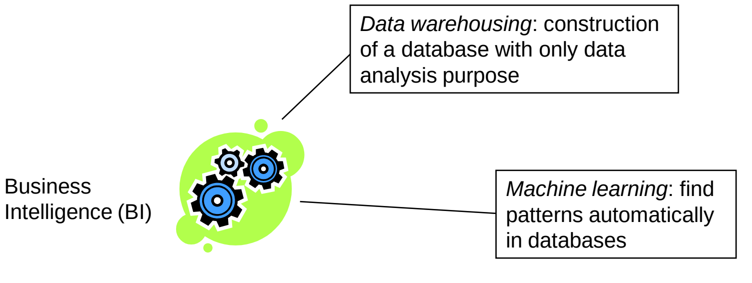
Introduction to Data Warehousing and Business Intelligence

Slides kindly borrowed from the course
"Data Warehousing and Machine Learning"
Aalborg University, Denmark

Christian S. Jensen
Torben Bach Pedersen
Christian Thomsen
{csj,tbp,chr}@cs.aau.dk

Course Structure

- Business intelligence
  - **Extract** knowledge from **large amounts** of data collected in a modern enterprise
  - Data warehousing, machine learning
- Purpose
  - Acquire theoretical background in lectures and literature studies
  - Obtain practical experience on (industrial) tools in practical exercises



2

Literature

- **Multidimensional Databases and Data Warehousing,** Christian S. Jensen, Torben Bach Pedersen, Christian Thomsen, Morgan & Claypool Publishers, 2010
- **Data Warehouse Design: Modern Principles and Methodologies**, Golfarelli and Rizzi, McGraw-Hill, 2009
- **Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications,** Elzbieta Malinowski, Esteban Zimányi, Springer, 2008
- **The Data Warehouse Lifecycle Toolkit**, Kimball et al., Wiley 1998
- **The Data Warehouse Toolkit**, 2nd Ed., Kimball and Ross, Wiley, 2002

Overview

- Why Business Intelligence?
- Data analysis problems
- Data Warehouse (DW) introduction
- DW topics
  - Multidimensional modeling
  - ETL
  - Performance optimization

What is Business Intelligence (BI)?

- From *Encyclopedia of Database Systems*:

  "[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."

5

What is Business Intelligence (BI)?

- BI is different from Artificial Intelligence (AI)
  - AI systems **make** decisions **for** the users
  - BI systems **help** the users make the **right** decisions, based on available data
- Combination of technologies
  - Data Warehousing (DW)
  - On-Line Analytical Processing (OLAP)
  - Data Mining (DM)
  - ……

6

Why is BI Important?

- Worldwide BI revenue in 2005 = US$ 5.7 billion
  - 10% growth each year
  - A market where players like IBM, Microsoft, Oracle, and SAP compete and invest
- BI is not only for large enterprises
  - Small and medium-sized companies can also benefit from BI
- The financial crisis has increased the focus on BI
  - You cannot afford *not* to use the “gold” in your data

BI and the Web

- The Web makes BI even more useful
  - Customers do not appear “physically” in a store; their behaviors cannot be observed by traditional methods
  - A website log is used to capture the behavior of each customer, e.g., sequence of pages seen by a customer, the products viewed
  - Idea: understand your customers using data and BI!
    - Utilize website logs, analyze customer behavior in more detail than before (e.g., what was **not** bought?)
    - Combine web data with traditional customer data

Case Study of an Enterprise

- Example of a chain (e.g., fashion stores or car dealers)
  - Each store maintains its own customer records and sales records
    - Hard to answer questions like: “find the total sales of Product X from stores in Aalborg”
  - The same customer may be viewed as different customers for different stores; hard to detect duplicate customer information
  - Imprecise or missing data in the addresses of some customers
  - Purchase records maintained in the operational system for limited time (e.g., 6 months); then they are deleted or archived
  - The same “product” may have different prices, or different discounts in different stores

- Can you see the problems of using those data for business analysis?

Data Analysis Problems

- The same data found in many different systems
  - Example: customer data across different stores and departments
  - The same concept is defined differently

- Heterogeneous sources
  - Relational DBMS, On-Line Transaction Processing (OLTP)
  - Unstructured data in files (e.g., MS Word)
  - Legacy systems
  - …

Data Analysis Problems (cont')

- Data is suited for operational systems
  - Accounting, billing, etc.
  - Do not support analysis across business functions
- Data quality is bad
  - Missing data, imprecise data, different use of systems
- Data are "volatile"
  - Data deleted in operational systems (6 months)
  - Data change over time – no historical information

Data Warehousing

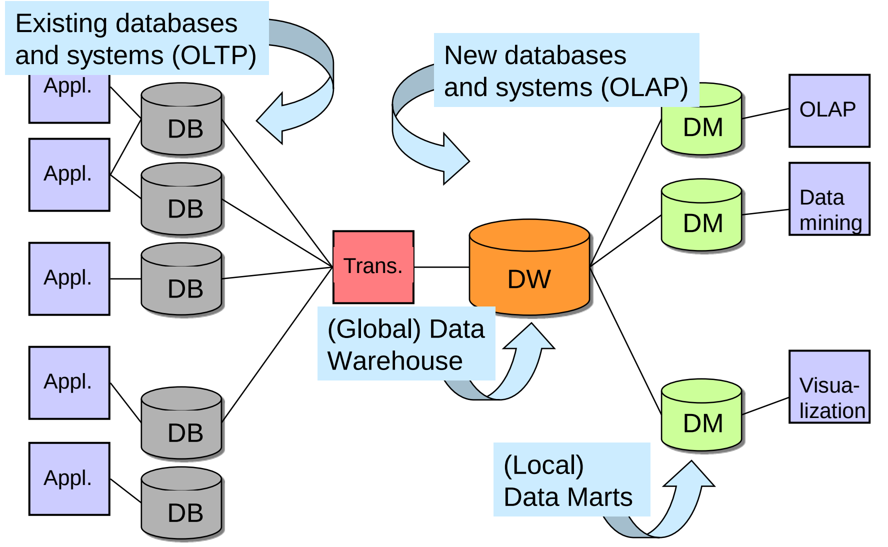
- Solution: new analysis environment (DW) where data are
  - Subject oriented (versus function oriented)
  - Integrated (logically and physically)
  - Time variant (data can always be related to time)
  - Stable (data not deleted, several versions)
  - Supporting management decisions (different organization)
- Data from the operational systems are
  - Extracted
  - Cleansed
  - Transformed
  - Aggregated (?)
  - Loaded into the DW
- A good DW is a **prerequisite** for successful BI

DW: Purpose and Definition

- DW is a **store of information** organized in a unified data model
- Data collected from a number of different sources
  - Finance, billing, website logs, personnel, …
- Purpose of a data warehouse (DW): support **decision making**
- Easy to perform advanced analysis
  - Ad-hoc analysis and reports
    - We will cover this soon ……
  - Data mining: discovery of hidden patterns and trends
    - You will study this in another course

13

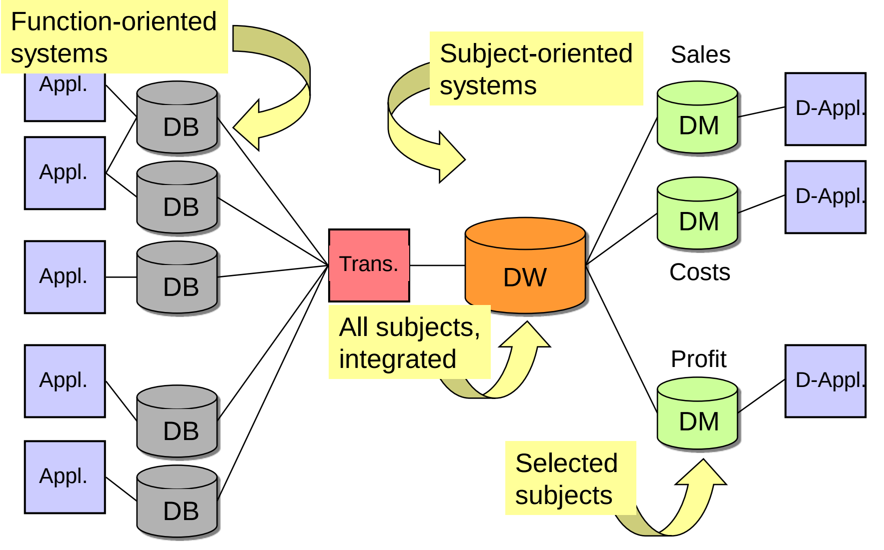
DW Architecture – Data as Materialized Views



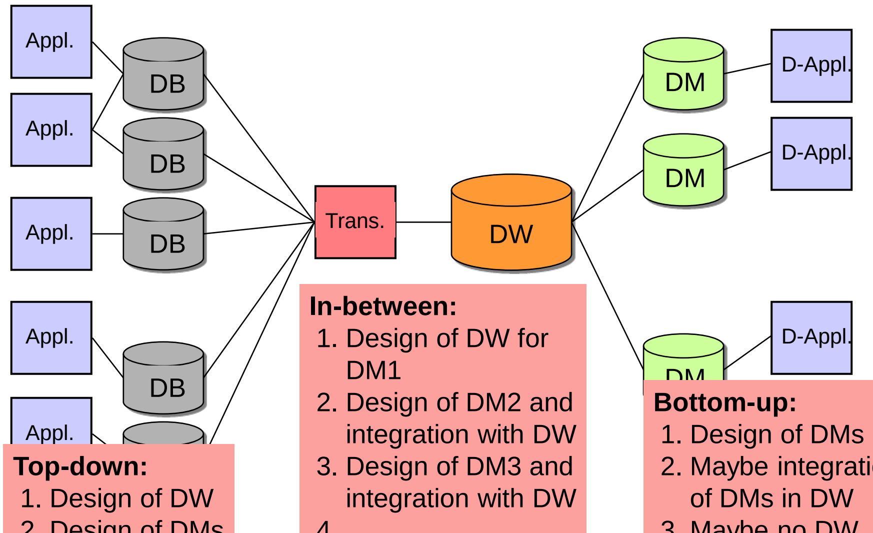
Analogy: (data) producers ↔ warehouse ↔ (data) consumers

14

Function vs. Subject Orientation



Top-down vs. Bottom-up



Top-down:
1. Design of DW
2. Design of DMs

In-between:
1. Design of DW for DM1
2. Design of DM2 and integration with DW
3. Design of DM3 and integration with DW
4. ...

Bottom-up:
1. Design of DMs
2. Maybe integration of DMs in DW
3. Maybe no DW

Hard/Infeasible Queries for OLTP

- Why not use the existing databases (OLTP) for business analysis?
- Business analysis queries
  - In the **past five years**, which product is the most profitable?
  - Which **public holiday** we have the largest sales?
  - Which **week** we have the largest sales?
  - Does the sales of **dairy products** increase over time?
- Difficult to express these queries in SQL
  - 3rd query: may extract the "week" value using a function
    - But the user has to learn many transformation functions …
  - 4th query: use a "special" table to store IDs of all dairy products, in advance
    - There can be many different dairy products; there can be many other product types as well …
- The need of multidimensional modeling

17

Multidimensional Modeling

- Example: sales of 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*

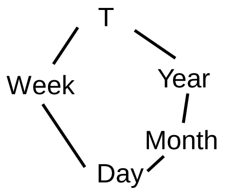
Product	**Type**	**Category**	**Store**	**City**	**County**	**Date**	**Sales**
Top	Beer	Beverage	Trøjborg	Århus	Århus	25 May, 2009	5.75

Product Store Time

18

Multidimensional Modeling

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



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

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

19

Quick Review: Normalized Database

Product ID	Type	Category	Price
001	Beer	Beverage	6.00
002	Rice	Cereal	4.00
003	Beer	Beverage	7.00
004	Wheat	Cereal	5.00

Product ID	TypeID	Price
001	013	6.00
002	052	4.00
003	013	7.00
004	067	5.00

TypeID	Type	CategoryID
013	Beer	042
052	Rice	099
067	Wheat	099

CategoryID	Category
042	Beverage
099	Cereal

- Normalized database avoids
  - Redundant data
  - Modification anomalies
- How to get the original table? (join them)
- No redundancy in OLTP, controlled redundancy in OLAP

20

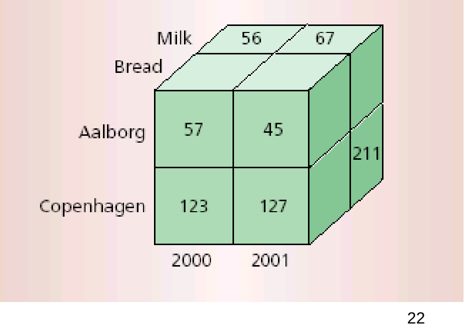
OLTP vs. OLAP

	OLTP	OLAP
Target	operational needs	business analysis
Data	small, operational data	large, historical data
Model	normalized	denormalized/ multidimensional
Query language	SQL	not unified – but MDX is used by many
Queries	small	large
Updates	frequent and small	infrequent and batch
Transactional recovery	necessary	not necessary
Optimized for	update operations	query operations

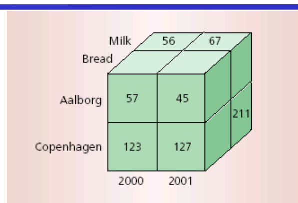
OLAP Data Cube

- Data cube
  - Useful data analysis tool in DW
  - Generalized GROUP BY queries
  - Aggregate facts based on chosen dimensions
    - Product, store, time dimensions
    - Sales measure of sale facts
- Why data cube?
  - Good for visualization (i.e., text results hard to understand)
  - Multidimensional, intuitive
  - Support interactive OLAP operations
- How is it different from a spreadsheet?

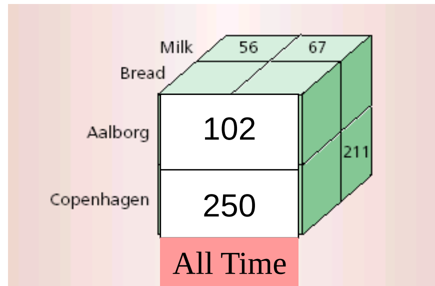
Store	Product	Time	Sales
Aalborg	Bread	2000	57
Aalborg	Milk	2000	56
Copenhagen	Bread	2000	123
…	…	…	…

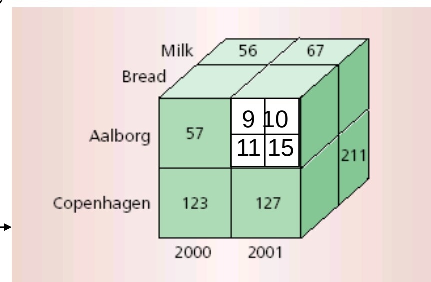


On-Line Analytical Processing (OLAP)



- On-Line Analytical Processing
  - Interactive analysis
  - Explorative discovery
  - Fast response times required
- OLAP operations/queries
  - Aggregation, e.g., SUM
  - Starting level, (Year, City)
    - Roll Up: Less detail
    - Drill Down: More detail
  - Slice/Dice: Selection, Year=2000





Advanced Multidimensional Modeling

- Changing dimensions
  - Some dimensions are not static. They change over time.
    - A store moves to a new location with more space
    - The name of a product changes
    - A customer moves from Aalborg Øst to Hasseris
  - How do we handle these changes?
- Large-scale dimensional modeling
  - How do we coordinate the dimensions in different data cubes and data marts?

Bus architecture

Dimensions

Data marts	Time	Customer	Product	Supplier
Sales	+	+	+	
Costs			+	+
Profit	+	+	+	+

Extract, Transform, Load (ETL)

- "Getting multidimensional data into the DW"
- Problems
  1. Data from different sources
  2. Data with different formats
  3. Handling of missing data and erroneous data
  4. Query performance of DW
- ETL
  - Extract (for problem #1)
  - Transformations / cleansing (for problems #2, #3)
  - Load (for problem #4)
- The most time-consuming process in DW development
  - 80% of development time spent on ETL

Performance Optimization

Sales

tid	pid	locid	sales
1	1	1	10
2	1	1	20
3	2	3	40
…	…	…	…

1 billion rows

- The data warehouse contains GBytes or even TBytes of data!
- OLAP users require fast query response time
  - They don't want to wait for the result for 1 hour!
  - Acceptable: answer within 10 seconds
- Idea: precompute some partial result in advance and store it
  - At query time, such partial result can be utilized to derive the final result very fast

Materialization Example

- Imagine 1 billion sales rows, 1000 products, 100 locations
- CREATE VIEW TotalSales (pid, locid, total) AS
  SELECT s.pid, s.locid, SUM(s.sales)
  FROM Sales s
  GROUP BY s.pid, s.locid
- The materialized view has 100,000 rows

Sales

tid	pid	locid	sales
1	1	1	10
2	1	1	20
3	2	3	40
…	…	…	…

1 billion rows

- Wish to answer the query:
  - SELECT p.category, SUM(s.sales)
    FROM Products p, Sales s WHERE p.pid=s.pid
    GROUP BY p.category
- Rewrite the query to use the view:
  - SELECT p.category, SUM(t.total)
    FROM Products p, **TotalSales t**
    WHERE p.pid=t.pid GROUP BY p.category
  - Query becomes 10,000 times faster!

VIEW TotalSales

pid	locid	sales
1	1	30
2	3	40
…	…	…

100,000 rows

27

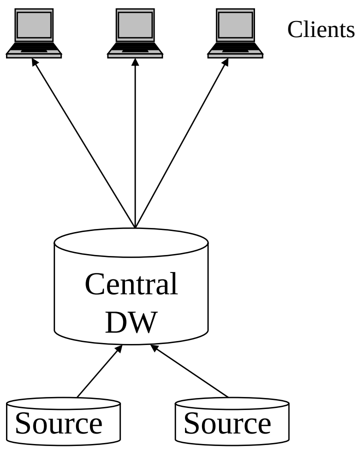
Data Warehouse Architecture

- Central
- Federated
- Tiered

28

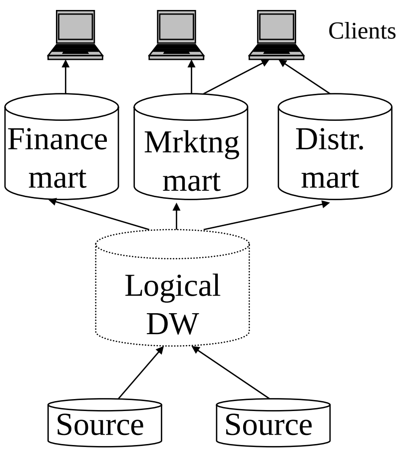
Central DW Architecture

- All data in one, central DW
- All client queries directly on the central DW
- Pros
  - Simplicity
  - Easy to manage
- Cons
  - Bad performance due to no redundancy/workload distribution



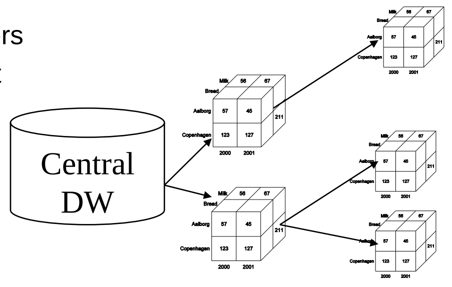
Federated DW Architecture

- Data stored in separate data marts, aimed at special departments
- Logical DW (i.e., virtual)
- Data marts contain detail data
- Pros
  - Performance due to distribution
- Cons
  - More complex



Tiered Architecture

- Central DW is materialized
- Data is distributed to data marts in one or more *tiers*
- Only aggregated data in cube tiers
- Data is aggregated/reduced as it moves through tiers
- Pros
  - Best performance due to redundancy and distribution
- Cons
  - Most complex
  - Hard to manage



Common DW Issues

- Metadata management
  - Need to **understand** data = metadata needed
  - Greater need in OLAP than in OLTP as “raw” data is used
  - Need to know about:
    - Data definitions, dataflow, transformations, versions, usage, security
- DW project management
  - DW projects are **large** and **different** from ordinary SW projects
    - 12-36 months and US$ 1+ million per project
    - Data marts are smaller and “safer” (bottom up approach)
  - Reasons for failure
    - Lack of proper design methodologies
    - High HW+SW cost
    - Deployment problems (lack of training)
    - Organizational change is hard… (new processes, data ownership,..)
    - Ethical issues (security, privacy,…)

Topics not Covered in the Course

- Privacy/security of data during ETL
  - Encryption may not work
  - During extraction/transformation, someone may need to know original values in order to check whether ETL performs correctly
- Data Visualization (VIS)
- Decision Analysis (What-if)
- Customer Relationship Management (CRM)

Summary

- Why Business Intelligence?
- Data analysis problems
- Data Warehouse (DW) introduction
- DW Topics
  - Multidimensional modeling
  - ETL
  - Performance optimization
- BI provide many advantages to your organization
  - A good DW is a prerequisite for BI

Multidimensional Databases

Overview

- Cubes: Dimensions, Facts, Measures
- OLAP Queries
- Relational Implementation
- Redundancy

ER Model vs. Multidimensional Model

- Why don't we use the ER model in data warehousing?

- ER model: a data model for **general** purposes
  - All types of data are "equal", difficult to identify the data that is important for business analysis
    - No difference between:
      - What **is** important
      - What just **describes** the important
    - Normalized databases **spread** information
    - When analyzing data, the information must be **integrated** again
  - Hard to overview a **large** ER diagram (e.g., over 100 entities/relations for an enterprise)

3

ER Model vs. Multidimensional Model

- The multidimensional model
  - Its only purpose: **data analysis**
    - It is not suitable for OLTP systems
  - More **built in** "meaning"
    - What **is** important
    - What **describes** the important
    - What we want to **optimize**
    - Easy for query operations
- Recognized by OLAP/BI tools
  - Tools offer powerful query facilities based on MD design
  - Example: TARGIT Analysis

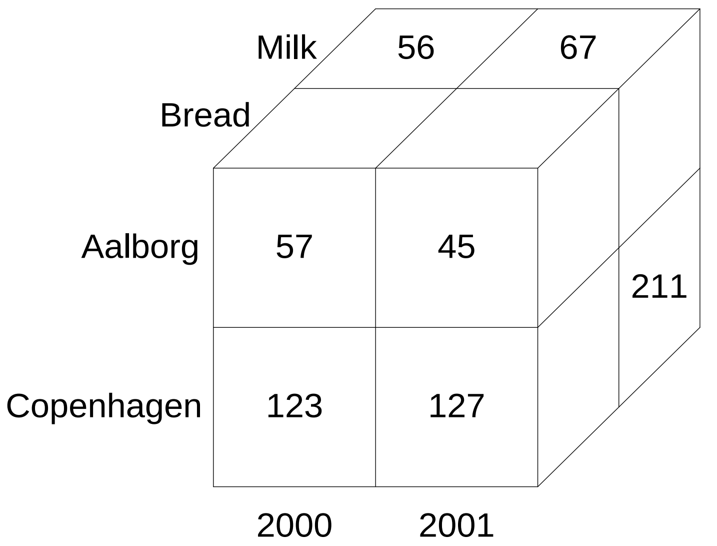
4

The multidimensional model

- Data is divided into:
  - **Facts**
  - **Dimensions**
- Facts are the **important** entity: a sale
- Facts have **measures** that can be aggregated: sales price
- Dimensions **describe** facts
  - A sale has the dimensions Product, Store and Time
- Facts “live” in a multidimensional **cube** (dice)
  - Think of an array from programming languages
- Goal for dimensional modeling:
  - Surround facts with as much context (dimensions) as possible
  - Hint: redundancy may be ok (in well-chosen places)
  - But you should **not** try to model **all** relationships in the data (unlike E/R and OO modeling!)

5

Cube Example



6

Cubes

- A "cube" may have **many** dimensions!
  - 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 (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

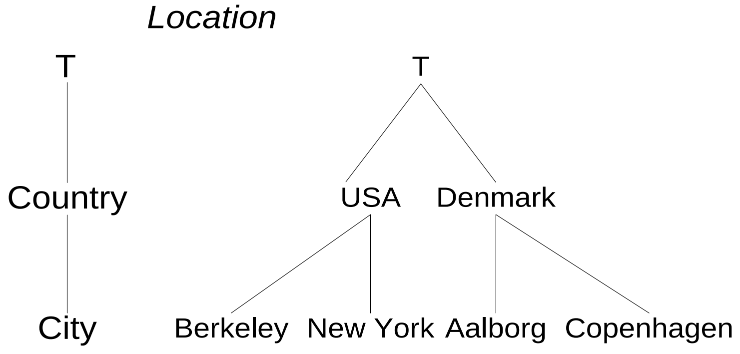
Dimensions

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

- Dimensions have **hierarchies** with **levels**
  - Typically 3-5 levels (of detail)
  - Dimension values are organized in a **tree structure**
  - **Product**: Product->Type->Category
  - **Store**: Store->Area->City->County
  - **Time**: Day->Month->Quarter->Year
  - Dimensions have a **bottom level** and a **top level** (ALL)
- Levels may have **attributes**
  - Simple, non-hierarchical information
  - 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



Schema Instance

Facts

- Facts represent the **subject** of the desired analysis
  - The ”important” in the business that should be analyzed
- A fact is identified via its dimension values
  - A fact is a non-empty cell
- Generally, a fact should
  - Be attached to **exactly one** dimension value in each dimension
  - Only be attached to dimension values in the bottom levels
  - Some models do not require this

Types of Facts

- **Event** fact (transaction)
  - A fact for every **business event** (sale)
- ”**Fact-less**” facts
  - A fact per event (customer contact)
  - **No** numerical measures
  - An event has happened for a given dimension value combination
- **Snapshot** fact
  - A fact for every dimension combination at given time intervals
  - Captures **current** status (inventory)
- **Cumulative snapshot** facts
  - A fact for every dimension combination at given time intervals
  - Captures **cumulative** status up to now (sales in year to date)
- Every type of facts answers **different** questions
  - Often both event facts and both kinds of snapshot facts exist

Granularity

- **Granularity** of facts is important
  - What does a single fact mean?
  - **Level of detail**
  - Given by combination of bottom levels
  - Example: ”total sales per store per day per product”
- Important for number of facts
  - Scalability
- 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
  - Except perhaps huge clickstreams etc.

13

Measures

- Measures represent the fact property that the users want to **study and optimize**
  - Example: total sales price
- A measure has two components
  - **Numerical value**: (sales price)
  - **Aggregation formula** (SUM): used for aggregating/combining a number of measure values into one
  - Measure value determined by dimension value combination
  - Measure value is meaningful for all aggregation levels

14

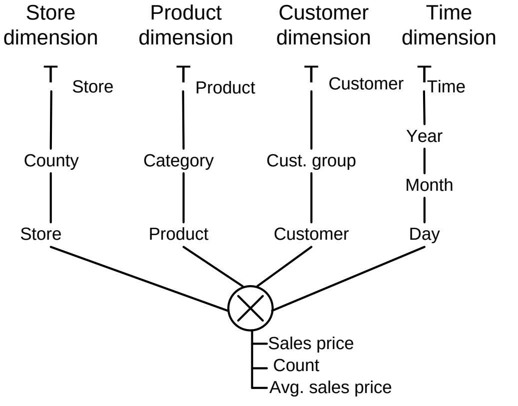
Types of Measures

- Three types of measures

- Additive
  - Can be aggregated over **all** dimensions
  - Example: **sales price**
  - Often occur in event facts
- Semi-additive
  - **Cannot** be aggregated over **some** dimensions - typically time
  - Example: **inventory**
  - Often occur in snapshot facts
- Non-additive
  - **Cannot** be aggregated over **any** dimensions
  - Example: **average sales price**
  - Occur in all types of facts

15

Schema Documentation



- No well-defined standard
- Our own notation
  - 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

16

Why the schema cannot answer question X

- Possible reasons
  - Certain **measures** not included in fact table
  - **Granularity** of facts too coarse
  - Particular **dimensions** not in DW
  - Descriptive **attributes** missing from dimensions
  - **Meaning** of attributes/measures deviate from the expectation of data analysts (users)
  - ……

ROLAP

- Relational OLAP
- 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
…	…	…

MOLAP

- Multidimensional OLAP
- Data stored in special multidimensional data structures
  - E.g., multidimensional array on hard disk
- Pros
  - Less storage use (“foreign keys” not stored)
  - Faster query response times
- 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

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

HOLAP

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

Relational Implementation

- 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?
  - Important for the size of the fact table
  - Often corresponding to a single business transaction (sale)
  - But it can be aggregated (sales per product per day per store)
- Some properties
  - Many-to-one relationship from fact to dimension
  - Many-to-one relationships from lower to higher levels in the hierarchies

Relational Design

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

Product Store Time

- One completely de-normalized table
  - Bad: inflexibility, storage use, bad performance, slow update
- Star schemas
- Snowflake schemas

Star Schema Example

- Star schemas
  - One fact table
  - De-normalized dimension tables
  - One column per level/attribute

ProductID	Product	Type	Category
1	Top	Beer	Beverage

TimeID	Day	Month	Year
1	25.	Maj	1997

ProductId	StoreId	TimeId	Sale
1	1	1	5.75

StoreID	Store	City	County
1	Trøjborg	Århus	Århus

Relational Implementation

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

- What are the disadvantages of using production codes as the key?
  - E.g., product dimension, production code: AABC1234
  - E.g., customer dimension, CPR number: 020208-1357
- Use surrogate key (“meaningless” integer key), which only allows the linking between its dimension table and the fact table

For Extract-Transform-Load, we need to keep a mapping from production key to surrogate key (more about this in lecture #4)

Snowflake Schema Example

TypeID	Type	CategoryID
1	Beer	1

ProductID	Product	TypeID
1	Top	1

MonthID	Month	YearID
1	May	1

TimeID	Day	MonthID
1	25.	1

ProductId	StoreId	TimeId	Sale
1	1	1	5.75

StoreID	Store	CityID
1	Trøjborg	1

CityID	City	CountyId
1	Århus	1

- Snowflake schemas
  - Dimensions are normalized
  - One dimension table per level
  - Each dimension table has integer key, level name, and one column per attribute

Question Time

- Suppose that we want to replace the original Store hierarchy **A** by a new hierarchy **B**
- How do we modify the star schema to reflect this?
- How do we modify the snowflake schema to reflect this?

T
|
County
|
City
|
Store

Store Schema A

T
|
Country
|
County
|
City
|
Store

Store Schema B

Star vs Snowflake

- Star Schemas
  - \+ Simple and easy overview → ease-of-use
  - \+ Relatively flexible
  - \+ 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
  - \+ Very flexible
  - \+ Dimension tables use less space
  - \- Harder to use due to many joins
  - \- Worse performance

27

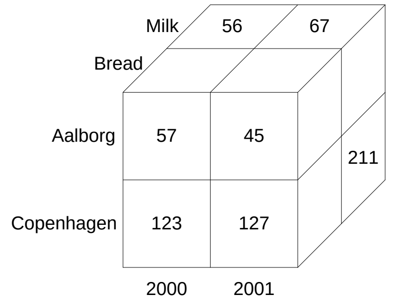
Redundancy in the DW

- Only very little or no redundancy in fact tables
  - The same fact data only stored in one fact table
- 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 good
  - Up to a certain limit

28

(Relational) OLAP Queries

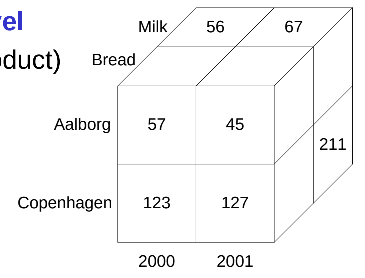
- **Two** kinds of queries
  - **Navigation queries** examine one dimension
    - SELECT DISTINCT l FROM d [WHERE p]
  - **Aggregation queries** summarize fact data
    - SELECT d1.l1, d2.l2, SUM(f.m) FROM d1, d2, f
      WHERE f.dk1 = d1.dk1 AND f.dk2 = d2.dk2 [AND p]
      GROUP BY d1.l1,d2.l2
- Fast,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



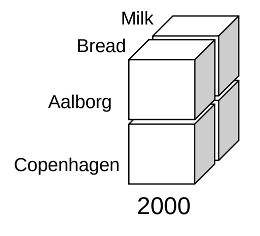
OLAP Queries

Starting level

(City, Year, Product)

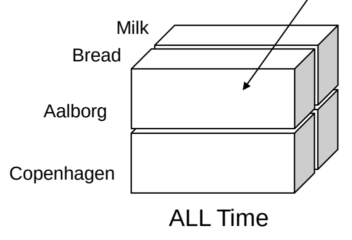


Slice/Dice:

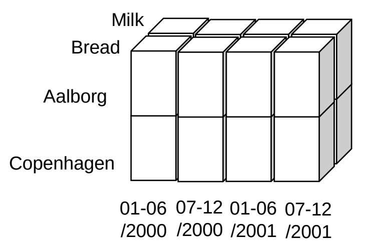


Roll-up: get overview

What is this value?



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	9861.25	23260.25	84286	164020.75
2002	30086.5	34731	0	15506.5	41619.5	74847	196790.5
2003	28740	28596	0	14213.5	45046	63580	180175.5
2004	24126.75	28292	0	9592	82226	54526.5	198763.25
2005	22695.5	20449	0	7803.25	75835	52044	178826.75
2006	25196	19958	0	6910.5	102746	47456	202266.5
2007	876	641	0	155.75	2094.5	1387.5	5154.75
Grand Total	205654.25	225447.5	8475	102580.75	408621.25	647096	1597874.75

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

31

Case Study: Grocery Store

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

- Task: Analyze how promotions affect sales

32

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
  - Are individual transactions necessary/feasible?
- Choose the **dimensions**
  - Time, Store, Promotion, Product
- Choose the **measures**
  - Dollar_sales, unit_sales, dollar_cost, customer_count
- Resisting normalization and preserving browsing
  - Flat dimension tables makes browsing easy and fast

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
  - Example of a **causal** dimension
  - Used to see if promotions work/are profitable
  - Ads, price reductions, end-of-aisle displays, coupons

The Grocery Store Measures

- All **additive** 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
  - **Non-additive** across all dimensions
- Customer_count
  - Additive across time, promotion, and store
  - **Non-additive** across product. Why?
  - **Semi-additive**

Data Warehouse Size

- Estimated number of fact records:
  - Time dimension: 2 years = 730 days
  - Store dimension: 300 stores reporting each day
  - Product dimension: 30,000 products, only 3000 sell per day
  - Promotion dimension: 5000 combinations, but a product only appears in one combination per day
  - 730*300*3000*1 = 657,000,000
- Total data warehouse size: 657,000,000 facts* 8 fields/fact * 4 bytes/field = 21 GB
  - Number of fields: 4 FKs + 4 measures = 8 fields
  - Assuming sizes of dimensions negligible
- Small size (by today's standard), feasible to store at transaction level detail

Summary

- Cubes: Dimensions, Facts, Measures
- OLAP Queries
- Relational Implementation
  - Star schema vs Snowflake schema
- Redundancy

Advanced MD Modeling and MD Database Implementation

Overview

- Handling Changes in Dimensions
- Coordinating Data Cubes / Data Marts
- Multidimensional Database Implementation

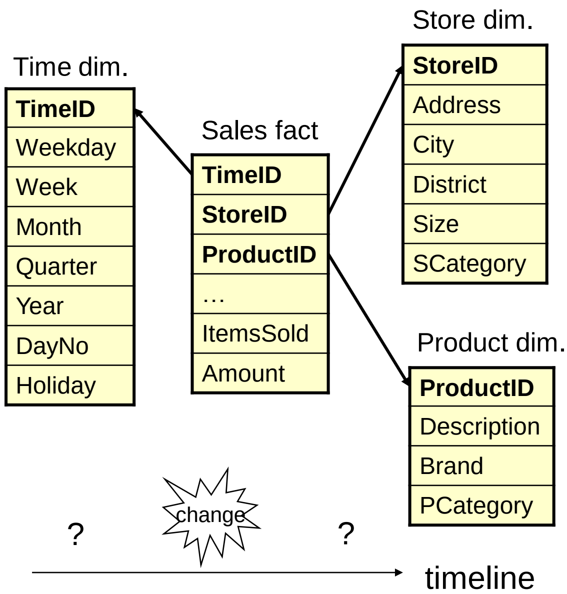
Changing Dimensions

- In the last lecture, we assumed that dimensions are stable over time
  - New rows in dimension tables can be inserted
  - Existing rows do not change
    - This is not a realistic assumption
- We now study techniques for handling changes in dimensions
- "Slowly changing dimensions" phenomenon
  - Dimension information change, but changes are not frequent
  - Still assume that the schema is fixed

Handling Changes in Dimensions

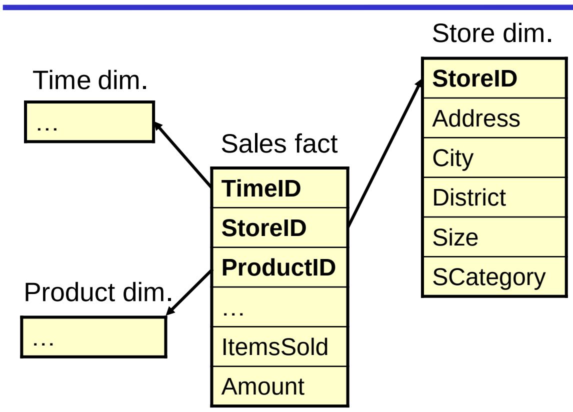
- Handling change over time
- Changes in dimensions
  - 1. No special handling
  - 2. Versioning dimension values
    - 2A. Special facts
    - 2B. Timestamping
  - 3. Capturing the previous and the current value
  - 4. Split into changing and constant attributes

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

Example



The store in Aalborg has the size of <u>250</u> sq. metres.

On a certain day, customers bought <u>2000</u> apples from that store.

Sales fact table

StoreID	…	ItemsSold	…
001		2000	

Store dimension table

StoreID	…	Size	…
001		250	

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?

8

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

9

Solution 2

- **Solution 2**: Versioning of rows with changing attributes
  - The *key* that links dimension and fact table, identifies a *version* of a row, not just a “row”
  - 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
    - e.g., relationship between the old store and the new store not captured

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 (old fact) refer to?

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

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 2A

- **Solution 2A**: Use special facts for capturing changes in dimensions via the Time dimension
  - Assume that no simultaneous, new fact refers to the new dimension row
  - Insert a new special fact that points to the new dimension row, and through its reference to the Time dimension, timestamps the row
- Pros
  - Possible to capture the development over time of the subjects that the dimensions describe
- Cons
  - Larger fact table

Solution 2A: Inserting Special Facts

StoreID	TimeID	…	ItemsSold	…
001	234		2000	

StoreID	…	Size	…
001		250	

special fact for capturing changes

StoreID	TimeID	…	ItemsSold	…
001	234		2000	
002	345		-	

StoreID	…	Size	…
001		250	
002		450	

StoreID	TimeID	…	ItemsSold	…
001	234		2000	
002	345		-	
002	456		3500	

StoreID	…	Size	…
001		250	
002		450	

Solution 2B

- **Solution 2B**: Versioning of rows with changing attributes like in Solution 2 + timestamping of rows
- Pros
  - Correct information captured in DW
- Cons
  - Larger dimension tables

Solution 2B: Timestamping

StoreID	TimeID	…	ItemsSold	…
001	234		2000	

StoreID	Size	From	To
001	250	98	-

attributes: “From”, “To”

StoreID	TimeID	…	ItemsSold	…
001	234		2000	

StoreID	Size	From	To
001	250	98	99
002	450	00	-

StoreID	TimeID	…	ItemsSold	…
001	234		2000	
002	456		3500	

StoreID	Size	From	To
001	250	98	99
002	450	00	-

Example of Using Solution 2B

- 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
- Unlike alternative categorizations that one wants to choose between

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 2B 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)

CustID
Name
PostalAddress
Gender
DateofBirth
Customerside
…
NoKids
MaritialStatus
CreditScore
BuyingStatus
Income
Education
…

CustID
Name
PostalAddress
Gender
DateofBirth
Customerside
…

Customer dimension (new):

relatively static attributes

DemographyID
NoKids
MaritialStatus
CreditScoreGroup
BuyingStatusGroup
IncomeGroup
EducationGroup
…

Demographics dimension:

often-changing attributes

20

Solution 4

- Solution 4
  - Make a “minidimension” with the often-changing (demograhic) attributes
  - Convert (numeric) attributes with many possible values into attributes with few discrete or banded values
    - E.g., Income group: [0,10K), [0,20K), [0,30K), [0,40K)
    - **Why? Any Information Loss?**
  - Insert rows for all combinations of values from these new domains
    - With 6 attributes with 10 possible values each, the dimension gets 10^6=1,000,000 rows
  - If the minidimension is too large, it can be further split into more minidimensions
    - Here, synchronous/correlated attributes must be considered (and placed in the same minidimension)
    - The same attribute can be repeated in another minidimension

21

Solution 4 (Changing Dimensions)

- Pros
  - DW size (dimension tables) is kept down
  - Changes in a customer's demographic 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

Changing Dimensions - Summary

- Why are there changes in dimensions?
  - Applications change
  - The modeled reality changes
- Multidimensional models realized as star schemas support change over time to a large extent
- A number of techniques for handling change over time at the instance level was described
  - Solution 2 and the derived 2A and 2B are the most useful
  - Possible to capture change precisely

- Coordinating Data Cubes / Data Marts

DW Bus Architecture

- What method for DW construction?
  - Everything at once, top-down DW (”monoliths”)
  - Separate, independent marts (”stovepipes”, ”data islands”)
- None of these methods work in practice
  - Both have different ”built-in” problems
- Architecture-guided step-by-step method
  - Combines the advantages of the first two methods
- A data mart can be built much faster than a DW
  - ETL is always the hardest - minimize risk with a simple mart
  - But: data marts must be compatible
  - Otherwise, incomparable views of the enterprise result
- Start with **single-source** data marts
  - **Facts** from only one source makes everything easier

DW Bus Architecture

- Data marts built independently by departments
  - Good (small projects, focus, independence,…)
  - Problems with ”stovepipes” (reuse across marts impossible)
- **Conformed** dimensions and facts/measures
- Conformed dimensions
  - Same structure **and content** across data marts
  - Take data from the **best** source
  - Dimensions are **copied** to data marts (not a space problem)
- Conformed fact **definitions**
  - The same **definition** across data marts (price excl. sales tax)
  - Observe **units of measurement** (also currency, etc.)
  - Use the same name only if it is **exactly** the same concept
  - Facts **are not** copied between data marts (facts > 95% of data)
- This allows several data marts to work together
  - Combining data from several fact tables is no problem

26

DW Bus Architecture

- Dimension content managed by **dimension owner**
  - The Customer dimension is made and published in **one** place
- Tools query each data mart separately
  - Separate queries to each data mart
  - Results combined by tool or OLAP server
- It is **hard** to make conformed dimensions and facts
  - Organizational and political challenge, not technical
  - Get everyone together **and**
  - Get a **top manager** (CIO) to back the conformance decision.
  - **No-one** must be allowed to ”escape”
- Exception: if business areas are totally separate
  - No common management/control

27

Large Scale Cube Design

- The design is never ”finished”
  - The dimensional modeler is always looking for new information to include in dimensions and facts
  - A sign of success!
- New dimensions and measures introduced **gracefully**
  - Existing queries will give same result
  - Example: Location dimension can be added for **old**+new facts
  - Can usually be done if data has **sufficiently fine** granularity
- Data mart granularity
  - Always as **fine** as possible (transaction level detail)
  - Makes the mart insensitive to changes

Coordinating Data Marts

- Multi-source data marts
  - Not built initially due to too large complexity
  - Combine several single-source data marts (building blocks)
  - Built ”on top of” several single-source marts
  - Relatively simple due to conformed dimensions and facts
  - Can be done physically or virtually (in OLAP server)
  - Example: profitability data mart
  - Important to have fine (single transaction?) granularity

Matrix Method

- DW Bus Architecture Matrix
- Two-dimensional matrix
  - X-axis: dimensions
  - Y-axis: data marts
- Planning Process
  - Make list of data marts
  - Make list of dimensions
  - Mark co-occurrences (which marts have which dimensions)
  - Time dimension occurs in (almost) all marts

Matrix Example

	Time	Customer	Product	Supplier
Sales	+	+	+	
Costs	+		+	+
Profit	+	+	+	+

- Multidimensional database implementation
  - MS SQL Server
  - MS Analysis Services

MS SQL Server 2008

- Microsoft's RDBMS
  - Runs on Windows OS only
- Nice features built-in
  - Analysis Services
  - Integration Services
  - Reporting Services
- Easy to use
  - Graphical "Management Studio" and "BI Developer Studio"
  - Watch the demonstration videos from Microsoft to get a quick introduction

MS Analysis Services

- Cheap, easy to use, good, and widely used
- Support ROLAP, MOLAP, HOLAP technology
- Intelligent pre-aggregation (for improving query performance)
- Programming: MS OLE DB for OLAP interface
- Uses the query language MDX (***M***ulti***D***imensional e***X***pressions)

Summary

- Handling Changes in Dimensions
- Coordinating Data Cubes / Data Marts
- Multidimensional Database Implementation: MS SQL Server and Analysis Services

Extract, Transform, Load (ETL)

ETL Overview

- The ETL Process
- General ETL issues
  - Building dimensions
  - Building fact tables
  - Extract
  - Transformations/cleansing
  - Load
- SQL Server Integration Services

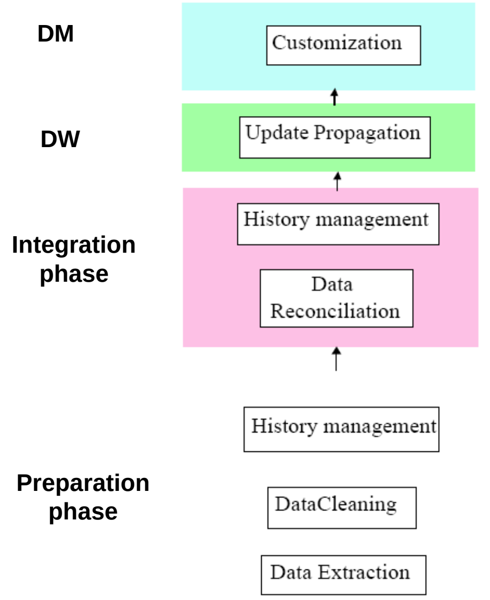
The ETL Process

- The **most underestimated** process in DW development
- The **most time-consuming** process in DW development
  - Up to 80% of the development time is spent on ETL!
- Extract
  - Extract relevant data
- Transform
  - Transform data to DW format
  - Build DW keys, etc.
  - Cleansing of data
- Load
  - Load data into DW
  - Build aggregates, etc.

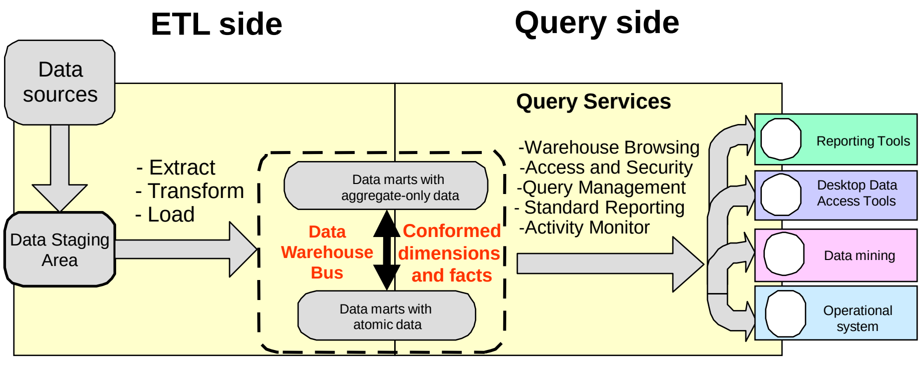
Phases

- Design phase
  - Modeling, DB design, source selection,…
- Loading phase
  - First load/population of the DW
  - Based on all data in sources
- Refreshment phase
  - Keep the DW up-to-date wrt. source data changes

ETL/DW Refreshment



ETL in the Architecture



Data Staging Area (DSA)

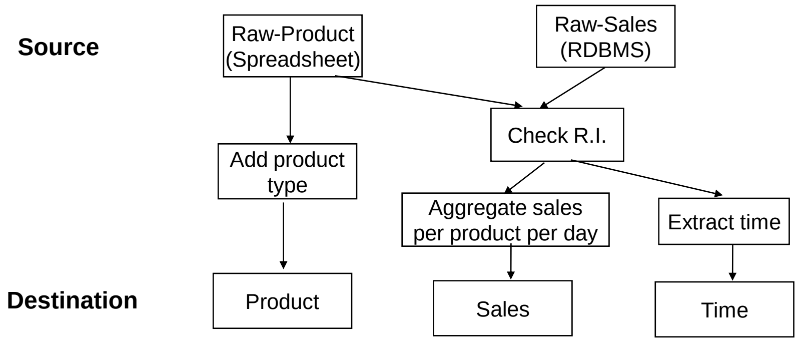
- Transit storage for data in the ETL process
  - Transformations/cleansing done here
- No user queries
- Sequential operations on large data volumes
  - Performed by central ETL logic
  - Easily restarted
  - No need for locking, logging, etc.
  - RDBMS or flat files? (DBMS have become better at this)
- Finished dimensions copied from DSA to relevant marts
- Allows centralized backup/recovery
  - Backup/recovery facilities needed
  - Better to do this centrally in DSA than in all data marts

ETL Construction Process

- Plan
  1) Make high-level diagram of source-destination flow
  2) Test, choose and implement ETL tool
  3) Outline complex transformations, DW key generation and job sequence for every destination table
- Construction of dimensions
  4) Construct and test building static dimension
  5) Construct and test change mechanisms for one dimension
  6) Construct and test remaining dimension builds
- Construction of fact tables and automation
  7) Construct and test initial fact table build
  8) Construct and test incremental update
  9) Construct and test aggregate build
  10) Design, construct, and test ETL automation

High-level diagram

1) Make high-level diagram of source-destination flow
   - Mainly used for communication purpose
   - One page only, highlight sources and destinations
   - Steps: extract, transform, load



Building Dimensions

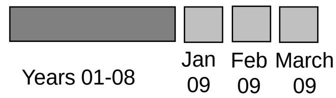
- Static dimension table
  - DW key assignment: production keys to DW keys using table
  - Check one-one and one-many relationships (using sorting)
- Handling dimension changes
  - Described in last lecture
  - Find the **newest** DW key for a given production key
  - Table for mapping production keys to DW keys must be maintained and updated
- Load of dimensions
  - Small dimensions: replace
  - Large dimensions: load only changes

Key mapping for the Product dimension

pid	DW_pid	Time
11	1	100
22	2	100
35	3	200
11	4	700
……	……	……

Product dimension of FClub vs. Product dimension of a supermarket

Building Fact Tables



- Two types of load
- Initial load
  - ETL for all data up till now
  - Done when DW is started the first time
  - Very heavy - large data volumes
- Incremental update
  - Move only changes since last load
  - Done periodically (e.g., month or week) after DW start
  - Less heavy - smaller data volumes
- Dimensions must be updated **before** facts
  - The relevant dimension rows for new facts must be in place
  - Special key considerations if initial load must be performed again

11

Extract

12

Types of Data Sources

- Non-cooperative sources
  - Snapshot sources – provides only full copy of source, e.g., files
  - Specific sources – each is different, e.g., legacy systems
  - Logged sources – writes change log, e.g., DB log
  - Queryable sources – provides query interface, e.g., RDBMS
- Cooperative sources
  - Replicated sources – publish/subscribe mechanism
  - Call back sources – calls external code (ETL) when changes occur
  - Internal action sources – only internal actions when changes occur
    - DB triggers is an example
- Extract strategy depends on the source types

13

Extract

- Goal: fast extract of relevant data
  - Extract from source systems can take **long** time
- Types of extracts:
  - Extract applications (SQL): co-existence with other applications
  - DB unload tools: faster than SQL-based extracts
    - e.g., MS SQL Export Wizard, MySQL DB dump
- Extract applications the only solution in some scenarios
- **Too** time consuming to ETL all data at each load
  - Can take days/weeks
  - Drain on the operational systems and DW systems
- Extract/ETL only changes since last load (delta)

14

Computing Deltas

- Delta = changes since last load
- Store sorted total extracts in DSA
  - Delta can easily be computed from current + last extract
  - + Always possible
  - + Handles deletions
  - - High extraction time
- Put update timestamp on all rows (in sources)
  - Updated by DB trigger
    - - Source system must be changed, operational overhead
  - Extract only where “timestamp > time for last extract”
    - + Reduces extract time
  - - Cannot (alone) handle deletions.

Last extract time: 300

Timestamp	DKK	…
100	10	…
200	20	…
300	15	…
400	60	…
500	33	…

Changed Data Capture

- Messages
  - Applications insert messages in a ”queue” at updates
  - + Works for all types of updates and systems
  - - Operational applications must be changed+operational overhead
- DB triggers
  - Triggers execute actions on INSERT/UPDATE/DELETE
  - + Operational applications need **not** be changed
  - + Enables real-time update of DW
  - - Operational overhead
- Replication based on DB log
  - Find changes directly in DB log which is written anyway
  - + Operational applications need **not** be changed
  - + No operational overhead
  - - Not possible in some DBMS (SQL Server, Oracle, DB2 can do it)

Transform

Common Transformations

- Data type conversions
  - EBCDIC → ASCII/Unicode
  - String manipulations
  - Date/time format conversions
    - E.g., Unix time 1201928400 = what time?
- Normalization/denormalization
  - To the desired DW format
  - Depending on source format
- Building keys
  - Table matches production keys to surrogate DW keys
  - Correct handling of history - especially for total reload

Data Quality

- Data almost **never** has decent quality
- Data in DW must be:
  - Precise
    - DW data must match known numbers
  - Complete
    - DW has all relevant data
  - Consistent
    - No contradictory data: aggregates fit with detail data
  - Unique
    - The same thing is called the same and has the same key
  - Timely
    - Data is updated ”frequently enough” and the users know when

Cleansing

- Why cleansing? Garbage In Garbage Out
- BI does not work on “raw” data
  - Pre-processing necessary for BI analysis
- Handle inconsistent data formats
  - Spellings, codings, …
- Remove unnecessary attributes
  - Production keys, comments,…
- Replace codes with text for easy understanding
  - City name instead of ZIP code, e.g., *Aalborg* vs. *DK-9000*
- Combine data from multiple sources with common key
  - E.g., customer data from customer address, customer name, …

Types of Cleansing

- Conversion and normalization
  - Most common type of cleansing
  - Text coding, date formats
    - does 3/2 mean 3rd February or 2nd March?
- Special-purpose cleansing
  - Look-up tables, dictionaries to find valid data, synonyms, abbreviations
  - Normalize spellings of names, addresses, etc.
    - Dorset *Rd* or Dorset *Road*? København or Copenhagen? Aalborg or Ålborg?
  - Remove duplicates, e.g., duplicate customers
- Domain-independent cleansing
  - Approximate, “fuzzy” joins on records from different sources
  - E.g., two customers are regarded as the same if their respective values match for most of the attributes (e.g., address, phone number)
- Rule-based cleansing
  - User-specified rules: if-then style
  - Automatic rules: use data mining to find patterns in data
    - Guess missing sales person based on customer and item

Cleansing

- Don’t use “special” values (e.g., 0, -1) in your data
  - They are hard to understand in query/analysis operations
- Mark facts with Data Status dimension
  - Normal, abnormal, outside bounds, impossible,…
  - Facts can be taken in/out of analyses
- Uniform treatment of NULL
  - Use NULLs only for measure values (estimates instead?)
  - Use special dimension key (i.e., surrogate key value) for NULL dimension values
    - E.g., for the time dimension, instead of NULL, use special key values to represent “Date not known”, “Soon to happen”
    - Avoids problems in joins, since NULL is not equal to NULL

Data Status Dimension

SID	Status
1	Normal
2	Abnormal
3	Out of bounds
…	…

Sales fact table

Sales	SID	…
10	1	…
20	1	…
10000	2	…
-1	3	…

Improving Data Quality

- Appoint “data steward”
  - Responsibility for data quality
  - Includes manual inspections and corrections!
- DW-controlled improvement
  - Default values
  - ”Not yet assigned 157” note to data steward
- Source-controlled improvements
- Construct programs that check data quality
  - Are totals as expected?
  - Do results agree with alternative source?
  - Number of NULL values?

Load

Load

- Goal: fast loading into DW
  - Loading deltas is much faster than total load
- SQL-based update is **slow**
  - Large overhead (optimization, locking, etc.) for every SQL call
  - DB load tools are much faster
- Index on tables **slows** load a lot
  - Drop index and rebuild after load
  - Can be done per index partition
- Parallellization
  - Dimensions can be loaded concurrently
  - Fact tables can be loaded concurrently
  - Partitions can be loaded concurrently

Load

- Relationships in the data
  - Referential integrity and data consistency must be ensured before loading
    - Because they won't be checked in the DW again
  - Can be done by loader
- Aggregates
  - Can be built and loaded at the same time as the detail data
- Load tuning
  - Load without log
  - Sort load file first
  - Make only simple transformations in loader
  - Use loader facilities for building aggregates

ETL Tools

- ETL tools from the big vendors
  - Oracle Warehouse Builder
  - IBM DB2 Warehouse Manager
  - Microsoft SQL Server Integration Services (SSIS)
- Offers much functionality
  - Data modeling
  - ETL code generation
  - Scheduling DW jobs
- … but (some) have steep learning curves and high costs
- The “best” tool does not exist
  - Choose based on your own needs
  - You may also have to code your own

Issues

- Pipes
  - Redirect output from one process to input of another process
    `cat payments.dat | grep 'payment' | sort -r`
- Files versus streams/pipes
  - Streams/pipes: no disk overhead, fast throughput
  - Files: easier restart, often only possibility
- Use ETL tool or write ETL code
  - Code: easy start, co-existence with IT infrastructure, maybe the only possibility
  - Tool: better productivity on subsequent projects, “self-documenting”
- Load frequency
  - ETL time dependent of data volumes
  - Daily load is much faster than monthly
  - Applies to all steps in the ETL process

SQL Server Integration Services

- A concrete ETL tool
- Example ETL flow

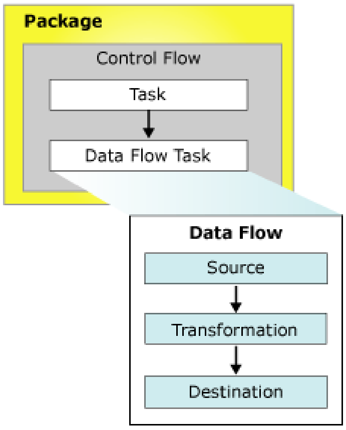
Integration Services (IS)

- Microsoft's ETL tool
  - Part of SQL Server 2008
- Tools
  - Import/export wizard - simple transformations
  - BI Development Studio - advanced development
- Functionality available in several ways
  - Through GUI - basic functionality
  - Programming - advanced functionality

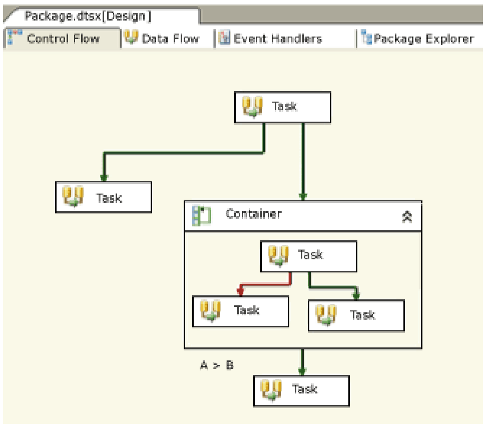
Packages

- A package is a collection of
  - Data flows (Sources → Trans-formations → Destinations)
  - Connections
  - Control flow: Tasks, Workflows
  - Variables
  - …

- A package may also invoke other packages and/or processes
- It is somehow similar to a "program"



A package



Arrows show precendence constraints

Constraint values:

• success (green)

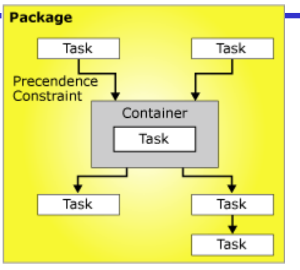
• failure (red)

• completion (blue)

Conditional expressions may also be given (A > B)

Package Control Flow

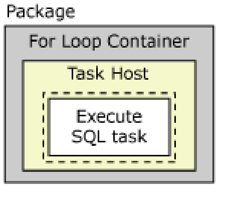
- Containers provide
  - Structure to packages
  - Services to tasks
- Control flow
  - Foreach loop container
    - Repeat tasks by using an enumerator
  - For loop container
    - Repeat tasks by testing a condition
  - Sequence container
    - Groups tasks and containers into control flows that are subsets of the package control flow
- Task host container
  - An abstract container class which is used implicitly



33

Tasks

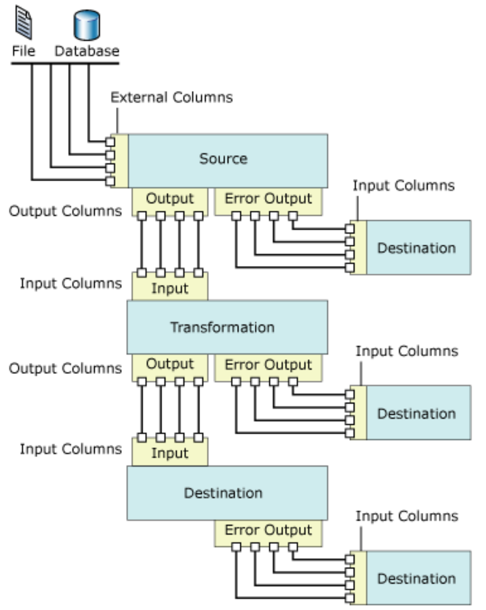
- A *task* is a unit of work
- Workflow Tasks
  - Execute package – execute other SSIS packages, good for structure!
  - Execute Process – run external application/batch file
- SQL Servers Tasks
  - Bulk insert – fast load of data
  - Execute SQL – execute any SQL query
- Data Preparation Tasks
  - File System – operations on files
  - FTP – up/download data
- Scripting Tasks
  - Script – execute .NET code
- Maintenance Tasks – DB maintenance
- Data Flow Tasks – run data flows from *sources* through *transformations* to *destinations* (this is where the work is done)



34

Data Flow Elements

- Sources
  - Make external data available
  - All ODBC/OLE DB data sources: RDBMS, Excel, Text files, ……
- Transformations
  - Update, summarize, cleanse, merge
- Destinations
  - Write data to specific store
- Input, Output, Error output



36

Transformations

- Row Transformations
  - Character Map - applies string functions to character data
  - Derived Column – populates columns using expressions
- Rowset Transformations (*rowset = tabular data*)
  - Aggregate - performs aggregations
  - Sort - sorts data
  - Percentage Sampling - creates sample data set by setting %
- Split and Join Transformations
  - Conditional Split - routes data rows to different outputs
  - Merge - merges two sorted data sets
  - Lookup Transformation - looks up ref values by exact match
- Other Transformations
  - Export Column - inserts data from a data flow into a file
  - Import Column - reads data from a file and adds it to a data flow
  - Slowly Changing Dimension - configures update of a SCD

37

A Few Hints on ETL Design

- **Don't** implement all transformations in one step!
  - Build first step and check that result is as expected
  - Add second step and execute both, check result (How to check?)
  - Add third step ……
- Test SQL statements before putting into IS
- Do **one** thing at the time
  - Copy source data one-by-one to the data staging area (DSA)
  - Compute deltas
    - Only if doing incremental load
  - Handle versions and DW keys
    - Versions only if handling slowly changing dimensions
  - Implement complex transformations
  - Load dimensions
  - Load facts

Summary

- The ETL Process
- Extract
- Transformations/cleansing
- Load