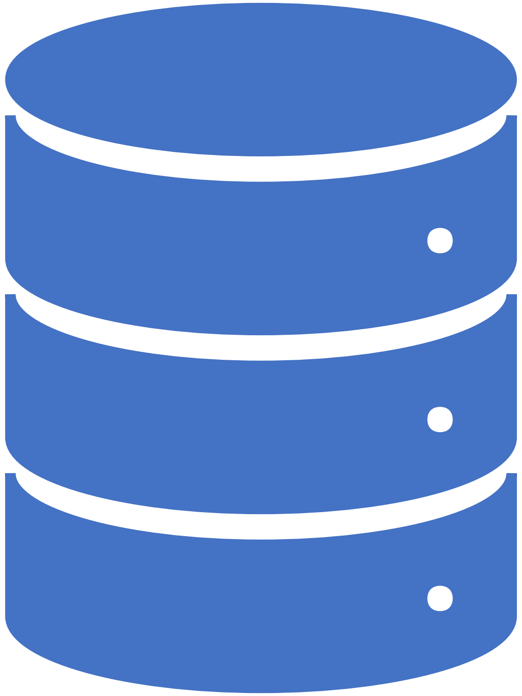
MUNI
FACULTY
OF INFORMATICS



PA220: Database systems for data analytics

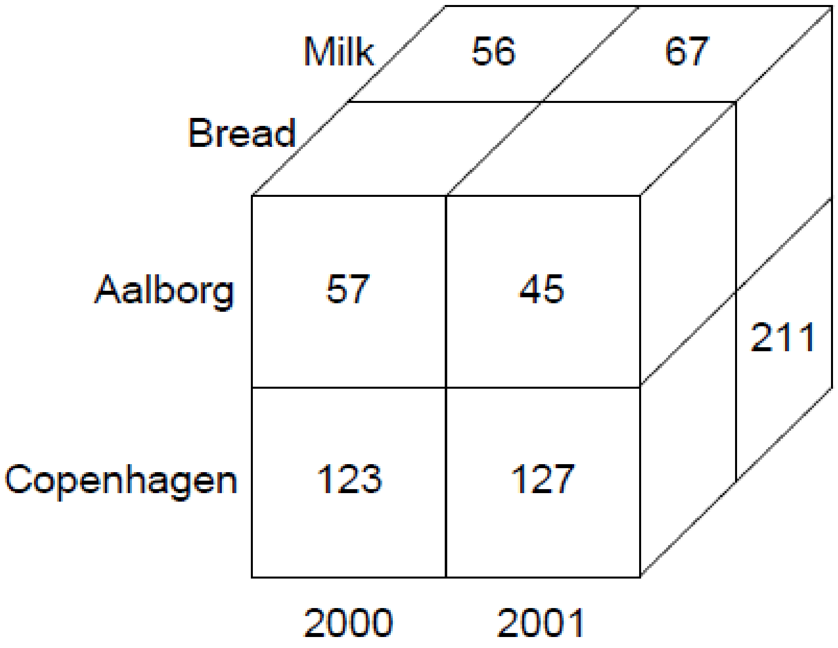
Data Warehouse Implementation & Querying

Contents

- Implementation of dimensional model
- Querying by dimensions
  - Grouping possibilities
  - Aggregate functions
  - Window functions
- Case Study: Grocery Store

Data Cube

- Implication of using dimensional model (star schema)
- A symbolic representation of aggregated data



(Data) Cubes

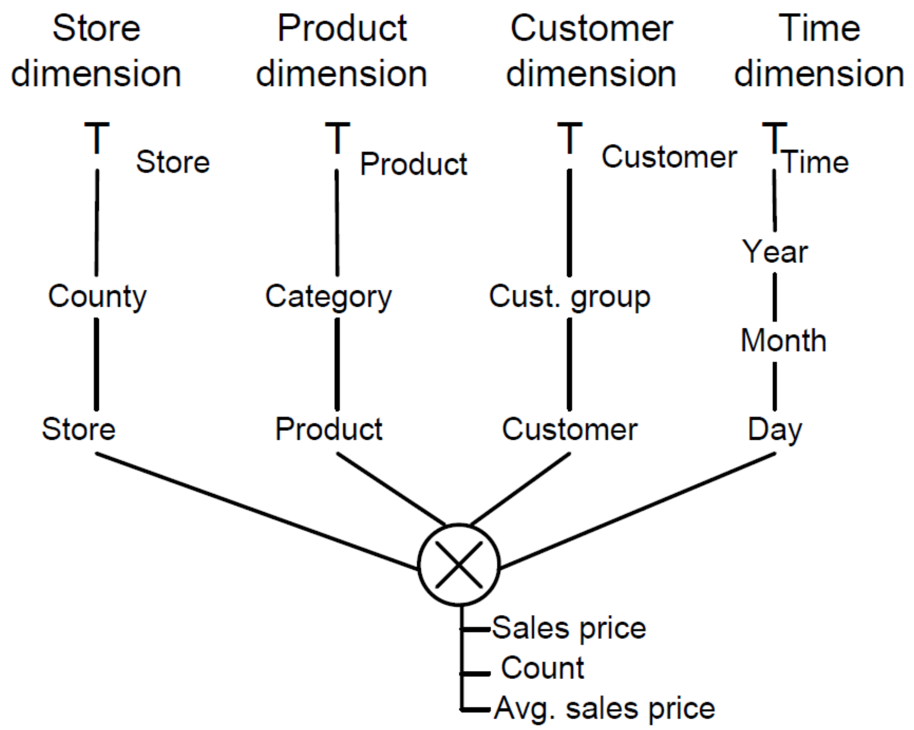
- A “cube” may have **many** dimensions!
    - More than 3 - the term ”**hyper-cube**” can sometimes be used
    - Theoretically no limit for the number of dimensions
    - Typical cubes have 4-12 dimensions
- But only 2-4 dimensions can be viewed at a time
    - Dimensionality reduced by queries via projection/aggregation
- A cube consists of **cells**
    - A given combination of dimension values
    - A cell can be empty (no data for this combination)
    - A **sparse** cube has few non-empty cells
    - A **dense** cube has many non-empty cells
    - Cubes become sparser for many/large dimensions

Dimensions

- Dimensions are used for
  - Selection of data
  - Grouping of data at the right level of detail
- Dimensions consist of dimension values
  - Product dimension have values ”milk”, ”cream”, …
  - Time dimension have values ”1/1/2001”, ”2/1/2001”,…
- Dimension values may have an ordering
  - Used for comparing cube data across values
  - Example: ”percent sales increase compared with last month”
  - Especially used for Time dimension

- Dimensions are the core of multidimensional databases
  - Other types of databases do not support dimensions

Schema Documentation



- No well-defined standard
  - T level corresponds to ALL
  - Record the measures
- You could also use a UML like notation
- Modeling and OLAP tools may have their own notation

OLAP Systems

- A key concept of OLAP systems is multidimensional analysis:
  - Examining data from many dimensions.
    - Show total sales across all products at increasing aggregation levels for a geography dimension, from state to country to geographical region, for 1999 and 2000.
    - Create a cross-tabular analysis of our operations showing expenses by territory in South America for 1999 and 2000. Include all possible subtotals.
    - List the top 10 sales representatives in Asia according to 2000 sales revenue for food products and rank their commissions.
- Organization of cubes to efficiently answer the requests
  - Response time of seconds / few minutes

Relational OLAP (ROLAP)

- Store data in relational databases and simulate multidimensionality with special schemas
  - Data stored in relational tables
    - Star (or snowflake) schemas used for modeling
    - SQL used for querying
- Pros
  - Leverages investments in relational technology
  - Scalable (billions of facts)
  - Flexible, designs easier to change
  - New, performance enhancing, techniques adapted from MOLAP
  - Indices, materialized views
- Cons
  - Storage use (often 3-4 times MOLAP)
  - Response times

Product ID	Store ID	Sales
1	3	2
2	1	7
3	2	3
…	…	…

Multidimensional OLAP (MOLAP)

- Physically stages the processed multidimensional information to deliver consistent and rapid response times to end users
  - Data stored in special multidimensional data structures
    - E.g., multidimensional array A on hard disk
- Pros
  - Less storage use (“foreign keys” not stored)
  - Faster query response times
    - Direct access to a cell by giving position, e.g. A[1][2] → 2
- Cons
  - Up till now not so good scalability
  - Less flexible, e.g., cube must be re-computed when design changes
  - Does not reuse an existing investment (but often bundled with RDBMS)
  - Not as open technology

MOLAP data cube

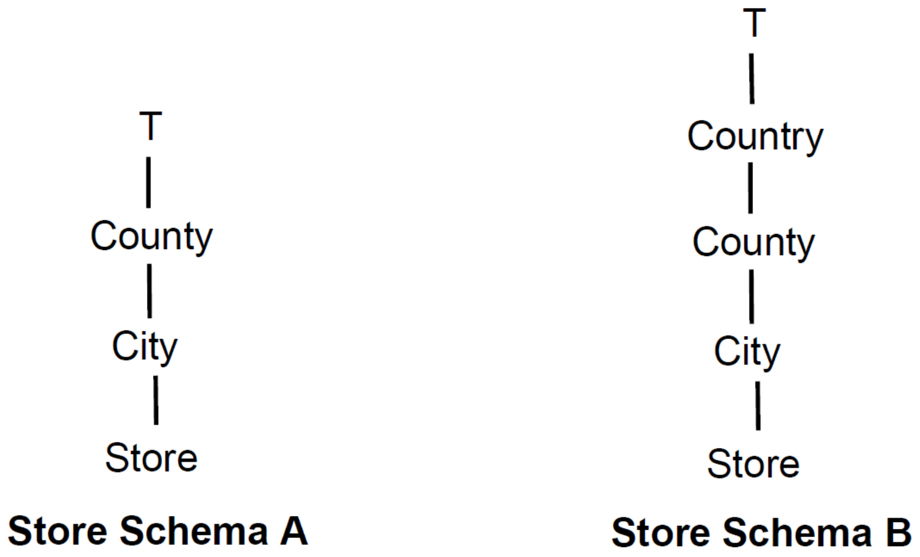
$d_2 \setminus d_1$	**1**	**2**	**3**
1	0	7	0
2	2	0	0
3	0	0	3

Hybrid OLAP (HOLAP)

- Detail data stored in relational tables (ROLAP)
- Aggregates stored in multidimensional structures (MOLAP)
- Pros
  - Scalable (as ROLAP)
  - Fast (as MOLAP)
- Cons
  - High complexity

Question time

- Suppose that we want to replace the original Store hierarchy A by a new hierarchy B
- How do we modify the schema to reflect it in ROLAP / MOLAP?

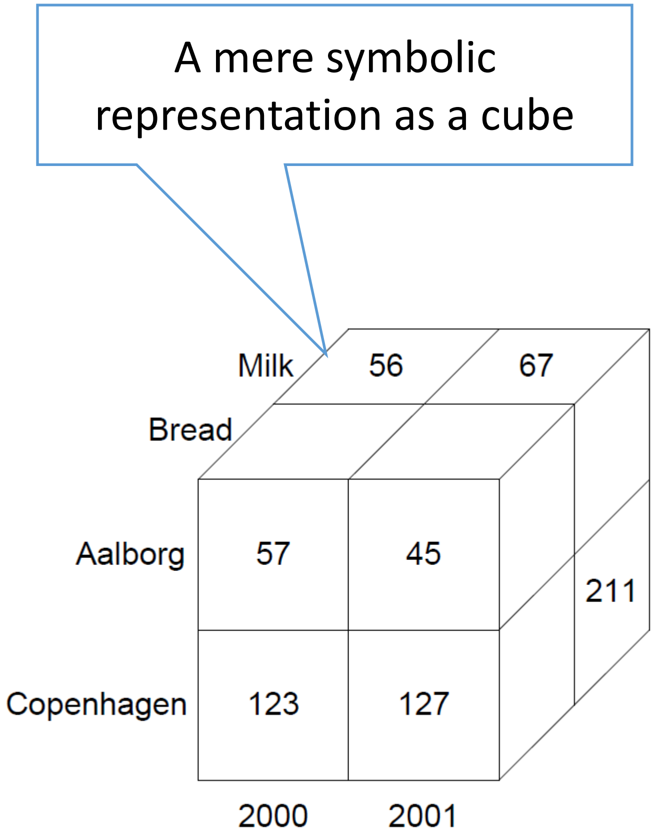


Question time (2)

- New store is being open and fact table needs to be populated
- How do we modify the fact table in ROLAP / MOLAP ?

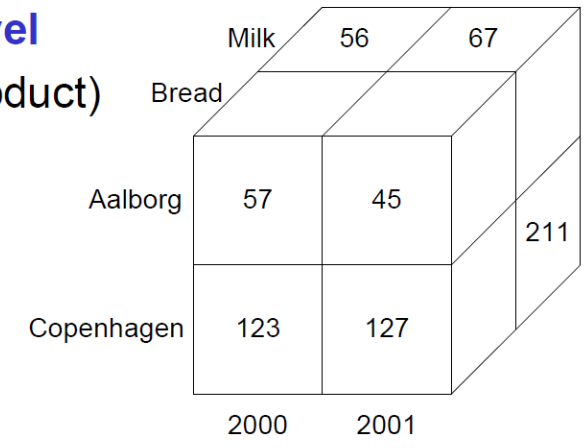
Relational OLAP Cubes

- Two kinds of queries
  - **Navigation** queries examine one dimension
    - SELECT DISTINCT l FROM d [WHERE p]
  - **Aggregation** queries summarize fact data
    - SELECT d1.l1, d2.l2, SUM(f.m)
      FROM d1, d2, f
      WHERE f.dk1 = d1.dk1 AND f.dk2 = d2.dk2 [AND p]
      GROUP BY d1.l1,d2.l2
- Fast, interactive analysis of large amounts of data

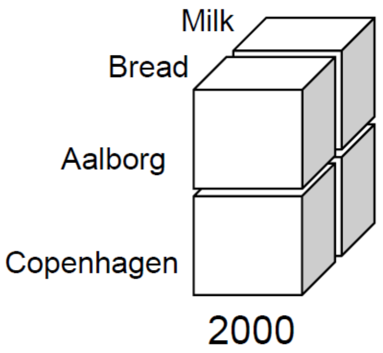


OLAP Queries

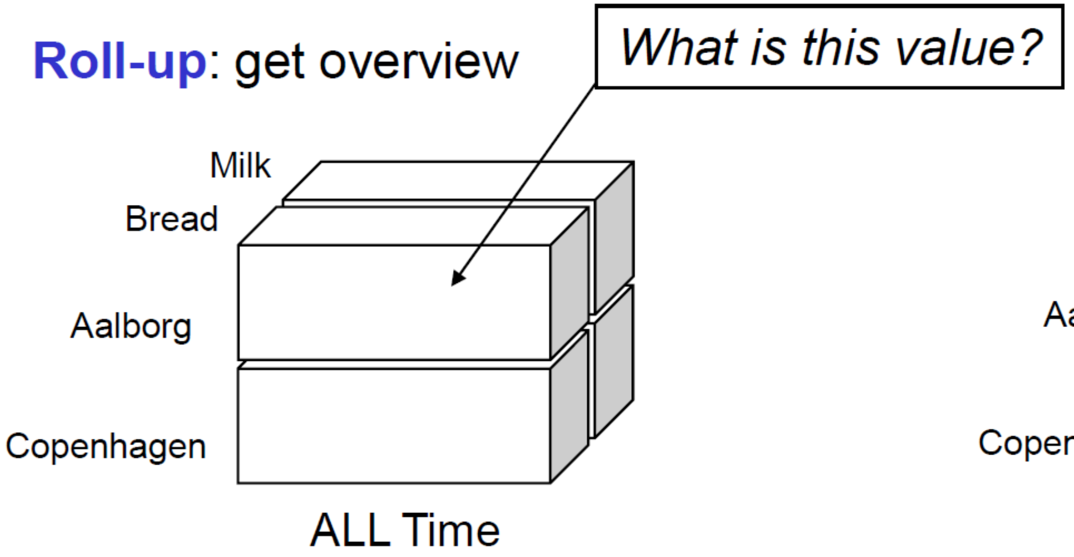
Starting level
(City, Year, Product)



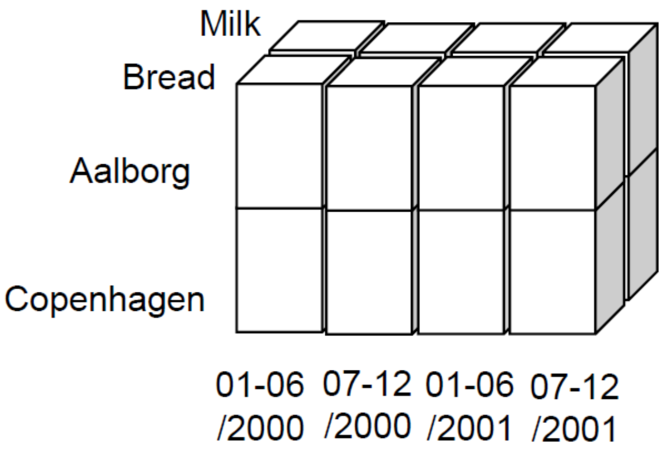
Slice/Dice:



Roll-up: get overview



Drilll-down: more detail



OLAP Cube in MS Analysis Services Project

	Prod Group / Name						
	cacao	flask	kaffe	milk	others	vand	Grand Total
Year / Month / Day	Sales	Sales	Sales	Sales	Sales	Sales	Sales
1996	369	471		229		813	1882
1997	2161.75	3985		1727	144	15576	23593.75
1998	16082	20591		12887.25	6908	80492	136960.25
1999	17325	20626	2535	13063.25	7609.5	90644	151802.75
2000	21095	17395	5940	10631.5	21132.5	81444	157638
2001	16900.75	29712.5	0	[illegible]	[illegible]	[illegible]	[illegible]
2002	30086.5	34731	0				
2003	28740	28596	0				
2004	24126.75	28292	0				
2005	22695.5	20449	0				
2006	25196	19958	0				
2007	876	641	0				
Grand Total	205654.25	225447.5	8[illegible]				

drill down

	Prod Group / Name								
	cacao			flask	kaffe	milk	others	vand	Grand Total
	½L Matilde cacao	Cocio	Total						
Year / Month / Day	Sales	Sales	Sales	Sales	Sales	Sales	Sales	Sales	Sales
1996	174	195	369	471		229		813	1882
1997	1501.75	660	2161.75	3985		1727	144	15576	23593.75
1998	13767	2315	16082	20591		12887.25	6908	80492	136960.25
1999	13050	4275	17325	20626	2535	13063.25	7609.5	90644	151802.75
2000	17430	3665	21095	17395	5940	10631.5	21132.5	81444	157638
2001	12403.5	4497.25	16900.75	29712.5	0	9861.25	23260.25	84286	164020.75
2002	25425.75	4660.75	30086.5	34731	0	15506.5	41619.5	74847	196790.5
2003	25524.25	3215.75	28740	28596	0	14213.5	45046	63580	180175.5
2004	20286	3840.75	24126.75	28292	0	9592	82226	54526.5	198763.25
2005	18152.75	4542.75	22695.5	20449	0	7803.25	75835	52044	178826.75
2006	22968.5	2227.5	25196	19958	0	6910.5	102746	47456	202266.5
2007	876		876	641	0	155.75	2094.5	1387.5	5154.75
Grand Total	171559.5	34094.75	205654.25	225447.5	8475	102580.75	408621.25	647096	1597874.75

Multidimensional database implementation

- Microsoft SQL Server Analysis Services (SSAS) in MS SQL Server
- Oracle Database OLAP Option within Oracle database

Microsoft SQL Server Analysis Services (SSAS)

- Introduced in SQL Server 2008
- Nice built-in features
  - Analysis Services
  - Integration Services
  - Reporting Services
- Easy to use
  - Graphical “Management Studio” and “BI Developer Studio”
- Allows all flavors of MOLAP, ROLAP and HOLAP to be used within the same model
  - Intelligent pre-aggregation (for improving query performance)
  - Uses the query language MDX (MultiDimensional eXpressions)

OLAP Operations & Queries

- Redundancy is necessity
  - Materialized views, special purpose indexes, denormalized schemas
- Data is refreshed periodically
  - Daily or weekly
- DW queries are big queries
  - Imply a large portion of the data
  - Mostly read queries

OLAP Queries

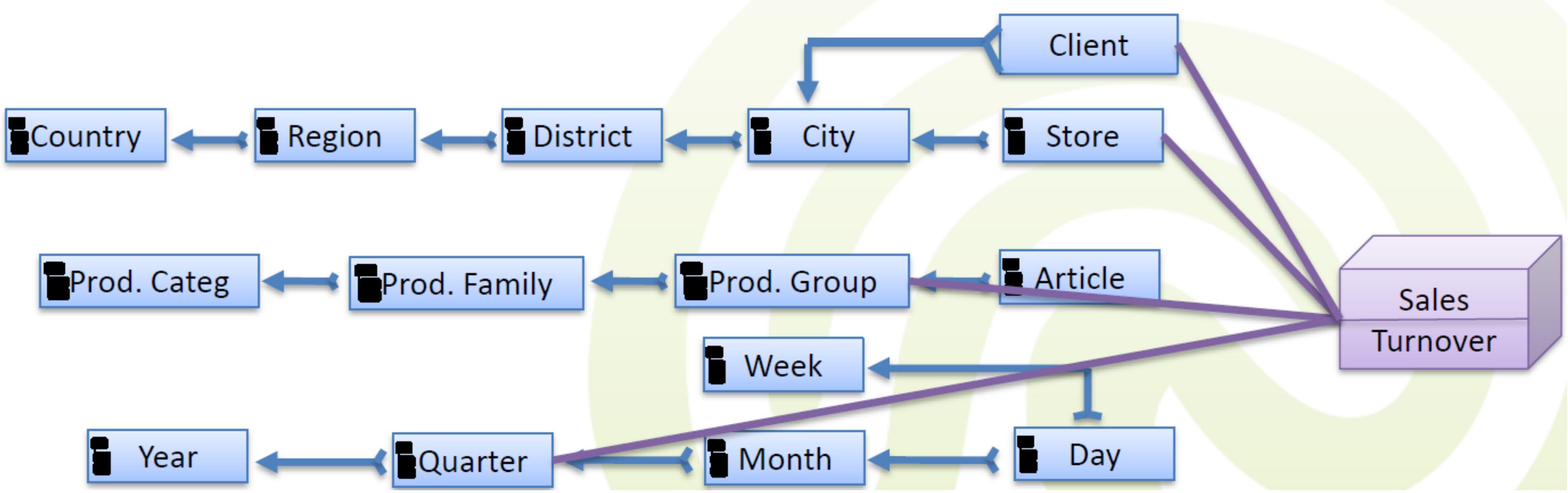
- Typical OLAP query operations
    - Roll up
    - Drill down
    - Slice and dice
    - Pivot (rotate) (aka crosstab)
- Other operations
    - Drill across
        - navigate from a fact table to another one by using a shared dimension
    - Drill through
        - allows users to view relational transactions that make up a multidimensional point in cube
    - Data densification (partitioned outer join)
- Based on
    - Aggregate functions
    - Ranking and comparing operators

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 (dimensional roll up)
    - Or by a mix of these 2 techniques
  - Used for obtaining an increased generalization
    - E.g., from Time.Week to Time.Year

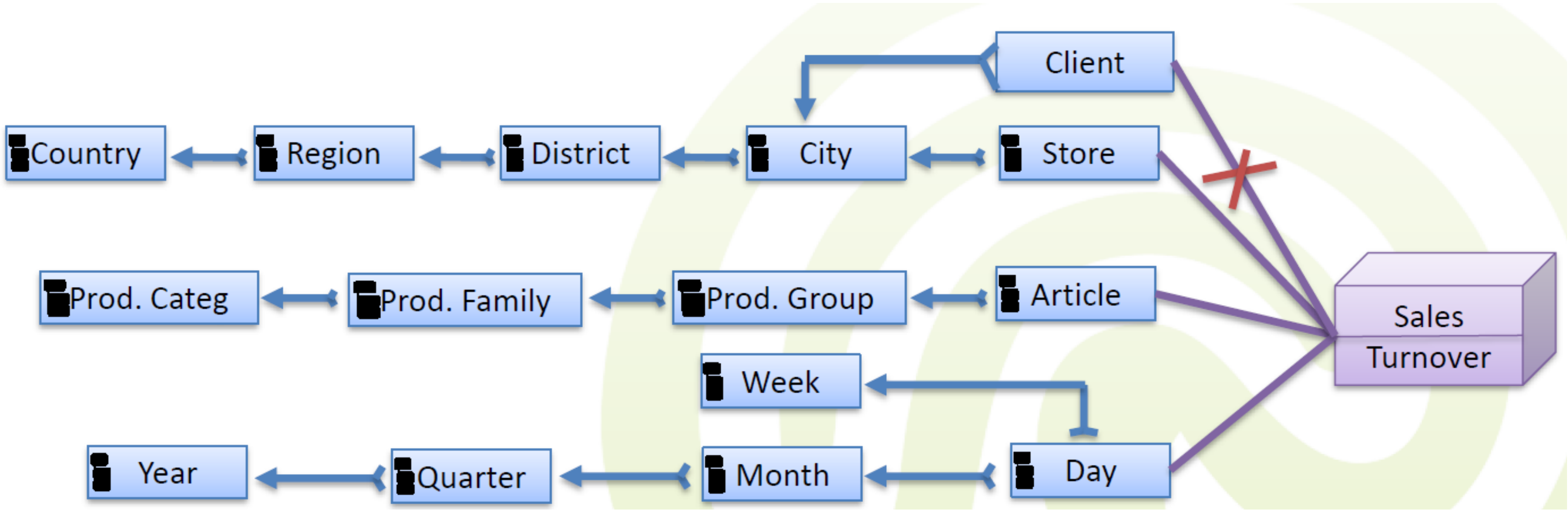
Roll Up

- 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



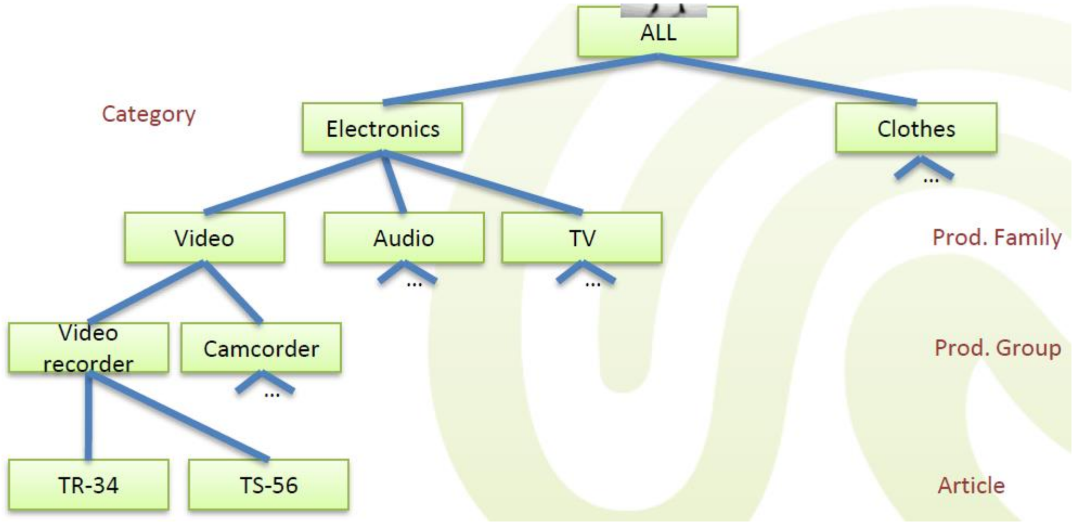
Roll Up

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



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



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 not drill if you do not have the data

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

Slice

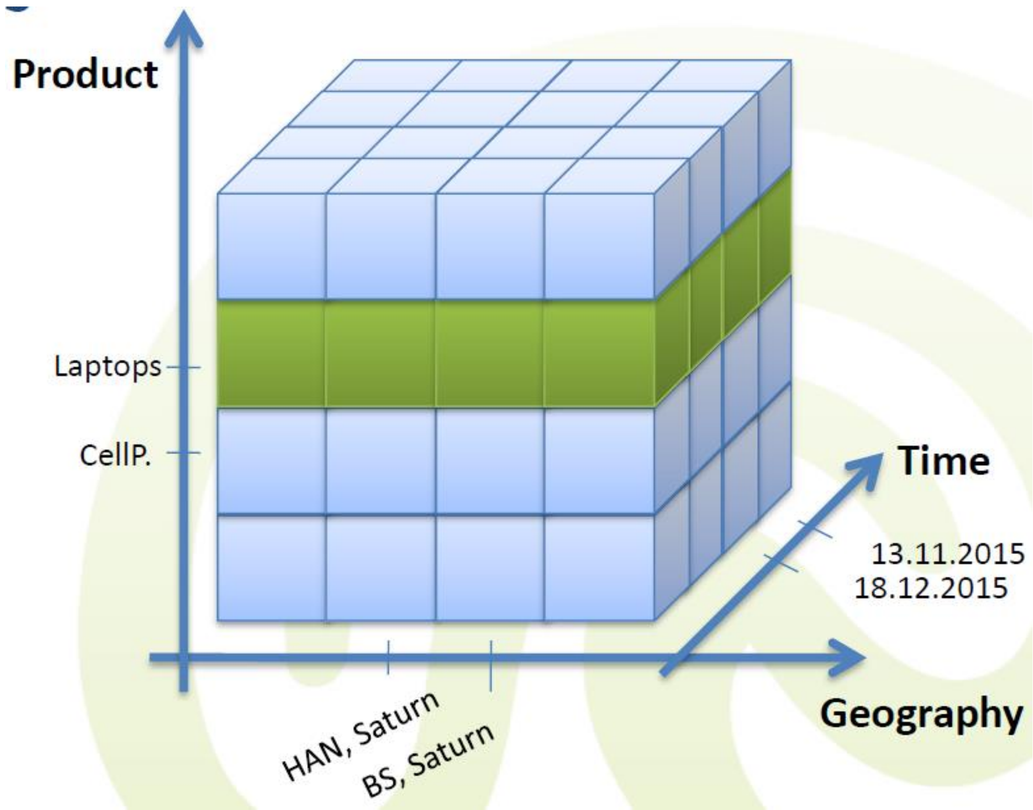
- Slice: a subset of the multi-dimensional array corresponding to a single value of one or more dimensions and projection on the rest of dimensions
  - E.g., project on Geo (store) and Time from values corresponding to Laptops (cat. of products) in the product dimension

```
SELECT store_id, time_id, amount
FROM Sales
WHERE article_id = <laptop_id>
```

$$\pi_{StoreId,TimeId,Amount}\left(\sigma_{ArticleId=<LaptopId>}(Sales)\right)$$

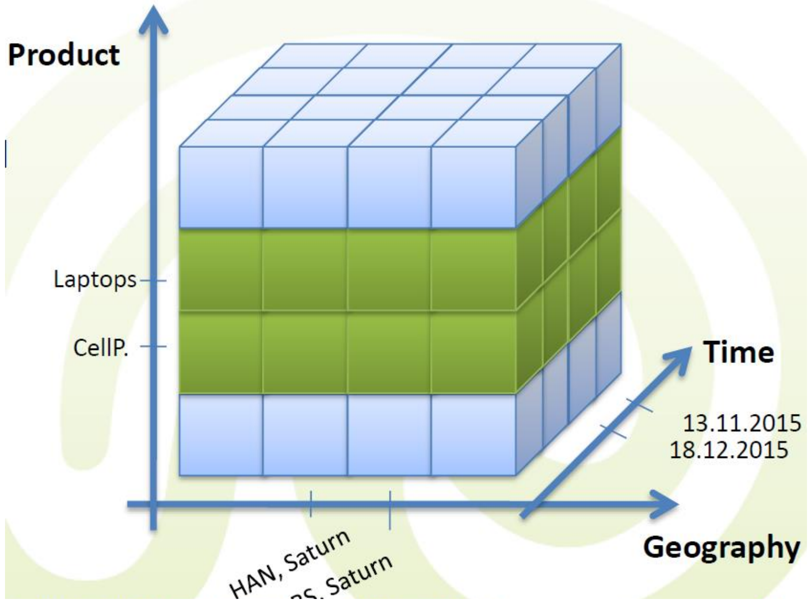
Slice

- Amounts to <u>equality</u> select condition on one dimension
- WHERE clause in SQL
  - E.g., slice Laptops



Dice

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



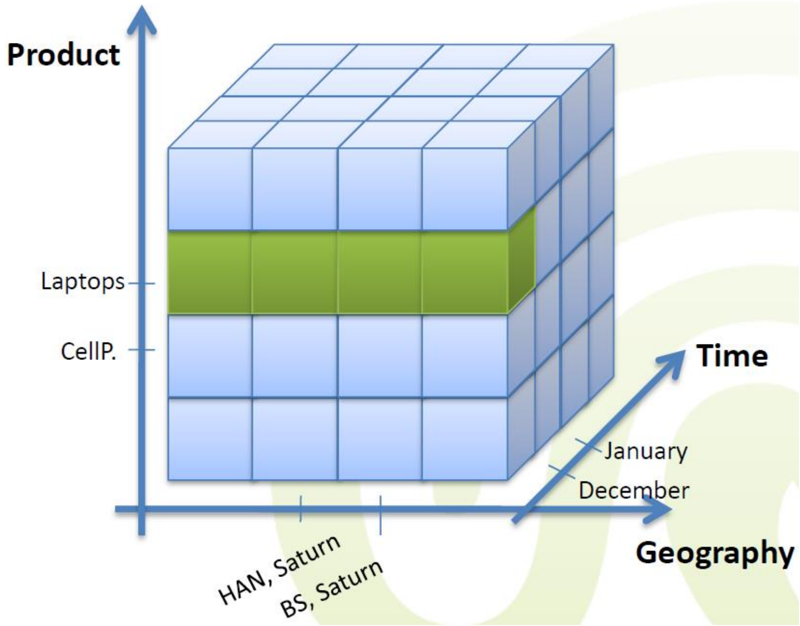

```
SELECT store_id, time_id, amount
FROM Sales
WHERE article_id = <laptop_id>
   OR article_id = <cellphone_id>
```

$$\pi_{StoreId,TimeId,Amount}\left(\sigma_{ArticleId=<LaptopId>\vee ArticleId=<CellPhoneId>}(Sales)\right)$$

Dice

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

```
SELECT store_id, time_id, amount
FROM Sales
WHERE article_id = <laptop_id>
  AND month_id = <December>
```



$$\pi_{StoreId,TimeId,Amount}\left(\sigma_{ArticleId=<LaptopId>\wedge MonthId=<December>}(Sales)\right)$$

Pivoting

- Pivot (rotate): re-arrange 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
    - Shows data at different "granularities"
  - The result of pivoting is also called **cross tabulation**
  - This is space efficient for dense data only (thus, few dimensions)

Media	**Country**		
	France	**USA**	**Total**
Internet	9,597	124,224	133,821
Direct Sales	61,202	638,201	699,403
Total	70,799	762,425	833,224

Pivoting in ROLAP

- Tabular representation for the cross-tabular report with totals.
  - ALL is a dummy value and stands for all or multiple values. (≈ NULL in SQL)
  - Probably not as nice to read as the crosstab.
- Information content is the same as in the crosstab.
- Is more space efficient than crosstab if the data is sparse.

Media	Country	Total
Internet	France	9,597
Internet	USA	133,821
Direct Sales	France	61,202
Direct Sales	USA	638,201
Internet	ALL	133,821
Direct Sales	ALL	699,403
ALL	France	70,799
ALL	USA	762,425
ALL	ALL	833,224

SQL & OLAP

- 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  d1.x, d2.y, d3.z, sum(f.t1), avg(f.t2)
  FROM Fact f, Dim1 d1, Dim2 d2, Dim3 d3
  WHERE a < d1.field < b AND d2.field = c
  GROUP BY d1.x, d2.y, d3.z;
```

SQL & OLAP

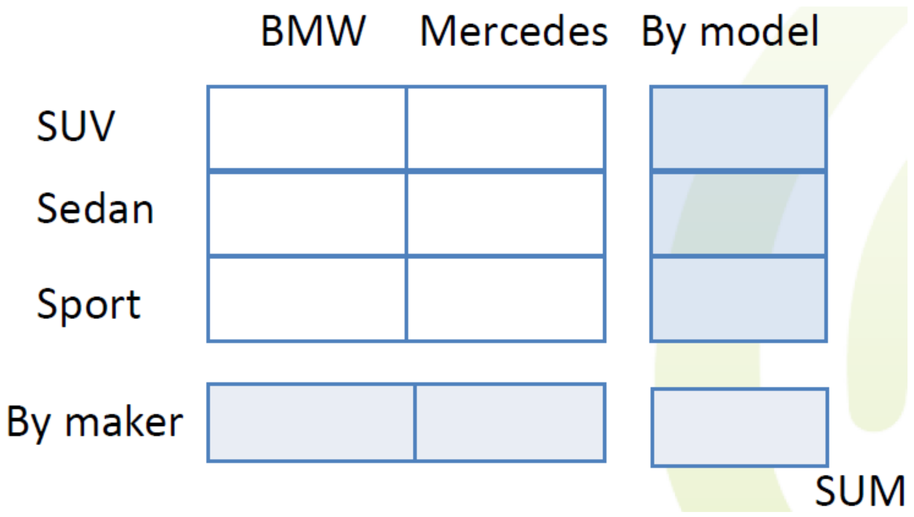
- No standard query language for OLAP
  - SQL99 for ROLAP
  - SQL:2003 OLAP extensions
- New SQL SELECT clauses
  - GROUPING SETS
  - ROLLUP
  - CUBE
- Queries of type “top k”
- New aggregate functions

SQL & OLAP

- 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

SQL92

- 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

SQL92

- 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

SQL92

- Reporting features in SQL/92
  - Too complex to express
    - RANK (top k) and NTILE (“top X%” of all products)
    - Median
    - Running/rolling/moving 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 NATURAL JOIN d_time start, v_sales NATURAL JOIN d_time end
      WHERE end.time >= start.time AND end.time <=start.time + 2
      GROUP BY end.time
      ```

SQL99: Grouping Operators

- GROUP BY **ROLLUP**(gcols)
  - Roll-up hierarchically
- GROUP BY **CUBE** (gcols)
  - Roll-up to all possible combinations
- GROUP BY gcols1, CUBE(gcols2)
  - Partial roll-up
- GROUP BY **GROUPING SETS** (gcols1, …, gcolsN)
  - Explicit specification of roll-ups
- GROUP BY groupings1, groupings2, …
  - Cross-product of groupings

- SELECT … GROUPING_ID(gcols)…
  - Identification of roll-up level

Roll Up

- ROLLUP creates subtotals at n+1 levels, where n is the number of grouping columns
  - Rows that would be produced by GROUP BY without ROLLUP
  - First-level subtotals
  - Second-level subtotals
  - …
  - A grand total row
- It is very helpful for subtotaling along a hierarchical dimensions such as time or geography
  - ROLLUP(y, m, day) or ROLLUP(country, state, city)
- Order of attributes is significant!

Roll Up

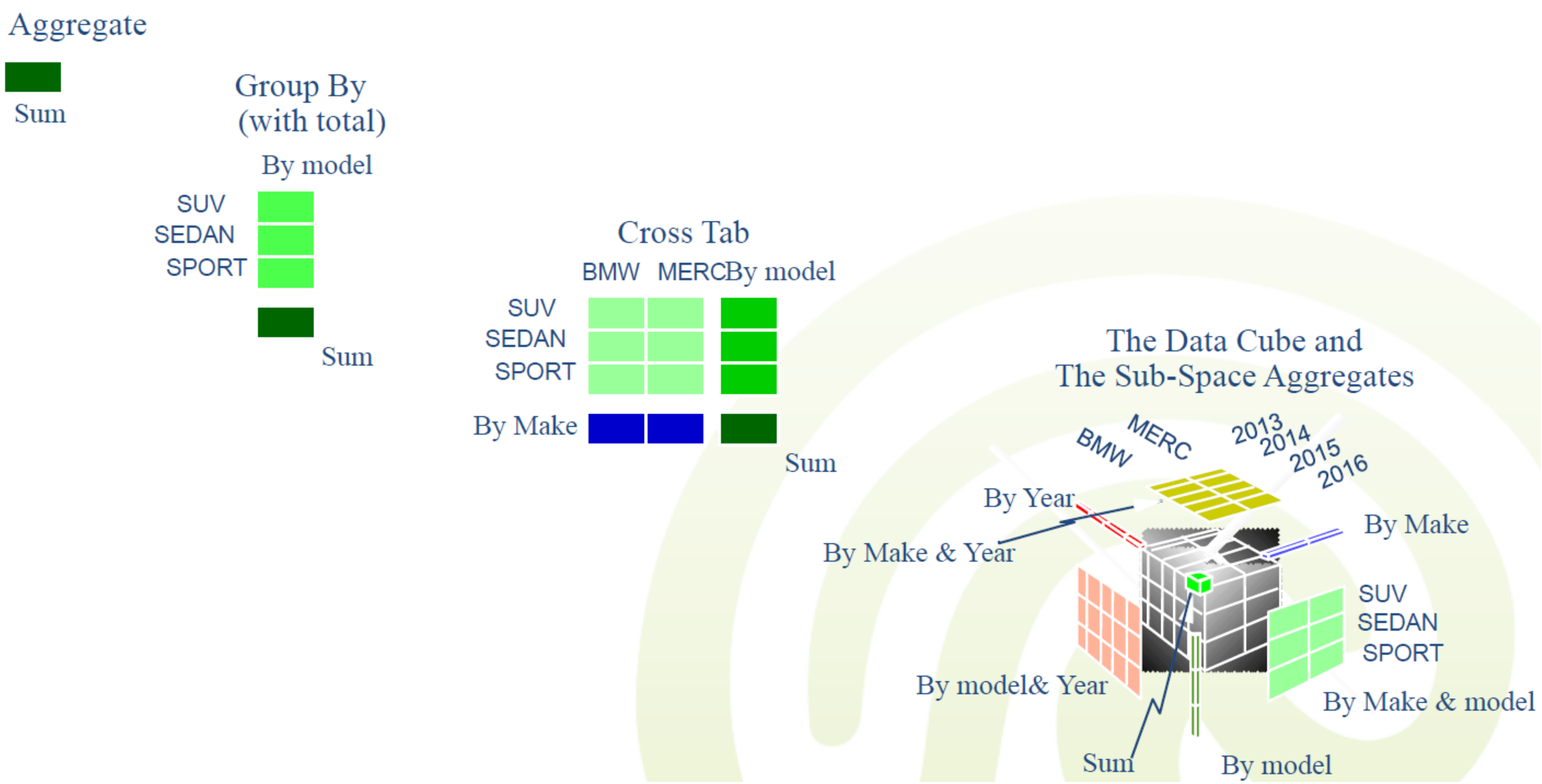
- Roll up operation, e.g.:
  - SELECT year, brand, SUM(qty) FROM sales GROUP BY ROLLUP(year, brand);

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)

Cube

- CUBE creates 2^n combinations of subtotals, where n is the number of grouping columns
  - Includes all the rows produced by ROLLUP
- CUBE is typically most suitable in queries that
  - use columns from multiple dimensions
  - rather than columns representing different levels of a single dimension
    - e.g., subtotals for all combinations of month, state, and product
- Partial CUBE similar to partial ROLLUP

Cube



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
BMW
MERC
2013
2014
2015
2016
By Year
By Make
By Make & Year
SUV
SEDAN
SPORT
By model& Year
By Make & model
Sum
By model

Cube

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

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

Grouping Sets

- Grouping sets produce just the specified groupings.
  - No (automatic) rollup is performed.
    - E.g., GROUPING SETS ((A,B), (D), (C, E, F))
      - Collection of columns in paratheses → **composite column**
- 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 rewritten as:
  - SELECT dept_name , job_title , COUNT(*) FROM Personnel GROUP BY GROUPING SETS
    ((dept_name), (job_title));

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	
2016	BMW	300	(year, brand)
2016	VW	450	
2016	Gen. Null	1000	(year)

Grouping Operators: Equivalences

- CUBE(a,b) ≡ GROUPING SETS ((a,b), (a), (b), ())
- ROLLUP(a,b,c) ≡ GROUPING SETS ((a,b,c), (a,b), (a), ())
- GROUP BY GROUPING SETS (a,b,c) ≡
  GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
- GROUP BY GROUPING SETS ((a,b,c)) ≡ GROUP BY a, b, c
- GROUP BY GROUPING SETS (a,b,(b,c)) ≡
  GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
- GROUP BY GROUPING SETS (a,ROLLUP(b,c)) ≡
  GROUP BY a UNION ALL GROUP BY ROLLUP(b, c)

Identification of Groupings

- With rollup and cube, we must provide a possibility to determine the rollup level programmatically.
- The **GROUPING_ID function** is designed for this.
  - GROUPING_ID takes a list of grouping columns as an argument.
  - For each column it returns 1 if its value is NULL because of a rollup, and 0 otherwise.
  - The list of binary digits is interpreted as a binary number and returned as a base-10 number.
- Example for CUBE(a,b):

a	**b**	**Bit vector**	**GROUPING_ID(a,b)**
1	2	0 0	0
1	NULL	0 1	1
NULL	1	1 0	2
NULL	NULL	1 1	3

Concatenated Groupings

- A **concatenated** grouping is specified by listing multiple
  - grouping sets, cubes, and rollups,

  and **produces** the cross-product of groupings from each grouping set
- Example:
  - GROUP BY GROUPING SETS (a,b), GROUPING SETS (c,d)
    produces (a,c), (a,d), (b,c), (b,d)
- A concise and easy way to generate useful combinations of groupings
  - A small number of concatenated groupings can generate a large number of final groups
  - One of the most important uses for concatenated groupings is to generate the aggregates for a hierarchical cube

Hierarchical Cubes

- A hierarchical cube is a data set where the data is aggregated along the rollup hierarchy of each of its dimensions.
  - The aggregations are combined across dimensions
- Example:
  - ROLLUP(year, quarter, month),
    ROLLUP(category,subcategory,name),
    ROLLUP(region,subregion,country,state,city)
  - Groups: (year,category,region), (quarter,category,region),
    (month,category,region), …
  - Produces a total of 4x4x6=96 aggregate groups
    - Compare to 2^11 = 2048 groupings by CUBE and 96 explicit group specifications

Window Functions

- 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>])
  - Moving average of 3 rows:
    - ```
      SELECT … ,
          AVG(sales) OVER (PARTITION BY region
                           ORDER BY month ASC
                           ROWS 2 PRECEDING) AS SMA3, …
      FROM …
      ```
      ```

Ranking Operators

- 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

Ranking Operators

- 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 nondeterministic
    - Each tied value could have its number switched!
- We need something deterministic

Ranking Operators

- 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)
  - DENSE_RANK leaves no gaps in ranking sequence when there are ties
  - PERCENT_RANK ←→ (rank - 1) / (total rows - 1)
  - CUME_DIST - the cumulative distribution
    - the number of partition rows preceding (or peers with) the current row / total partition rows
    - The value ranges from 1/N to 1

Ranking Operators

- E.g.,

```
SELECT channel, calendar,
  TO_CHAR(TRUNC(SUM(amount_sold), -6), '9,999,990’) AS sales,
  RANK() OVER (ORDER BY TRUNC(amount_sold, -6)) DESC) AS rank,
  DENSE_RANK() OVER (ORDER BY TRUNC(SUM(amount_sold), -6)) DESC) AS dense_rank
FROM sales, products WHERE …
GROUP BY channel, calendar
```

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

Ranking Operators

- 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.
  - PARTITION BY clause

```
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

Ntile

- NTILE splits a set into equal-sized 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

```
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

- Not a part of the SQL99 standard, but adopted by major vendors

Window Frame

- Obtain a value of a particular row of a window frame defined by window clause (PARTITION BY…)
  - first_value(expression)
  - last_value(expression)
  - nth_value (expression)

CHANNEL	CALENDAR	SALES	LOWEST_SALE
Direst sales	02.2016	10,000	4,000
Direst sales	03.2016	9,000	4,000
Internet	02.2016	6,000	4,000
Internet	03.2016	6,000	4,000
Partners	03.2016	4,000	4,000

SELECT … FIRST_VALUE(sales) OVER (ORDER BY sales) AS lowest_sale

SELECT … FIRST_VALUE(sales) OVER (PARTITION BY channel ORDER BY sales) AS lowest_sale_per_channel

Window Frame

- Access to a row that comes before the current row at a specified physical offset with the current window frame (partition)
  - LAG(expression [,offset [,default_value]])
- … after the current row
  - LEAD(expression [,offset [,default_value]]

CHANNEL	CALENDAR	SALES	PREV_SALE
Direst sales	02.2016	10,000	NULL
Direst sales	03.2016	9,000	10,000
Internet	02.2016	6,000	NULL
Internet	03.2016	6,000	6,000
Partners	03.2016	4,000	NULL

SELECT … LAG(sales, 1) OVER (PARTITION BY channel ORDER BY calendar) AS prev_sale

Data Densification

- Enrich the existing “holey” data with default values

PROD	YEAR	WEEK	SALES
Deluxe	2001	25	5560
Mouse P	2001	24	2083
Mouse P	2001	26	2501
Standar	2001	24	2394
Standar	2001	26	1280

- Goal: produce dense result for all products
  - i.e., incl. the weeks 24, 25, and 26.

Data Densification

- Partitioned Outer Join
  - Apply outer join on each partition
  - Implemented in Oracle

```
SELECT p_Name, t.Year, t.Week, NVL(Sales,0) dense_sales
FROM ( SELECT   P_Name, T_Cal_Year Year, t_Cal_Week_num Week,
                SUM(S_Amnt_Sold) Sales
       FROM     bi.spctmn
       GROUP BY p_Name, T_Cal_Year, t_Cal_Week_num ) v
     PARTITION BY (v.p_Name)
     RIGHT OUTER JOIN
     ( SELECT DISTINCT t_Cal_Week_num Week, T_Cal_Year Year
       FROM bi.spctmn
       WHERE T_Cal_Year IN (2000, 2001)
       AND t_Cal_Week_num BETWEEN 24 AND 26 ) t
     ON (v.week = t.week AND v.Year = t.Year)
ORDER BY p_name, year, week;
```

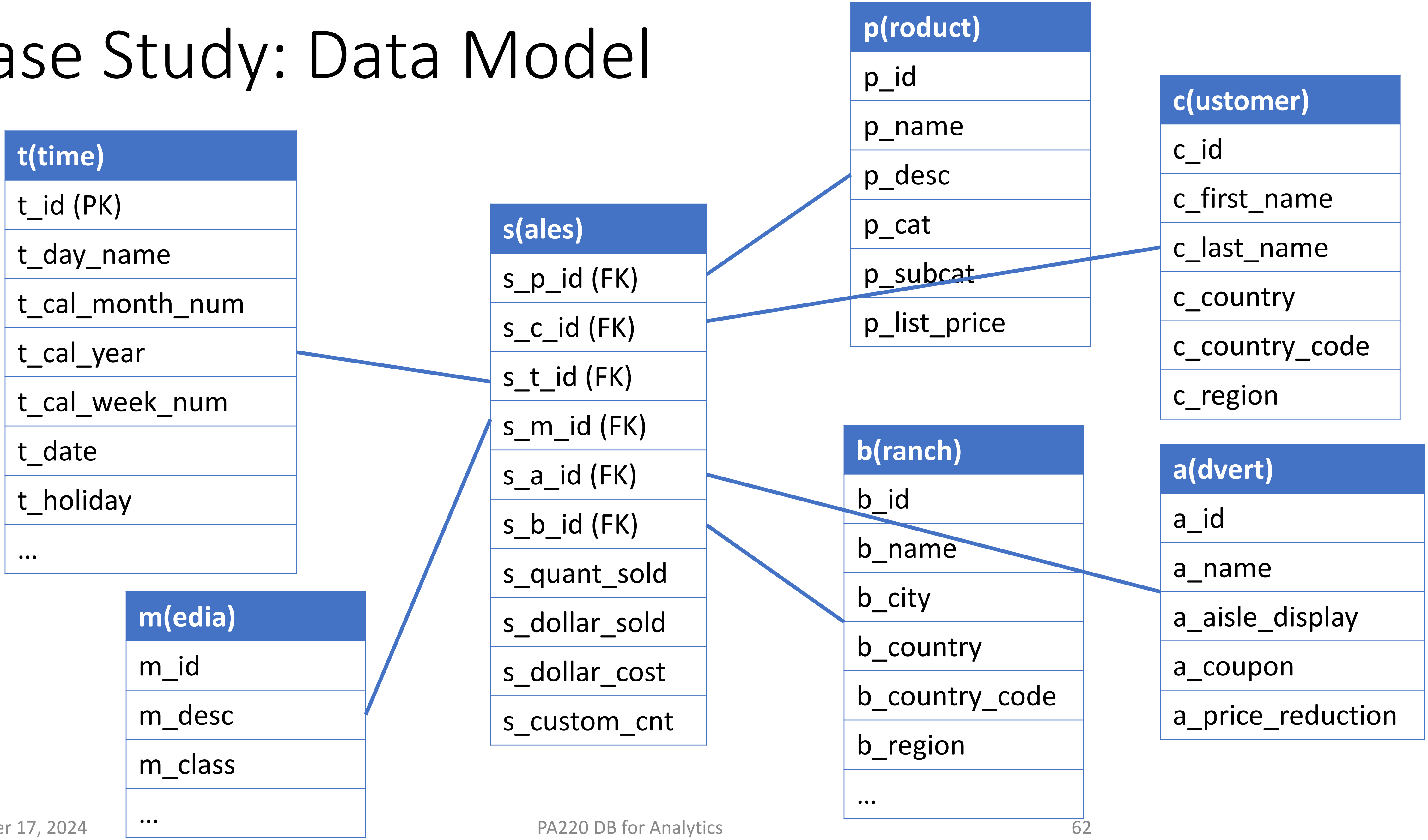
PROD	YEAR	WEEK	DENSE_SALES
Deluxe	2000	24	0.0
Deluxe	2000	25	0.0
Deluxe	2000	26	0.0
Deluxe	2001	24	2260.72
Deluxe	2001	25	1871.3
Deluxe	2001	26	5560.51
Mouse P	2000	24	1685.52
Mouse P	2000	25	494.91
Mouse P	2000	26	1548.2
Mouse P	2001	24	2083.29
Mouse P	2001	25	0.0
Mouse P	2001	26	2501.79
Standar	2000	24	1007.37
Standar	2000	25	339.36
Standar	2000	26	183.92
Standar	2001	24	2394.04
Standar	2001	25	0.0
Standar	2001	26	1280.97

Case Study: Grocery Store (from prev. lecture)

- Stock Keeping Units (SKUs)
- Point Of Sale (POS) system
- Stores/Branches
- Promotions

- Task: Analyze how promotions (adverts) affect sales

Case Study: Data Model



Case Study: Dimensions

- Time dimension
  - Explicit time dimension is needed (events, holidays,..)
- Product dimension
  - Many-level hierarchy allows drill-down/roll-up
  - Many descriptive attributes (often more than 50)
- Branch (store) dimension
  - Many descriptive attributes
- Advert (promotion) dimension
  - Example of a causal dimension
  - Used to see if promotions work/are profitable
  - Ads, price reductions, end-of-aisle displays, coupons

Case Study: Measures (Facts)

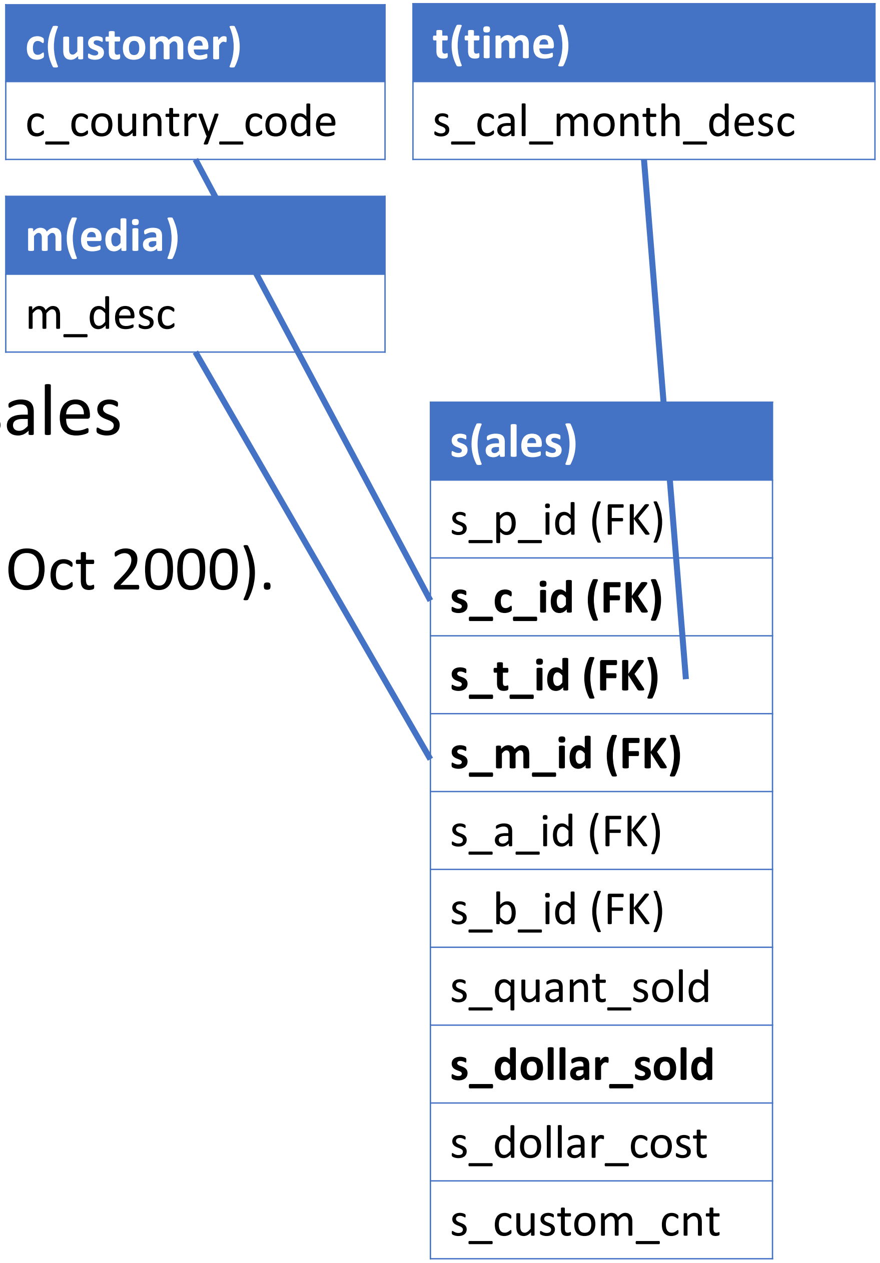
- Dollar_sales, Unit_sales, Dollar_cost
  - All additive across all dimensions
- Gross profit (derived)
  - Computed from sales and cost: `sales – cost`
  - Additive
- Gross margin (derived)
  - Computed from gross profit and sales: `(sales – cost)/cost`
  - Non-additive
    - sum() makes not sense across any dimension since it is a ratio; rather avg() is sensible
- Customer_count
  - Semi-additive
    - i.e., additive across time, promotion, and store
    - i.e., non-additive across product. Why?

Case Study: Common View

- A common view is created to simplify queries

```
CREATE VIEW spcbatm
SELECT *
FROM sales s JOIN product p ON (s_p_id=p_id)
     JOIN customer c ON (s_c_id=c_id)
     JOIN branch b ON (s_b_id=b_id)
     JOIN advert a ON (s_a_id=a_id)
     JOIN time t ON (s_t_id=t_id)
     JOIN media m ON (s_m_id=m_id)
```

Case Study: Query Formulation



- Task: Analyze how promotions (adverts) affect sales
  - Assume specific customer countries (GB and US), media types (Direct and Internet) and months (Sep, Oct 2000).

```
SELECT m_desc, t_cal_month_desc, c_country_code,
       SUM(s_dollar_sold)
FROM spcbatm
WHERE m_desc IN ('Direct Sales', 'Internet')
AND t_cal_month_desc IN ('2000-09', '2000-10')
AND c_country_code IN ('GB', 'US')
GROUP BY m_desc, t_cal_month_desc, c_country_code;
```

Case Study: Make a pivot table

- Compute subtotals:
  - Rollup from right to left
- Computes and combines the following groupings
  - m_desc, t_cal_month_desc, c_country_code
  - m_desc, t_cal_month_desc
  - m_desc
  - _

```
SELECT m_desc, t_cal_month_desc, c_country_code,
       SUM(s_dollar_sold)
FROM spcbatm
WHERE m_desc IN ('Direct Sales', 'Internet')
AND t_cal_month_desc IN ('2000-09', '2000-10')
AND c_country_code IN ('GB', 'US')
GROUP BY ROLLUP(m_desc, t_cal_month_desc, c_country_code);
```

Case Study: Roll up Example

M_DESC	T_CAL_MO	C_	SUM(S_amount_sold)
Internet	2000-09	GB	16569.36
Internet	2000-09	US	124223.75
Internet	2000-10	GB	14539.14
Internet	2000-10	US	137054.29
Direct Sales	2000-09	GB	85222.92
Direct Sales	2000-09	US	638200.81
Direct Sales	2000-10	GB	91925.43
Direct Sales	2000-10	US	682296.59
Internet	2000-09		140793.11
Internet	2000-10		151593.43
Direct Sales	2000-10		774222.02
Direct Sales	2000-09		723423.73
Internet			292386.54
Direct Sales			1497645.75
			1790032.29

Case Study: Make a pivot table (2)

- Partial Roll up
  - m_desc is always present and not part of the rollup hierarchy
- Computes and combines the following groupings
  - m_desc, t_cal_month_desc, c_country_code
  - m_desc, t_cal_month_desc
  - m_desc

```
SELECT m_desc, t_cal_month_desc, c_country_code,
       SUM(s_dollar_sold)
FROM spcbatm
WHERE m_desc IN ('Direct Sales', 'Internet')
AND t_cal_month_desc IN ('2000-09', '2000-10')
AND c_country_code IN ('GB', 'US')
GROUP BY m_desc, ROLLUP(t_cal_month_desc, c_country_code);
```

Case Study: Partial Roll up

M_DESC	T_CAL_MO	N_	SUM(S_amount_sold))
Internet	2000-09	GB	16569.36
Internet	2000-09	US	124223.75
Internet	2000-10	GB	14539.14
Internet	2000-10	US	137054.29
Direct Sales	2000-09	GB	85222.92
Direct Sales	2000-09	US	638200.81
Direct Sales	2000-10	GB	91925.43
Direct Sales	2000-10	US	682296.59
Internet	2000-09		140793.11
Internet	2000-10		151593.43
Direct Sales	2000-09		723423.73
Direct Sales	2000-10		774222.02
Internet			292386.54
Direct Sales			1497645.75

Case Study: Make a pivot table (3)

- Produces all possible roll-up combinations
  - CUBE operator
- Computes and combines the following groupings
  - m_desc, t_cal_month_desc, c_country_code
  - m_desc, t_cal_month_desc
  - m_desc, c_country_code
  - m_desc
  - t_cal_month, c_country_code
  - t_cal_month
  - c_country_code
  - -

```
SELECT m_desc, t_cal_month_desc, c_country_code,
       SUM(s_dollar_sold)
FROM spcbatm
WHERE m_desc IN ('Direct Sales', 'Internet')
AND t_cal_month_desc IN ('2000-09', '2000-10')
AND c_country_code IN ('GB', 'US')
GROUP BY CUBE(m_desc, t_cal_month_desc,
              c_country_code);
```

Case Study: Cube Example

M_DESC	T_CAL_MO	N_	SUM(S_DOLLAR_SOLD)
Internet	2000-09	GB	16569.36
Internet	2000-09	US	124223.75
Internet	2000-10	GB	14539.14
Internet	2000-10	US	137054.29
Direct Sales	2000-09	GB	85222.92
Direct Sales	2000-09	US	638200.81
Direct Sales	2000-10	GB	91925.43
Direct Sales	2000-10	US	682296.59
	2000-09	GB	101792.28
	2000-09	US	762424.56
	2000-10	GB	106464.57
	2000-10	US	819350.88
Internet		GB	31108.5
Internet		US	261278.04
Direct Sales		GB	177148.35
Direct Sales		US	1320497.4
Internet	2000-09		140793.11
Internet	2000-10		151593.43
Direct Sales	2000-09		723423.73
Direct Sales	2000-10		774222.02
Internet			292386.54
Direct Sales			1497645.75
	2000-09		864216.84
	2000-10		925815.45
		GB	208256.85
		US	1581775.44
			1790032.29

Case Study: Make a pivot table (4)

- Specific group-by's
  - Grouping sets

```
SELECT m_desc, t_cal_month_desc, c_country_code,
       SUM(s_dollar_sold)
FROM spcbatm
WHERE m_desc IN ('Direct Sales', 'Internet’)
  AND t_cal_month_desc IN ('2000-09', '2000-10’)
  AND c_country_code IN ('GB', 'US')
GROUP BY GROUPING SETS ((m_desc, t_cal_month_desc, c_country_code),
                        (m_desc, c_country_code),
                        (t_cal_month_desc, c_country_code));
```

Grouping Sets

M_DESC	T_CAL_MO	C_	SUM(S_DOLLAR_SOLD)
Internet	2000-09	GB	16569.36
Direct Sales	2000-09	GB	85222.92
Internet	2000-09	US	124223.75
Direct Sales	2000-09	US	638200.81
Internet	2000-10	GB	14539.14
Direct Sales	2000-10	GB	91925.43
Internet	2000-10	US	137054.29
Direct Sales	2000-10	US	682296.59
	2000-09	GB	101792.28
	2000-09	US	762424.56
	2000-10	GB	106464.57
	2000-10	US	819350.88
Internet		GB	31108.5
Internet		US	261278.04
Direct Sales		GB	177148.35
Direct Sales		US	1320497.4

Grouping ID Example

- Replaces all NULLs from rollup with string ‘*’.
- Leaves NULL that are not the result of rollup untouched.
- Could easily make selective replacements of NULL.

```
SELECT
  CASE WHEN GROUPING_ID(m_desc)=1 THEN ‘*' ELSE m_desc END,
  CASE WHEN GROUPING_ID(c_country_code)=1 THEN ‘*' ELSE c_country_code END,
  SUM(s_dollar_sold)
FROM spcbatm
WHERE m_desc IN ('Direct Sales', 'Internet’)
  AND t_cal_month_desc= '2000-09’
  AND c_country_code IN ('GB', 'US')
GROUP BY CUBE(m_desc, c_country_code);
```

Grouping ID Example

CASEWHENGROUPING(M_D	CAS	SUM(S_DOLLAR_SOLD)
Internet	GB	16569.36
Internet	US	124223.75
Direct Sales	GB	85222.92
Direct Sales	US	638200.81
Direct Sales	*	723423.73
Internet	*	140793.11
*	GB	101792.28
*	US	762424.56
*	*	864216.84

Case Study: Ranking sales

- Rank dollar sales by the media (‘Internet’ versus ‘Direct sales’)
  - Do the analysis for August until November 2000.
  - Use the number of unit sales to break ties.

```
M_DESC       T_CAL_MONTH  SUM(s_dollar_s SUM(s_quant_s  Rank
------------------------------------------------------------
Direct Sales 2000-11            799830.48         396180      1
Direct Sales 2000-10            774222.02         306868      2
Direct Sales 2000-09            723423.73         286950      3
Direct Sales 2000-08            723423.73         286045      4
Internet     2000-10            151593.43         100425      5
Internet     2000-11            145874.92          98760      6
Internet     2000-09            140793.11          96461      7
Internet     2000-08            138953.02          94568      8
```

Case Study: Ranking sales

- Rank dollar sales by the media (‘Internet’ versus ‘Direct sales’)
  - Do the analysis for August until November 2000.
  - Use the number of unit sales to break ties.

```
SELECT m_desc, t_cal_month_desc, SUM(s_dollar_sold), SUM(s_quant_sold),
       RANK() OVER (ORDER BY SUM(s_dollar_sold) DESC,
                             SUM(s_quant_sold) DESC) AS Rank
FROM spcbatm
WHERE m_desc IN ('Direct Sales', 'Internet'),
AND t_cal_month_desc IN ('2000-08', '2000-09’, '2000-10', '2000-11')
GROUP BY m_desc, t_cal_month_desc;
```

Case Study: Ranking sales (2)

- Determine the two least and most successful sales media, respectively (in terms of total amount sold).

```
SELECT * FROM (SELECT m_desc, SUM(s_dollar_sold),
                      RANK() OVER (ORDER BY SUM(s_dollar_sold)) worst,
                      RANK() OVER (ORDER BY SUM(s_dollar_sold) DESC) best
               FROM spcbatm
               GROUP BY m_desc)
WHERE worst < 3 OR best < 3;
```

M_DESC	SUM(S_DOLLAR_SOLD)	Worst	Best
Direct Sales	57875260	4	1
Partners	26346342	3	2
Internet	13706802	2	3
Tele Sales	277426	1	4

Case Study: Ranking sales (3)

- Determine the output of the following statement:

```
SELECT c_id, p_id, RANK() OVER (ORDER BY p_id) AS r1,
                   RANK() OVER (ORDER BY c_id) AS r2,
                   RANK() OVER (ORDER BY 1) AS r3,
                   RANK() OVER (PARTITION BY c_id ORDER BY p_id) AS r4,
                   RANK() OVER (PARTITION BY p_id ORDER BY c_id) AS r5
FROM spcbatm
WHERE c_id in (214, 608, 699)
  AND p_id in (42, 98, 123)
GROUP BY c_id, p_id;
```

C_ID	P_ID	R1	R2	R3	R4	R5
214	123	3	1	1	1	1
608	42	1	2	1	1	1
608	123	3	2	1	2	2
699	42	1	4	1	1	2
699	123	3	4	1	2	3

Summary (Takeaways)

- ROLAP is a good option that exploits existing investments
  - You should know difference between ROLAP and MOLAP
- SQL:2003 has added a lot of support for OLAP operations
  - SQL is not just select-from-where
- Types of queries
  - Navigational vs aggregation
  - Operations: roll up / drill down, slicing, dicing, pivoting
    - dimensional vs hierarchical roll up
- Data densification