Window Function Tasks

```sql
-- find total sales for all orders in every row
SELECT
    OrderID,
    Sales,
    SUM(Sales) OVER() AS TotalSales
FROM Sales.Orders;


-- find the average sales for all orders beside order id, and sales
SELECT
      OrderID,
      Sales,
      AVG(Sales) OVER() AS AverageSales
FROM Sales.Orders;


-- show order id, product id, sales, and total sales for each product
SELECT
      OrderID,
      ProductID,
      Sales,
      SUM(Sales) OVER(PARTITION BY ProductID) AS TotalByProduct
FROM Sales.Orders;


-- show order id, customer id, sales, total sales per customer
SELECT
      OrderID,
      CustomerID,
      Sales,
      SUM(Sales) OVER(PARTITION BY CustomerID) AS TotalByCustomer
FROM Sales.Orders;


-- count orders per customers
SELECT
      OrderID,
      CustomerID,
      COUNT(*) OVER(PARTITION BY CustomerID) AS OrdersCount
FROM Sales.Orders;



-- show product id, order id, number of orders per product
SELECT
      ProductID,
      OrderID,
      COUNT(*) OVER(PARTITION BY ProductID) AS OrdersPerProduct
FROM Sales.Orders;

-----------------------------------------------------------------------
```

```sql
-- sorts orders by date
-- show order id, order date, sales, running total of sales over time
-- So total keeps increasing.
SELECT
       OrderID,
       OrderDate,
       Sales,
       SUM(Sales) OVER(ORDER BY OrderDate) AS RunningTotal
FROM Sales.Orders;


-- show order id, creation time, sales, running total orders by creation time
SELECT
       OrderID,
       CreationTime,
       Sales,
       SUM(Sales) OVER(ORDER BY CreationTime) AS RunningTotal
FROM Sales.Orders;


-- show product id, order id, sales, running total per product
-- Split rows by ProductID
-- Sort each group by date
-- Calculate running total inside each group
SELECT
       ProductID,
       OrderDate,
       Sales,
       SUM(Sales) OVER(
       PARTITION BY ProductID
       ORDER BY OrderDate
       ) AS RunningTotalPerProduct
FROM Sales.Orders;


-- show customer id, order date, sales, running total per customer
SELECT
       CustomerID,
       OrderDate,
       Sales,
       SUM(Sales) OVER(
       PARTITION BY CustomerID
       ORDER BY OrderDate
       ) AS RunningTotalPerCustomer
FROM Sales.Orders;


/*
       -----------------------
       -- RANKING Functions --
       -----------------------
       | Function       | Meaning                                            |
       | -------------- | -------------------------------------------------- |
       | `ROW_NUMBER()` | Gives a unique number to each row (no ties)        |
       | `RANK()`       | Gives rank, same rank for ties, skips next ranks   |
       | `DENSE_RANK()` | Gives rank, same rank for ties, no skipped numbers |
       | `NTILE(n)`     | Splits rows into n groups, assigns group numbers   |
        ------------
        -- syntax --
        ------------
        <function>() OVER(
    [PARTITION BY column]
    ORDER BY column
)
*/


-- Assign a number to each order by OrderDate
-- No ties, every row is unique
SELECT
        OrderID,
        OrderDate,
        Sales,
        ROW_NUMBER() OVER(ORDER BY OrderDate) AS RowNumber
FROM Sales.Orders;


-- Rank products by Sales
-- Biggest sale gets rank 1
-- Tied sales get the same rank
-- Next rank skips numbers if tie occurs
SELECT
        ProductID,
        Sales,
        RANK() OVER(ORDER BY Sales DESC) AS SalesRank
FROM Sales.Orders;


-- Dense rank products by Sales
-- does NOT skip ranks
-- Useful for continuous ranking
SELECT
    ProductID,
    Sales,
    DENSE_RANK() OVER(ORDER BY Sales DESC) AS SalesDenseRank
FROM Sales.Orders;


-- Split customers into 4 groups by Score
-- Highest score → group 1
-- Lowest score → group 4
-- Divides rows evenly into 4 groups
SELECT
    CustomerID,
    Score,
    NTILE(4) OVER(ORDER BY Score DESC) AS ScoreQuartile
FROM Sales.Customers;
```

```sql
-- Running Total per Customer (with ORDER BY)
-- Splits rows by customer
-- Sorts by date
-- Adds sales cumulatively
SELECT
    CustomerID,
    OrderDate,
    Sales,
    SUM(Sales) OVER(
        PARTITION BY CustomerID
        ORDER BY OrderDate
    ) AS RunningTotalPerCustomer
FROM Sales.Orders;


-- Previous Order Sales using LAG()
-- Use: Compare current order vs previous for same customer
SELECT
    OrderID,
    CustomerID,
    Sales,
    LAG(Sales) OVER(
        PARTITION BY CustomerID
        ORDER BY OrderDate
    ) AS PreviousOrderSales
FROM Sales.Orders;


-- Next Order Sales using LEAD()
-- Use: Forecast or analyze trends
SELECT
    OrderID,
    CustomerID,
    Sales,
    LEAD(Sales) OVER(
        PARTITION BY CustomerID
        ORDER BY OrderDate
    ) AS NextOrderSales
FROM Sales.Orders;


-- Quartiles / NTILE for Customers by Score
-- Use: Divide customers into top/bottom groups
SELECT
    CustomerID,
    Score,
    NTILE(4) OVER(
        ORDER BY Score DESC
    ) AS ScoreQuartile
FROM Sales.Customers;
```