锁由 MySQL Server 实现，行锁则是存储引擎实现，不同的引擎实现的不同。在 MySQL 的常用引擎中 InnoDB 支持行锁，而 MyISAM 则只能使用 MySQL Server 提供的表锁。

两种锁的比较

表锁：加锁过程的开销小，加锁的速度快；不会出现死锁的情况；锁定的粒度大，发生锁冲突的几率大，并发度低；

- 一般在执行DDL语句时会对整个表进行加锁，比如说 ALTER TABLE 等操作；
- 如果对InnoDB的表使用行锁，被锁定字段不是主键，也没有针对它建立索引的话，那么将会锁整张表；
- 表级锁更适合于以查询为主，并发用户少，只有少量按索引条件更新数据的应用，如Web 应用。

行锁：加锁过程的开销大，加锁的速度慢；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；

- 最大程度的支持并发，同时也带来了最大的锁开销。
- 在 InnoDB 中，除单个 SQL 组成的事务外，锁是逐步获得的，这就决定了在 InnoDB 中发生死锁是可能的。
- 行级锁只在存储引擎层实现，而 MySQL 服务器层没有实现。行级锁更适合于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理（OLTP）系统。

✔ MyISAM表锁

MyISAM表级锁模式

- 表共享读锁（Table Read Lock）：不会阻塞其他线程对同一个表的读操作请求，但会阻塞其他线程的写操作请求；
- 表独占写锁（Table Write Lock）：一旦表被加上独占写锁，那么无论其他线程是读操作还是写操作，都会被阻塞；

默认情况下，写锁比读锁具有更高的优先级；当一个锁释放后，那么它会优先相应写锁等待队列中的锁请求，然后再是读锁中等待的获取锁的请求。

> This ensures that updates to a table are not "starved" even when there is heavy SELECT activity for the table. However, if there are many updates for a table, SELECT statements wait until there are no more updates.

这种设定也是MyISAM表不适合于有大量更新操作和查询操作的原因。大量更新操作可能会造成查询操作很难以获取读锁，从而过长的阻塞。同时一些需要长时间运行的查询操作，也会使得线程"饿死"，应用中应尽量避免出现长时间运行的查询操作（在可能的情况下可以通过使用中间表等措施对SQL语句做一定的"分解"，使每一步查询都能在较短的时间内完成，从而减少锁冲突。如果复杂查询不可避免，应尽量安排在数据库空闲时段执行，比如一些定期统计可以安排在夜间执行。）

我们可以通过一些设置来调节MyISAM的调度行为：

- 通过指定启动参数 `low-priority-updates`，使MyISAM引擎默认给予读请求以优先的权利；
- 通过执行命令 `SET LOW_PRIORITY_UPDATES=1`，使该连接发出的更新请求优先级降低；
- 通过指定INSERT、UPDATE、DELETE语句的 `LOW_PRIORITY` 属性，降低该语句的优先级；
- 给系统参数 `max_write_lock_count` 设置一个合适的值，当一个表的读锁达到这个值后，MySQL就暂时将写请求的优先级降低，给读进程一定获得锁的机会。

MyISAM对表加锁分析

MyISAM在执行查询语句（SELECT）前，会自动给涉及的所有表加读锁，在执行更新操作（UPDATE、DELETE、INSERT等）前，会自动给涉及的表加写锁，这个过程并不需要用户干预，因此用户一般不需要直接用 LOCK TABLE 命令给 MyISAM 表显式加锁。在自动加锁的情况下，MyISAM 总是一次获得 SQL 语句所需要的全部锁，这也正是 MyISAM 表不会出现死锁（Deadlock Free）的原因。

MyISAM存储引擎支持并发插入，以减少给定表的读操作和写操作之间的争用：

如果MyISAM表在数据文件中没有空闲块（由于删除或更新导致的空行），则行始终插入数据文件的末尾。在这种情况下，你可以自由混合并发使用MyISAM表的 INSERT 和 SELECT 语句而不需要加锁（你可以在其他线程进行读操作的情况下，同时将行插入到MyISAM表中）。如果文件中有空闲块，则并发插入会被禁止，但当所有的空闲块重新填充有新数据时，它又会自动启用。要控制此行为，可以使用MySQL的concurrent_insert系统变量。

- 当concurrent_insert=0时，不允许并发插入功能。
- 当concurrent_insert=1时，允许对没有空闲块的表使用并发插入，新数据位于数据文件结尾（缺省）。
- 当concurrent_insert=2时，不管表有没有空想块，都允许在数据文件结尾并发插入。

显式加表锁的应用

上面已经提及了表锁的加锁方式，一般表锁都是隐式加锁的，不需要我们去主动声明，但是也有需要显式加锁的情况，这里简单做下介绍：

给MyISAM表显式加锁，一般是为了一定程度模拟事务操作，实现对某一时间点多个表的一致性读取。例如，有一个订单表orders，其中记录有订单的总金额total，同时还有一个订单明细表 order_detail，其中记录有订单每一产品的金额小计 subtotal，假设我们需要检查这两个表的金额合计是否相等，可能就需要执行如下两条SQL：

```
SELECT SUM(total) FROM orders;
SELECT SUM(subtotal) FROM order_detail;
```

这时，如果不先给这两个表加锁，就可能产生错误的结果，因为第一条语句执行过程中，order_detail表可能已经发生了改变。因此，正确的方法应该是：

```
LOCK tables orders read local,order_detail read local;
SELECT SUM(total) FROM orders;
SELECT SUM(subtotal) FROM order_detail;
Unlock tables;
```

查看表锁争用情况：

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的争夺，如果 Table_locks_waited 的值比较高，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 | 1151552 |
| Table_locks_waited | 15324 |
+-----------------------+----------+
```

✔ InnoDB行锁与表锁

InnoDB锁模式

1）InnoDB中的行锁

InnoDB实现了以下两种类型的行锁：

- 共享锁（S）：加了锁的记录，所有事务都能去读取但不能修改，同时阻止其他事务获得相同数据集的排他锁；
- 排他锁（X）：允许已经获得排他锁的事务去更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁；

2）InnoDB表锁——意向锁

由于表锁和行锁虽然锁定范围不同，但是会相互冲突。当你要加表锁时，势必要先遍历该表的所有记录，判断是否有排他锁。这种遍历检查的方式显然是一种低效的方式，MySQL引入了意向锁，来检测表锁和行锁的冲突。

> Intention locks are table-level locks that indicate which type of lock (shared or exclusive) a transaction requires later for a row in a table。
>
> The intention locking protocol is as follows:
>
> - Before a transaction can acquire a shared lock on a row in a table, it must first acquire an IS lock or stronger on the table.
> - Before a transaction can acquire an exclusive lock on a row in a table, it must first acquire an IX lock on the table.

意向锁也是表级锁，分为读意向锁（IS锁）和写意向锁（IX锁）。当事务要在记录上加上行锁时，要首先在表上加上意向锁。这样判断表中是否有记录正在加锁就很简单了，只要看下表上是否有意向锁就行了，从而就能提高效率。

意向锁之间是不会产生冲突的，它只会阻塞表级读锁或写锁。意向锁不于行级锁发生冲突。

锁模式的兼容矩阵

下面表显示了了各种锁之间的兼容情况：

	X	IX	S	IS
X				
IX		兼容		兼容
S			兼容	兼容
IS		兼容	兼容	兼容

（注意上面的X与S是说表级的X锁和S锁，意向锁不和行级锁发生冲突）

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；如果两者不兼容，那么该事务就需要等待锁的释放。

InnoDB的加锁方法

- 意向锁是 InnoDB 自动加的，不需要用户干预；
- 对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及的数据集加上排他锁；
- 对于普通的SELECT语句，InnoDB不会加任何锁；事务可以通过以下语句显示给记录集添加共享锁或排他锁：
  - 共享锁（S）：`select * from table_name where ... lock in share mode`。此时其他 session 仍然可以查询记录，并也可以对该记录加 share mode 的共享锁。但是如果当前事务需要对该记录进行更新操作，则很有可能造成死锁。
  - 排他锁（X）：`select * from table_name where ... for update`。其他session可以查询记录，但是不能对该记录加共享锁或排他锁，只能等待锁释放后在加锁。

select for update

在执行这个 select 查询语句的时候，会将对应的索引访问条目加上排他锁（X锁），也就是说这个语句对应的锁就相当于update带来的效果；

使用场景：为了让确保自己查找到的数据一定是最新数据，并且查找到后的数据值允许自己来修改，此时就需要用到select for update语句；

性能分析：select for update语句相当于一个update语句。在业务繁忙的情况下，如果事务没有及时地commit或者rollback可能会造成事务长时间的等待，从而影响数据库的并发使用效率。

select lock in share mode

in share mode 子句的作用就是将查找的数据加上一个share锁，这个就是表示其他的事务只能对这些数据进行简单的 select 操作，而不能进行 DML 操作。

使用场景：为了确保自己查询的数据不会被其他事务正在修改，也就是确保自己查询到的数据是最新的数据，并且不允许其他事务来修改数据。与select for update不同的是，本事务在查找完之后不一定能去更新数据，因为有可能其他事务也对同数据集使用了 in share mode 的方式加上了S锁；

性能分析：select lock in share mode 语句是一个给查找的数据上一个共享锁（S 锁）的功能，它允许其他的事务也对该数据上S锁，但是不能够允许对该数据进行修改。如果不及时的commit 或者rollback 也可能会造成大量的事务等待。

InnoDB的锁争用情况

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

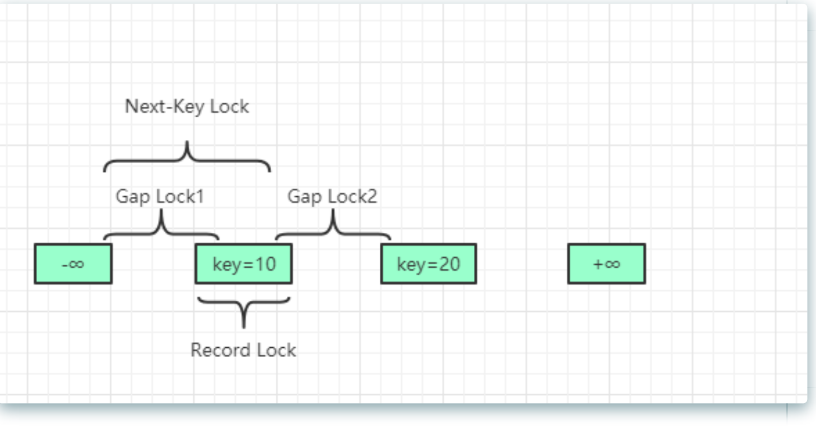
```
mysql> show status like 'innodb_row_lock%';
+-------------------------------+-------+
| Variable_name | Value |
+-------------------------------+-------+
| InnoDB_row_lock_current_waits | 0 |
| InnoDB_row_lock_time | 0 |
| InnoDB_row_lock_time_avg | 0 |
| InnoDB_row_lock_time_max | 0 |
| InnoDB_row_lock_waits | 0 |
+-------------------------------+-------+
5 rows in set (0.01 sec)
```

✔ 行锁的类型

所以记录锁、间隙锁、临键锁、插入意向GAP锁都属于行锁

上面我们根据了锁的粒度将锁分为了行锁与表锁，接下来根据使用场景的不同，又可以将行锁进行进一步的划分：Next-Key Lock、Gap Lock、Record Lock以及插入意向GAP锁。

不同的锁锁定的位置是不同的，比如说记录锁只锁定对应的记录，而间隙锁锁住记录和记录之间的间隙，Next-key Lock则锁住所属记录之间的间隙。不同的锁类型锁定的范围大致如图所示：



记录锁（Record Lock）

记录锁最简单的一种行锁形式，上面我们以及稍微提及过了。这里补充下的点就是：行锁是加在索引上的，如果当你的查询语句不走索引的话，那么它就会升级到表锁，最终造成效率低下，所以在写SQL语句时需要特别注意。

间隙锁（Gap Lock）

> A gap lock is a lock on a gap between index records, or a lock on the gap before the first or after the last index record。

当我们使用范围条件而不是相等条件去检索，并请求锁时，InnoDB就会给符合条件的记录的索引项加上锁；而对于键值在条件范围内但并不存在（参考上面所说的空闲块）的记录，就叫做间隙，InnoDB在此时也会对间隙加锁，这种记录锁+间隙锁的机制叫Next-Key Lock。额，扯的有点快。

从上面这句话可以表明间隙锁是所在两个存在的索引之间，是一个开区间，像最开始的那张索引图，15和18之间，是有（16，17）这个间隙存在的。

> Gap locks in InnoDB are "purely inhibitive", which means that their only purpose is to prevent other transactions from inserting to the gap. Gap locks can co-exist. A gap lock taken by one transaction does not prevent another transaction from taking a gap lock on the same gap. There is no difference between shared and exclusive gap locks. They do not conflict with each other, and they perform the same function.

上面这段话表明间隙锁是可以共存的，共享间隙锁与独占间隙锁之间是没有区别的，两者之间并不冲突。其存在的目的都是防止其他事务往间隙中插入新的纪录，故而一个事务所采取的间隙锁是不会去阻止另外一个事务在同一个间隙中加锁的。

当然也不是在什么时候都会去加间隙锁的：

> Gap locking can be disabled explicitly. This occurs if you change the transaction isolation level to READ COMMITTED. Under these circumstances, gap locking is disabled for searches and index scans and is used only for foreign-key constraint checking and duplicate-key checking.

这段话表明，在 RU 和 RC 两种隔离级别下，即使你使用 select in share mode 或 select for update，也无法防止 **幻读**（读后写的场景）。因为这两种隔离级别下只会有 **行锁**，而不会有 **间隙锁**。而如果是 RR 隔离级别的话，就会在间隙上加上间隙锁。

临键锁（Next-key Lock）

> A next-key lock is a combination of a record lock on the index record and a gap lock on the gap before the index record.

临键锁是记录锁与与间隙锁的结合，所以临键锁与间隙锁是一个同时存在的概念，并且临键锁是个左开有闭的却比如(16, 18]。

关于临键锁与幻读，官方文档有这么一条说明：

> By default, InnoDB operates in REPEATABLE READ transaction isolation level. In this case, InnoDB uses next-key locks for searches and index scans, which prevents phantom rows.

就是说 MySQL 默认隔离级别是RR，在这种级别下，如果你使用 select in share mode 或者 select for update 语句，那么InnoDB会使用临键锁（记录锁 + 间隙锁），因而可以防止幻读；

但是我也在网上看到相关描述：即使你的隔离级别是 RR，如果你这是使用普通的select语句，那么此时 InnoDB 引擎将是使用快照读，而不会使用任何锁，因而还是无法防止幻读。（其实普通读应该是快照读没错，但是快照读应该是不会有幻读幻读问题，mmp）。

插入意向锁（Insert Intention Lock）

> An insert intention lock is a type of gap lock set by INSERT operations prior to row insertion. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6, respectively, each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

官方文档已经解释得很清楚了，这里我做个翻译机：

插入意图锁是一种间隙锁，在行执行 INSERT 之前的插入操作设置。如果多个事务 INSERT 到同一个索引间隙之间，但没有在同一位置上插入，则不会产生任何的冲突。假设有值为4和7的索引记录，现在有两事务分别尝试插入值为 5 和 6 的记录，在获得插入行的排他锁之前，都使用插入意向锁锁住 4 和 7 之间的间隙，但两者之间并不会相互阻塞，因为这两行并不冲突。

插入意向锁只会和 间隙或者 Next-key 锁冲突，正如上面所说，间隙锁作用就是防止其他事务插入记录造成幻读，正是由于在执行 INSERT 语句时需要加插入意向锁，而插入意向锁和间隙锁冲突，从而阻止了插入操作的执行。

不同类型锁之间的兼容

不同类型的锁之间的兼容如下表所示：

	RECORED	GAP	NEXT-KEY	II GAP（插入意向锁）
RECORED		兼容		兼容
GAP	兼容	兼容	兼容	兼容
NEXT-KEY		兼容		兼容
II GAP	兼容			兼容

（其中行表示已有的锁，列表示意图加上的锁）

其中，第一行表示已有的锁，第一列表示要加的锁。插入意向锁较为特殊，所以我们先对插入意向锁做个总结，如下：

- 插入意向锁不影响其他事务加其他任何锁。也就是说，一个事务已经获取了插入意向锁，对其他事务是没有任何影响的；
- 插入意向锁与间隙锁和 Next-key 锁冲突。也就是说，一个事务想要获取插入意向锁，如果有其他事务已经加了间隙锁或 Next-key 锁，则会阻塞。

其他类型的锁的规则较为简单：

- 间隙锁不和其他锁（不包括插入意向锁）冲突；
- 记录锁和记录锁冲突，Next-key 锁和 Next-key 锁冲突，记录锁和 Next-key 锁冲突；

> 文章写到这里吧，再写下去有点长了。上面文章中很多信息都来源网络，我只是个搬运工，假若哪里有表述错误，请评论区留言。

参考资料：

- InnoDB Locking
- 把MySQL中的各种锁及其原理都画出来
- MySQL中的锁（表锁、行锁）
- 关于MySQL MyISAM 表并发

分类：Middleware
标签：MySQL

« 上一篇：【Java】final 域的内存语义
» 下一篇：【负载均衡】常见的负载均衡算法的实现与应用

posted @ 2020-11-15 22:14 周二鸭 阅读(5772) 评论(5) 编辑 收藏 举报