Week 01 Introduction to Data Warehousing

Reading:

"A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process."—W. H. Inmon

	OLTP	OLAP
users	clerk, IT professional	knowledge worker
function	day to day operations	decision support
DB design	application-oriented	subject-oriented
data	current, up-to-date detailed, flat relational isolated	historical, summarized, multidimensional integrated, consolidated
usage	repetitive	ad-hoc
access	read/write index/hash on prim. key	lots of scans
unit of work	short, simple transaction	complex query
# records accessed	tens	millions
#users	thousands	hundreds
DB size	100MB-GB	100GB-TB
metric	transaction throughput	query throughput, response

- **Modeling data warehouses: dimensions:**
  - Star schema: A fact table in the middle connected to a set of dimension tables
  - Snowflake schema: A refinement of star schema where some dimensional hierarchy is normalized into a set of smaller dimension tables, forming a shape similar to snowflake
  - Fact constellations: Multiple fact tables share dimension tables, viewed as a collection of stars, therefore called galaxy schema or fact constellation
- **Modeling data warehouses: measures:**
  - distributive: if the result derived by applying the function to *n* aggregate values is the same as that derived by applying the function on all the data without partitioning.
    - E.g., count(), sum(), min(), max().

- algebraic: if it can be computed by an algebraic function with *M* arguments (where *M* is a bounded integer), each of which is obtained by applying a distributive aggregate function.
    - E.g., avg(), min_N(), standard_deviation().
- holistic: if there is no constant bound on the storage size needed to describe a subaggregate.
    - E.g., median(), mode(), rank().

Typical OLAP Operations

- Roll up (drill-up): summarize data
    - *by climbing up hierarchy or by dimension reduction*
- Drill down (roll down): reverse of roll-up
    - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
- Slice and dice:
    - *project and select*
- Pivot (rotate):
    - *reorient the cube, visualization, 3D to series of 2D planes.*
- Other operations
    - *drill across: involving (across) more than one fact table*
    - *drill through: through the bottom level of the cube to its back-end relational tables (using SQL)*

- **Design of Data Warehouse: A Business Analysis Framework**
- Ø Four views regarding the design of a data warehouse
- § Top-down view
- ● allows selection of the relevant information necessary for the data warehouse
- § Data source view
- ● exposes the information being captured, stored, and managed by operational systems
- § Data warehouse view
- ● consists of fact tables and dimension tables
- § Business query view
- ● sees the perspectives of data in the warehouse from the view of end-user

- **Data Warehouse Design Process**
- Ø Top-down, bottom-up approaches or a combination of both
- § Top-down: Starts with overall design and planning (mature)
- § Bottom-up: Starts with experiments and prototypes (rapid)
- Ø From software engineering point of view
- § Waterfall: structured and systematic analysis at each step before proceeding to the next
- § Spiral: rapid generation of increasingly functional systems, short turn around time, quick turn around
- Ø Typical data warehouse design process
- § Choose a business process to model, e.g., orders, invoices, etc.
- § Choose the *grain* (*atomic level of data*) of the business process
- § Choose the dimensions that will apply to each fact table record
- § Choose the measure that will populate each fact table record

Three Data Warehouse Models

- Enterprise warehouse
  - collects all of the information about subjects spanning the entire organization
- Data Mart
  - a subset of corporate-wide data that is of value to a specific groups of users. Its scope is confined to specific, selected groups, such as marketing data mart
    - Independent vs. dependent (directly from warehouse) data mart
- Virtual warehouse
  - A set of views over operational databases
  - Only some of the possible summary views may be materialized

Data Warehouse Back-End Tools and Utilities

- Data extraction:
  - get data from multiple, heterogeneous, and external sources
- Data cleaning:
  - detect errors in the data and rectify them when possible
- Data transformation:
  - convert data from legacy or host format to warehouse format

- Load:
    - sort, summarize, consolidate, compute views, check integrity, and build indicies and partitions
- Refresh
    - propagate the updates from the data sources to the warehouse

Metadata Repository

- Meta data is the data defining warehouse objects. It has the following kinds
    - Description of the structure of the warehouse
        - schema, view, dimensions, hierarchies, derived data defn, data mart locations and contents
    - Operational meta-data
        - data lineage (history of migrated data and transformation path), currency of data (active, archived, or purged), monitoring information (warehouse usage statistics, error reports, audit trails)
    - The algorithms used for summarization
    - The mapping from operational environment to the data warehouse
    - Data related to system performance
        - warehouse schema, view and derived data definitions
    - Business data
        - business terms and definitions, ownership of data, charging policies

Data Warehouse Usage

- Three kinds of data warehouse applications
    - Information processing
        - supports querying, basic statistical analysis, and reporting using crosstabs, tables, charts and graphs

- Analytical processing
  - multidimensional analysis of data warehouse data
  - supports basic OLAP operations, slice-dice, drilling, pivoting
- Data mining
  - knowledge discovery from hidden patterns
  - supports associations, constructing analytical models, performing classification and prediction, and presenting the mining results using visualization tools.

- Differences among the three tasks

PPT:

What is Data Warehouse?

Ø Defined in many different ways, but not rigorously.

§ A decision support database that is maintained separately from the organization's operational database

§ Support information processing by providing a solid platform of consolidated, historical data for analysis.

Ø "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process."—W. H. Inmon

Ø Data warehousing:

§ The process of constructing and using data warehouses

Data Warehouse: Subject-Oriented

Ø Organized around major subjects, such as customer, product, sales

Ø Focusing on the modeling and analysis of data for decision makers, not on daily operations or transaction processing

Ø Provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process

Data Warehouse—Integrated

Ø Constructed by integrating multiple, heterogeneous data sources

§ relational databases, flat files, on-line transaction records

Ø Data cleaning and data integration techniques are applied.

§ Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources

§ When data is moved to the warehouse, it is converted.

Data Warehouse—Time Variant

Ø The time horizon for the data warehouse is significantly longer than that of operational systems

§ Operational database: current value data

§ Data warehouse data: provide information from a historical perspective (e.g., past 5-10 years)

Ø Every key structure in the data warehouse

§ Contains an element of time, explicitly or implicitly

§ But the key of operational data may or may not contain "time element"

Data Warehouse—Nonvolatile

Ø A physically separate store of data transformed from the operational environment

Ø Operational update of data does not occur in the data warehouse environment

§ Does not require transaction processing, recovery, and concurrency control mechanisms

§ Requires only two operations in data accessing:

● initial loading of data and access of data

Data Warehouse vs. Operational DBMS

Ø OLTP (on-line transaction processing)

§ Major task of traditional relational DBMS

§ Day-to-day operations: purchasing, inventory, banking, manufacturing, payroll, registration, accounting, etc.

Ø OLAP (on-line analytical processing)

§ Major task of data warehouse system

§ Data analysis and decision making

Ø Distinct features (OLTP vs. OLAP):

§ User and system orientation: customer vs. market

§ Data contents: current, detailed vs. historical, consolidated

§ Database design: ER + application vs. star + subject

§ View: current, local vs. evolutionary, integrated

§ Access patterns: update vs. read-only but complex queries

	OLTP	OLAP
users	clerk, IT professional	knowledge worker
function	day to day operations	decision support
DB design	application-oriented	subject-oriented
data	current, up-to-date detailed, flat relational isolated	historical, summarized, multidimensional integrated, consolidated
usage	repetitive	ad-hoc
access	read/write index/hash on prim. key	lots of scans
unit of work	short, simple transaction	complex query
# records accessed	tens	millions
#users	thousands	hundreds
DB size	100MB-GB	100GB-TB
metric	transaction throughput	query throughput, response

Why Separate Data Warehouse?

Ø High performance for both systems

§ DBMS— tuned for OLTP: access methods, indexing, concurrency control, recovery

§ Warehouse—tuned for OLAP: complex OLAP queries, multidimensional view, consolidation

Ø Different functions and different data:

§ missing data: Decision support requires historical data which operational DBs do not typically maintain

§ data consolidation: DS requires consolidation (aggregation, summarization) of data from

heterogeneous sources

§ data quality: different sources typically use inconsistent data representations, codes and formats which have to be reconciled

Ø Note: There are more and more systems which perform OLAP analysis directly on relational databases (Agile DW)

Design of Data Warehouse: A Business Analysis Framework

Ø Four views regarding the design of a data warehouse

§ Top-down view

● allows selection of the relevant information necessary for the data warehouse

§ Data source view

● exposes the information being captured, stored, and managed by operational systems

§ Data warehouse view

● consists of fact tables and dimension tables

§ Business query view

● sees the perspectives of data in the warehouse from the view of end-user

Data Warehouse Design Process

Ø Top-down, bottom-up approaches or a combination of both

§ Top-down: Starts with overall design and planning (mature)

§ Bottom-up: Starts with experiments and prototypes (rapid)

Ø From software engineering point of view

§ Waterfall: structured and systematic analysis at each step before proceeding to the next

§ Spiral: rapid generation of increasingly functional systems, short turn around time, quick turn around

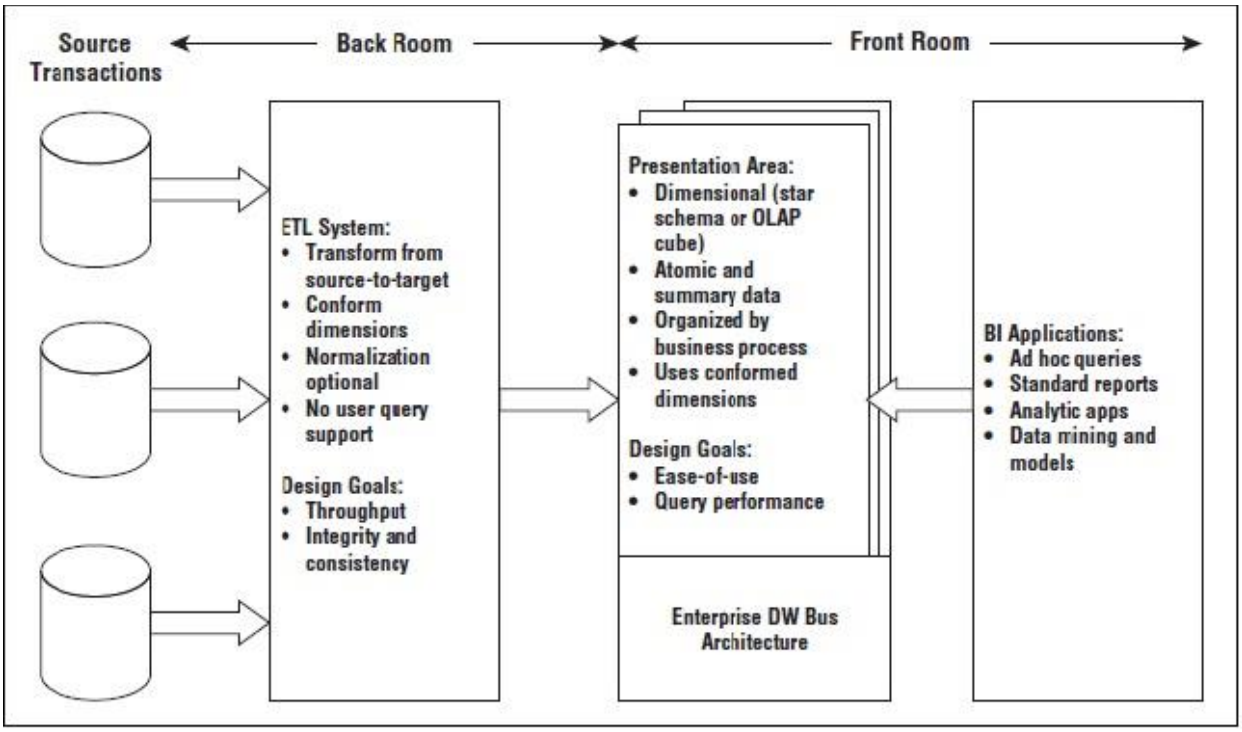
Ø Typical data warehouse design process

§ Choose a business process to model, e.g., orders, invoices, etc.

§ Choose the grain (atomic level of data) of the business process

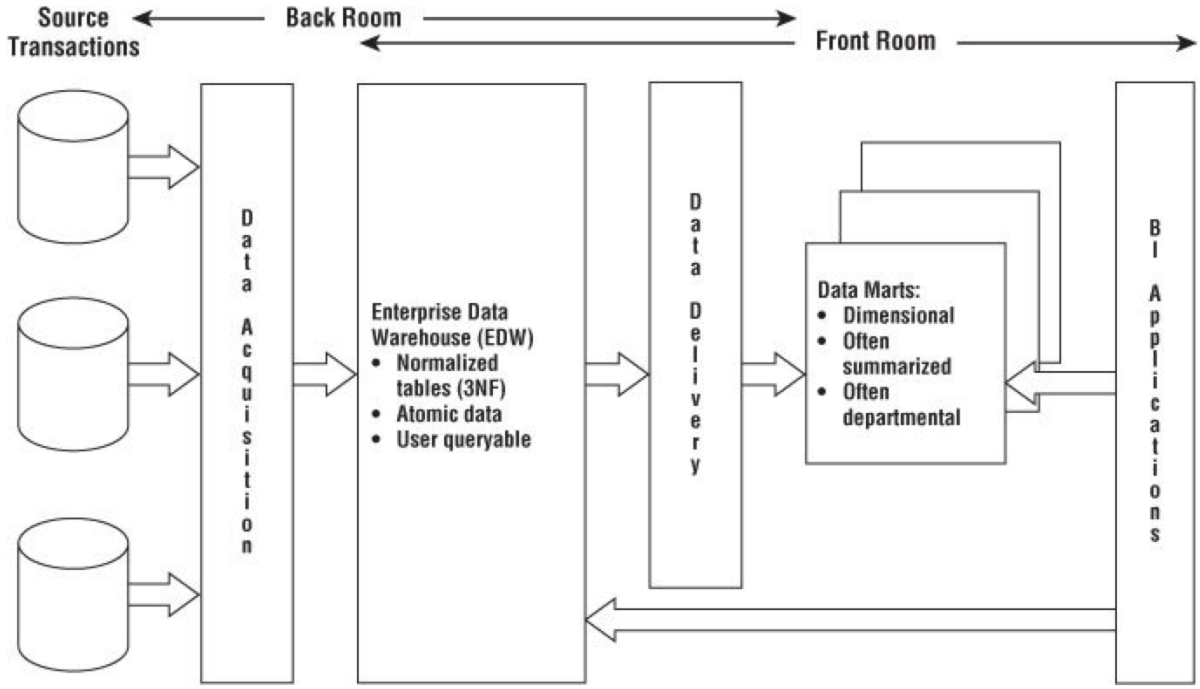
§ Choose the dimensions that will apply to each fact table record

§ Choose the measure that will populate each fact table record



Core Elements Kimball DW/BI Architecture.

William Inman's Architecture



Three Data Warehouse Models

Ø Enterprise warehouse

§ collects all of the information about subjects spanning the entire organization

Ø Data Mart

§ a subset of corporate-wide data that is of value to a specific groups of users. Its scope is confined to specific, selected groups, such as marketing data mart

● Independent (landing zone) vs. dependent (directly from warehouse) data mart

Ø Virtual warehouse

§ A set of views over operational databases

§ Only some of the possible summary views may be materialized

Data Warehouse Back-End Tools and Utilities

Ø Data extraction

§ get data from multiple, heterogeneous, and external sources

Ø Data cleaning

§ detect errors in the data and rectify them when possible

Ø Data transformation

§ convert data from legacy or host format to warehouse format

Ø Load

§ sort, summarize, consolidate, compute views, check integrity, and build indicies and partitions

Ø Refresh

§ propagate the updates from the data sources to the warehouse (Disruptive, Incremental, Real-time, etc.)

Metadata Repository

Ø Meta data is the data defining warehouse objects. It stores:

Ø Description of the structure of the data warehouse (Technical spec)

§ schema, view, dimensions, hierarchies, derived data definition, data mart locations and contents

Ø Operational meta-data

§ data lineage (history of migrated data and transformation path), currency of data (active, archived, or purged), monitoring information (warehouse usage statistics, error reports, audit trails)

Ø The algorithms used for summarization

Ø The mapping from operational environment to the data warehouse

Ø Business data

§ business terms and definitions, ownership of data, charging policies

Data Warehouse Usage

Ø Three kinds of data warehouse applications

§ Information processing

● supports querying, basic statistical analysis, and

reporting using crosstabs, tables, charts and graphs

§ Analytical processing

● multidimensional analysis of data warehouse data

● supports basic OLAP operations, slice-dice, drilling, pivoting

§ Data mining

● knowledge discovery from hidden patterns

● supports associations, constructing analytical models, performing classification and prediction, and presenting the mining results using visualization tools

Week 02 Dimensional Modeling: Fundamental Concepts

Reading:

#1 Mistake in Data Modeling

- Modeling something to take on human characteristics or characteristics of our world

Data Model Anthropomorphism 拟人论；拟人观

- We sometimes create objects in our Data Models are they exist in the real world, not in the applications

Over Engineering

- Additional flexibility that is not required does not simplify the solution, it overly complicates the solution

Two design methods

- Dimensional
  - Dimensional modeling always uses the concepts of facts (measures), and dimensions (context). Facts are typically (but not always) numeric values that can be aggregated, and dimensions are groups of hierarchies and descriptors that define the facts
- Dimensional Analysis
  - Star Schema/Snowflake
  - Database is optimized for analytical processing. (OLAP)
  - Facts and Dimensions optimized for retrieval
    - Facts – Business events – Transactions
    - Dimensions – context for Transactions
      - People
      - Accounts
      - Products
      - Date

Normal forms

- **1st** - Under first normal form, all occurrences of a record type must contain the same number of fields.
- **2nd** - Second normal form is violated when a non-key field is a fact about a subset of a key. It is only relevant when the key is composite
- **3rd** - Third normal form is violated when a non-key field is a fact about another non-key field

Dimensional

- Dimensional Analysis
  - Star Schema/Snowflake
  - Database is optimized for analytical processing. (OLAP)
  - Facts and Dimensions optimized for retrieval
    - Facts – Business events – Transactions
    - Dimensions – context for Transactions
      - People
      - Accounts
      - Products
      - Date

Relational

- 3 Dimensions
- Spatial Model
  - No historical components except for transactional tables
- Relational – Models the one truth of the data
  - One account '11'
  - One person 'Terry Bunio'
  - One transaction of '$100.00' on April 10th

Inmon-ians

- Top-down
  - Corporate Information Factory
  - Operational systems feed the Data Warehouse
  - Enterprise Data Warehouse is a corporate relational model that Data Marts are sourced from
  - Enterprise Data Warehouse is the source of Data Marts

Kimball-lytes

- Bottom-up - incremental
  - Operational systems feed the Data Warehouse
  - Data Warehouse is a corporate dimensional model that Data Marts are sourced from
  - Data Warehouse is the consolidation of Data Marts
  - Sometimes the Data Warehouse is generated from Subject area Data Marts

The gist…

- Kimball's approach is easier to implement as you are dealing with separate subject areas, but can be a nightmare to integrate
- Inmon's approach has more upfront effort to avoid these consistency problems, but takes longer to implement.

Fact Tables

- Contains the measurements or facts about a business process
- Are thin and deep
- Usually is:
  - Business transaction
  - Business Event
- The grain of a Fact table is the level of the data recorded.
- Contains the following elements
  - Primary Key - Surrogate

  - Timestamp
  - Measure or Metrics
    - Transaction Amounts
  - Foreign Keys to Dimensions
  - Degenerate Dimensions
    - Transaction indicators or Flags
- Types of Measures
  - Additive - Measures that can be added across any dimensions.
    - Amounts
  - Non Additive - Measures that cannot be added across any dimension.
    - Rates
  - Semi Additive - Measures that can be added across some dimensions.
    - Don't have a good example
- Types of Fact tables
  - **Transactional** - A transactional table is the most basic and fundamental. The grain associated with a transactional fact table is usually specified as "one row per line in a transaction".
  - **Periodic snapshots** - The periodic snapshot, as the name implies, takes a "picture of the moment", where the moment could be any defined period of time.
  - **Accumulating snapshots** - This type of fact table is used to show the activity of a process that has a well-defined beginning and end, e.g., the processing of an order. An order moves through specific steps until it is fully processed. As steps towards fulfilling the order are completed, the associated row in the fact table is updated.

Special Fact Tables

- Degenerate Dimensions
  - Degenerate Dimensions are Dimensions that can typically provide additional context about a Fact
    - For example, flags that describe a transaction

- Degenerate Dimensions can either be a separate Dimension table or be collapsed onto the Fact table
  - My preference is the latter
- If Degenerate Dimensions are not collapsed on a Fact table, they are called Junk Dimensions and remain a Dimension table
- Junk Dimensions can also have attributes from different dimensions
  - Not recommended

Dimension Tables

- Unlike *fact* tables, *dimension* tables contain descriptive attributes that are typically textual fields
- These attributes are designed to serve two critical purposes:
  - query constraining and/or filtering
  - query result set labeling.
- Shallow and Wide
- Usually corresponds to entities that the business interacts with
  - People
  - Locations
  - Products
  - Accounts
- Contains the following elements
  - Primary Key – Surrogate
  - Business Natural Key
    - Person ID
  - Effective and Expiry Dates
  - Descriptive Attributes
    - Includes de-normalized reference tables

Time Dimension

- All Dimensional Models need a time component
- This is either a:
  - Separate Time Dimension (recommended)
  - Time attributes on each Fact Table

Behavioural Dimensions

- A Dimension that is computed based on Facts is termed a behavioural dimension

Mini-Dimensions

- Splitting a Dimension up due to the activity of change for a set of attributes
- Helps to reduce the growth of the Dimension table

Slowly Changing Dimensions

- Type 1 – Overwrite the row with the new values and update the effective date
  - Pre-existing Facts now refer to the updated Dimension
  - May cause inconsistent reports
- Type 2 – Insert a new Dimension row with the new data and new effective date
  - Update the expiry date on the prior row
- Don't update old Facts that refer to the old row
  - Only new Facts will refer to this new Dimension row
- Type 2 Slowly Changing Dimension maintains the historical context of the data
- A type 2 change results in multiple dimension rows for a given natural key
- Type 3 – The Dimension stores multiple versions for the attribute in question
- This usually involves a current and previous value for the attribute
- When a change occurs, no rows are added but both the current and previous attributes are updated
- Like Type 1, Type 3 does not retain full historical context

Type 1/Type 2 Hybrid

- Most common hybrid
- Used when you need history AND the current name for some types of statutory reporting

Frozen Attributes

- Sometimes it is required to freeze some attributes so that they are not Type 1, Type 2, or Type 3
- Usually for audit or regulatory requirements

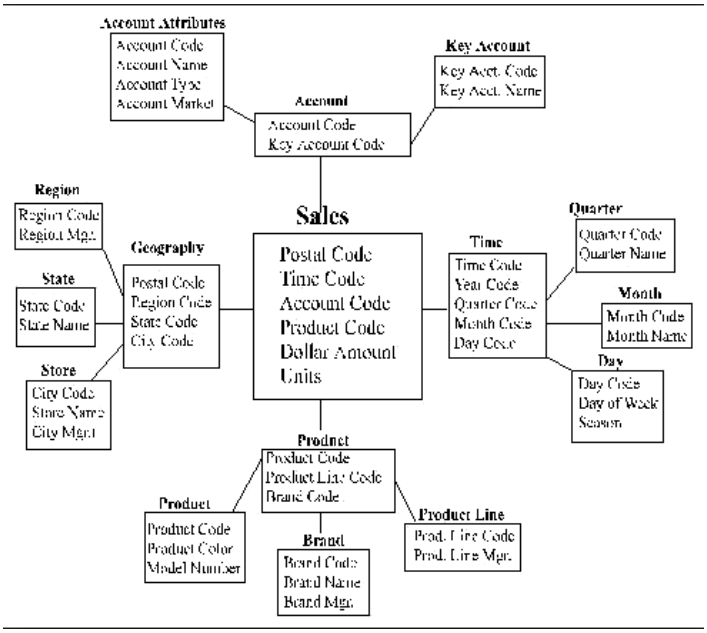
Conforming Dimension

- A Dimension is said to be conforming if:
  - A conformed dimension is a set of data attributes that have been physically referenced in multiple database tables using the same key value to refer to the same structure, attributes, domain values, definitions and concepts. A conformed dimension cuts across many facts.
- Dimensions are conformed when they are either exactly the same (including keys) or one is a perfect subset of the other.

Complex Concept Introduction

- Snowflake vs Star Schema
- Multi-Valued Dimensions and Bridges
- Multi-Valued Attributes
- Factless Facts
- Recursive Hierarchies

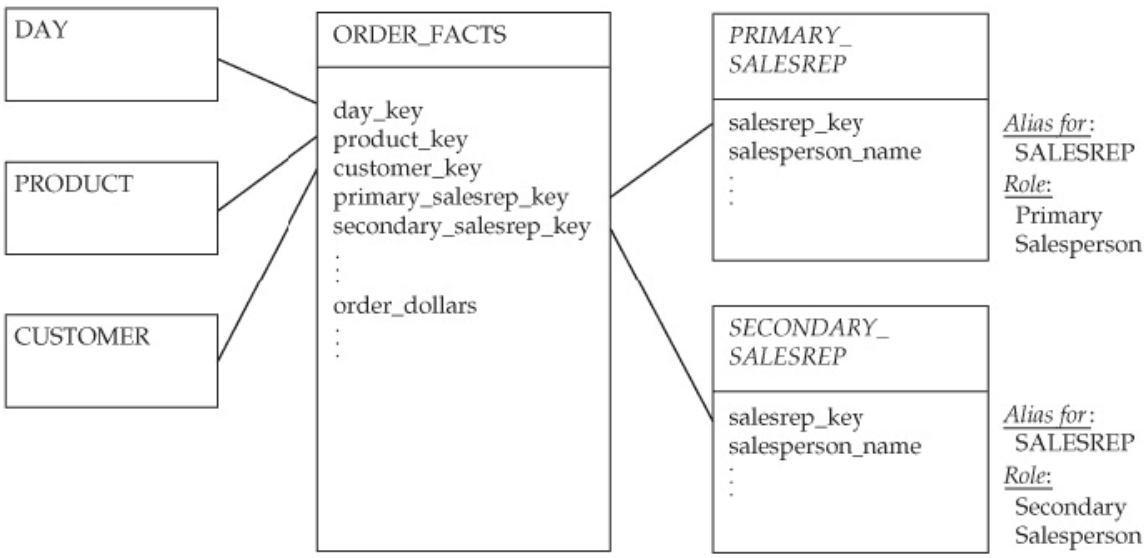
Snowflake vs Star Schema

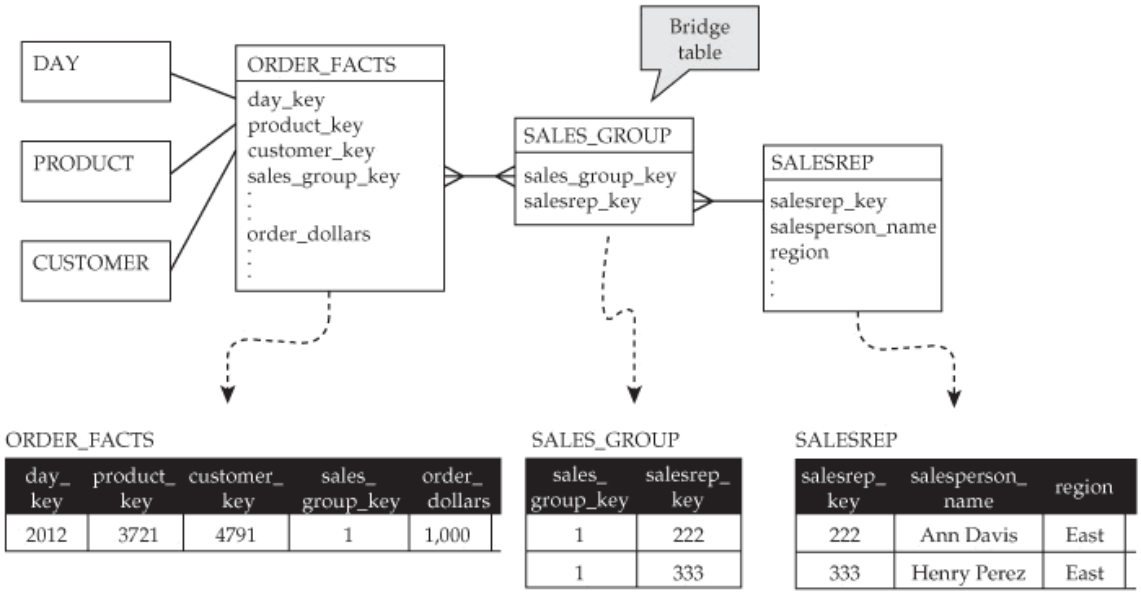


- These extra table are termed outriggers
- They are used to address real world complexities with the data
  - Excessive row length
  - Repeating groups of data within the Dimension
- I will use outriggers in a limited way for repeating data

Multi-Valued Dimensions

- Multi-Valued Dimensions are when a Fact needs to connect more than once to a Dimension
  - Primary Sales Representative
  - Secondary Sales Representative
- Two possible solutions
  - Create copies of the Dimensions for each role
  - Create a Bridge table to resolve the many to many relationship





ORDER_FACTS

day_ key	product_ key	customer_ key	sales_ group_key	order_ dollars
2012	3721	4791	1	1,000

SALES_GROUP

sales_ group_key	salesrep_ key
1	222
1	333

SALESREP

salesrep_ key	salesperson_ name	region
222	Ann Davis	East
333	Henry Perez	East

Bridge Tables

- Bridge Tables can be used to resolve any many to many relationships
- This is frequently required with more complex data areas
- These bridge tables need to be considered a Dimension and they need to use the same Slowly Changing Dimension Design as the base Dimension
  - My Recommendation

Multi-Valued Attributes

- In some cases, you will need to keep multiple values for an attribute or sets of attributes
- Three solutions
  - Outriggers or Snowflake (1:M)
  - Bridge Table (M:M)
  - Repeat attributes on the Dimension
    - Simplest solution but can be hard to query and causes long record length

Factless Facts

- Fact table with no metrics or measures
- Used for two purposes:

  - Records the occurrence of activities. Although no facts are stored explicitly, these events can be counted, producing meaningful process measurements.
  - Records significant information that is not part of a business activity. Examples of conditions include eligibility of people for programs and the assignment of Sales Representatives to Clients

Hierarchies and Recursive Hierarchies

- Solution involves defining Dimension tables to record the Hierarchy with a special solution to address the Slowly Changing Dimension Hierarchy
- Any change in the Hierarchy can result in needing to duplicate the Hierarchy downstream

Process

- Start with your simplest Dimension and Fact tables and define the Natural Keys for them
  - i.e. People, Product, Transaction, Time
- De-Normalize Reference tables to Dimensions (And possibly Facts based on how large the Fact tables will be)
  - I place both codes and descriptions on the Dimension and Fact tables
- Look to De-normalize other tables with the same Cardinality into one Dimension
  - Validate the Natural Keys still define one row
- Don't force entities on the same Dimension
  - Tempting but you will find it doesn't represent the data and will cause issues for loading or retrieval
  - Bridge table or mini-snowflakes are not bad
    - I don't like a deep snowflake, but shallow snowflakes can be appropriate
    - Don't fall into the Star-Schema/Snowflake Holy War – Let your data define the solution

Top 10

1. Copy the design for the Time Dimension from the Web. Lots of good solutions with scripts to prepopulate the dimension
2. Make all your attributes Not-Null. This makes Self-Service Report writing easy

3. Create a single Surrogate Primary Key for Dimensions – This will help to simplify the design and table width
    - These FKs get created on Fact tables !
4. Never reject a record
    - Create an Dummy Invalid record on Each Dimension. Allows you to store a Fact record when the relationship is missing
5. Choose a Type 2 Slowly Changing Dimension as your default
6. Use Effective and Expiry dates on your Dimensions to allow for maximum historical information
    - If they are Type 2!
7. SSIS 2012 has some built-in functionality for processing Slowly Changing Dimensions – Check it out!
8. Add "Current_ind" and "Dummy_ind" attributes to each Dimension to assist in Report writing
9. Iterate, Iterate, Iterate
10. Read this book

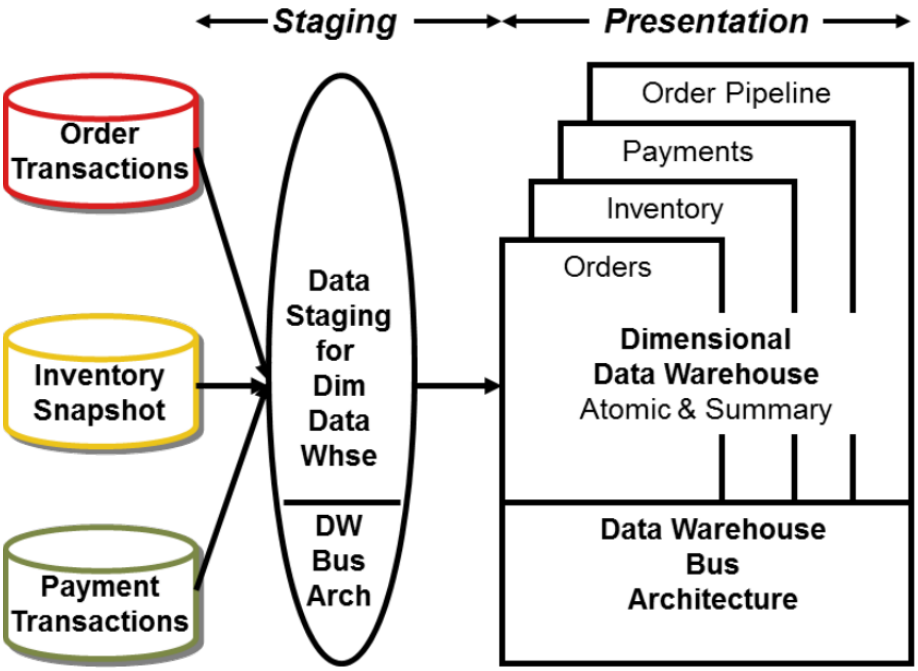
Reading: Differences of Opinion - KimballGroup

The Kimball bus architecture and the Corporate Information Factory: What are the fundamental differences?

get data → staging (a consists of extract-transform-load (ETL) processes and support)

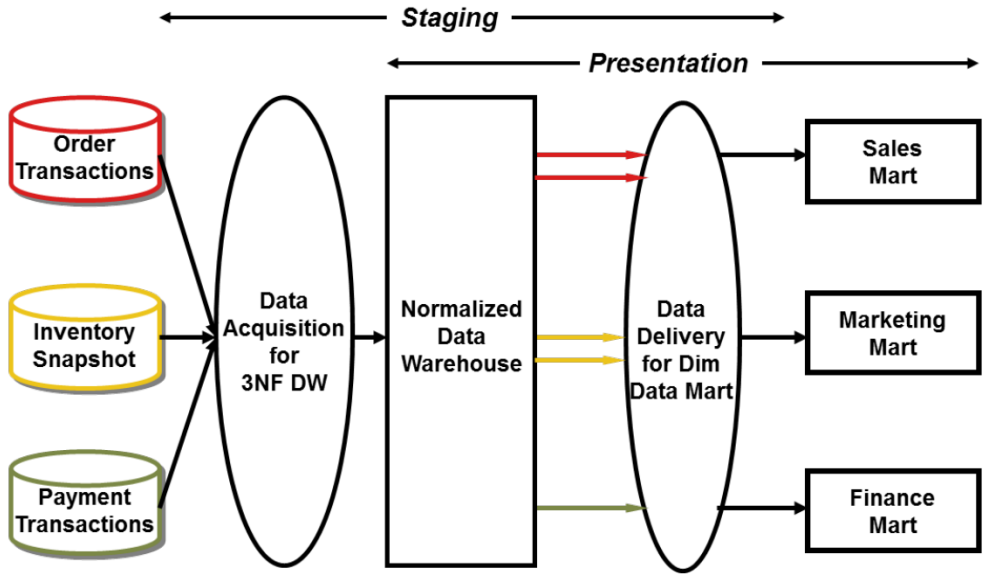
make it easily accessible to decision makers → presentation

Kimball Bus Architecture: (Dimensional data warehouse)



Dimensional models are built by business process (corresponding to a business measurement or event).

Corporate Information Factory: (known as EDW approach; Normalized data warehouse with summary dimensional marts (CIF))



There are two fundamental differentiators between the CIF and Kimball approaches. The first concerns the need for a normalized data structure before loading the dimensional models. The second primary difference between the two approaches is the treatment of atomic data. The CIF

says atomic data should be stored in the normalized data warehouse. The Kimball approach says atomic data must be dimensionally structured.

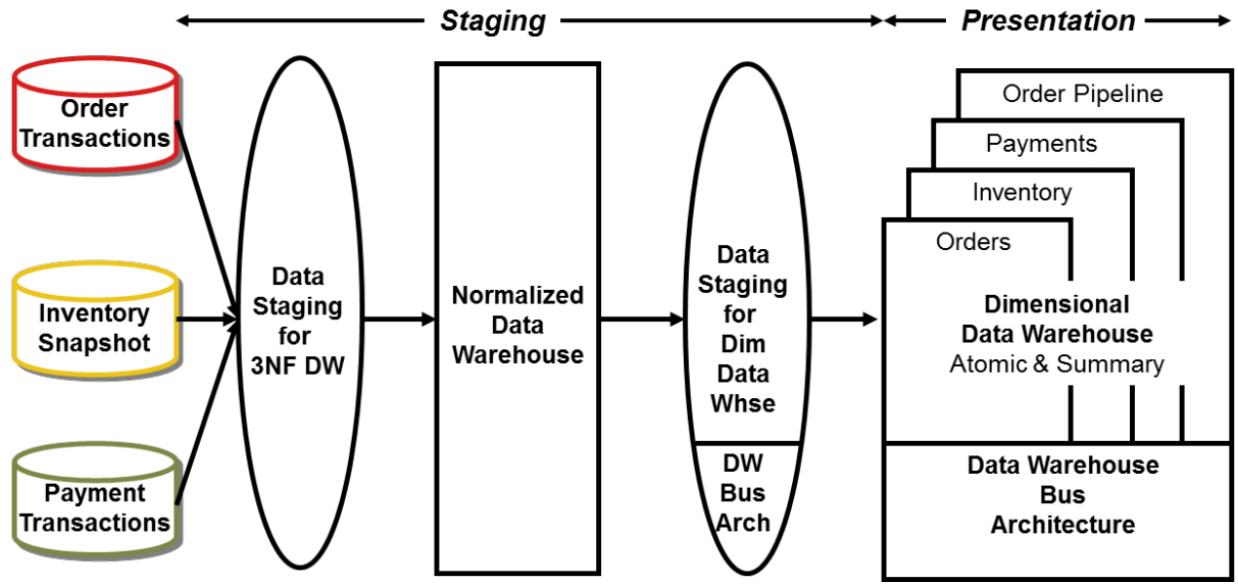


Figure 3 Hybrid of normalized data warehouse and dimensional data warehouse (不好)

In our opinion, the success of any data warehouse is measured by the business's acceptance of the analytic environment and their benefits realized from it. You should choose the data warehouse architecture that best supports these success criteria, regardless of the label.

PPT:

Business-Driven Goals of DW/BI

Ø Business needs we hear frequently:

§ "We collect tons of data, but we can't access it."

§ "We need to slice and dice the data every which way."

§ "Business people need to get at the data easily."

§ "Just show me what is important."

§ "We spend entire meetings arguing about who has the right numbers rather than making decisions."

§ "We want people to use information to support more fact-based decision making."

Goals of DW/BI:

Ø Make information accessible and understandable to the business users

Ø Present information consistently in a timely manner

Ø Ensure DW/BI deliverables accepted by business community to support their decision making

DW/BI Deliverables:

Dimensional models

§ Relational star schemas

- Fact tables for measurements
- Dimension tables for descriptors

§ Multidimensional online analytical processing (OLAP) cubes § Goals:

- Easy to use
- Fast queries

Ø Business intelligence applications

§ Reporting tools, analytic tools, etc.

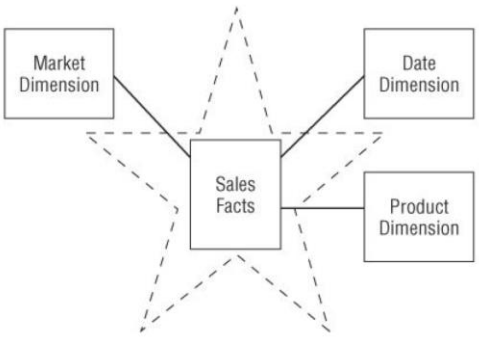
Star Schema VS. OLAP Cubes

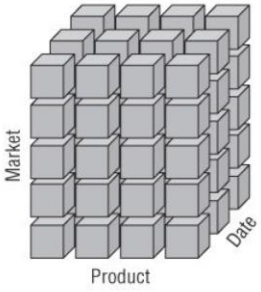
Ø At a logical level, there is no difference

Ø It is a matter of physical database implementation.

Ø Star schema is implemented in a relational database and is queried through SQL

Ø OLAP Cubes (multidimensional databases) are implemented for extreme performance and are queried through MDX.





Ø The star schema can store large amounts of detailed data.

Ø OLAP Cubes provide higher performance with precalculated summary data.

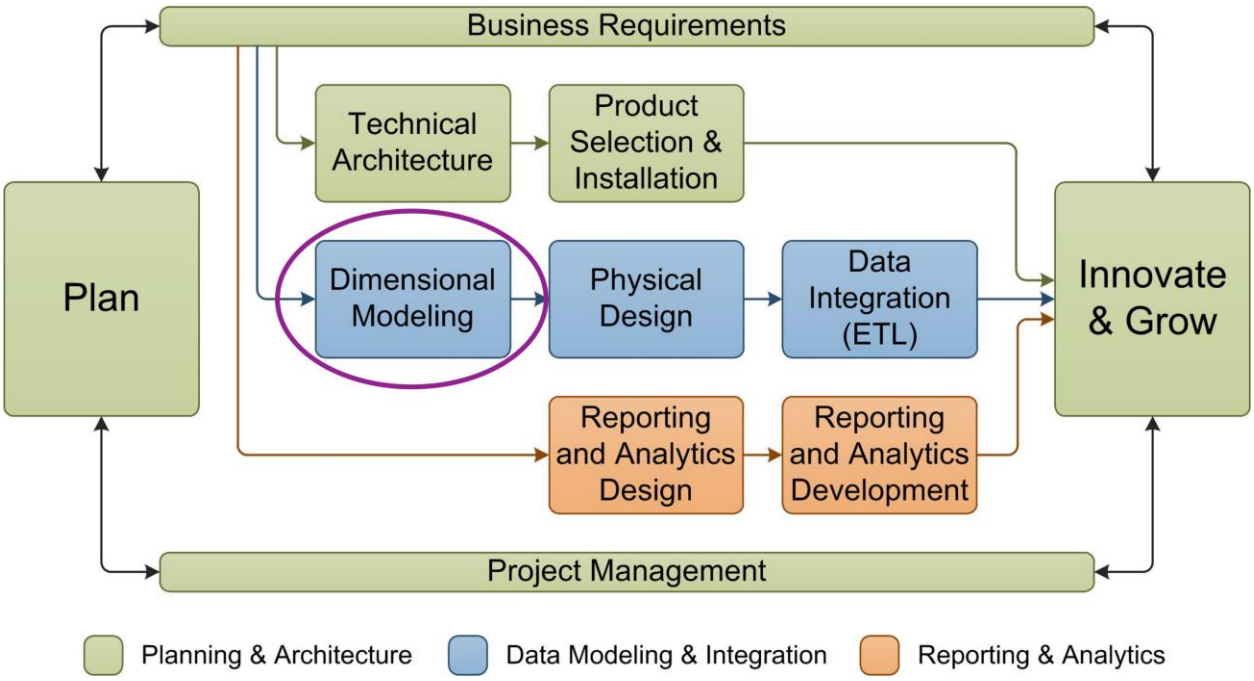
Ø In general, OLAP cubes are populated from the star schema

• Kimball focuses on the star schema rather than the OLAP cubes

• Star schema usually has 15 dimensions

• OLAP usually has 8-10 dimensions

Kimball Lifecycle Approach



Dimensional Modeling -- Fundamental Concepts: Make it Simple

Ø 3NF: Immensely useful in operational processing

Ø However, it's too complicated for users to use. It looks like a subway map

Ø Unpredictable queries cause performance Problems

Ø Dimensional modeling came to a rescue in the presentation layer

Ø Dimensional modeling provides

§ Understandability

§ Query performance

§ Resilience to change 适应力；弹性

Ø Making it simple via Denormalization

Ø Widely accepted model for presenting analytic data

Ø Techniques for making database simple through denormalization

Ø "We sell products in various stores in different regions and measure our performance over time": emphasis on

§ Products, Stores, Regions, Time

§ Performance: sales volume, profit →Make a simple modeling

Ø Simple modeling is important

§ Resist temptation to over-engineer

Dimensions – Physical Table Elements

Ø Contain descriptive attributes that are typically textual fields

Ø Shallow and wide

Ø Corresponds to entities that business interacts with

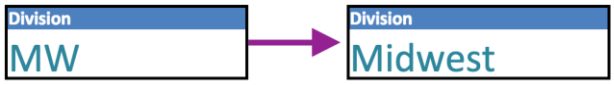
§ Customer, Employee, Products, Accounts

Ø Single column PK (typically a surrogate key) with a single column natural key

Descriptive Dimension Attributes

Ø Describe "Who, What, Where, When and Why"

Ø Consists of words rather than cryptic abbreviation



Ø DW is only as good as the dimension attributes

Ø Embedded meaning within codes as separate attributes

Ø Denormalized many-to-one hierarchies

Ø Operational natural business key as attribute, not primary key

Date Dimension

Ø All dimensional models need a time component

Ø Generally the calendar date dimension with the granularity of a single day

Ø Surprisingly has many attributes

§ Holidays, work days, fiscal periods, week numbers, last day of month flags, etc.

Slowly Changing Dimensions (SCD)

Ø Dimensions contains relatively static data such as

§ Geo locations, customers, or products

Ø Data in the dimensions change slowly and in unpredictable time

Ø This can cause referential 有参考内容的 integrity issue between a fact and dimension tables (e.g. an employee left the job)

Ø SCD is a mechanism to deal with these changes in dimensions

SCD Type 1

Ø Overwrite the old with new data

§ Pre-existing facts now refer to the updated Dimension § May cause inconsistent reports

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	IL

SCD Type 2

Ø Insert a new dimension row with the new data and new effective start date

Ø Update the effective end date on the prior row

Ø Maintains the historical context of the data

Ø Fact tables reference to the correct snapshot information of a SCD type 2 dimension

Ø Results in multiple dimension rows for a given natural key

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
123	ABC	Acme Supply Co	CA	2000-01-01	2004-12-21
124	ABC	Acme Supply Co	IL	2004-12-22	NULL

Fact Tables

Ø Contains the measurements or facts about a business process

Ø Are thin and deep

Ø Usually is:

§ Business transaction

§ Business Event

Ø The grain of a fact table is the level of the data recorded

Fact Table – Grain

Ø The grain of a fact table: what a fact table record exactly represents

Ø Examples:

§ an individual line item on a customer's retail sales ticket as measured by a scanner device

§ an individual transaction against an insurance policy

§ a line item on a bill received from a doctor

§ an individual boarding pass used by someone making an airplane flight

Fact Tables – Physical Table Elements

Ø Contains the following elements

§ Primary key – surrogate key (optional)

§ Foreign keys to dimensions

§ Degenerate dimensions

- Transaction indicators or flags

§ Measure or metrics

- Transaction amounts

Fact Table - Types of Measures

Ø Additive 加添的 facts - Measures that can be added across any dimensions.

§ Amounts

Ø Non additive facts - Measures that cannot be added across any dimension.

§ Rates

Ø Semi additive facts - Measures that can be added across some dimensions.

§ Balances

Fact Tables - Types of Tables

Ø Transactional - A transactional table is the most basic and fundamental. The grain associated with a transactional fact table is usually specified as "one row per line in a transaction".

Ø Periodic snapshots - The periodic snapshot, as the name implies, takes a "picture of the moment", where the moment could be any defined period of time.

Ø Accumulating snapshots - This type of fact table is used to show the activity of a process that has a well-defined beginning and end, e.g., the processing of an order. An order moves through specific steps until it is fully processed. As steps towards fulfilling the order are completed, the associated row in the fact table is updated.

Dimensional Star Schema

Ø Atomic fact table per business process event (grain)

Ø Benefits:

§ Easy to understand

§ Better query performance from fewer joins

§ Resilience to change via extensibility (adding attributes, dimensions)

"A model that starts simple has a chance of remaining simple at the end of the design. A model that starts complicated surely will be overly complicated at the end, resulting in slow query performance and business user rejection." – Kimball

Independent Data Mart Architecture

Ø ETL System

§ Multiple independent extracts from the same source data

- Inefficient
- Potentially different transformation business rules applied to create each data mart

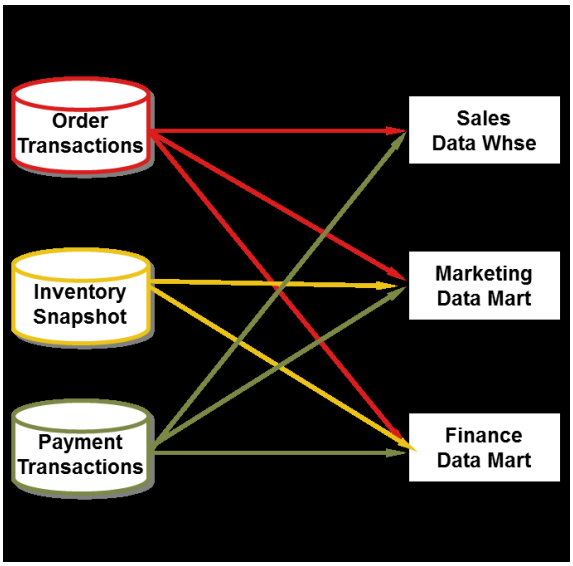
Ø Advantages: Shorter time to delivery

Ø Disadvantages:

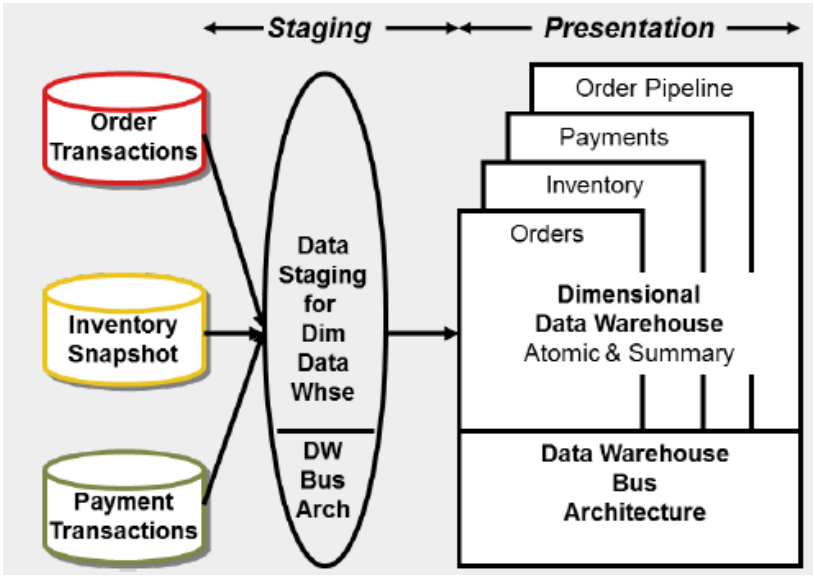
§ Inconsistency across data marts for the same source data

§ Undue burden on the source system

Ø This approach is NOT recommended



Kimball Architecture



Ø ETL System

§ "Back room" and is off limit to the business users

§ No user query Support

§ Design Goals

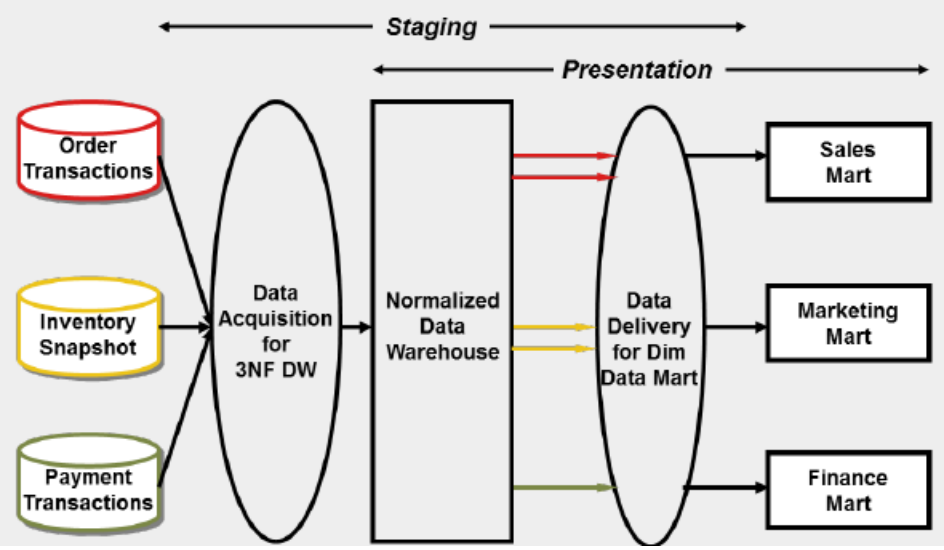
- Staging throughput, Integrity and Consistency

Ø Presentation Server

§ Stores dimensional models (star or OLAP cube)

§ Dimensional models typically contain atomic details in star schema; in addition, summary data may be stored in OLAP for query performance

Simplified Hub-and-Spoke Corp Info Factory (CIF) Architecture



Ø Similarities to Kimball Architecture

§ Deliver single version of the truth

§ Dimensional presentation in dimensional summary data

Ø Unique characteristics in CIF

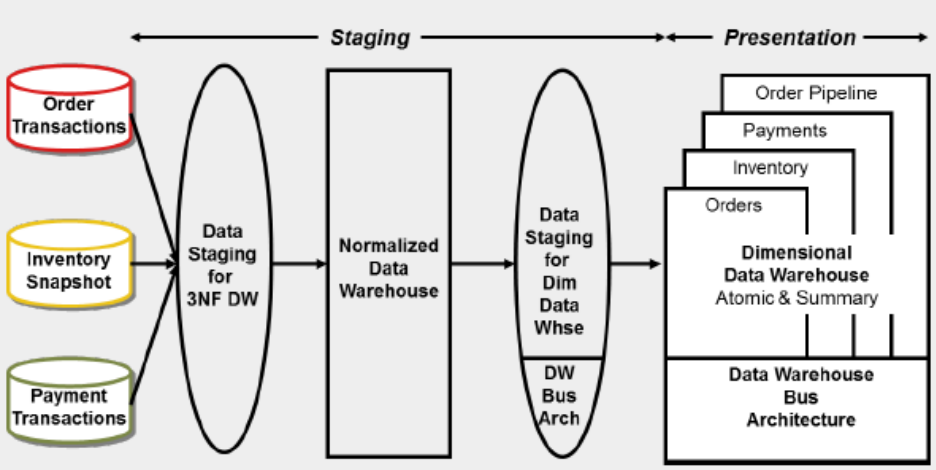
§ The first ETL to EDW (normalized tables) and the second ETL to the dimensional structure

§ Users are given access to the EDW

§ Dimensional structures are typically for summary data only

§ The EDW and dimensional data can be out of sync depending on the ETL's timing

Hybrid Architecture



Ø A normalized data warehouse from the CIF plus dimensional data warehouse of atomic detail

Ø Comes with high incremental costs and time lags

Ø It would work if there is already a normalized data warehouse built (e.g. SAP ERP system)

4-Step Dimensional Design Process

Ø Identity the Business Process

§ Source of "measurements"

Ø Identity the Grain

§ What does 1 row in fact table represent/mean?

§ Lowest atomic grain delivers most flexibilty

Ø Identity the Dimensions

§ Descriptive context, true to the grain

Ø Identify the Facts

§ Numeric additive measurements, true to the grain

Week 03 Dimensional Modeling: Basic Fact Table Techniques

Reading: Surrogate Keys

A surrogate key is an artificial or synthetic key that is used as a substitute for a natural key. They are just anonymous 无特色的 integers.

Let's be very clear: Every join between dimension tables and fact tables in a data warehouse environment should be based on surrogate keys, not natural keys.

In the production transaction processing environment, the meaning of a product key or a customer key is directly related to the record's content. In the data warehouse environment, however, a dimension key must be a generalization of what is found in the record.

Reasons:

1. SCD
2. the need to encode uncertain knowledge.
3. Save memory
4. is bound to improve join performance

Reading: Design Tip #81 Fact Table Surrogate Key

Although for the logical design of a fact table, the answer is no, surprisingly we find a fact table surrogate key may be helpful at the physical level.

Surrogate keys provide a number of important benefits for dimensions including avoiding reliance on awkward "smart" keys made up of codes from the dimension's source systems, protecting the data warehouse from changes in the source systems, enabling integration of data from disparate source systems, support for type 2 slowly changing dimensions attributes, space savings in the fact tables when these dimension keys are embedded in the fact tables as foreign keys, and improved indexing and query performance.

Assigning a surrogate key to the rows in a fact table is beneficial:

1. Sometimes the business rules of the organization legitimately allow multiple identical rows to exist for a fact table.

2. Certain ETL techniques for updating fact rows are only feasible if a surrogate key is assigned to the fact rows.

3. A similar ETL requirement is to determine exactly where a load job was suspended, either to resume loading or back put the job entirely.

Remember, surrogate keys for dimension tables are a great idea. Surrogate keys for fact tables are not logically required but can be very helpful in the back room ETL processing.

Reading: Fact Tables

Fact tables contain the fundamental measurements of the enterprise

Stay True to the Grain

The first and most important design step is declaring the fact table grain.

The grain is the description of the measurement event in the physical world that gives rise to a measurement.

The real purpose of the fact table is to be the repository of the numeric facts that are observed during the measurement event.

Notice that we don't store the price of the product being scanned because the price is nonadditive.

Build Up from the Lowest Possible Grain

Fact tables at the lowest grain are the most expressive because they have the most complete set of possible dimensions for that business process

Three Kinds of Fact Tables

If you stay true to the grain, then all of your fact tables can be grouped into just three types: transaction grain, periodic snapshot grain and accumulating snapshot grain (the three types are shown in Figure 1). In Figure 1, the dimensions are designated by FK (foreign key) and the numeric facts are italicized.

Figure 1. The Three Fact Table Types

Transaction Grain	Periodic Snapshot Grain	Accumulating Snapshot Grain
Date (FK)	Month (FK)	Order Date (FK)
Product (FK)	Account (FK)	Ship Date (FK)
Store (FK)	Branch (FK)	Delivery Date (FK)
Customer (FK)	Household (FK)	Payment Date (FK)
Cashier (FK)	*balance*	Return Date (FK)
Manager (FK)	*fees paid*	Warehouse (FK)
Promotion (FK)	*interest earned*	Customer (FK)
Weather (FK)	*transaction count*	Promotion (FK)
Basket (FK)		Order Status (FK)
time of day		*quantity*
quantity		*extended list price*
extended price		*discounts*
		extended net price

Reading: Design Tip #46: Another Look At Degenerate Dimensions

退化维度，看起来像是事实表的一个维度关键字，但实际上并没有对应的维度表，其中，事实表的粒度就是文档本身或文档中的一个分列项。

什么是退化维度（Degenerate Dimension,DD），就是那些看起来像是事实表的一个维度关键字，但实际上并没有对应的维度表，其中，事实表的粒度就是文档本身或文档中的一个分列项。 具体怎么理解呢？在传统的父子关系型数据库中，事务编号是事物标题记录的关键字，比如订单编号、发票编号，这样的纪录包含了诸如事务日期、供应商标示这样在总体上对事务有效的所有信息。但在给出的维度模型中，已经将这些令人感兴趣的标题信息抽取出来放到其它维度中去了。但这个事务编号仍然十分有用，因为它可以作为组关键字而将单个事务中处理的明细集中在一起。

尽管这个事务维度看起来是一个维度关键字，但当把事务维度所有的描述性项目进行剔出后，形成维度为空。诸如这种事务编号、固有的操作型票据编号，应该自然的放入事实表中，而不用连接到维度表。退化维度在事实表粒度表示单个事务或事务分列项目时是很常见的，因为它标示父实体的惟一标示。订单号，发票号与提货单编号等几乎总是以退化维度的形式出现在维度模型之中。

同时，退化维度在事实表主关键字方面也有一定作用。比如将订单事实表主关键字可以由退化的订单编号和产品组关键字组成。

A degenerate dimension (DD) acts as a dimension key in the fact table, however does not join to a corresponding dimension table because all its interesting attributes have already been placed in other analytic dimensions.

Degenerate dimensions commonly occur when the fact table's grain is a single transaction (or transaction line).

Transaction control header numbers assigned by the operational business process are typically degenerate dimensions, such as order, ticket, credit card transaction, or check numbers.

We typically don't implement a surrogate key for a DD.

Reading: Declaring the Grain

Declaring the grain means saying exactly what a fact table record represents. Remember that a fact table record captures a measurement. Example declarations of the grain include:

- An individual line item on a customer's retail sales ticket as measured by a scanner device
- An individual transaction against an insurance policy

- A line item on a bill received from a doctor
- A boarding pass used by someone on an airplane flight
- An inventory measurement taken every week for every product in every store.

In a properly executed dimensional design, the grain is first anchored to a clear business object (no pun intended) and a set of business rules. Then, the dimensions that implement that grain become obvious.

In summary, try to do your dimensional designs using the following four steps, in order:

1. Decide on your sources of data.

2. Declare the grain of the fact table (preferably at the most atomic level).

3. Add dimensions for "everything you know" about this grain.

4. Add numeric measured facts true to the grain.

Reading: Keep to the Grain in Dimensional Modeling

The grain of a fact table is the business definition of the measurement event that creates a fact record. The grain is exclusively determined by the physical realities of the source of the data.

PPT:

Fact Tables Revisited

Ø Contains the measurements or facts about a business process

Ø Stay true to the grain

Ø Build up from the lowest grain possible

Ø Use conformed facts (identical field name for identical technical definition)

Ø Three types of Fact tables

§ Transaction Fact

§ Periodic Snapshot Fact

§ Accumulating Snapshot Fact

Fact Tables Revisited: Physical Table Elements

Ø Contains the following elements

§ Primary key – surrogate key: not common (only if ETL tools require)

§ Foreign keys to dimensions

§ Degenerate dimensions

● Transaction indicators or flags

§ Measure or metrics

● Transaction amounts

Surrogate Key

Ø One of the basic elements of data warehouse design

Ø Every join between dimension tables and fact tables should be based on surrogate keys, not natural keys

Ø Essential for Slowly Changing Dimension Type 2

Ø Essential when no natural key is available

Ø Improves join performance and saves storage space

Three Types of Fact Tables: Transaction Fact

Ø The grain corresponds to a measurement taken at a single instant

§ e.g. The grocery store beep

Ø Unpredictably sparse or dense

Ø Can be enormous, with the largest containing many billions of records.

Three Types of Fact Tables: Periodic Snapshot Fact

Ø The grain corresponds to a predefined span of time

§ e.g. a financial reporting period

Ø All of the reporting entities appears in each snapshot

Ø Is predictably dense and can get large as well

§ e.g. 2 million accounts for 10 years' of monthly snapshot will generate 2.4 billion fact rows

Three Types of Fact Tables: Accumulating Snapshot Fact

Ø The grain corresponds to well-defined predictable processes

§ e.g. Order processing and college admissions

Ø Fact rows are revisited and overwritten as the process progresses through its steps from beginning to end

Ø Is much smaller than the other two types

	Periodic Snapshot	Transaction	Accumulating Snapshot
Time period represented	Regular predictable intervals	Point in time	Indeterminate time span, typically short lived
Grain	One row per period	One row per transaction event	One row per life
Table loads	Insert	Insert	Insert and update
Row updates	Not revisited	Not revisited	Revisited whenever activity
Date dimension	End-of-period	Transaction date	Multiple dates for standard milestones
Facts	Performance for predefined time interval	Transaction activity	Performance over finite time

Degenerate Dimension

A dimension key in the fact table that does not have its own dimension table, because all the interesting attributes have been placed in analytic dimension

Ø Does not join to a corresponding dimension

Ø Becomes a part of the fact table's primary key

Ø Commonly occurs when the fact table's grain is a single line item transaction

The Grain Revisited

Ø An event that creates a fact record

Ø Start at the lowest, most atomic grain

§ Atomic data is the most expressive data and versatile to unexpected requests

Ø Avoid mixed granularity

Ø A clear definition of grain provides rich information about dimensions

4-Step Dimensional Design Process

1. Identity the Business Process

    § Source of "measurements"

    § e.g. taking orders, invoicing, receiving payments, handling service calls, etc.

2. Identity the Grain

    § What does 1 row in fact table represent/mean?

    § Lowest atomic grain delivers most flexibility

3. Identity the Dimensions

    § Descriptive context, true to the grain

4. Identify the Facts

§ Numeric additive measurements, true to the grain

4-Step Dimensional Design Process: An Example

1. Identify Business Process

    § Claim Billing

2. Identify the Grain

    § A line item of a doctor's bill

3. Identify Dimensions

    § Date (of treatment)

    § Doctor (maybe called "provider")

    § Patient

    § Procedure

    § Primary Diagnosis

    § Location (presumably the doctor's office)

    § Billing Organization (an organization the doctor belongs to)

    § Responsible Party (either the patient, or the patient's legal guardian)

    § Primary Payer (often an insurance plan)

    § Secondary Payer (maybe the responsible party's spouse's insurance plan) and quite possibly others.

4. Identify the Facts

    § Billed Amount

Week 04 Dimensional Modeling: Basic Dimension Table Techniques

Reading: Design Tip #51: Latest Thinking On Time Dimension Tables

The most common and useful time dimension is the calendar date dimension with the granularity of a single day.

It is one of the only dimensions that is completely specified at the beginning of the data warehouse project. It also doesn't have a conventional source.

Every calendar date dimension needs a Date Type attribute and a Full Date attribute. These two fields comprise the natural key of the dimension table.

The calendar date primary key ideally should be a meaningless surrogate key but many ETL teams can't resist the urge to make the key a readable quantity such as 20040718 meaning July 18, 2004.

In some fact tables, time is measured below the level of calendar day, down to minute or even second.

We recommend a design with a calendar date dimension foreign key and a full SQL date-time stamp, both in the fact table. The calendar day component of the precise time remains as a foreign key reference to our familiar calendar day dimension. But we also embed a full SQL date-time stamp directly in the fact table for all queries requiring the extra precision.

If the enterprise has well defined attributes for time slices within a day, such as shift names, or advertising time slots, an additional time-of-day dimension can be added to the design where this dimension is defined as the number of minutes (or even seconds) past midnight.

Reading: Slowly Changing Dimensions

The Three Types: Types 1, 2 and 3.

Type 1: Overwrite

While the Type 1 SCD is the simplest and seemingly cleanest change, there are a number of fine points to think about:

1. Type 1 destroys the history of a particular field.

2. Precomputed aggregates (including materialized views and automatic summary tables) that depend on the Home City field must be taken offline at the moment of the overwrite and must be recomputed before being brought back online.

3. In financial reporting environments with month end close processes and in any environment subject to regulatory or legal compliance, Type 1 changes may be outlawed. In these cases, the Type 2 technique must be used.

4. Overwriting a single dimension field in a relational environment has a pretty small impact but can be disastrous in an online analytical processing (OLAP) environment if the overwrite causes the cube to be rebuilt.

5. All distributed copies of the Employee dimension, as well as aggregates, must be updated simultaneously across the enterprise when Type 1 changes occur, or else the logic of drilling across will be corrupted.

6. In a pure Type 1 dimension where all fields in the dimension are subject to overwriting, a Type 1 change like the Home City change for Ralph Kimball will typically affect only one record (the record for Ralph Kimball). But in a more typical complex environment, where some fields are Type 1 and other fields are Type 2, the act of overwriting the Home City field must overwrite all the records for Ralph Kimball. In other words, Type 1 affects all history, not just the current perspective.

Type 2: Add a New Dimension Record

This has many interesting side effects:

1. Type 2 requires that we generalize the primary key of the Employee dimension.

2. In addition to the primary surrogate key, I recommend adding five additional fields to a dimension that is undergoing Type 2 processing.

3. With a dimension undergoing Type 2 processing, great care must be taken to use the correct contemporary surrogate keys from this dimension in every affected fact table.

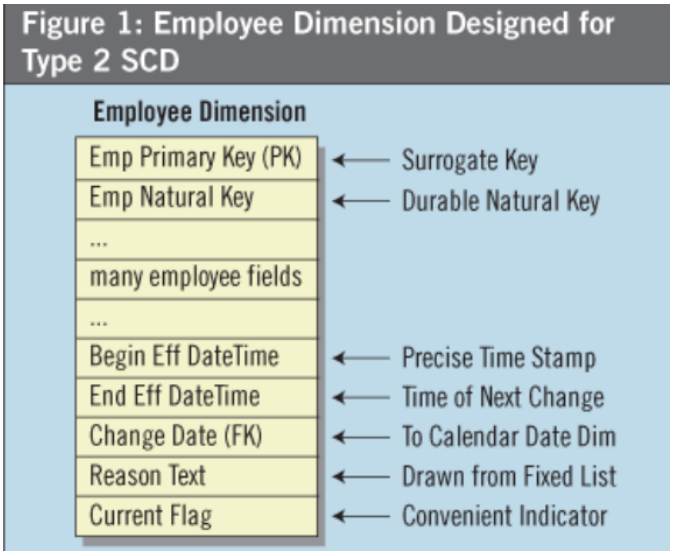


Figure 1: Employee Dimension Designed for Type 2 SCD

Type 3: Add a New Field

With Type 3 machinery in place, end users and applications can switch seamlessly between these alternate realities. If the environment requires more than one alternate reality, this approach can be generalized by adding more Alternate fields, although obviously this approach does not scale gracefully beyond a few choices.

Reading: Design Tip #105 Snowflakes, Outriggers, and Bridges

When a dimension table is snowflaked, the redundant many-to-one attributes are removed into separate dimension tables. . With snowflakes, the dimension tables are normalized to third normal form.

Outriggers are dimension tables joined to other dimension tables, but they're just one more layer removed from the fact table, rather than being fully normalized snowflakes. Outriggers are most frequently used when one standard dimension table is referenced in another dimension, such as a hire date attribute in the employee dimension table. If the users want to slice and-dice the hire date by non-standard calendar attributes, such as the fiscal year, then a date dimension table (with unique column labels such as Hire Date Fiscal Year) could serve as an outrigger to the employee dimension table joined on a date key.

Bridge tables are used in two more complicated scenarios. The first is where a many-to-many relationship can't be resolved in the fact table itself (where M:M relationships are normally handled) because a single fact measurement is associated with multiple occurrences of a dimension, such as multiple customers associated with a single bank account balance. Bridge tables are also used to represent a ragged or variable depth hierarchical relationship which cannot be reasonably forced into a simpler fixed depth hierarchy of many-to-one attributes in a dimension table.

Reading: Design Tip #48: De-Clutter With Junk (Dimensions)

A junk dimension is a convenient grouping of flags and indicators.

The benefits of a junk dimension include:

- Provide a recognizable, user-intuitive location for related codes, indicators and their descriptors in a dimensional framework.
- Clean up a cluttered design that already has too many dimensions. There might be five or more indicators that could be collapsed into a single 4-byte integer surrogate key in the fact table.
- Provide a smaller, quicker point of entry for queries compared to performance from constraining directly on these attributes in the fact table. If your database supports bitmapped indices, this potential benefit may be irrelevant, although the others are still valid.

There are two approaches for creating junk dimensions. The first is to create the junk dimension table in advance. Each possible, unique combination generates a row in the junk dimension table. The second approach is to create the rows in the junk dimension on the fly during the extract, transformation, and load (ETL) process. As new unique combinations are encountered, a new row with its surrogate key is created and loaded into the junk dimension table.

Reading: Design Tip #113 Creating, Using, and Maintaining Junk Dimensions

Three aspects of junk dimension processing: building the initial dimension, incorporating it into the fact processing, and maintaining it over time.

Admit_Type_Source

Admit _ Type_ID	Admit_Type_ Descr
1	Walk-in
2	Appointment
3	ER
4	Transfer

Care_Level_Source

Care_ Level_ ID	Care_Level_ Descr
1	ICU
2	Pediatric ICU
3	Medical Floor

Fact_Admissions_Source

Admit_ Type_ID	Care_ Level_ID	Admission_Count
1	1	1
2	1	1
2	2	1
5	3	1

SELECT ROW_NUMBER() OVER(ORDER BY Admit_Type_ID, Care_Level_ID) AS

Admission_Info_Key,

Admit_Type_ID, Admit_Type_Descr, Care_Level_ID, Care_Level_Descr

FROM Admit_Type_Source

CROSS JOIN Care_Level_Source;

SELECT ROW_NUMBER() OVER(ORDER BY F.Admit_Type_ID) AS

Admission_Info_Key,

F.Admit_Type_ID, ISNULL(Admit_Type_Descr, 'Missing Description')

Admit_Type_Descr,

F.Care_Level_ID, ISNULL(Care_Level_Descr, 'Missing Description')

Care_Level_Descr — substitute NVL() for ISNULL() in Oracle

FROM Fact_Admissions_Source F

LEFT OUTER JOIN Admit_Type_Source C ON

F.Admit_Type_ID = C.Admit_Type_ID

LEFT OUTER JOIN Care_Level_Source P ON

F.Care_Level_ID = P.Care_Level_ID;

Result:

Admission_Info_Key	Admit_ Type_ID	Admit_Type_Descr	Care_Level_ID	Care_Level_Descr
1	1	Walk-In	1	ICU
2	2	Appointment	1	ICU
3	2	Appointment	2	Pediatric ICU
4	5	Missing Description	3	Medical Floor

You could apply the second set of SQL code to the incremental fact rows and select out only the new rows to be appended to the junk dimension as shown below.

SELECT * FROM ({Select statement from second SQL code listing}) TabA

WHERE TabA.Care_Level_Descr = 'Missing Description'

OR TabA.Admit_Type_Descr = 'Missing Description' ;

Reading: Design Tip #43: Dealing With Nulls In The Dimensional Model

Nulls as Fact Table Foreign Keys

Most of our users get nervous when data disappears, so we recommend using a surrogate key, which joins to a special record in the date dimension tablewith a description like "Data not yet available."

Nulls as Facts

In this case, the null value has two potential meanings. Either the value did not exist, or our measurement system failed to capture the value. Substituting a zero instead would improperly skew these aggregated calculations.

Nulls as Dimension Attributes

We generally encounter dimension attribute nulls due to timing or dimension sub-setting. We recommend substituting an appropriately descriptive string, like "Unknown" or "Not provided."

Reading: Design Tip #128 Selecting Default Values for Nulls

Handling Null Foreign Keys in Fact Tables

There are a number of reasons why we have no foreign key:

- There is a data quality issue because the key value provided by the source system is invalid or incorrect.
- The dimension itself is not applicable for the particular fact row.

- The foreign key value is missing from the source data. In some case, this missing data is another data quality issue. In other cases, the foreign key legitimately is not known because the event being tracked has not yet occurred as frequently happens with accumulating snapshot fact tables.

Handling Null Attribute Values in Dimension Tables

There are a several reasons why the value of a dimension attribute may not be available:

- Missing Value – The attribute was missing from the source data.
- Not Happened Yet – The attribute is not yet available due to source system timing issues.
- Domain Violation – Either we have a data quality issue, or we don't understand all the business rules surrounding the attribute. The data provided by the source system is invalid for the column type or outside the list of valid domain values.
- Not Applicable – The attribute is not valid for the dimension row in question.

Avoid tricks we've seen, such as populating the default attributes with a space or meaningless string of symbols like @@@ as these only confuse the business users.

Text: real explanation

Numeric: 0

PPT:

Surrogate Key: Implementation Considerations

Ø Most databases have a way of generating a surrogate key for a table

Ø Most ETL tools can generate a surrogate key with a counter as well

Ø An ETL generated key works no matter what type of database under the hood

Ø Database surrogate key is easier to implement

Ø Special care is required not to reset the database surrogate key sequence

Date Dimension Considerations

Ø Predictable stable dimension

Ø Key assigned chronically

Ø YYYYMMDD instead of surrogate sequence number

§ Query YYYYMMDD to bypass Date dimension table is not advised

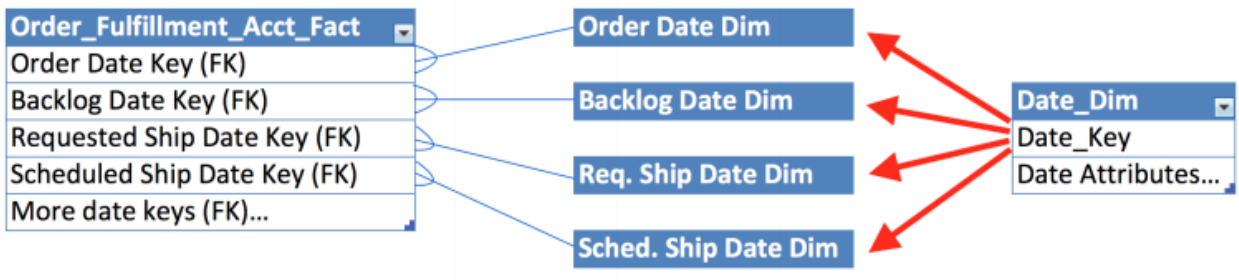
§ Useful for partitioning large fact table

Ø Need default (dummy) row/key for unknown or to be determined date

Ø Handle time-of-day separate from Date dimension for day parts

Ø Consider transaction date/time stamp as fact

Roll-Playing Dimensions



Ø A single physical dimension table playing multiple logical roles

Ø Options for implementing roll-playing dimensions

§ Database views

§ Aliases or synonyms of a single physical table

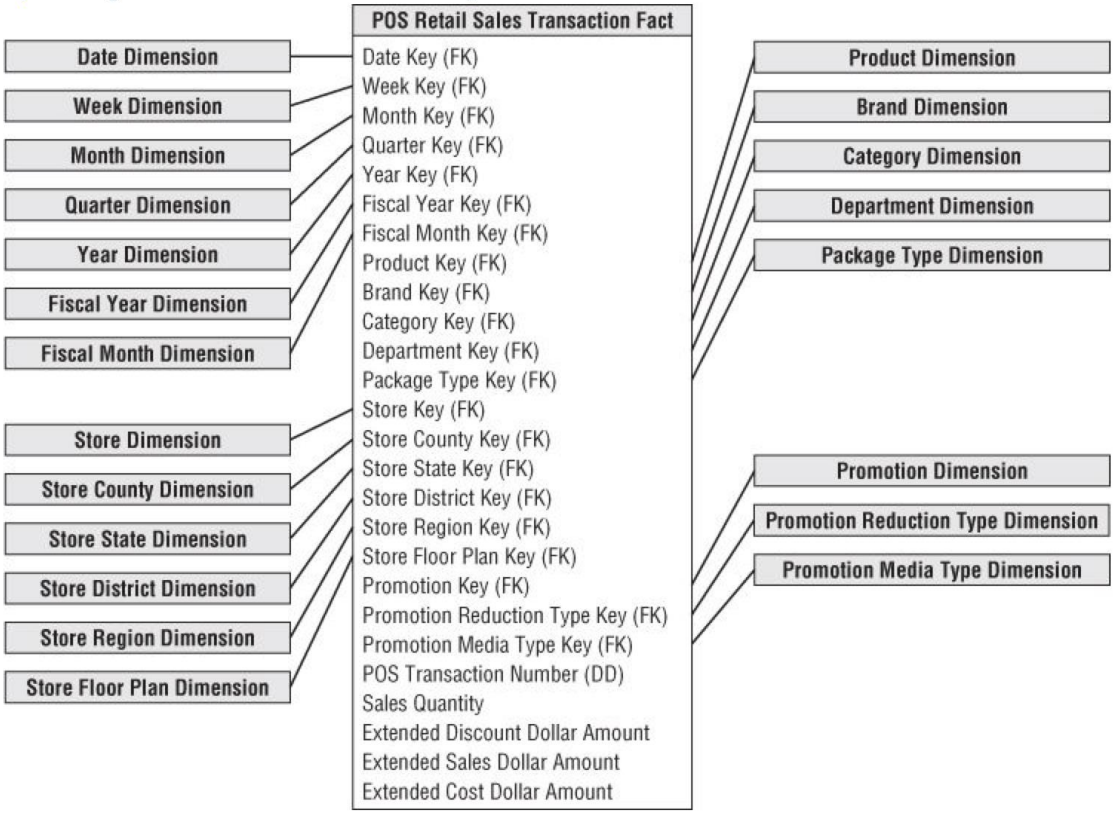
§ Aliasing within the BI tool's semantic layer

§ Other examples: Employee (Cashier/Associate),

Location (Origin/Destination), Physicians (Primary/Referring), etc.

Avoid Too Many Dimension "Centipede" Fact Tables

Figure 3.17 Centipede fact table with too many normalized dimensions.



Junk Dimensions

Ø Combine miscellaneous transaction flags/indicators into junk dimension

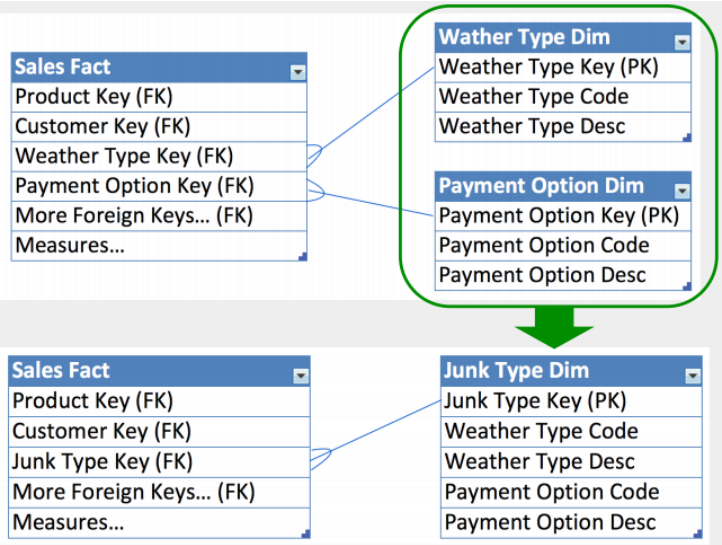
Ø Potentially less desirable alternative:

§ Multiple fact table keys to low-cardinality dimensions

Ø Undesirable alternatives:

§ Place flags/indicators directly in fact table as text facts or DDs

§ Place flags/indicators in transaction dimension



Dealing with Nulls: Operations with Nulls

Ø Operations with Null can be tricky sometimes

```
SELECT 1 + null col
  FROM dual; -- null

SELECT CASE WHEN null = null THEN 1 ELSE 2 END AS col
  FROM dual; -- 2

SELECT CASE WHEN null IS null THEN 1 ELSE 2 END AS col
  FROM dual; -- 1

SELECT CASE WHEN 1 IS null THEN 1 ELSE 2 END AS col
  FROM dual; -- 2

SELECT DECODE (null, null, 1, 2, 3) as col
  FROM dual; -- 1

SELECT DECODE (1, null, 1, 2, 3) as col
  FROM dual; -- null
```

Ø Operations with Null can result in missing information

```
SELECT deptid, SUM(annual_salary) AS annual_salary_by_dept
  FROM (
        SELECT deptid, empid, month_pay*12+bonus AS annual_salary
          FROM (
                SELECT 'd01' AS deptid, 'e0001' AS empid,
                       10000 AS month_pay, 1000 AS bonus
                  FROM dual UNION ALL
                SELECT 'd01' AS deptid, 'e0002' AS empid,
                      100000 AS month_pay, null AS bonus
                  FROM dual
               )
       )
 GROUP BY deptid
;
```

```
DEPTID ANNUAL_SALARY_BY_DEPT
------ ---------------------
d01                    121000 -- $1,200,000 missing!
```

Ø Aggregate functions handle Null gracefully

```
SELECT room_no, AVG(math) AS avg_math_by_room, AVG(writing) AS
avg_writing_by_dept
  FROM (
        SELECT room_no, student_id, math, writing
          FROM (
                SELECT 'rm2001' AS room_no, 's0001' AS student_id,
                        80 AS math,   90 writing
                  FROM dual UNION ALL
                SELECT 'rm2001' AS room_no, 's0002' AS student_id,
                       100 AS math, null writing
                  FROM dual
               )
       )
 GROUP BY room_no;

ROOM_NO AVG_MATH_BY_ROOM AVG_WRITING_BY_DEPT
------- ---------------- ------------------
rm2001                90                 90
```

Dealing with Nulls: Rule of Thumb in Data Warehousing

Ø NULL fact table foreign keys

§ No NULL is allowed as it breaks referential integrity

§ Substitute key to special dimension row (a.k.a. dummy dimension row)

Ø NULL dimension attributes

§ Strongly discouraged to avoid unexpected query results (e.g. invalidating index strategy)

§ Use default values instead – N/A, Unknown, Invalid, To be determined,…

Ø NULL facts

§ Use ONLY IF it truly means N/A, Unknown, and Invalid, not zero

Week 05 Dimensional Modeling: Integration Via Conformed Dimensions

Reading: Design Tip #135 Conformed Dimensions as the Foundation for Agile 灵活的； Data Warehousing

A conformed dimension is descriptive master reference data that's referenced in multiple dimensional models.

Reading: The Soul of the Data Warehouse

The three most important are drilling down, drilling across, and handling time.

Drilling Down

Drilling down in a relational database means "adding a row header" to an existing SELECT statement.

Incidentally, we often call a row header a "grouping column" because everything in the select list that's not aggregated with an operator such as SUM must be mentioned in the SQL GROUP BY clause.

I can now make some precise technical comments about drilling down:

1. Drilling down is the most basic end-user maneuver in the data warehouse, which must support it in as general and flexible a manner as possible because there's no way to predict the user's drill-down path.

2. The data warehouse must therefore support drilling down at the user interface level, at all times, with the most atomic data possible because the most atomic data is the most dimensional. \

3. The atomic data must be in the same schema format as any aggregated form of the data: The atomic data must be a smoothly accessible target for drill-down paths using standard ad hoc query tools.

4. To build a practical system for drilling down, you want standard ad hoc query tools to present the drill-down choices without special schema-dependent programming, and you want these tools to emit the correct resulting SQL without schema-dependent programming.

5. Only one standard schema methodology exists that's capable of expressing data in a single, uniform format that looks the same at the atomic layers as in all aggregated layers, and at the same time requires no schema-dependent programming: the star schema, otherwise known as the dimensional model.

6. The star schema design in the presentation layer smoothly supports prebuilt aggregation tables and snowflake designs. Aggregations and snowflakes are related.

A modern data warehouse environment uses a query-rewrite facility called an aggregate navigator to choose a prebuilt aggregate table whenever possible.

Drilling Across

Drilling across adds more data to an existing row. It's better described as the column accretion from separate queries.
Drilling across by adding another measured fact to the SELECT list from the existing fact table mentioned in the query is a trivial accomplishment.

Drill-across observation: The new fact table called for in the drill-across operation must share certain dimensions with the fact table in the original query.

Drill-across choice: Either send a single, simultaneous SQL request to the two fact tables or send two separate requests.

Implementing Drill-Across

1. All fact tables in a drill-across query must use conformed dimensions.

2. The actual drill-across query consists of a multi-pass set of separate requests to the target fact tables followed by a simple sort/merge on the identical row headers returned from each request.

If two fact tables have a "Customer" dimension, then "Customer" is conformed if the two dimensions are exactly the same.

When you bring two separate queries together in a drill-across operation, both queries must have the same number of row headers, arranged from left to right in the same order.

To sort/merge the two queries, you must sort them the same way.

In my classes, I sometimes describe conformed dimensions as either dimensions that are exactly equal (the trivial case) or dimensions where "one is a subset of the other."

Handling Time

This pledge generates three main requirements for the data warehouse:

First, every piece of data in the data warehouse must have a clearly understood time validity.

Second, if the detailed description of a data warehouse entity has changed over time, you must correctly associate each version of that entity with the contemporary versions of other measurements and entities in the data warehouse.

Last, the data warehouse must support the natural ways people have of viewing data over time. These natural ways include seeing instantaneous events, regular periodic reports, and latest status.

Time Validity

Every fact table has a time dimension.

You want the time stamp in the fact table to be a surrogate key rather than a real date for three reasons: First, the rare time stamp that is inapplicable, corrupted, or hasn't happened yet needs a value that cannot be a real date. Second, most end-user calendar navigation constraints, such as fiscal periods, end-of-periods, holidays, day numbers, and week numbers aren't supported by database time stamps. Therefore, they need to come from a table with a verbose time dimension, rather than computed in the requesting query tool. Third, integer time keys take up much less disk space than full dates.

When the source system provides a detailed time stamp for the measurement down to the minute or the second, the time of day needs to be a separate dimension.

Correct Association

When the data warehouse encounters a legitimate revised description of, for example, a customer, there are three fundamental choices for handling this slowly changing dimension:

1. Overwrite the changed attribute, thereby destroying previous history. This approach is justifiable only when correcting an error, if you're living by the pledge.

2. Issue a new record for the customer, keeping the customer natural key, but creating (by necessity) a new surrogate primary key.

3. Create an additional field in the existing customer record, and store the old value of the attribute in the additional field. Overwrite the original attribute field. This strategy is called for when the attribute can have simultaneous "alternate realities."

The most sophisticated treatment of a Type 2 SCD record involves five fields:

- Begin effective date/time stamp (not a surrogate key pointer)
- End effective date/time stamp
- Effective date surrogate key (daily grain) connecting to date dimension as a snowflake
- Change description field (text)
- Most recent flag.

Natural Views

Fact-table measurements all fall into just three classes. These types correspond to instantaneous events, regular periodic reports, and latest status. In dimensional modeling, these three fact-table types are the transaction grain, the periodic-snapshot grain, and the accumulating-snapshot grain.

Reading: Drill Down to Ask Why

An analytic application consists of five stages:

1. Publish reports.

2. Identify exceptions.

3. Determine causal factors.

4. Model alternatives. Provide a backdrop to evaluate different decision alternatives.

5. Track actions. Evaluate the effectiveness of the recommended actions and feed the decisions back to both the operational systems and DW, against which stage one reporting will be conducted, thereby closing the loop.

In stage three, we determine the causal factors behind the exceptions. Why?

Imagine five ways in which the fare planner might ask why. I'll arrange these in order of increasing breadth and complexity:

1. Give me more detail.

2. Give me a comparison. Compare before and now

3. Let me search for other factors. Jump to nonyield databases, such as a weather database, a holiday/special events database, a marketing promotions database or a competitive pricing database to see if any of these exogenous factors could have played a role.

4. Tell me what explains the variance. Perform a data mining analysis, perhaps using decision trees, examining hundreds of marketplace conditions to see which of these conditions correlates most strongly with the drop in yield (explaining the variance in data mining terminology).

5. Search the Web for information about the problem.

Reading: Design Tip #155 Going Agile? Start with the Bus Matrix

The bus matrix provides a master plan for agile development, plus it identifies the reusable common descriptive dimensions that provide both data consistency and reduced time-to-market delivery in the long run.

Collaboration is critical to identifying the business's core processes.

Reading: Enterprise Data Warehouse Bus Architecture

It decomposes the DW/BI planning process into manageable pieces by focusing on the organization's core business processes, along with the associated conformed dimensions.

Conformed dimensions are common, standardized, master dimensions that are managed once in the extract, transformation, and load (ETL) system and then reused by multiple fact tables. Conformed dimensions deliver consistent descriptive attributes across dimensional models. They support the ability to drill across and integrate data from multiple business processes. Finally, reusing conformed dimensions shortens the time-to-market by eliminating redundant design and development efforts.

Reading: The Matrix

A first-level data mart is a collection of related fact tables and dimension tables that is typically:

- Derived from a single data source
- Supported and implemented by a single department
- Based on the most atomic data possible to collect from the source
- Conformed to the “data warehouse bus.”

Inviting Data Mart Groups to the Conforming Meeting

Conformed dimensions are the basis for distributed data warehouses, and using conformed dimensions is the way to avoid stovepipe data marts.

Communicating With the Boss

Second-Level Data Marts

A second-level data mart is a combination of two or more first-level marts.

Matrix Columns for Reference Data

Data Stewardship 管理员、服务员等的职位和职责 Required

Process-Centric Rows

Each row of the bus matrix corresponds to a business process within the organization.

Associate Matrix Columns and Rows

Common Matrix Mishaps

Row mishaps commonly fall into the following two categories:

- Departmental or overly encompassing rows.
- Report-centric or too narrowly defined rows.

When defining the matrix columns, architects naturally fall into the similar traps of defining columns that are either too broad or narrow:

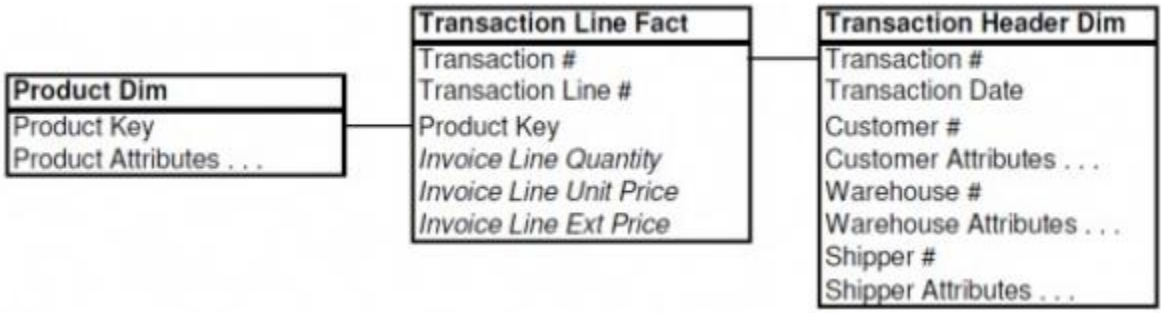
- Overly generalized columns.
- Separate columns for each level of a hierarchy.

Matrix Extensions

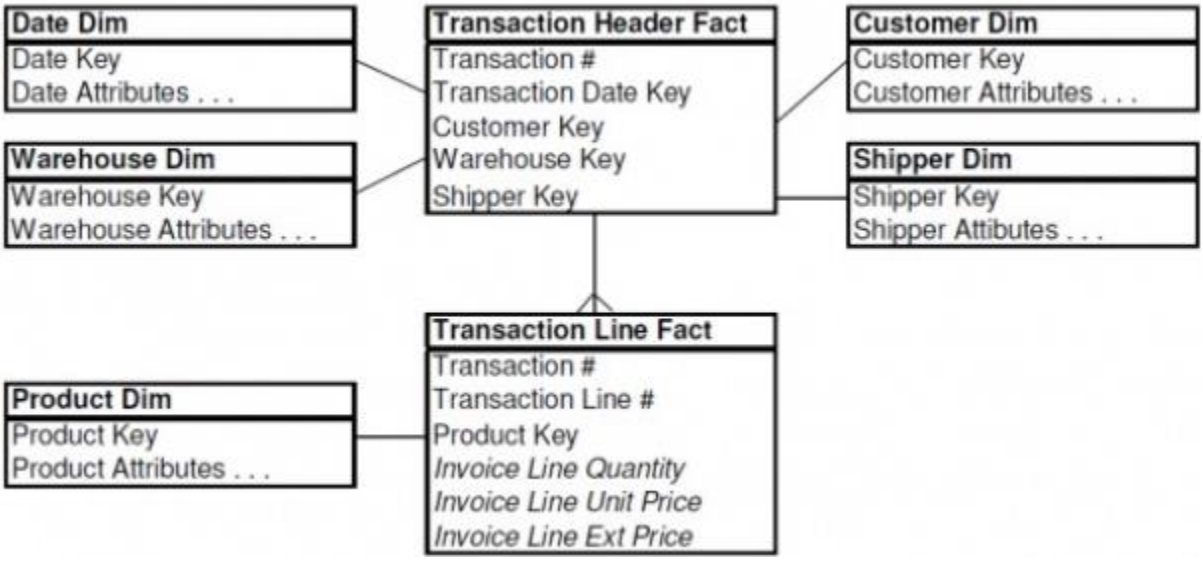
- Opportunity matrix. Once the bus matrix rows have stabilized, replace the dimension columns with business stakeholders, such as marketing, sales and finance.
- Detailed implementation bus matrix. A single business process matrix row sometimes spawns multiple fact tables or OLAP cubes.

Reading: Design Tip #95 Patterns to Avoid when Modeling Header/Line Item Transactions

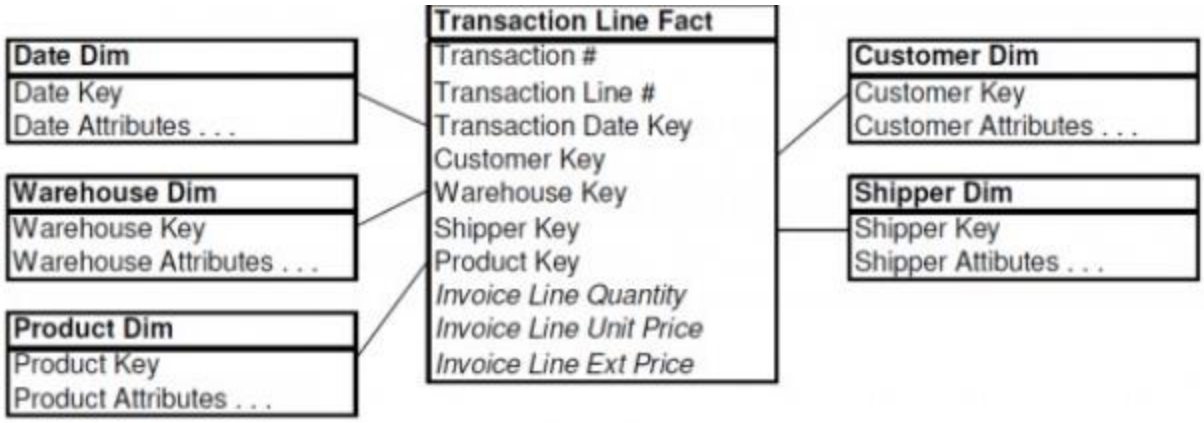
Bad Idea #1



Bad Idea #2



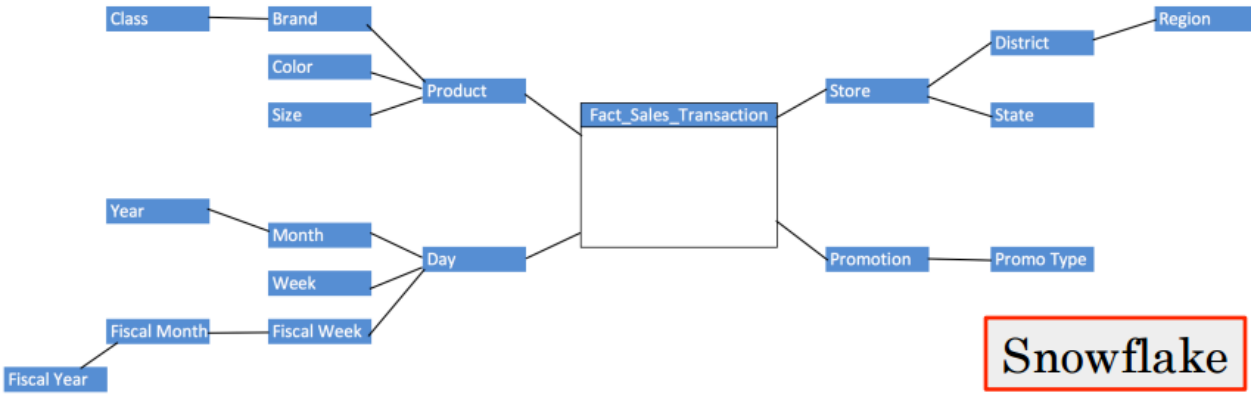
Recommended Structure for Header/Line Item Transactions

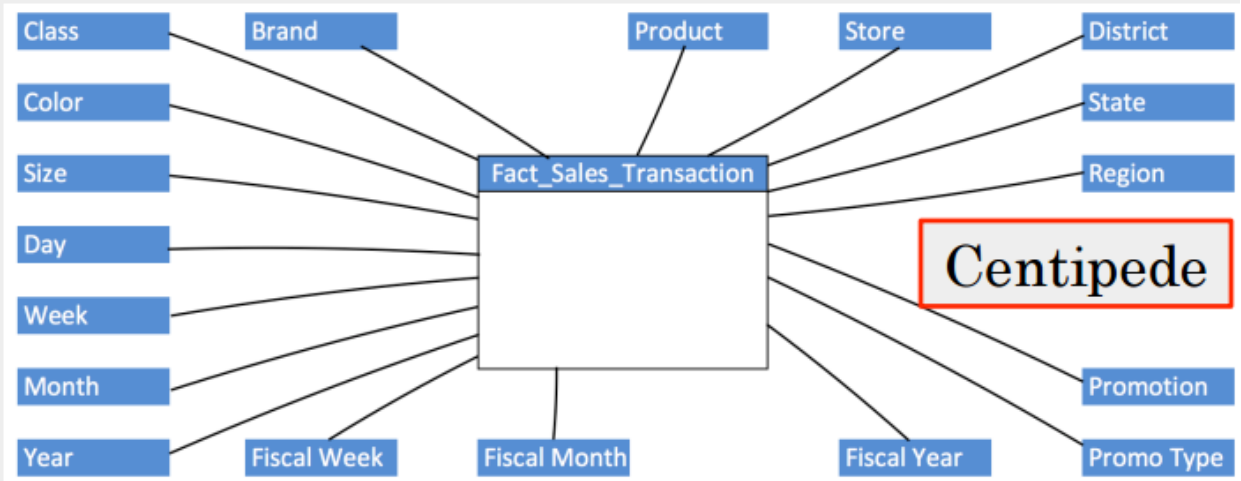


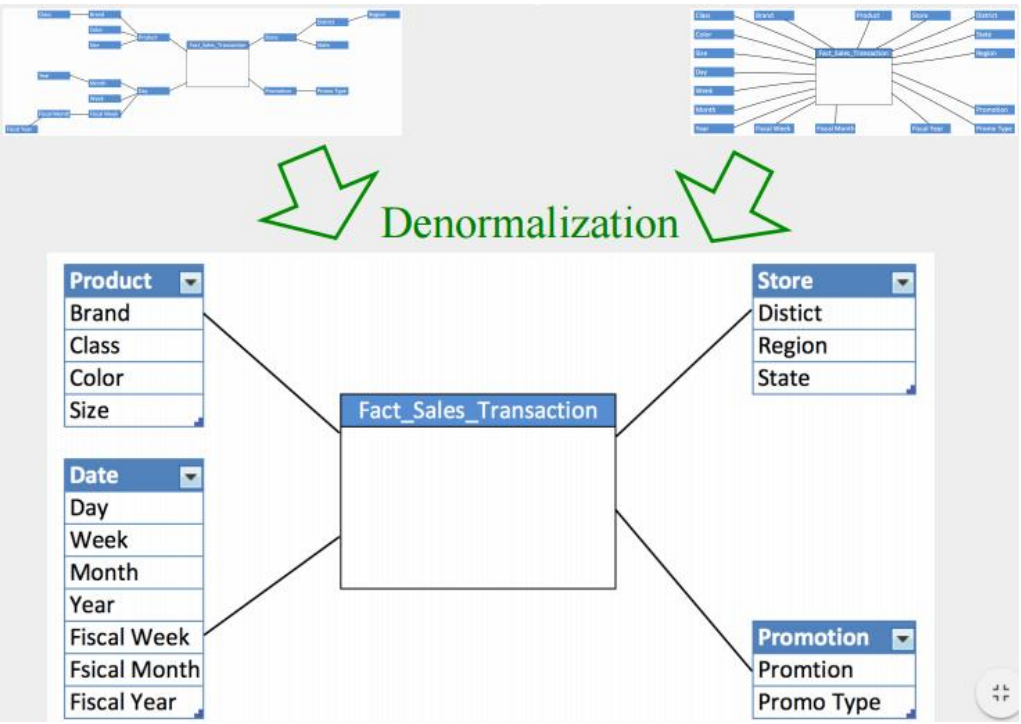
Rather than holding onto the operational notion of a transaction header "object," we recommend that you bring all the dimensionality of the header down to the line items.

PPT:

Avoid Snowflake or Centipede Model via Denormalization







Dealing with Nulls Revisited

Ø NULL dimension attributes

§ Strongly discouraged to avoid unexpected query results (e.g. invalidating index strategy)

§ Use default values instead – N/A, Unknown, Invalid, To be determined,…

Ø NULL facts

§ Use ONLY IF it truly means N/A, Unknown, and Invalid, not zero

Default/Dummy Row in Dimension Tables

Ø Every foreign key in a fact table should reference a dimension row

Ø Default/dummy row prevents missing rows in building fact tables

SAELS_SOURCE

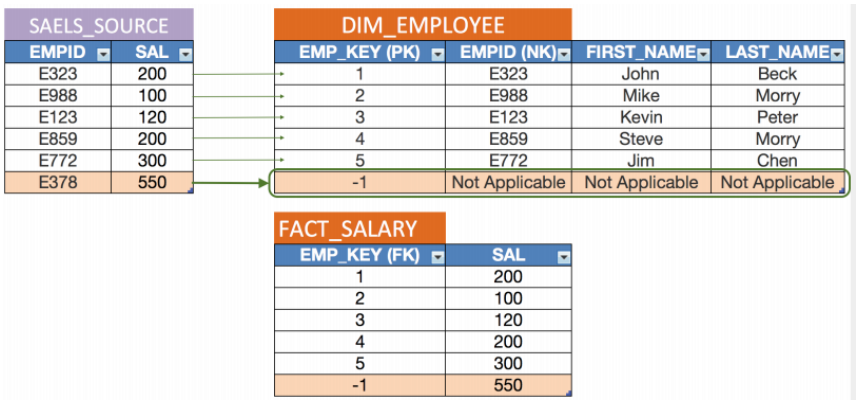
EMPID	SAL
E323	200
E988	100
E123	120
E859	200
E772	300
E378	550

DIM_EMPLOYEE

EMP_KEY (PK)	EMPID (NK	FIRST_NAM	LAST_NAM
1	E323	John	Beck
2	E988	Mike	Morry
3	E123	Kevin	Peter
4	E859	Steve	Morry
5	E772	Jim	Chen
?			

FACT_SALARY

EMP_KEY (FK)	SAL
1	200
2	100
3	120
4	200
5	300



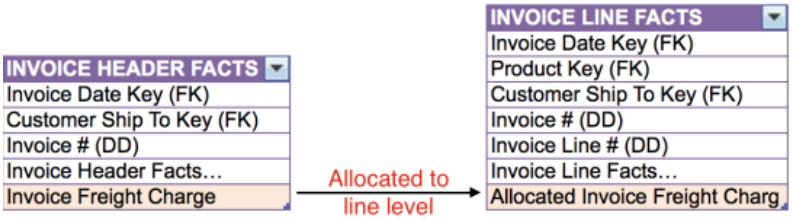
Ø Default values for dimension attributes:

"Missing Value", "Not Happened Yet", "Domain Violation", "Not Applicant", etc.

Allocating Header Facts

Ø Recommended

§ Option 1: Allocate to line-grained fact table



§ Option 2: Leave unallocated fact in header-grain fact table

Ø Not recommended

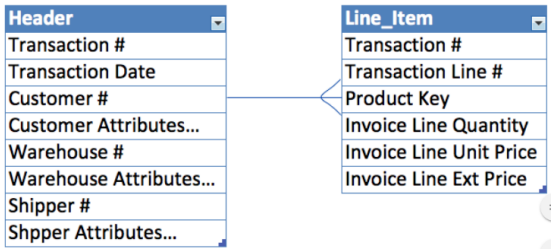
§ Include unallocated fact on every line fact row

§ Include unallocated fact on first or last fact row

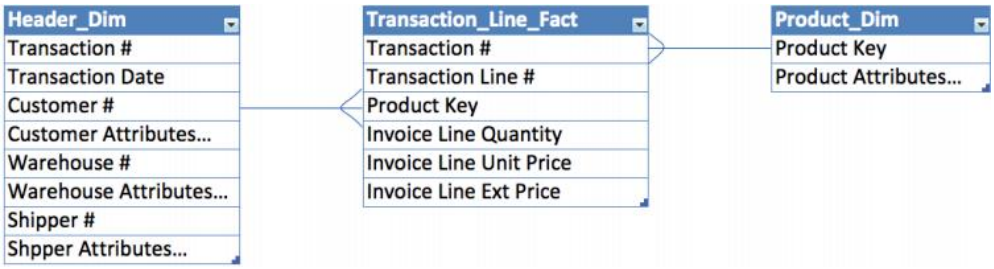
§ Store in transaction dimension

Patterns to Avoid when Modeling Header/Line Item Transactions

Ø ERD in OLTP Source System:



Ø Pattern #1 to Avoid

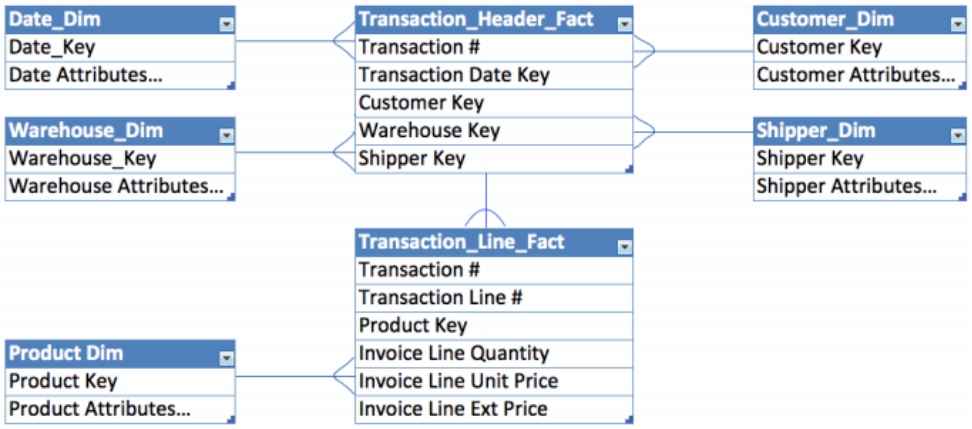


§ The transaction header dimension will get very large

§ Business user's requests will have to go through this very large dimension

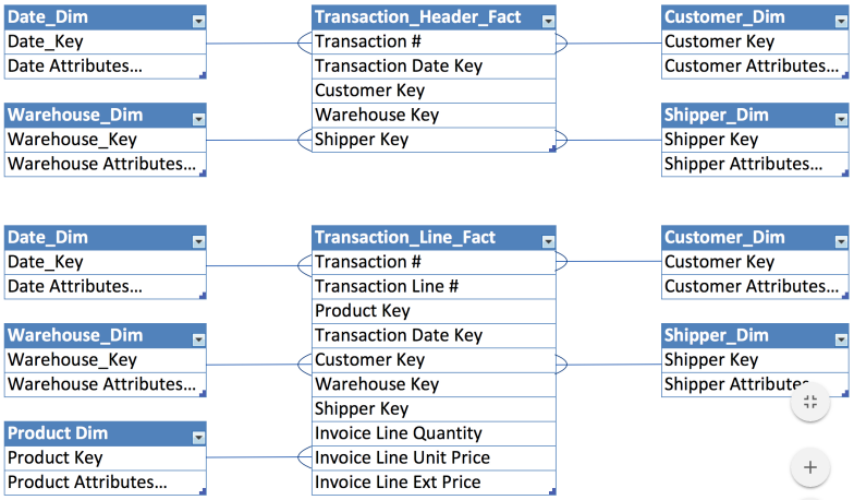
§ This can get even larger when applying SCD Type 2 (preserving change history)

Ø Pattern #2 to Avoid

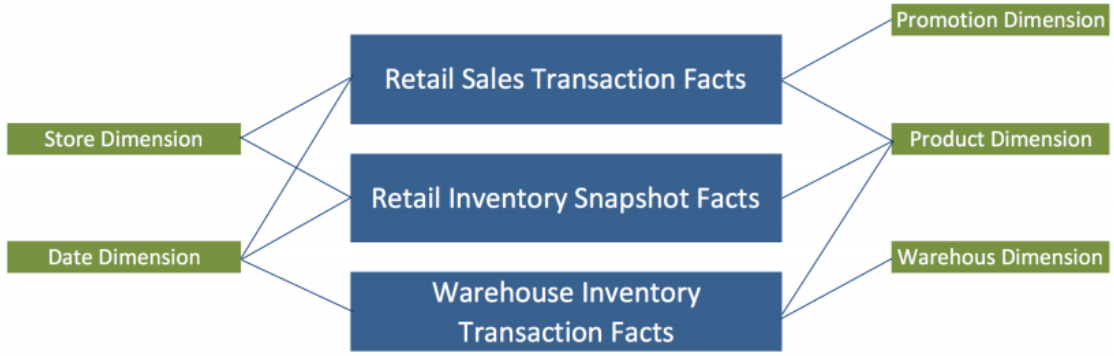


§ It is fine to build Header fact for header metrics, but Line fact table should inherit header dimensions

Ø Suggested pattern



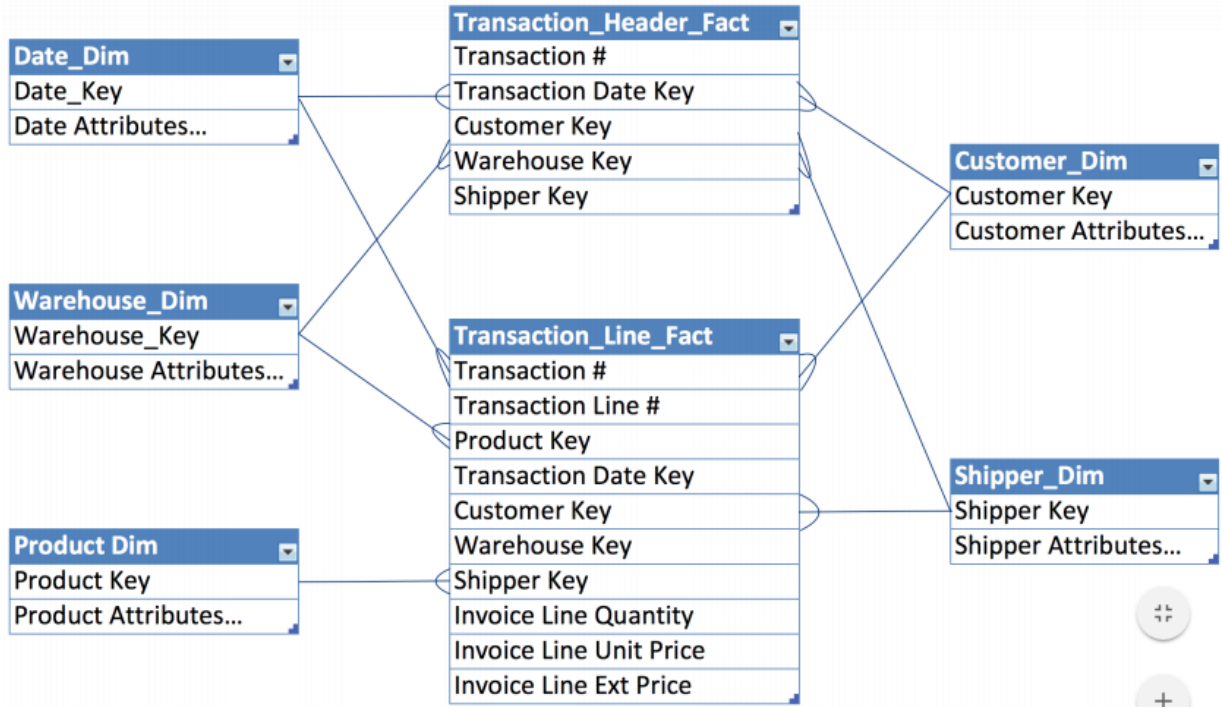
Conformed Dimensions Tables Shared by Fact Tables



Ø Each business process typically is represented by one or more fact tables

Ø Using shared, common dimensions is absolutely critical for data marts integrated seamlessly

➢ Suggested pattern – Logical View

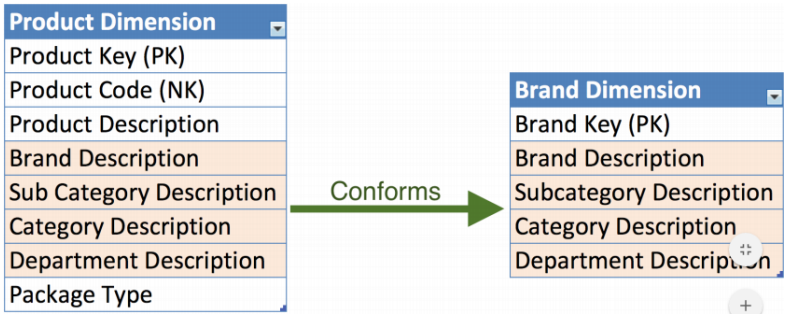


Conformed Dimensions

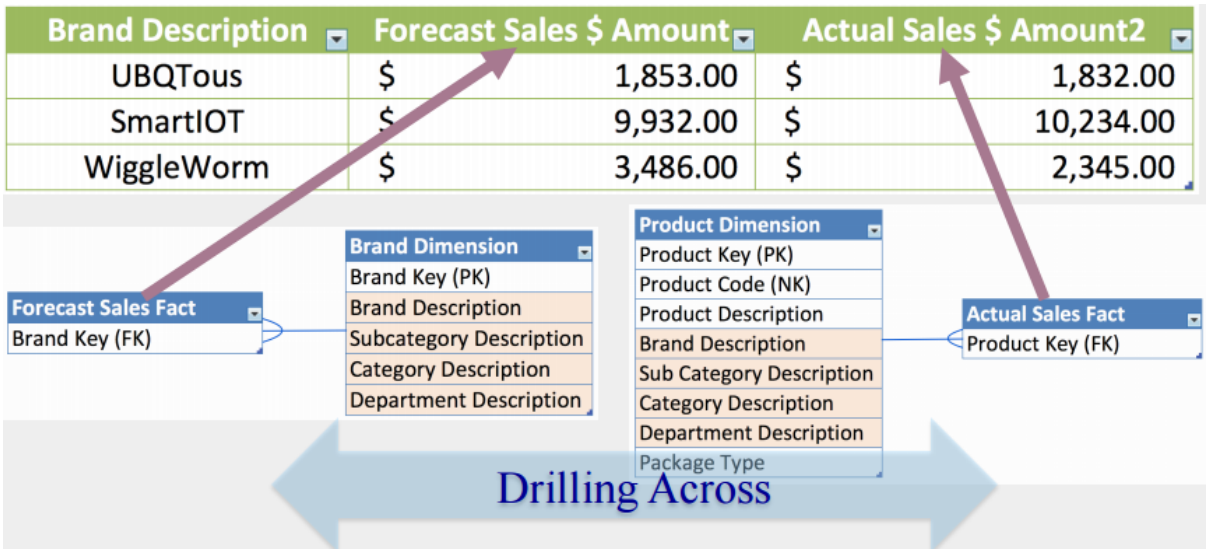
Identical dimensions conform

Ø Shrunken rollup dimensions conform

§ Domain values of conform columns must match



Drilling Across Fact Tables via Conformed Dimension Tables



Ø Open separate connection to each source

Ø Assemble each answer set

Ø Merge answer sets on conformed row head

Enterprise Data Warehouse Bus Architecture

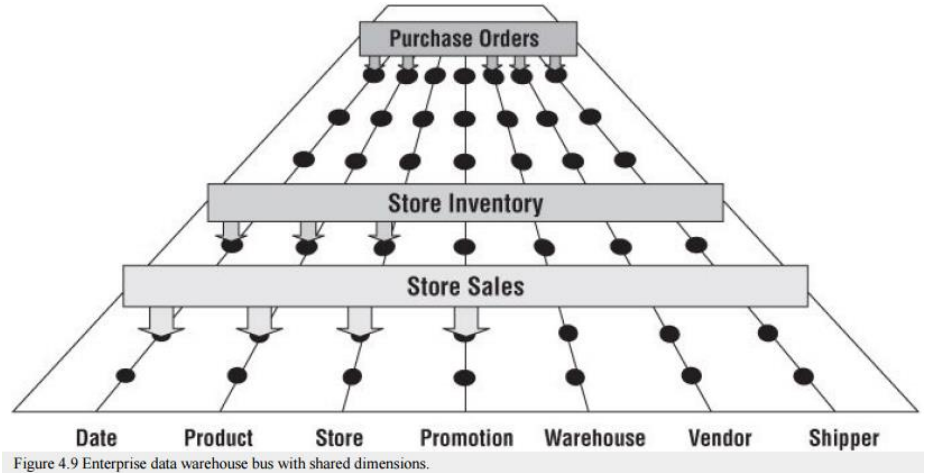


Figure 4.9 Enterprise data warehouse bus with shared dimensions.

Ø The Enterprise Data Warehouse Bus Architecture provides a standardized master set of conformed dimensions and conformed facts used through the data warehouse

Ø It is analogous to the bus in your computer, providing a standard interface that allows many different kinds of devices to connect to your computer and co-exist

Ø Conformed dimensions are standard dimensions that are shared among dimensional models.

Ø The use of conformed dimension is the central technique for building an enterprise data warehouse from a set of dimensional models

Ø As the separate dimensional models are developed, they plug into the Bus, fitting together like pieces of the puzzle

Ø Isolated data marts that cannot be tied together are disastrous. Stovepipe data marts merely perpetuate incompatible views of the business

Enterprise Data Warehouse Bus Matrix

Figure 4.10 Sample enterprise data warehouse bus matrix for a retailer. Kimball, Ralph; Ross, Margy (2013-07-01). The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. Wiley. Kindle Edition.

BUSINESS PROCESSES	Date	Product	Warehouse	Store	Promotion	Customer	Employee
Issue Purchase Orders	X	X	X				
Receive Warehouse Deliveries	X	X	X				X
Warehouse Inventory	X	X	X				
Receive Store Deliveries	X	X	X	X			X
Store Inventory	X	X		X			

Ø Rows translate into fact tables

Ø Columns represent common dimensions used across the enterprise. Mark the intersections where the dimensions are relevant to the business processes. The resulting matrix will be surprising dense

Ø Sharing conformed dimensions across the data warehouse is absolutely critical

- § Ensures consistent definition of common data
- § Ensures consistent row/column heading labels and roll-ups
- § Ensures consistent "values" for consistently defined dimensions and attributes
- § Reduce time to market
- § Support integration and drilling across fact tables

Ø Committing to use conformed dimensions is a business policy. It represents more political challenges than technical hurdles

Week 07 More Dimension Patterns and Considerations

Reading: Design Tip #97 Modeling Data as Both a Fact and Dimension Attribute

Remember that numerical facts usually have an implicit time series of observations, and usually participate in numerical computations such as sums and averages, or more complex functional expressions. Dimension attributes, on the other hand, are the targets of constraints, and provide the content of "row headers" (grouping columns) in a query.

Reading: Design Tip #105 Snowflakes, Outriggers, and Bridges

When a dimension table is snowflaked, the redundant many-to-one attributes are removed into separate dimension tables.

We generally encourage you to handle many-to-one hierarchical relationships in a single dimension table rather than snowflaking. Snowflakes may appear optimal to an experienced OLTP data modeler, but they're suboptimal for DW/BI query performance.

Outriggers are similar to snowflakes in that they're used for many-to-one relationships, however they're more limited. Outriggers are dimension tables joined to other dimension tables, but they're just one more layer removed from the fact table, rather than being fully normalized snowflakes.

Bridge tables are used in two more complicated scenarios. The first is where a many-to-many relationship can't be resolved in the fact table itself (where M:M relationships are normally handled) because a single fact measurement is associated with multiple occurrences of a dimension, such as multiple customers associated with a single bank account balance. Placing a customer dimension key in the fact table would require the unnatural and unreasonable divvying of the balance amongst multiple customers, so a bridge table with dual keys to capture the many-to-many relationship between customers and accounts is used in conjunction with the measurement fact table. Bridge tables are also used to represent a ragged or variable depth hierarchical relationship which cannot be reasonably forced into a simpler fixed depth hierarchy of many-to-one attributes in a dimension table.

Reading: Design Tip #142 Building Bridges

We use a bridge table to capture this many-to-many relationship.

There are two major classes of bridge tables. The first, and easiest to model, captures a simple set of values associated with a single fact row. The second kind of many-to-many relationship exists independent of the transactions being measured. The relationship between Customer and Account is a good example.

Historical Load

The steps involved in creating the historical bridge table depend on how the data is captured in the source system.

Create the Initial List of Groups

SELECT Row_Number() OVER (ORDER BY Diagnosis_Code_List) AS

Diagnosis_Group_Key, Diagnosis_Code_List

INTO Diagnosis_Group

FROM(

SELECT DISTINCT Diagnosis_Code_List

FROM

(SELECT DISTINCT OuterTrans.ER_Admittance_ID,

STUFF((SELECT ', ' + CAST(Diagnosis_Code AS VARCHAR(1024))

FROM ER_Admittance_Transactions InnerTrans

WHERE InnerTrans.ER_Admittance_ID = OuterTrans.ER_Admittance_ID

ORDER BY InnerTrans.Diagnosis_Code

FOR XML PATH('')),1,2,'') AS Diagnosis_Code_List

FROM ER_Admittance_Transactions OuterTrans

) OuterList

) FinalList;

ER_Admittance_Transactions

ER_Admittance_ID	Diagnosis_ Code
27	T41.201
27	Z77.22
28	K35.2
28	B58.09
28	I13.10
29	T41.201
29	Z77.22

Diagnosis_Group

Diagnosis_ Group_Key	Diagnosis_Code_List
1	B58.09, I13.10, K35.2
2	T41.201, Z77.22

gure 1 –Source transaction data and associated diagnosis group table.

Create the Bridge Table

Diagnosis_Bridge

Diagnosis Group Key	Diagnosis _Key
1	1
1	3
1	5
2	4
2	5

ICD10_Diagnosis

Diagnosis _Key	Diagnosis _Code	Diagnosis_Description
1	B58.09	Other toxoplasma oculopathy
2	I13.10	Hypertensive heart and chronic kidney disease without heart failure
3	K35.2	Acute appendicitis with generalized peritonitis
4	T41.201	Poisoning by unspecified general anesthetics, accidental (unintentional)
5	Z77.22	Contact with and (suspected) exposure to environmental tobacco smoke

Figure 2 –Diagnosis Bridge table and associated ICD10_Diagnosis dimension.

WITH XMLTaggedList AS (

SELECT Diagnosis_Group_Key,

CAST('<I>' + REPLACE(Diagnosis_Code_List, ', ', '</I><I>') + '</I>'AS XML)

AS Diagnosis_Code_List

FROM Diagnosis_Group

)

SELECT Diagnosis_Group_Key,

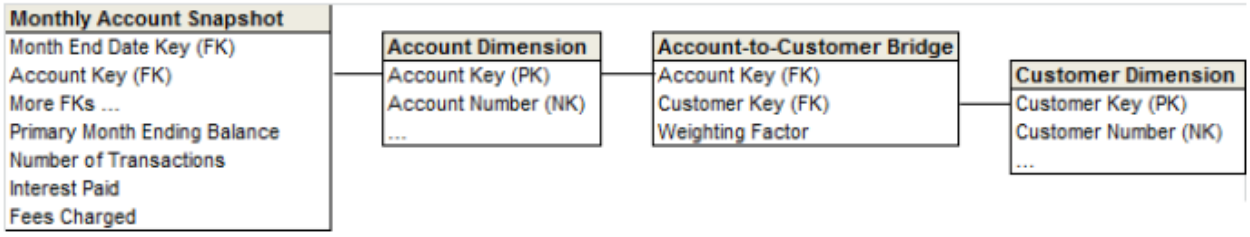
ExtractedDiagnosisList.X.value('.', 'VARCHAR(MAX)') AS Diagnosis_Code_List

FROM XMLTaggedList

CROSS APPLY Diagnosis_Code_List.nodes('//I') AS ExtractedDiagnosisList(X);

Incremental Processing

Reading: Design Tip #166 Potential Bridge (Table) Detours 绕路；便道



Here are several potential techniques to avoid bridge tables. However, be aware that each comes with its own potential downsides, too.

1. Alter the fact table's grain to resolve the many-valued dimension relationship, allocating the metrics accordingly.
2. Designate a "primary" value.
3. Add multiple named attributes to the dimension table
4. Add a single concatenated text string with delimited attribute values to the dimension.

Reading: Fistful of Flaws

What's the Grain?

Mixed-Grain or Textual Facts?

Once you have established the fact table granularity, identify the facts that are consistent with this grain.

Fact tables typically consist of foreign keys plus numeric counts and amounts measuring business performance. Optimally, the facts are additive, meaning they can be summed across any dimension.

You should also prohibit text fields, including cryptic indicators and flags, from entering the fact table.

Dimension Descriptors and Decodes?

Handling of Hierarchies?

Be careful to avoid abusing the outrigger technique; outriggers should be the exception rather than the rule. Similarly, if your design is riddled with bridge tables to capture many-valued dimension relationships, you need to go back to the drawing board.

Explicit Date Dimension?

Control Numbers as Degenerate Dimensions?

In transaction-oriented fact tables, treat the operational control numbers (such as the purchase order or invoice number) as degenerate dimensions. They reside as dimension keys on the fact table, but don't join to a corresponding dimension table.

Teams are sometimes tempted to create a dimension table with information from the operational header record, such as the transaction number, transaction date, transaction type, or transaction terms. In this case, you'd end up with a dimension table that has nearly as many rows as the fact table.

Surrogate Keys?

Surrogate keys let you integrate data with multiple operational keys from multiple sources. They are also required to support the dominant technique for tracking changes to dimension table attributes.

Slowly Changing Dimension Strategies?

Well-Understood Business Requirements?

Reading: The 10 Essential Rules of Dimensional Modeling

Rule #1: Load detailed atomic data into dimensional structures.

Rule #2: Structure dimensional models around business processes.

Rule #3: Ensure that every fact table has an associated date dimension table.

Rule #4: Ensure that all facts in a single fact table are at the same grain or level of detail.

Rule #5: Resolve many-to-many relationships in fact tables

Use a many-to-many, dual-keyed bridge table in conjunction with the fact table.

Rule #6: Resolve many-to-one relationships in dimension tables.

Using the fact table to resolve M:1 relationships should be done sparingly.

Rule #7: Store report labels and filter domain values in dimension tables

Rule #8: Make certain that dimension tables use a surrogate key.

Rule #9: Create conformed dimensions to integrate data across the enterprise.

Rule #10: Continuously balance requirements and realities to deliver a DW/BI solution that's accepted by business users and that supports their decision-making.

Reading: Design Tip #120 Design Review Dos and Don'ts

Before the design review…

- Do invite the right players. Obviously, the modeling team needs to participate, but you'll also want representatives from the BI development team (to ensure that proposed changes enhance usability) and ETL development team (to ensure that the changes can be populated). Perhaps most importantly, it's critical that folks who are really knowledgeable about the business and their needs are sitting at the table. While diverse perspectives should participate in a review, don't invite 25 people to the party.

- Do designate someone to facilitate the review. Group dynamics, politics, and the design challenges themselves will drive whether the facilitator should be a neutral resource or involved party. Regardless, their role is to keep the team on track to achieving a common goal. Effective facilitators need the right mix of intelligence, enthusiasm, confidence, empathy, flexibility, assertiveness, and the list goes on. Don't forget a sense of humor.
- Do agree upon the scope of the review (e.g., dimensional models focused on several tightly coupled business processes.) Ancillary topics will inevitably arise during the review, but agreeing in advance on the scope makes it easier to stay focused on the task at hand.
- Do block off time on everyone's calendar. We typically conduct dimensional model reviews as a focused 2- day effort. The entire review team needs to be present for the full two days. Don't allow players to float in and out to accommodate other commitments. Design reviews require undivided attention; it's disruptive when participants leave intermittently.
- Do reserve the right space. The same conference room should be blocked for the full two days. Optimally, the room has a large white board; it's especially helpful if the white board drawings can be saved or printed. If a white board is unavailable, have flip charts on hand. Don't forget markers and tape; drinks and food never hurt.
- Do assign homework. For example, ask everyone involved to make a list of their top five concerns, problem areas, or opportunities for improvement with the existing design. Encourage participants to use complete sentences when making their list so it's meaningful to others. These lists should be emailed to the facilitator in advance of the design review for consolidation. Soliciting advance input gets people engaged and helps avoid "group think" during the review.

During the design review…

- Do check attitudes at the door. While it's easier said than done, don't be defensive about prior design decisions. Do embark on the review thinking change is possible; don't go in resigned to believing nothing can be done to improve the situation.
- Unless needed to support the review process, laptops and smartphones should also be checked at the door (at least figuratively). Allowing participants to check email during the sessions is no different than having them leave to attend an alternative meeting.
- Do exhibit strong facilitation skills. Review ground rules. Ensure that everyone is participating and communicating openly. Do keep the group on track; ban side conversations and table discussions that are out of scope or spirally into the death zone. There are tomes written on facilitation best practices, so we won't go into detail here.
- Do ensure a common understanding of the current model before delving into potential improvements. Don't presume everyone around the table already has a comprehensive perspective. It may be worthwhile to dedicate the first hour to walking through the current design and reviewing objectives. Don't be afraid to restate the obvious.

- Do designate someone to act as scribe, taking copious notes about both the discussions and decisions being are made.
- Do start with the big picture. Just as when you're designing from a blank slate, begin with the bus matrix, then focus on a single high priority business process, starting with the granularity then moving out to the corresponding dimensions. Follow this same "peeling back the layers of the onion" method with your design review, starting with the fact table, then tackling dimension-related issues. Do undertake the meatiest issues first; don't defer the tough stuff to the afternoon of the second day.
- Do remind everyone that business acceptance is the ultimate measure of DW/BI success; the review should focus on improving the business users' experience.
- Do sketch out sample rows with data values during the review sessions to ensure everyone has a common understanding of the recommended changes.
- Do close the meeting with a recap; don't let participants leave the room without clear expectations about their assignments and due dates. Do establish a time for the next follow-up.

Following the design review…

- Do anticipate that some open issues will remain after the 2-day review. Commit to wrestling these issues to the ground, even though this can be challenging without an authoritative party involved. Don't fall victim to analysis paralysis.
- Don't let your hard work gather dust. Do evaluate the cost/benefit for the potential improvements; some changes will be more painless (or painful) than others. Then develop action plans for implementing the improvements.
- Do anticipate similar reviews in the future. Plan to reevaluate every 12 to 24 months. Do try to view inevitable changes to your design as signs of success, rather than failure.

PPT:

Fact Attributes or Dimension Attributes?

Ø Fact Attributes

§ Numerical measurements

§ Pertain implicit time series of observations

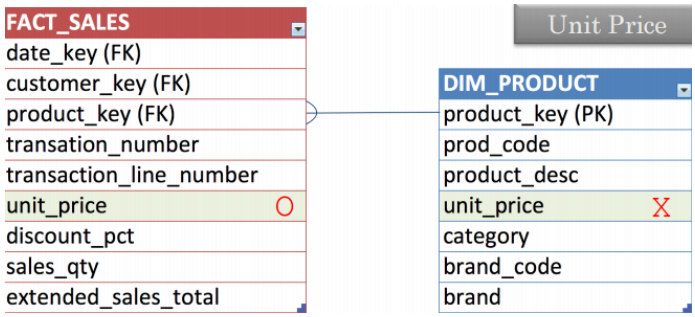
§ Participate in numerical computations (sum, averages, etc.)

Ø Dimension Attributes

§ Textual descriptors

§ Targets of constraints

§ Provide the content of "row headers" (grouping columns)

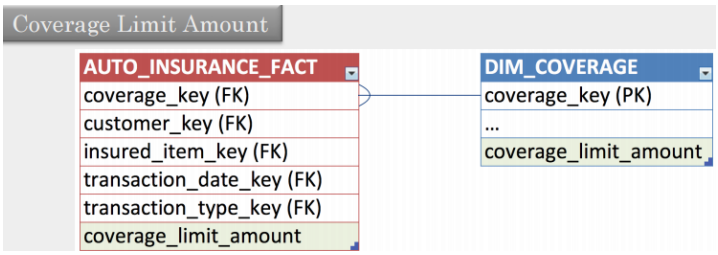


Ø The unit price varies over time and over location

Ø It is a rather rapidly changing, not a good fit for a SCD Type 2 dimension attribute

Ø It is not a good row header item as it is a continuous (not discrete) value

Ø Thus, it is a fact attribute



Ø The coverage limit participates as a query constrain

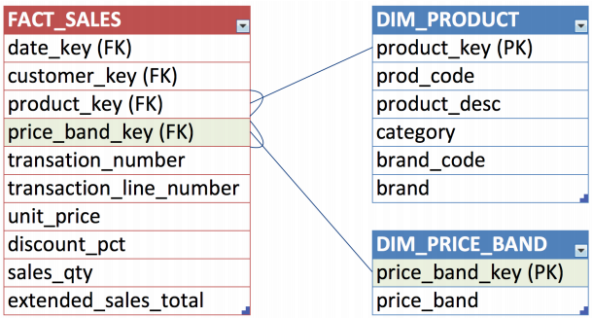
Ø It is generally a discrete value ($300k, $400k, ...)

Ø It slowly changes over time

Ø It participates in computations such as sum, average on policies and coverage

Ø For a truly continuous numeric dimension attribute, a value band can be an excellent alternative

§ unit price, GRE score, credit score, etc.

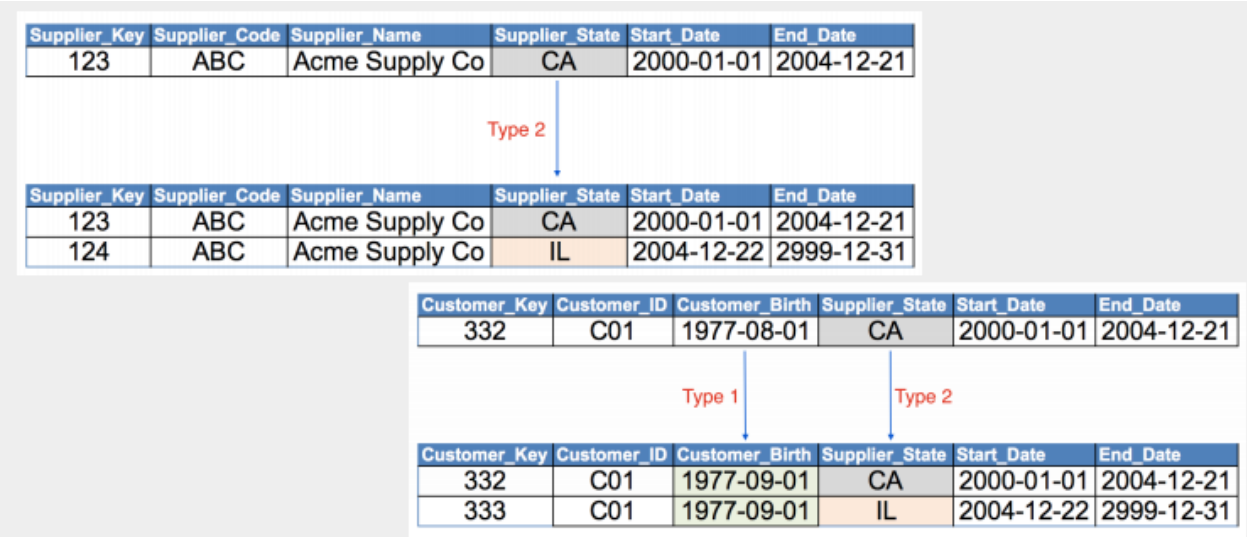


Dealing with Rapidly Changing Monster Dimensions: SCD Type 2 Revisited

Ø Changes in dimensions arrive

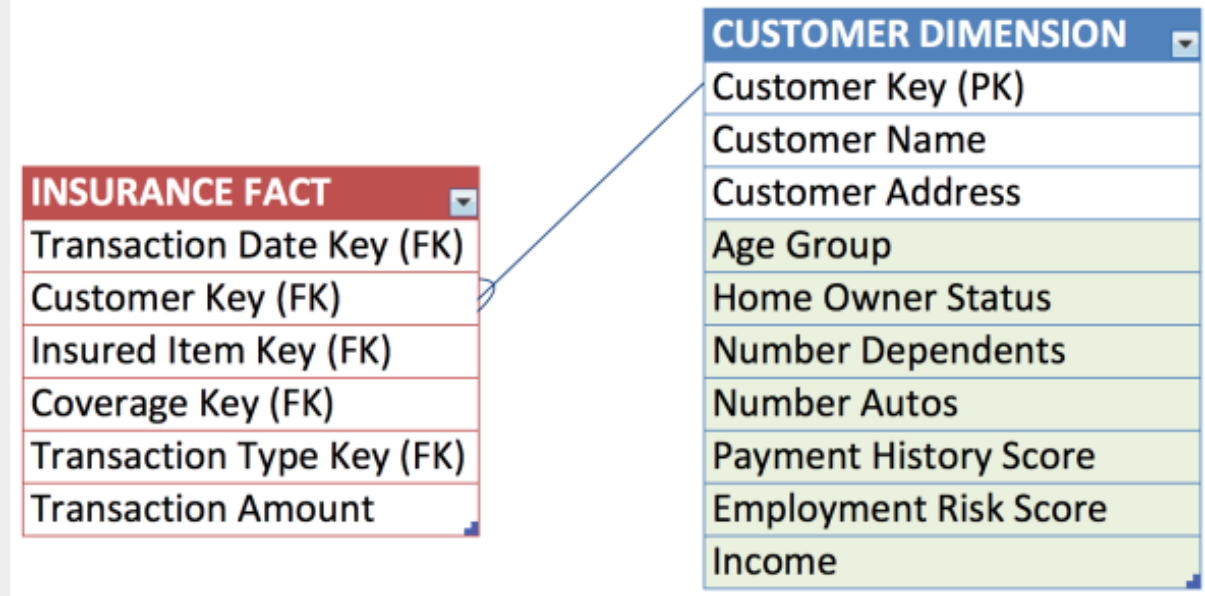
§ Far less frequently than fact table measurements → Slow changing

§ Type 2: Insert a new dimension row with the new data and new effective date



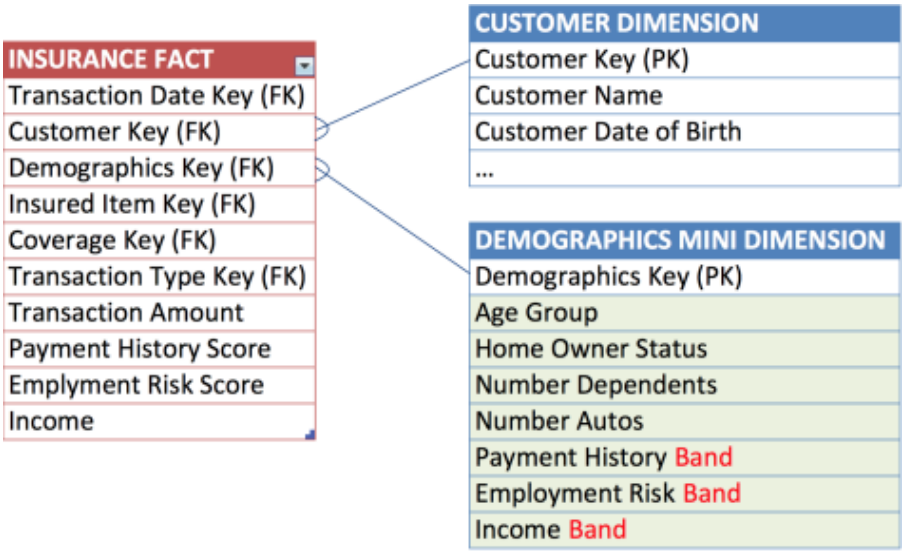
Dealing with Rapidly Changing Monster Dimensions: Monster Dimensions

Ø Imagine an insurance company with a big customer dimension (e.g. 30 million) with rapidly changing demographics (in green)



Ø The dimension table size can be easily doubled within a short period making this a rapidly changing monster dimension

Ø The solution is to break off the hot attributes into their own separate mini dimension



Ø The mini dimension contains one row for each possible combination of the attributes

Ø Value bands are used in the mini-dimension to reduce the number of rows overall

```
Customer dimension sample row:
Customer Key   Customer Name   Date of Birth
------------   -------------   -------------
123456         John Smith      1984-02-10

Demographics mini-dimension sample row:
Demographics Key  Age Group  Income Band
----------------  ---------  -----------------
1                 25-29      $50,000 - $59,999
2                 30-34      $50,000 - $59,999
3                 30-34      $60,000 - $69,999

Fact table sample row:
Transaction Date Key  Customer Key  Demographics Key
--------------------  ------------  ----------------
20140131              123456        1
20140228              123456        2
20140331              123456        2
20140430              123456        3
```

Outriggers

Ø Dimension tables joined to other dimension tables

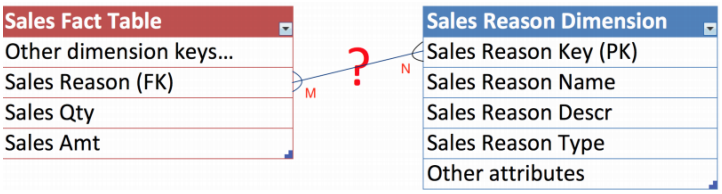
Ø In this case, a date dimension serves as an outrigger to the employee dimension via role-playing

Ø Outriggers are acceptable in moderation but should be viewed as the exception rather than the rule

Resolving Multivalued Relationships: Using Bridge Tables

Ø In a classic dimensional schema, each dimension attached to a fact table has a single value consistent with the fact table's grain

Ø But there are a number of situations in which a dimension is legitimately multivalued



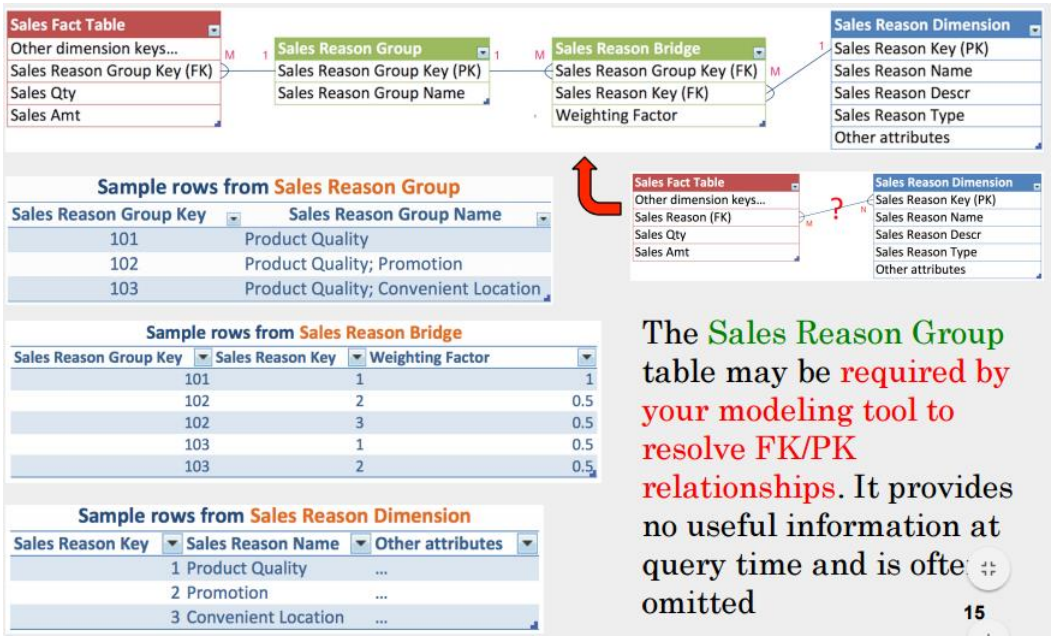
Ø Many sales reasons on a single transaction

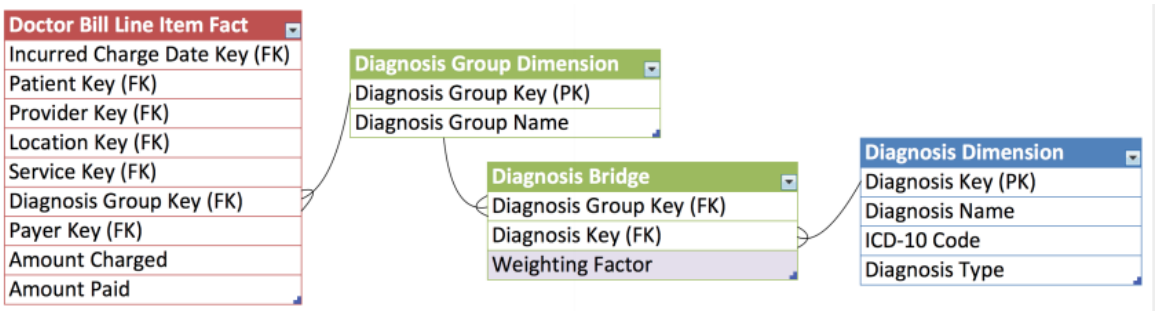
Ø Many customers in a bank account

Ø Many diagnoses at the time of a treatment

Ø Many witnesses to an accident

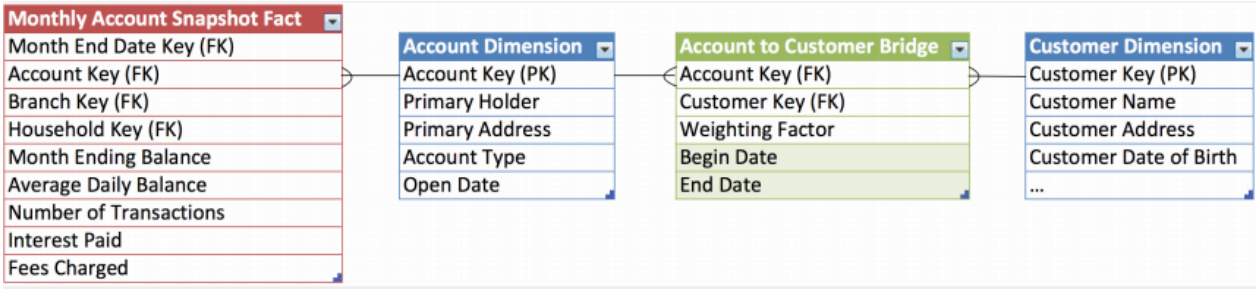
Ø Many options on a car





Ø The weighting factor is an explicit allocation

Ø Records in the Diagnosis Group Dimension can be made for each patient, but in this case it seems reasonable to re-use diagnosis groups, especially for out patient treatments where many groups would be repeated



Ø Associate customers to accounts where these have a many-to-many relationship

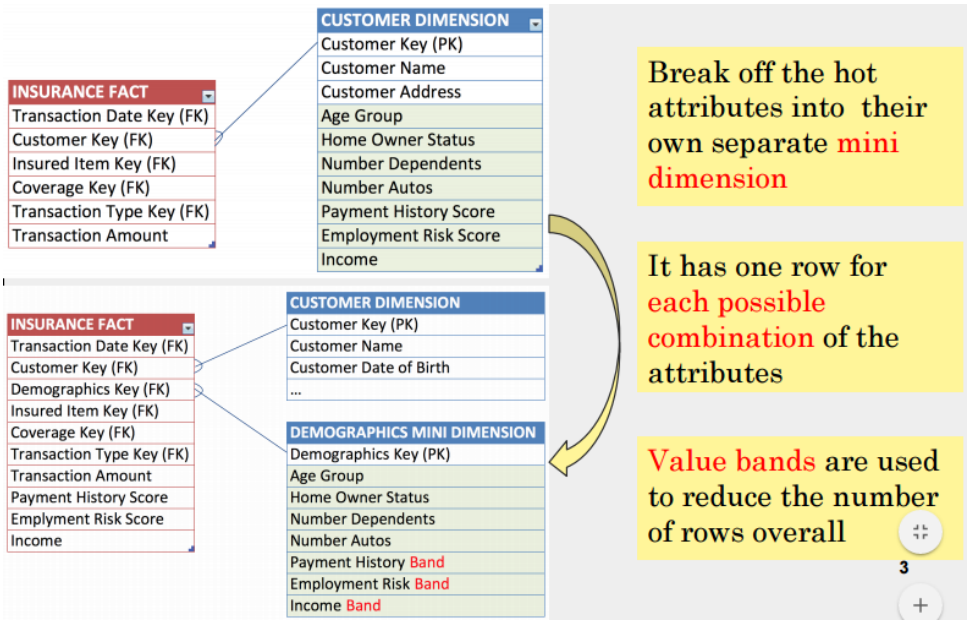
Ø Query account balances by individual customer or groups of customers

Ø Show account balances correctly weighted (prorated) by individual customers to avoid double counting

Ø Show account balances by customer "impact" (unweighted)

Week 08 Dimensional Modeling: Dealing with Hierarchies

Dealing with Monster Dimensions: Mini-Dimension to the Rescue

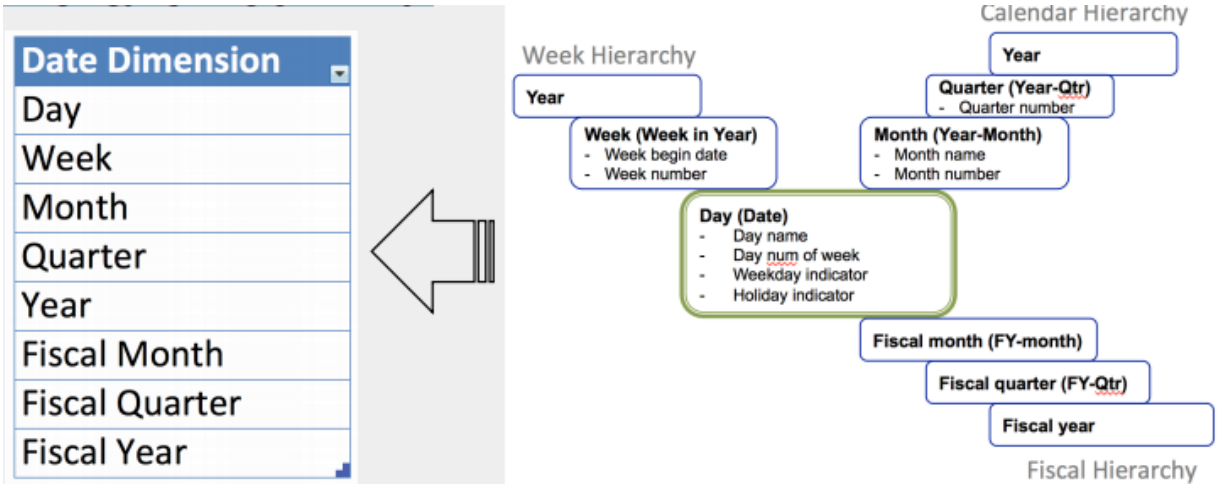


Dealing with Hierarchies: Fixed Depth Positional Hierarchies

Ø A series of many-to-one relationships

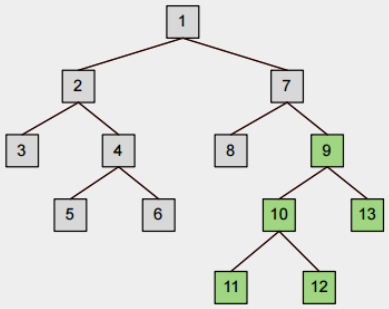
Ø The hierarchy levels become separate positional attributes in a dimension table

Ø A dimension can have multiple fixed depth hierarchies in it



Dealing with Hierarchies: Ragged/Variable Depth Hierarchies

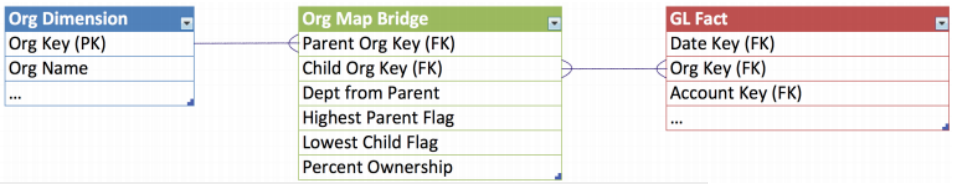
- Ragged hierarchies of indeterminate depth are difficult to model and query in a relational database
- A specially constructed bridge table to the rescue



Parent Key	Child Key	Depth From Parent	Highest Parent Flag	Lowest Child Flag
9	9	0	FALSE	FALSE
9	10	1	FALSE	FALSE
9	11	2	FALSE	TRUE
9	12	2	FALSE	TRUE
9	13	1	FALSE	TRUE
10	10	0	FALSE	FALSE
10	11	1	FALSE	TRUE
10	12	1	FALSE	TRUE
11	11	0	FALSE	TRUE
12	12	0	FALSE	TRUE
13	13	0	FALSE	TRUE

Ø Child element references to the fact

Ø Parent element references to the dim




```
SELECT SUM(f.balance * m.percent_ownership)
  FROM fact_gl f,
       bridget_org_map b,
       dim_org d
 WHERE d.org_name       = 'Divisional Rollup'
   AND f.org_key        = b.child_org_key
   AND d.org_key        = b.parent_org_key
```

Kimball Lifecycle Approach