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

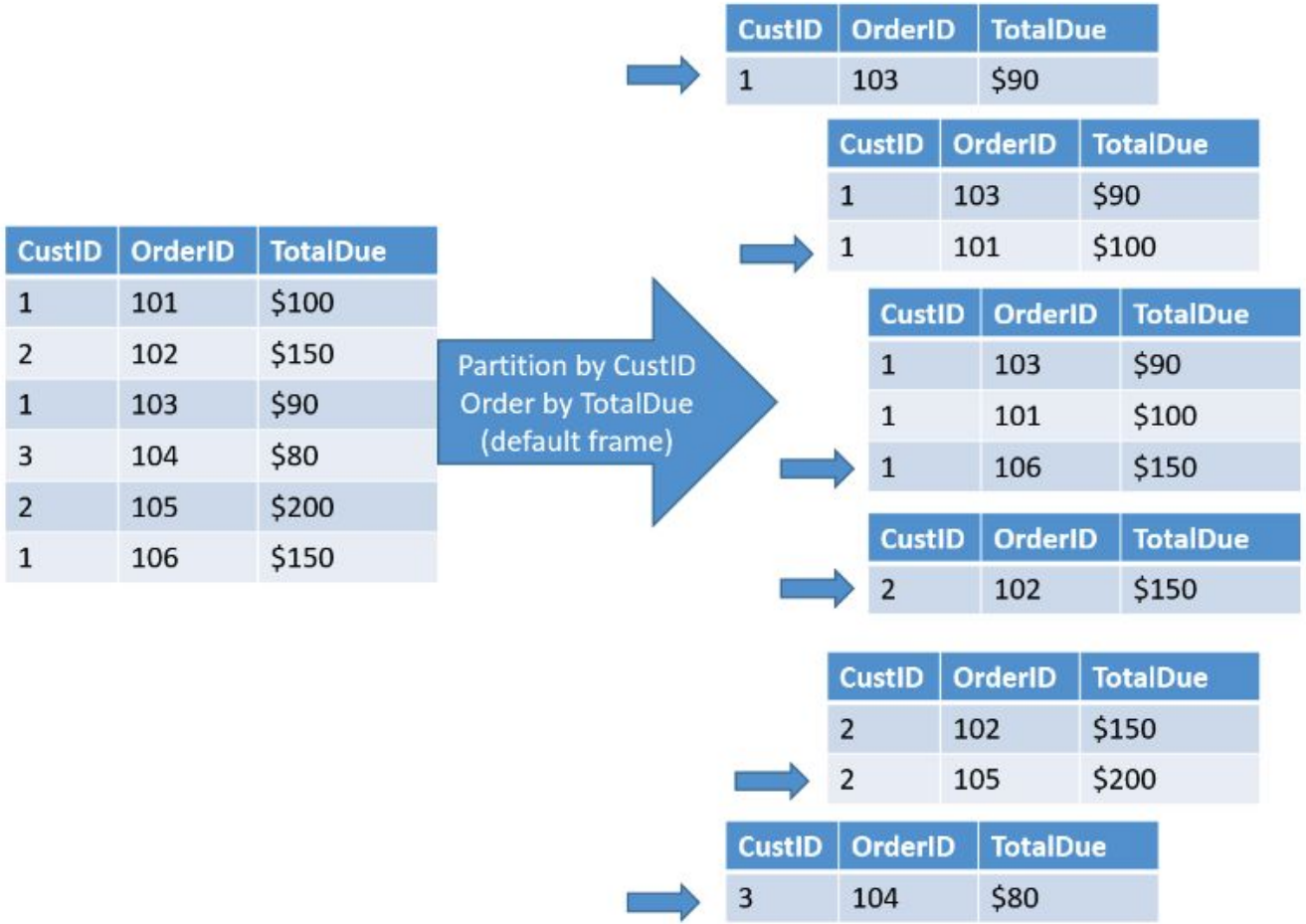
CustID	OrderID	TotalDue
3	104	$80

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
```

All query elements are processed in a very strict order:

- **FROM** - the database gets the data from tables in FROM clause and if necessary performs the JOINs,
- **WHERE** - the data are filtered with conditions specified in WHERE clause,
- **GROUP BY** - the data are grouped by with conditions specified in WHERE clause,
- **Aggregate functions** - the aggregate functions are applied to the groups created in the GROUP BY phase,
- **HAVING** - the groups are filtered with the given condition,
- **Window functions,**
- **SELECT** - the database selects the given columns,
- **DISTINCT** - repeated values are removed,
- **UNION/INTERSECT/EXCEPT** - the database applies set operations,
- **ORDER BY** - the results are sorted,
- **OFFSET** - the first rows are skipped,
- **LIMIT/FETCH/TOP** - only the first rows are selected

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

Extract the farmer’s products that have prices above the market date’s average product cost only for vendor having id 8

- 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.

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***

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.

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
```

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

Let’s say you want to find out if the total sales on each market date are higher or lower than they were on the previous market date.

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

```
SELECT * FROM
(SELECT
        market_date,
    vendor_id,
    booth_number,
    LAG(booth_number, 1) OVER (partition by vendor_id order by
market_date) AS prev_booth
FROM farmers_market.vendor_booth_assignments) AS x
WHERE x.market_date = "2019-04-10"
        AND
    (x.booth_number <> x.prev_booth OR x.prev_booth IS NULL)
```

Reference

Window functions.

Question: From each market_start_datetime, extract the following:

- day of week,
- month of year,
- year,
- hour and
- minute from the timestamp

Question: From each market_start_datetime, extract the following:

- day of week,
- month of year,
- year,
- hour and
- minute from the timestamp

```
SELECT
        market_start_datetime,
    EXTRACT(DAY FROM
market_start_datetime) AS start_day,
    EXTRACT(YEAR FROM
market_start_datetime) AS date_year,
    EXTRACT(MONTH FROM
market_start_datetime) AS month_of_year,
    EXTRACT(HOUR FROM
market_start_datetime) AS hour_of_day,
    EXTRACT(MINUTE FROM
market_start_datetime) AS minute_of_time
FROM farmers_market.datetime_demo;
```

Question: Let’s say you want to calculate how many sales occurred within the first 30 minutes after the farmer’s market opened, how would you dynamically determine what cutoff time to use?

Question: Let's say you want to calculate how many sales occurred within the first 30 minutes after the farmer's market opened, how would you dynamically determine what cutoff time to use?

```
SELECT
        market_start_datetime,
    DATE_ADD(market_start_datetime, INTERVAL 30
DAY) AS mkt_plus_30mins
FROM farmers_market.datetime_demo;
```

Question: Let’s say we wanted to get a profile of each farmer’s market customer’s habits over time.

Question: Let’s say we wanted to get a profile of each farmer’s market customer’s habits over time.

```
SELECT customer_id,
    MIN(market_date) AS first_purchase,
    MAX(market_date) AS last_purchase,
    COUNT(DISTINCT market_date) AS
count_of_purchase_dates,
    DATEDIFF(MAX(market_date),
MIN(market_date)) AS
days_between_first_
 last_purchase
 FROM
farmers_market.customer_purchases
 GROUP BY customer_id
```

Question: Let’s say we wanted to get a profile of how long it’s been since the customer last made a purchase,

Question: Let’s say we wanted to get a profile of how long it’s been since the customer last made a purchase,

```
SELECT customer_id,
    MIN(market_date) AS first_purchase,
    MAX(market_date) AS last_purchase,
    COUNT(DISTINCT market_date) AS count_of_purchase_dates,
    DATEDIFF(MAX(market_date), MIN(market_date)) AS days_between_first_
last_purchase,
    DATEDIFF(CURDATE(), MAX(market_date)) AS days_since_last_purchase
FROM farmers_market.customer_purchases
GROUP BY customer_id
```

Question: Write a query that gives us the days between each purchase a customer makes.

Question: Write a query that gives us the days between each purchase a customer makes.

```
SELECT
    x.customer_id,
    x.market_date,
    RANK() OVER (PARTITION BY x.customer_id ORDER BY x.market_date)
        AS purchase_number,
    LEAD(x.market_date,1) OVER (PARTITION BY x.customer_id ORDER BY x.market_date) AS next_purchase,
    DATEDIFF(
        LEAD(x.market_date,1) OVER
        (PARTITION BY x.customer_id ORDER BY x.market_date),
        x.market_date
        ) AS days_between_purchases

FROM (
    SELECT DISTINCT customer_id, market_date
    FROM farmers_market.customer_purchases
    WHERE customer_id = 1
) AS x
```

Question: today’s date is May 31, 2019, and the marketing director of the farmer’s market wants to give infrequent customers an incentive to return to the market in April.

Question: today's date is May 31, 2019, and the marketing director of the farmer's market wants to give infrequent customers an incentive to return to the market in April.

```
SELECT x.customer_id,
    COUNT(DISTINCT x.market_date) AS market_count
FROM (
    SELECT DISTINCT customer_id, market_date
    FROM farmers_market.customer_purchases
    WHERE DATEDIFF(market_date, '2019-03-31') <= 31
)x
GROUP BY x.customer_id
HAVING COUNT(DISTINCT market_date) = 1
```