1. Ranking Rows Using ROW_NUMBER()

Problem:
Assign a unique sequential number to rows within each department ordered by salary descending.
Solution:

```
SELECT
    EmployeeID,
    Department,
    Salary,
    ROW_NUMBER() OVER (PARTITION BY Department ORDER BY Salary
DESC) AS RankInDept
FROM Employees;
```

2. Assigning RANK() and DENSE_RANK()

Problem:
Rank employees by salary within departments, accounting for ties.
Solution:

```
SELECT
    EmployeeID,
    Department,
    Salary,
    RANK() OVER (PARTITION BY Department ORDER BY Salary DESC)
AS Rank,
    DENSE_RANK() OVER (PARTITION BY Department ORDER BY Salary
DESC) AS DenseRank
FROM Employees;
```

3. Calculate Running Total of Sales

Problem:
Calculate cumulative sales over time ordered by sale date.

Solution:

```
SELECT
    SalesDate,
    SalesAmount,
    SUM(SalesAmount) OVER (ORDER BY SalesDate) AS RunningTotal
FROM Sales;
```

4. Calculate Moving Average

Problem:
Compute 3-day moving average of sales amount.
Solution:

```
SELECT
    SalesDate,
    SalesAmount,
    AVG(SalesAmount) OVER (ORDER BY SalesDate ROWS BETWEEN 2
PRECEDING AND CURRENT ROW) AS MovingAvg3Days
FROM Sales;
```

5. Use LEAD() and LAG() to Compare Current and Previous Rows

Problem:
Show the difference between current day's sales and previous day's sales.
Solution:

```
SELECT
    SalesDate,
    SalesAmount,
    LAG(SalesAmount) OVER (ORDER BY SalesDate) AS
PreviousDaySales,
    SalesAmount - LAG(SalesAmount) OVER (ORDER BY SalesDate) AS
SalesDifference
```

```
FROM Sales;
```

6. Find Duplicates Using Window Functions

Problem:
Find duplicate employees by name with counts.
Solution:

```
SELECT
    EmployeeID,
    FirstName,
    LastName,
    COUNT(*) OVER (PARTITION BY FirstName, LastName) AS
DuplicateCount
FROM Employees
WHERE DuplicateCount > 1;
```

7. Use NTILE() to Distribute Rows into Buckets

Problem:
Divide employees into 4 quartiles based on their salaries.
Solution:

```
SELECT
    EmployeeID,
    Salary,
    NTILE(4) OVER (ORDER BY Salary DESC) AS SalaryQuartile
FROM Employees;
```

8. Conditional Aggregation with CASE and Window Functions

Problem:
Count how many employees in each department earn above 50,000.
Solution:

```
SELECT
    EmployeeID,
    Department,
    Salary,
    SUM(CASE WHEN Salary > 50000 THEN 1 ELSE 0 END) OVER
(PARTITION BY Department) AS HighEarnersInDept
FROM Employees;
```

9. Calculate Rank with Ties, Then Filter Top N per Group

Problem:
Get the top 3 highest paid employees per department.
Solution:
sql

```
WITH RankedEmployees AS (
    SELECT
        EmployeeID, Department, Salary,
        RANK() OVER (PARTITION BY Department ORDER BY Salary
DESC) AS SalaryRank
    FROM Employees
)
SELECT * FROM RankedEmployees
WHERE SalaryRank <= 3;
```

10. Use Window Functions in Pagination

Problem:
Paginate employee records with row numbers and return rows 11 to 20 ordered by EmployeeID.
Solution:

```
WITH NumberedEmployees AS (
    SELECT
        EmployeeID,
        FirstName,
```

```
        LastName,
        ROW_NUMBER() OVER (ORDER BY EmployeeID) AS RowNum
    FROM Employees
)
SELECT * FROM NumberedEmployees WHERE RowNum BETWEEN 11 AND 20;
```