

Data Warehouse

Md. Shafiuzzaman Rajib
CSE SUST 99

What We will Discuss?

- Data Warehouse
- Dimensional Modeling
- Dimension
- Fact
- Star Schema
- Snow-Flake Schema

History of Data Warehouse

The history of data warehouse dates back to 1960 when General Mills and Dartmouth College, in a joint research project, develop the terms dimensions and facts. In 1970, ACNielsen, a global marketing research company, published sales data pertaining to retail industry in the form of dimensional data mart. Earlier than this, the concept of data warehouse for analysis was only a subject of academic pursuit.

History of Data Warehouse Continued

In the year 1992 Bill Inmon published a book - Building the Data Warehouse - which gave us a widely accepted definition of what a data warehouse is. Another author - Ralph Kimball - who 4 years later in 1996 wrote another book - Data warehouse toolkit - showing us yet another approach of defining and building a data warehouse. Since then, both Inmon and Kimball approaches are widely accepted and implemented throughout the globe.

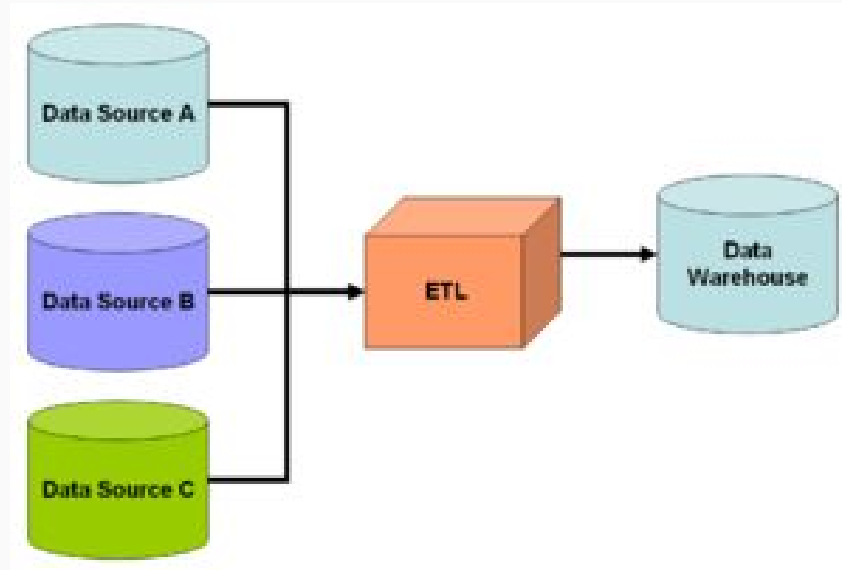
What is Data Warehouse?

A data warehouse is an electronically stored collection of integrated data that can be used for the purpose of intelligent analysis.

Why Data Warehouse?

Data warehouse provides historical, current and predictive views of business operations by analyzing the present and historical business data. Data analysis is often done using visualization techniques that turn complex data into images that tell a compelling story. Raw data by this process of analysis helps management take right decisions.

Schematic View of a data warehouse



Difference between OLTP and OLAP

OLTP is the transaction system that collects business data. Whereas OLAP is the reporting and analysis system on that data.

OLTP systems are optimized for INSERT, UPDATE operations and therefore highly normalized. On the other hand, OLAP systems are deliberately denormalized for fast data retrieval through SELECT operations.

Difference between OLTP and OLAP Continued

In a departmental shop, when we pay the prices at the check-out counter, the sales person at the counter keys-in all the data into a "Point-Of-Sales" machine. That data is transaction data and the related system is a OLTP system.

On the other hand, the manager of the store might want to view a report on out-of-stock materials, so that he can place purchase order for them. Such report will come out from OLAP system.

Dimensional Modeling

Dimensional model consists of dimension and fact tables. Fact tables store different transactional measurements and the foreign keys from dimension tables that qualifies the data. The goal of Dimensional model is not to achieve high degree of normalization but to facilitate easy and faster data retrieval.

Ralph Kimball is one of the strongest proponents of this very popular data modeling technique which is often used in many enterprise level data warehouses.

Goals and Benefits of Dimensional Modelling

The goal of dimensional modelling can be summarized as:

- Faster Data Retrieval
- Better Understandability
- Extensibility

What is Dimension?

A dimension is something that qualifies a quantity (measure). For an example, consider this: If I just say... "20kg", it does not mean anything. But if I say, "20kg of Rice (Product) is sold to Ratul (customer) on 5th April (date)", then that gives a meaningful sense. These product, customer and dates are some dimension that qualified the measure - 20kg.

Dimensions are mutually independent. Technically speaking, a dimension is a data element that categorizes each item in a data set into non-overlapping regions.

Types of Dimension

In a data warehouse model, dimension can be of following types,

- Conformed Dimension
- Junk Dimension
- Degenerated Dimension
- Role Playing Dimension

Types of Dimension Continued

Based on how frequently the data inside a dimension changes, we can further classify dimension as

- Unchanging or static dimension (UCD)
- Slowly changing dimension (SCD)
- Rapidly changing Dimension (RCD)

Conformed Dimension

A conformed dimension is the dimension that is shared across multiple subject area. Consider 'Customer' dimension. Both marketing and sales department may use the same customer dimension table in their reports. Similarly, a 'Time' or 'Date' dimension will be shared by different subject areas. These dimensions are conformed dimension.

Degenerated Dimension

A degenerated dimension is a dimension that is derived from fact table and does not have its own dimension table.

A dimension key, such as transaction number, receipt number, Invoice number etc. does not have any more associated attributes and hence can not be designed as a dimension table.

Junk Dimension

A junk dimension is a grouping of typically low-cardinality attributes (flags, indicators etc.) so that those can be removed from other tables and can be junked into an abstract dimension table.

These junk dimension attributes might not be related. The only purpose of this table is to store all the combinations of the dimensional attributes which you could not fit into the different dimension tables otherwise. Junk dimensions are often used to implement Rapidly Changing Dimensions in data warehouse.

Role-Playing Dimension

Dimensions are often reused for multiple applications within the same database with different contextual meaning. For instance, a "Date" dimension can be used for "Date of Sale", as well as "Date of Delivery", or "Date of Hire". This is often referred to as a 'role-playing dimension'

What is Fact?

A fact is something that is quantifiable (Or measurable). Facts are typically (but not always) numerical values that can be aggregated. Facts can be:

- Non-additive
- Semi Additive
- Additive

Non-Additive Measures

Non-additive measures are those which can not be used inside any numeric aggregation function (e.g. SUM(), AVG() etc.). One example of non-additive fact is any kind of ratio or percentage. Example, 5% profit margin, revenue to asset ratio etc. A non-numerical data can also be a non-additive measure when that data is stored in fact tables, e.g. some kind of varchar flags in the fact table.

Semi Additive Measures

Semi-additive measures are those where only a subset of aggregation function can be applied. Let's say account balance. A `sum()` function on balance does not give a useful result but `max()` or `min()` balance might be useful. Consider price rate or currency rate. Sum is meaningless on rate; however, average function might be useful.

Additive Measures

Additive measures can be used with any aggregation function like `Sum()`, `Avg()` etc. Example is Sales Quantity etc.

Fact-Less-Fact

A fact table that does not contain any measure is called a fact-less-fact. This table will only contain keys from different dimension tables. This is often used to resolve a many-to-many cardinality issue.

Fact-Less-Fact Continued

Consider a school, where a single student may be taught by many teachers and a single teacher may have many students. To model this situation in dimensional model, one might introduce a fact-less-fact table joining teacher and student keys. Such a fact table will then be able to answer queries like,

- Who are the students taught by a specific teacher.
- Which teacher teaches maximum students.
- Which student has highest number of teachers.etc.

Coverage Fact

A fact-less-fact table can only answer 'optimistic' queries (positive query) but can not answer a negative query. Again consider the illustration in the above example. A fact-less-fact containing the keys of tutors and students can not answer a query like below,

- Which teacher did not teach any student?
- Which student was not taught by any teacher?

Coverage Fact Continued

Coverage fact table attempts to answer this - often by adding an extra flag column. Flag = 0 indicates a negative condition and flag = 1 indicates a positive condition. To understand this better, let's consider a class where there are 100 students and 5 teachers. So coverage fact table will ideally store $100 \times 5 = 500$ records (all combinations) and if a certain teacher is not teaching a certain student, the corresponding flag for that record will be 0.

Incident and Snapshot Facts

A fact table stores some kind of measurements. Usually these measurements are stored (or captured) against a specific time and these measurements vary with respect to time. Now it might so happen that the business might not be able to capture all of its measures always for every point in time. Then those unavailable measurements can be kept empty (Null) or can be filled up with the last available measurements. The first case is the example of incident fact and the second one is the example of snapshot fact.

Classifying Data for Successful Modeling

We have to understand the natural characteristics of data in general. Knowing these characteristics help us classify the data appropriately while doing data modeling.

What is data?

Data are values of qualitative or quantitative variables, belonging to a set of items. Simply put, it's an attribute or property or characteristics of an object. Point to note here is, data can be both qualitative (brown eye color) and quantitative (20cm long).

Types of Data

One of the fundamental aspects you must learn before attempting to do any kind of data modeling is the fact that how we model the data depends completely on the nature or type of data.

Data can be:

- Qualitative
- Quantitative

Qualitative Data

Qualitative data are also called categorical data as they represent distinct categories rather than numbers. In case of dimensional modeling, they are often termed as "dimension". Mathematical operations such as addition or subtraction do not make any sense on that data.

Example of qualitative data are, eye color, zip code, phone number etc.

Classification of Qualitative Data

We can classify qualitative data as:

- Nominal
- Ordinal

Nominal

Nominal data represents data where order of the data does not represent any meaningful information. Consider your passport number. There is no information as such if your passport number is greater or lesser than someone else's passport number. Consider Eye color of people, does not matter in which order we represent the eye colors, order does not matter.

ID, ZIP code, Phone number, eye color etc. are example of nominal class of qualitative data.

Ordinal

Order of the data is important for ordinal data. Consider height of people - tall, medium, short. Although they are qualitative but the order of the attributes does matter, in the sense that they represent some comparative information. Similarly, letter grades, scale of 1-10 etc. are examples of Ordinal data.

In the field of dimensional modeling, this kind of data are sometimes referred as non-additive facts.

Quantitative Data

Quantitative data are also called numeric data as they represent numbers. In case of dimensional data modeling approach, these data are termed as "Measure".

Example of quantitative data is, height of a person, amount of goods sold, revenue etc.

Classification of Quantitative Data

We can classify quantitative data as:

- Interval
- Ratio

Interval

Interval classification is used where there is no true zero point in the data and division operation does not make sense. Bank balance, temperature in Celsius scale, GRE score etc. are the examples of interval class data. Dividing one GRE score with another GRE score will not make any sense. In dimensional modeling this is synonymous to semi-additive facts.

Ratio

Ratio class is applied on the data that has a true "zero" and where division does make sense. Consider revenue, length of time etc. These measures are generally additive.

Step by Step Approach to Dimensional Modeling

The business objective is to create a data model that can store and report number of burgers and fries sold from a specific McDonalds outlet per day.

Step 1: Identify the Dimensions

Dimensions are the object or context. That is - dimensions are the 'things' about which something is being spoken. In the above statement, we are speaking about 3 different things - we are speaking about some "food", some specific McDonalds "store" and some specific "day". So we have 3 dimensions - "food" (e.g. burgers and fries), "store" and "day". Burgers and fries are 2 different members of "food" dimension. We will have to create separate tables for separate dimensions.

Step 2: Identify the Measures

Measures are the quantifiable subjects and these are often numeric in nature. In the above statement, the number of burgers/fries sold is a measure. Measures are not stored in the dimension tables. A separate table is created for storing measures. This table is called Fact Table.

Step 3: Identify the Attributes of Dimensions

We have decided we need 3 tables to store the information of 3 dimensions, next we need to know what are the properties or attributes of each dimension that we need to store in our table. This is important since knowing the properties let us decide what columns are required to be created in each dimension table.

Step 3 Continued

Each dimension might have number of different properties, but for a given context, not all of them are relevant for us. As an example, let's take the dimension "food". We can think of so many different attributes of food - e.g. names of the food, price of the food, total calories in the food, color of the food and so on.

Step 3 Continued

As for the given statement above, we just need to know only one attribute of the "food" dimension - i.e. name of the food (burger or fries). So the structure of our food dimension will be rather easy.

KEY	NAME
1	Burger
2	Fries

Step 3 Continued

The structure of our store dimension will be like:

KEY	NAME
1	Store 1
2	Store 2

Step 3 Continued

As for the given statement above, we just need to know only one attribute of the "food" dimension - i.e. name of the food (burger or fries). So the structure of our food dimension will be rather easy.

KEY	NAME
1	Burger
2	Fries

Step 3 Continued

The structure of our day dimension will be like:

KEY	DAY
1	01-Jan-2017
2	02-Jan-2017

Each dimension table above has a key column. Key is a not null column and it's a unique column which helps us identify each record of the table.

Step 4: Identify the Granularity of the Measures

Granularity refers to the lowest (or most granular) level of information stored in any table. If a table contains sales data for each and every day, then it has a daily granularity. If a table contains total sales data for each month, then it has monthly granularity

Step 4 Continued

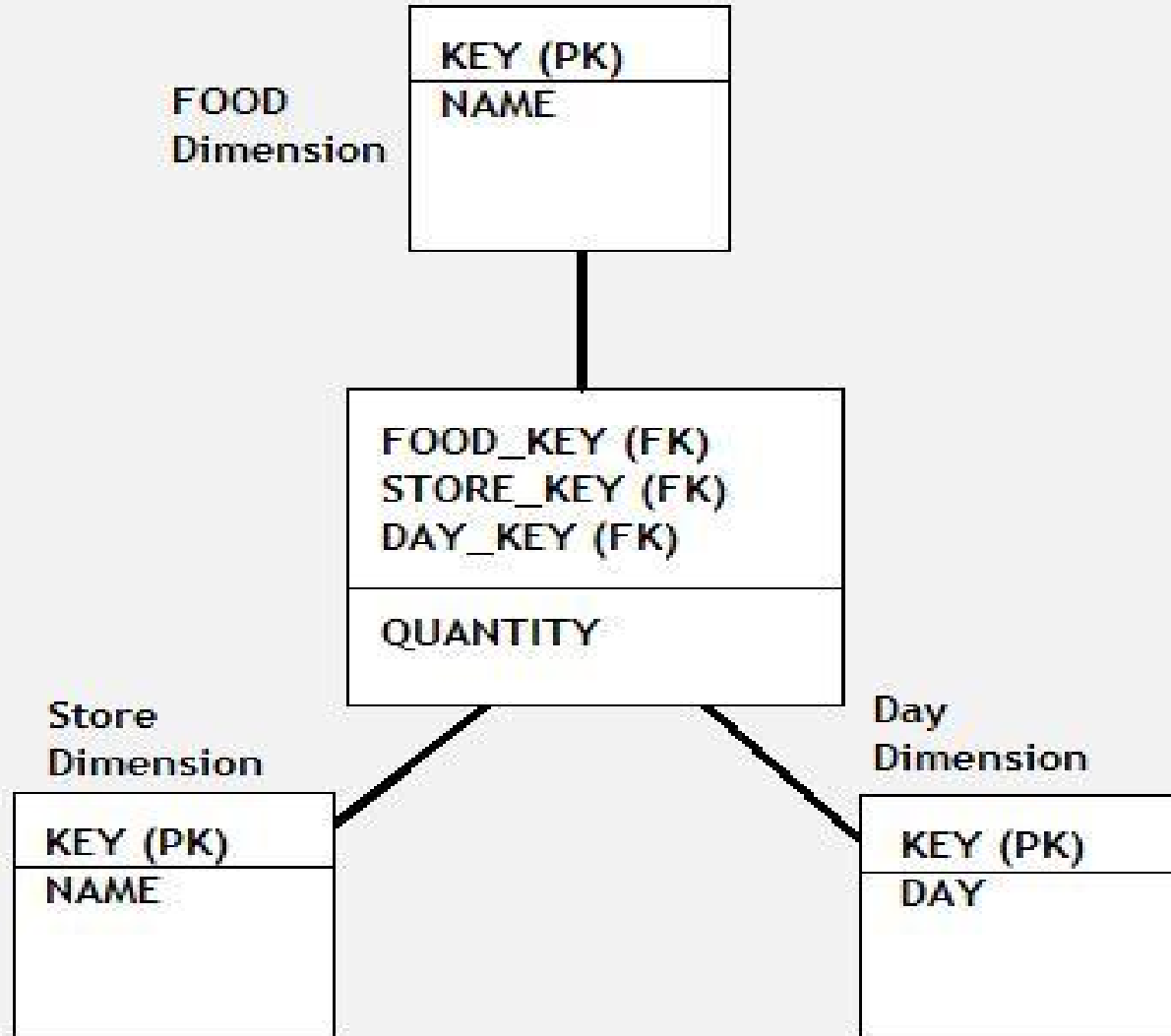
In our case, we need the information on a daily basis. But if our requirement was "To store how many burgers and fries are getting sold from a specific McDonalds outlet per month", required granularity would have changed to monthly from daily. This is important because this information help us decide what columns are required to be stored in our fact table.

Step 4 Continued

For example, since in our case the granularity is food getting sold per store per day, we will need to add key columns from food/store and Day dimensions to the Fact table like below:

Step 4 Continued

Dimensional Model of
our business case and
it is a star schema



Star Schema

Star schema is used in data warehouse models where one centralized fact table references number of dimension tables so as the keys (primary key) from all the dimension tables flow into the fact table (as foreign key) where measures are stored. This entity-relationship diagram looks like a star, hence the name.

Star schema provides a denormalized design.

Benefit of Star Schema Design

Simple and flexible. Any information can be obtained just by traversing a single join, meaning this type of schema will be ideal for faster information retrieval. All the hierarchies (or levels) of the members of a dimension are stored in the single dimension table - that means, let's say if we wish to group (veggie burger and chicken burger) in "burger" category and (french fries and twister fries) in "fries" category, you have to store that category information in the same dimension table.

Storing Hierarchy in Star Schema

KEY	NAME	TYPE
1	Chicken Burger	Burger
2	Veggie Burger	Burger
3	French Fries	Fries
4	Twister Fries	Fries

Snow-Flake Schema

Snow flake schema is just like star schema but the difference is, here one or more dimension tables are connected with other dimension table as well as with the central fact table.

We are storing the food type in one dimension ("type" table as shown below) and food in other dimension.

Snow-Flake Schema Design

Type Dimension:

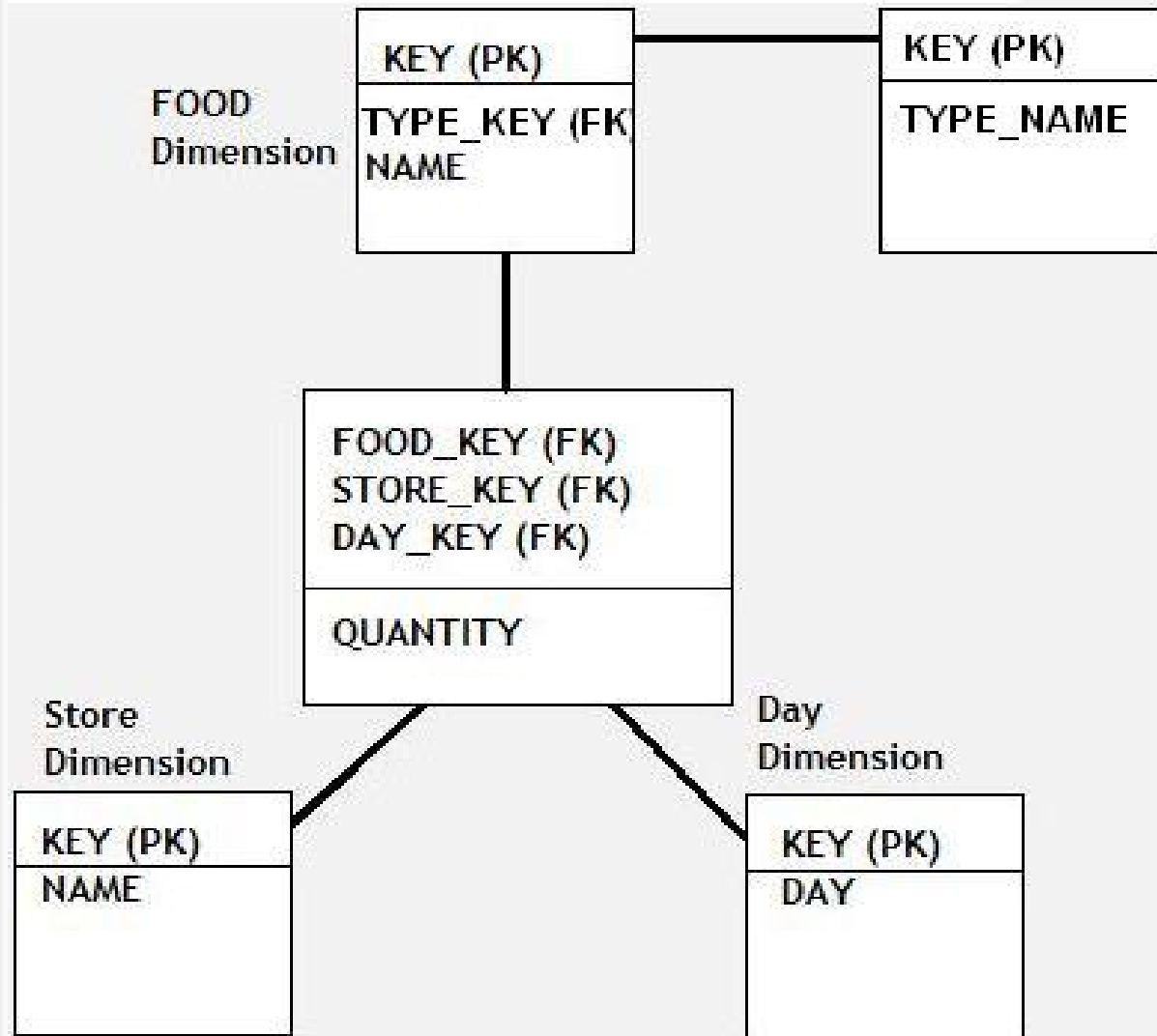
KEY	TYPE_NAME
1	Burger
2	Fries

Snow-Flake Schema Design

Food Dimension:

KEY	TYPE_KEY	NAME
1	1	Chicken Burger
2	1	Veggie Burger
3	2	French Fries
4	2	Twister Fries

Snow-Flake Schema Diagram



Step 5: History Preservation (Optional)

Let's say McDonalds decided to change the name of the food "burger" to "jumbo burger" for some promotional reason. If they do that, they would update the burger record in the dimension table and update the name to "jumbo burger". So far so good. But the problem is we will lose the old information once they change the name. This means, after they change the name if you look at the data in the model, you will not know that until now the product was called "burger" and not "jumbo burger".

Nothing Lasts Forever

Let's add a "price" column to our "Food" dimension table:

KEY	NAME	TYPE	PRICE
1	Chicken Burger	Burger	3.70
2	Veggie Burger	Burger	3.20
3	French Fries	Fries	2.00
4	Twister Fries	Fries	2.20

How Do We Manage Changes?

The table in the previous slide only tells us the price of the food as of current point in time. It does not tell us what the price was, let's say 6 months ago. If the price of Veggie Burger changes from \$3.20 to \$3.25 tomorrow, the new price will be updated in the table and then we will have no way to know what was the earlier price. So our objective is to change the above table structure in such a way so that we can store all the historical and future prices of the foods.

Types of Changing Dimensions

- Unchanging Dimension
- Slowly Changing Dimension
- Rapidly Changing Dimensions

Unchanging Dimension

There are some dimensions that do not change at all. For example, let's say you have created a dimension table called "Gender". Below are the structure and data of this dimension table: Gender

KEY	VALUE
1	FEMALE
2	MALE

Slowly Changing Dimension

These are the dimensions where one or more attributes can change slowly with respect to time. Look at the "food" dimension from our earlier example. "Price" is one such attribute which is variable in this dimension. But "price" of french fries or burgers do not change very often, may be they change once in a season. This is an example of slowly changing dimension.

Types of Slowly Changing Dimension

Slowly changing dimensions, referred as SCD henceforth, can be modeled basically in 3 different ways based on whether we want to store no history, partial history or full histories. These different types are called Type 1, Type 3 and Type 2 respectively.

SCD Type 1

We do not store any history in SCD Type 1. This is not same as "Unchanged Dimension". In case of an unchanged dimension, we assume that the values of the attributes of that dimension will not change at all. On the other hand, here in case of a SCD Type 1 dimension, we assume that the values of the attributes will change slowly, however, we are not interested to store those changes. We are only interested to store the current or latest value. So every time it changes we will update the old value with new ones.

Handling SCD Type 1 Dimension in ETL Process

From ETL design perspective SCD Type 1 dimensions are loaded using "Merge" operation which is also known as "UPSERT" as an abbreviation of "Update else Insert".

SCD Type 2

In order to design the "Food" table as SCD Type 2, we will have to add 3 more columns in that table, "Date From", "Date To" and "Latest Flag". These columns are called type 2 metadata columns.

KEY	NAME	TYPE_KEY	PRICE	DATE_FROM	DATE_TO	LATEST_FLG
1	Chicken Burger	1	3.70	01-Jan-11	31-Dec-99	Y
2	VeggieBurger	1	3.20	01-Jan-11	31-Dec-99	Y
3	French Fries	2	2.00	01-Jan-11	31-Dec-99	Y
4	Twister Fries	2	2.20	01-Jan-11	31-Dec-99	Y

SCD Type 2 Continued

Let's assume, today is 15 March 2011, and McDonald has decided to increase the price of "Veggie Burger" from \$3.20 to \$3.25. If this happens we will not straight away update the price from \$3.20 to \$3.25. Instead to store this new information (and also the old information), we will insert a new record in the "Food" dimension table which will look like below:

SCD Type 2 Continued

KEY	NAME	TYPE_KEY	PRICE	DATE_FROM	DATE_TO	LATEST_FLG
1	Chicken Burger	1	3.70	01-Jan-11	31-Dec-99	Y
2	VeggieBurger	1	3.20	01-Jan-11	14-Mar-11	N
3	French Fries	2	2.00	01-Jan-11	31-Dec-99	Y
4	Twister Fries	2	2.20	01-Jan-11	31-Dec-99	Y
5	VeggieBurger	1	3.25	15-Mar-11	31-Dec-99	Y

Surrogate Key for SCD Type 2 Dimension

Note from the above example that, each time we generate a new row in the dimension table, we also assign a new key to the record. This is the key that flows down to the fact table in a typical Star schema design. The value of this key, that is the numbers like 1, 2, 3, , 7 etc. are not coming from the source systems. Instead those numbers are just like sequential running numbers which are generated automatically at the time of inserting these records. These numbers are unique, so as to uniquely identify each record in the table, and are called "Surrogate Key" of the table.

Alternate Design of SCD Type 2: Addition of Version Number

KEY	NAME	TYPE_KEY	PRICE	DATE_FROM	DATE_TO	VERSION
1	Chicken Burger	1	3.70	01-Jan-11	19-Dec-11	1
2	VeggieBurger	1	3.20	01-Jan-11	14-Mar-11	1
3	French Fries	2	2.00	01-Jan-11	31-Dec-99	1
4	Twister Fries	2	2.20	01-Jan-11	31-Dec-99	1
5	VeggieBurger	1	3.25	15-Mar-11	19-Dec-11	2
6	Chicken Burger	1	3.80	20-Dec-11	31-Dec-99	2
7	VeggieBurger	1	3.20	20-Dec-11	31-Dec-99	3

Handling SCD Type 2 Dimension in ETL Process

Unlike SCD Type 1, Type 2 requires us to insert new records in the table as and when any attribute changes. We will need to update old record (e.g. changing the latest flag from "Y" to "N", updating the "Date To") as well as we will need to insert a new record.

Performance Considerations of SCD Type 2 Dimension

Let's say we have an "employee" dimension table which we have designed as SCD Type 2. The employee dimensions has 20 different attributes and there are 10 attributes in this table which change at least once in a year on average (e.g. employee grade, manager's name, department, salary, band, designation etc.). This means if we have 1,000 employees in our company, at the end of just one year, we are going to get 10,000 records in this dimension table (i.e. assuming on an average 10 attributes change per year - resulting into 10 different rows in the dimension table).

SCD Type 3

Type 3 design is used to store partial history. Although theoretically it is possible to use the type 3 design to store full history, that would be not possible practically. In case of type 3, we add new column to the table to store the history. Let's say, we have a table where we have 2 column initially - "Key" and "attribute".

SCD Type 3 Continued

KEY	ATTRIBUTE
1	A
2	B
3	C

SCD Type 3 Continued

If the record 1 changes its attribute from A to D, we will add one extra column to the table to store this change.

KEY	ATTRIBUTE	ATTRIBUTE_OLD
1	D	A
2	B	
3	C	

SCD Type 3 Continued

If the record again change attribute values, we will again have to add columns to store the history of the changes

KEY	ATTRIBUTE	ATTRIBUTE_OLD	ATTRIBUTE_OLD_1
1	E	D	A
2	B		
3	C		

SCD Type 3 Continued

As we can see, storing the history in terms of changing the structure of the table in this way is quite cumbersome and after the attributes are changed a few times the table will become unnecessarily big and fat and difficult to manage. But that does not mean SCD Type 3 design methodology is completely unusable. In fact, it is quite usable in a particular circumstance - where we just need to store the partial history information.

SCD Type 3 Continued

Let's think about a special circumstance where we only need to know the "current value" and "previous value" of an attribute. That is, even though the value of that attribute may change numerous times, at any time we are only concerned about its current and previous values. In such circumstances, we can design the table as type 3 and keep only 2 columns - "current value" and "previous value" like below.

SCD Type 3 Continued

KEY	CURRENT_VALUE	PREVIOUS_VALUE
1	D	A
2	B	
3	C	

SCD Type 3 Continued

Example from telecom domain, wherein a certain calculated field in the report used to depend on the latest and previous values of the customer status. That calculated attribute is called "Churn Indicator" (churn in telecom business generally means leaving a telephone connection) and the rule to populate the churn indicator was (in a very very simplified way) like below:

SCD Type 3 Continued

```
Churn Indicator  
= "Voluntary Churn"  
(if customer's current status = 'Inactive' and previous status = 'Active')  
  
= "Involuntary Churn",  
(if customer's current status = 'Inactive' and previous status = 'Suspended')
```

As we can see, in order to find out the correct value of churn indicator, we do not need to know complete history of changes of customer's status. All we need to know is the current and previous status. In this kind of partial history scenario, SCD Type 3 design is very useful

SCD Type 3 Continued

Compared to SCD Type 2, type 3 does not increase the number of records in the table thereby easing out performance concerns.

Rapidly Changing Dimensions

Let's say we have a "Subscriber" dimension where we store the details of all the subscribers to a particular pre-paid mobile service plan. We have a "status" column in the "Subscriber" dimension table which can have several different values based on the current account balance of the subscriber.

Rapidly Changing Dimensions Continued

For example, if your balance is less than \$0.1, the status becomes "No Outgoing call". If your balance is less than \$5, the status becomes "Restricted Call Service". If your balance is less than \$10, the status becomes "No Long Distance Call" and if the balance is greater than \$10 then status becomes "Full Service", etc. Every month, the status of any subscriber keeps on changing multiple times based on his or her account balance thereby making the "Subscriber" dimension one rapidly changing dimension.

Implementing Rapidly Changing Dimension

Handling rapidly changing dimensions are tricky due to various performance implications. SCD Type 2 dimensions is not a very good fit for rapidly changing scenarios as it increases the number of rows in the table. Let's explore a method to tackle rapidly changing dimension.

Junk Dimension

The method that we are going to consider here assumes the fact that, not all the attributes of a dimension table are rapidly changing in nature. There might be a few attributes which are changing quite often and some other attributes which seldom change. If we can separate the fast changing attributes from the slowly changing ones and move them in some other table while maintaining the slowly changing attributes in the same table, we can get rid of the issue of bulking up the dimension table.

Junk Dimension Continued

Let's say CUSTOMER dimension has following columns:

- CUSTOMER_KEY
- CUSTOMER_NAME
- CUSTOMER_GENDER
- CUSTOMER_MARITAL_STATUS
- CUSTOMER_TIER
- CUSTOMER_STATUS

Junk Dimension Continued

While attributes like name, gender, marital status etc. do not change at all or rarely change, let's assume customer tier and status change every month based on customer's buying pattern. If we decide to keep status and tier in the same SCD Type 2 Customer dimension table, we could risk filling-up the table too much too soon. Instead, we can pull out those two attributes in yet another table, which some people refer as JUNK DIMENSION. Here is how our junk dimension will look like. In this case, it will have 3 columns as shown below.

Junk Dimension Continued

Junk Dimension Structure

- SEGMENTATION_KEY
- TIER
- STATUS

The column SEGMENTATION_KEY is a surrogate key. This acts as the primary key of the table.

Junk Dimension Continued

Since we have removed status and tier from our main dimension table, the dimension table now looks like this:

- CUSTOMER_KEY
- CUSTOMER_NAME
- CUSTOMER_GENDER
- CUSTOMER_MARITAL_STATUS

Junk Dimension Continued

We must create a linkage between the above customer dimension to our newly created JUNK dimension. By creating one more mini table in between the original customer dimension and the junk dimension. This mini dimension table acts as a bridge between them. We also put "start date" and "end date" columns in this mini table so that we can track the history. Here is how our new mini table looks like:

Junk Dimension Continued

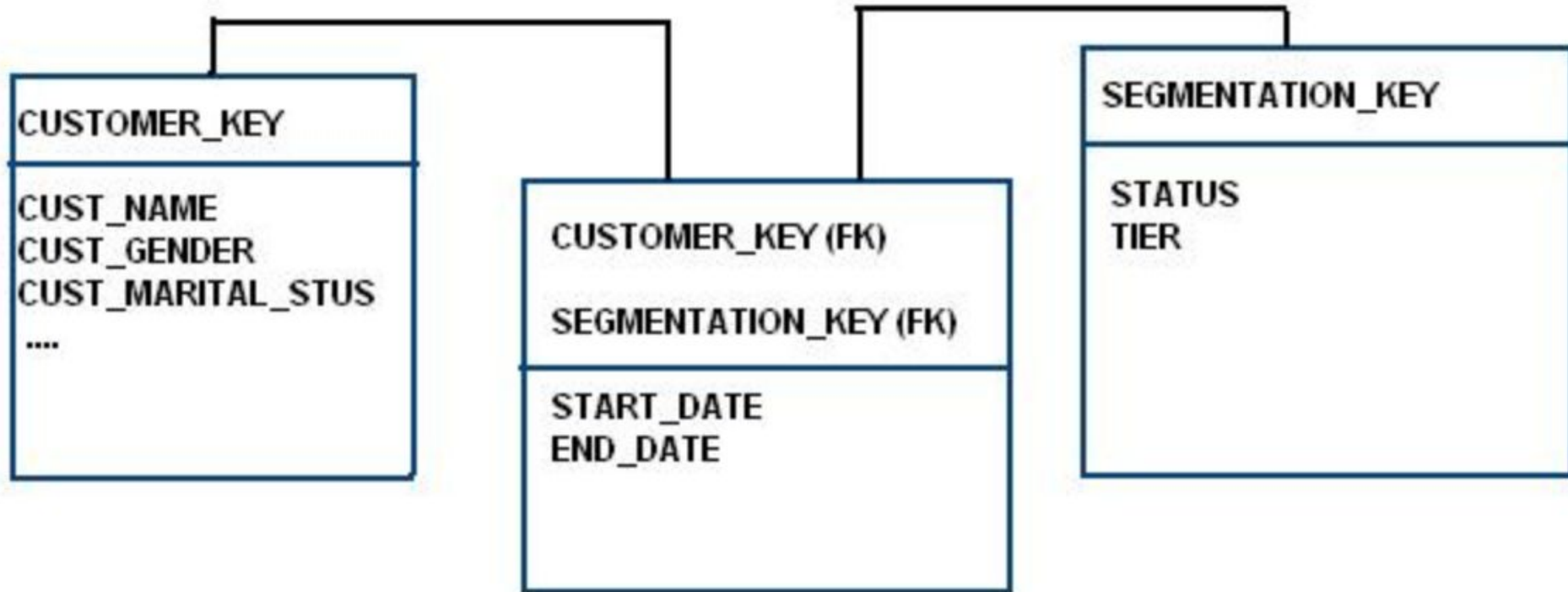
Mini Dimension Structure

- CUSTOMER_KEY
- SEGMENTATION_KEY
- START_DATE
- END_DATE

Junk Dimension Continued

This table does not require any surrogate key. However, one may include one "CURRENT FLAG" column in the table if required. Now the whole model looks like this:

Junk Dimension Continued



Maintaining the Junk Dimension













If number of attributes and the number of possible distinct values per attributes are not very large in the Junk dimension, we can pre-populate the junk dimension once and for all. In our example, let's say possible values of status are only "Active" and "Inactive" and possible values of Tier are only "Platinum", "Gold" and "Silver". That means there can be only $3 \times 2 = 6$ distinct combinations of records in this table. We can pre-populate the table with these 6 records from segmentation key = 1 to 6 and assign one key to each customer based on the customers status and tier values.

How does this Junk dimension help?

Since the connection between the segmentation key and customer key is maintained in the mini dimension table, frequent changes in tier and status do not change the number of records in the dimension table. Whenever a customer's status or tier attribute changes, a new row is added in the mini dimension (with `START_DATE` = date of change of status) signifying the current relation between the customer and the segmentation.

DB-Engines Ranking

337 systems in ranking, November 2017

Rank			DBMS	Database Model	Score		
Nov 2017	Oct 2017	Nov 2016			Nov 2017	Oct 2017	Nov 2016
1.	1.	1.	Oracle 	Relational DBMS	1360.05	+11.25	-52.96
2.	2.	2.	MySQL 	Relational DBMS	1322.03	+23.20	-51.53
3.	3.	3.	Microsoft SQL Server 	Relational DBMS	1215.08	+4.76	+1.27
4.	4.	4.	PostgreSQL 	Relational DBMS	379.92	+6.64	+54.10
5.	5.	5.	MongoDB 	Document store	330.47	+1.07	+5.00
6.	6.	6.	DB2 	Relational DBMS	194.06	-0.53	+12.61
7.	7.	 8.	Microsoft Access	Relational DBMS	133.31	+3.86	+7.34
8.	8.	 7.	Cassandra 	Wide column store	124.21	-0.58	-9.76
9.	9.	9.	Redis 	Key-value store	121.18	-0.87	+5.64
10.	10.	 11.	Elasticsearch 	Search engine	119.41	-0.82	+16.84

References

<https://dwbi.org/data-modelling/dw-design/8-what-is-a-data-warehouse-guide.html>

<https://dwbi.org/data-modelling/dimensional-model/1-dimensional-modeling-guide.html>

<https://dwbi.org/data-modelling/dimensional-model/16-classifying-data-for-successful-modeling.html>

<https://dwbi.org/data-modelling/dimensional-model/17-dimensional-modeling-schema.html>

<https://dwbi.org/data-modelling/dimensional-model/19-modeling-for-various-slowly-changing-dimension.html>

<https://dwbi.org/data-modelling/dimensional-model/20-implementing-rapidly-changing-dimension.html>

<https://dwbi.org/>

<https://db-engines.com/en/ranking>

Q&A

Thank you :)

Contact:

Md. Shafiuzzaman Rajib

Skype: md.shafiuzzaman.rajib

Email: rajibsust@gmail.com

LinkedIn: <https://www.linkedin.com/in/md-shafiuzzaman-rajib-5bbb626/>