SQL INTERVIEW PREPARATION PART 3.2

JOIN QUESTIONS CONTINUED

16. Compare Employee Salaries Across Departments

Scenario:
You have an Employees table with EmployeeID, Name, DepartmentID, and Salary.
Question: Write a query to compare salaries of employees in the same department and list those earning more than the department average.
Solution:

1. Using Subquery approach

```
SELECT e.EmployeeID,
    e.Name,
    e.DepartmentID,
    e.Salary
FROM Employees e
WHERE e.Salary > (
  SELECT AVG(Salary)
  FROM Employees
  WHERE DepartmentID = e.DepartmentID
);
```

2. Using CTE

```
WITH AverageSalaryCTE AS (
  SELECT
    DepartmentID,
    AVG(Salary) AS Avg_Salary
  FROM Employees
  GROUP BY DepartmentID
)
SELECT
  e.EmployeeID,
  e.Name,
  e.DepartmentID,
  e.Salary
FROM Employees e
INNER JOIN AverageSalaryCTE a
ON e.DepartmentID = a.DepartmentID
WHERE e.Salary > a.Avg_Salary;
```

17. List Customers Who Bought Products from Multiple Categories

Scenario:

- Orders table includes customer IDs and product IDs.
- Products table includes product IDs and category IDs.

Question: Write a query to find customers who have purchased products from more than one category.

Solution:

```
SELECT o.customerID
FROM Orders o
INNER JOIN Products p
ON o.productID = p.productID
GROUP BY o.customerID
HAVING COUNT(DISTINCT p.categoryID) > 1;
```

18. Match Employees with Department Heads

Scenario:

- Employees table includes employee details and DepartmentID.
- Departments table includes department details and ManagerID (which corresponds to an employee).

Question: Write a query to list each employee along with their department head's name.

Solution:

```
SELECT e1.name AS Employee_Name,
    d.DepartmentID,
    e2.name AS Manager_Name
FROM Employees e1
INNER JOIN Departments d
  ON e1.DepartmentID = d.DepartmentID
INNER JOIN Employees e2
  ON d.ManagerID = e2.EmployeeID;
```

19. Identify Overlapping Time Periods

Scenario:

- Bookings table includes booking details with BookingID, StartDate, and EndDate.

Question: Write a query to find bookings that overlap with one another**.**

Solution:

```
SELECT b1.BookingID AS Booking1,
    b2.BookingID AS Booking2
FROM Bookings b1
SELF JOIN Bookings b2
ON b1.BookingID <> b2.BookingID
  AND b1.StartDate < b2.EndDate
  AND b1.EndDate > b2.StartDate;
```

20. Merging Financial Transactions

Scenario:

- BankTransactions table includes transaction IDs and amounts.
- CreditCardTransactions table includes transaction IDs and amounts.

Question: Write a query to combine all transactions from both tables, avoiding duplicates.
Solution:

```
SELECT transactionID, amount
FROM BankTransactions
UNION
SELECT transactionID, amount
FROM CreditCardTransactions;
```

Alternative using FULL OUTER JOIN and COALESCE:

```
SELECT
  COALESCE(b.transactionID, c.transactionID) AS transactionID,
  COALESCE(b.amount, c.amount) AS amount
FROM BankTransactions b
FULL OUTER JOIN CreditCardTransactions c
ON b.transactionID = c.transactionID;
```

21. List Common Products in Two Databases

Scenario:

- OnlineStoreProducts and RetailStoreProducts tables both include product IDs.

Question: Write a query to find products available in both stores.
Solution:

```
SELECT productID
FROM OnlineStoreProducts
INTERSECT
SELECT productID
FROM RetailStoreProducts;
```

Alternative Using Joins:
If the database system does not support INTERSECT, the same result can be achieved with an INNER JOIN:

```
SELECT o.productID
FROM OnlineStoreProducts o
INNER JOIN RetailStoreProducts r
ON o.productID = r.productID;
```

22. Monthly Sales Analysis

Scenario:

- Sales table includes sales data with SaleID, Date, and Amount.

Question: Write a query to calculate the total sales amount for each month, showing months even if no sales occurred.

Solution:

```
WITH MonthDimension AS (
   SELECT 1 AS Month, 'January' AS MonthName
   UNION ALL
   SELECT 2, 'February'
   UNION ALL
   SELECT 3, 'March'
   UNION ALL
   SELECT 4, 'April'
   UNION ALL
   SELECT 5, 'May'
   UNION ALL
   SELECT 6, 'June'
   UNION ALL
   SELECT 7, 'July'
   UNION ALL
   SELECT 8, 'August'
   UNION ALL
   SELECT 9, 'September'
   UNION ALL
   SELECT 10, 'October'
   UNION ALL
   SELECT 11, 'November'
   UNION ALL
   SELECT 12, 'December'
)
SELECT
   m.MonthName,
   COALESCE(SUM(s.Amount), 0) AS TotalSales
FROM
   MonthDimension m
LEFT JOIN
   Sales s ON MONTH(s.Date) = m.Month
GROUP BY
   m.MonthName
ORDER BY
   m.Month;
```

23. Find Products Never Purchased

Scenario:

- Products table includes product details.
- Orders table includes product IDs of purchased items.

Question: Write a query to find products that have never been purchased.

Solution:

```
SELECT p.productID
FROM Products p
LEFT JOIN Orders o
ON p.productID = o.productID
WHERE o.productID IS NULL;
```

24. Joining Hierarchical Data

Scenario: You have a Company table with EmployeeID and ManagerID.
Question: Write a query to list all employees along with their manager's name.
Solution:

```
SELECT c1.EmpName as EmployeeName,
        c2.EmpName as ManagerName
FROM Company c1
LEFT JOIN Company c2
ON c1.ManagerID = c2.EmployeeID;
```

25. Invoice Payment Status

Scenario:

- Invoices table includes invoice IDs and amounts.
- Payments table includes invoice IDs and payment amounts.

Question: Write a query to find invoices that are partially paid, fully paid, or unpaid.
Solution:

```
SELECT i.invoiceID,
    CASE
      WHEN p.payment_amt = i.invoice_amt THEN 'fully paid'
      WHEN p.payment_amt IS NULL OR p.payment_amt = 0 THEN 'unpaid'
      WHEN p.payment_amt > 0 AND p.payment_amt < i.invoice_amt THEN 'partially paid'
      ELSE 'unknown'
    END AS payment_status
FROM Invoices i
LEFT JOIN Payments p
ON i.invoiceID = p.invoiceID;
```

26. Revenue by Customer and Year

Scenario:

- Orders table includes customer IDs, order amounts, and order dates.

Question: Write a query to calculate the total revenue generated by each customer for each year.
Solution:

```
SELECT
  customerID,
```

```
    YEAR(order_date) AS order_year,
    SUM(order_amt) AS total_revenue
FROM Orders
GROUP BY
    customerID,
    YEAR(order_date);
```

27. Duplicate Customer Records

Scenario:

- Customers table includes customer IDs and names.

Question: Write a query to identify duplicate customer entries (same name or email).

Solution:

```
SELECT
    c1.customerID AS DuplicateCustomerID,
    c1.name AS CustomerName,
    c1.email AS CustomerEmail
FROM Customers c1
INNER JOIN Customers c2
ON c1.customerID <> c2.customerID
WHERE c1.name = c2.name OR c1.email = c2.email;
```

28. Active Users by Region

Scenario:

- Users table includes user IDs, regions, and active statuses.

Question: Write a query to find the number of active users in each region.

Solution:

```
SELECT region, COUNT(userID) AS active_user_count
FROM Users
WHERE active_status = TRUE
GROUP BY region;
```

29. Orders Without Valid Products

Scenario:

- Orders table includes product IDs and order details.
- Products table includes product IDs and details.

Question: Write a query to find orders placed for products that are not in the product catalog.

Solution:

```
SELECT o.productID
FROM Orders o
LEFT JOIN Products p
ON o.productID = p.productID
```

WHERE p.productID IS NULL;

30. Combine Employee and Contractor Data

Scenario:

- Employees table includes employee IDs and salaries.
- Contractors table includes contractor IDs and hourly rates.

Question: Write a query to combine data from both tables to create a unified resource list.
Solution:

```
SELECT employeeID AS ResourceID, salaries AS Compensation
FROM Employees
UNION
SELECT contractorID AS ResourceID, hourly_rates AS Compensation
FROM Contractors;
```

Alternatively, using FULL OUTER JOIN:

```
SELECT e.employeeID AS ResourceID, e.salaries AS EmployeeCompensation,
    c.contractorID AS ContractorID, c.hourly_rates AS ContractorCompensation
FROM Employees e
FULL OUTER JOIN Contractors c
ON e.employeeID = c.contractorID;
```

When to Use Which Approach:

1. **UNION**:
    - Use this when you need a single unified list of resources (employees and contractors) with similar attributes.
2. **FULL OUTER JOIN**:
    - Use this when you need to display data from both tables side by side, even if they don't overlap.

Note: Both approaches solve the problem based on how the data is to be presented.