ANJALI KUMARI

DAY- 9,10 SQL

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.

-- Retrieve all columns from the 'customers' table

```
SELECT * FROM customers;
```

-- Modify to return only the customer name and email address for customers in a specific city

```
SELECT customer_name, email_address
FROM customers
WHERE city = 'specific_city';
```

Replace 'specific_city' with the actual city you want to query for. This SQL statement will retrieve the customer name and email address from the 'customers' table for customers located in the specified

---

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.

-- Combine 'orders' and 'customers' tables using INNER JOIN for customers in a specified region

```
SELECT orders.*, customers.*
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id
WHERE customers.region = 'specified_region';
```

-- Use LEFT JOIN to display all customers including those without orders

```
SELECT customers.*, orders.*
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

Replace 'specified_region' with the actual region you want to query for. The first query combines the 'orders' and 'customers' tables using INNER JOIN based on the customer ID and filters customers based on the specified region. The second query uses LEFT JOIN to display all customers along with their corresponding orders, including those customers who don't have any orders.

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.

Using a subquery to find customers who have placed 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);
```

This query selects the customer IDs from the 'orders' table, groups them by customer ID, and then filters out those customers whose average order value is above the overall average order value.

Writing a UNION query to combine two SELECT statements with the same number of columns:

```
SELECT customer_name, email_address

FROM customers

WHERE city = 'specific_city'

UNION

SELECT customer_name, email_address

FROM customers

WHERE region = 'specified_region';
```

This query selects the customer name and email address from the 'customers' table for customers in a specific city and then combines it with the customer name and email address for customers in a specified region using UNION. 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.

-- BEGIN a transaction

BEGIN TRANSACTION;

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

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

VALUES (NEW_ID(), 'customer_id_value', '2024-05-10', 100.00); -- Assuming NEW_ID() generates a new unique order ID and 'customer_id_value' is the customer ID for whom the order is being placed.

-- COMMIT the transaction

COMMIT;

-- UPDATE the 'products' table

UPDATE products

SET stock_quantity = stock_quantity - 1

WHERE product_id = 'product_id_value'; -- Assuming 'product_id_value' is the ID of the product being updated.

-- ROLLBACK the transaction

ROLLBACK;

Replace 'customer_id_value' with the actual customer ID for whom the order is being placed, and 'product_id_value' with the actual product ID being updated in the 'products' table. This sequence of SQL statements will start a transaction, insert a new record into the 'orders' table, commit the transaction, update the 'products' table, and then rollback the transaction in case of any issues.

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.

```
-- BEGIN a transaction

BEGIN TRANSACTION;

-- Perform a series of INSERTs into 'orders', setting a SAVEPOINT after each

SAVEPOINT savepoint1;

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

VALUES (NEW_ID(), 'customer_id_1', '2024-05-10', 100.00);

SAVEPOINT savepoint2;

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

VALUES (NEW_ID(), 'customer_id_2', '2024-05-11', 150.00);

-- Rollback to the second SAVEPOINT

ROLLBACK TO SAVEPOINT savepoint2;

-- COMMIT the overall transaction

COMMIT;
```

Replace 'customer_id_1' and 'customer_id_2' with the actual customer IDs for the orders being inserted, and ensure that NEW_ID() generates a new unique order ID. This sequence of SQL statements will start a transaction, insert orders into the 'orders' table with savepoints set after each insertion, rollback to the second savepoint, and finally commit the overall transaction.

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.

Transaction logs play a vital role in ensuring data integrity and facilitating recovery in the event of system failures or unexpected shutdowns. These logs record every transaction that modifies the database, providing a detailed history of changes made over time. By replaying these logged transactions, it's possible to restore the database to a consistent state, thus minimizing data loss and ensuring business continuity.

Scenario:

In a hypothetical scenario, consider a large e-commerce platform that relies heavily on its database to manage orders, inventory, and customer information. During a routine system upgrade, an unexpected power outage occurs, causing the database server to shut down abruptly. As a result, the database becomes corrupted, and crucial transactional data is lost.

However, due to the presence of transaction logs, the database administrators are able to initiate a recovery process. By analyzing the transaction logs, they identify the last committed transactions before the shutdown. Using this information, they roll forward the transactions, reapplying the changes recorded in the logs to restore the database to its state just before the outage.

For example, let's say the transaction log recorded that a customer placed an order for a product and the inventory was updated accordingly. After the unexpected shutdown, the database rollback process would replay this transaction from the logs, ensuring that the order is not lost and the inventory reflects the correct stock levels.

By leveraging transaction logs, the database administrators successfully recover the database without significant data loss or downtime. This highlights the critical role of transaction logs in ensuring data resilience and enabling efficient recovery procedures in the face of unforeseen events.

In conclusion, transaction logs serve as a valuable mechanism for data recovery, allowing organizations to maintain data integrity and minimize the impact of system failures. Implementing robust logging practices and backup strategies is essential for safeguarding against data loss and ensuring the continuity of business operations.
