```
/* ROW_NUMBER()
Problem 1: List the top 5 highest salaries in each department. */
WITH RankedSalaries AS (
SELECT
EmployeeID,
Department,
Salary,
ROW_NUMBER() OVER (PARTITION BY Department ORDER BY Salary DESC) AS SalaryRank
FROM
Employees
)
SELECT
EmployeeID,
Department,
Salary
FROM
RankedSalaries
WHERE
SalaryRank <= 5;

/* Problem 2: Rank students by their grades within each class. */
WITH RankedGrades AS (
SELECT
StudentID,
ClassID,
Grade,
ROW_NUMBER() OVER (PARTITION BY ClassID ORDER BY Grade DESC) AS GradeRank
FROM
Grades
)
SELECT
```

```
StudentID,
ClassID,
Grade
FROM
RankedGrades
WHERE
GradeRank <= 10;

/* Problem 3: Number all orders within each month, ordered by order date. */
WITH OrderRanks AS (
SELECT
OrderID,
OrderDate,
MONTH(OrderDate) AS OrderMonth,
ROW_NUMBER() OVER (PARTITION BY MONTH(OrderDate) ORDER BY OrderDate) AS OrderRank
FROM
Orders
)
SELECT
OrderID,
OrderDate,
OrderMonth,
OrderRank
FROM
OrderRanks;

/* Problem 4: List the top 3 products by sales within each region. */
WITH RankedProducts AS (
SELECT
ProductID,
Region,
```

```
SUM(SalesAmount) AS TotalSales,
ROW_NUMBER() OVER (PARTITION BY Region ORDER BY SUM(SalesAmount) DESC) AS SalesRank
FROM
Sales
GROUP BY
ProductID, Region
)
SELECT
ProductID,
Region,
TotalSales
FROM
RankedProducts
WHERE
SalesRank <= 3;

/* Problem 5: Assign a unique number to each customer based on their total purchase amount in descending order. */
WITH CustomerRanks AS (
SELECT
CustomerID,
SUM(PurchaseAmount) AS TotalPurchase,
ROW_NUMBER() OVER (ORDER BY SUM(PurchaseAmount) DESC) AS PurchaseRank
FROM
Purchases
GROUP BY
CustomerID
)
SELECT
CustomerID,
TotalPurchase,
```

```
PurchaseRank
FROM
CustomerRanks;

/* Problem 6: List the 5 most recent hires in each department. */
WITH RecentHires AS (
SELECT
EmployeeID,
Department,
HireDate,
ROW_NUMBER() OVER (PARTITION BY Department ORDER BY HireDate DESC) AS HireRank
FROM
Employees
)
SELECT
EmployeeID,
Department,
HireDate
FROM
RecentHires
WHERE
HireRank <= 5;

/* Problem 7: Rank the top 10 sales transactions within each quarter. */
WITH RankedTransactions AS (
SELECT
TransactionID,
TransactionDate,
SalesAmount,
ROW_NUMBER() OVER (PARTITION BY DATEPART(QUARTER, TransactionDate) ORDER BY
SalesAmount DESC) AS TransactionRank
```

```
FROM
Transactions
)
SELECT
TransactionID,
TransactionDate,
SalesAmount
FROM
RankedTransactions
WHERE
TransactionRank <= 10;

/* Problem 8: Number the reviews within each product, ordered by review date. */
WITH ReviewRanks AS (
SELECT
ReviewID,
ProductID,
ReviewDate,
ROW_NUMBER() OVER (PARTITION BY ProductID ORDER BY ReviewDate) AS ReviewRank
FROM
Reviews
)
SELECT
ReviewID,
ProductID,
ReviewDate,
ReviewRank
FROM
ReviewRanks;

/* Problem 9: List the 5 most recent orders for each customer. */
```

```
WITH RecentOrders AS (
SELECT
OrderID,
CustomerID,
OrderDate,
ROW_NUMBER() OVER (PARTITION BY CustomerID ORDER BY OrderDate DESC) AS OrderRank
FROM
Orders
)
SELECT
OrderID,
CustomerID,
OrderDate
FROM
RecentOrders
WHERE
OrderRank <= 5;

/* Problem 10: Number all products by their launch date within each category. */
WITH ProductRanks AS (
SELECT
ProductID,
CategoryID,
LaunchDate,
ROW_NUMBER() OVER (PARTITION BY CategoryID ORDER BY LaunchDate) AS LaunchRank
FROM
Products
)
SELECT
ProductID,
CategoryID,
```

```
LaunchDate,
LaunchRank
FROM
ProductRanks;

/* RANK()
Problem 1: Rank employees by their bonus amount within each department, handling ties. */
SELECT
EmployeeID,
Department,
BonusAmount,
RANK() OVER (PARTITION BY Department ORDER BY BonusAmount DESC) AS BonusRank
FROM
Employees;

/* Problem 2: Rank the top 10 performing students in each subject based on their grades. */
WITH RankedStudents AS (
SELECT
StudentID,
SubjectID,
Grade,
RANK() OVER (PARTITION BY SubjectID ORDER BY Grade DESC) AS GradeRank
FROM
StudentGrades
)
SELECT
StudentID,
SubjectID,
Grade
FROM
RankedStudents
```

```
WHERE
GradeRank <= 10;

/* Problem 3: Rank the most popular items in each store based on the number of sales. */
WITH ItemRanks AS (
SELECT
ItemID,
StoreID,
COUNT(*) AS SalesCount,
RANK() OVER (PARTITION BY StoreID ORDER BY COUNT(*) DESC) AS SalesRank
FROM
Sales
GROUP BY
ItemID, StoreID
)
SELECT
ItemID,
StoreID,
SalesCount
FROM
ItemRanks
WHERE
SalesRank <= 5;

/* Problem 4: Rank authors by their total book sales, considering ties. */
WITH AuthorSales AS (
SELECT
AuthorID,
SUM(SalesAmount) AS TotalSales,
RANK() OVER (ORDER BY SUM(SalesAmount) DESC) AS SalesRank
FROM
```

```
Books
GROUP BY
AuthorID
)
SELECT
AuthorID,
TotalSales,
SalesRank
FROM
AuthorSales;

/* Problem 5: Rank customers based on their total number of orders, handling ties. */
WITH CustomerOrders AS (
SELECT
CustomerID,
COUNT(OrderID) AS OrderCount,
RANK() OVER (ORDER BY COUNT(OrderID) DESC) AS OrderRank
FROM
Orders
GROUP BY
CustomerID
)
SELECT
CustomerID,
OrderCount,
OrderRank
FROM
CustomerOrders;

/* Problem 6: Rank students by their average test scores in each grade level. */
WITH StudentAverages AS (
```

```
SELECT
StudentID,
GradeLevel,
AVG(TestScore) AS AverageScore,
RANK() OVER (PARTITION BY GradeLevel ORDER BY AVG(TestScore) DESC) AS ScoreRank
FROM
TestScores
GROUP BY
StudentID, GradeLevel
)
SELECT
StudentID,
GradeLevel,
AverageScore
FROM
StudentAverages
WHERE
ScoreRank <= 10;

/* Problem 7: Rank the top 5 highest paid employees in each city. */
WITH CityEmployeeRanks AS (
SELECT
EmployeeID,
City,
Salary,
RANK() OVER (PARTITION BY City ORDER BY Salary DESC) AS SalaryRank
FROM
Employees
)
SELECT
EmployeeID,
```

```
City,
Salary
FROM
CityEmployeeRanks
WHERE
SalaryRank <= 5;

/* Problem 8: Rank products by their total sales in each category, handling ties. */
WITH ProductSales AS (
SELECT
ProductID,
CategoryID,
SUM(SalesAmount) AS TotalSales,
RANK() OVER (PARTITION BY CategoryID ORDER BY SUM(SalesAmount) DESC) AS SalesRank
FROM
Sales
GROUP BY
ProductID, CategoryID
)
SELECT
ProductID,
CategoryID,
TotalSales,
SalesRank
FROM
ProductSales;

/* Problem 9: Rank the top 10 most reviewed products in each store. */
WITH ProductReviewRanks AS (
SELECT
ProductID,
```

```
StoreID,
COUNT(*) AS ReviewCount,
RANK() OVER (PARTITION BY StoreID ORDER BY COUNT(*) DESC) AS ReviewRank
FROM
Reviews
GROUP BY
ProductID, StoreID
)
SELECT
ProductID,
StoreID,
ReviewCount
FROM
ProductReviewRanks
WHERE
ReviewRank <= 10;

/* Problem 10: Rank employees by their annual performance score in each department. */
WITH EmployeeScores AS (
SELECT
EmployeeID,
Department,
SUM(PerformanceScore) AS TotalScore,
RANK() OVER (PARTITION BY Department ORDER BY SUM(PerformanceScore) DESC) AS ScoreRank
FROM
Performance
GROUP BY
EmployeeID, Department
)
SELECT
EmployeeID,
```

```
Department,
TotalScore
FROM
EmployeeScores
WHERE
ScoreRank <= 5;

/* DENSE_RANK()
Problem 1: Rank products by their total sales in each category, ensuring no gaps in rank for ties. */
WITH ProductSales AS (
SELECT
ProductID,
CategoryID,
SUM(SalesAmount) AS TotalSales,
DENSE_RANK() OVER (PARTITION BY CategoryID ORDER BY SUM(SalesAmount) DESC) AS SalesRank
FROM
Sales
GROUP BY
ProductID, CategoryID
)
SELECT
ProductID,
CategoryID,
TotalSales,
SalesRank
FROM
ProductSales;

/* Problem 2: Rank employees by their tenure in the company, ensuring no gaps in ranking. */
WITH EmployeeTenure AS (
SELECT
```

```
EmployeeID,
HireDate,
DENSE_RANK() OVER (ORDER BY DATEDIFF(YEAR, HireDate, GETDATE()) DESC) AS TenureRank
FROM
Employees
)
SELECT
EmployeeID,
HireDate,
TenureRank
FROM
EmployeeTenure;

/* Problem 3: Rank the top 10 best-selling products by sales amount in each store, with no gaps for ties. */
WITH StoreProductSales AS (
SELECT
ProductID,
StoreID,
SUM(SalesAmount) AS TotalSales,
DENSE_RANK() OVER (PARTITION BY StoreID ORDER BY SUM(SalesAmount) DESC) AS SalesRank
FROM
Sales
GROUP BY
ProductID, StoreID
)
SELECT
ProductID,
StoreID,
TotalSales
FROM
```

```
StoreProductSales
WHERE
SalesRank <= 10;

/* Problem 4: Rank authors by their total number of published books, ensuring no gaps in ranking. */
WITH AuthorPublication AS (
SELECT
AuthorID,
COUNT(BookID) AS BookCount,
DENSE_RANK() OVER (ORDER BY COUNT(BookID) DESC) AS PublicationRank
FROM
Books
GROUP BY
AuthorID
)
SELECT
AuthorID,
BookCount,
PublicationRank
FROM
AuthorPublication;

/* Problem 5: Rank the top 5 sales representatives by their total sales in each region, without gaps.
*/
WITH SalesRepRank AS (
SELECT
SalesRepID,
Region,
SUM(SalesAmount) AS TotalSales,
DENSE_RANK() OVER (PARTITION BY Region ORDER BY SUM(SalesAmount) DESC) AS SalesRank
FROM
```

```
Sales
GROUP BY
SalesRepID, Region
)
SELECT
SalesRepID,
Region,
TotalSales
FROM
SalesRepRank
WHERE
SalesRank <= 5;


/* Problem 6: Rank students by their average test scores within each grade level, ensuring no gaps.
*/
WITH StudentAvgScores AS (
SELECT
StudentID,
GradeLevel,
AVG(TestScore) AS AverageScore,
DENSE_RANK() OVER (PARTITION BY GradeLevel ORDER BY AVG(TestScore) DESC) AS ScoreRank
FROM
TestScores
GROUP BY
StudentID, GradeLevel
)
SELECT
StudentID,
GradeLevel,
AverageScore
FROM
```

```
StudentAvgScores

WHERE

ScoreRank <= 10;


/* Problem 7: Rank the top 3 most reviewed products in each category, with no gaps in rank for ties.
*/

WITH ProductReviewRanks AS (

SELECT

ProductID,

CategoryID,

COUNT(*) AS ReviewCount,

DENSE_RANK() OVER (PARTITION BY CategoryID ORDER BY COUNT(*) DESC) AS ReviewRank

FROM

Reviews

GROUP BY

ProductID, CategoryID

)

SELECT

ProductID,

CategoryID,

ReviewCount

FROM

ProductReviewRanks

WHERE

ReviewRank <= 3;


/* Problem 8: Rank the highest spenders in each customer segment by their total spending, with no
gaps. */

WITH SegmentSpending AS (

SELECT

CustomerSegment,

CustomerID,
```

```
SUM(SpendingAmount) AS TotalSpending,
DENSE_RANK() OVER (PARTITION BY CustomerSegment ORDER BY SUM(SpendingAmount) DESC) AS SpendingRank
FROM
CustomerSpending
GROUP BY
CustomerSegment, CustomerID
)
SELECT
  CustomerID,
  CustomerSegment,
  TotalSpending
FROM
  SegmentSpending
WHERE
  SpendingRank <= 5;

/* Problem 9: Rank departments by their average employee age, ensuring no gaps in the ranks. */
WITH DepartmentAge AS (
  SELECT
    DepartmentID,
    AVG(DATEDIFF(YEAR, BirthDate, GETDATE())) AS AvgAge,
    DENSE_RANK() OVER (ORDER BY AVG(DATEDIFF(YEAR, BirthDate, GETDATE())) DESC) AS AgeRank
  FROM
    Employees
  GROUP BY
    DepartmentID
)
SELECT
  DepartmentID,
  AvgAge,
```

```
    AgeRank
FROM
    DepartmentAge;

/* Problem 10: Rank the top 5 sales transactions in each store by transaction amount, with no gaps.
*/
WITH StoreTransactionRanks AS (
    SELECT
        TransactionID,
        StoreID,
        TransactionAmount,
        DENSE_RANK() OVER (PARTITION BY StoreID ORDER BY TransactionAmount DESC) AS Transac-
tionRank
    FROM
        Transactions
)
SELECT
    TransactionID,
    StoreID,
    TransactionAmount
FROM
    StoreTransactionRanks
WHERE
    TransactionRank <= 5;

/* NTILE()
Problem 1: Divide employees into 4 quartiles based on their salary. */
SELECT
    EmployeeID,
    Salary,
    NTILE(4) OVER (ORDER BY Salary DESC) AS SalaryQuartile
FROM
```

```
    Employees;

/* Problem 2: Partition products into 10 equal-sized groups based on their sales amount. */
SELECT
    ProductID,
    SalesAmount,
    NTILE(10) OVER (ORDER BY SalesAmount DESC) AS SalesDecile
FROM
    Sales;

/* Problem 3: Divide students into 5 groups based on their average grades. */
WITH StudentAvgGrades AS (
    SELECT
        StudentID,
        AVG(Grade) AS AvgGrade
    FROM
        Grades
    GROUP BY
        StudentID
)
SELECT
    StudentID,
    AvgGrade,
    NTILE(5) OVER (ORDER BY AvgGrade DESC) AS GradeGroup
FROM
    StudentAvgGrades;

/* Problem 4: Partition transactions into 3 groups based on transaction amount. */
SELECT
    TransactionID,
    TransactionAmount,
```

```
    NTILE(3) OVER (ORDER BY TransactionAmount DESC) AS AmountGroup
FROM
    Transactions;

/* Problem 5: Divide customers into 4 equal groups based on their total spending. */
WITH CustomerSpending AS (
    SELECT
        CustomerID,
        SUM(SpendingAmount) AS TotalSpending
    FROM
        Purchases
    GROUP BY
        CustomerID
)
SELECT
    CustomerID,
    TotalSpending,
    NTILE(4) OVER (ORDER BY TotalSpending DESC) AS SpendingQuartile
FROM
    CustomerSpending;

/* Problem 6: Partition the top 20% of sales transactions into a group based on transaction amount.
*/
WITH TopSales AS (
    SELECT
        TransactionID,
        TransactionAmount,
        NTILE(5) OVER (ORDER BY TransactionAmount DESC) AS AmountGroup
    FROM
        Transactions
)
```

```
SELECT
    TransactionID,
    TransactionAmount
FROM
    TopSales
WHERE
    AmountGroup = 1;

/* Problem 7: Divide employees into 3 groups based on their tenure in the company. */
WITH EmployeeTenure AS (
    SELECT
        EmployeeID,
        DATEDIFF(YEAR, HireDate, GETDATE()) AS TenureYears
    FROM
        Employees
)
SELECT
    EmployeeID,
    TenureYears,
    NTILE(3) OVER (ORDER BY TenureYears DESC) AS TenureGroup
FROM
    EmployeeTenure;

/* Problem 8: Partition products into 5 groups based on their profitability. */
WITH ProductProfitability AS (
    SELECT
        ProductID,
        (SalesAmount - Cost) AS Profit
    FROM
        Sales
)
```

```
SELECT
    ProductID,
    Profit,
    NTILE(5) OVER (ORDER BY Profit DESC) AS ProfitGroup
FROM
    ProductProfitability;

/* Problem 9: Divide stores into 4 groups based on their total sales amount. */
WITH StoreSales AS (
    SELECT
        StoreID,
        SUM(SalesAmount) AS TotalSales
    FROM
        Sales
    GROUP BY
        StoreID
)
SELECT
    StoreID,
    TotalSales,
    NTILE(4) OVER (ORDER BY TotalSales DESC) AS SalesQuartile
FROM
    StoreSales;

/* Problem 10: Partition employees into 10 groups based on their performance score. */
WITH EmployeePerformance AS (
    SELECT
        EmployeeID,
        PerformanceScore
    FROM
        Performance
```

```
)
SELECT
    EmployeeID,
    PerformanceScore,
    NTILE(10) OVER (ORDER BY PerformanceScore DESC) AS PerformanceDecile
FROM
    EmployeePerformance;
```