Lab Manual

Task 5: Writing Join Queries, Equivalent, and/or Recursive Queries

Case Study: Online Food Ordering System

Objective: To perform advanced query processing and test its heuristics by designing optimized complex queries and their equivalent forms, including recursive queries, for the Online Food Ordering System.

1. Join Queries

Query 1: Retrieve all orders along with the corresponding customer's name.

```
SELECT o.Order_ID, o.Order_Date, o.Order_Total, c.Cust_Name
FROM OrderTable o
JOIN Customer c ON o.Cust_ID = c.Cust_ID;
```

Expected Output:

Order_ID	Order_Date	Order_Total	Cust_Name
1	2025-01-20	800	Alice
2	2025-01-21	500	Bob
3	2025-01-22	700	Charlie

Query 2: Retrieve all menu items along with the restaurant name that offers them.

```
SELECT m.Item_Name, m.Price, r.Rest_Name
FROM Menu_Item m
JOIN Restaurant r ON m.Rest_ID = r.Rest_ID;
```

Expected Output:

Item_Name	Price	Rest_Name
Pizza	450	Food Paradise
Burger	270	Food Paradise
Sushi	720	Tasty Treats
Pasta	360	Food Paradise
Noodles	315	Global Eats

Query 3: Retrieve all orders and their delivery status.

```sql
SELECT o.Order_ID, o.Order_Total, d.Delivery_Status, d.Delivery_Time
FROM OrderTable o
LEFT JOIN Delivery d ON o.Order_ID = d.Order_ID;
```

Expected Output:

Order_ID	Order_Total	Delivery_Status	Delivery_Time
1	800	Delivered	2025-01-20 14:30:00
2	500	Pending	NULL
3	700	Delivered	2025-01-22 16:00:00

INNER JOIN

An **INNER JOIN** retrieves records that have matching values in both tables.

Query: Retrieve all orders along with their customer names.

```sql
SELECT o.Order_ID, o.Order_Date, o.Order_Total, c.Cust_Name
FROM OrderTable o
INNER JOIN Customer c ON o.Cust_ID = c.Cust_ID;
```

Expected Output:

Order_ID	Order_Date	Order_Total	Cust_Name
1	2025-01-20	800	Alice
2	2025-01-21	500	Bob
3	2025-01-22	700	Charlie

LEFT OUTER JOIN

A **LEFT OUTER JOIN** retrieves all records from the left table and the matched records from the right table. If no match is found, NULL is returned for columns from the right table.

Query: Retrieve all customers, even those who haven't placed any orders.

```sql
SELECT c.Cust_Name, o.Order_ID, o.Order_Total
FROM Customer c
LEFT JOIN OrderTable o ON c.Cust_ID = o.Cust_ID;
```

Expected Output:

Cust_Name	Order_ID	Order_Total
Alice	1	800
Bob	2	500
Charlie	3	700

RIGHT OUTER JOIN

A **RIGHT OUTER JOIN** retrieves all records from the right table and the matched records from the left table. If no match is found, NULL is returned for columns from the left table.

Query: Retrieve all orders and the names of customers who placed them. Include orders even if the customer details are missing.

```
SELECT o.Order_ID, o.Order_Total, c.Cust_Name
FROM OrderTable o
RIGHT JOIN Customer c ON o.Cust_ID = c.Cust_ID;
```

Expected Output:

Order_ID	Order_Total	Cust_Name
1	800	Alice
2	500	Bob
3	700	Charlie

FULL OUTER JOIN

A **FULL OUTER JOIN** retrieves all records from both tables. If no match is found, NULL is returned for unmatched rows from either table.

Query: Retrieve all customers and all orders, even if there is no match.

```
SELECT c.Cust_Name, o.Order_ID, o.Order_Total
FROM Customer c
FULL OUTER JOIN OrderTable o ON c.Cust_ID = o.Cust_ID;
```

Expected Output:

Cust_Name	Order_ID	Order_Total
Alice	1	800
Bob	2	500
Charlie	3	700

Note: Oracle doesn't support FULL OUTER JOIN directly. Use `UNION` of LEFT JOIN and RIGHT JOIN.

CROSS JOIN

A **CROSS JOIN** returns the Cartesian product of the two tables. Every row from the first table is combined with every row from the second table.

Query: Retrieve all possible combinations of customers and menu items.

```
SELECT c.Cust_Name, m.Item_Name, m.Price
FROM Customer c
CROSS JOIN Menu_Item m;
```

Expected Output:

Cust_Name	Item_Name	Price
Alice	Pizza	450
Alice	Burger	270
Alice	Sushi	720
Alice	Pasta	360
Alice	Noodles	315
Bob	Pizza	450
Bob	Burger	270
Bob	Sushi	720
Bob	Pasta	360
Bob	Noodles	315

SELF JOIN

A **SELF JOIN** joins a table with itself. It is useful for hierarchical or comparison data.

Query: Retrieve all menu items that belong to the same restaurant as another item.

```sql
SELECT m1.Item_Name AS Item1, m2.Item_Name AS Item2, r.Rest_Name
FROM Menu_Item m1
JOIN Menu_Item m2 ON m1.Rest_ID = m2.Rest_ID AND m1.Item_ID != m2.Item_ID
JOIN Restaurant r ON m1.Rest_ID = r.Rest_ID;
```

Expected Output:

Item1	Item2	Rest_Name
Pizza	Burger	Food Paradise
Pizza	Pasta	Food Paradise
Burger	Pizza	Food Paradise
Burger	Pasta	Food Paradise
Pasta	Pizza	Food Paradise
Pasta	Burger	Food Paradise

2. Equivalent Queries

Query 1: Retrieve all customers who placed orders using a join (equivalent to a subquery).

Using Join:

```sql
SELECT DISTINCT c.Cust_Name
FROM Customer c
JOIN OrderTable o ON c.Cust_ID = o.Cust_ID;
```

Equivalent Subquery:

```sql
SELECT Cust_Name
FROM Customer
WHERE Cust_ID IN (SELECT Cust_ID FROM OrderTable);
```

Query 2: Retrieve the restaurant offering the most expensive menu item.

Using Join:

```sql
SELECT r.Rest_Name, MAX(m.Price) AS Max_Price
FROM Menu_Item m
JOIN Restaurant r ON m.Rest_ID = r.Rest_ID
GROUP BY r.Rest_Name
HAVING MAX(m.Price) = (SELECT MAX(Price) FROM Menu_Item);
```

Equivalent Subquery:

```
SELECT Rest_Name
FROM Restaurant
WHERE Rest_ID = (
    SELECT Rest_ID
    FROM Menu_Item
    WHERE Price = (SELECT MAX(Price) FROM Menu_Item)
);
```

3. Recursive Queries

Oracle SQL supports recursion using the **WITH** clause for hierarchical data.

Query 1: Generate a recursive query to find all ancestors of a given category in a hypothetical "Menu Category" table.

Assume we have a table:

```
CREATE TABLE Menu_Category (
    Cat_ID INT PRIMARY KEY,
    Cat_Name VARCHAR(50),
    Parent_Cat_ID INT
);
```

Sample Data:

INSERT INTO Menu_Category (Cat_ID, Cat_Name, Parent_Cat_ID) VALUES (1, 'Food', NULL);

INSERT INTO Menu_Category (Cat_ID, Cat_Name, Parent_Cat_ID) VALUES (2, 'Italian', 1);

INSERT INTO Menu_Category (Cat_ID, Cat_Name, Parent_Cat_ID) VALUES (3, 'Chinese', 1);

INSERT INTO Menu_Category (Cat_ID, Cat_Name, Parent_Cat_ID) VALUES (4, 'Pizza', 2);

INSERT INTO Menu_Category (Cat_ID, Cat_Name, Parent_Cat_ID) VALUES (5, 'Pasta', 2);

Recursive Query:

```sql
WITH CategoryHierarchy AS (
    SELECT Cat_ID, Cat_Name, Parent_Cat_ID
    FROM Menu_Category
    WHERE Cat_Name = 'Pizza'
    UNION ALL
    SELECT mc.Cat_ID, mc.Cat_Name, mc.Parent_Cat_ID
    FROM Menu_Category mc
    JOIN CategoryHierarchy ch ON mc.Cat_ID = ch.Parent_Cat_ID
)
SELECT * FROM CategoryHierarchy;
```

Expected Output for 'Pizza':

Cat_ID	Cat_Name	Parent_Cat_ID
4	Pizza	2
2	Italian	1
1	Food	NULL

4. Optimizing Complex Queries

Query 1: Find customers who placed orders totaling more than 1000 across all their orders.

```sql
SELECT c.Cust_Name, SUM(o.Order_Total) AS Total_Spent
FROM Customer c
JOIN OrderTable o ON c.Cust_ID = o.Cust_ID
GROUP BY c.Cust_Name
HAVING SUM(o.Order_Total) > 1000;
```

Query 2: Retrieve all restaurants and their total number of menu items using a join.

```sql
SELECT r.Rest_Name, COUNT(m.Item_ID) AS Total_Items
FROM Restaurant r
LEFT JOIN Menu_Item m ON r.Rest_ID = m.Rest_ID
GROUP BY r.Rest_Name;
```