**Assignment 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.**

**Ans:**

**To retrieve all columns:**

**Query:**

SELECT \* FROM customers;

**To retrieve only the** **customer name and email address for customers in a specific city:**

**Query:**

SELECT customer\_name, email\_address FROM customers WHERE city = 'Mumbai';

**Assignment 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.**

**Ans:**

**1. INNER JOIN - Customers with Orders in a Specified Region**

**Query:**

SELECT o.customer\_id, c.name, c.region FROM orders o INNER JOIN customers c ON o.customer\_id = c.customer\_id WHERE c.region = 'Delhi';

This query uses an INNER JOIN to combine the orders and customers tables. It selects the customer\_id, name, and region columns. The ON clause specifies that the rows should be matched where the customer\_id in the orders table is equal to the customer\_id in the customer table. The WHERE clause filters the results to include only customers in the specified region i.e. Mumbai ,Delhi etc.

**2. LEFT JOIN - All Customers (Including Those Without Orders)**

**Query:**

SELECT c.customer\_id, c.name, c.region, o.order\_id FROM customers c  
LEFT JOIN orders o ON o.customer\_id = c.customer\_id;

This query uses a LEFT JOIN to combine the customers and orders tables. It selects the customer\_id, name, and region columns from the customers table. The LEFT JOIN ensures that all rows from the customers table are included, even if there are no matching orders in the orders table. The ON clause specifies the join condition as before. You can optionally include the order\_id from the orders table, but it will be NULL for customers without orders.

**Explanation of the Difference:**

* INNER JOIN only returns rows where there's a match in both tables (customers with orders in the specified region).
* LEFT JOIN returns all rows from the left table (customers) and matching rows from the right table (orders), with NULL values for missing order information.

**Assignment 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.**

**Ans:**

**Subquery to find customers who have placed orders above the average order value:**

SELECT customer\_id, customer\_name FROM orders WHERE order \_value > (SELECT avg(order\_value) FROM orders);

This subquery retrieves the customer IDs and customer\_name of those who have placed orders above the average order value.

**Combining with UNION:**

**Query:**

SELECT customer\_id FROM orders WHERE order\_value > (SELECT avg(order\_value) FROM orders);

UNION

SELECT customer\_id FROM customers WHERE customer\_id NOT IN (SELECT customer\_id FROM orders);

This query first selects the customer IDs who have placed orders above the average value and then combines them with customer IDs who haven't placed any orders using the UNION operator. Both SELECT statements in the UNION query have the same number of columns.

**Assignment 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.**

**Ans:**

**Query:**

BEGIN TRANSACTION;

**Inserting a new record into the 'orders' table:**

INSERT INTO orders (id, order\_date, order\_value) VALUES (1, '2024-05-25', 200);

**Committing the transaction:**

COMMIT;

**Updating the 'products' table:**

UPDATE products SET price = price \* 1.15 WHERE category = ' grocery ';

**Rolling back the transaction:**

ROLLBACK;

**Explanation:**

We begin a transaction with BEGIN TRANSACTION.

We insert a new record into the 'orders' table.

We commit the transaction with COMMIT, making the changes permanent in the database.

We update the 'products' table, increasing the price of products in the ' grocery ' category by 15%.

We decide to roll back the transaction using ROLLBACK, reverting any changes made after the transaction began.

**Assignment 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.**

**Ans:**

**Query:**

BEGIN TRANSACTION;

**Inserting First record:**

INSERT INTO orders (customer\_id, order\_date, order\_value) VALUES (1, '2024-05-23', 250);

SAVEPOINT savepoint1;

**Inserting Second record:**

INSERT INTO orders (customer\_id, order\_date, order\_value) VALUES (2, '2024-05-24', 150);

SAVEPOINT savepoint2;

**Inserting Third record:**

INSERT INTO orders (customer\_id, order\_date, order\_value)

VALUES (3, '2024-05-30', 200);

**Rollback to the second SAVEPOINT:**

ROLLBACK TO SAVEPOINT savepoint2;

**COMMIT the overall transaction:**

COMMIT;

**Explanation:**

In this script, we begin a transaction with BEGIN TRANSACTION. Then we perform a series of INSERTs into the 'orders' table, setting a SAVEPOINT after each INSERT. After the third INSERT, we rollback to the second SAVEPOINT using ROLLBACK TO SAVEPOINT savepoint2. Finally, we COMMIT the overall transaction to make the changes permanent.

**Assignment 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.**

**Ans:**

**Report:** The Use of Transaction Logs for Data Recovery

**Introduction:**

Transaction logs play a crucial role in database management systems by recording all transactions and modifications made to the database. These logs serve as a detailed record of changes, enabling recovery to a consistent state in case of system failures, crashes, or other unforeseen events. In this report, we'll explore the importance of transaction logs for data recovery and present a hypothetical scenario to illustrate their significance.

**Importance of Transaction Logs:**

Transaction logs are essential for ensuring data integrity and facilitating recovery operations. Here's why they are vital:

**Redundancy:** Transaction logs provide redundancy by storing a chronological record of all database modifications, including inserts, updates, and deletes.

**Point-in-Time Recovery:** Transaction logs allow for point-in-time recovery, enabling restoration of the database to a specific timestamp before the occurrence of a failure or error.

**Rollback and Rollforward Operations:** Transaction logs support rollback and rollforward operations, allowing DBAs to undo or redo transactions as needed to recover from failures.

**Minimized Data Loss:** With transaction logs, organizations can minimize data loss by replaying logged transactions to restore lost or corrupted data.

**Audit Trail:** Transaction logs serve as an audit trail, providing a detailed history of database changes for compliance and forensic purposes.

**Hypothetical Scenario:**

Consider a scenario in which a retail company operates a database to manage its inventory and sales transactions. During a routine system update, an unexpected power outage occurs, leading to an abrupt shutdown of the database server. As a result, the database becomes inaccessible, and critical sales data may be at risk of loss or corruption.

In this situation, the transaction logs prove instrumental in data recovery. Here's how:

**Identification of Incomplete Transactions:** Upon restarting the database server, the system recognizes the abrupt shutdown and identifies any incomplete transactions at the time of the failure by analyzing the transaction logs.

**Transaction Replay:** The transaction logs contain a record of all committed and uncommitted transactions up to the point of failure. DBAs can use this information to replay the transactions and bring the database to a consistent state.

**Point-in-Time Recovery:** With the help of transaction logs, the DBAs can perform a point-in-time recovery, selecting a timestamp just before the unexpected shutdown occurred. This allows them to restore the database to a state where data integrity is ensured.

**Data Reconstruction:** In cases where data loss or corruption has occurred, transaction logs can be used to reconstruct the lost or damaged data by applying logged changes to the database.

**Conclusion:**

Transaction logs are indispensable for ensuring data consistency, integrity, and recoverability in database systems. By maintaining a detailed record of all transactions, these logs enable organizations to recover from system failures and other disruptions while minimizing data loss and ensuring business continuity.

In the hypothetical scenario described above, the transaction log serves as a lifeline for data recovery, allowing the retail company to restore its database to a consistent state and safeguard critical business information. As such, investing in robust transaction logging mechanisms and disaster recovery strategies is essential for any organization that relies on data availability and reliability.