Lecture Notes – NTILE()

NTILE() Window Function - Complete Guide

Section 1: Lecture Summary

The **NTILE()** function is a ranking window function that divides rows into a specified number of equally-sized groups or "tiles." Unlike ROW_NUMBER(), RANK(), and DENSE_RANK() which assign unlimited sequential ranks, NTILE() limits rankings to a fixed number you specify, making it ideal for segmenting data into quartiles, deciles, or any custom number of categories.

Section 2: Key Concepts and Explanations

What is NTILE()?

NTILE() distributes rows across N equally-sized buckets. If you have 15 products and use NTILE(4), they are divided into 4 groups (quartiles) with approximately equal row counts per group. The function assigns rank values from 1 to N.

Core Differences from Other Ranking Functions

Unlike RANK() and DENSE_RANK() which create unlimited ranking levels based on row count, NTILE() caps rankings at your specified value. This enables clear categorization rather than granular individual rankings.

Basic Syntax Structure

The general syntax is:

NTILE(n) OVER (ORDER BY column_name)

Where n is the number of tiles you want to create.

Partitioning and Ordering

NTILE() works within window partitions using PARTITION BY, allowing you to:

- Create tiles independently within each group

- Order rows within those tiles using ORDER BY

- Generate separate tile assignments for each partition

Section 3: Example Code and Use Cases

Example 1: Basic Product Segmentation into Four Quartiles

Divide all products into 4 equal segments:

```
SELECT
    ProductID,
    ProductName,
    CategoryID,
    Price,
    Stock,
    NTILE(4) OVER (ORDER BY Price) AS Quartile
FROM Products;
```

Result interpretation: Products are ranked 1-4 based on price distribution. Products with Quartile = 1 represent the lowest 25% of prices, Quartile = 4 represents the highest 25%.

Example 2: Price Segmentation within Each Category

Divide products into quartiles independently within each category:

```
SELECT
    ProductID,
    ProductName,
    CategoryID,
    Price,
    Stock,
    NTILE(4) OVER (PARTITION BY CategoryID ORDER BY Price) AS PriceSegment
FROM Products;
```

This creates category-specific quartiles. A category with 5 products assigns Quartiles 1, 2, 3, 4 to those 5 rows based on price order. A category with only 2 products receives Quartiles 1 and 2 only (no 3 or 4, as there aren't enough rows).

Example 3: Customer Order Value Deciles

Segment customers by their order spending into 10 deciles:

```
SELECT
    c.CustomerID,
    c.FirstName,
    c.LastName,
    SUM(o.TotalAmount) AS TotalSpent,
    NTILE(10) OVER (ORDER BY SUM(o.TotalAmount) DESC) AS SpendingDecile
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName;
```

Customers in Decile 1 are top spenders; those in Decile 10 are lowest spenders.

Example 4: Product Rating Segments by Category

Divide products into 5 performance segments based on average rating within each category:

```
SELECT
    p.ProductID,
    p.ProductName,
    p.CategoryID,
    AVG(r.Rating) AS AvgRating,
    NTILE(5) OVER (PARTITION BY p.CategoryID ORDER BY AVG(r.Rating) DESC)
AS PerformanceSegment
FROM Products p
LEFT JOIN Reviews r ON p.ProductID = r.ProductID
GROUP BY p.ProductID, p.ProductName, p.CategoryID;
```

Section 4: Key Takeaways

NTILE() creates fixed-size segments rather than unlimited ranks, making data analysis and categorization straightforward and meaningful.

Use NTILE(4) for quartiles, NTILE(10) for deciles, or any number matching your analytical needs. This is particularly useful when you want to classify customers, products, or orders into performance tiers or value segments.

PARTITION BY enables independent tile assignment within groups, allowing category-wise or customer-wise segmentation simultaneously.

The ORDER BY clause within NTILE() determines the distribution logic—whether by price, rating, amount, or any other metric. Ordering direction (ASC or DESC) affects which tiles receive higher or lower values.

NTILE() is simpler than RANK() when you need clear groupings rather than individual rankings, improving readability and reducing calculation complexity for business intelligence queries.