International Journal of Computer Applications (0975 – 8887)
Volume 148 – No.7, August 2016

Graphical Representation of Optimistic 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

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

ABSTRACT

In a multiuser database environment, multiple simultaneous transactions may update the same data. Transactions executing simultaneously must produce meaningful and consistent results. In multiuser database environment conflicts are common. If conflicting situations are not dealt properly then it can harm the database. To minimize the concurrency problem the locking approach is used. Our study focus on implementation of optimistic lock through trigger on data objects of temporal database to resolve the conflicts among multiple user sessions. Through step by step graphical representation this study highlights how to acquire and release an optimistic lock on data objects in case of conflict. This experimental study shows each locking, unlocking situations along with conflicting situations graphically through Oracle 12C enterprise manager.

General Terms

Concurrency, Concurrency Control, Consistency, Locking, Temporal Database, Pessimistic Lock, Optimistic Lock.

Keywords

CC, PCC, OCC, RCN, Oracle 12c.

1. INTRODUCTION

Concurrency is conflicting situation where more than one users or transactions tires to access the same database resource at the same time. In such an environment each user must be given the equal priority to perform their operation. The situation must be avoided in which one user is updating an object in the database, while other users are waiting [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 control 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 mechanism avoids any concurrent execution of transactions as soon as potential conflicts between these transactions are detected. Alternately, Optimistic concurrency control allows such transactions to proceed at the risk of having to restart them in case this suspected conflict actually occurs. In optimistic concurrency mechanism the concentration is on the fact that the resource should not be blocked for longer period of time [5].

The main aim of concurrency control method is to preserve the consistency of database without any overhead. This can be achieved through serializabillity and serial execution of transactions. An execution is serializable if it is computationally equivalent to a serial execution. A serial execution of two or more transactions means that all operations of one transaction are executed before any operation from another transaction can execute. Since serial executions preserve consistency by definition and every serializable execution is equivalent to a serial one, every serializable execution also preserves consistency. The optimistic concurrency control method differs since; detection of conflicts and their resolution are deferred until committed. The underlying assumption here is that such conflicts are rare [6].

Optimistic concurrency control method differ from the pessimistic method in a way that here in contrast to pessimistic concurrency control approach the assumption is that very few transactions will conflict in normal operation, so there is no prerequisite sequence, synchronization and execution of transaction until transaction terminates.

A pessimistic locking technique suffers from two major problems namely frequent lockouts and deadlocks. The optimistic locking provides efficient solution to the problems. Optimistic locking does not lock records when they are read, and proceeds on the assumption that the data being updated has not changed since the read. Since no locks are taken out during the read, the deadlocks are eliminated since users should never have to wait on each other's locks. The Oracle database uses optimistic locking by default [5].

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.

2. CONCURRENCY CONTROL

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.

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 [10, 11]. 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].

3. LOCKING

3.1 Locking Mechanism

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].

3.2 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 focus is on data only, one should 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.

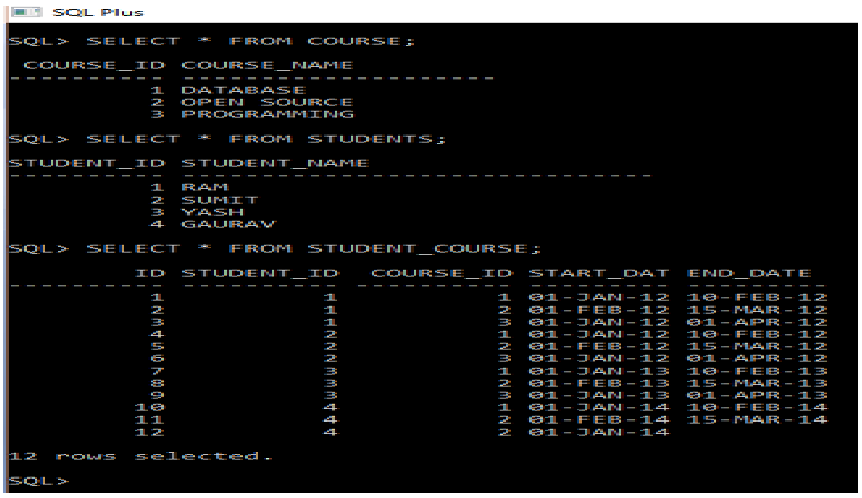
4. ACQUIRING OPTIMISTIC LOCK ON TEMPORAL RELATION

4.1 Temporal Table Creation

As visible from the below figure 1, this experimental study starts with designing 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:

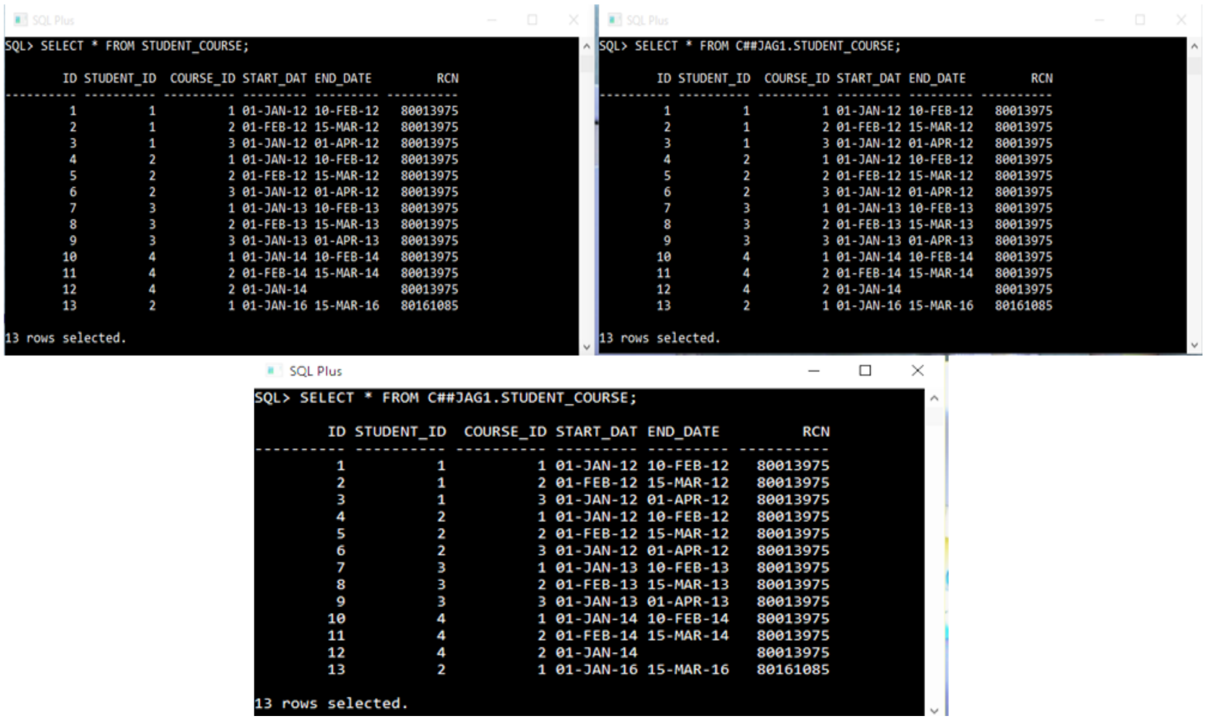


```
SQL> SELECT * FROM COURSE;

 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>
```

Fig 1: Structure of three tables along with its data

4.2 Session view at the start of experiment

At the beginning of experiment as shown in the below figure 2, the experiment starts three distinct sessions for three different users namely C##JAG1, C##JAG2 and C##JAG3. Initially as visible for the below image the owner of the STUDENT_COURSE table C##JAG1 grants ALL permissions of STUDENT_COURSE table to other two users namely C##JAG2 and C##JAG3 respectively. Initially all three uses issues a select command on STUDENT_COURSE table.



```
SQL> SELECT * FROM STUDENT_COURSE;

        ID STUDENT_ID  COURSE_ID START_DAT END_DATE         RCN
---------- ---------- ---------- --------- --------- ----------
         1          1          1 01-JAN-12 10-FEB-12   80013975
         2          1          2 01-FEB-12 15-MAR-12   80013975
         3          1          3 01-JAN-12 01-APR-12   80013975
         4          2          1 01-JAN-12 10-FEB-12   80013975
         5          2          2 01-FEB-12 15-MAR-12   80013975
         6          2          3 01-JAN-12 01-APR-12   80013975
         7          3          1 01-JAN-13 10-FEB-13   80013975
         8          3          2 01-FEB-13 15-MAR-13   80013975
         9          3          3 01-JAN-13 01-APR-13   80013975
        10          4          1 01-JAN-14 10-FEB-14   80013975
        11          4          2 01-FEB-14 15-MAR-14   80013975
        12          4          2 01-JAN-14             80013975
        13          2          1 01-JAN-16 15-MAR-16   80161085

13 rows selected.
```

```
SQL> SELECT * FROM C##JAG1.STUDENT_COURSE;

        ID STUDENT_ID  COURSE_ID START_DAT END_DATE         RCN
---------- ---------- ---------- --------- --------- ----------
         1          1          1 01-JAN-12 10-FEB-12   80013975
         2          1          2 01-FEB-12 15-MAR-12   80013975
         3          1          3 01-JAN-12 01-APR-12   80013975
         4          2          1 01-JAN-12 10-FEB-12   80013975
         5          2          2 01-FEB-12 15-MAR-12   80013975
         6          2          3 01-JAN-12 01-APR-12   80013975
         7          3          1 01-JAN-13 10-FEB-13   80013975
         8          3          2 01-FEB-13 15-MAR-13   80013975
         9          3          3 01-JAN-13 01-APR-13   80013975
        10          4          1 01-JAN-14 10-FEB-14   80013975
        11          4          2 01-FEB-14 15-MAR-14   80013975
        12          4          2 01-JAN-14             80013975
        13          2          1 01-JAN-16 15-MAR-16   80161085

13 rows selected.
```

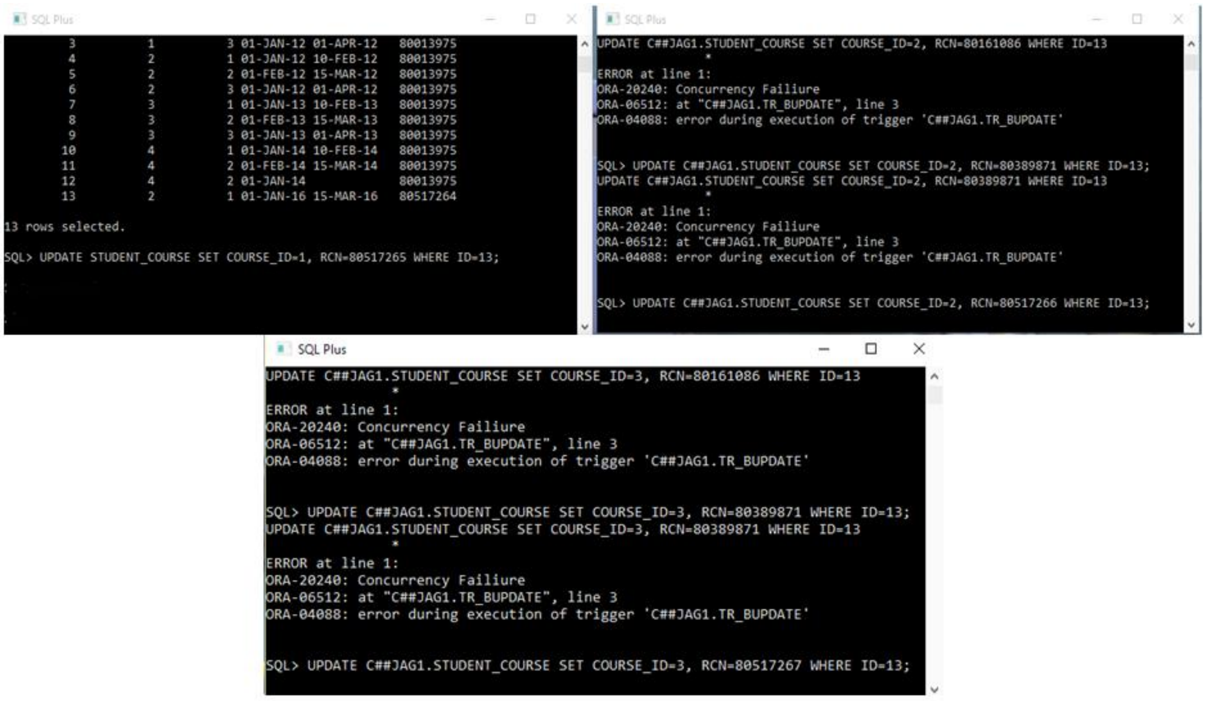
```
SQL> SELECT * FROM C##JAG1.STUDENT_COURSE;

        ID STUDENT_ID  COURSE_ID START_DAT END_DATE         RCN
---------- ---------- ---------- --------- --------- ----------
         1          1          1 01-JAN-12 10-FEB-12   80013975
         2          1          2 01-FEB-12 15-MAR-12   80013975
         3          1          3 01-JAN-12 01-APR-12   80013975
         4          2          1 01-JAN-12 10-FEB-12   80013975
         5          2          2 01-FEB-12 15-MAR-12   80013975
         6          2          3 01-JAN-12 01-APR-12   80013975
         7          3          1 01-JAN-13 10-FEB-13   80013975
         8          3          2 01-FEB-13 15-MAR-13   80013975
         9          3          3 01-JAN-13 01-APR-13   80013975
        10          4          1 01-JAN-14 10-FEB-14   80013975
        11          4          2 01-FEB-14 15-MAR-14   80013975
        12          4          2 01-JAN-14             80013975
        13          2          1 01-JAN-16 15-MAR-16   80161085

13 rows selected.
```

Fig 2: Session view at the start of experiment

4.3 Implementation of optimistic lock using a trigger

By default the optimistic locking mechanism is used by Oracle. To understand the concept of optimistic locking through experiment we have designed and implemented a trigger which allows the transactions to update the record based on Record Change Number (RCN), which is generated automatically by trigger and stored as a column value in a table. The trigger uses an integer as a concurrency key, and it is combined with the get_time function within the dbms_utility package which allows lock resolutions to 100th second.

Initially after implementing trigger the three users C#JAG1, C##JAG2 and C##JAG3 tries to update a same row of STUDENT_COURSE relation shown in the below figure 3.



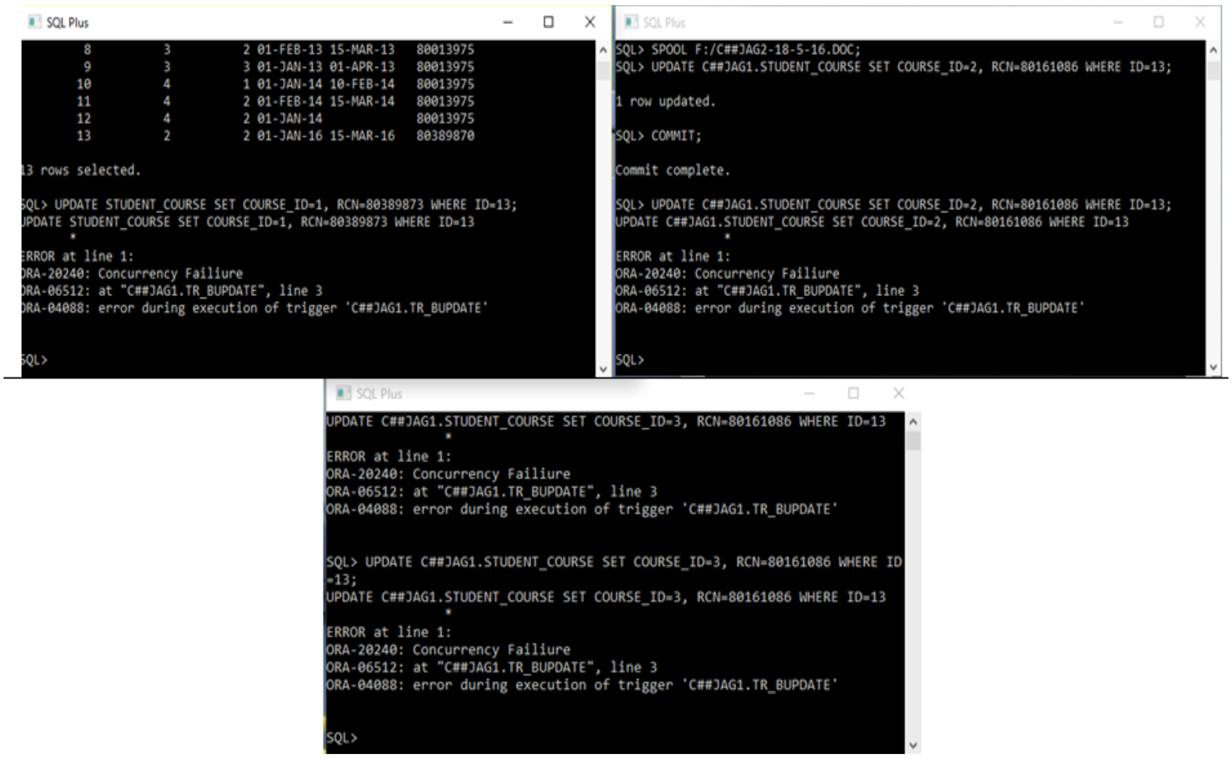
```
        3          1          3 01-JAN-12 01-APR-12   80013975
        4          2          1 01-JAN-12 10-FEB-12   80013975
        5          2          2 01-FEB-12 15-MAR-12   80013975
        6          2          3 01-JAN-12 01-APR-12   80013975
        7          3          1 01-JAN-13 10-FEB-13   80013975
        8          3          2 01-FEB-13 15-MAR-13   80013975
        9          3          3 01-JAN-13 01-APR-13   80013975
       10          4          1 01-JAN-14 10-FEB-14   80013975
       11          4          2 01-FEB-14 15-MAR-14   80013975
       12          4          2 01-JAN-14             80013975
       13          2          1 01-JAN-16 15-MAR-16   80517264

13 rows selected.

SQL> UPDATE STUDENT_COURSE SET COURSE_ID=1, RCN=80517265 WHERE ID=13;
```

```
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80389871 WHERE ID=13;
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80389871 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80517266 WHERE ID=13;
```

```
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80389871 WHERE ID=13;
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80389871 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80517267 WHERE ID=13;
```

Fig 3: Three users trying to update the same row

Following figure 4 highlights that if all the users tries to update the same row of same table with incorrect RCN number i.e. row number does not match with RCN number generated by trigger, then it shows concurrency control error message as per trigger description.



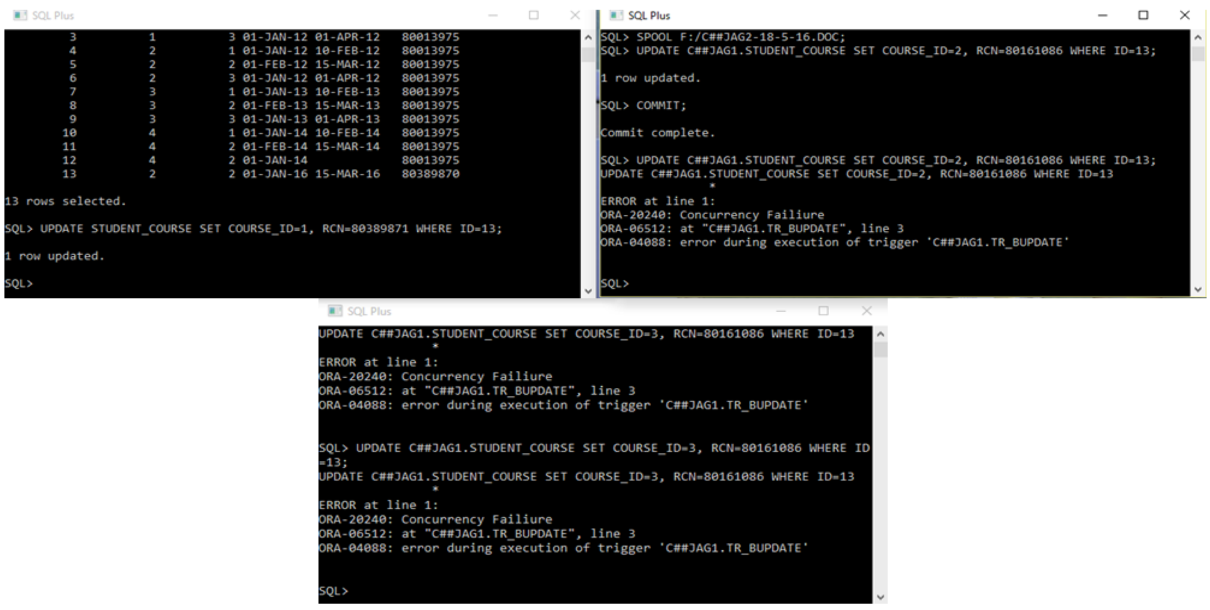
```
        8          3          2 01-FEB-13 15-MAR-13   80013975
        9          3          3 01-JAN-13 01-APR-13   80013975
       10          4          1 01-JAN-14 10-FEB-14   80013975
       11          4          2 01-FEB-14 15-MAR-14   80013975
       12          4          2 01-JAN-14             80013975
       13          2          2 01-JAN-16 15-MAR-16   80389870

13 rows selected.

SQL> UPDATE STUDENT_COURSE SET COURSE_ID=1, RCN=80389873 WHERE ID=13;
UPDATE STUDENT_COURSE SET COURSE_ID=1, RCN=80389873 WHERE ID=13
       *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL>
```

```
SQL> SPOOL F:/C##JAG2-18-5-16.DOC;
SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13;

1 row updated.

SQL> COMMIT;

Commit complete.

SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13;
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL>
```

```
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID
=13;
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL>
```

Fig 4: Users are denied update operation based on incorrect RCN number

As a next step in experiment the C##JAG1 user get the access of the table and updates the row successfully with correct RCN number without committing itself and other two users waiting in a queue i.e. C##JAG2 and C##JAG3 denies the update operation on the same table showing concurrency failure error message as per the concurrency rule described in the trigger. The situation is shown in the following figure 5.



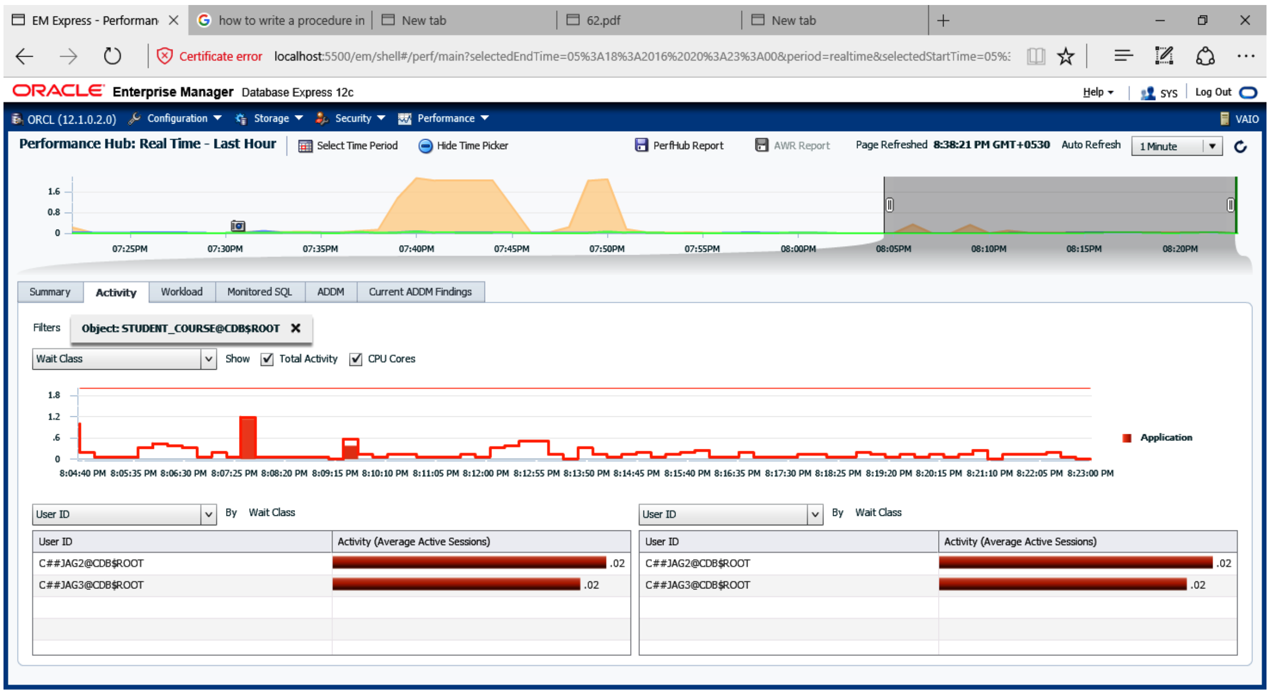
```
         3          1          3 01-JAN-12 01-APR-12   80013975
         4          2          1 01-JAN-12 10-FEB-12   80013975
         5          2          2 01-FEB-12 15-MAR-12   80013975
         6          2          3 01-JAN-12 01-APR-12   80013975
         7          3          1 01-JAN-13 10-FEB-13   80013975
         8          3          2 01-FEB-13 15-MAR-13   80013975
         9          3          3 01-JAN-13 01-APR-13   80013975
        10          4          1 01-JAN-14 10-FEB-14   80013975
        11          4          2 01-FEB-14 15-MAR-14   80013975
        12          4          2 01-JAN-14             80013975
        13          2          2 01-JAN-16 15-MAR-16   80389870

13 rows selected.

SQL> UPDATE STUDENT_COURSE SET COURSE_ID=1, RCN=80389871 WHERE ID=13;

1 row updated.

SQL>
```

```
SQL> SPOOL F:/C##JAG2-18-5-16.DOC;
SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13;

1 row updated.

SQL> COMMIT;

Commit complete.

SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13;
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL>
```

```
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL> UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID
=13;
UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID=13
                *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL>
```

Fig 5: View of three sessions after update operation of C##JAG1 with correct RCN number

The following image shows the view of oracle enterprise manager of oracle 12c at the start of three distinct user sessions. It is visible from the image that the C##JAG2 and C##JAG3 are waiting for C##JAG1 to release the lock on STUDENT_COURSE object, but not in deadlock state as it was in pessimistic locking experiment [5]. They can continue their normal operation which is not related to locked resource.



User ID	Activity (Average Active Sessions)
C##JAG2@CDB$ROOT	.02
C##JAG3@CDB$ROOT	.02

User ID	Activity (Average Active Sessions)
C##JAG2@CDB$ROOT	.02
C##JAG3@CDB$ROOT	.02

Fig 6: Two user sessions in waiting in a queue

4.4 Concurrency Situation

Concurrency control can be provided through optimistic locking approach using a trigger. When more than one user session tries to access the same resource at the same time in our case STUDENT_COURSE table, the trigger allows one user session to continue with correct RCN number and other user sessions are just in waiting state but do not lock the resource. So the problem of deadlock is resolved through optimistic approach which is there in pessimistic locking experiment [5]. So concurrent executions of update transactions on the locked recourse are now permitted in optimistic locking.

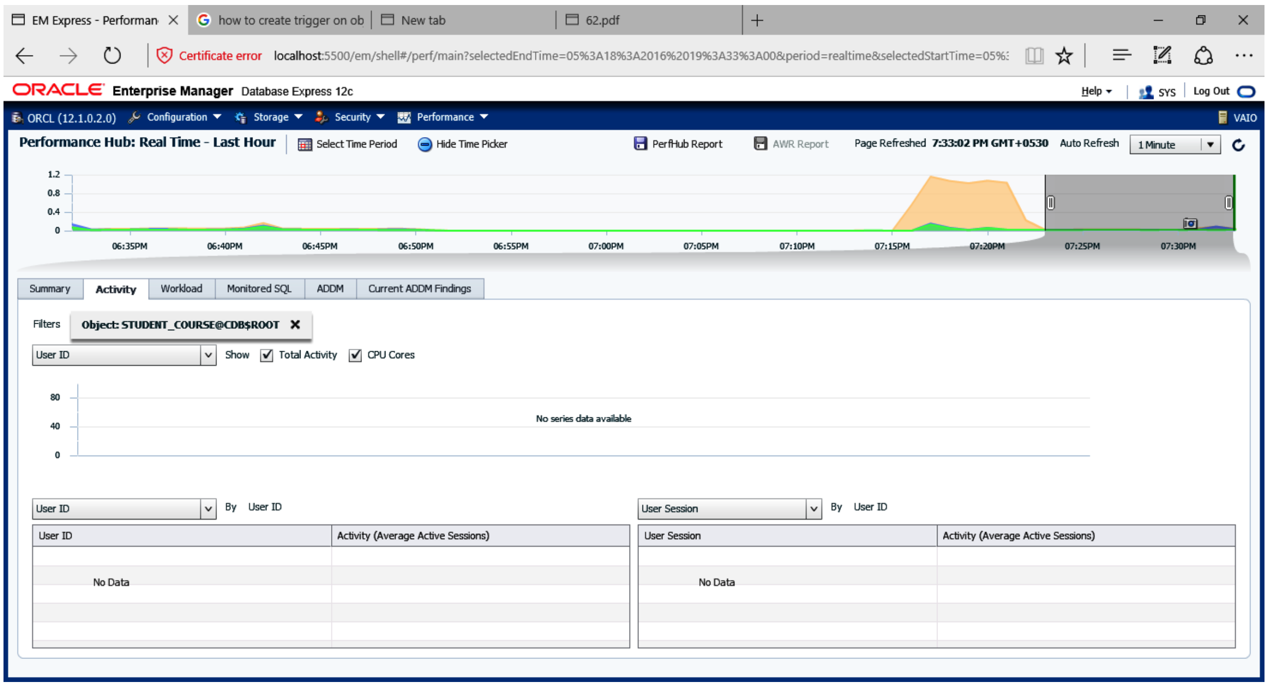
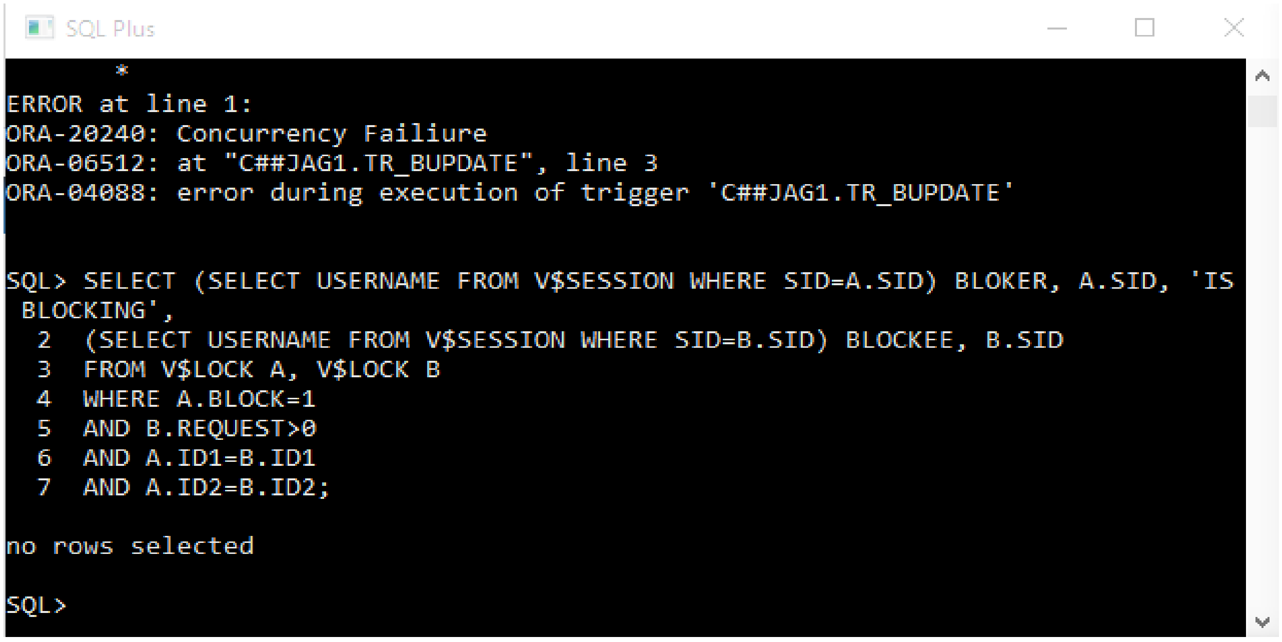


Fig 7: Enterprise Manager showing concurrency and resource

The above image shows the concurrency environment for proposed experiment. It shows the resource is not locked for other users and other user sessions can continue their normal operation. There will be no concurrency situation.

As per the mechanism of our study the trigger imposes a pessimistic lock on the object. So optimistic lock is exists on the object irrespective of the session or transaction which holds the locks commits itself. Hence in our experiment the optimistic lock is held by C##JAG1 user and other user's i.e. C##JAG2 and C##JAG3 are waiting for their turn but not in deadlock mode. So they can perform their normal operations in normal manner until lock is acquired by them on locked recourse. So one can say that C##JAG1 user's session is not blocking C##JAG2 and C#JAG3 users respectively as is the case in pessimistic locking experiment [5]. The following figure shows the details about who is blocking whom.



```
         *
ERROR at line 1:
ORA-20240: Concurrency Failiure
ORA-06512: at "C##JAG1.TR_BUPDATE", line 3
ORA-04088: error during execution of trigger 'C##JAG1.TR_BUPDATE'


SQL> SELECT (SELECT USERNAME FROM V$SESSION WHERE SID=A.SID) BLOKER, A.SID, 'IS
 BLOCKING',
  2  (SELECT USERNAME FROM V$SESSION WHERE SID=B.SID) BLOCKEE, B.SID
  3  FROM V$LOCK A, V$LOCK B
  4  WHERE A.BLOCK=1
  5  AND B.REQUEST>0
  6  AND A.ID1=B.ID1
  7  AND A.ID2=B.ID2;

no rows selected

SQL>
```

Fig 8: Who is blocking whom

5. RELEASING OPTIMISTIC LOCK ON TEMPORAL RELATION

The COMMIT command is used to release both pessimistic and optimistic locks on an object. Through this experiment it is quite evident that lock on the object acquired by one user session does not prevent other user sessions to continue their normal operation. In experiment the user C##JAG1 gets the lock first in sequence but due to concurrency control provisions implemented in trigger it allows other users C##JAG2 and C##JAG3 to continue without waiting for a long time for STUDENT_COURSE relation. So whenever lock is released through COMMIT command the next user sessions waiting in a queue gets the access of STUDENT_COURSE relation as per correct RCN number according to the rules of a trigger.

6. SESSION OUTLINE VIEW

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

Table 1. Session outline view

Time	Session 1 for user C##JAG1	Session 2 for user C##JAG2	Session 3 for user C##JAG3	Explanation
T1	GRANT ALL ON STUDENT_COURSE TO C#JAG2,C#JAG3;			The C##JAG1 user grants all the permission on STUDENT_COURSE relation to two distinct users' C##JAG2 and C##JAG3.
T2	SELECT * FROM STUDENT_COURSE;	SELECT * FROM C##JAG1.STUDENT_COURSE;	SELECT * FROM C##JAG1.STUDENT_COURSE;	The three user session C##JAG1, C##JAG2 and C##JAG3 issues a select statement on STUDENT_COURSE relation owned by C##JAG1 user.
T3	UPDATE STUDENT_COURSE SET COURSE_ID=1, RCN=80389873 WHERE ID=13;			In session 1 the user C##JAG1 tries to update a row with wrong RCN number and trigger denies the update operation showing concurrency failure message generated by a trigger.
T4		UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13;		In session 2 the user C##JAG2 tries to update a row with wrong RCN number and trigger denies the update operation showing concurrency failure message generated by a trigger.
T5			UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=3, RCN=80161086 WHERE ID=13;	In session 3 the user C##JAG3 tries to update a row with wrong RCN number and trigger denies the update operation showing concurrency failure message generated by a trigger.
T6	UPDATE STUDENT_COURSE SET COURSE_ID=1, RCN=80389873 WHERE ID=13;	UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161086 WHERE ID=13;	UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2, RCN=80161087 WHERE ID=13;	In the next step three different sessions of users' C##JAG1, C##JAG2 and C##JAG3 respectively again tries to update the same row of STUDENT_COURSE relation but this time with correct RCN number sequence.
T7	1 row updated	No wait…	No wait….	At this point after the successful update operation of user C##JAG1 the other two transactions of user C##JAG2 and C##JAG3 are not in waiting state, they can perform other normal operations which are not related to common row of STUDENT_COURSE relation.
T8	Commit;	1 row updated	No wait….	At this point C##JAG1 user issues a commit command which allows the C##JAG2 to successfully update the row as per correct RCN number. Still at this point C##JAG3 can perform its normal operations.
T9		Commit;	1 row updated	At this moment the C##JAG2 user issues a commit statement so now lock which is being held by user C##JAG2 is released and granted to next user C##JAG3 which is waiting in queue with correct RCN number. At this point now lock is with user C##JAG3.
T10			Commit;	Finally the user C##JAG3 commits its operation and releases the lock.

7. SESSION WAITING TIME STATISTICS

The optimistic locking approach using a trigger allows other users to continue their other non conflicting operations in normal manner. So waiting time for the conflicting transactions is in very negligible fractions, which is vast improvement over pessimistic locking experiment [5]. As visible from the chart the session waiting time for sessions 2 and sessions 3 of users C##JAG2 and C##JAG3 is around 0.02 sec. So from the below chart it is quite clear and evident that in optimistic locking approach the sessions waiting time is very less which is almost negligible, which is not the case with pessimistic locking approach [5].

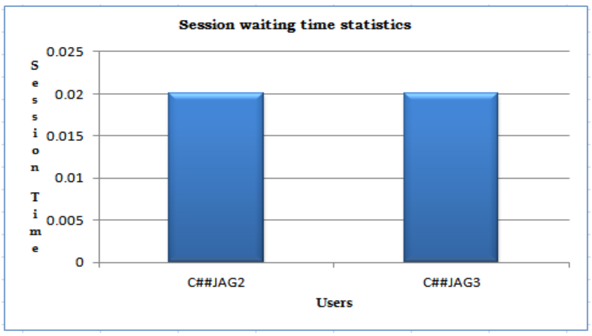


Fig 9: Session waiting time statistics

8. SUMMARY

In optimistic concurrency control method detection of conflicts and their resolution are deferred until committed. The underlying assumption here is that such conflicts are rare. The main disadvantage of pessimistic approach is deadlocks. They may be a situation in which deadlock can arise in a system through serial execution of transactions. An optimistic concurrency control approach let the transaction to execute itself without worry of conflict with other transactions. As the name implies the optimistic concurrency control mechanism is based on the assumption that conflicts between transactions are not frequent and regular. Finally this experimental study implements the optimistic locking through trigger which allows other conflicting transactions to do their normal operation without waiting for indefinite time. So there is a vast decrease in average session waiting time, which is almost at negligible level.

9. CONCLUSION

Locking is an efficient mechanism to provide concurrency control in database system environment. Locking approaches can be classified in either pessimistic or optimistic categories. Optimistic concurrency control requires transactions to operate in a private workspace, so their modifications are not visible to other until they commit. When a transaction is ready to commit, a validation is performed on all the data items to see whether the data conflicts with operations of other transactions. If the validation fails, then the transaction will have to be aborted and restarted later. Optimistic control is clearly overcomes the problem of deadlock. Optimistic approach is deadlock free and avoids any time consuming node-locked scenarios. This approach is generic in the sense if the transactions become query dominant; the concurrency control overhead becomes almost negligible. In this approach reading operations are completely unrestricted whereas write operations of transactions are severely restricted. The optimistic concurrency control can be implemented to real time database system where time dependant distributed transactions are frequent. The inherent nature of distributed transactions for temporal database systems can be blend together with optimistic concurrency control mechanism which is more reliable and suitable for temporal database environment as compared to optimistic approach where session waiting time is more and deadlocks are common.

10. 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] Jaypalsinh A. Gohil, Dr. Prashant M. Dolia, “Graphical Representation of Pessimistic Locking and Concurrency Control for Temporal Database using Oracle 12c Enterprise Manager”, International Journal of Innovations in Engineering and Technology (IJIET), Volume 6, Issue 4, April 2016

[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”.