National University of Computer & Emerging Sciences, Karachi
FAST School of Computing
Fall 2024 Lab Manual - 11

Course Code: CL-2005	Database Systems Lab
Instructor(s):	**Sohail Ahmed Malik, Mr. Mubashir, Mr. Sameer Faisal, Ms. Mehak Mazhar, Ms. Fatima, Ms. Mahnoor Javed, Ms. Filza Akhlaq, Ms. Bushra Sattar, Ms. Syeda Ravia Ejaz, Ms. Yumna Asif**

Contents:

1. Introduction to Database Transactions:

Definition of a Transaction:

A **transaction** in a database is a sequence of one or more SQL statements that perform a specific unit of work, ensuring data integrity. A transaction allows multiple actions to be grouped so that they succeed or fail together, creating a consistent change in the data.

Components of a Database Transaction:

A database transaction can consist of:

- **DML (Data Manipulation Language) statements** that represent a consistent change in the database.
- A single **DDL (Data Definition Language)** statement, which implicitly commits.
- A single **DCL (Data Control Language)** statement, which also implies a commit.

Example:

Consider a fund transfer between two bank accounts, where debiting from one account and crediting to another must be successful simultaneously. If either action fails, the transaction should fail entirely, maintaining data integrity.

2. Transaction Control Commands:

1. **COMMIT** - Finalizes the transaction, making changes permanent in the database.
2. **ROLLBACK** - Reverts all changes made since the start of the transaction or since the last SAVEPOINT.
3. **SAVEPOINT** - Sets a marker within the transaction, allowing partial rollbacks to specific points.
4. **SET TRANSACTION** - Configures transaction properties such as read-write mode.
5. **AUTOCOMMIT** - Automatically commits each individual DML statement.

Flow of Transaction Control:

T#	Transaction	Explanation
t1	SET TRANSACTION NAME 'sal_update';	Begins a transaction and names it sal_update.
t2	UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';	Updates salary for Banda to 7000.
t3	SAVEPOINT after_banda_sal;	Sets a savepoint named after_banda_sal.

t4	UPDATE employees SET salary = 12000 WHERE last_name = 'Greene';	Updates salary for Greene to 12000.
t5	SAVEPOINT after_greene_sal;	Sets a savepoint named after_greene_sal.
t6	ROLLBACK TO SAVEPOINT after_banda_sal;	Rolls back to after_banda_sal, undoing Greene's update.
t7	UPDATE employees SET salary = 11000 WHERE last_name = 'Greene';	Updates Greene's salary to 11000.
t8	ROLLBACK;	Rolls back all changes, ending sal_update.
t9	SET TRANSACTION NAME 'sal_update2';	Begins a new transaction named sal_update2.
t10	UPDATE employees SET salary = 7050 WHERE last_name = 'Banda';	Updates Banda's salary to 7050.
t11	UPDATE employees SET salary = 10950 WHERE last_name = 'Greene';	Updates Greene's salary to 10950.
t12	COMMIT;	Commits all changes, finalizing sal_update2.

3. Implicit and Explicit Transaction Control:

Implicit Transaction Control:

Oracle Database performs automatic commits and rollbacks in specific situations:

- **Automatic Commit:** Occurs when a DDL or DCL statement is issued or upon a normal exit from SQL*Plus.
- **Automatic Rollback:** Happens on an abnormal exit or system failure.

Explicit Transaction Control:

Users can manually control transactions using **COMMIT**, **ROLLBACK**, and **SAVEPOINT** commands to ensure precise control over data modifications.

Advantages of COMMIT and ROLLBACK:

- Ensures data consistency.
- Allows previewing changes before committing.
- Groups related operations for atomic execution.

Creating and Modifying a Table in a Transaction

1. **Create a Table** worker with attributes worker_id, worker_name, and salary.

```
CREATE TABLE worker (
    worker_id NUMBER PRIMARY KEY,
    worker_name VARCHAR2(50),
    salary NUMBER
);
```

2. **Insert a Record** in worker:

```
INSERT INTO worker (worker_id, worker_name, salary) VALUES (1, Sohail, 5000);
```

This action is a DML statement, so a commit is required to finalize it.

3. **Update the Salary** without committing:

```
UPDATE worker SET salary = 6000 WHERE worker_id = 1;
```

4. **Activity on Different Transactions:**
   Open a **new unshared worksheet**.
   Attempt to **update** the same record from the unshared worksheet:

```
UPDATE worker SET salary = 7000 WHERE worker_id = 1;
```

   The transaction will lock the row until the previous worksheet's transaction is committed or rolled back.
5. **Commit the transaction** in the original worksheet:

```
COMMIT;
```

   Retry the update from the unshared worksheet. This time, it should succeed.

Savepoint and Rollback within a Transaction

1. **Start Transaction** and Insert Data:

```
SET TRANSACTION NAME 'test_transaction';
INSERT INTO worker (worker_id, worker_name, salary) VALUES (2, 'Erum', 5500);
SAVEPOINT sp1;
```

2. **Update Data** and set a savepoint:

```
UPDATE worker SET salary = 6000 WHERE worker_name = 'Erum';
SAVEPOINT sp2;
```

3. **Rollback to sp1** (Undo the Update):

```
ROLLBACK TO SAVEPOINT sp1;
```

   Erum's salary returns to 5500.

4. **Commit the Transaction**:

```
COMMIT;
```

Using AUTOCOMMIT

1. **Enable AUTOCOMMIT**:

```
SET AUTOCOMMIT ON;
```

2. **Insert Data**:

```
INSERT INTO worker (worker_id, worker_name, salary) VALUES (3, 'FAST-NU', 5000);
```

   Since AUTOCOMMIT is on, the insert will commit automatically.

3. **Disable AUTOCOMMIT**:

```
SET AUTOCOMMIT OFF;
```

Scenario: Customer and Order Transactions

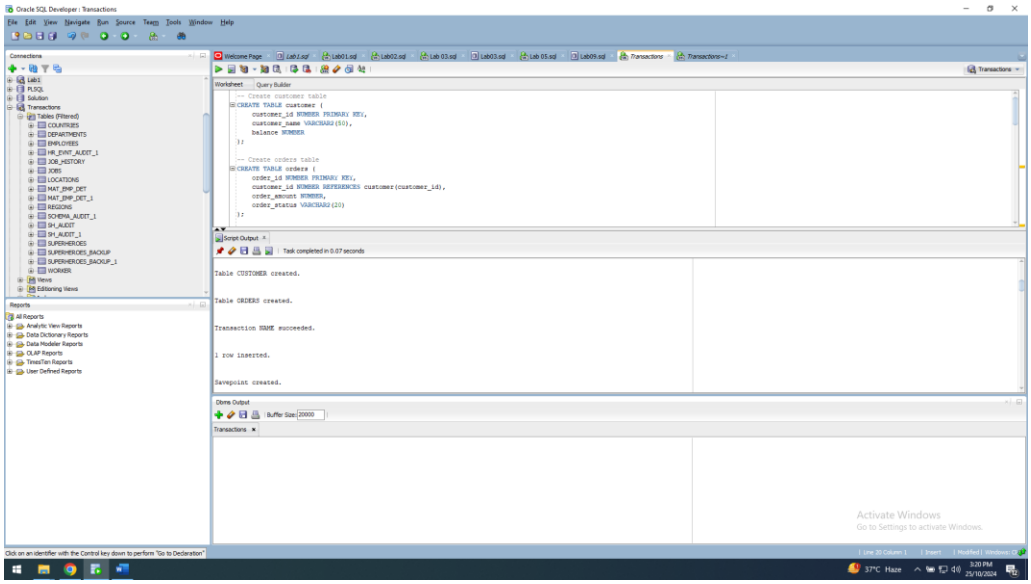
Problem Statement:

We want to handle a customer's order placement, ensuring that both customer details and order records are updated only if all actions in the transaction succeed. If any action fails, all changes should be undone to maintain data integrity.

Tables Involved:

1. **customer**: Stores customer information.
2. **orders**: Stores order information related to each customer.

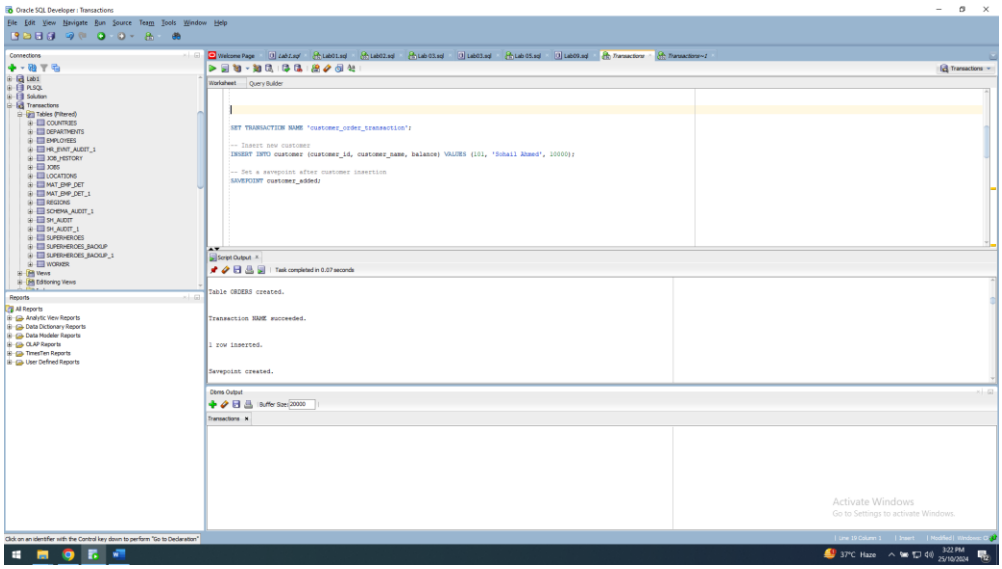
Table Structure:



Transaction Steps

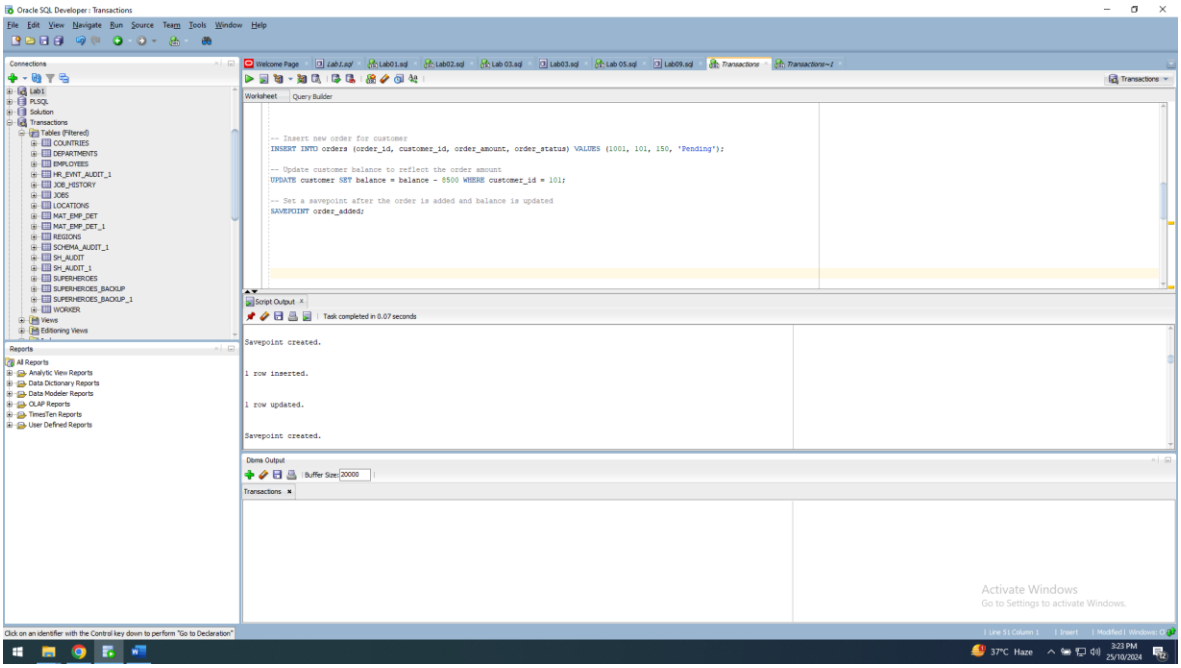
Step 1: Insert a New Customer

Start by inserting a new customer into the customer table and creating a savepoint to rollback to this step if necessary.



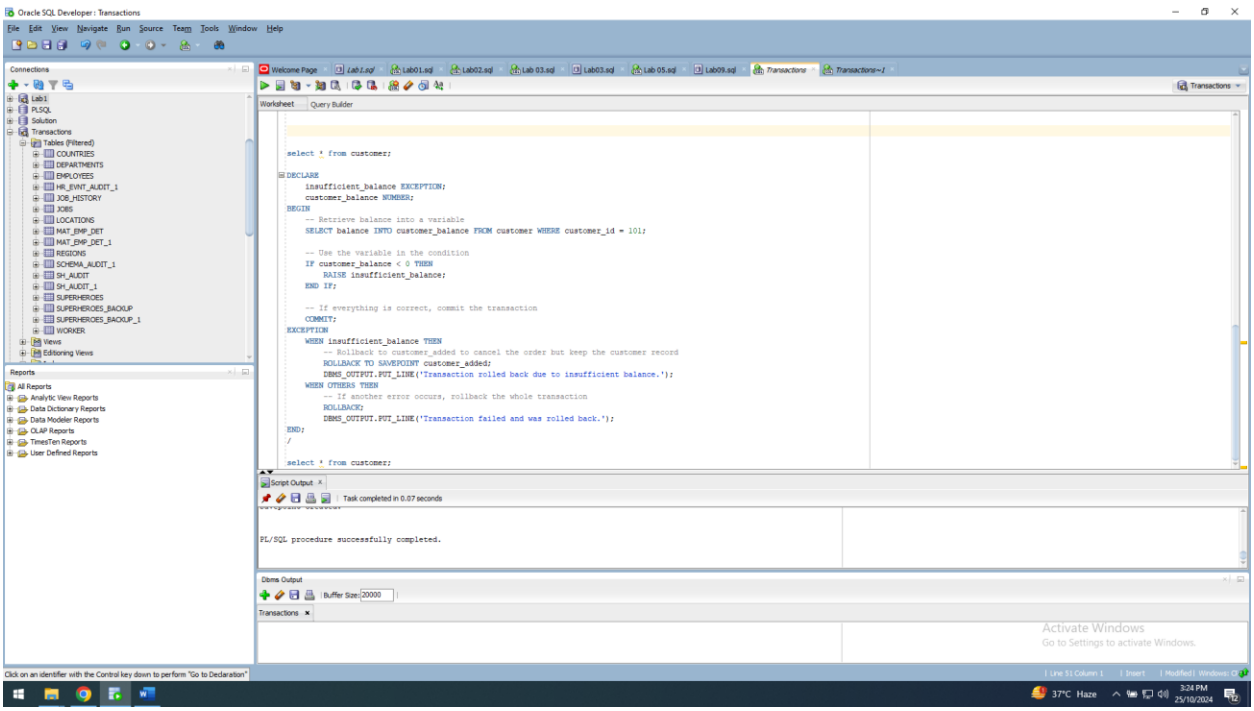
Step 2: Insert an Order and Deduct Balance

Now, insert a new order for the customer and update their balance. If updating the balance fails, we will rollback to the savepoint.



Step 3: Commit or Rollback

If all statements execute successfully, we can commit the transaction. However, if any statement fails (such as insufficient balance), rollback to the appropriate savepoint or to the beginning of the transaction.



Explanation of Flow:

1. **Start the Transaction** by setting a name (customer_order_transaction).
2. **Insert Customer Data**: Insert a new customer with an initial balance and set a savepoint (customer_added).
3. **Insert Order Data** and **Update Balance**: Deduct the order amount from the customer's balance and create another savepoint (order_added).
4. **Commit or Rollback**:
    - If all operations succeed and the balance is sufficient, commit the transaction.
    - If there's an issue (like insufficient balance), rollback to customer_added to cancel the order but retain the customer.
    - If there's another type of error, rollback the entire transaction.

Class Task:

Try to create a similar example where:

- You add a new product to a product table.
- Add an inventory entry and set a savepoint.
- Deduct from inventory on order placement.
- Rollback if the inventory quantity goes negative, ensuring inventory consistency.