**National University of Computer & Emerging Sciences, Karachi Computer Science Department**

**Spring 2023, Lab Manual - 11**

| **Course Code: CL-2005** | **Course : Database Systems Lab** |
| --- | --- |
| **Instructor(s) :** | **Ali Fatmi** |

Contents:

1. Transactions(Commit,Rollback,savepoint,Autocommit).
2. Activity on Different Transactions
3. Tasks

DATABASE TRANSACTION:

A **TRANSACTION** consists of a collection of DML statements that form a logical unit of work.

A Database Transaction consists of one of the following statements.

1. DML statements that make up one consistent change to the data.
2. One DDL statement.
3. One DCL statement.

Transactions consist of DML statements that make up one consistent change to the data.

Example:

A transfer of funds between two accounts should include the debit to one account and credit to another account in the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

You can use the COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION

command to control the transaction.

[**COMMIT**](https://way2tutorial.com/plsql/plsql-transaction.php#commit)**:** COMMIT command to make changes permanent save to a database during the current transaction.

[**ROLLBACK**](https://way2tutorial.com/plsql/plsql-transaction.php#rollback): ROLLBACK command executes at the end of current transaction and undo/undone any changes made since the begin transaction.

[**SAVEPOINT**](https://way2tutorial.com/plsql/plsql-transaction.php#savepoint)**:** SAVEPOINT command saves the current point with the unique name in the processing of a transaction.

[**AUTOCOMMIT**](https://way2tutorial.com/plsql/plsql-transaction.php#autocommit)**:** Set AUTOCOMMIT ON to execute COMMIT Statement automatically.

[**SET TRANSACTION**](https://way2tutorial.com/plsql/plsql-transaction.php#set_transaction)**:** PL/SQL SET TRANSACTION command set the transaction properties such as read-write/read only access.

**Isolation Level:**

Oracle supports different isolation levels, including READ COMMITTED,

SERIALIZABLE, READ UNCOMMITTED, and Repeatable read.

**SET TRANSACTION ISOLATION LEVEL READ COMMITTED;**

**Read Write / Read Only:**

Specifies whether the transaction can perform write operations or is limited to read operations only.

**SET TRANSACTION READ WRITE;**

**Name:**

Assigns a name to the transaction, which can be useful for distributed transactions.

**SET TRANSACTION NAME 'MyTransaction';**

**Deferrable / Not Deferrable:**

Specifies whether constraints should be checked immediately or at the end of the transaction.

**SET TRANSACTION DEFERRABLE;**

Database Transaction begins when the first DML SQL statement is executed. Ends with one of the following events

* COMMIT or ROLLBACK is issued.
* DDL or DCL statement executes (Automatic commit).
* Users normally exit.

## Advantages of Commit and Rollback Statements.

-Ensure Data Consistency.

-Preview data changes before making any permanent changes.

-Group logically related operations.

## Implicit Transaction:

Automatic commit occurs under the following conditions

-DDL Statement is issued.

-DCL statement is issued.

-Normal Exit with command Exit from SQL \* PLUS.

Automatic Rollback occurs under the following conditions.

-Abnormal Exit or termination of SQL \* PLUS.

-System Failure.



Flow of Transaction Control

| T# | Transaction | Explanation |
| --- | --- | --- |
| t1 | SET TRANSACTION  NAME 'sal\_update'; | This statement begins a transaction and names it sal\_update. |
| t2 | UPDATE employees SET salary = 7000 WHERE last\_name  ='Banda'; | This statement updates the salary for Banda to 7000. |
| t3 | SAVEPOINT  after\_banda\_sal; | This statement creates a savepoint  named after\_banda\_sal, enabling changes in this transaction to be rolled back to this point. |
| t4 | UPDATE employees SET salary = 12000 WHERE last\_name =  'Greene'; | This statement updates the salary for Greene to 12000. |
| t5 | SAVEPOINT  after\_greene\_sal; | This statement creates a savepoint  named after\_greene\_sal, enabling changes in this transaction to be rolled back to this point. |
| t6 | ROLLBACK TO SAVEPOINT  after\_banda\_sal; | This statement rolls back the transaction to t3, undoing the update to Greene's salary at t4.  The sal\_update transaction has *not* ended. |
| t7 | UPDATE employees SET salary = 11000 WHERE last\_name =  'Greene'; | This statement updates the salary for Greene to 11000 in transaction sal\_update. |
|  |  |  |



| t8 | ROLLBACK; | This statement rolls back all changes in transaction sal\_update, ending the transaction. |
| --- | --- | --- |
| t9 | SET TRANSACTION  NAME 'sal\_update2'; | This statement begins a new transaction in the session and names it sal\_update2. |
| t10 | UPDATE employees SET salary = 7050 WHERE last\_name =  'Banda'; | This statement updates the salary for Banda to 7050. |
| t11 | UPDATE employees SET salary = 10950 WHERE last\_name =  'Greene'; | This statement updates the salary for Greene to 10950. |
| t12 | COMMIT; | This statement commits all changes made in transaction sal\_update2, ending the transaction. The commit guarantees that the changes are saved in the online redo log files. |

## Automatic Transaction Control

To execute a COMMIT automatically whenever an INSERT, UPDATE or DELETE command is executed, you can set the AUTOCOMMIT environment variable as −

You can turn-off the auto commit mode using the following command −



## Executing Queries on Different Transactions

Activity:

Create a table worker by having the following attributes.



**Note** that since the creation of a table is a [**DDL**](https://docs.oracle.com/cd/B14117_01/server.101/b10759/statements_1001.htm#i2099120) statement, there is no need for commit because Oracle SQL auto-commits it.

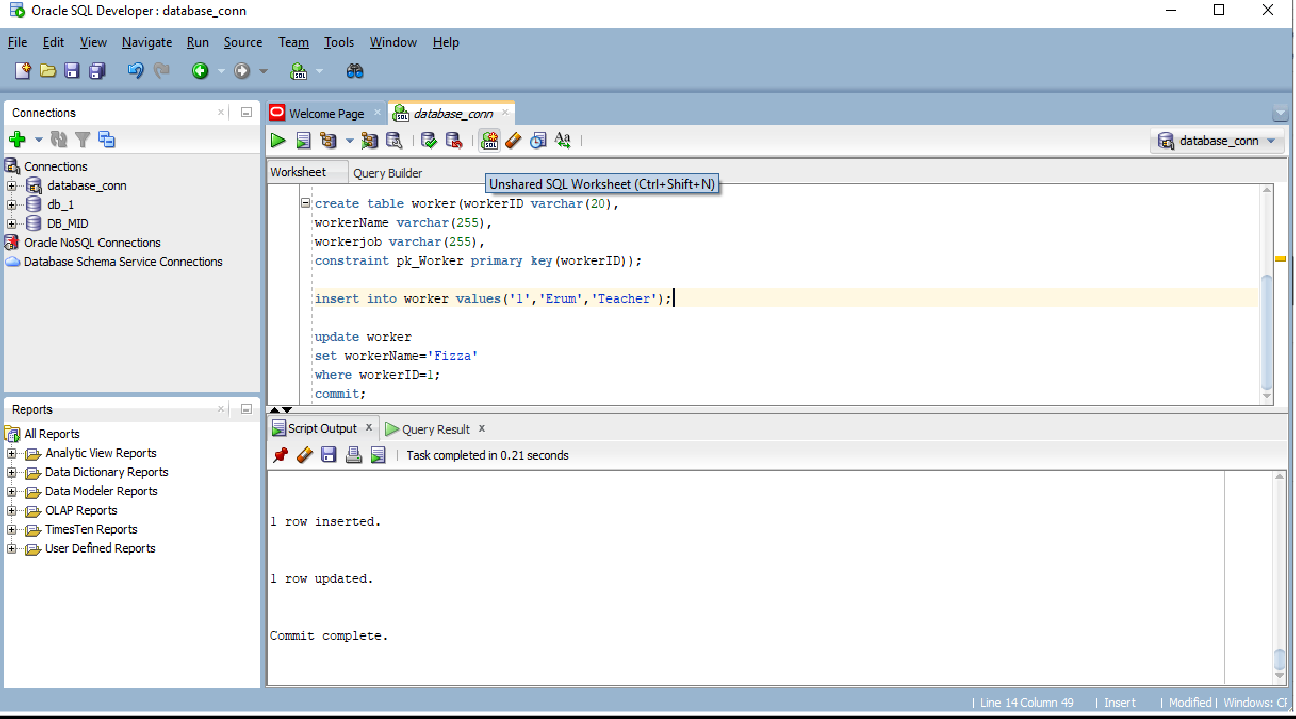
Insert a row in the worker table.



After that, we should now have the new record on our database. In order to leave the record locked, we will now do an update on it, without committing the transaction.



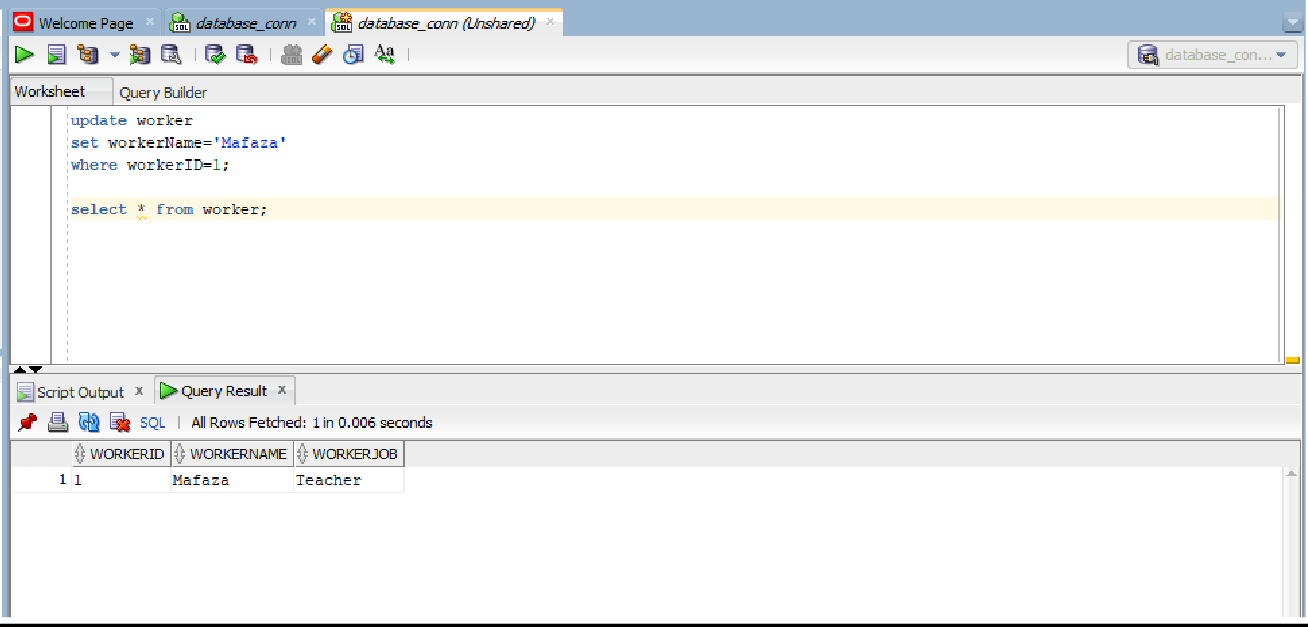
Now, we will open a new Unshared SQL Worksheet, as indicated in figure 1. The SQL statements executed in this sheet will be in a different transaction.



Now, to test if this is working correctly, execute another update statement on the same record, on the new unshared worksheet.



Since we had a transaction that updated that same record and haven’t yet committed, the transaction on the new unshared worksheet will not update the record so we need to go back to the previous sheet and commit the transaction.



**Tasks:**

# **Taks:01** Create table persons having personID as primary key, FirstName,LastName,Address,City and age as attributes.

Add 10 records into persons and create a savepoint after inserting five records.

# Update the record on personID=7 where FirstName=Erum and After the update it should be FirstName=Rida.

Now Rollback the transaction to savepoint1 and see the changes.

# **Task:02** By using above persons table update a record on personID=8 and change the age from 28 to 30 then without commiting change create a new session and update the record on PersonID=8 and set age=31. State what is the reason behind no update.

# **Task:03** Create another table named cars and create a 1-to-many from person to cars. Now insert some valid values into cars as well. Add a savepoint after every insertion and don’t commit any of the changes yet. Using the sample code below, create a PL/SQL code that checks if the number of cars owned by a user exceeds 3, if so then rollback to the previous statement else keep moving forward. Once you have made all of the changes, commit the transaction.

set autocommit off;

SET SERVEROUTPUT ON;

DECLARE

sal integer;

BEGIN

UPDATE testing SET price = 5500 WHERE u\_id = 1;

select price into sal from testing;

dbms\_output.put\_line('Salary : ' || sal);

SAVEPOINT after\_price\_up;

UPDATE testing SET price = 2500 WHERE u\_id = 1;

select price into sal from testing;

dbms\_output.put\_line('Salary : ' || sal);

ROLLBACK TO SAVEPOINT after\_price\_up;

rollback;

END;