Part V

Data Warehouse Queries

Data Warehouse Queries

Overview

Introduction

- Typical queries to data warehouses include aggregations, e.g.

> *How many units were sold from the product groups soft drinks and wine in Saxony-Anhalt and Thuringia per month and place in the years 2010 and 2011 and which turnover did this generate?*

- Characteristics of typical data warehouse queries:
  - From the large quantity of existing facts only one particular one becomes, in most dimensions limited data range requested

Introduction (2)

- Multidimensional query:
  - Restriction, which is usually based on simple classification node references
- Special optimization techniques make sense!
- Problem: Aggregations on large amounts of data
- Example beverage retail chain
  - 2,000 branches, per branch: 1,000 customers daily with 5 Articles
  - per purchase: 1 item soft drink, 0,5 items wine
  - per day: 10.000.000 records in fact table `Sales`, record size 63 Byte
  - fact table: approx. 600 MB/day, with 310 shopping days 182 GB/year
  - scan of the fact table over 10 years: 6,5 hours at 80 MB/s !!

Relational implementation of multidimensional queries

- Basically depending on illustration for schemas
  - star vs. snowflake schema
  - Classification hierarchies
- Frequent query pattern
  - $(n+1)$-way connection between n dimension tables and the fact table and
  - Restrictions via dimension tables

Star-Join: Example

```
SELECT L_City, YEAR_MONTH(T_Date),
   SUM(S_Number) AS Units,
   SUM(S_Number * P_Sales price) AS Turnover
FROM Sale, Time, Product, Place
 WHERE S_Time_ID = T_ID AND S_Product_ID = P_ID AND
   S_Location_ID = L_ID AND
   YEAR(T_Date) BETWEEN 2010 AND 2011 AND
   L_RegionIN ('Saxony-Anhalt', 'Thuringia') AND
   P_ProductGroup IN ('soft drink', 'wine')
GROUP BY L_City, YEAR_MONTH(T_Date)
```

Star-Join: Structure

- **SELECT** clause
  - Parameters with aggregate function
  - Result granularity (e.g. month, region)
- **FROM** clause
  - Fact and dimension tables
- **WHERE** clause
  - Bonding conditions
  - Restrictions (e.g.: P_ProductGroup **IN** ('Soft drink', 'Wine') **AND** L_Region **IN** ('Saxony-Anhalt', 'Thuringia') **AND** **YEAR**(T_Date) **BETWEEN** 2010 **AND** 2011)

Grouping and Aggregation

Grouping and Aggregation

- Data Analysis: Aggregation of multidimensional data
- Aggregate function: "dimensionless" response
  - standard: **`SUM`**, **`MIN`**, **`MAX`**, **`COUNT`**
  - extensions: statistical, physical and financial functions
  - User-defined aggregate functions
- Grouping: "1-dimensional"’ Answer
  - Result: Table with aggregate values indexed by set of attributes

Grouping and Aggregation (2)

- SQL: **GROUP BY** attrib_list [**HAVING** condition]
  - Grouping with respect to equal values of grouping attributes
  - Final projection only via grouping attributes or Aggregations
- Restrictions
  - Calculation of histograms: Aggregations over calculated Categories
    `... GROUP BY func(time) AS week ...`
  - Calculation of subtotals and grand totals
  - Calculation of crosstabs

Aggregate Functions

- standard SQL functions like **MIN**, **MAX**, **SUM**, **COUNT**, **AVG**
- new functions in SQL:2003 for variance **VAR_POP** (x), standard deviation **STDDEV_POP** (x), Covariance **COVAR_POP** (x, y) and Correlation coefficients **CORR** (x, y)
- for the whole population (**_POP**) or with Bessel correction (**_SAMP**)

Aggregate functions: examples

- Is there a (linear) relationship between the number of sold products and their selling price?

```
SELECT COVAR_POP(S_number, P_sales price)
FROM Sales, Product
 WHERE S_Product_ID = P_ID
```

- values close to zero ≈ not stronger than statistical coincidence

Aggregate functions: examples

- covariance does not provide information about the strength of the correlation, better correlation coefficient

```
SELECT CORR(P_sales price, P_purchase price),
   P_ProductGroup
FROM Sales, Product
 WHERE S_Product_ID = P_ID
GROUP BY P_ProductGroup
```

- values from $0{,}5$ indicate medium to strong correlation

Aggregate functions: examples

- regression analysis for correlation between number and sales price
- Calculation of line slope **REGR_SLOPE**, Regression coefficients **REGR_R2**, mean price **REGR_AVGX** and average number **REGR_AVGY**

```
SELECT S_Channel,
   REGR_SLOPE(S_number, P_sales price) AS increase,
   REGR_R2(S_number, P_sales price) AS coefficient,
   REGR_COUNT(S_Number, P_Sales price) AS Number,
   REGR_AVGX(S_number, P_sales price) AS MPreis,
   REGR_AVGY(S_number, P_selling price) AS MAnzahl
FROM Sales, Product, Time
 WHERE S_Product_ID = P_ID AND
S_Time_ID = T_ID AND YEAR(T_Date) = 2011
 GROUP BY S_Channel
```

Calculation of subtotals and grand totals

PGroup	Year	Region	Turnover PGroup-Year-Region	Turnover PGroup-Year	Turnover PGroup	Turnover
Wine	2010	Saxony-Anhalt Thuringia	45 43			
				88		
	2011	Saxony-Anhalt	47			
				47		
					135	
Bier	2011	Thuringia	42			
				42		
					42	
						177

Calculation of subtotals and grand totals (2)

```
-- subtotal (1) "across all product groups, years and federal states"
SELECT P_ProductGroup AS PGroup, YEAR(T_Date), L_Province,
   SUM(S_Number * P_Sales price) AS Turnover
FROM Sale, Time, Product, Place
 WHERE S_Time_ID = T_ID AND S_Product_ID = P_ID AND
   S_Location_ID = L_ID
GROUP BY P_ProductGroup, YEAR (T_Date), L_Province
UNION ALL
 -- subtotal (2) "over all product groups and years
SELECT P_ProductGroup AS PGroup, YEAR (T_Date),
   CAST(NULL AS VARCHAR(50)),
   SUM(S_Number * P_Sales price) AS Turnover
FROM Sale, Time, Product, Place
 WHERE S_Time_ID = T_ID AND S_Product_ID = P_ID AND
   S_Location_ID = L_ID
GROUP BY P_ProductGroup, YEAR(T_Date)
UNION ALL
```

Calculation of subtotals and grand totals (3)

```
-- subtotal (3) "across all product groups
SELECT P_ProductGroup AS PGroup, CAST(NULL AS INT),
   CAST(NULL AS VARCHAR(50)),
   SUM(S_Number * P_Sales price) AS Turnover
FROM Sale, Time, Product, Place
 WHERE S_Time_ID = T_ID AND S_Product_ID = P_ID AND S_Location_ID = L_
GROUP BY P_ProductGroup
UNION ALL
 -- Total
 SELECT CAST(NULL AS VARCHAR(50)) AS Group, CAST(NULL AS INT),
   CAST(NULL AS VARCHAR(50)),
   SUM(S_Number * P_Sales price) AS Turnover
FROM Sale, Time, Product, Place
 WHERE S_Time_ID = T_ID AND S_Product_ID = P_ID AND S_Location_ID = L_
```

Selection of subtotals and grand totals

Group	Year	L_Region	Turnover
wine	2010	Saxony-Anhalt	45
wine	2010	Thuringia	43
wine	2011	Saxony-Anhalt	47
beer	2011	Thuringia	42
wine	2010	*NULL*	88
wine	2011	*NULL*	47
beer	2011	*NULL*	42
wine	*NULL*	*NULL*	135
beer	*NULL*	*NULL*	42
NULL	*NULL*	*NULL*	177

Disadvantages of the UNION variant

- involves a great deal of effort:
  - calculation of all subtotals for n grouping attributes requires 2^n partial requests
  - Possible compound operations must be repeated several times
- elaborate formulation:
  - However, possibly generated by OLAP tools
  - adherence to the structure

Calculation of cross tables

- Symmetric Aggregation
- Also pivot tables

Sales	2010	2011	Total
Thuringia	120	135	255
Property content	135	140	275
Total	255	275	530

PIVOT in SQL Server

```
SELECT Year, [THUR] AS Thuringia,
   [SANH] AS Saxony-Anhalt FROM Sale
 PIVOT (SUM(sales) FOR
   Region IN ([THUR], [SANH])
```

Year	Thuringia	Saxony-Anhalt
2010	135	120
2011	140	135

CUBE and ROLLUP

Cube Operator

- "‘short form"’ for request samples for the calculation of part and Totals
- Generation of all possible grouping combinations from given set of grouping attributes
- Result: Table with aggregated values
- total aggregate:

$$\texttt{NULL}, \texttt{NULL}, \ldots, \texttt{NULL}, f(*)$$

- Higher dimensional levels with fewer `NULL` values

Cube Operator: Example

PGroup	Region	Year	Turnover
Wine	Saxony-Anhalt	2010	45
Wine	Thuringia	2010	43
Wine	Saxony-Anhalt	2011	47
Beer	Thuringia	2011	42

CUBE

PGroup	Year	Region	Turnover
wine	2010	Saxony-Anhalt	45
wine	2010	Thuringia	43
...	...	...	...
wine	2010	NULL	88
wine	2011	NULL	47
beer	2011	NULL	42
Wine	NULL	Saxony-Anhalt	92
Wine	NULL	Thuringia	43
beer	NULL	Thuringia	42
Wine	NULL	NULL	135
beer	NULL	NULL	42
NULL	2010	Saxony-Anhalt	45
...	...	...	...
NULL	NULL	Saxony-Anhalt	92
NULL	NULL	Thuringia	85
...	...	...	...
NULL	2010	NULL	88
NULL	2011	NULL	89
NULL	NULL	NULL	177

Cube: Details

- cardinality
  - N attributes with cardinality $C_1, C_2, ..., C_N$
  - total cardinality of the **CUBE**:

$$\prod_{i=1}^{N}(C_i + 1)$$

- number of super aggregate values
  - N attributes in the **SELECT** clause
  - super aggregates: $2^N - 1$

Cube operator: SQL syntax

- Implementation in SQL Server, DB2, Oracle
- Syntax ORACLE:

```
SELECT P_ProductGroup AL_S PGroup,
   L_Region, YEAR(T_Date),
   SUM(S_Number * P_Sales price) AS Turnover
FROM Sale, Time, Product, Location
 WHERE ...
 GROUP BY CUBE(P_ProductGroup, L_Region,
   YEAR(T_Date)
```

- Function **GROUPING**(Attribute)
  - Returns value = 1 if aggregated via attribute
  - Returns value = 0 if grouped by attribute
- Suppression of partial sums, e.g. the total sum

```
...   HAVING NOT (GROUPING(P_ProductGroup) = 1 AND
                  GROUPING(L_Region) = 1 AND
                  GROUPING(YEAR(T_Date)) = 1)
```

Rollup Operator

- **CUBE** operator: interdimensional
  - applicable for attributes from different dimensions
  - Too complex for roll-up or drill-down operations
- **ROLLUP** operator: intradimensional
  - Generation of the attribute combinations

$$(A_1, ..., A_N), (A_1, ..., A_{N-1}), (A_1, A_2), (A_1), ()$$

for given attribute*list* $A_1, ..., A_N$

ROLLUP operator: Example (simple)

- request:

```
SELECT P_group, T_day, T_month, T_year,
   SUM(S_Number * P_Sales price) AS Turnover
FROM Sale, Time, Product, Place
 WHERE S_Product_ID = P_ID AND
   S_Location_ID = L_ID AND
   S_Time_ID = T_ID AND YEAR(T_Date) = 2011 AND
   P_ProductGroup = 'Red wine'
GROUP BY ROLLUP(T_year, T_month, T_day)
```

- evaluation:
  - rollup: `(T_year,T_month,T_day),` `(T_year,T_month),(T_year),()`

ROLLUP operator: Example (simple)

group	day	month	year	turnover
Red wine	1	January	2011	100
Red wine	2	January	2011	100
...	...	...	...	...
Red wine	31	January	2011	100
Red wine	NULL	January	2011	2000
Red wine	1	February	2011	100
Red wine	2	February	2011	100
...	...	...	...	...
Red wine	28	February	2011	100
Red wine	NULL	February	2011	2000
...	...	...	...	...
Red wine	NULL	NULL	2011	24000
Red wine	NULL	NULL	NULL	24000

ROLLUP operator: Example (composite)

- request:

```
SELECT P_ProductCategory, P_Group, L_Country, L_Re
   SUM(S_Number) AS Sales
FROM Sale, Time, Product, Place
 WHERE S_Time_ID = T_ID AND S_Product_ID = P_ID
   AND S_Location_ID = L_ID AND
   YEAR(T_Date) = 2011
GROUP BY ROLLUP(P_ProductCategory, P_Group),
   ROLLUP(L_Country, (L_Region))
```

- evaluation:
  - 1. Rollup: `(P_ProductCategory,P_group), (P_ProductCategory),()`
  - 2. Rollup: `(L_Country,L_Region),(L_Country),()` *cross product of both combinations*

ROLLUP operator: Example (composite)

P_ProductCategory	P_Group	Country	Region	Sales
Wine	White wine	D	SANH	102
Wine	Red wine	D	SANH	98
Wine	NULL	D	SANH	200
...	...	...	...	...
Wine	White wine	D	NULL	541
wine	red wine	D	NULL	326
Wine	NULL	D	NULL	867
...	...	...	...	...
Wine	NULL	D	NULL	1232
...	...	...	...	...
...NULL	NULL	D	NULL	1432
...	...	...	...	...
...NULL	NULL	NULL	NULL	3456

CUBE vs. ROLLUP operator

- CUBE operator:
  - Generates all 2^n combinations:
    - e.g. for 4 grouping attributes 16 combinations
- ROLLUP operator:
  - Generates only combinations with super aggregates:
    - $(f_1, f_2, ..., f_n)$,
    - ...
    - $(f_1, NULL, ..., NULL)$,
    - $(NULL, NULL, ..., NULL)$
  - $n + 1$ combinations

GROUPING SETS

- SQL: 2003 grouping

```
GROUP BY ...  GROUPING SETS (grouping)
```

- grouping:
  - Simple grouping combination, e.g: `(L_Region, L_City)`
  - Complex grouping condition with **`CUBE`** or **`ROLLUP`**

GROUPING SETS: Example

- request

```
...
GROUP BY
   ROLLUP(P_ProductGroup, P_ProductCategory),(1)
   GROUPING SETS((L_City), (L_Province)),(2)
   GROUPING SETS(
      ROLLUP(year, quarter, month), (week))(3)
```

- meaning
  (1) along the classification hierarchy
  (2) only for cities and federal states
  (3) Using the parallel hierarchy `(year→quarter→month)` and `(week)`

Iceberg-Cube: Motivation

- Problems of the CUBE calculation (example)
  - 9-dimensional data set (data from weather stations)
  - $1,015,367$ tuples (about $39MB$)
  - CUBE: $210.343.580$ Tuple (approx. 8 GB $\approx 200\times$ input data)
  - 20% of all GROUP-BYs almost without aggregation (size: approx. 1)
  - calculation of GROUP-BYs with at least 2 input tuples: only $50\times$ input data!
  - For at least 10 tuples: only $5\times$ input data!
- idea Iceberg-Cube: Calculate only aggregations that have a have minimal support

Iceberg-Cube

- calculation of the groupings (partitions), which fulfill aggregate selection condition

```
SELECT A, B, C, COUNT(*), SUM(X)
FROM R
 GROUP BY CUBE(A, B, C)
HAVING COUNT(*) >= N
```

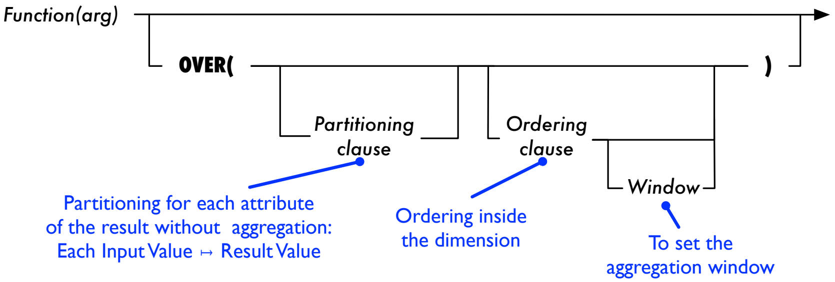
- N: minimum support of a partition
- Special optimization possible
  - Pruning: "Truncate" partitions that contain minimal Failure to meet support requirements

OLAP functions in SQL:2003

SQL:2003 – Sequence-based operations

- Since SQL:1999 – Extension with OLAP functions for attribute and sequence-based evaluation
- attribute- and tuple-based aggregation
- implementation in Oracle and DB2
- Supported query types
  - Ratio-To-Total
  - Current totals (accumulation)
  - Moving average
  - Ranking Analysis

OLAP functions: syntax



View definition for following examples

```
CREATE VIEW DailyTurnover AS
   SELECT P_ProductGroup, T_Date,
      SUM(S_Number * P_Sales price) AS Turnover
   FROM Sales, Time, Product
   WHERE S_Time_Id = T_Id AND S_Product_Id = P_Id
   GROUP BY P_ProductGroup, T_Date
```

OLAP functions: Motivation

- ratio-to-total analysis
  - calculation of the daily turnover on the total turnover of the month
  - Classical SQL query:

```
SELECT T_Date, Turnover,Total Turnover AS MonthTotal
   100.0*Turnover/Total Turnover AS Share,
FROM DailyTurnover,
   (SELECT SUM(Turnover) AS Total Turnover
   FROM DailyTurnover
   WHERE P_ProductGroup = 'Wine' AND
      YEAR_MONTH(T_Date) = 201108) Total
WHERE P_ProductGroup = 'Wine' AND
   YEAR_MONTH(T_Date) = 201108
```

  - Internal subquery calculates the total quantity for the Share calculation:

```
( SELECT SUM(Turnover) AS Total Turnover
  FROM DailyTurnover WHERE ...)
```

Formulation using OLAP function

- request:

```
SELECT T_Date, Turnover,
   100.0*Turnover/SUM(turnover) OVER() AS Share,
   SUM(Turnover) OVER() AS MonthTotal
FROM DailyTurnover
WHERE P_ProductGroup = 'Wine' AND
   YEAR_MONTH(T_Date) = 201108
```

- OLAP function

```
SUM(turnover) OVER()
```

  - aggregation over the entire entrance area
  - partition for aggregation is created locally for each entry generates

Result Relation

Date	Sales	Share	MonthTotal
01-AUG-2011	58	4,669	1242
02-AUG-2011	52	4,186	1242
03-AUG-2011	64	5,152	1242
04-AUG-2011	0	0,000	1242
...			
31-AUG-2011	47	3,784	1242
...			

Attribute local partitioning

- Partitioning of the input stream of an OLAP function (similar grouping)
- Partitioning is done per attribute/statement of the Aggregation operation
  - Enables regrouping
- example: Determination of the share of daily turnover in the Comparison to monthly sales

```
SELECT P_ProductGroup, T_Date, Sales,
   100.0*Turnover/SUM(turnover)
      OVER( PARTITION BY YEAR_MONTH(T_Date),
         P_ProductGroup) AS Monthly share,
   SUM(turnover)
      OVER( PARTITION BY YEAR_MONTH(T_Date),
         P_ProductGroup) AS MonthTotal
FROM DailyTurnover
```

Attribute local partitioning: Details

- principle:

```
SUM(quantity) OVER(PARTITION BY MONTH(T_Date))
```

- specification text after **OVER** means partitioning scheme
- No conflicts due to different partitioning schemes within a request
  - All entries of a partition included in calculation

Sequence-oriented analysis

- Specification of an attribute local order for partitions
- Application: running total, moving average, etc.
- Example: cumulated sales figures of the wines over total period and per month

```
SELECT T_Date,
   SUM(turnover) OVER(
         ORDER BY T_Date) AS TotalTotal,
   SUM(turnover) OVER(
         PARTITION BY YEAR_MONTH(T_Date)
         ORDER BY T_Date) AS TotalMonth
FROM DailyTurnover
WHERE P_ProductCategory = 'Wine
```

Sequence-oriented analysis: principle

- Number of tuples included in a result tuple Position of the tuple with respect to the given order
- input tuple t_i, result tuple s_i

$$\begin{array}{lclcl} t_1 & \longrightarrow & SUM(\{t_1\}) & \longrightarrow & s_1 \\ t_2 & \longrightarrow & SUM(\{t_1, t_2\}) & \longrightarrow & s_2 \\ t_3 & \longrightarrow & SUM(\{t_1, t_2, t_3\}) & \longrightarrow & s_3 \\ & & \ldots & & \end{array}$$

- Step by step enlargement of the analysis window

Use for ranking analyses

- Functions
  - **RANK**(): get rank of a tuple regarding the given order within the partition
    - For duplicates of equal rank (with gaps)
  - **DENSE_RANK**(): like **RANK**(), but without gaps
- example: Ranking by sales

```
SELECT T_Date, RANK()
   OVER(ORDER BY Turnover DESC) AS Rank
 FROM DailyTurnover
WHERE P_ProductGroup = 'Wine
```

Ranking analysis: Example

- restriction of "‘hitlists
- example: Top 3 days with the highest sales figures per month
- request:

```
SELECT P.T_Date, P.TopMonth
FROM (SELECT T_Date, P_ProductGroup,
      RANK() OVER(
         PARTITION BY YEAR_MONTH(T_Date)
         ORDER BY Sales DESC) AS Top Month
      FROM DailyTurnover) P
WHERE P.TopMonth <= 3 AND
   P.P_ProductGroup = 'Wine
ORDER BY P.TopMonth DESC
```

Formation of dynamic windows

- So far: only increasing window size for partition
- Now: explicit specification of the window
  - **ROWS**: Number of tuples
  - **RANGE**: Number of different tuples
- Application: moving average
- Starting from defined starting point to the current tuple
  - **UNBOUNDED PRECEDING**: first tuple of respective partition
  - n **PRECEDING**: n-ter predecessor relative to current position
  - **CURRENT ROW**: current tuple (only with **RANGE** and duplicates makes sense)

Building dynamic windows (2)

- specification of the lower and upper limits

```
BETWEEN lower limit AND upper limit
```

- Specification of the limits
  - **UNBOUNDED PRECEDING**
  - **UNBOUNDED FOLLOWING**
  - `n` **PRECEDING**
  - `n` **FOLLOWING**
  - **CURRENT ROW**
- `upper limit` must be higher than `lower limit` specify

Dynamic Windows: Example

- Moving Average with 5-day window on monthly level

```
SELECT T_Date, AVG(Sales) OVER(
      PARTITION BY YEAR_MONTH(T_Date)
      ORDER BY T_Date
      ROWS BETWEEN 2 PRECEDING
      AND 2 FOLLOWING) AS Through5days
FROM DailyTurnover
WHERE P_ProductCategory = 'Wine
```

Summary

- star join queries as a typical pattern of an SQL query
- special SQL extensions for groupings and aggregations