

Data Warehouse Newsletter #2

Database Design Methods

Version 1.0

16th October 2002

Author: Peter Nolan

pnolan@ozemail.com.au

Table of Contents

1. CHANGE CONTROL LOG	3
2. AUDIENCE	4
3. OVERVIEW	4
4. CONCLUSIONS	4
5. DATA WAREHOUSE DATABASE DESIGN METHODS.....	4
6. STAR SCHEMA	5
6.1. Main Benefits	5
6.2. Multi-Dimensional Databases and Relational Database	7
7. TIME VARIANT PLUS STABILITY ANALYSIS	8
8. SOME TECHNICAL DETALS	10
8.1. Generated Integer Keys	10
8.2. Ability to Build and Store Multiple Levels of Summary Data	11
8.3. Ability to Drill Down to Detail Data	11
8.4. What's Wrong With Third Normal Form?	12

1. CHANGE CONTROL LOG

#	Date	Name	Description
1.0	16/10/02	P. Nolan	Initial publication to the web.

2. AUDIENCE

The intended audiences for this Newsletter are:

- IT Managers responsible for Data Warehouse Initiatives.
- Database Administrators who are interested in how to design a Data Warehouse.

3. OVERVIEW

Many IT staff begin new Data Warehouse development projects using traditional database design techniques, such as Third Normal Form. Thanks to the pioneering publications by Ralph Kimball the world has now accepted dimensional models as a valid way of designing a Data Warehouse. Today there are three widely used database design techniques depending on the business use of the Data Warehouse. This Newsletter discusses the 'other two' popular database design techniques, their purposes, advantages and disadvantages.

4. CONCLUSIONS

This Newsletter concludes that there are three popular Data Warehouse Database Design Methods:

- Third Normal Form
- Star Schema
- Time Variant plus Stability Analysis

In any Data Warehouse project each method should be considered for various parts of the Data Warehouse.

5. DATA WAREHOUSE DATABASE DESIGN METHODS

There are three popular mechanisms commonly used for developing the database model for a Data Warehouse. They are:

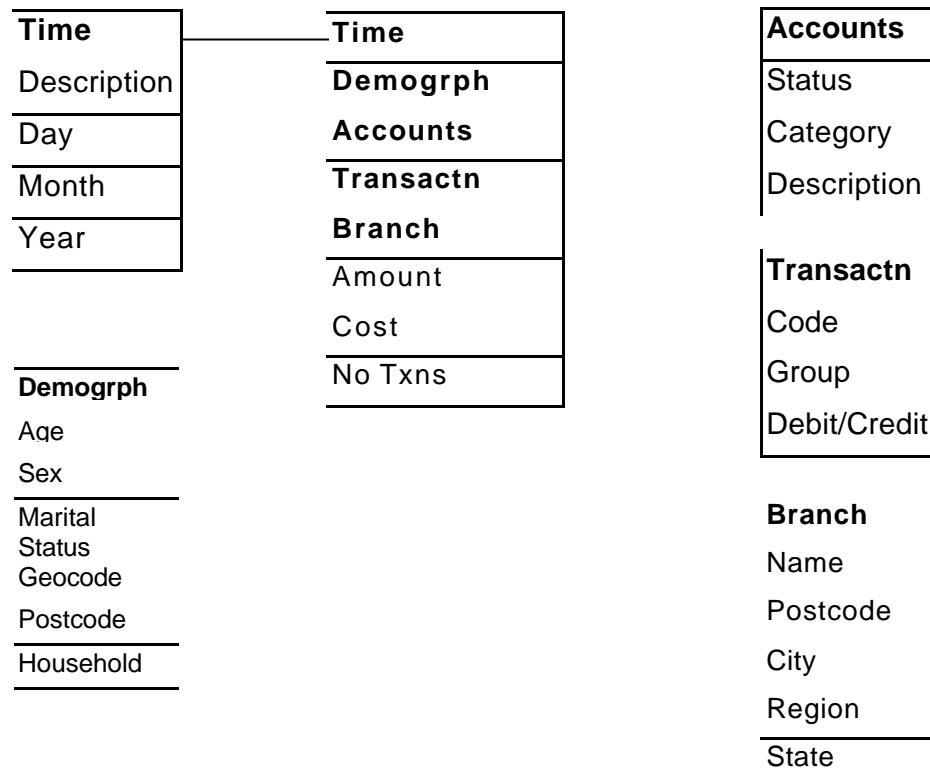
- Third Normal Form
- Star Schema
- Time Variant plus Stability Analysis

There is little need to elaborate on the Third Normal Form model here as this mechanism is widely known amongst Database Administrators. The Star Schema and the Time Variant plus Stability Analysis will be discussed further in this Newsletter.

6. STAR SCHEMA

The Star Schema is essentially an implementation of a multi-dimensional data model using a relational database manager as the data store. Thus the Star Schema enables multi-dimensional analysis on open standard relational databases. There is no truth in the rumour that a multi-dimensional database is required to perform multi-dimensional analysis.

Firstly, what does a Star Schema look like? The following diagram is a simple Star Schema that might be useful for a banking business.



One can see by the diagram that the name star comes from the idea that all the 'facts' or numbers about the business are stored in a central (and often **very** large) table. The large central table is called the Fact Table and the surrounding tables are called Dimension Tables. The Fact Table is generally a transaction table or a summary of transactions table. It records every 'event' that occurs in the business. The tables surrounding the Fact Table hold the natural dimensions of the business. In the banking business these dimensions are typically "who did what, when, where, on what product". Thus we see dimensions of Time, Demographics, Accounts, Transactions and Branch.

In the banking business there is always a Fact Table recording the transactions that have occurred in the bank. You can see from the above database design that it is possible to analyse transaction amounts, number of transactions and transaction costs by Time, Demographic Groupings, Product Groupings, Transaction Types or Branches.

6.1. Main Benefits

The main benefits of using a Star Schema are:

- Easy for Business Users to Understand (intuitively obvious).

One can see that if using a Star Schema an analyst performing ad-hoc decision support can select a

number from a Fact Table by any column (or any combination of columns) in the Dimension Tables. The model is based on business lines, who did what transaction, where and when. It is intuitively obvious to the business user that the key 'facts' can be analysed by any field in the surrounding Dimension Tables.

- Performance

Star Schema is the fastest performing data model for complex queries. That is, queries which are of the format select xxxx, yyyy, sum(a), sum(b) where group by xxxx, yyyy and there are many conditions in which the 'where' clause runs fastest on the Star Schema model. These types of queries form 90+% of all analytical queries.

The Star Schema also allows for multiple levels of summaries to be stored, so long term trends can be stored as summaries and this can provide rapid reliable response times.

Star Schema can be generally expected to perform between 10-1000 times faster than any other form of database design.

- Flexibility

The Star Schema is the most flexible database design for decision support. For example, should you want to slice products using a different field that was not initially defined in the database you could do this by simply adding that field to the appropriate Dimension Table. As the table is small, this is easily done. For example, say you want to have a new attribute of product you wish to use to analyse products by, you could just add that field to the product table and the Star Schema inherently supports the new field.

- Multiple Levels of Summary

The Star Schema also supports multiple levels of summaries in the Fact Table so that one can drill down through data in the one structure and one set of tables. In a project a long time ago I developed new techniques that allow the creation of new levels of summaries using records in a summary control table. Thus adding a new summary is as simple as adding a new record to the summary control table. There are no new database objects to maintain and no new code to write to support the new summary. This is very different to the usual approach of writing new code and creating new tables for every new summary level that is required. This code is also published on my website under 'A Data Warehouse Toolkit'.

- No Wrong Answers Due to Join Problems

One of the most problematic areas of end user queries is that frequently a query that 'looks' correct can produce incorrect results. The two areas where this occurs frequently are the 'inner join' and 'cartesian join' problems.

The 'inner join' problem is where two tables are joined together and both tables do not contain all the valid information. For example, if a report was being produced to show all new accounts opened in the last month and then drill down to new accounts by branch manager for the last month and there was a new branch manager who's new branch manager number had not been placed into the Data Warehouse there is a chance that the report that does not specify branch managers will not reconcile with the report that does specify branch managers. This problem is very common and erodes confidence in the Data Warehouse.

The cartesian join problem is less common. This occurs where not all appropriate joins have been defined and some records are double counted. An example would be producing a report by branch manager where there are two branch managers and only the branch number was used to join records instead of branch number and branch manager number. Because it is typically the IT department that defines joins, this problem occurs less frequently. However, when an end user query tool like MS Access is used, this problem occurs frequently.

The Star Schema eliminates both of these problems as a part of the database design.

6.2. *Multi-Dimensional Databases and Relational Database*

There are different schools of thought amongst industry experts on performing Online Analytical Processing (OLAP) on a Multi-Dimensional Database (MDDB). Multi-Dimensional Database vendors recommendations may be that OLAP can only be done or can achieve the best results done on a MDDB. We believe the following facts need to be considered:

- Most OLAP functions (90+%) can be performed using a Star Schema on a Relational Database using new, smarter, tools. There are some functions which MDDBs contain that relational databases do not. For example date awareness and time variant data.
- MDDBs can manage databases up to 40-50GB of data. As far as I am aware there is no truth to the 'rumours' that MDDBs can actually effectively manage large volumes of data. Thus all the MDDB vendors are building drill-through into the Data Warehouse to be able to access the larger volumes of data typically held in the Data Warehouse. (Note that at least one Multi-Dimensional Database vendor has recently announced a product that is claimed to manage up to 16TB of data. To my knowledge there are no proven large (500GB+) sites using Multi-Dimensional Databases as the primary data store. This may change.)
- MDDBs are closed, proprietary and frequently quite expensive. Far above the price of a relational database manager for a similar power machine. Most interestingly Microsoft SQL Server 2000 is very inexpensive and contains both the RDBMS and the MDDB. For many companies SQL Server 2000 will be a viable Data Warehouse database.
- MDDB vendors do not need to comply to any standards for their language or database and thus have the ability to add new technology at a much greater rate than the relational database vendors.

7. TIME VARIANT PLUS STABILITY ANALYSIS

The third method of storing base data in a Data Warehouse is to use Time Variant plus Stability Analysis Data Model Design Methods.

This method of storing data is frequently used when end users:

- do not know what questions they will ask.
- do not know what the KPIs for the business are.
- do require the ability to answer any question that may be posed in the future that could be answered from any data that was stored in the operational systems at any point in time.

Meeting this type of stringent, ill-defined requirement is a significant challenge.

An example of a time variant plus stability analysis data model is provided below:

Natural Key	Natural Key	Natural Key
Date From	Date From	Date From
Date To	Date To	Date To
Highly	Medium	Low
Volatile	Volatile	Volatile
Data	Data	Data
Elements	Elements	Elements

The basic concept is to send to the Data Warehouse all records that have changed for the last period, usually daily.

Each record that is passed to the Data Warehouse is broken into a number of constituent records based on volatility of the data in that record. Frequently three levels of volatility are chosen though there may be only one or two depending on the benefits the breakdown can offer. Often there is quite a number of data elements that rarely change on a record, such as date of birth, sex, country of citizenship, etc.

Every record is stored with its natural key from the operational system and in addition is assigned a Date_From and Date_To effective date range so that one can tell the value of any field on any date by querying a date between Date_From and Date_To.

Every field on every record that is passed to the Data Warehouse is inspected to see if it has changed since the last period. If it has changed then a new record is placed into the appropriate table. This allows for the Data Warehouse to store every value of every field over every period on a timely extraction basis, such as daily.

From this data structure one can easily see that the database will contain all data that it is possible to store over an extended period. This allows the 'answer any question' capability.

Naturally this data structure requires a very significant volume of storage for records that are highly volatile.

This data structure is not appropriate for transaction based records such as an insurance claim record or a banking transaction record.

This data structure is not appropriate to be queried by end user tools. Generally data is summarised from these detail tables into a structure where there is only one date such as 'effective_day' or 'effective_month' so that end user queries do not specify a between clause on date fields.

Consider a typical example from banking. Let us say that the business managers want to be able to answer any question about any account from data that is available on the account record from any time in the past. This is not possible to support in a Star Schema. An example could be, "Show me the balance of account 123 on 21/6/1996". Another example could be, "Show me the total value of balances for product X on 21/6/1996". This is another example that is frequently difficult to support in a Star Schema because Star Schema snapshot periods are typically monthly or weekly.

In these types of examples one would need a table that contained the daily ending balance for every account to be stored. This is the Time Variant model.

8. SOME TECHNICAL DETAILS

The next section discusses some of the more technical details of these two models and how they need to be built in order to actually provide the base of data for the Data Warehouse. If you are reading this section we expect that you understand the fundamentals of how computers and databases work.

8.1. Generated Integer Keys

By far the most crucial element of any large Star Schema design is that of the generated integer keys for Fact and Dimension Tables. In the Star Schema diagram previously shown the fields in bold are the keys to the tables. All these fields would be generated integer keys for a large Star Schema design. There are a number of reasons why one needs to use generated integer keys for Star Schema databases of any significant size.

1. Smaller keys.

Integers are typically smaller than the real key. For example, in a bank account the account number is frequently 16 characters (check your Visa card). An integer is typically four characters. This saving does not just occur once as the account key is always in an index, thus 24 characters of data are saved for every snapshot of every account record. If the bank has 1M accounts and takes monthly snapshots for three years this is $12 \times 2 \times 36 \times 1\text{M} = 864$ Mbytes of unnecessary data. This is nominal against how much extra data might be carried on the transaction record. For example, if every account had 10 transactions per month the unnecessary data stored for transaction records could be 10GB. The bigger the bank the more disk wasted. But in the end, it's not much disk.

The real reason making the keys smaller is a good idea is that when scanning indexes this data must be moved from the disk, through the IO subsystems and through the processors. This is still a time consuming process in a computer, and you just can't keep buying more memory to try and store it all in memory. This movement of extra data is enhanced by using integer keys in indexes.

2. Faster Processing and Reduced CPU consumption.

Integers are the datatype that computers deal with best. Integer comparisons are faster than character comparisons. A compare for two integers happens a lot faster than a character comparison of two 16 byte character strings. When these comparisons are going to be done billions upon billions of times it pays to optimise them where possible. As I like to say now, if you are going to do something a few hundred billion times it might be worthwhile optimizing it. (How much things have changed in 20 years!!)

3. Multiple Level Summary Support.

The most potent tool the database designer has at his/her disposal for reducing run times of trend analysis queries is the use of summary data. All good database designers realise that looking through every transaction a bank has performed over the last 3 years to get a trend line of the transaction volumes is a waste of processing power and that a summary table will improve that performance significantly. This is one of the attractions of the Multi-Dimensional Databases where data ONLY occurs at the summary level. One of the major issues of Multiple Levels of Summary is 'what keys do you use?'. A summary, by definition, does not have a defined key on the operational system because it does not exist. Thus a key must be generated. If all keys are uniform, generated, meaningless and hidden from the user then multiple levels of summarisation can occur without the end user even being aware of them. This creates one of the most potent advantages of a Star Schema where multiple levels of summaries can be stored in the one or more tables.

These are the three major reasons why all keys in a large Star Schema database should be generalised integers of consistent length.

8.2. *Ability to Build and Store Multiple Levels of Summary Data*

Once generated integer keys are used you get the next benefit - the ability to store multiple levels of summary data in one or more tables. In the analysis of information the end user typically starts at a high level of summarisation and 'drills-down' to detail levels of data. There is a lot of documentation on the process of drill down so we will not discuss it here. However, the drill down process can be most efficiently and effectively managed by creating some summaries in the Star Schema and always using the highest level of summary that supports the query actually being asked. Large mature sites can have in excess of 30 summary levels of data for a single Fact Table, supporting hundreds of users asking for information at all sorts of summary levels within the Fact Table.

Storing multiple levels of summaries is **not optional** in a Data Warehouse. There are those who say that all data should always be stored at the detail level and no summaries should ever exist. In every mature site I have ever come across there is summary data present. **Every single one.** Those summaries might be in the database or in multi-dimensional tables or in end user tools. But what the end user looks at is a summary, there is no denying it.

If you are a DBA and you understand how queries against databases work we suggest, when faced with the 'no summaries' message, you consider the following. Consider keeping 3 years of transactions and wanting to see a trend of these transaction volumes over three years. With no stored summary data the query will read three years worth of transactions. Now lets say that we use stored summaries rather than calculated summaries and we get 1000 times fewer rows. The size of the task is cut 1000 times. A significant cost reduction and useability improvement!

Another point to keep in mind is to read the SQL that users, who say they only want detailed data, actually generate. Most queries are of the form:

```
select aaa,bbb,ccc,sum(xxx),sum(yyy),sum(zzz) from tables group by aaa,bbb,ccc.
```

As a DBA you know that this is a summary. The only question is: should there be some pre-calculated and pre-stored summary data?

8.3. *Ability to Drill Down to Detail Data*

Once it is possible to store multiple levels of summaries and drill down through the data it is only a matter of time before the end user wants to see some of the actual detail records that make up the summary.

In a bank, it is only natural that the end of the analysis frequently ends up with, "list me the transactions that make up this chart" or "list me the customers that make up this chart". In the end, the bank must deal with individual customers to make changes. Thus, as the Data Warehouse technical staff, you should expect that the end user will ask to drill down to detailed data.

This drill down to individual transactions and customers can be supported in both the database designs that are discussed in this Newsletter.

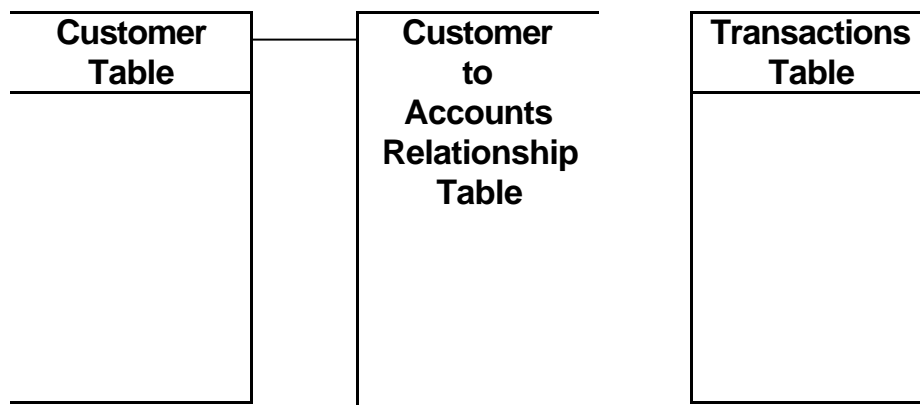
8.4. What's Wrong With Third Normal Form?

Absolutely Nothing!!!

Many DBAs when first reading about Star Schemas and Time Variant data models feel that the concept of Third Normal Form is under some attack and that Data Warehouse designers are invalidating the Third Normal Form model. Nothing could be further from the truth. Third Normal Form is an excellent method of storing data and delivers on its purposes - minimising the volume of data to be stored, supporting referential integrity, and making updates to the data as quickly as possible. None of these purposes are required in the Data Warehouse. Minimising the volume of data or providing the fastest update time is not much of an issue. Both these goals are constantly traded off for fast response times and fast reading times. Thus it is common for data to be replicated, summarised, distributed or whatever, if it will improve the reading times of the huge volumes of data stored in the Data Warehouse.

One of the main issues with Third Normal Form is the CPU it consumes in a large Data Warehouse. To implement a Third Normal Form at the detail level data model one must commit to using a huge amount of processing power, usually in a Massively Parallel Processing (MPP) architecture. This amount of processing power is usually not necessary and hundreds of sites have proven, over and over again, that database design is a far more potent tool for maximising performance than raw computing power.

Consider the following case. A bank wanting to perform demographic analysis on transaction volumes. Who is using auto tellers? What amount of money is being withdrawn by what types of people? If performing this analysis on a Third Normal Form model one would typical see a table design as follows:



The customer table contains all customer information and is keyed by a customer number. The account transactions table contains transaction information and contains the account number as a part of the key. The customer to account relationships table stores both the customer number and account number, as well as some relationship code.

Now, we have assumed that the reading audience is from the IT department so we may ask you to investigate the performance of the types of questions we are suggesting.

Suppose we ask, "What demographic groups, by age band, are using auto tellers?". How is this very simple query performed in the above model?

Firstly, there is no age band field on a customer record to group by, so from the very start we need to scan the customer table once for each age band we are interested in. Say five year bands from 0-60, which means this query would have to run 12 times.

Secondly, because we need age band and transaction volumes we must join the customer table, the customer accounts relationship table and the transaction tables. Let us investigate the size of these tables and the keys of these tables.

In a midrange bank:

- the customer table will be 3 million rows.
- the transaction table will be 200 million for one year worth of transactions.
- The customer to account relationship may be around 6 million rows where customers have a couple of different possible relationships with accounts.

The keys of these tables will be customer number (which may be an integer) and account number (likely to be a 16 character string). To join these tables requires the customer table to be scanned, and all relevant customers selected, a join to the customer account relationship table, a sort (which will be a huge sort) and then a join to the transaction table. We will do this 12 times!!!

Consider the CPU processing power required to support this query.

By using Star Schemas, generated integer keys, multiple levels of summaries and intelligent database design decisions I (and many other database designers) have managed to reduce these types of queries from many thousands of CPU seconds to just a few seconds. Indeed, in original research work I did in the area of demographic analysis of large customer and account bases I found that queries that always consumed over 1000 CPU seconds could be routinely reduced to between 3-10 CPU seconds.

That is a more than 100 fold improvement, to get the same result.