一.

二. 死锁

1 死锁是怎么被发现的?

1.1 死锁成因&&检测方法

与java中死锁的成因一样,都是互相等待资源造成,

innodb是怎么探知死锁的?

直观方法是在两个事务相互等待时，当一个等待时间超过设置的某一阀值时，对其中一个事务进行回滚，另一个事务就能继续执行。这种方法简单有效，在innodb中，参数innodb_lock_wait_timeout用来设置超时时间。

仅用上述方法来检测死锁太过被动，innodb还提供了wait-for graph算法来主动进行死锁检测，每当加锁请求无法立即满足需要并进入等待时，wait-for graph算法都会被触

发

1.2 wait-for graph原理

他们相互等待对方的资源，而且形成环路！我们将每辆车看为一个节点，当节点1需要等待节点2的资源时，就生成一条有向边指向节点2，最后形成一个有向图。我们只要检测这个有向图是否出现环路即可，出现环路就是死锁！这就是wait-for graph算法

2 innodb隔离级别、索引与锁

3 死锁成因

3.1不同表相同记录行锁冲突

这种情况很好理解，事务A和事务B操作两张表，但出现循环等待锁情况

3.2相同表记录行锁冲突

这种情况比较常见，之前遇到两个job在执行数据批量更新时，jobA处理的的id列表为[1,2,3,4]，而job处理的id列表为[8,9,10,4,2]，这样就造成了死锁。

3.3不同索引锁冲突

这种情况比较隐晦，事务A在执行时，除了在二级索引加锁外，还会在聚簇索引上加锁，在聚簇索引上加锁的顺序是[1,4,2,3,5]，而事务B执行时，只在聚簇索引上加锁，加锁顺序是[1,2,3,4,5]，这样就造成了死锁的可能性

3.4 gap锁(间隙锁)冲突

innodb在RR级别下，如下的情况也会产生死锁，比较隐晦。不清楚的同学可以自行根据上节的gap锁原理分析下。

4 如何尽可能避免死锁

1）以固定的顺序访问表和行。比如对第2节两个job批量更新的情形，简单方法是对id列表先排序，后执行，这样就避免了交叉等待锁的情形；又比如对于3.1节的情形，将两个事务的sql顺序调整为一致，也能避免死锁。

2）大事务拆小。大事务更倾向于死锁，如果业务允许，将大事务拆小。

3）在同一个事务中，尽可能做到一次锁定所需要的所有资源，减少死锁概率。

4）降低隔离级别。如果业务允许，将隔离级别调低也是较好的选择，比如将隔离级别从RR调整为RC，可以避免掉很多因为gap锁造成的死锁。

5）为表添加合理的索引。可以看到如果不走索引将会为表的每一行记录添加上锁，死锁的概率大大增大。

5.怎么解决死锁

1. 通过应用业务日志定位到问题代码，找到相应的事务对应的sql；
2. 查看数据库死锁相关日志,确定死锁情况；
3. 找到死锁的进程,手动kill一个或多个进程(破坏环路)

一般来说,数据库会帮我们检测死锁并破坏死锁情况