Lab Lecture

SQL Support for OLAP

SQL Extensions for Complex Data Analysis

Early versions of SQL provide limited support for complex data analysis

Processing scenario

- retrieve data from the database into the client application environment
- perform analysis on the client
- overhead by retrieval of huge amounts of data

Data analysis features of SQL:99

SQL/OLAP

- aggregate functions
- user-defined functions
- GROUP BY and HAVING
- CUBE and ROLLUP
- Window

SQL:1992 Select – An example

SELECT

Country.region, Time.year, Manufacturer, SUM(salesAmt)

FROM

Sales, Time, Product, Country

WHERE

Sales.product_ID = Product.product_ID
AND Sales.time_ID = Time.time_ID
AND Sales.country_ID = Country.country_ID
AND Product.product_category = 'Fridge'
AND Country.country_name = 'New Zealand'
AND Time.year = 2011

GROUP BY

Country.region, Time.year, Manufacturer

Relational treatment of multi-dimensional queries:

- o depends on Schema - Star vs. Snowflake
- o joins between n-dimension-tables and the fact-table
- o restrictions are on the dimension-table

Aggregation (Subtotals and Totals)

```
SELECT Product, NULL, NULL, SUM(turnover)
FROM Sales
WHERE Product = 'Fridge'
GROUP BY product

UNION

SELECT Product, Year, NULL, SUM(turnover)
From Sales
WHERE Product = 'Fridge'
GROUP BY Product, Year

UNION
SELECT Product, Year, Region, SUM(turnover)
FROM Sales
WHERE Product = 'Fridge'
GROUP BY Product, Year, Region
```

Calculation of Subtotals and Totals

Product	Year	Region	Turnover per year and region	Turnover per year	Turnover
Fridge	2005	Northland	135		
		Auckland	120		
				255	
	2006	Northland	140		
		Auckland	135		
				275	
					530

Product	Year	Region	Turnover
Fridge	2005	Northland	135
Fridge	2005	Auckland	120
Fridge	2005		255
Fridge	2006	Northland	140
Fridge	2006	Auckland	135
Fridge	2005		275
Fridge			530

Disadvantages of the UNION Version

Not efficient!

- many sub queries are necessary for calculation of subtotal
- grouping attributes specified more than once
- Join operations may have to be done more than once

Awkward to define!

- There might be an easier way
  - by using SQL:99 (SQL/OLAP) extensions

The ROLLUP Operator

- ROLLUP is a modification ('enlargement') of the GROUP BY Clause
- ROLLUP produces subtotals
  - delivers subtotals for each aggregation level, up to the total
- ROLLUP greatly simplifies the amount of computation

SELECT…GROUP BY ROLLUP(columnList)

The ROLLUPOperator

SELECT Company, Year, **SUM**(quantity) **AS** Sum
FROM Sales …
GROUP BY ROLLUP (Company, year)

Company	Year	Sum
Siemens	1999	2.000
Siemens	2000	3.000
Siemens	2001	3.500
Motorola	1999	1.000
Motorola	2000	1.000
Motorola	2001	1.500
Nokia	1999	1.000
Nokia	2000	1.500
Nokia	2001	2.000
Siemens	NULL	8.500
Motorola	NULL	3.500
Nokia	NULL	4.500
NULL	NULL	16.500

GROUP BY Company, Year

GROUP BY Company

Without **GROUP BY** Company

- ROLLUP simplifies the amount of computation.
- Only one sort is needed.
- Thereafter, only aggregate (sum) operations are required.

The CUBEOperator

- The Cube-Operator is a generalization of the ROLLUPoperator
- It computes all combinations of aggregates across the dimensions included
- The sequence of operation is not dependent on the sequence given in the query

CUBE Operator - Example

```
SELECT Producer, Year, SUM(quantity) AS Total
FROM Sales
Group BY Cube (Producer, Year)
```

The aggregation levels are:

1. Producer, Year
2. Producer
3. Year
4. Total

CUBE Operator – Example…

SELECT Producer, Year, Sum (quantity) as Total
FROM Sales
Group BY Cube (Producer, Year)

The aggregation levels are:

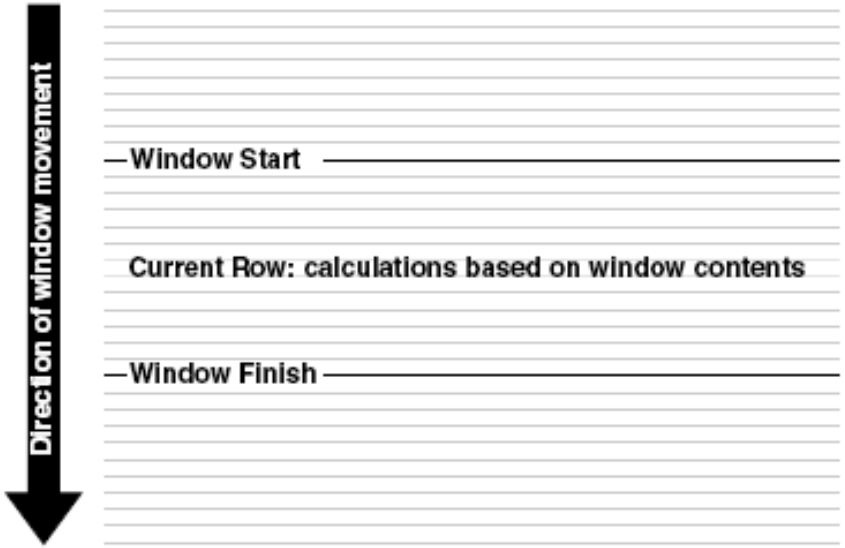
1. Producer, Year
2. Producer
3. Year
4. Grand Total

Producer	Year	Total	
Siemens	1999	2000	**GROUP BY** (Producer,Year)
Siemens	2000	3000	
Siemens	2001	3500	
Motorola	1999	1000	
Motorola	2000	1000	
Motorola	2001	1500	
Nokia	1999	1000	
Nokia	2000	1500	
Nokia	2001	2000	
Siemens	NULL	8500	**GROUP BY** (Producer)
Motorola	NULL	3500	
Nokia	NULL	4500	
NULL	1999	4000	**GROUP BY** (Year)
NULL	2000	5500	
NULL	2001	7000	
NULL	NULL	16500	Without **GROUP BY** clause

The WINDOW clause

- Intuitively identifies an ordered 'window' of rows 'around' each row in a table.
  - Allows us to apply a rich collection of aggregate functions (SUM, AVG) to the window of a row and extend the row with the results

Figure 21–2 Sliding Window Example



Source: Oracle Data Warehousing Guide

GROUP BY v WINDOW

- GROUP BY clause allows us to:
  - Create groupings (partitions) of rows
  - Apply aggregates to groups
  - However,
    - There is a single output row per partition
    - Rows within each partition are unordered
- A window is a combination of rows to which a function is applied, giving us:
  - cumulative totals
  - sliding averages
  - calculation over a partition or range

Window example

```
SELECT L.State, T.month, AVG(S.Sales) OVER W AS movavg
FROM Sales S, Time T, Locations L
WHERE S.timeID = T.timeID
AND S.LocID = L.LocID
WINDOW  W  AS      (PARTITION BY L.State
                    ORDER BY T.Month
                    RANGE BETWEEN INTERVAL '1' MONTH
                                  PRECEDING
                    AND INTERVAL '1'  MONTH FOLLOWING)
```

3 Steps in Defining a Window

1. Define *partitions* of the table
   - Partitions are based on the L.state column
2. Specify the *ordering* of rows within a partition
   - Rows within each partition are ordered by T.month
3. Frame windows – establish the boundaries of the window associated with each row in terms of the ordering of rows within partitions
   - Window for a row includes the row itself plus
   - all rows whose month value is within a month before or after
   - ie. A row whose month value is June 2009 has a window containing all rows with *month* equal to May, June or July 2009

- The answer row corresponding to a given row is constructed by first identifying its window
- Then, for each answer column (defined using a window aggregate function), the aggregate is computed using the rows in the window

Framing a Window

Two ways to frame a window in SQL:99

1. Using a RANGE construct
2. Directly specifying how many rows before and after the given row are in its window

```
WINDOW W AS (PARTITION BY L.State
             ORDER BY T.Month
             RANGE BETWEEN 1 PRECEDING
                   AND 1 FOLLOWING)
```

Framing a Window

- A window can consist of all rows of a partition or it can be defined as a "gliding window" with the smallest size of one row of the partition.
- The window defines the selection of rows for calculation.
  - The actual row serves as reference point for the starting and ending point of a window.
- The size of an window can be defined physically (ROWS) by defining the number of rows before and after the actual row.
- The size of a window can be defined logically by values in relation to the actual row (see RANGE definition)
- A window has a starting and an ending row.
- Depending on the definition a window can move on both sides. Such a "moving window" is defined by UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING e.g.:
  - A window for a cumulative sum has a starting row, and can glide from there to the last row of the partition.
  - A window for a gliding average has a starting and an ending row, and can glide over the whole partition

Sliding Window with a gliding average

```
SELECT Producer, Year, Sum (Sales) AS Total, Avg(Total) OVER w AS Moving_Avg
FROM SalesFact
WHERE Year BETWEEN 1999 and 2001
WINDOW w AS (PARTITION BY Producer
            ORDER BY Year
            ROWS 1 PRECEDING)
```

Producer	Year	Total	Moving_Avg
Motorola	1999	1000	1000
Motorola	2000	1000	1000
Motorola	2001	1500	1250
Nokia	1999	1000	1000
Nokia	2000	1500	1250
Nokia	2001	2000	1750
Siemens	1999	2000	2000
Siemens	2000	3000	2500
Siemens	2001	3500	3250

Window WINDOW

```
SELECT country, month, group, sales,
    SUM(sales) OVER w AS moving_sum
FROM Sales AS s
WINDOW w AS (
    PARTITION BY s.country, s.group
    ORDER BY s.month
    ROWS 2 PRECEDING )
```

window partitioning

window ordering

window framing

```
SELECT country, month, group, sales,
    SUM(sales) OVER (
        PARTITION BY s.country,
                s.group
        ORDER BY s.month
        ROWS 2 PRECEDING )
    AS moving_sum
FROM Sales AS s
```

in-line window specification

Calculated column

country	month	group	sales	moving_sum
F	01/04	100	500	500
F	02/04	100	1000	1500
F	03/04	100	250	1750
F	04/04	100	750	2000
F	02/04	200	1250	1250
F	03/04	200	2000	3250
F	04/04	200	500	3750
GB	01/04	100	400	400
GB	02/04	100	800	1200
GB	03/04	100	300	1500
GB	04/04	100	2000	3100
GB	05/04	100	2500	4800
GB	02/04	200	100	100
GB	03/04	200	100	200
GB	04/04	200	100	300
GB	05/04	200	100	300
I	01/04	300	250	250
I	02/04	300	750	1000
I	03/04	300	200	1200
I	04/04	300	1500	2450
I	05/04	300	800	2500

Window Framing

```
...ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
/* physical aggregation group on a dense column */

... ROWS BETWEEN 5 PRECEDING AND CURRENT ROW ... ROWS 5 PRECEDING
/* preceding 5 rows and current row */ ... ROWS UNBOUNDED PRECEDING
/* all preceding rows and current row */

... ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
 /* current row and all following rows */

... ROWS BETWEEN 6 PRECEDING AND 3 PRECEDING
/*the 3 rows starting 6 rows before the current row, current row is not included */
```

```
... ORDER BY month   RANGE BETWEEN INTERVAL '1' MONTH PRECEDING
                      AND INTERVAL '1' MONTH FOLLOWING
                     /* logical aggregation group including the last, current, and next month */

... ORDER BY month   ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
                     /* physical aggregation group on a dense column
```

Ranking Functions

- Allows you to rank items in a group (window, partition)
  - *Eg.* Finding the top 3 products sold in Auckland last year
- Three functions
  - **RANK**
  - **DENSE_RANK**
  - **PERCENT_RANK**
- Example: Ranking a competition where 3 people have tied for 2nd place
  - Using **RANK**

    1, 2, 2, 2, 5,…
  - Using **DENSE_RANK** leaves no gaps

    1, 2, 2, 2, 3,…

RANK OPERATORS

- Example

```
SELECT Producer, Year, Sum(Sales),
RANK() OVER (ORDER BY Sum(Sales) DESC) AS Rank
FROM SalesFact
WHERE Year = 2010
GROUP BY Producer
```

Producer	Year	Sum(Sales)	Rank
Siemens	2001	3500	1
Nokia	2000	2000	2
Motorola	1999	1500	3

- To produce the table on the next slide:

```
SELECT Producer, Year, Sum(Sales),
RANK() OVER (ORDER BY Sum(Sales) DESC) AS Rank  DENSE_RANK()
OVER (ORDER BY Sum(Sales) DESC) AS Drank  PERCENT_RANK()
OVER (ORDER BY Sum(Sales) DESC) AS PCT_Rank
FROM SalesFact
WHERE Year BETWEEN 2008 and 2010
GROUP BY Producer
```

RANK OPERATORS

DENSE_RANK calculates a rank as well but DOES NOT produce gaps

Producer	Year	Total	Rank	Drank	Pct_Rank
Siemens	2010	3500	1	1	0
Siemens	2009	3000	2	2	0.17
Nokia	2010	2000	3	3	0.33
Siemens	2008	2000	3	3	0.33
Nokia	2009	1500	5	4	0.67
Motorola	2010	1500	5	4	0.67
Nokia	2008	1000	7	5	1.00
Motorola	2009	1000	7	5	1.00
Motorola	2008	1000	7	5	1.00

References

- Ramakrishnan & Gherke “Database Management Systems”, 3rd edition.
- Oracle Data Warehousing Guide 10g, Chapters 20, 21.