InnoDB的事务分析

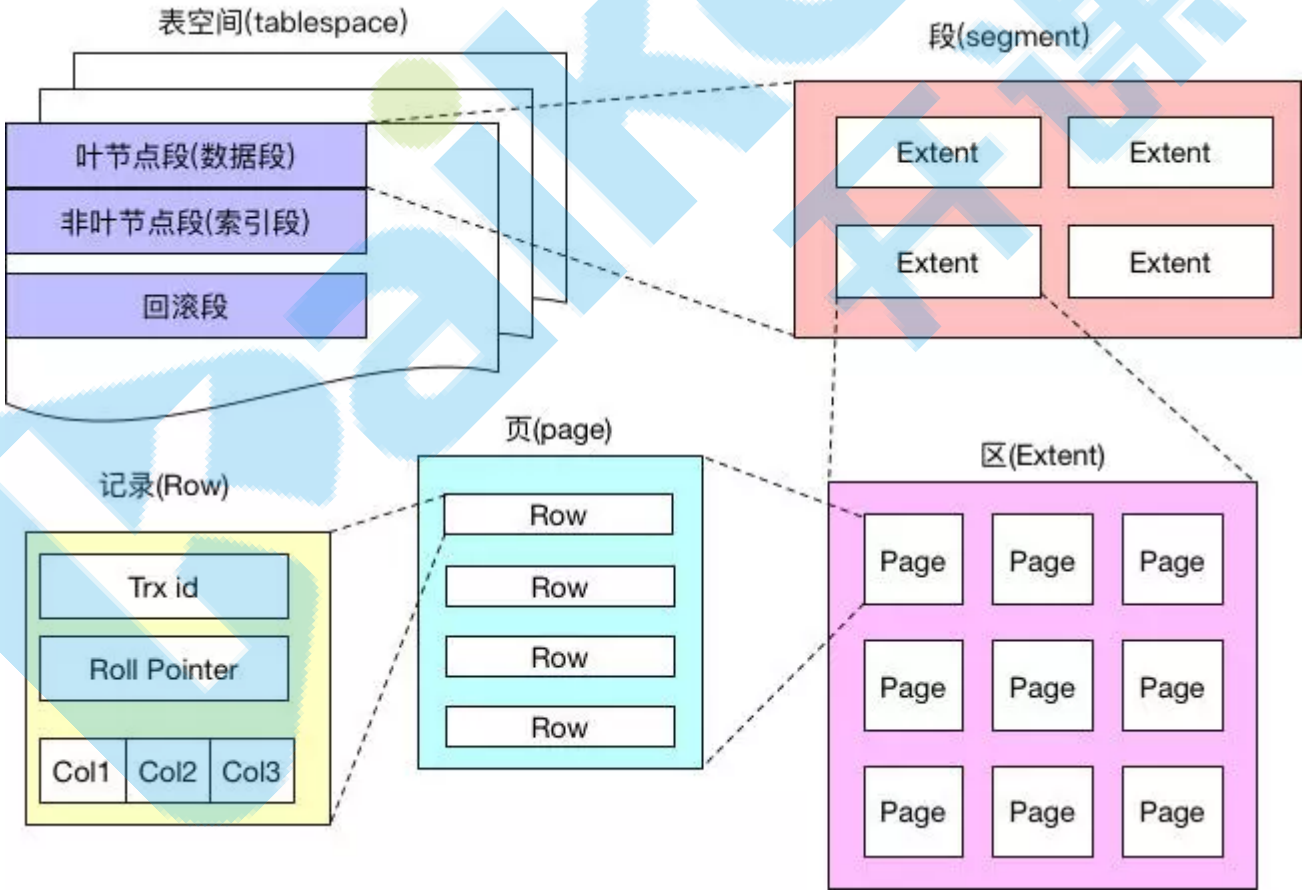
原子性，持久性和一致性

原子性，持久性和一致性主要是通过redo log、undo log和Force Log at Commit机制机制来完成的。redo log用于在崩溃时恢复数据，undo log用于对事务的影响进行撤销，也可以用于多版本控制。而Force Log at Commit机制保证事务提交后redo log日志都已经持久化。

开启一个事务后，用户可以使用COMMIT来提交，也可以用ROLLBACK来回滚。其中COMMIT或者ROLLBACK执行成功之后，数据一定是会被全部保存或者全部回滚到最初状态的，这也体现了事务的原子性。但是也会有很多的异常情况，比如说事务执行中途连接断开，或者是执行COMMIT或者ROLLBACK时发生错误，Server Crash等，此时数据库会自动进行回滚或者重启之后进行恢复。

我们先来看一下redo log的原理，redo log顾名思义，就是重做日志，每次数据库的SQL操作导致的数据变化它都会记录一下，**具体来说，redo log是物理日志，记录的是数据库页的物理修改操作。**如果数据发生了丢失，数据库可以根据redo log进行数据恢复。

在事务执行的过程中，除了记录redo log，还会记录一定量的undo log。undo log记录了数据在每个操作前的状态，如果事务执行过程中需要回滚，就可以根据undo log进行回滚操作。



数据和回滚日志的逻辑存储结构.jpg

undo log的存储不同于redo log，它存放在数据库内部的一个特殊的段(segment)中，这个段称为回滚段。回滚段位于共享表空间中。undo段中的以undo page为更小的组织单位。**undo page和存储数据库数据和索引的页类似。因为redo log是物理日志，记录的是数据库页的物理修改操作。所以undo log（也看成数据库数据）的写入也会产生redo log，也就是undo log的产生会伴随着redo log的产生，这是因为undo log也需要持久性的保护。**如上图所

表空间中有回滚段和叶节点段和非叶节点段，而三者都有对应的页结构。

隔离性

事务并发问题

在事务的**并发操作**中可能会出现一些问题：

- **丢失更新**：两个事务针对同一数据都发生修改操作时，会存在丢失更新问题。
- **脏读**：一个事务读取到另一个事务未提交的数据。
- **不可重复读**：一个事务因读取到另一个事务**已提交的update或者delete数据**。导致对同一条记录读取两次以上的结果不一致。
- **幻读**：一个事务因读取到另一个事务**已提交的insert数据**。导致*对同一张表读取*两次以上的结果不一致。

事务隔离级别

- **四种隔离级别（SQL92标准）：**

  现在来看看MySQL数据库为我们提供的四种隔离级别（*由低到高*）：

  ① Read uncommitted (读未提交)：最低级别，任何情况都无法保证。

  ② Read committed (RC，读已提交)：可避免脏读的发生。

  ③ **Repeatable read** (RR，可重复读)：可避免脏读、不可重复读的发生。

  （**注意事项：InnoDB的RR还可以解决幻读，主要原因是Next-Key锁，只有RR才能使用Next-Key锁**）

  ④ Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

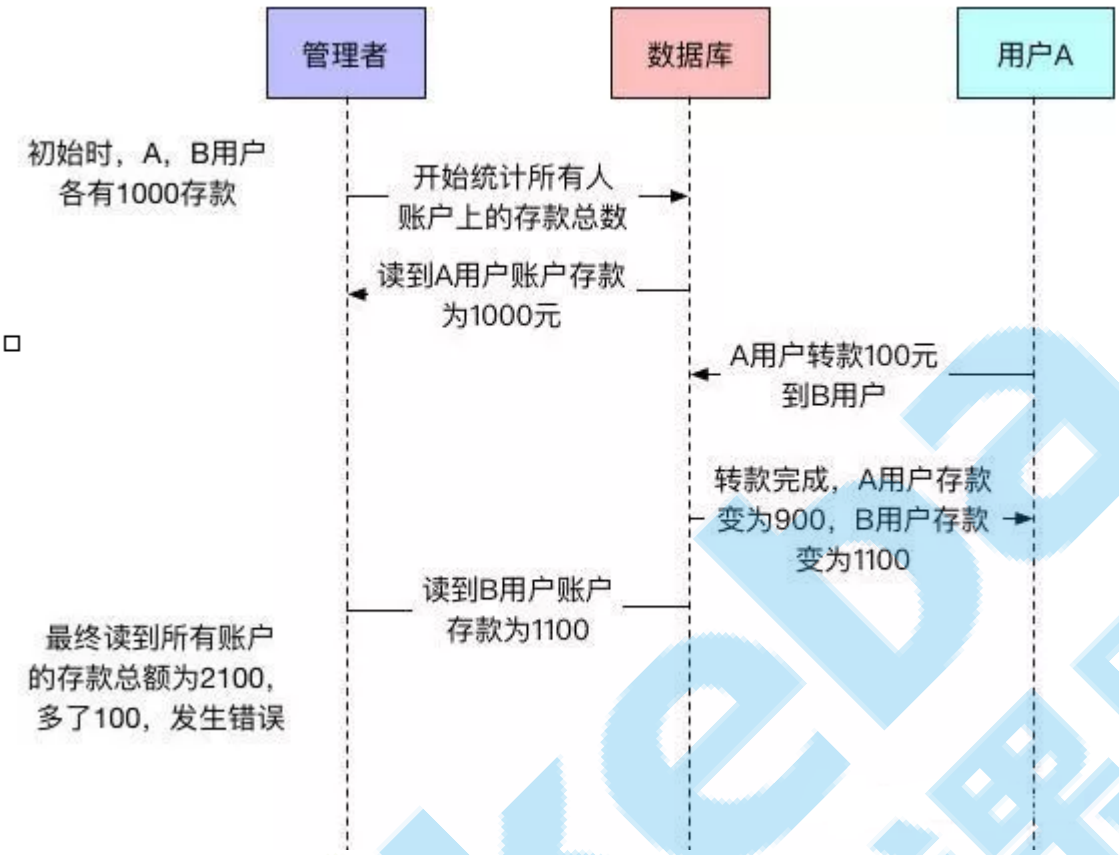
  （**由MVCC降级为Locking-Base CC**）

事务隔离级别	脏读	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
不可重复读、读已提交（read-committed）	否	是	是
可重复读（repeatable-read）	否	否	是
串行化（serializable）	否	否	否

考虑一个现实场景：**

管理者要查询所有用户的存款总额，假设除了用户A和用户B之外，其他用户的存款总额都为0，A、B用户各有存款1000，所以所有用户的存款总额为2000。但是在查询过程中，用户A会向用户B进行转账操作。转账操作和查询总额操作的时序图如下图所示。

转账和查询的时序图：



如果没有任何的并发控制机制，查询总额事务先读取了用户A的账户存款，然后转账事务改变了用户A和用户B的账户存款，最后查询总额事务继续读取了转账后的用户B的账号存款，导致最终统计的存款总额多了100元，发生错误。

InnoDB的MVCC实现

我们首先来看一下wiki上对MVCC的定义：

> **Multiversion concurrency control (MCC or MVCC)**, is a concurrency control method commonly used by database management systems to provide concurrent access to the database and in programming languages to implement transactional memory.

由定义可知，MVCC是用于数据库提供并发访问控制的并发控制技术。与MVCC相对的，是基于锁的并发控制，`Lock-Based Concurrency Control`。MVCC最大的好处，相信也是耳熟能详：**读不加锁，写依然加锁，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，这也是为什么现阶段，几乎所有的RDBMS，都支持了MVCC。

多版本并发控制仅仅是一种技术概念，并没有统一的实现标准， 其**核心理念就是数据快照，不同的事务访问不同版本的数据快照，从而实现不同的事务隔离级别。**虽然字面上是说具有多个版本的数据快照，但这并不意味着数据库必须拷贝数据，保存多份数据文件，这样会浪费大量的存储空间。InnoDB通过事务的undo日志巧妙地实现了多版本的数据快照。MVCC 在mysql 中的实现依赖的是 undo log 与 read view 。

一致性非锁定读

一致性非锁定读(consistent nonlocking read)是指InnoDB存储引擎通过多版本控制(MVCC)读取当前数据库中行数据的方式。如果读取的行正在执行DELETE或UPDATE操作，这时读取操作不会因此去等待行上锁的释放。相反地，InnoDB会去读取行的一个最新可见快照。

行锁原理分析

一条简单SQL的加锁分析

在介绍完一些背景知识之后，接下来将选择几个有代表性的例子，来详细分析MySQL的加锁处理。当然，还是从最简单的例子说起。经常有朋友发给我一个SQL，然后问我，这个SQL加什么锁？就如同下面两条简单的SQL，他们加什么锁？

- SQL1：

```
select * from t1 where id = 10;
```

- SQL2：

```
delete from t1 where id = 10;
```

针对这个问题，该怎么回答？能想象到的一个答案是：

- SQL1：**不加锁。**因为MySQL是使用多版本并发控制的，读不加锁。
- SQL2：对id = 10的记录**加写锁** (走主键索引)。

这个答案对吗？说不上来。即可能是正确的，也有可能是错误的，已知条件不足，这个问题没有答案。必须还要知道以下的一些前提，前提不同，能给出的答案也就不同。要回答这个问题，还缺少哪些前提条件？

- **前提一：**id列是不是主键？
- **前提二：**当前系统的隔离级别是什么？
- **前提三：**id列如果不是主键，那么id列上有索引吗？
- **前提四：**id列上如果有二级索引，那么这个索引是唯一索引吗？
- **前提五：**两个SQL的执行计划是什么？索引扫描？全表扫描？

没有这些前提，直接就给定一条SQL，然后问这个SQL会加什么锁，都是很业余的表现。而当这些问题有了明确的答案之后，给定的SQL会加什么锁，也就一目了然。下面，我们将这些问题的答案进行组合，然后按照从易到难的顺序，逐个分析每种组合下，对应的SQL会加哪些锁？

注：下面的这些组合，需要做一个前提假设，也就是有索引时，执行计划一定会选择使用索引进行过滤 (索引扫描)。但实际情况会复杂很多，真正的执行计划，还是需要根据MySQL输出的为准。

```
组合一：id列是主键，RC隔离级别

组合二：id列是二级唯一索引，RC隔离级别

组合三：id列是二级非唯一索引，RC隔离级别

组合四：id列上没有索引，RC隔离级别

组合五：id列是主键，RR隔离级别

组合六：id列是二级唯一索引，RR隔离级别

组合七：id列是二级非唯一索引，RR隔离级别

组合八：id列上没有索引，RR隔离级别

组合九：Serializable隔离级别
```

if列组合还没有列举完全，但是看起来，已经很多了。真的有必要这么复杂吗？事实上，要分析加锁，就是需要这么复杂。但是从另一个角度来说，只要你选定了一种组合，SQL需要加哪些锁，其实也就确定了。接下来，就让我们来逐个分析这9种组合下的SQL加锁策略。

扩展点

要做到完全掌握MySQL/InnoDB的加锁规则，甚至是其他任何数据库的加锁规则，需要具备以下的一些知识点：

- 了解数据库的一些基本理论知识：数据的存储格式 (堆组织表 vs 聚簇索引表)；并发控制协议 (MVCC vs Lock-Based CC)；Two-Phase Locking；数据库的隔离级别定义 (Isolation Level)；
- 了解SQL本身的执行计划 (主键扫描 vs 唯一键扫描 vs 范围扫描 vs 全表扫描)；
- 了解数据库本身的一些实现细节 (过滤条件提取；Index Condition Pushdown；Semi-Consistent Read)；
- 了解死锁产生的原因及分析的方法 (加锁顺序不一致；分析每个SQL的加锁顺序)