Atitit 数据库并发控制

# 13.2. 事务隔离

SQL标准用三个必须在并发的事务之间避免的现像定义了四个级别的事务隔离。 这些不希望发生的现像是：

脏读

一个事务读取了另一个未提交事务写入的数据。

不可重复读

一个事务重新读取前面读取过的数据，发现该数据已经被另一个已提交事务修改。

幻读

一个事务重新执行一个查询，返回一套符合查询条件的行， 发现这些行因为其它最近提交的事务而发生了改变。

这四种隔离级别和对应的行为在Table 13-1里描述。

**Table 13-1. SQL事务隔离级别**

| **隔离级别** | **脏读** | **不可重复读** | **幻读** |
| --- | --- | --- | --- |
| 读未提交 | 可能 | 可能 | 可能 |
| 读已提交 | 不可能 | 可能 | 可能 |
| 重复读 | 不可能 | 不可能 | 可能 |
| 可串行化 | 不可能 | 不可能 | 不可能 |

在PostgreSQL里，你可以请求四种可能的事务隔离级别中的任意一种。 但是在内部，实际上只有两种独立的隔离级别，分别对应读已提交和可串行化。 如果你选择了读未提交的级别，实际上你用的是读已提交，在你选择可重复读级别的时候， 实际上你用的是可串行化，所以实际的隔离级别可能比你选择的更严格。这是 SQL 标准允许的： 四种隔离级别只定义了哪种现像不能发生，但是没有定义那种现像一定发生。 PostgreSQL只提供两种隔离级别的原因是， 这是把标准的隔离级别与多版本并发控制架构映射相关的唯一合理方法。 可用的隔离级别的行为在下面小节里描述。

要设置一个事务的隔离级别，使用SET TRANSACTION命令。

## 13.2.1. 读已提交隔离级别

读已提交是 PostgreSQL 里的缺省隔离级别。 当一个事务运行在这个隔离级别时， SELECT查询(没有FOR UPDATE/SHARE的子句)只能看到查询开始之前已提交的数据 而无法看到未提交的数据或在查询执行期间其它事务已提交的数据。 不过SELECT看得见其自身所在事务中前面尚未提交的更新结果。 实际上，SELECT 查询看到一个在查询开始运行的瞬间该数据库的一个快照。 请注意，在同一个事务里两个相邻的SELECT命令可能看到不同的快照， 因为其它事务会在第一个SELECT执行期间提交。

UPDATE，DELETE，SELECT FOR UPDATE和SELECT FOR SHARE命令在搜索目标行时的行为和 SELECT 一样： 它们只能找到在命令开始的时候已经提交的行。 不过，找到这样的目标行的时候可能已经被其它并发事务更新、删除、锁住。在这种情况下， 即将进行的更新将等待第一个事务提交或者回滚(如果它还在处理)。 如果第一个事务回滚，那么它的作用将忽略，而第二个事务将继续更新最初发现的行。 如果第一个事务提交，那么如果第一个事务删除了该行，则第二个事务将忽略该行， 否则它将试图在该行的已更新的版本上施加它的操作。系统将重新计算命令搜索条件(WHERE子句)， 看看该行已更新的版本是否仍然符合搜索条件。如果符合，则第二个事务从该行的已更新版本开始继续其操作。 如果是SELECT FOR UPDATE和SELECT FOR SHARE 则意味着把已更新的行版本锁住并返回给客户端。

: 因为上面的规则，正在更新的命令可能会看到不一致的快照： 它们可以看到影响它们更新的并发命令的效果，但是却看不到那些命令对数据库里其它行的作用。 这样的行为令读已提交模式不适合用于哪种涉及复杂搜索条件的命令。 不过，它对于简单的情况而言是正确的。 比如，假设我们用类似下面这样的命令更新银行余额：

BEGIN;

UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 12345;

UPDATE accounts SET balance = balance - 100.00 WHERE acctnum = 7534;

COMMIT;

如果两个并发事务试图同时修改帐号12345的余额，那我们很明显希望第二个事务是从已更新过的行版本上进行更新。 因为每个命令只是影响一个已经决定了的行，因此让它看到更新后的版本不会导致任何不一致的问题。

在读取已提交模式中，更复杂的使用可能产生不需要的结果。比如： 考虑删除命令操作正在数据中添加和删除其都限制条件由另一个命令。 website是个有两行数据表。website.hits分别为9和 10：

BEGIN;

UPDATE website SET hits = hits + 1;

-- run from another session: DELETE FROM website WHERE hits = 10;

COMMIT;

DELETE不会影响在UPDATE前后website.hits = 10的行。 这是因为发生更新前值9的行跳过，当UPDATE完成，DELETE获取一个锁。 新行值不再是10而是11，没有匹配的条件。

因为在读已提交模式里，每个命令都是从一个新的快照开始的，而这个快照包含所有到该时刻为止已提交的事务， 因此同一事务中后面的命令将看到任何已提交的其它事务的效果。 这里关心的问题是在单个命令里是否看到数据库里绝对一致的视图。

读已提交模式提供的部分事务隔离对于许多应用而言是足够的，并且这个模式速度快，使用简单。 不过，对于做复杂查询和更新的应用，可能需要保证数据库有比读已提交模式更加严格的一致性视图。

## 13.2.2. 可串行化隔离级别

可串行化级别提供最严格的事务隔离。 这个级别模拟串行的事务执行，就好像事务将被一个接着一个那样串行(而不是并行)的执行。 不过，使用这个级别的应用必须准备在串行化失败的时候重新启动事务。

当一个事务处于可串行化级别时， SELECT查询只能看到在事务开始之前已提交的数据而看不到未提交的数据或事务执行期间其它并发事务已提交的修改。 不过，SELECT 看得见其自身所在事务中前面尚未提交的更新结果。 可串行化的SELECT看到的是该事务开始时的快照，而不是该事务内部当前查询开始时的快照，这个行为和读已提交级别是不一样的。 这样，同一个事务内部后面的SELECT命令总是看到同样的数据。 它们看不到其它事务所提交在它们自己的事务开始后所做的更改。 （这种行为可以报表应用程序的理想选择。）

UPDATE,DELETE,SELECT FOR UPDATE, 和SELECT FOR SHARE在搜索目标行时的行为和 SELECT 一样： 它们将只寻找在事务开始的时候已经提交的目标行。 但是，这样的目标行在被发现的时候可能已经被另外一个并发的事务更新、删除、锁住。在这种情况下， 可串行化的事务将等待第一个正在更新的事务提交或者回滚(如果它仍然在处理中)。 如果第一个事务回滚，那么它的影响将被忽略，而这个可串行化的就可以继续更新它最初发现的行。 但是如果第一个事务被提交了(并且实际上更新或者删除了该行，而不只是锁住它)那么可串行化事务将回滚，并返回下面信息：

ERROR: could not serialize access due to concurrent update

因为一个可串行化的事务在开始之后不能更改或者锁住被其它事务更改过的行。

当应用收到这样的错误信息时，它应该退出当前的事务然后从新开始进行整个事务。 第二次运行时，该事务看到的快照将包含前一次已提交的修改，所以不会有逻辑冲突。

请注意只有更新事务才需要重试，只读事务从来没有串行化冲突。

可串行化事务级别提供了严格的保证：每个事务都看到一个完全一致的数据库视图。 不过，如果并发更新令数据库不能维持串行执行的样子，那么应用必须准备重试事务。 因为重做复杂事务的开销可能是非常可观的， 所以我们只建议当更新命令中包含复杂逻辑并且在读已提交级别中可能导致错误结果的时候才使用可串行化级别。 可串行化模式只是在这样的情况下是必要的： 一个事务连续做若干个命令，而这几个命令必须看到完全相同的数据库视图。

### 13.2.2.1. 可串行化隔离与真正的可串行化之比较

执行的"可串行化"的直观含义(以及数学定义)是两个成功提交的并发事务将显得好像严格地串行执行一样， 一个跟着一个(尽管我们可能无法预期哪个首先执行)。 但是我们必须明白，禁止那些在表12-1里面列出的行为并不能保证真正的可串行化， 并且，实际上PostgreSQL的可串行化模式并不保证在这种含义下的可串行化。 举例来说，假设一个表mytab，最初包含：

class | value

-------+-------

1 | 10

1 | 20

2 | 100

2 | 200

假设可串行化事务 A 计算：

SELECT SUM(value) FROM mytab WHERE class = 1;

然后把结果(30)作为value字段值插入到表中，并令新行的class= 2。 同时，另一个并发的可串行化的事务B进行下面计算:

SELECT SUM(value) FROM mytab WHERE class = 2;

然后把结果(300)作为 class 字段值插入到表中，并令新行的class= 1。 然后两个事务都提交。所有列出的禁止行为都不会发生，但是我们拿到的结果是不可能在任何一种串行执行下看到的。 如果 A 在 B 之前执行，B 应该计算出总和 330 ，而不是 300 ，如果 B 在 A 之前执行，那么 A 计算出的总和也会不同。

为了保证真正数学上的可串行化，有必要对一个数据库系统强制谓词锁定，这就意味着一个事务不能插入或者更改这样的数据行： 这个数据行的数据匹配另外一个并发事务的 WHERE 条件。 比如，一旦事务 A 执行了查询SELECT ... WHERE class = 1， 那么一个谓词锁定系统将禁止事务B插入任何 class 为 1 的新行，直到 A 提交。 [1] 这样的锁系统实现起来非常复杂，并且执行起来代价高昂，因为每个会话都必须要知道每个并发事务的每个查询的执行细节。 并且这样大量的开销在大部分情况下都是不必要的浪费，因为在实际情况下大部分应用都不做这种制造麻烦的事情。 当然，上面的例子是精心设计的，不能代表真实的软件。 因此，PostgreSQL并未实现谓词锁定。

在那些非串行化执行真的可能有危险的场合，可以通过使用明确的锁定来避免问题的发生。 更多的讨论在下面的小节进行

### Notes

|  |  |
| --- | --- |
| [1] | 实际上，一个谓词锁定系统避免了幻读，方法是约束写入的东西， 而 MVCC 避免幻读的方法是约束它读取的东西。 |