Advanced Data Management Technologies

Unit 11 — SQL Analytic Functions

J. Gamper

Free University of Bozen-Bolzano
Faculty of Computer Science
IDSE

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

Outline

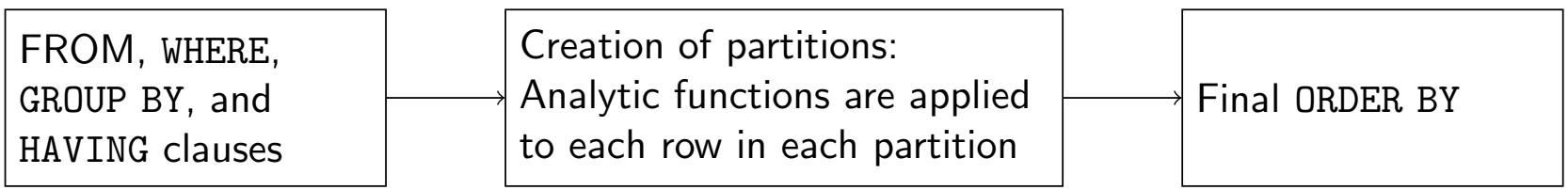
Outline

Window Functions/1

- ISO SQL:2003 has enhanced SQL's analytical processing capabilities by introducing so-called **analytic functions** (also known as window functions).
- All major database systems 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

Window Functions/2

- Basic syntax:
  `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:



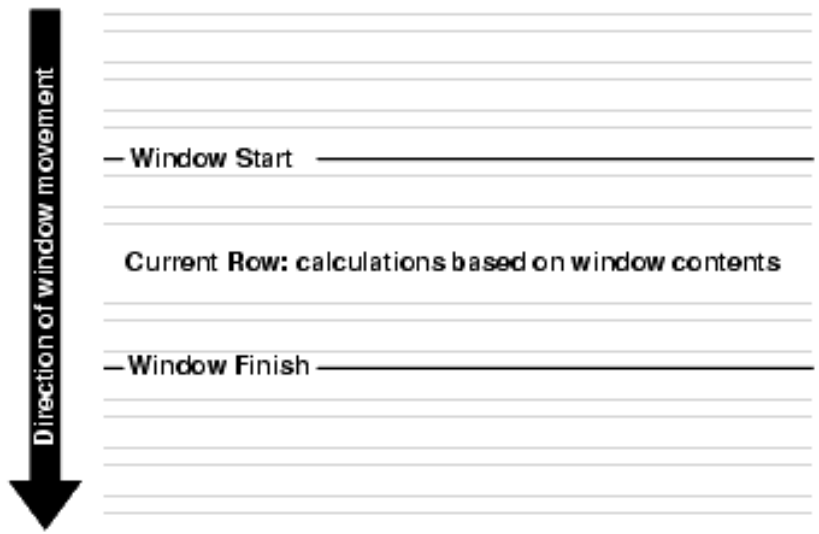
- Result set of first step is made available to window/analytic functions.

Window Functions/3

- Basic syntax:
  `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) 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

Ranking and Percentiles

- 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 )`

RANK Example/1

```
SELECT    m_desc,
          SUM(s_amount_sold),
          RANK() OVER (ORDER BY SUM(s_amount_sold))
FROM      bi.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

```
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;
```

M_DESC	SUM(S_AMNT_SOLD)	RANK
Internet	261278.04	1
Partners	800871.37	2
Direct Sales	1320497.4	3

- 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      bi.tpch
WHERE     t_cal_month_desc IN
              ('2000-08','2000-09','2000-10','2000-11')
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.

RANK with Partitioning Example/2

Media	Month	AmntSold	RankByMedia
Direct Sales	2000-08	1236104.31	1
Direct Sales	2000-10	1225584.31	2
Direct Sales	2000-09	1217807.75	3
Direct Sales	2000-11	1115239.03	4
Internet	2000-11	284741.77	1
Internet	2000-10	239236.26	2
Internet	2000-09	228241.24	3
Internet	2000-08	215106.56	4

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
```

Media	Month	AmntSold	RankByMedia	RankByMonth
Direct Sales	2000-08	1236104.31	1	1
Direct Sales	2000-10	1225584.31	2	1
Direct Sales	2000-09	1217807.75	3	1
Direct Sales	2000-11	1115239.03	4	1
Internet	2000-11	284741.77	1	2
Internet	2000-10	239236.26	2	2
Internet	2000-09	228241.24	3	2
Internet	2000-08	215106.56	4	2

DENSE_RANK

- DENSE_RANK leaves no gaps in the ranking sequence when there are ties.

Example: Rank and dense rank of amount sold.

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

Media	Month	AmntSold	Rank	Dense_Rank
Direct Sales	2000-09	1200000	1	1
Direct Sales	2000-10	1200000	1	1
Partners	2000-09	600000	3	2
Partners	2000-10	600000	3	2
Internet	2000-09	200000	5	3
Internet	2000-10	200000	5	3

Ranking Examples/1

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      bi.tpch
WHERE     m_desc IN ('Direct Sales', 'Internet'),
AND       t_cal_month_desc IN
              ('2000-08', '2000-09', '2000-10', '2000-11')
GROUP BY m_desc, t_cal_month_desc;
```

Ranking Examples/2

M_DESC	T_CAL_MO	SUM(S_AMNT_SOLD)	SUM(S_QUANTITY_SOLD)	RANK
Direct Sales	2000-08	1236104.31	12230	1
Direct Sales	2000-10	1225584.31	12584	2
Direct Sales	2000-09	1217807.75	11995	3
Direct Sales	2000-11	1115239.03	11357	4
Internet	2000-11	284741.77	1913	5
Internet	2000-10	239236.26	1450	6
Internet	2000-09	228241.24	1887	7
Internet	2000-08	215106.56	1132	8

Ranking Examples/3

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

```
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   bi.tpch
           GROUP BY m_desc
         )
WHERE    worst < 3 OR best < 3;
```

M_DESC	SUM(S_AMNT_SOLD)	Worst	Best
Direct Sales	57875260	4	1
Partners	26346342	3	2
Internet	13706802	2	3
Tele Sales	277426	1	4

Ranking Examples/4

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      bi.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

M_DESC	N_	SUM(S_AMNT_SOLD)	RANK_PER_GROUP
Direct Sales	US	638200.81	1
Partners	US	376813.18	2
Internet	US	124223.75	3
Direct Sales	JP	81073.81	4
Partners	JP	43347.12	5
Internet	JP	23862.29	6
Direct Sales	DK	17640.33	7
Partners	DK	16561.62	8
Internet	DK	2060.56	9
Direct Sales		736914.95	1
Partners		436721.92	2
Internet		150146.6	3
	US	1139237.74	1
	JP	148283.22	2
	DK	36262.51	3

Ranking Examples/6

Determine the output of the following statement:

```
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     bi.tpch
WHERE    c_id in (214, 608, 699)
AND      p_id in (42, 98, 123)
GROUP BY c_id, p_id;
```

C_ID	P_ID	R1	R2	R3	R4	R5
214	123					
608	42					
608	123					
699	42					
699	123					

CUME_DIST Example/1

```
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     bi.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.,
  - `CUME_DIST(x)` = (# of values smaller or equal to x) / (total # of values).
- `PERCENT_RANK()` is similar, but uses rank values rather than row counts in the denominator, i.e.,
  - `PERCENT_RANK()` = (rank of row - 1) / (# of rows - 1).
  - Row with rank 1 has percent rank 0.

CUME_DIST Example/2

MONTH	M_DESC	SUM(S_AMNT	CUME_DIST	(PERCENT_RANK)
2000-07	Internet	140423.34	.333333333	0.0
2000-07	Partners	611064.35	.666666667	0.5
2000-07	Direct Sales	1145275.13	1	1.0
2000-08	Internet	215106.56	.333333333	0.0
2000-08	Partners	661044.92	.666666667	0.5
2000-08	Direct Sales	1236104.31	1	1.0
2000-09	Internet	228241.24	.333333333	0.0
2000-09	Partners	666171.69	.666666667	0.5
2000-09	Direct Sales	1217807.75	1	1.0

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     bi.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

MONTH	SUM(S_AMNT_SOLD)	TILE4
2000-02	242416.38	1
2000-01	257285.89	1
2000-03	280010.94	1
2000-06	315950.95	2
2000-05	316824.18	2
2000-04	318105.67	2
2000-07	433823.77	3
2000-08	477833.26	3
2000-12	553534.39	3
2000-10	652224.76	4
2000-11	661146.75	4
2000-09	691448.94	4

ROW_NUMBER Example/1

```
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      bi.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

M_DESC	MONTH	SUM(S_AMNT_SOLD)	ROW_NUMBER
Direct Sales	2001-09	1100000	1
Direct Sales	2001-10	1000000	2
Internet	2001-09	500000	3
Internet	2001-10	700000	4
Partners	2001-09	600000	5
Partners	2001-10	600000	6

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

Outline

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     bi.tpch
WHERE    s_t_id = TO_DATE('11-OCT-2000')
GROUP BY m_desc;
```

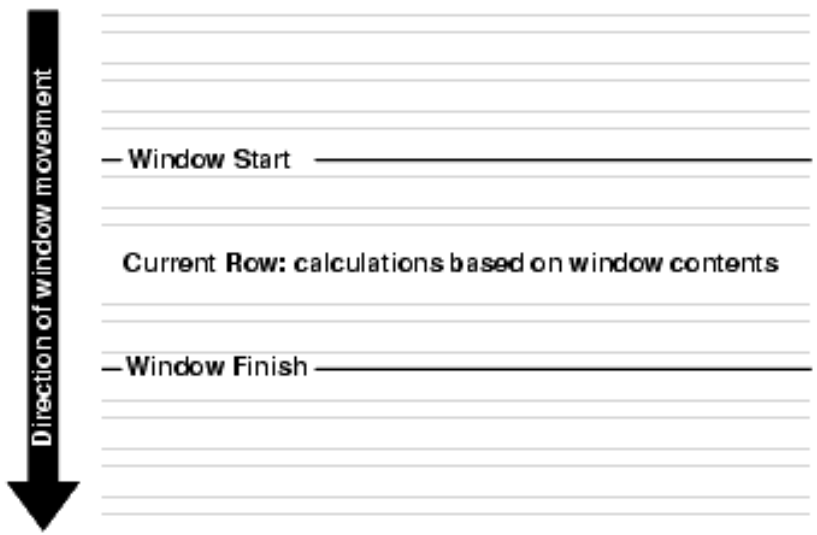
M_DESC	SALES	TOTAL_SALES	RATIO
Direct Sales	14447.23	23183.45	.623169977
Internet	345.02	23183.45	.014882168
Partners	8391.2	23183.45	.361947855

Outline

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.

Centered Aggregate Example/1

The centered 3 day moving average of all sales during week 51 in 1999.

```
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      bi.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

T_ID	SALES	CENTERED_3_DAY_AVG
20-DEC-99	134336.84	106675.93
21-DEC-99	79015.02	102538.713
22-DEC-99	94264.28	85341.7533
23-DEC-99	82745.96	93322.3067
24-DEC-99	102956.68	82936.7
25-DEC-99	63107.46	87062.2167
26-DEC-99	95122.51	79114.985

- The window frame in the first and last row contains only two rows.
  - e.g., $(63107.46 + 95122.51)/2 = 79114.985$

Ranking Example/1

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      bi.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

```
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     bi.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;
```

M_DESC	T_CAL_MO	RANK
Direct Sales	2000-08	1
Direct Sales	2000-10	2
Direct Sales	2000-09	3
Direct Sales	2000-11	4
Internet	2000-11	5
Internet	2000-10	6
Internet	2000-09	7
Internet	2000-08	8

LAG and LEAD Example/1

- 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      bi.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

S_T_ID	SUM(S_AMNT	LAG1	LEAD1
10-OCT-00	238479.49		23183.45
11-OCT-00	23183.45	238479.49	24616.04
12-OCT-00	24616.04	23183.45	76515.61
13-OCT-00	76515.61	24616.04	29794.78
14-OCT-00	29794.78	76515.61	

Outline

Densification/1

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

PROD	YEAR	WEEK	SALES
Deluxe	2001	25	5560
Mouse P	2001	24	2083
Mouse P	2001	26	2501
Standar	2001	24	2394
Standar	2001	26	1280

- Goal: produce dense report for weeks 24, 25, and 26.
- Can be important for reports or subsequent aggregations (3 months average), time series analysis, etc.

Densification/2

- 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.

Densification Example/1

```
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     bi.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   bi.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

PROD	YEAR	WEEK	DENSE_SALES
Deluxe	2000	24	0.0
Deluxe	2000	25	0.0
Deluxe	2000	26	0.0
Deluxe	2001	24	2260.72
Deluxe	2001	25	1871.3
Deluxe	2001	26	5560.51
Mouse P	2000	24	1685.52
Mouse P	2000	25	494.91
Mouse P	2000	26	1548.2
Mouse P	2001	24	2083.29
Mouse P	2001	25	0.0
Mouse P	2001	26	2501.79
Standar	2000	24	1007.37
Standar	2000	25	339.36
Standar	2000	26	183.92
Standar	2001	24	2394.04
Standar	2001	25	0.0
Standar	2001	26	1280.97

Reporting Examples

- Use the reporting functions to determine the answers to the following queries:
  1. Media that contributed with more than 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.

Reporting Examples – Query 3

```
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     bi.tpch
         WHERE    s_t_id = TO_DATE('11-OCT-2001')
         GROUP BY p_cat, n_region
       )
WHERE  sales = MAX_REG_SALES;
```

P_CAT	N_REGION	SALES
Electron	Americas	581.92
Hardware	Americas	925.93
Peripher	Europe	4290.38
Software	Americas	4445.7

Reporting Examples – Query 4

```
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      bi.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