MySql事务与锁

事务

1、事务简介

事务(Transaction) 是指作为单个逻辑工作单元执行的一系列操作。

2、事务的特性

A、原子性 (Atomicity)
表示组成一个事务的多个数据库操作是一个不可分隔的原子单元，只有所有的操作执行成功，整个事务才提交，事务中任何一个数据库操作失败，已经执行的任何操作都必须撤销，让数据库返回到初始状态。
B、一致性 (Consistency)
事务操作成功后，数据库所处的状态和它的业务规则是一致的，即数据不会被破坏。
C、隔离性 (Isolation)
在并发数据操作时，不同的事务拥有各自数据空间，它们的操作不会对对方产生干扰。数据库规定了多种事务隔离级别，不同隔离级别对应不同的干扰程度，隔离级别越高，数据一致性越好，但并发性越弱。
D、持久性 (Durabiliy)
一旦事务提交成功后，事务中所有的数据操作都必须被持久化到数据库中，即使提交事务后，数据库马上崩溃，在数据库重启时，也必须能保证能够通过某种机制恢复数据

锁

1.锁简介

数据库中的锁是指一种软件机制，用来控制防止某个用户（进程会话）在已经占用了某种数据资源时，其他用户做出影响本用户数据操作或导致数据非完整性和非一致性问题发生的手段

2. 锁的划分

(1)根据锁的级别划分可以划分为共享锁,排它锁
　　共享锁:针对同一条数据,多个读操作可以同时进行而不会互相影响.共享锁只针对update时候加锁,在未对update操作提交之前,其他事务只能获取最新的记录但不能够update操作
　　排它锁:当前的写操作没有完成前,阻断其他的写锁和读锁
(2)根据锁的粒度划分
　　行锁： 开销大，加锁慢，会出现死锁，锁定力度最小，发生锁冲突的概率最低，并发度高。
　　表所： 开销小，加锁快，不会出现死锁，锁定力度大，发生冲突所的概率高，并发度低
　　页面锁： 开销和加锁时间介于表锁和行锁之间，会出现死锁，锁定力度介于表和行行级锁之间，并发度一般

3.MySQL存储引擎和锁机制

MySQL的锁机制比较简单，最显著的特点是不同的存储引擎支持不同的锁机制
MyISAM和MEMORY存储引擎采用表级锁。
InnoDB支持行级锁、表级锁，默认情况采用行级锁。

4.表锁

表加锁的命令Lock Tables，给表解锁的命令Unlock Tables

```
#  ------------------------------添加表级读锁---------------------------------
#1.在一个数据库中创建两张表 t1,t2:
    在t1中插入两条记录
    # 在会话1中对t1 加锁  mysql> lock tables t1 read;
        加锁后只可以查询已经加锁的表t1,查询没有加锁的表t2时候会报错,而且当进行更新操作时候也会报错
        ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
        ERROR 1099 (HY000): Table 't1' was locked with a READ lock and can't be updated
    #另起一个会话2(另一个进程),对已经加锁的表t1进行查询,成功,但是当要进行更新操作会holding 即出现锁等待
        此时将t1解锁,会话2中的更新操作会成功，unlock tables;
        Query OK, 1 row affected (13 min 22.04 sec)

# ----------- 结论 ---------------
#1.READ锁是共享锁，不影响其他会话(进程)的读取，但不能更新已经加READ锁的数据。
#2.MyISAM表的读写是串行的，但是总体而言的，在一定条件下，MyISAM表也支持查询和插入操作的并发进行
    使用local，Local参数允许在表尾并发插入，只锁定表中当前记录，其他会话(进程)可以插入新的记录，但当前会话不能读取新的记录
```

```
#  ------------------------------Local 设置表锁的并发性---------------------------------
MyISAM存储引擎有一个系统变量concurrent_insert，用以控制其并发插入的行为，其值分别可以为0、1或2。
#0：不允许并发操作
    设置concurrent_insert为0
    在会话1对表t1加锁，lock tables t1 read local;
    在会话2插入一条记录，此时t1表被锁定，进入等待
    在会话1解锁表t1，此时会话2插入成功 unlock tables;

#1：如果MyISAM表中没有空洞（即表的中间没有被删除的行），MyISAM允许在一个进程读表的同时，另一个进程从表尾插入记录，是MySQL的默认设置。
        设置concurrent_insert为1
        在会话1删除ID为2的记录
            在会话1对表t1加锁，lock tables t1 read local;
            在会话2插入一条记录，此时t1表被锁定，并且表中有空洞，进入等待
            在会话1解锁表t1，此时会话2插入成功，此时表中已经没有空洞
        在会话1对表t1加锁
        在会话2插入一条记录，插入成功，支持有条件并发插入，不会等待

 #2：无论MyISAM表中有没有空洞，都允许在表尾并发插入记录。在MySQL配置文件添加，concurrent_insert=2，重启mySQL服务设置生效。
        设置concurrent_insert为2
        在会话1删除ID为5的记录，创造一个空洞
        在会话1对表t1加锁
        在会话2插入一条记录，插入成功，支持无条件并发插入
```

```
#  ------------------------------添加表级写锁---------------------------------
添加表级写锁语法如下：
LOCK TABLES tablename WRITE;
不允许其他会话查询、修改、插入记录。
```

5.行锁

行锁简介

InnoDB存储引擎实现的是基于多版本的并发控制协议--MVCC (Multi-Version Concurrency Control)。

MVCC的优点是读不加锁，读写不冲突。在读多写少的OLTP应用中，读写不冲突是非常重要的，极大的增加了系统的并发性能。

在MVCC并发控制中，读操作可以分成两类：快照读 (snapshot read)与当前读 (current read)

快照度：读取的是记录的可见版本 (有可能是历史版本)，不用加锁。

当前读：读取的是记录的最新版本，并且当前读返回的记录都会加上锁，保证其他事务不会再并发修改。事务加锁，是针对所操作的行，对其他行不进行加锁处理。

基于不同对的sql语句对读操作分类

快照读：简单的select 操作,属于快照读,不加锁

```
select * from table where ?;
```

当前读：特殊的读操作,insert,update,delete,属于当前读,需要加锁

```
select * from table where ? lock in share mode;
select * from table where ? for update;
insert into table values (…);
update table set ? where ?;
delete from table where ?;
```

以上sql操作属于当前读,读取记录的最新版本,并且读取之后,还需要保证其他并发事务不能修改当前记录，对读取记录加锁。除了第一条语句，对读取记录加S锁 (共享锁)外，其他的操作，都加的是X锁 (排它锁)。

```
#  ----------------------快照读 ------------------------
打开会话1 创建表和数据  (使用innoDB引擎)
    #在会话1中开启事务,并查询t3表中id =1 的数据 start transaction;
        在会话2中将t3表更新id =1 的age为100,并查询数据内容，发现数据已经修改
        在会话1中查看id为1的数据信息，发现此时age = 19，并未修改
    #但是当会话1中提交事务后 再次查询时候此时数据已进行了更改 commit，数据已进行了修改
```

```
# ---------------------- 当前读 -------------------------
#会话1开启事务,并且给select语句添加共享锁
    start transaction;
    Query OK, 0 rows affected (0.00 sec)
    mysql> select * from t3 where id =1 lock in share mode
    #在会话2中更新id = 1 的数据,此时会进入锁等待
    mysql> update t3 set age=66 where id =1;

#当会话1提交事务后,会话2的更新操作成功
    会话1 commit
    会话2 Query OK, 1 row affected (11.54 sec)
```

```
# ---------------------- 验证事务给记录加锁 -------------------------
1.会话1 开启事务，在会话1中更新 id = 1 的数据
    mysql> start transaction;
    mysql> update t3 set age=300 where id=1;

2.在会话2 开启事务 ,更新id为2 的数据 ,此时会进入锁等待
    mysql> start transaction;
    mysql> update t3 set age=200 where id=2;

#由于 t3 表没有指定主键,事务不支持行级锁。会话1的事务给整张表加了锁
```

```
在会话1提交事务,此时会话2的修改成功   ,在会话2提交事务,解除对表的锁定

# ------------------------------------------------------------------
1.在会话1 给表的 id增加主键
    alter table t3 add primary key(id);
2.在会话1 开启事务,并且更新id 为1 的数据
    mysql> start transaction;
    Query OK, 0 rows affected (0.00 sec)

    mysql> update t3 set age=300 where id=1;
    Query OK, 1 row affected (0.01 sec)

3.在会话2上开启事务,并修改id为2的的数据,更新成功   ,说明会话1只锁定了ID为1的行
    mysql> start transaction;
    mysql> update t3 set age=400 where id=2;
    Query OK, 1 row affected (0.00 sec)
  #但是在会话2 中更新id = 1 的值,出现了等待,因为会话1给ID为1的行添加了独占锁
```

```
# ------------------死锁的产生-------------------------
#A事务添加共享锁后，B事务也可以添加共享锁。A事务update锁定记录，处于等待中，于此同时B事务也update更新锁定的记录，就产生死锁

会话1 开启事务,查询id=1 的数据,并添加共享锁
    mysql> start transaction;
    Query OK, 0 rows affected (0.00 sec)
    mysql> select * from t3 where id =1 lock in share mode;

    同样,在会话2中 开启事务,查询id=1 的数据,并添加共享锁
    mysql> start transaction;
    Query OK, 0 rows affected (0.00 sec)
    mysql> select * from t3 where id =1 lock in share mode;

在会话1 中更新 id=1 的数据,等待会话2释放共享锁
    mysql> update t3 set age =199 where id =1;
在会话2更新ID为1的age为，会话2发现死锁，回滚事务。
    mysql> update t3 set age =199 where id =1;
在会话1提交事务
commit
```

6.在普通索引中，是否引发表锁取决于普通索引的高效程度。

常用的索引有三类：主键、唯一索引、普通索引。

- **主键** 不由分说，自带最高效的索引属性；
- **唯一索引**指的是该属性值重复率为0，一般可作为业务主键，例如学号；
- **普通索引** 与前者不同的是，属性值的重复率大于0，不能作为唯一指定条件，例如学生姓名。接下来我要说明是“普通索引对并发的影响”。

1.结论：

- 当 Where 查询条件中的字段没有索引时，更新操作会锁住全表！
- 可以看到，在有索引的情况下，更新不同的行，InnoDB 默认的行锁不会阻塞。

- Where 条件中的查询字段虽然有索引，但是索引失效时（本例子中是字符串没有加单引号），InnoDB 默认的行锁更新操作变为表锁。
- **当“值重复率”低时，甚至接近主键或者唯一索引的效果，“普通索引”依然是行锁；当“值重复率”高时，MySQL 不会把这个“普通索引”当做索引，即造成了一个没有索引的 SQL，此时引发表锁。**