ASSIGNMENT –(DAY 9 & 10TH)

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.

Solution :-

- **SELECT * FROM EMP;**
- **SELCT CNAME, EMAIL FROM EMP WHERE CITY='BANGALORE';**

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.

Solution :-

```
SELECT c.customer_name, c.email_address, o.order_id, o.order_date
FROM customers c LEFT JOIN orders o
ON c.customer_id = o.customer_id
WHERE c.region = 'Bangalore';
```

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.

Solution :-

```
SELECT customer_id, customer_name FROM customers c
WHERE c.customer_id IN (
  SELECT customer_id
  FROM orders o
  WHERE o.order_value > (
    SELECT AVG(order_value)
    FROM orders
  )
);
```

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.

Solution :-

```
BEGIN TRANSACTION;

-- Insert a new record into the 'orders' table (replace with your actual column names and values)
INSERT INTO orders (customer_id, order_date, order_total)
VALUES (123, '2024-05-23', 100.00);

-- Assuming the insert is successful, commit the transaction
COMMIT;

-- Update the 'products' table (replace with your actual column names and values)
-- This update will be rolled back since we use ROLLBACK later
UPDATE products
SET stock_level = stock_level - 1
WHERE product_id = 456;

-- Simulate an error or unexpected issue to demonstrate rollback
RAISE ERROR ('Something unexpected happened!');

-- Rollback the entire transaction, effectively undoing the insert
ROLLBACK;
```

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.

Solution :-

```
BEGIN TRANSACTION;

-- Insert 1st order (replace with your actual column names and values)
INSERT INTO orders (customer_id, order_date, order_total)
VALUES (123, '2024-05-23', 100.00);

-- Savepoint after 1st insert (name can be anything descriptive)
SAVEPOINT order1_inserted;

-- Insert 2nd order
INSERT INTO orders (customer_id, order_date, order_total)
VALUES (456, '2024-05-23', 50.00);

-- Savepoint after 2nd insert
SAVEPOINT order2_inserted;
```

```
-- Insert 3rd order (optional, for demonstration purposes)
INSERT INTO orders (customer_id, order_date, order_total)
VALUES (789, '2024-05-23', 200.00);

-- Simulate an issue after the 3rd insert (can be removed for actual use)
RAISE WARNING ('Issue encountered processing order!');

-- Rollback to the second savepoint (order2_inserted)
ROLLBACK TO SAVEPOINT order2_inserted;

-- We only want the first two orders, so commit the transaction from here
COMMIT;
```

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.

Solution :-

Transaction Logs: Guardians of Data Integrity

Transaction logs are the unsung heroes of database management. These chronological records of all database modifications play a vital role in ensuring data consistency and enabling recovery from unexpected events.

How Transaction Logs Work:

Each database transaction, encompassing a series of reads and writes, is meticulously documented in the transaction log. This includes:

- **Start and End:** The timestamp marking the beginning and completion of the transaction.
- **Data Changes:** Detailed records of all modifications made to the database, including inserts, updates, and deletes.
- **Rollback Information:** Essential data needed to undo the transaction if necessary.

Data Recovery with Transaction Logs:

There are two primary ways transaction logs facilitate data recovery:

1. **Rollback:** If a transaction fails midway due to errors or system crashes, the log allows the database to revert to its state before the transaction began. This ensures data integrity and prevents partial or incomplete updates.
2. **Redoing Uncommitted Changes:** In case of a system crash after a successful transaction but before it's committed to the database, the log helps redo the changes. This ensures that valid modifications are not lost.

Scenario: Recovering from the Unexpected

Imagine a busy online store experiencing a sudden power outage. The system abruptly shuts down while a customer is finalizing a purchase. Panic sets in! Did the order get recorded? Did the customer's payment go through?

Fortunately, the database utilizes transaction logs. Here's how it helps:

1. **Identifying Incomplete Transactions:** Upon restarting, the database system analyzes the transaction log. It identifies the unfinished purchase as the last logged transaction without a "commit" mark.
2. **Data Recovery:** Using the logged information, the system can either:
    - **Rollback the Transaction:** If the payment processing wasn't complete, the log facilitates reversing any temporary changes made to the inventory or customer account.
    - **Redo the Transaction:** If the payment was successful but not committed, the log allows the system to redo the purchase, ensuring the order is recorded and the customer's account is updated.