ifis

Institut für Informationssysteme
Technische Universität Braunschweig

Data Warehousing & Mining Techniques

Wolf-Tilo Balke
Muhammad Usman
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de

6. OLAP Operations & Queries

6.1 OLAP Operations

6.2 OLAP Queries: SQL 99, MDX

```
(SELECT lead_in, ROW_NUMBER()
        OVER (PARTITION BY zip_code
              ORDER BY lead_date)
        AS lead_link
 FROM Leads) AS L
FULL OUTER JOIN
(SELECT dealer_id, ROW_NUMBER()
        OVER (PARTITION BY zip_code
              ORDER BY dealer_priority DESC)
        AS dealer_link
 FROM Dealers) AS D
ON D.dealer_link = L.lead_link AND D.zip_code = L.zipcode;
```

6.0 DW Queries

- DW queries are **big queries**
  - Imply a large portion of the data
  - Mostly read queries
- Redundancy a necessity
  - Materialized views, special-purpose indexes, de-normalized schemas
- Data is refreshed periodically
  - Daily or weekly
- Their purpose is to analyze data
  - OLAP (OnLine Analytical Processing)

6.0 DW Queries

- OLAP usage fields
  - Management Information
    - Sales per product group / area / year
  - Government
    - Population census
  - Scientific databases
    - Geo-, Bio-Informatics
  - Etc.
- Goal: Response time of seconds / few minutes

6.1 OLAP Operations

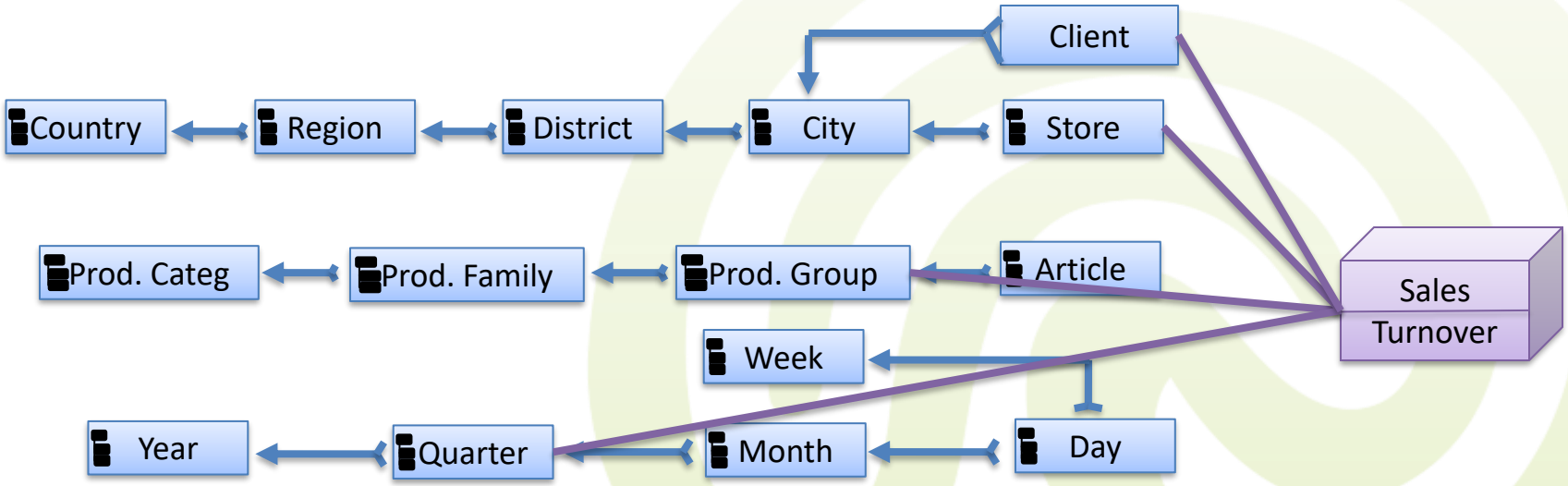
- Typical OLAP operations
  - Roll-up
  - Drill-down
  - Slice and dice
  - Pivot (rotate)
- Other operations
  - Aggregate functions
  - Ranking and comparing
  - Drill-across
  - Drill-through

6.1 Roll-up

- Roll-up (drill-up)
  - Taking the current aggregation level of fact values and doing a **further aggregation**
  - **Summarize** data by
    - **Climbing up hierarchy** (hierarchical roll-up)
    - By **dimensional reduction**
    - Or by a mix of these 2 techniques
  - Used for obtaining an increased generalization
    - E.g., from Time.Week to Time.Year

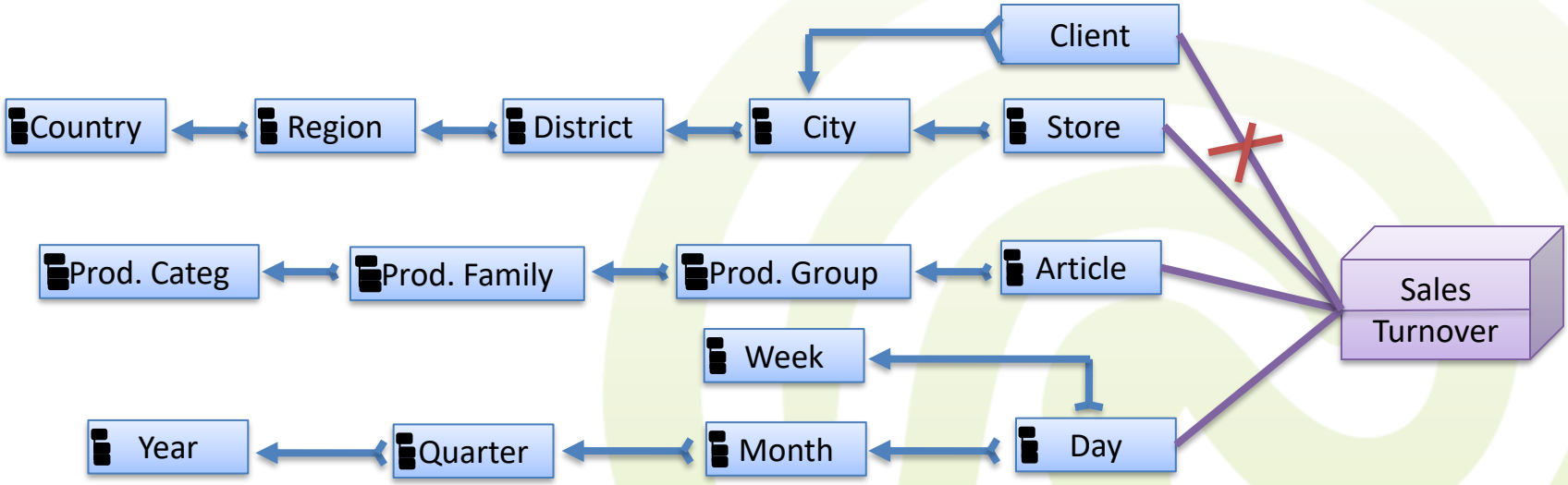
6.1 Roll-up

- **Hierarchical roll-ups**
  - Performed on the **fact table and some dimension tables** by **climbing up** the attribute hierarchies
    - E.g., climbed the Time hierarchy to Quarter and Article hierarchy to Prod. group



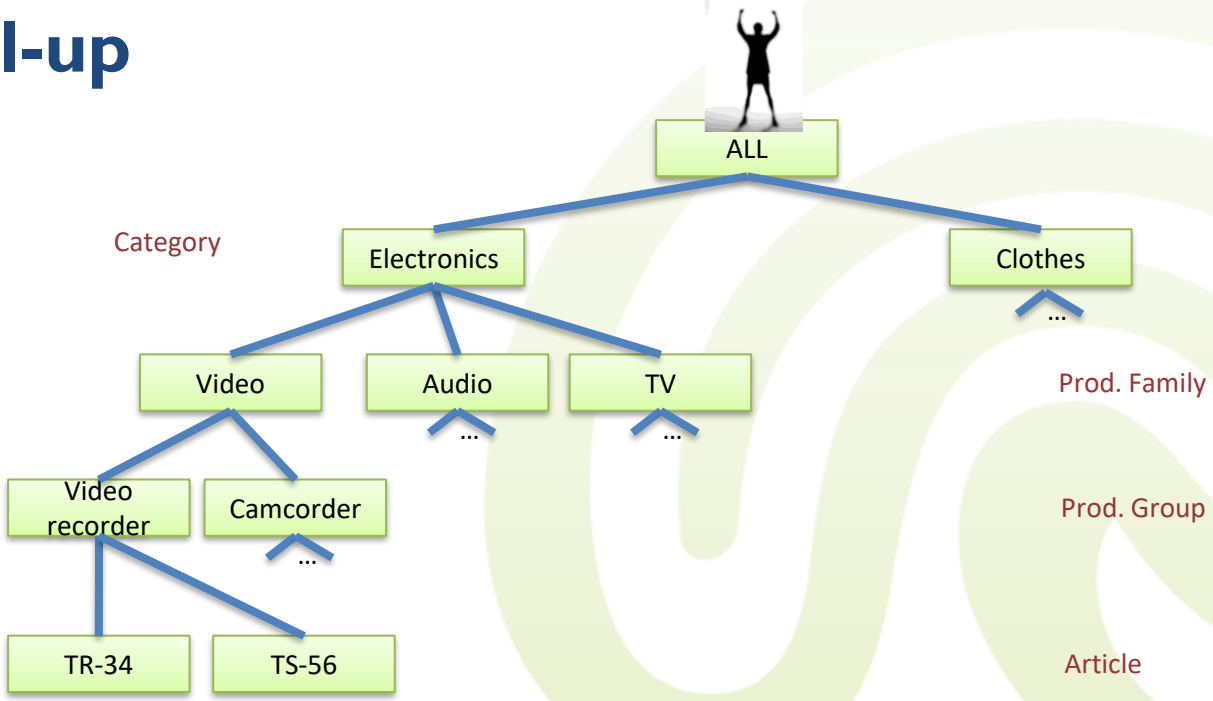
6.1 Roll-up

- **Dimensional roll-ups**
  - Are done solely on the **fact table** by **dropping** one or more dimensions
    - E.g., drop the Client dimension



6.1 Roll-up

- Climbing above the top in hierarchical roll-up
  - In an ultimate case, **hierarchical roll-up** above the top level of an attribute hierarchy (attribute "ALL") **can be viewed** as converting to a **dimensional roll-up**



6.1 Drill-down

- Drill-down (roll-down)
  - **Reverse of roll-up**
  - Represents a **de-aggregate** operation
    - From higher level of summary to lower level of summary – detailed data
  - Introducing new dimensions
  - Requires the **existence** of materialized **finer grained data**
    - **You can't drill if you don't have the data**

6.1 Roll-up Drill-down Example

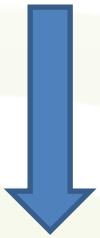
€ by BAR/Time

	Week1	Week2	Week3
Joe's	450	330	300
Salitos	500	360	420
Roots	380	310	400

Roll-up by BAR

€ by Time

Week1	Week2	Week3
1330	1000	1120



Drill-down by Brand

€ by Brand/Time

	Week1	Week2	Week3
Wolters	480	400	400
Becks	450	310	370
Krombacher	400	290	350

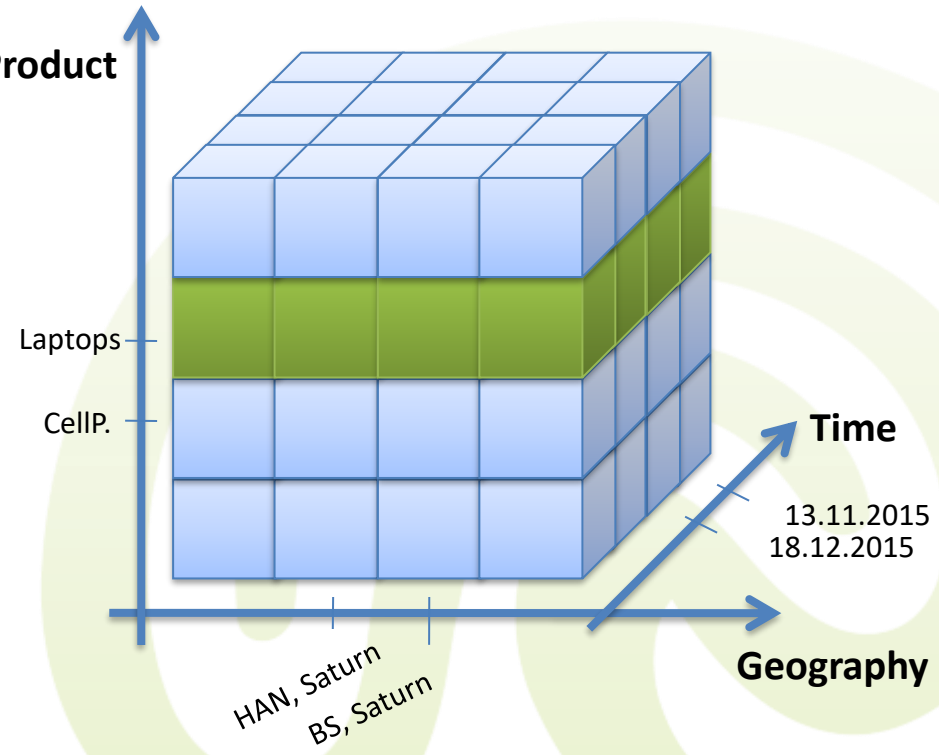
6.1 Slice

- Slice: a **subset** of the multi-dimensional array corresponding to **a single value** of one or more dimensions **and projection on the rest** of dimensions
  - E.g., project on Geo (store) and Time from values corresponding to Laptops in the product dimension

$$\pi_{\text{StoreId,TimeId,Amount}}(\sigma_{\text{ArticleId=LaptopId}}(\text{Sales}))$$

6.1 Slice

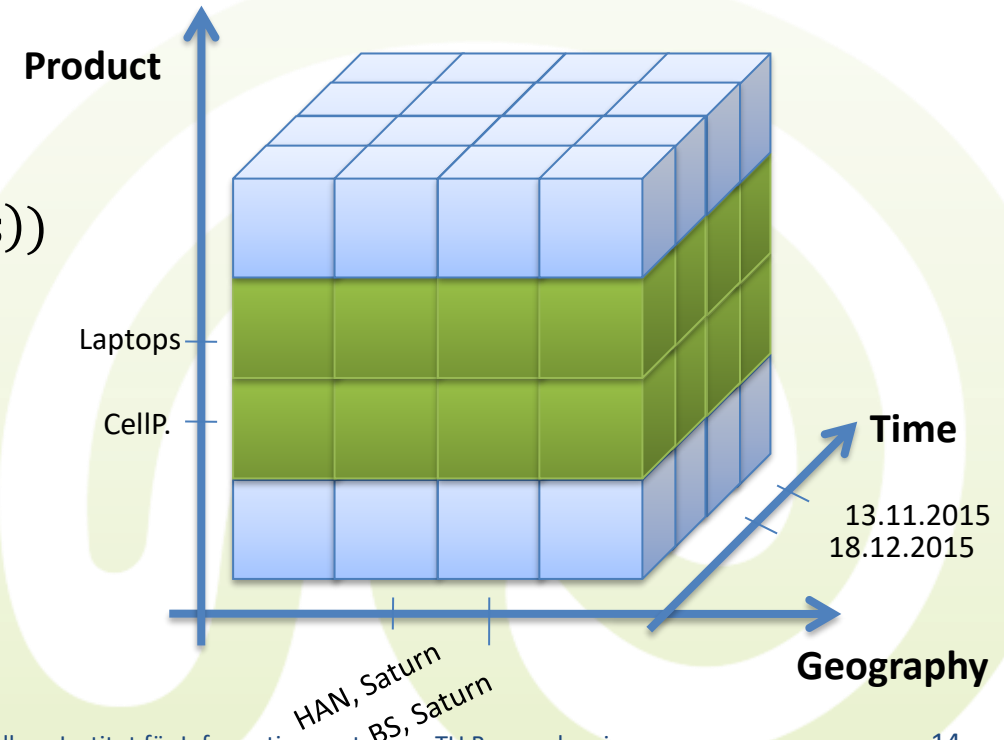
- Amounts to equality select condition
- WHERE clause in SQL
  - E.g., slice Laptops



6.1 Dice

- Dice: amounts to **range select condition** on one dimension, or to **equality select condition** on **more** than one dimension
  - E.g. range SELECT

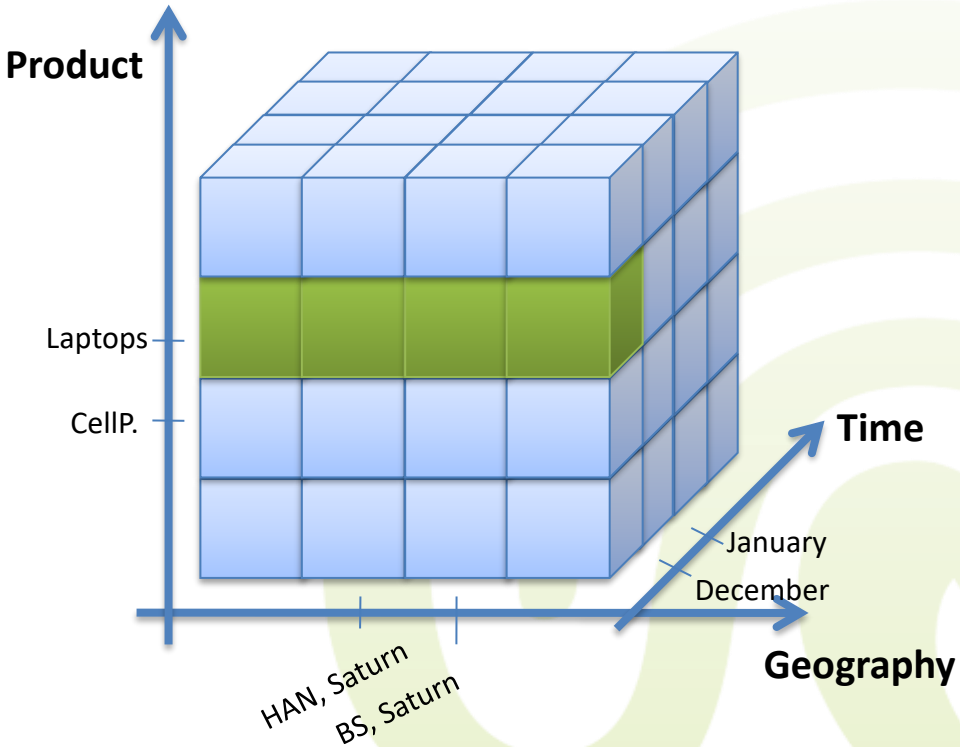
$$\pi_{\text{StoreID,TimeId,Amount}}(\sigma_{\text{ArticleId} \in \{\text{Laptop,CellP}\}}(\text{Sales}))$$



6.1 Dice

- E.g. equality SELECT on 2 dimensions Product and Time

$$\pi_{\text{StoreId,Amount}}(\sigma_{\text{ArticleId=Laptop} \wedge \text{MonthId=December}}(\text{Sales}))$$



6.1 Pivot

- Pivot (rotate): re-arranging data for viewing purposes
  - The simplest view of pivoting is that it selects two dimensions to **aggregate the measure**
    - The aggregated values are often displayed in **a grid** where each point in the (x, y) coordinate system corresponds to an aggregated value of the measure
    - The x and y coordinate values are the values of the selected two dimensions
  - The result of pivoting is also called **cross–tabulation**

6.1 Pivot

- Consider pivoting the following data

Location	
CityId	City
1	Braunschweig
2	Hannover
3	Hamburg

Sales			
CityId	PerId	TimId	Amnt
1	1	1	230
1	1	2	300
1	1	8	310
1	2	7	50
2	3	1	550
2	3	5	100
3	4	6	880
3	5	1	60
3	5	2	60
3	5	4	140

Time	
TimId	Day
1	Mon
2	Tue
3	Wed
4	Thu
5	Fri
6	Sat
7	San
8	Mon

6.1 Pivot

- Pivoting on City and Day

	Mon	Tue	Wed	Thu	Fri	Sat	San	SubTotal
Hamburg	60	60	0	140	0	880	0	1140
Hannover	550	0	0	0	100	0	0	650
Braunschweig	540	300	0	0	0	0	50	890
SubTotal	1150	360	0	140	100	880	50	2680

	Hamburg	Hannover	Braunschweig	SubTotal
Mon	60	550	540	1150
Tue	60	0	300	360
Wed	0	0	0	0
Thu	140	0	0	140
Fri	0	100	0	100
Sat	880	0	0	880
San	0	0	50	50
SubTotal	1140	650	890	2680

6.1 Typical Analytical Requests

- OLAP operations are hard to express in query languages
  - Most analysts and decision makers won't enjoy it

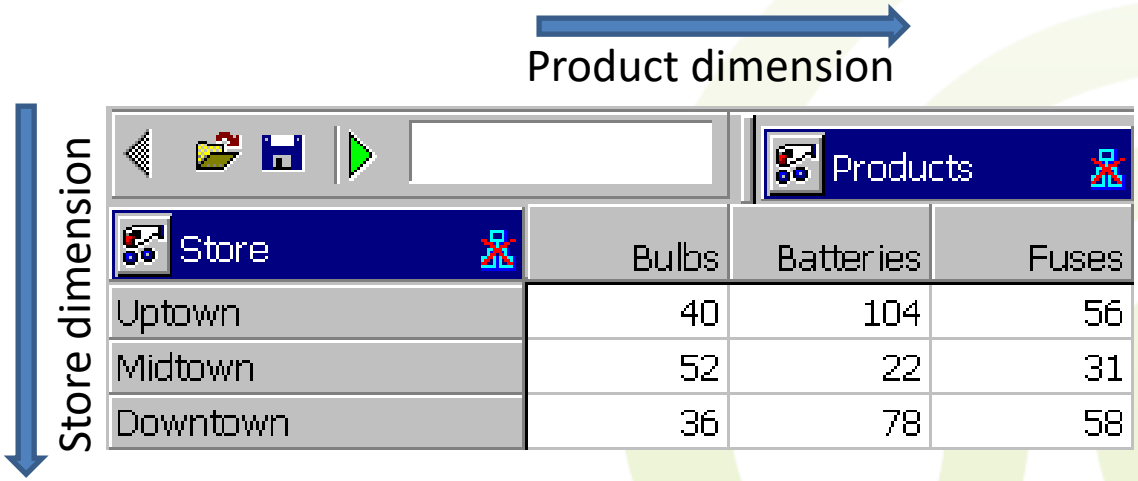
```
SELECT  f.region, z.month, sum(a.price * a.volume)
FROM    Order a, Time z, PoS f
WHERE   a.pos = f.name AND a.date = z.date
GROUP BY f.region, z.month
```

  - OLAP clients allow operations to be performed through GUIs

QUARTER	Store Name:	PRODTYPE	Quantity:	Line Cost Of Goods Sold
Q2	Audio Expert	Digital	111,421	28,064,250.00
Q1	Audio Expert	Digital	105,983	25,092,678.00
Q1	eMart	Digital	108,221	24,990,368.00
Q2	eMart	Digital	115,102	24,971,512.00
Q1	eMart	Analog	97,128	21,152,262.00
Q2	eMart	Analog	74,737	16,789,403.00
Q1	Audio Expert	Analog	78,449	16,467,146.00
Q4	eMart	Digital	72,126	14,000,951.00
Q3	eMart	Digital	66,156	13,867,709.00
Q2	Audio Expert	Analog	57,944	11,868,758.00
Q3	Audio Expert	Digital	50,076	11,210,406.00
Q4	Audio Expert	Digital	53,275	11,190,923.00
Q1	TV City	Digital	41,307	10,128,967.00
Q4	eMart	Analog	39,515	9,383,389.00
Q3	eMart	Analog	36,306	8,308,647.00
Q2	TV City	Digital	29,627	6,732,303.00
Q2	AV VideoTown	Digital	27,377	5,928,507.00
Q4	Audio Expert	Analog	25,897	5,916,936.00

6.1 OLAP Data Visualization

- How do these operations look like for the user?
  - E.g. Crystal Decisions later bought by SAP and integrated into Business Objects
    - 2 dimensions … is trivial
    - E.g. Products by Store

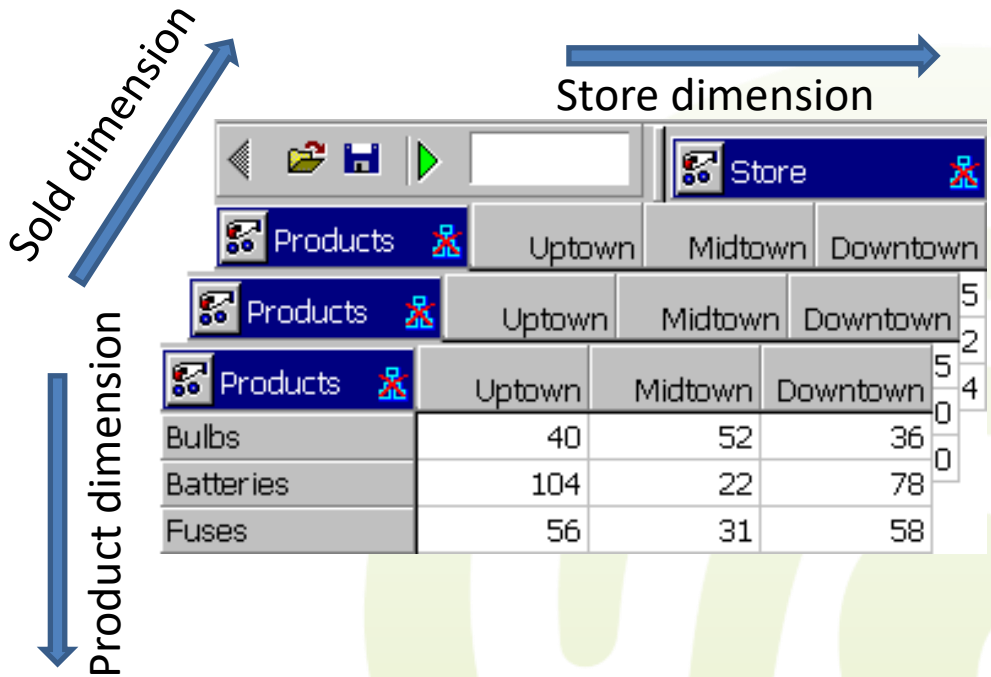


crystal decisions™
A SEAGATE COMPANY

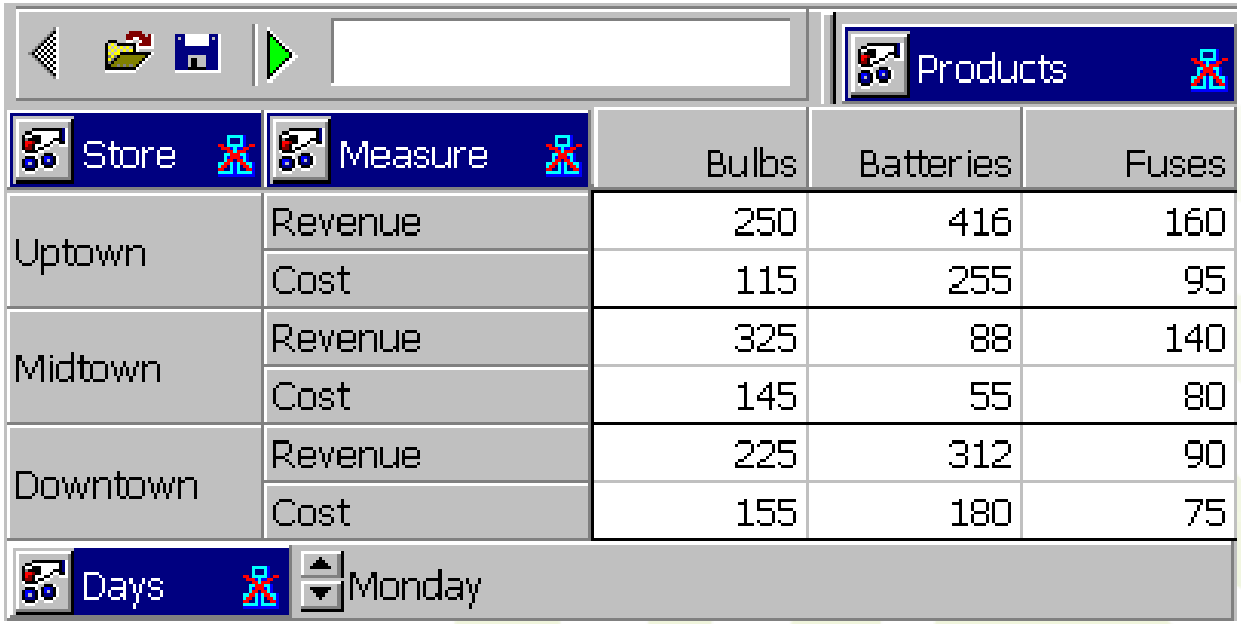
6.1 OLAP Data Visualization

Detour

- 3 dimensions: We can visualize sold quantity on 3 dimensions as **layers**



- Another way is by **nesting** on the same axis



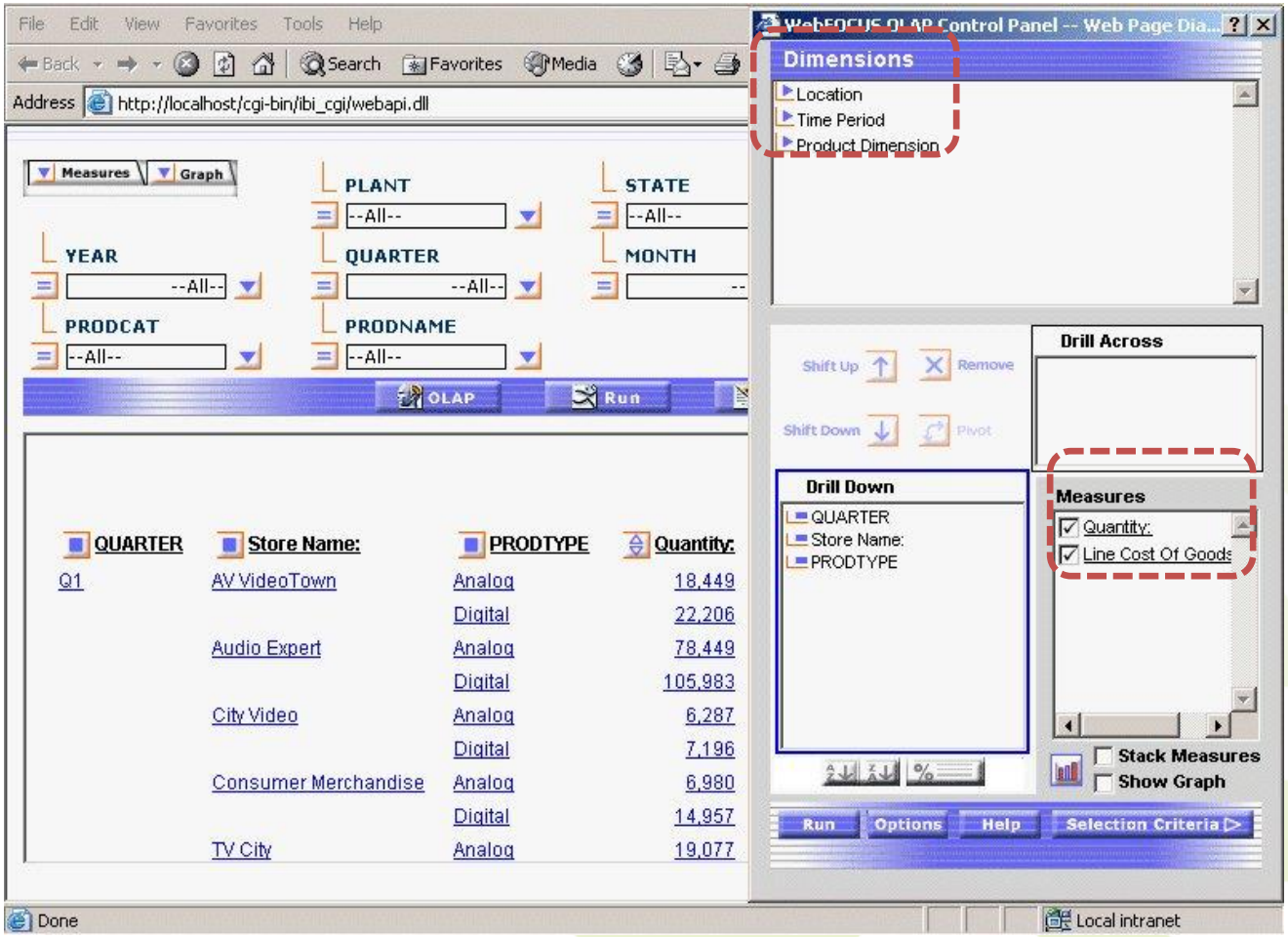
Store	Measure	Products: Bulbs	Products: Batteries	Products: Fuses
Uptown	Revenue	250	416	160
	Cost	115	255	95
Midtown	Revenue	325	88	140
	Cost	145	55	80
Downtown	Revenue	225	312	90
	Cost	155	180	75

Days: Monday

6.1 OLAP Data Visualization

Detour

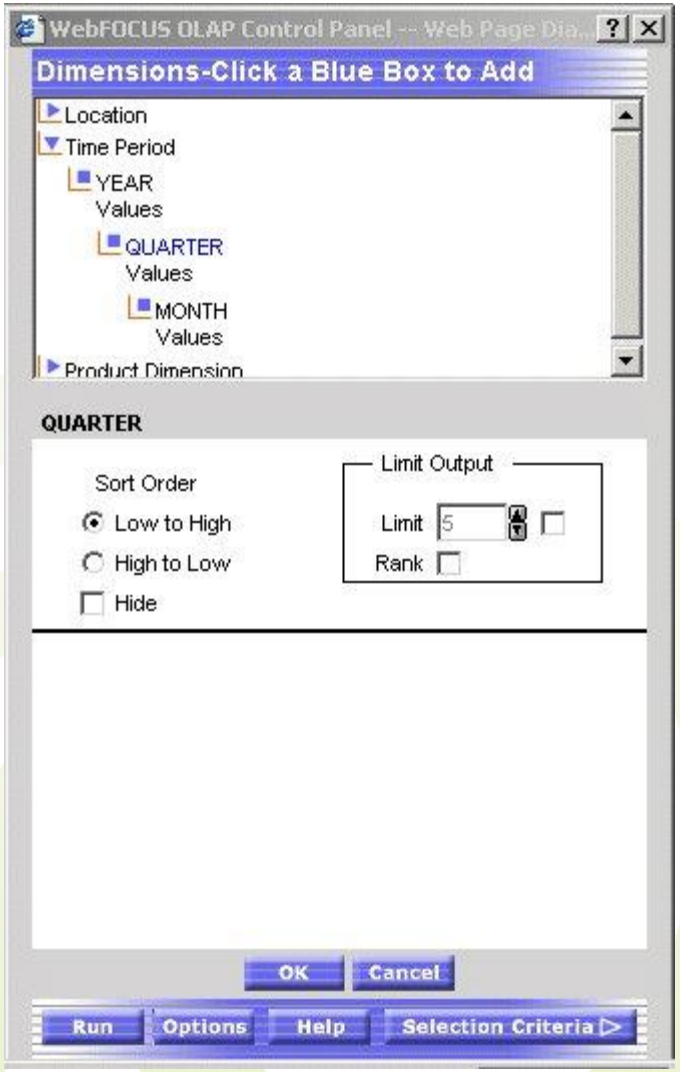
- OLAP reporting has to be very flexible
  - The IBM Infosphere - OLAP web based report



6.1 OLAP Data Visualization

Detour

- **Drill-down** operation
  - Can be performed easily
    by going down the hierarchy
    and choosing the granularity



6.1 OLAP Data Visualization

Detour

- Trends Visualization
  - With the help of charts

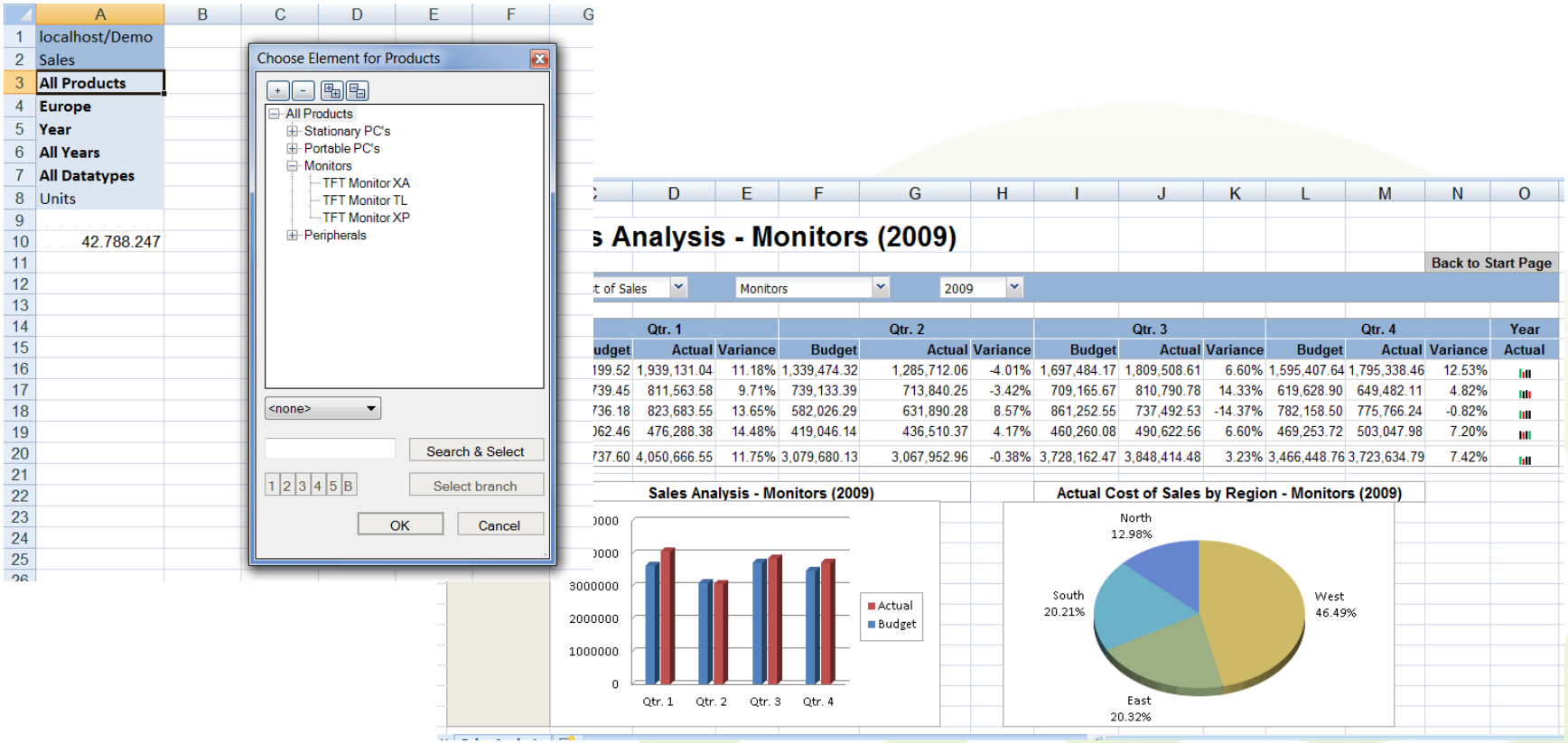
Click to sort

QUARTER	Store Name:	PRODTYPE	Quantity:	Line Cost Of Goods Sold
Q1	AV VideoTown	Analog	18,449	3,969,296.00
		Digital	22,206	5,109,400.00
	Audio Expert	Analog	78,449	16,467,146.00
		Digital	105,983	25,092,678.00
	City Video	Analog	6,287	1,315,015.00
		Digital	7,196	1,607,513.00
	Consumer Merchandise	Analog	6,980	1,542,036.00
		Digital	14,957	3,251,090.00
	TV City	Analog	19,077	3,772,119.00
		Digital	41,307	10,128,967.00
	Web Sales	Analog	545	124,366.00
		Digital	829	190,201.00
	eMart	Analog	97,128	21,152,262.00
		Digital	108,221	24,990,368.00
Q2	AV VideoTown	Analog	11,781	2,663,655.00
		Digital	27,377	5,928,507.00
	Audio Expert	Analog	57,944	11,868,758.00
		Digital	111,421	28,064,250.00

6.1 OLAP Data Visualization

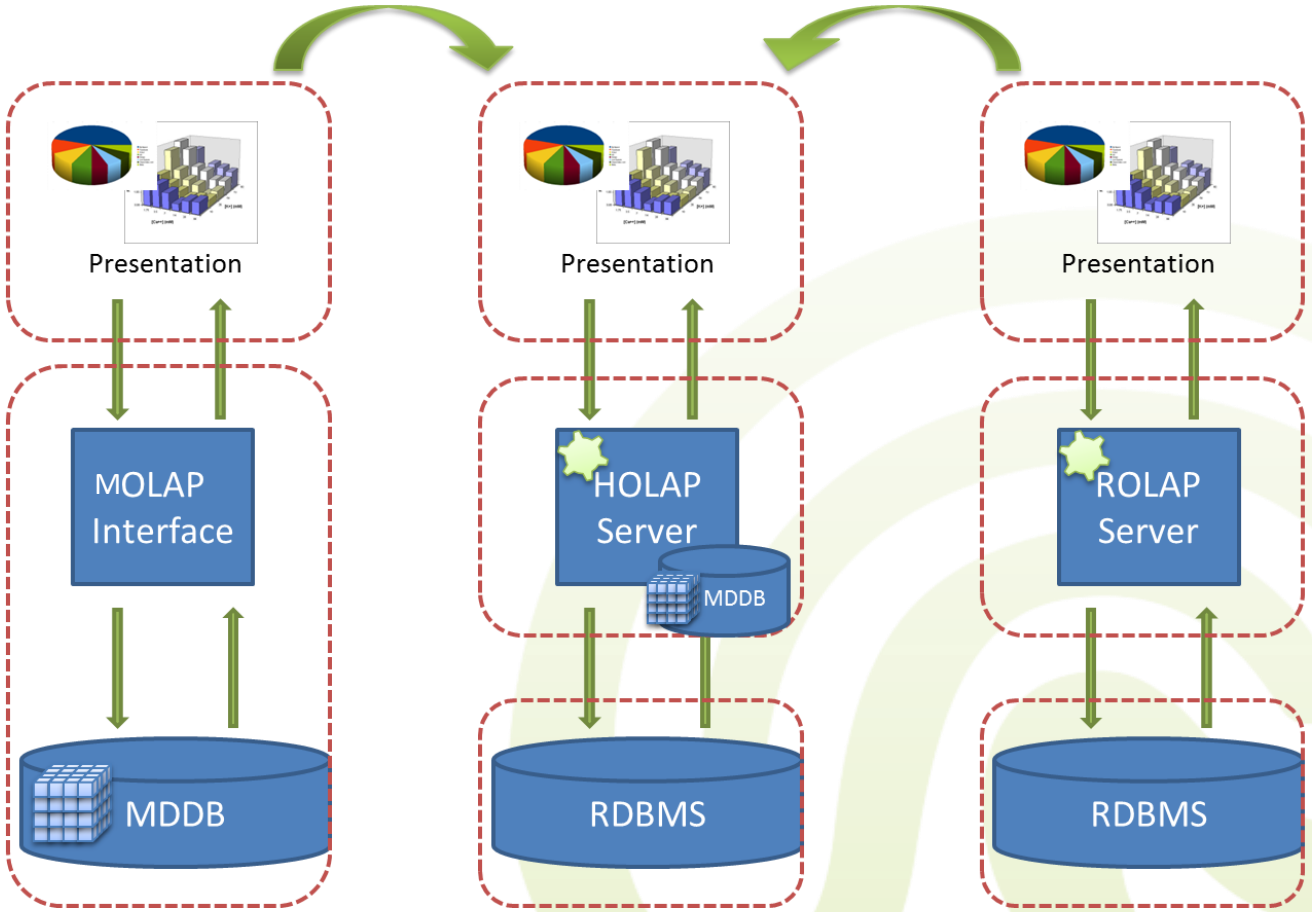
Detour

- Palo Technologies, integrated into Excel
  - Cubes are defined in a Web interface



6.1 From Presentation to Data

- Remember the DW architecture?



6.1 From Presentation to Data

- Client/server architecture
  - The client **displays** reports and **allows interaction** with the end user to perform the **OLAP operations** and other custom queries
  - The server is responsible for providing the requested data. How? It depends on whether it is MOLAP, ROLAP, HOLAP, etc.

6.1 OLAP Server

- High-capacity, multi-user **data manipulation engine** specifically designed to **support and operate on multidimensional** data structures
- It is optimized for fast, flexible calculation and transformation of raw data based on formulaic relationships

6.1 OLAP Server

- OLAP server may either
  - **Physically stage** the processed multidimensional information to deliver consistent and rapid response times to end users **(MOLAP)**
  - Store data in **relational databases** and **simulate multidimensionality** with special schemas **(ROLAP)**
  - Or offer a **choice of both (HOLAP)**

6.1 From Presentation to Data

- Getting from OLAP operations to the data
  - As in the relational model, **through queries**
    - In OLTP we have SQL as the standard query language
  - However, OLAP operations are hard to express in SQL
  - There is **no standard** query language for OLAP
  - Choices are:
    - **SQL-99** for ROLAP
    - **MDX** (Multidimensional expressions) for both MOLAP and ROLAP

6.2 Typical OLAP Queries

- The idea is to
  - Select by Attributes of Dimensions
    - E.g., region = „Europe“
  - Group by Attributes of Dimensions
    - E.g., region, month, quarter
  - Aggregate on measures
    - E.g., sum(price * volume)
- OLAP queries in SQL

SELECT $d_1.x, d_2.y, d_3.z, \text{sum}(f.t_1), \text{avg}(f.t_2)$
FROM Fact f, Dim1 d_1, Dim2 d_2, Dim3 d_3
WHERE $a < d_1.\text{field} < b$ AND $d_2.\text{field} = c$
GROUP BY $d_1.x, d_2.y, d_3.z$;

6.2 OLAP Query Languages

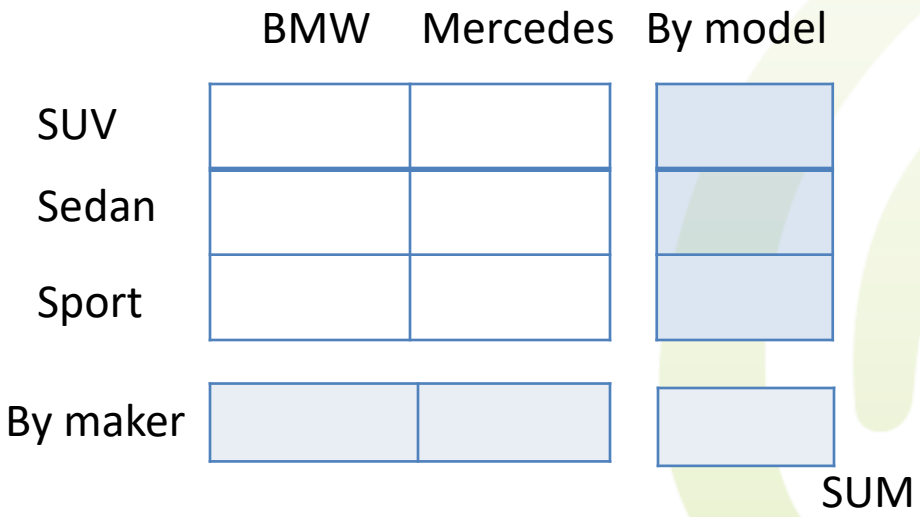
- SQL-99
  - Prepare SQL for OLAP queries
  - New SQL commands
    - GROUPING SETS
    - ROLLUP
    - CUBE
  - New aggregate functions
  - Queries of type "top k"

6.2 SQL-92

- Shortcomings of SQL/92 with regard to OLAP queries
  - Hard or impossible to express in SQL
    - Multiple aggregations
    - Comparisons (with aggregation)
    - Reporting features
  - Performance penalty
    - Poor execution of queries with many AND and OR conditions
  - Lack of support for advanced statistical functions

6.2 SQL-92

- Multiple aggregations in SQL/92
  - Create a 2D spreadsheet that shows sum of *sales* by *maker* as well as *car model*
  - Each subtotal requires a separate aggregation query




```
SELECT model, maker, sum(amt)
FROM sales GROUP BY model,
maker
union
SELECT model, sum(amt)
FROM sales GROUP BY model
union
SELECT maker, sum(amt)
FROM sales GROUP BY maker
union
SELECT sum(amt) FROM sales
```

6.2 SQL-92

- Comparisons in SQL/92
  - This year's sales vs. last year's sales for each product
    - **Requires a self-join**
    - CREATE VIEW v_sales AS SELECT prod_id, year, sum(qty) AS sale_sum FROM sales GROUP BY prod_id, year;
    - SELECT cur.prod_id, cur.year, cur.sale_sum, last.year, last.sale_sum FROM v_sales cur, v_sales last WHERE cur.year = (last.year+1) AND cur.prod_id = last.prod_id;

6.2 SQL-92

- Reporting features in SQL/92
  - Too complex to express
    - RANK (top k) and NTILE ("top X%" of all products)
    - Median
    - Running total, moving average, cumulative totals
  - E.g. **moving average** over a 3 day window of total sales for each product
    - CREATE OR REPLACE VIEW v_sales AS SELECT prod_id, time_id, sum(qty) AS sale_sum FROM sales GROUP BY prod_id, time_id;
    - SELECT end.time, avg(start.sale_sum) FROM v_sales start, v_sales end WHERE end.time >= start.time AND end.time <= start.time + 2 GROUP BY end.time;

6.2 OLAP Functions

Detour

- **Moving-averages**, **Percentiles**, **Ranks** are all hard to compute with SQL-92
  - Involves multiple self joins of the fact table
- SQL-99 introduced the **window clause** for creating dynamical windows of data

6.2 Window Clause

Detour

- The window clause specifies an **action** to perform **over a set of rows**
  - 3 sub-clauses: **Partitioning, ordering** and **aggregation grouping**
  - <aggregate function> OVER ([**PARTITION** BY <column list>] **ORDER BY** <sort column list> [<**aggregation grouping**>])
    - SELECT … AVG(sales) OVER (PARTITION BY region ORDER BY month ASC ROWS 2 PRECEDING) AS SMA3…

    (moving average of 3 rows)

6.2 Ranking in SQL

Detour

- **Ranking** operators in SQL
  - Row numbering is the most basic ranking function
    - Old style: ROW_NUMBER() returns a column that contains the row's number within the result set
    - E.g., SELECT SalesOrderID, CustomerID, ROW_NUMBER() OVER (ORDER BY SalesOrderID) as RunningCount FROM Sales WHERE SalesOrderID > 10000 ORDER BY SalesOrderID;

SalesOrderID	CustomerID	RunningCount
43659	543	1
43660	234	2
43661	143	3
43662	213	4
43663	312	5

6.2 Ranking in SQL

Detour

- ROW_NUMBER doesn't consider **tied values**
  - Each 2 equal values get 2 different row numbers

SalesOrderID	RunningCount
43659	1
43659	2
43660	3
43661	4

  - The behavior is **non-deterministic**
    - Each tied value could have its number switched!
- **We need something deterministic**

6.2 Ranking in SQL

Detour

- RANK and DENSE_RANK functions
  - Allow ranking items in a group
  - Syntax:
    - RANK ( ) OVER ([query_partition_clause] order_by_clause)
    - DENSE_RANK ( ) OVER ([query_partition_clause] order_by_clause)

6.2 Ranking in SQL

Detour

- SQL99 Ranking e.g.

SELECT channel, calendar, TO_CHAR(TRUNC(SUM(amount_sold),-6), '9,999,999') SALES, RANK() OVER (ORDER BY Trunc(SUM(amount_sold),-6) DESC) AS RANK, DENSE_RANK() OVER (ORDER BY TRUNC(SUM(amount_sold),-6) DESC) AS DENSE_RANK FROM sales, products …

CHANNEL	CALENDAR	SALES	RANK	DENSE_RANK
Direct sales	02.2015	10,000	1	1
Direct sales	03.2015	9,000	2	2
Internet	02.2015	6,000	3	3
Internet	03.2015	6,000	3	3
Partners	03.2015	4,000	5	4

  - DENSE_RANK **leaves no gaps in ranking sequence** when there are ties

6.2 Ranking in SQL

Detour

- Other flavours of ranking
  - **Group ranking**
    - RANK function can operate within groups: the rank gets **reset** whenever the group changes
    - A single query can contain more than one ranking function, each partitioning the data into different groups

6.2 Group Ranking

Detour

- This is accomplished with the **PARTITION BY** clause
  - E.g. SELECT … RANK() OVER (PARTITION BY channel ORDER BY SUM(amount_sold) DESC) AS RANK_BY_CHANNEL

CHANNEL	CALENDAR	SALES	RANK _BY_CHANNEL
Direct sales	02.2016	10,000	1
Direct sales	03.2016	9,000	2
Internet	02.2016	6,000	1
Internet	03.2016	6,000	1
Partners	03.2016	4,000	1

6.2 NTILE

Detour

- Not a part of the SQL99 standard, but adopted by major vendors
- NTILE splits a set into equal groups
  - It **divides** an ordered partition into **buckets** and assigns a bucket number to each row in the partition
  - Buckets are calculated so that each bucket has **exactly the same number of rows** assigned to it or at most 1 row more than the others

6.2 NTILE

Detour

- SELECT … NTILE(3) OVER (ORDER BY sales) NT_3 FROM …

CHANNEL	CALENDAR	SALES	NT_3
Direct sales	02.2016	10,000	1
Direct sales	03.2016	9,000	1
Internet	02.2016	6,000	2
Internet	03.2016	6,000	2
Partners	03.2016	4,000	3

  - NTILE(4) – quartile
  - NTILE(100) – percentage

6.2 SQL-99

- Grouping operators
  - Extensions to the GROUP BY operator
    - GROUPING SET
    - ROLLUP
    - CUBE

6.2 Grouping Operators

- GROUPING SET
  - **Efficiently** replaces the series of UNIONed queries
    - SELECT dept_name, CAST(NULL AS CHAR(10)) AS job_title, COUNT(*) FROM personnel
      GROUP BY dept_name
      UNION ALL
      SELECT CAST(NULL AS CHAR(8)) AS dept_name, job_title, COUNT(*) FROM personnel
      GROUP BY job_title;
    - **Can be re-written as:**
      SELECT dept_name, job_title, COUNT(*) FROM Personnel
      GROUP BY GROUPING SET (dept_name, job_title);

6.2 Grouping Sets

- The issue of **NULL values**
  - The new grouping functions generate NULL values at the subtotal levels
    - How do we tell the difference between "generated NULLs" and "real NULLs" from the data itself?
    - The GROUPING function call returns 0 for NULL in the data and 1 for generated NULL

Year	Brand	SUM(qty)	
2016	Real NULL	250	(year, brand)
2016	BMW	300	
2016	VW	450	
2016	Gen. Null	1000	(year)

6.2 Roll-up

- **Roll-up:** produces a result set that contains subtotal rows in addition to regular grouped rows
  - GROUP BY ROLLUP (a, b, c) is equivalent to GROUP BY GROUPING SETS (a, b, c), (a, b), (a), ()
  - N elements of the Roll-up operation translate to (N+1) grouping sets
  - **Order** is significant for Roll-up!
    - GROUP BY ROLLUP (c, b, a) is equivalent with grouping sets of (c, b, a), (c, b), (c), ()

6.2 Roll-up

- Roll-up operation, e.g.:
  - SELECT year, brand, SUM(qty) FROM sales GROUP BY ROLLUP(year, brand);

Year	Brand	SUM(qty)	
2015	Mercedes	250	(year, brand)
2015	BMW	300	
2015	VW	450	
2015	NULL	1000	(year)
2016	Mercedes	50	(year, brand)
…	…	…	
2016	NULL	400	(year)
NULL	NULL	1400	(ALL)

6.2 Grouping Operators

- Cube operator: contains all the subtotal rows of a Roll-up and in addition cross-tabulation rows
  - Can also be thought as a series of GROUPING SETs
  - All permutations of the cubed grouping expressions are computed along with the grand total
    - N elements of a CUBE translate to 2^n grouping sets:
      - GROUP BY CUBE (a, b, c) is equivalent to
        GROUP BY GROUPING SETS(a, b, c) (a, b) (a, c) (b, c) (a) (b) (c) ()

6.2 CUBE Operator

Aggregate

Sum

Group By
(with total)

By model

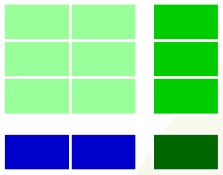
SUV
SEDAN
SPORT

Sum

Cross Tab

BMW MERC By model

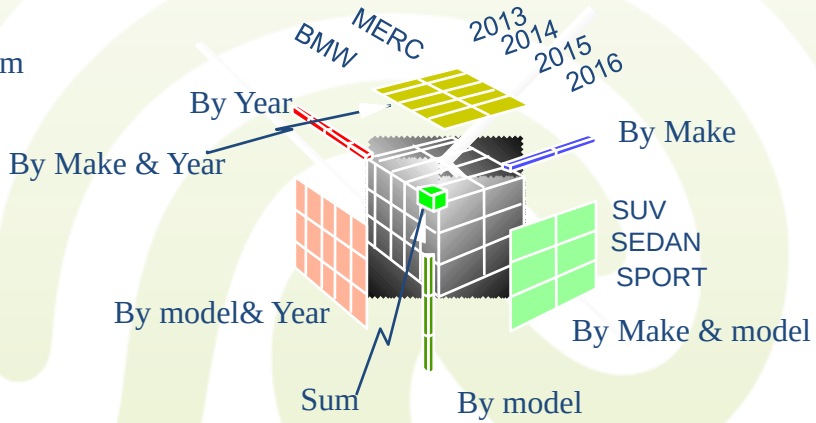
SUV
SEDAN
SPORT



By Make

Sum

The Data Cube and
The Sub-Space Aggregates

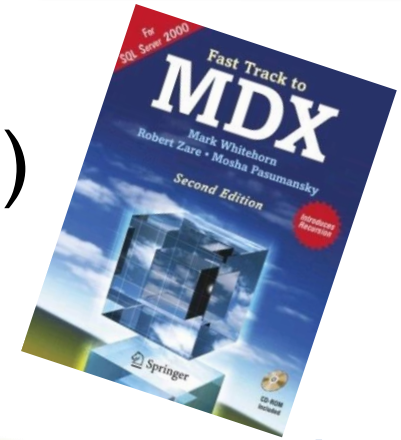


6.2 CUBE Operator

- Example
  - SELECT year, brand, SUM(qty) FROM sales GROUP BY CUBE (year, brand);

Year	Brand	SUM(qty)	
2015	Mercedes	250	(year, brand)
2015	BMW	300	
2015	VW	450	
2015	NULL	1000	(year)
2016	Mercedes	50	(year, brand)
…	…	…	
2016	NULL	400	(year)
NULL	Mercedes	300	(brand)
NULL	BMW	350	
NULL	VW	650	
NULL	NULL	1400	(ALL)

6.2 MDX



- **MDX** (MultiDimensional eXpressions)
  - Developed by Microsoft
    - Not really brilliant
    - But adopted by major OLAP providers due to Microsoft's market leader position
  - Used in
    - OLE DB for OLAP (ODBO) with API support
    - XML for Analysis (XMLA): specification of web services for OLAP
  - For ROLAP to support MDX, it is usually translated into SQL

6.2 MDX Basics

- Similar to **SQL syntax**

```
SELECT {Germany, Niedersachsen, Bayern, Frankfurt} ON COLUMNS,
       {Qtr1.CHILDREN, Qtr2, Qtr3} ON ROWS
       FROM SalesCube
       WHERE (Measures.Sales, Time.[2015], Products.[All Products]);
```

  - SELECT
    - Dimensions, on columns and rows
  - FROM
    - Data source cube specification
    - If joined, data cubes must share dimensions
  - WHERE
    - **Slicer** - restricts the data area
    - Specifies the measures to return

6.2 MDX Basic Elements

- Lists: Enumeration of elementary nodes from different classification levels
  - E.g. {Germany, Niedersachsen, [Frankfurt am Main], USA}
- Generated elements: Methods which lead to new sets of the classification levels
  - Germany.CHILDREN generates: {Niedersachsen, Bayern,…}
  - Niedersachsen.PARENT generates Germany
  - Time.Quarter.MEMBERS generates all the elements of the classification level
- Functional generation of sets
  - DESCENDENT(USA, Cities): children of the provided classification level *Cities*
  - GENERATE ({USA, France}, DESCENDANTS(Geography.CURRENT, Cities)): enumerates all the cities in USA and France

6.2 MDX Basics

- **Sets nesting** combines individual coordinates to reduce dimensionality
  - SELECT CROSSJOIN({Germany, Sachsen, Hannover, BS}{Ikea, [H&M-Möbel]})
    ON COLUMNS,
    {Qtr1.CHILDREN, Qtr2} ON ROWS
    FROM salesCube
    WHERE (Measure.Sales, Time.[2015], Products.[All Products]);

	Germany		Sachsen		Hannover		BS	
	Ikea	H&M-Möbel	Ikea	H&M-Möbel	Ikea	H&M-Möbel	Ikea	H&M-Möbel
Jan 14								
Feb 14								
Mar 14								
Qtr2								

sales numbers

6.2 MDX Basics

- Relative selection
  - Uses the **order** in the **dimensional structures**
    - Time.[2015].LastChild : last quarter of 2015
    - [2015].NextMember : {[2016]}
    - [2015].[Qtr4].Nov.Lead(2) : Jan 2016
    - [2010]:[2015] represents [2010], .., [2015]
- Methods for hierarchy information extraction
    - Germany.LEVEL : country
    - Time.LEVELS(1) : Year
- Brackets
    - {}: Sets, e.g. {Hannover, BS, John}
    - []: text interpretation of numbers, empty spaces between words or other symbols
      - E.g. [2015], [Frankfurt am Main], [H&M]
    - (): tuple e.g. WHERE (Measure.Sales, Time.[2015], Products.[All Products])

6.2 MDX Basics

- **Special functions and filters**
  - Special functions TOPCOUNT(), TOPPERCENT(), TOPSUM()
    - e.g. top 5 areas of Germany by turnover on rows

```
SELECT {Time.CHILDREN} ON COLUMNS,
       {TOPCOUNT(Germany.CHILDREN, 5, Sales.turnover)} ON ROWS
  FROM salesCube
  WHERE (Measure.Sales,  Time.[2018]);
```

  - FILTER function, e.g. areas of Germany with increased turnover for 2018 compared to 2017

```
SELECT FILTER(Germany.CHILDREN, ([2018], Turnover) > ([2017], Turnover))
       ON COLUMNS, Quarters.MEMBERS ON ROWS
  FROM salesCube
  WHERE (Measure.Sales,  Time.[2018], Products.Electronics);
```

6.2 MDX Basics

- **Time series**
  - Set Value Expressions e.g., choosing time intervals
    - PERIODSTODATE(Quarter, [15-Nov-2018]): returns 1.10.-15.11.2018
  - Member Value Expressions e.g. pre-periods
    - PARALLELPERIOD(Year, 3, [Sep-2018]): returns [Sep-2015]
  - Numerical functions: covariance, correlation, linear regression

6.2 mdXML

- **XMLA** (XML for Analysis)
  - Most recent attempt at a standardized API for OLAP
  - Allows client applications to talk to multi-dimensional data sources
    - In XMLA, mdXML is a MDX wrapper for XML
  - Underlying technologies
    - XML, SOAP, HTTP
  - Service primitives
    - DISCOVER
      - Retrieve information about available data sources, data schemas, server infos…
    - EXECUTE
      - Transmission of a query and the corresponding conclusion

Summary

- OLAP Operations:
  - Roll-up: hierarchical, dimensional
  - Drill-down: You can't drill if you don't have the data
  - Slice, dice, Pivot
- Operations affect data through query languages OLAP Query languages: SQL 99, MDX
  - SQL99: Grouping Set, Roll-up, Cube operators
  - MDX: Similar to SQL, used especially MOLAP solutions, in ROLAP it is mapped to SQL

Next lecture

- Building the DW
  - The DW Project
  - Data Extract/Transform/Load (ETL)
  - Metadata