SQL-6 Window functions

Lecture 10

Order of execution

- FROM, including JOINs
- WHERE
- GROUP BY
- HAVING
- WINDOW functions
- SELECT
- DISTINCT
- UNION
- ORDER BY
- OFFSET
- LIMIT

Quiz

- Consider a transaction table. To Find customers who have made an average sale of more than $200? Which clause to use?
  - Having - correct
  - Where
- In a transaction table, consider every rows as a sale value per transaction. We want to analyze only sale value per transaction above $200? Which clause to use?
  - Having
  - Where – correct
- Which is the right query to get overall sales per customer in India?
  - Select cus_id, sum(sales) from DB.tbl having country = ‘India’ sales group by cus _id
  - Select cus_id, sum(sales) from DB.tbl where country=‘India’ group by cus _id - correct
  - Select country, cus _id, sum(sales) from DB.tbl group by cus _id, market_date
- Right Order of execution for these 6 clauses
  - SELECT>window function>FROM>WHERE>GroupBy>HAVING
  - FROM>WHERE > GroupBy>HAVING>window functions>SELECT - correct
  - FROM>WHERE > GroupBy>window functions> HAVING>SELECT
- When grouping, We can have a column B in the select statement which is not present in the GROUP BY
  - True
  - False - correct

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.

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

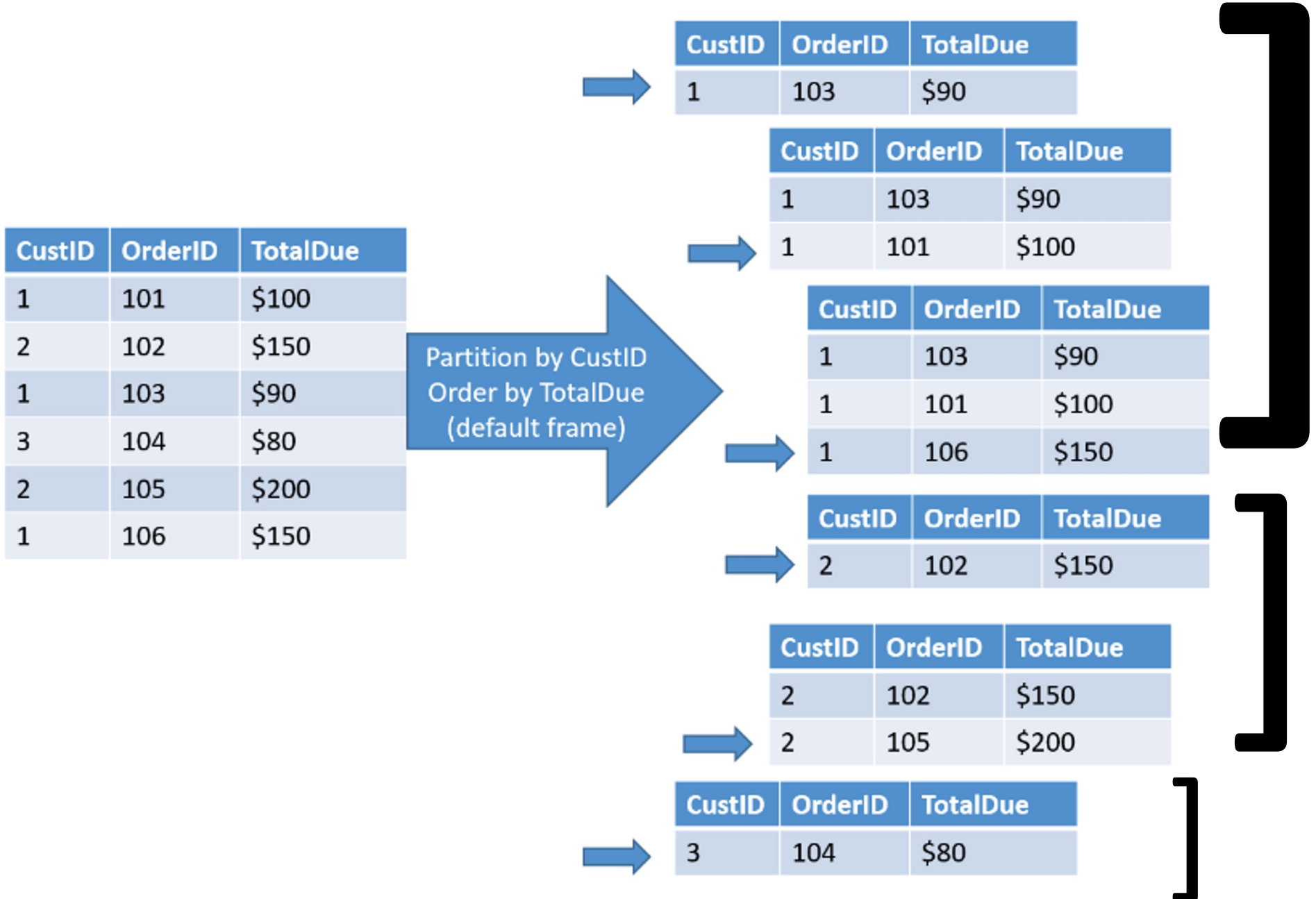
CustID	OrderID	TotalDue
3	104	$80

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

```
SELECT
        vendor_id,
        MAX(original_price) AS highest_price
FROM farmers_market.vendor_inventory
GROUP BY vendor_id
```

New Question: Rank the products in each vendor's inventory. Expensive products get a lower rank.

```
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
```

Get me all the products per vendor that have a price rank of 1.

```
SELECT * FROM
  (
     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) x
  WHERE x.price_rank = 1;
```

Question: The record of the highest-priced item per vendor

i.e Get me all the products per vendor that have a price rank of 1.

```
SELECT * FROM
 (
    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) x
 WHERE x.price_rank = 1;
```

Since row_number() does not logically look at the price, Use Dense_rank instead

```
SELECT * FROM
 (
    SELECT
        vendor_id,
        market_date,
        product_id,
        original_price,
        ROW_NUMBER() OVER (PARTITION BY vendor_id ORDER BY original_price DESC) AS price_rank_row_num,
        RANK() OVER (PARTITION BY vendor_id ORDER BY original_price DESC) AS price_rank_,
        DENSE_RANK() OVER (PARTITION BY vendor_id ORDER BY original_price DESC) AS price_dense_rank
FROM farmers_market.vendor_inventory ORDER BY vendor_id) x
 WHERE x.price_dense_rank = 1;
```

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;
```

Follow-up Question: Extract the farmer's products with prices above the market date's average product cost for vendor id 8?.

```
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 display that count on each row.

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