Data Intensive Systems (DIS)
KBH-SW7 E25

10. Data Warehouse

AALBORG
UNIVERSITY

Agenda

- Introduction
- Fundamental Multidimensional Modelling
- Relational Representations
- Querying
- Changing Dimensions

Motivation Example

- Say you manage a retail chain. Each day, it generates massive data.
  - Sales transactions, inventory levels, customer memberships, etc.
  - The data may reside in *different* stores or *different* IT systems.
- As the business continuously expands, it is increasingly challenging to make use of the data for decision making inside your organization.
  - **Data fragmentation**: Local data stores vs. comprehensive view of the business
  - **Complex queries**: Inefficient to run across multiple systems.
  - **Inconsistent reporting**: Different shops may generate reports differently.
  - **Limited historical analysis**: Purchase records are maintained in the operational system only for a limited time, e.g., 6 months.

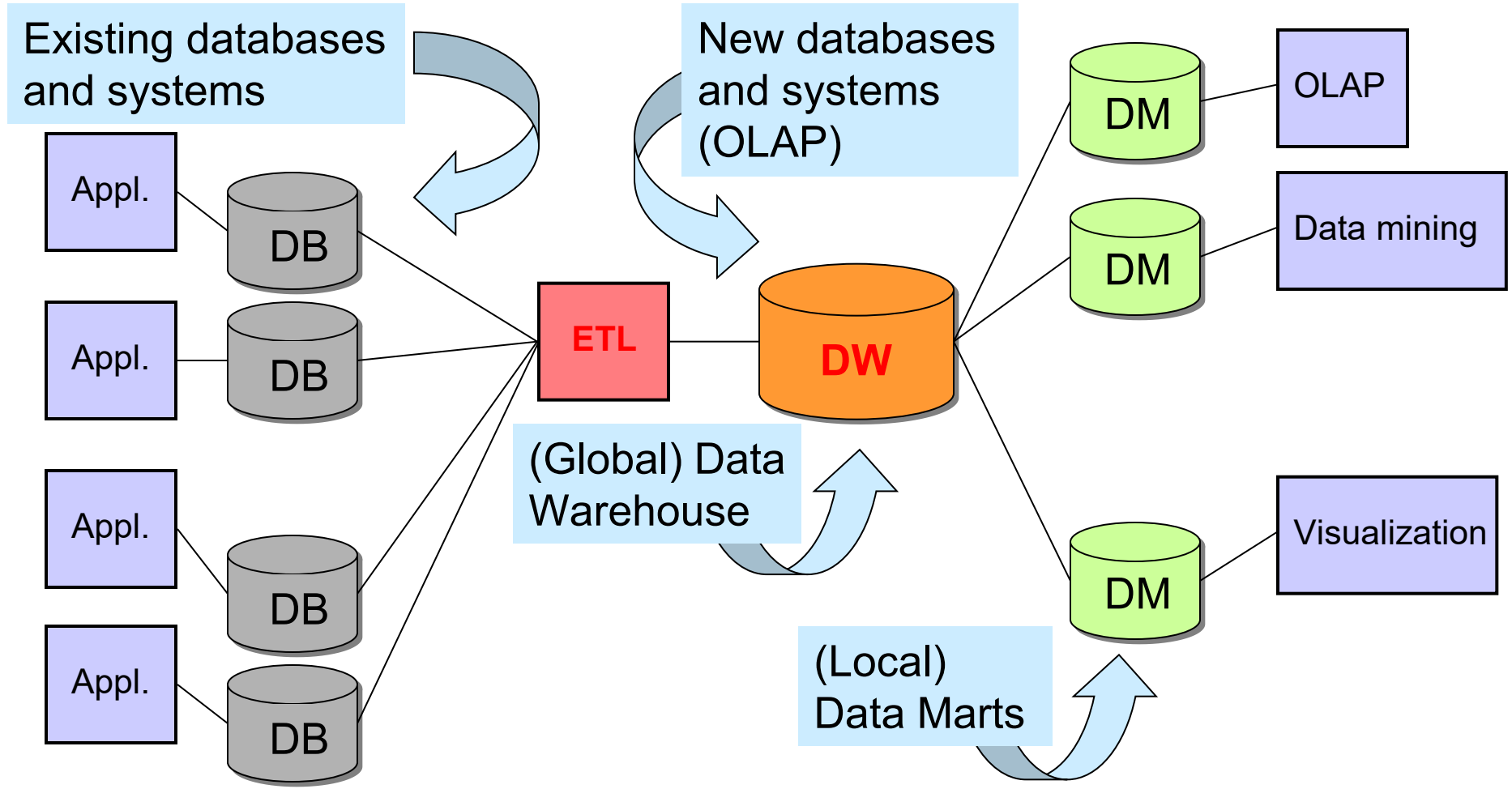
Business Intelligence (BI)

- "[BI] refers to a set of **tools and techniques** that enable a company to **transform its business data** into **timely and accurate information** for the decisional process, to be made available to the right persons **in the most suitable form**." --- Encyclopedia of Database Systems
  - BI is different from Artificial Intelligence (AI)
    - AI *makes* decisions for the users.
    - BI *helps* the users make the right decisions based on available data.
  - BI combines technologies
    - Data Warehousing
    - Online Analytical Processing (OLAP
    - Data Mining (DM)
    - ……

Data Warehouse (DW)

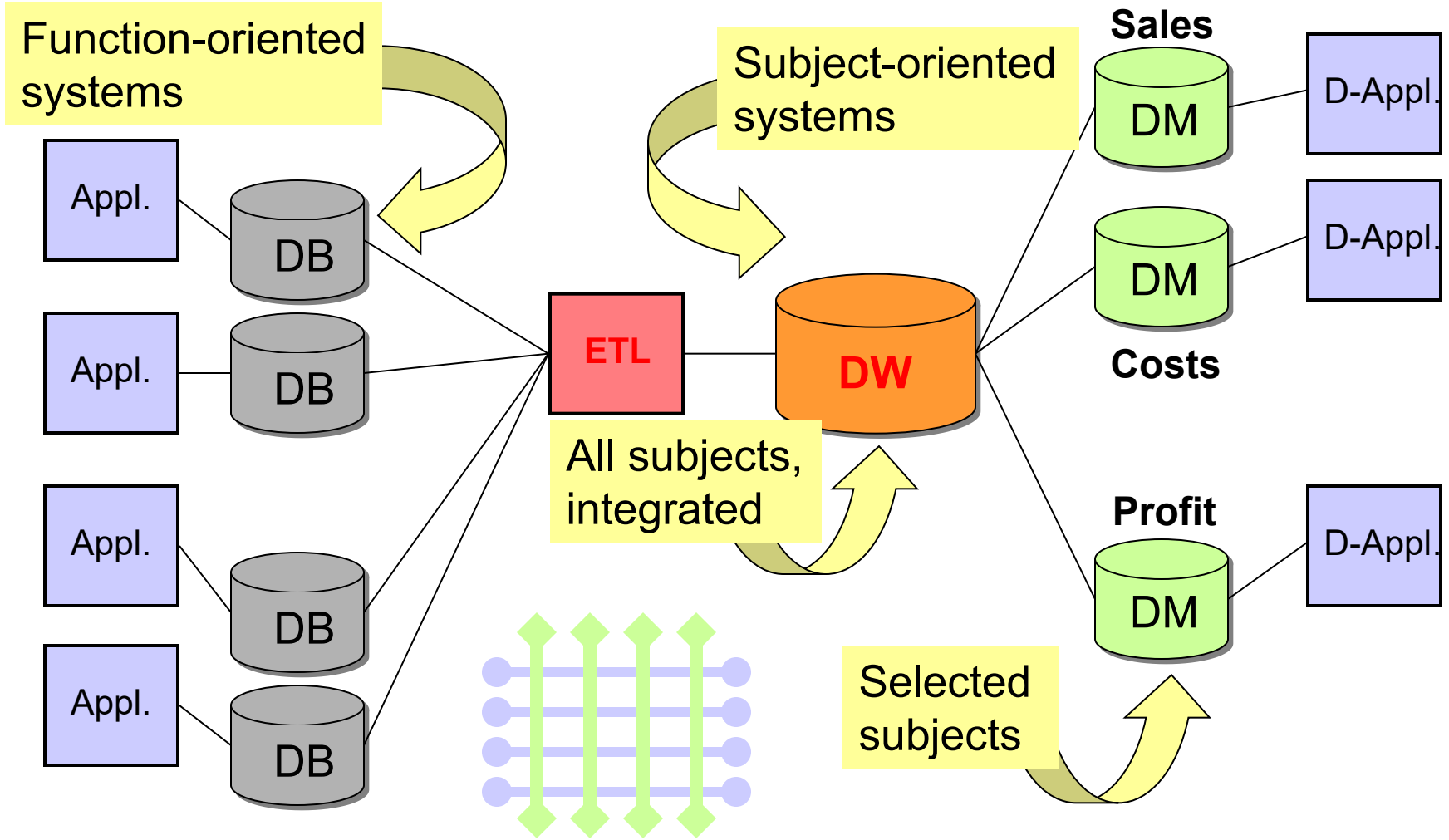
- DW is a system used for **reporting** and **data analysis**. It is considered as a core component of BI. Inside DW, the data is
  - Subject oriented (versus function oriented)
  - Integrated (logically and physically)
  - Time variant (data can always be related to time)
  - Stable (data not deleted, several versions)
  - Supporting management decisions (in or across different organizations)
- Data from the operational systems is
  - **E**xtracted
  - Cleansed
  - **T**ransformed
  - Aggregated (?)
  - **L**oaded into the DW

DW Architecture: Data as Materialized Views



Analogy: (data) producers ↔ warehouse ↔ (data) consumers

Function vs. Subject Orientation



Properties and Benefits of DW

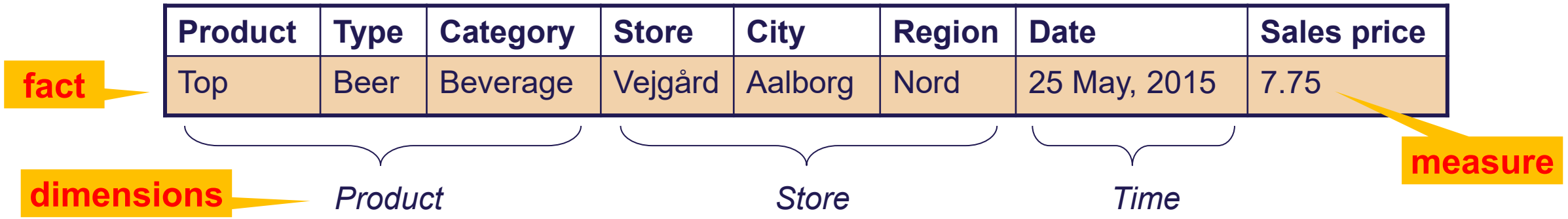
- Centralized Data Repository
- Consistent Data
- Historical Analysis
- Support for Complex Queries
- Performance Optimization
- Business Intelligence (BI) and Reporting
- Support for Decision-making
- Separation of Operational and Analytical Workloads
- Scalability

Agenda

- Introduction
- Fundamental Multidimensional Modelling
- Relational Representations
- Querying
- Changing Dimensions

Multidimensional Modeling: Basic Concepts

- Example: sales from supermarkets
- Facts and measures
  - Each sales record is a **fact**, and its sales value is a **measure**
- Dimensions
  - Group correlated attributes into the same **dimension** ➔ easier for analysis tasks
  - Each sales record is associated with its values of *Product*, *Store*, *Time*



Multidimensional Modeling: Hierarchy

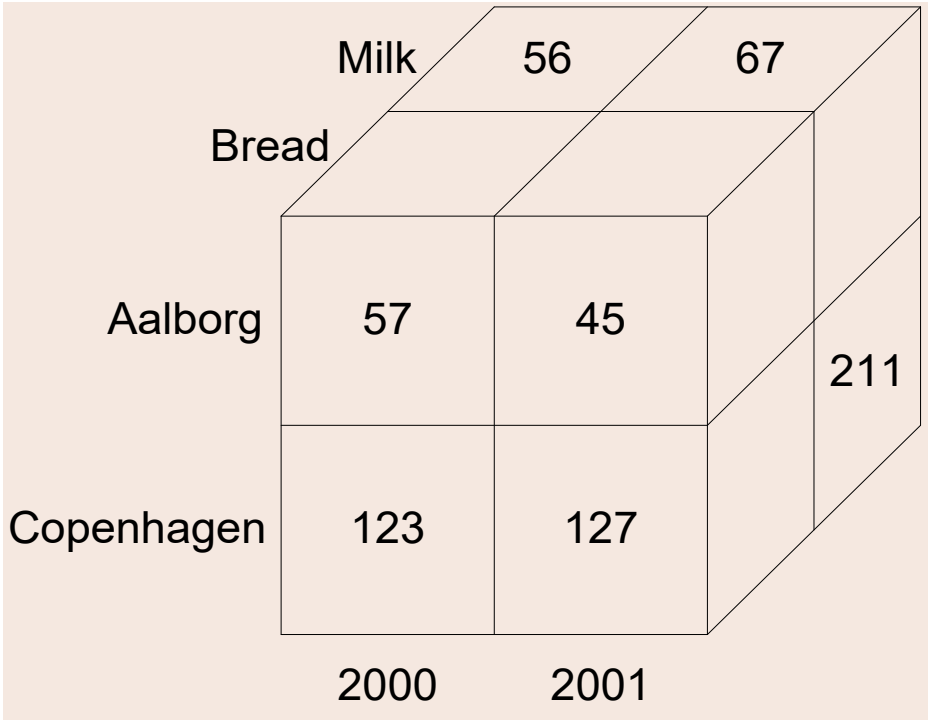
- How do we model the *Time* dimension?
  - **Hierarchy** with multiple levels
  - Attributes, e.g., holiday, event

tid	day	day #	month #	year	week day	work day	…
1	January 1st 2009	1	1	2009	Thurs day	No	…
2	January 2nd 2009	2	1	2009	Fri day	Yes	…
…	…	…	…	…	…	…	…

- Advantage of this model?
  - Easy for querying (more about this later)
- Disadvantage?
  - Data redundancy (but controlled redundancy is acceptable)

Multidimensional Modeling: Cube

- Data is divided into:
  - **Facts**
  - **Dimensions**
- Facts are the *important* entity: a sale
- Facts have **measures** that can be aggregated: sales
- Dimensions *describe* facts
  - A sale has the dimensions Product, Store and Time
- Facts "live" in a multidimensional **cube**



- **Goal** for dimensional modeling:
  - Surround facts with as much context (dimensions) as possible
  - Hint: redundancy may be okay (in well-chosen places)
  - You do not have to model *all* relationships in the data (unlike ER and OO modeling)

Cubes

- A “cube” may have *many* dimensions!
  - It can have more than 3---the term ”hypercube” is sometimes 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 (i.e., there is no data for this combination)
  - A *sparse* cube has many empty cells
  - A *dense* cube has few empty cells
  - Cubes become sparser for many/large dimensions

Dimensions

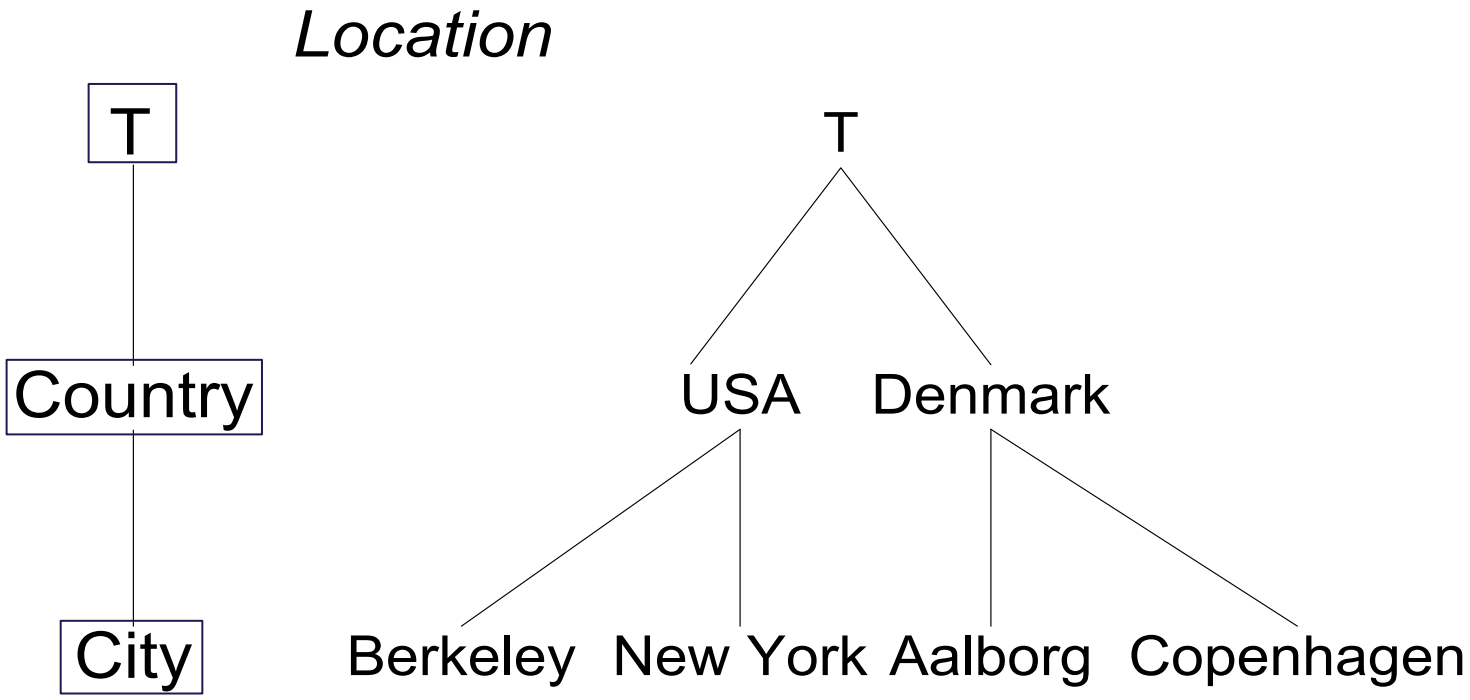
- Dimensions are the core of multidimensional databases
- Dimensions are used for
  - **Selection** of data
  - **Grouping** of data at the right level of detail
- Dimensions consist of **dimension values**
  - The Product dimension has the values ”milk”, ”cream”, …
  - The Time dimension has the values ”1/1/2001”, ”2/1/2001”,…
- Dimension values *can* have an **ordering**
  - Used for comparing cube data across values
  - E.g., ”percent sales increase compared with last month”
  - Especially used for the Time dimension

Hierarchies and Levels

- Dimensions have **hierarchies** with **levels**
  - Typically 3-5 levels
  - **Product**: Product → Type → Category → T
  - **Store**: Store → Area → City → County → T
  - **Time**: Day → Month → Quarter → Year → T
  - Dimensions have a **bottom level** and a **top level** (“T” or “ALL”)
  - Dimension values are organized in a **tree structure**
- Levels *may* have **attributes**
  - Simple, non-hierarchical information
  - E.g., Day has Workday as attribute
- Dimensions should contain much information
  - Time dimensions may contain holiday, season, events,…
  - Good dimensions have 50-100 or more attributes/levels

Dimension Example

- Dimension of *Location*



Facts

- Facts represent the **subject** of the desired analysis
  - What is important to the business
- A fact is identified via its dimension values
  - A fact is a *non-empty* cell

Granularity

- **Granularity** of facts is important.
  - **Level of detail**
  - Given by combination of bottom levels
- Often the granularity is a single business transaction
  - Example: sale
  - Sometimes the data is aggregated (***total*** sales per store per day per product)
  - Might be necessary due to scalability
- Generally, transaction detail can be handled.

Measures

- Measures represent the fact property that the users want to **study and optimize**
  - E.g., sales price
- A measure has two components
  - **Numerical value** (e.g., sales price)
  - **Aggregation formula** (e.g., SUM): used for aggregating/combining a number of measure values into one
  - Measure value is determined by dimension value combination
    - E.g., the sales volume of “milk” and “cream”
  - Measure value is meaningful for *all* aggregation levels (incl. the top level T)

Agenda

- Introduction
- Fundamental Multidimensional Modelling
- Relational Representations
- Querying
- Changing Dimensions

Relational Representation

- Goal for dimensional modeling: surround the facts with as much context (dimensions) as we can
- **Granularity** of the fact table is important
  - What does one fact table row represent?
- Many-to-one relationships
  - Facts vs. dimension values
  - Lower vs. higher levels in the hierarchies

Relational Design

Product	Type	Category	Store	City	County	Date	Sales price
Top	Beer	Beverage	Trøjborg	Århus	Århus	25 May 2009	5.75

Product Store Time

- Naïve solution: One completely de-normalized table
  - This is bad due to inflexibility, storage use, bad performance, slow updates
- Instead, we use **star schemas** or **snowflake schemas** with fact tables and dimension tables

Relational Implementation

- The **fact table** stores facts
  - One row for each fact
  - One column for each measure
  - One column for each dimension (foreign key to dimension table)
  - The dimension keys make up a composite primary key

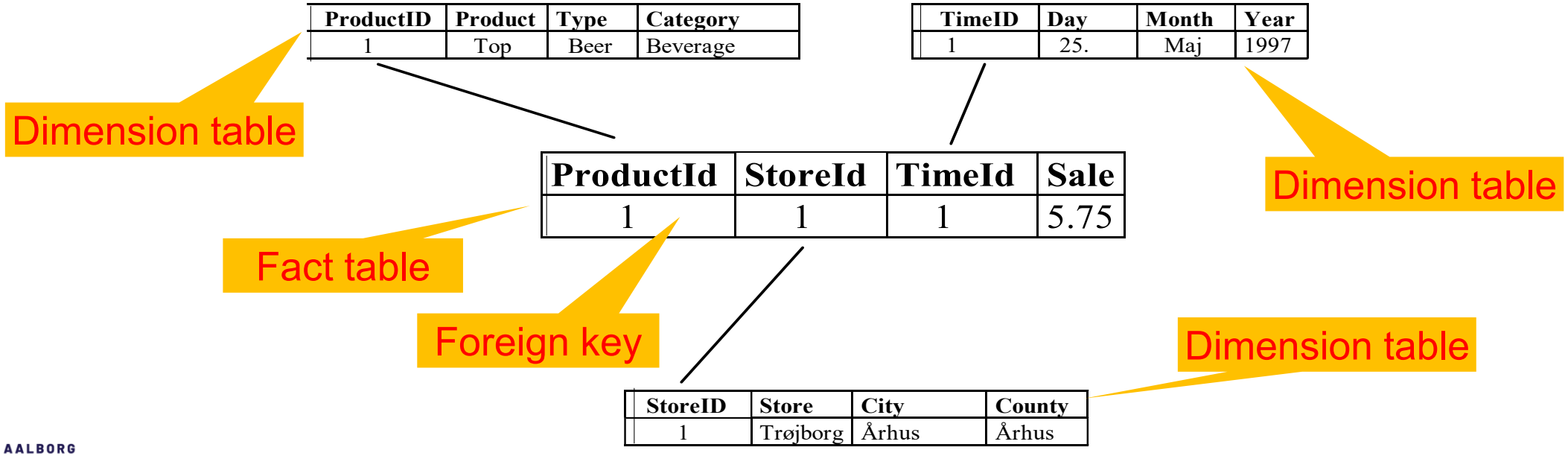
ProductId	StoreId	TimeId	Sale
1	1	1	5.75

- A **dimension table** stores dimension data

- For the key, we don't use information-bearing keys from the source systems.
  - E.g., product dimension, production code: AABC1234
  - E.g., customer dimension, CPR number: 020208-1357
- Use a surrogate key ("meaningless" integer key), which only allows the linking between its dimension table and the fact table

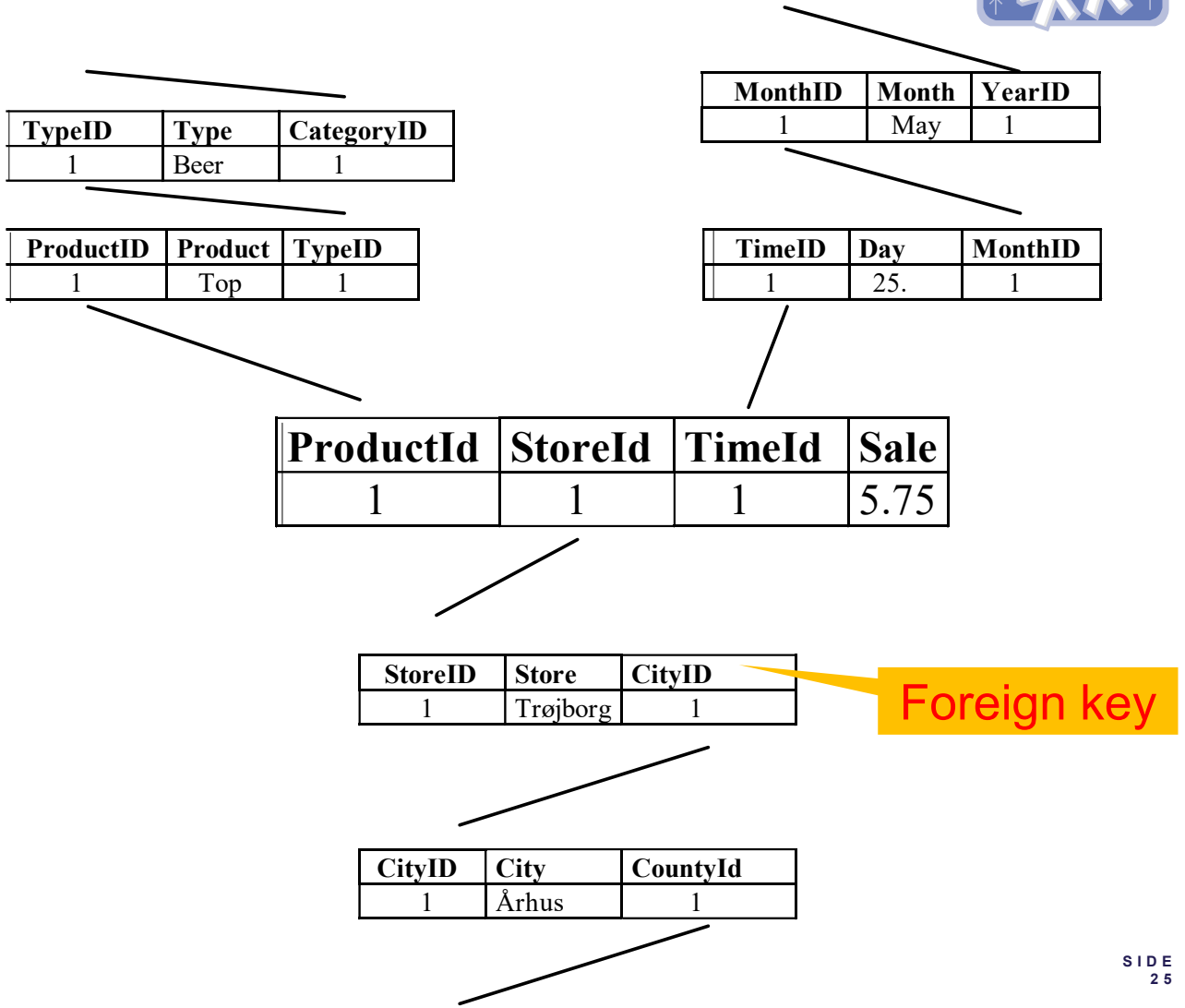
Star Schema

- **Star schema** tables
  - One fact table (for each business process)
  - One *de-normalized* dimension table for each dimension with one column for each level and attribute (except T)



Snowflake Schema

- **Snowflake schema** tables
  - Dimensions are *normalized*
  - One dimension table per level
  - Each dimension table has
    - an integer key
    - a level name
    - one column per attribute
    - a foreign key to the next level



Star vs. Snowflake

- Star Schemas
  - \+ Simple and easy overview → ease-of-use
  - \+ Dimension tables often relatively small
  - \+ “Recognized” by many RDBMSes → good performance
  - \- Hierarchies are ”hidden” in the columns
  - \- Dimension tables are de-normalized

- Snowflake schemas
  - \+ Hierarchies are made explicit/visible
  - \+ Dimension tables use less space
  - \- Harder to use due to many joins
  - \- Worse performance

Redundancy in DW

- Only very little or no redundancy in fact tables
- Redundancy is mostly in dimension tables.
  - Star dimension tables have redundant entries for the higher levels.
- Redundancy problems?
  - Inconsistent data: The central load process helps with this.
  - Update time: The DW is optimized for querying, not updates.
  - Space use: Dimension tables typically take up less than 5% of DW.

- So, **controlled** redundancy is acceptable.

Agenda

- Introduction
- Fundamental Multidimensional Modelling
- Relational Representations
- Querying
- Changing Dimensions

Case Study: Grocery Store

- Products sold from a POS system in Stores with Promotions
  - Point Of Sale
- Task: Analyze how promotions affect sales

DW Design Steps

- Choose the **business process(es)** to model
  - Sales
- Choose the **granularity** of the business process
  - Sales by Product by Store by Promotion by Day
  - Low granularity is needed
- Choose the **dimensions**
  - Time, Store, Promotion, Product
- Choose the **measures**
  - Dollar_sales, unit_sales, dollar_cost, customer_count

The Grocery Store Dimensions

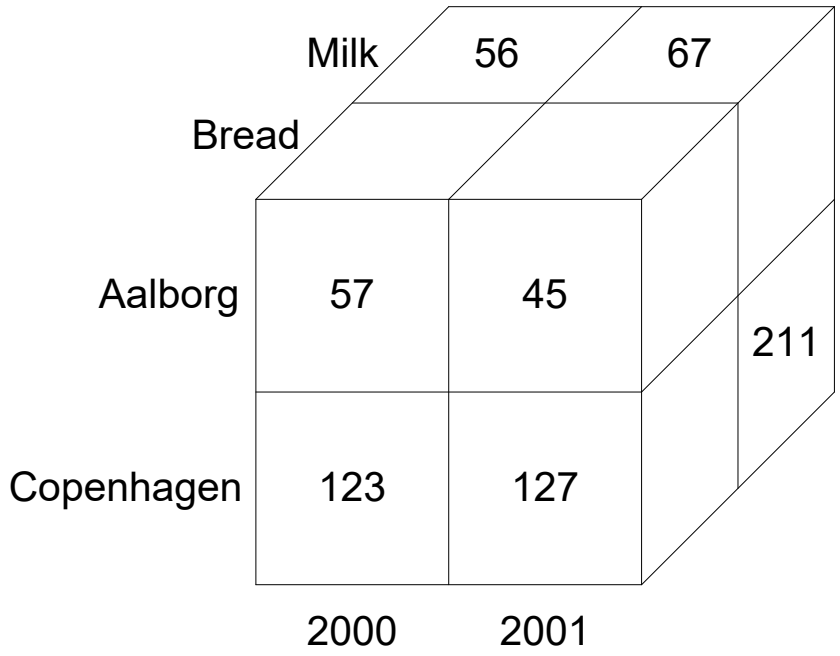
- Time dimension
  - Explicit time dimension is needed (events, holidays,..)
- Product dimension
  - Many-level hierarchy allows drill-down/roll-up
  - **Many** descriptive attributes (often more than 50)
- Store dimension
  - Many descriptive attributes
- Promotion dimension
  - Used to see if promotions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

The Grocery Store Measures

- All **additive measures** across all dimensions
  - Dollar_sales
  - Unit_sales
  - Dollar_cost
- Gross profit (derived)
  - Computed from sales and cost: sales – cost
  - Additive
- Gross margin (derived)
  - Computed from gross profit and sales: (sales – cost)/cost
- Customer_count
  - Additive **measure** across time, promotion, and store

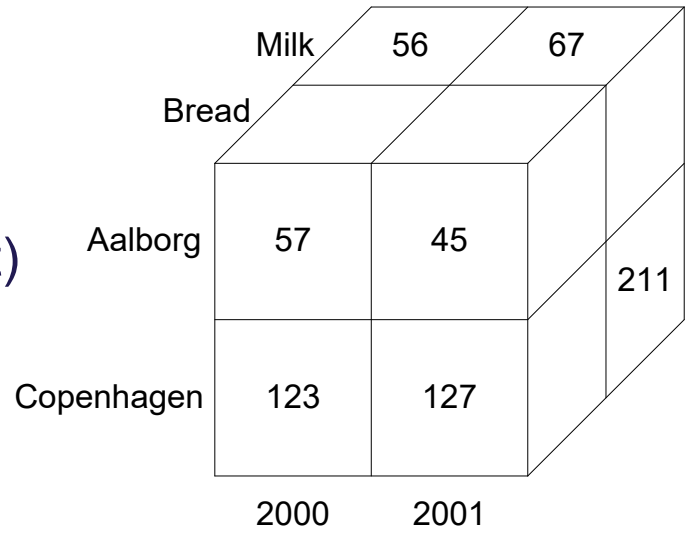
OLAP Queries

- 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

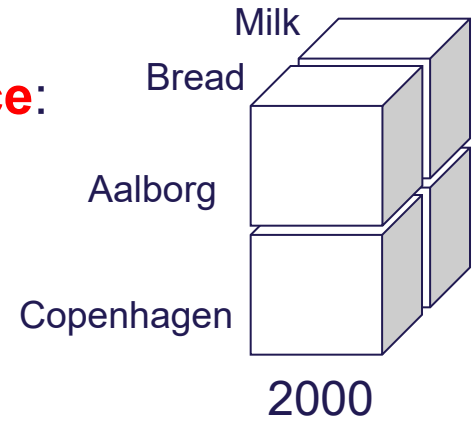


OLAP Queries

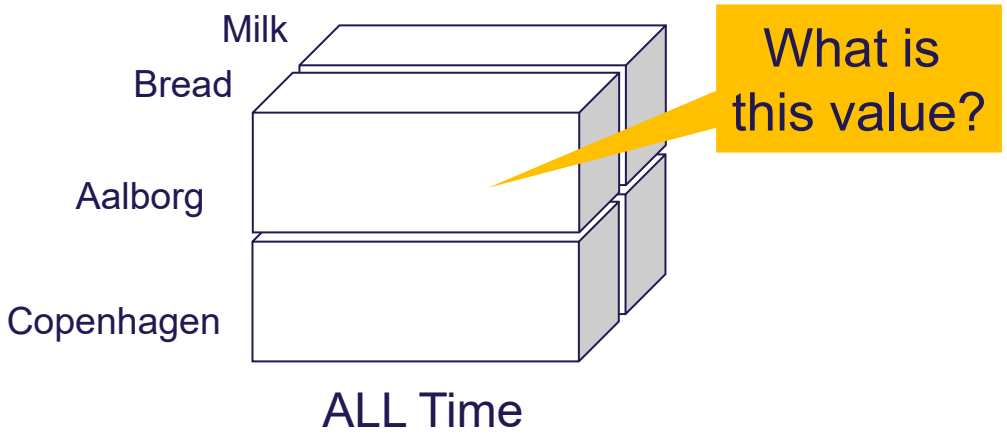
Starting level
(City, Year, Product)



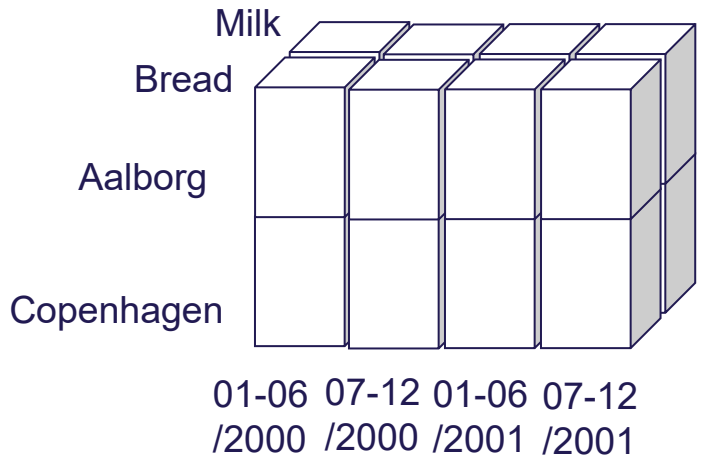
Slice/Dice:



Roll-up: get overview

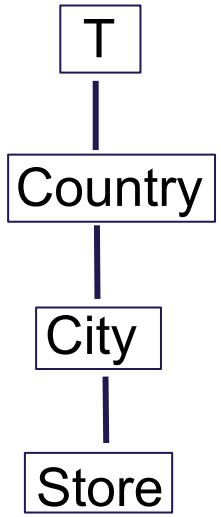


Drill-down: more detail

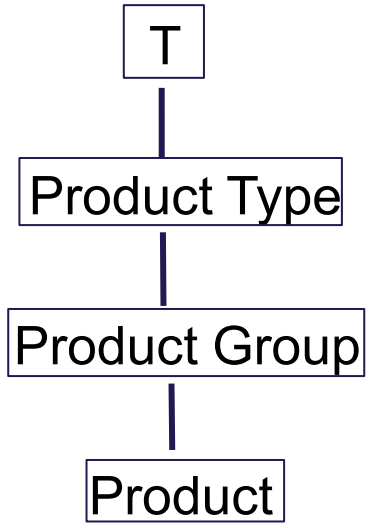


“Drill-down” vs. “Drill-out”

- We “**drill-down**” when we go downwards from one (non-T) level in a hierarchy to another level in the same hierarchy
- We “**drill-out**” when we include a level from another hierarchy in the analysis
- Consider the following hierarchies:



Store Dimension



Product Dimension

Drill-down Example

Country	Sales
Denmark	4200
Sweden	5500

Drill-down from Country level to City level

City	Sales
Aalborg	2000
Copenhagen	2200
Lund	2500
Stockholm	3000

Drill-out Example

Country	Sales
Denmark	4200
Sweden	5500

Drill-out to include the Product Type level

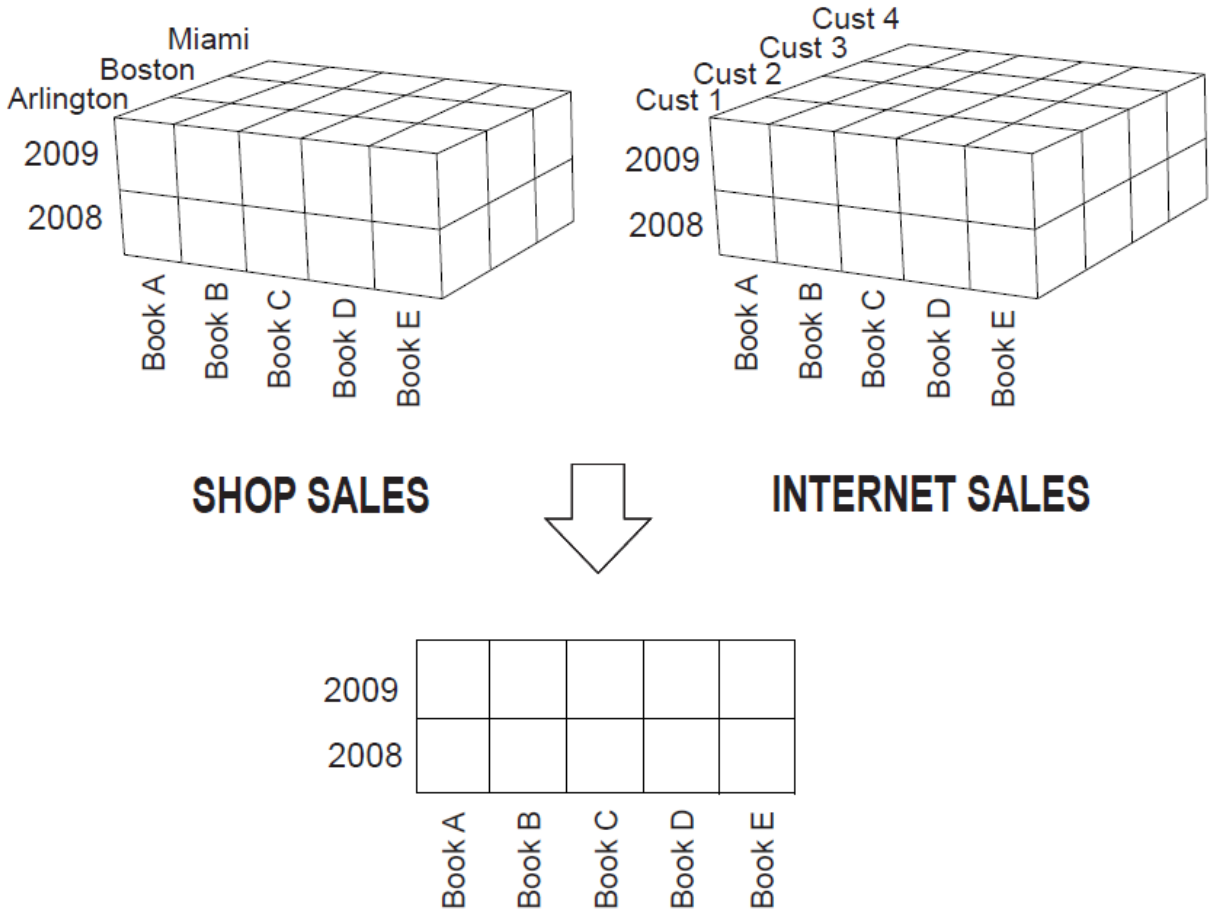
Country	Product Type	Sales
Denmark	Food	3000
Denmark	Non-food	1200
Sweden	Food	4000
Sweden	Non-food	1500

Drill-across

- To **drill-across** means to combine two cubes by means of one or more *shared dimensions*.
  - In relational terms this corresponds to a join
- The resulting cube holds measures from both cubes.
  - For non-shared dimensions, it is like that we roll up to their top level.

Drill-across Example

- A book retailer with two cubes
  - **ShopSales** with dimensions Book, Date, and Shop
  - **InternetSales** with dimensions Book, Date, and Customer
- To find the total sales, the book retailer drills-across and considers the (calculated) measure
  shop_sales + internet_sales



Data Warehouse Software and Tools

- Many out there. How to choose?
  - Open source or not?
  - Local vs. cloud?
  - Performance and scalability
  - Integration
  - 10 Best Data Warehouse Tools to Explore in 2023: https://hevodata.com/learn/data-warehouse-tools/
- Apache Kylin
  - https://kylin.apache.org/
- MS Excel Power Pivot
  - https://www.youtube.com/watch?v=rB_IiYbOo7w

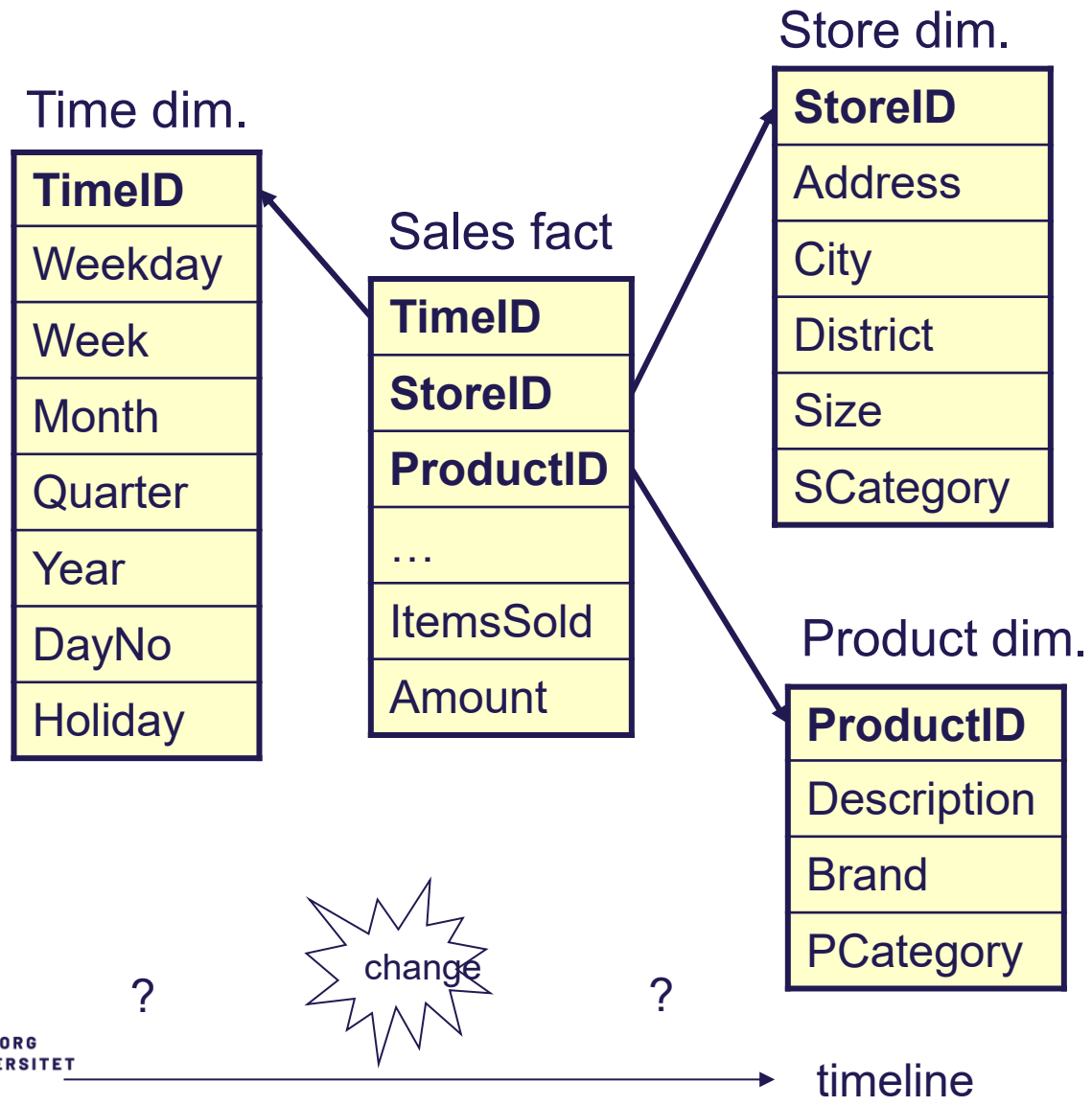
Agenda

- Introduction
- Fundamental Multidimensional Modelling
- Relational Representations
- Querying
- Changing Dimensions
  - NB: Mainly for self-study

Slowly Changing Dimensions

- So far, we've assumed that dimensions are stable over time:
  - New rows in dimension tables can be inserted
  - Existing rows do not change
- We now study techniques for handling changes in dimensions.
- "Slowly changing dimensions" phenomenon
  - Dimension information can change, but changes are not frequent.
  - We (still) assume that the schema is fixed.

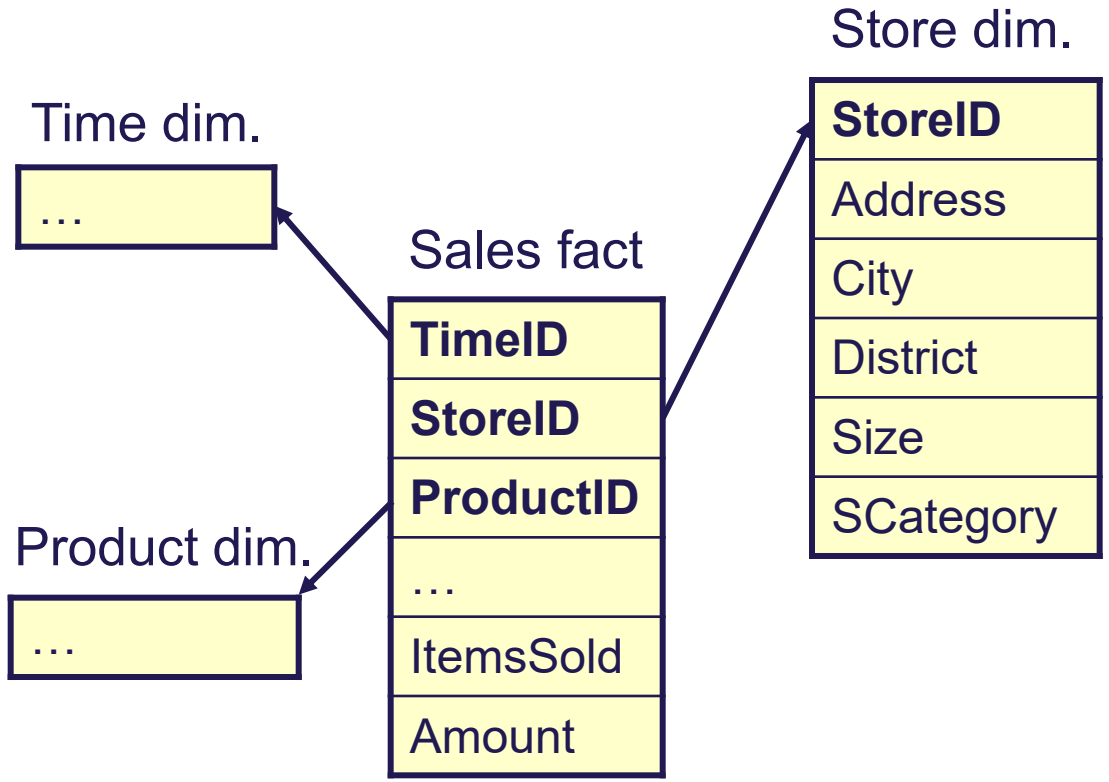
Example



- Attribute values in dimensions vary over time
  - A store changes Size
  - A product changes Description
  - Districts are changed

- Problems
  - Dimensions not updated
    ➔ DW is not up-to-date
  - Dimensions updated in a straightforward way
    ➔ incorrect information in historical data,
    - E.g., wrong analysis on sales/m^2

Example, cont.



- The store in Aalborg has the size of 250 m^2.
- On a certain day, customers bought 2000 apples from that store.

Sales fact table

StoreID	…	ItemsSold	…
001		2000	

Store dimension table

StoreID	…	Size	…
001		250	

Handling Slowly Changing Dimensions

1. No special handling
2. Versioning of dimension values
   - + Use of Timestamping
3. Capturing the previous and the current value
4. Split into changing and constant attributes

Solution 1: No Special Handling

Sales fact table

StoreID	…	ItemsSold	…
001		2000	

Store dimension table

StoreID	…	Size	…
001		250	

The size of a store expands

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	Size	…
001		450	

A new fact arrives

StoreID	…	ItemsSold	…
001		2000	
001		3500	

StoreID	…	Size	…
001		450	

What's the problem here?

Solution 1

- **Solution 1**: Overwrite the old values in the dimension tables
- Consequences
  - Old facts point to rows in the dimension tables with incorrect information!
  - New facts point to rows with correct information

- Pros
  - Easy to implement
  - Useful if the updated attribute is not significant, or the old value should be updated for error correction
- Cons
  - Old facts may point to “incorrect” rows in dimensions

Solution 2: Versioning of Rows

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	Size	…
001		250	

different versions of a store

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	Size	…
001		250	
002		450	

A new fact arrives

StoreID	…	ItemsSold	…
001		2000	
002		3500	

StoreID	…	Size	…
001		250	
002		450	

Which store does the new fact refer to?

Solution 2

- **Solution 2**: Versioning of rows with changing attributes
  - The *key* that links the dimension table and the fact table, identifies a *version* of a dimension member, not just a dimension member
  - Surrogate keys make this easier to implement
    - – what if we had used, e.g., the shop's zip code as key?
    - Always use surrogate keys!!!

- Consequences
  - Larger dimension tables
- Pros
  - Correct information captured in DW
  - No problems when formulating queries
- Cons
  - Cannot capture the development over time of the subjects the dimensions describe in the simplest form (but we can fix that)

Solution 2 with Timestamping

StoreID	TimeID	…	ItemsSold	…
001	234		2000	

StoreID	Size	From	To
001	250	1998	-

attributes: “From”, “To”

StoreID	TimeID	…	ItemsSold	…
001	234		2000	

StoreID	Size	From	To
001	250	1998	1999
002	450	2000	-

StoreID	TimeID	…	ItemsSold	…
001	234		2000	
002	456		3500	

StoreID	Size	From	To
001	250	1998	1999
002	450	2000	-

Solution 2 with Timestamping

- Versioning of rows with changing attributes. Use timestamping of row versions in the SCD with From and To attributes

- Pros
  - Correct information captured in DW
- Cons
  - Larger dimension tables
    - (not really a problem)

Example

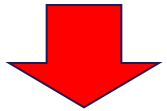
- Product descriptions are versioned, when products are changed, e.g., new package sizes
  - Old versions are still in the stores, new facts can refer to both the newest and older versions of products
  - Time value for a fact not necessarily between “From” and “To” values in the fact’s Product dimension row

- Unlike changes in Size for a store, where all facts from a certain point in time will refer to the newest Size value

Solution 3: Two Versions of Changing Attribute

StoreID	…	ItemsSold	…
001		2000	

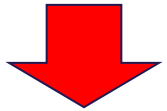
StoreID	…	DistrictOld	DistrictNew	…
001		37	37	



versions of an attribute

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	DistrictOld	DistrictNew	…
001		37	73	



StoreID	…	ItemsSold	…
001		2000	
001		2100	

StoreID	…	DistrictOld	DistrictNew	…
001		37	73	

We cannot find out **when** the district changed.

Solution 3

- **Solution 3**: Create two versions of each changing attribute
  - One attribute contains the current value
  - The other attribute contains the previous value
- Consequences
  - Two values are attached to each dimension row
- Pros
  - Possible to compare across the change in dimension value (which is a problem with Solution 2)
    - Such comparisons are interesting when we need to work simultaneously with two alternative values
    - Example: Categorization of stores and products
- Cons
  - Not possible to see when the old value changed to the new
  - Only possible to capture the two latest values

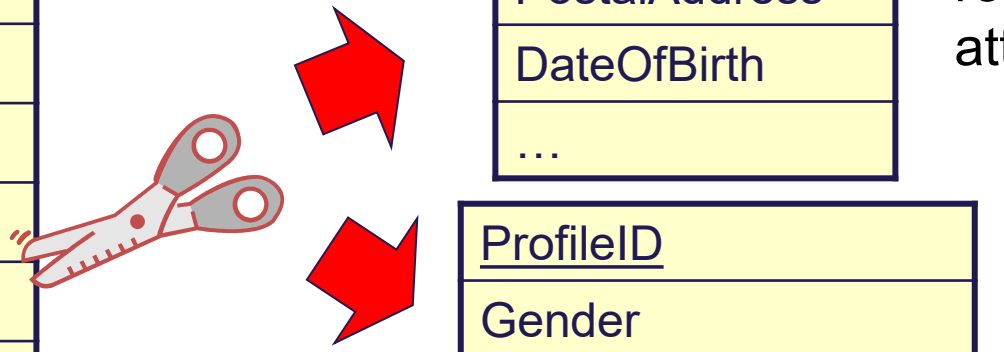
Rapidly Changing Dimensions

- Difference between “slowly” and “rapidly” is subjective
  - Solution 2 is often still feasible
  - The problem is the size of the dimension
- Example
  - Assume an Employee dimension with 100,000 employees, each using 2K bytes and many changes every year
  - Solution 2 is recommended
- Examples of (large) dimensions with many changes: Product and Customer
- The more attributes in a dimension table, the more changes per row are expected
- Example
  - A Customer dimension with 100M customers and many attributes
  - Solution 2 yields a dimension that is too large

Solution 4: Dimension Splitting

Customer dimension (original)

CustomerID
Name
PostalAddress
DateOfBirth
…
Gender
NumberOfKids
MaritalStatus
CreditScore
BuyingStatus
Income
Age
…



CustomerID
Name
PostalAddress
DateOfBirth
…

(New) Customer dimension (SCD): relatively static attributes

ProfileID
Gender
NumberOfKids
MaritalStatus
CreditScoreBand
BuyingStatusBand
IncomeBand
AgeBand
…

Profile dimension (not an SCD): often-changing attributes; describing different profiles;

Solution 4: What we do

- Make a “minidimension” with the often-changing attributes
- Convert (numeric) attributes with many possible values into attributes with few discrete or banded values
  - E.g., Income group: [0,10K), [10,20K), [20,30K), [30,40K)
  - Why? Any Information Loss?
- Insert rows for all combinations of values from these new domains
  - What do we do, if there are too many (theoretical) combinations?
- If the minidimension is too large, it can be further split into more minidimensions
  - Synchronous/correlated attributes should be in the same minidimension.

Solution 4: Pros and Cons

- Pros
  - DW size (dimension tables) is kept down
  - Changes in a customer's profile values do not result in changes in dimensions
- Cons
  - More dimensions and more keys in the star schema
  - Navigation of customer attributes is more cumbersome as these are in more than one dimension
  - Using value groups gives less detail
  - The construction of groups is irreversible

Summary

- Multidimensional data modelling
  - Cubes, Dimensions, Facts, Measures
- Relational Implementation
  - Star schema
  - Snowflake schema
- OLAP Queries
  - Slice/Dice, Roll-up
  - Drill-down, Drill-out, Drill-across
- Changing Dimensions
  - Why are there changes in dimensions?
  - Star schemas support change over time to a large extent
  - Different techniques to handle changes over time at the instance level exist
    - Solution 2 is the most useful

Readings

- Mandatory readings
  - Christian S. Jensen, Torben Bach Pedersen, and Christian Thomsen: Multidimensional Databases and Data Warehousing, Morgan & Claypool 2010.
    - Chapters 1, 2 and 3 (only 3.1 and 3.2.1)

- Acknowledgements
  - A large portion of the slides are (adapted) from Torben Bach Pedersen

Exercises

- JPT book Section 2.10
  - 2, 3, 4, 5, 7, 8, 9
- JPT book Section 3.5
  - 4, 5