DATA WAREHOUSING
Changing Dimensions

Floriano Alves

University of Minho, Braga, Portugal
pg13213@alunos.uminho.pt

Abstract. Dimension tables have a very important mission in any Data Warehouse. Their main role is to support Fact Table records with descriptions and other information about the entities involved on these records. By crossing data with other dimension tables and the fact table we can get very important results that managers and decision makers can support on, by exploring all dimensional width of the data contained in the Data Warehouse or simply analyzing it through the attribute structure hierarchy where time has the main role. Ideally data in the dimensional tables should be static, not changing through time making information consistent and predictable but in some application scenarios this is not true. Some times data in the dimensional tables changes occasionally (slowly) and some times it changes frequently (rapidly). These changes in the dimension records can lead to irregular situations if not treated well.

1 Context of the Problem

"The Data Warehouse systems aim to provide the resources and technologies needed to centralize information on the business of a company in one place." [1] This information is often distributed through various places, such as a department or a dependency store. By centralizing data so that those responsible in the company can support on them to manage business processes, then making decisions can be taken easier and confidently by the executives.

The information that a Data Warehouse contains is targeted to specific issues and its data is used for the analysis of these issues and not for normal processing as a common database system The Data Warehouse systems provide information that does not exist naturally in sources, such as historical information and data aggregations. Before building the Data Warehouse we need to know and decide what data will be stored in its tables and at each time we have to know any kind of change that occurred in the data sources and how they affect or not the data that was already in the Data Warehouse system.

We can define a Data Mart as a "subset of an organizational Data Warehouse usually targeted for a specific purpose or a matter of utmost importance for the organization and can be distributed to support the needs of the business" [1]. Data Marts are analytical data stores designed to focus on specific business functions to a specific community within an organization. Data Marts are often derived from subsets of data in a Data Warehouse, though in the bottom-up design methodology the Data Ware-

house is created from the union of organizational Data Marts. They offer users the information they analyze more frequently but with smaller data volumes. The costs of implementation are often better.

The star schema, which can also be referred to as star join schema (see Fig. 1), is the simplest style in Data Warehouse design. This schema consists of some fact tables, probably only one therefore its name, referencing any number of dimension tables. The star schema is considered to be an important particular case of the snowflake schema (see Fig. 2).

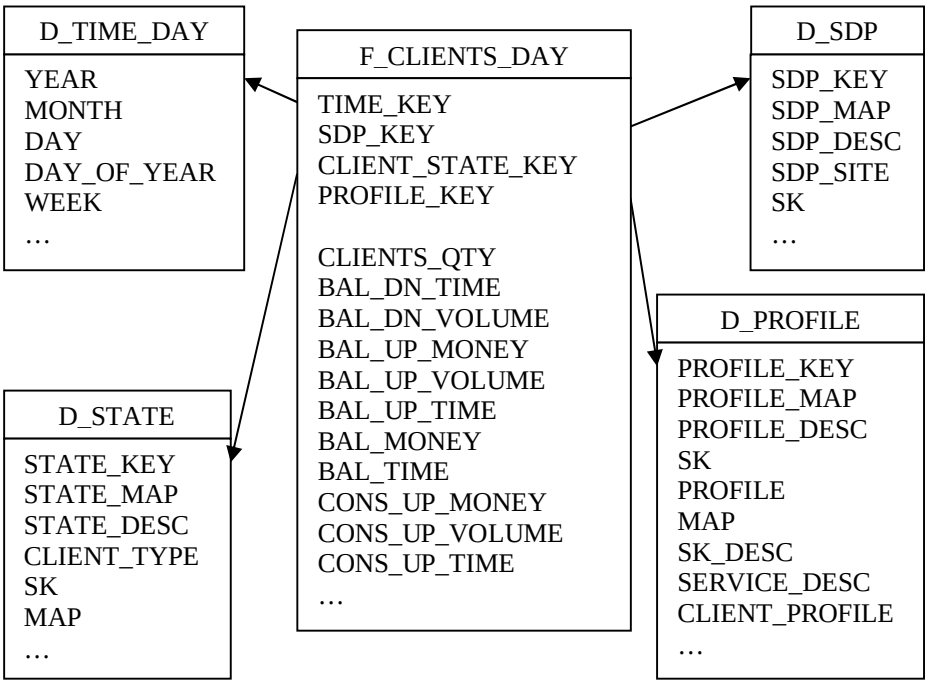

Fig. 1. Star Schema[1]

A snowflake schema is a logical arrangement of tables in a relational database such that the entity-relationship diagram reminds the shape of a snowflake. Closely related to the star schema, the snowflake schema is represented by centralized fact tables that are connected to multiple dimensions. In the snowflake schema however, the dimensions are normalized into several related tables, while the dimensions of the star schema are de-normalized, i.e. each dimension is represented by a single table. When the dimensions of a snowflake schema are elaborate with various levels of relationship and where child tables have multiple parent tables, a complex snowflake schema begins to take shape. The "snowflaking" effect only affects the dimension tables and not the fact tables.

[1] Source: PT Inovação, SA – Business Intelligence division.

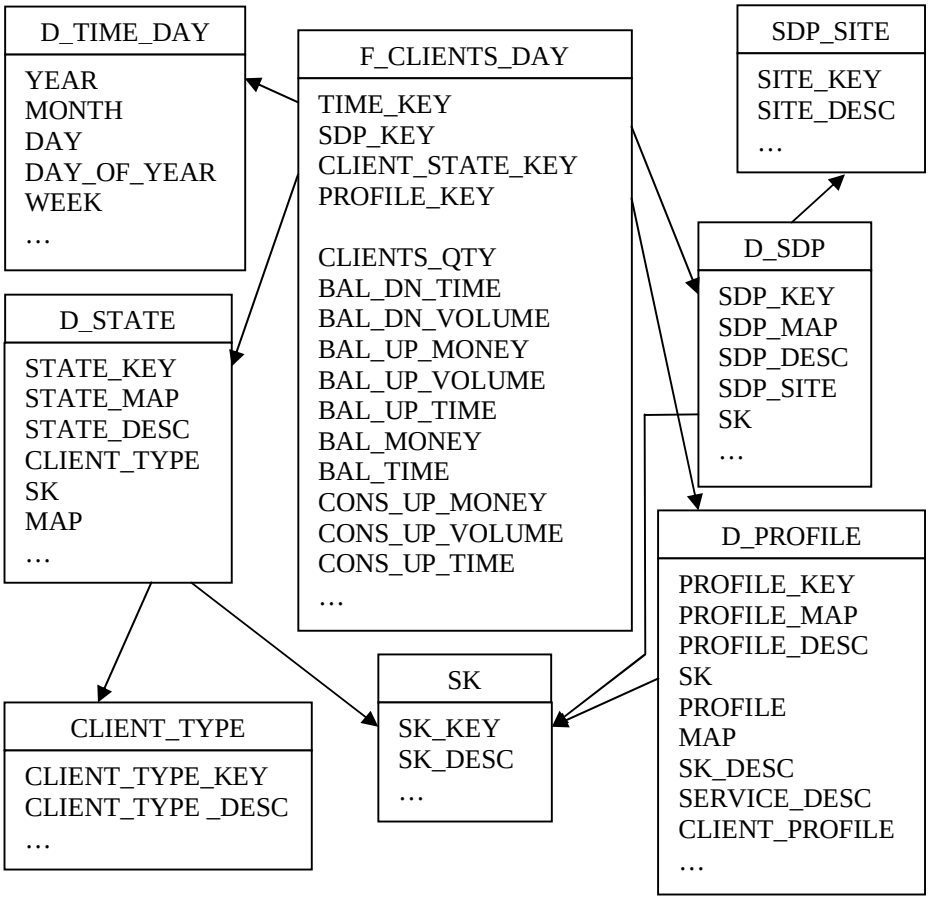

Fig. 2. Snowflake Schema[1]

2 The Problem

The problem of handling with data changes in dimensions represented as a star schema or a snowflake schema was addressed by Kimbal et al. [1]. These authors define Changing Dimensions as dimensions that have data that changes through time in the other hand dimensions that don't change are called Static Dimensions. In some cases dimensions change slowly and asynchronously, in other cases they change rapidly. A significant problem of dimensional processing is decide what to do with a dimension description or other attribute of a client or product that is not compatible with what is already stored in the Data Warehouse. Another example is if we have a dimension that tracks client plafonds in a telecommunications company and a particular plafond is set to be a money plafond and it changes to become a volume plafond,

[1] Source: PT Inovação, SA – Business Intelligence division.

then we have a problem because we have a unit conflict, money cannot be matched with volume. For instance euros can't be compared with bytes, events or time and the other way around and with each other. We shouldn't perform mathematical operations between plafonds of different units, even being possible to do so, it would give misleading information. So how do we solve this?

3 Dealing with the Problem

If the revised description or plafond unit is taken as a legitimate and reliable update to prior information, then we must use the techniques of **"Slowly Changing Dimensions"** [1]. If the revised description is merely an informal entry, then changed fields are ignored. Dealing with these issues involves Slowly Changing Dimensions management methodologies referred to as **Type 1, 2, 3, 4, and 6**. Type 6 Slowly Changing Dimensions are also sometimes called **Hybrid Slowly Changing Dimensions**. The most common Slowly Changing Dimensions techniques are Types 1, 2, and 3.

We must recognize what has changed in the input data and generate the proper dimension **surrogate key**. All data warehouse keys should be a surrogate key because the data warehouse architect must have the flexibility to react to mutant descriptions and abnormal conditions in the primary data. If the real physical join key between a dimension table and the data table is a direct derivation of a production key, sooner or later the data warehouse architect will face an impossible situation. Production keys can be reused or re-formatted. Sometimes the dimension value can be "unknown". The most common need for a generalized key is when the data warehouse attempts to trace a revised dimension description and the production key has not changed.

The first step in preparing a dimension record is deciding whether we already have the record or not. The primary data will generally have a production key value. This production key value must be crossed with the same field in the "current" dimension record. Note that in the Data Warehouse dimension, this is only a common dimension attribute. The current dimension record can be found quickly with an updatable flag in the dimension record that identifies whether the record is or is not the current one.

This combination of the input production key with its partnership in the current dimension record can be achieved with a sequential simple passage through the primary data and the dimension table data, both classified by the production key. If the input dimension data information matches to what we already have, then no further action is required. If anything in the dimension input information as changed, then we should implement a change in the dimension of type 1, type 2, type 3 or special derivation cases of these such as type 4 and 6.

- **Type 1**: “Overwrite the dimension table record with the new values from the raw data” [1]. In this case we loose history. It can be used to correct an error for example.
- **Type 2**: “Create a new additional dimension record using a new value of the surrogate key” [1]. We then update this record with fields that have changed and add any other fields that are needed. By doing so we can historically track a true physical change that has happened to the dimension entity. In this case we save history,

but this is also the most laborious technique for handling occasional changes in a dimension record.

– **Type 3**: “Create an "old" field in the dimension record to store the immediate previous attribute value” [1]. This is used when a change is simple or is just an attempt to change the value, or when we want to track history with the old value of the attribute, and the new one. We can act as if the change had not occurred, which allows us to choose which attribute to use.

4 How to Proceed

Regardless of the type of change that we make in processing the input dimension data, our choice will always result in a single step of sequential processing.

Nevertheless there is always one valid choice that we can take regarding Changing Dimensions that is: even if the values in the record change they are never updated under any circumstances. To that we can call Slowly Changing Dimensions **“Type 0”** technique.

4.1 Rapidly Changing Dimensions

In this scenario of Changing Dimensions in a Data Warehouse we can identify three cases: “Rapidly Changing Small Dimensions, Rapidly Changing Large Dimensions and the worst case scenario, Rapidly Changing Monster Dimensions”. [1]

Dealing with the first case is standard, we can use Slow Changing Dimensions Type 2 technique, we probably want to track all the versions of a product or phone plafond that changes "many" times. After all what is slow and fast? Changes happening every day or every month or some times a year? Although we call it Slow Changing Dimensions they apply for handling Rapidly Changing Dimensions as well.

If the dimension tends to be large, we must not create more records to deal with the Slowly Changing Dimension problem. Type 2 approaches to address a Rapidly Changing Dimension should not be used.

The worst case is the Rapidly Changing of Monster Dimensions. In this scenario, we cannot use the proper techniques for small and medium sized, due to the size of the table. Taking for example a dimension table for Client Plafond, the solution to this problem is to separate the most significant attributes of the table in their own dimension table. Now we can track the changing nature of the descriptions of the Plafond descriptions. But we must also make some compromises, all the continues variable facts must be converted into ranges of values such as initial balance, max balance and rating unit will be converted in the initial balance range, max balance range and unit rating range. We force the attributes of this new Plafond Details dimension to have a relatively small number of discrete values and then we build the dimension table with all possible discrete attributes combinations. Combining the Plafond table with the Plafond Details table in a record on a fact table, we describe a relation for these two entities and we can also track historical values, simply changing the Plafond Details key for a Plafond on a new record in a fact table.

A reasonable predetermined number of possible values for the Plafond Details dimension table are about 100,000 records. In that way we can have 5 attributes and 10 possible combinations. If we need more than 5 attributes the best option is to create another dimension table to support these attributes creating a new foreign key in the fact table.

4.2 Type 1 Slowly Changing Dimensions

The methodology of the Type 1 rewrites the old data with new data and therefore does not store historical information. This method is most appropriate to address certain types of errors such as spelling errors, for instance. Assuming that we will not ever need to know how they used to be written incorrectly in the past. Another example would be the dimension table that holds information about Client Plafond on a telecommunications company (see table 1).

Now imagine that a specified plafond changes its description from "Pay As You Go" to "Sooner Or Later You Pay". The updated table would simply replace this record (see table 2).

Plafond_Key	**Plafond_Name**	**Plafond_Unit**
1	Pay As You Go	Money

Table 1. Dimension Table Before Type 1[1]

Plafond_Key	**Plafond_Name**	**Plafond_Unit**
1	Sooner or Later You Pay	Money

Table 2. Dimension Table After Type 1[2]

The obvious disadvantage of this method of managing Slowly Changing Dimensions is that there is no historical record of the data stored in the Data Warehouse. We can not say if a plafond was used with another name in the past. The advantage of using this method is that these are very easy to maintain.

[1] Source: PT Inovação, SA – Business Intelligence division.
[2] Source: PT Inovação, SA – Business Intelligence division.

4.3 Type 2 Slowly Changing Dimensions

The Type 2 method records historical data, because it creates several entries in the dimension tables with different keys. In type 2, we keep all the history of all changes made to a dimension table field, when we insert a new record each time a change is made. First we try to get the previous version of the dimension record. If it exists in the table we copy it, creating a new entry in the dimension table with a new surrogate key. If there is an earlier version of the dimension record, then we must create a completely new one. Then we update this record with the fields that were changed, and any other ones required. For example, we may receive a record of a plafond, showing that its unit has changed. If this is a real change rather than a correction, we should use this type. The new key is substituted for example by the next sequential integer number that represents the Max (surrogate key) + 1 for the dimension in question. To expedite the processing of the replacement of the replacement key value, we can store it explicitly as metadata instead of calculating it each time it is needed.

In the same example, if the plafond moves to the volume unit, the table would look like table 3. Another method which can be used for versioning is to add columns to the dimension tables with dates of entry and update of the records (see table 4.).

Plafond_Key	Plafond_Type	Plafond_Name	Plafond_Unit	Version
1	3	Pay As You Go	Money	0
2	3	Pay As You Go	Volume	1

Table 3. Dimension Table of Type 2 - Version 1[1]

Plafond_ Key	Plafond_ Type	Plafond_ Name	Plafond_Unit	Entry_Date	Update_Date
1	3	Pay As You Go	Money	20-Jul-2003	21-Nov-2005
2	3	Pay As You Go	Volume	21-Nov-2005	

Table 4. Dimension Table of Type 2 - Version 2[2]

[1] Source: PT Inovação, SA – Business Intelligence division.
[2] Source: PT Inovação, SA – Business Intelligence division.

If the Update_Date field is null it means that this is the current version of the record. In some cases if necessary, we use a pseudo infinite end date (e.g. 31/12/9999), so the field can be included in an index.

The transactions that reference the surrogate key (Plafond_Key) are then necessarily linked to the time slices defined by each row of the table with the Slowly Changing Dimension of Type 2. If there are posterior changes made to the contents of the dimension, or whether a new set of attributes are added to the dimension (such as a column "Plafond Way" with possible ascending or descending values), which have different effective dates from those that are already defined, then this may result in the need to make changes to existing operations, to reflect the new situation. This can be an expensive database operation, so the Type 2 technique of handling Slowly Changing Dimensions is not a good choice if the dimensional model or data are subject to change.

4.4 Type 3 Slowly Changing Dimensions

The Slowly Changing Dimension treatment technique of Type 3 monitors changes using separate columns. Unlike the technique of type 2 that could keep unlimited preservation of history, the technique of Type 3 has the preservation of historical limited, since the history is set to the number of columns that are designated for the storage of historical data.

In this case, we believe that an attribute will have controlled changes which allow the user refers to one of the attributes explicitly, the old value or the new one. For example, if a plafond is set for a new unit recently identified and if we will only have two units available, then it is possible to register the unit change in the old or the new designated one. The plafond of the old unit is described as an "alternate reality".

The structure of the dimension tables for types 1 and 2 is very similar in the case of Type 3 we have to add more columns in the tables (see table 5).

Plafond_Key	**Plafond_Name**	**Original_ Plafond_Unit**	**Update_Date**	**Original_ Plafond_Unit**
1	Pay As You Go	Money	22-Jan-2006	Volume

Table 5. Dimension Table of Type 3[1]

Note that this register cannot track all historical changes, such as when a unit of a plafond has changed twice to different values. This technique is generally defined by the business after the main ETL process has been implemented.

[1] Source: PT Inovação, SA – Business Intelligence division.

5 How to Proceed - Advanced

Now that we know the three basic techniques to handle Slowly Changing Dimensions, let's have a look on two more advanced ones that are variations of Types 1, 2 and 3.

5.1 Type 4 Slowly Changing Dimensions

The technique of Type 4 is usually associated with the use of "historical tables', which maintains a current data table and an additional table that's used to keep track of some or all changes in the dimension table. The granularity of the history table is a record per rating per plafond, while the granularity of the primary dimension is a record per plafond. The number of rows in the dimension that keeps the history may be in the order of millions, but the number of rows of the current data dimension is a fraction of that. We can get a relation between the history table and the primary table in the fact table. When an event is represented by a row in the fact table, then this row has a foreign key to the primary dimension and another to the record in effect at the time of the event registered in the history table.

Following the example above, the original table might be called Plafond (see table 6) and the history table might be called Plafond_History (see table 7).

Plafond_Key	Plafond_Name	Plafond_Unit
1	Pay As You Go	Volume

Table 6. Plafond Dimension Table (Type 4)[1]

Plafond_Key	Plafond_Name	Plafond_Unit	Create_Date
1	Pay As You Go	Money	22-Jan-2006

Table 7. Plafond_History Dimension Table (Type 4)[2]

[1] Source: PT Inovação, SA – Business Intelligence division.
[2] Source: PT Inovação, SA – Business Intelligence division.

5.2 Type 6/Hybrid Slowly Changing Dimensions

The Type 6 is a technique that combines the approaches of types 1, 2 and 3. We can see a certain analogy to the values of the basic techniques by adding them together and realizing that the result is 6. It is not used often because it has the potential to complicate access to the end user, but has some advantages over the other technical approaches, especially when other techniques are employed to soften complexity further downstream.

The approach is to use the technique of Slowly Changing Dimensions of type 1, but adding one more pair of columns of date type to indicate the range of dates when a particular record in the dimension table applies.

This approach has the following advantages:

- The user may choose to query using the current values of the dimensional table, restricting the rows in the table and using a filter to select only the current values.
- Another way is to use the "as at the time of the event" values, using one of the date fields on the transaction as a constraint on the dimension table.
- If there is a series of date columns on the transaction (e.g. Access Date, Generation Date, Dump Date), then an user can choose which date to analyze the data from the table of facts. This is not possible using other approaches.

Table 8 shows how the Plafond table would look using Type 6 Slowly Changing Dimensions.

Row_ Key	Plafond_ Key	Plafond_ Type	Plafond_ Name	Plafond_ Unit	Start_ Date	End_ Date	Current_ Indicator
1	1	3	Pay As You Go	Money	22-Jan-2006	15-Jan-2008	N
2	1	3	Pay As You Go	Volume	15-Jan-2008	1-Jan-2199	Y

Table 8. Dimension Table of Type 6[1]

Alternative implementations of Type 6 may include a null end date. Other approaches include the use of a Revision Number, instead of a Row Key.

Every transaction should have the plafond key as a foreign key (even if this does not join to a single column) in combination with a filter by date. The join between the records should not be to the primary key of the table (Row Key). The transaction could also have one or more Row Keys (to allow use of type 2 Slowly Changing Dimensions in addition to Type 6), depending on which date columns the user wants to

[1] Source: PT Inovação, SA – Business Intelligence division.

be analyzed. But remember the problems with this approach described in the section of type 2 Slowly Changing Dimensions. The columns Current_Indicator and Row_Key are optional, they are not necessary for analysis, but may simplify the consultation and management respectively. Ideally one fact should have a single date of event representing the date the transaction occurred. If there are several dates, the best option is to consider splitting the fact table out, in the most granular events possible.

5.3 Type 6/Hybrid Slowly Changing Dimensions - Alternative Implementation

One of the problems of the application of the above technique is that a relationship between the dimension table and fact table of type many-to-many is never resolved in the data model. This many-to-many relationship cannot be solved from the logical and physical data model point of view. The way it was designed, it will be resolved only at runtime, at a report level, when the end user completes the parameter "As at Date" The consequence is that no referential integrity can be enforced between the dimension table and the fact table at a level of a Relational Database Management System (RDBMS). A variant of this technique to the Type 6 of Slowly Changing Dimensions exists and it implements all the advantages of Type 6, without this inconvenience.

This variant is based on primary key of the dimension table made of a surrogate key and a version number (not a start date) (see table 9). The current version number of the record will be always 0. Before updating a dimension record, the version 0 is copied to a new record (version n + 1) and version 0 may be updated.

Version_ Number	Plafond_ Key	Plafond_ Type	Plafond_ Name	Plafond_ Unit	Start_Date	End_Date
1	1	3	Pay As You Go	Money	22-Jan-2006	15-Jan-2008
0	1	3	Pay As You Go	Volume	15-Jan-2008	

Table 9. Dimension Table of Type 6 - Alternative[1]

In the fact table (see table 10), when a fact is added, it may be so by using the current dimension record. Consequently, all records of the fact table will have a version number equal to 0.

[1] Source: PT Inovação, SA – Business Intelligence division.

Transaction_ Key	Plafond_ Key	Plafond_ Ver_ Numb	Billed	Service	Generation_ Date	Access_ Date
1	1	0	2	SMS	22-Jan-2007	16-Nov-2006
2	1	0	1	MMS	15-Jan-2008	5-Jan-2008

Table 10. Fact table for Type 6 - Alternative[1]

The advantage of this implementation of handling Type 6 Slowly Changing Dimensions is that the relationship of many-to-many between the dimension table and the fact table is resolved and the priority is to the "AS IS" version, where the version number is equal to 0.

Another advantage is that the RDBMS can guarantee referential integrity between the fact table and dimension table. Finally, when a using a design tool for the physical model, the fact tables are linked to the dimension tables. The surrogate key and the version number are a composite primary key, and can be stored in the fact table, although the version number is always 0.

In the implementation of the technique using the start date in the primary key, all tables in the physical data model will appear as isolate tables, and no integrity constraint can be enforced by the RDBMS, even though this is not always desirable.

6 ETL - Extract, Transform and Load

Different types of techniques that were previously presented can be applied to different columns in a table. For example, we can apply the Type 1 to the Plafond Name column and to the column of the same table Plafond Unit Type 2. "The technique of dealing with Slowly Changing Dimensions most useful and difficult to manage is the type 2" [2] and its derivatives, because we must create a new record in the dimension and a new surrogate key each time we find a dimension record that changed. The only attributes that differ in the new record from the original are the surrogate key and any other fields that triggered the change of name or value in the dimension table.

When Transforming data from the source systems "we must identify changed values and create surrogate keys for Slowly Changing Dimensions" [2], this is a tricky process but not impossible to apply.

"We must design and implement a process for Loading data into Slowly Changing Dimensions depending on wich type of policy we are using." [2]

[1] Source: PT Inovação, SA – Business Intelligence division.

The lookup logic and allocation of keys to deal with a changed dimension record during the extraction process is described in the following two algorithms In the first one, an upstream process has identified the record has changed and then all we need do is generate a new surrogate key from the sequence and update the surrogate keys lookup table.

In the second one, we have the current version of the entire dimension table and we must determine which records were changed. Once we have the changed records, the logic of processing the surrogate key is the same as the first scenario.

```
-- The lookup and key assignment logic for handling a
changed dimension record when we know that the input
represents only changed records.

Begin

   Select * from Known Changed Dimension Records Table;

   Select Max Plafond Key;

   Select * from Most Recent Key Map Table;

   Update Key Map;

   Update Most Recent Key Map Table;

   Add Surrogate Key to Load Record;

   Insert * into Dimension Records with new surrogate
key;

   Load to DBMS Dimension table;

End;


-- The logic for determining if a dimension record has
been changed.

Begin

   Select * from Today’s complete set of Dimension Re-
cords Table;

   Select * from Yesterday’s complete set of Dimension
Records Table;

   Diff AND Compare AND Policy;

   Insert * into Changed Dimension Records Table;

   Input into previous process;

End;
```

7 Slowly Changing Dimensions Processing Issues

The production key has a relation of 1-to-many to the surrogate key for the type 2 technique Slowly Changing Dimensions and its derivations and this type saves the change history for the dimension, on the other hand, the production key has a 1-to-1 in relationship with the surrogate key for static dimensions and Slowly Changing Dimensions of type 1 and type 3. The sources don't notify their changes and don't register by time/date their own updates. For small tables, we can use a brute force approach of comparing each field received with all the fields in DW to see if anything has changed. For larger tables, "we can use the CRC, or cyclic-redundancy checksum, a number about 20 digits" [6], like this:

- Treat the entire incoming record as a string.
- Compute the CRC value of the string (some numeric value).
- Compare the CRC value of today's record with the CRC value of yesterdays record.
- If the CRC values match, today's record is identical to yesterdays record.
- If the CRC values do not match, do a field by field comparison to see what has changed.
- Depending on whether the changed field is a type 1, type 2 or type 3 change, do the necessary updates.

To identify deletions:

- Read the source log file, if one exists.
- Determine that the CRC number of a record in the DW does not match some incoming record.
- Use the MINUS set operator to compare the DW dimension with the incoming dimension. This usually works if the ETL staging table with today's dimension and the DW are part of the same database.
- Delete the record if this is valid for the business. In most cases we don't want to delete anything from the DW.

The rewriting of values used by the type 1 results in an SQL UPDATE, when the value changes. If a column is of type 1, the ETL subsystem should add the dimension record if it is a new value or update the attribute in the dimension otherwise. We also need to update any staging tables, so that any subsequent loading of the DW from the staging tables will preserve the update, this update does not affect the surrogate key, but it affects materialized aggregates that were built on the value that has changed. We must be careful with ETL tools such as "Update else Insert", which are tempting to use, but inefficient, some developers use "UPDATE else INSERT" for rapidly changing dimensions and "INSERT else UPDATE" for very slow changing dimensions. The best approach is to separate INSERTS from UPDATES and load the DW separately for the updates and the inserts. There is no need to invoke a bulk loader for small tables, we can just run the SQL updates and the performance impact is unperceptive, even with the DW logging SQL statements. For larger tables, it is better a loader, because SQL updates will result in massive database logging. We should turn the logger off before updating the DW with SQL Updates and separate SQL Inserts or

use a bulk loader. We must first prepare a new dimension in a staging file, then drop the old dimension table and finally we load the new dimension table using the bulk loader.

In type 2 of Slowly Changing Dimensions the production key does not change, but the attribute Plafond_Unit changes. We can constraint our search for Plafond_Unit or Plafond_Name. Now it is time to face the greatest burden, the load of the facts and speed is essential. Natural identifications must be stripped off the fact records and replaced with the correct surrogate keys as soon as possible. Small lookup tables for each dimension which reflects natural keys to surrogate keys are constructed from the dimension tables in the warehouse RDBMS. By doing so, the lookup tables will include the surrogate keys that correspond to new facts and will support fast hash lookups. However, this simple and efficient lookup technique is not enough if we're loading a significant range of historical facts in this first load. The lookup table would have to contain the historical dimension surrogate keys for each plafond detail change probably over a long period of time. The simple hash lookup comparable to SQL:

```
Select plafond_key from plafond_lookup PL
where PL.plafond_ID = factfile.plafond_id
```

must be replaced with:

```
Select plafond_key from plafond_lookup PL
where plafond_ID = factfile.plafond_id
and factfile.transaction_date between PL.effective_date
and PL.end_date
```

The “between date logic” in this SQL statement will slow the process of loading hundreds of millions of fact rows with ten such dimensions to a crawl. This is also true for each DML operation or Query operation in Slowly Changing Dimensions of Type 2 and its derivations. Although we have to do some sorts of compromises, the answer to this problem has an attractive simplicity. We simply do not build the complex bulk job, only the incremental building job! This technique provides a better and truer version of the past than simply link all historical events to the current dimension records.

Maintaining type 3 Slowly Changing Dimensions carries much overhead and is not used very frequently. After we change the dimension with type 3 we still have a single record in the data warehouse.

Before using the hybrid sophisticated techniques to support change tracking, we cannot forget to maintain the balance between flexibility and complexity. User's question and answer sets will vary depending on the size attributes are used for constraining or grouping.

One of the biggest problems with using these techniques for Slowly Changing Dimensions has always been performance. The Slowly Changing Dimension component actually slows the process, but can be improved very easily and quickly. We can obtain a significant improvement in performance of the dimension processing packages if we implement some optimizations in our data flows such as lookup tables and other metadata tables or SQL statement splitting.

Even if disk space and memory are cheap we have to consider degradation of space and available memory. Perhaps if an attribute on a dimension changes many times maybe its a good idea making it a metric in the fact table and record its changes through time this way, avoiding these problems when using SCD techniques.

Conclusions

The above techniques can have a very profound impact in the Data Warehouse, phisically and logically speaking. If we do not take some wize measures they will degrade Data Warehouse processes very much.

Even if these techniques for handling Slowly Changing Dimensions are very well documented and have already given practical proofs, in the end it will always be necessary for the Data Warehouse architect to analyse each particular case and decide the best way to implement them, because there's always specific situations that need to be attended and particular issues to be resolved.

In my opinion the Type 2 technique is the most useful for most cases. Managers will always want to know how and when changed the information about their business. They don't care if the processes get slower or heavier, they just want to access as much valid data as possible and with precision.

A good implemetation of these techniques for handling Slowly Changing Dimensions will make that Data Warehouse very robust and powerful, giving managers all the valid information they need to make the best decisions about their businesses.

References

1. Kimball R., Reeves L., Ross M., Thornthwaite W.:The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses, John Wiley & Sons, 1998
2. Kimball R., Casserta J.:The Data Warehouse ETL Toolkit:Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data, John Wiley & Sons, 2004
3. Ponniah, P:Data Warehousing Fundamentals:A Comprehensive Guide for IT Professionals, Wiley-IEEE, 2001
4. Todman, C.:Designing a Data Warehouse:Supporting Customer Relationship Management, Prentice Hall PTR, 2001
5. Malinowski, E.:Advanced Data Warehouse Design:From Conventional to Spatial and Temporal Applications, Springer, 2008
6. Corey, M., Abbey, M., Abramson, I., Taub, B.:Oracle8i Data Warehousing:Plan and Build a Robust Data Warehousing and Analisys Solution, Osborne/McGraw-Hill, 2001

7. Stackowiak, R., Rayman, J., Greenwald, R.:Oracle Data Warehousing and Business Intelligence Solutions:With Business Intelligence Solutions, John Wiley & Sons, 2006
8. Marco, D.:Building and Managing the Meta Data Repository:A Full Lifecycle Guide, Wiley, 2000
9. Mundy, J., Thornthwaite, W., Kimball, R.:The Microsoft Data Warehouse Toolkit: With SQL Server 2005 and the Microsoft Business Intelligence Toolset, John Wiley & Sons, 2006
10. The Data Warehouse Architect, http://www.dbmsmag.com/9604d05.html