TUTORIAL Windowing: Aggregate, Ranking, dan Offset

Oleh: Yan Watequlis Syaifudin, Ph.D.

1. KONSEPTUAL

A. Konsep Dasar Windowing pada T-SQL

Windowing dalam T-SQL merujuk pada fitur SQL yang memungkinkan Anda melakukan perhitungan dan analisis pada sekumpulan baris (window) yang berhubungan dengan baris saat ini dalam query. Berbeda dengan fungsi agregat biasa yang mengembalikan satu baris hasil, window functions membiarkan setiap baris dalam hasil lengkap, tetapi menambahkan kolom baru berdasarkan perhitungan yang dilakukan pada jendela baris yang ditentukan.

Elemen Kunci dari Windowing:

1. **Fungsi Window**: Terdiri dari beberapa jenis fungsi yang digunakan dalam query:
   - **Aggregate Functions**: Seperti SUM(), AVG(), COUNT(), MIN(), dan MAX() yang menghitung nilai berdasarkan subset data.
   - **Ranking Functions**: Seperti ROW_NUMBER(), RANK(), DENSE_RANK(), dan NTILE() yang memberikan peringkat pada baris dalam partisi.
   - **Offset Functions**: Seperti LAG() dan LEAD() yang mengakses baris sebelumnya atau berikutnya dalam urutan data.
2. **Clause OVER()**: Bagian yang digunakan dengan window functions yang menentukan batas jendela:
   - **PARTITION BY**: Untuk membagi hasil menjadi subgroup berdasarkan satu atau beberapa kolom.
   - **ORDER BY**: Untuk menentukan urutan baris dalam jendela, sangat penting untuk fungsi-fungsi yang menghitung nilai terkait urutan.
3. **Frame Specification**: Anda dapat menentukan spesifikasi frame untuk batasan dalam jendela menggunakan:
   - **ROWS**: Menentukan jumlah baris yang menjadi bagian dari frame.
   - **RANGE**: Menentukan rentang nilai, terutama untuk perhitungan agregasi yang memerlukan lebih dari satu baris/nilai.
   -

Struktur Query dalam Windowing

Struktur query yang menggunakan window functions umumnya mengikuti pola berikut:

```
SELECT    column1, column2,
    WindowFunction(arguments) OVER (
        [PARTITION BY partition_column]
        [ORDER BY order_column]
        [ROWS | RANGE frame_specification]
    ) AS new_column_name
FROM  table_name
WHERE  condition
ORDER BY column1;
```

Manfaat Penggunaan Windowing

1. **Meningkatkan Pemahaman Data**: Windowing functions memberikan wawasan dari data yang lebih dalam dan konteks yang lebih jelas dengan melakukan perhitungan yang mempertimbangkan nilai lain di set data, tanpa kehilangan detail data lainnya.
2. **Fleksibilitas dalam Analisis**: Memungkinkan analisis data dengan berbagai cara yang berbeda dan menghasilkan agregasi tanpa harus merangkum data. Anda bisa menggunakan windowing untuk memisahkan analisis dalam populasinya, sekaligus tetap mempertahankan detail baris.
3. **Pengurangan Kompleksitas Query**: Mengurangi kebutuhan untuk membuat subquery atau tabel sementara yang akan mempersulit query menjadi lebih rumit. Dengan satu query yang sederhana, Anda dapat mendapatkan hasil dan analisis yang kompleks.
4. **Kinerja yang Lebih Baik**: Dalam banyak kasus, penggunaan windowing functions lebih efisien daripada metode tradisional dengan berkali-kali melakukan join atau subquery, terutama pada basis data yang besar.
5. **Dukungan untuk Tren dan Analisis Waktu**: Windowing memungkinkan untuk menghitung nilai kumulatif atau tren sepanjang waktu, yang sangat berguna dalam analisis finansial dan pengumpulan data.

B. Aggregate Functions dalam Windowing T-SQL

Aggregate Functions dalam T-SQL adalah fungsi yang digunakan untuk melakukan perhitungan pada satu set nilai, mengumpulkan data, dan mengembalikan satu nilai. Saat digunakan dalam konteks windowing dengan OVER(), fungsi agregat dapat memberikan hasil yang lebih fleksibel dan dinamis, memungkinkan analisis yang lebih mendalam pada dataset tanpa mempengaruhi jumlah baris hasil.

Konsep Dasar

Windowing functions memungkinkan pengguna untuk melakukan perhitungan agregat tanpa kehilangan detail baris individual. Dengan menggunakan **Aggregate Functions** bersama **Windowing**, Anda dapat menghitung nilai agregat seperti total, rata-rata, minimum, atau maksimum dalam subset data yang ditentukan, menggunakan jendela yang ditetapkan oleh PARTITION BY dan ORDER BY.

Penggunaan Umum

- **Analisis Kumulatif**: Menghitung total kumulatif, rata-rata kumulatif, dan sebagainya.
- **Statistik Per Grup**: Menghitung nilai agregat berdasarkan kelompok tertentu, seperti rata-rata gaji per departemen.
- **Memperoleh Nilai Relatif**: Menghitung nilai minimum, maksimum, atau jumlah dalam konteks grup.

Fungsi-Fungsi Agregat yang Umum Digunakan

Berikut adalah beberapa fungsi agregat yang umum digunakan dalam windowing di T-SQL:

1. **SUM()**: Menghitung jumlah total dari nilai dalam kolom.
2. **AVG()**: Menghitung rata-rata nilai dalam kolom.
3. **COUNT()**: Menghitung jumlah entri atau baris dalam hasil.
4. **MIN()**: Mengambil nilai minimum dari kolom.

5. **MAX()**: Mengambil nilai maksimum dari kolom.

Struktur Query

Struktur query yang menggunakan Aggregate Functions dalam windowing dengan OVER() mengikuti pola berikut:

```
SELECT <columns>,
    AggregateFunction(column_name) OVER (
        [PARTITION BY partition_column]
        [ORDER BY order_column]
        [ROWS | RANGE frame_specification]
    ) AS new_column_name
FROM  table_name
WHERE  <conditions>
ORDER BY  <columns>;
```

Penjelasan Struktur Query

- **SELECT** : Memilih kolom yang ingin ditampilkan dari tabel.
- **AggregateFunction(column_name)**: Menetapkan fungsi agregat seperti SUM(), AVG(), COUNT(), dan lain-lain.
- **OVER()**: Menentukan jendela untuk fungsi agregat:
    - **PARTITION BY**: Memisahkan hasil ke dalam grup berdasarkan kolom tertentu.
    - **ORDER BY**: Menentukan urutan data dalam setiap partisi yang mempengaruhi hasil fungsi.
    - **ROWS | RANGE**: Menentukan jumlah baris dalam jendela yang diperhitungkan.
- **FROM table_name**: Menentukan tabel dari mana data diambil.
- **WHERE** : Memfilter hasil query.
- **ORDER BY** : Menentukan urutan akhir hasil yang ditampilkan.

C. Ranking Functions dalam Windowing T-SQL

Ranking Functions dalam T-SQL adalah fungsi yang digunakan untuk memberikan peringkat atau urutan pada data dalam hasil query berdasarkan kriteria tertentu. Fungsi-fungsi ini sering digunakan dalam konteks analytic queries untuk membantu pengguna memahami posisi relatif dari setiap baris dalam satu set data. Ranking functions sangat berguna dalam berbagai analisis, termasuk persaingan, urutan, atau untuk menentukan posisi suatu entitas dalam satu kelompok.

Konsep Dasar

Ranking functions bekerja dengan cara menghitung urutan atau peringkat angka untuk setiap baris dalam set hasil. Fungsi ini beroperasi dalam konteks window (jendela) yang ditentukan oleh klausa OVER(), yang memungkinkan kita untuk memperhitungkan baris lain dalam window tersebut saat menentukan peringkat.

Penggunaan Umum

- **Mengurutkan Data**: Memberikan letak atau urutan nilai dalam dataset, seperti peringkat siswa berdasarkan nilai.

- **Analisis Kompetitif**: Menentukan posisi produk berdasarkan penjualan, rating, atau metrik lainnya.
- **Pembuatan Laporan**: Menghasilkan laporan peringkat berdasarkan kriteria yang relevan untuk menampilkan kinerja.

Fungsi-Fungsi Ranking

Di dalam T-SQL, terdapat beberapa fungsi ranking utama yang dapat digunakan, antara lain:

1. **ROW_NUMBER()**: Memberikan nomor urut unik untuk setiap baris dalam partisi, tanpa ada duplikasi.
2. **RANK()**: Memberikan peringkat untuk setiap baris dalam partisi, dengan peringkat yang sama untuk nilai yang sama, tetapi meninggalkan celah untuk peringkat berikutnya.
3. **DENSE_RANK()**: Mirip dengan RANK(), tetapi tidak ada celah di antara peringkat nilai yang sama (sequential).
4. **NTILE(n)**: Membagi set data menjadi n kelompok dan memberikan nomor kelompok untuk tiap baris.

Struktur Query

Struktur query untuk menggunakan fungsi ranking dengan OVER() mengikuti pola berikut:

```
SELECT <columns>,
    RankingFunction() OVER (
        [PARTITION BY partition_column]
        [ORDER BY order_column]
    ) AS new_column_name
FROM table_name
WHERE  <conditions>
ORDER BY  <columns>;
```

Penjelasan Struktur Query

- **SELECT** : Memilih kolom yang ingin ditampilkan dari tabel.
- **RankingFunction()**: Menentukan fungsi ranking seperti ROW_NUMBER(), RANK(), DENSE_RANK(), atau NTILE(n).
- **OVER()**: Menentukan konteks jendela untuk fungsi ranking:
  - **PARTITION BY**: Memisahkan hasil ke dalam grup berdasarkan kolom tertentu.
  - **ORDER BY**: Menentukan urutan data dalam setiap partisi yang mempengaruhi hasil fungsi.
- **FROM table_name**: Menentukan tabel dari mana data diambil.
- **WHERE** : Memfilter hasil query.
- **ORDER BY** : Menentukan urutan akhir hasil yang ditampilkan.

D. Offset Functions dalam Windowing T-SQL

Offset Functions dalam T-SQL adalah fungsi yang digunakan untuk mengakses baris lain dalam dataset relatif terhadap baris saat ini. Fungsi ini sangat berguna untuk analisis data yang memerlukan perbandingan antar nilai yang berada pada posisi berbeda dalam dataset. Offset functions umumnya digunakan dalam konteks analisis temporal atau untuk membandingkan nilai dari satu baris ke baris lainnya tanpa perlu membuat subquery yang rumit.

Konsep Dasar

Offset functions bekerja dengan menyediakan akses ke nilai baris lainnya berdasarkan posisi relatif terhadap baris saat ini dalam hasil query. Dengan menggunakan OVER() untuk mendefinisikan window, Anda dapat menentukan bagaimana offset function akan mempengaruhi perhitungan dari setiap baris.

Penggunaan Umum

- **Perbandingan Data**: Menggunakan offset functions untuk membandingkan nilai suatu baris dengan baris sebelumnya atau berikutnya.
- **Analisis Tren**: Menganalisis perubahan nilai sepanjang waktu, seperti penjualan dari hari ke hari.
- **Kumulatif dan Rolling Calculation**: Melihat perubahan kumulatif atau perhitungan bergerak dibandingkan dengan nilai sebelumnya.

Fungsi-Fungsi Offset

Terdapat beberapa fungsi offset yang umum digunakan dalam T-SQL:

1. **LAG()**: Memungkinkan akses nilai dari baris sebelumnya dalam set hasil berdasarkan urutan tertentu.
2. **LEAD()**: Memungkinkan akses nilai dari baris berikutnya dalam set hasil berdasarkan urutan tertentu.
3. **FIRST_VALUE()**: Mengambil nilai pertama dalam jendela.
4. **LAST_VALUE()**: Mengambil nilai terakhir dalam jendela.

Struktur Query

Struktur query untuk menggunakan offset functions dalam windowing mengikuti pola berikut:

```
SELECT  <columns>,
    OffsetFunction(arguments) OVER (
        [PARTITION BY partition_column]
        [ORDER BY order_column]
    ) AS new_column_name
FROM  table_name
WHERE  <conditions>
ORDER BY  <columns>;
```

Penjelasan Struktur Query

- **SELECT** : Memilih kolom yang ingin ditampilkan dari tabel.
- **OffsetFunction(arguments)**: Menentukan fungsi offset seperti LAG(), LEAD(), FIRST_VALUE(), atau LAST_VALUE().
- **OVER()**: Menentukan jendela untuk fungsi offset:
    - **PARTITION BY**: Memisahkan hasil ke dalam grup berdasarkan kolom tertentu.
    - **ORDER BY**: Menentukan urutan data dalam setiap partisi yang mempengaruhi hasil fungsi.
- **FROM table_name**: Menentukan tabel dari mana data diambil.
- **WHERE** : Memfilter hasil query.
- **ORDER BY** : Menentukan urutan akhir dari hasil yang ditampilkan.

2. TUTORIAL (STUDI KASUS: Penjualan Perusahaan)

A. Definisi Tabel

1. Tabel Employees

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DepartmentID INT NOT NULL,
    Salary DECIMAL(10, 2) NOT NULL
);
```

2. Tabel Sales

```
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    SaleDate DATE NOT NULL,
    Amount DECIMAL(10, 2) NOT NULL,
    EmployeeID INT NOT NULL,
    ProductID INT NOT NULL,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

3. Tabel Product

```
CREATE TABLE Product (
    ProductID INT PRIMARY KEY,
    ProductName NVARCHAR(100) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL,
    Stock INT NOT NULL
);
```

Pengisian Data

1. Mengisi Tabel Employees

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID,
Salary) VALUES
(1, '<Nama Unik>', '<Unik 2>', 1, 60000.00),
(2, 'Jane', 'Smith', 1, 70000.00),
(3, 'Sam', 'Brown', 2, 45000.00),
(4, 'Linda', 'Jones', 2, 50000.00),
(5, 'Mike', 'Davis', 3, 75000.00),
(6, 'Emily', 'Clark', 1, 80000.00),
(7, 'Jacob', 'Williams', 3, 55000.00),
(8, '<Nama mhs>', '<Belakang mhs>', 4, 62000.00);
```

2. Mengisi Tabel Sales

```
INSERT INTO Sales (SaleID, SaleDate, Amount, EmployeeID, ProductID)
VALUES
```

```
(1, '2024-01-01', 500.00, 1, 1),
(2, '2024-01-02', 600.00, 1, 2),
(3, '2024-01-03', 700.00, 2, 2),
(4, '2024-01-04', 800.00, 3, 3),
(5, '2024-01-05', 900.00, 3, 1),
(6, '2024-01-06', 300.00, 4, 3),
(7, '2024-01-07', 400.00, 5, 1),
(8, '2024-01-08', 200.00, 6, 2),
(9, '2024-01-09', 1000.00, 7, 3),
(10, '2024-01-10', 1200.00, 8, 2);
```

3. Mengisi Tabel Product

```
INSERT INTO Product (ProductID, ProductName, Price, Stock) VALUES
(1, 'Laptop', 1500.00, 10),
(2, 'Smartphone', 800.00, 20),
(3, 'Tablet', 300.00, 15),
(4, 'Printer', 200.00, 5),
(5, 'Headphones', 100.00, 50),
(6, 'Monitor', 250.00, 8),
(7, 'Keyboard', 50.00, 30),
(8, 'Mouse', 20.00, 40),
(9, 'Webcam', 90.00, 25),
(10, 'External Hard Drive', 120.00, 12);
```

B. Fungsi Agregat dengan Windowing

Fungsi agregat pada windowing di T-SQL merupakan alat yang sangat kuat untuk analisis data di SQL Server. Konsep ini memungkinkan pengguna untuk melakukan perhitungan agregat di dalam konteks yang spesifik (disebut jendela atau window) tanpa mengelompokkan hasil menjadi satu baris.

Konsep Dasar Fungsi Agregat

1. **Fungsi Agregat**: Fungsi agregat adalah fungsi yang menghitung nilai berdasarkan kelompok data. Contoh fungsi agregat yang umum digunakan termasuk:
    - SUM(): Menghitung total nilai.
    - AVG(): Menghitung rata-rata nilai.
    - COUNT(): Menghitung jumlah entri.
    - MIN(): Menemukan nilai minimum.
    - MAX(): Menemukan nilai maksimum.
2. **Window Functions**: Fungsi ini memungkinkan analisis data dengan cara yang tidak mungkin dicapai dengan fungsi agregat tradisional. Ketika menggunakan window functions, Anda dapat mendapatkan hasil yang mempertahankan detail baris sekaligus memberikan perhitungan berbasis konteks.

Struktur Umum Window Functions

Struktur umum untuk menggunakan fungsi agregat dalam windowing adalah:

```
AggregateFunction(column_name) OVER (
    [PARTITION BY partition_expression]
    [ORDER BY order_expression]
    [ROWS | RANGE frame_specification]
)
```

- **AggregateFunction(column_name)**: Fungsi agregat yang ingin digunakan (misalnya, SUM(), AVG(), dll.).
- **OVER(...)**: Bagian ini menentukan jendela di mana fungsi agregat diterapkan.
- **PARTITION BY**: Digunakan untuk membagi hasil menjadi kelompok. Setiap grup dihitung secara terpisah.
- **ORDER BY**: Menentukan urutan dalam jendela untuk fungsi yang menghitung yang bergantung pada urutan.
- **ROWS atau RANGE**: Menentukan sejumlah baris yang akan digunakan dalam perhitungan. Ini memungkinkan kontrol lebih besar atas seberapa banyak data yang dimanfaatkan untuk melakukan perhitungan.

1. Menghitung Total Gaji Kumulatif untuk Karyawan
Tujuan: Menghitung total gaji kumulatif berdasarkan EmployeeID.

```
SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary,
    SUM(Salary) OVER (ORDER BY EmployeeID) AS CumulativeSalary
FROM  Employees
ORDER BY  EmployeeID;
```

Penjelasan:

- **Fungsi SUM()**: Menghitung total dari kolom Salary.
- **OVER (ORDER BY EmployeeID)**: Mengatur urutan penghitungan berdasarkan EmployeeID, sehingga total kumulatif dihitung dari awal hingga karyawan yang sedang diproses.
- **CumulativeSalary**: Kolom baru yang menunjukkan total kumulatif gaji hingga setiap karyawan.

2. Menghitung Rata-rata Penjualan Per Produk
Tujuan: Menghitung rata-rata jumlah penjualan untuk setiap produk.

```
SELECT p.ProductID, p.ProductName,
    AVG(s.Amount) OVER (PARTITION BY p.ProductID) AS AvgSales
FROM  Product p
JOIN Sales s ON p.ProductID = s.ProductID;
```

Penjelasan:

- **Fungsi AVG()**: Menghitung rata-rata dari Amount di tabel Sales.
- **PARTITION BY p.ProductID**: Memisahkan perhitungan rata-rata untuk setiap produk secara terpisah.
- **JOIN**: Menggabungkan tabel Product dan Sales berdasarkan ProductID untuk mendapatkan data penjualan yang relevan.

3. Menghitung Juml ah Penjualan Kumulatif Berdasarkan Tanggal
Tujuan: Menghitung total kumulatif penjualan berdasarkan tanggal penjualan.

```
SELECT SaleDate, Amount,
    SUM(Amount) OVER (ORDER BY SaleDate) AS CumulativeSales
FROM  Sales
ORDER BY  SaleDate;
```

Penjelasan:

- **Fungsi SUM()**: Menghitung total kumulatif dari kolom Amount.
- **ORDER BY SaleDate**: Memberikan urutan berdasarkan SaleDate, sehingga total kumulatif dihitung sesuai dengan tanggal penjualan.
- **CumulativeSales**: Kolom baru yang menampilkan total kumulatif penjualan hingga setiap tanggal.

4. Menghitung Total Penjualan Setiap Karyawan
Tujuan: Menghitung total penjualan yang dilakukan oleh setiap karyawan.

```
SELECT  e.EmployeeID, e.FirstName, e.LastName,
    SUM(s.Amount) OVER (PARTITION BY e.EmployeeID) AS TotalSales
FROM  Employees e
JOIN  Sales s ON e.EmployeeID = s.EmployeeID;
```

Penjelasan:

- **Fungsi SUM()**: Menjumlahkan total penjualan dari kolom Amount.
- **PARTITION BY e.EmployeeID**: Membagi hasil berdasarkan setiap karyawan untuk menghitung total penjualan secara terpisah.
- **JOIN**: Menyambungkan tabel Employees dengan Sales untuk mendapatkan jumlah setiap penjualan yang dilakukan oleh karyawan.

5. Menghitung Gaji Karyawan Sebagai Persentase dari Rata-rata Gaji Departemen
Tujuan: Menghitung persentase gaji setiap karyawan dibandingkan dengan rata-rata gaji di departemen mereka.

```
SELECT   e.EmployeeID, e.FirstName, e.LastName, e.Salary,
    AVG(e.Salary)   OVER   (PARTITION   BY   e.DepartmentID)   AS
AvgDepartmentSalary,
    (e.Salary   *   100.0   /   AVG(e.Salary)   OVER   (PARTITION   BY
e.DepartmentID)) AS SalaryPercentage
FROM  Employees e;
```

Penjelasan:

- **Fungsi AVG()**: Menghitung rata-rata gaji di setiap departemen.
- **PARTITION BY e.DepartmentID**: Memisahkan penghitungan agar rata-rata dihitung per departemen.

- **SalaryPercentage**: Menghitung persentase gaji karyawan dibandingkan rata-rata gaji di departemen dengan formula "(gaji karyawan / rata-rata gaji di departemen) * 100.0".

6. Menghitung Penjualan Kumulatif dengan Informasi Produk

Tujuan: Menghitung penjualan kumulatif untuk setiap produk dan melibatkan informasi nama produk.

```
SELECT p.ProductID, p.ProductName, s.SaleDate, s.Amount,
    SUM(s.Amount) OVER (PARTITION BY p.ProductID ORDER BY s.SaleDate)
AS CumulativeSales
FROM Product p
JOIN Sales s ON p.ProductID = s.ProductID
ORDER BY  p.ProductID, s.SaleDate;
```

Penjelasan:

- **Fungsi SUM()**: Menghitung total kumulatif dari kolom Amount untuk penjualan setiap produk.
- **PARTITION BY p.ProductID**: Memisahkan penghitungan untuk setiap produk.
- **ORDER BY s.SaleDate**: Mengatur urutan penjualan berdasarkan tanggal untuk mendapatkan total kumulatif secara tepat dari waktu ke waktu.
- **CumulativeSales**: Kolom baru yang menunjukkan total kumulatif penjualan untuk setiap produk sampai tanggal penjualan yang sedang diproses.

7. Menghitung Jumlah Karyawan Berdasarkan Departemen

Tujuan: Menghitung jumlah karyawan di setiap departemen dan memberikan informasi ini pada setiap baris karyawan.

```
SELECT e.EmployeeID, e.FirstName, e.LastName, e.DepartmentID,
    COUNT(e.EmployeeID)  OVER  (PARTITION  BY  e.DepartmentID)  AS
NumberOfEmployees
FROM     Employees e;
```

Penjelasan:

- **Fungsi COUNT()**: Menghitung jumlah karyawan.
- **PARTITION BY e.DepartmentID**: Memisahkan hasil berdasarkan setiap departemen, sehingga tidak mempengaruhi penghitungan karyawan di departemen lain.
- **NumberOfEmployees**: Kolom baru ini menunjukkan total jumlah karyawan dalam departemen yang sama untuk setiap baris karyawan, memberikan konteks tambahan dalam laporan.

Kesimpulan

Penggunaan fungsi agregat dengan windowing di T-SQL memberikan fleksibilitas dan kekuatan dalam analisis data. Dengan fungsi-fungsi seperti SUM(), AVG(), dan COUNT(), Anda dapat melakukan berbagai perhitungan yang relevan dengan data yang ada dalam tabel-tabel **Employees**, **Sales**, dan **Product**. Fungsi-fungsi tersebut, saat digunakan dalam konteks windowing dan bersamaan dengan **JOIN** untuk menggabungkan data dari beberapa tabel, memungkinkan analisis yang lebih mendalam dan mendetail mengenai kinerja karyawan, penjualan produk, dan keadaan finansial dalam perusahaan.

C. Fungsi Ranking dengan Windowing

Fungsi ranking pada windowing di T-SQL merupakan alat yang powerful untuk memberikan peringkat atau nomor urut kepada baris dalam result set berdasarkan kriteria tertentu. Fungsi-fungsi ini melibatkan perhitungan yang memerlukan kontekstualisasi data untuk memberi urutan atau peringkat kepada data dalam satu set.

Jenis Fungsi Ranking

T-SQL menyediakan beberapa fungsi ranking, termasuk:

- **ROW_NUMBER()**: Menghasilkan nomor urut unik untuk setiap baris dalam hasil, tanpa mempertimbangkan nilai yang sama. Nomor urut akan dimulai dari 1 dan meningkat satu per baris.
- **RANK()**: Memberikan peringkat yang sama untuk nilai yang sama. Jika dua baris memiliki nilai yang sama, mereka akan mendapat peringkat yang sama. Namun, peringkat berikutnya akan memiliki celah (misalnya, jika dua karyawan berada di peringkat 1, karyawan berikutnya akan berada di peringkat 3).
- **DENSE_RANK()**: Mirip dengan RANK(), tetapi tidak memiliki celah antara peringkat. Jika dua baris memiliki nilai yang sama, mereka akan menerima peringkat yang sama, tetapi peringkat berikutnya akan melanjutkan dengan angka berikutnya.
- **NTILE(n)**: Membagi set data ke dalam n kelompok yang hampir sama dan memberikan nomor kelompok untuk setiap baris.

Struktur Umum Fungsi Ranking

Struktur umum untuk menggunakan fungsi ranking dalam windowing adalah sebagai berikut:

```
RankingFunction() OVER (
    [PARTITION BY partition_expression]
    [ORDER BY order_expression]
)
```

- **RankingFunction()**: Ini adalah fungsi ranking yang ingin diterapkan (seperti ROW_NUMBER(), RANK(), dll.).
- **OVER(...)**: Menandai bahwa kita sedang menggunakan window function.
    - **PARTITION BY**: Mencetak hasil ke dalam grup berdasarkan kolom tertentu. Fungsi ranking dihitung di dalam setiap partisi secara terpisah.
    - **ORDER BY**: Menentukan urutan baris yang akan dinilai saat memberikan peringkat.

1. Menggunakan ROW_NUMBER() untuk Mengurutkan Karyawan Berdasarkan Gaji

Tujuan: Memberikan nomor urut untuk setiap karyawan berdasarkan gaji di setiap departemen.

```
SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary,
    ROW_NUMBER() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC)
AS SalaryRank
FROM Employees;
```

Penjelasan:

- **ROW_NUMBER()** memberikan nomor urut berdasarkan posisi masing-masing karyawan yang diurutkan berdasarkan gaji.
- **PARTITION BY DepartmentID**: Memisahkan perhitungan berdasarkan department, sehingga peringkat dimulai dari 1 untuk setiap department.
- **ORDER BY Salary DESC**: Mengurutkan gaji dari yang tertinggi, sehingga karyawan dengan gaji tertinggi mendapatkan nomor urut 1.

2. Menggunakan RANK() untuk Memberikan Peringkat Berdasarkan Gaji

Tujuan: Memberikan peringkat kepada karyawan berdasarkan gaji, dengan mempertimbangkan karyawan yang memiliki gaji yang sama.

```
SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary,
    RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS
SalaryRank
FROM   Employees;
```

Penjelasan:

- **RANK()** memberikan angka peringkat yang sama untuk karyawan dengan gaji yang sama. Jika dua karyawan di departemen yang sama memiliki gaji yang identik, mereka akan memiliki peringkat yang sama, tetapi peringkat berikutnya akan memiliki celah.
- **PARTITION BY DepartmentID** membatasi perhitungan sehingga setiap departemen memiliki peringkat masing-masing.
- **ORDER BY Salary DESC** menentukan urutan peringkat berdasarkan gaji tertinggi.

3. Menggunakan DENSE_RANK() untuk Memberikan Peringkat Tanpa Celah

Tujuan: Memberikan peringkat kepada karyawan tanpa kelemahan, meskipun terdapat karyawan dengan gaji yang sama.

```
SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary,
    DENSE_RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC)
AS SalaryRank
FROM Employees;
```

Penjelasan:

- **DENSE_RANK()** mirip dengan RANK(), tetapi tanpa celah di antara peringkat yang sama, yang memudahkan untuk komparasi nilai.
- Setiap grup akan mendapatkan peringkat yang konsisten meskipun ada nilai yang sama.
- **PARTITION BY DepartmentID** memastikan bahwa pencacahan peringkat dilakukan di dalam departemen.

4. Menggunakan NTILE() untuk Membagi Karyawan Menjadi Kuartil Berdasarkan Gaji

Tujuan: Membagi karyawan menjadi 4 kelompok berdasarkan kuartil gaji di seluruh tabel.

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

Penjelasan:

- **NTILE(4)** membagi seluruh karyawan ke dalam 4 kelompok (kuartil), berdasarkan gaji mereka.
- **ORDER BY Salary DESC** menentukan urutan gaji dari yang tertinggi sehingga kelompok tersebut dipisahkan sesuai dengan perbandingan relatif.
- **SalaryQuartile** memberikan nomor kuartil untuk setiap karyawan berdasarkan posisi gaji mereka.

5. Menggunakan Fungsi Ranking dengan JOIN untuk Menghitung Penjualan Kumulatif Karyawan

Tujuan: Menghitung penjualan kumulatif yang dilakukan oleh setiap karyawan berdasarkan urutan tanggal penjualan.

```
SELECT e.EmployeeID, e.FirstName, e.LastName, s.SaleDate, s.Amount,
    SUM(s.Amount) OVER (PARTITION BY e.EmployeeID ORDER BY s.SaleDate)
AS CumulativeSales
FROM Employees e
JOIN Sales s ON e.EmployeeID = s.EmployeeID
ORDER BY e.EmployeeID, s.SaleDate;
```

Penjelasan:

- **SUM(s.Amount) OVER (PARTITION BY e.EmployeeID ORDER BY s.SaleDate)**: Menghitung total kumulatif dari jumlah penjualan setiap karyawan berdasarkan urutan tanggal. Kumulatif akan diperbarui setiap kali ada penjualan baru yang ditambahkan.
- **PARTITION BY e.EmployeeID** memisahkan perhitungan sehingga masing-masing karyawan mendapat total kumulatif penjualannya sendiri.
- **JOIN** digunakan untuk menggabungkan tabel Employees dan Sales berdasarkan EmployeeID, sehingga kita bisa mengaitkan penjualan dengan karyawan yang melakukan penjualan.

6. Menggunakan Peringkat Penjualan Karyawan

Tujuan: Memberikan peringkat penjualan bulanan untuk setiap karyawan berdasarkan total penjualan yang dilakukan.

```
SELECT e.EmployeeID, e.FirstName, e.LastName,
    SUM(s.Amount) AS TotalSales,
    RANK() OVER (ORDER BY SUM(s.Amount) DESC) AS SalesRank
FROM Employees e
JOIN  Sales s ON e.EmployeeID = s.EmployeeID
GROUP BY e.EmployeeID, e.FirstName, e.LastName
ORDER BY SalesRank;
```

Penjelasan:

- **SUM(s.Amount)**: Menghitung total penjualan untuk masing-masing karyawan.

- **RANK() OVER (ORDER BY SUM(s.Amount) DESC)**: Menetapkan peringkat untuk setiap karyawan berdasarkan total penjualannya, di mana karyawan dengan total penjualan terbanyak mendapatkan peringkat tertinggi (1).
- **GROUP BY e.EmployeeID, e.FirstName, e.LastName**: Mengelompokkan data berdasarkan karyawan agar total penjualan dihitung dengan tepat.
- **ORDER BY SalesRank**: Mengurutkan hasil berdasarkan peringkat penjualan sehingga peringkat dengan total penjualan tertinggi muncul di atas.

Kesimpulan
Fungsi ranking pada windowing di T-SQL memberikan kemampuan untuk memberikan peringkat dan nomor urut kepada baris dalam hasil query, memberikan wawasan yang lebih dalam tentang data. Dengan menggunakan fungsi-fungsi seperti ROW_NUMBER(), RANK(), DENSE_RANK(), dan NTILE(), Anda dapat menganalisis posisi relatif dari satu set data berdasarkan kriteria tertentu.
Pada contoh-contoh di atas, kami menunjukkan bagaimana fungsi ranking dapat digabungkan dengan operasi **JOIN** untuk memberikan analisis lengkap tentang karyawan, penjualan, dan produk. Ini memberikan persepsi yang lebih kaya dalam pengambilan keputusan berbasis data.

D. Fungsi Agregat dengan Windowing

Fungsi offset dalam konteks windowing di T-SQL adalah fitur yang memungkinkan Anda untuk mengakses nilai dari baris lain dalam dataset berdasarkan posisi relatif terhadap baris yang saat ini sedang diproses. Fungsi ini sangat berguna untuk analisis data yang memerlukan komparasi antar nilai dalam satu set hasil, memungkinkan Anda untuk melakukan analisis yang lebih tematik dan mendalam.

1. Jenis-Jenis Fungsi Offset
Dua fungsi offset yang paling umum di T-SQL adalah:
- **LAG()**: Fungsi ini mengizinkan Anda untuk mengakses nilai dari baris sebelumnya dalam urutan yang ditentukan. (Mengambil nilai dari baris sebelumnya).
- **LEAD()**: Fungsi ini mengizinkan Anda untuk mengakses nilai dari baris berikutnya dalam urutan yang ditentukan. (Mengambil nilai dari baris berikutnya).

2. Sintaks Umum
Berikut adalah sintaks umum untuk menggunakan fungsi offset dalam windowing:

```
LAG(column_name, offset, default_value) OVER (ORDER BY order_column)
```

atau

```
LEAD(column_name, offset, default_value) OVER (ORDER BY order_column)
```

- **column_name**: Nama kolom yang nilainya akan diambil.
- **offset**: Jumlah baris yang ingin Anda lihat relatif terhadap baris saat ini (default adalah 1).
- **default_value**: Nilai yang akan diberikan jika tidak ada baris sebelumnya atau berikutnya (default adalah NULL).

- **OVER**: Menunjukkan bahwa fungsi ini adalah window function.
- **ORDER BY**: Menentukan urutan pengolahan baris.

1. Menggunakan LAG() untuk Membandingkan Gaji Karyawan dengan Gaji Sebelumnya
Tujuan: Menampilkan gaji setiap karyawan berserta gaji karyawan sebelumnya dalam urutan EmployeeID.

```
SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary,
    LAG(Salary, 1, 0) OVER (ORDER BY EmployeeID) AS PreviousSalary
FROM Employees
ORDER BY EmployeeID;
```

Penjelasan:
- **Fungsi LAG(Salary, 1, 0)**: Mengambil gaji dari karyawan sebelumnya. Jika tidak ada karyawan sebelumnya, mengembalikan 0 sebagai default.
- **OVER (ORDER BY EmployeeID)**: Menentukan bahwa perhitungan dilakukan berdasarkan urutan EmployeeID, sehingga kita dapat membandingkan gaji satu karyawan dengan karyawan sebelumnya.
- **PreviousSalary**: Kolom baru yang menunjukkan gaji karyawan sebelum karyawan saat ini.

2. Menggunakan LEAD() untuk Menampilkan Perubahan Gaji Karyawan
Tujuan: Menampilkan gaji setiap karyawan berserta gaji karyawan berikutnya dalam urutan EmployeeID.

```
SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary,
    LEAD(Salary, 1, 0) OVER (ORDER BY EmployeeID) AS NextSalary
FROM Employees
ORDER BY EmployeeID;
```

Penjelasan:
- **Fungsi LEAD(Salary, 1, 0)**: Mengambil gaji dari karyawan setelahnya. Jika tidak ada karyawan berikutnya, mengembalikan 0 sebagai default.
- **OVER (ORDER BY EmployeeID)**: Menentukan urutan perhitungan berdasarkan EmployeeID, sehingga kita dapat melihat gaji berikutnya.
- **NextSalary**: Kolom baru yang menunjukkan gaji karyawan berikutnya.

3. Menggunakan LAG() untuk Menghitung Selisih Penjualan Karyawan
Tujuan: Menghitung selisih penjualan antara satu hari dengan hari sebelumnya untuk setiap karyawan.

```
SELECT  s.EmployeeID, e.FirstName, e.LastName, s.SaleDate, s.Amount,
    LAG(s.Amount, 1, 0) OVER (PARTITION BY s.EmployeeID ORDER BY
s.SaleDate) AS PreviousDayAmount,
    s.Amount - LAG(s.Amount, 1, 0) OVER (PARTITION BY s.EmployeeID ORDER
BY s.SaleDate) AS ChangeInSales
FROM Sales s JOIN  Employees e ON s.EmployeeID = e.EmployeeID
ORDER BY s.EmployeeID, s.SaleDate;
```

Penjelasan:

- **Fungsi LAG(s.Amount, 1, 0)**: Mengambil jumlah penjualan karyawan dari hari sebelumnya. Mengembalikan 0 jika tidak ada penjualan sebelumnya.
- **PARTITION BY s.EmployeeID**: Memisahkan perhitungan untuk setiap karyawan, sehingga penghitungan dilakukan secara terpisah untuk setiap karyawan.
- **ChangeInSales**: Kolom baru yang menghitung selisih penjualan hari ini dengan penjualan hari sebelumnya.

4. Menggunakan LEAD() untuk Melihat Tren Penjualan Produk

Tujuan: Melihat penjualan setiap produk untuk hari ini dibandingkan dengan hari berikutnya.

```
SELECT s.SaleDate, s.ProductID, p.ProductName, s.Amount,
    LEAD(s.Amount, 1, 0) OVER (PARTITION BY s.ProductID ORDER BY
s.SaleDate) AS NextDayAmount
FROM Sales s
JOIN Product p ON s.ProductID = p.ProductID
ORDER BY s.ProductID, s.SaleDate;
```

Penjelasan:

- **Fungsi LEAD(s.Amount, 1, 0)**: Mengambil jumlah penjualan produk untuk hari berikutnya. Jika tidak ada penjualan untuk produk tersebut di hari berikutnya, maka fungsi ini akan mengembalikan 0 sebagai nilai default.
- **PARTITION BY s.ProductID**: Memisahkan penghitungan berdasarkan setiap produk sehingga kita bisa melihat perbandingan penjualan berdasarkan produk secara terpisah.
- **NextDayAmount**: Kolom baru yang menunjukkan jumlah penjualan produk untuk hari berikutnya. Ini memungkinkan Anda untuk menganalisis bagaimana penjualan produk berubah dari satu hari ke hari berikutnya.

5. Menghitung Persentase Perubahan Penjualan Karyawan

Tujuan: Mengukur persentase perubahan penjualan dari satu hari ke yang berikutnya untuk masing-masing karyawan.

```
SELECT s.EmployeeID, e.FirstName, e.LastName, s.SaleDate, s.Amount,
    LAG(s.Amount, 1, 0) OVER (PARTITION BY s.EmployeeID ORDER BY s.SaleDate)
AS PreviousDayAmount,
    CASE WHEN LAG(s.Amount, 1, 0) OVER (PARTITION BY s.EmployeeID ORDER BY
s.SaleDate) = 0
    THEN NULL  -- Menghindari pembagian dengan nol
    ELSE ((s.Amount - LAG(s.Amount, 1, 0) OVER (PARTITION BY s.EmployeeID
ORDER BY s.SaleDate)) * 100.0 / LAG(s.Amount, 1, 0) OVER (PARTITION BY
s.EmployeeID ORDER BY s.SaleDate)) AS PercentChange
    END
FROM Sales s
JOIN  Employees e ON s.EmployeeID = e.EmployeeID
ORDER BY  s.EmployeeID, s.SaleDate;
```

Penjelasan:

- **Fungsi LAG(s.Amount, 1, 0)**: Mengambil nilai penjualan dari hari sebelumnya untuk karyawan yang sama.
- **CASE Statement**: Menghindari pembagian dengan nol. Jika jumlah penjualan sebelumnya memang nol, kita tidak ingin melakukan perhitungan, sehingga mengembalikan NULL.
- **Penghitungan Persentase Perubahan**: Menghitung persentase perubahan antara penjualan hari ini dan penjualan hari sebelumnya; hasilnya merupakan presentase perubahan penjualan dari hari sebelumnya ke hari ini.

Kesimpulan

Fungsi offset pada windowing di T-SQL, seperti **LAG()** dan **LEAD()**, sangat berguna dalam analisis data untuk:

- Membandingkan nilai antar baris dalam konteks yang relevan.
- Memungkinkan analisis tren dan perubahan antara nilai NCX.
- Mengoptimalkan query dengan mengurangi kebutuhan untuk subqueries kompleks.

Dengan menggabungkan fungsi offset ini dengan teknik **JOIN**, Anda dapat melakukan analisis komprehensif yang mengaitkan data dari beberapa tabel untuk mendapatkan wawasan yang lebih mendalam.