**Using Oracle Locks to Manage Data Concurrency and Consistency**

how oracle manages data concurrency and consistency by using Locks.

In multi-user systems, many users may update the same information at the same time. Locking allows only one user to update a particular data block; another person cannot modify the same data.

The basic idea of locking is that when a user modifies data through a transaction, that data is locked by that transaction until the transaction is committed or rolled back. The lock is held until the transaction is complete.

When the user commits the first transaction only then it is made available to other users of the system. The locks are thus released by Oracle only if the user does a commit or rollback on a transaction.

There are two basic types of locks

1. Row Level Lock
2. Table Level Lock

The aim of a row level lock is to lock a particular row or set of rows for the transaction. This enables other users of the system to update rows in other tables; they cannot modify the row(s) that are being modified by an earlier transaction.

All the rows of the table can only be viewed. This includes the row, which is currently being modified. The original copy of the row is displayed to other users. Once the transaction is committed the users will be able to see the new values and since the row is now un-locked, they will also be able to modify the same row.

This type of locking is more efficient than table level locking. Once you lock a particular table, others will have to wait to modify any data in any of the rows of that table.

An example of row level or set of rows level locking is the **SELECT...FOR UPDATE** statement. This statement allows you to lock a row or a set of rows that you will be updating. This statement makes sure that any other users do not update the rows that you will be updating. Once the transaction is committed, the rows are made available to other users for modification. You can lock as many rows as you like; there is no limit.

Note that a table lock is also acquired when any row-level lock is obtained by the transaction. This is because no DDL statements can be fired on the table in which the data is currently being modified. For example if you are updating a value in a row, you cannot use the ALTER TABLE command or any such DDL statement(s) for that particular table.

### Examples

Consider the table below;

SQL> desc room\_bookings

Name Null? Type

----------------------------------------- -------- ----------------

ROOM\_CODE VARCHAR2(5)

SCHEDULE VARCHAR2(200)

BOOKING\_CODE VARCHAR2(10)

SQL> Select \* from ROOM\_BOOKING;

ROOM\_ SCHEDULE BOOKING\_CD

---------- ----------------------------------------------------- -------------------

R0010 Booked for Tele-meeting scheduled on 15/07/2003 B00089

R0020 Booked for Tele-conference scheduled on 16/07/2003 B00090

R0010 Booked for Tele-conference scheduled on 17/07/2003 B00091

R0010 Booked for Tele-meeting scheduled on 18/07/2003 B00092

SQL>

### Example I

We want to change the Schedule from Tele-meeting to Tele-conference. We will lock such rows for Update so that other users cannot update them.

declare

v\_schedule varchar2(200);

cursor my\_cur is

select replace(schedule, 'Tele-meeting', 'Tele-conference')

from room\_bookings

for update of schedule;

begin

open my\_cur;

loop

fetch my\_cur into v\_schedule;

exit when my\_cur%NOTFOUND;

update room\_bookings set schedule = v\_schedule

where current of my\_cur;

end loop;

close my\_cur;

end;

/

**Example II**

Open two SQL sessions and in each session issue the following

**Session I**

SQL> SELECT SCHEDULE

FROM room\_bookings

WHERE booking\_code = 'B00090'

FOR UPDATE OF schedule;

**Session II**

SQL> update room\_bookings

set schedule='Test'

where booking\_code='B00090';

The transaction in this session will fail to move ahead. This is because in Session I, the row has been locked. The transaction is Session II will move ahead if we issue the COMMIT or ROLLBACK command.

**Session I**

SQL> SELECT SCHEDULE

FROM room\_bookings

WHERE booking\_code = 'B00090'

FOR UPDATE OF schedule;

ROOM\_ SCHEDULE BOOKING\_CD

---------- --------------------------------------------------- -------------------

R0020 Booked for Tele-conference scheduled on 16/07/2003 B00090

SQL> rollback;

Rollback complete.

SQL>

**Session II**

SQL> update room\_bookings

set schedule='Test'

where booking\_code='B00090';

1 row updated.

SQL>

Explicit locking can be achieved on tables by using the LOCK TABLE command. You can use this command to lock the entire table and unlock it. There are many risks in using this method. For example, a table can be held in locked mode for more time than is required. Thus, the users who want to update the values of this table will have to wait until the lock is released. The application looses the flexibility.

It is better that we let Oracle handle the locking. It is also better to understand that Oracle will release the lock on rows once the transaction is committed or roll backed. Since this entire process is transparent to the users, we have to take care that long running transactions should be committed or else it will cause problems for the system.

### Dead Lock

When two or more users are waiting for access to data that has been locked by each other, it is known as deadlock. When a deadlock occurs, the transactions fail to go ahead - they are stuck. In such cases, Oracle breaks the deadlock by forcing one or more transactions to rollback.

**Note:**

If Oracle's default locking is overridden at any level, the database administrator or application developer should ensure that the overriding locking procedures operate correctly. The locking procedures must satisfy the following criteria: data integrity is guaranteed, data concurrency is acceptable and deadlocks are not possible or are appropriately handled.

### Summary

Locking is a very important part of any database system. It is in our best interests that we let Oracle use the locking implicitly. Only if it is very necessary should you use manual locking. Letting Oracle handle locks will guarantee data concurrency and consistency.