Date: 06/06/2012
Procedure: SQL - Locking for Concurrency in MySQL
Source: LINK
Permalink: LINK
Created by: HeelpBook Staff
Document Version: 1.0

SQL – LOCKING FOR CONCURRENCY IN MYSQL

When and how to lock tables

The first and most important point is that the primary use of locking is to solve concurrency problems. If scripts are being implemented that write to the database but aren't multistep operations susceptible to the problems described in the last section, **locks** aren't needed.

Simple scripts that insert one row, delete one row, or update one row, and that don't use results of a previous **SELECT** or data entered by the user as input, don't require a lock.

Locking is required only when developing scripts that first read a value from a database and later write that value to the database.

- Locks are never required for self-contained insert, update, or delete operations such as updating a customer's details, adding a region to the *region* table, or unconditionally deleting an inventory.
- Locking may not be required for all parts of a web database application: parts of the application can still be safely used without violating any locking conditions.

Locks are variables with a special property. With its default settings, each **MySQL** table has an associated lock variable. If a user sets the lock variable for a particular table, no other user can perform particular actions on that table. The user who has set the lock variable *holds* the lock on the table.

In practice, there are two kinds of locks for each table: `READ LOCKs`, when a user is only reading from a table, and **WRITE LOCKs**, when a user is both reading and writing to a table.

Having locks in a **DBMS** leads to four rules of use:

- If a user wants to write to a table, and she is performing a transaction susceptible to a concurrency problem, she must obtain a **WRITE LOCK** on that table.
- If a user only wants to read from a table, and he is performing a transaction susceptible to a concurrency problem, he must obtain a **READ LOCK** on that table.
- If a user requires a lock, she must lock all tables used in the transaction.
- A user must release all locks when a database transaction is complete.

When a user holds a **WRITE LOCK** on a table, no other users can read or write to that table. When a user holds a **READ LOCK** on a table, other users can also read or hold a **READ LOCK**, but no user can write or hold a **WRITE LOCK** on that table.

- **SELECT, UPDATE, INSERT, or DELETE** operations that don't use LOCK TABLES don't proceed if locks are held that would logically prevent their operation. For example, if a user holds a WRITE LOCK on a table, no other user can issue a **SELECT, UPDATE, INSERT, DELETE, or LOCK** operation on that table.

The following segment of an interaction with the **MySQL** command interpreter illustrates the use of locks in a summarization task that requires locking:

```
mysql> LOCK TABLES items READ, temp_report WRITE;
mysql> SELECT sum(price) FROM items WHERE cust_id=1;
+------------+
| sum(price) |
+------------+
|     438.65 |
+------------+
1 row in set (0.04 sec)
mysql> UPDATE temp_report SET purchases=438.65
       WHERE cust_id=1;
mysql> UNLOCK TABLES;
```

In this example, a temporary table called **temp_report** is updated with the result of a **SELECT** operation on the items table. If locks aren't used, the items table can be modified by another user, possibly altering the summary value of **$438.65** used as input to the **UPDATE** operation.

There are two locks obtained for this transaction: first, a **READ LOCK** on items, since we don't need to change items but we don't want another user to make a change to it; and, second, a **WRITE LOCK** on **temp_report**, because we want to change the table, and we don't want other users to read or write to the report while we make changes. The **UNLOCK TABLES** operation releases all locks held; locks can't be progressively released.

It isn't permitted by **MySQL** to lock only one of the two tables used in the transaction above. The following rules apply to locks:

- If a lock is held, all other tables that are to be used must also be locked. Failing to do so results in a **MySQL** error.
- If aliases are used in queries-for example:

  ```
  SELECT * from customer c where c.custid=1
  ```

  the alias must be locked with:

  ```
  LOCK TABLES customer c READ
  ```

  or:

  ```
  LOCK TABLES customer c WRITE
  ```

  (depending on the transaction requirements).
- If different aliases for the same table are used, each different alias must be locked.

In many cases-including those in which locking is required if the tasks are implemented intuitively-locking can be avoided. When designing transactions, careful use of **mysql_insert_id( )** (as opposed to using **max( )** to find the next available identifier), use of temporary summary tables, and updates that are relative (such as **UPDATE customer SET discount=discount*1.1**) are practical techniques to avoid using the output of previous **SELECT** statements.