

ETL and Advanced Modeling

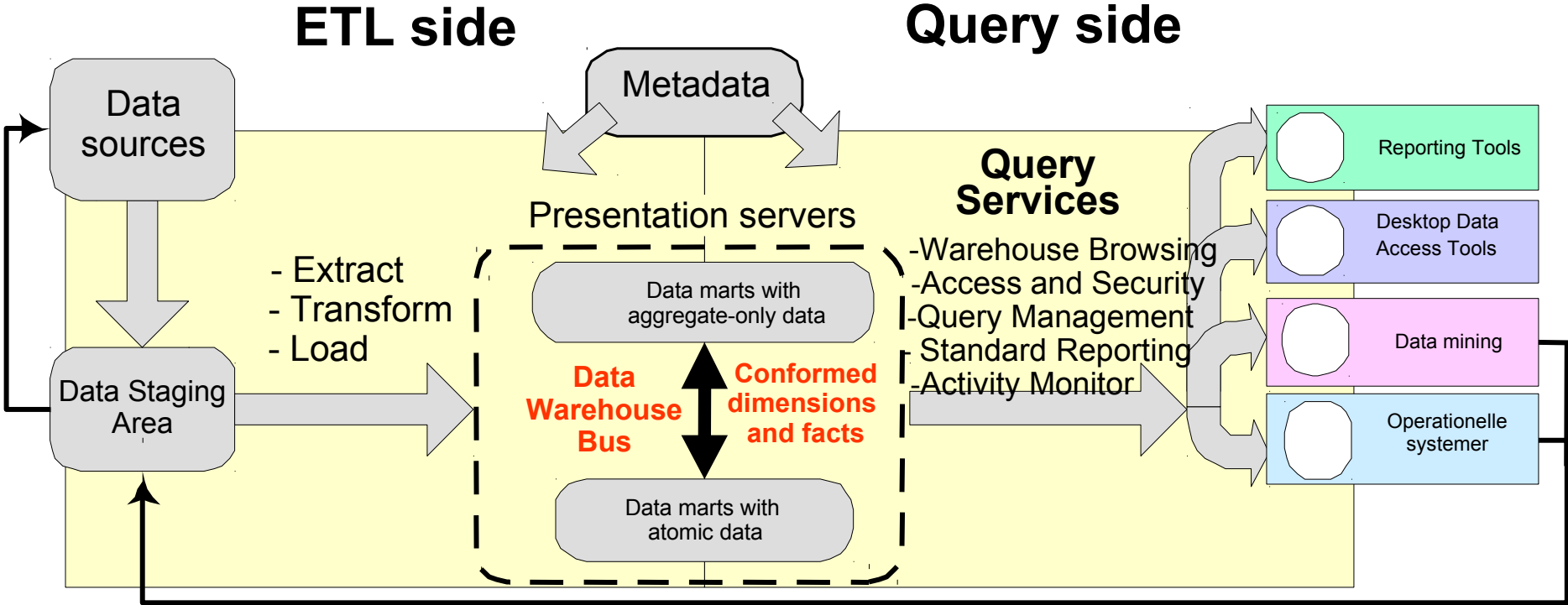
1. Extract-Transform-Load (ETL)
 - The ETL process
 - Building dimension and fact tables
 - Extract, transform, load
2. Advanced Multidimensional Modeling
 - Handling changes in dimensions
 - Large-scale dimensional modeling

Acknowledgements: I am indebted to Michael Böhlen and Stefano Rizzi for providing me their slides, upon which these lecture notes are based.

The ETL Process

- The **most underestimated** and **time-consuming** process in DW development
 - Often, 80% of development time is spent on ETL
- The ETL system is the **foundation** of the DW/BI project
 - its success makes or breaks the data warehouse.
- Extract
 - Extract relevant data
- Transform
 - Transform data to DW format
 - Build keys, etc.
 - Cleansing of data
- Load
 - Load data into DW
 - Build aggregates, etc.

ETL – Big Picture



Data Staging Area

- Transit storage for data underway in the ETL process
 - Transformations/cleansing done here
- No user queries (some do it)
- Sequential operations (few) 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

The 34 ETL Subsystems

- Kimball et al. report 34 subsystems to compose the ETL system
 - Extracting
 - ◆ Gathering raw data from the source systems and usually writing it to disk in the ETL environment
 - ◆ 3 subsystems
 - Cleaning and conforming:
 - ◆ Sending source data through a series of processing steps in the ETL system to improve the quality of the data and merging data
 - ◆ 4 subsystems
 - Delivering:
 - ◆ Physically structuring and loading the data into the dimensional model
 - ◆ 13 subsystems
 - Managing (usually considered a separate component):
 - ◆ Managing the related systems and processes of the ETL environment
 - ◆ 13 subsystems

ETL Construction Process

- Plan
 - Make high-level diagram of source-destination flow
 - Test, choose and implement ETL tool
 - Outline complex transformations, key generation and job sequence for every destination table
- Construction of dimensions
 - Construct and test building static dimension
 - Construct and test change mechanisms for one dimension
 - Construct and test remaining dimension builds
- Construction of fact tables and automation
 - Construct and test initial fact table build
 - Construct and test incremental update
 - Construct and test aggregate build (will be done later)
 - Design, construct, and test ETL automation

Building Dimensions

- Static dimension table
 - Relatively easy?
 - Assignment of keys: production keys to DW keys using mapping table
 - Combination of data sources: find common key?
 - Check one-one and one-many relationships using sorting
- Handling dimension changes
 - Find newest DW key for a given production key
 - Table for mapping production keys to DW keys must be updated
- Load of dimensions
 - Small dimensions: replace
 - Large dimensions: load only changes

Building Fact Tables

- Two types of load
- Initial load
 - ETL for all data up till now
 - Done when DW is started the first time
 - Often problematic to get correct historical data
 - Very heavy - large data volumes
- Incremental update
 - Move only changes since last load
 - Done periodically (../month/week/day/hour/...) 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

Extract

- Goal: fast extract of relevant data
 - Extract from source systems can take a **long** time
- Types of extracts:
 - Extract applications (SQL): co-existence with other applications
 - DB unload tools: much faster than SQL-based extracts
 - Extract applications sometimes the only solution
- Often **too** time consuming to ETL all data at each load
 - Extracts can take days/weeks
 - Drain on the operational systems
 - Drain on DW systems
 - => Extract/ETL only changes since last load (delta)

E - Computing Deltas

- Much faster to only "ETL" changes since last load
 - A number of methods can be used
- Store sorted total extracts in DSA
 - Delta can easily be computed from current+last extract
 - + Always possible
 - + Handles deletions
 - - Does not reduce extract time
- Put update timestamp on all rows
 - Updated by DB trigger
 - Extract only where "timestamp > time for last extract"
 - + Reduces extract time
 - +/- Less operational overhead
 - - Cannot (alone) handle deletions
 - - Source system must be changed

E - Capturing Changed Data

- 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)

E - Data Quality

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

T – Data Transformations

- Data type conversions
 - EBCDIC->ASCII/UniCode
 - String manipulations
 - **Date/time format conversions**
- Normalization/denormalization
 - To the desired DW format
 - Depending on source format
- Building keys
 - Table that maps production keys to surrogate DW keys
 - Observe correct handling of history - especially for total reload

T - Cleansing/1

- BI does not work on "raw" data
 - Pre-processing necessary for good results
 - Can "disturb" BI analyses if not handled (e.g., duplicates)
- Handle inconsistent data formats
 - Spellings, codings, ...
- Remove unneeded attributes
 - Production keys, comments,...
- Replace codes with text
 - City name instead of ZIP code, 0/1 by Yes/No, ...
- Combine data from multiple sources with common key
 - Customer data,...
- Find attributes with several uses
 - Extra values used for "programmer hacks"

T – Cleansing/2

- Mark facts with Data Status dimension
 - Normal, abnormal, outside bounds, impossible,...
 - Facts can be taken in/out of analyses
- Recognize random or noise values
 - Can disturb BI tools
 - Replace with NULLs (or estimates)
- Uniform treatment of NULL
 - Use explicit NULL value rather than "normal" value (0,-1,...)
 - Use NULLs only for measure values (estimates instead?)
 - Use special dimension keys for NULL dimension values
- Mark facts with changed status
 - Customer about to cancel contract,...
- Aggregate fact data?
 - Performance
 - Statistical significance only for higher levels

T – Data Mining Transformations

- Can often **not** be done generally for the whole DW
- Divide data into training-, test-, and evaluation sets
 - Training: used to train model
 - Test: used to check the model's generality (overfitting)
 - Evaluation: model uses evaluation set to find "real" clusters,...
- Add computed attributes as inputs or "targets"
 - Derived attributes often more interesting for mining (profit,..)
 - Means more possibilities for data mining
- Mapping
 - Continuous values to intervals, textual values to numerical
- Normalization of values between 0 and 1
 - Demanded by some tools (neural nets)
- Emphasizing the rare case
 - Duplicate rare elements in training set

T – Improving Data Quality

- Appoint "data stewards" - responsible for data quality
 - A given steward has the responsibility for certain tables
 - Includes manual inspections and corrections!
- DW-controlled improvement
 - Default values
 - "Not yet assigned 157" note to data steward
- Source-controlled improvements
 - The optimal?
- Construct programs that check data quality
 - Are totals as expected?
 - Do results agree with alternative source?
- Do not fix **all** problems with data quality
 - Allow management to see "weird" data in their reports?

Load/1

- 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
 - Some load tools can also perform UPDATES
- Index on tables **slows** load a lot
 - Drop index and rebuild after load
 - Can be done per partition
- Parallelization
 - Dimensions can be loaded concurrently
 - Fact tables can be loaded concurrently
 - Partitions can be loaded concurrently

Load/2

- Relationships in the data
 - Referential integrity must be ensured
 - Can be done by loader
- Aggregates
 - Must be built and loaded at the same time as the detail data
 - Today, RDBMSes can often do this
- Load tuning
 - Load without log
 - Sort load file first
 - Make only simple transformations in loader
 - Use loader facilities for building aggregates
 - Use loader within the same database
- Should DW be on-line 24*7?
 - Use partitions or several sets of tables

ETL Tools

- ETL tools from the big vendors
 - Oracle Warehouse Builder / Oracle Data Integrator
 - IBM DB2 Warehouse Manager
 - Microsoft Data Transformation Services
- Offers much functionality at a reasonable price (included...)
 - Data modeling
 - ETL code generation
 - Scheduling DW jobs
 - ...
- **Many** others
 - Hundreds of tools
 - Often specialized tools for certain jobs (insurance cleansing,...)
- The "best" tool does not exist
 - Choose based on your own needs
 - Check first if the "standard tools" from the big vendors are ok

Issues

- Files versus streams/pipes
 - Streams/pipes: no disk overhead, fast throughput
 - Files: easier restart, often only possibility
- ETL tool or not
 - Code: easy start, co-existence with IT infrastructure
 - Tool: better productivity on subsequent projects
- Load frequency
 - ETL time dependent of data volumes
 - Daily load is much faster than monthly
 - Applies to all steps in the ETL process

A Few Hints on ETL Design

- Do **not** try to implement all transformations in one step!
- Do **one** (or just a few) thing(s) at the time
 - Copy source data one-one to 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

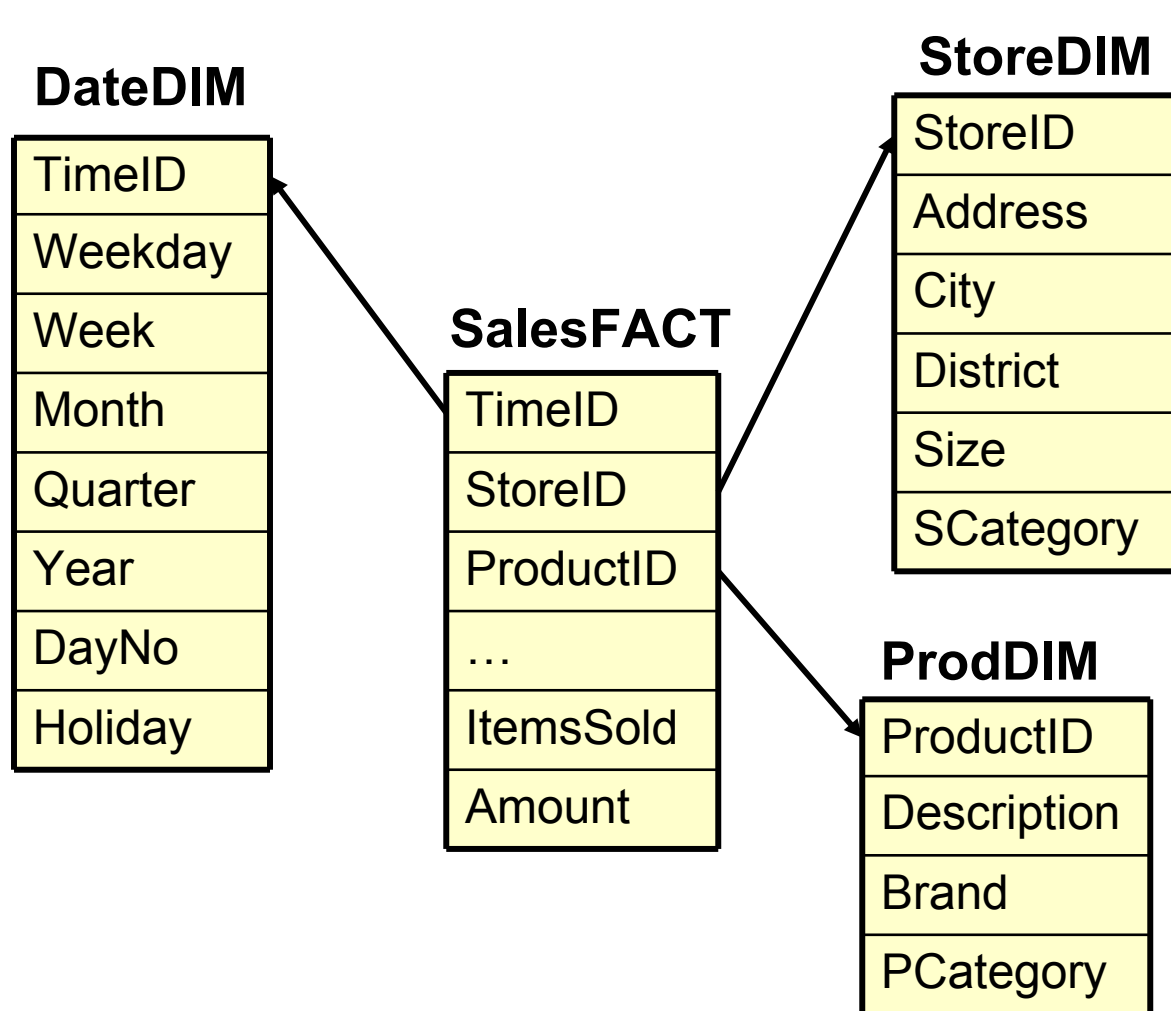
Advanced MD Modeling

- Handling change over time
- Changes in dimensions
 - No special handling
 - Versioning dimension values
 - Capturing the previous and the actual value
 - Timestamping
 - Split into changing and constant attributes
- Large-scale dimensional modeling

Changing Dimensions

- So far, we have implicitly assumed that dimensions are stable over time.
 - At most, new rows in dimension tables are inserted.
 - The existing rows do not change.
- This assumption is not valid in practice.
 - The phenomenon is called “slowly changing dimensions”.
 - The intuition is, that dimension information change, but changes are (relatively) rare.
- We will look at a number of techniques for handling changes in dimensions.
- Schema changes are not considered now.
 - Then it becomes really funny!

Changing Dimensions/2



- Descriptions of stores and products vary over time.
- A store is enlarged and changes Size.
- A product changes Description.
- Districts are changed.
- Problems
 - If we update the dimensions, wrong information will result.
 - If we don't update the dimensions, the DW is not up-to-date.

Changing Dim – Overwrite Old Values/1

- **Solution 1:** Overwrite the old values that change, 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.
 - ◆ New facts are facts that are inserted after the dimension rows they point to are inserted/changed.
- Pros
 - Easy to implement
 - Ideal if the changes are due to erroneous registrations.
 - In some cases, the "imprecision" can be disregarded.
- Cons
 - "The solution" does not solve the problem of capturing change.

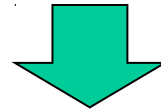
Changing Dim – Overwrite Old Values/2

SalesFACT

StoreID	...	ItemsSold	...
001		2000	

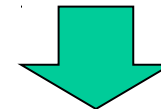
StoreDIM

StoreID	...	Size	City
001		250	Chikago



StoreID	...	ItemsSold	...
001		2000	

StoreID	...	Size	City
001		450	Chicago



StoreID	...	ItemsSold	...
001		2000	
001		2500	

StoreID	...	Size	City
001		450	Chicago

Changing Dim – Versioning of Rows/1

- **Solution 2:** Versioning of rows with changing attributes.
 - The *key* that links dimension and fact table should now identify a *version* of a row, not just a "row".
 - The key is generalized.
 - If "stupid" ("non information-bearing", "surrogate") keys are used, there is no need for changes.
- Consequences
 - Larger dimension tables
- Pros
 - Correct information captured in DW
 - No problems when formulating queries
- Cons
 - It is not possible to capture the development over time of the subjects the dimensions describe since time of change is not recorded.

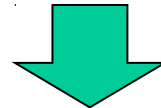
Changing Dim – Versioning of Rows/2

SalesFACT

StoreID	...	ItemsSold	...
001		2000	

StoreDIM

StoreID	...	Size	...
001		250	



StoreID	...	ItemsSold	...
001		2000	

StoreID	...	Size	...
001		250	
002		450	



StoreID	...	ItemsSold	...
001		2000	
002		2500	

StoreID	...	Size	...
001		250	
002		450	

Changing Dim – Two Attribute Values/1

- **Solution 3:** Create two versions of each changing attribute
 - One attribute contains the actual value
 - The other attribute contains the previous value
- **Consequences**
 - Two values are attached to each dimension row.
- **Pros**
 - It is possible to compare across the change in dimension value (which is a problem with Solution 2).
 - Such comparisons are interesting in certain situations, where it is logical 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.

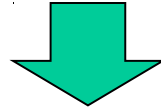
Changing Dim – Two Attribute Values/2

SalesFACT

StoreID	...	ItemsSold	...
001		2000	

StoreDIM

StoreID	...	DistrictOld	DistrictNew	...
001		37	37	



StoreID	...	ItemsSold	...
001		2000	

StoreID	...	DistrictOld	DistrictNew	...
001		37	73	



StoreID	...	ItemsSold	...
001		2000	
001		2100	

StoreID	...	DistrictOld	DistrictNew	...
001		37	73	

Changing Dim – Special Facts/1

- **Solution 2.1:** Use special facts for capturing changes in dimensions via the Time dimension.
 - When a change occurs and there is no simultaneous, new fact referring to the new dimension row, a new special fact is created that points to the new dimension row and thus timestamps the row via the fact row's reference to the Time dimensions.
- **Pros**
 - It is possible to capture the development over time of the subjects that the dimensions describe.
- **Cons**
 - Complexity of working with this solution.

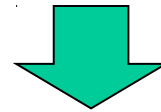
Changing Dim – Special Facts/2

SalesFACT

StoreID	TimeID	...	ItemsSold	...
001	234		2000	

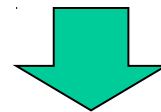
StoreDIM

StoreID	...	Size	...
001		250	



StoreID	TimeID	...	ItemsSold	...
001	234		2000	
002	345		-	

StoreID	...	Size	...
001		250	
002		450	



StoreID	TimeID	...	ItemsSold	...
001	234		2000	
002	345		-	
002	456		2500	

StoreID	...	Size	...
001		250	
002		450	

Changing Dim – Timestamping of Rows/1

- **Solution 2.2:** Versioning of rows with changing attributes like in Solution 2 + timestamping of rows.
- Pros
 - Correct information captured in DW
- Cons
 - Larger dimension tables
 - Time dimension values and timestamps do not describe the same aspect of time => subtle to use

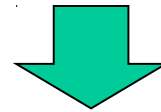
Changing Dim – Timestamping of Rows/2

SalesFACT

StoreID	TimeID	...	ItemsSold	...
001	234		2000	

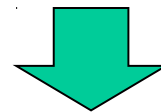
StoreDIM

StoreID	Size	From	To
001	250	98	-



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		2500	

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

Changing Dim – Timestamping of Rows/3

- **Solution 2.2:** Examples
- Product descriptions are versioned, when products are changed, e.g., new package sizes.
- New facts can refer to both the newest and older versions of products, as old versions are still in the stores.
- Thus, the Time value for a fact should not necessarily be between the From and To values in the fact's Product dimension row.
- This is unlike changes in Size for a store, where all facts from a certain point in time will refer to the newest Size value .
- This is also unlike alternative categorizations (e.g, department) that one wants to choose between.

Rapidly Changing Dimensions/1

- Handling “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 and many changes every year.
 - Kimball recommends Solution 2.2.
- Other typical examples of (large) dimensions with many changes are Product and Customer.
- Example
 - Some Customer dimensions can have 10M customers.
 - Use Solution 2 and suitable indexing!

Rapidly Changing Dimensions/2

- Handling “rapidly changing monster dimensions”.
- The more attributes in a dimension table, the more changes per row can be expected.
- Solution 2 yields a dimension that is too large.
- Example
 - A Customer dimension with 100M customers and many attributes.

Rapidly Changing Dimensions/3

- **Solution with Mini-dimension**

- Make a "minidimension" with the often-changing (demographic) attributes.
- Convert (numeric) attributes with many possible values into attributes with few possible values, representing groups of the original values.
- Insert rows for all combinations of values from these new domains.
 - ◆ With 6 attributes with 10 possible values each, the dimension gets 1,000,000 rows.
 - ◆ Alternatively, (combination) rows can be inserted when needed.
- If the minidimension is too large, it can be split into two or more minidimensions.
 - ◆ Here, synchronous attributes must be considered (and placed in the same minidimension).
 - ◆ The same attribute can be repeated in another minidimension.

Rapidly Changing Dimensions/4

CustID
Name
PostalAddress
Gender
DateofBirth
Customerside
...
NoKids
MaritalStatus
CreditScore
BuyingStatus
Income
Education
...



CustID
Name
PostalAddress
Gender
DateofBirth
Customerside
...



DemographyID
NoKids
MaritalStatus
CreditScoreGroup
BuyingStatusGroup
IncomeGroup
EducationGroup
...

Rapidly Changing Dimensions/5

- Pros
 - DW size (dimension tables) is kept down.
 - Changes in a customer's demographic values do not result in changes in dimensions.
 - ◆ With the alternative solution, rows must be inserted into the minidimension.
- Cons
 - More dimensions and more keys in the star schema.
 - Using value groups gives less detail.
 - The construction of groups is irreversible and makes it hard to make other groupings.
 - Navigation of customer attributes is more cumbersome as these are in more than one dimension.
 - ◆ An ActualDemography attribute can be added to the dimension with the stable values.

Changing Dimensions - Summary

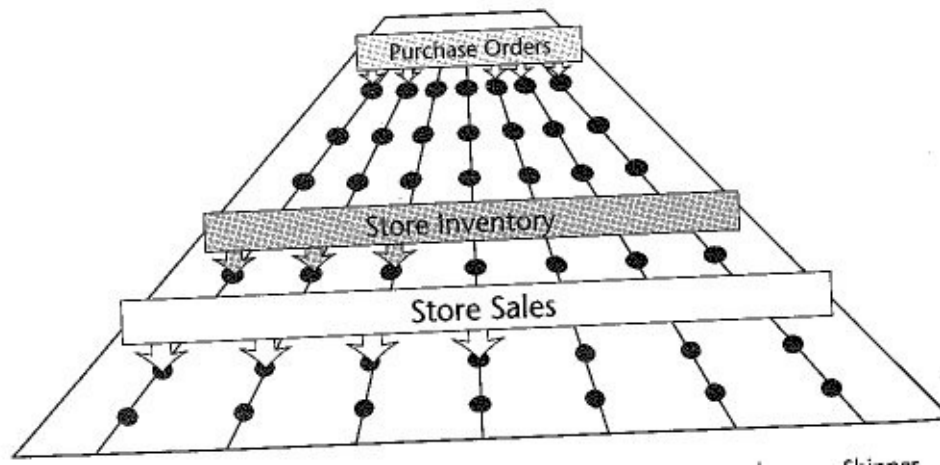
- Multidimensional models realized as star schemas support change over time to a large extent.
- This is important!
 - Applications change.
 - The modeled reality changes.
- A number of techniques for handling change over time at the instance level was described.
 - In general, solution 2 is the most useful.
 - There exist application scenarios for all solutions
 - It is possible to capture change precisely.

What Design Methodology to Use?

- What method for DW construction?
 - Everything at once, top-down DW ("monoliths")
 - Separate, independent, bottom-up data 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/1

- A standard bus interface for a DW environment that allows to implement separate data marts that can be successfully integrated
- Based on conformed (similar) dimensions that are shared by the data marts



- Guides the overall design and breaks down the development process into small chunks (DMs)

DW Bus Architecture/2

- 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)
 - Facts **are not** copied between data marts (facts > 95% of data)
 - Observe **units of measurement** (also currency, etc.)
 - Use the same name only if it is **exactly** the same concept
- This allows several data marts to work together
 - Combining data from several fact tables is no problem

DW Bus Architecture/3

- Dimension content managed by **dimension owner**
 - The Customer dimension is made and published in **one** place
- Tools query each data mart separately
 - Separate (SQL) queries to each data mart
 - Results combined (outer join) 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
 - Build several DWs

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

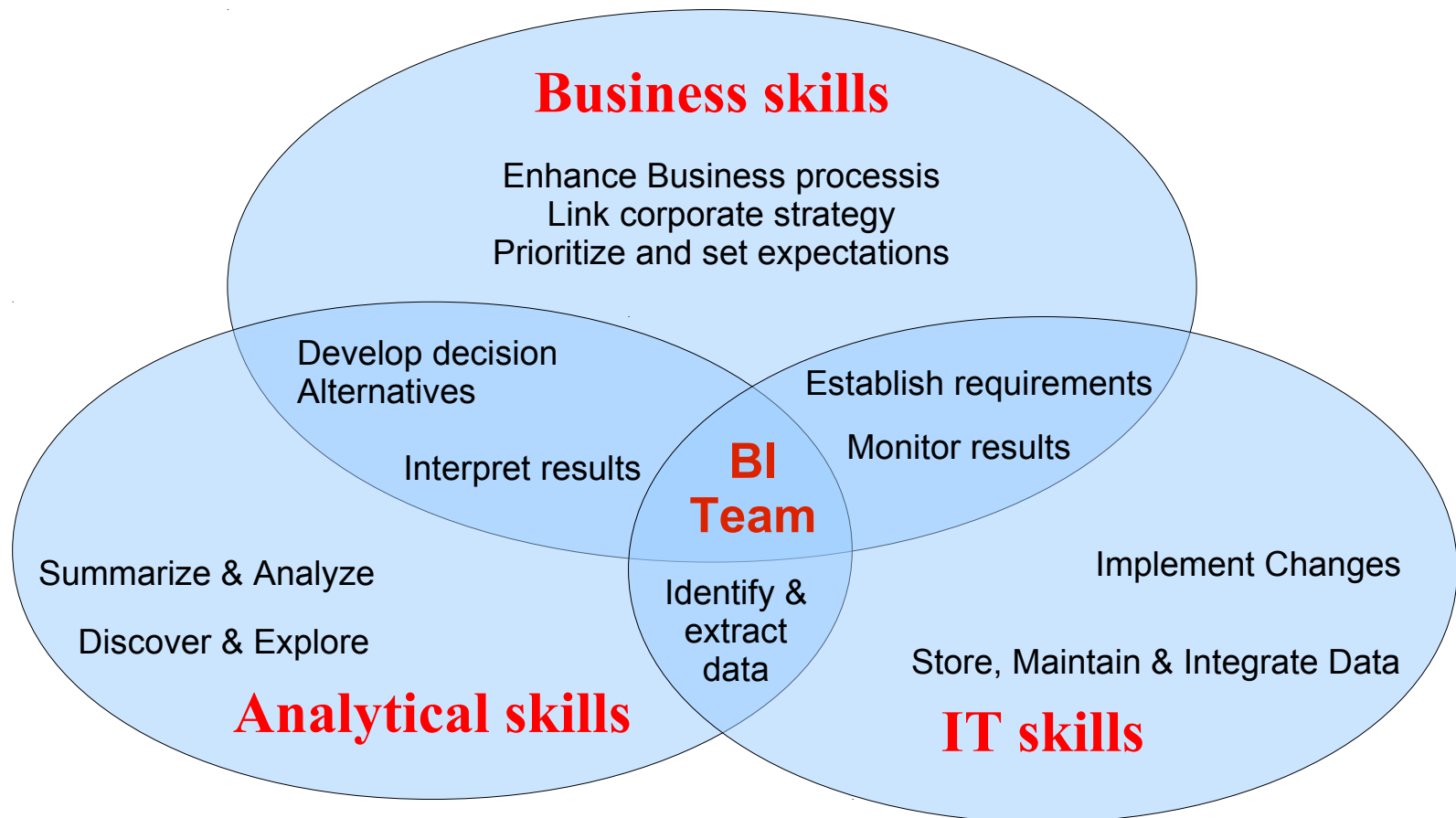
- Multi-source data marts
 - Not built initially due to too large complexity
 - Combine several single-source data marts (building blocks)
 - Built "on top of" several single-source marts
 - Relatively simple due to conformed dimensions and facts
 - Can be done physically or virtually (in OLAP server)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

DW Project Management/1

- 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,...)

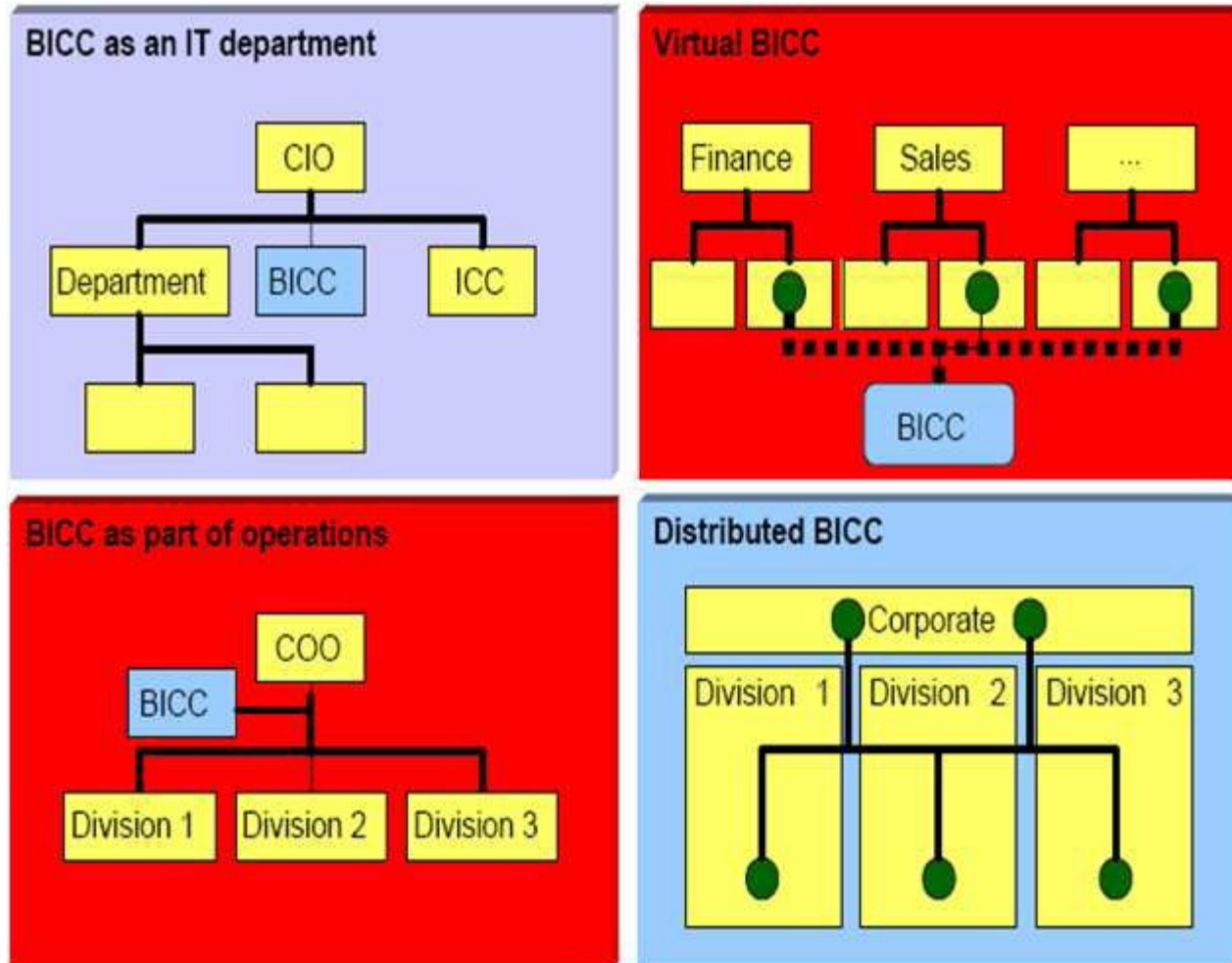
DW Project Management/2

- Create **Business Intelligence Competence Center (BICC)** which is responsible for the DW project



DW Project Management/2

- BICC requires a **change** in the organization (difficult!)
- No best place, but has strategic importance



Summary

- ETL
 - is very time consuming (80% of entire project)
 - consists of many small steps
- Advanced multidimensional modeling
 - Mainly handling changes in dimensions
- Large-scale dimensional modeling
 - Coordinating cubes/data marts
 - Conformed dimensions and facts
- DW Project Management
 - Complex projects and long-term
 - Establishment of a BICC is crucial for success