Database Support for Business Intelligence Applications

PKDD 2000

Dr.-Ing. Wolfgang Lehner
University of Erlangen-Nuremberg / Germany
Dept. of Computer Science VI (Database Systems)
E-Mail: wolfgang@lehner.net

Outline
- Motivation and Introduction
  - Business Intelligence
  - DB-Support for BI
- Part A: Multiple Grouping Extension
  - GROUPING SETS(), CUBE(), ROLLUP()
- Part B: OLAP Functions
  - partitioning, windowing, ordering
- Part C: Combining A and B
  - ... we will end up in a mess
- Part D: MultiDimensional eXpressions
  - ... the Microsoft approach ...
- Summary and Conclusion

What is Business Intelligence?

Application Areas
- Online Analytical Processing
- Data Mining
- Reporting
- Supply chain Management
- Customer Relationship Management

Data Characteristics
- consolidated
- historic
- integrated

User Requirements
- take advantage of existing data
- high performance

Paper-based Reporting

Graphical Reporting-Tool
What is Business Intelligence? (3)

What is Business Intelligence? (4)

BI Application Stack

N-tier Architecture

High Performance? Rich Functionality?

Preliminary Step

Sample Scenario
University Erlangen-Nuremberg

Sample Star-Schema

Store Dimension Table
- store_id
- name
- city
- region
- zip_code
- country
- state

Sales Fact Table
- store_id
- product_id
- period_id
- sales
- units
- price

Product Dimension Table
- product_id
- brand
- color
- group
- family
- area

Period Dimension Table
- period_id
- description
- year
- quarter
- month
- week
- day

Customer Fact Table
- customer_id
- cc_limit
- cc_date
- zip_code
- marital_status

Customer Dimension Table
- cust_id
- cc_date
- cc_transaction

Virtual Super Star Schema

CREATE VIEW sales AS
SELECT fact.*, store.*, period.*, product.*, customer.*
FROM fact, store, period, product, customer
WHERE fact.store_id = store.store_id
AND fact.period_id = period.period_id
AND fact.product_id = product.product_id
AND fact.customer_id = customer.customer_id;

Sample Query

SELECT brand, month, region, sales
FROM sales
WHERE brand = 'XYZ';

<table>
<thead>
<tr>
<th>BRAND</th>
<th>MONTH</th>
<th>REGION</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>NORTH</td>
<td>3</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>NORTH</td>
<td>1</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>SOUTH</td>
<td>2</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>SOUTH</td>
<td>2</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>SOUTH</td>
<td>3</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>EAST</td>
<td>5</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>EAST</td>
<td>1</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>EAST</td>
<td>7</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>NORTH</td>
<td>1</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>NORTH</td>
<td>2</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>NORTH</td>
<td>1</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>SOUTH</td>
<td>7</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>SOUTH</td>
<td>3</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>SOUTH</td>
<td>7</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>SOUTH</td>
<td>4</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>EAST</td>
<td>2</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>EAST</td>
<td>18</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>EAST</td>
<td>1</td>
</tr>
</tbody>
</table>

PART A

Extended Grouping

SELECT brand, month, region, SUM(sales)
FROM sales
WHERE brand = 'XYZ'
GROUP BY brand, month, region
HAVING AVG(price) < 100;

<table>
<thead>
<tr>
<th>BRAND</th>
<th>MONTH</th>
<th>REGION</th>
<th>SUM(SALES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>EAST</td>
<td>11</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>SOUTH</td>
<td>12</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>NORTH</td>
<td>4</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>EAST</td>
<td>21</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>SOUTH</td>
<td>22</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>NORTH</td>
<td>4</td>
</tr>
</tbody>
</table>

... remember good old SQL days ...

- that was all we were able to provide ...
  - two places for extensions
    - group-by clause
    - aggregation operators

Sample Query - GROUP BY

SELECT brand, month, region, SUM(sales)
FROM sales
WHERE brand = 'XYZ'
GROUP BY brand, month, region
HAVING AVG(price) < 100;

<table>
<thead>
<tr>
<th>BRAND</th>
<th>MONTH</th>
<th>REGION</th>
<th>SUM(SALES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>EAST</td>
<td>11</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>SOUTH</td>
<td>12</td>
</tr>
<tr>
<td>XYZ</td>
<td>MAY</td>
<td>NORTH</td>
<td>4</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>EAST</td>
<td>21</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>SOUTH</td>
<td>22</td>
</tr>
<tr>
<td>XYZ</td>
<td>JUNE</td>
<td>NORTH</td>
<td>4</td>
</tr>
</tbody>
</table>
Extensions to the GROUP BY clause

- **GROUPING SETS()**
  - explicitly enumerate the participating grouping combinations

  ```sql
  ... GROUP BY GROUPING SETS(  
  (brand, month, region),  
  (brand, year, region)  
  )
  ```

  - grouping is done for every member of the grouping set

---

Sample Query - GROUPING SETS()

```sql
... GROUP BY GROUPING SETS (brand, month),  
  (color, year), (brand), (year), () ;
```

---

Super-Aggregates

- multiple grouping produces 'super-aggregate' values to identify
  - sub-totals / marginals
  - the grand total

- SQL representation
  - database system generates NULL values
  - **GROUPING()** function helps to distinguish 'super-aggregate' values from naturally occurring NULL values

---

Multidimensional Grouping

- **CUBE()**
  - multidimensional view of data
  - forms all possible grouping combinations for a set of N grouping attributes
  - results in 2^N combinations

  ```sql
  GROUP BY CUBE(color, month)
  ```

---

Sample Query - CUBE()

```sql
SELECT color, month, grouping(color),  
  GROUPING (month), SUM(sales)  
FROM sales  
GROUP BY CUBE(color, month);
```
Nested Data Cubes

- Data cubes are allowed within a grouping set

... GROUP BY GROUPING SETS (CUBE (color, month), CUBE (brand, month))

Results in two data cubes:
- (color, month), (color), (month), ()
- (brand, month), (brand), (month), ()

Hierarchical Data Cubes

- ROLLUP ()
  - Aggregation along a dimension hierarchy
  - Specify a hierarchical data cube as a list of distinct ROLLUPs:

... GROUP BY ROLLUP(a, b, c, d, e)

Corresponds to:
... GROUP BY GROUPING SETS ( (a, b, c, d, e), (a, b, c), (a, b), (a), () )

Sample Query - ROLLUP ()

SELECT ...
FROM sales
GROUP BY ROLLUP(area, family, group),
ROLLUP(year, quarter, month),
ROLLUP(country, state, region);

- We end up in 64 combinations
- Difference to CUBE():
  - Do not consider functionally dependent grouping combinations
- Use GROUPING to compute the hierarchy level

... SELECT ...
GROUPING(area) + GROUPING(family) + GROUPING(group)

Semantics of Grouping Combinations

- Different semantics
- GROUP BY level multiplication
- GROUPING SETS() level concatenation

GROUP BY a, b
GROUP BY GROUPING SETS ((a, b), (b))
GROUP BY a, GROUPING SETS ((b), (c))
GROUP BY a, b, GROUPING SETS ((b), (c))

Parallel Classifications

- Use ROLLUP () to address a single path in a classification hierarchy
- Use GROUPING SETS () with nested ROLLUP () to handle parallel classifications
- Example: time dimension

... GROUP BY GROUPING SETS ( ROLLUP (year, quarter, month), (week) ),
<Further dimensions...>
'Advanced' Grouping

- conditional drill-down
  - drill-down only for certain dimensional elements
- Example
  - sales per state and month and a split only for cities 'Paris' and 'Lyon'

```sql
SELECT ...
FROM ...
GROUP BY ROLLUP(state,
CASE WHEN city IN ('Paris', 'Lyon') THEN city
ELSE 'other cities'
END,
MONTH);
```

- conditional rollup ☺ Homework!

Summary Part A

- Simple group by
  - only a single grouping combination per query
- Multiple grouping combinations
  - powerful technique to return data grouped into different directions
  - very flexible to specify complex grouping conditions
  - against the (original) relational idea
  - (data of different schema within a single table)
  - useful in Data Mining
    - compute different item sets within a single database query

Types of Functions

- Scalar Functions
  - operate on values from a row and return a single value
  - Example: `SELECT price*sales AS turnover` FROM ...
- Aggregation Functions
  - operate on values from a set of rows (= group) and aggregate them into a single result
  - Example: `SELECT brand, SUM(sales)` FROM ... GROUP BY brand
- OLAP Functions
  - hot new stuff ...

OLAP Functions

- provide the ability to perform ranking and row numbering
- apply existing column-functions (like AVG, SUM, COUNT, ...) to retrieve scalar values
- main difference
  - aggregation functions
    - N-values of a single group
    - return a single value (e.g. SUM)
  - OLAP functions
    - N-values of a single group
    - return N values over a specified window
    - need to divide a row set into partitions
    - need of sequential ordering within the partition

... more on Aggregation Functions

- Small set of standardized aggregation functions
  - SUM, COUNT, MIN, MAX, AVG
- Huge variety of non-standardized analytic functions
  - Correlation
  - Covariance
  - Family of linear regression functions
    - fitting of an ordinary least squares regression line of the form \( y = a \cdot x + b \) to a set of number pairs
    - `REGR_SLOPE, REGR_INTERCEPT, REGR_R2, REGR_COUNT, REGR_R2, REGR_AVG, REGR_AVGV, REGR_AVGX, REGR_SXX, REGR_SYY, REGR_SXY`
Example: Analytic Functions

- get the correlation and the linear regression slope of sales as a function of income

```sql
SELECT country, state,
       CORRELATION(sales, cc_balance) AS correlation,
       REGR_SLOPE(cc_balance, sales) AS slope
FROM ...
WHERE ...
```

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>CITY</th>
<th>CORRELATION</th>
<th>SLOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germany</td>
<td>MUC</td>
<td>-0.13</td>
<td>-0.28</td>
</tr>
<tr>
<td>Germany</td>
<td>NUE</td>
<td>0.29</td>
<td>0.62</td>
</tr>
<tr>
<td>Germany</td>
<td>FRA</td>
<td>0.20</td>
<td>0.35</td>
</tr>
<tr>
<td>Germany</td>
<td>BER</td>
<td>0.28</td>
<td>0.73</td>
</tr>
<tr>
<td>Germany</td>
<td>STU</td>
<td>0.14</td>
<td>0.84</td>
</tr>
<tr>
<td>Germany</td>
<td>HDH</td>
<td>-0.47</td>
<td>-5.84</td>
</tr>
<tr>
<td>Germany</td>
<td>HHA</td>
<td>0.15</td>
<td>0.84</td>
</tr>
</tbody>
</table>

OLAP Functions in Detail

- OLAP Functions
  - extension of aggregation functions
  - to derive a single output value, the underlying data set is determined algorithmically
  - the set is computed using
    - ordering
    - partitioning
    - window: aggregation group specification

```
Function(arg)
  OVER (partition-clause order-clause window-agg-group)
```

Ranking and Numbering Function

- three new aggregation functions
  - `RANK()`: computes the ordinal rank of a row
  - in case of duplicate rows assigns the same rank
  - produces gaps in the ordering
  - produces gaps in the ordering
  - `DENSERANK()`: = 1 plus the number of rows that are distinct wrt. ordering
  - `ROWNUMBER()`: = 1 plus the number of rows that strictly precede the row

Example Queries - `RANK()`

```sql
SELECT cust_name,
       RANK() OVER (ORDER BY cc_limit) AS rank_limit,
       RANK() OVER (ORDER BY cc_balance) AS rank_balance
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>RANK_LIMIT</th>
<th>RANK_BALANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hans</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Chris</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Mike</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Mary</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Example Queries - `DENSERANK()`

```sql
SELECT cust_name,
       DENSERANK() OVER (ORDER BY cc_limit) AS rank_limit,
       DENSERANK() OVER (ORDER BY cc_balance) AS rank_balance
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>RANK_LIMIT</th>
<th>RANK_BALANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hans</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Chris</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Mike</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Mary</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Sample Queries - **ROWNUMBER()**

```sql
SELECT cust_name,
       ROWNUMBER() OVER (ORDER BY cc_limit) AS rowno_limit,
       ROWNUMBER() OVER (ORDER BY cc_balance) AS rowno_balance
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>ROWNO_LIMIT</th>
<th>ROWNO_BALANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hans</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Chris</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Mike</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Mary</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Example: Advanced Ranking

- **Ranking of Aggregations**

```sql
SELECT marital_status, AVG(cc_limit) AS avg_limit,
       RANK() OVER (ORDER BY AVG(cc_limit) DESC)
       AS avg_rank_limit
FROM cust
GROUP BY marital_status;
```

<table>
<thead>
<tr>
<th>MARITAL_STATUS</th>
<th>AVG_LIMIT</th>
<th>AVG_RANK_LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>single</td>
<td>13000</td>
<td>1</td>
</tr>
<tr>
<td>married</td>
<td>12500</td>
<td>2</td>
</tr>
</tbody>
</table>

- **TOP(n) Queries**

```sql
SELECT cust_name, cc_limit, rank_limit
FROM (SELECT cust_name, cc_limit,
          RANK() OVER (ORDER BY cc_limit desc) AS rank_limit
       FROM cust)
WHERE rank_limit <= 3;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>CC_LIMIT</th>
<th>RANK_LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chris</td>
<td>16000</td>
<td>1</td>
</tr>
<tr>
<td>Hans</td>
<td>15000</td>
<td>2</td>
</tr>
<tr>
<td>Mike</td>
<td>12000</td>
<td>3</td>
</tr>
</tbody>
</table>

Sample Queries - **SUM()**

- **Problem:** compute the ratio of the customer's balance to the total balance

```sql
SELECT SUM(cc_balance) AS sum_bal
FROM cust;
```

<table>
<thead>
<tr>
<th>SUM_BAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>29000</td>
</tr>
</tbody>
</table>

```sql
SELECT cust_name,
       SUM(cc_balance) OVER () AS sum_row_bal
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>SUM_ROW_BAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfgang</td>
<td>29000</td>
</tr>
<tr>
<td>Hans</td>
<td>29000</td>
</tr>
<tr>
<td>Chris</td>
<td>29000</td>
</tr>
<tr>
<td>Mike</td>
<td>29000</td>
</tr>
<tr>
<td>Mary</td>
<td>29000</td>
</tr>
</tbody>
</table>

```sql
SELECT cust_name, cc_balance,
       100.0 * cc_balance / SUM(cc_balance) OVER () AS ratio_bal
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>CC_BALANCE</th>
<th>RATIO_BAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfgang</td>
<td>5000</td>
<td>17.24</td>
</tr>
<tr>
<td>Hans</td>
<td>4000</td>
<td>13.79</td>
</tr>
<tr>
<td>Chris</td>
<td>5000</td>
<td>17.24</td>
</tr>
<tr>
<td>Mike</td>
<td>7000</td>
<td>24.13</td>
</tr>
<tr>
<td>Mary</td>
<td>8000</td>
<td>27.58</td>
</tr>
</tbody>
</table>

... lets make the problem a little more complex ...

- additionally consider the marital status, i.e. compute the ratio to the total balance per marital status

... need to partition the data set using a windowing mechanism ...
Windowing in OLAP Functions

- **Windowing**
  - allows the specification of a partitioning scheme plus order specification
  - defines the rows over which the function is applied, and in what order

- **Difference between grouping and partitioning**
  - grouping is table based (on query level)
  - partitioning is column based (individually for each column)

### Processing semantics

- apply selection condition (**WHERE**)
- build the grouping combinations (**GROUP-BY**)
- apply selection condition on groups (**HAVING**)
- sequentially compute each OLAP function (**OVER**- clause)
- generate partitions
- order the partitions
- apply given rank, numbering or aggregation function
- order the query’s result set (**ORDER BY**)

### Example: Ranking per Partition

```sql
SELECT cust_name, marital_status, cc_limit,
       RANK() OVER (PARTITION BY marital_status
       ORDER BY cc_limit desc) AS rank_limit
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>MARITAL_STATUS</th>
<th>CC_LIMIT</th>
<th>RANK_LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hans</td>
<td>married</td>
<td>15000</td>
<td>1</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>married</td>
<td>10000</td>
<td>2</td>
</tr>
<tr>
<td>Chris</td>
<td>single</td>
<td>14000</td>
<td>1</td>
</tr>
<tr>
<td>Mike</td>
<td>single</td>
<td>12000</td>
<td>2</td>
</tr>
<tr>
<td>Mary</td>
<td>single</td>
<td>11000</td>
<td>3</td>
</tr>
</tbody>
</table>
```

### Example: SUM and Partitioning

Recall the problem of computing the ratio of each customer’s balance to the total balance per marital status.

```sql
SELECT marital_status, SUM(cc_balance) AS sum_bal
FROM cust
GROUP BY marital_status;
```

<table>
<thead>
<tr>
<th>MARITAL_STATUS</th>
<th>SUM_BAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>married</td>
<td>9000</td>
</tr>
<tr>
<td>single</td>
<td>20000</td>
</tr>
</tbody>
</table>

```sql
SELECT cust_name, marital_status,
       SUM(cc_balance) OVER (PARTITION BY marital_status) AS sum_row_bal
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>MARITAL_STATUS</th>
<th>SUM_ROW_BAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfgang</td>
<td>married</td>
<td>9000</td>
</tr>
<tr>
<td>Hans</td>
<td>married</td>
<td>9000</td>
</tr>
<tr>
<td>Chris</td>
<td>single</td>
<td>20000</td>
</tr>
<tr>
<td>Mike</td>
<td>single</td>
<td>20000</td>
</tr>
<tr>
<td>Mary</td>
<td>single</td>
<td>20000</td>
</tr>
</tbody>
</table>

```sql
SELECT cust_name, marital_status, cc_balance,
       100.0 * cc_balance / SUM(cc_balance) OVER (PARTITION BY marital_status) AS sum_row_bal
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>MARITAL_STATUS</th>
<th>CC_BALANCE</th>
<th>SUM_ROW_BAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfgang</td>
<td>married</td>
<td>5000</td>
<td>55.55</td>
</tr>
<tr>
<td>Hans</td>
<td>married</td>
<td>4000</td>
<td>44.44</td>
</tr>
<tr>
<td>Chris</td>
<td>single</td>
<td>5000</td>
<td>25.00</td>
</tr>
<tr>
<td>Mike</td>
<td>single</td>
<td>7000</td>
<td>35.00</td>
</tr>
<tr>
<td>Mary</td>
<td>single</td>
<td>8000</td>
<td>40.00</td>
</tr>
</tbody>
</table>
```

### Example: Standard Deviation

```sql
SELECT marital_status, SUM(cc_balance) AS sum_bal,
       AVG(cc_balance) OVER (PARTITION BY marital_status) AS avg_bal,
       STDDEV(cc_balance) OVER (PARTITION BY marital_status) AS stddev_bal,
       STDDEV_AVG
FROM cust;
```

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>MARITAL_STATUS</th>
<th>CC_BALANCE</th>
<th>AVG_BAL</th>
<th>STDDEV_BAL</th>
<th>STDDEV_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfgang</td>
<td>married</td>
<td>5000</td>
<td>4500</td>
<td>+500.00</td>
<td>+1.00</td>
</tr>
<tr>
<td>Hans</td>
<td>married</td>
<td>4000</td>
<td>4500</td>
<td>+500.00</td>
<td>-1.00</td>
</tr>
<tr>
<td>Chris</td>
<td>single</td>
<td>5000</td>
<td>6666</td>
<td>+1247.21</td>
<td>-1.33</td>
</tr>
<tr>
<td>Mike</td>
<td>single</td>
<td>7000</td>
<td>6666</td>
<td>+1247.21</td>
<td>+0.26</td>
</tr>
<tr>
<td>Mary</td>
<td>single</td>
<td>8000</td>
<td>6666</td>
<td>+1247.21</td>
<td>+1.06</td>
</tr>
</tbody>
</table>

```

**NOTE:** OLAP functions are applied locally to each partition, e.g. **RANK()**/ **ROWNUMBER()** re-starts with value 1 within each partition.
Specifying an Aggregation Group

- **ROWS**
  - aggregation group is defined by counting rows

- **RANGE**
  - aggregation group is defined by an offset with regard to the current ordering within the partition

- **Size of aggregation group**
  - alternative 1: from somewhere to current row
    - starting point is addressable, and point is current row
    - UNBOUNDED PRECEDING (from first row to current row)
    - <uint> PRECEDING (from current row to <uint> to current row)
    - CURRENT ROW (only current row)

Specifying an Aggregation Group (2)

- **Example:**
  - sliding window operations (moving average)
    - ... usually used to smooth data
    - Suppose a centered window size of \( 3 \) (previous, the current, and the following) to compute the average value
      \[
      \text{OUTPUT}(N) = \frac{\text{VAL}(N-1) + \text{VAL}(N) + \text{VAL}(N+1)}{3}
      \]
    - cumulative sum
      - each value of the input stream is replaced by the sum of all previous values plus the current value
      \[
      \text{OUTPUT}(N) = \text{VAL}(N) + \text{SUM}(N-1)
      \]
  - Note:
    - an aggregation group is always specified INSIDE a column-based partitioning scheme

Example: Cumulative Sum

```sql
SELECT cust_name, cc_date, cc_transaction, SUM(cc_transaction) OVER (PARTITION BY month(cc_date) ORDER BY cc_date) AS cum_sum_month FROM cust;
```

Aggregation Group Scenario

```sql
SELECT cust_name, cc_date, cc_transaction, SUM(cc_transaction) OVER (PARTITION BY month(cc_date) ORDER BY cc_date) AS cum_sum_month FROM cust;
```

Specifying an Aggregation Group (2)

- Alternative 2: BETWEEN lower_bound AND upper_bound
  - lower bound
    - UNBOUNDED PRECEDING
    - includes the entire partition preceding the current row
    - <uint> PRECEDING
    - CURRENT ROW
  - upper bound
    - UNBOUNDED FOLLOWING
    - includes the entire partition following the current row
    - <uint> FOLLOWING
    - CURRENT ROW
Example: Data Smoothing

```
SELECT cust_name, cc_date, cc_transaction,
  AVG(cc_transaction) OVER (ORDER BY cc_date
  ROWS BETWEEN 1 PRECEDING
  AND 1 FOLLOWING)
AS cc_3mvg_avg
FROM cust;
```

Example: Data Smoothing (2)

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>CC_DATE</th>
<th>CC_TRANSACTION</th>
<th>CC_3MVG_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfgang</td>
<td>08/15/2000</td>
<td>177</td>
<td>190</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/16/2000</td>
<td>203</td>
<td>327</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/17/2000</td>
<td>601</td>
<td>477</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/18/2000</td>
<td>627</td>
<td>679</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/19/2000</td>
<td>809</td>
<td>660</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/20/2000</td>
<td>545</td>
<td>663</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/21/2000</td>
<td>635</td>
<td>594</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/22/2000</td>
<td>604</td>
<td>533</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/23/2000</td>
<td>360</td>
<td>481</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/24/2000</td>
<td>480</td>
<td>496</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/25/2000</td>
<td>648</td>
<td>418</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/26/2000</td>
<td>127</td>
<td>378</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/27/2000</td>
<td>361</td>
<td>380</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/28/2000</td>
<td>654</td>
<td>519</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/29/2000</td>
<td>542</td>
<td>445</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/30/2000</td>
<td>139</td>
<td>331</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/31/2000</td>
<td>312</td>
<td>412</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/01/2000</td>
<td>787</td>
<td>479</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/02/2000</td>
<td>338</td>
<td>596</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/03/2000</td>
<td>663</td>
<td>452</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/04/2000</td>
<td>355</td>
<td>440</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/05/2000</td>
<td>304</td>
<td>475</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/06/2000</td>
<td>768</td>
<td>393</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/07/2000</td>
<td>107</td>
<td>337</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/08/2000</td>
<td>136</td>
<td>121</td>
</tr>
</tbody>
</table>

Example: Data Smoothing (3)

```
/* this is an uncentered 7 day average
   6 preceding to current row
   current row to 6 following */
SELECT cust_name, cc_date, cc_transaction,
  AVG(cc_transaction) OVER (ORDER BY cc_date
  ROWS BETWEEN 6 PRECEDING
  AND CURRENT ROW)
AS cc_p7mvg_avg,
  AVG(cc_transaction) OVER (ORDER BY cc_date
  ROWS BETWEEN CURRENT ROW
  AND 6 FOLLOWING)
AS cc_f7mvg_avg
FROM cust;
```

Example: Data Smoothing (4)

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>CC_DATE</th>
<th>CC_TRANSACTION</th>
<th>CC_P7MVG_AVG</th>
<th>CC_F7MVG_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfgang</td>
<td>08/15/2000</td>
<td>177</td>
<td>177</td>
<td>513</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/16/2000</td>
<td>203</td>
<td>190</td>
<td>574</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/17/2000</td>
<td>601</td>
<td>327</td>
<td>597</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/18/2000</td>
<td>627</td>
<td>402</td>
<td>580</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/19/2000</td>
<td>809</td>
<td>483</td>
<td>583</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/20/2000</td>
<td>545</td>
<td>493</td>
<td>485</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/21/2000</td>
<td>635</td>
<td>513</td>
<td>459</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/22/2000</td>
<td>604</td>
<td>574</td>
<td>462</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/23/2000</td>
<td>360</td>
<td>597</td>
<td>453</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/24/2000</td>
<td>480</td>
<td>580</td>
<td>421</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/25/2000</td>
<td>648</td>
<td>583</td>
<td>397</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/26/2000</td>
<td>127</td>
<td>485</td>
<td>417</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/27/2000</td>
<td>361</td>
<td>459</td>
<td>447</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/28/2000</td>
<td>654</td>
<td>462</td>
<td>490</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/29/2000</td>
<td>542</td>
<td>453</td>
<td>448</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/30/2000</td>
<td>139</td>
<td>421</td>
<td>414</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>08/31/2000</td>
<td>312</td>
<td>397</td>
<td>503</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/01/2000</td>
<td>787</td>
<td>417</td>
<td>474</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/02/2000</td>
<td>338</td>
<td>447</td>
<td>381</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/03/2000</td>
<td>663</td>
<td>490</td>
<td>388</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/04/2000</td>
<td>355</td>
<td>448</td>
<td>334</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/05/2000</td>
<td>304</td>
<td>414</td>
<td>328</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/06/2000</td>
<td>768</td>
<td>503</td>
<td>337</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/07/2000</td>
<td>107</td>
<td>474</td>
<td>121</td>
</tr>
<tr>
<td>Wolfgang</td>
<td>09/08/2000</td>
<td>136</td>
<td>381</td>
<td>136</td>
</tr>
</tbody>
</table>

Summary Part B

- OLAP functions
  - extends the classical technique of grouping
  - column-wise partitioning
  - sequence-based processing (aggregation groups)
- SQL Standard???
  - predecessor: RSQL
  - joint proposal of IBM and Oracle
  - passed the initial draft status
  - fully implemented in IBM/DB2
  - implemented in the upcoming Oracle8i

Part C

Interaction of Extended Grouping and OLAP Functions
Hierarchical Ranking

- Recall the concepts
  - group by is done globally
  - partitioning is done locally for each OLAP function column
  - how to do these concepts interact?

- Consider ranking ...
  - ranking does only make sense with regard to the same grouping combination
  - use GROUPING() function to control the partitioning

Example:

```
SELECT zip_code, country, SUM(cc_limit) AS sum_limit,
       RANK() OVER (PARTITION BY GROUPING(zip_code) + GROUPING(country)
                   ORDER BY SUM(cc_limit) DESC) AS hierarchical_rank
FROM cust
GROUP BY ROLLUP(country, zip_code);
```

<table>
<thead>
<tr>
<th>ZIP_CODE</th>
<th>COUNTRY</th>
<th>SUM_LIMIT</th>
<th>HIERARCHICAL_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>140000</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>France</td>
<td>76000</td>
<td>1</td>
</tr>
<tr>
<td>40233</td>
<td>France</td>
<td>47000</td>
<td>2</td>
</tr>
<tr>
<td>40112</td>
<td>40112</td>
<td>28000</td>
<td>3</td>
</tr>
<tr>
<td>91058</td>
<td>Germany</td>
<td>37000</td>
<td>2</td>
</tr>
<tr>
<td>91053</td>
<td>Germany</td>
<td>27000</td>
<td>4</td>
</tr>
</tbody>
</table>

Hierarchical Ranking (2)

- ranking might be further refined to reset the rank for each new parent within a dimension hierarchy

Example:

```
SELECT zip_code, country, SUM(cc_limit) AS sum_limit, GROUPING(zip_code) + GROUPING(country) hier_level,
       RANK() OVER (PARTITION BY GROUPING(zip_code) + GROUPING(country),
                   CASE WHEN GROUPING(zip_code)=0 THEN country END
                   ORDER BY SUM(cc_limit) DESC) AS hierarchical_rank
FROM cust
GROUP BY ROLLUP(country, zip_code);
```

<table>
<thead>
<tr>
<th>ZIP_CODE</th>
<th>COUNTRY</th>
<th>SUM_LIMIT</th>
<th>HIER_LEVEL</th>
<th>HIERARCHICAL_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>140000</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>France</td>
<td>76000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>Germany</td>
<td>64000</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>40233</td>
<td>France</td>
<td>47000</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>40112</td>
<td>France</td>
<td>28000</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>91058</td>
<td>Germany</td>
<td>37000</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>91053</td>
<td>Germany</td>
<td>27000</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Summary Part C

- OLAP functions in the multidimensional context
  - still a very powerful mechanism to compute ranks, moving average, year-to-date analyses in a (hierarchical) data cube
  - specification gets quite tricky
    - OLAP servers usually generate such SQL code
  - ... do not dive into detail here!
Multidimensional Expressions

- Proposed by Microsoft
  - OLE DB for OLAP
  - now part of OLE DB V2 (SQL Server 7)
  - >40 OLAP vendors support/announced support
- User language to easily specify BI-style queries
  - Multidimensional context (cubes, dimensions, levels, cells, ...)
  - huge portfolio of commands specifying the query context (exploits hierarchical structures)
  - result set is kind of a classical spreadsheet table
- Spec prescribes the mapping to a relational backend
  - rather straightforward
  - star-schema like

Microsoft OLAP Architecture

Components of a MDX Statement

- SELECT (axis specification)
  - set value of descriptive elements ON COLUMNS
  - set value of descriptive elements ON ROWS
  - ... also ON PAGES, SECTIONS, CHAPTERS, ...
- FROM (cube specification)
  - list of data cubes (must be join compatible, i.e. share common dimensions)
- WHERE (slicer specification)
  - defines the scope of the query

Sample MDX Query

Sample MDX Query (2)

MDX: SetValues Expressions

- Simple Expressions
  - ... {USA, France}, (Mike, John) ON ROWS
- Specifying Sets
  - Germany.CHILDREN
  - Time.Quarterly.MEMBERS
  - DESCENDANTS(USA, Regions)
- Generating Sets
  - GENERATE {USA, France}, DESCENDANTS(Geography.CURRENT, Cities)
- Nesting of Sets
  - CROSSJOIN({USA, France}, (Mike, John))
- Ordering of Sets
  - consider dimension hierarchies / flat ordering
**MDX: Query Context**

- **WHERE clause**
  ```
  SELECT ... 
  FROM Sales 
  WHERE (Sales, [1999], [Products].[ALL]) 
  ```

- **TOPCOUNT**
  ```
  SELECT TOPCOUNT(Germany.CHILDREN, 5, Sales) 
  ON COLUMNS, Quarters.MEMBERS ON ROWS 
  FROM Sales 
  WHERE (BudgetSales, 1999, Products.ALL, 
  SalesPeople.ALL) 
  ```

**MDX: Query Context (2)**

- **FILTER expressions**
  ```
  generate axis values depending on data content 
  ```

- **FILTER expressions**
  ```
  SELECT FILTER({Germany.CHILDREN}, 
  (1998, ActualSales) > 500) ON COLUMNS, 
  Quarters.MEMBERS ON ROWS 
  FROM Sales 
  WHERE (BudgetSales, 1999, Products.ALL, 
  SalesPeople.ALL) 
  ```

**MDX Time Series Functionality**

- **TS Set Value Expressions**
  ```
  PERIODSTART(Quarter, [15-Sept-2000]) 
  Quarter 3 
  LAGPERIOD(3, [Sept-2000]) 
  ```

- **TS Member Value Expressions**
  ```
  PARALLELPERIOD(Year, 2, [Sept-2000]) 
  ```

- **TS (Numeric) Functions**
  ```
  COVARIANCE, CORRELATION 
  LINEAR REGRESSION 
  ```

**Summary Part D: MDX**

- **Powerful syntax to specify BI queries**
  - SQL like query specification
  - set/member expressions are directly exploiting the hierarchical structures of the data schema

- **Rich portfolio of:**
  - numeric functions
  - time series analysis

- **but**
  - mixture of content and presentation issues
  - procedural flavor for query specification

- **... an important player in the OLAP business**

**What’s left…**

- **...recall the two directions of database support**
  - extend the API ...
    - SQL extensions (grouping and OLAP functions)
    - MDX
  - provide support inside the database engine
    - already a huge variety available
    - index structures (bitmap, join index)
    - parallel processing / replication
    - materialized views (routing and maintenance)
    - there is still a lot of work to do
    - processing of complex grouping expressions
    - common-deriving of multiple OLAP function columns
    - result caching of OLAP function (unbounded preceding)
Summary

- Business Intelligence
  - support for different applications like
    - Reporting
    - Online Analytical Processing
    - Data Mining
  - relational database system is THE platform for
    - efficiently performing complex OLAP and Data Mining queries

- BUT !!! ... users have
  - to know about the existence
  - to take advantage
  of BI features

Questions ...

- Download
- Contact:
  - E-Mail: lehner@informatik.uni-erlangen.de
  - WWW: http://www6.informatik.uni-erlangen.de/wl.html