SQL-05 | Window Functions

Lecture Queries

WINDOW Functions

Window Functions

Window fns give the ability to put the values from one row of data into context compared to a group of rows, or partition.

We can answer questions like

- If the dataset were sorted, where would this row land in the results?
- How does a value in this row compare to a value in the prior row?
- How does a value in the current row compare to the average value for its group?

So, window functions **return group aggregate calculations alongside individual row-level** information for items in that group, or partition.

Question: Get the price of the most expensive item per vendor?

CustID	OrderID	TotalDue
1	101	$100
2	102	$150
1	103	$90
3	104	$80
2	105	$200
1	106	$150

CustID	OrderID	TotalDue
1	101	$100
1	103	$90
1	106	$150

CustID	OrderID	TotalDue
2	102	$150
2	105	$200

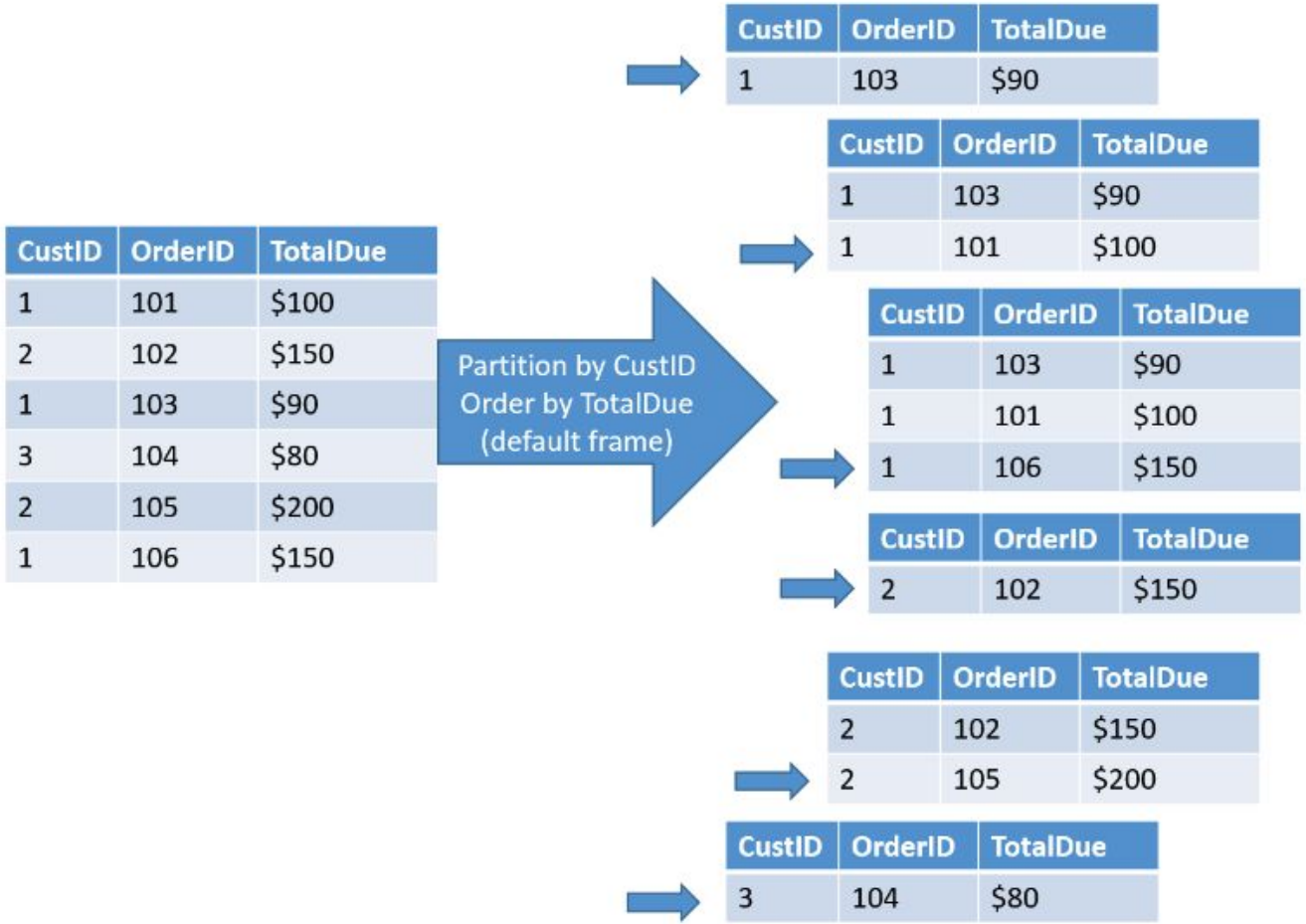
CustID	OrderID	TotalDue
3	104	$80

CustID	OrderID	TotalDue
1	101	$100
2	102	$150
1	103	$90
3	104	$80
2	105	$200
1	106	$150

CustID	OrderID	TotalDue
1	103	$90
1	101	$100
1	106	$150

CustID	OrderID	TotalDue
2	102	$150
2	105	$200

CustID	OrderID	TotalDue
3	104	$80


CustID OrderID TotalDue
1 101 $100
2 102 $150
1 103 $90
3 104 $80
2 105 $200
1 106 $150
Partition by CustID
Order by TotalDue
(default frame)
CustID OrderID TotalDue
1 103 $90
CustID OrderID TotalDue
1 103 $90
1 101 $100
CustID OrderID TotalDue
1 103 $90
1 101 $100
1 106 $150
CustID OrderID TotalDue
2 102 $150
CustID OrderID TotalDue
2 102 $150
2 105 $200
CustID OrderID TotalDue
3 104 $80

Question: Rank the products on their price per vendor and the associated **product_id**.

```
SELECT
        vendor_id,
        market_date,
        product_id,
        original_price,
    ROW_NUMBER() OVER (PARTITION BY vendor_id ORDER BY
original_price DESC) AS price_rank
FROM farmers_market.vendor_inventory
ORDER BY vendor_id, original_price DESC
```

RANK()

The RANK function numbers the results just like ROW_NUMBER does, but gives rows with the same value the same ranking.

```
SELECT
    vendor_id,
    market_date,
    product_id,
    original_price,
    RANK() OVER (PARTITION BY vendor_id ORDER BY
original_price DESC) AS
 price_rank
    FROM farmers_market.vendor_inventory
```

DENSE_RANK()

If you don’t want to skip rank numbers for tied values like in case of RANK, use the DENSE_RANK function.

```
SELECT
    vendor_id,
    market_date,
    product_id,
    original_price,
    DENSE_RANK() OVER (PARTITION BY vendor_id ORDER
BY original_price DESC) AS
 price_rank
    FROM farmers_market.vendor_inventory
```

Return the “**top tenth**” of the inventory, when sorted by price?

The dynamic solution is to use the NTILE function.

```
SELECT
        vendor_id,
        market_date,
        product_id,
        original_price,
        NTILE(10) OVER (ORDER BY original_price DESC) AS price_ntile
FROM farmers_market.vendor_inventory
ORDER BY original_price DESC
```

Question: As a farmer, you want to figure out which of your products were above the average price per product on each market date?

```
SELECT
        vendor_id,
        market_date,
        product_id,
        original_price,
        AVG(original_price) OVER (PARTITION BY market_date) AS
average_cost_product_by_market_date
 FROM farmers_market.vendor_inventory
```

```
SELECT *
FROM
(
SELECT
        vendor_id,
        market_date,
        product_id,
        original_price,
        AVG(original_price) OVER
(PARTITION BY market_date) AS
average_cost_product_by_market_
date
 FROM
farmers_market.vendor_inventory
 ) x
 where x.original_price >
x.average_cost_product_by_market
_date
```

Question: As a farmer, you want to figure out which of your products were above the average price per product on each market date?

```
SELECT
        vendor_id,
        market_date,
        product_id,
        original_price,
        AVG(original_price) OVER (PARTITION BY market_date ORDER BY
       market_date) AS average_cost_product_by_market_date
 FROM farmers_market.vendor_inventory
```

Extract the farmer's products that have prices above the market date's average product cost.

- Using a **subquery**, we can filter the results to a single vendor, with **vendor_id 8**, and
- only **display products that have prices above the market date's average product** cost.

```
SELECT * FROM
        (
            SELECT
                vendor_id,
                    market_date,
                    product_id,
                    original_price,
                    ROUND(AVG(original_price) OVER (PARTITION BY market_date ORDER BY
          market_date), 2) AS average_cost_product_by_market_date
FROM farmers_market.vendor_inventory )x
WHERE x.vendor_id = 8
                    AND x.original_price > x.average_cost_product_by_market_date
ORDER BY x.market_date, x.original_price DESC
```

Question: Count how many different products each vendor brought to market on each date, and displays that count on each row.

```
SELECT
    vendor_id,
    market_date,
    product_id,
    original_price,
    COUNT(product_id) OVER (PARTITION BY market_date, vendor_id)
 vendor_product_count_per_market_date
    FROM farmers_market.vendor_inventory
 ORDER BY vendor_id, market_date, original_price DESC
```

Question: Calculate the running total of the cost of items purchased by each customer, sorted by the date and time and the ***product_id***

```
SELECT customer_id,
    market_date,
    vendor_id,
    product_id,
    quantity * cost_to_customer_per_qty AS price,
    SUM(quantity * cost_to_customer_per_qty) OVER (PARTITION BY
 customer_id ORDER BY market_date, transaction_time, product_id) AS
 customer_spend_running_total
 FROM farmers_market.customer_purchases
```

Question: Using the **vendor_booth_assignments** table in the Farmer's Market database, display each vendor's booth assignment for each ***market_date*** alongside their previous booth assignments.

```
SELECT
        market_date,
        vendor_id,
        booth_number,
        LAG(booth_number,1) OVER (PARTITION BY vendor_id ORDER BY
market_date, vendor_id) AS previous_booth_number
FROM farmers_market.vendor_booth_assignments
ORDER BY market_date, vendor_id, booth_number
```

Question: The Market manager may want to filter these query results to a specific market date to determine which vendors are new or changing booths that day, so we can contact them and ensure setup goes smoothly.

Check it for date: 2019-04-10

```
SELECT * FROM
  (
     SELECT
          market_date,
           vendor_id,
           booth_number,
           LAG(booth_number,1) OVER (PARTITION BY vendor_id ORDER BY market_
           date, vendor_id) AS previous_booth_number
FROM farmers_market.vendor_booth_assignments
ORDER BY market_date, vendor_id, booth_number
  ) x WHERE x.market_date = '2019-04-10'
          AND (x.booth_number <> x.previous_booth_number OR x.previous_
  booth_number IS NULL)
```

Reference

Window functions.