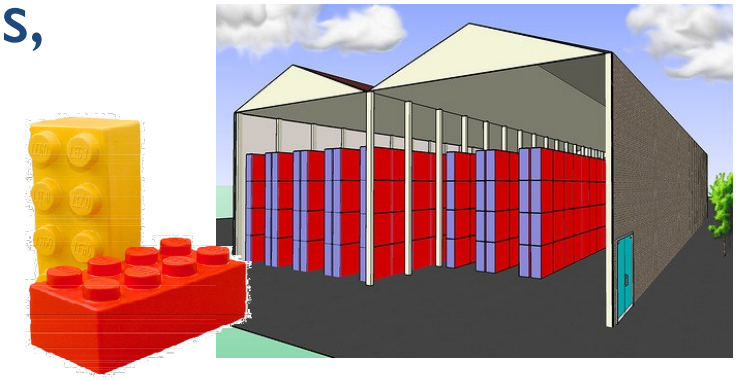
Lecture 06

Data Warehouse Architecture (cont'd.)

Physical Model

Summary – last week

- Logical Model
  - Cubes, Dimensions, Hierarchies, Classification Levels



- This week:
- Physical Model
  - Relational Implementation through:
    - Star schema: improves query performance for often-used data
    - Snowflake schema: reduce the size of the dimension tables
  - Multidimensional implementation

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Physical Model

- Defining the **physical structures**
  - Setting up the database environment
  - Performance tuning strategies (next lecture)
    - Indexing
    - Partitioning
    - Materialization
- Goal:
  - Define the actual storage architecture
  - Decide on how the data is to be accessed and how it is arranged

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Physical Model (cont'd.)

- Physical implementation of the multidimensional paradigm model can be:
  - Relational
    - Snowflake-schema
    - Star-schema
  - Multidimensional
    - Matrixes

Relational Model

- **Relational** model, goals:
  - As low loss of semantically knowledge as possible e.g., classification hierarchies
  - The translation from multidimensional queries must be efficient – as they will be in SQL
  - The maintenance of the present tables should be easy and fast e.g., when loading new data

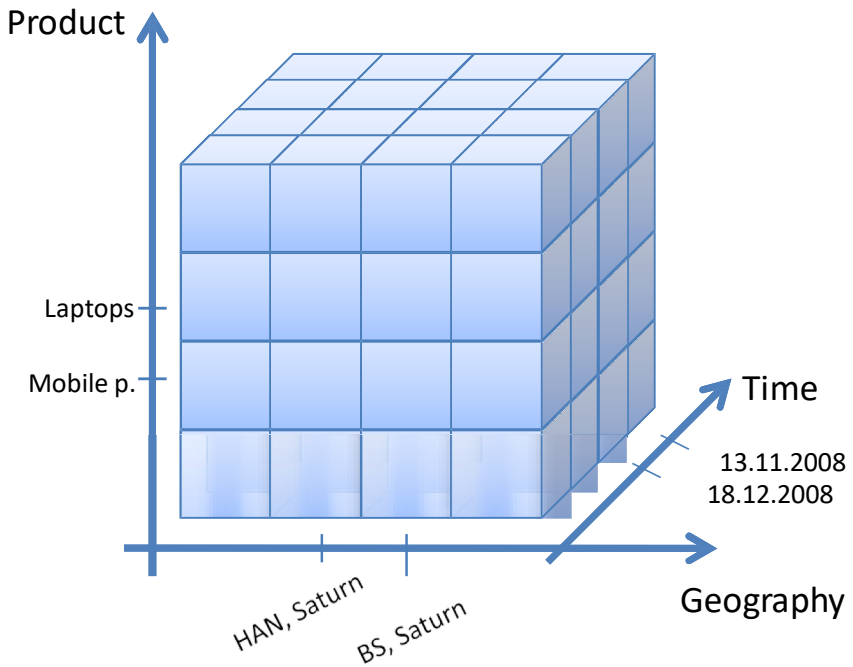
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Relational Model (cont'd.)

- Going from **multidimensional** to **relational**
  - **Representations** for cubes, dimensions, classification hierarchies and attributes
  - **Implementation** of cubes without the classification hierarchies is easy
    - A table can be seen as a *cube*
    - A column of a table can be considered as a dimension mapping
    - A tuple in the table represents a cell in the cube
    - If we interpret only a part of the columns as dimensions we can use the rest as measures
    - The resulting table is called a **fact table**

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Relational Model (cont'd.)



Article	Store	Day	Sales
Laptops	Hannover, Saturn	13.11.2008	6
Mobile Phones	Hannover Saturn	18.12.2008	24
Laptops	Braunschweig Saturn	18.12.2008	3

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Snowflake Schema

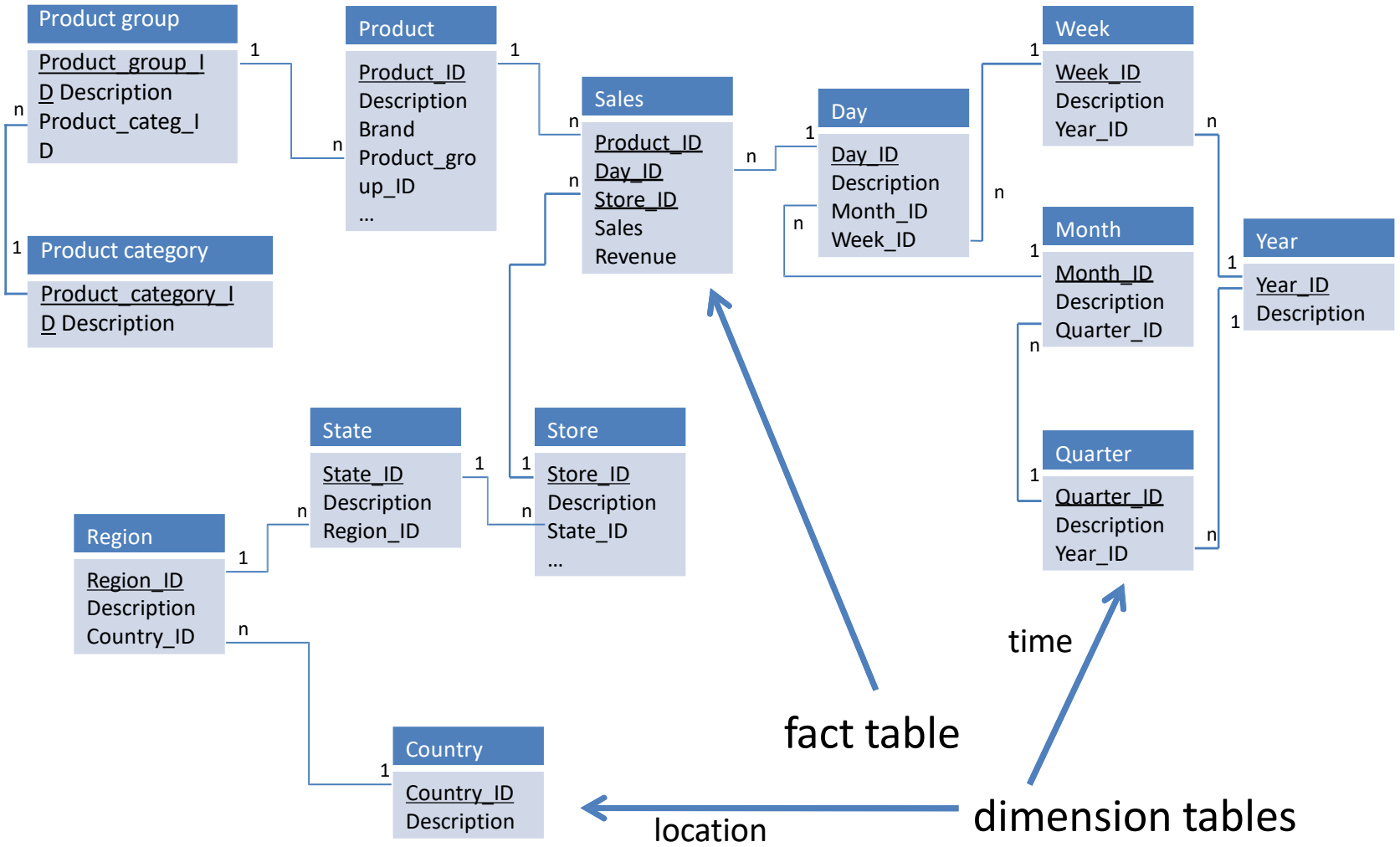
- Snowflake-schema
  - Simple idea: use a table for each classification level
    - This table includes the ID of the classification level and other attributes
    - 2 neighbor classification levels are connected by 1:n connections e.g., from n Days to 1 Month
    - The measures of a cube are maintained in a fact table
    - Besides measures, there are also the foreign key IDs for the smallest classification levels

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Snowflake Schema (cont'd.)

- Snowflake?
  - The facts/measures are in the center
  - The dimensions spread out in each direction and branch out with their granularity

Snowflake Schema (cont'd.)



Snowflake Schema (cont'd.)

- Snowflake schema – **Advantages**
  - With a snowflake schema the **size of the dimension tables** will be reduced and queries will run faster
    - If a dimension is **very sparse** (most measures corresponding to the dimension have no data)
    - And/or a dimension **has long list of attributes** which may be queried

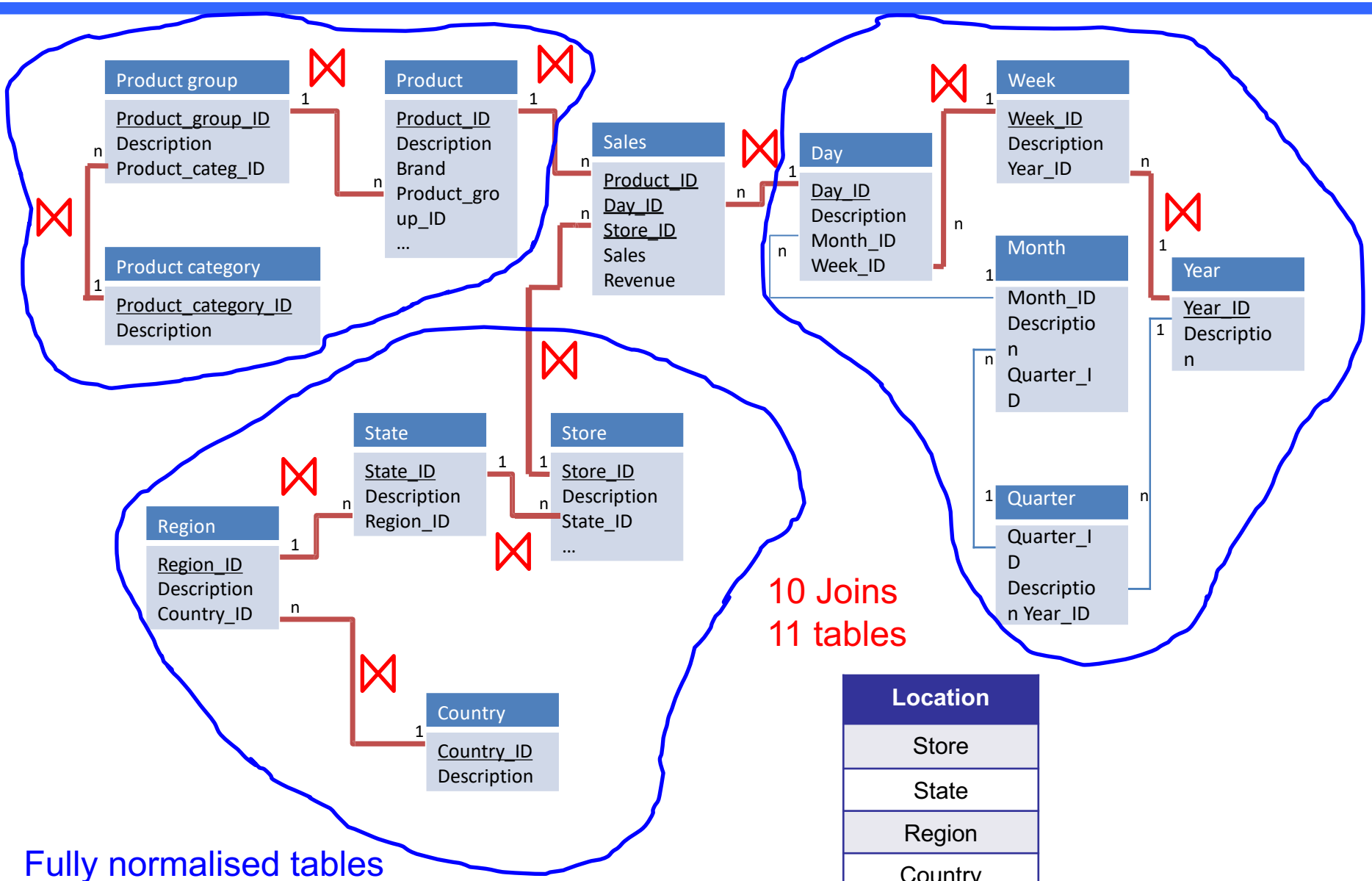
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Snowflake Schema (cont'd.)

- Snowflake schema – **Disadvantages**
  - Fact tables are responsible for 90% of the storage requirements (due to FKs)
    - Thus, normalizing the dimensions usually lead to insignificant improvements
  - Normalization of the dimension tables can reduce the performance of the DW because it leads to a large number of tables
    - E.g., when connecting dimensions with coarse granularity these tables are joined with each other during queries
    - A query which connects Product category with Year and Country is clearly not performant (10 tables need to be connected)

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Snowflake Schema (cont'd.)



10 Joins
11 tables

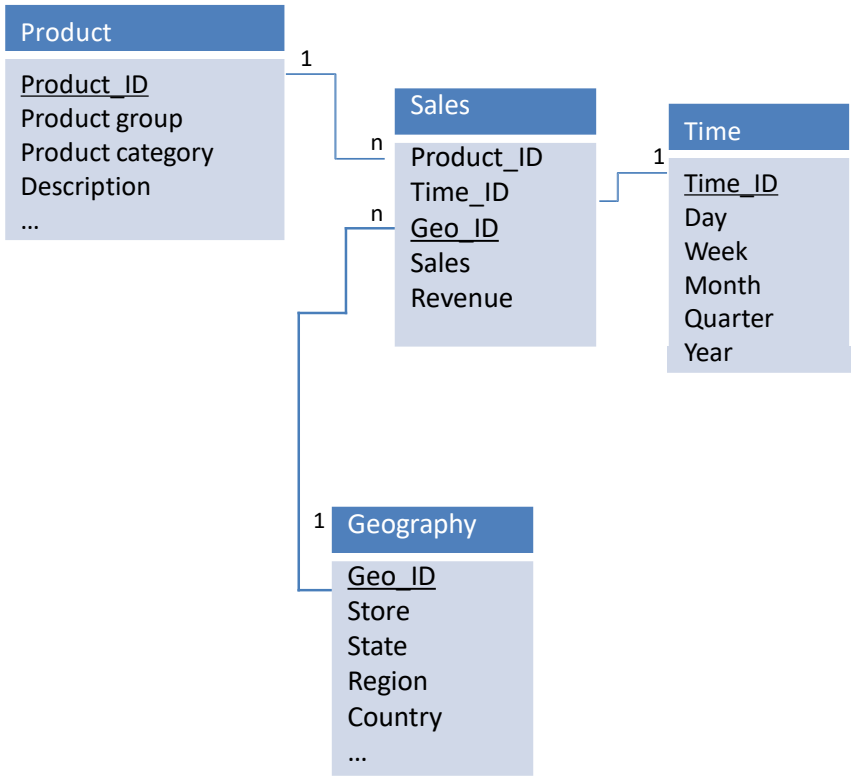
Location
Store
State
Region
Country

Fully normalised tables

Star Schema

- Star schema
  - Basic idea: use a denormalized schema for all the dimensions
    - A star schema can be obtained from the snowflake schema through the denormalization of the tables belonging to a dimension

Star Schema (cont'd.)



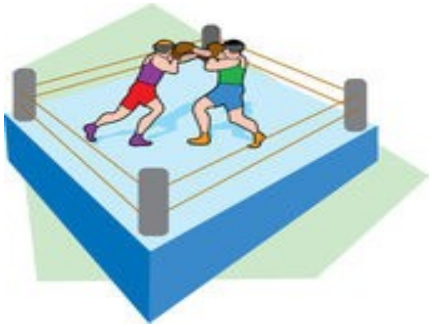
Star Schema (cont'd.)

- Advantages
  - Improves query performance for often-used data
  - Less tables and simple structure
  - Efficient query processing with regard to dimensions
- Disadvantages
  - In some cases, high overhead of redundant data

Snowflake vs. Star Schema

Snowflake vs. Star

- The structure of the classifications are expressed in table schemas
- The fact table and dimension tables are normalized



- The entire classification is expressed in just one table
- The fact table is normalized while in the dimension table the normalization is broken
  - This leads to redundancy of information in the dimension tables

If dimension table is bigger then star schema is a bad idea. While, if query needs high level of granularity (e.g. country or year etc.) then snowflake is a bad idea.

Examples

- Snowflake

Product_ID	Description	Brand	Prod_group_ID
10	E71	Nokia	4
11	PS-42A	Samsung	2
12	5800	Nokia	4
	Bold	Berry	4

Prod_group_ID	Description	Prod_categ_ID
2	TV	11
4	Mobile Pho..	11

Prod_categ_ID	Description
11	Electronics

- Star

Product _ ID	Description	...	Prod. group	Prod. categ
10	E71	...	Mobile Ph..	Electronics
11	PS-42A	...	TV	Electronics
12	5800		Mobile Ph..	Electronics
13	Bold		Mobile Ph..	Electronics

Snowflake to Star Schema

- When should we go from Snowflake to star?
  - Heuristics-based decision
    - When typical queries relate to coarser granularity (like product category)
    - When the volume of data in the dimension tables is relatively low compared to the fact table
      - In this case a star schema leads to negligible overhead through redundancy, but performance is improved
    - When modifications on the classifications are rare compared to insertion of fact data
      - In this case these modifications controlled through the data load process of the ETL reducing the risk of data anomalies

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Do we have a winner?

- Snowflake or Star?
  - It depends on the necessity
    - Fast query processing or efficient space usage
  - However, most of the time a mixed form is used
    - The Starflake schema: some dimensions stay normalized corresponding to the snowflake schema, while others are denormalized according to the star schema

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

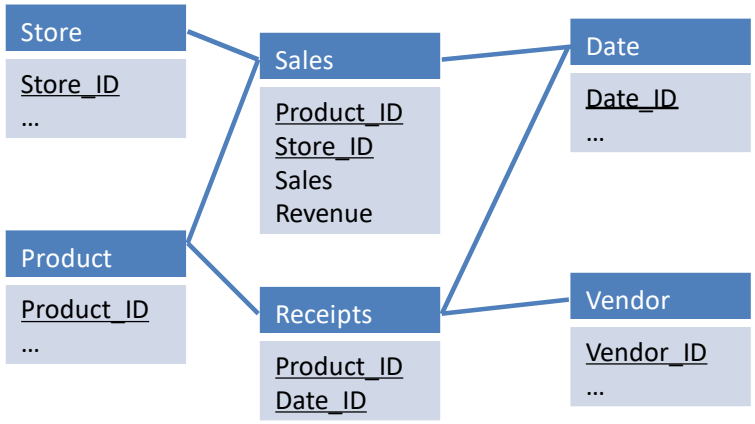
Our Focuses Combined

- The **Starflake schema**
  - The decision on how to deal with the dimensions is influenced by
    - **Frequency** of the modifications: if the dimensions change often, normalization leads to better results
    - **Amount** of dimension elements: the bigger the dimension tables, the more space normalization saves
    - **Number of classification levels** in a dimension: more classification levels introduce more redundancy in the star schema
    - **Materialization of aggregates** for the dimension levels: if the aggregates are materialized, a normalization of the dimension can bring better response time

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

More Schemas

- Galaxies
  - In pratice we usually have more measures described by different dimensions
    - Thus, more fact tables



Even More Schemas

- Other schemas
  - Fact constellations
    - Pre-calculated aggregates
  - Factless fact tables
    - Fact tables do not have non-key data
      - Can be used for event tracking or to inventory the set of possible occurrences
  - …

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Relational Model

- Relational model - **disadvantages**
  - The representation of the multidimensional data can be implemented relationally with a finite set of transformation steps, however:
    - Multidimensional queries have to be first **translated** to the relational representation
    - A **direct interaction** with the relational data model is not fit for the end user

- What about storing the data multidimensionally?

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Multidimensional Model

- The basic data structure for multidimensional data storage is the **array**
- The elementary data structures are the cubes and the dimensions

  $C=((D_1,\ldots,D_n),(M_1,\ldots,M_m))$
- The storage is intuitive as arrays of arrays, physically **linearized**

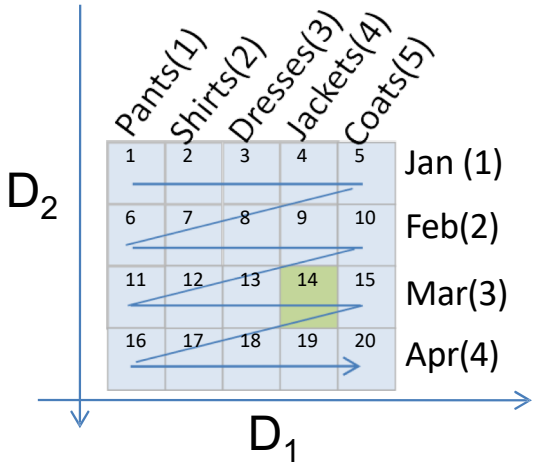
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Multidimensional Model (cont'd.)

- Linearization example: 2D cube $|D_1| = 5$, $|D_2| = 4$, cube cells = 20
  - Query: Jackets sold in March?
    - Measure stored in cube cell $D_1[4], D_2[3]$
    - The 2D cube is physically stored as a linear array, so

      $D_1[4], D_2[3]$ becomes array cell 14

      - (Index(D2) – 1) * |D1| + Index(D1)
      - Linearized Index = 2 * 5 + 4 = 14



- Generalization

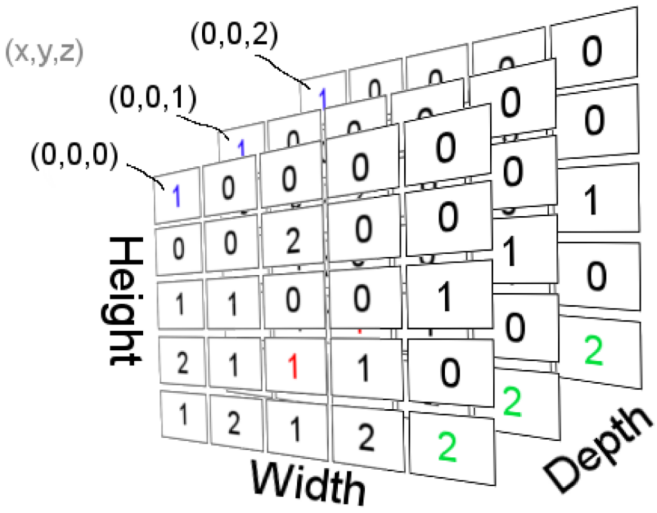
Physically	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Linearization

- Generalization:
  - Given a cube $C = ((D_1, D_2, ..., D_n), (M_1{:}Type_1, M_2{:}Type_2, ..., M_m{:}Type_m))$, the index of a cube cell z with coordinates $(x_1, x_2, ..., x_n)$ can be linearized as follows:
    - $Index(z) = x_1 + (x_2 - 1) * |D_1| + (x_3 - 1) * |D_1| * |D_2| + ... + (x_n - 1) * |D_1| * ... * |D_{n-1}| =$

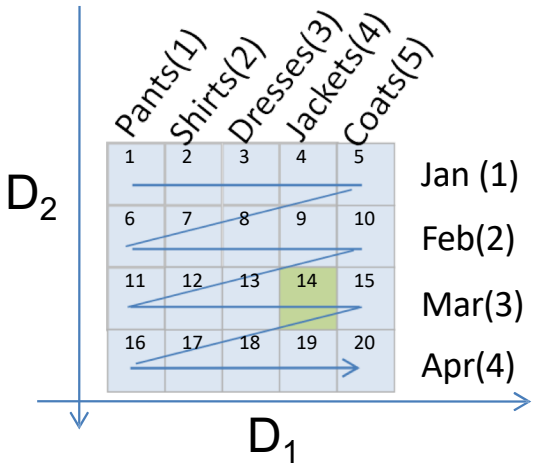
$$= 1 + \sum_{i=1}^{n} ((x_i - 1) * \prod_{j=1}^{i-1} |D_j|)$$



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Problems in Array-Storage

- Influence of the **order of the dimensions** in the cube definition
  - In the cube the cells of D_2 are ordered one under the other
    e.g., sales of all pants involves a **column** in the cube
  - After linearization, the information is **spread** among more data blocks or pages
  - If we consider a data block can hold 5 cells, a query over **all products sold in January** can be answered with just 1 block read, but a query of **all sold pants**, involves reading 4 blocks



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Problems in Array-Storage (cont'd.)

- Solution: use **caching techniques**
  - But…caching and swapping is performed also by the operating system
  - MDBMS has to manage its caches such that the OS doesn't perform any damaging swaps

Problems in Array-Storage (cont'd.)

- Storage of **dense cubes**
  - If cubes are **dense**, array storage is **more efficient**. However, operations suffer due to the large cubes
  - Solution: store dense cubes not linear but on **2 levels**
    - The first contains **indexes** and the second **the data cells** stored in blocks
    - Optimization procedures like **indexes** (trees, bitmaps), physical partitioning, and **compression** (run-length- encoding) can be used

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

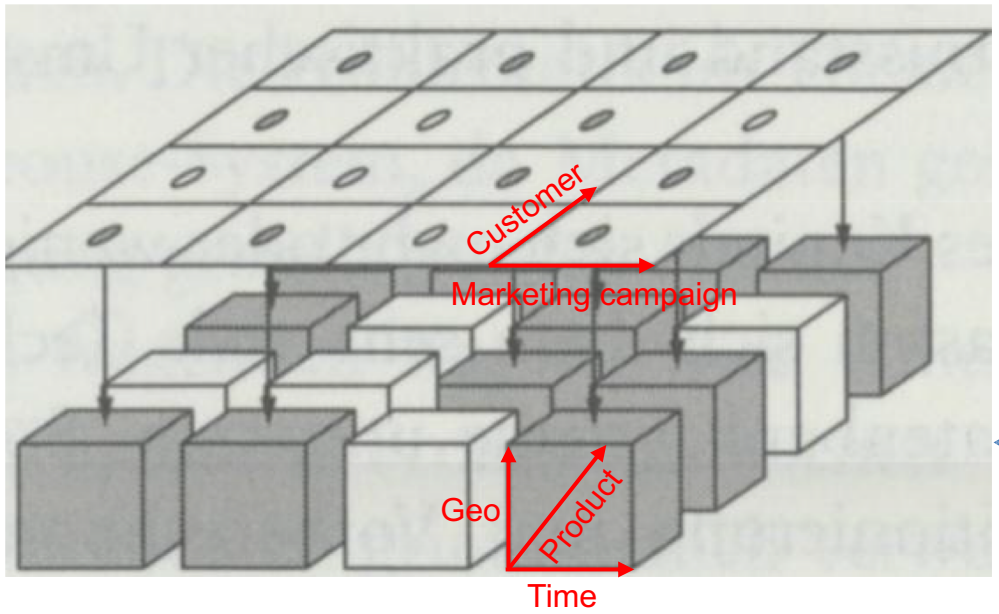
Problems in Array-Storage (cont'd.)

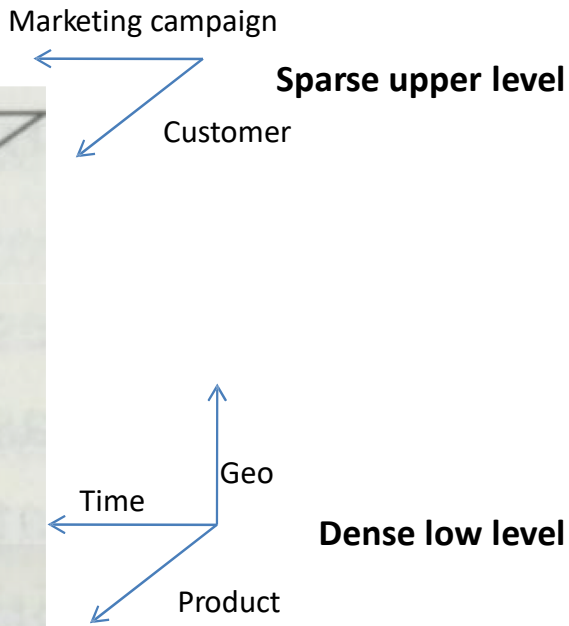
- Storage of **sparse cubes**
  - **All** the cells of a cube, **including empty ones**, have to be stored
  - Sparseness leads to data being stored in many physical blocks or pages
    - The query speed is affected by the large number of block accesses on the secondary memory
  - **Solution**:
    - Do not store empty blocks or pages but adapt the index structure
    - 2 level data structure: upper layer holds all possible combinations of the sparse dimensions, lower layer holds dense dimensions

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Problems in Array-Storage (cont'd.)

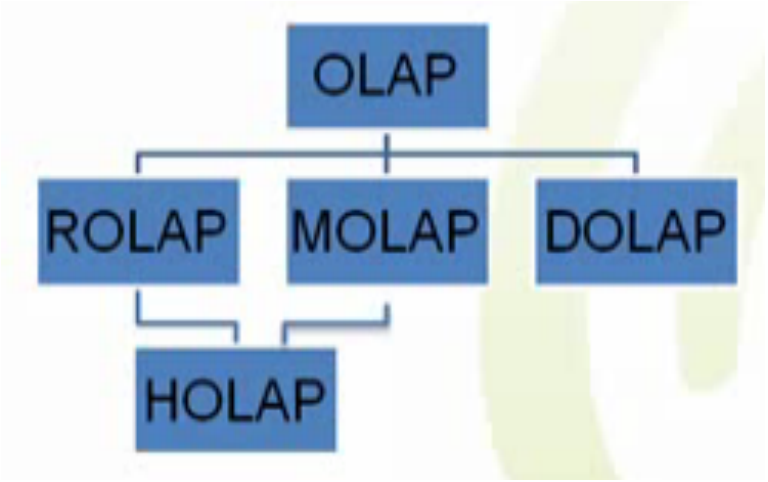
- 2 level cube storage





Physical Models

- Based on the physical model used:
  - DOLAP (Desktop OLAP)
  - MOLAP (Multidimensional OLAP)
  - ROLAP (Relational OLAP)
  - HOLAP (Hybrid OLAP)



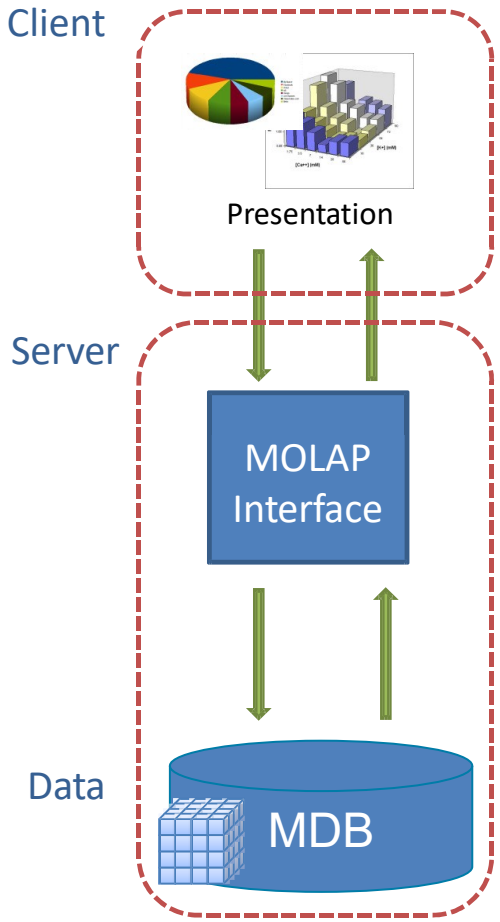
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

DOLAP

- **DOLAP**
  - Developed as extension to the production system reports
    - It downloads a small hypercube from a central point (data mart or DW)
    - Performs multidimensional analysis while disconnected from the data source
    - The computation occurs on the client
  - Requires little investment
  - They lack the ability to manage large data sets

MOLAP

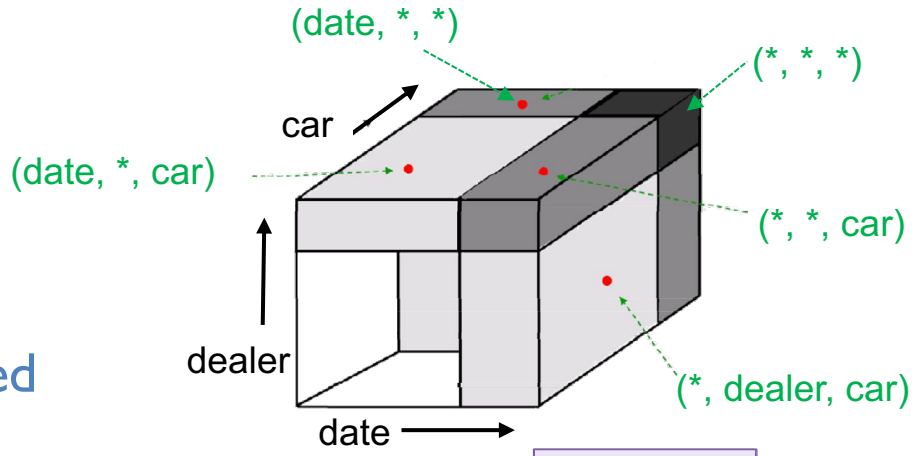
- **MOLAP**
  - Presentation layer provides the multidimensional view
  - The OLAP server stores data in a multidimensional structure
    - Computation occurs in this layer during the loading step (not at query)

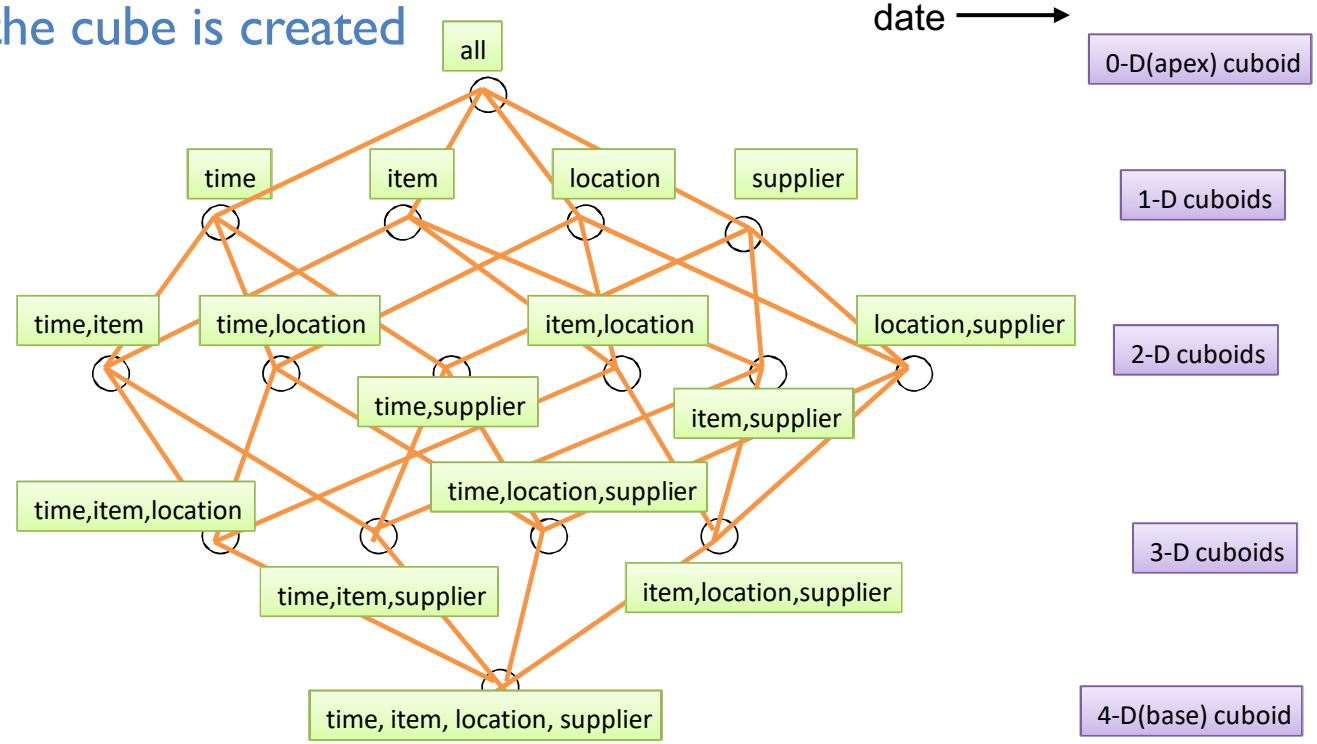


Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

MOLAP (cont'd.)

- Advantages
  - Excellent performance
    - Fast data retrieval
    - Optimal for slicing and dicing
    - All calculations are pre-generated when the cube is created





Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

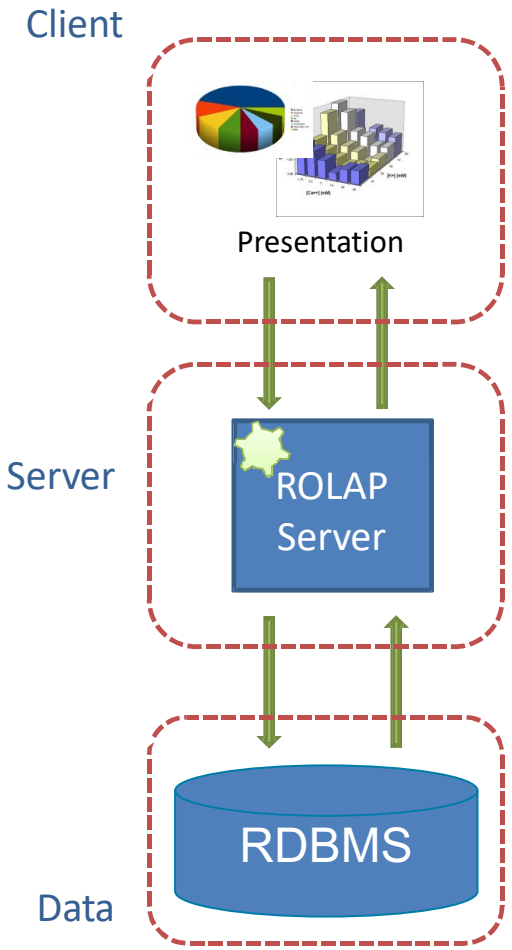
MOLAP (cont'd.)

- Disadvantages
  - Limited amount of data it can handle – due to large calculations
    - Cube can be derived from large amount of data, but only **summary level information** will be included in the cube
  - Requires additional investment
    - Cube technology are often proprietary
  - Enormous amount of overhead
    - An input file of 200 MB can expand to 5 GB with calculations
- Products:
  - Cognos (IBM), Essbase (Oracle), Microsoft Analysis Service, Palo (open source)

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

ROLAP

- ROLAP
  - Presentation layer provides the multidimensional view
  - The ROLAP Server generates SQL queries, from the OLAP OLAP requests, to query the RDBMS
  - Data is stored in **RDBs**



ROLAP (cont'd.)

- Special schema design: e.g., star, **snowflake**
- Special indexes: e.g., bitmap, R-Trees (next lecture)
- Advantages
  - Proven technology (relational model, DBMS)
  - Can handle large amounts of data (VLDBs)
- Disadvantages
  - Limited SQL functionalities
- Products
  - Microsoft Analysis Service, Siebel Analytics (now Oracle BI), Micro Strategy, Mondrian (open source)

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

MOLAP vs. ROLAP

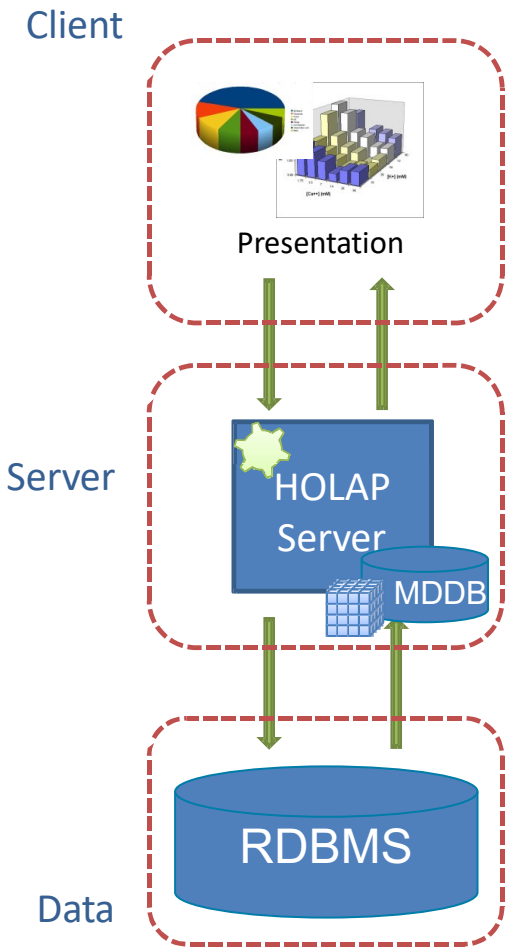
- Based on OLAP needs

OLAP needs		MOLAP	ROLAP
User Benefits	Multidimensional View	√	√
	Excellent Performance	√	-
	Analytical Flexibility	√	-
	Real-Time Data Access	-	√
	High Data Capacity	-	√
MIS Benefits	Easy Development	√	-
	Low Structure Maintenance	-	√
	Low Aggregate Maintenance	√	-

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

HOLAP

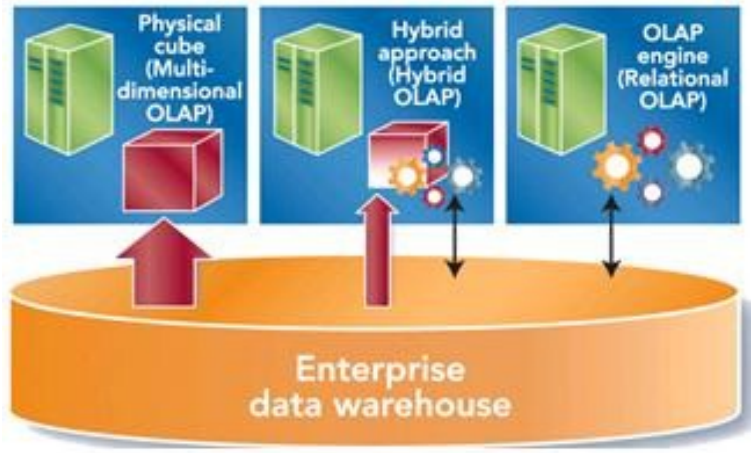
- HOLAP
  - Best of both worlds
  - Split the data b/w MOLAP and ROLAP
    - Vertical partitioning
      - Storing detailed data in RDBs
      - Storing aggregated data in MDBs
    - Horizontal partitioning
      - Storing the recent slice (w.r.t. time) of data in MOLAP
      - Storing the old slice of data in ROLAP



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

HOLAP (cont'd.)

- Other approaches
  - Store some cubes in MOLAP and others in ROLAP, leveraging the fact that in a large cuboid, there will be dense and sparse sub-regions



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Conclusions

- ROLAP
  - RDBMS - star/snowflake schema
  - For detailed and larger volumes of data (TB)
- MOLAP
  - MDBMS - Cube structures, array based storage
  - For summarized and relatively "small" volumes of data (50GB)
- HOLAP is emerging as the OLAP server of choice

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Summary

- Physical Level
  - Relational Implementation through:
    - Star schema: improves query performance for often-used data
      - Less tables and simple structure
      - Efficient query processing with regard to dimensions
      - In some cases, high overhead of redundant data
    - Snowflake schema: reduce the size of the dimension tables
      - However, through dimension normalization - large number of tables

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

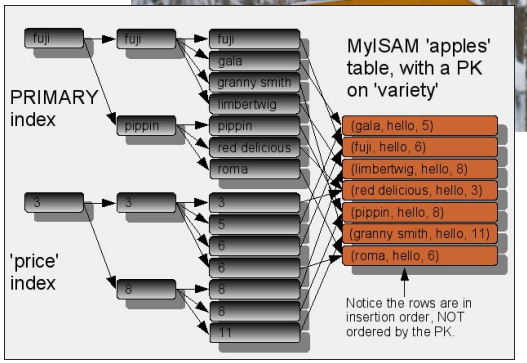
Summary (cont'd.)

- Physical Level
  - Array based storage
    - How to perform linearization
    - Problems:
      - Order of dimensions – solution:caching
      - Dense Cubes,Sparse Cubes - solution:2 level storage
  - MOLAP, ROLAP, HOLAP

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Next Lecture

- DW Optimization / Indexes
  - Bitmap indexes
  - Tree based indexes
  - Hash indexes



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides