# Assignment No 8

**1]** Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

Solution:-

1. Retrieve all columns from the 'customers' table:

```
select * from customer;
```

2.Return only the customer name and email address for customers in a specific city (let's assume the city is 'Satara'):

```
select customer_name,email_add from customer where city = ‘Satara’;
```

**2]** Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

Solution:-

1] INNER JOIN QUERY:-

```
SELECT c.customer_id, c.customer_name, c.region, o.order_id
  FROM customer c
  INNER JOIN orders o ON c.customer_id = o.customer_id
 WHERE c.region = 'Specified_Region';
```

2] LEFT JOIN QUERY:-

```
SELECT c.customer_id, c.customer_name, c.region, o.order_id

FROM customer c

LEFT JOIN orders o ON c.customer_id = o.customer_id

 WHERE c.region = 'Specified_Region';
```

**3]** Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

Solution:-

1.Subquery to find customers with orders above the average order value:

```
SELECT customer_id

 FROM orders

GROUP BY customer_id

HAVING AVG(order_value) > (SELECT AVG(order_value) FROM orders);
```

2.UNION query to combine two SELECT statements with the same number of columns:

```
(SELECT customer_id, customer_name, order_id, order_value

 FROM customer

INNER JOIN orders ON customers.customer_id = orders.customer_id)

UNION
```

```
(SELECT customer_id, customer_name, NULL AS order_id, NULL AS
order_value

FROM customer

WHERE customer_id IN (

SELECT customer_id

FROM orders

GROUP BY customer_id

HAVING AVG(order_value) > (SELECT AVG(order_value) FROM   orders)));
```

**4]** Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

Solution:-

```
-- BEGIN a transaction

    BEGIN TRANSACTION;



-- INSERT a new record into the 'orders' table

INSERT INTO orders (order_id, customer_id, order_date,   total_amount)

VALUES (12345, 1001, '2024-05-25', 150.00);
```

```
-- COMMIT the transaction

COMMIT;



-- UPDATE the 'products' table

UPDATE products

SET stock_quantity = stock_quantity - 1

WHERE product_id = 'ABC123';



-- ROLLBACK the transaction (assuming something went wrong)

ROLLBACK;
```

**5]** Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

Solution:-

```
-- BEGIN a transaction

BEGIN TRANSACTION;
```

```
-- Perform first INSERT into 'orders' and set SAVEPOINT

INSERT INTO orders (order_id, customer_id, order_date,
total_amount)

VALUES (1, 1001, '2024-05-25', 150.00);

SAVEPOINT sp1;



-- Perform second INSERT into 'orders' and set SAVEPOINT

INSERT INTO orders (order_id, customer_id, order_date,
total_amount)

VALUES (2, 1002, '2024-05-26', 200.00);

SAVEPOINT sp2;



-- Perform third INSERT into 'orders' and set SAVEPOINT

INSERT INTO orders (order_id, customer_id, order_date,
total_amount)

VALUES (3, 1003, '2024-05-27', 175.00);

SAVEPOINT sp3;



-- Rollback to the second SAVEPOINT
```

```
ROLLBACK TO SAVEPOINT sp2;


-- COMMIT the overall transaction

COMMIT;
```

**6]** Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Solution:-

## Report on the Use of Transaction Logs for Data Recovery

### Introduction:

Transaction logs play a crucial role in ensuring data integrity and facilitating recovery in database management systems. They record all changes made to a database, providing a detailed history of transactions. In the event of system failures, such as unexpected shutdowns or hardware malfunctions, transaction logs enable the restoration of the database to a consistent state, minimizing data loss and ensuring business continuity.

### Functionality of Transaction Logs:

Transaction logs track every transaction executed on a database, including INSERTs, UPDATEs, DELETEs, and schema modifications. For each transaction, the log records the before and after images of the modified data, along with a timestamp and other relevant metadata. This allows for the reconstruction of database changes, even in the face of system failures.

### Data Recovery Process:

In the event of an unexpected shutdown or system failure, the database management system (DBMS) utilizes transaction logs to restore the database to a consistent state. The recovery process typically involves the following steps:

1. **Analysis Phase:** The DBMS examines the transaction logs to determine the last committed transaction before the failure occurred. This information helps identify the point at which the database become inconsistent.
2. **Rollback or Rollforward:** Depending on the analysis, the DBMS may perform a rollback to undo incomplete transactions that occurred after the last committed transaction. Alternatively, it may execute a rollforward process to reapply transactions that were committed but not yet written to disk at the time of the failure.
3. **Redo and Undo Operations:** Using the transaction logs, the DBMS performs redo operations to reapply committed transactions that were not persisted to disk. It also executes undo operations to reverse any incomplete transactions and restore the database to its pre-failure state.
4. **Database Recovery:** Once the redo and undo operations are completed, the database is brought online, and normal operations resume. The transaction logs are then archived or truncated to free up storage space for future transactions.

### Hypothetical Scenario:

Consider a hypothetical scenario where a retail company operates a centralized database to manage its inventory and sales transactions. During peak business hours, the database server experiences an unexpected shutdown due to a power outage. As a result, the database becomes inaccessible, leading to concerns about data integrity and potential losses.

However, thanks to the robust transaction logging mechanism implemented by the DBMS, the company's IT team can initiate the data recovery process with confidence. By analyzing the transaction logs, they identify the point of failure and determine the last committed transaction before the shutdown.

Using this information, the DBMS performs a roll forward process to reapply committed transactions that were not yet written to disk. Additionally, it executes rollback operations to undo any incomplete transactions and restore the database to a consistent state.

As a result of the transaction log's meticulous recording of database changes, the company successfully recovers all inventory and sales data up to the moment of the unexpected shutdown. Business operations resume seamlessly, and the company can continue serving its customers without significant disruptions.

## Conclusion:

Transaction logs are indispensable for ensuring data recovery and maintaining data integrity in database management systems. By recording all transactions and changes to the database, transaction logs enable the restoration of databases to consistent states following system failures or unexpected shutdowns. As demonstrated in the hypothetical scenario, the use of transaction logs is instrumental in mitigating data loss and ensuring business continuity in the face of adversity