

Data Warehouse Newsletter #9

Insurance Industry Data Warehousing

Large Star Schema Database Designs

For

Demographic Analysis of Large Customer Bases

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. PURPOSE	4
4. ASSUMPTIONS IN THIS DOCUMENT	4
5. THE ANALYSIS OF LARGE CUSTOMER BASES PROBLEM	5
5.1. The Multi-Level Summary	8
6. ALLOCATING INTEGER KEYS, KEY RANGES AND KEY SEQUENCE	9
6.1. Allocating Integer Keys	9
6.2. Allocating Key Ranges	9
6.3. Key Sequencing	9
7. BUILDING MULTILEVEL DIMENSION TABLES.....	10
8. CONTROLLING AGGREGATIONS	11
9. INCREMENTAL UPDATE OF AGGREGATES.....	11
10. AGGREGATE LEVEL NAVIGATION.....	12
11. OVERVIEW OF THE DATA WAREHOUSE LOAD PROCESS	13

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 document are:

- Insurance Data Warehouse Project Management.
- Insurance Data Warehouse Database Designers.

3. PURPOSE

Database designs methods for analysing large customer bases have been emerging from the very rudimentary to the very sophisticated over the last fifteen (15) years. This trend is likely to continue. The current research on how to design and build analytical databases for large customer bases continues to push the boundaries of what is economically possible in terms of analysing large customer bases.

The purpose of this document is to outline a series of ideas that have been used in the building of large star schema data warehouses in large organisations. I have changed it so that it is more specific for insurance companies. I believe the ideas are valuable for people implementing insurance data warehouses.

If you would like to ask questions please direct them to pnolan@ozemail.com.au.

4. ASSUMPTIONS IN THIS DOCUMENT

For the remainder of the document I will make the following assumptions: I assume:

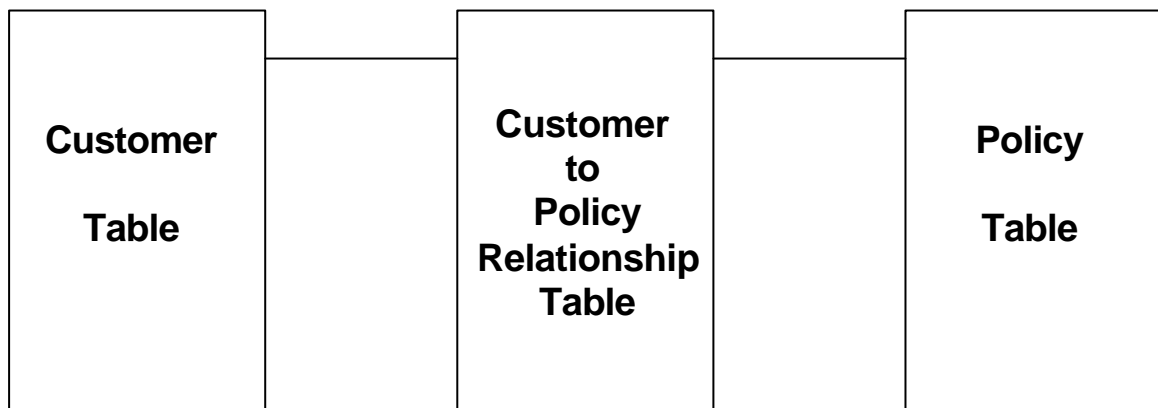
- You are very familiar with the star schema design mechanisms and the various issues in such technical issues as the need for integer keys and the performance aspects of computer systems and IO subsystems.
- You have read or are familiar with the concepts expressed in Newsletter #2 'Data Warehousing Database Design Methods' also available on my web site.
- You understand that there is significant business advantage to be gained from being able to analyse the policy information and transactions on policy information within your insurance company. Thus, I will not explain these business benefits in this document.

5. THE ANALYSIS OF LARGE CUSTOMER BASES PROBLEM

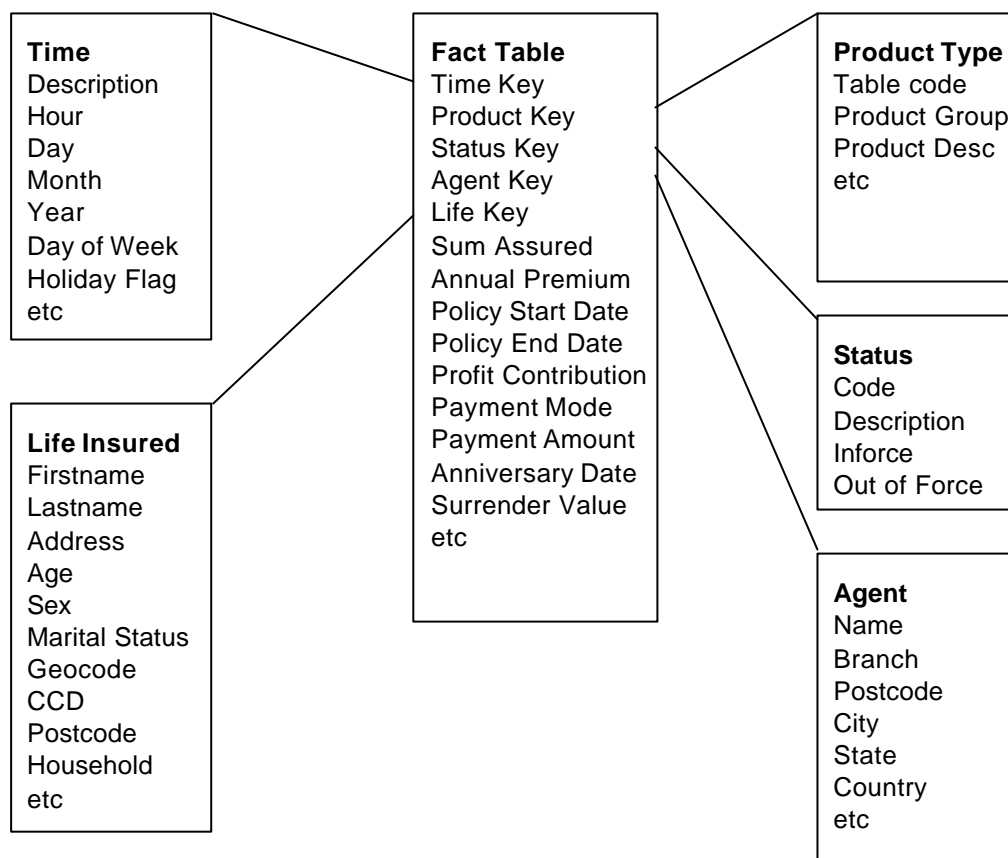
This section describes the analysis of “Large Customer Bases Problem”. The section is included simply to set the level of the discussion and to describe the problem at hand.

Given that there is a need to analyse the transactions and policy behaviour of large customer bases in insurance companies I assume you understand the performance issues of third normal form database designs for analysis of large customer bases. For example, I assume that you understand the issues of a data model that follows.

Naturally the joins and the sorts for customer demographic analysis is too expensive to be able to afford. Or, more correctly, there is no need to spend the money required for the computing power to afford analysis on such a data model since star-schemas provide the same capability at far lower cost.



To solve the problem of analysing very large policy and payment volumes of very large customer bases in a reasonable cost of computing it is possible to apply the star-schema database design. Thus, a simple (or oversimplified) star schema for an insurance company may look as follows:



This data model removes the massive sorts required because the Life Insured Key (customer key) is on the fact table. This data structure will perform much faster than the third normal form model and this type of data structure is excellent for storing the base level policy and transaction information at the lowest level of a Data Warehouse.

With the advent of Oracle 8 it is now possible to resolve such a query without performing a Cartesian product. This has dramatically improved the performance of star schema joins in Oracle.

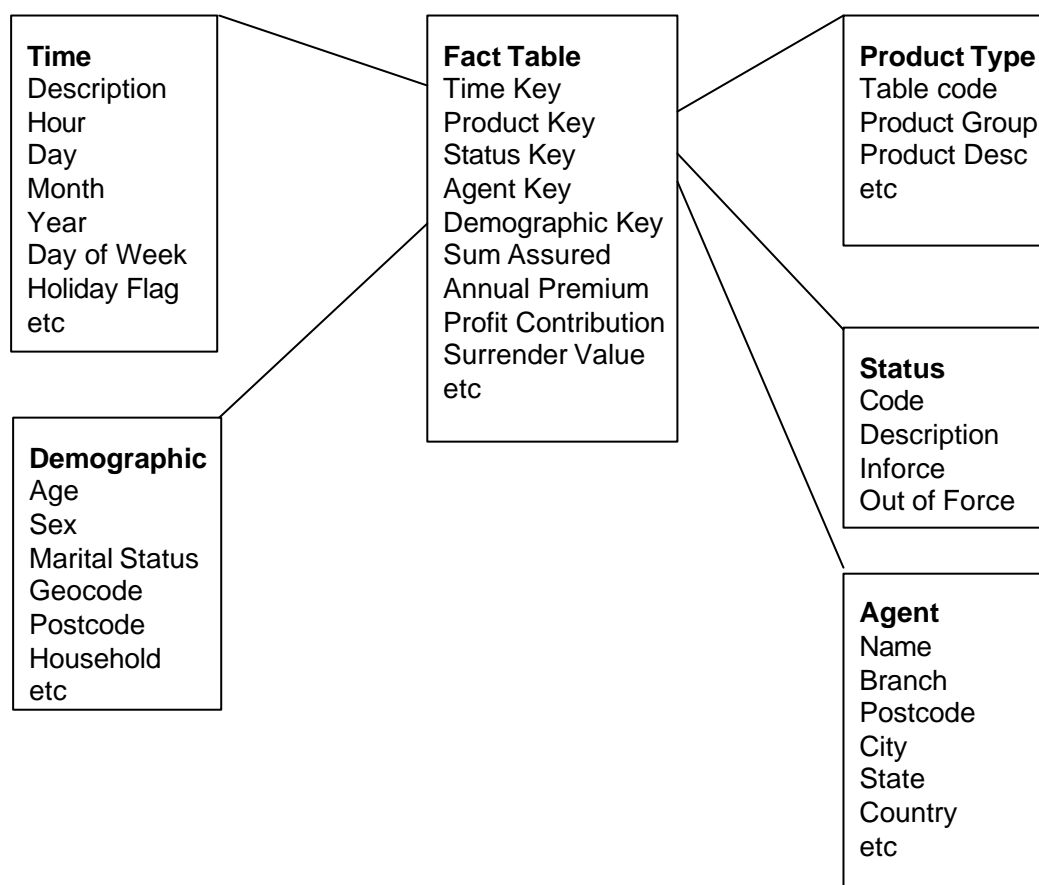
The introduction of bit mapped indexes has allowed us, for the first time, to be able to sensibly scan vast numbers of detailed level transactions on databases other than NCR/Teradata.

However, how often do we need to scan the detailed transaction fact or policy fact table in an insurance company?

If one stops to consider what is actually required of analytics on large customer bases one soon realises that no-one is actually interested in the specific customer or the specific policy in most cases (though access to this data is required). What business people are actually interested in are the trends over time and the changes in the mix of products or customers, or the success of product launches or other marketing campaigns. Thus, for the majority of cases the real business question does not look at the individual policy or the individual person. The business question looks at some level of summary.

The concept of using some sort of summary is extremely well accepted in Data Warehousing today. Thus, the simplified data model on the previous page may group the customer data into demographic groups.

This type of data is far more efficient at analysing demographic groups, by product groups by agents and branches because there are thousands of times fewer records in the demographic dimension table which leads to far fewer rows being produced in the cartesian product intermediate table and then far fewer lookups into the fact table.



Thus, the best solution available today to analyse very large customer bases such as policies in insurance companies is a combination of detailed level fact tables and summary level fact tables.

5.1. The Multi-Level Summary

The last point to be made in this introduction on large stars for insurance companies is this. How many summaries does one have? And how are they generated?

It is possible to generate Multi-Level Summaries in a star schema and to navigate into the multi-level summary fact tables using levels on the dimension tables. Databases I've implemented over the years have allowed N levels of summary in M numbers of fact tables. Most people will be aware that there has been discussion on dwlist as to whether each summary should have it's own physical table. There are pros and cons to doing this. I am saying that if you choose to have one fact table per summary that's fine, and there is an overhead for doing that. I am also saying that by implementing multiple level summaries you can avoid some of that overhead, at the cost of having to navigate the summaries.

Thus, it is possible to create multi-level dimension tables that then generate multi-level summary levels in one or more fact tables and be able to navigate into these tables using intelligent tools.

This design method has been used at numerous large customers. This begs the question, what are the issues in developing such tables? The remainder of this paper describes some of the most problematic areas of developing large star schema databases.

The Data Warehouse Generator on my web page publishes the actual code required to build such a set of summaries.

6. ALLOCATING INTEGER KEYS, KEY RANGES AND KEY SEQUENCE

A major problem when building a very large star-schema database is the allocation of integer keys, the sequence of those keys and the key ranges that are used. Each of these problems will be described in detail. The Data Warehouse Generator has effectively solved all these problems.

6.1. Allocating Integer Keys

The issue of allocating integer keys is a major problem. The problem is what integers should be generated? The method that is used for multi-level summary tables is that the integer keys that make up the primary key of the summary record define the uniqueness of the record and thus the specific summary record. Therefore, if a fact table is to contain multiple levels of summary the keys for each level of summary must be unique.

In fact, for any given dimension the keys used for the detail level and the keys for all summary levels for the dimension must all be unique across the dimension if one is to have freedom of placement of summaries into any fact tables in any combination.

Thus, these keys must be carefully allocated. For very small dimensions these keys can be allocated by the DBAs. For the very large dimension tables, such as customer, these keys must be allocated in code. Thus, for very large dimensions they must be allocated in key ranges. Some of my star schema designs allocate keys as the customer and account records are updated in the detail level dimension tables. The Data Warehouse Generator builds data models with specific tables to manage the key allocations by any field on the detail level record for the dimension.

6.2. Allocating Key Ranges

The issue of allocating key ranges immediately raises questions such as how are the key ranges defined and how is the next key generated? The usual solution is to create a control table that stores the key ranges and the aggregate key level ranges for all the system generated keys. Thus, the code that maintains the dimension tables must be aware of new records entering the dimension and the setting of integer keys for aggregate levels of the dimension record as the record enters the dimension table.

6.3. Key Sequencing

Key sequencing was a mechanism that was particularly useful on the 3390 type disk storage on DB2/MVS. It brings only some of its value to the open systems RAID type disks and Oracle 8. In the past, I have allocated integer keys that forced the clustering of the data into the order in which the data is most often read, then placed that data in cylinders.

Oracle seems to have introduced some of these types of capability in Oracle 8, in particular the physical partitioning of tables by key sequence. Though I am yet to test whether data can be forced to be in certain row orders and forced to be retrieved by some sort of sequential pre-fetch.

I trust that knowledge of key generation, key range allocation, key sequencing and where/how to store the integer keys is of value to you.

7. **BUILDING MULT-LEVEL DIMENSION TABLES**

The biggest problem table in a Insurance star schema is the customer (person insured) dimension tables. Having already established that summaries are highly beneficial and that integer keys give one great flexibility and performance improvements one is left with the problem of what to do about this large dimension table.

My experience has been that it is necessary to allow multi-level summary keys to direct queries back into the fact table by the summary dimension tables. Whether this is done by putting levels of dimensions/facts into different tables or the same table makes no real difference to the logical view of the model. Let's first take the Customer dimension as an example.

Analytics on the customer dimension are desirable for any number of attributes. For example:

- Age (or age band)
- Sex
- Martial Status
- Occupation
- Employer Asic code
- Geodemographic code (a kind of combination of income and location generated from Census data)
- Geographic code of residence including CCD and Postcode.
- Nationality
- Ethnic/Racial background

are all useful attributes to understand the customer base using a specific product.

In the average Insurance system some of these things are recorded and some are not. For example the insurance company would collect date of birth (age), but geodemographic code would have to be supplied through an algorithm based on census data which would not be stored in the Insurance systems.

Almost always, these attributes are placed on the customer record at the customer level. Thus, if one wants to perform analysis it would make sense to create some summary level of the dimension. However, all these fields are not used in every query. The query based on age is very common, but queries based on nationality or ethnic background are far less frequent. Thus, it would be more efficient to create a customer dimension level of age and age band, and another one for ethnic background.

The problems with this situation are:

- how many summary levels can be created?
- what summary levels should be created?
- how would the summary levels be created?

The answers I believe that are most useful are:

- as many summary levels as one likes, the Data Warehouse Generator defaults this to nine (9) levels (detail + summary levels =10). (Why? I have 10 fingers.)
- any summary levels one likes as long as the attributes are available on the detailed client record.
- automatically with the minimum of system changes.

It is possible to create multi-level summary dimensions for large dimensions such as customer based on any field on the input table, for as many levels as required (9 seems to be enough) without the need to be constantly changing code.

I trust that this is of value to you.

8. CONTROLLING AGGREGATIONS

By enabling the development of multi-level aggregates for large (number of rows) dimension tables as well as small dimension tables using integer keys for all keys for the levels of the dimensions these star schema designs have another major benefit.

Many years ago a major bank I was asked to work with had significant performance problems with a large star schema data warehouse. The problem was that reports required some fields only available at a quite low level of summary. If these fields could be placed at a higher level of summary significant performance improvements could be enjoyed. I made this recommendation. However, the change cost the customer 55 days programmer time.

As a result of this direct experience, I thought it would be useful for my star schema designs to able to create new levels of summaries forward and backward in time with no code changes. This is managed by a summary control table which defines every level of summary for every dimension for every fact table. Thus, if a new level of summary is required in a fact table it is only necessary for one of the DBA staff to add a new row to the summary control table and that summary will be created forward in time as new data is loaded into the data warehouse. This same code also supports the ability to create summaries backwards in time. Naturally the unloading and creation of aggregates backward in time requires the IT department to perform this work due to the massive amount of computing resources required to do so.

Thus, the Data Warehouse Generator can control the creation and generation of aggregates in a Star Schema database, both forward and backwards in time, using an aggregate control table and cobol code to control the generation of these aggregates.

9. INCREMENTAL UPDATE OF AGGREGATES

Once aggregates have been defined there is the issue of how to update the aggregates. The problem is, if transactions are sent to the data warehouse on a daily basis (highly recommended) and there are aggregates by week, month, quarter, year etc then how are these detailed transactions consolidated into the multi-level summaries above the frequency of update applied?

The Red-Brick load utility allows for in place consolidation of detail records to aggregate levels of data. As far as I am aware no other database load utility does this.

Since most of my projects were DB2/Oracle the Data Warehouse Generator has code that allows the incremental update of aggregate levels in many multi-level summary aggregate fact tables using static SQL. (Dynamic SQL performs very poorly). This mechanism allows as many aggregates as desired to be created at as many levels as desired and for the detail records to be aggregated and then compared with the records in the various aggregate level tables. It does this by using the primary key of the aggregate record to define the summary record and it consolidates the new record flowing through to the data warehouse where the primary keys match.

I firmly believe that the knowledge of incremental updates to multi-level summary tables in a star schema is of value, especially if the batch window from extraction of data to delivery of data in the data warehouse is constrained. Why do I believe this? Because if summaries are created from the detailed data on a nightly basis the extra processor time required is very large. I know this because this is the way that you (just about) have to implement summaries in most of the ETL tools.

10. AGGREGATE LEVEL NAVIGATION

Having created a sophisticated multi-level summary star schema database there is then the issue of properly accessing the database and ensuring that a customer does not double count any rows. This issue is still largely unresolved to this point in time.

There are tools available that contain what are called 'Aggregate Level Navigators'. This is a concept that Bill Inmon and Ralph Kimball have championed for years though it has not been widely implemented.

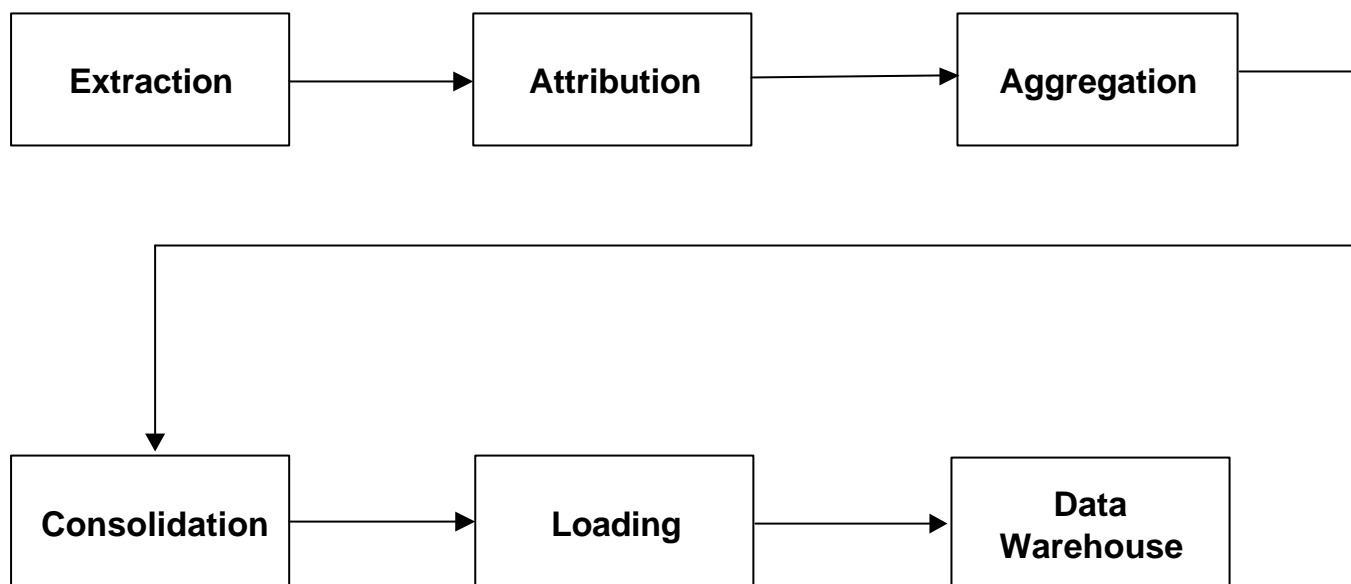
The concept of an aggregate level navigator is that all questions are always developed by the end user at the most detailed level of the data. The SQL is sent to the server which analyses a set of metadata tables similar to a regular database catalog to determine the highest level of aggregated data that will still answer the question accurately. Once these tables are determined the names of the detailed level tables and the name of the summary level tables are substituted into the SQL and the SQL is then forwarded to the database engine.

This is a particularly useful way of navigating aggregate level tables where all the aggregation is hidden from the end user.

An example of implementing this type of navigator is Micro Strategys DSS Agent. Most tool vendors have not really included aggregate navigation into their products. This seems to be a shame. But then again, none of the vendors are motivated to save you money on your hardware purchases.

11. OVERVIEW OF THE DATA WAREHOUSE LOAD PROCESS

All these functions that we have been talking about require code to populate the data warehouse. This section provides a very brief overview of the Star Schema Data Warehouse Load Process.



There are five (5) processes required to take data from a source and then place the data into the Data Warehouse.

1. Extraction – which is self explanatory.
2. Attribution
This is the process of allocating the integer keys to the detailed record.
3. Aggregation
This is the process of creating aggregate records from the detail fact table records.
4. Consolidation
This is the process of consolidating new aggregate with possibly existing aggregate records in the Data Warehouse.
5. Loading/Updating
This is the process of insert/updating aggregate levels and loading detail records into the Data Warehouse.

Please note. Most processing in the Data Warehouse load process has initially been written in cobol working on sequential files. This is done to improve performance of the process.