# 

# Lab 3 – Transaction Management:

# A solution to Data Concurrency and Resilience to System Failure

## Objectives

After completing this lab, you will be able to:

* understand data concurrency and consistency
* understand preventable phenomena and isolation levels
* understand how Oracle manages data concurrency and consistency
* how to set isolation level

## 

## Tools/Software Requirement

To accomplish this lab, students should have Oracle Database 18c express edition or later version installed on their machines.

We will be using SQLPlus, which comes along with oracle.

## Introduction to Data Concurrency and Consistency in a Multiuser Environment

Data concurrency means that many users can access data at the same time.

Data consistency means that each user sees a consistent view of the data, including visible changes made by the user's own transactions and transactions of other users.

## Preventable Phenomena and Transaction Isolation Levels

Dirty reads: A transaction reads data that has been written by another transaction that has not been committed yet.

Nonrepeatable (fuzzy) reads: A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data.

Phantom reads (or phantoms): A transaction re-runs a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition.

## Preventable Read Phenomena by Isolation Level

A screenshot of a computer

Description automatically generated

## How Oracle Manages Data Concurrency and Consistency

* Multiversion Concurrency Control
* Statement-Level Read Consistency
* Transaction-Level Read Consistency
* Read Consistency with Real Application Clusters
* Oracle Isolation Levels
* Comparison of Read Committed and Serializable Isolation
* Choice of Isolation Level

## Oracle Isolation Levels

Graphical user interface, text, application, email

Description automatically generated

## What is the difference between transaction and query?

A query is a single SQL statement that does Select, Update, Insert or Delete of rows while a transaction is a consecutive sequence of SQL statements (from the application viewpoint) that have the "ACID" properties:

* Atomicity: All statements or none are executed.
* Consistency: Data integrity is always maintained.
* Isolation: Transaction A can never affect Transaction B.
* Durability: Changes that are committed by a transaction persist, even in event of system failure.

## Setting the Isolation Level

|  |
| --- |
| SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  SQL> SET TRANSACTION READ ONLY; |

Changing the isolation level

|  |
| --- |
| SQL> ALTER SESSION SET ISOLATION\_LEVEL= READ COMMITTED;  SQL> ALTER SESSION SET ISOLATION\_LEVEL= SERIALIZABLE; |

## Lab Tasks 1:

In this task you will see the behavior of transactions under different isolation level using SQL Plus.

Please open SQL plus and provide same username and password which you used for SQL developer;

Here is the screenshot of SQL Plus command prompt:

Text

Description automatically generated

Please open three sessions of SQL PLUS and try out following operation in the same order.

|  |  |  |
| --- | --- | --- |
| **Session 1** | **Session 2** | **Session 3** |
| **SQL>** create table R (A number(2), B number(2));  insert into R values (5,10);  insert into R values (6,8); | **set transaction isolation level serializable;**  **SQL>** select \* from R;  --**no rows selected** | **SQL> set transaction isolation level read committed;**  Transaction set.  select \* from R;  --no rows selected |
| **Commit;** |  |  |
|  | select \* from R;  --**no rows selected** | **select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8** |
| **set transaction isolation level serializable;**  **SQL> insert into R values(20,30);**  **1 row created.**  **SQL> select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8**  **20 30** | select \* from R;  --no rows selected  Note session 2 and 3 will not reflect changes until we commit changes in session 1. | **SQL> select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8** |
| **Commit;** |  |  |
|  | select \* from R;  no rows selected.  If transaction submitted under serializability it will not see any change in the middle of transaction. | **SQL> select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8**  **20 30**  **If transaction submitted under read committed it will see any change made by other transaction, in this case changes made by session 1.** |
|  | insert into R values (12,15);  select \* from R;  A B  ---------- ----------  12 15 |  |
| **SQL> select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8**  **20 30**  **No changes reflected in session 1 and 3 made by session 2, until we commit session 2.** |  | **select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8**  **20 30** |
|  | **commit;** |  |
| **select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8**  **20 30**  **12 15**  **After committing session 2, the database state is consistent across all session** | select \* from R;  A B  ---------- ----------  5 10  6 8  20 30  12 15 | **select \* from R;**  **A B**  **---------- ----------**  **5 10**  **6 8**  **20 30**  **12 15** |

## **Lab Tasks 2: In the second task, please create copy of employee table and perform following operation by using 2 sessions.**

**2(a): Read committed**

| **Session 1** | **Session 2** |
| --- | --- |
| Create table employee\_copy  AS (select \* from employee); |  |
| SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  FNAME SALARY  --------------- ----------  James 400  Jennifer 43000  Jared 85000 |  |
| SQL> SQL> UPDATE employee\_copy set salary=10000 where fname='James';  SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  FNAME SALARY  --------------- ----------  James 10000  Jennifer 43000  Jared 85000 |  |
|  | SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED; |
|  | SQL>SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  FNAME SALARY  --------------- ----------  James 400  Jennifer 43000  Jared 85000 |
|  | SQL> UPDATE employee\_copy set salary=91000 where fname='Jennifer'; |
| SQL> INSERT INTO employee\_copy VALUES  ('Martin','B','Smith','123456710','09-Jan-55','731 Fondren, Houston, TX','M',30000,'333445555',5); |  |
|  | SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  FNAME SALARY  --------------- ----------  James 400  Jennifer 91000  Jared 85000 |
|  | SQL> update employee\_copy set salary =63000 where fname='James';  -- prompt does not return –read is ok, but new insert record by session one locked the table |
| SQL> COMMIT; |  |
|  | 1 row updated.  SQL> |
|  | SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  FNAME SALARY  --------------- ----------  James 63000  Jennifer 91000  Jared 85000 |
|  | SQL> COMMIT; |
| SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  **FNAME SALARY**  **--------------- ----------**  **James 63000**  **Jennifer 91000**  **Jared 85000** |  |

### **2(b) – Serializable**

|  |  |
| --- | --- |
| **Session 1** | **Session 2** |
| SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  **FNAME SALARY**  **--------------- ----------**  **James 63000**  **Jennifer 91000**  **Jared 85000** |  |
| SQL> update employee\_copy set salary =7000 where fname='James'; |  |
|  | SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; |
|  | SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  **FNAME SALARY**  **--------------- ----------**  **James 63000**  **Jennifer 91000**  **Jared 85000** |
|  | SQL> update employee\_copy set salary =9900 where fname='Jennifer'; |
| SQL> INSERT INTO employee\_copy VALUES  ('Eric','B','Smith','123456710','09-Jan-55','731 Fondren, Houston, TX','M',30000,'333445555',5); |  |
| SQL> COMMIT; |  |
| SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  FNAME SALARY  --------------- ----------  James 7000  Jennifer 91000  Jared 85000  Note salary inconsistency in both sessions | SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  **FNAME SALARY**  **--------------- ----------**  **James 63000**  **Jennifer 9900**  **Jared 85000** |
|  | SQL> COMMIT; |
| SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  **FNAME SALARY**  **--------------- ----------**  **James 7000**  **Jennifer 9900**  **Jared 85000** | SQL> SELECT fname, salary  from employee\_copy  where fname in ('James', 'Jennifer', 'Jared');  **FNAME SALARY**  **--------------- ----------**  **James 7000**  **Jennifer 9900**  **Jared 85000** |