08 搞清事务隔离级别，理解数据并发

更新时间：2020-03-18 10:18:04

> 学习要注意到细处，不是粗枝大叶的，这样可以逐步学习、摸索，找到客观规律。 —— 徐特立

相信大家对 “事务隔离级别” 这个词都不会陌生，我们在编码中、面试中一定都会遇到，由此也不禁感叹它出现的频率太高了。但同时，它也是 MySQL 中比较难理解的概念，你会发现，真的很难用几句话把它解释清楚。这一节里，我除了会对事务隔离级别基本的理论、概念进行讲解之外，还会去演示在不同的隔离级别设定下对事务的影响。

1 你真的理解 MySQL 中的事务吗 ？

在解读事务隔离级别之前，我们先来看一看事务。这里我将会说明三个话题：事务是什么、事务的 ACID 特性、并发事务会带来什么样的问题。好的，我们先来解读下事务的概念吧。

1.1 事务是什么

事务是作为单个逻辑工作单元执行的一系列操作，它所表达的语义是：要么全部执行，要么全部不执行。我们在读书时可能就听过银行取款的例子，用它来解释事务就再合适不过了。假如 A 要去银行取款 100 元，这次的取款过程会涉及两个操作：

- 将 A 的余额减少 100 元
- A 获得 100 元取款

那么，这两个操作就是一次事务，因为这两个操作只能全部成功或全部失败，任何一个部分成功或失败，将会是非常严重的系统漏洞。事务的目标是保证数据库的完整性，避免各种原因引起的数据库内容不一致的问题。所以，事务可以保证数据安全，事务控制实际上就是在控制数据的安全访问。

1.2 事务的 ACID 特性

事务必须要有四个属性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）。深刻理解这四个属性是理解事务的基础，下面，我们依次来看一看这四个属性的含义与要求。

- 原子性：原子本身是化学中的一个名词，它指的是构成化学元素的最小粒子，即不能再更细的分割了。事务操作必须是原子的，对于一个事务中的所有操作，要么全部执行（COMMIT），要么全部不执行（ROLLBACK）
- 一致性：一致性指的是数据的完整性，即执行事务的前后，数据整体应该是一致的。事务必须能够让数据库从一个一致性状态变换到另一个一致性状态。对于取款的案例来说，A 的数据总值就是一致的
- 隔离性：它指的是一个事务的执行不能被其他事务所干扰，这里又涉及到并发的概念。一个事务内部的操作对其他并发的事务是隔离的，简单的说，每个事务都认为是自己独占数据库
- 持久性：这个属性简单易懂，它是说一个事务一旦提交（COMMIT），它对数据库中数据的改变就是永久性的。任何操作甚至是系统故障都不应该对其产生影响

可以看到，事务有着严格的判定标准，想要同时实现它们又要求有很高的性能，可谓是难上加难。所以，在各大数据库厂商的实现中，真正能够满足这四个特性的事务寥寥无几。例如：InnoDB 存储引擎默认的事务隔离级别是可重复读，它不能满足隔离性要求；而 MySQL 的 NDB Cluster 事务则不满足持久性和隔离性。所以，与其说 ACID 是事务必须满足的条件，不如说它们是衡量事务的严谨性标准。

1.3 并发事务会带来什么样的问题

如果你写过多线程的程序，那就一定对并发的概念不会陌生了。并发事务的概念是多个事务并发运行，那么，如果在并发运行的过程中对相同的数据进行了修改，就可能会引起一些问题。总结下来，可能出现的问题一共有三种：脏读、不可重复读、幻读。

- 脏读：事务 A 读取了事务 B 当前更新的数据，但是事务 B 出现了回滚或未提交修改，事务 A 读到的数据就被称为 “脏数据”。通常情况下，使用 “脏数据” 会造成系统数据不一致，出现错误
- 不可重复读：事务 A 在执行过程中多次读取同一数据，但是事务 B 在事务 A 的读取过程中对数据做了多次修改并提交，则会导致事务 A 多次读取的数据不一致，进而无法做出准确性判断
- 幻读：事务 A 在执行过程中读取了一些数据，但是事务 B 随即插入了一些数据，那么，事务 A 重新读取时，发现多了一些原本不存在的数据，就像是幻觉一样，称之为幻读

仔细品味，可以发现，不可重复读与幻读从概念上来说，是非常相似的。区分它们只要记住：不可重复读指的是对原来存在的数据做修改，而幻读指的是新增或者删除数据。

2 解读事务隔离级别

SQL 标准定义了四种隔离级别，由低到高依次为：READ-UNCOMMITTED（未提交读）、READ-COMMITTED（提交读）、REPEATABLE-READ（可重复读）、SERIALIZABLE（串行化）。它们可以逐个解决脏读、不可重复读、幻读这几类问题。

2.1 SQL 标准定义的四种事务隔离级别

这里，我将按照隔离级别由低到高的顺序依次对它们进行解读：

- **READ-UNCOMMITTED**：它是最低的隔离级别，正如它的名称一样，它允许一个事务读取其他事务未提交的数据。这个隔离级别很少在工业环境中应用，因为它的性能并不会比其他高级别的性能好很多
- **READ-COMMITTED**：它可以保证一个事务修改的数据提交之后才能被其他的事务读取。这个隔离级别是大多数数据库系统的默认隔离级别，但并不是 MySQL 默认的
- **REPEATABLE-READ**：它的核心在于“可重复”，即在一个事务内，对同一字段的多次读取结果都是相同的，也是 MySQL 的默认事务隔离级别
- **SERIALIZABLE**：它是最高的隔离级别，花费的代价也是最高的，事务的处理是顺序执行的。在这个级别上，可能会导致大量的锁超时现象和锁竞争。同样，在工业级环境中，很少被使用

仔细分析这四种隔离级别，是不是发现：除 SERIALIZABLE 之外的另外三种都不能解决所有的问题。所以，在实际的应用中，一定是有所取舍的。

2.2 不同事务隔离级别可能会产生的问题

隔离级别越低，事务请求的锁也就越少，所以，可能出现的问题也就越多。而 SERIALIZABLE，它通过强制事务排序，并按顺序执行，使各个事务之间不可能会产生冲突，从而才能够解决脏读、不可重复读、幻读所有的问题。下面，我用一张表来总结各个事务隔离级别能够解决的问题（使用 Y 标识）与不能够解决的问题（使用 N 标识）。

隔离级别	脏读	不可重复读	幻读
READ-UNCOMMITTED	N	N	N
READ-COMMITTED	Y	N	N
REPEATABLE-READ	Y	Y	N
SERIALIZABLE	Y	Y	Y

具体选择哪一种隔离级别应该是多个维度的考虑，例如：事务请求锁的多少（性能问题）、能够解决什么问题、业务特点等等。一般情况下，使用 InnoDB 存储引擎，我们会选择 READ-COMMITTED。

3 实践不同事务隔离级别对事务的影响

理论知识总是不直观的，下面，我将使用示例的形式直观的感受下不同事务隔离级别对事务的影响。首先，我们先去看一看 InnoDB 存储引擎系统级的隔离级别和会话级的隔离级别（默认情况下的）：

```
-- global.tx_isolation 代表系统级的隔离级别，即对所有的会话都生效
-- tx_isolation 代表会话级隔离级别，只对当前会话生效
mysql> SELECT @@global.tx_isolation, @@tx_isolation;
+-----------------------+-----------------+
| @@global.tx_isolation | @@tx_isolation  |
+-----------------------+-----------------+
| REPEATABLE-READ       | REPEATABLE-READ |
+-----------------------+-----------------+
```

修改隔离级别的方式也非常简单，示例如下：

```
-- 在 session1 中修改会话的隔离级别
mysql> SET session tx_isolation='read-uncommitted';
Query OK, 0 rows affected, 1 warning (0.00 sec)

-- session1 的 tx_isolation 修改成功
mysql> SELECT @@global.tx_isolation, @@tx_isolation;
+-----------------------+------------------+
| @@global.tx_isolation | @@tx_isolation   |
+-----------------------+------------------+
| REPEATABLE-READ       | READ-UNCOMMITTED |
+-----------------------+------------------+

-- 对 session2 并没有产生影响
mysql> SELECT @@global.tx_isolation, @@tx_isolation;
+-----------------------+-----------------+
| @@global.tx_isolation | @@tx_isolation  |
+-----------------------+-----------------+
| REPEATABLE-READ       | REPEATABLE-READ |
+-----------------------+-----------------+
```

默认情况下，SQL 语句是自动提交的，我们也可以将自动提交关闭：

```
-- 查看默认的 SQL 语句自动提交
mysql> SHOW VARIABLES LIKE 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | ON    |
+---------------+-------+

-- 关闭自动提交
mysql> SET autocommit = off;

-- 检验自动提交是否已关闭
mysql> SHOW VARIABLES LIKE 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | OFF   |
+---------------+-------+
```

在实际操作中还会使用到一些并发控制语句：

- start transaction：显式地开启一个事务
- commit：提交事务，使得对数据库做的所有修改成为永久性
- rollback：回滚结束用户的事务，并撤销正在进行的所有未提交的修改

最后，我们还需要有一张示例表并填充一些示例数据，如下所示 worker 表：

```
mysql> SELECT id, type, name, salary FROM worker;
+----+------+--------+--------+
| id | type | name   | salary |
+----+------+--------+--------+
|  1 | A    | tom    |   1800 |
|  2 | B    | jack   |   2100 |
|  3 | C    | pony   |   NULL |
|  4 | B    | tony   |   3600 |
|  5 | B    | marry  |   1900 |
|  6 | C    | tack   |   1200 |
|  7 | A    | tick   |   NULL |
|  8 | B    | clock  |   2000 |
|  9 | C    | noah   |   1500 |
| 10 | C    | jarvis |   1800 |
+----+------+--------+--------+
```

好的，准备工作已经就绪了，接下来，我们开启两个会话（MySQL 客户端），看一看数据库处于不同的事务隔离级别会有怎样的状况发生。

3.1 READ-UNCOMMITTED 级别

```
-- 修改 “会话 A” 的事务隔离级别
mysql> SET session tx_isolation='read-uncommitted';

-- 查看 “会话 A” 的事务隔离级别（之后将不再重复查看）
mysql> SELECT @@tx_isolation;
+------------------+
| @@tx_isolation   |
+------------------+
| READ-UNCOMMITTED |
+------------------+

-- “会话 A” 开启事务
mysql> start transaction;

-- “会话 A” 查询 worker 表中 id 为1的记录
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   1800 |
+----+------+------+--------+

-- “会话 B” 开启事务
mysql> start transaction;

-- “会话 B” 修改 id 为1的记录，将 salary 修改为 2000，但是并不提交
mysql> UPDATE worker SET salary = 2000 WHERE id = 1;

-- “会话 A” 再次查询 worker 表中 id 为1的记录
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   2000 |
+----+------+------+--------+
```

3.2 READ-COMMITTED 级别

```
-- 修改 “会话 A” 的事务隔离级别
mysql> SET session tx_isolation='read-committed';

-- “会话 A” 开启事务
mysql> start transaction;

-- “会话 A” 查询 worker 表中 id 为1的记录
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   1800 |
+----+------+------+--------+

-- “会话 B” 开启事务
mysql> start transaction;

-- “会话 B” 修改 id 为1的记录，将 salary 修改为 2000，但是并不提交
mysql> UPDATE worker SET salary = 2000 WHERE id = 1;

-- “会话 A” 再次查询 worker 表中 id 为1的记录，数据并没有发生变化，因为 “会话 B” 并未提交
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   1800 |
+----+------+------+--------+

-- “会话 B” 提交
mysql> commit;

-- “会话 A” 再次查询 worker 表中 id 为1的记录
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   2000 |
+----+------+------+--------+
```

3.3 REPEATABLE-READ 级别

```
-- 修改 “会话 A” 的事务隔离级别
mysql> SET session tx_isolation='repeatable-read';

-- “会话 A” 开启事务
mysql> start transaction;

-- “会话 A” 查询 worker 表中 id 为1的记录
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   1800 |
+----+------+------+--------+

-- “会话 B” 开启事务
mysql> start transaction;

-- “会话 B” 修改 id 为1的记录，将 salary 修改为 2000，但是并不提交
mysql> UPDATE worker SET salary = 2000 WHERE id = 1;

-- “会话 A” 再次查询 worker 表中 id 为1的记录，数据并没有发生变化
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   1800 |
+----+------+------+--------+

-- “会话 B” 提交
mysql> commit;

-- “会话 A” 再次查询 worker 表中 id 为1的记录，数据仍没有发生变化
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   1800 |
+----+------+------+--------+
```

3.4 SERIALIZABLE 级别

```
-- 修改 “会话 A” 的事务隔离级别
mysql> SET session tx_isolation='serializable';

-- “会话 A” 开启事务
mysql> start transaction;

-- “会话 A” 查询 worker 表中 id 为1的记录
mysql> SELECT * FROM worker WHERE id = 1;
+----+------+------+--------+
| id | type | name | salary |
+----+------+------+--------+
|  1 | A    | tom  |   1800 |
+----+------+------+--------+

-- “会话 B” 开启事务
mysql> start transaction;

-- “会话 B” 修改 id 为1的记录，将 salary 修改为 2000，由于 “会话 A” 事务没有提交，“会话 B” 的事务一直处于等待状态，直到超时
mysql> UPDATE worker SET salary = 2000 WHERE id = 1;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

综上可以看到，每一种事务隔离级别有着不一样的操作效果。想要更好的理解它们，就应该立刻动手去实验，但是需要特别注意事务开启、提交、回滚顺序的正确性，否则，可能会得出不一样的结论。

4 总结

事务隔离级别是 MySQL 中的进阶知识点，想要把它完全搞明白并不是一件简单的事。同时，也正是由于它的存在，才保证了数据并发的正确性。虽然目前的各种 ORM 框架已经封装了事务的开启、提交、回滚过程，但是，完全的掌握还是非常有必要的。这将有利于你在工作中写出正确的代码，以及在面试中能够驾轻就熟。

5 问题

你能举一个并发事务中出现幻读和不可重复读的例子吗？

修改系统级的隔离级别（SET global tx_isolation），验证对所有的会话都是生效的？

挑选一个事务隔离级别，例如：READ-COMMITTED，演示并验证它可能会导致的问题？

当会话处于 REPEATABLE-READ 隔离级别时，读取到的数据就一定是一样的吗？

6 参考资料

《高性能 MySQL（第三版）》

《MySQL 技术内幕：InnoDB 存储引擎》

MySQL 官方文档：InnoDB Locking and Transaction Model

MySQL 官方文档：SET TRANSACTION Statement

}

07 掌握数据备份与恢复是很有必要的

09 通过锁解决并发数据问题