## 一．事务隔离&MVCC

1. 事务的概念是什么?事务的特性是什么？

事务的概念：保证一组数据库操作，要么全部成功，要么全部失败。

事务的特性：原子性、一致性、隔离性、持久性。

2. 多事务同时执行的时候，可能会出现的问题：脏读、不可重复读、幻读。

脏读：事务1执行修改操作，还未提交，事务2读到事务1的修改后的值进行后续操作，此时事务1进行回滚，事务2读到的其实是脏数据。

不可重复读：事务1读表t的所有数据后，事务2修改了表t中的若干行数据并提交，事务1再次读表t的数据，发现和第一次读的时候不一致。

幻读：事务1读表t的所有数据后，事务2向表t中插入了若干行数据并提交，事务1再次读表t的数据，发现和第一次读的时候相比多了若干行。

3. mysql的事务隔离级别读未提交, 读已提交, 可重复读, 串行各是什么意思?

读未提交：一个事务还未提交，它所做的变更就可以被别的事务看到

读提交：一个事务提交之后，它所做的变更才可以被别的事务看到

可重复读：一个事务执行过程中看到的数据和该事务启动时看到的数据是一致的。

串行化：对应一个记录会加读写锁，出现冲突的时候，后访问的事务必须等前一个事务执行完成才能继续执行

4. 读已提交, 可重复读是怎么通过视图构建实现的?

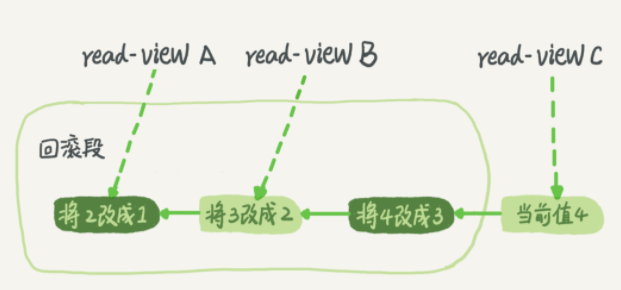
在“可重复读”隔离级别下，这个视图是在事务启动时创建的，整个事务存在期间都用这个视图；在“读提交”隔离级别下，这个视图是在每个 SQL 语句开始执行的时候创建的；“读未提交”隔离级别下直接返回记录上的最新值，没有视图概念；而“串行化”隔离级别下直接用加锁的方式来避免并行访问。

5. 可重复读的使用场景举例? 对账的时候应该很有用?

假设银行有一项业务是对账，一张表存的是每个月月底的余额（如果没到月底就用最新的余额作为当月余额），另一张表存的是账单明细，判断每个月月底的余额之差和账单明细是否能对得上。假设计算过程中有的账单来临，也不会影响正在进行的校对过程。

6. 事务隔离是怎么通过read-view(读视图)实现的?

在 MySQL 中，实际上每条记录在更新的时候都会同时记录一条回滚操作。记录上的最新值，通过回滚操作，都可以得到前一个状态的值。假设一个值从 1 被按顺序改成了 2、3、4，在回滚日志里面就会有类似下面的记录。



当前值是 4，但是在查询这条记录的时候，不同时刻启动的事务会有不同的 read-view。如图中看到的，在视图 A、B、C 里面，这一个记录的值分别是 1、2、4，同一条记录在系统中可以存在多个版本，就是数据库的多版本并发控制（MVCC）。对于 read-view A，要得到 1，就必须将当前值依次执行图中所有的回滚操作得到。

7. 并发版本控制(MCVV)的概念是什么, 是怎么实现的?

概念：在事务中保存数据的快照，这意味着在一个事物里能够看到数据一致的视图，而不用担心这个事务运行多长时间，同时也意味着在同一个时刻不同事务看到的相同表里的数据可能是不同的。

实现方式：在mysql中，每条记录都会记录更新时会同时记录一条回滚操作，通过回滚操作，可以得到前一个状态的值。具体参考上一个问题。

8. 使用长事务的弊病? 为什么使用常事务可能拖垮整个库?

长事务意味着系统里面会存在很老的事务视图。由于这些事务随时可能访问数据库里面的任何数据，所以这个事务提交之前，数据库里面它可能用到的回滚记录都必须保留，这就会导致大量占用存储空间。

9. 事务的启动方式有哪几种?

一、显式启动事务语句，begin或者start transaction,提交commit，回滚rollback；二、set autocommit=0，该命令会把这个线程的自动提交关掉。这样只要执行一个select语句，事务就启动，并不会自动提交，直到主动执行commit或rollback或断开连接。

二．索引

1.索引的常见模型：哈希表、有序数组、搜索树

三种模型的比较

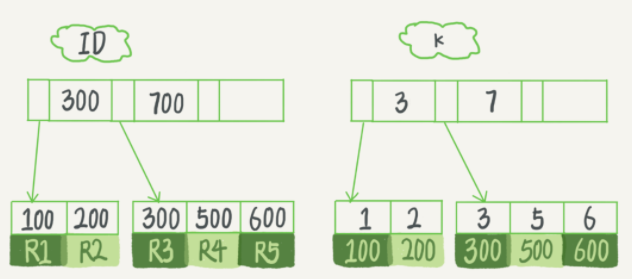
|  |  |  |  |
| --- | --- | --- | --- |
|  | 优点 | 缺点 | 适用场景 |
| 哈希表 | 新增和等值查询很快 | 范围查询慢 | 等值查询 |
| 有序数组 | 等值和范围查询都很快 | 新增效率慢 | 静态存储引擎（不大会修改数据） |
| 搜索树 | 兼顾了新增和查询的速度 |  | 大部分场景 |

2.InnoDB的索引模型（因为该引擎最常用）

InnoDB 使用了 B+ 树索引模型，所以数据都是存储在 B+ 树中的。

假设，我们有一个主键列为 ID 的表，表中有字段 k，并且在 k 上有索引。

表中 R1~R5 的 (ID,k) 值分别为 (100,1)、(200,2)、(300,3)、(500,5) 和 (600,6)，两棵树的示例示意图如下。



从图中不难看出，根据叶子节点的内容，索引类型分为主键索引和非主键索引。

主键索引的叶子节点存的是整行数据。在 InnoDB 里，主键索引也被称为聚簇索引（clustered index）。

非主键索引的叶子节点内容是主键的值。在 InnoDB 里，非主键索引也被称为二级索引（secondary index）。

**基于主键索引和普通索引的查询有什么区别？**

如果语句是 select \* from T where ID=500，即主键查询方式，则只需要搜索 ID 这棵 B+ 树；如果语句是 select \* from T where k=5，即普通索引查询方式，则需要先搜索 k 索引树，得到 ID 的值为 500，再到 ID 索引树搜索一次。这个过程称为回表。

也就是说，基于非主键索引的查询需要多扫描一棵索引树。因此，我们在应用中应该尽量使用主键查询。