LECTURE 11

Process of Dimensional Modeling

Learning Goals

Business Process

Grain

Choose the Facts

Choose the Dimensions

The Process of Dimensional Modeling

Four Step Method from ER to DM

1. Choose the Business Process
2. Choose the Grain
3. Choose the Facts
4. Choose the Dimensions

A typical ER diagram covers the entire spectrum of the business, actually covers every possible business process. However, in reality those multitudes of process do not co-exist in time and space (tables). As a consequence, an ER diagram is overly complex, and is a demerit to itself. This is precisely the point that differentiates a DM from an ER diagram, as a single ER diagram can be divided into multiple DM diagrams. Thus a step-wise approach is followed to separate the DMs from an ER diagram, and this consists of four steps.

Step-1: Separate the ER diagram into its discrete business processes and to model each business process separately.

Step-2: Grain of a fact table = the meaning of one fact table row. Determines the maximum level of detail of the warehouse.

Step-3: Select those many-to-many relationships in the ER model containing numeric and additive non-key items and designate them as fact tables. Actually all business events to be analyzed are gathered into fact tables.

Step-4: De-normalize all of the remaining tables into flat tables with single-part keys that connect directly to the fact tables. These tables become the dimension tables. They are like reference

© St Paul's University

tables that define how to analyze the fact information. They are typically small and relatively static.

Let's discuss each of the steps in detail, one by one.

Step-1: Choose the Business Process

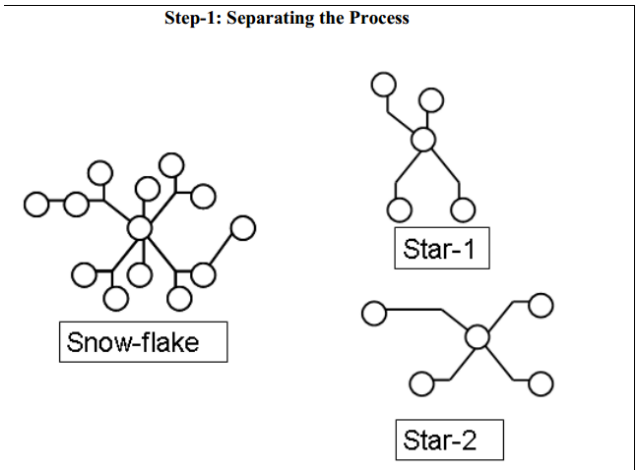
A business process is a major operational process in an organization.

Typically supported by a legacy system (database) or an OLTP.
Examples: Orders, Invoices, Inventory etc.
Business Processes are often termed as Data Marts and that is why many people criticize DM as being data mart oriented.

The first step in DM is the business process selection. What do we mean by a process? A process is a natural business activity in the organization supported by a legacy source data-collection system (database or OLTP). Example business processes include purchasing, orders, shipments, invoicing, inventory, and general ledger etc.

Many people consider business processes as Data marts i.e. in their view organizational or departmental function is referred as the business process. That is why such people criticize DM as being a data mart oriented approach. However, in Kimball's view, it is a wrong approach, the two must not be confused e.g. a single model is built to handle orders data rather than building separate models for marketing and sales departments, which both access the orders data. By focusing on business processes, rather than departments, consistent information can be delivered economically throughout the organization. Building departmental data models may result in data duplication, data inconsistencies, and data management issues. What is a possible solution? Yes, publishing data once can not only reduce the consistency problems, but can also reduce ETL development, data management and disk storage issues as well.

© St Paul's University



Step-1: Separating the Process

shows an interesting concept i.e. separating business processes to be modeled from a complex set of processes. This translates to splitting a snow-flake schema into multiple star schemas. Note that as the processes move into the star schema all the hierarchies collapse

Step-2: Choosing the Grain

Grain is the fundamental, atomic level of data to be represented.

Grain is also termed as the unit of analyses.

Example grain statements

- Typical grains
    - Individual Transactions
    - Daily aggregates (snapshots)
    - Monthly aggregates
- Relationship between grain and expressiveness.
- Grain vs. hardware trade-off.

Grain is the lowest level of detail or the atomic level of data stored in the warehouse. The lowest level of data in the warehouse may not be the lowest level of data recorded in the business system. It is also termed as the unit of analysis e.g. unit of weight is Kg etc.

Example grain statements: (*one fact row represents a…*)

© St Paul's University

- Entry from a cash register receipt
- Boarding pass to get on a flight
- Daily snapshot of inventory level for a product in a warehouse
- Sensor reading per minute for a sensor
- Student enrolled in a course

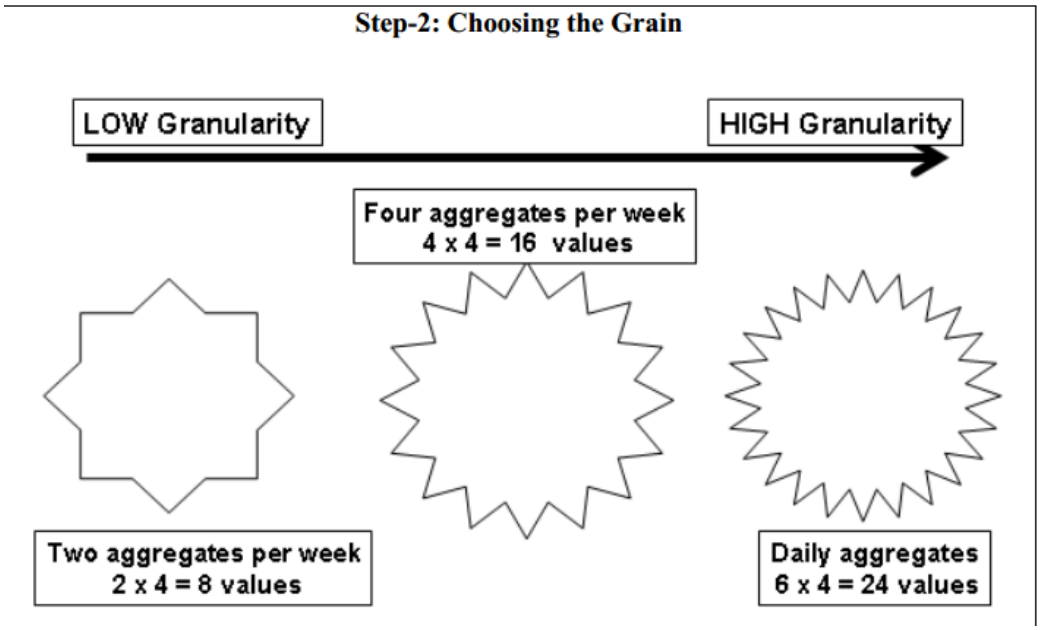
Finer-grained fact tables:

- are more expressive
- have more rows

Trade-off between performance and expressiveness

- Rule of thumb: Err in favor of expressiveness
- Pre-computed aggregates can solve performance problems

In the absence of aggregates, there is a potential to waste millions of dollars on hardware upgrades to solve performance problems that could have been otherwise addressed by aggregates.



Step-2: Choosing the Grain

Note that you may come across definitions of grain as given in the notes and discussed in the lectures, but you may also come across definitions that are different from those discussed. This depends on the interpretation of the writer. We will follow the definition as per Fig-14.2.

© St Paul's University

The case FOR data aggregation

- Works well for repetitive queries.
- Follows the known thought process.
- Justifiable if used for max number of queries.
- Provides a "big picture" or macroscopic view.
- Application dependent, usually inflexible to business changes (remember lack of absoluteness of conventions).

There are both positives and negatives to data aggregation. These are a list of the reasons for the utilization of summary or aggregate data. As you can see, they all really fall under the area of "performance".

The negative side is that summary data does not allow a total solution with the flexibility and capabilities that some businesses truly require as compared to other businesses.

The case AGAINST data aggregation

- Aggregation is irreversible.
- Can create monthly sales data from weekly sales data, but the reverse is not possible.
- Aggregation limits the questions that can be answered.
- What, when, why, where, what-else, what-next
- Aggregation can hide crucial facts.
- The average of 100 & 100 is same as 150 & 50

Aggregation is one-way i.e. you can create aggregates, but cannot dissolve aggregates to get the original data from which the aggregates were created. For example 3+2+1 = 6 at the same time 2+4 also equals 6, so does 5+1 and if we consider reals, then infinitely many ways of adding numbers to get the same result.

If you think about the "5 W's of journalism", these are the "6 W's of data analysis". Again it highlights the types of questions that end users want to ask and cannot be answered by summary

© St Paul's University

data.

By definition, a summarization will consider at least one of these points irrelevant. For example, a summary across the company takes out the dimension of "WHERE" and a summary by quarters takes out the element of "WHEN". The point to be noted is that although summary data has a purpose, yet one can take any summary and ask a question that the system cannot answer.

Aggregation hides crucial facts

	Week-1	Week-2	Week-3	Week-4	Average
Zone-1	Just Looking at the averages i.e. aggregates				100
Zone-2					100
Zone-3					100
Zone-4					100
Average	100	100	100	100	

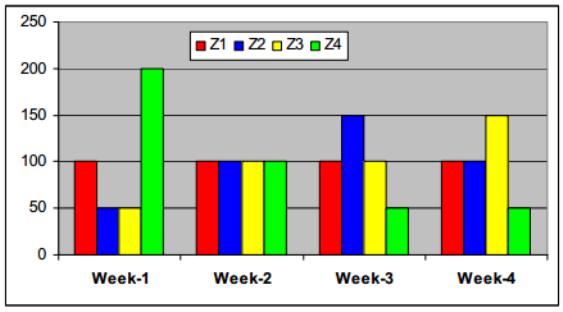
Aggregation hides crucial facts

Consider the sales data of an item sold in a chain store in four zones, such that the sales data is aggregated across the weeks also. For this simple example, for the sake of conserving space the average sales across each zone and for each week is stored. Therefore, instead of storing 16 values only 8 values are stored i.e. a saving of 50% space.

Assume that a promotional scheme or advertisement campaign was run, and then the sales data was recorded to analyze the effectiveness of the campaign. If, we look at the averages (as shown in the table) there is no change in sales i.e. neither across time nor across the geography dimension. On the face of it, it was an in effective campaign. Now let's raise the curtain and look at the detailed sales records. The numbers are NOT constant! Drawing the graphs of the sales records shows a very different picture

© St Paul's University

Aggregation hides crucial facts



Z1: Sale is constant (need to work on it)
Z2: Sale went up, then fell (need of concern)
Z3: Sale is on the rise, why?
Z4: Sale dropped sharply, need to look deeply.
W2: Static sale

Z1: Sale is constant throughout the month (need to work on it)

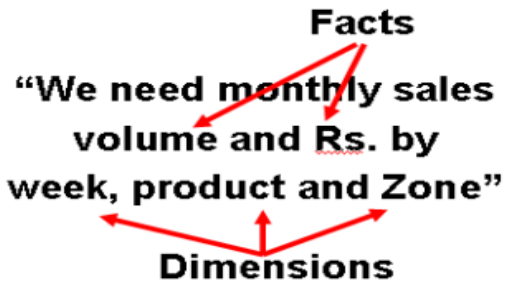
Z2: Sale went up, then fell (need of concern) i.e. the campaign was effective, but after week it fizzled down.

Z3: Sale is on the rise, why?

Z4: Sale dropped sharply, need to look deeply. It seems that the campaign had a negative effect on the sales?

W2: Static sale across all zones, very unique indeed.

Step 3: Choose Facts



© St Paul's University

Step 3: Choose Facts

Numeric facts are identified by answering the question "what are we measuring?" Many- to-many relationships in the ER model containing numeric and additive non-key items are selected and designated as fact tables. In the example numeric additive figures volume (quantity ordered) and Rs. (Rupees cost amount) are the facts because the numeric values of the two are of keen interest for the business user.

Step 3: Choose Facts

Choose the facts that will populate each fact table record.

Remember that best Facts are Numeric, Continuously Valued and Additive.

Example: Quantity Sold, Amount etc.

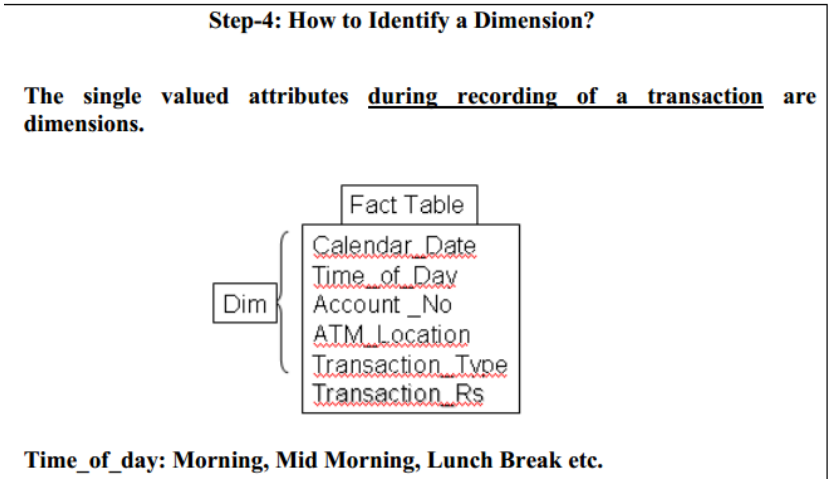
It should be remembered that facts are numeric, continuous, additive and non-key items that will populate the fact table. Example facts for a point of sales terminal (POS) are sales quantity, per unit sales price, and the sales amount in rupees. All the candidate facts in a design must be true to the grain described in previous slides. Facts that clearly belong to a different grain must be in a separate fact table.

Step 4: Choose Dimensions

Choose the dimensions that apply to each fact in the fact table.

- Typical dimensions: time, product, geography etc.
- Identify the descriptive attributes that explain each dimension.
- Determine hierarchies within each dimension.

© St Paul's University



Step-4: How to Identify a Dimension?

The dimension tables usually represent textual attributes that are already known about things such as the product, the geography, or the time. If the database designer is very clear about the grain of the fact table, then choosing the appropriate dimensions for the fact table is usually easy. What is so special about it, seems to be pretty intuitive, but is not.

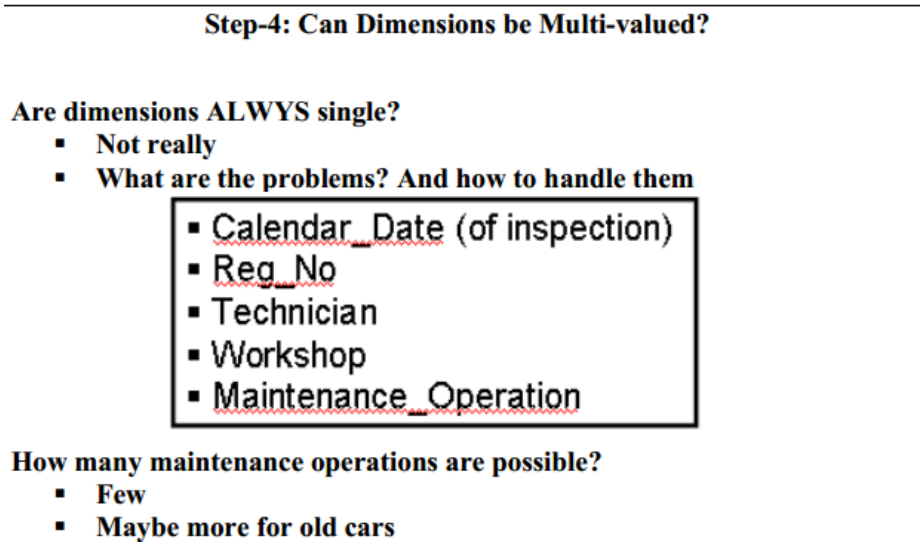
The success in selecting the right dimensions for a given fact table is dependent on correctly identifying any description that has a single value for an individual fact table record or transaction. Note that the fact table record considered could be a single transaction or weekly aggregate or monthly sums etc i.e. a grain is associated. Once this is correctly identified and settled, then as many dimensions can be added to the fact able as required. For the ATM customer transaction example, the following dimensions all have a single value during the recording of the transaction, as none of the above dimensions change during a single transaction:

? Calendar Date
? Time_of_Day
? Account _No
? ATM_Location
? Transaction_Type (withdrawal, deposit, balance inquiry etc.)

Over here Time_of_Day refers to specific periods such as Morning, Mid Morning, Lunch Break, Office_Off etc. Note that during an atomic transaction, the value of Time_of_Day does not

© St Paul's University

change (as a transaction takes less than a minute), hence it is a dimension. In the context of the ATM example, the only numeric attribute is the Transaction_Rs, so it is a fact. Observe that we use this convention in real life also, when people say we will visit you first time or second time



of the day etc.

Step-4: Can Dimensions be Multi-valued?

After convincing ourselves that dimensions are really single valued, perhaps we should consider whether there are ever legitimate exceptions i.e. is it possible to have multi-valued dimension in a fact table? If this is conceivable, what problems might arise?

Consider the following example from vehicle maintenance system used at a vehicle service center. You are handed a data sheet for which the grain is the individual line item on the customer's bill. The data source could be your periodic car maintenance visits to the company workshop or individual replacement charges on a repair/change bill. These individual line items have a rich set of dimensions such as:

? Calendar_Date (of inspection)
? Reg_No (of vehicle)
? Technician_ID
? Workshop
? Maintenance Operation

The numeric additive facts in this design (which are the core of every fact table in a dimensional

© St Paul's University

design) would include Amount_Charged and perhaps others including Amount_Paid, depending upon if the vehicle was insured etc.

On the face of it, this seems to be a very straightforward design, with obvious single values for all the dimensions. But there is a surprise. In many situations, there may be multiple values for services performed, such as oil change, air filter change, spark plug change etc. What do you do if for a certain car there are three separate changes at the moment the service was performed? How about really old and ill-kept cars that might have upto 5+ such changes? How do you encode the Maintenance_Operation dimension if you wish to represent this information?

Step-4: Dimensions & Grain

Several grains are possible as per business requirement.

For some aggregations certain descriptions do not remain atomic.

Example: Time_of_Day may change several times during daily aggregate, but not during a transaction.

Choose the dimensions that are applicable within the selected grain.

Strangely, there is a relationship between the grain and the dimensions. When building a fact table, the most important step is to declare the grain (aggregation level) of the fact table. The grain declares the exact meaning of an individual fact record. Consider the case of transactions for an ATM machine. The grain could be individual customer transaction, or number of transaction per week or the amount drawn per month.

? Calendar_Date
? Time_of_Day
? Account _No
? ATM_Location
? Transaction_Type (withdrawal, deposit, balance inquiry etc.)

Note that none of the above dimensions change during a single transaction. However, for weekly transactions probably only Account _No and ATM_Location can be treated as a dimension. Note that higher the level of aggregation of the fact table, the fewer will be the number of

© St Paul's University

dimensions you can attach to the fact records. The converse of this is surprising. The more granular the data, the more dimensions make sense. Hence the lowest-level data in any organization is the most dimensional.

Issues of Dimensional Modeling

Learning Goals

- Additive vs. Non-Additive facts
- Classification of Aggregation Functions
- Not recording Facts
- A Fact-less Fact Table
- OLTP & Slowly Changing Dimensions

Step 3: Additive vs. Non-Additive facts

- **Additive facts are easy to work with**
  - **Summing the fact value gives meaningful results**
  - **Additive facts:**
    - **Quantity sold**
    - **Total Rs. sales**
  - **Non-additive facts:**
    - **Averages (average sales price, unit price)**
    - **Percentages (% discount)**
    - **Ratios (gross margin)**
    - **Count of distinct products sold**

Month	Crates of Bottles Sold
May	14
Jun.	20
Jul.	24
TOTAL	58

Month	% discount
May	10
Jun.	8
Jul.	6
TOTAL	24% ← Incorrect!

There can be two types of facts i.e. additive and non-additive. Additive facts are those facts which give the correct result by an addition operation. Examples of such facts could be number of items sold, sales amount etc. Non-additive facts can also be added, but the addition gives incorrect results. Some examples of non-additive facts are average, discount, ratios etc. Consider three instances of 5, with the sum being 15 and average being 5. Now consider two numbers i.e. 5 and 10, the sum being 15, but the average being 7.5. Now, if the average of 5 and 7.5 is taken this comes to be 6.25, but if the average of the actual numbers is taken, the sum comes to be 30

© St Paul's University

and the average being 6. Hence averages, if added gives wrong results. Now facts could be averages, such as average sales per week etc, thus they are perfectly legitimate facts.

Step-3: Classification of Aggregation Functions

How hard to compute aggregate from sub-aggregates?
Three classes of aggregates:

- Distributive

1. Compute aggregate directly from sub-aggregates
2. Examples: MIN, MAX ,COUNT, SUM

- Algebraic

1. Compute aggregate from constant-sized summary of subgroup
2. Examples: STDDEV, AVERAGE
3. For AVERAGE, summary data for each group is SUM, COUNT

- Holistic

1. Require unbounded amount of information about each subgroup
2. Examples: MEDIAN, COUNT DISTINCT
3. Usually impractical for a data warehouses!

We see that calculating aggregates from aggregates is desirable, but is not possible for non-additive facts. So we deal with three types of aggregates i.e. distributive that are additive in nature, and then algebraic which are non-additive in nature. Therefore, such aggregates have to be computed from summary of subgroups to avoid the problem of incorrect results. The of course are the holistic aggregates that give a complete picture of the data, such as median, or distinct values. However, such aggregates are not desirable for a data warehouse environment, as it requires a complete scanning, which is highly undesirable as it consumes lot of time.

Step-3: Not recording Facts

Transactional fact tables don't have records for events that don't occur Example:No records(rows) for products that were not sold. This has both advantage and disadvantage.
Advantage:
Benefit of sparsity of data

© St Paul's University

Significantly less data to store for "rare" events

Disadvantage: Lack of information

Example: What products on promotion were not sold?

Fact tables usually don't records events that don't happen, such as items that were not sold. The advantage of this approach is getting around the problem of sparsely. Recall that when we discussed MOLAP, we discussed the sales of different items not occurring in different geographies and in different time frames, resulting in sparse cubes. If however this data is not recorded, then significantly less data will be required to be stored. But what if, from the point of view of decision making, such data has to be retrieved, how to retrieve data corresponding to those items? To find such items, additional queries will be required to check the current item balance with the item balance when the items where (say) brought into the store. So the biggest disadvantage of this approach is key data is not recorded.

Step-3: A Fact-less Fact Table

"Fact-less" fact table

- A fact table without numeric fact columns
- Captures relationships between dimensions
- Use a dummy fact column that always has value 1

The problem of not recording non-events is solved by using fact-less fact tables, as not recording such information resulted in loss of data. Such a fact-less fact table is one which does not have numeric values stored in the corresponding column, as such tables are used to capture the relationships between dimensions. Fact less fact table captures the many-to-many relationships between dimensions, but contains no numeric or textual facts. To achieve this dummy value of 1 is used in the corresponding column.

Step-3: Example: Fact-less Fact Tables

Examples:

Department/Student mapping fact table

© St Paul's University

What is the major for each student?

Which students did not enroll in ANY course

Promotion coverage fact table

Which products were on promotion in which stores for which days?

Kind of like a periodic snapshot fact

Some of the examples of fact-less fact tables: Consider the case of a department/student mapping fact table. The data is recorded for each student who registers for a course, but there may be students that do not register in any course. If data is useful from the point of view of identifying those students which are skipping a semester, there is no direct or simple way to identify such students, the solution is a fact-less fact table. Similarly which items on promotion are not selling, as the sales records are for only those items that are sold.

Step-4: Handling Multi-valued Dimensions?

One of the following approaches is adopted:

Drop the dimension. Use a primary value as a single value.

Add multiple values in the dimension table.

Use “Helper” tables.

For handling the exceptions in dimensions, designers adopt one of the following approaches:

? Drop the Maintenance Operation dimension as it is multi-valued.
? Choose one value (as the "primary" maintenance) and omit the other values.
? Extend the dimension list and add a fixed number of maintenance dimensions.
? Put a helper table in between this fact table and the Maintenance dimension table.

Instead of ignoring the problem and dropping the dimension altogether, let’s tackle the problem.

Usually the designers go for the second alternative, as a consequence this will show up as the

© St Paul’s University

primary, or main maintenance operation. Such as20,000Km maintenance or 40,000 Km maintenance. It is known what would constitute for each mileage based maintenance, and these maintenance are also mutually exclusive i.e. single valued. In many cases, you may actually come across this practice being observed in the OLTP systems. The obvious advantage is that the modeling problem is resolved, but the disadvantage is that the usefulness of the data becomes questionable. Why? Because with the passage of time or with new models of vehicles coming or because of company policy, what constitutes service at 20,000 Km may actually change. Will need meta data to resolve this issue.

The third alternative of creating a fixed number of additional columns in the dimension table is a quick and dirty approach and should be avoided. There is likely to be car that may require more changes then reflected in the table, or the company policy changes and more items fall under the maintenance, and a long list will result in many null entries for a typical car, especially new ones. Furthermore, it is not easy to query the multiple separate maintenance dimensions and will result in slow queries. Therefore, multiple dimensions style of design should be avoided.

The last alternative is usually adopted, and a "helper" table is placed between the Maintenance dimension and the fact table, although this adulterates or dilutes the star schema. More details are beyond the scope of this course.

Step-4: OLTP & Slowly Changing Dimensions

OLTP systems not good at tracking the past. History never changes.

OLTP systems are not “static” always evolving, data changing by overwriting.

Inability of OLTP systems to track history, purged after 90 to 180 days. Actually don’t want to keep historical data for OLTP system.

One major difference between an OLTP system and a data warehouse is the ability and the responsibility to accurately describe the past. OLTP systems are usually very poor at correctly representing a business as of a month or a year ago for several reasons as discussed before. A good OLTP system is always evolving. Orders are being filled and, thus, the order backlog is constantly changing. Descriptions of products, suppliers, and customers are constantly being

© St Paul’s University

updated, usually by overwriting. The large volume of data in an OLTP system is typically purged every 90 t o 180 days. For these reasons, it is difficult for an OLTP system to correctly represent the past. In an OLTP system, do you really want to keep old order statuses, product descriptions, supplier descriptions, and customer descriptions over a multiyear period?

Step-4: DWH Dilemma: Slowly Changing Dimensions

The responsibility of the DWH to track the changes.

Example: Slight change in description, but the product ID (SKU) is not changed.

Dilemma: Want to track both old and new descriptions, what do they use for the key? And where do they put the two values of the changed ingredient attribute?

The data warehouse must accept the responsibility of accurately describing the past. By doing so, the data warehouse simplifies the responsibilities of the OLTP system. Not only does the data warehouse relieve the OLTP system of almost all forms of reporting, but the data warehouse contains special structures that have several ways of tracking historical data.

A dimensional data warehouse database consists of a large central fact table with a multipart key. This fact table is surrounded by a single layer of smaller dimension tables, each containing a single primary key. In a dimensional database, these issues of describing the past mostly involve slowly changing dimensions. A typical slowly changing dimension is a product dimension in which the detailed description of a given product is occasionally adjusted. For example, a minor ingredient change or a minor packaging change may be so small that production does not assign the product a new SKU number (which the data warehouse has been using as the primary key in the product dimension), but nevertheless gives the data warehouse team a revised description of t he product. The data warehouse team faces a dilemma when this happens. If they want the data warehouse to track both the old and new descriptions of the product, what do they use for the key? And where do they put the two values of the changed ingredient

Step-4: Explanation of Slowly Changing Dimensions…

Compared to fact tables, contents of dimension tables are relatively stable.

- New sales transactions occur constantly.

© St Paul's University

- New products are introduced rarely.
- New stores are opened very rarely.

The assumption does not hold in some cases

- Certain dimensions evolve with time
- e.g. description and formulation of products change with time
- Customers get married and divorced, have children, change addresses etc.
- Land changes ownership etc.
- Changing names of sales regions.

For example, a minor ingredient change or a minor packaging change may be so small that production does not assign the product a new SKU number (which the data warehouse has been using as the primary key in the product dimension), but nevertheless gives the data warehouse team a revised description of t he product. The data warehouse team faces a dilemma when this happens. If they want the data warehouse to track both the old and new descriptions of the product, what do they use for the key? And where do they put the two values of the changed ingredient attribute?

Other common slowly changing dimensions are the district and region names for a sales force. Every company that has a sales force reassigns these names every year or two. This is such a common problem that this example is something of a joke in data ware housing classes. When the teacher asks, "How many of your companies have changed the organization of your sales force recently?" everyone raises their hands.

Step-4: Explanation of Slowly Changing Dimensions…

Although these dimensions change but the change is not rapid, therefore called “Slowly” Changing Dimensions.

There can be many examples. For a young customer who is single, then after a while the customer gets married. After sometime there are children, in unfortunate cases the marriage breaks so the customer is separated or the husband dies and the customer becomes a widow. This

© St Paul’s University

just does not typically happen overnight but takes a while. Another example is inheritance, consider the example of land. Over a period of time the land changes hands, is split because of inheritance or its size increases by buying. Again things don't happen overnight, but take a while, hence slowly changing dimensions.

Step-4: Handling Slowly Changing Dimensions

Option-1: Overwrite History

- Example: Code for a city, product entered incorrectly
- Just overwrite the record changing the values of modified attributes.
- No keys are affected.
- No changes needed elsewhere in the DM.
- Cannot track history and hence not a good option in DSS.

The first technique is the simplest and fastest. But it doesn't maintain past history! Nevertheless, overwriting is frequently used when the data warehouse team legitimately decides that the old value of the changed dimension attribute is not interesting .For example, if you find incorrect values in the city and state attributes in a customer record, then overwriting would almost certainly be used. After the overwrite,certain old reports that depended on the city or state values would not return exactly the same values. Most of us would argue that this is the correct outcome.

Step-4: Handling Slowly Changing Dimensions

Option-2: Preserve History

Example: The packaging of a part change from glued box to stapled box, but the code assigned (SKU) is not changed.

Create an additional dimension record at the time of change with new attribute values.

Segments history accurately between old and new description Requires adding two to three version numbers to the end of key. SKU#+1, SKU#+2 etc.

© St Paul's University

Suppose you work in a manufacturing company and one of your main data warehouse schemas is the company's shipments. The product dimension is one of the most important dimensions in this dimensional schema. A typical product dimension would have several hundred detailed records, each representing a unique product capable of being shipped. A good product dimension table would have at least 50 attributes describing the products, including hierarchical attributes such as brand and category, as well as nonhierarchical attributes such as color and package type. An important attribute provided by manufacturing operations is the SKU number assigned to the product. You should start by using the SKU number as the key to the product dimension table. Suppose that manufacturing operations makes a slight change in packaging of SKU #38, and the packaging description changes from "glued box" to "pasted box." Along with this change, manufacturing operations decides not to change the SKU number of the product, or the bar code (UPC) that is printed on the box. If the data warehouse team decides to track this change, the best way to do this is to issue another product record, as if the pasted box version were a brand new product. The only difference between the two product records is the packaging description. Even the SKU numbers are the same. The only way you can issue another record is if you generalize the key to the product dimension table to be something more than the SKU number. A simple technique is to use the SKU number plus two or three version digits. Thus the first instance of the product key for a given SKU might be SKU# + 01. When, and if, another version is needed, it becomes SKU# +, and so on. Notice that you should probably also park t he SKU number in a separate dimension attribute (field) because you never want an application to be parsing the key to extract the underlying SKU number. Note the separate SKU attribute in the Product dimension in This technique for tracking slowly changing dimensions is very powerful because new dimension records automatically partition history in the fact table. The old version of the dimension record points to all history in the fact table prior to the change. The new version of the dimension record points to all history after the change. There is no need for a timestamp in the product table to record the change. In fact, a timestamp in the dimension record may be meaningless because the event of interest is t he actual use of the new product type in a shipment. This is best recorded by a fact table record with the correct new product key.

Another advantage of this technique is that you can gracefully track as many changes to a dimensional item as you wish. Each change generates a new dimension record,and each record partitions history perfectly. The main drawbacks of the technique are the requirement to

© St Paul’s University

generalize the dimension key, and the growth of the dimension table itself.

Step-4: Handling Slowly Changing Dimensions

Option-3: Create current valued field

Example: The name and organization of the sales regions change over time, and want to know how sales would have looked with old regions.

- Add a new field called current region rename old to previous region.
- Sales record keys are not changed.
- Only TWO most recent changes can be tracked.

Creating a Current Value Field

You use the third technique when you want to track a change in a dimension value, but it is legitimate to use the old value both before and after the change. This situation occurs most often in the infamous sales force realignments, where although you have changed the names of your sales regions, you still have a need to state today's sales in terms of yesterday's region names, just to "see how they would have done" using the old organization. You can attack this requirement, not by creating a new dimension record as in the second technique, but by creating a new "current value" field. Suppose in a sales team dimension table, where the records represent sales teams, you have a field called "region." When you decide to rearrange the sales force and assign each team to newly named regions, you create a new field in the sales dimension table called "current region." You should probably rename the old field "previous region." No alterations are made to the sales dimension record keys or to the number of sales team records. These two fields now allow an application to group all sales fact records by either the old sales assignments (previous region) or the new sales assignments (current region). This schema allows only the most recent sales force change to be tracked, but it offers the immense flexibility of being able to state all of the history by either of the two sales force assignment schemas. It is conceivable, although somewhat awkward, to generalize this approach to the two most recent changes. If many of these sales force realignments take place and it is desired to track them all, then the

© St Paul's University

second technique should probably be used.

Step-4: Pros and Cons of Handling

Option-1: Overwrite existing value
Simple to implement
No tracking of history

Option-2: Add a new dimension row
Accurate historical reporting
Pre-computed aggregates unaffected
Dimension table grows over time
Option-3: Add a new field
Accurate historical reporting to last TWO changes
Record keys are unaffected
Dimension table size increases

There are number of ways of handling slowly changing dimensions. Some of the methods are simple, but not desirable; but all have their own pros and cons. The simplest possible "solution" is to overwrite history. If the customer was earlier single, and gets married, just change his/here status from single to married. Very simple to implement, but not desirable, as a DWH is about recording historical data, and by virtue of overwriting, the historical data is destroyed. Another option is to add a row when the dimension changes, the obvious benefit is that history is not lost, but over the period of time the dimension table will grow as new rows are added corresponding to the changes in the dimensions. The third, rather desirable approach is to add an additional column, that does increase the table size, but the increase is not non-deterministic. The column records the last two changes, if the dimension changes more than twice, then historical data is lost.

Step-4: Junk Dimension

Sometimes certain attributes don't fit nicely into any dimension

- Payment method (Cash vs. Credit Card vs. Check)
- Bagging type (Paper vs. Plastic vs. None)

© St Paul's University

Create one or more “mix” dimensions

- Group together leftover attributes as a dimension even if not related
- Reduces number of dimension tables, width of fact table
- Works best if leftover attributes are

Few in number

Low in cardinality

Correlated

Other options

Each leftover attribute becomes a dimension

Eliminate leftover attributes that are not useful

A junk dimension is a collection of random transactional codes, flags and/or text attributes that are unrelated to any particular dimension. The junk dimension is simply a structure that provides a convenient place to store the junk attributes. A good example would be a trade fact in a company that brokers equity trades.

The need for junk dimensions arises when we are considering single level hierarchies. The only problem with the single level hierarchies is that you may have a lot of them in any given dimensional model. Ideally, the concatenated primary key of a fact table should consist of fewer than 10 foreign keys. Sometimes, if all of the yes/no flags are represented as single level hierarchy dimensions, you may end up with 30 or more. Obviously, this is an overly complex design.

A technique that allows reduction of the number of foreign keys in a fact table is the creation of "junk" dimensions. These are just "made up" dimensions where you can put several of these single level hierarchies. This cuts down the number of foreign keys in the fact table dramatically.

As to the number of flags before creating a junk dimension, if there are more than 15 dimensions, where five or more are single level hierarchies, I start seriously thinking about combining them into one or more junk dimensions. One should not indiscriminately combine 20 or 30 or 80 single level hierarchies.

© St Paul’s University