Acknowledgements: I am indebted to M. Böhlen for providing me the lecture notes.
Outline

1 SQL Analytic/Window Functions
2 Ranking and Percentiles
3 Nested Aggregates
4 Moving Windows
5 Densification
Outline

1. SQL Analytic/Window Functions
2. Ranking and Percentiles
3. Nested Aggregates
4. Moving Windows
5. Densification
ISO SQL:2003 has enhanced SQL’s analytical processing capabilities by introducing so-called **window/analytic functions**.

All major database systems (have to) support window functions.

These window functions permit things such as:

- Rankings and percentiles: cumulative distributions, percent rank, and N-tiles.
- Reporting aggregate functions (nested aggregations, moving averages)
- Lag and Lead
- Data densification
- Linear regression
Basic syntax:
\[ \text{WFctType(expr) OVER (WPartitioning WOrdering Wframe)} \]

Window functions may only be used in the `SELECT` (or `ORDERING`) clause.

Query processing takes place in three stages:

1. FROM, WHERE, GROUP BY, and HAVING clauses
2. Creation of partitions: Analytic functions are applied to each row in each partition
3. Final ORDER BY

Result set of first step is made available to window/analytic functions.
Window Functions/3

- Basic syntax:
  \[ \text{WFctType(expr) OVER (WPartitioning WOrdering Wframe)} \]

- **WPartitioning**
  - Divides the table into partitions, i.e., groups of rows.
  - Division can be based upon any number of columns or expressions (e.g., aggregates).
  - Partitions are created after groupings (GROUP BY and its extensions) and aggregations, and can therefore refer to any aggregate results.

- **WOrdering**
  - Determines the ordering in which the rows are passed to the window function.
  - Many window functions are sensitive to the ordering of rows.
Window Functions/4

- **Basic syntax:**
  
  $WFctType(expr) \text{ OVER (WPartitioning WOrdering Wframe)}$

- **WFrame**
  - Each calculation with an window function is based on the current row.
  - For each (current) row within a partition, a sliding frame of data can optionally be defined.
  - The window frame determines the rows that are used to calculate values for the current row.
  - Windows can be defined as
    - physical number of rows or
    - logical range of rows.
Outline

1. SQL Analytic/Window Functions
2. Ranking and Percentiles
3. Nested Aggregates
4. Moving Windows
5. Densification
A **ranking function** computes the rank of a record (row) compared to other records in the data set based on the values of a set of measures.

The following ranking functions are available:

- `RANK()` OVER ( [WPartitioning] WOrdering )
- `DENSE_RANK()` OVER ( [WPartitioning] WOrdering )
- `CUME_DIST()` OVER ( [WPartitioning] WOrdering )
- `PERCENT_RANK()` OVER ( [WPartitioning] WOrdering )
- `NTILE(expr)` OVER ( [WPartitioning] WOrdering )
- `ROW_NUMBER()` OVER ( [WPartitioning] WOrdering )
SELECT m_desc,  
    SUM(s_amount_sold),  
    RANK() OVER (ORDER BY SUM(s_amount_sold))  
FROM tpch  
WHERE m_desc IN ('Direct Sales', 'Internet')  
AND t_cal_month_desc IN ('2000-09', '2000-10')  
AND n_iso_code = 'US'  
GROUP BY m_desc;

- **RANK()** assigns the rank to each row according to the order of the total amount sold.
- The ordering attribute or expression must be specified.
- The ordering can be ASC (default) or DESC.
### RANK Example/2

```sql
SELECT m_desc,
       SUM(s_amount_sold),
       RANK() OVER (ORDER BY SUM(s_amount_sold))
FROM tcph
WHERE m_desc IN ('Direct Sales', 'Internet', 'Partners')
AND t_cal_month_desc IN ('2000-09', '2000-10')
AND n_iso_code = 'US'
GROUP BY m_desc;
```

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>SUM(S_AMNT_SOLD)</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>261278.04</td>
<td>1</td>
</tr>
<tr>
<td>Partners</td>
<td>800871.37</td>
<td>2</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>1320497.4</td>
<td>3</td>
</tr>
</tbody>
</table>

Note that the aggregate result `SUM(s_amount_sold)` need not necessarily be reported.
RANK with Partitioning Example/1

SELECT  m_desc,  
t_cal_month_desc,  
SUM(s_amnt_sold),  
RANK() OVER ( PARTITION BY m_desc  
          ORDER BY SUM(s_amnt_sold) DESC)  
) AS RankByMedia  
FROM  tpch  
WHERE  t_cal_month_desc IN  
AND  m_desc IN ('Direct Sales', 'Internet')  
GROUP BY m_desc, t_cal_month_desc;

- If a PARTITION BY clause is specified, the rank is computed independently for each group specified by the partitioning,
- i.e., the rank is reset for each group.
- Partitions are created on top of the groups produced by the GROUP BY clause.
<table>
<thead>
<tr>
<th>Media</th>
<th>Month</th>
<th>AmntSold</th>
<th>RankByMedia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>2000-08</td>
<td>1236104.31</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>1225584.31</td>
<td>2</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>1217807.75</td>
<td>3</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-11</td>
<td>1115239.03</td>
<td>4</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-11</td>
<td>284741.77</td>
<td>1</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>239236.26</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>228241.24</td>
<td>3</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-08</td>
<td>215106.56</td>
<td>4</td>
</tr>
</tbody>
</table>
Multiple RANK Functions

- A query block can contain more than one ranking function, each partitioning the data into different groups.

**Example:** Rank products based on their dollar sales within each month and within each channel.

```
RANK() OVER ( PARTITION BY m_desc
            ORDER BY SUM(amount_sold) ) AS RankByMedia,
RANK() OVER ( PARTITION BY cal_month_desc
            ORDER BY SUM(amount_sold) ) AS RankByMonth
```

<table>
<thead>
<tr>
<th>Media</th>
<th>Month</th>
<th>AmntSold</th>
<th>RankByMedia</th>
<th>RankByMonth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>2000-08</td>
<td>1236104.31</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>1225584.31</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>1217807.75</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-11</td>
<td>1115239.03</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-11</td>
<td>284741.77</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>239236.26</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>228241.24</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-08</td>
<td>215106.56</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
DENSE_RANK leaves no gaps in the ranking sequence when there are ties.

**Example:** Rank and dense rank of amount sold.

```sql
SELECT ...
    RANK() OVER ( ORDER BY SUM(amount_sold) ) AS Rank,
    DENSE_RANK() OVER ( ORDER BY SUM(amount_sold) ) AS Dense_Rank
FROM ...
```

<table>
<thead>
<tr>
<th>Media</th>
<th>Month</th>
<th>AmntSold</th>
<th>Rank</th>
<th>Dense_Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>1200000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>1200000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Partners</td>
<td>2000-09</td>
<td>600000</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Partners</td>
<td>2000-10</td>
<td>600000</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>200000</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>200000</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
Rank the media (‘Internet’ versus ‘Direct sales’) used for selling products according to their dollar sales. Use the number of unit sales to break ties. Do the analysis for August until November 2000.

```
SELECT m_desc,
t_cal_month_desc,
SUM(s_amnt_sold),
SUM(s_quantity_sold),
RANK() OVER ( ORDER BY SUM(s_amnt_sold) DESC,
              SUM(s_quantity_sold) DESC)
  ) AS Rank
FROM tpch
WHERE m_desc IN (‘Direct Sales’, ‘Internet’),
AND t_cal_month_desc IN
GROUP BY m_desc, t_cal_month_desc;
```
## Ranking Examples/2

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>T_CAL_MO</th>
<th>SUM(S_AMNT_SOLD)</th>
<th>SUM(S_QUANTITY_SOLD)</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>2000-08</td>
<td>1236104.31</td>
<td>12230</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>1225584.31</td>
<td>12584</td>
<td>2</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>1217807.75</td>
<td>11995</td>
<td>3</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-11</td>
<td>1115239.03</td>
<td>11357</td>
<td>4</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-11</td>
<td>284741.77</td>
<td>1913</td>
<td>5</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>239236.26</td>
<td>1450</td>
<td>6</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>228241.24</td>
<td>1887</td>
<td>7</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-08</td>
<td>215106.56</td>
<td>1132</td>
<td>8</td>
</tr>
</tbody>
</table>
### Ranking Examples/3

Determine the two least and the two most successful sales media, respectively (in terms of total amount sold).

```sql
SELECT *
FROM ( SELECT m_desc,
            SUM(s_amnt_sold),
            RANK() OVER ( ORDER BY SUM(s_amnt_sold) ) worst,
            RANK() OVER ( ORDER BY SUM(s_amnt_sold) DESC ) best
      FROM tpch
      GROUP BY m_desc
    )
WHERE worst < 3 OR best < 3;
```

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>SUM(S_AMNT_SOLD)</th>
<th>Worst</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>57875260</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Partners</td>
<td>26346342</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>13706802</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Tele Sales</td>
<td>277426</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Rank sales per media and country, per media, and per country, respectively. Consider US, JP, and DK during September 2000.

```
SELECT m_desc,
n_iso_code,
SUM(s_amnt_sold),
RANK() OVER (PARTITION BY GROUPING_ID(m_desc,n_iso_code)
ORDER BY SUM(s_amnt_sold) DESC
) AS RANK_PER_GROUP
FROM tpch
WHERE t_cal_month = '2000-09'
AND n_iso_code IN ('DK', 'US', 'JP')
GROUP BY CUBE(m_desc, n_iso_code)
HAVING GROUPING_ID(m_desc, n_iso_code) <> 3
ORDER BY GROUPING_ID(m_desc, n_iso_code);
```
## Ranking Examples/5

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>N_</th>
<th>SUM(S_AMNT_SOLD)</th>
<th>RANK_PER_GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>US</td>
<td>638200.81</td>
<td>1</td>
</tr>
<tr>
<td>Partners</td>
<td>US</td>
<td>376813.18</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>US</td>
<td>124223.75</td>
<td>3</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>JP</td>
<td>81073.81</td>
<td>4</td>
</tr>
<tr>
<td>Partners</td>
<td>JP</td>
<td>43347.12</td>
<td>5</td>
</tr>
<tr>
<td>Internet</td>
<td>JP</td>
<td>23862.29</td>
<td>6</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>DK</td>
<td>17640.33</td>
<td>7</td>
</tr>
<tr>
<td>Partners</td>
<td>DK</td>
<td>16561.62</td>
<td>8</td>
</tr>
<tr>
<td>Internet</td>
<td>DK</td>
<td>2060.56</td>
<td>9</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>US</td>
<td>736914.95</td>
<td>1</td>
</tr>
<tr>
<td>Partners</td>
<td></td>
<td>436721.92</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td>150146.6</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>US</td>
<td>1139237.74</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>JP</td>
<td>148283.22</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>DK</td>
<td>36262.51</td>
<td>3</td>
</tr>
</tbody>
</table>
Determine the output of the following statement:

```sql
SELECT c.id, p.id,
       RANK() OVER (ORDER BY p.id) AS r1,
       RANK() OVER (ORDER BY c.id) AS r2,
       RANK() OVER (ORDER BY 1) AS r3,
       RANK() OVER (PARTITION BY c.id ORDER BY p.id) AS r4,
       RANK() OVER (PARTITION BY p.id ORDER BY c.id) AS r5
FROM tpch
WHERE c.id in (214, 608, 699)
AND p.id in (42, 98, 123)
GROUP BY c.id, p.id;
```

<table>
<thead>
<tr>
<th>C_ID</th>
<th>P_ID</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
</tr>
</thead>
<tbody>
<tr>
<td>214</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>608</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>608</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>699</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>699</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**CUME_DIST Example/1**

```sql
SELECT t_cal_month_desc AS MONTH,
       m_desc,
       SUM(s_amnt_sold),
       CUME_DIST() OVER ( PARTITION BY t_cal_month_desc
                          ORDER BY SUM(s_amnt_sold)
                      ) AS CUME_DIST
FROM tpch
WHERE t_cal_month_desc IN ('2000-09', '2000-07', '2000-08')
GROUP BY t_cal_month_desc, m_desc;
```

- **CUME_DIST()** (cumulative distribution) computes the position of a value relative to a set of values, i.e.,
  - \( \text{CUME\_DIST}(x) = \frac{\text{# of values smaller or equal to } x}{\text{total \# of values}} \).

- **PERCENT_RANK()** is similar, but uses rank values rather than row counts in the denominator, i.e.,
  - \( \text{PERCENT\_RANK}() = \frac{\text{rank of row} - 1}{\text{# of rows} - 1} \).
  - Row with rank 1 has percent rank 0.
## CUME_DIST Example/2

<table>
<thead>
<tr>
<th>MONTH</th>
<th>M_DESC</th>
<th>SUM(S_AMNT)</th>
<th>CUME_DIST</th>
<th>(PERCENT_RANK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000-07</td>
<td>Internet</td>
<td>140423.34</td>
<td>.3333333333333</td>
<td>0.0</td>
</tr>
<tr>
<td>2000-07</td>
<td>Partners</td>
<td>611064.35</td>
<td>.666666666666666</td>
<td>0.5</td>
</tr>
<tr>
<td>2000-07</td>
<td>Direct Sales</td>
<td>1145275.13</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2000-08</td>
<td>Internet</td>
<td>215106.56</td>
<td>.3333333333333</td>
<td>0.0</td>
</tr>
<tr>
<td>2000-08</td>
<td>Partners</td>
<td>661044.92</td>
<td>.666666666666666</td>
<td>0.5</td>
</tr>
<tr>
<td>2000-08</td>
<td>Direct Sales</td>
<td>1236104.31</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2000-09</td>
<td>Internet</td>
<td>228241.24</td>
<td>.3333333333333</td>
<td>0.0</td>
</tr>
<tr>
<td>2000-09</td>
<td>Partners</td>
<td>666171.69</td>
<td>.666666666666666</td>
<td>0.5</td>
</tr>
<tr>
<td>2000-09</td>
<td>Direct Sales</td>
<td>1217807.75</td>
<td>1</td>
<td>1.0</td>
</tr>
</tbody>
</table>
NTILE Example/1

```
SELECT t_cal_month_desc AS MONTH,
       SUM(s_amnt_sold),
       NTILE(4) OVER (ORDER BY SUM(s_amnt_sold)) AS TILE4
FROM   tpch
WHERE  t_cal_year=2000
AND    p_cat = 'Electronics'
GROUP BY t_cal_month_desc;
```

- **NTILE(n)** divides an ordered partition into \( n \) equal sized **buckets** and assigns to each bucket a number.
- Each bucket shall contain the same number of rows.
- If the rows cannot be distributed evenly, the highest buckets have one row less.
## NTILE Example/2

<table>
<thead>
<tr>
<th>MONTH</th>
<th>SUM(S_AMNT_SOLD)</th>
<th>TILE4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000-02</td>
<td>242416.38</td>
<td>1</td>
</tr>
<tr>
<td>2000-01</td>
<td>257285.89</td>
<td>1</td>
</tr>
<tr>
<td>2000-03</td>
<td>280010.94</td>
<td>1</td>
</tr>
<tr>
<td>2000-06</td>
<td>315950.95</td>
<td>2</td>
</tr>
<tr>
<td>2000-05</td>
<td>316824.18</td>
<td>2</td>
</tr>
<tr>
<td>2000-04</td>
<td>318105.67</td>
<td>2</td>
</tr>
<tr>
<td>2000-07</td>
<td>433823.77</td>
<td>3</td>
</tr>
<tr>
<td>2000-08</td>
<td>477833.26</td>
<td>3</td>
</tr>
<tr>
<td>2000-12</td>
<td>553534.39</td>
<td>3</td>
</tr>
<tr>
<td>2000-10</td>
<td>652224.76</td>
<td>4</td>
</tr>
<tr>
<td>2000-11</td>
<td>661146.75</td>
<td>4</td>
</tr>
<tr>
<td>2000-09</td>
<td>691448.94</td>
<td>4</td>
</tr>
</tbody>
</table>
SELECT m_desc, 
    t_cal_month_desc, 
    SUM(s_amnt_sold), 
    ROW_NUMBER() OVER ( ORDER BY SUM(s_amnt_sold) DESC ) 
    AS Row_Number
FROM tpch 
WHERE t_cal_month_desc IN ('2001-09', '2001-10') 
GROUP BY m_desc, t_cal_month_desc;

**ROW_NUMBER** assigns a unique number (sequentially, starting from 1, as defined by ORDER BY) to each row within the partition.
### ROW_NUMBER Example/2

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>MONTH</th>
<th>SUM(S_AMNT_SOLD)</th>
<th>ROW_NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>2001-09</td>
<td>1100000</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2001-10</td>
<td>1000000</td>
<td>2</td>
</tr>
<tr>
<td>Internet</td>
<td>2001-09</td>
<td>500000</td>
<td>3</td>
</tr>
<tr>
<td>Internet</td>
<td>2001-10</td>
<td>700000</td>
<td>4</td>
</tr>
<tr>
<td>Partners</td>
<td>2001-09</td>
<td>600000</td>
<td>5</td>
</tr>
<tr>
<td>Partners</td>
<td>2001-10</td>
<td>600000</td>
<td>6</td>
</tr>
</tbody>
</table>

- **Ties** can be reported in any order (see last 2 rows)
- Use additional column(s) in ORDER BY clause to break ties.
Outline

1. SQL Analytic/Window Functions
2. Ranking and Percentiles
3. Nested Aggregates
4. Moving Windows
5. Densification
Nested Aggregates

- After a query has been processed (FROM, WHERE, GROUP BY, HAVING), aggregate values like the number of rows or an average value or sum in a column can be made available to window functions.
- This yields **nested aggregations**, which are frequently used in analytic aggregate functions.
- Nested aggregate functions return the same value for each row in a window.
- For example, **reporting functions** often relate partial totals to grand totals, etc.
  - They are based on nested aggregations.
- The **RATIO_TO_REPORT** function computes the ratio of a value to the sum of a set of values.
RATIO_TO_REPORT Example

For each media, compute the total amount sold and the ratio wrt the overall total amount sold (across all media) for October 11, 2000.

```
SELECT m_desc,
    SUM(s_amnt_sold) AS SALES,
    SUM(SUM(s_amnt_sold)) OVER () AS TOTAL_SALES,
    RATIO_TO_REPORT(SUM(s_amnt_sold)) OVER () AS RATIO
FROM tpch
WHERE s_t_id = TO_DATE('11-OCT-2000')
GROUP BY m_desc;
```

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>SALES</th>
<th>TOTAL_SALES</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>14447.23</td>
<td>23183.45</td>
<td>.623169977</td>
</tr>
<tr>
<td>Internet</td>
<td>345.02</td>
<td>23183.45</td>
<td>.014882168</td>
</tr>
<tr>
<td>Partners</td>
<td>8391.2</td>
<td>23183.45</td>
<td>.361947855</td>
</tr>
</tbody>
</table>
Outline

1. SQL Analytic/Window Functions
2. Ranking and Percentiles
3. Nested Aggregates
4. Moving Windows
5. Densification
Window Frame

- **Syntax:** `WFctType(expr) OVER (WPartitioning WOrdering Wframe)`

- **Window frames** are used to compute cumulative, moving and centered aggregates.

- Window frames return a value for each row that depends on the other rows in the window.

- Window frames provide access to more than one row **without a self join**.

- **FIRST_VALUE** and **LAST_VALUE** return the first and last value of the window, respectively.
Examples of Window Frame Specifications

- **ROWS UNBOUNDED PRECEDING**
  - Takes all rows in the window/partition up to and including the current row.

- **ROWS 2 PRECEDING**
  - Takes the 2 preceding rows.

- **RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND INTERVAL '1' DAY FOLLOWING**
  - Takes all rows that fall within the given logical offset (wrt the expression in the ORDERING clause).
  - In this example rows with a timestamp that differs by at most 1 day.

- **RANGE BETWEEN INTERVAL '10' DAY PRECEDING AND CURRENT ROW**
  - Takes all rows with a timestamp that is at most 10 days before the timestamp of the current row.
The centered 3 day moving average of all sales during week 51 in 1999.

```sql
SELECT t_id,
  SUM(s_amnt_sold) AS SALES,
  AVG(SUM(s_amnt_sold))
  OVER ( ORDER BY t_id
    RANGE BETWEEN INTERVAL '1' DAY PRECEDING
    AND INTERVAL '1' DAY FOLLOWING
  )
FROM tpch
WHERE t_cal_week_num = 51
AND t_cal_year = 1999
GROUP BY t_id
ORDER BY t_id;
```

- Notice the use of nested aggregates.
## Centered Aggregate Example/2

<table>
<thead>
<tr>
<th>T_ID</th>
<th>SALES</th>
<th>CENTERED_3_DAY_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-DEC-99</td>
<td>134336.84</td>
<td>106675.93</td>
</tr>
<tr>
<td>21-DEC-99</td>
<td>79015.02</td>
<td>102538.713</td>
</tr>
<tr>
<td>22-DEC-99</td>
<td>94264.28</td>
<td>85341.7533</td>
</tr>
<tr>
<td>23-DEC-99</td>
<td>82745.96</td>
<td>93322.3067</td>
</tr>
<tr>
<td>24-DEC-99</td>
<td>102956.68</td>
<td>82936.7</td>
</tr>
<tr>
<td>25-DEC-99</td>
<td>63107.46</td>
<td>87062.2167</td>
</tr>
<tr>
<td>26-DEC-99</td>
<td>95122.51</td>
<td>79114.985</td>
</tr>
</tbody>
</table>

- The window frame in the first and last row contains only two rows.
  - e.g., \((63107.46 + 95122.51)/2 = 79114.985\)
Rewrite the following statement to a semantically equivalent one that does not use the RANK function.

```
SELECT m_desc,
       t_cal_month_desc,
       RANK() OVER ( ORDER BY SUM(s_amnt_sold) DESC ) AS rank
FROM tpch
WHERE t_cal_month_desc
       IN ('200008', '200009', '200010', '200011')
AND m_desc IN ('Direct sales', 'Internet')
GROUP BY m_desc, t_cal_month_desc;
```

- Consider also the case that several rows might have the same rank!
- Hint: Rank of a row is the number of rows with equal or larger values — number of rows with the same value + 1


**Ranking Example/2**

```sql
SELECT m_desc, t_cal_month_desc,
       ( COUNT(*) OVER ( ORDER BY SUM(s_amnt_sold) DESC ROWS UNBOUNDED PRECEDING )
         - COUNT(*) OVER ( ORDER BY SUM(s_amnt_sold) DESC ROWS CURRENT ROW )
       ) + 1 AS Rank
FROM tpch
WHERE t_cal_month_desc IN ('200008', '200009', '200010', '200011')
AND m_desc IN ('Direct sales', 'Internet')
GROUP BY m_desc, t_cal_month_desc;
```

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>T_CAL_MO</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>2000-08</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>2</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>3</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-11</td>
<td>4</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-11</td>
<td>5</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>6</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>7</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-08</td>
<td>8</td>
</tr>
</tbody>
</table>
LAG and LEAD functions give access to rows that are at a certain distance from the current row.

- LAG(): row at a given offset prior to the current position.
- LEAD(): row at a given offset after the current position.

**Example:** Report with amounts sold between 10.8.2000 and 14.8.2000. Include with each row the amount of the previous and the following day.

```
SELECT s.t_id,
       SUM(s.amnt_sold),
       LAG(SUM(s.amnt_sold),1) OVER (ORDER BY s.t_id),
       LEAD(SUM(s.amnt_sold),1) OVER (ORDER BY s.t_id)
FROM tpch
WHERE s.t_id >= TO_DATE('10-OCT-2000')
AND    s.t_id <= TO_DATE('14-OCT-2000')
GROUP BY s.t_id;
```
### LAG and LEAD Example/2

<table>
<thead>
<tr>
<th>ST_ID</th>
<th>SUM(S_AMNT)</th>
<th>LAG1</th>
<th>LEAD1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-OCT-00</td>
<td>238479.49</td>
<td></td>
<td>23183.45</td>
</tr>
<tr>
<td>11-OCT-00</td>
<td>23183.45</td>
<td>238479.49</td>
<td>24616.04</td>
</tr>
<tr>
<td>12-OCT-00</td>
<td>24616.04</td>
<td>23183.45</td>
<td>76515.61</td>
</tr>
<tr>
<td>13-OCT-00</td>
<td>76515.61</td>
<td>24616.04</td>
<td>29794.78</td>
</tr>
<tr>
<td>14-OCT-00</td>
<td>29794.78</td>
<td>76515.61</td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. SQL Analytic/Window Functions
2. Ranking and Percentiles
3. Nested Aggregates
4. Moving Windows
5. Densification
Densification

Example of sparse data set, i.e., some combinations of Prod, Year, Week do not have any values.

<table>
<thead>
<tr>
<th>PROD</th>
<th>YEAR</th>
<th>WEEK</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deluxe</td>
<td>2001</td>
<td>25</td>
<td>5560</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2001</td>
<td>24</td>
<td>2083</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2001</td>
<td>26</td>
<td>2501</td>
</tr>
<tr>
<td>Standar</td>
<td>2001</td>
<td>24</td>
<td>2394</td>
</tr>
<tr>
<td>Standar</td>
<td>2001</td>
<td>26</td>
<td>1280</td>
</tr>
</tbody>
</table>

Can be important for reports or subsequent aggregations (3 months average), time series analysis, etc.
Data is often stored in sparse form, e.g., in relational tables.
  * e.g., if no value exists for a given combination of dimension values, no row exists in the fact table.

For reporting or analysis purposes, it can make sense to selectively densify data.

**Data densification** is the process of converting sparse data into dense form.

The key technique is a *partitioned outer join*.

A partitioned outer join extends the regular outer join by applying the outer join to each partition.

This allows to fill in values for the partitioned attributes.
SELECT  p_Name,  
t.Year,  
t.Week,  
NVL(Sales,0) AS dense_sales  
FROM  ( SELECT  P_Name, T_Cal_Year Year, t_Cal_Week_num AS Week,  
    SUM(S_Amnt_Sold) AS Sales  
    FROM  tpch  
    GROUP BY  p_Name, T_Cal_Year, t_Cal_Week_num  
) AS v  
PARTITION BY (v.p_Name)  
RIGHT OUTER JOIN  
( SELECT  DISTINCT t_Cal_Week_num Week, T_Cal_Year AS Year  
    FROM  tpch  
    WHERE  T_Cal_Year IN (2000, 2001)  
    AND  t_Cal_Week_num BETWEEN 24 AND 26  
) AS t  
ON (v.week = t.week AND v.Year = t.Year)  
ORDER BY  p_name, year, week;
## Densification Example/2

<table>
<thead>
<tr>
<th>PROD</th>
<th>YEAR</th>
<th>WEEK</th>
<th>DENSE SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deluxe</td>
<td>2000</td>
<td>24</td>
<td>0.0</td>
</tr>
<tr>
<td>Deluxe</td>
<td>2000</td>
<td>25</td>
<td>0.0</td>
</tr>
<tr>
<td>Deluxe</td>
<td>2000</td>
<td>26</td>
<td>0.0</td>
</tr>
<tr>
<td>Deluxe</td>
<td>2001</td>
<td>24</td>
<td>2260.72</td>
</tr>
<tr>
<td>Deluxe</td>
<td>2001</td>
<td>25</td>
<td>1871.3</td>
</tr>
<tr>
<td>Deluxe</td>
<td>2001</td>
<td>26</td>
<td>5560.51</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2000</td>
<td>24</td>
<td>1685.52</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2000</td>
<td>25</td>
<td>494.91</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2000</td>
<td>26</td>
<td>1548.2</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2001</td>
<td>24</td>
<td>2083.29</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2001</td>
<td>25</td>
<td>0.0</td>
</tr>
<tr>
<td>Mouse P</td>
<td>2001</td>
<td>26</td>
<td>2501.79</td>
</tr>
<tr>
<td>Standar</td>
<td>2000</td>
<td>24</td>
<td>1007.37</td>
</tr>
<tr>
<td>Standar</td>
<td>2000</td>
<td>25</td>
<td>339.36</td>
</tr>
<tr>
<td>Standar</td>
<td>2000</td>
<td>26</td>
<td>183.92</td>
</tr>
<tr>
<td>Standar</td>
<td>2001</td>
<td>24</td>
<td>2394.04</td>
</tr>
<tr>
<td>Standar</td>
<td>2001</td>
<td>25</td>
<td>0.0</td>
</tr>
<tr>
<td>Standar</td>
<td>2001</td>
<td>26</td>
<td>1280.97</td>
</tr>
</tbody>
</table>
Use the reporting functions to determine the answers to the following queries:

1. Media that contributed with more than $\frac{1}{3}$ to the total sales. Formulate with and without analytic functions.

2. For customer 6510 determine the 3 month moving average of sales (current month plus preceding two months) in 1999.

3. For each product category find the region in which it had maximum sales on Oct 11, 2001.

4. On October 11, 2000, find the 5 top-selling products for each product subcategory that contributes more than 20% of the sales within its category.
SELECT p_cat, n_region, sales
FROM ( SELECT p_cat, n_region,
    SUM(s_amnt_sold) AS sales,
    MAX(SUM(s_amnt_sold)) OVER (PARTITION BY p_cat)
    AS MAX_REG_SALES
    FROM tpch
    WHERE s_t_id = TO_DATE('11-OCT-2001')
    GROUP BY p_cat, n_region )
WHERE sales = MAX_REG_SALES;

<table>
<thead>
<tr>
<th>P_CAT</th>
<th>N_REGION</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electron</td>
<td>Americas</td>
<td>581.92</td>
</tr>
<tr>
<td>Hardware</td>
<td>Americas</td>
<td>925.93</td>
</tr>
<tr>
<td>Peripher</td>
<td>Europe</td>
<td>4290.38</td>
</tr>
<tr>
<td>Software</td>
<td>Americas</td>
<td>4445.7</td>
</tr>
</tbody>
</table>
SELECT p_cat, p_subcat, p_id, SALES
FROM ( 
    SELECT p_cat, p_subcat, p_id,
        SUM(S_Amnt_Sold) AS Sales,
        SUM(SUM(S_Amnt_Sold)) OVER (PARTITION BY p_cat) AS Cat_Sales,
        SUM(SUM(S_Amnt_Sold)) OVER (PARTITION BY p_subcat) AS Subcat_Sales,
        RANK() OVER (PARTITION BY p_subcat ORDER BY SUM(s_amnt_sold)) DESC AS Rank_in_line
    FROM tpch
    WHERE s_t_id = TO_DATE('11-OCT-2000')
    GROUP BY p_cat, p_subcat, p_id
    ORDER BY p_cat, p_subcat
)
WHERE subcat_Sales > 0.2 * Cat_Sales
AND Rank_in_line <= 5;
Summary

- **Window/Analytic Functions**
  - WFuncType(Expr) OVER (WPartition WOrder WFrame)
  - RANK, CUME, NTILE
  - RATIO_TO_REPORT, LAG, LEAD
  - CURRENT ROW AND INTERVAL ’1’ DAY FOLLOWING

- Provide an **important SQL extension** for analysis and reporting in DW environments.

- Window frames allow efficient access to more than one row without a self-join.

- **Nested aggregates** are frequently used in combination with analytic functions.
  - Allow to relate subtotals to grand totals, compute percentages, etc.

- **Densification** allows to convert sparse data into dense data.
  - Essential technique for this is a partitioned outer join