Chapter 7 – Windows and Query Functions in SQL

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Inhalt

SQL:2003 Built-in Functions for OLAP

- 34 new built-in functions:
  - 7 new numeric functions
  - 16 new aggregate functions
  - 5 new windowed table functions
  - 4 new hypothetical aggregate functions
  - 2 new inverse distribution functions
- Windowed table functions provide facilities for calculating moving sums, moving averages, ranks, correlation, standard deviation, regression, etc.
- Significant functionality and performance advantages for OLAP applications

New Built-in Functions

- 7 new numeric functions
  - LN (expr)
  - EXP (expr)
  - POWER (expr, expr)
  - SQRT (expr)
  - FLOOR (expr)
  - CEIL[ING] (expr)
  - WIDTH_BUCKET(expr, expr, expr, expr)
    EX: WIDTH_BUCKET (age,0,100,10)
- 16 new aggregate functions
  - STDDEV_POP (expr)
  - STDDEV_SAMP (expr)
  - VAR_POP (expr)
  - VAR_SAMP (expr)
  - COVAR_POP (expr, expr)
  - COVAR_SAMP (expr, expr)
  - CORR (expr, expr)
  - REGR_SLOPE (expr, expr)
  - REGR_INTERCEPT (expr, expr)
  - REGR_COUNT (expr, expr)
  - REGR_R2 (expr, expr)
  - REGR_AVGX (expr, expr)
  - REGR_AVGY (expr, expr)
  - REGR_SXX (expr, expr)
  - REGR_SYY (expr, expr)
  - REGR_SXY (expr, expr)

Windowed Table Functions

- A windowed table function operates on a window of a table and returns a value for every row in that window. The value is calculated by taking into consideration values from the set of rows in that window.
- 5 new windowed table functions
  - RANK () OVER ...
  - DENSE_RANK () OVER ...
  - PERCENT_RANK () OVER ...
  - CUME_DIST () OVER ...
  - ROW_NUMBER () OVER ...
- In addition, 8 old aggregate functions and 16 new aggregate functions can also be used as windowed table functions:

  Example: sum(salary) OVER ...
- Allows calculation of moving and cumulative aggregate values.

Windowed Table Functions

- The set of rows is defined using the window-clause
- This set has three primary attributes
  - An Ordering
  - A Partitioning
  - A Window Aggregation Group
- This set is defined with the OVER clause

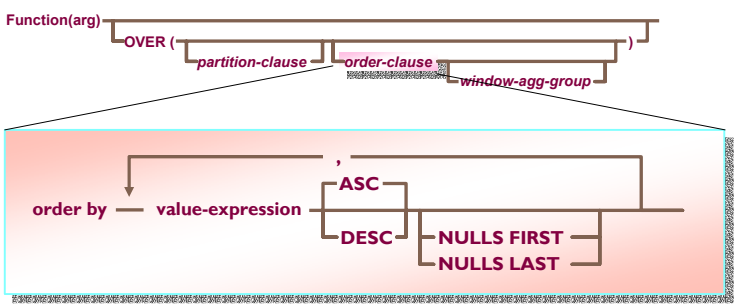
The RANK Function

- RANK
  - The Rank function returns the relative position of a value in an ordered group
  - Equal values (ties) are ranked the same
- In order to RANK, the ordering for the ranking must be specified

Rank	Test_Score
1	98
2	97
3	93
3	93
5	89
6	88
...	

The Order Clause

- The order-clause may contain multiple order items
  - Each item includes a value-expression
    - It may be a complex expression
    - Correlation is not allowed
- Note this clause is completely independent of the query's ORDER BY clause



Rank Example

Find the ranking of each employee in descending order of salary

```
select empnum, dept, salary,
       rank() over (order by salary desc) as rank
from emptab;
```

EMPNUM	DEPT	SALARY	RANK
0	-	-	1
4	2	-	1
3	-	84000	3
8	3	79000	4
6	1	78000	5
2	1	75000	6
7	1	75000	6
12	3	75000	6
10	3	55000	9
11	1	53000	10
5	1	52000	11
9	2	51000	12
1	1	50000	13

Ranks the salaries of all employees, highest salaries first.

Nulls collate high, so they rank first for descending ranks

Ranking Nulls First or Last

```
select empnum, dept, salary,
       rank() over (order by salary desc nulls last) as ranknl
from emptab;
```

EMPNUM	DEPT	SALARY	RANKNL
3	-	84000	1
8	3	79000	2
6	1	78000	3
2	1	75000	4
7	1	75000	4
12	3	75000	4
10	3	55000	7
11	1	53000	8
5	1	52000	9
9	2	51000	10
1	1	50000	11
0	-	-	12
4	2	-	12

Rank on Aggregations

- Windowed table functions are computed in the select list
  - After applying FROM, WHERE, GROUP BY, HAVING
  - They may not be referenced in any of these clauses
  - If you wish to reference them, you must nest them, or use a common table expression

Find rankings of each department's total salary

```
select dept, sum(salary) as sumsal,
     rank() over (order by sum(salary) desc nulls last) as rankdept
from emptab
group by dept;
```

```
DEPT        SUMSAL              RANKDEPT
--------    -----------         --------
       1        383000                1
       3        209000                2
       -         84000                3
       2         51000                4
```


Top n queries using rank

- In this example, we use a common table expression to compute the rank of the salary, and then reference the rank in the where clause of the outer query.

Find the three employees with the highest salaries

```
with dt as (
       select empnum, dept, salary,
            rank() over (order by salary desc nulls last) as ranksal
       from emptab )
select empnum, salary
from dt
where ranksal <= 3;
```

```
EMPNUM              SALARY
-----------         -----------
          3             84000
          8             79000
          6             78000
```

Rank, Dense_Rank, and Rownumber

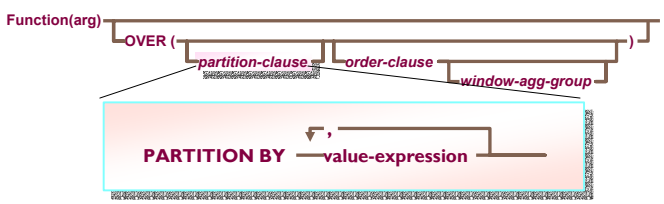
- RANK
- DENSE_RANK
  - Like RANK, but no gaps in rankings in the case of ties
- ROW_NUMBER
  - Ties are non-deterministically numbered

```
select empnum, dept, salary,
      rank() over (order by salary desc nulls last) as rank,
      dense_rank() over (order by salary desc nulls last) as denserank,
      row_number() over (order by salary desc nulls last) as rownum
      from emptab;

EMPNUM    DEPT      SALARY         RANK          DENSERANK     ROWNUM
--------  ------    --------       --------      ---------     --------
       3       -       84000            1              1             1
       8       3       79000            2              2             2
       6       1       78000            3              3             3
       2       1       75000            4              4             4
       7       1       75000            4              4             5
      12       3       75000            4              4             6
      10       3       55000            7              5             7
      11       1       53000            8              6             8
```


The Partitioning Clause

- What if we want to rank the employees' salaries within their departments?
  - The partition-clause allows you to subdivide the rows into partitions, much like the group by clause



Rank with Ordering and Partitioning

Find rankings of each employee's salary within her department

```
select empnum, dept, salary,
     rank() over (partition by dept order by salary desc nulls last)
       as rank_in_dept,
     rank() over (order by salary desc nulls last) as globalrank
 from emptab;
```

EMPNUM	DEPT	SALARY	RANK_IN_DEPT	GLOBALRANK
6	1	78000	1	3
2	1	75000	2	4
7	1	75000	2	4
11	1	53000	4	8
5	1	52000	5	9
1	1	50000	6	11
9	2	51000	1	10
4	2	-	2	12
8	3	79000	1	2
12	3	75000	2	4
10	3	55000	3	7
3	-	84000	1	1
0	-	-	2	12

Review of Ranking Functions

- The OVER clause defines the function window
  - PARTITION BY defines the set
  - ORDER BY defines the ordering within the set
- Other OLAP functions use the same window specification

```
rank() over (partition by dept
              order by salary desc nulls last) as rank
```

EMPNUM	DEPT	SALARY	RANK
6	1	78000	1
2	1	75000	2
7	1	75000	2
11	1	53000	4
5	1	52000	5
1	1	50000	6
9	2	51000	1
4	2	-	2
8	3	79000	1
12	3	75000	2
10	3	55000	3
3	-	84000	1
0	-	-	2

Reporting Functions - SUM

Find the sum of all salaries

```
select sum(salary) as sum
      from emptab;

SUM
-----------
     727000
```

- The OVER clause changes the SUM function into a windowed table function
  - Rather than aggregating the rows together, the function operates on the set and returns a single value per row
  - If nothing is specified, then all rows are in the set, so it just computes the sum for all rows, and returns it per row

Find the sum of all salaries, per row

```
select empnum, sum(salary) over ()
  as rsum
from emptab;
EMPNUM    RSUM
------- --------
      0   727000
      1   727000
      2   727000
      3   727000
      4   727000
      5   727000
      6   727000
      7   727000
      8   727000
      9   727000
     10   727000
     11   727000
     12   727000
```


Reporting Functions with Partitioning

Find each department's total salaries

```
select dept, sum(salary) as deptsum
from emptab
group by dept;

DEPT         DEPTSUM
---------- ----------
         1     383000
         2      51000
         3     209000
         -      84000
```

- If PARTITION BY is specified, then the reporting function is computed per partition
  - This is similar to the GROUP BY by for an aggregating function, but the rows aren't collapsed
- Traditionally, these queries were done with a join

For each employee, get his/her dept, and the sum of all salaries within his/her department

```
select empnum, dept,
   sum(salary) over (partition by dept)
      as deptsum
from emptab;
EMPNUM       DEPT         DEPTSUM
---------- ---------- ----------
         1          1     383000
         2          1     383000
         5          1     383000
         6          1     383000
         7          1     383000
        11          1     383000
         4          2      51000
         9          2      51000
         8          3     209000
        10          3     209000
        12          3     209000
         0          -      84000
         3          -      84000
```

Using Reporting Sums - Ratios

For each employee, find the ratio of his/her salary to the sum of all salaries in her department

```
select empnum, dept, salary,
decimal(salary,17,0) * 100 / sum(salary) over (partition by dept)
   as salratio
from emptab
order by dept, salratio desc;
```

EMPNUM	DEPT	SALARY	SALRATIO
6	1	78000	20.365
2	1	75000	19.582
7	1	75000	19.582
11	1	53000	13.838
5	1	52000	13.577
1	1	50000	13.054
4	2	-	-
9	2	51000	100.000
8	3	79000	37.799
12	3	75000	35.885
10	3	55000	26.315
0	-	-	-
3	-	84000	100.000

Cumulative Functions

- When an ORDER BY is specified, the window is defined as all rows equal to or preceding the current row

Find the total sales per quarter, and cumulative sales in quarter order for 1993-1995

```
select year(pdate) as year,quarter(pdate) as quarter,
    sum(ti.amount) as month_sales,
    sum(sum(ti.amount)) over (order by year(pdate), quarter(pdate))
     as cume_sales
from stars.trans t, stars.transitem ti
where t.transid=ti.transid
      and year(pdate) between 1993 and 1995
group by year(pdate), quarter(pdate);
```

YEAR	QUARTER	MONTH_SALES	CUME_SALES
1993	1	1270775.75	1270775.75
1993	2	1279171.45	2549947.20
1993	3	1050825.44	3600772.64
1993	4	1062329.99	4663102.63
1994	1	1176312.84	5839415.47
1994	2	1132602.73	6972018.20
1994	3	1241437.72	8213455.92
1994	4	1103020.49	9316476.41
1995	1	1193343.62	10509820.03
1995	2	1194296.14	11704116.17
1995	3	1418400.68	13122516.85
1995	4	1182153.01	14304669.86

Cumulative Functions with Partitioning

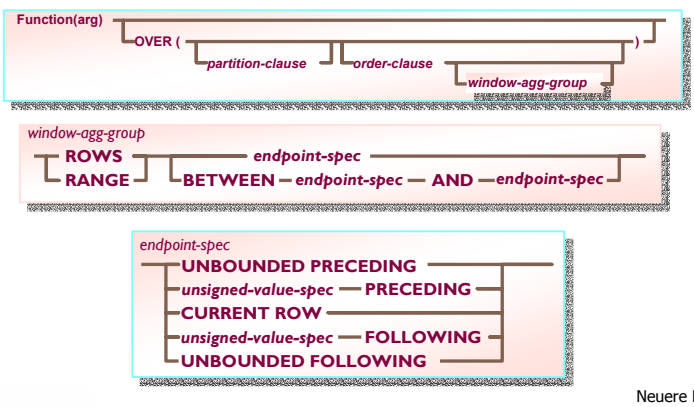
Find the total sales per quarter, and cumulative sales in quarter order PER YEAR for 1993-1995

```
select year(pdate) as year,quarter(pdate) as quarter,
      sum(ti.amount) as month_sales,
      sum(sum(ti.amount)) over (partition by year(pdate) order by quarter(pdate))
       as cume_sales_year,
      sum(sum(ti.amount)) over (order by year(pdate), quarter(pdate)) as cume_sales
from stars.trans t, stars.transitem ti
where t.transid=ti.transid
       and year(pdate) between 1993 and 1995
group by year(pdate),quarter(pdate);
```

YEAR	QUARTER	MONTH_SALES	CUME_SALES_YEAR	CUME_SALES
1993	1	1270775.75	1270775.75	1270775.75
1993	2	1279171.45	2549947.20	2549947.20
1993	3	1050825.44	3600772.64	3600772.64
1993	4	1062329.99	4663102.63	4663102.63
1994	1	1176312.84	**1176312.84**	5839415.47
1994	2	1132602.73	2308915.57	6972018.20
1994	3	1241437.72	3550353.29	8213455.92
1994	4	1103020.49	4653373.78	9316476.41
1995	1	1193343.62	**1193343.62**	10509820.03
1995	2	1194296.14	2387639.76	11704116.17
1995	3	1418400.68	3806040.44	13122516.85
1995	4	1182153.01	4988193.45	14304669.86

Advanced Windowing

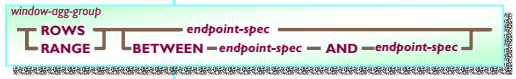
- It is possible to further refine the set of rows in a function's window when an order by is present
  - This is done with the window aggregation group clause
  - Allows inclusion/exclusion of ranges of values or rows within the ordering

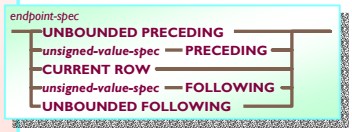


Aggregation Group

Find the total sales per quarter, and cumulative sales in quarter order for 1993-1995

```
select year(pdate) as year,quarter(pdate) as quarter,
      sum(ti.amount) as quarter_sales,
      sum(sum(ti.amount)) over (order by year(pdate), quarter(pdate)
      rows between unbounded preceding and current row)
       as cume_sales
from stars.trans t, stars.transitem ti
where t.transid=ti.transid
       and year(pdate) between 1993 and 1995
group by year(pdate), quarter(pdate);
```





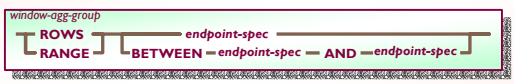
YEAR	QUARTER	QUARTER_SALES	CUME_SALES
1993	1	861246.41	861246.41
1993	2	1006042.32	1867288.73
1993	3	1217157.17	3084445.90
1993	4	1073961.01	4158406.91
1994	1	925632.22	5084039.13
1994	2	984755.32	6068794.45
1994	3	1079267.37	7148061.82
1994	4	1140777.68	8288839.50
1995	1	1722983.85	10011823.35
1995	2	910403.39	10922226.74
1995	3	1045813.60	11968040.34
1995	4	1261523.19	13229563.53

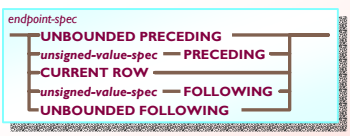
Same as a cumulative sum

Aggregation Group

How can you look at rows following instead of rows preceding?

```
select year(pdate) as year,quarter(pdate) as quarter,
      sum(ti.amount) as quarter_sales,
      sum(sum(ti.amount)) over (order by year(pdate), quarter(pdate)
                  rows between current row and unbounded following)
      as rcume_sales
from stars.trans t, stars.transitem ti
where t.transid=ti.transid
       and year(pdate) between 1993 and 1995
group by year(pdate),quarter(pdate);
```





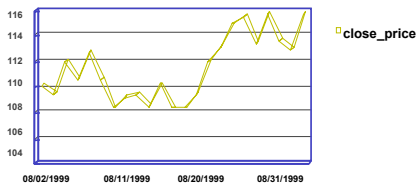
YEAR	QUARTER	QUARTER_SALES	RCUME_SALES
1993	1	861246.41	13229563.53
1993	2	1006042.32	12368317.12
1993	3	1217157.17	11362274.80
1993	4	1073961.01	10145117.63
1994	1	925632.22	9071156.62
1994	2	984755.32	8145524.40
1994	3	1079267.37	7160769.08
1994	4	1140777.68	6081501.71
1995	1	1722983.85	4940724.03
1995	2	910403.39	3217740.18
1995	3	1045813.60	2307336.79
1995	4	1261523.19	1261523.19

Curve Smoothing

```
select date,symbol, close_price
from stocktab
where symbol = 'IBM' and
 date between '1999-08-01' and '1999-09-01';
```

DATE	SYMBOL	CLOSE_PRICE
08/02/1999	IBM	110.125
08/03/1999	IBM	109.500
08/04/1999	IBM	112.000
08/05/1999	IBM	110.625
08/06/1999	IBM	112.750
08/09/1999	IBM	110.625
08/10/1999	IBM	108.375
08/11/1999	IBM	109.250
08/12/1999	IBM	109.375
08/13/1999	IBM	108.500
08/16/1999	IBM	110.250
08/17/1999	IBM	108.375
08/18/1999	IBM	108.375
08/19/1999	IBM	109.375
08/20/1999	IBM	112.000
08/23/1999	IBM	113.125
08/24/1999	IBM	114.875
08/25/1999	IBM	115.500
08/26/1999	IBM	113.375
08/27/1999	IBM	115.625
08/30/1999	IBM	113.625
08/31/1999	IBM	112.875
09/01/1999	IBM	115.625

- Often, users desire the ability to smooth their data. The ROWS clause allows this.



closing price vs. date

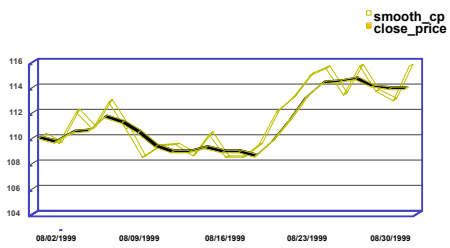
Curve Smoothing

Find the three day historical average of IBM stock for each day it traded

```
select date,symbol, close_price,
avg(close_price) over (order by date rows 2 preceding) as
 smooth_cp
from stocktab
where symbol = 'IBM' and date between '1999-08-01' and '1999-09-01';
```

DATE	SYMBOL	CLOSE_PRICE	SMOOTH_CP
08/02/1999	IBM	110.125	110.1250
08/03/1999	IBM	109.500	109.8125
08/04/1999	IBM	112.000	110.5416
08/05/1999	IBM	110.625	110.7083
08/06/1999	IBM	112.750	111.7916
08/09/1999	IBM	110.625	111.3333
08/10/1999	IBM	108.375	110.5833
08/11/1999	IBM	109.250	109.4166
08/12/1999	IBM	109.375	109.0000
08/13/1999	IBM	108.500	109.0416
08/16/1999	IBM	110.250	109.3750
08/17/1999	IBM	108.375	109.0416
08/18/1999	IBM	108.375	109.0000
08/19/1999	IBM	109.375	108.7083
08/20/1999	IBM	112.000	109.9166
08/23/1999	IBM	113.125	111.5000
08/24/1999	IBM	114.875	113.3333
08/25/1999	IBM	115.500	114.5000
08/26/1999	IBM	113.375	114.5833
08/27/1999	IBM	115.625	114.8333
08/30/1999	IBM	113.625	114.2083
08/31/1999	IBM	112.875	114.0416
09/01/1999	IBM	115.625	114.0416

- Now the curve is smooth, but it is uncentered
- Centered average:
  - … rows between 1 preceeding and 1 following …



three day historical average

Uncentered Windows

- Up to this point, all windows have contained the current row
  - It is possible to define windows that don't contain the current row
- ROWS 1 PRECEDING **EXCLUDE CURRENT ROW** gets the value of the expression for the immediately preceding row
  - Note that MAX doesn't do anything for a single row

For each month, find the average closing price of IBM stock for that month, and the month preceding

```
select year(date) as year, month(date) as month, avg(close_price) as avg_close,
 max(avg(close_price)) over (order by year(date), month(date)
   rows 1 preceding exclude current row)
 as prev_avg_close
from stocktab
where symbol = 'IBM'
group by year(date), month(date);
```

YEAR	MONTH	AVG_CLOSE	PREV_AVG_CLOSE
1999	7	110.29687	-
1999	8	111.29545	110.29687
1999	9	113.68181	111.29545
1999	10	118.66406	113.68181

Uncentered Windows - Percent-change

- This example uses the previous value to compute the percent change from one month to the next

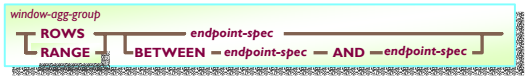
For IBM stock, what was the average price of the stock per month, as well as the percent change vs. the average price the preceding month?

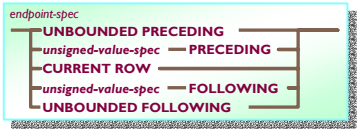
```
select year(date) as year, month(date) as month,
 avg(close_price) as avg_close,
 (avg(close_price) * 100 / avg(avg(close_price))
       over (order by year(date), month(date)
             rows 1 preceding exclude current row) - 100)
  as avg_pct_chg
from stocktab
where symbol = 'IBM'
group by year(date), month(date);
```

YEAR	MONTH	AVG_CLOSE	AVG_PCT_CHG
1999	7	110.2968	-
1999	8	111.2954	1.17768595
1999	9	113.6818	2.41605241
1999	10	118.6640	5.01244469

RANGE Based Windows

```
DATE       SYMBOL CLOSE_PRICE
---------- ------ ------------
08/02/1999 IBM         110.125
08/03/1999 IBM         109.500
08/04/1999 IBM         112.000
08/05/1999 IBM         110.625
08/06/1999 IBM         112.750
values missing for the weekend!
08/09/1999 IBM         110.625
08/10/1999 IBM         108.375
08/11/1999 IBM         109.250
08/12/1999 IBM         109.375
08/13/1999 IBM         108.500
  .. and here
08/16/1999 IBM         110.250
```





- ROW based windows work great when the data is dense
  - duplicate values and missing rows can cause problems
- In other situations, it would be nice to specify the aggregation group in terms of values, not absolute row position
  - For example, the stock table doesn't have any entries for weekends
  - Looking at the last 6 rows gives you more than the last week

RANGE Based Window Example

For IBM stock, what is the 7 calendar day historical average, and the 7 trade day historical average for each day in the month of August, 1999

```
select date,substr(dayname(date),1,9), close_price,
       avg(close_price) over (order by date rows 6 preceding) as avg_7_rows,
       count(close_price) over (order by date rows 6 preceding) as count_7_rows,
       avg(close_price) over (order by date range interval '6' day preceding) as avg_7_range,
       count(close_price) over (order by date range interval '6' day preceding) as count_7_range
from stocktab
where symbol = 'IBM' and date between '1999-08-01' and '1999-09-01';
```

DATE	2	CLOSE_PRICE	AVG_7_ROWS	COUNT_7_ROWS	AVG_7_RANGE	COUNT_7_RANGE
08/02/1999	Monday	110.125	110.12	1	110.12	1
08/03/1999	Tuesday	109.500	109.81	2	109.81	2
08/04/1999	Wednesday	112.000	110.54	3	110.54	3
08/05/1999	Thursday	110.625	110.56	4	110.56	4
08/06/1999	Friday	112.750	111.00	5	111.00	5
08/09/1999	Monday	110.625	110.93	6	111.10	5
08/10/1999	Tuesday	108.375	110.57	**7**	110.87	**5**
08/11/1999	Wednesday	109.250	110.44	**7**	110.32	**5**
08/12/1999	Thursday	109.375	110.42	**7**	110.07	**5**
08/13/1999	Friday	108.500	109.92	**7**	109.22	**5**
08/16/1999	Monday	110.250	109.87	**7**	109.15	**5**
08/17/1999	Tuesday	108.375	109.25	**7**	109.15	**5**
...						

Explicit Window Clause

- So far, a window was specified "in-line" in the SELECT clause of a query
- Alternative syntax uses and explicit WINDOW clause

```
select date,symbol, close_price,
avg(close_price) over w as smooth_cp
from stocktab
where symbol = 'IBM' and date between '1999-08-01' and '1999-09-01'
window w as (order by date rows 2 preceding)
```

- Advantages
  - window has a name, which can be used by multiple window table function invocations in the SELECT clause

Hypothetical Aggregate Functions

- 4 new hypothetical aggregate functions:
  - RANK (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
  - DENSE_RANK (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
  - PERCENT_RANK (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
  - CUME_DIST (expr, expr ...) WITHIN GROUP (ORDER BY <sort specification list>)
- Hypothetical aggregate functions evaluate the aggregate over the window extended with a new row derived from the specified values.
  - "What if" scenarios

Inverse Distribution Functions

- 2 new inverse distribution functions:
  - PERCENTILE_DISC (expr) WITHIN GROUP (ORDER BY <sort specification list>)
  - PERCENTILE_CONT (expr) WITHIN GROUP (ORDER BY <sort specification list>)
- Argument must evaluate to a value between 0 and 1.
- Return the values of expressions specified in <sort specification list> that correspond to the specified percentile value.