13.3. 显式锁定

PostgreSQL提供了多种锁模式用于控制对表中数据的并发访问。 这些模式可以用于在MVCC无法给出期望行为的情境中由应用控制的锁。 同样，大多数PostgreSQL命令会自动要求恰当的锁以保证被引用的表在命令的执行过程中 不会以一种不兼容的方式删除或修改（例如，TRUNCATE无法安全地与同一表中上的其他操作并发地执行，因此它在表上获得一个ACCESS EXCLUSIVE 锁来强制这种行为）。

要检查在一个数据库服务器中当前未解除的锁列表，可以使用pg_locks系统视图。 有关监控锁管理器子系统状态的更多信息，请参考第 28 章。

13.3.1. 表级锁

下面的列表显示了可用的锁模式和PostgreSQL自动使用它们的场合。 你也可以用LOCK命令显式获得这些锁。请记住所有这些锁模式都是表级锁，即使它们的名字包含“row”单词（这些名称是历史遗产）。 在一定程度上，这些名字反应了每种锁模式的典型用法 — 但是语意却都是一样的。 两种锁模式之间真正的区别是它们有着不同的冲突锁模式集合（参考表 13.2）。 两个事务在同一时刻不能在同一个表上持有属于相互冲突模式的锁（但是，一个事务决不会和自身冲突。例如，它可以在同一个表上获得ACCESS EXCLUSIVE锁然后接着获取ACCESS SHARE锁）。非冲突锁模式可以由许多事务同时持有。 请特别注意有些锁模式是自冲突的（例如，在一个时刻ACCESS EXCLUSIVE锁不能被多于一个事务持有）而其他锁模式不是自冲突的（例如，ACCESS SHARE锁可以被多个事务持有）。

表级锁模式

ACCESS SHARE

只与ACCESS EXCLUSIVE锁模式冲突。

SELECT命令在被引用的表上获得一个这种模式的锁。通常，任何只***读取***表而不修改它的查询都将获得这种锁模式。

ROW SHARE

与EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。

SELECT FOR UPDATE和SELECT FOR SHARE命令在目标表上取得一个这种模式的锁 （加上在被引用但没有选择FOR UPDATE/FOR SHARE的任何其他表上的ACCESS SHARE锁）。

ROW EXCLUSIVE

与SHARE、 SHARE ROW EXCLUSIVE、 EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。

命令UPDATE、 DELETE和INSERT在目标表上取得这种锁模式（加上在任何其他被引用表上的ACCESS SHARE锁）。通常，这种锁模式将被任何***修改表中数据***的命令取得。

SHARE UPDATE EXCLUSIVE

与SHARE UPDATE EXCLUSIVE、 SHARE、 SHARE ROW EXCLUSIVE、 EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。这种模式保护一个表不受并发模式改变和VACUUM运行的影响。

由VACUUM（不带FULL）、 ANALYZE、 CREATE INDEX CONCURRENTLY、 REINDEX CONCURRENTLY、 CREATE STATISTICS以及某些ALTER INDEX 和 ALTER TABLE的变体获得(详细内容请参考这些命令的文档)。

SHARE

与ROW EXCLUSIVE、 SHARE UPDATE EXCLUSIVE、 SHARE ROW EXCLUSIVE、 EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。这种模式保护一个表不受并发数据改变的影响。

由CREATE INDEX（不带CONCURRENTLY）取得。

SHARE ROW EXCLUSIVE

与ROW EXCLUSIVE、 SHARE UPDATE EXCLUSIVE、 SHARE、 SHARE ROW EXCLUSIVE、 EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。这种模式保护一个表不受并发数据修改所影响，并且是自排他的，这样在一个时刻只能有一个会话持有它。

由CREATE TRIGGER和某些形式的 ALTER TABLE所获得。

EXCLUSIVE

与ROW SHARE、 ROW EXCLUSIVE、 SHARE UPDATE EXCLUSIVE、 SHARE、 SHARE ROW EXCLUSIVE、 EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。这种模式只允许并发的ACCESS

SHARE锁，即只有来自于表的读操作可以与一个持有该锁模式的事务并行处理。

由REFRESH MATERIALIZED VIEW CONCURRENTLY获得。

ACCESS EXCLUSIVE

与所有模式的锁冲突（ACCESS SHARE、 ROW SHARE、 ROW EXCLUSIVE、 SHARE UPDATE EXCLUSIVE、 SHARE、 SHARE ROW EXCLUSIVE、 EXCLUSIVE和ACCESS EXCLUSIVE）。这种模式保证持有者是访问该表的唯一事务。

由ALTER TABLE、 DROP TABLE、 TRUNCATE、 REINDEX、 CLUSTER、 VACUUM FULL和REFRESH MATERIALIZED VIEW（不带CONCURRENTLY）命令获取。 很多形式的ALTER INDEX和ALTER TABLE也在这个层面上获得锁（见ALTER TABLE）。这也是未显式指定模式的LOCK TABLE命令的默认锁模式。

提示

只有一个ACCESS EXCLUSIVE锁阻塞一个SELECT（不带FOR UPDATE/SHARE）语句。

一旦被获取，一个锁通常将被持有直到事务结束。 但是如果在建立保存点之后才获得锁，那么在回滚到这个保存点的时候将立即释放该锁。 这与ROLLBACK取消保存点之后所有的影响的原则保持一致。 同样的原则也适用于在PL/pgSQL异常块中获得的锁：一个跳出块的错误将释放在块中获得的锁。

表 13.2. 冲突的锁模式

请求的锁模式	已存在的锁模式							
	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	SHARE	SHARE ROW EXCL.	EXCL.	ACCESS EXCL.
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCL.					X	X	X	X
SHARE UPDATE EXCL.				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCL.			X	X	X	X	X	X
EXCL.		X	X	X	X	X	X	X

请求的锁模式	已存在的锁模式							
	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	SHARE	SHARE ROW EXCL.	EXCL.	ACCESS EXCL.
ACCESS EXCL.	X	X	X	X	X	X	X	X

13.3.2. 行级锁

除了表级锁以外，还有行级锁，在下文列出了行级锁以及在哪些情境下PostgreSQL会自动使用它们。 行级锁的完整冲突表请见表 13.3。注意一个事务可能会在相同的行上保持冲突的锁，甚至是在不同的子事务中。 但是除此之外，两个事务永远不可能在相同的行上持有冲突的锁。行级锁不影响数据查询，它们只阻塞对同一行的***写入者和加锁者***。 行级锁在事务结束时或保存点回滚的时候释放，就像表级锁一样。

行级锁模式

`FOR UPDATE`

> `FOR UPDATE`会导致由`SELECT`语句检索到的行被锁定，就好像它们要被更新。这可以阻止它们被其他事务锁定、修改或者删除，一直到当前事务结束。也就是说其他尝试`UPDATE`、`DELETE`、`SELECT FOR UPDATE`、`SELECT FOR NO KEY UPDATE`、`SELECT FOR SHARE`或者`SELECT FOR KEY SHARE`这些行的事务将被阻塞，直到当前事务结束。反过来，`SELECT FOR UPDATE`将等待已经在相同行上运行以上这些命令的并发事务，并且接着锁定并且返回被更新的行（或者没有行，因为行可能已被删除）。不过，在一个`REPEATABLE READ`或`SERIALIZABLE`事务中，如果一个要被锁定的行在事务开始后被更改，将会抛出一个错误。进一步的讨论请见第 13.4 节。
>
> 任何在一行上的`DELETE`命令也会获得`FOR UPDATE`锁模式，以及修改某些列的值的`UPDATE`也会获得该锁模式。 当前`UPDATE`情况中被考虑的列集合是那些具有能用于外键的唯一索引的列（所以部分索引和表达式索引不被考虑），但是这种要求未来有可能会改变。

`FOR NO KEY UPDATE`

> 行为与`FOR UPDATE`类似，不过获得的锁较弱：这种锁将不会阻塞尝试在相同行上获得锁的`SELECT FOR KEY SHARE`命令。任何不获取`FOR UPDATE`锁的`UPDATE`也会获得这种锁模式。

`FOR SHARE`

行为与FOR NO KEY UPDATE类似，不过它在每个检索到的行上获得一个共享锁而不是排他锁。一个共享锁会阻塞其他事务在这些行上执行UPDATE、DELETE、SELECT FOR UPDATE或者SELECT FOR NO KEY UPDATE，但是它不会阻止它们执行SELECT FOR SHARE或者SELECT FOR KEY SHARE。

FOR KEY SHARE

行为与FOR SHARE类似，不过锁较弱：SELECT FOR UPDATE会被阻塞，但是SELECT FOR NO KEY UPDATE不会被阻塞。一个键共享锁会阻塞其他事务执行修改键值的DELETE或者UPDATE，但不会阻塞其他UPDATE，也不会阻止SELECT FOR NO KEY UPDATE、SELECT FOR SHARE或者SELECT FOR KEY SHARE。

PostgreSQL不会在内存里保存任何关于已修改行的信息，因此对一次锁定的行数没有限制。 不过，锁住一行会导致一次磁盘写，例如， SELECT FOR UPDATE将修改选中的行以标记它们被锁住，并且因此会导致磁盘写入。

表 13.3. 冲突的行级锁

要求的锁模式	当前的锁模式			
	FOR KEY SHARE	FOR SHARE	FOR NO KEY UPDATE	FOR UPDATE
FOR KEY SHARE				X
FOR SHARE			X	X
FOR NO KEY UPDATE		X	X	X
FOR UPDATE	X	X	X	X

13.3.3. 页级锁

除了表级别和行级别的锁以外，页面级别的共享/排他锁被用来控制对共享缓冲池中表页面的读/写。 这些锁在行被抓取或者更新后马上被释放。应用开发者通常不需要关心页级锁，我们在这里提到它们只是为了完整。

13.3.4. 死锁

显式锁定的使用可能会增加*死锁*的可能性，死锁是指两个（或多个）事务相互持有对方想要的锁。例如，如果事务 1 在表 A 上获得一个排他锁，同时试图获取一个在表 B 上的排他锁， 而事务 2 已经持有表 B 的排他锁，同时却正在请求表 A 上的一个排他锁，那么两个事务就都不能进行下去。PostgreSQL能够自动检测到死锁情况并且会通过中断其

中一个事务从而允许其它事务完成来解决这个问题（具体哪个事务会被中断是很难预测的，而且也不应该依靠这样的预测）。

要注意死锁也可能会作为行级锁的结果而发生（并且因此，它们即使在没有使用显式锁定的情况下也会发生)。考虑如下情况，两个并发事务在修改一个表。第一个事务执行：

```
UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 11111;
```

这样就在指定帐号的行上获得了一个行级锁。然后，第二个事务执行：

```
UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 22222;
UPDATE accounts SET balance = balance - 100.00 WHERE acctnum = 11111;
```

第一个UPDATE语句成功地在指定行上获得了一个行级锁，因此它成功更新了该行。 但是第二个UPDATE语句发现它试图更新的行已经被锁住了，因此它等待持有该锁的事务结束。事务二现在就在等待事务一结束，然后再继续执行。现在，事务一执行：

```
UPDATE accounts SET balance = balance - 100.00 WHERE acctnum = 22222;
```

事务一试图在指定行上获得一个行级锁，但是它得不到：事务二已经持有了这样的锁。所以它要等待事务二完成。因此，事务一被事务二阻塞，而事务二也被事务一阻塞：一个死锁。 PostgreSQL将检测这样的情况并中断其中一个事务。

防止死锁的最好方法通常是保证所有使用一个数据库的应用都以一致的顺序在多个对象上获得锁。在上面的例子里，如果两个事务以同样的顺序更新那些行，那么就不会发生死锁。 我们也应该保证一个事务中在一个对象上获得的第一个锁是该对象需要的最严格的锁模式。如果我们无法提前验证这些，那么可以通过重试因死锁而中断的事务来即时处理死锁。

只要没有检测到死锁情况，寻求一个表级或行级锁的事务将无限等待冲突锁被释放。这意味着一个应用长时间保持事务开启不是什么好事（例如等待用户输入）。

13.3.5. 咨询锁

PostgreSQL提供了一种方法创建由应用定义其含义的锁。这种锁被称为*咨询锁*，因为系统并不强迫其使用 — 而是由应用来保证其正确的使用。咨询锁可用于 MVCC 模型不适用的锁定策略。例如，咨询锁的一种常用用法是模拟所谓“平面文件”数据管理系统典型的悲观锁策略。虽然一个存储在表中的标志可以被用于相同目的，但咨询锁更快、可以避免表膨胀并且会由服务器在会话结束时自动清理。

有两种方法在PostgreSQL中获取一个咨询锁：在会话级别或在事务级别。一旦在会话级别获得了咨询锁，它将被保持直到被显式释放或会话结束。不同于标准锁请求，会话级咨询锁请求不尊重事务语义：在一个后来被回滚的事务中得到的锁在回滚后仍然被保持，并且同样即使调用它的事务后来失败一个解锁也是有效的。一个锁在它所属的进程中可以被获取多次；对于每一个完成的锁请求必须有一个相应的解锁请求，直至锁被真正释放。在另一方面，事务级锁请求的行为更像普通锁请求：在事务结束时会自动释放它们，并且没有显式的解锁操作。这种行为通常比会话级别的行为更方便，因为它使用一个咨询锁的时间更短。对于同一咨询锁标识符的会话级别和事务级别的锁请求按照期望将彼此阻塞。如果一个会话已经持有了一个给定的咨询锁，由它发出的附加请求将总是成功，即使有其他会话在等待该锁；不管现有的锁和新请求是处在会话级别还是事务级别，这种说法都是真的。

和所有PostgreSQL中的锁一样，当前被任何会话所持有的咨询锁的完整列表可以在`pg_locks`系统视图中找到。

咨询锁和普通锁都被存储在一个共享内存池中，它的尺寸由max_locks_per_transaction和max_connections配置变量定义。 必须当心不要耗尽这些内存，否则服务器将不能再授予任何锁。这对服务器可以授予的咨询锁数量设置了一个上限，根据服务器的配置不同，这个限制通常是数万到数十万。

在使用咨询锁方法的特定情况下，特别是查询中涉及显式排序和`LIMIT`子句时，由于 SQL 表达式被计算的顺序，必须小心控制锁的获取。例如：

```
SELECT pg_advisory_lock(id) FROM foo WHERE id = 12345; -- ok
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100; -- danger!
SELECT pg_advisory_lock(q.id) FROM
(
  SELECT id FROM foo WHERE id > 12345 LIMIT 100
) q; -- ok
```

在上述查询中，第二种形式是危险的，因为不能保证在锁定函数被执行之前应用`LIMIT`。这可能导致获得某些应用不期望的锁，并因此在会话结束之前无法释放。 从应用的角度来看，这样的锁将被挂起，虽然它们仍然在`pg_locks`中可见。

提供的操作咨询锁函数在第 9.27.10 节中描述。