**Mysql 锁基础知识**

1. **锁的种类**
   * **粒度划分：**

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。

* + **上锁时间划分：**

悲观锁：先校验在操作（写多读少场景）。毫无疑问就是每次操作前就给资源上锁。

乐观锁：先操作再校验（读多写少场景）。对于悲观锁的校验很简单就是判断资源是否被锁，如果被锁则等待，否则操作资源。那么乐观锁怎么判断资源是否被操作过呢？

* + - 版本号机制

所谓的版本号机制就是给数据的修改操作加一个编号Version，每次修改这个数据版本号就是自增（当然这个是原子性的），每次我们修改数据前先记录这个数据的版本号，然后操作完以后判断这个版本号是否和我们操作前一致，如果一致则修改不一致则放弃修改，重试整个过程。

* + - CAS算法（Compare And Swap）

与版本号机制殊途同归，这里是先记录内存中的值V将它存储到A变量中，然后我们操作完比较A变量和内存中的值V是否一致，如果一致则进行修改（这里要保证比较和修改是一个原子性操作），否则进行自旋然后重试。那CAS同版本号机制相对比有哪些缺陷呢？

1.无法解决ABA问题，什么是ABA问题呢？还是以刚刚的修改过程来举例子，内存中有一个变量v=1，要是v=2这个操作是原子性的，按着CAS的流程  
（1）A=1；

（2）B=2  
（3）判断A和如今内存中的v是否相等并且等于1，如果相等并且等于1 则将B更新成v的值，否则则自旋重新进行操作。

但是如果在（3）步骤之前v从1->2…->3->1的话，v的值还是会更新的，这就达不到原子性的功能了。  
2.CAS每次只能保证一个变量是原子性的。

3.CAS可能长时间自旋，导致cpu空耗。

* + **可重入性**

意向锁（优化目的）：意向锁分为意向共享锁和意向排他锁。意向锁是个表锁，和行锁相互独立，是用来优化的。当事务想要获取表锁的时候，如果不存在意向锁，则需要扫描整张表看看是否已经有行锁，性能差。而意向锁则是当事务给表加行锁的时候同时给表加上意向锁（行锁是共享锁则加意向共享锁IS，行锁是写锁则加意向排他锁IX），避免加表锁的时候全表扫描。

共享锁（读锁）：多个事务可以同时读取数据，但不允许其他连接修改数据。共享锁的加锁过程：判断该请求是否需要读取整张表，如果需要则判断该表上是否具有意向排他锁（IX），如果有则等待，否则给表加意向共享锁（IS），然后给需要读取的数据加共享锁。

排他锁（写锁）：只有一个连接可以修改数据，其他连接既不能读也不能改。排他锁的加锁过程：首先判断该锁是否是表锁，如果是表锁则获取数据表判断该表是否有意向排他锁（IX）或者意向共享锁（IS）如果存在则等待，不存在则给该表加上意向排他锁，然后给数据行加写锁。

* + **锁的范围**

记录锁：是对索引记录的锁定。比如使用in或者等于号命中索引的时候，锁住多条记录的行锁。

间隙锁：是对索引记录中间的数据的锁定。比如使用>号或者<号命中索引的区间锁的时候，锁住这段区间的数据。

临键锁：记录锁加间隙锁。

1. **死锁的排查**
   * **查找最近一次死锁**

**show engine innodb status;**

=====================================

2020-10-03 10:13:30 0x7f8aa15c4700 INNODB MONITOR OUTPUT。//输出当前时间

=====================================

Per second averages calculated from the last 54 seconds //距离上次输出的时间

-----------------

BACKGROUND THREAD

-----------------

srv\_master\_thread loops: 119 srv\_active, 0 srv\_shutdown, 8289617 srv\_idle//主线程每秒loop循环的次数(119激活的次数 0 停止的次数 8289617等待的次数)

srv\_master\_thread log flush and writes: 8289308//主线程日志刷新和写入的次数

----------

SEMAPHORES//信号量 事件计数器以及可选的当前等待线程的列表

----------

OS WAIT ARRAY INFO: reservation count 2361

OS WAIT ARRAY INFO: signal count 14317

RW-shared spins 0, rounds 40369, OS waits 704

RW-excl spins 0, rounds 151995, OS waits 265

RW-sx spins 10838, rounds 174759, OS waits 1128

Spin rounds per wait: 40369.00 RW-shared, 151995.00 RW-excl, 16.12 RW-sx

------------------------

LATEST DETECTED DEADLOCK //最后一次死锁

------------------------

2020-06-29 20:31:29 0x7f8aa1542700

\*\*\* (1) TRANSACTION://事物1

TRANSACTION 1931, ACTIVE 8 sec fetching rows

mysql tables in use 2, locked 2。

LOCK WAIT 5 lock struct(s), heap size 1136, 3 row lock(s)

MySQL thread id 6, OS thread handle 140233389127424, query id 179 localhost root optimizing

update test1 set num1 = (select count(1) from test)

\*\*\* (1) WAITING FOR THIS LOCK TO BE GRANTED://等待哪个表的什么类型的锁

RECORD LOCKS space id 27 page no 3 n bits 72 index PRIMARY of table `ecm`.`test` trx id 1931 lock mode S waiting

Record lock, heap no 3 PHYSICAL RECORD: n\_fields 4; compact format; info bits 0

0: len 4; hex 80000002; asc ;;

1: len 6; hex 00000000078a; asc ;;

2: len 7; hex 5e000001620110; asc ^ b ;;

3: len 4; hex 8000007b; asc {;;

\*\*\* (2) TRANSACTION://事物2

TRANSACTION 1930, ACTIVE 15 sec starting index read //这行显示次事务处于活跃状态已经188s，可能的所有状态有not started，active，prepared和committed in memory，一旦事务日志落盘了就会变成not started状态。在时间后面会显示出当前事务正在做什么（在这里为空没有显示出来）

mysql tables in use 1, locked 1

3 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1 //锁了几个结构

MySQL thread id 7, OS thread handle 140233388861184, query id 180 localhost root updating //哪个链接的那个账号

update test set num = 123 where id = 1 //具体哪条语句

\*\*\* (2) HOLDS THE LOCK(S): //持有哪些锁

RECORD LOCKS space id 27 page no 3 n bits 72 index PRIMARY of table `ecm`.`test` trx id 1930 lock\_mode X locks rec but not gap

Record lock, heap no 3 PHYSICAL RECORD: n\_fields 4; compact format; info bits 0

0: len 4; hex 80000002; asc ;;

1: len 6; hex 00000000078a; asc ;;

2: len 7; hex 5e000001620110; asc ^ b ;;

3: len 4; hex 8000007b; asc {;;

\*\*\* (2) WAITING FOR THIS LOCK TO BE GRANTED://等待哪些锁

RECORD LOCKS space id 27 page no 3 n bits 72 index PRIMARY of table `ecm`.`test` trx id 1930 lock\_mode X locks rec but not gap waiting

Record lock, heap no 2 PHYSICAL RECORD: n\_fields 4; compact format; info bits 0

0: len 4; hex 80000001; asc ;;

1: len 6; hex 000000000780; asc ;;

2: len 7; hex 590000015d0110; asc Y ] ;;

3: len 4; hex 8000007b; asc {;;

\*\*\* WE ROLL BACK TRANSACTION (2)//回滚了哪个事物

------------

TRANSACTIONS

------------

Trx id counter 22113

Purge done for trx's n:o < 22110 undo n:o < 0 state: running but idle

History list length 60

LIST OF TRANSACTIONS FOR EACH SESSION:

---TRANSACTION 421708541421392, not started

0 lock struct(s), heap size 1136, 0 row lock(s)

--------

FILE I/O //文件IO

--------

I/O thread 0 state: waiting for completed aio requests (insert buffer thread)

I/O thread 1 state: waiting for completed aio requests (log thread)

I/O thread 2 state: waiting for completed aio requests (read thread)

I/O thread 3 state: waiting for completed aio requests (read thread)

I/O thread 4 state: waiting for completed aio requests (read thread)

I/O thread 5 state: waiting for completed aio requests (read thread)

I/O thread 6 state: waiting for completed aio requests (write thread)

I/O thread 7 state: waiting for completed aio requests (write thread)

I/O thread 8 state: waiting for completed aio requests (write thread)

I/O thread 9 state: waiting for completed aio requests (write thread)

Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0, 0] ,

ibuf aio reads:, log i/o's:, sync i/o's:

Pending flushes (fsync) log: 0; buffer pool: 0

256 OS file reads, 22626 OS file writes, 21136 OS fsyncs

0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s

-------------------------------------

INSERT BUFFER AND ADAPTIVE HASH INDEX

-------------------------------------

Ibuf: size 1, free list len 0, seg size 2, 0 merges

merged operations:

insert 0, delete mark 0, delete 0

discarded operations:

insert 0, delete mark 0, delete 0

Hash table size 34679, node heap has 2 buffer(s)

Hash table size 34679, node heap has 3 buffer(s)

Hash table size 34679, node heap has 1 buffer(s)

Hash table size 34679, node heap has 2 buffer(s)

Hash table size 34679, node heap has 1 buffer(s)

Hash table size 34679, node heap has 1 buffer(s)

Hash table size 34679, node heap has 17 buffer(s)

Hash table size 34679, node heap has 2 buffer(s)

0.00 hash searches/s, 0.00 non-hash searches/s

---

LOG

---

Log sequence number 12663136

Log flushed up to 12663136

Pages flushed up to 12663136

Last checkpoint at 12663127

0 pending log flushes, 0 pending chkp writes

20542 log i/o's done, 0.00 log i/o's/second

----------------------

BUFFER POOL AND MEMORY

----------------------

Total large memory allocated 137428992

Dictionary memory allocated 472525

Buffer pool size 8192

Free buffers 7535

Database pages 628

Old database pages 211

Modified db pages 0

Pending reads 0

Pending writes: LRU 0, flush list 0, single page 0

Pages made young 0, not young 0

0.00 youngs/s, 0.00 non-youngs/s

Pages read 227, created 401, written 1685

0.00 reads/s, 0.00 creates/s, 0.00 writes/s

No buffer pool page gets since the last printout

Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s

LRU len: 628, unzip\_LRU len: 0

I/O sum[0]:cur[0], unzip sum[0]:cur[0]

--------------

ROW OPERATIONS

--------------

0 queries inside InnoDB, 0 queries in queue

0 read views open inside InnoDB

Process ID=1529, Main thread ID=140233264801536, state: sleeping

Number of rows inserted 31742, updated 10024, deleted 20564, read 78831

0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s

----------------------------

END OF INNODB MONITOR OUTPUT

根据日志查看数据库最近的一次死锁，查看事务持有的锁，和等待的锁，找出循环引用的表和sql语句。

1. **优化方向**

① 为查询缓存优化查询

② EXPLAIN 我们的SELECT查询(可以查看执行的行数)

③ 当只要一行数据时使用LIMIT 1（常用于判断是否存在）

④ 为搜索字段建立索引

⑤ 在Join表的时候使用相当类型的列，并将其索引

⑥ 千万不要 ORDER BY RAND ()

⑦ 避免SELECT \*

⑧ 永远为每张表设置一个ID

⑨ 可以使用ENUM 而不要VARCHAR

⑩ 尽可能的使用NOT NULL

⑪ 固定长度的表会更快

⑫ 垂直分割

⑬ 拆分打的DELETE或INSERT语句

⑭ 越小的列会越快

⑮ 选择正确的存储引擎

⑯ 小心 "永久链接"