

LABORATORY WORK BOOK

For The Course

SE-403 Data Warehouse Methods



Name : _____

Roll No. : _____

Batch : _____

Year : _____

Dept. : _____

Department of Computer Science and Information Technology
N.E.D. University of Engineering & Technology, Karachi –75270, Pakistan

LABORATORY WORK BOOK
For The Course
SE-403 Data Warehouse Methods

Prepared By:
Mr. Umar Farooq

Approved By:

Chairman

Department of Computer Science and Information Technology

Data Warehouse Methods

CONTENTS

Task. no.	List of Experiments	Page no.
	<u>To design a Warehouse Model</u>	
01	<i>To understand the steps for scrubbing data.</i>	04
02	<i>Map source data to subject area</i>	09
03	<i>Verify data integrity</i>	18
04	<i>Map subject areas to the warehouse model</i>	24
05	<i>Modeling the Virtual Data Warehouse</i>	37
06	<i>Identify different kinds of temporal data and construct temporal queries</i>	40
07	<i>To Design a Dimensional Model</i>	58
08	<i>Retail Case Study</i>	66

Objective: To Design Warehouse Model.

Task # 1

To understand the steps for scrubbing data.

Theory

Introduction

The first step in designing a data warehouse is to identify and analyze data that will be stored in the data warehouse. Before extracting data from the operational systems to a data warehouse, data needs to be validated and corrected. In addition, data should be transformed and mapped to the subject area model of a data warehouse. All transformations and conversions of data need to be well documented. Migration of data could lead to problems, such as inconsistencies between the extracted and original data. Apart from these issues, integrity of data needs to be checked and verified. Various tools are available in the market for solving problems related to transformation and migration of data to data warehouses.

Source Data

The operational data level of the architected data warehouse environment provides the source data for the enterprise data warehouse. This level consists of the online transaction systems. Identification and analysis of source data requires documenting the structure of source data and mapping it to subject area models. It also requires procedures to detect and clean incorrect and inconsistent data.

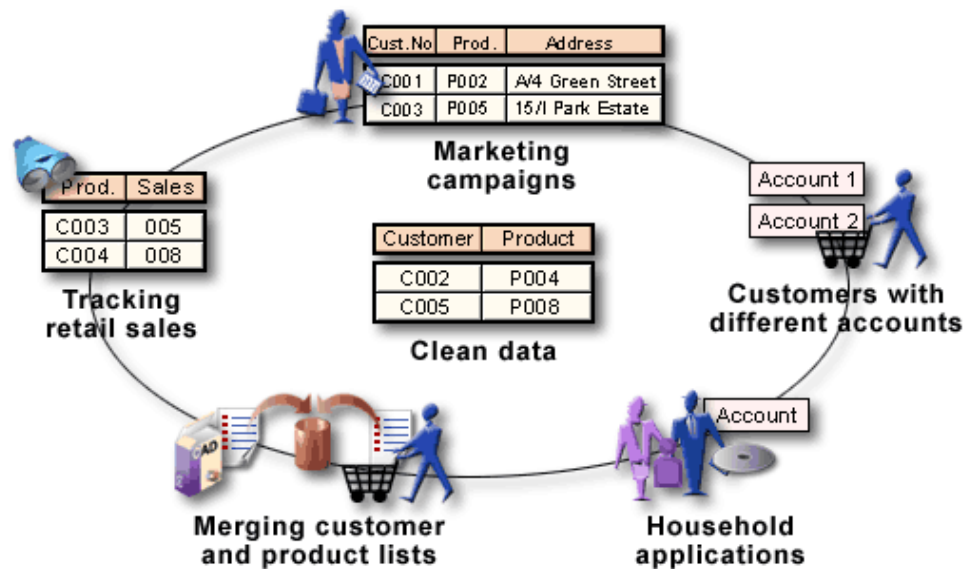


Dirty Data

Data that is incorrect, inconsistent, or badly structured is called **dirty data**. Business analysis, if performed on dirty data, results in erroneous decision-making and can be the undoing of an organization. The applications listed below show the importance of clean data.

- Marketing campaigns require accurate data to generate selective customer lists and avoid missing or duplicating contacts.

- Accurate data is needed for identifying customers with different accounts.
- Household applications require accurate information to identify customers in the same family, or subsidiaries of a parent company, to facilitate cross-selling.
- Merging customer and product lists from internal and/or external sources requires clean data.
- Tracking retail sales requires accurate product descriptions.

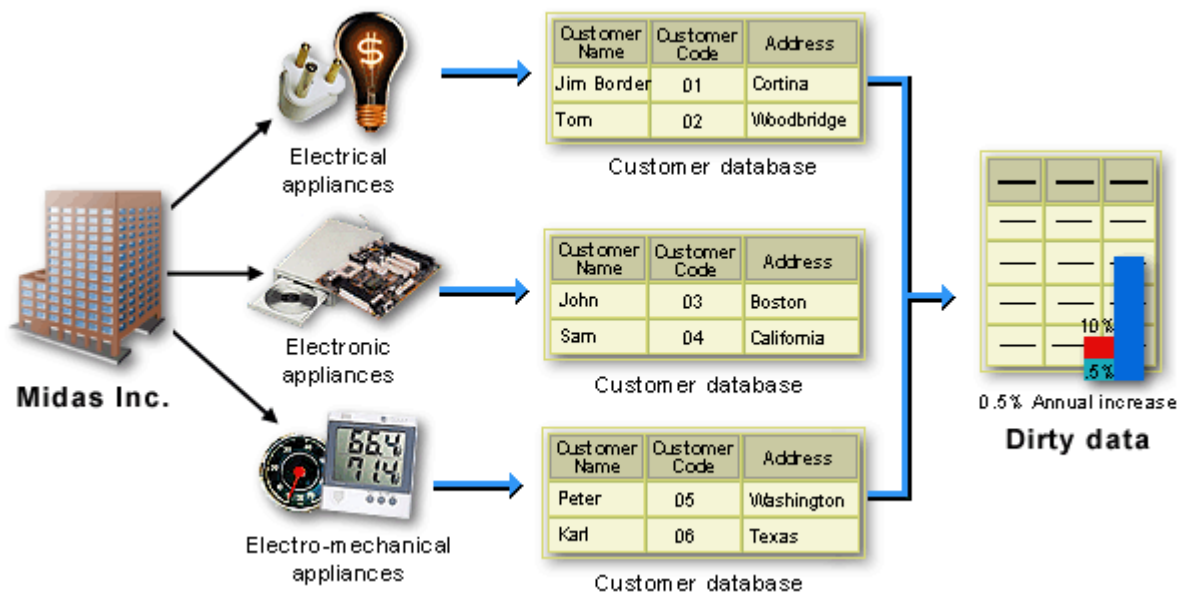


Exercise:

The Cost of Dirty Data

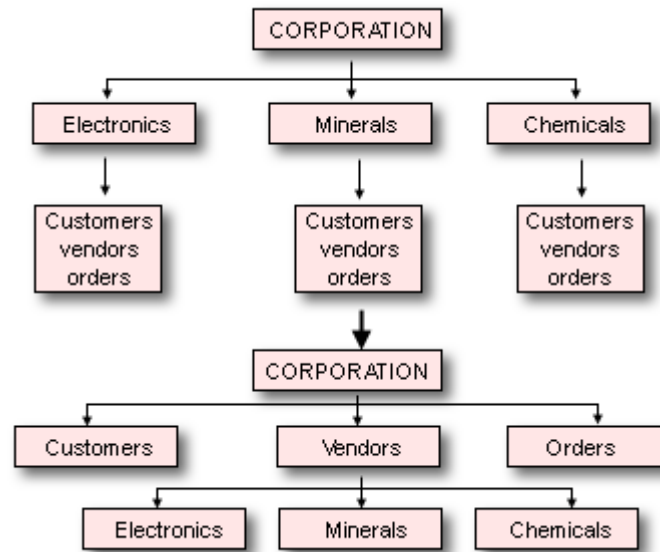
Consider, **Midas Inc.**, a manufacturer of domestic appliances that has a strong direct sales network. It has three product groups - electrical appliances, electronic appliances, and electro-mechanical appliances. Each product group maintains a customer database of its own. The total number of customers is one million. To achieve its aggressive growth targets, **Midas** plans to sell products of all three categories across all the customers. It estimates that **10%** of the customer file is inaccurate, of which **5%** can be repaired, resulting in **0.5%** annual increase in active customers. It further estimates that the annual value of selling products of all three categories to all the customers is anywhere between **\$100** and **\$1000** per customer. It concludes that with a customer base of one million and a **0.5%** annual increase in active customers made possible by cleaning the data, the loss due to dirty data is **\$500,000** to **\$5M** annually.

Similar analysis applies to products, sales, medical procedures, and other types of data.



Data Scrubbing

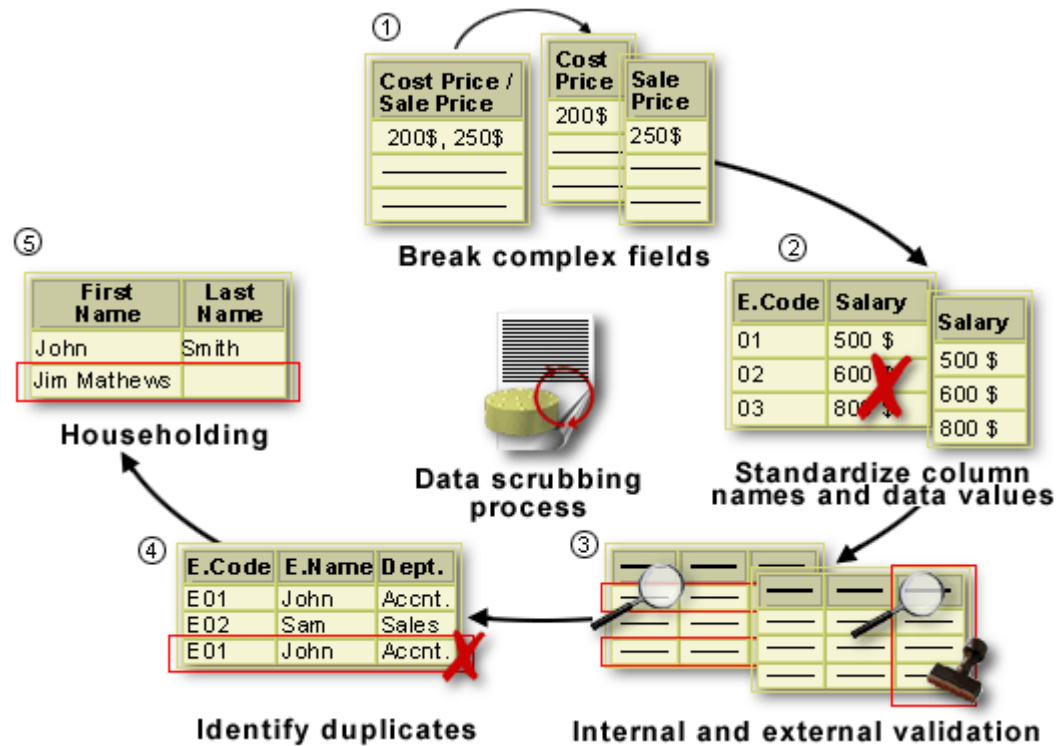
The process of restructuring data by content and cleaning data is called data scrubbing. In most companies, each branch manages customer, vendor, and order data separately. This prevents cross-selling of products from one branch to customers of another. Such companies can easily organize their data by content. For instance, they can divide the data according to subjects, such as customer, vendor, and order. This data can be further categorized by products.



The Data Scrubbing Process

Organizing data by content results in a subject area model. In other words, it achieves the first stage of the model synchronization problem, that is, mapping source documentation to subject areas. After organizing the data, further data scrubbing involves the following five steps.

1. Break complex fields into simple columns that contain just one kind of information.
2. Standardize both column names and data values. Replace confusing abbreviations with standardized abbreviations.
3. Validate the data. Validations can be external and internal. External validation checks that a record does not conflict with any other related records in the database. Internal validation checks that all the fields within a record are correct and consistent.
4. Identify duplicates, condense these to one record, and assign a primary key value.
5. Group related records under a single value. This is called householding.



Question # 1:

Data scrubbing is exclusively used for further refining the clean data.

- ☐ True
- ☐ False

Question # 2:

Which one of the following options is a characteristic of scrubbed data?

- ☐ Complex fields
- ☐ Standard Abbreviations
- ☐ Duplicate Records

Task # 02.

Map source data to subject area

Theory

Mapping Source Data to Subject Area

The data scrubbing process is a subset of a larger process, that is, of mapping source documentation to subject area models. The subject area model can be further mapped to the warehouse model. The warehouse model in turn needs to be mapped to the dimension model.



Mapping Source Data to Subject Area

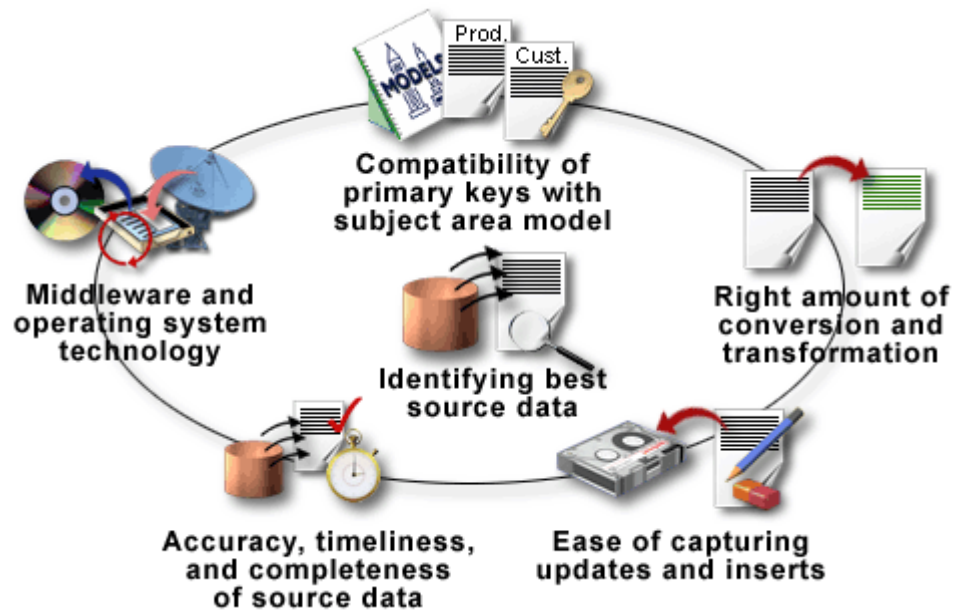
Mapping source data to subject areas involves the nine tasks listed here.

1. Identify the best source data.
2. Unpack overloaded fields.
3. Normalize tables.
4. Standardize column names.
5. Household related data.
6. Create a common primary key.
7. Document functional transformations.
8. Capture delta data.
9. Check data integrity.

Identifying the Best Source Data

The first task while mapping source data to subject area is to identify the source data. When there are several alternative data sources to choose from, consider the following issues.

- compatibility of primary keys with subject area models
- right amount of conversion and transformation
- ease of capturing updates and inserts
- accuracy, timeliness, and completeness of source data
- middleware and operating system technology

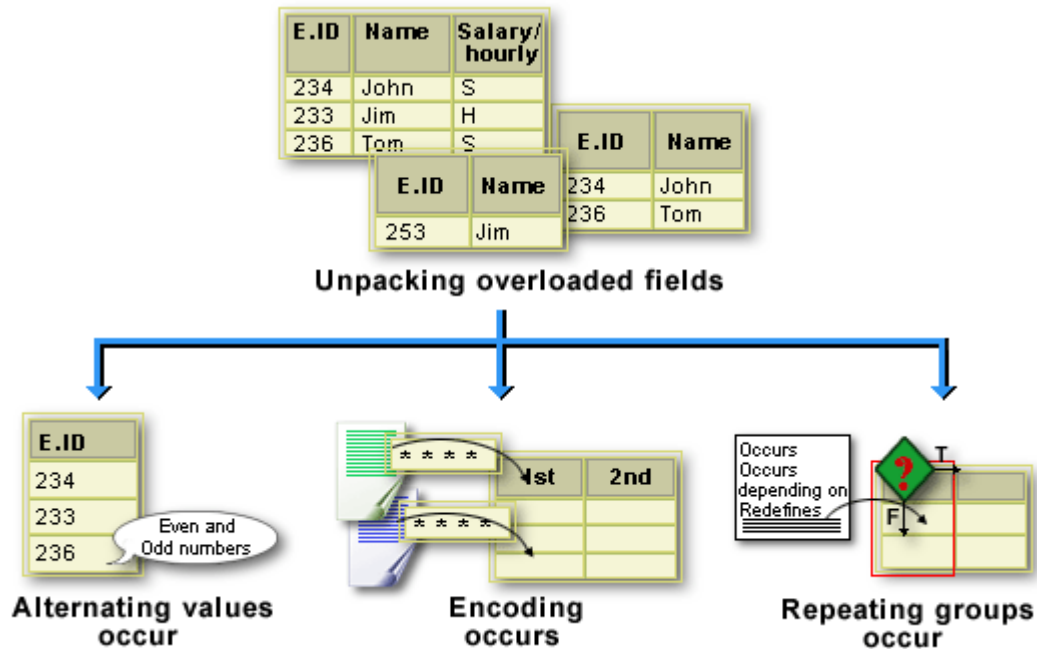


Unpacking Overloaded Fields

An overloaded field contains more than one type of data. It might occur in any one of the following three ways.

- **Alternating values** occur when data with slight differences is stored in the same field for different records.
- **Encoding** occurs when several kinds of data are stored in one field as a result of string concatenation or numeric encoding.
- **Repeating groups** occur when several values are stored for one logical field, for instance, the values **Occurs**, **Occurs depending on**, and **Redefines** might be stored for a field called **Occurrence**.

Regardless of the specific type of overloading, you must extract individual values from the overloaded field, and store each value in separate columns of the subject area model.



Exercise:

Overloaded Fields

Consider **FastPlace, Inc.**, a software development agency that is in the process of mapping its source documentation to the subject area model. During this process, two instances of overloading are discovered. In the first instance, data with slight differences is stored in the same field for different records. In the **Employees** table, the **Quantity** field signifies the lines of code written by a programmer. However, for salespersons, the **Quantity** field signifies the amount of orders converted by a salesperson.

Employee_ID	Job_type	Quantity
E1	Programmer	123
E2	Salesperson	3
E3	Programmer	345

Quantity means lines of code for programmer but orders for salesperson

Overloaded Fields

The second instance of overloading in the **FastPlace** database occurs due to encoding. Different kinds of data are concatenated or encoded in the same field of a record. In this instance, the **Employee_ID** field contains three numeric characters. The third numeric character is used to denote a salaried employee, if the number is even. If the number is odd, the field denotes an hourly employee.

Employee_ID	SS_number	Full_name
234	123-45-56756	Charlie Chapman
233	678-67-68790	J Smith
236	111-22-24567	Charence

Even numbers denote salaried employees. Odd number denotes hourly employee.

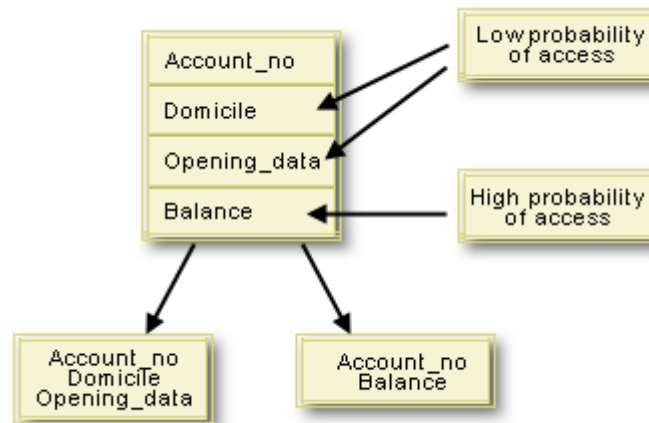
Overloaded Fields

The solution to the problem of overloading fields is to divide an overloaded field into logical fields. For instance, the **Quantity** field in the **FastPlace** database can be divided into the **Order_amount** field and the **Code_lines** field. Also, logical components of a concatenated field can be separated into different fields.

Employee_ID	Job_type	Order_amount	Code_lines
E1	Programmer	Null	123
E2	Salesperson	3	Null
E3	Programmer	Null	345

Normalizing Tables

Normalization means designing tables that have little or no redundant data. In addition, data that have high probability of access are stored separately from data that have less probability of access. For instance, a table might contain the **Account_no**, **domicile**, **opening_data**, and **balance** fields. The **balance** field has a very different probability of access than the **domicile** and **opening_data** fields. The **balance** field is frequently accessed while the other fields are hardly accessed. Therefore, to make the data access more efficient and to store the data more compactly, the table needs to be separated into two. One of the tables will contain the **Account_no**, **domicile**, and **opening_data** fields. The second table will contain **Account_no** and **balance** fields.



Creating a Common Primary Key

To create a common key, you need to map both column names and values of the source data to a common primary key in the subject area model.

If there are several primary keys in the source data, choose one that is stable, short, and concise. Usually, numbers or mnemonics work best. If all the source primary keys are lengthy or unstable, create an artificial primary key, such as an integer field. It is usually a good idea to keep a record of the values of source keys, and the corresponding values of subject area keys. This helps the warehouse administrator trace the source of warehouse data and isolate integrity problems.

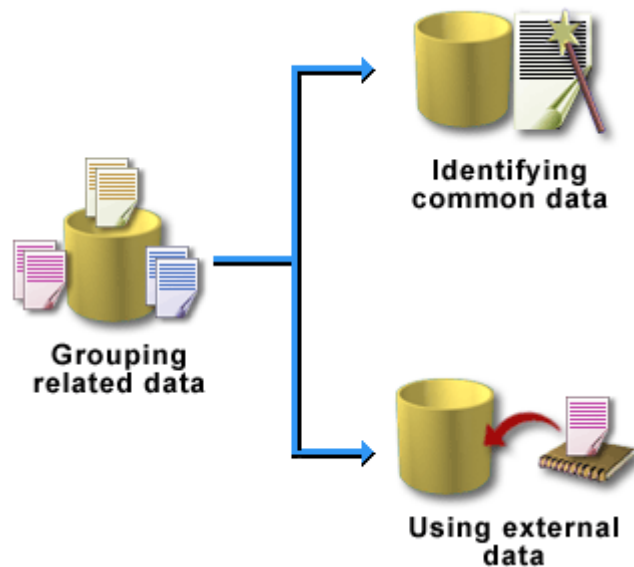
UNIVERSAL_ CUSTOMER_KEY TABLE			
Customer_ID	Billing_num	Cust_Acct	Reg#
1	232	142-578925	GT88
2	234	142-545725	RT34
3	235	142-578926	WE56

Householding Related Data

To group related data:

- look for common data, such as names, addresses, tax identification numbers
- use external data such as property titles, marriage, and birth records

Data scrubbing tools, such as the Integrity Data Reengineering Tool from Vality Technology aid complex tasks like standardizing values and householding.



Standardizing Column Names

Most data administrators use a three-part naming convention to standardize column names and values. If your organization has no naming conventions, take some time to develop and document them. Without naming conventions, it is difficult to achieve integration and easy access required in a warehouse system.

Standardizing column values is difficult, because there are many more values in a database than column names. To standardize column values:

- map codes to standard values, such as state code and name suffixes
- map character strings to common spelling, such as first names and company names
- Standardizing Column Names
- The graphic displays a table containing fields for storing naming conventions and their description as an example.

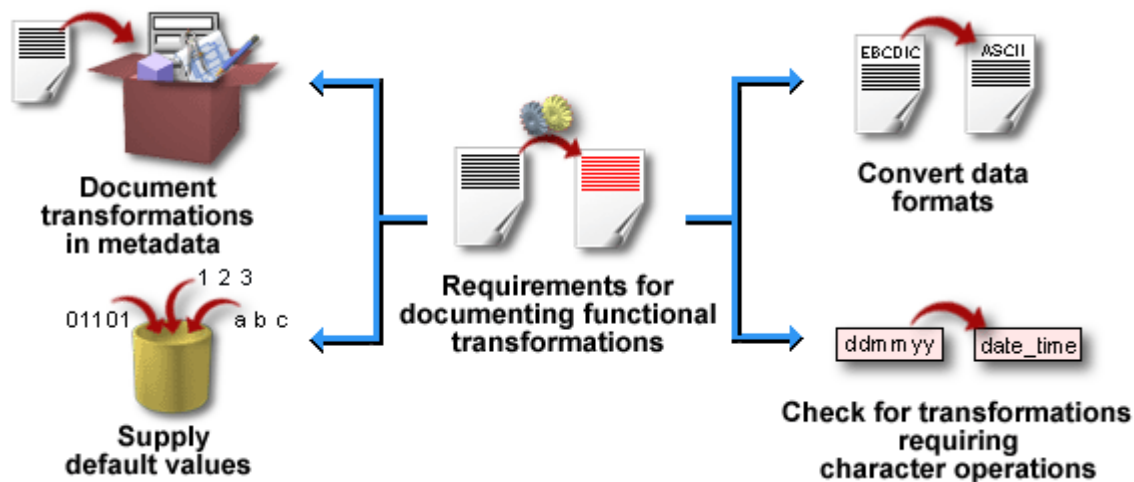
Employee_Id	Project_ID	Job_type	Job_level	Name
E1	P1	Programmer	Sr.	Alice Phillips
E2	P2	Agent	Jr.	Melee Ohr
E3	P3	Project manager	Sr.	Gracie Gnomes

Standardizing column names

Documenting Functional Transformations

Functional transformations include numeric transformations, such as converting Fahrenheit to Celsius, and textual transformations, such as date conversions. The simplest functional transformations map one source field to one subject area column. More complex transformations are one-to-many, many-to-one, or many-to-many. To document these transformations comprehensively, you need to:

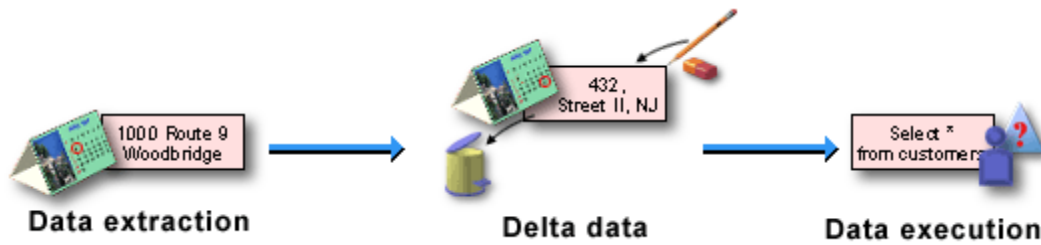
- document all transformations in the metadata of the warehouse
- supply default values when source values are not available
- convert data formats, such as EBCDIC to ASCII
- check for some transformations that may require character operations, such as **ddmmyy** to **date_time**



Delta Data

While taking decisions based on data, it is very important that data is correct at the time of execution of the decision taken based on that data. However, data can change between extraction and execution. Such changed data within source data is called **delta data**. The changes are incremental and can be in terms of inserts, deletes, and updates. For instance, assume that you extracted the address of a customer at midnight. If that customer changes residence the next morning, the new address is termed delta data. This delta data must be loaded to the warehouse database.

The problem with delta data is distinguishing between new and old information. If you are not able to differentiate between the two, you have to reload the entire warehouse database every midnight. This is inefficient at best, and more likely, impossible. Your options are either to scan the source data or develop routines for identifying delta data.



Question # 1

Explain the steps of capturing the delta data and draw the architecture of capturing the delta data.

Answer

Question # 2

An overloaded field can occur due to:

- ☐ Encoding
- ☐ Householding
- ☐ Alternating Values
- ☐ External Validations

Question # 3

Normalization optimizes tables by enabling you to remove redundant data.

- ☐ True
- ☐ False

Question # 4

Which one of the following convention types refers to the logical data type of a column?

- ☐ Root
- ☐ Class
- ☐ Qualifier

Question # 5

Delta data:

- ☐ groups related records under a single value
- ☐ contains more than one type of data
- ☐ contains changes after extraction to the data warehouse

Task # 03.

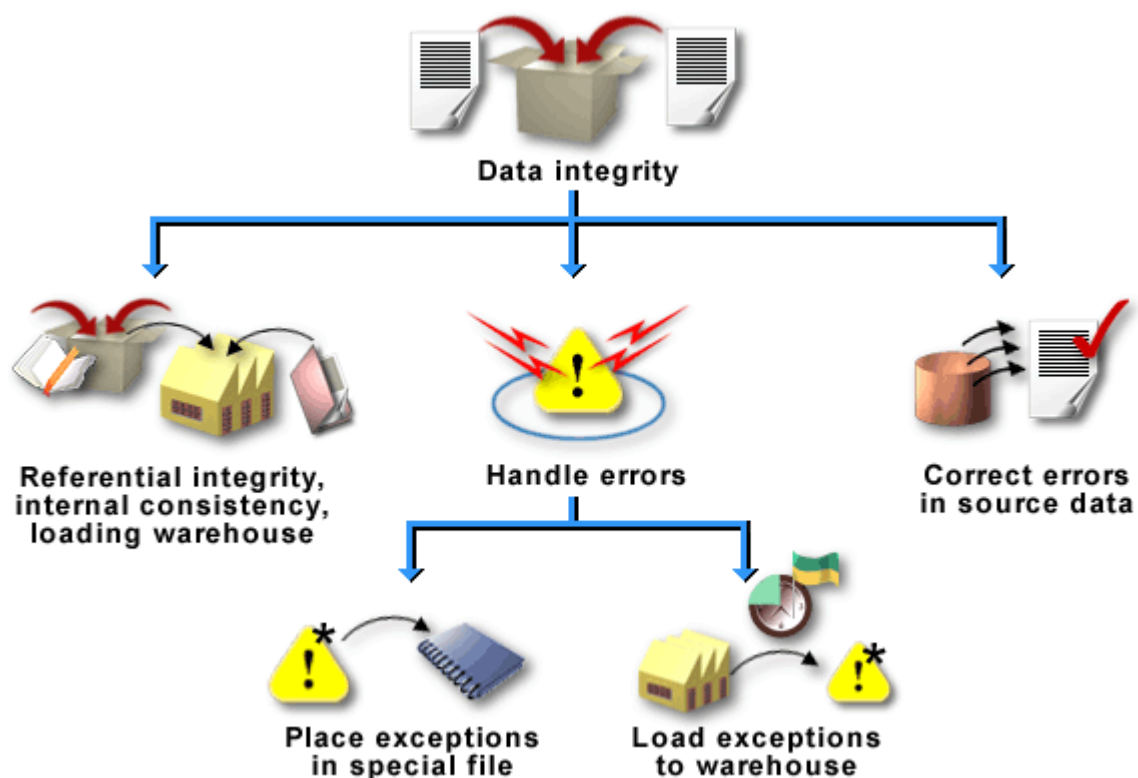
Verify data integrity

Theory

Checking Data Integrity

Data integrity is a broad issue that includes structural aspects of a database, such as [referential integrity](#) and primary key uniqueness. It also includes conformance to an organization's business rules and to external rules, such as consistency of zip code and state. You can check for data integrity using the following guidelines.

- Check referential integrity, internal consistency, and other business rules while loading the warehouse.
- Apply one of the following ways to handle errors.
 - Place exceptions in a special file.
 - Load exceptions to the warehouse with a temporary **exception** flag. This is usually the best alternative since it enables correction within the warehouse database.
- Always correct errors in source data before loading for consistency.

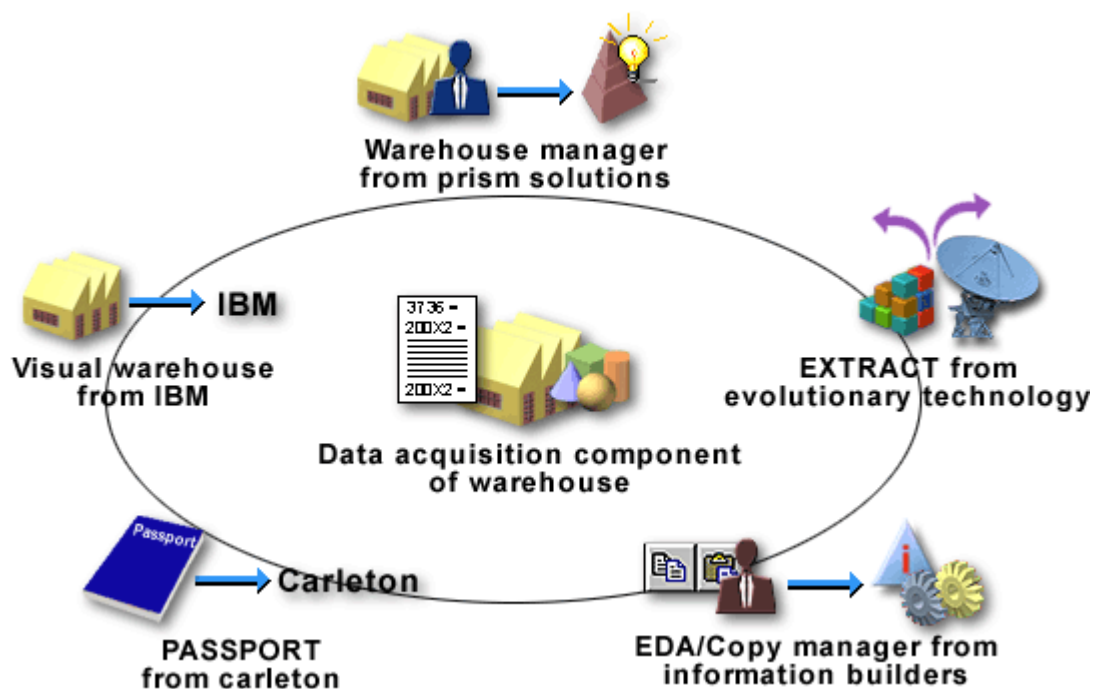


Data Mapping Tools

Data extraction and transformation tools extract data from source systems and help transform the data to an enterprise warehouse or data mart. The following products correspond to the data acquisition component of the warehouse roadmap.

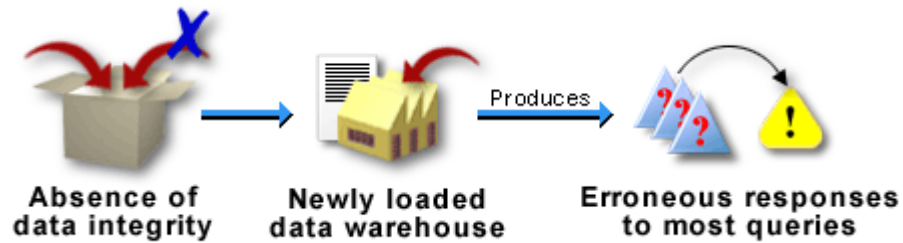
- Warehouse Manager from Prism Solutions
- EXTRACT from Evolutionary Technology
- EDA/Copy Manager from Information Builders
- PASSPORT from Carleton
- Visual Warehouse from IBM

Most of these products use the [compromise approach](#) and transform source data directly to the warehouse model.



Importance of Verifying Data Integrity

In the absence of data integrity, newly loaded data warehouses usually produce erroneous responses to most queries. For instance, consider the case of a company whose performance depends on the accuracy of customer data. Typically, the biggest customers form a small percentage of the total number of customers. However, most of the queries would access data relating to this small group of customers. Therefore, even a small percentage of dirty data around these customers will compound to invalidate the information.



Exercise:

Cost of Ignoring Integrity Issues

Consider a bank, **FixDep Bank** that loads its source customer, business, and financial data to its warehouse on the last working day of every month. This data is loaded from various systems including their credit processing system. The credit processing system contains an entry that marks a customer's deposit as loan. Due to this erroneous data that remained unchecked, the bank sent the customer a debit note instead of crediting his account with the interest owed to him. Thus, the bank lost a valued customer.

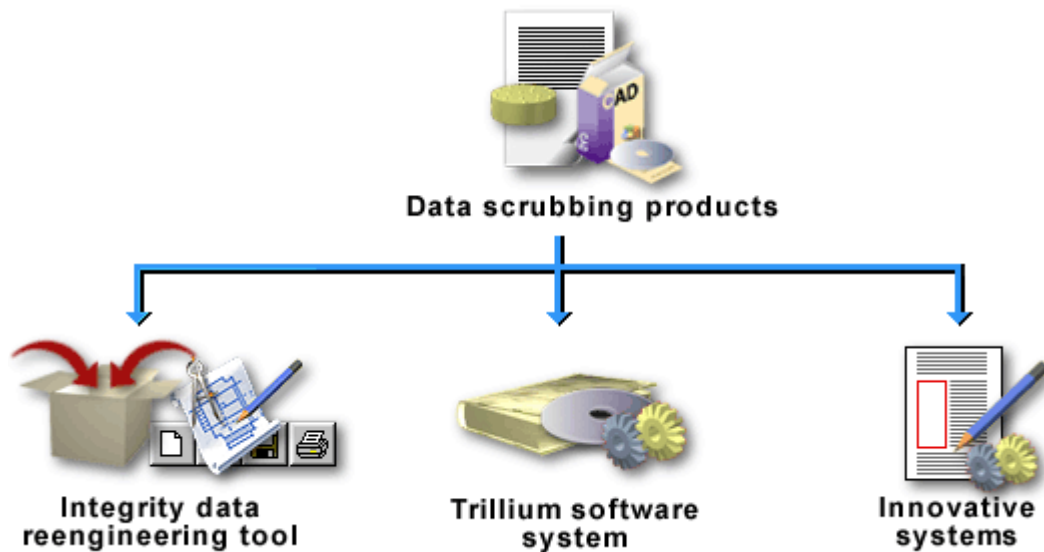
FIXED DEPOSIT BANK			
Cust ID	TR ID	Last Upd	Tot int
WER	D	05/04/99	4500.00
WER	L	06/04/99	4500.00
WER	D	06/04/99	0.00

Erroneous data

TR ID	TRANS TYPE
L	Loan
D	Deposit

Data Scrubbing Products

Along with checking of data integrity, most data scrubbing products apply the five basic scrubbing steps, but they differ significantly in their methodologies. The tools depicted in the graphic are widely used for data scrubbing.



Integrity Data Reengineering Tool

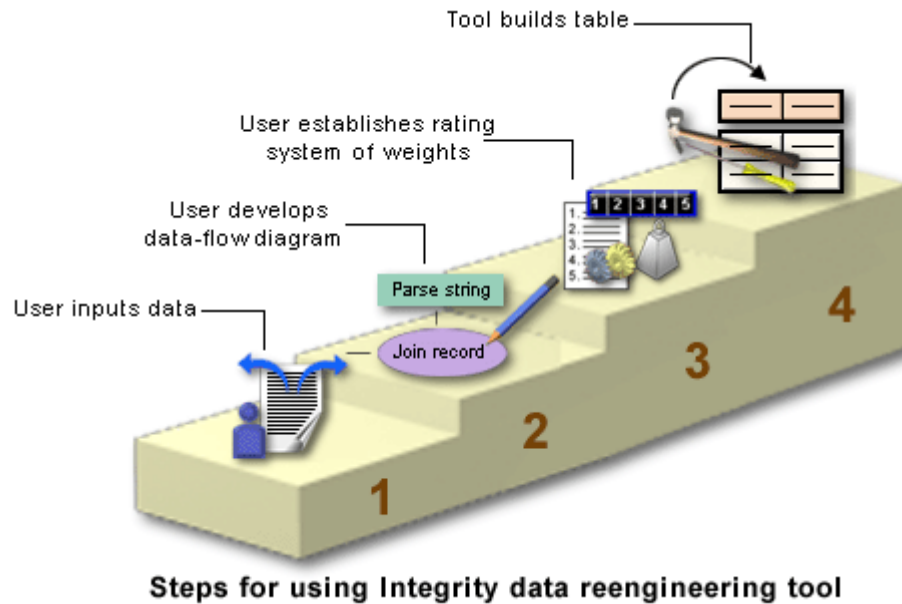
The **Integrity data reengineering tool** from Vality Technology enables the developer to specify a procedure for ranking data elements using the product's standard functions.



Using the Integrity Data Reengineering Tool

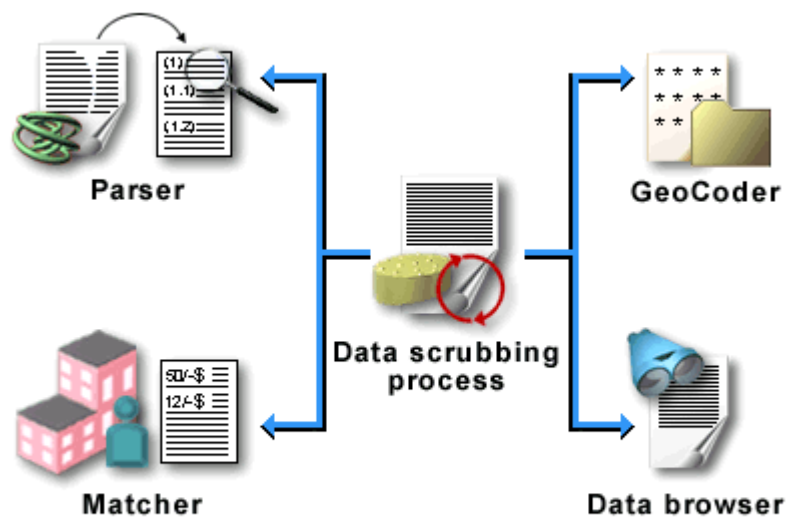
You can use the Integrity data reengineering tool to investigate, standardize, transform, and integrate data from multiple operational systems and external sources. The tool works as shown in the following steps.

1. The user inputs data as flat files.
2. The user develops a **data-flow diagram** of scrubbing steps from a toolkit of operators like **parse string** or **join record**.
3. The user establishes a rating system of weights, penalties, and thresholds. The tool merges strings that score above a threshold.
4. The tool builds a table that maps source strings to standardized values based on ratings.



Verifying Clean Data with Trillium

The **Trillium software system** has utilities for data discovery, lexical analysis, data scrubbing, and data transformation. Its components perform all five steps in the data scrubbing process. Click each component for a description of its function.



Verifying Clean Data with Innovative Systems

Innovative systems offers the following data scrubbing tools. Click each tool for a description of its function.

Question # 1

Data extraction and transformation tools represent the structural aspects of a database, such as referential integrity and primary key uniqueness.

- ☐ True
- ☐ False

Question # 2

The Integrity data reengineering tool is a data extraction and transformation tool.

- ☐ True
- ☐ False

Question # 3

Which one of the following Trillium components helps quality assurance staff inspect data and analyze integrity?

- ☐ Parser
- ☐ Coder
- ☐ GeoCoder

Task # 04.

Map subject areas to the warehouse model

Theory

Designing Enterprise Warehouse

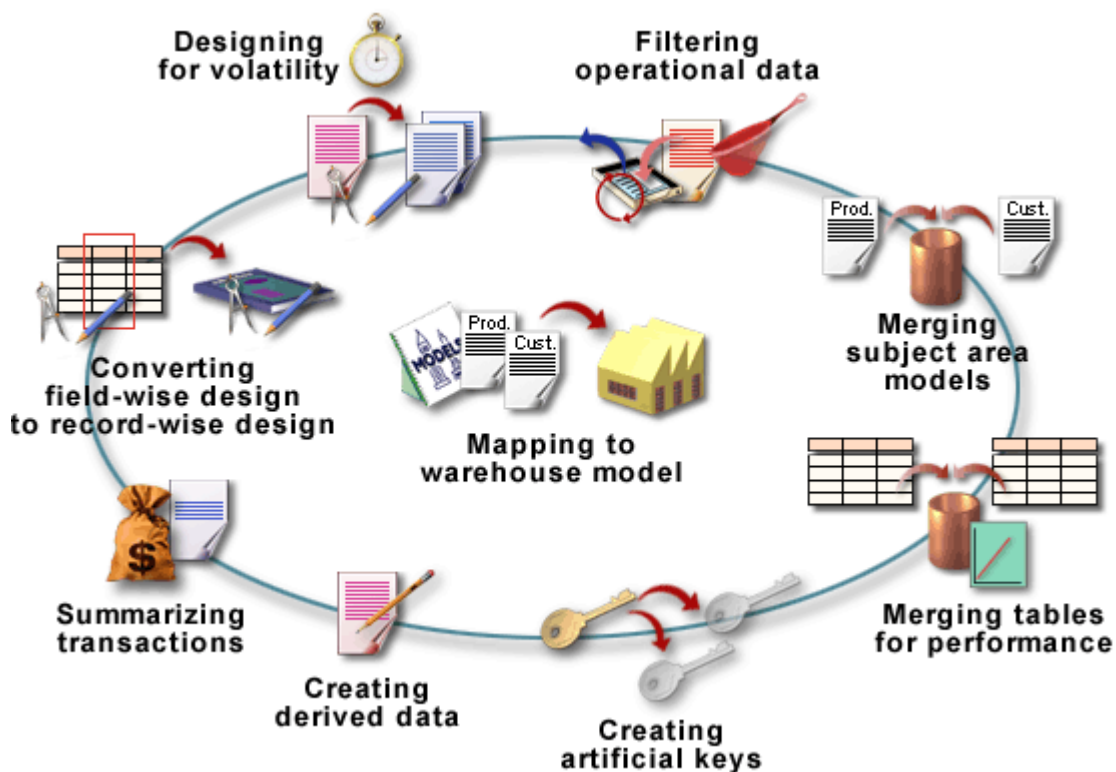
After source data is mapped to subject areas, the warehouse data is mapped to the warehouse model. The design of enterprise data warehouses is based on the warehouse model. The design of a virtual data warehouse is based on an intermediate model, which is midway between subject area and warehouse models. You should design a virtual data warehouse, if your analysis needs current operational data, instead of historic data.

At the end of this topic, you will be able to:

- map subject areas to the warehouse model
- describe the virtual data warehouse model

Mapping to the Warehouse Model

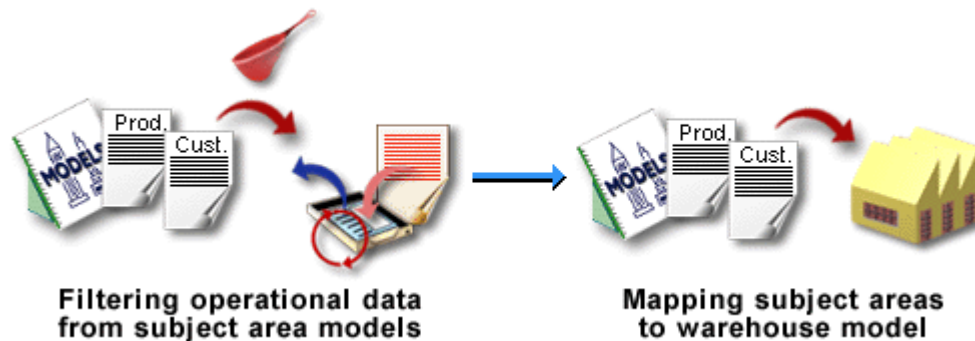
In designing the enterprise warehouse model, after you have mapped source data to subject area models, you need to map subject area models to the [warehouse model](#). This involves eight basic transformations as shown in the following graphic. **Warehouse model** is an integrated, denormalized, summarized, and historic version of the subject areas.



Filtering Operational Data

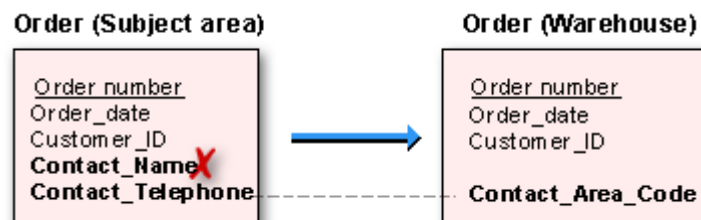
While mapping subject areas to the warehouse model, you need to first remove or filter operational data from the subject area models. Many fields or parts of fields, in subject area models contain only operational data, and are not useful to a business analyst. You need to eliminate these unnecessary fields from the warehouse model, to simplify the enterprise warehouse.

In some cases, information useful to business analysts is hidden in operational data. While filtering operational data, you need to extract such information and include it in the warehouse model.



Useful Information in Operational Data

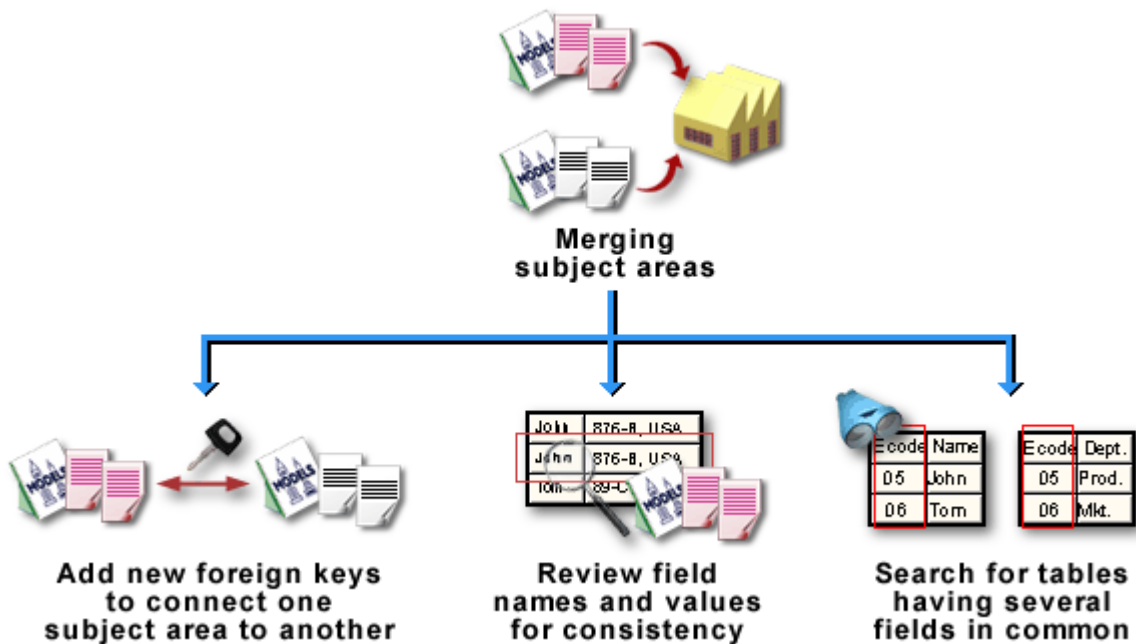
Consider **Christine**, a DSS architect, who is filtering operational data from the **Order** subject area. She finds that the **Contact_Name** field is not useful in business analysis. However, the **Contact_Telephone** field containing the area code is useful for demographic analysis of buying patterns. Therefore, she extracts the area code from the **Contact_Telephone** field and stores it in a field called **Contact_Area_Code** in the warehouse data.



Merging Subject Areas

After filtering operational data, you need to merge the subject areas. While combining several subject area models in an integrated warehouse model, the following three points might be useful.

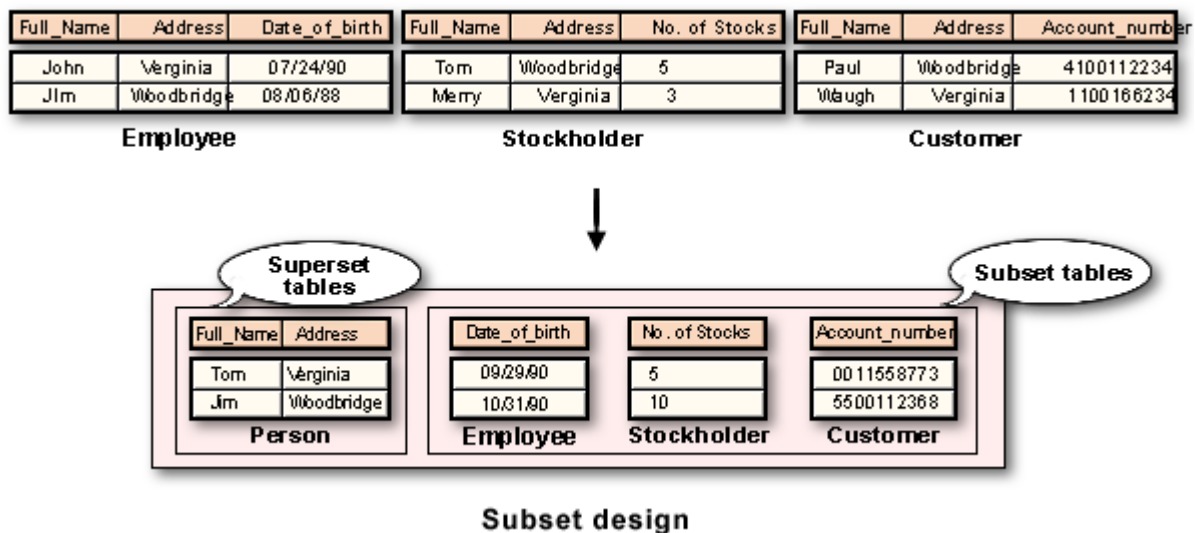
- Add new foreign keys to connect one subject area to another. For instance, when you merge the **Product** and **Department** subject areas, you might add a foreign key, **Dept_Code** to the **Product** subject area. This foreign key indicates the department that manages each product.
- Review field names and values for consistency between different subject areas, and revise them if necessary. Pay particular attention to the names of primary and foreign keys. Since keys are critical to the structure of a database, it is important that their names are consistent.
- Search for similar tables, such as tables that have several fields in common. Move the common fields to a new table, thus creating a **superset table**. Leave the other fields in **subset tables**.



Exercise:

Subset Design

Consider an organization, **BigFunds Inc.**, which has independent subject area models for its employees, stockholders, and customers. The three subject area tables are similar, as all of them contain the **Full_Name** and **Address** fields. The analysts at **BigFunds** want to merge these subject areas. They convert similar tables to superset and subset tables. All of the common fields move to a new superset table, **Person**. The fields that are not shared, remain in the subset tables, such as the **Employee**, **Stockholder**, and **Customer** fields.



Merging Tables for Performance

Merging of tables can improve business analysis that requires many complex queries. Queries that involve more than one table, called **join queries**, are slower and require more space. To improve the performance of an enterprise warehouse, you should merge two or more the tables that are used in long running join queries.



Enhancing Performance of Join Queries

You can merge tables in the following ways for improving performance of join queries.

- superset design
- denormalization
- one-to-one relationship merge

Full_name	Address	Date_of_Birth	Account_number
Tom	Virginia	09/29/80	0011558773
Jim	Woodbridge	10/31/80	5500112368

Superset design

Name	Address	Description	Quantity
Amsco	Woodbridge	Motor	34
Afsd	Virginia	Wheel	64

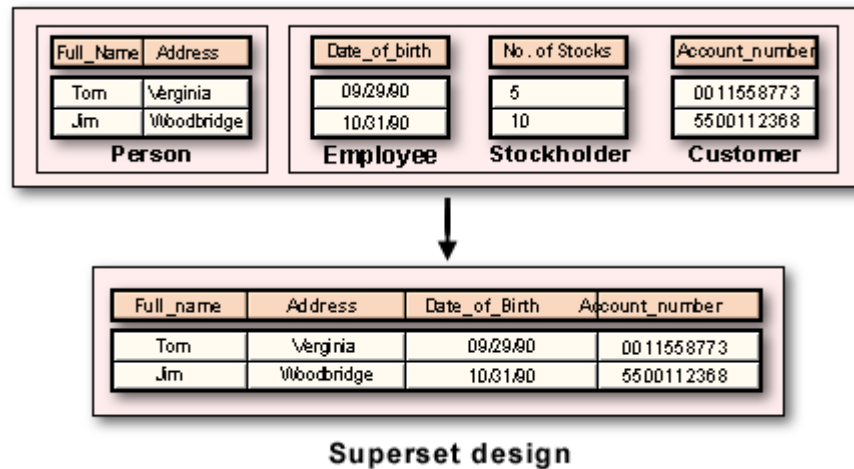
Denormalization

Employee_ID	Name	Salary	Dept.No	Dept.Name
E1	Tom	\$ 5000	1	Developers
E2	Jim	\$ 8000	2	Graphics

One-to-one relationship

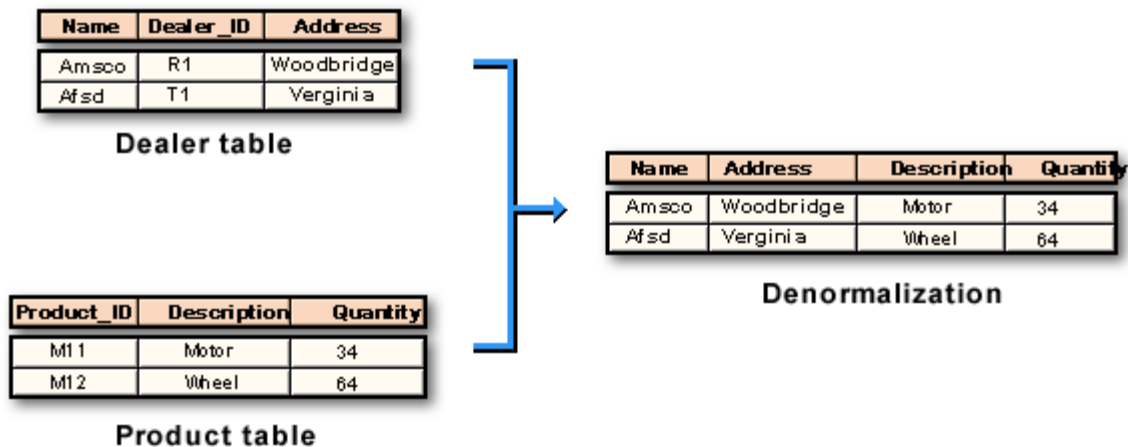
Superset Design

In the superset design, you merge subset and superset tables. Thus, you have fewer tables and hence, fewer join queries. This might generate additional null values. However, in an enterprise warehouse, the benefit of faster joins usually outweighs the disadvantage of more nulls.



Denormalization

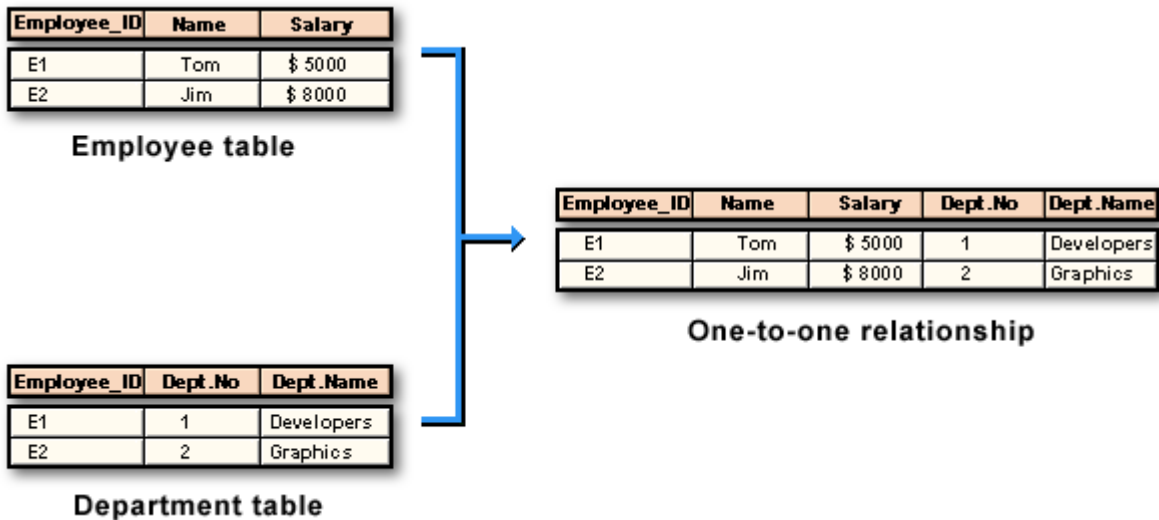
In denormalization, you reverse the normalization process by merging tables connected by [one-to-many relationships](#). By doing so, you might increase redundancy and physical storage requirements marginally. For instance, dealer names can be added to the **Product** table to enhance query.



One-to-one Relationship Merge

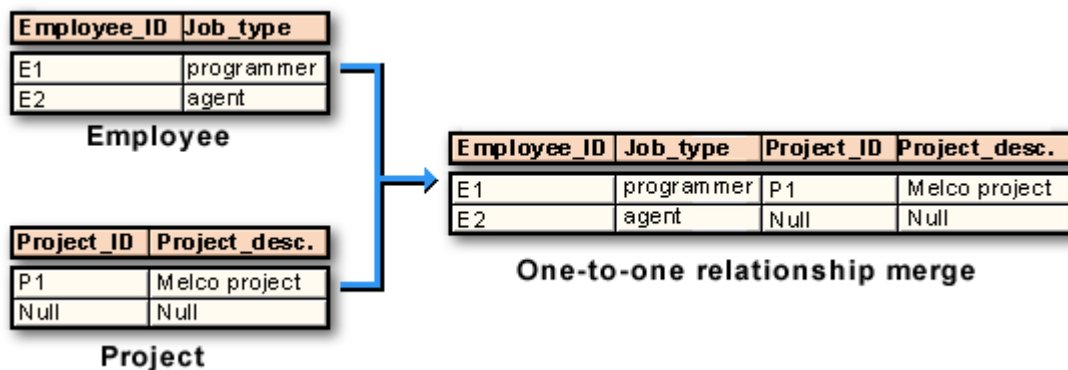
In [one-to-one relationship](#) merge, you merge tables connected by one-to-one relationship. Use one-to-one relationship merge in either of the following two cases.

- The primary keys of the two tables are identical.
- The relationship is required on both sides; for instance, every employee is assigned to at least one project, and vice versa.



One-to-one Relationship Merge

Typically, one employee is related only to a single project at a time. In such a situation, the **Employee** and the **Project** tables are related with a one-to-one relationship. However, the one-to-one relationship between the **Employee** and **Project** tables might be optional. For instance, some employees might not be assigned any projects. When tables connected by an optional relationship are merged, the resulting table has null values. For this reason, it is always best to merge only when the relationship is required, or when the primary keys are identical.



Natural Keys

After merging tables, you might create artificial keys in tables, while mapping subject areas to the warehouse model. However, before considering artificial keys, you need to understand natural keys.

A **natural key** is a primary key, quite familiar to a business analyst. A natural key can appear as a **foreign key** in a different table. Natural keys have the following advantages.

- It is easier to access tables with a natural key.
- Using a natural key avoids the additional field for an artificial key.
- Performance is better due to fewer joins in certain queries.

Employee_ID	Job_type	Dept_code	Project_ID	Description
E1	programmer	D1	P1	Melco project
E2	agent	D2	Null	Null
E3	programmer	D3	P3	Project F A
E4	accountant	D4	Null	Null

Natural key

Creating Artificial Keys

In contrast to natural keys, an **artificial key** is a field introduced by the warehouse designer or administrator to the subject area table. It is usually an integer.

Artificial keys have the following advantages.

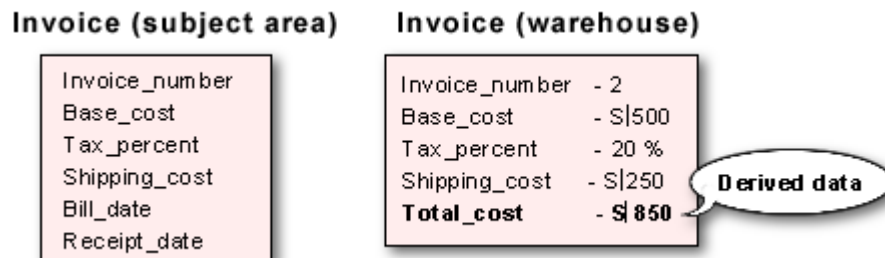
- An artificial key is smaller and more efficient.
- Artificial keys simplify warehouse design, since their values are simple and never change.
- Since artificial keys are constituted of only one field, whereas natural keys might involve several fields, it is usually easier to write join queries.

S.No.	Employee_ID	Job_type	Dept_code	Project_ID	Description
1	E1	programmer	D1	P1	Melco project
2	E2	agent	D2	Null	Null
3	E3	programmer	D3	P3	Project F A
4	E4	accountant	D4	Null	Null

Artificial key

Creating Derived Data

After creating artificial keys, further transformation of subject area models to warehouse data requires creation of derived data. **Derived data** is data, which can be calculated from other fields or tables. Storing derived data in a warehouse improves performance and simplifies query syntax by eliminating complex calculations.



Summarizing Transactions

You might create derived data in the form of summaries. A **summary** is a sum of values in one column across many rows. Sometimes, the term refers to other types of aggregates, such as average, minimum, or maximum computations.

As characteristic to derived data, summaries improve performance and simplify queries. However, summaries can take minutes or hours to compute, since they involve many more records. Therefore, you can limit the number and type of summaries based on the time available to load the warehouse.

Product_Line	Region	Quarter	revenue
Slates	Midwest	1Q94	4,784
Slates	Central	1Q94	5,777
...			
Dockers	Midwest	1Q94	23,342
Average			33,903

Summary

Vector Data

After creating derived data, such as summaries, you need to address the issue of vector data. A **vector** is a fixed number of related values. Vectors might be implemented in a field-wise or record-wise design. For instance, the number of students enrolled for a particular course in a particular year is vector data.

Year	Class	Enrollment
1981	fresh	143
1981	soph	102
1981	Junior	205
1981	Senior	257
1981	Grad	50
1982	Fresh	187
1982	Soph	99
1982	Junior	163
1982	Senior	120
1982	Grad	38

Vector data

Advantages of Record-wise Design

The record-wise implementation of vectors has the following advantages.

- Less storage space is required.
- Fewer tables are involved.
- Generating record-wise reports is simpler for users, although conversion from field-wise design to record-wise report might be slow and complex.

Year	Fresh	Soph	Junior	Senior	Grad
1981	143	102	205	257	50
1982	187	99	163	120	38

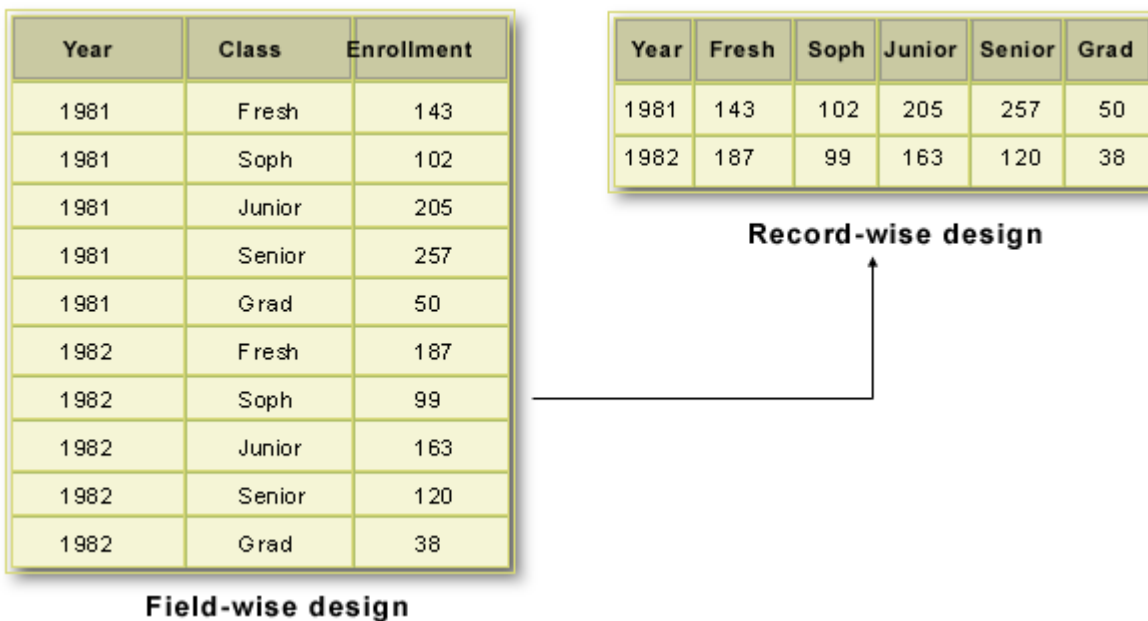
Record-wise design

Advantages of Field-wise Design

The field-wise implementation of vectors has the following advantages.

- Field-wise design does not limit the number of related values.
- There are fewer nulls, and less space is required when the vector is [sparse](#).
- Maintenance is easy if vector changes length.
- SQL functions are usually easier to write and less error prone.

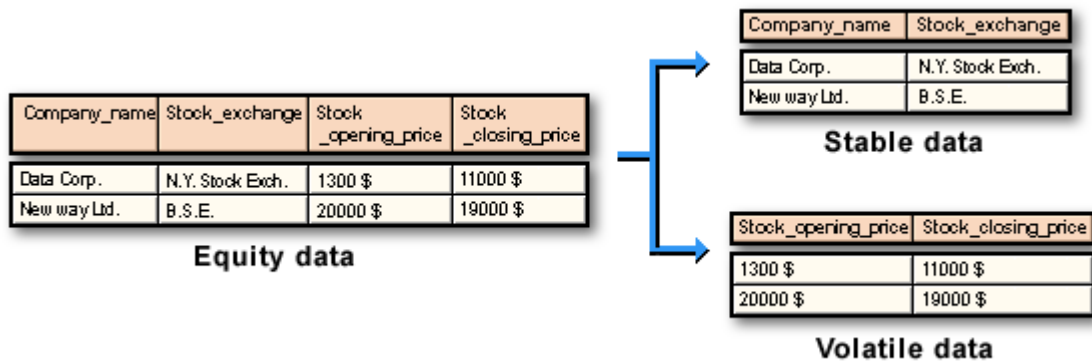
Although these advantages are significant for operational systems, query speed and database size are usually more important in a warehouse. As a rule, you should implement vectors record-wise in a warehouse model.



Volatile Data

Apart from implementing vectors while mapping subject areas to the warehouse model, you address the problem of volatile data. Rapidly changing data is called **volatile data**. You need to separate volatile data from stable data in a warehouse model. For instance, a table of equity data might include the symbol, which rarely changes, and the price, which changes continually. In this case, you might place symbol and price in separate tables.

The advantage of this design is that you need not alter the table containing static data when you load the warehouse. This reduces load time.



Question # 1

While mapping subject areas to the warehouse model, you need to first merge the subject areas.

- ☐ True
- ☐ False

Question # 2

You can improve the performance of an enterprise warehouse by merging the tables that are used in long running join queries.

- ☐ True
- ☐ False

Question # 3

In the superset design, you can merge tables because the primary keys of the merging tables are identical.

- ☐ True
- ☐ False

Question # 4

An artificial key appears as a foreign key in a different table.

- ☐ True
- ☐ False

Question # 5

In field-wise implementation of vectors:

- ☐ less storage space is required
- ☐ fewer tables are involved
- ☐ maintenance is easy if vector changes length

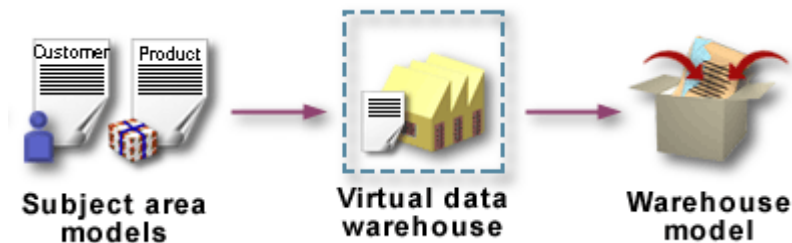
Task # 05.

Modeling the Virtual Data Warehouse

Theory:

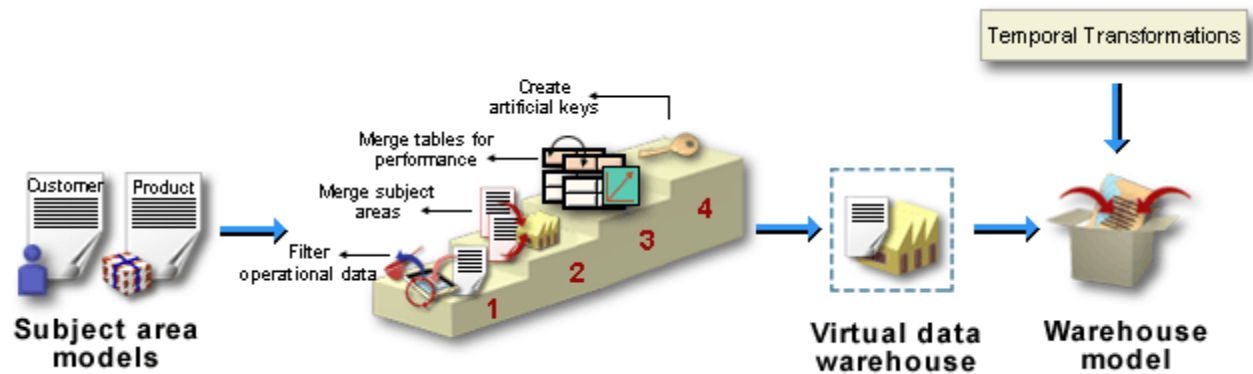
The Virtual Data Warehouse

In the transformation process of mapping subject areas to the warehouse model, the virtual data warehouse is created midway between the subject areas and the warehouse model.



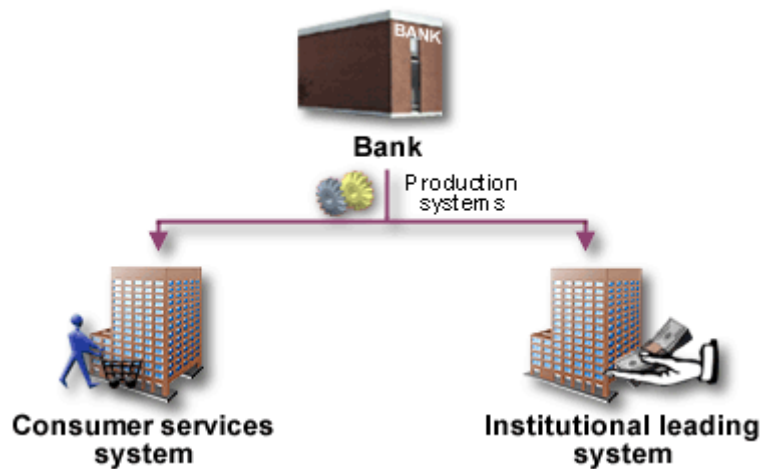
Modeling the Virtual Data Warehouse

The first four steps of the transformation process map subject areas to the virtual data warehouse. These four steps are: filtering operational data, merging subject areas, merging tables for performance, and creating artificial keys. The remaining four convert the virtual data warehouse to the warehouse model. You also require temporal transformations to finish the process of converting the virtual data warehouse to the warehouse model. **Temporal transformations** transform current data to historic.



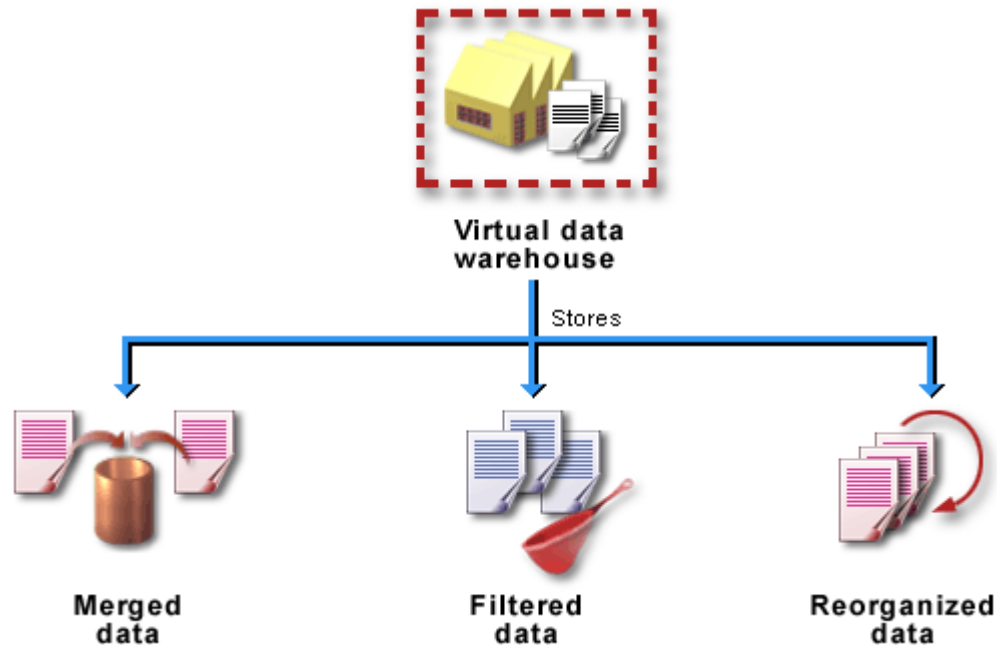
The Virtual Data Warehouse

Consider, **ABC bank**, which has two production systems, the consumer services system and the institutional lending system. The bank needs to design a warehouse for fast and easy querying. The queries mostly relate to customer balances, loan status, and interest statements.



The Virtual Data Warehouse

According to the requirements of **ABC bank**, the bank does not need a full-fledged enterprise warehouse since the queries require operational data and not historic data. The bank needs to design a virtual data warehouse. Data should be loaded into this warehouse at the end of each day's transactions. In addition, the warehouse should store merged, filtered, and reorganized data assembled from the production systems.



Summary

Mapping subject areas to the warehouse model involves eight data transformations. These are: filtering operational data, merging subject areas, merging tables for performance, creating artificial keys, creating derived data, summarizing transactions, converting field-wise design to record-wise design, and designing for volatility. The first four of the eight transformations map subject areas to the virtual data warehouse since it is midway between the subject areas and the warehouse model. In addition to the eight transformations, temporal transformations are also required while mapping subject areas to the warehouse model.

Exercise:

Question#1:

Which of the following steps are required to transform subject areas to a virtual data warehouse?

- ☐ temporal transformation
- ☐ creating artificial keys
- ☐ merging tables for performance improvement
- ☐ creating volatile data

Task # 6.

Identify different kinds of temporal data and construct temporal queries

Theory:

Modeling Temporal Data

Enterprise warehouses and data marts contain historic data that analysts use for studying trends. These warehouses might also contain projected data in the form of future business trends. To create such data, you need to perform temporal transformations that map current source data to historic warehouse data. Temporal data in the warehouse is managed and queried using temporal queries.

At the end of this topic, you will be able to:

- identify different kinds of temporal data
- develop temporal transformations from subject areas to the warehouse model
- construct temporal queries

Temporal Data

Most of the data that you encounter in OLTP systems is **current data**, which describe events as they exist in the present situation. For instance, the **Account_Balance** field of a **Customer** table in a bank's operational database tracks the customer's current account balance.

In contrast, in enterprise warehouses and data marts, you will encounter historic and projected data. Such data that has a time component attached to it is called **temporal data**. The terms, **time-series data** and **time-variant data** also refer to temporal data.

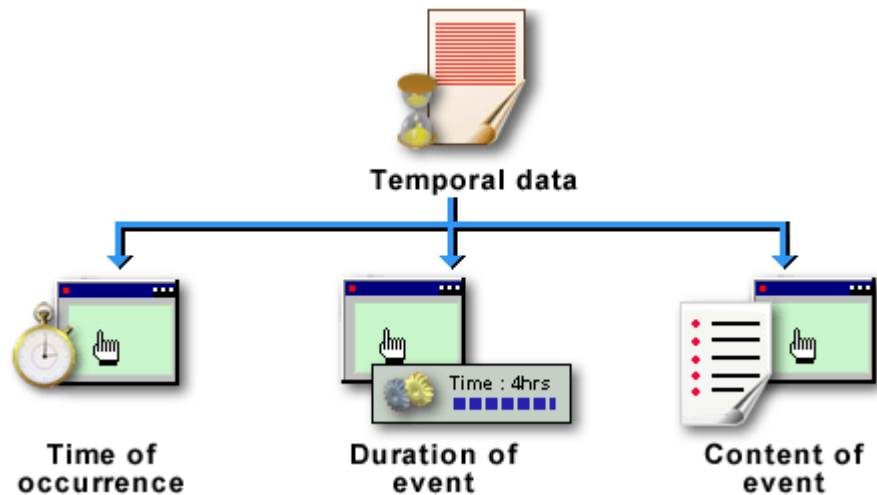
Depending on the variable it describes, temporal data can have different levels of precision. For instance, you measure fiscal time in quarter years, while machine idling time could be measured in minutes.



Classification of Temporal Data

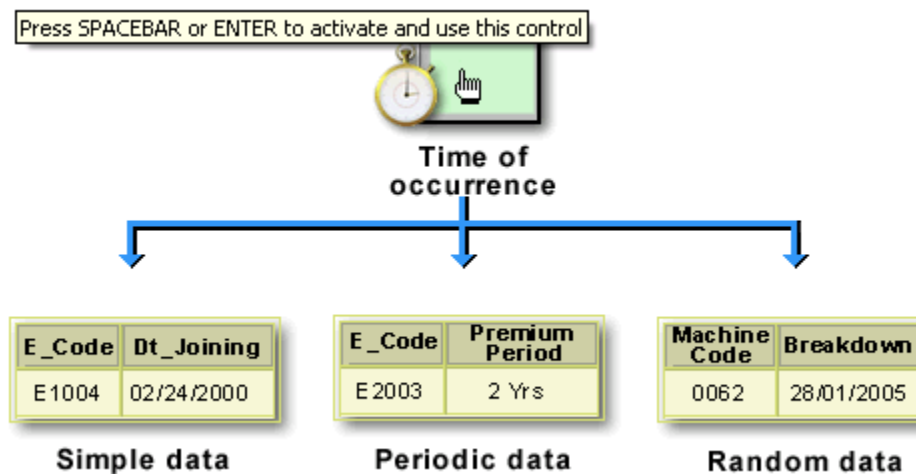
You can classify temporal data in three different ways based on the following three characteristics.

- time of occurrence
- duration of event
- content of event



Classifying Temporal Data

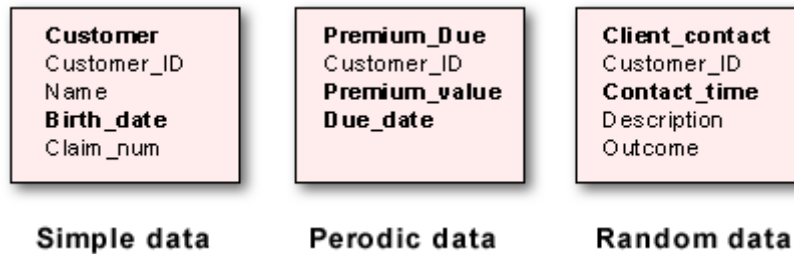
Based on time of occurrence, you can classify temporal data into three types: simple, periodic, and random. Click each type of temporal data in the graphic for a description.



Simple, Periodic, and Random Data

Simple data is independent of other data. For instance, in a **Customer** table, the **Birth_date** field is not dependent on fields of other tables, such as **Premium_value** and **Due_date**.

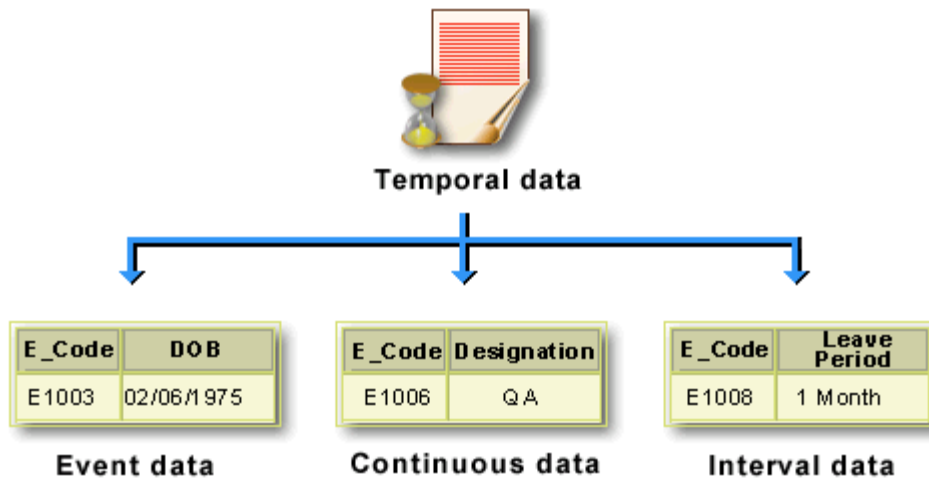
On the other hand, periodic and random data are dependent on other data of a table. For instance, the **Premium_value** field of a **Premium_Due** table is meaningless without the **Due_date** field. Random data, unlike periodic data does not repeat at regular intervals. For instance, the **Contact_time** field of the **Client_contact** table stores the time of customer contact that can occur any number of times during the day.



Event, Continuous, and Interval Data

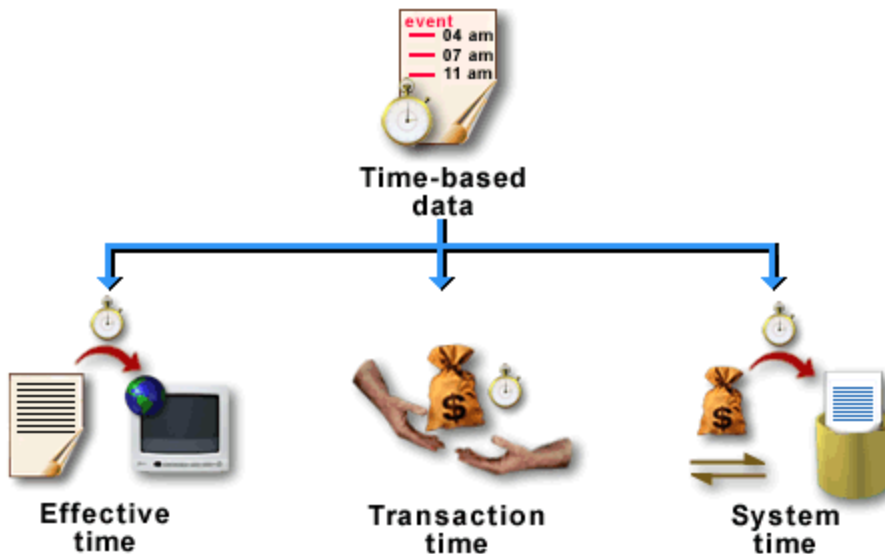
Simple, periodic, and random data are classified based on their time of occurrence. You can also classify temporal data based on duration. Event, continuous, and interval are the three types of data classified based on duration.

Click each type of temporal data in the graphic for a description.



Effective, Transaction, and System Time

In addition to time of occurrence and duration, you can classify time-based data on the basis of content of the event. Click each type of temporal data in the graphic for a description.



Effective, Transaction, and System Time

Consider the database of **LotLoans Inc.**, a company that finances consumer products. According to its business policy, all loans transacted between the first and the twenty-fifth of the month start their term on the thirtieth of the same month. All loans transacted after the twenty-fifth start their term on the seventh of the subsequent month. Transactions are on-line and entered on the database the same day.

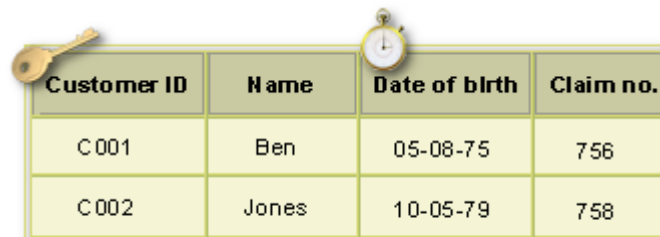
The graphic displays the **Loan_Detail** table of the database. Here, **LOAN_START_DATE** is effective time, **TRANSACTION_DATE** is transaction time, and **DATE_ENTERED** is system time.

CUST_NAME	CUST_ID	TRANSACTION_DATE	LOAN_START_DATE	DATE_ENTERED
Bill Smith	2098	01/02/99	01/30/99	01/02/99
R. Gibson	2399	01/24/99	02/07/99	01/24/99

Designing Simple Data Tables

The classification of temporal data into effective, transaction, and system time does not affect the design of a temporal table. However, the classification of temporal data into simple, periodic, and random data affects the design of temporal tables.

A **simple data table** has a single time point field. To design simple tables, you need to add a date or time field to the table. This field is not part of the primary key, since it is not dependent on other fields.



Customer ID	Name	Date of blrth	Claim no.
C001	Ben	05-08-75	756
C002	Jones	10-05-79	758

Simple data table

Designing Periodic Data Tables

In **periodic data tables**, the time field is part of the primary key, since other fields depend on it. Since the time period is regular, you can replace time with a sequence number, if the programmers and users know the starting time and length of the period. You have the following three options to choose from while designing a periodic table.

- Include a date/time field, representing the beginning of the period, in the primary key.
- Include a sequence number in the primary key.
- Design a record-wise table when the number of periods is fixed.



Insurance period (Yr.)	Premium value
5	5000
10	10000
15	15000
20	20000

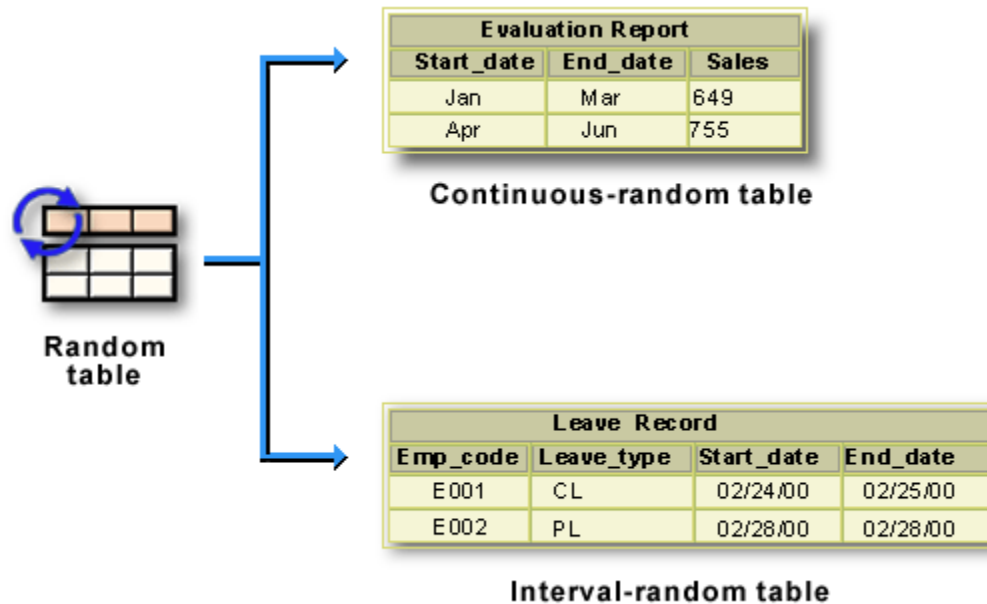
Periodic data table

Designing Random Data Tables

Random tables are similar to periodic tables. The primary key includes time. However, since the period is irregular, you cannot replace time with a sequence number.

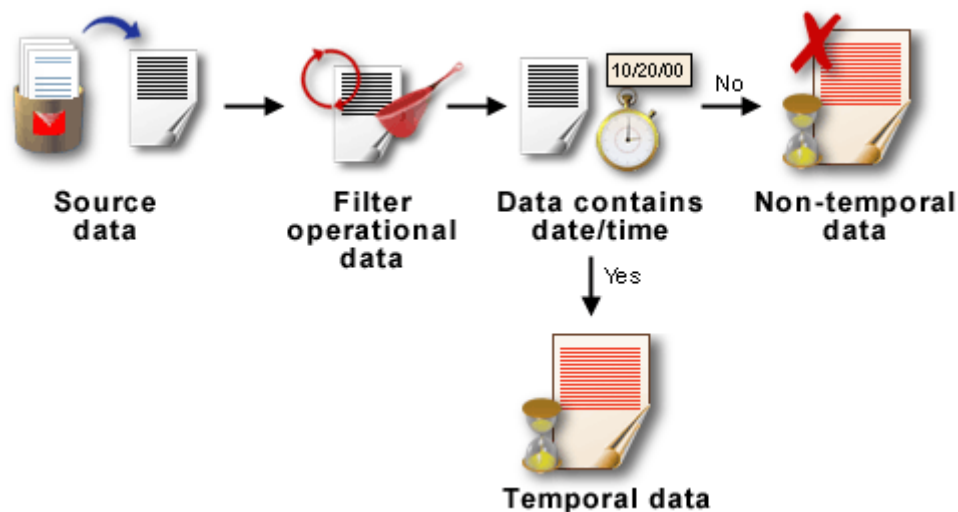
Random tables can also be continuous-random and interval-random. These two tables are complex to design. A **continuous-random table** includes a field that represents the beginning of the time period in the primary key. Optionally, you might add the end of the time period as a non-key field to simplify and accelerate queries.

Interval-random tables also include a field that represents the beginning of the time period in the primary key. However, unlike continuous-random tables, interval-random tables always need to include the end of the time period since the ending time is not necessarily the beginning of the next period. In case of indefinite begin and end time, use default values, such as 0 for begin time and 9999 for end time.



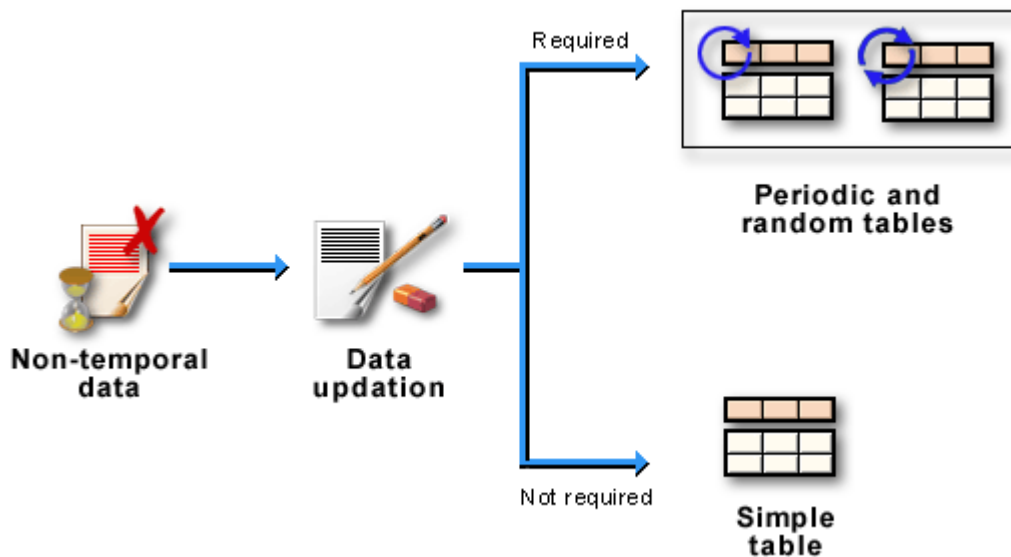
Basic Temporal Transformations

Temporal transformations can be classified into basic and advanced transformations. Basic temporal transformations start after filtering operational data from the source data. At this point, you need to examine whether the filtered data contains any time or date/time fields. No time or date/time fields indicate that the data is non-temporal. On the other hand, if the table contains time and date/time fields, you need to examine if further temporal transformations to periodic, random, continuous, or interval data are required.



Basic Transformations for Non-temporal Data

If the filtered source data contains non-temporal data, examine if the data requires updating from time to time. If the data does not require updating, then the data does not require any further transformation. You might add a time or date field to transform the data into a simple table. If the data requires updating from time to time, you need to examine if the updations are periodic or random. If the data updations are periodic, you need to transform the data into a periodic table. If the data requires random updations, you need to transform the data into a random table. However, sometimes, you might want to transform random tables to periodic tables. For instance, although a customer account is updated randomly, business analysts might prefer a monthly snapshot of balances.

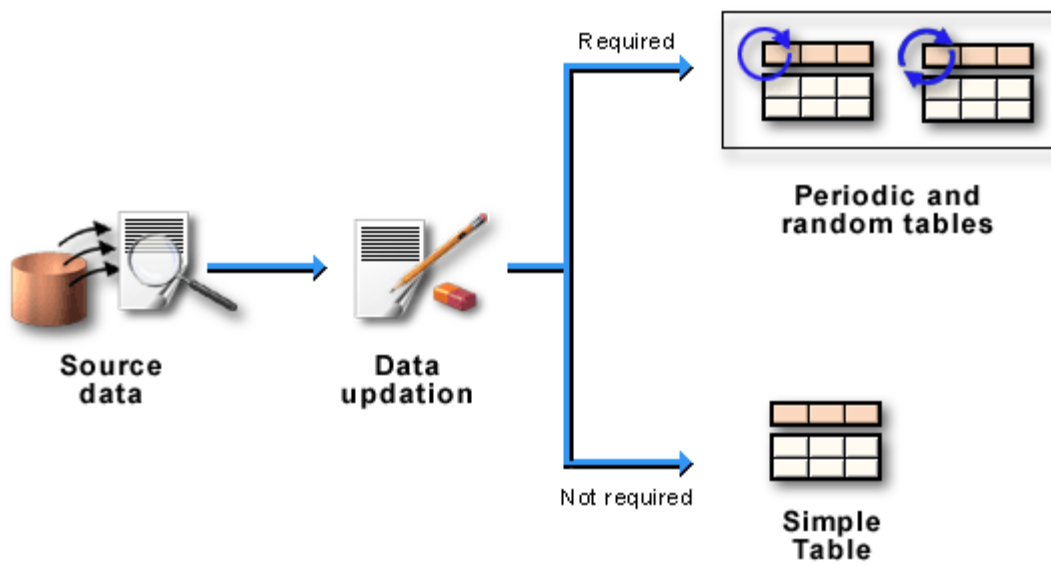


Basic Transformations for Temporal Tables

The sequence of transformations that you need to perform for temporal tables is different from that for non-temporal tables. You can perform the following transformations.

- If the filtered source data contains time and date/time fields, you need to determine whether the fields are simple, periodic, or random.
- If the fields are simple, and do not require updations, further transformation is not required.
- If the time and date/time fields require updations, you can further transform the data to periodic or random depending on the occurrence of updations.
- In case these fields are periodic, then no further transformation is required.

- If the fields are random, then you need to examine if the source data contains event, continuous, or interval data, and design accordingly.

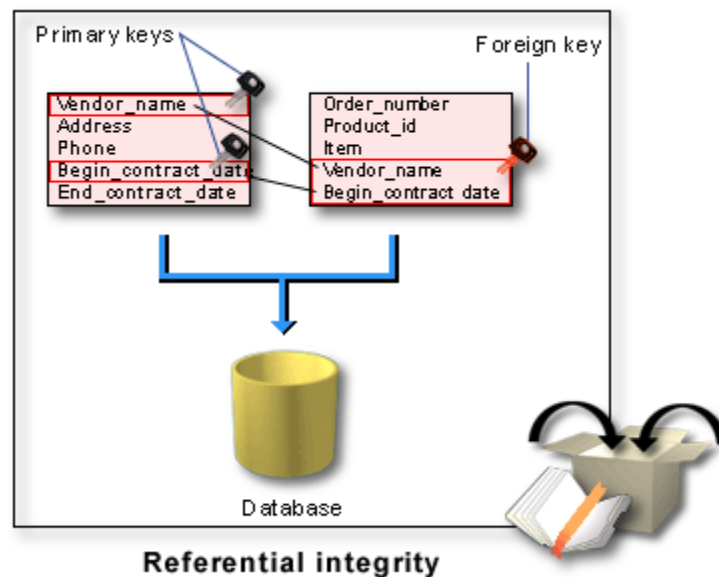


Exercise

Referential Integrity

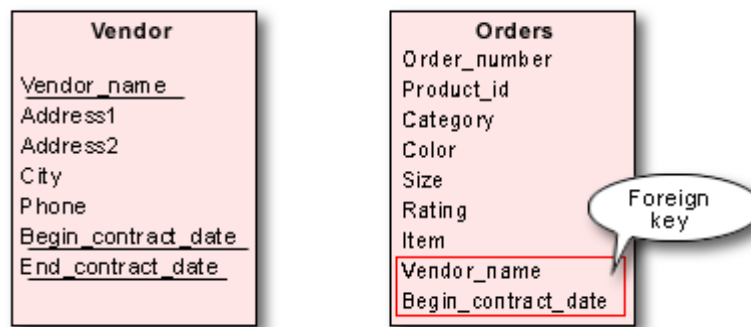
Referential integrity implies that every foreign key field in a table must match a primary key field in another table in the same database. Transformation to temporal data creates special problems with referential integrity, since the primary keys of temporal tables include either date/time, starting time, or sequence number. Foreign keys may no longer match these primary keys, and normal database utilities for referential integrity might not work.

To overcome this problem, you should write a specialized procedure that checks referential integrity for all temporal tables. Run the procedure in batch, when you load data to the warehouse.



Referential Integrity in Temporal data

Consider **IraWare Inc.**, a company that markets ceramic tiles. They purchase products from various vendors. **IraWare** has a half-yearly appraisal system. According to the system, the management renews contracts with the vendors who have supplied them with quality products and terminates contract with those who have not. The system maintains a **Vendor** table that has the **Vendor_name**, **Begin_contract_date**, and **End_contract_date** fields as primary keys. Another table, **Orders**, has **Vendor_name** and **Begin_contract_date** as its foreign keys.



Referential Integrity in Temporal data

As some vendors consistently supply quality products, their contracts get renewed, resulting in several records of fractured data in the **Vendor** table. While transforming temporal data, these records are repaired by merging contiguous records for each vendor. However, after this transformation, the **Begin_contract_date** field of the **Vendor** table might not match the **Begin_contract_date** field in the **Orders** table. Therefore, referential integrity needs to be checked during transformation of the database.

Vendor	Begin_contract_date
Spencer	01/99
Brial enterprises	01/99
Brial enterprises	0799

Vendor table

Fractured data

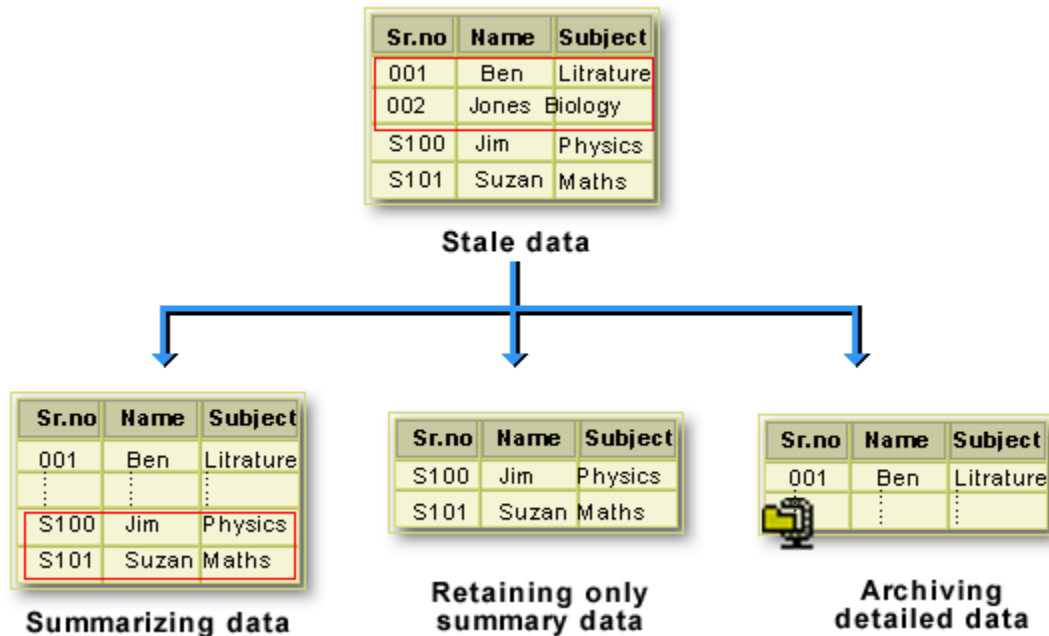
Vendor	Begin_contract_date
Spencer	09/99
Brial enterprises	06/99
Brial enterprises	12/99

Orders table

Stale Data

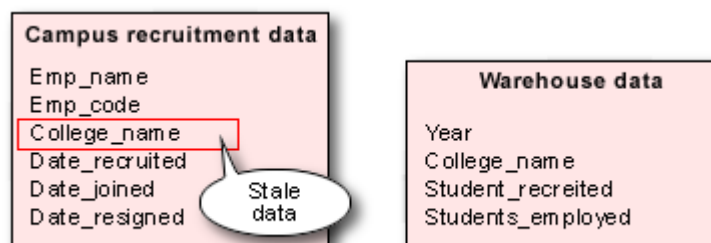
Stale data is detailed data that has aged and is no longer necessary. Stale data makes your warehouse unnecessarily large. Purge stale data by:

- summarizing data
- retaining only summary data in the warehouse
- archiving detailed data



Stale Data

Consider **2001 Technologies Inc.**, a company that visits technical colleges every year to recruit promising students. The human resource department has maintained data of each student recruited from colleges for the last twenty years. Carrying this huge data to the warehouse would make it unnecessarily large. Therefore, they summarize this data and carry only essential data to the warehouse.



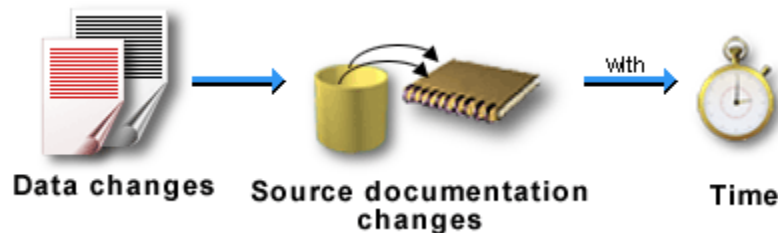
Temporal Metadata

Temporal data is not the only part of your warehouse that changes over time. The warehouse model also changes, because source documentation changes occasionally. Data about changing source documentation is called **temporal metadata**, and is stored in the warehouse directory.

Since source documentation changes at irregular times, temporal metadata is typically random data. Temporal metadata is either continuous data, such as changes in field names, or interval, such as

deleting a table and recreating it later. Either way, treat metadata just like other temporal data. First, determine the type of time and then apply the basic steps for temporal transformations.

If source documentation changes extensively, consider a subset design for the metadata. For instance, if your source database evolves through several versions, place metadata that is common to all versions in one superset table. Place metadata that is specific to each version in a different subset table.



Temporal Metadata

Consider the warehouse data of **XYZ Corporation**. The graphic displays a portion of the table containing the temporal metadata and the corresponding table. Since the data is random-interval, the **begin_date** and **end_date** fields have been added to the **Metadata** table. The **Customer_Id** and **Name** fields of the customer table are static. **Priority** was created in **8/96** and **ship_flag** was deleted from the source data in **5/95**. The null values in the **Customer** table reflect these changes for records that were created before **8/96** and after **5/95**.

Metadata table				
TABLE	COLUMN	BEGIN_DATE	END_DATE	DESCRIPTION
Customer	Customer_ID	0	9999	Artificial key for identifying customers
Customer	Name	0	9999	Official company name
Customer	Priority	8/96	9999	New code for preferred customers
Customer	Ship_flag	0	5/95	Shipping preference** discontinued**

Customer table			
Customer_ID	Name	Priority	Ship_flag
143	Acme Bolts	P	Ground
82	Crumpacker Inc	L	Ground
198	Alpha-beta food products	P	Null
24	Delta-norton	Null	overnight

Temporal Queries

You can extract temporal or historic data using **temporal queries**. The graphic displays a table containing special requirements for queries against interval data. Queries against continuous data have similar requirements, and queries against event data do not have such requirements.

Query on interval data	
Type of query	Requirement
INSERT	If the value for ending time is indefinite, set it to 9999
UPDATE	Replace 9999 in the ending time field with starting time of the new record
EXPIRE	Update the ending time field with the current date or time
PURGE	Archive all records with ending time prior to the purge date
DELETE	Apply only if you cannot use PURGE or EXPIRE
JOIN	Verify that time periods overlap for joined records

Constructing Temporal Joins

After identifying the information to be extracted, you need to identify the source tables that contain the information. In the case of **Trident Inc.**, the tables might be **Branch** and **Employee**. Next, you need to identify the fields for joining the two tables, such as the **Branch** field.

Employee					
Emp ID	Begin date	End date	Name	Branch	Salary (\$)
E1	1/93	12/94	Terry Brown	Hamburg	4500
E2	1/94	12/95	Terry Brown	Berlin	5200
E3	4/92	09/93	Cathy Vig	Bayern	4700

Temporal joins

Branch			
Branch	Begin date	End date	Building no
Hamburg	0	9/94	800
Berlin	10/94	9999	34
Bayern	3/90	6/93	100

Constructing Temporal Joins

While joining the tables, consider all the conditions before creating the join query. For instance, to locate the building numbers of the departments during the time **Terry Brown** worked for them, you need to identify the overlap between the begin dates and end dates of the employee and departments. The following example displays a query to extract the required data.

```
SELECT Full_Name, Dept_Name, Building_Number FROM Employee, Department WHERE  
Employee.Dept_Name=Department.Dept_Name AND Employee.Begin_Date <=  
Department.End_Date AND Employee.End_Date >= Department.Begin_Date
```

Question # 1

If the non-temporal data requires updating from time to time, you need to examine if the updations are periodic or random

- ☐ true
- ☐ false

Question # 2

Non-temporal data is created when two or more records of a table are contiguous and identical, except for the time fields.

- ☐ True
- ☐ False

Question # 3

Transformation to temporal data creates special problems with referential integrity, since the primary keys of temporal tables include either date/time, starting time, or sequence number.

- ☐ True
- ☐ False

Question # 4

Temporal metadata is typically:

- ☐ Periodic data
- ☐ Simple data
- ☐ Random data

Question # 5

Consider Sinew Software Inc., a company that executes software projects for its clients. Its warehouse maintains information, such as ending and beginning dates for each project that an employee has worked on. Suppose an employee, Jones, is abruptly transferred to Project Y from Project X due to a crisis in Project Y. In such a situation, which one of the following changes needs to be considered while running the query?

- ☐ Replace the beginning date of the old record with the current date.
- ☐ Replace ending date of the old record with 9999.
- ☐ Replace 9999 in the ending date field of the old record with beginning date of the new record

Question # 6

Data scrubbing involves:

- ☐ creating a common primary key
- ☐ validating the source data
- ☐ unpacking overloaded fields

Question # 7

Which one of the following tools provided by Innovative systems is used to group corporate data based on name, address, tax identification number, or user-defined fields?

- ☐ Innovative-Analyzer
- ☐ Innovative-Dictionary
- ☐ Innovative-Match
- ☐ Innovative Corp-Match

Question # 8

Which one of the following steps is used to model a virtual data warehouse?

- ☐ merging subject areas
- ☐ converting field-wise design to record-wise design
- ☐ designing for volatility

Question # 9

Most of the data in the virtual data warehouse is:

- ☐ historic data
- ☐ current data
- ☐ projected data

Question # 10

Consider Retail Credit Inc., a financial services firm. One of its database systems maintains a table called Loans, in which the Status_date field is updated as the status of an application for loan changes. For instance, when an application for a loan is received, the date for the application is stored in the Status_date field. When the loan is approved, the date of approval is stored in the Status_date field. The Status_date field is updated again at the time of disbursal. Thus, the Status_date field contains:

- ☐ periodic data
- ☐ random data
- ☐ simple data

Question # 11

Fractured data is:

- ☐ created when two or more records of a table are contiguous and identical, except for the time fields
- ☐ data about changing source documentation
- ☐ detailed data that has aged and is no longer necessary

Question # 12

Data modified between extraction and execution, within the source data, is called:

- ☐ dirty data
- ☐ delta data
- ☐ metadata
- ☐ temporal data

Question # 13

Which one of the following Trillium components breaks complex data into its elements, and standardizes and verifies data?

- ☐ Parser
- ☐ Matcher
- ☐ Geocoder
- ☐ Data browser

Question # 14

Current data is turned to historic data using:

- ☐ natural keys
- ☐ artificial keys
- ☐ temporal transformations
- ☐ non-temporal transformations

Question # 15

The time field is part of the primary key in:

- ☐ periodic data tables
- ☐ continuous-random tables
- ☐ interval-random tables
- ☐ simple data tables

Task # 7.

To design a dimensional model

Theory:

Basics of Dimensional Model

Enterprise warehouses can be divided into small subsets of data for specific business processes, thus, helping you create a number of data marts. However, most of the data marts are designed based on the dimensional model. The components of a dimensional model and their attributes enable business analysts to divide and shape the data in numerous ways. A dimensional model also offers a powerful method for analyzing data.

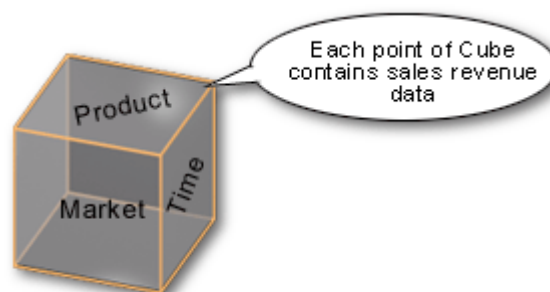
At the end of this topic, you will be able to:

- define the dimensional model and its components
- define the flow of products through a value chain
- convert the warehouse model to the dimensional model

Data as a Cube

You can view business data as a cube, where each edge of the cube represents a measurement of performance. For instance, business data in the cube might be sales revenue. The three dimensions of the cube represent product, market, and time respectively. Each point in this cube of data contains sales revenue data for a particular combination of product, market, and time. Depending on the complexity of the business, the cube can have more edges or dimensions.

Arranging data in this way makes it easy to identify specific values and quickly retrieve alternate views of different combinations of information. Thus, you can **slice and dice** the data cube along any of its edges or dimensions.



Business data as cube

Using the Cube of Data

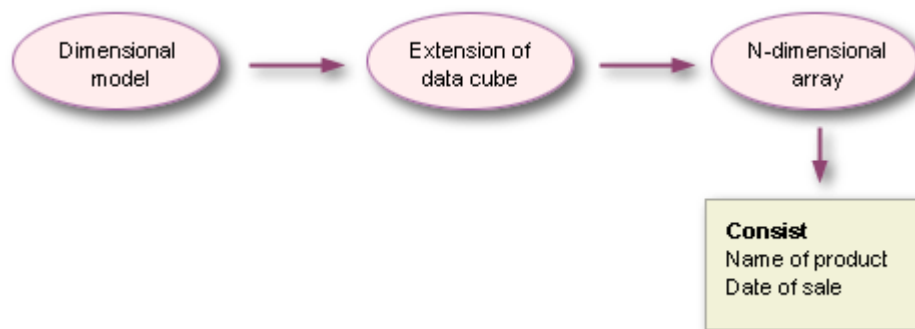
Consider the regional managers, financial manager and product manager, of **Ramjet Corp.**, who are presenting performance reviews at the annual meeting. Each manager can access data on product categories, market regions, and time durations.

Each regional manager analyzes the performance of all products in a particular region over a time. The financial manager analyzes the performance of all products across all market regions at a fixed point of time. Each product manager analyzes the performance of products across all market regions and time. Thus, each manager slices and dices the cube of data in different ways.

Corp. Managers	Access Data
Regional Managers	Products in region over time
Financial Manager	Products and Market regions at fixed point of time
Product Managers	Product, Market regions, and Time

Dimensional Model

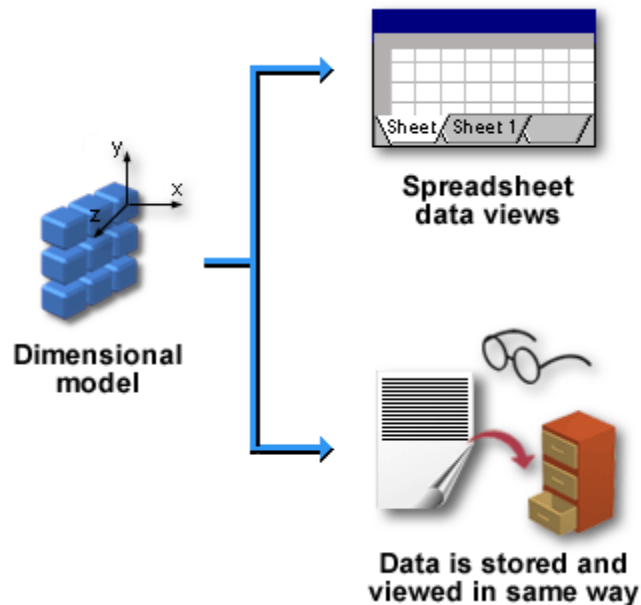
A **dimensional model** is an extension of a data cube, which represents data as a hypercube or an N-dimensional array. The values in the array are numeric facts about business, such as sales revenue, performance ratios, and quantity. The dimensions of the array consist of descriptive information, such as the name of a product or the date on which a sale is made.



Benefits of Dimensional Models

Apart from the dimensional model, the other three data models are source documentation, subject area model, and warehouse model. There are two major advantages of the dimensional model over these three data models.

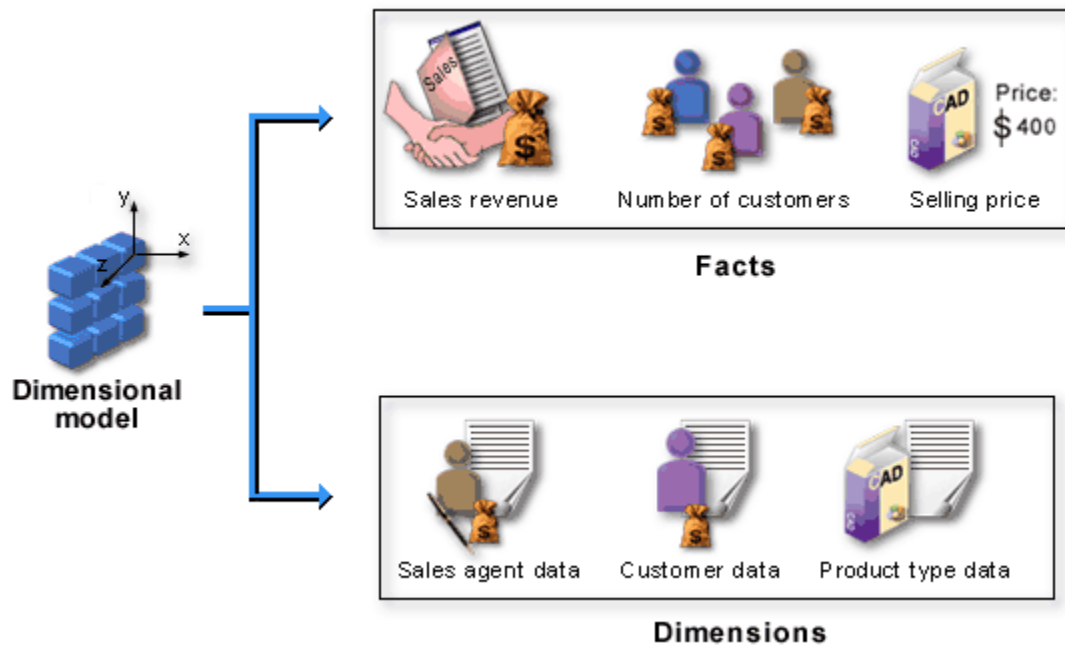
- Since spreadsheet-like data views are the natural output of dimensional models, you can easily analyze, present, and navigate through data.
- Since data is stored in the same way as it is viewed, you do not require additional overhead to translate user queries into requests for data.



Components of a Dimensional Model

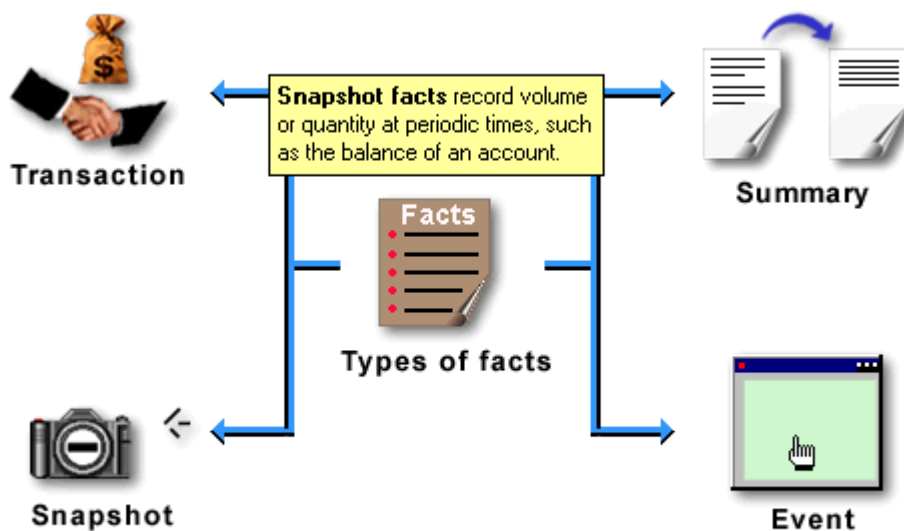
There are two fundamental components in any dimensional model. These are:

- facts - **Facts** are numerical measurements of a business. For instance, data, such as sales revenue, number of customers, and selling price, are facts.
- dimensions - **Dimensions** are textual data, which describe the facts. For instance, sales agent data, customer data, and product type data, are dimensions. Since business analysts need to assess trends in business performance, one of the dimensions is always an indicator of time, such as date, time, or date/time.



Types of Facts

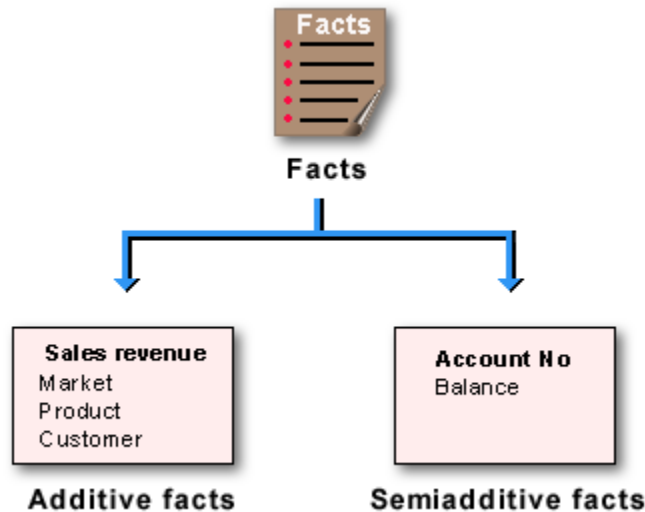
There are four types of facts in a dimensional model. Click each type in the graphic for a description.



Additivity of Facts

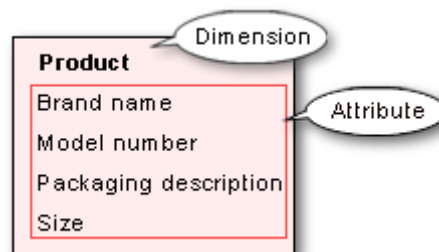
Facts are always numeric data that can be summed up to form high-level summaries. Some facts such as sales revenue, can be summed along every dimension. Such facts are **additive**. In contrast, some facts, such as balance in a bank account, can be added only along certain dimensions. These facts are called **semiadditive**.

For instance, you can add sales revenue along market region, time, product, and customer. On the other hand, you can add balances for different customers or accounts simultaneously. However, you would not want to add data for the same account at different times.



Attributes in a Dimension

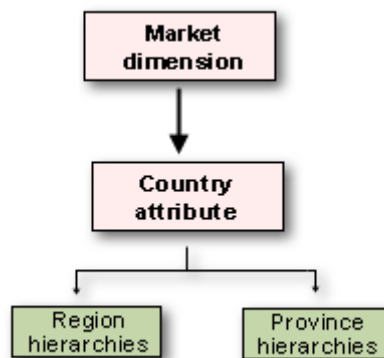
A fact is described by a dimension. In turn, you can split down each dimension into a set of **attributes** that describe different aspects of the dimension. For instance, the product dimension would typically have attributes, such as brand name, model number, packaging description, and size. Usually, attributes are textual, and are displayed as headings in reports.



Attributes and Hierarchies of Dimensions

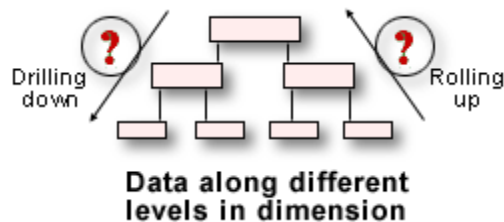
Dimensions have **hierarchies** that reflect natural summary levels. For instance, you can summarize the date dimension as week, month, quarter, year, or all dates.

Some dimensions have several independent hierarchies. For instance, the **Market** dimension can have two independent hierarchies, **Region** and **Province**, under the **Country** attribute.



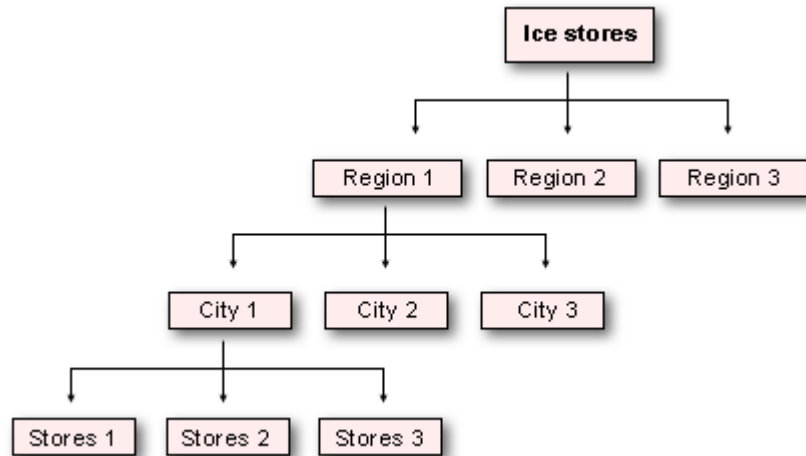
Moving Up and Down in a Hierarchy

Once the hierarchies in a dimension are defined, you can analyze and perform queries on data along different levels or attributes in the dimension. When you move up the levels, it is referred to as **rolling up**. Moving down the levels is called **drilling down**.



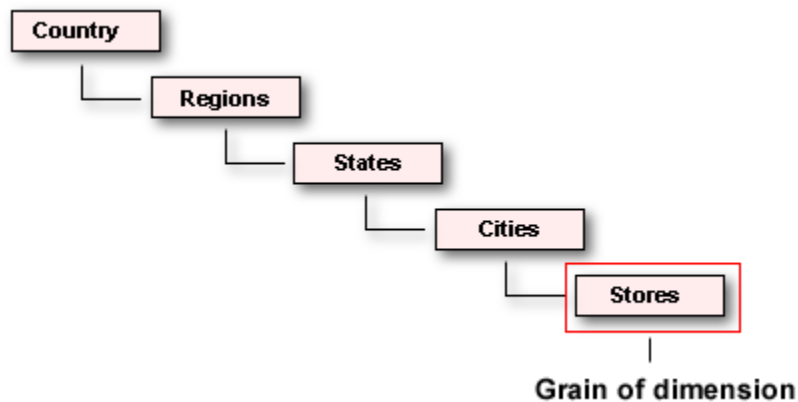
Rolling Up and Drilling Down

Rolling up and drilling down in a dimensional hierarchy can be demonstrated with an example. **IceStores** is a processor and retailer of frozen food. Their retail stores are spread across three regions. Not all retail stores are generating profits and the management wants to close down all the retail stores except the best three in each city, with a maximum of nine per region. The management also wants to determine the amount of revenue these stores generate for their respective cities and regions.



Grain of a Dimension

It is important to determine the lowest level in a dimension. Ideally, you would set this level as low as possible so that you could drill down to the minute details. However, the lower you go, the greater are the response time and storage space required by the data mart. Therefore, you must decide on an optimum level that satisfies both requirements. This lowest level is called the **grain of the dimension**. The combination of the grains along all dimensions is the **grain of the dimensional model**.

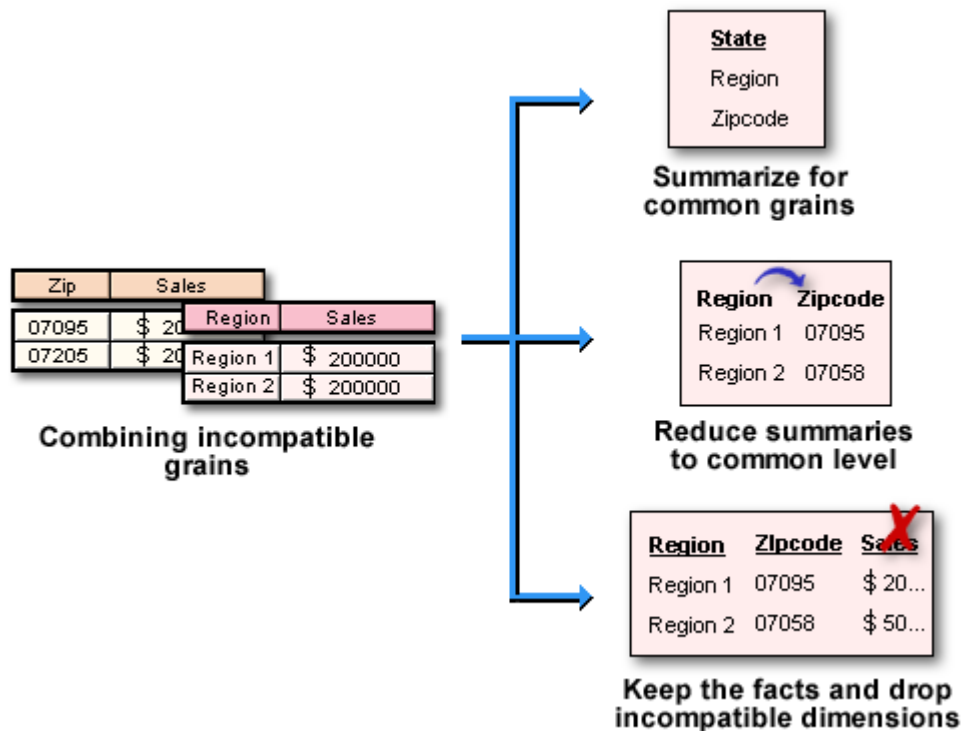


Incompatible Grains

Sometimes, summary data from different sources have incompatible grains. For instance, one source might provide sales summarized by region, and another by zip code. You can combine such data using the following three methods.

- Summarize further, until all sources have a common grain. For instance, you can summarize region and zip code by state. This is the simplest approach. However, you might lose some detailed data.

- Reduce the summaries to a common level. For instance, you can reduce region to zip code. This approach preserves the detail of the summary, but the reduction algorithm is usually arbitrary and inaccurate.
- Keep the facts and drop the incompatible dimension. For instance, you can eliminate the geographic dimension of the sales summary. This is usually not the best choice because you eliminate important dimensional information.



Mapping Warehouse Model to Dimensional Model

The design of most data marts is based on the dimensional model. The first step is to map source documentation to subject area models. The second step is to map subject areas to warehouse model. Mapping the warehouse model to the dimensional model is the third and final step in the model synchronization process.



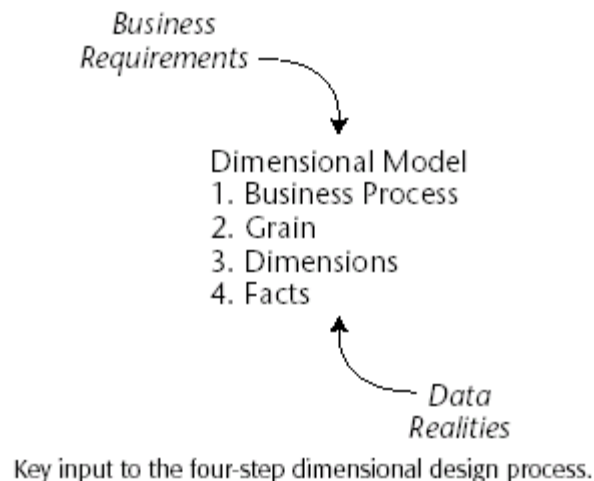
Task # 8.

Retail Case Study

Theory:

Retail Case Study

Let's start with a brief description of the retail business that we'll use in this case study to make dimension and fact tables more understandable. We begin with this industry because it is one to which we can all relate. Imagine that we work in the headquarters of a large grocery chain. Our business has 100 grocery stores spread over a five-state area. Each of the stores has a full complement of departments, including grocery, frozen foods, dairy, meat, produce, bakery, floral, and health/beauty aids. Each store has roughly 60,000 individual products on its shelves. The individual products are called *stock keeping units* (SKUs). About 55,000 of the SKUs come from outside manufacturers and have bar codes imprinted on the product package. These bar codes are called *universal product codes* (UPCs). UPCs are at the same grain as individual SKUs. Each different package variation of a product has a separate UPC and hence is a separate SKU.



The remaining 5,000 SKUs come from departments such as meat, produce, bakery, or floral. While these products don't have nationally recognized UPCs, the grocery chain assigns SKU numbers to them. Since our grocery chain is highly automated, we stick scanner labels on many of the items in these other departments. Although the bar codes are not UPCs, they are certainly SKU numbers. Data is collected at several interesting places in a grocery store. Some of the most useful data is collected at the cash registers as customers purchase products. Our modern grocery store scans the bar codes directly into the point-of-sale (POS) system. The POS system is at the front door of the grocery store where consumer takeaway is measured. The back door, where vendors make deliveries, is another interesting data-collection point. At the grocery store, management is concerned with the logistics of ordering, stocking, and selling products while maximizing profit. The profit ultimately comes from charging as much as possible for each product, lowering costs for product acquisition and overhead, and at the same time attracting as many customers as possible in a highly competitive pricing environment. Some of the most significant management decisions have to do with pricing and promotions. Both store management and headquarters marketing spend a great deal of time tinkering with pricing and promotions. Promotions in a grocery store include temporary price reductions, ads in newspapers and newspaper inserts, displays in

the grocery store (including end-aisle displays), and coupons. The most direct and effective way to create a surge in the volume of product sold is to lower the price dramatically. A 50-cent reduction in the price of paper towels, especially when coupled with an ad and display, can cause the sale of the paper towels to jump by a factor of 10. Unfortunately, such a big price reduction usually is not sustainable because the towels probably are being sold at a loss. As a result of these issues, the visibility of all forms of promotion is an important part of analyzing the operations of a grocery store.

Exercise:

To Design Dimensional model for the retail case study.

Now that we have described our business case study, we'll begin to design the dimensional model.

Step 1. Select the Business Process

The first step in the design is to decide what business process(es) to model by combining an understanding of the business requirements with an understanding of the available data. In our retail case study, management wants to better understand customer purchases as captured by the POS system. Thus the business process we're going to model is POS retail sales. This data will allow us to analyze what products are selling in which stores on what days under what promotional conditions.

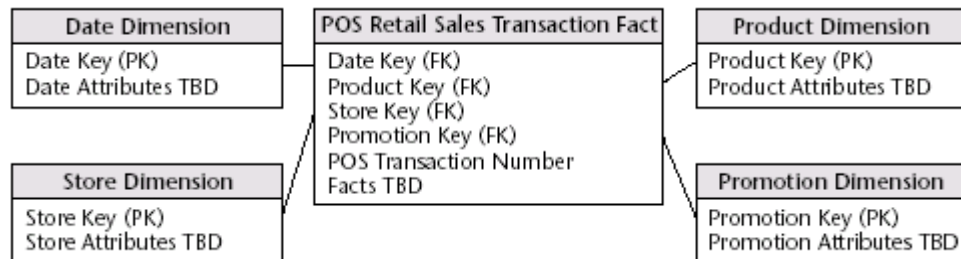
Step 2. Declare the Grain

Once the business process has been identified, the data warehouse team faces a serious decision about the granularity. What level of data detail should be made available in the dimensional model? This brings us to an important design tip.

Tackling data at its lowest, most atomic grain makes sense on multiple fronts. Atomic data is highly dimensional. The more detailed and atomic the fact measurement, the more things we know for sure. All those things we know for sure translate into dimensions. In this regard, atomic data is a perfect match for the dimensional approach. Atomic data provides maximum analytic flexibility because it can be constrained and rolled up in every way possible. Detailed data in a dimensional model is poised and ready for the ad hoc attack by business users. Of course, you can always declare higher-level grains for a business process that represent an aggregation of the most atomic data. However, as soon as we select a higher-level grain, we're limiting ourselves to fewer and/or potentially less detailed dimensions. The less granular model is immediately vulnerable to unexpected user requests to drill down into the details. Users inevitably run into an analytic wall when not given access to the atomic data. In our case study, the most granular data is an individual line item on a POS transaction. To ensure maximum dimensionality and flexibility, we will proceed with this grain. It is worth noting that this granularity declaration represents a change from the first edition of this text. Previously, we focused on POS data, but rather than representing transaction line item detail in the dimensional model, we elected to provide sales data rolled up by product and promotion in a store on a day. At the time, these daily product totals represented the state of the art for syndicated retail sales databases. It was unreasonable to expect then-current hardware and software to deal effectively with the volumes of data associated with individual POS transaction line items. Providing access to the POS transaction information gives us with a very detailed look at store sales. While users probably are not interested in analyzing single items associated with a specific POS transaction, we can't predict all the ways that they'll want to cull through that data. For example, they may want to understand the difference in sales on Monday versus Sunday. Or they may want to assess whether it's worthwhile to stock so many individual sizes of certain brands, such as cereal. Or they may want to understand how many shoppers took advantage of the 50-cents-off promotion on shampoo. Or they may want to determine the impact in terms of decreased sales when a competitive diet soda product was promoted heavily. While none of these queries calls for data from one specific transaction, they are broad questions that require detailed data sliced in very precise ways. None of them could have been answered if we elected only to provide access to summarized data.

Step 3. Choose the Dimensions

Once the grain of the fact table has been chosen, the date, product, and store dimensions fall out immediately. We assume that the calendar date is the date value delivered to us by the POS system. Later, we will see what to do if we also get a time of day along with the date. Within the framework of the primary dimensions, we can ask whether other dimensions can be attributed to the data, such as the promotion under which the product is sold. We express this as another design principle:



In our case study we've decided on the following descriptive dimensions: date, product, store, and promotion. In addition, we'll include the POS transaction ticket number as a special dimension.

Step 4. Identify the Facts

The fourth and final step in the design is to make a careful determination of which facts will appear in the fact table. Again, the grain declaration helps anchor our thinking. Simply put, the facts must be true to the grain: the individual line item on the POS transaction in this case. When considering potential facts, you again may discover that adjustments need to be made to either our earlier grain assumptions or our choice of dimensions.

The facts collected by the POS system include the sales quantity (e.g., the number of cans of chicken noodle soup), per unit sales price, and the sales dollar amount. The sales dollar amount equals the sales quantity multiplied by the unit price. More sophisticated POS systems also provide a standard dollar cost for the product as delivered to the store by the vendor. Presuming that this cost fact is readily available and doesn't require a heroic activity-based costing initiative, we'll include it in the fact table. Three of the facts, sales quantity, sales dollar amount, and cost dollar amount, are beautifully additive across all the dimensions. We can slice and dice the fact table with impunity, and every sum of these three facts is valid and correct.



We can compute the gross profit by subtracting the cost dollar amount from the sales dollar amount, or revenue. Although computed, this gross profit is also perfectly additive across all the dimensions—we can calculate the gross profit of any combination of products sold in any set of stores on any set of days. Dimensional modelers sometimes question whether a calculated fact should be stored physically in the database. We generally recommend that it be stored physically. In our case study, the gross profit

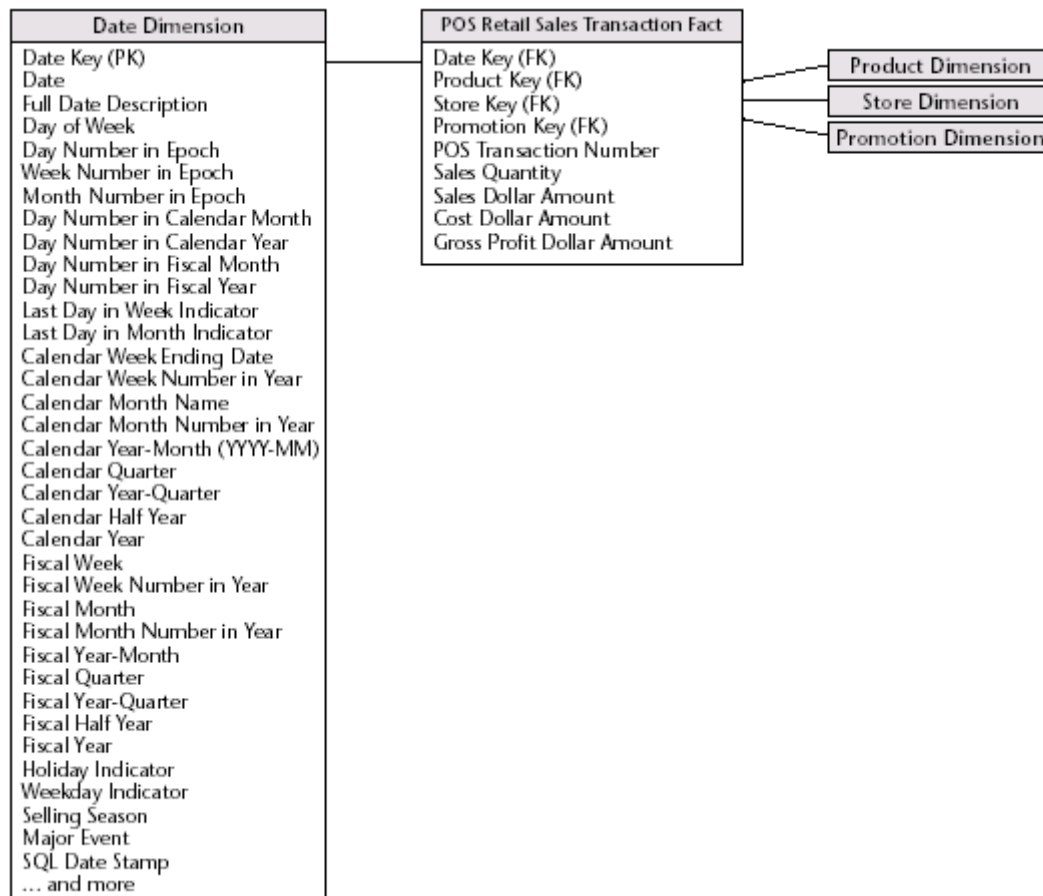
calculation is straightforward, but storing it eliminates the possibility of user error. The cost of a user incorrectly representing gross profit overwhelms the minor incremental storage cost. Storing it also ensures that all users and their reporting applications refer to gross profit consistently. Since gross profit can be calculated from adjacent data within a fact table row, some would argue that we should perform the calculation in a view that is indistinguishable from the table. This is a reasonable approach if *all* users access the data via this view and *no* users with ad hoc query tools can sneak around the view to get at the physical table. Views are a reasonable way to minimize user error while saving on storage, but the DBA must allow no exceptions to data access through the view. Likewise, some organizations want to perform the calculation in the query tool. Again, this works if *all* users access the data using a common tool (which is seldom the case in our experience). The gross margin can be calculated by dividing the gross profit by the dollar revenue. Gross margin is a nonadditive fact because it can't be summarized along any dimension. We can calculate the gross margin of any set of products, stores, or days by remembering to add the revenues and costs before dividing. This can be stated as a design principle: Unit price is also a nonadditive fact. Attempting to sum up unit price across any of the dimensions results in a meaningless, nonsensical number. In order to analyze the average selling price for a product in a series of stores or across a period of time, we must add up the sales dollars and sales quantities before dividing the total dollars by the total quantity sold. Every report writer or query tool in the data warehouse marketplace should automatically perform this function correctly, but unfortunately, some still don't handle it very gracefully. At this early stage of the design, it is often helpful to estimate the number of rows in our largest table, the fact table. In our case study, it simply may be a matter of talking with a source system guru to understand how many POS transaction line items are generated on a periodic basis. Retail traffic fluctuates significantly from day to day, so we'll want to understand the transaction activity over a reasonable period of time. Alternatively, we could estimate the number of rows added to the fact table annually by dividing the chain's annual gross revenue by the average item selling price. Assuming that gross revenues are \$4 billion per year and that the average price of an item on a customer ticket is \$2.00, we calculate that there are approximately 2 billion transaction line items per year. This is a typical engineer's estimate that gets us surprisingly close to sizing a design directly from our armchairs. As designers, we always should be triangulating to determine whether our calculations are reasonable.

Dimension Table Attributes

Now that we've walked through the four-step process, let's return to the dimension tables and focus on filling them with robust attributes.

Date Dimension

We will start with the date dimension. The date dimension is the one dimension nearly guaranteed to be in every data mart because virtually every data mart is a time series. In fact, date is usually the first dimension in the underlying sort order of the database so that the successive loading of time intervals of data is placed into virgin territory on the disk. Each column in the date dimension table is defined by the particular day that the row represents. The day-of-week column contains the name of the day, such as Monday. This column would be used to create reports comparing the business on Mondays with Sunday business. The day number in calendar month column starts with 1 at the beginning of each month and runs to 28, 29, 30, or 31, depending on the month. This column is useful for comparing the same day each month. Similarly, we could have a month number in year (1, ... , 12). The day number in epoch is effectively a Julian day number (that is, a consecutive day number starting at the beginning of some epoch). We also could include



absolute week and month number columns. All these integers support simple date arithmetic between days across year and month boundaries. For reporting, we would want a month name with values such as January. In addition, a yearmonth (YYYY-MM) column is useful as a report column header. We likely also will want a quarter number (Q1, ... , Q4), as well as a year quarter, such as 2001-Q4. We would have similar columns for the fiscal periods if they differ from calendar periods. The holiday indicator takes on the values of Holiday or Nonholiday. Remember that the dimension table attributes serve as report labels. Simply populating the holiday indicator with a Y or an N would be far less useful. Imagine a report where we're comparing holiday sales for a given product versus nonholiday sales. Obviously, it would be helpful if the columns had meaningful values such as Holiday/Nonholiday versus a cryptic Y/N. Rather than decoding cryptic flags into understandable labels in a reporting application, we prefer that the decode be stored in the database so that a consistent value is available to all users regardless of their reporting environment.

A similar argument holds true for the weekday indicator, which would have a value of Weekday or Weekend. Saturdays and Sundays obviously would be assigned the Weekend value. Of course, multiple date table attributes can be jointly constrained, so we can easily compare weekday holidays with weekend holidays, for example. The selling season column is set to the name of the retailing season, if any. Examples in the United States could include Christmas, Thanksgiving, Easter, Valentine's Day, Fourth of July, or None. The major event column is similar to the season column and can be used to mark special outside events such as Super Bowl Sunday or Labor Strike. Regular promotional events usually are not handled in the date table but rather are described more completely by means of the promotion dimension, especially since promotional events are not defined solely by date but usually are defined by a combination of date, product, and store. Some designers pause at this point to ask why an explicit date dimension table is needed. They reason that if the date key in the fact table is a date-type field, then any SQL query can directly constrain on the fact table date key and use natural SQL date semantics to filter

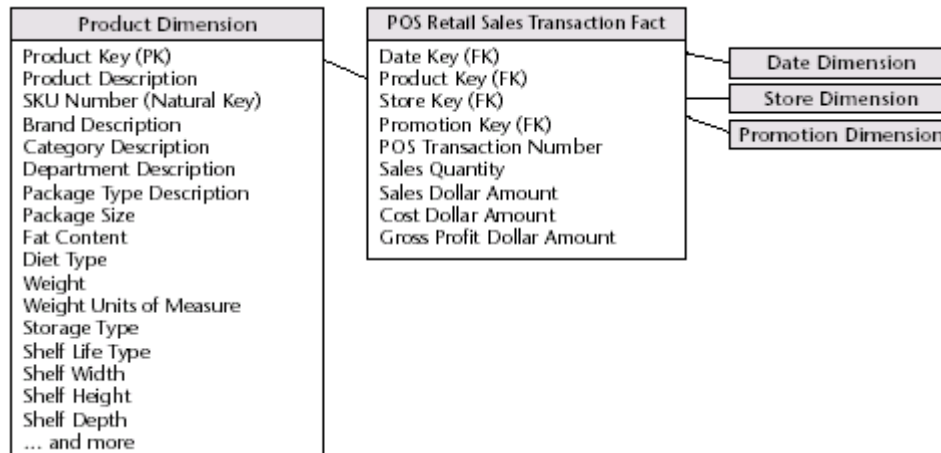
on month or year while avoiding a supposedly expensive join. This reasoning falls apart for several reasons. First of all, if our relational database can't handle an efficient join to the date dimension table, we're already in deep trouble. Most database optimizers are quite efficient at resolving dimensional queries; it is not necessary to avoid joins like the plague. Also, on the performance front, most databases don't index SQL date calculations, so queries constraining on an SQL-calculated field wouldn't take advantage of an index.

Date Key	Date	Full Date Description	Day of Week	Calendar Month	Calendar Year	Fiscal Year-Month	Holiday Indicator	Weekday Indicator
1	01/01/2002	January 1, 2002	Tuesday	January	2002	F2002-01	Holiday	Weekday
2	01/02/2002	January 2, 2002	Wednesday	January	2002	F2002-01	Non-Holiday	Weekday
3	01/03/2002	January 3, 2002	Thursday	January	2002	F2002-01	Non-Holiday	Weekday
4	01/04/2002	January 4, 2002	Friday	January	2002	F2002-01	Non-Holiday	Weekday
5	01/05/2002	January 5, 2002	Saturday	January	2002	F2002-01	Non-Holiday	Weekend
6	01/06/2002	January 6, 2002	Sunday	January	2002	F2002-01	Non-Holiday	Weekend
7	01/07/2002	January 7, 2002	Monday	January	2002	F2002-01	Non-Holiday	Weekday
8	01/08/2002	January 8, 2002	Tuesday	January	2002	F2002-01	Non-Holiday	Weekday

Product Dimension

The product dimension describes every SKU in the grocery store. While a typical store in our chain may stock 60,000 SKUs, when we account for different merchandising schemes across the chain and historical products that are no longer available, our product dimension would have at least 150,000 rows and perhaps as many as a million rows. The product dimension is almost always sourced from the operational product master file. Most retailers administer their product master files at headquarters and download a subset of the file to each store's POS system at frequent intervals. It is headquarters' responsibility to define the appropriate product master record (and unique SKU number) for each new UPC created by packaged goods manufacturers. Headquarters also defines the rules by which SKUs are assigned to such items as bakery goods, meat, and produce. We extract the product master file into our product dimension table each time the product master changes. An important function of the product master is to hold the many descriptive attributes of each SKU. The merchandise hierarchy is an important group of attributes. Typically, individual SKUs roll up to brands. Brands roll up to categories, and categories roll up to departments. Each of these is a many-to-one relationship. For each SKU, all levels of the merchandise hierarchy are well defined. Some attributes, such as the SKU description, are unique. In this case, there are at least 150,000 different values in the SKU description column. At the other extreme, there are only perhaps 50 distinct values of the department attribute. Thus, on average, there are 3,000 repetitions of each unique value in the department attribute. This is all right! We do not need to separate these repeated values into a second normalized table to save space. Remember that dimension table space requirements pale in comparison with fact table space considerations.

Product Key	Product Description	Brand Description	Category Description	Department Description	Fat Content
1	Baked Well Light Sourdough Fresh Bread	Baked Well	Bread	Bakery	Reduced Fat
2	Fluffy Sliced Whole Wheat	Fluffy	Bread	Bakery	Regular Fat
3	Fluffy Light Sliced Whole Wheat	Fluffy	Bread	Bakery	Reduced Fat
4	Fat Free Mini Cinnamon Rolls	Light	Sweeten Bread	Bakery	Non-Fat
5	Diet Lovers Vanilla 2 Gallon	Coldpack	Frozen Desserts	Frozen Foods	Non-Fat
6	Light and Creamy Butter Pecan 1 Pint	Freshlike	Frozen Desserts	Frozen Foods	Reduced Fat
7	Chocolate Lovers 1/2 Gallon	Frigid	Frozen Desserts	Frozen Foods	Regular Fat
8	Strawberry Ice Creamy 1 Pint	Icy	Frozen Desserts	Frozen Foods	Regular Fat
9	Icy Ice Cream Sandwiches	Icy	Frozen Desserts	Frozen Foods	Regular Fat



Many of the attributes in the product dimension table are not part of the merchandise hierarchy. The package-type attribute, for example, might have values such as Bottle, Bag, Box, or Other. Any SKU in any department could have one of these values. It makes perfect sense to combine a constraint on this attribute with a constraint on a merchandise hierarchy attribute. For example, we could look at all the SKUs in the Cereal category packaged in Bags. To put this another way, we can browse among dimension attributes whether or not they belong to the merchandise hierarchy, and we can drill up and drill down using attributes whether or not they belong to the merchandise hierarchy. We can even have more than one explicit hierarchy in our product dimension table. A reasonable product dimension table would have 50 or more descriptive attributes. Each attribute is a rich source for constraining and constructing row headers. Viewed in this manner, we see that drilling down is nothing more than asking for a row header that provides more information. Let's say we have a simple report where we've summarized the sales dollar amount and quantity by department.

Department Description	Sales Dollar Amount	Sales Quantity
Bakery	\$12,331	5,088
Frozen Foods	\$31,776	15,565

If we want to drill down, we can drag virtually any other attribute, such as brand, from the product dimension into the report next to department, and we automatically drill down to this next level of detail. Atypical drill down within the merchandise hierarchy would look like this:

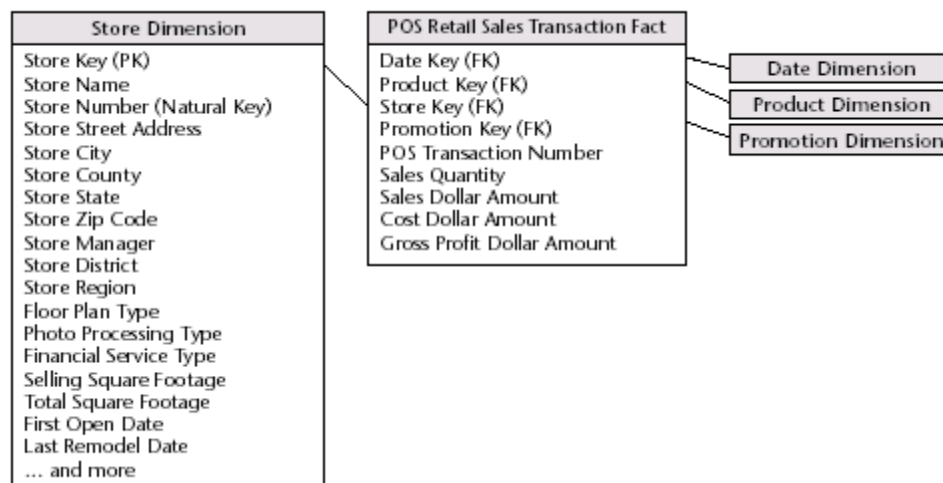
Department Description	Brand Description	Sales Dollar Amount	Sales Quantity
Bakery	Baked Well	\$3,009	1,138
Bakery	Fluffy	\$3,024	1,476
Bakery	Light	\$6,298	2,474
Frozen Foods	Coldpack	\$5,321	2,640
Frozen Foods	Freshlike	\$10,476	5,234
Frozen Foods	Frigid	\$7,328	3,092
Frozen Foods	Icy	\$2,184	1,437
Frozen Foods	QuickFreeze	\$6,467	3,162

Or we could drill down by the fat-content attribute, even though it isn't in the merchandise hierarchy roll-up.

Department Description	Fat Content	Sales Dollar Amount	Sales Quantity
Bakery	Non-Fat	\$6,298	2,474
Bakery	Reduced Fat	\$5,027	2,086
Bakery	Regular Fat	\$1,006	528
Frozen Foods	Non-Fat	\$5,321	2,640
Frozen Foods	Reduced Fat	\$10,476	5,234
Frozen Foods	Regular Fat	\$15,979	7,691

Store Dimension

The store dimension describes every store in our grocery chain. Unlike the product master file that is almost guaranteed to be available in every large grocery business, there may not be a comprehensive store master file. The product master needs to be downloaded to each store every time there's a new or changed product. However, the individual POS systems do not require a store master. Information technology (IT) staffs frequently must assemble the necessary components of the store dimension from multiple operational sources at headquarters. The store dimension is the primary geographic dimension in our case study. Each store can be thought of as a location. Because of this, we can roll stores up to any geographic attribute, such as ZIP code, county, and state in the United States. Stores usually also roll up to store districts and regions. These two different hierarchies are both easily represented in the store dimension because both the geographic and store regional hierarchies are well defined for a single store row.



The floor plan type, photo processing type, and finance services type are all short text descriptors that describe the particular store. These should not be one-character codes but rather should be 10- to 20-character standardized descriptors that make sense when viewed in a pull-down list or used as a report row header. The column describing selling square footage is numeric and theoretically additive across stores. One might be tempted to place it in the fact table. However, it is clearly a constant attribute of a store and is used as a report constraint or row header more often than it is used as an additive element in a summation. For these reasons, we are confident that selling square footage belongs in the store dimension table. The first open date and last remodel date typically are join keys to copies of the date dimension table. These date dimension copies are declared in SQL by the VIEW construct and are

semantically distinct from the primary date dimension. The VIEW declaration would look like `CREATE VIEW FIRST_OPEN_DATE (FIRST_OPEN_DAY_NUMBER, FIRST_OPEN_MONTH ...) AS SELECT DAY_NUMBER, MONTH, ...FROM DATE`

Now the system acts as if there is another physical copy of the date dimension table called `FIRST_OPEN_DATE`. Constraints on this new date table have nothing to do with constraints on the primary date dimension table. The first open date view is a permissible outrigger to the store dimension. Notice that we have carefully relabeled all the columns in the view so that they cannot be confused with columns from the primary date dimension.

Promotion Dimension

The promotion dimension is potentially the most interesting dimension in our schema. The promotion dimension describes the promotion conditions under which a product was sold. Promotion conditions include temporary price reductions, end-aisle displays, newspaper ads, and coupons. This dimension is often called a *causal* dimension (as opposed to a casual dimension) because it describes factors thought to cause a change in product sales. Managers at both headquarters and the stores are interested in determining whether a promotion is effective or not. Promotions are judged on one or more of the following factors:

___ Whether the products under promotion experienced a gain in sales during the promotional period. This is called the *lift*. The lift can only be measured if the store can agree on what the baseline sales of the promoted products would have been without the promotion. Baseline values can be estimated from prior sales history and, in some cases, with the help of sophisticated mathematical models.

___ Whether the products under promotion showed a drop in sales just prior to or after the promotion, canceling the gain in sales during the promotion (time shifting). In other words, did we transfer sales from regularly priced products to temporarily reduced-priced products?

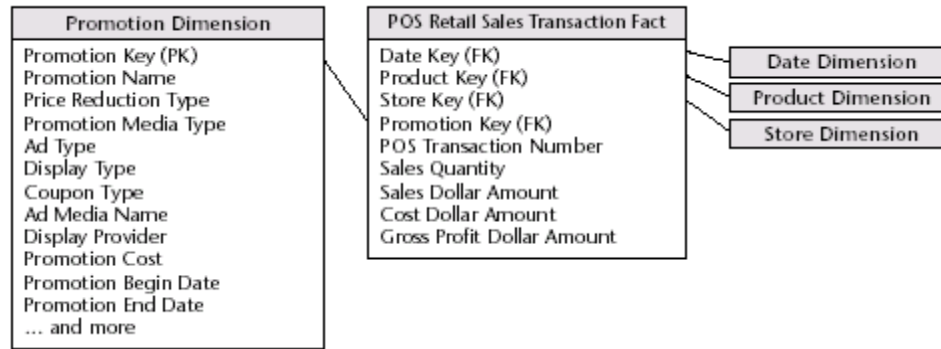
___ Whether the products under promotion showed a gain in sales but other products nearby on the shelf showed a corresponding sales decrease (cannibalization).

___ Whether all the products in the promoted category of products experienced a net overall gain in sales taking into account the time periods before, during, and after the promotion (market growth).

___ Whether the promotion was profitable. Usually the profit of a promotion is taken to be the incremental gain in profit of the promoted category over the baseline sales taking into account time shifting and cannibalization, as well as the costs of the promotion, including temporary price reductions, ads, displays, and coupons.

The causal conditions potentially affecting a sale are not necessarily tracked directly by the POS system. The transaction system keeps track of price reductions and markdowns. The presence of coupons also typically is captured with the transaction because the customer either presents coupons at the time of sale or does not. Ads and in-store display conditions may need to be linked from other sources.

The various possible causal conditions are highly correlated. A temporary price reduction usually is associated with an ad and perhaps an end-aisle display. Coupons often are associated with ads. For this reason, it makes sense to create one row in the promotion dimension for each combination of promotion conditions that occurs. Over the course of a year, there may be 1,000 ads, 5,000 temporary price reductions, and 1,000 end-aisle displays, but there may only be 10,000 combinations of these three conditions affecting any particular product. For example, in a given promotion, most of the stores would run all three promotion mechanisms simultaneously, but a few of the stores would not be able to deploy the end-aisle displays. In this case, two separate promotion condition rows would be needed, one for the normal price reduction plus ad plus display and one for the price reduction plus ad only.



From a purely logical point of view, we could record very similar information about the promotions by separating the four major causal mechanisms (price reductions, ads, displays, and coupons) into four separate dimensions rather than combining them into one dimension. Ultimately, this choice is the designer's prerogative. The tradeoffs in favor of keeping the four dimensions together include the following:

- Since the four causal mechanisms are highly correlated, the combined single dimension is not much larger than any one of the separated dimensions would be.
- The combined single dimension can be browsed efficiently to see how the various price reductions, ads, displays, and coupons are used together.
- However, this browsing only shows the possible combinations. Browsing in the dimension table does not reveal which stores or products were affected by the promotion. This information is found in the fact table.
- The tradeoffs in favor of separating the four causal mechanisms into distinct dimension tables include the following:
- The separated dimensions may be more understandable to the business community if users think of these mechanisms separately. This would be revealed during the business requirement interviews.
- Administration of the separate dimensions may be more straightforward than administering a combined dimension.

Keep in mind that there is no difference in the information content in the data warehouse between these two choices. Typically, many sales transaction line items involve products that are not being promoted. We will need to include a row in the promotion dimension, with its own unique key, to identify "No Promotion in Effect" and avoid a null promotion key in the fact table. Referential integrity is violated if we put a null in a fact table column declared as a foreign key to a dimension table. In addition to the referential integrity alarms, null keys are the source of great confusion to our users because they can't join on null keys.