

Chapter 6 – Data Analysis in SQL

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

Neuere Entwicklungen für
Datenmodelle und
Anfragesprachen

1

Inhalt

Überblick

I. Objektorientierung und Erweiterbarkeit

1. Benutzerdefinierte Datentypen und getypte Tabellen
2. Objekt-relationale Sichten und Kollektionstypen
3. Benutzerdefinierte Routinen und Objektverhalten
4. Anbindung an Anwendungsprogramme
5. Objekt-relationales SQL und Java

II. Online Analytic Processing

6. **Datenanalyse in SQL**
7. Windows und Query Functions

III. XML

8. XML und Datenbanken
9. SQL/XML
10. Xquery

On-line Analytic Processing (OLAP)

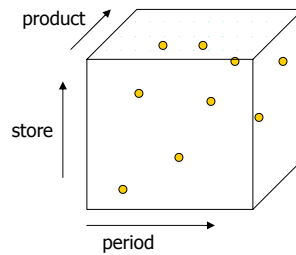
- Goal: analyze all (or large amounts of) company data to discover patterns or trends of importance
- Queries (OLAP queries, decision support queries) are very complex
 - joins, aggregations, ...
- Often based on an integrated data warehouse
 - organize and centralize corporate data from various sources
 - can involve additional integration work, data cleansing and normalization
 - provide an execution platform for complex, potentially long-running OLAP queries without slowing down regular transaction processing systems
 - periodically refreshed from the various data sources
- In contrast to On-Line Transaction Processing (OLTP) applications

OLTP vs. OLAP

OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
many simple queries	few (repeated) complex queries
small amounts of data accessed	large amounts of data accessed
usually operate on the current data	operate on current and historical data
optimize for fast update, high throughput	optimize for fast calculation
→ hard (sometimes impossible) for DBMS to optimize for both OLTP and OLAP at the same time	
Parallel execution of OLAP and OLTP queries on operational database may impact OLTP performance	

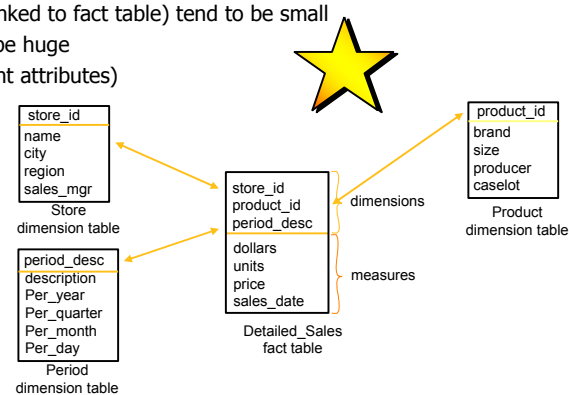
Multidimensional View of OLAP Data

- Fact Table
 - central relation or collection of data in an OLAP application
 - represents events or objects of interest
 - e.g., a sales event, with information about the product sold, the store, the sales date and price
 - objects can often be thought of as arranged in a multi-dimensional space, or **cube**
 - e.g., sales events have store, product, and time period dimensions
 - each point is a single sales event, dimensions represent properties of the sale



(R)OLAP Schema

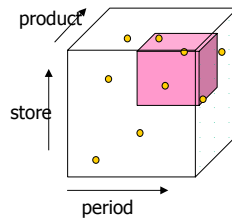
- Typically uses a **"STAR"** structure
 - Dimension tables (linked to fact table) tend to be small
 - Fact table tends to be huge
 - Measures (dependent attributes)
- OLAP queries usually
 - GROUP BY the dimensions
 - compute aggregate values of measures



CREATE VIEW Sales AS
 (SELECT ds.*, YEAR (sales_date) AS year, MONTH (sales_date) AS month, DAY (sales_date) AS day
 FROM (Detailed_Sales NATURAL JOIN Store NATURAL JOIN Product NATURAL JOIN Period) ds

Slicing and Dicing

- Think of the set of points in the cube as partitioned along each dimension at some level of granularity
 - brand, size, producer for product
 - name, city, region for store
 - month, week, year for time period
- Choice of partition (GROUP BY) for each dimension "**dices**" the cube
 - divides cube into smaller cubes that contain points whose measure are aggregated
- Focusing on particular partions (through WHERE clause) "**slices**" the cube
 - selects partitions along one or more dimensions
- General form
 - SELECT** grouping attributy and aggregates
 - FROM** fact table joined with zero or more dimension tables
 - WHERE** certain attributes are constant
 - GROUP BY** grouping attributes



Neuere Entwicklungen für
Datenmodelle und
Anfragesprachen

Drill-down, Roll-Up and Data Cubes

- Drill-down, Roll-up: common patterns in sequences of queries that slice & dice
 - Drill-down gradually partitions more finely or focuses on specific values in the dimensions
 - Example:

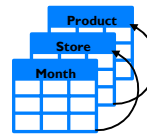
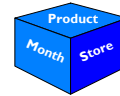

```
SELECT month, city, producer, SUM(units) AS sum_units
FROM Sales
WHERE year = 1998
GROUP BY month, city, producer
```



```
SELECT month, city, producer, SUM(units) AS sum_units
FROM Sales
WHERE year = 1998
      AND region = 'Central'
GROUP BY month, city, producer
```
 - Roll-up gradually partitions more coarsely
- Data Cube operator performs systematic (pre-)aggregation along all dimensions

SQL99 OLAP SQL Extensions

- Extension to GROUP BY clause
- Produces "super aggregate" rows
- ROLLUP equivalent to "control breaks"
- CUBE equivalent to "cross tabulation"
- GROUPING SETS equivalent to multiple GROUP BYs
- Provides "data cube" collection capability
 - Often used with data visualization tool



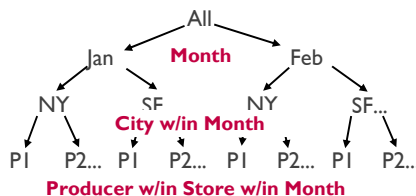
ROLLUP

- Extends grouping semantics to produce "subtotal" rows
 - Produces "regular" grouped rows
 - Produces same groupings reapplied down to grand total



```

SELECT month, city, producer, SUM(units) AS sum_units
FROM Sales
WHERE year = 1998
GROUP BY ROLLUP (month, city, producer)
    
```



ROLLUP

- Find the total sales per region and sales manager during each month of 1996, with subtotals for each month, and concluding with the grand total:

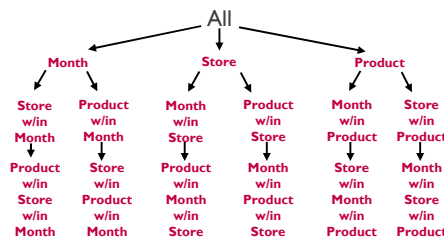
```
SELECT month, region, sales_mgr, SUM(price)
FROM Sales
WHERE year = 1996
GROUP BY ROLLUP
(month, region, sales_mgr)
```

MONTH	REGION	SALES_MGR	SUM(price)
April	Central	Chow	25 000
April	Central	Smith	15 000
April	Central	-	40 000
April	NorthWest	Smith	15 000
April	NorthWest	-	15 000
April	-	-	55 000
May	Central	Chow	25 000
May	Central	-	25 000
May	NorthWest	Smith	15 000
May	NorthWest	-	15 000
May	-	-	40 000
			95 000

CUBE

- Further extends grouping semantics to produce multidimensional grouping and "subtotal" rows
 - Superset of ROLLUP
 - Produces "regular" grouped rows
 - Produces same groupings reapplied down to grand total
 - Produces additional groupings on all variants of the CUBE clause

```
SELECT month, city, product_id, SUM(units)
FROM Sales
WHERE year = 1998
GROUP BY CUBE (month, city, product_id)
```



... GROUP BY CUBE

```
SELECT month, region, sales_mgr, SUM(price)
FROM Sales
WHERE year = 1996
GROUP BY CUBE (month, region, sales_mgr)
```

MONTH	REGION	SALES_MGR	SUM(price)
April	Central	Chow	25 000
April	Central	Smith	15 000
April	Central	-	40 000
April	NorthWest	Smith	15 000
April	NorthWest	-	15 000
April	-	Chow	25 000
April	-	Smith	30 000
April	-	-	55 000
May	Central	Chow	25 000
May	Central	-	25 000
May	NorthWest	Smith	15 000
May	NorthWest	-	15 000
May	-	Chow	25 000
May	-	Smith	15 000
May	-	-	40 000
-	Central	Chow	50 000
-	Central	Smith	15 000
-	Central	-	65 000
-	NorthWest	Smith	30 000
-	NorthWest	-	30 000
-	-	Chow	50 000
-	-	Smith	45 000
-	-	-	95 000

GROUPING SETS

- Multiple "groupings" in a single pass
 - Used in conjunction with usual aggregation (MAX, MIN, SUM, AVG, COUNT, ...)
 - Allows multiple groups e.g. (month, region) and (month, sales_mgr)
 - Result can be further restricted via HAVING clause
- Example:
Find the total sales during each month of 1996, per region and per sales manager:

```
SELECT month, region, sales_mgr,
       SUM(price)
```

```
FROM Sales
```

```
WHERE year = 1996
```

```
GROUP BY
```

```
GROUPING SETS ((month, region),
               (month, sales_mgr))
```

MONTH	REGION	SALES_MGR	SUM(price)
April	Central	-	40 000
April	NorthWest	-	15 000
April	-	Chow	25 000
April	-	Smith	30 000
May	Central	-	25 000
May	NorthWest	-	15 000
May	-	Chow	25 000
May	-	Smith	15 000

Generating Grand Total Rows

- Special syntax available to include a "grand total" row in the result
 - Grand totals are generated implicitly with ROLLUP and CUBE operations
 - Syntax allows grand totals to be generated without additional aggregates
- Get total sales by month, region, and sales manager and also the overall total sales:

```
SELECT month, region, sales_mgr, SUM (price)
FROM Sales
WHERE year = 1996
GROUP BY GROUPING SETS ( (month, region, sales_mgr), ( ) )
```

The GROUPING Function

- New column function
 - Allows detection of rows that were generated during the execution of CUBE and ROLLUP i.e. generated nulls to be distinguished from naturally occurring ones
- Example:
Run a rollup, and flag the generated rows...

```
SELECT month, region, sales_mgr, SUM(price), GROUPING(sales_mgr) AS
grouping
FROM Sales
WHERE year = 1996
GROUP BY ROLLUP (month, region, sales_mgr)
```


Result...

MONTH	REGION	SALES_MGR	SUM(price)	GROUPING
April	Central	Chow	25 000	0
April	Central	Smith	15 000	0
April	Central	-	40 000	1
April	NorthWest	Smith	15 000	0
April	NorthWest	-	15 000	1
April	-	-	55 000	1
May	Central	Chow	25 000	0
May	Central	-	25 000	1
May	NorthWest	Smith	15 000	0
May	NorthWest	-	15 000	1
May	-	-	40 000	1
			95 000	1

Selecting Nongrouped Columns

- Nongrouped columns can sometimes be selected based on functional dependencies:

```

SELECT e.deptno, d.location, AVG (e.salary) AS average
FROM Emp e, Dept d
WHERE e.deptno = d.deptno
GROUP BY e.deptno
    
```



e.deptno determines d.deptno (equals in WHERE clause), and d.deptno determines d.location (deptno is PK of Dept); therefore, d.deptno and d.location are consistent within any group. This is functional dependency analysis in action.



```

SELECT e.deptno, e.name, AVG (e.salary) AS Average
FROM Emp e, Dept d
WHERE e.deptno =d.deptno
GROUP BY e.deptno
    
```