Graphical Representation of Pessimistic Locking and Concurrency Control for Temporal Database using Oracle 12c Enterprise Manager

Jaypalsinh A. Gohil
Research Scholar, Department of Computer Science
MK Bhavnagar University, Bhavnagar, Gujarat, India

Dr. Prashant M. Dolia
Associate Professor & Research Guide, Department of Computer Science
MK Bhavnagar University, Bhavnagar, Gujarat, India

Abstract- Conflicts are common in multiuser database environment. Concurrency can lead to an adverse effect on database if it is not efficiently controlled. Consistency requirements are at high importance in such situations. Locking is an efficient and more reliable mechanism to provide concurrency control in database system. In database system the pessimistic locking approach always provide consistency by serial execution of transactions. In this study we have shown how one can implement the pessimistic locks on data object of temporal database to provide concurrency control and consistency. The study comprises an experiment which shows step by step procedures for acquisition and release of pessimistic lock on data object by series of multiple transactions. The study also highlights the details of other transactions which are waiting in a queue for a transaction to complete its work and waiting for release of lock. In this experiment work we have also highlighted each locking and unlocking situations, concurrency and conflicting situations graphically with the help of Oracle 12C enterprise manager.

Keywords – Concurrency Control, Pessimistic Locking, Temporal Database, Oracle 12c, Oracle enterprise manager

I. INTRODUCTION

The consistency of database can be achieved through efficient concurrency control. The serial executions of transactions always provide consistency by generating consistent database state. In serial execution of transactions all operations of one transaction are executed before any operation from another transaction can execute. The serial execution of transaction preserves consistency [1].

The concurrency control approaches can be categorized as either pessimistic or optimistic. Pessimistic Concurrency Control approach [2,3], prevents execution of concurrent transactions if any conflict is detected between the concurrent transactions. One can also follow Optimistic Concurrency approach which allows the concurrent transactions to proceed at the time of conflict with a risk of having to restart them in case of conflicts [4].

Pessimistic concurrency control can be implemented through locking. If any particular transaction acquires a lock on data object, the data object remains locked for other transactions waiting in a queue for the same data object. When lock is being released by the transaction which initially granted the lock, then other transactions can acquire a lock on the same data object [5].

In pessimistic locking method data to be updated is locked in advance. Once the data to be updated has been locked, the application can make the required changes, and then commit or rollback - during which the lock is automatically dropped. If anyone else attempts to acquire a lock of the same data during this process, they will be forced to wait until the first transaction has completed [6]. This approach is called pessimistic because it assumes that another transaction might change the data between the read and the update. In order to prevent that change from happening and the data inconsistency that would result the read statement locks the data to prevent any other transaction from changing it.

Temporal databases provide a uniform and systematic way of dealing with historical data [7,8]. It provides mechanisms to store and manipulate time-varying information.. Temporal databases encompass all database applications that require some aspect of time when organizing their information. So consistency in temporal database is a critical area needs to be addressed by database administrator. Oracle introduced Oracle Database 12c on June 25, 2013, which is considered to be the important architectural transformation in the legacy of the world's leading database in its 25 years with respect to market presence and dominance [9]. Oracle 12c supports temporal database consistency through efficient locking mechanism.

II. Concurrency Control

Concurrency is a common phenomenon in multiuser database environment. When tow more transactions or users tries to access the same database resource at the same time there will be a conflict, and that conflicting situation is known as concurrency. In such an environment each user must be given the equal priority to perform their operation. We must avoid the situation in which one user is updating an object in the database, while another user is reading it [6].

Concurrency control is the problem of synchronizing concurrent transactions such that the following two properties are achieved:

(1) The consistency of the transaction and database is maintained.

(2) The maximum degree of concurrency of operations is achieved.

Obviously, the serial execution of a set of transaction achieves consistency, if each single transaction is consistent. The efficient concurrency control mechanism should ensure the consistency of the database when transactions are executed concurrently. Concurrency Control is an integral part of database system.

The conflicts can be detected between more than one transaction in the following two ways. In pessimistic method the conflicts are detected before the access of the data object. [10]. In pessimistic method whenever a transaction tries to access some data item, the concurrency control manager (CC Manager) determines the request and will decide whether to grant the permission or not [7].

However, two or more transactions can conflict in a variety of ways: they can require common resources that must be allocated exclusively, or they can access common data items in incompatible modes. In such a case it will generally be necessary to have some transactions wait, or backup, or restart certain transactions, until the transactions they conflict with have run to completion. If the probability of 'conflict is high, then only a few transactions can run concurrently so that all run to completion. In such a case a limit to increased transaction rates will soon be encountered, and this limit is determined by the nature of the transactions [6, 11].

The following compatibility matrix of two transactions shows conflicting and non conflicting sets of operations. Two operations Oi(x) and Oj(x) of transactions Ti and Tj are in conflict if and only if at least one of the operations is a write, i.e.,

- Oi = read(x) and Oj = write(x)
- Oi = write(x) and Oj = read(x)
- Oi = write(x) and Oj = write(x)

The following table shows compatibility matrix of conflicting transactions Ti and Tj.

Table – 1 Compatibility Matrix for Transactions Ti and Tj

TABLE I.

Compatibility Matrix for Ti and Tj		**Ti**	
		read(x)	**write(x)**
Tj	**read(x)**	✓	✗
	write(x)	✗	✗

Generally, a conflict between two operations indicates that their order of execution is important. Read operations do not conflict with each other, hence the ordering of read operations does not matter [13].

III. Locking

A. *Locking Concept –*

Locking is one of the most important and complex topic in oracle. In general explicit and implicit locking schemes are used. Explicit locks can be implemented through LOCK TABLE command, where as implicit locking uses DML statements like insert, update, delete, select for update. The implicit locking is considered to be more efficient.

Two or more transactions can be in deadlock situation. The locking mechanism in Oracle is quite complex and it is hard to find the answer of the question, why session is blocked.

Lock based concurrency control mechanism works on simple lock mechanism to control the concurrent access to the data item. If lock is acquired by the transaction then and then only permission is given to access the data item.

In Lock Based Protocols the Lock mechanism is used for concurrent access to a data item. Permission is given to access a data item only if it is currently holding a lock on that item. Data items can be locked in two modes; either write lock (w) – also called exclusive lock which is denoted by (X) or read lock (r) – also called shared lock which is denoted by (S) [1]. The transaction which performs both read and write from the data item X, exclusive-mode lock is given. The transaction which is only reading the data item, but cannot write on data item, shared-mode lock is given to data item. Transaction can continue its operation only after request is granted [5].

B. *Lock Types –*

Serialize access of conflicting resource can be possible through lock. Using this locking scheme concurrent session will wait for the resource, as in a queue, they are also called enqueues, and this is the term used in wait events to measure the time waited [12]. As our focus is on data only, we target data locks which are also called DML locks because they are used for Data Manipulation Language. The various lock types in DML are as follows:

- Row level locks are called transaction locks (TX) because, even if they are triggered by a concurrent DML on a row, the locked resource is the transaction. TX enqueues are not waiting for a row, but for the completion of the transaction that has updated the row. The TX lock is identified by the transaction id v$transaction
- Table level locks are called table locks (TM) and the locked resource is the database object (table, index, partition…). In addition to DML or DDL, they can be acquired explicitly with the LOCK TABLE statement. The TM locks are identified by an object_id (as in dba_objects).
- User defined locks (UL) resource is not an Oracle object but just a number that has a meaning only for the application. They are managed by the dbms_lock package.

IV. Acquiring pessimistic lock on temporal relation

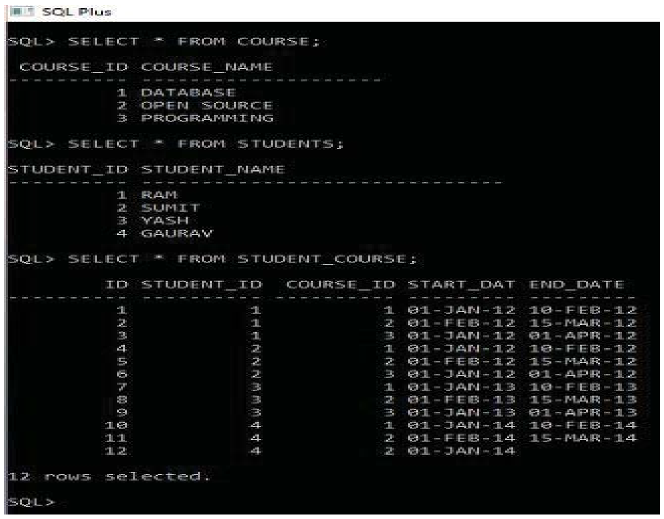
A. *Temporal Table Creation –*

As visible from the below figure, for this study we have designed three relations namely COURSE, STUDENTS and third temporal relation STUDENT_COURSE by using the PERIOD FOR clause at the time of creation of relation STUDENT_COURSE suing following query [13, 14].

```
CREATE TABLE COURSE
(
 COURSE_ID      NUMBER(10) PRIMARY KEY,
 COURSE_NAME    VARCHAR2(20) NOT NULL
);

CREATE TABLE STUDENTS
(
 STUDENT_ID     NUMBER(10) PRIMARY KEY,
 STUDENT_NAME VARCHAR2(30) NOT NULL
);
```

```
CREATE TABLE STUDENT_COURSE
(
 ID NUMBER(10) PRIMARY KEY,
 STUDENT_ID NUMBER(10) REFERENCES
 STUDENTS(STUDENT_ID),
 COURSE_ID NUMBER(10) REFERENCES
 COURSE(COURSE_ID),
 START_DATE DATE,
 END_DATE   DATE,
 PERIOD FOR student_course_period
 (START_DATE,END_DATE)
);
```

The structures of three tables along with its records are as follows:



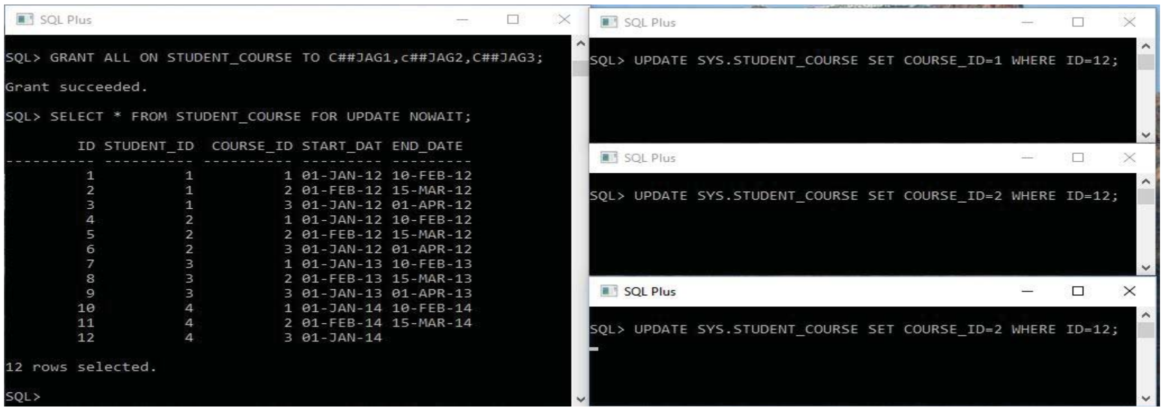
```
SQL> SELECT * FROM COURSE;

 COURSE_ID COURSE_NAME
---------- --------------------
         1 DATABASE
         2 OPEN SOURCE
         3 PROGRAMMING

SQL> SELECT * FROM STUDENTS;

STUDENT_ID STUDENT_NAME
---------- ------------------------------
         1 RAM
         2 SUMIT
         3 YASH
         4 GAURAV

SQL> SELECT * FROM STUDENT_COURSE;

        ID STUDENT_ID  COURSE_ID START_DAT END_DATE
---------- ---------- ---------- --------- ---------
         1          1          1 01-JAN-12 10-FEB-12
         2          1          2 01-FEB-12 15-MAR-12
         3          1          3 01-JAN-12 01-APR-12
         4          2          1 01-JAN-12 10-FEB-12
         5          2          2 01-FEB-12 15-MAR-12
         6          2          3 01-JAN-12 01-APR-12
         7          3          1 01-JAN-13 10-FEB-13
         8          3          2 01-FEB-13 15-MAR-13
         9          3          3 01-JAN-13 01-APR-13
        10          4          1 01-JAN-14 10-FEB-14
        11          4          2 01-FEB-14 15-MAR-14
        12          4          2 01-JAN-14

12 rows selected.

SQL>
```

Figure 1. Structure of three table along with its data

B. *Transaction view at the start of experiment –*

At the beginning of experiment as shown in the below figure, we have started four distinct sessions for four different users namely SYS, C##JAG1, C#JAG2 and C#JAG3. Initially as visible for the below image the owner of the STUDENT_COURSE table grants ALL permissions of STUDENT_COURSE table to other three users namely C##JAG1, C#JAG2 and C#JAG3. After granting the permissions the SYS user issues a SELECT command on STUDENT_COURSE relation using FOR UPDATE NOWAIT option.

The user C##JAG1 issues update command on STUDENT_COURSE relation, but it has to wait for some time since already a pessimistic lock is being acquired by SYS user on object STUDENT_COURSE. The cursor of window of user C##JAG1 is just blinking and not showing SQL prompt, which is indicating that the session invoked by user C##JAG1 is waiting for SYS to finish then and then it can continue. In the similar way the other users C##JAG2 and C##JAG3 are issued a same update command and waiting in a queue.



```
SQL> GRANT ALL ON STUDENT_COURSE TO C##JAG1,c##JAG2,C##JAG3;

Grant succeeded.

SQL> SELECT * FROM STUDENT_COURSE FOR UPDATE NOWAIT;

        ID STUDENT_ID  COURSE_ID START_DAT END_DATE
---------- ---------- ---------- --------- ---------
         1          1          1 01-JAN-12 10-FEB-12
         2          1          2 01-FEB-12 15-MAR-12
         3          1          3 01-JAN-12 01-APR-12
         4          2          1 01-JAN-12 10-FEB-12
         5          2          2 01-FEB-12 15-MAR-12
         6          2          3 01-JAN-12 01-APR-12
         7          3          1 01-JAN-13 10-FEB-13
         8          3          2 01-FEB-13 15-MAR-13
         9          3          3 01-JAN-13 01-APR-13
        10          4          1 01-JAN-14 10-FEB-14
        11          4          2 01-FEB-14 15-MAR-14
        12          4          3 01-JAN-14

12 rows selected.

SQL>
```

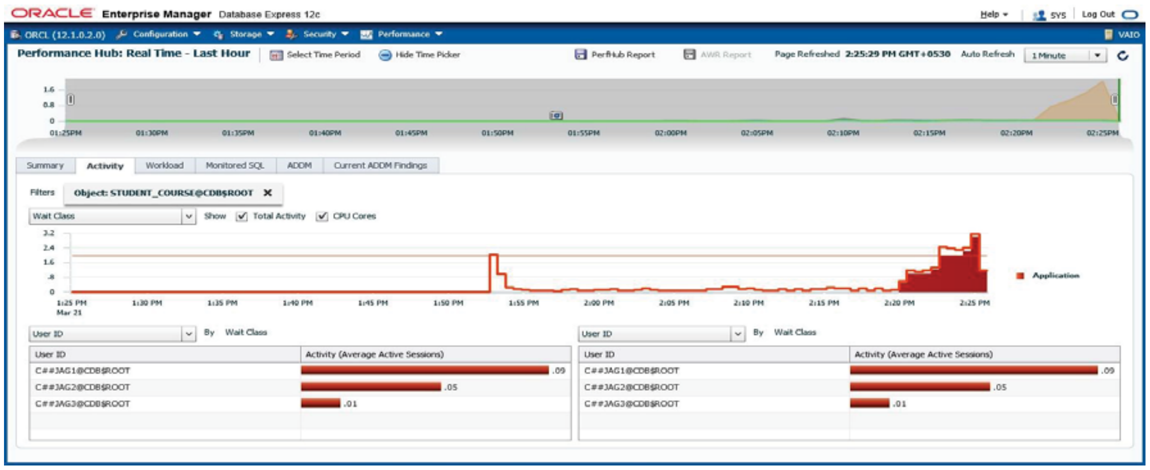
```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=1 WHERE ID=12;
```

```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=12;
```

```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=12;
```

Figure 2. Transaction view at the start of experiment

The following image shows the view of oracle enterprise manager of oracle 12c at the start of experiment.



User ID	Activity (Average Active Sessions)
C##JAG1@CDB$ROOT	.09
C##JAG2@CDB$ROOT	.05
C##JAG3@CDB$ROOT	.01

User ID	Activity (Average Active Sessions)
C##JAG1@CDB$ROOT	.09
C##JAG2@CDB$ROOT	.05
C##JAG3@CDB$ROOT	.01

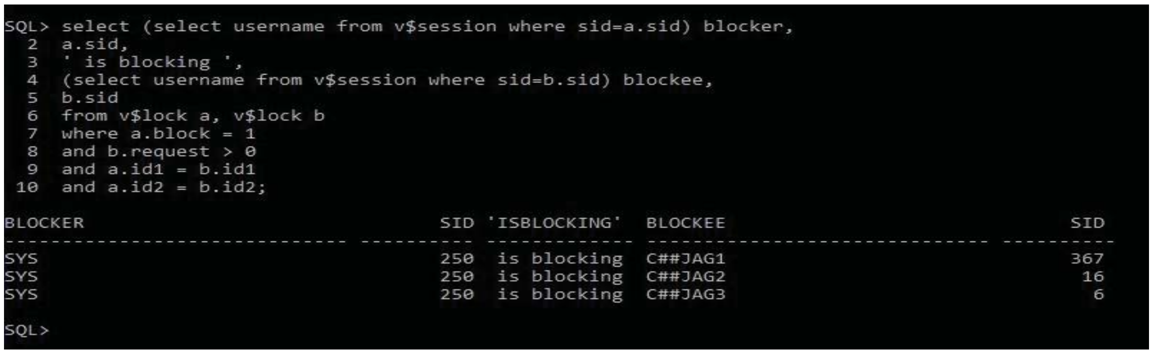
Figure 3. View of Oracle 12c Enterprise Manager the start of experiment

C. *Implementation of pessimistic lock using FOR UPDATE NOWAIT option–*

The "for update nowait" statement locks the row against updates by other sessions. This is why this approach is called pessimistic locking. We lock the row before we attempt to update it because we doubt that the row will remain unchanged otherwise. We'll get three outcomes from this statement:

(1) If the underlying data has not changed, we get our row back and this row will be locked from updates by others (but not reads). (2) If another user is in the process of modifying that row, we Resource Busy error. We are blocked and must wait for the other user to finish with it. (3) If in the time between selecting the data and indicating our intention to update, someone has already changed the row then we will get zero rows back. The data will be stale. The application needs to re-query the data and lock it before allowing the end user to modify any of the data in order to avoid a lost update scenario [15].

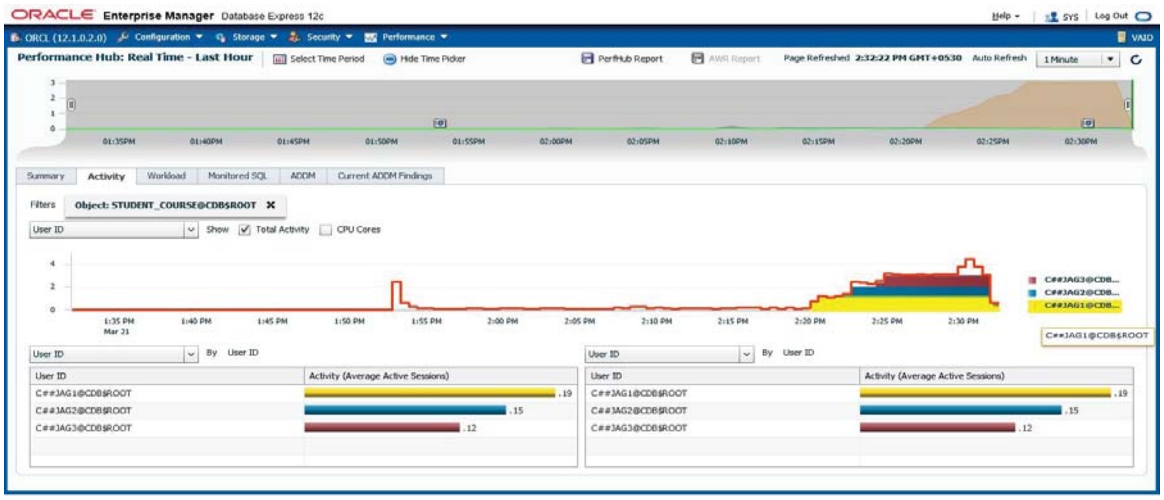
As per the mechanism of SELECT statement with FOR UPDATE NOWAIT clause imposes a pessimistic lock on the object. So pessimistic lock is exists on the object until and unless the session or transaction which holds the locks commits itself. So lock will be released when transaction commits. Hence in our experiment the pessimistic lock is held by SYS user and other user's i.e C#JAG1, C#JAG2 and C#JAG3 are waiting for the lock to be released on STUDENT_COURSE table by SYS release. So we can say that SYS user's session is blocking C#JAG1, C##JAG2 and C#JAG3 users respectively. The following figure shows the details about who is blocking whom.



```
SQL> select (select username from v$session where sid=a.sid) blocker,
  2  a.sid,
  3  ' is blocking ',
  4  (select username from v$session where sid=b.sid) blockee,
  5  b.sid
  6  from v$lock a, v$lock b
  7  where a.block = 1
  8  and b.request > 0
  9  and a.id1 = b.id1
 10  and a.id2 = b.id2;

BLOCKER                               SID 'ISBLOCKING'  BLOCKEE                               SID
------------------------------ ---------- ------------- ------------------------------ ----------
SYS                                   250  is blocking  C##JAG1                               367
SYS                                   250  is blocking  C##JAG2                                16
SYS                                   250  is blocking  C##JAG3                                 6

SQL>
```

Figure 4. Who is blocking whom

The following image shows the view of oracle enterprise manager of oracle 12c at the start of four distinct sessions. It is visible from the image that the C##JAG1, C##JAG2 and C##JAG3 are waiting for SYS to release the lock on STUDENT_COURSE object.



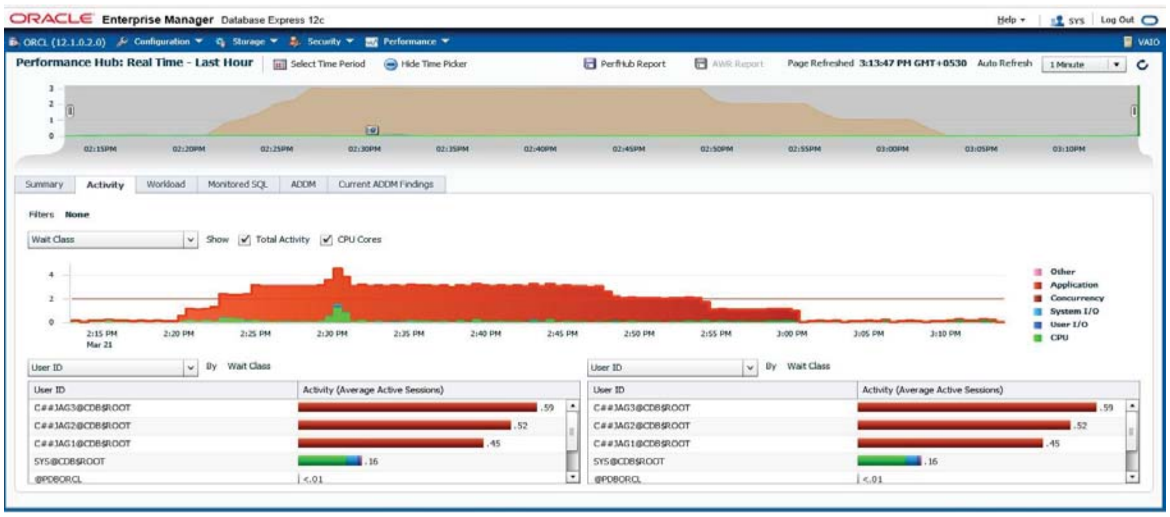
User ID	Activity (Average Active Sessions)
C##JAG1@CDB$ROOT	.19
C##JAG2@CDB$ROOT	.15
C##JAG3@CDB$ROOT	.12

User ID	Activity (Average Active Sessions)
C##JAG1@CDB$ROOT	.19
C##JAG2@CDB$ROOT	.15
C##JAG3@CDB$ROOT	.12

Figure 5. Three user sessions in waiting state

D. Concurrency Situation–

By implementing pessimistic locking using FOR UPDATE NOWAIT option generates the problem of concurrency. Since the same resource needed to be accessed by multiple transactions or users at the same time the concurrent update operation is not possible. The resource is blocked by one transaction and other transactions are waiting in a queue for release of lock. So at a time more than one transaction can be in waiting state, which results in concurrency problem. Concurrent executions of update transactions on the locked recourse are not permitted in pessimistic locking.


ORACLE Enterprise Manager Database Express 12c
Help SYS Log Out
ORCL (12.1.0.2.0) Configuration Storage Security Performance VAIO
Performance Hub: Real Time - Last Hour
Select Time Period
Hide Time Picker
PerfHub Report
AWR Report
Page Refreshed 3:13:47 PM GMT+0530 Auto Refresh 1 Minute
Summary Activity Workload Monitored SQL ADDM Current ADDM Findings
Filters None
Wait Class Show Total Activity CPU Cores
Other
Application
Concurrency
System I/O
User I/O
CPU
User ID By Wait Class

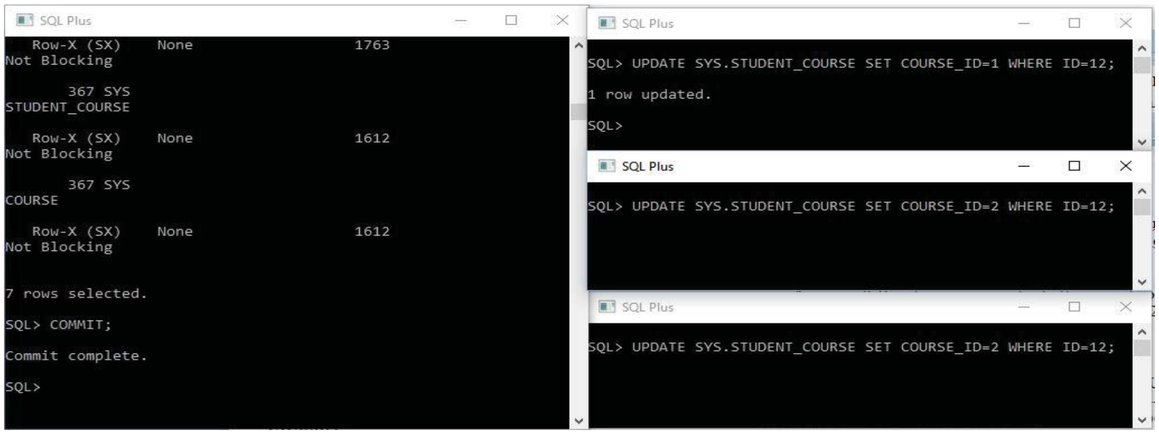
User ID	Activity (Average Active Sessions)
C##JAG3@CDB$ROOT	.59
C##JAG2@CDB$ROOT	.52
C##JAG1@CDB$ROOT	.45
SYS@CDB$ROOT	.16
@PDBORCL	<.01

Figure 6. Enterprise Manager showing concurrency

The above image shows the concurrency environment for proposed experiment. It shows three transactions of users C##JAG1, C##JAG2 and C##JAG3 are waiting concurrently in a queue for SYS to release the lock. This results in concurrency, which is highlighted by red colored region by enterprise manager.

V. RELEASING PESSIMISTIC LOCK ON TEMPORAL RELATION

Through a COMMIT command a pessimistic lock on an object can be released. It evident from the below image when SYS users gives the COMMIT COMMAND, it releases a lock and frees the STUDENT_COURSE relation as a resource. Now in a queue user C##JAG1 is waiting for the lock to be released. Since at this point the lock is released by SYS it can be granted to the user C##JAG1. As lock permission is granted to C##JAG1 and the update operation of C##JAG1 is executed successfully as shown in below figure. The output shows the result as 1 row is updated. So at this point the lock is acquired by use C##JAG1 on STUDENT_COURSE table and other two user's namely C##JAG2 and C##JAG3 are still waiting in a queue for user C##JAG1 to commit itself.



```
   Row-X (SX)    None                   1763
Not Blocking

       367 SYS
STUDENT_COURSE

   Row-X (SX)    None                   1612
Not Blocking

       367 SYS
COURSE

   Row-X (SX)    None                   1612
Not Blocking


7 rows selected.

SQL> COMMIT;

Commit complete.

SQL>
```

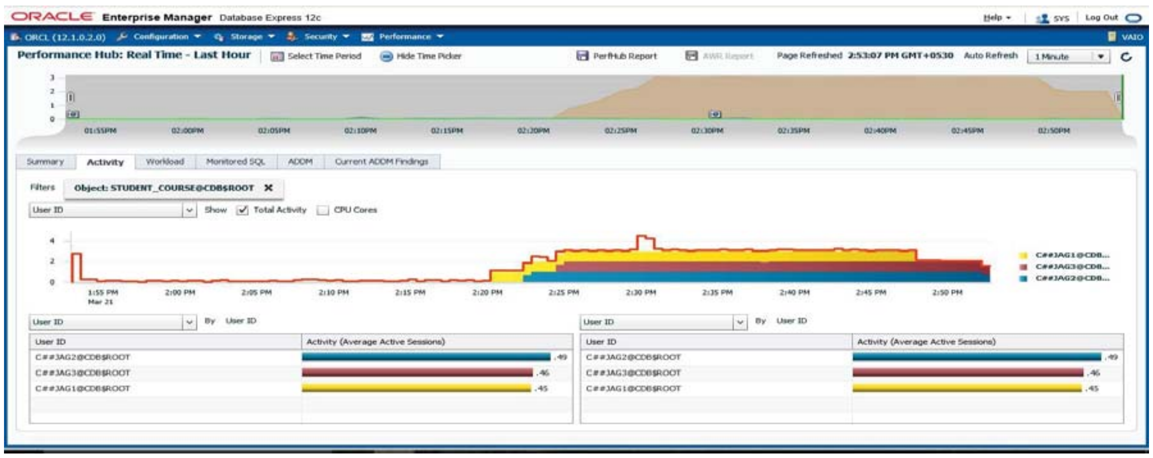
```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=1 WHERE ID=12;

1 row updated.

SQL>
```

```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=12;
```

```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=12;
```

Figure 7. View of user sessions at the time of commit operation of SYS

A view of oracle enterprise manager when user SYS commits. As visible for the below image of EM the session of user C##JAG1 come out form the waiting phase and resumes operations in normal manner (see the yellow bar for user C#JAG1).



User ID By User ID

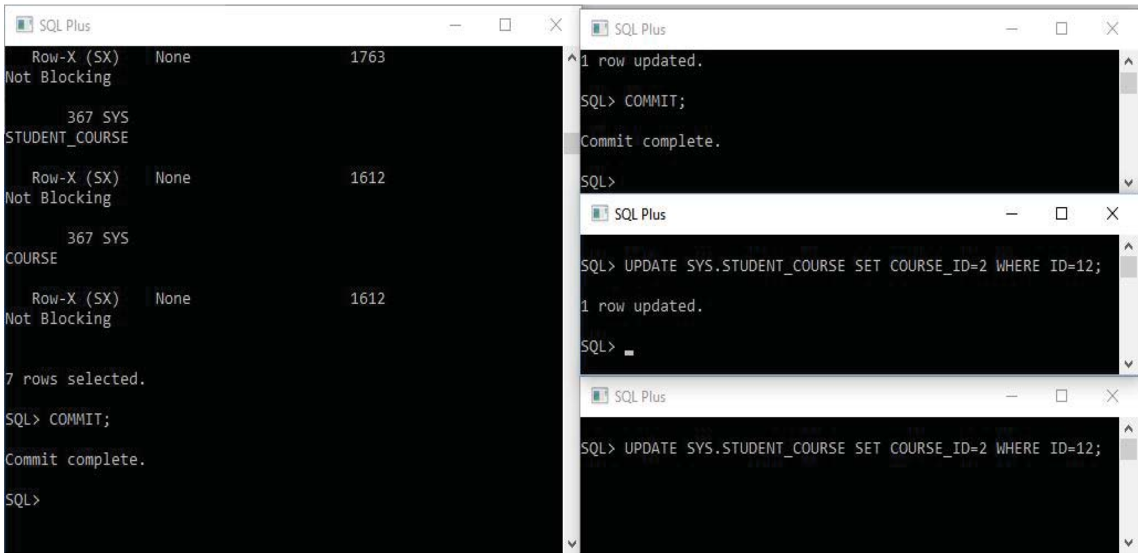
User ID	Activity (Average Active Sessions)
C##JAG2@CDB$ROOT	.49
C##JAG3@CDB$ROOT	.46
C##JAG1@CDB$ROOT	.45

User ID By User ID

User ID	Activity (Average Active Sessions)
C##JAG2@CDB$ROOT	.49
C##JAG3@CDB$ROOT	.46
C##JAG1@CDB$ROOT	.45

Figure 8. View of enterprise manager at the time of commit operation of SYS user

Now in the similar manner to SYS, in the next turn we are committing transaction for user C##JAG1. Now lock is released by C#JAG1 and can be acquired by C##JAG2, which is the next transaction waiting in a queue to acquire a lock on STUDENT_COURSE recourse. The situation is highlighted in below mention image. Now the user C##JAG2 can execute the UPDATE statement on STUDENT_COURSE table.



```
   Row-X (SX)    None                  1763
Not Blocking

       367 SYS
STUDENT_COURSE

   Row-X (SX)    None                  1612
Not Blocking

       367 SYS
COURSE

   Row-X (SX)    None                  1612
Not Blocking


7 rows selected.

SQL> COMMIT;

Commit complete.

SQL>
```

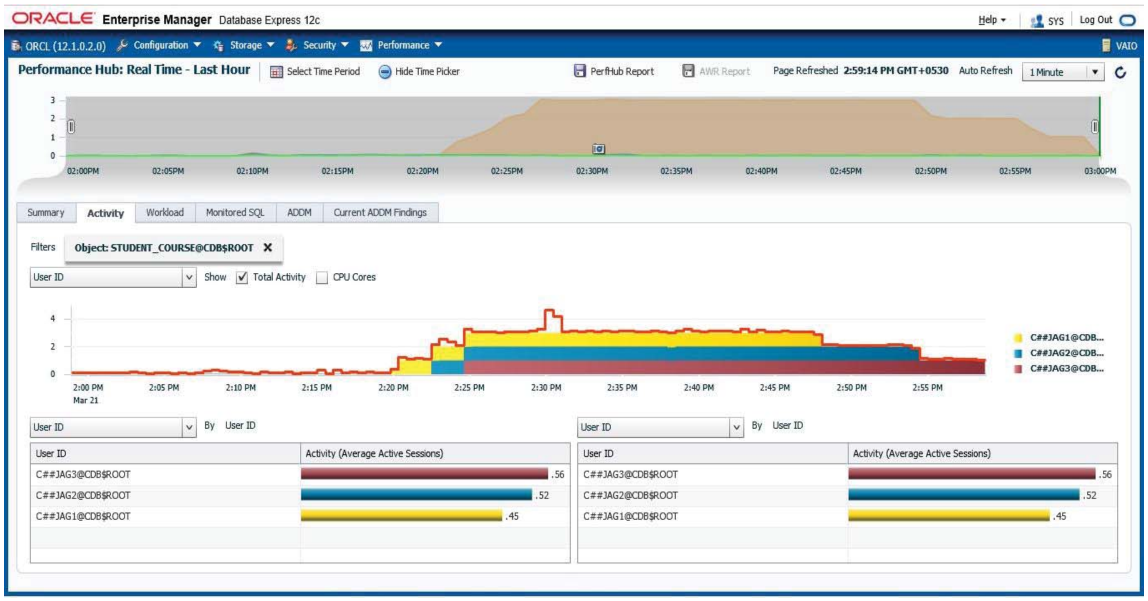
```
1 row updated.

SQL> COMMIT;

Commit complete.

SQL>
```

```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=12;

1 row updated.

SQL> _
```

```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=12;
```

Figure 9. View of user sessions at the time of commit operation of C##JAG1

The snapshot of oracle enterprise manager after the commit operation of user C##JAG1.



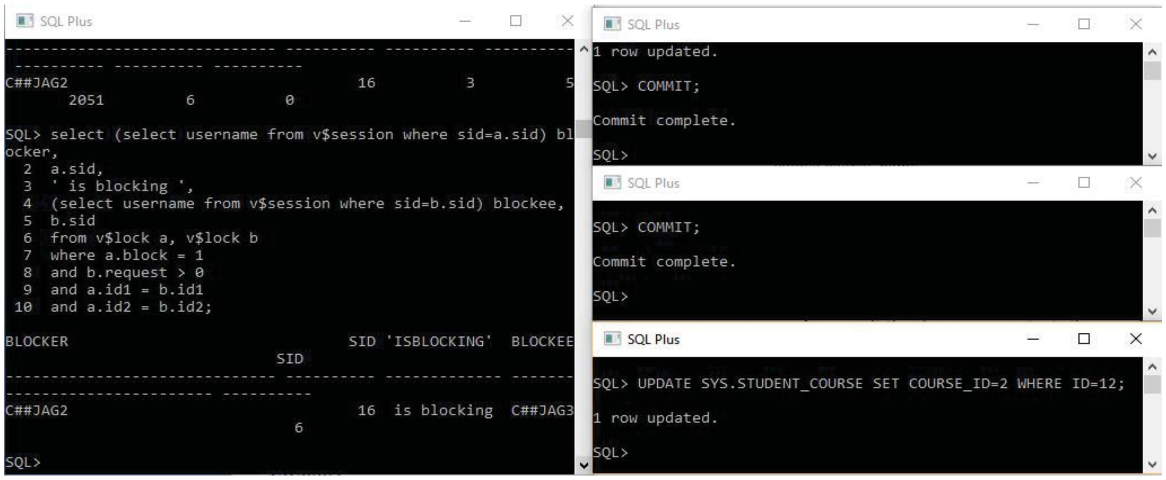
User ID	Activity (Average Active Sessions)
C##JAG3@CDB$ROOT	.56
C##JAG2@CDB$ROOT	.52
C##JAG1@CDB$ROOT	.45

User ID	Activity (Average Active Sessions)
C##JAG3@CDB$ROOT	.56
C##JAG2@CDB$ROOT	.52
C##JAG1@CDB$ROOT	.45

Figure 10. View of enterprise manager at the time of commit operation of C##JAG1 user

As evident for the above image now the session of user C##JAG2 is resumes its operation in normal manner (see the blue bar for user C#JAG2).

Finally we do the similar commit operation for user session C##JAG2 and the effect is same. Lock on object STUDENT_COURSE is being released by C##JAG2 and acquired by C##JAG3. The commit operation of user C##JAG2 is shown below.



```
------------------------------ ---------- ---------- ----------
---------- ---------- ----------
C##JAG2                                16          3          5
      2051          6          0

SQL> select (select username from v$session where sid=a.sid) bl
ocker,
  2  a.sid,
  3  ' is blocking ',
  4  (select username from v$session where sid=b.sid) blockee,
  5  b.sid
  6  from v$lock a, v$lock b
  7  where a.block = 1
  8  and b.request > 0
  9  and a.id1 = b.id1
 10  and a.id2 = b.id2;

BLOCKER                               SID 'ISBLOCKING'  BLOCKEE
                                 SID
------------------------------ ---------- ------------- -------
------------------------- ----------
C##JAG2                                16  is blocking  C##JAG3
                                   6

SQL>
```

```
1 row updated.

SQL> COMMIT;

Commit complete.

SQL>
```

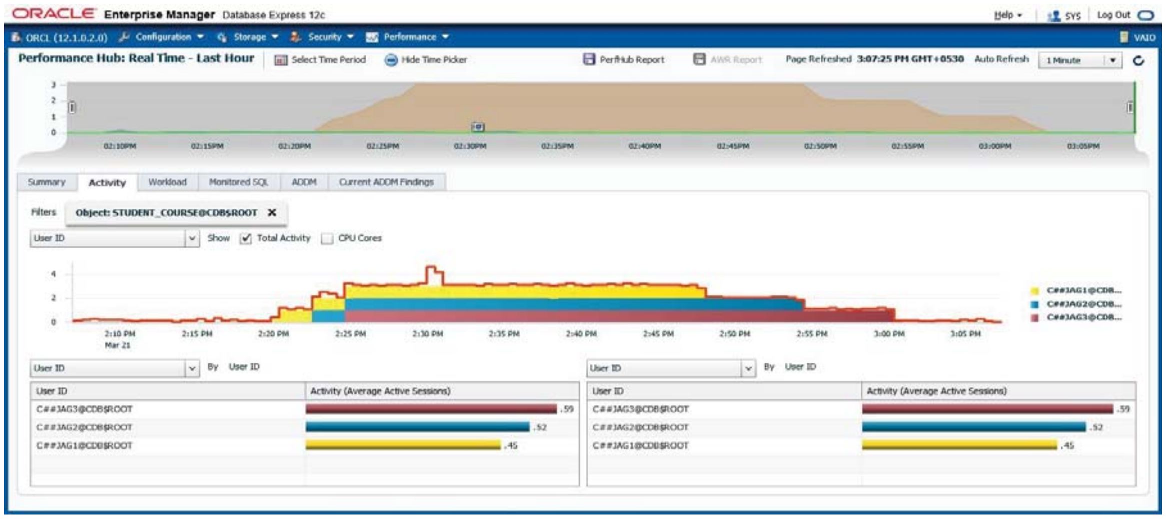
```
SQL> COMMIT;

Commit complete.

SQL>
```

```
SQL> UPDATE SYS.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=12;

1 row updated.

SQL>
```

Figure 11. View of user sessions at the time of commit operation of C##JAG2

Now the update operation of C##JAG3 user can no longer has to wait and it finishes itself by resulting a message 1 row is updated. Now at this point lock is granted to C##JAG3.

After commit operation of C##JAG2 the snapshot of enterprise manager shows us the following picture. Now it is visible clearly that all user sessions are running independently and not waiting for each other to acquire or to release any lock.



Page Refreshed 3:07:25 PM GMT+0530

Auto Refresh: 1 Minute

Filters: Object: STUDENT_COURSE@CDB$ROOT

User ID; Show: [✓ Total Activity]

User ID By User ID

User ID	Activity (Average Active Sessions)
C##JAG3@CDB$ROOT	.59
C##JAG2@CDB$ROOT	.52
C##JAG1@CDB$ROOT	.45

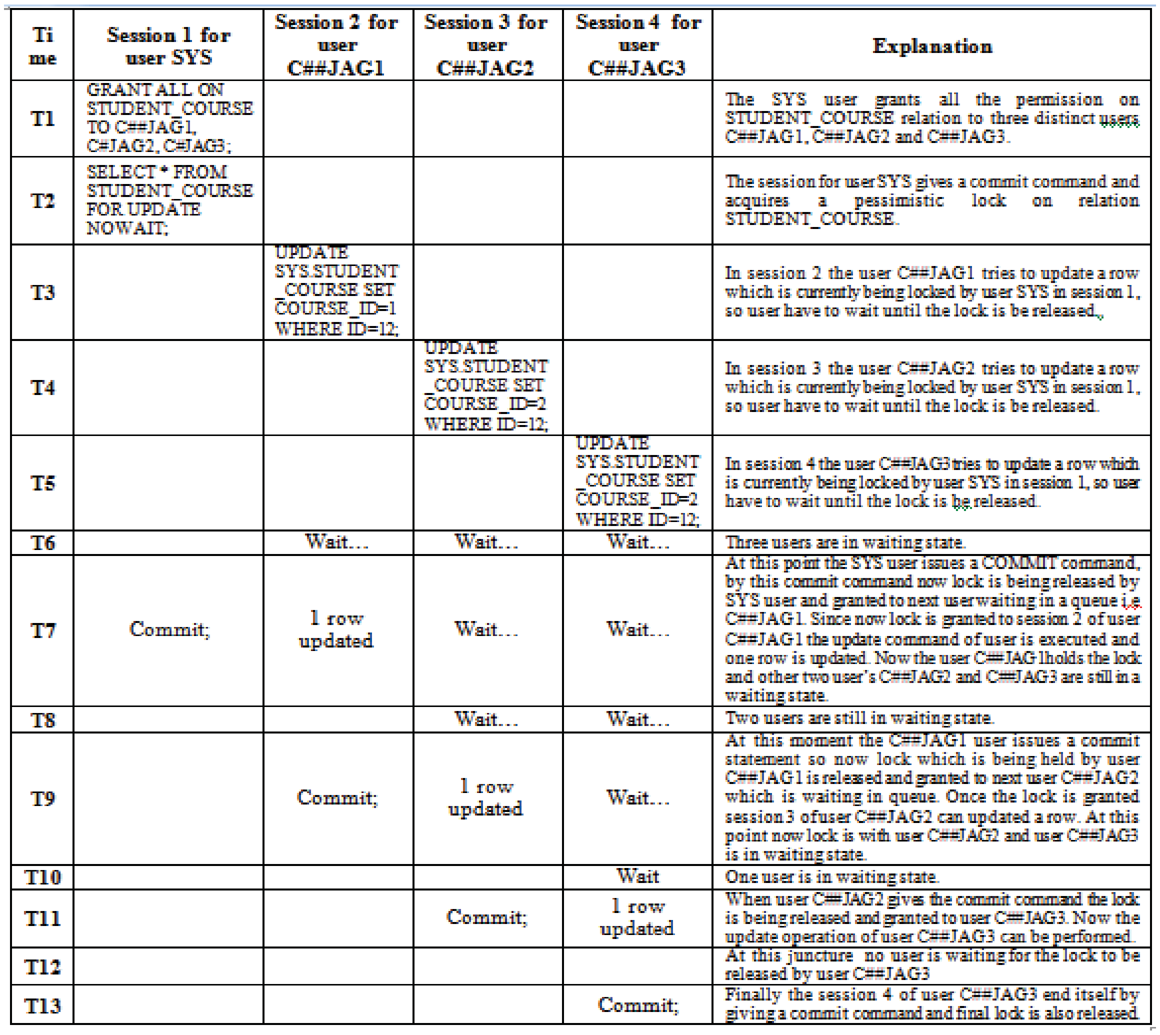
User ID By User ID

User ID	Activity (Average Active Sessions)
C##JAG3@CDB$ROOT	.59
C##JAG2@CDB$ROOT	.52
C##JAG1@CDB$ROOT	.45

Figure 12. View of enterprise manager at the time of commit operation of C##JAG2 user

VI. Session Outline View

The following session outline view of user sessions shows the step by step sequence of events when four different sessions of four distinct users tries to modify the same resource roughly the same time.



Ti me	Session 1 for user SYS	Session 2 for user C##JAG1	Session 3 for user C##JAG2	Session 4 for user C##JAG3	Explanation
T1	GRANT ALL ON STUDENT_COURSE TO C##JAG1, C#JAG2, C#JAG3;				The SYS user grants all the permission on STUDENT_COURSE relation to three distinct users C##JAG1, C##JAG2 and C##JAG3.
T2	SELECT * FROM STUDENT_COURSE FOR UPDATE NOWAIT;				The session for user SYS gives a commit command and acquires a pessimistic lock on relation STUDENT_COURSE.
T3		UPDATE SYS.STUDENT _COURSE SET COURSE_ID=1 WHERE ID=12;			In session 2 the user C##JAG1 tries to update a row which is currently being locked by user SYS in session 1, so user have to wait until the lock is be released.
T4			UPDATE SYS.STUDENT _COURSE SET COURSE_ID=2 WHERE ID=12;		In session 3 the user C##JAG2 tries to update a row which is currently being locked by user SYS in session 1, so user have to wait until the lock is be released.
T5				UPDATE SYS.STUDENT _COURSE SET COURSE_ID=2 WHERE ID=12;	In session 4 the user C##JAG3 tries to update a row which is currently being locked by user SYS in session 1, so user have to wait until the lock is be released.
T6		Wait…	Wait…	Wait…	Three users are in waiting state.
T7	Commit;	1 row updated	Wait…	Wait…	At this point the SYS user issues a COMMIT command, by this commit command now lock is being released by SYS user and granted to next user waiting in a queue i.e. C##JAG1. Since now lock is granted to session 2 of user C##JAG1 the update command of user is executed and one row is updated. Now the user C##JAG1holds the lock and other two user's C##JAG2 and C##JAG3 are still in a waiting state.
T8			Wait…	Wait…	Two users are still in waiting state.
T9		Commit;	1 row updated	Wait…	At this moment the C##JAG1 user issues a commit statement so now lock which is being held by user C##JAG1 is released and granted to next user C##JAG2 which is waiting in queue. Once the lock is granted session 3 of user C##JAG2 can updated a row. At this point now lock is with user C##JAG2 and user C##JAG3 is in waiting state.
T10				Wait	One user is in waiting state.
T11			Commit;	1 row updated	When user C##JAG2 gives the commit command the lock is being released and granted to user C##JAG3. Now the update operation of user C##JAG3 can be performed.
T12					At this juncture no user is waiting for the lock to be released by user C##JAG3
T13				Commit;	Finally the session 4 of user C##JAG3 end itself by giving a commit command and final lock is also released.

Figure 13. Session outline view

VII. Session Waiting Time Statistics

The following chart shows average session waiting time for each user. As visible from the chart the session waiting time for session 2 of user C##JAG1 is 0.45 sec, the average waiting session time for session 3 of user C##JAG2 is 0.52 sec and for sessions 4 of user C##JAG3 is 0.59 sec. So, form the chart it is evident that the user C##JAG3 has waited in a queue for longest period of time for the locked resource as compared to other two users C##JAG1 and C##JAG2. C##JAG1 got the locked resource earlier than other two users i.e. C##JAG2 and C##JAG3.

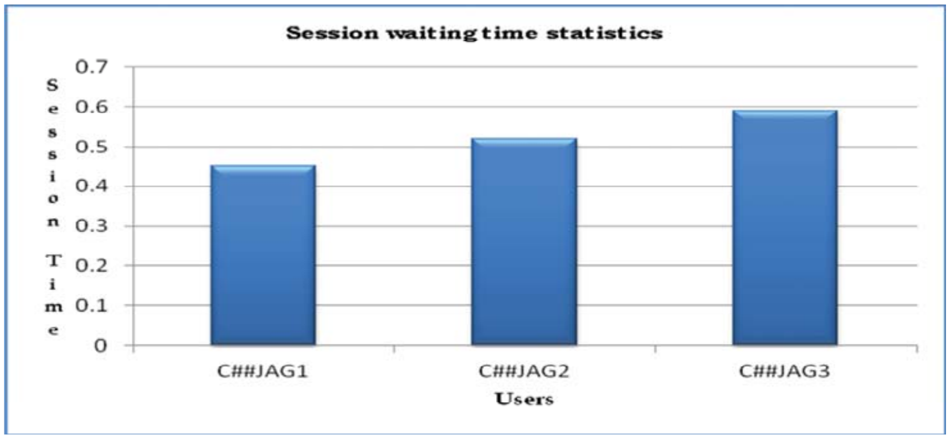


Figure 14. Session waiting time statistics

VIII. Summary

In the pessimistic locking technique the data to be updated is locked in advance. This can be done by using SELECT…FOR UPDATE clause. In our experiment once the data to be updated has been locked by the user session using SELECT...FOR UPDATE statement, and then it can be changed by user session who holds the lock and at the end it commits. At the time of commit operation the lock is automatically dropped. If any user tries to acquire the lock on the object or data item during this process, they will be forced to wait until the first transaction has completes its operation and release the lock. The approach is termed as "pessimistic" because it assumes that any other transaction or user session might change the data between the read and update operation. To prevent the update operation of other transactions and data consistency it would result, the read statement locks the data to prevent any other transaction from updating or changing it.

IX. Conclusion

Concurrency control through locking mechanism is an important aspect of any database system. Through either Pessimistic or Optimistic approach locking is possible in oracle. Serial execution of transactions always provides consistency. The pessimistic concurrency control locking mechanism maintains and confirms serialization. In case of pessimistic locking scheme the transactions waiting in a queue cannot access the resource which is locked by the other transaction until the transaction holding the lock release it by committing itself. In this scheme a transaction acquires a lock on object that will be accessed and after finishing operations on the object it unlocks the object. The pessimistic approach is error prone for lockouts and deadlocks, but ensures concurrency through serial execution of transactions.

References

[1] P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems", ACM Computing Surveys, Vol. 13(2), June 1981, pp. 186 - 221.

[2] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger "The notions of consistency and predicate locks in a database system", Communications of the ACM, 19(11):624{633, November 1976.

[3] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger "Granularity of locks and degrees of consistensy in a shared data base", In G. M. Nijssen, editor, Modeling in Data Base Management Systems, pages 365{395. North-Holland, Amsterdam, The Netherlands, 1976.

[4] J.A.Gohil, Dr.P.M.Dolia "Comparative Study and Performance Analysis of Optimistic and Pessimistic Approaches for Concurrency Control Suitable for Temporal Database Environment", National Conference on Emerging Trends in Information & Communication Technology (NCETICT), 2013

[5] Joe Hellerstein "Concurrency Control, Locking, Optimistic, Degrees of Consistency", Advanced Topics in Computer Systems ,Spring 2008 UC Berkeley.

[6] Jaypalsinh A. Gohil, Dr. Prashant M. Dolia "Study and Comparative Analysis of Basic Pessimistic and Optimistic Concurrency Control Methods for Database Management System", International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 1, January 2016

[7] Kung H. T. and Robinson J. T. "On Optimistic Methods for Concurrency Control", ACM Trans. on Database Systems, V. 6. No. 2, 1981.

[8] Richard T. Snodgrass and Ilsoo Ahn, "Temporal Databases", IEEE Computer 19(9), pp. 35–42, September, 1986

[9] Oracle DBA "Tips and Techniques Gavin Soorma Oracle 12c New Feature - Temporal Validity" http://gavinsoorma.com/2013/08/oracle-12c-new-feature-emporal-validity/.

[10] Franaszek, P. and J. Robinson "Limitation of concurrency in transaction processing", ACM Trans. Database Syst., 10: 1-28, 1985.

[11] Amer Abu Ali "On Optimistic Concurrency Control for Real-Time Database Systems", American Journal of Applied Sciences 3 (2): 1706-1710, 2006

[12] Franck Pachot "All about locks: DML, DDL, foreign key, online operations, dbi services", Switzerland

[13] J. A. Gohil, P.M.Dolia, "Testing Temporal Data Validity in Oracle 12c using Valid Time Temporal Dimension and Queries", Journal of Engineering Computers & Applied Sciences(JECAS), Volume 4, No.4, April 2015.

[14] Jaypalsinh A. Gohil, Dr. Prashant M. Dolia "Checking and Verifying Temporal Data Validity using Valid Time Temporal Dimension and Queries In Oracle 12c", International Journal of Database Management Systems (IJDMS), Vol.7, No.4, August 2015.

[15] Yongdong Wang, Lawrence A. Rowe "Cache Consistency and Concurrency Control in Client/Server DBMS Architecture".