Data Query Language Assignment

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.

Let us take an example of a SELECT query to retrieve all columns from a 'customers' table, and a modified version 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;

- Retrieve customer name and email for customers in a specific city:

  SELECT customer_name, email

  FROM customers

  WHERE city = 'New York';

This modified query will return only the 'customer_name' and 'email' columns from the 'customers' table, but it will filter the results to include only the rows where the 'city' column matches 'New York'.

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.

INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region:

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
WHERE c.region = 'West'
```

- The INNER JOIN ensures that only the matching rows between the two tables are included in the result set. If a customer has no orders or an order has no matching customer, those rows will not be included.

LEFT JOIN to display all customers including those without orders:

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
```

- The LEFT JOIN ensures that all rows from the left table ('customers') are included in the result set, even if there are no matching rows in the right table ('orders'). In such cases, the columns from the right table will contain NULL values.

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.

Subquery to find customers with orders above the average order value

```
SELECT c.customer_id, c.customer_name, o.order_value
FROM customers c
```

```
INNER JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_value > (
   SELECT AVG(order_value)
   FROM orders
)
```

- The subquery (SELECT AVG(order_value) FROM orders) is used to get the average order value, which is then used in the main query's WHERE clause.

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

```
SELECT customer_id, customer_name, email
FROM customers
WHERE region = 'East'

UNION
SELECT customer_id, customer_name, email
FROM customers
WHERE region = 'West'
```

- This query uses the UNION operator to combine the results of two SELECT statements. Both SELECT statements have the same number of columns (customer_id, customer_name, email) and compatible data types.

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 TRANSACTION;

#Insert a new record into the 'orders' table

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

VALUES (101, 1, 10, '2023-05-25', 100.00);

#Commit the transaction

COMMIT TRANSACTION;

#Update the 'products' table

UPDATE products

SET product_name = 'Updated Product'

WHERE product_id = 10;

#Rollback the transaction

ROLLBACK TRANSACTION;
```

1. BEGIN TRANSACTION: This statement starts a new transaction, which groups multiple SQL statements together as a single unit of work.
2. INSERT INTO orders: Inside the transaction block, a new record is inserted into the 'orders' table. This INSERT statement is part of the transaction.
3. COMMIT TRANSACTION: If all the statements within the transaction block execute successfully, the COMMIT statement permanently saves the changes to the database. In this case, the new order record will be inserted into the 'orders' table.

4. UPDATE products: After the transaction is committed, an UPDATE statement is executed to modify a record in the 'products' table.
5. ROLLBACK TRANSACTION: If an error occurs or if you want to undo the changes made within the transaction, the ROLLBACK statement is used. In this example, the UPDATE statement on the 'products' table will be rolled back, and the changes will not be saved to the database.

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 the transaction

BEGIN TRANSACTION;

# Insert the first record into the 'orders' table

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

VALUES (101, 1, 10, '2023-05-25', 100.00);

# Set the first SAVEPOINT

SAVEPOINT savepoint1;

# Insert the second record into the 'orders' table

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

VALUES (102, 2, 11, '2023-05-26', 150.00);

# Set the second SAVEPOINT

SAVEPOINT savepoint2;

# Insert the third record into the 'orders' table
```

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

VALUES (103, 3, 12, '2023-05-27', 200.00);

# Rollback to the second SAVEPOINT

ROLLBACK TO SAVEPOINT savepoint2;

# Commit the overall transaction

COMMIT TRANSACTION;
```

In this sequence of SQL statements:

- The transaction is initiated with BEGIN TRANSACTION.
- Three INSERT statements are executed to add records to the 'orders' table.
- SAVEPOINTs are set after the first and second INSERTs to mark specific points in the transaction.
- After the third INSERT, a ROLLBACK TO SAVEPOINT is used to revert the transaction back to the state at the second SAVEPOINT, effectively undoing the last INSERT.
- Finally, the changes up to the second SAVEPOINT are committed using COMMIT 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 for Data Recovery

- Transaction logs, also known as redo logs or write-ahead logs, are an essential component of database management systems. They serve as a record of all the changes made to the database, including INSERT, UPDATE, and DELETE operations. Transaction logs play a crucial role in ensuring data integrity and enabling

data recovery in the event of system failures, hardware malfunctions, or unexpected shutdowns.

- When a transaction is executed, the changes are first recorded in the transaction log before being applied to the actual database files. This process ensures that the database can be restored to a consistent state even if an unexpected shutdown occurs before the changes are fully committed to the database.
- In the event of a system failure or unexpected shutdown, the database management system can use the transaction log to recover the database by:
- Undoing any uncommitted transactions that were interrupted by the shutdown.
- Redoing any committed transactions that were not fully applied to the database files before the shutdown occurred.
- This process ensures that the database is restored to a consistent state, with all committed transactions applied and any uncommitted transactions rolled back.

Hypothetical Scenario: Data Recovery after Unexpected Shutdown

- Consider a scenario where a company's e-commerce website is running on a database server. The website allows customers to place orders, update their profiles, and manage their shopping carts. The database consists of tables such as orders, customers, products, and carts.
- One day, during peak business hours, an unexpected power outage occurs, causing the database server to shut down abruptly. When the power is restored and the server is brought back online, the database management system automatically initiates the recovery process using the transaction log.
- The recovery process involves the following steps:

1. Undo
2. Redo