

Instituto Politecnico Nacional

Escuela Superior de Computo

Class: Distributed Data Base

Group: 3cm4

Professor: Hernandez Contreras Euler

Student: Urban Reyes Adan

Boleta: 2012630450

Email: [ayanializita@gmail.com](mailto:ayanializita@gmail.com)

Team: Locks

Index

[Theoretical Framework 3](#__RefHeading___Toc502_1246705353)

[Instructions and Screen Shot 5](#__RefHeading___Toc177_1196317776)

[Transaction with Locks 7](#__RefHeading___Toc354_949229052)

[References 9](#__RefHeading___Toc207_1196317776)

Theoretical Framework

Transaction locks are an important feature of any transactional storage engine. There are two types of transaction locks – table locks and row locks. Table locks are used to avoid a table being altered or dropped by one transaction when another transaction is using the table. It is also used to prohibit a transaction from accessing a table, when it is being altered. InnoDB supports [**multiple granularity locking (MGL)**](http://en.wikipedia.org/wiki/Multiple_granularity_locking). So to access rows in a table, intention locks must be taken on the tables.

Row locks are at finer granularity than table level locks, different threads can work on different parts of the table without interfering with each other. This is in contrast with MyISAM where the entire table has to be locked when updating even unrelated rows. Having row locks means that multiple transactions can read and write into a single table. This increases the concurrency level of the storage engine. InnoDB being an advanced transactional storage engine, provides both table and row level transaction locks.

This article will provide information about how transaction locks are implemented in InnoDB storage engine. The lock subsystem of InnoDB provides many services to the overall system, like:

* Creating, acquiring, releasing and destroying row locks.
* Creating, acquiring, releasing and destroying table locks.
* Providing multi-thread safe access to row and table locks.
* Data structures useful for locating a table lock or a row lock.
* Maintaining a list of user threads suspended while waiting for transaction locks.
* Notification of suspended threads when a lock is released.
* Deadlock detection

The lock subsystem helps to isolate one transaction from another transaction. This article will provide information about how transaction locks are created, maintained and used in the InnoDB storage engine. All reference to locks means transaction locks, unless specified otherwise.

**What Happens Internally When Acquiring Table Locks**

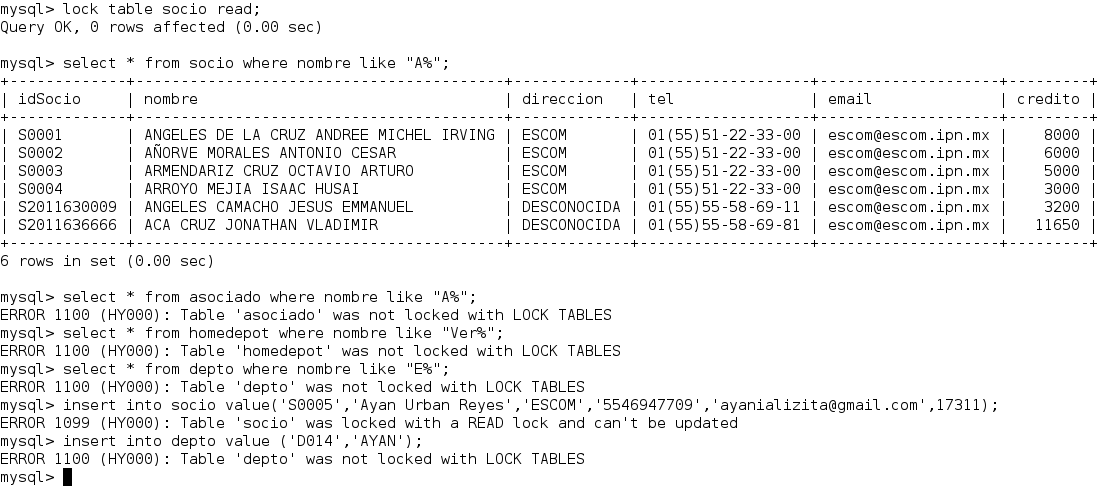
Each table is uniquely identified within the InnoDB storage engine using the table descriptor object of type dict\_table\_t. In this section we will see the steps taken internally by InnoDB to obtain a table lock. The function to refer to is lock\_table() in the source code. Necessary mutexes are taken during these steps to ensure that everything works correctly in a multi-thread environment. This aspect is not discussed here.

1. The request for a table lock comes with the following information – the table to lock (dict\_table\_t), the mode in which to lock (enum lock\_mode), and the transaction which wants the lock (trx\_t).
2. Check whether the transaction is already holding an equal or stronger lock on the given table. Each transaction maintains a list of table locks obtained by itself (trx\_t::trx\_lock\_t::table\_locks). Searching this list for the given table and mode is sufficient to answer this question. If the current transaction is already holding an equal or stronger lock on the table, then the lock request is reported as success. If not, then go to next step.
3. Check if any other transaction has an incompatible lock request in the lock queue of the table. Each table descriptor has a lock queue (dict\_table\_t::locks). Searching this queue is sufficient to answer this question. If some other transaction holds an incompatible lock on the table, then the lock request needs to wait. Waiting for a lock can lead to time out or deadlock error. If there is no contention from other transactions for this table, then proceed further.
4. Allocate a lock struct object (lock\_t). Initialize with table, trx and lock mode information. Add this object to the queue in dict\_table\_t::locks object of the table as well as the trx\_t::trx\_lock\_t::table\_locks.
5. Complete.

You can re-visit the above scenario (௨) and then follow the above steps and verify that the number of lock structs created is 2.

Instructions and Screen Shot

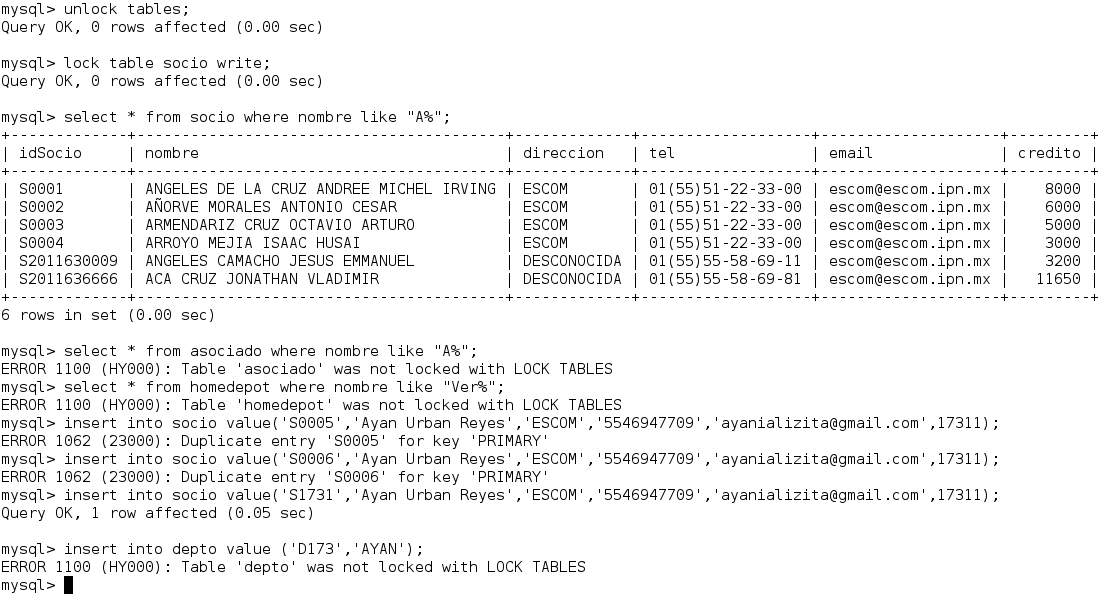
First analyzer what happen the types of locks, start with the block of read.



How can observe when we lock the table with read just can make read on this. Not can write/read over other table.

Now observe the lock write.

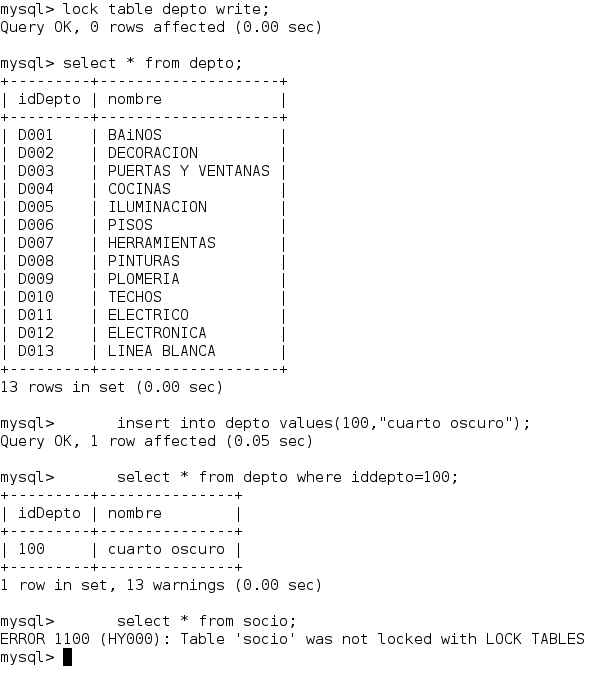
**lock table socio read** → lock table socio on mode read



How can observe when we lock by write can read and write but only on table locked. Not can write or read on other table.

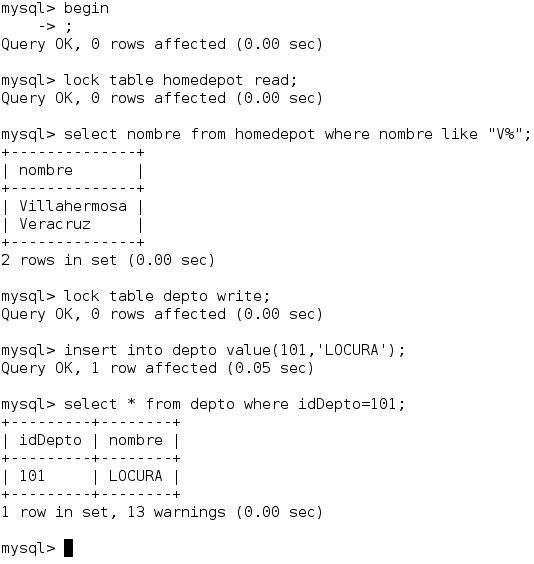
**lock table socio read** → lock table socio on mode read

**unlock tables** → unlock all tables locked

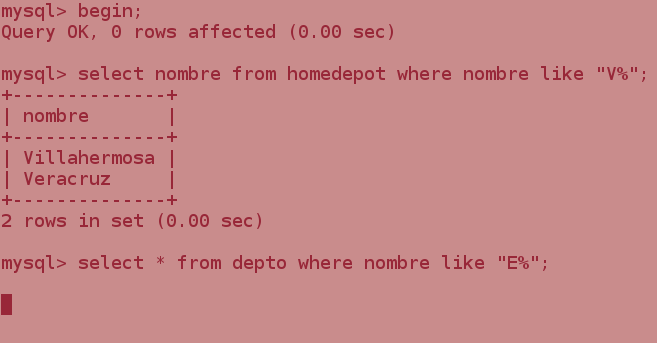


Transaction with Locks

Well start open new terminal.



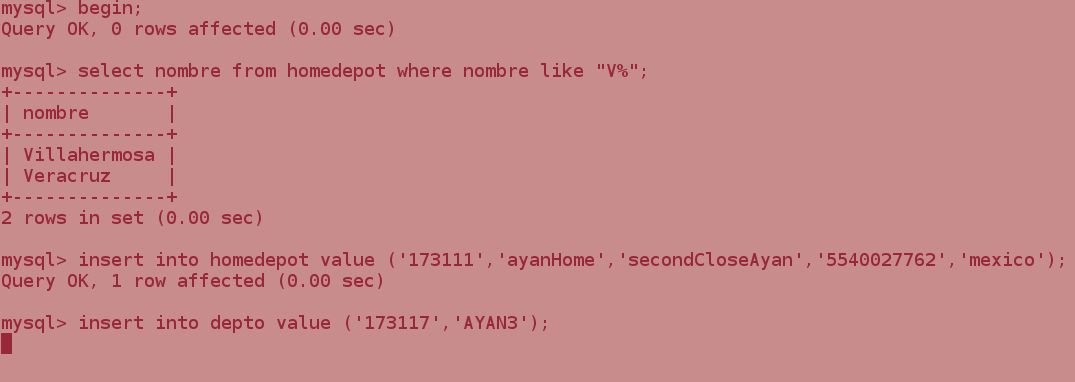
Well in the queries can observe the properties called before time.

Open other terminal and make the next queries.

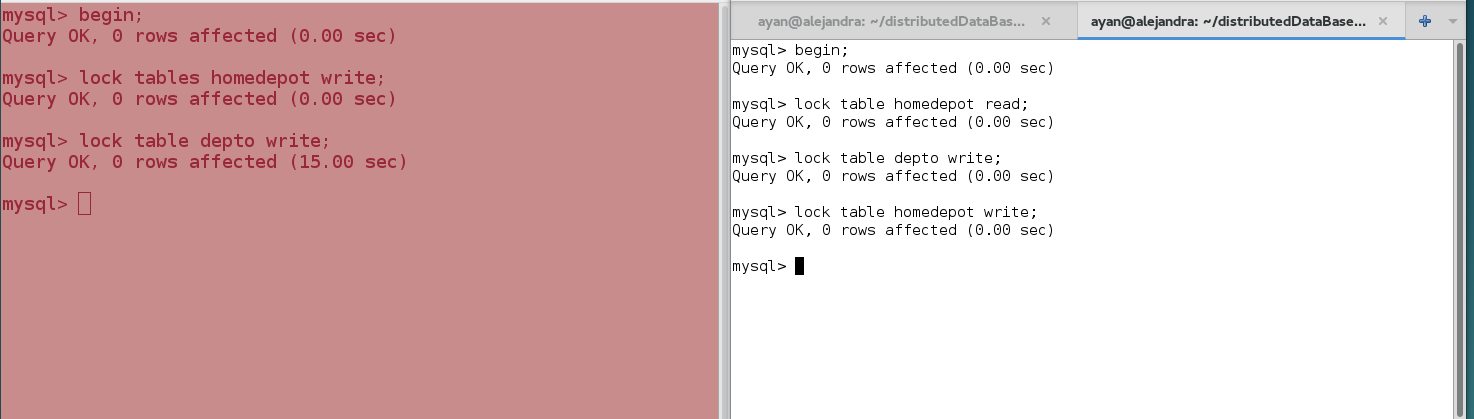
How can observe for first query:

**select nombre from homedepot where nombre like “V%”;** can make because when we lock by read all transaction can operations of read or write, but we lock write any transaction can operations of read/ write also that finish transaction that lock table.

So for unlock terminal 2 (red) need first execute make unlock tables over terminal 1 (white).

****

The image show if we make the insert (operation write) this lock because the terminal 1 lock depto as write. When we execute a unlock tables on terminal 1 (white) the insert make it;

****

we can observe that a transaction only can have one table locket by write is by this that when we lock homedepot on terminal 2 by write unlock terminal 1;

References

<https://blogs.oracle.com/mysqlinnodb/entry/introduction_to_transaction_locks_in>

<http://dev.mysql.com/doc/refman/5.7/en/lock-tables.html>

http://stackoverflow.com/questions/11034504/show-all-current-locks-from-get-lock-on-mysql