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, for the online food ordering system.

1. Join Queries

Query1 : 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;
```

Query2 : 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 = r.Rest_ID;
```

Query3 : Retrieve all orders and their delivery status

```
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;
```

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

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

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

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

Writin

Case s

Objec

1.

Q

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

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

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

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

Inner Join

An Inner Join retrieves records that have matching values in both tables.

Query : Retrieves all orders along with their customer names.

```
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;
```

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.

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

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.

Item 1	Item2	Rest_Name
Pizza	Burger	FoodParadise
Pizza	Pasta	FoodParadise
Burger	Pizza	FoodParadise
Burger	Pasta	Food Paradise
Pasta	Pizza	Food Paradise
Pasta	Burger	Food Paradise

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

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;
```

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 mat
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;
```

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.Items_Name m.Price
FROM Customer c
CROSS JOIN Menu_Item m;
```

SELF JOIN

A self join joins a table with itself. It is used for hierarchial or comparison data.

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

```
SELECT m1.Item_Name AS Item1, m2.Item_Name AS Item2, r.Rest_
       Name.
FROM Menu_Item m1.
JOIN Menu_Item AS Item1, m2.Item_Name AS Item2, r.Rest_Name
JOIN Restaurant r ON m1.Rest_ID = r.Rest_ID;
```

2. Equivalent Queries:

Query1: Retrieve all customers who placed orders using a join.

Using Join:

```
SELECT DISTINCT c.cust_name
FROM customer c
JOIN OrderTable o ON c.cust_ID = o.Cust_ID;
```

Equivalent Subquery:

```
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:

```
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)
);
```

Recursive Queries:

Oracle SQL supports recursion using the WITH clause for hierarchial data.

Query 1: Generate a recursive query to find all ancestors of a given category in a hypothetical "Menu Category" 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);
```

4. Optimizing Complex Queries

Query1 : Find customers who placed orders totalling more than 1000 across all their orders.

```
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 ;
```