13.2. 事务隔离

SQL标准定义了四种隔离级别。最严格的是可序列化，在标准中用了一整段来定义它，其中说到一组可序列化事务的任意并发执行被保证效果和以某种顺序一个一个执行这些事务一样。其他三种级别使用并发事务之间交互产生的现象来定义，每一个级别中都要求必须不出现一种现象。注意由于可序列化的定义，在该级别上这些现象都不可能发生（这并不令人惊讶--如果事务的效果与每个时刻只运行一个的相同，你怎么可能看见由于交互产生的现象？）。

在各个级别上被禁止出现的现象是：

脏读

一个事务读取了另一个并行未提交事务写入的数据。

不可重复读

一个事务重新读取之前读取过的数据，发现该数据已经被另一个事务（在初始读之后提交）修改。

幻读

一个事务重新执行一个返回符合一个搜索条件的行集合的查询， 发现满足条件的行集合因为另一个最近提交的事务而发生了改变。

序列化异常

成功提交一组事务的结果与这些事务所有可能的串行执行结果都不一致。

SQL 标准和 PostgreSQL 实现的事务隔离级别在 表 13.1中描述。

表 13.1. 事务隔离级别

隔离级别	脏读	不可重复读	幻读	序列化异常
读未提交	允许，但不在 PG 中	可能	可能	可能
读已提交	不可能	可能	可能	可能
可重复读	不可能	不可能	允许，但不在 PG 中	可能
可序列化	不可能	不可能	不可能	不可能

在PostgreSQL中，你可以请求四种标准事务隔离级别中的任意一种，但是内部只实现了三种不同的隔离级别，即 PostgreSQL 的读未提交模式的行为和读已提交相同。这是因为把标准隔离级别映射到 PostgreSQL 的多版本并发控制架构的唯一合理的方法。

该表格也显示 PostgreSQL 的可重复读实现不允许幻读。而 SQL 标准允许更严格的行为：四种隔离级别只定义了哪种现像不能发生，但是没有定义哪种现像***必须***发生。可用的隔离级别的行为在下面的小节中详细描述。

要设置一个事务的事务隔离级别，使用SET TRANSACTION命令。

重要

某些PostgreSQL数据类型和函数关于事务的行为有特殊的规则。特别是，对一个序列的修改（以及用`serial`声明的一列的计数器）是立刻对所有其他事务可见的，并且在作出该修改的事务中断时也不会被回滚。见第 9.17 节和第 8.1.4 节。

13.2.1. 读已提交隔离级别

*读已提交*是PostgreSQL中的默认隔离级别。 当一个事务运行使用这个隔离级别时， 一个查询（没有`FOR UPDATE/SHARE`子句）只能看到查询开始之前已经被提交的数据， 而无法看到未提交的数据或在查询执行期间其它事务提交的数据。实际上，`SELECT`查询看到的是一个在查询开始运行的瞬间该数据库的一个快照。不过`SELECT`可以看见在它自身事务中之前执行的更新的效果，即使它们还没有被提交。还要注意的是，即使在同一个事务里两个相邻的`SELECT`命令可能看到不同的数据， 因为其它事务可能会在第一个`SELECT`开始和第二个`SELECT`开始之间提交。

`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： 它们将只找到在命令开始时已经被提交的行。 不过，在被找到时，这样的目标行可能已经被其它并发事务更新（或删除或锁住）。在这种情况下， 即将进行的更新将等待第一个更新事务提交或者回滚（如果它还在进行中）。 如果第一个更新事务回滚，那么它的作用将被忽略并且第二个事务可以继续更新最初发现的行。 如果第一个更新事务提交，若该行被第一个更新者删除，则第二个更新事务将忽略该行，否则第二个更新者将试图在该行的已被更新的版本上应用它的操作。该命令的搜索条件（`WHERE`子

句）将被重新计算来看该行被更新的版本是否仍然符合搜索条件。如果符合，则第二个更新者使用该行的已更新版本继续其操作。在SELECT FOR UPDATE和SELECT FOR SHARE的情况下，这意味着把该行的已更新版本锁住并返回给客户端。

带有ON CONFLICT DO UPDATE子句的 INSERT行为类似。在读已提交模式，要插入的 每一行将被插入或者更新。除非有不相干的错误出现，这两种结果之一是肯定 会出现的。如果在另一个事务中发生冲突，并且其效果对于INSERT 还不可见，则UPDATE子句将会 影响那个行，即便那一行对于该命令来说***没有***惯常的可见版本。

带有ON CONFLICT DO NOTHING子句的 INSERT有可能因为另一个效果对 INSERT快照不可见的事务的结果无法让插入进行 下去。再一次，这只是读已提交模式中的情况。

因为上面的规则，正在更新的命令可能会看到一个不一致的快照： 它们可以看到并发更新命令在它尝试更新的相同行上的作用，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涉及复杂搜索条件的命令。不过，它对于更简单的情况是正确的。 例如，考虑用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;
```

如果两个这样的事务同时尝试修改帐号 12345 的余额，那我们很明显希望第二个事务从账户行的已更新版本上开始工作。 因为每个命令只影响一个已经决定了的行，让它看到行的已更新版本不会导致任何麻烦的不一致性。

在读已提交模式中，更复杂的使用可能产生不符合需要的结果。例如： 考虑一个在数据上操作的DELETE命令，它操作的数据正被另一个命令从它的限制条件中移除或者加入，例如，假定website是一个两行的表，两行的website.hits等于9和10：

```
BEGIN;
UPDATE website SET hits = hits + 1;
-- run from another session:  DELETE FROM website WHERE hits = 10;
COMMIT;
```

即便在UPDATE之前有一个website.hits = 10的行，DELETE将不会产生效果。这是因为更新之前的行值9被跳过，并且当UPDATE完成并且DELETE获得一个锁，新行值不再是10而是11，这再也不匹配条件了。

因为在读已提交模式中，每个命令都是从一个新的快照开始的，而这个快照包含在该时刻已提交的事务， 因此同一事务中的后续命令将看到任何已提交的并行事务的效果。以上的焦点在于***单个***命令是否看到数据库的绝对一致的视图。

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并且使用简单。 不过，它不是对于所有情况都够用。做复杂查询和更新的应用可能需要比读已提交模式提供的更严格一致的数据库视图。

13.2.2. 可重复读隔离级别

*可重复读*隔离级别只看到在事务开始之前被提交的数据；它从来看不到未提交的数据或者并行事务在本事务执行期间提交的修改（不过，查询能够看见在它的事务中之前执行的更新，即使它们还没有被提交）。这是比SQL标准对此隔离级别所要求的更强的保证，并且阻止表 13.1中描述的除了序列化异常之外的所有现象。如上面所提到的，这是标准特别允许的，标准只描述了每种隔离级别必须提供的***最小***保护。

这个级别与读已提交不同之处在于，一个可重复读事务中的查询可以看见在***事务***中第一个非事务控制语句开始时的一个快照，而不是事务中当前语句开始时的快照。因此，在一个***单一***事务中的后续SELECT命令看到的是相同的数据，即它们看不到其他事务在本事务启动后提交的修改。

使用这个级别的应用必须准备好由于序列化失败而重试事务。

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： 它们将只找到在事务开始时已经被提交的行。 不过，在被找到时，这样的目标行可能已经被其它并发事务更新（或删除或锁住）。在这种情况下， 可重复读事务将等待第一个更新事务提交或者回滚（如果它还在进行中）。 如果第一个更新事务回滚，那么它的作用将被忽略并且可重复读事务可以继续更新最初发现的行。 但是如果第一个更新事务提交（并且实际更新或删除该行，而不是只锁住它），则可重复读事务将回滚并带有如下消息

```
ERROR:  could not serialize access due to concurrent update
```

因为一个可重复读事务无法修改或者锁住被其他在可重复读事务开始之后的事务改变的行。

当一个应用接收到这个错误消息，它应该中断当前事务并且从开头重试整个事务。在第二次执行中，该事务将见到作为其初始数据库视图一部分的之前提交的改变，这样在使用行的新版本作为新事务更新的起点时就不会有逻辑冲突。

注意只有更新事务可能需要被重试；只读事务将永远不会有序列化冲突。

可重复读模式提供了一种严格的保证，在其中每一个事务看到数据库的一个完全稳定的视图。不过，这个视图并不需要总是和同一级别上并发事务的某些序列化（一次一个）执行保持一致。例如，即使这个级别上的一个只读事务可能看到一个控制记录被更新，

这显示一个批处理已经被完成但是***不能***看见作为该批处理的逻辑组成部分的一个细节记录，因为它读取空值记录的一个较早的版本。如果不小心地使用显式锁来阻塞冲突事务，尝试用运行在这个隔离级别的事务来强制业务规则不太可能正确地工作。

可重复读隔离级别是使用学术数据库文献和一些其他数据库产品中称为*Snapshot Isolation*的已知的技术来实现的。 与使用传统锁技术并降低并发性的系统相比，可以观察到行为和性能方面的差异。 一些其他系统甚至可以提供可重复读取和快照隔离作为具有不同行为的不同隔离级别。 直到SQL标准开发出来之后，数据库研究人员才正式确定区分这两种技术的允许现象，并且超出了本手册的范围。全面的阐述，请参阅[berenson95]。

> **注意**
>
> 在PostgreSQL版本 9.1 之前，一个对于可序列化事务隔离级别的请求会提供和这里描述的完全一样的行为。为了保持可序列化行为，现在应该请求可重复读。

13.2.3. 可序列化隔离级别

*可序列化*隔离级别提供了最严格的事务隔离。这个级别为所有已提交事务模拟序列事务执行；就好像事务被按照序列一个接着另一个被执行，而不是并行地被执行。但是，和可重复读级别相似，使用这个级别的应用必须准备好因为序列化失败而重试事务。事实上，这个隔离级别完全像可重复读一样地工作，除了它会监视一些条件，这些条件可能导致一个可序列化事务的并发集合的执行产生的行为与这些事务所有可能的序列化（一次一个）执行不一致。这种监控不会引入超出可重复读之外的阻塞，但是监控会产生一些负荷，并且对那些可能导致*序列化异常*的条件的检测将触发一次*序列化失败*。

例如，考虑一个表`mytab`，它初始时包含：

```
 class | value
-------+-------
     1 |    10
     1 |    20
     2 |   100
     2 |   200
```

假设可序列化事务 A 计算：

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

并且接着把结果（30）作为一个新行的value插入，新行的class = 2。同时，可序列化事务 B 计算：

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

并得到结果 300，它会将其与class = 1插入到一个新行中。然后两个事务都尝试提交。如果其中一个事务运行在可重复读隔离级别，两者都被允许提交；但是由于不同的执行顺序可能导致不同结果，使用可序列化事务将允许一个事务提交并且将回滚另一个并伴有这个消息：

```
ERROR:  could not serialize access due to read/write dependencies among transactio
```

这是因为，如果 A 在 B 之前执行，B 将计算得到合计值 330 而不是 300，而且相似地另一种顺序将导致 A 计算出一个不同的合计值。

当依赖可序列化事务来阻止异常时，重要的一点是任何从一个持久化用户表读出数据都不被认为是有效的，直到读它的事务已经成功提交为止。即便是对只读事务也是如此，除了在一个*可推迟的*只读事务中读取的数据是读出以后立刻有效的，因为这样的一个事务在开始读取任何数据之前会等待，直到它能获得一个快照保证来避免这种问题为止。在所有其他情况下，应用不能依靠在一个后来被中断的事务中读取的结果；相反，它们应当重试事务直到它成功。

要保证真正的可序列化，PostgreSQL使用了*谓词锁*，这意味着它会保持锁，这些锁让它能够判断在它先运行的情况下，什么时候一个写操作会对一个并发事务中之前读取的结果产生影响。在PostgreSQL中，这些锁并不导致任何阻塞，并且因此***不***会导致一个死锁。它们被用来标识和标志并发可序列化事务之间的依赖性，这些事务的组合可能导致序列化异常。相反，一个想要保证数据一致性的读已提交或可重复读事务可能需要拿走一个在整个表上的锁，这可能阻塞其他尝试使用该表的用户，或者它可能会使用不仅会阻塞其他事务还会导致磁盘访问的SELECT FOR UPDATE或SELECT FOR SHARE。

像大部分其他数据库系统，PostgreSQL中的谓词锁基于被一个事务真正访问的数据。这些谓词锁将显示在pg_locks系统视图中，它们的mode为SIReadLock。这种在一个查询执行期间获得的特别的锁将依赖于该查询所使用的计划，并且在事务过程中多个细粒度锁（如元组锁）可能和少量粗粒度锁（如页面锁）相结合来防止耗尽用于跟踪锁的内存。如果一个READ ONLY事务检测到不会有导致序列化异常的冲突发生，它可以在完成前释放其 SIRead 锁。事实上，READ ONLY事务将常常可以在启动时确立这一事实并避免拿到任何谓词锁。如果你显式地请求一个SERIALIZABLE READ ONLY DEFERRABLE事务，它将阻塞直到它能够确立这一事实（这是***唯一***一种可序列化事务阻塞但可重复读事务不阻塞的情

况）。在另一方面，SIRead 锁常常需要被保持到事务提交之后，直到重叠的读写事务完成。

坚持使用可序列化事务可以简化开发。成功提交的并发可序列化事务的任意集合将得到和一次运行一个相同效果的这种保证意味着，如果你能证明一个单一事务在独自运行时能做正确的事情，则你可以相信它在任何混合的可序列化事务中也能做正确的事情，即使它不知道那些其他事务做了些什么，否则它将不会成功提交。重要的是使用这种技术的环境有一种普遍的方法来处理序列化失败（总是会返回一个 SQLSTATE 值'40001'），因为它将很难准确地预计哪些事务可能为读/写依赖性做贡献并且需要被回滚来阻止序列化异常。读/写依赖性的监控会产生开销，如重启被序列化失败中止的事务，但是作为在该开销和显式锁及SELECT FOR UPDATE或SELECT FOR SHARE导致的阻塞之间的一种平衡，可序列化事务是在某些环境中最好性能的选择。

虽然PostgreSQL的可序列化事务隔离级别只允许并发事务在能够证明有一种串行执行能够产生相同效果的前提下提交，但它却不能总是阻止在真正的串行执行中不会发生的错误产生。尤其是可能会看到由于可序列化事务重叠执行导致的唯一约束被违背的情况，这些情况即便在尝试插入键之前就显式地检查过该键不存在也会发生。避免这种问题的方法是，确保***所有***插入可能会冲突的键的可序列化事务首先显式地检查它们能不能那样做。例如，试想一个要求用户输入新键的应用，它会通过尝试查询用户给出的键来检查键是否已经存在，或者是通过选取现有最大的键并且加一来产生一个新键。如果某些可序列化事务不遵循这种协议而直接插入新键，则也可能会报告唯一约束被违背，即便在并发事务串行执行的情况下不会发生唯一约束被违背也是如此。

当依赖可序列化事务进行并发控制时，为了最佳性能应该考虑一下问题：

- 在可能时声明事务为READ ONLY。
- 控制活动连接的数量，如果需要使用一个连接池。这总是一个重要的性能考虑，但是在一个使用可序列化事务的繁忙系统中这尤为重要。
- 只在一个单一事务中放完整性目的所需要的东西。
- 不要让连接不必要地“闲置在事务中”。配置参数idle_in_transaction_session_timeout可以被用来自动断开拖延会话的连接。
- 在那些由于使用可序列化事务自动提供的保护的地方消除不再需要的显式锁、SELECT FOR UPDATE和SELECT FOR SHARE。
- 当系统因为谓词锁表内存短缺而被强制结合多个页面级谓词锁为一个单一的关系级谓词锁时，序列化失败的比例可能会上升。你可以通过增加max_pred_locks_per_transaction、max_pred_locks_per_relation和max_pred_locks_per_page来避免这种情况。

- 一次顺序扫描将总是需要一个关系级谓词锁。这可能导致序列化失败的比例上升。通过缩减random_page_cost和/或增加cpu_tuple_cost来鼓励使用索引扫描将有助于此。一定要在事务回滚和重启数目的任何减少与查询执行时间的任何全面改变之间进行权衡。

可序列化隔离级别是使用学术数据库文献中称为可序列化快照隔离的技术实现的，通过添加序列化异常事务的检查的方式构建在快照隔离的基础之上。 与使用传统锁技术的其他系统相比，可以观察到行为和性能方面的一些差异。详细信息请参阅[ports12]。

上一页 13.1. 介绍 | 上一级 起始页 | 下一页 13.3. 显式锁定