15丨初识事务隔离：隔离的级别有哪些，它们都解决了哪些异常问题？

2019-07-15 陈旸

上一篇文章中，我们讲到了事务的四大特性ACID，分别是原子性、一致性、隔离性和持久性，其中隔离性是事务的基本特性之一，它可以防止数据库在并发处理时出现数据不一致的情况。最严格的情况下，我们可以采用串行化的方式来执行每一个事务，这就意味着事务之间是相互独立的，不存在并发的情况。然而在实际生产环境下，考虑到随着用户量的增多，会存在大规模并发访问的情况，这就要求数据库有更高的吞吐能力，这个时候串行化的方式就无法满足数据库高并发访问的需求，我们还需要降低数据库的隔离标准，来换取事务之间的并发能力。

有时候我们需要牺牲一定的正确性来换取效率的提升，也就是说，我们需要通过设置不同的隔离等级，以便在正确性和效率之间进行平衡。同时，随着RDBMS种类和应用场景的增多，数据库的设计者需要统一对数据库隔离级别进行定义，说明这些隔离标准都解决了哪些问题。

我们今天主要讲解事务的异常以及隔离级别都有哪些，如果你已经对它们有所了解，可以跳过本次章节，当然你也可以通过今天的课程快速复习一遍：

1. 事务并发处理可能存在的三种异常有哪些？什么是脏读、不可重复读和幻读？
2. 针对可能存在的异常情况，四种事务隔离的级别分别是什么？
3. 如何使用MySQL客户端来模拟脏读、不可重复读和幻读？

事务并发处理可能存在的异常都有哪些？

在了解数据库隔离级别之前，我们需要了解设定事务的隔离级别都要解决哪些可能存在的问题，也就是事务并发处理时会存在哪些异常情况。实际上，SQL-92标准中已经对3种异常情况进行了定义，这些异常情况级别分别为脏读（Dirty Read）、不可重复读（Nnrepeatable Read）和幻读（Phantom Read）。

脏读、不可重复读和幻读都代表了什么，我用一个例子来给你讲解下。比如说我们有个英雄表heros_temp，如下所示：

id	name
1	张飞
2	关羽
3	刘备

这张英雄表，我们会记录很多英雄的姓名，假设我们不对事务进行隔离操作，那么数据库在进行事务的并发处理时会出现怎样的情况？

第一天，小张访问数据库，正在进行事务操作，往里面写入一个新的英雄“吕布”：

```
SQL> BEGIN;
SQL> INSERT INTO heros_temp values(4, '吕布');
```

当小张还没有提交该事务的时候，小李又对数据表进行了访问，他想看下这张英雄表里都有哪些英雄：

```
SQL> SELECT * FROM heros_temp;
```

这时，小李看到的结果如下：

id	name
1	张飞
2	关羽
3	刘备
4	吕布

你有没有发现什么异常？这个时候小张还没有提交事务，但是小李却读到了小张还没有提交的数据，这种现象我们称之为“脏读”。

那么什么是不可重复读呢？

第二天，小张想查看id=1的英雄是谁，于是他进行了SQL查询：

```
SQL> SELECT name FROM heros_temp WHERE id = 1;
```

运行结果：

name
张飞

然而此时，小李开始了一个事务操作，他对id=1的英雄姓名进行了修改，把原来的“张飞”改成了“张翼德”：

```
SQL> BEGIN;
SQL> UPDATE heros_temp SET name = '张翼德' WHERE id = 1;
```

然后小张再一次进行查询，同样也是查看id=1的英雄是谁：

```
SQL> SELECT name FROM heros_temp WHERE id = 1;
```

运行结果：

name
张翼德

这个时候你会发现，两次查询的结果并不一样。小张会想这是怎么回事呢？他明明刚执行了一次查询，马上又进行了一次查询，结果两次的查询结果不同。实际上小张遇到的情况我们称之为“不可重复读”，也就是同一条记录，两次读取的结果不同。

什么是幻读？

第三天，小张想要看下数据表里都有哪些英雄，他开始执行下面这条语句：

```
SQL> SELECT * FROM heros_temp;
```

id	name
1	张飞
2	关羽
3	刘备

这时当小张执行完之后，小李又开始了一个事务，往数据库里插入一个新的英雄“吕布”：

```
SQL> BEGIN;
SQL> INSERT INTO heros_temp values(4, '吕布');
```

不巧的是，小张这时忘记了英雄都有哪些，又重新执行了一遍查询：

```
SQL> SELECT * FROM heros_temp;
```

id	name
1	张飞
2	关羽
3	刘备
4	吕布

他发现这一次查询多了一个英雄，原来只有3个，现在变成了4个。这种异常情况我们称之为“幻读”。

我来总结下这三种异常情况的特点：

1. 脏读：读到了其他事务还没有提交的数据。
2. 不可重复读：对某数据进行读取，发现两次读取的结果不同，也就是说没有读到相同的内容。这是因为有其他事务对这个数据同时进行了修改或删除。

3. 幻读：事务A根据条件查询得到了N条数据，但此时事务B更改或者增加了M条符合事务A查询条件的数据，这样当事务A再次进行查询的时候发现会有N+M条数据，产生了幻读。

事务隔离的级别有哪些？

脏读、不可重复读和幻读这三种异常情况，是在SQL-92标准中定义的，同时SQL-92标准还定义了4种隔离级别来解决这些异常情况。

解决异常数量从少到多的顺序（比如读未提交可能存在3种异常，可串行化则不会存在这些异常）决定了隔离级别的高低，这四种隔离级别从低到高分别是：读未提交（READ UNCOMMITTED ）、读已提交（READ COMMITTED）、可重复读（REPEATABLE READ）和可串行化（SERIALIZABLE）。这些隔离级别能解决的异常情况如下表所示：

	脏读	不可重复读	幻读
读未提交（READ UNCOMMITTED）	允许	允许	允许
读已提交（READ COMMITTED）	禁止	允许	允许
可重复读（REPEATABLE READ）	禁止	禁止	允许
可串行化（SERIALIZABLE）	禁止	禁止	禁止

你能看到可串行化能避免所有的异常情况，而读未提交则允许异常情况发生。

关于这四种级别，我来简单讲解下。

读未提交，也就是允许读到未提交的数据，这种情况下查询是不会使用锁的，可能会产生脏读、不可重复读、幻读等情况。

读已提交就是只能读到已经提交的内容，可以避免脏读的产生，属于RDBMS中常见的默认隔离级别（比如说Oracle和SQL Server），但如果想要避免不可重复读或者幻读，就需要我们在SQL查询的时候编写带加锁的SQL语句（我会在进阶篇里讲加锁）。

可重复读，保证一个事务在相同查询条件下两次查询得到的数据结果是一致的，可以避免不可重复读和脏读，但无法避免幻读。MySQL默认的隔离级别就是可重复读。

可串行化，将事务进行串行化，也就是在一个队列中按照顺序执行，可串行化是最高级别的隔离等级，可以解决事务读取中所有可能出现的异常情况，但是它牺牲了系统的并发性。

使用MySQL客户端来模拟三种异常

我在讲解这三种异常的时候举了一个英雄数据表查询的例子，你还可以自己写SQL来模拟一下这三种异常。

首先我们需要一个英雄数据表heros_temp，具体表结构和数据，你可以从GitHub上下载heros_temp.sql文件。

你也可以执行下面的SQL文件，来完成heros_temp数据表的创建。

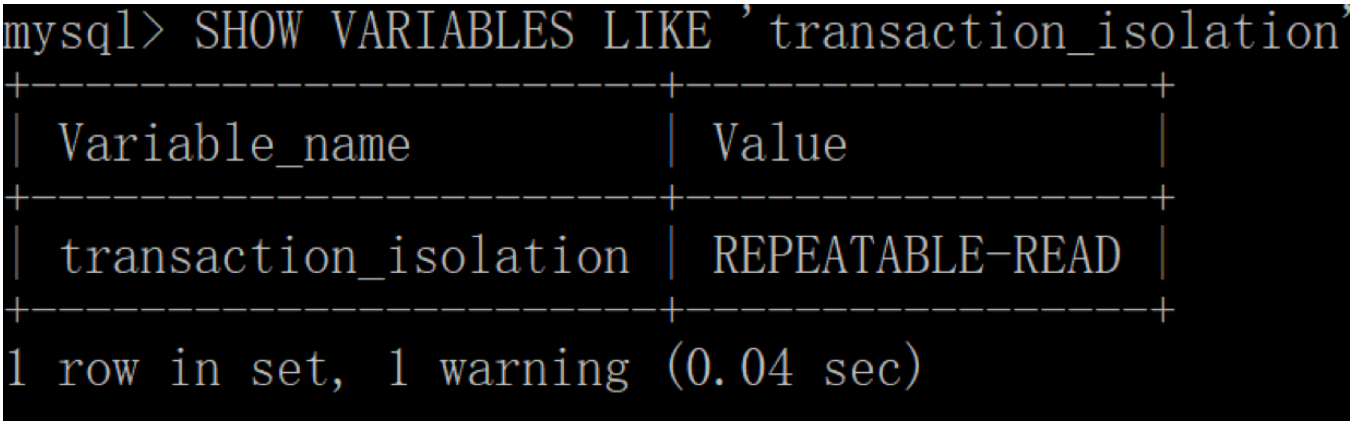
```
-- ----------------------------
-- Table structure for heros_temp
-- ----------------------------
DROP TABLE IF EXISTS `heros_temp`;
CREATE TABLE `heros_temp` (
  `id` int(11) NOT NULL,
  `name` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Records of heros_temp
-- ----------------------------
INSERT INTO `heros_temp` VALUES (1, '张飞');
INSERT INTO `heros_temp` VALUES (2, '关羽');
INSERT INTO `heros_temp` VALUES (3, '刘备');
```

模拟的时候我们需要开两个MySQL客户端，分别是客户端1和客户端2。

在客户端1中，我们先来查看下当前会话的隔离级别，使用命令：

```
mysql> SHOW VARIABLES LIKE 'transaction_isolation';
```

然后你能看到当前的隔离级别是REPEATABLE-READ，也就是可重复读。



现在我们把隔离级别降到最低，设置为READ UNCOMMITTED（读未提交）。

```
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

然后再查看下当前会话（SESSION）下的隔离级别，结果如下：

```
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'transaction_isolation';
+-----------------------+------------------+
| Variable_name         | Value            |
+-----------------------+------------------+
| transaction_isolation | READ-UNCOMMITTED |
+-----------------------+------------------+
1 row in set, 1 warning (0.00 sec)
```

因为MySQL默认是事务自动提交，这里我们还需要将autocommit参数设置为0，命令如下：

```
mysql> SET autocommit = 0;
```

然后我们再来查看SESSION中的autocommit取值，结果如下：

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW VARIABLES LIKE 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | OFF   |
+---------------+-------+
1 row in set, 1 warning (0.00 sec)
```

接着我们以同样的操作启动客户端2，也就是将隔离级别设置为READ UNCOMMITTED（读未提交），autocommit设置为0。

模拟“脏读”

我们在客户端2中开启一个事务，在heros_temp表中写入一个新的英雄“吕布”，注意这个时候不要提交。

```
mysql> BEGIN;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO heros_temp values(4, '吕布');
Query OK, 1 row affected (0.00 sec)
```

然后我们在客户端1中，查看当前的英雄表：

```
mysql> SELECT * FROM heros_temp;
+----+----------+
| id | name     |
+----+----------+
|  1 | 张飞     |
|  2 | 关羽     |
|  3 | 刘备     |
|  4 | 吕布     |
+----+----------+
4 rows in set (0.00 sec)
```

你能发现客户端1中读取了客户端2未提交的新英雄“吕布”，实际上客户端2可能马上回滚，从而造成了“脏读”。

模拟“不可重复读”

我们用客户端1来查看id=1的英雄：

```
mysql> SELECT name FROM heros_temp WHERE id = 1;
+--------+
| name   |
+--------+
| 张飞   |
+--------+
1 row in set (0.00 sec)
```

然后用客户端2对id=1的英雄姓名进行修改：

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE heros_temp SET name = '张翼德' WHERE id = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

这时用客户端1再次进行查询：

```
mysql> SELECT name FROM heros_temp WHERE id = 1;
+------------+
| name       |
+------------+
| 张翼德     |
+------------+
1 row in set (0.00 sec)
```

你能发现对于客户端1来说，同一条查询语句出现了“不可重复读”。

模拟“幻读”

我们先用客户端1查询数据表中的所有英雄：

```
mysql> SELECT * FROM heros_temp;
+----+---------+
| id | name    |
+----+---------+
|  1 | 张飞    |
|  2 | 关羽    |
|  3 | 刘备    |
+----+---------+
3 rows in set (0.00 sec)
```

然后用客户端2，开始插入新的英雄“吕布”：

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO heros_temp values(4, '吕布');
Query OK, 1 row affected (0.00 sec)
```

这时，我们再用客户端1重新进行查看：

```
mysql> SELECT * FROM heros_temp;
+----+---------+
| id | name    |
+----+---------+
|  1 | 张飞    |
|  2 | 关羽    |
|  3 | 刘备    |
|  4 | 吕布    |
+----+---------+
4 rows in set (0.00 sec)
```

你会发现数据表多出一条数据。

如果你是初学者，那么你可以采用heros_temp数据表简单模拟一下以上的过程，加深对脏读、不可重复读以及幻读的理解。对应的，你也会更了解不同的隔离级别解决的异常问题。

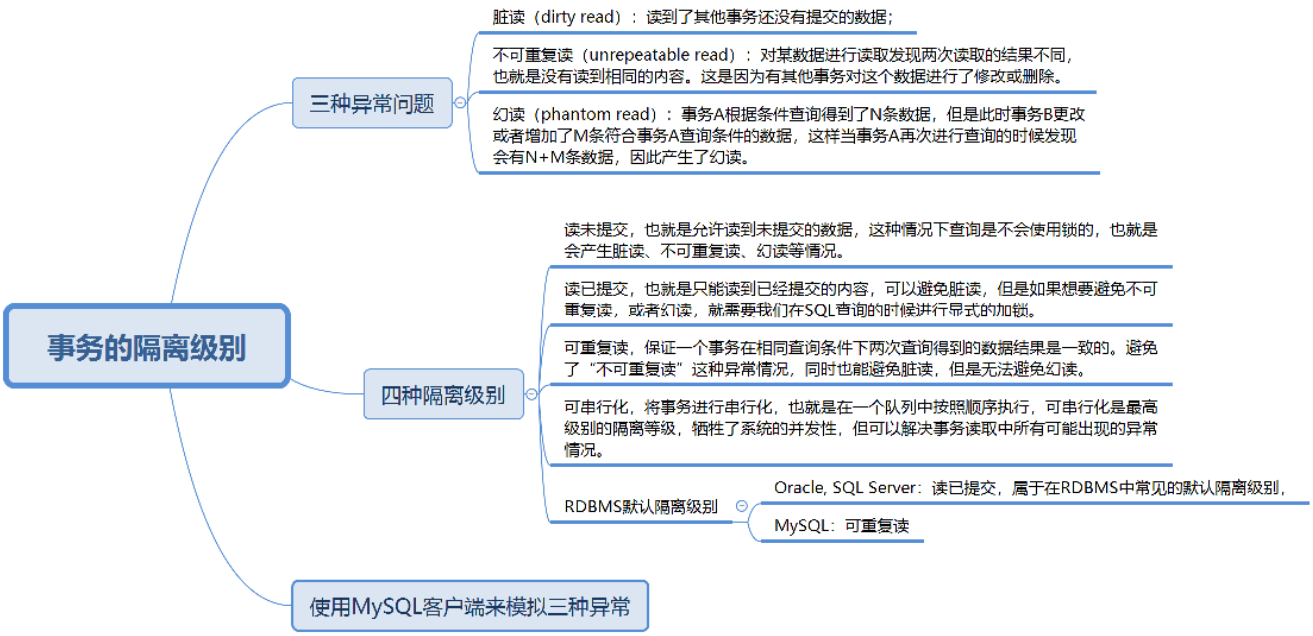
总结

我们今天只是简单讲解了4种隔离级别，以及对应的要解决的三种异常问题。我会在优化篇这一模块里继续讲解隔离级别以及锁的使用。

你能看到，标准的价值在于，即使是不同的RDBMS都需要达成对异常问题和隔离级别定义的共识。这就意味着一个隔离级别的实现满足了下面的两个条件：

1. 正确性：只要能满足某一个隔离级别，一定能解决这个隔离级别对应的异常问题。
2. 与实现无关：实际上RDBMS种类很多，这就意味着有多少种RDBMS，就有多少种锁的实现方式，因此它们实现隔离级别的原理可能不同，然而一个好的标准不应该限制其实现的方式。

隔离级别越低，意味着系统吞吐量（并发程度）越大，但同时也意味着出现异常问题的可能性会更大。在实际使用过程中我们往往需要在性能和正确性上进行权衡和取舍，没有完美的解决方案，只有适合与否。



今天的内容到这里就结束了，你能思考一下为什么隔离级别越高，就越影响系统的并发性能吗？以及不可重复读和幻读的区别是什么？

欢迎你在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事。

精选留言

大牛凯

2

老师好，对幻读有些迷惑，从网上看到幻读并不是说两次读取获取的结果集不同，幻读侧重的方面是某一次的 select 操作得到的结果所表征的数据状态无法支撑后续的业务操作。更为具体一些：select 某记录是否存在，结果显示不存在，准备插入此记录，但执行 insert 时发现此记录已存在，无法插入，此时就发生了幻读。

2019-07-15

作者回复

你说的这种情况属于幻读。
当你INSERT的时候，也需要隐式的读取，比如插入数据时需要读取有没有主键冲突，然后再决定是否能执行插入。如果这时发现已经有这个记录了，就没法插入。

官方对幻读的定义是：The so-called phantom problem occurs within a transaction when the same query produces different sets of rows at different times.
For example, if a SELECT is executed twice, but returns a row the second time that was not returned the first time, the row is a “phantom” row.（详见：https://dev.mysql.com/doc/refman/8.0/en/innodb-next-key-locking.html ）

需要说明下，不可重复读 VS 幻读的区别：
不可重复读是同一条记录的内容被修改了，重点在于UPDATE或DELETE
幻读是查询某一个范围的数据行变多了或者少了，重点在于INSERT

所以，SELECT 显示不存在，但是INSERT的时候发现已存在，说明符合条件的数据行发生了变化，也就是幻读的情况，而不可重复读指的是同一条记录的内容被修改了。

2019-07-15

石维康

5

老师可以详细讲解下脏读和幻读的区别吗？看文中的例子几乎是一样的。

2019-07-15

这个需求做不了

啦啦啦

3

打卡打卡

2019-07-15

flow

1

关于事务隔离和异常问题的举例不够详细和严谨，具体可以看这个 https://www.liaoxuefeng.com/wiki/1177760294764384/1179611198786848

以下是自己的理解：
读未提交：在这个隔离级别下，事务A会读到事务B未提交的数据，在事务B回滚后，事务A读到的数据无意义，是脏数据，称为 脏读
读已提交：在这个隔离级别下，只有在事务B已提交时，事务A才能读到，如果事务A先查询id为1的记录，之后事务B修改这条记录并提交，事务A再读取，两次结果会不一致，所以不可重

复读。
可重复读：在这个隔离级别下，就算事务B的修改已经提交，事务A读到的数据依旧是一致的。当事务B插入一条新数据并提交之后，事务A查询不到当前数据，查询不到就以为不存在，但是事务A却可以更新这条数据成功，并且更新后再次查询，数据出现了。一开始查询不到，但能修改，再次查询又出现了，跟幻觉一样，所以称为　幻读。

2019-07-15

JackPn　1

老师我感觉幻读也是不了重复读啊，都是一个事务过程中两次读到了另一个事务修改提交后的数据

2019-07-15

作者回复

首先，不可重复读 和 幻读都是在先后两次读取的时候发现不一致的情况，但是两种读取略有差别：

不可重复读是对于同一条记录内容的“不可重复读”

幻读是对于某一范围的数据集，发现查询数据集的行数多了或者少了，从而出现的不一致。

所以不可重复读的原因是 对于要查询的那条数据进行了UPDATE或DELETE

而幻读是对于要查询的 那个范围的数据集，进行了INSERT。

2019-07-15

0 error 0 warning 0 bug　1

不可重复读和幻读还不是很理解，老师可以再详细讲讲吗

2019-07-15

作者回复

不可重复读 VS 幻读的区别：

不可重复读是同一条记录的内容被修改了，重点在于UPDATE或DELETE

幻读是查询某一个范围的数据行变多了或者少了，重点在于INSERT

2019-07-15

一叶知秋　0

两个问题个人理解是：
1）四个级别分别是无限制（可以并发读写）、写事务加锁（可以并发读、读完立刻释放锁而不是等事务结束）、读写事务都加锁（可以并发读、读写都是事务结束才释放锁）、表锁（读写事务序列化执行、单线程执行）。 时间开销依次递增所以随着隔离等级递增并发性能会降低。
2）区别在于不可重复读是由于其他事务的update、delete操作对数据进行了修改 重点在修改（内容修改）、幻读是其他事务由于delete、insert对表数据进行了修改重点在于数量新增、减少（数量变更）

嘿嘿个人理解，不对希望指出 >o<

2019-07-15

dbtiger　0

【隔离级别越高，就越影响系统的并发性能】

1,首先隔离的实现机制是锁，隔离级别越高锁的代价越大（锁的粒度越小，表级锁到行级锁，共享锁到独占锁），终极为了一致性读写，只能是串行化操作读写（类似于操作系统的多进程原理，看着像是并行性执行，实则是单元分配CPU资源串行执行的过程）。
【不可重复读和幻读的区别】
2.我认为没啥区别，前者只是列值改变了，后者侧重是记录数变了。都是2次读的时候中间夹了一个已经执行了的事务，从而产生2次读的数据不一致的情况。

另外，请教一下陈老师，存储过程里面有很多dml操作，每个dml语句加begin...end好，还是不加好，还是一样?两种状态对锁的持有时间是不是相同的？

2019-07-15

Hail hydra

0

高并发的情况要保证数据的一致性就必须排队一个个来，那这不就不能算并发了么

2019-07-15

ack

0

事务的隔离级别是需要通过锁来保证的，想要解决的问题越多，加的锁就越多，从文章也可以看出，当想要解决幻读的时候，需要的隔离级别已经是串行化了。

2019-07-15

作者回复

对 加锁是底层的实现环节，不同的事务隔离级别对应能解决不同的异常问题。在选择隔离级别的时候，我们要在正确性和性能上进行权衡取舍

2019-07-15

L荀

0

不可重复读，和幻读例子中事物不用提交么

2019-07-15

作者回复

一个好问题，一般来说第二个事务需要进行提交。不过在文章中，我将客户端1的隔离级别设置为了 读未提交，因此不论客户端2是否提交，都会对客户端1造成影响。
如果将客户端1的隔离级别设置为 读已提交，或者 可重复度。就需要对客户端2的事务进行提交，这时才会对客户端1在执行的事务产生影响。

2019-07-15

墨禾

0

老师，分库分表是不是可以一定程度上保证隔离性和并发访问呢？

2019-07-15