Chapter 6 – Windowed Tables and Window Functions in SQL

Outline

Overview

I. Object-Relational Database Concepts
   1. User-defined Data Types and Typed Tables
   2. Object-relational Views and Collection Types
   3. User-defined Routines and Object Behavior
   4. Application Programs and Object-relational Capabilities

II. Online Analytic Processing
   5. Data Analysis in SQL
   6. Windowed Tables and Window Functions in SQL

III. XML
   7. XML and Databases
   8. SQL/XML
   9. XQuery

IV. More Developments (if there is time left)
   temporal data models, data streams, databases and uncertainty, ...
Windowed Table Functions

- Windowed table function
  - operates on a window of a table
  - 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.

Concept (Compared To Set Functions)

- Set functions (aggregate functions)
  ```sql```
  ```sql```
  ```sql```
  ```sql```

- Windowed Table Functions (tuple-based aggregation)
  ```sql```
  ```sql```
Windowed Tables and Window Functions

- Windowed table
  - table (result of a table expression) together with one or more windows
  - windows are independent from each other
- Window
  - defines, for each row in the table, a set of rows (current row window) that is used to compute additional attributes
  - specified using a window specification (OVER clause)
  - based on three main concepts
    - window partitioning is similar to forming groups, but rows are retained
    - window ordering defines an order (sequence) of rows within each partition
    - window frame is defined relative to each row to further restrict the set of rows
- Window function
  - is applied for each row, over the current row window, returning a single value
  - used in column expressions in the select-list

\[
\text{Function}(\text{arg}) \quad \text{OVER (}
\begin{align*}
\text{partition-clause} & \\
\text{order-clause} & \\
\text{frame-clause} & 
\end{align*}
\)]

The Partitioning Clause

- The partition-clause allows to subdivide the rows into partitions, much like the group by clause

\[
\text{OVER (}
\begin{align*}
\text{partition-clause} & \\
\text{order-clause} & \\
\text{frame-clause} & 
\end{align*}
\)]

- Without further clauses, the current row window contains all the rows of the same partition (i.e., all the rows that are not distinct from the current row, including the current row)
  - if no partitioning clause is specified, then there is a single partition that contains the complete table
  - Windows do not reach across partition boundaries!
Set Functions as Window Functions

- The OVER clause turns a set function into a window function
  - Aggregated value is computed per current row window (here: per partition)
  ```sql
  select empnum, dept, salary, 
  avg(salary) over (partition by dept) as dept_avg
  from emptab;
  ```

<table>
<thead>
<tr>
<th>EMPNUM</th>
<th>DEPT</th>
<th>SALARY</th>
<th>DEPT_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>76000</td>
<td>63833</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>75000</td>
<td>63833</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>75000</td>
<td>63833</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>51000</td>
<td>63833</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>51000</td>
<td>63833</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>51000</td>
<td>63833</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>51000</td>
<td>51000</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>51000</td>
<td>51000</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>51000</td>
<td>69667</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>51000</td>
<td>69667</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>51000</td>
<td>69667</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>84000</td>
<td>84000</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The Order Clause

- The order-clause defines an order (sequence) within a partition
- May contain multiple order items
  - Each item includes a value-expression
  - NULLS FIRST/LAST defines ordering semantics for NULL values
- This clause is completely independent of the query's ORDER BY clause
Ranking Functions For Sequences

- **RANK**
  - returns the relative position of a value in an ordered group
  - equal values (ties) are ranked the same
- **DENSE_RANK**
  - like RANK, but no gaps in rankings in the case of ties
- **ROW_NUMBER**
  - ties are non-deterministically numbered
  - Ordering is required!
- **Example:**
  ```sql
  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;
  ```

Ranking Functions Example

<table>
<thead>
<tr>
<th>EMPNUM</th>
<th>DEPT</th>
<th>SALARY</th>
<th>RANK</th>
<th>DENSERANK</th>
<th>ROWNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-</td>
<td>84000</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>79000</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>78000</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>75000</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>75000</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>75000</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>55000</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>53000</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>52000</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>51000</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>50000</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>-</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>
Example: Rank with Ordering and Partitioning

- Find rankings of each employee's salary within her department
  
  ```sql
  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;
  ```

<table>
<thead>
<tr>
<th>EMPNUM</th>
<th>DEPT</th>
<th>SALARY</th>
<th>RANK_IN_DEPT</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
<td>78000</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>75000</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>75000</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>71000</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>70000</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>70000</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>51000</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>50000</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>75000</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>71000</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>51000</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>84000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

---

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
  - May use aggregation functions in window specification expressions
  - If you wish to reference them, you must nest them, or use a common table expression

- Example: Find rankings of each department's total salary
  
  ```sql
  select dept, sum(salary) as sumsal,
  rank() over (order by sum(salary) desc nulls last) as rankdept
  from emptab
  group by dept;
  ```

<table>
<thead>
<tr>
<th>DEPT</th>
<th>SUMSAL</th>
<th>RANKDEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>383000</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>209000</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>84000</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>51000</td>
<td>4</td>
</tr>
</tbody>
</table>
Cumulative Functions with Partitioning

Without a frame-clause, the current row window is now restricted to all rows equal to or preceding the current row within the current partition.

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

```
select year, quarter, sum(s.dollars) as q_sales,
    sum(sum(s.dollars)) over (partition by year order by quarter) as cume_sales_year
from sales s
where year between 1993 and 1995
group by year, quarter;
```

<table>
<thead>
<tr>
<th>YEAR</th>
<th>QUARTER</th>
<th>Q_SALES</th>
<th>CUME_SALES_YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993</td>
<td>1</td>
<td>1270775.75</td>
<td>1270775.75</td>
</tr>
<tr>
<td>1993</td>
<td>2</td>
<td>1279171.45</td>
<td>2549947.20</td>
</tr>
<tr>
<td>1993</td>
<td>3</td>
<td>1080825.44</td>
<td>3600722.64</td>
</tr>
<tr>
<td>1993</td>
<td>4</td>
<td>1062329.97</td>
<td>4063102.63</td>
</tr>
<tr>
<td>1994</td>
<td>1</td>
<td>1176312.84</td>
<td>1176312.84</td>
</tr>
<tr>
<td>1994</td>
<td>2</td>
<td>1132602.73</td>
<td>2208915.57</td>
</tr>
<tr>
<td>1994</td>
<td>3</td>
<td>1241437.72</td>
<td>3495363.29</td>
</tr>
<tr>
<td>1994</td>
<td>4</td>
<td>1103020.49</td>
<td>4553373.78</td>
</tr>
<tr>
<td>1995</td>
<td>1</td>
<td>1199343.62</td>
<td>1199343.62</td>
</tr>
<tr>
<td>1995</td>
<td>2</td>
<td>1194296.14</td>
<td>2387639.76</td>
</tr>
<tr>
<td>1995</td>
<td>3</td>
<td>1418400.68</td>
<td>3800640.44</td>
</tr>
<tr>
<td>1995</td>
<td>4</td>
<td>1192153.01</td>
<td>4988193.45</td>
</tr>
</tbody>
</table>

Window Frames

Further refine the set of rows in a function's window when an order by is present.

- Allows inclusion/exclusion of ranges of values or rows within the ordering.
Example: 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';
```

<table>
<thead>
<tr>
<th>DATE</th>
<th>SYMBOL</th>
<th>CLOSE_PRICE</th>
<th>SMOOTH_CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/02/1999</td>
<td>IBM</td>
<td>110.125</td>
<td>110.1250</td>
</tr>
<tr>
<td>08/03/1999</td>
<td>IBM</td>
<td>109.500</td>
<td>109.8125</td>
</tr>
<tr>
<td>08/04/1999</td>
<td>IBM</td>
<td>112.000</td>
<td>110.0414</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08/16/1999</td>
<td>IBM</td>
<td>110.250</td>
<td>109.3750</td>
</tr>
</tbody>
</table>

Now the curve is smooth, but it is uncentered.
Centered average:

```
... rows between 1 preceding and 1 following ...
```

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

```
RANGE Based Windows
```

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
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';
```

<table>
<thead>
<tr>
<th>DATE</th>
<th>2</th>
<th>CLOSE_PRICE</th>
<th>AVG_7_ROWS</th>
<th>COUNT_7_ROWS</th>
<th>AVG_7_RANGE</th>
<th>COUNT_7_RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/02/1999</td>
<td>Monday</td>
<td>110.125</td>
<td>110.12</td>
<td>1</td>
<td>110.12</td>
<td>1</td>
</tr>
<tr>
<td>08/03/1999</td>
<td>Tuesday</td>
<td>109.500</td>
<td>109.81</td>
<td>2</td>
<td>109.81</td>
<td>2</td>
</tr>
<tr>
<td>08/04/1999</td>
<td>Wednesday</td>
<td>112.000</td>
<td>110.54</td>
<td>3</td>
<td>110.54</td>
<td>3</td>
</tr>
<tr>
<td>08/05/1999</td>
<td>Thursday</td>
<td>110.625</td>
<td>110.56</td>
<td>4</td>
<td>110.56</td>
<td>4</td>
</tr>
<tr>
<td>08/06/1999</td>
<td>Friday</td>
<td>112.750</td>
<td>111.00</td>
<td>5</td>
<td>111.00</td>
<td>5</td>
</tr>
<tr>
<td>08/09/1999</td>
<td>Monday</td>
<td>110.250</td>
<td>110.44</td>
<td>7</td>
<td>110.44</td>
<td>7</td>
</tr>
<tr>
<td>08/10/1999</td>
<td>Tuesday</td>
<td>110.375</td>
<td>109.92</td>
<td>7</td>
<td>109.92</td>
<td>7</td>
</tr>
<tr>
<td>08/11/1999</td>
<td>Wednesday</td>
<td>110.250</td>
<td>109.92</td>
<td>7</td>
<td>109.92</td>
<td>7</td>
</tr>
<tr>
<td>08/12/1999</td>
<td>Thursday</td>
<td>110.375</td>
<td>109.92</td>
<td>7</td>
<td>109.92</td>
<td>7</td>
</tr>
<tr>
<td>08/13/1999</td>
<td>Friday</td>
<td>110.250</td>
<td>109.92</td>
<td>7</td>
<td>109.92</td>
<td>7</td>
</tr>
<tr>
<td>08/16/1999</td>
<td>Monday</td>
<td>110.250</td>
<td>109.92</td>
<td>7</td>
<td>109.92</td>
<td>7</td>
</tr>
<tr>
<td>08/17/1999</td>
<td>Tuesday</td>
<td>110.375</td>
<td>109.92</td>
<td>7</td>
<td>109.92</td>
<td>7</td>
</tr>
</tbody>
</table>

Explicit Window Definition Clause

- So far, a window was specified "in-line" in the SELECT clause of a query
- Alternative syntax uses an 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
Additional Capabilities

- 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.
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)
  - SORT (expr)
  - FLOOR (expr)
  - CEILING (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_AVGX (expr, expr)
  - REGR_SXX (expr, expr)
  - REGR_AVGY (expr, expr)
  - REGR_SYY (expr, expr)
  - REGR_SXY (expr, expr)
Summary

- **OLAP-Functionality in SQL**
  - extension of classical application of aggregation functions
- **Windowed tables, window functions**
  - tuple-based, attribute-based partitioning and analysis/aggregation of data
  - rows in a partition are preserved/expanded
    - in contrast to group-by/aggregation
  - window order defines sequence for sequence-based analysis
    - cumulative aggregation, ranking
  - window frame defines current row window dynamically for ordered windows
    - moving aggregates
- **Multiple windows can be defined for the same table**
  - windows are independent
- **SQL query execution model enhancement**
- **This functionality provides powerful infrastructure for optimized data analysis in the scope of OLAP**