POSTGRESQL - LOCKS

Locks or *Exclusive Locks* or *Write Locks* prevent users from modifying a row or an entire table. Rows modified by UPDATE and DELETE are then exclusively locked automatically for the duration of the transaction. This prevents other users from changing the row until the transaction is either committed or rolled back.

The only time when users must wait for other users is when they are trying to modify the same row. If they modify different rows, no waiting is necessary. SELECT queries never have to wait.

The database performs locking automatically. In certain cases, however, locking must be controlled manually. Manual locking can be done by using the LOCK command. It allows specification of a transaction's lock type and scope.

Syntax for LOCK command

The basic syntax for LOCK command is as follows:

```
LOCK [ TABLE ]
name
 IN
lock_mode
```

- **name**: The name (optionally schema-qualified) of an existing table to lock. If ONLY is specified before the table name, only that table is locked. If ONLY is not specified, the table and all its descendant tables (if any) are locked.
- **lock_mode**: The lock mode specifies which locks this lock conflicts with. If no lock mode is specified, then ACCESS EXCLUSIVE, the most restrictive mode, is used. Possible values are: ACCESS SHARE, ROW SHARE , ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE.

> *Once obtained, the lock is held for the remainder of the current transaction. There is no UNLOCK TABLE command; locks are always released at transaction end.*

DeadLocks

Deadlocks can occur when two transactions are waiting for each other to finish their operations. While PostgreSQL can detect them and end them with a ROLLBACK, deadlocks can still be inconvenient. To prevent your applications from running into this problem, make sure to design them in such a way that they will lock objects in the same order.

Advisory Locks

PostgreSQL provides a means for creating locks that have application-defined meanings. These are called *advisory locks*. As the system does not enforce their use, it is up to the application to use them correctly. Advisory locks can be useful for locking strategies that are an awkward fit for the MVCC model.

For example, a common use of advisory locks is to emulate pessimistic locking strategies typical of so called "flat file" data management systems. While a flag stored in a table could be used for the same purpose, advisory locks are faster, avoid table bloat, and are automatically cleaned up by the server at the end of the session.

Example

Consider the table COMPANY having records as follows:

```
testdb# select * from COMPANY;
 id | name  | age | address   | salary
----+-------+-----+-----------+--------
  1 | Paul  |  32 | California|  20000
  2 | Allen |  25 | Texas     |  15000
  3 | Teddy |  23 | Norway    |  20000
  4 | Mark  |  25 | Rich-Mond |  65000
  5 | David |  27 | Texas     |  85000
  6 | Kim   |  22 | South-Hall|  45000
  7 | James |  24 | Houston   |  10000
(7 rows)
```

The following example locks the COMPANY table within the testdb database in ACCESS EXCLUSIVE mode. The LOCK statement works only in a transaction mode:

```
testdb=#BEGIN;
LOCK TABLE company1 IN ACCESS EXCLUSIVE MODE;
```

Above PostgreSQL statement will produce the following result:

```
LOCK TABLE
```

The above message indicates that the table is locked until the transaction ends and to finish the transaction you will have to either rollback or commit the transaction.