MVCC实现不同隔离级别

之前说到ReadView的机制只在Read Committed和Repeatable Read隔离级别下生效，所以只有这两种隔离级别才有MVCC。在Read Committed隔离级别下，每次读取数据时都会生成ReadView；而在Repeatable Read隔离级别下只会在事务首次读取数据时生成ReadView，之后的读操作都会沿用此ReadView。

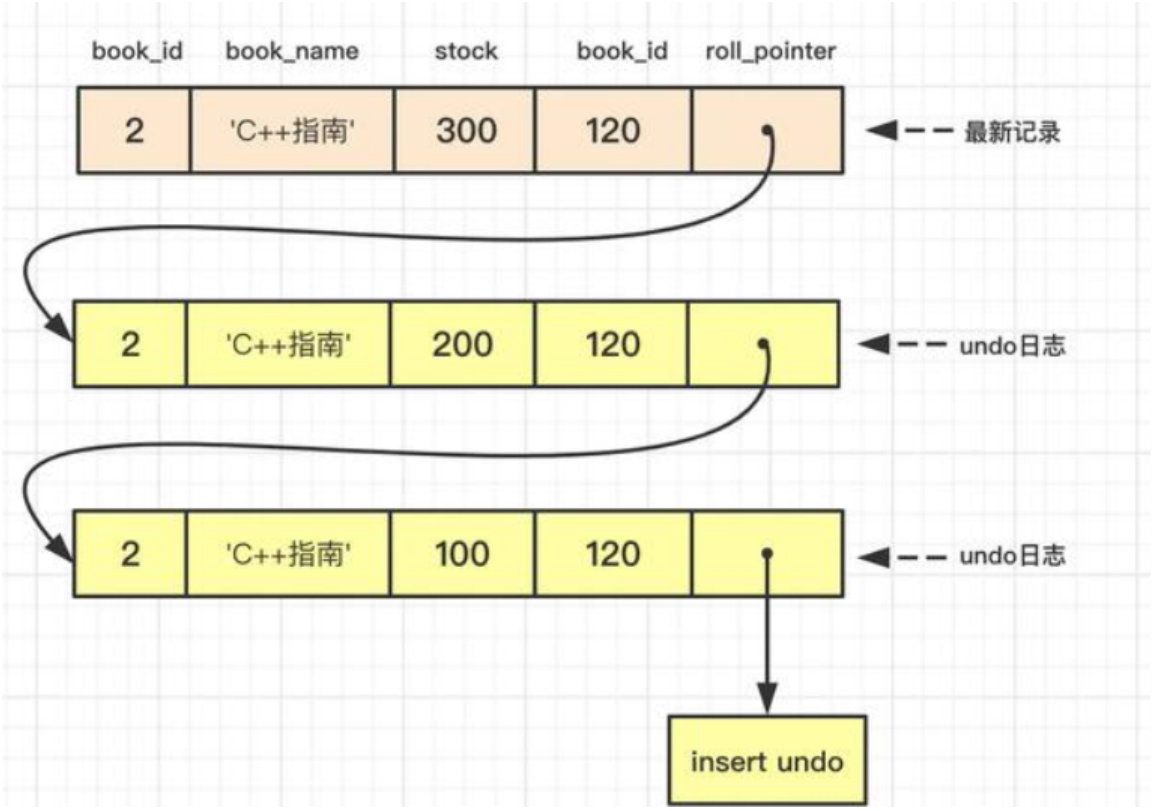
下面我们通过例子来看看Read Committed和Repeatable Read隔离级别下MVCC的不同表现。我们继续以表book为例进行演示。

Read Committed隔离级别下MVCC工作原理

假设在Read Committed隔离级别下，有如下事务在执行，事务id为10：

```
BEGIN; // 开启Transaction 10

UPDATE book SET stock = 200 WHERE id = 2;

UPDATE book SET stock = 300 WHERE id = 2;
12345
```

此时该事务尚未提交，id为2的记录版本链如下图所示：



浅谈MySQL并发控制：隔离级别、锁与MVCC
然后我们开启一个事务对id为2的记录进行查询：

```
BEGIN;
```

当执行SELECT语句时会生成一个ReadView，该ReadView中的m_ids为[10]，min_trx_id为10，max_trx_id为11，creator_trx_id为0（因为事务中当执行写操作时才会分配一个单独的事务id，否则事务id为0）。按照我们之前所述ReadView的工作原理，我们查询到的版本记录为

```
+----------+-----------+-------+
| book_id  | book_name | stock |
+----------+-----------+-------+
| 2        | C++指南    |  100  |
+----------+-----------+-------+
12345
```

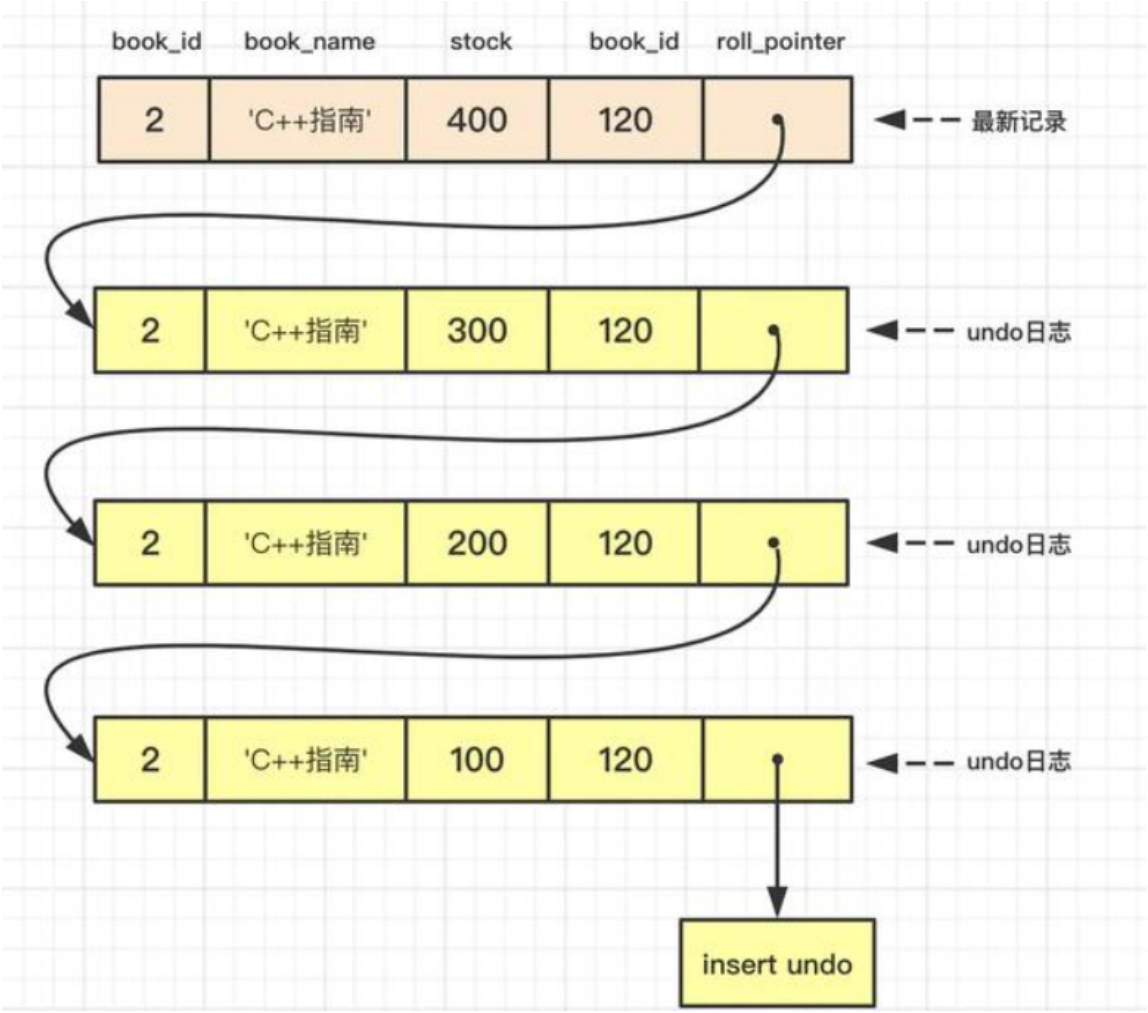
然后我们将事务id为10的事务提交：

```
BEGIN; // 开启Transaction 10

UPDATE book SET stock = 200 WHERE id = 2;

UPDATE book SET stock = 300 WHERE id = 2;

COMMIT;
1234567
```

同时开启执行另一事务id为11的事务，但不提交：

```
BEGIN; // 开启Transaction 11

UPDATE book SET stock = 400 WHERE id = 2;
123
```

此时id为2的记录版本链如下图所示：



浅谈MySQL并发控制：隔离级别、锁与MVCC
然后我们回到刚才的查询事务中再次查询id为2的记录：

```
BEGIN;

SELECT * FROM book WHERE id = 2; // 此时Transaction 10 未提交

SELECT * FROM book WHERE id = 2; // 此时Transaction 10 已提交
```

当第二次执行SELECT语句时会再次生成一个ReadView，该ReadView中的m_ids为[11]，min_trx_id为11，max_trx_id为12，creator_trx_id为0。按照ReadView的工作原理进行分析，我们查询到的版本记录为

```
+----------+-----------+-------+
| book_id  | book_name | stock |
+----------+-----------+-------+
| 2        | C++指南     | 300   |
+----------+-----------+-------+
```

从上述分析可以发现，因为每次执行查询语句都会生成新的ReadView，所以在Read Committed隔离级别下的事务读取到的是查询时刻表中已提交事务修改之后的数据。

Repeatable Read隔离级别下MVCC工作原理

我们在Repeatable Read隔离级别下重复上面的事务操作：

```
BEGIN; // 开启Transaction 20

UPDATE book SET stock = 200 WHERE id = 2;

UPDATE book SET stock = 300 WHERE id = 2;
```

此时该事务尚未提交，然后我们开启一个事务对id为2的记录进行查询：

```
BEGIN;

SELECT * FROM book WHERE id = 2;
```

当事务第一次执行SELECT语句时会生成一个ReadView，该ReadView中的m_ids为[20]，min_trx_id为20，max_trx_id为21，creator_trx_id为0。根据ReadView的工作原理，我们查询到的版本记录为

```
+----------+-----------+-------+
| book_id  | book_name | stock |
+----------+-----------+-------+
| 2        | C++指南    |  100  |
+----------+-----------+-------+
```

然后我们将事务id为20的事务提交：

```
BEGIN; // 开启Transaction 20

UPDATE book SET stock = 200 WHERE id = 2;

UPDATE book SET stock = 300 WHERE id = 2;

COMMIT;
```

同时开启执行另一事务id为21的事务，但不提交：

```
BEGIN; // 开启Transaction 21

UPDATE book SET stock = 400 WHERE id = 2;
```

然后我们回到刚才的查询事务中再次查询id为2的记录：

```
BEGIN;

SELECT * FROM book WHERE id = 2; // 此时Transaction 10 未提交

SELECT * FROM book WHERE id = 2; // 此时Transaction 10 已提交
```

当第二次执行SELECT语句时不会生成新的ReadView，依然会使用第一次查询时生成ReadView。因此我们查询到的版本记录跟第一次查询到的结果是一样的：

```
+----------+-----------+-------+
| book_id  | book_name | stock |
+----------+-----------+-------+
| 2        | C++指南    |  100  |
+----------+-----------+-------+
```

从上述分析可以发现，因为在Repeatable Read隔离级别下的事务只会在第一次执行查询时生成ReadView，该事务中后续的查询操作都会沿用这个ReadView，因此此隔离级别下一个事务中多次执行同样的查询，其结果都是一样的，这样就实现了可重复读。