SQL ASSIGNMENTS(1-6) 22-05

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.

SQL Commands:

```
CREATE TABLE Customers (
   ID NUMBER(2) PRIMARY KEY,
   Name VARCHAR(50) NOT NULL,
   Age NUMBER(2),
   City VARCHAR(50),
   Email_Address NUMBER NOT NULL);

INSERT INTO Customers VALUES (1, ‘Ravi’, 23, ‘Rajapalayam’, ‘ravi@gmail.com’);
INSERT INTO Customers VALUES  (2, ‘Shan’, 22, ‘Chennai’, ‘shan@gmail.com’);
INSERT INTO Customers VALUES  (3, ‘Siva’,  25,‘Rajapalayam’, ‘siva@gmail.com’);
INSERT INTO Customers VALUES  (4, ‘Suriya', 24, ‘Madurai’, ‘suriya@gmail.com’);

SELECT * FROM Customers;
```

Results:

ID	**Name**	**Age**	**City**	**Email_Address**
1	Ravi	23	Rajapalayam	ravi@gmail.com
2	Shan	22	Chennai	shan@gmail.com
3	Siva	25	Rajapalayam	siva@gmail.com
4	Suriya	24	Madurai	suriya@gmail.com

SQL Commands:

```
SELECT Name, Email_Address
FROM Customers
WHERE City = ‘Rajapalayam’;
```

Results:

Name	**Email_Address**
Ravi	ravi@gmail.com
Siva	siva@gmail.com

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.

SQL Commands:

```
SELECT *
FROM Customers
LEFT JOIN Orders ON Customers.id = Orders.id
WHERE Customers.City = 'rajapalayam';
```

Results:

Customer_ID	**Customer_Name**	**Customer_Age**	**Customer_City**	**Order_Item**	**Order_Quantity**	**Order_ID**
1	Ravi	23	rajapalayam	Mobile Phone	1	1
2	Siva	26	rajapalayam	Laptop	1	2

Explanation:

- **SELECT *** : using to selecting all columns from both the **'Customers'** and '**Orders'** tables.
- **LEFT JOIN** : to ensure that all customers from the **'Customers'** table are included in the result, regardless of whether they have corresponding orders in the 'Orders' table.
- **INNER JOIN** : implicitly when we specify **Customers.id = Orders.id**, which combines only those customers who have orders and are in the **City = ‘rajapalayam’**.
- **WHERE** : This clause filters the result to only include customers from the **City = ‘rajapalayam’**.

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.

SQL Commands:

```
// Find Customers with Orders Above Average Order Value

SELECT Customer_ID, Name
FROM Customer
WHERE Customer_ID IN (
    SELECT o.Customer_ID
    FROM Order o
    GROUP BY o.Customer_ID
    HAVING AVG(o.TotalAmount) > (
        SELECT AVG(TotalAmount)
        FROM Order
    )
);


//  Union Query

SELECT Customer_ID, Name
FROM Customer_A
UNION
SELECT Customer_ID, Name
FROM Customer_B;
```

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.

SQL Commands:

```
BEGIN TRANSACTION;

INSERT INTO orders (customer_id, customer_name, order_item,order_date)
VALUES (5, 'Guna', ‘Laptop’, ‘2024-05-22’);

COMMIT;

UPDATE products
SET stock_quantity = stock_quantity - 1
WHERE product_id = 456;

ROLLBACK;
```

Explanation:

- **BEGIN TRANSACTION** command for start a transaction.
- Insert a new record into the **'orders'** table.
- **COMMIT** command for commit the transaction to make the changes permanent.
- Update the **'products'** table to decrement the stock quantity of a particular product.
- **ROLLBACK** command for rollback the transaction rollback to undo any changes made since 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.

SQL Commands:

```
BEGIN TRANSACTION;

SAVEPOINT savepoint1;
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES (1, 123, '2024-05-24', 100.00);

SAVEPOINT savepoint2;
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES (2, 456, '2024-05-25', 150.00);

ROLLBACK TO SAVEPOINT savepoint2;

COMMIT;
```

Explanation:

- **BEGIN TRANSACTION** command for start a transaction using.
- **INSERT** statement, we set a savepoint using **SAVEPOINT** to mark a point to which we can rollback later.
- After the second **INSERT**, we rollback to the second savepoint using **ROLLBACK TO SAVEPOINT** savepoint2. This will undo the changes made after the second savepoint.
- **COMMIT** command for commit the overall transaction, which makes the changes up to that point 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.

Solution:

Uses of Transaction Logs:

- **Point-in-Time Recovery:** Transaction logs allow for point-in-time recovery, enabling administrators to restore the database to a specific moment before the occurrence of a failure or error.
- **Data Integrity:** By recording all transactions, including commits and rollbacks, transaction logs help maintain data integrity and consistency.
- **Minimized Data Loss:** In the event of a system failure, transaction logs enable administrators to replay transactions and recover lost or corrupted data, minimizing data loss.
- **Audit Trail:** Transaction logs serve as an audit trail, providing a chronological record of database activities for compliance and forensic purposes.
- **Performance Monitoring:** Analyzing transaction logs can help identify performance issues, optimize database performance, and troubleshoot errors.

Hypothetical scenario :

Imagine a scenario where a multinational retail corporation operates a centralized database system to manage inventory and sales data across its global chain of stores.

We have to analyze the transaction logs to identify the last successfully committed transactions before the shutdown. Using this information, we can roll back incomplete transactions and replay committed transactions from the transaction logs, restoring the database to a consistent state as of the last recorded transaction.

As a result, the retail corporation successfully recovers its inventory and sales data with minimal disruption to operations. The transaction logs prove instrumental in ensuring data integrity and facilitating rapid recovery, demonstrating the critical importance of transaction logging for data management and recovery in enterprise environments.