Automatic Initial Load of Data Warehouses as a Workflow Process

- DRAFT -

Ignacio Larrañaga
Grupo de Concepción de Sistemas de Información (CSI)
Facultad de Ingeniería, Universidad de la Republica.
Montevideo, Uruguay.
ilarra@fing.edu.uy

Abstract

In Data Warehouses Load and Maintenance area, the main line sees the Load and Maintenance as a View maintenance problem. Other line says that the problem cannot be seen in this way, but can be model as Workflow process. That is, a process whose nodes performs activities that cover the extraction, transformation, cleaning and integration.
Following the second line, we present a proposition to automatically design a basic workflow process that performs the load process.

Keywords: Data Warehouses, Initial Load, Workflow.

1. Introduction

In this paper we not try to explain if the view of the problem focused on the view maintenance problem is right or wrong. Simply because this work is not focused on this topic. But we must note that some authors, that was study the problem, present and explain other line which is explored here.

Following this line, our intention is to explore who the workflow view can be exploited, and get easily, practical results. In the followings paragraphs we present some the authors that present this line. After that we introduce how we can make a contribution.

In [MFM99], Mokrane, Fabret and Maja explain that data refreshment in data warehouses is generally confused with data loading as done during the initial phase or with update propagation through a set of materialized views. At this paper the authors organized the refreshment process as workflow.

In [GFSS00] Galhardas, Florescu, Shasha and Simon present a declarative SQL extension to solve the cleaning problem. As we know this is a part of data warehouse load and refreshment process. According to our proposal this is modeling as a graph of transformations, which easily can be viewed as a workflow process.

In [LWMG] the resumption of interrupted warehouse loads is treated. The first hypothesis taken for the authors is that the load process is a workflow, then they present an algorithm that works over this workflow.

Now that we introduce that the problem can be see as a workflow process, we try to show you, how in a small problem much of the load process can be designed automatically. In this paper we only consider a small part of the work that can be done (for example, only the load process is considered). The main intention is show that the work can be done, and the result can be useful.

The following section intend to present some of whose are the elements, the tools, that we can consider when we are trying to design a process automatically.

1.1. Integrity Constraints in a Data Warehouse

Integrity constraints provide a mechanism for ensuring that data conforms to guidelines specified by the designer. The most common types of constraints include:

- **UNIQUE constraints**. To ensure that a given column is unique.
- **NOT NULL constraints**. To ensure that a column has no null values.
- **DOMAIN constraints**. To be assure that the values of some column are the desired values.
- **FOREIGN KEY constraints**. To ensure that the values in a table have the correct relation with other table.

Constraints can be used for these purposes in a data warehouse:

- **Data integrity**. Constraints verify that the data in the data warehouse conforms to a basic level of data consistency and correctness, avoiding the introduction of dirty data.
- **Load and Refreshment**. The constraints can be useful to rewrite the load SQL sentences to optimize those processes.
- **Query optimization**. Some databases utilize constraints when optimizing SQL queries. Although constraints can be particularly important for query rewrite of materialized views.

For data warehousing, many users have discovered that such constraints may be prohibitively costly to build and maintain. By example A UNIQUE constraint is typically enforced using a UNIQUE index. However, in a data warehouse whose tables can be extremely large (millions or even billions of rows in a FAT table), creating a unique index can be costly both in processing time and in disk space.

A common technique before modifying this table was drops the constraint. Then, make all necessary data modifications. Finally, re-create the constraint. However, this approach does not guarantee that data added to the table while the constraint has been dropped is unique.

However we argument that taken they carefully the associated cost must be maintained under control. Moreover, various techniques can be used to reduce this cost. For example, all constraints can be validated in parallel. When validating constraints on very large tables, parallelism is good for performance goals. Partitioning also can improve constraint management.

We can present other type of constraints that can help us, for example, OLAP constraints. If we know that a table is a cube, and detect another table/cube that have less degree of granularity, then we can

consider if it is better to perform a direct load from the source, or make a drill up from the previous cube. Because this work only introduce the problem, we consider that the presented elements are enough.

The rest of this work is organized as follows. Section 2, present a the load process, which is divided into an insertion process (Section 2.1), an deletion process (Section 2.2), and finally, the full load process that can be designed. Section 3, present the conclusions and the future work.

2. Load Workflow Process

Suppose the following problem, a data warehouse associated with the university student's information and you course inscriptions. There are four tables. From the student's we know the name, sex, address, phone and where they come from (PLACEID). The student's come from different places, which is represented in the PLACES table.
Also we know the courses of the university (COURSES table), which have, course identification and name. The students make inscriptions for different courses; the inscription is represented as an association between one student and one course (INSCRIPTIONS table).

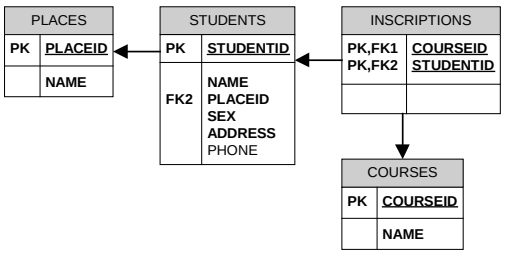

Figure 1: University student's inscriptions problem.

Then consider a load process for this database, it must get data from the operational databases and put into this tables.

Suppose the most simple situation, the operational database have the same scheme as the data warehouse, the load process is the initial load and the data warehouses tables are empty. Then the process is a massive insert of data from the operational tables to the data warehouse tables.

As definition of massive insert we will denote this as a task that take all data from the source and put it into one data warehouse table.
The data structure that is taken from the source databases is represented as one extraction SQL sentence.
The schema of the target table defines the schema data that is putting into the database.
The only restriction that we impose is that the SQL instruction data schema is the same as the target database table.

In the example of the Figure 1 (taking into account that the source and data warehouse schema is the same) the massive insert task for PLACES data warehouse table is described by a SQL sentence over the source data "*select * from PLACES*" and the target table PLACES of the data warehouse.
Over STUDENTS table is described by the SQL "*select * from STUDENTS*" and the STUDENTS table as target, and so on.

But the massive insert must assure some constraints, particularly the foreign key constraints of the data warehouse, because if not, the insertion fails.
Suppose that you try to insert the data from the STUDENTS table before the PLACES table. Easily you can see that the foreign key between the STUDENTS and PLACES will be

broken, because the PLACES table is empty.

Extending this observation over all tables we have some dependencies between the massive insert tasks of the load process. The load process can look like as Figure 2, where the nodes are the massive insert tasks (as defined before), and the lines defines the dependencies between the tasks. Notice, this is an extremely basic workflow definition.

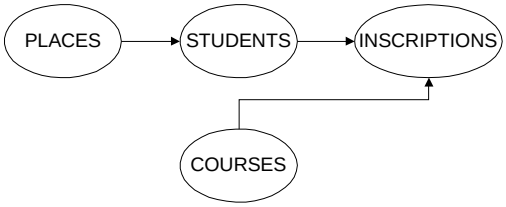

Figure 2: University data warehouse load process.

A partial result that will be demonstrated is that if the foreign key between two tables is from A to B (Figure 3), then the task dependency between the massive insertion for A and B is in the opposite direction and must exist.

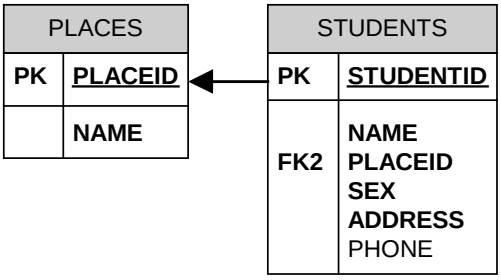

Figure 3: A is STUDENTS and B is PLACES.

The probe is quite simple, suppose that the dependency not exists (Figure 4), then the insertion can fail if the tuple of A try to be inserted before the referenced tuple of B, but the tuple of A exists, then the process must not fail.

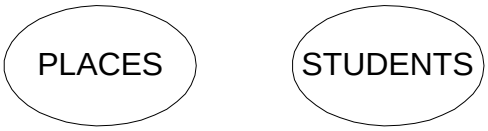

Figure 4: The dependency between the tasks not exists.

Now suppose that the dependency exists, but in the same direction of the foreign key (Figure 5). That is the same case considered before, any tuple of A will be inserted before the tuples of B, then the referenced tuple of B never exists when the tuple of A is inserted, then the foreign key fails. As before the insertion process must not fail because the referenced tuple exists in B.

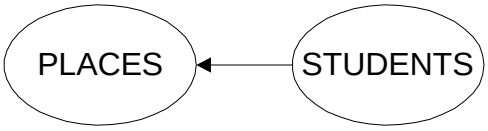

Figure 5: The dependency between the tasks is in the foreign key direction.

This little example shows a more general assertion that we can make over all load process.

2.1. Insertion Workflow Process

Proposition 1: *A relational data warehouse and a group of the extraction SQL sentences for all data warehouse tables define a workflow process called "**Insertion**". In this process, the tasks are massive insertion tasks, defined by the target table and the SQL sentence associated. The dependencies between tasks are in the opposite direction of the foreign keys for the target data warehouse tables associated to the process tasks. Also are not other dependencies between tasks (except by un-benefit dependencies, which are not considered)*[1].

[1] That is a necessary and sufficient condition. This is, this process must have those

Demonstration: By definition of the massive insertion nodes they are defined by one extraction SQL sentence and the associated table. The hypothesis says that this element exists, and then the massive insertion tasks are defined.

The previous result assures that if a foreign key exists between two of those tasks, the task dependency must exists and in the opposite direction of the foreign key.

Now suppose that there is not the case that exists a foreign key between two tables, and suppose that we make a dependency between the associated tasks. Because there are no constraints between them, the process will get the same result in any case (that is, if the dependency exists in some direction, or not), then the dependency does not benefit the workflow in any case, then it must not exist.

Now if it is considered as a simple workflow process definition the result is confirmed.

Now we will try to raise the assumption that the source relational database has the same schema as the data warehouse. This makes no effects over our last proposition.

Suppose that the schema is not the same, obviously this affect the extraction SQL sentences, but if the structure of the SQL result tuple is the same of the data warehouse tuple the Proposition 1 is not affected, because the hypothesis remains and the massive insertion nodes can be created.

dependencies, and is enough that have those dependencies.

The assumption that the data warehouse is empty can be raise too, maintaining that the process is a initial load process (later in this document we treat other load processes).

To raise this assumption we will introduce another kind of tasks, the massive delete tasks. The massive delete tasks are defined by a data warehouse table, then its have an associated SQL sentence to be executed that is (supposing that the table name is A): "*delete from A*".

The massive delete tasks for example at Figure 1 have associated the followings SQL instruction that gives your semantic:

- PLACES: "*delete from PLACES*"
- STUDENTS: "*delete from STUDENTS*"
- COURSES: "*delete from COURSES*"
- INSCRIPTIONS: "*delete from INSCRIPTIONS*"

Now we show partial results much similar to the massive insert tasks. If the foreign key between two tables is from A to B (Figure 3), then the task dependency between the massive delete for A and B is in the same direction and must exist.

The probe is equal to the previous one. Suppose that there is no dependency, then the delete can fail if the tuple of B try to be deleted before the referenced tuple of A. But if the tuple of A is deleted first the process success. Then, the process must not fail.

Now suppose that the dependency exists, but in the opposite direction of the foreign key. Any tuple of B will be deleted before the tuples of A, then the foreign key fails, because the tuple of B is referenced by tuples of A. As before the delete process must not fail, then this option is not valid.

In conclusion, the partial result is true because the other available options can fail, then the other options are not correct.

Now as before this example shows a more general assertion that we can make over all load process.

2.2. Deletion Workflow Process

Proposition 2: *A relational data warehouse define a workflow process called "**Deletion**". In this process, the tasks are massive delete tasks, defined by the target table. The dependencies between tasks are in the same direction of the foreign keys for the target data warehouse tables associated to the process tasks. Also are not other dependencies between tasks (except by un-benefit dependencies, which are not considered)*[2].

Demonstration: By definition of the massive delete tasks they are defined by the associated table. The hypothesis says that this element exists, and then the massive delete tasks are defined.

The previous result assures that if a foreign key exists between two of those tasks, the task dependency must exists and in the same direction of the foreign key.

Now suppose that there is not a foreign key between two tables, and suppose that we make a dependency between the associated tasks. Because there are no constraints between them, the process will get the same result in any case (that is, if the dependency exists in some direction, or not), then the dependency does not benefit the workflow in any case, then it must not exist.

Now if it is considered as a simple workflow process definition the result is confirmed.

At this moment our load process for an non empty data warehouse look like this:

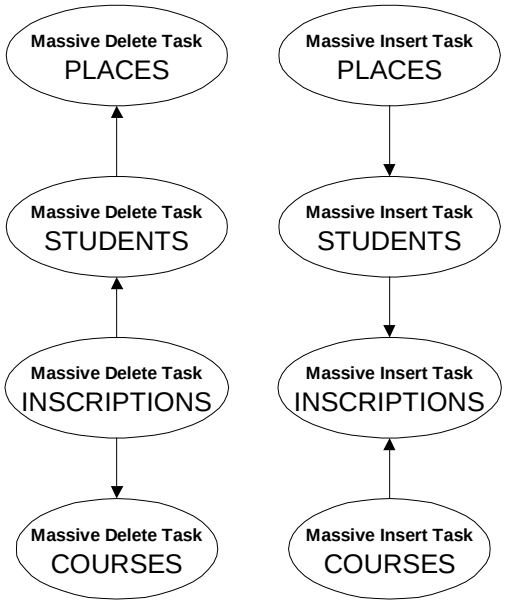

Figure 6: The **Insertion** process (at right) and the **Deletion** process (at left).

In Figure 6 all is coherent except that Insertion and Deletion processes are not connected. That is correct with the actual status because they are studied separately. The following step in our study is the connection of this two process to make the global process.

The first consideration that we must take into account is that the connection between those two processes could be focus by two different approaches, inter-process or intra-process.

As inter-process we mean that the connection is of the type Insertion process and it must be executed before Deletion process, or Deletion process

[2] That is a necessary and sufficient condition. This is, this process must have those dependencies, and is enough that have those dependencies.

will be the first or they have no order to be executed.

As intra-process we understand that the connection is of type task A into Insertion process must be executed before task B into Deletion process, or in the opposite way, or they have no order.

The approach follows in this paper is the second one. The main reasons are that the second allows better representation of the task relation and also allows more degree of parallelism. We not discard the first one we only focus the second.

The results that we goes to show is that the task dependencies between task of both process are from the massive delete task of target table A to the massive insert task of target table B, and no any one dependency is needed. At our example it is show at Figure 7.

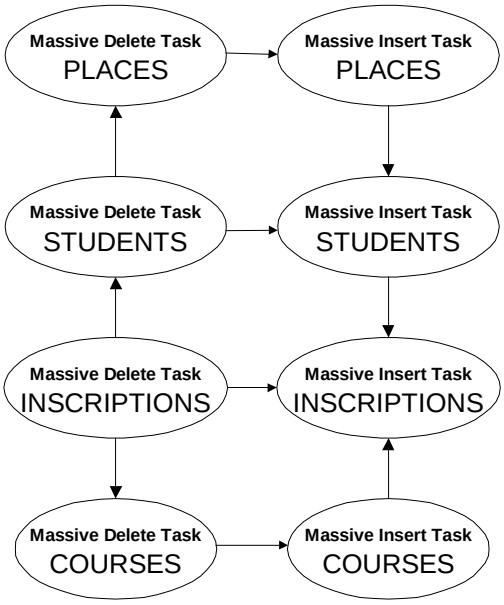

Figure 7: Full-Load Workflow Process for Student's inscription problem.

2.3. Full-Load Workflow Process

Proposition 3: *The intra-process relation between an Insertion and Deletion Workflow process defines a workflow process called "**Full-Load**" which have all tasks and dependencies of both Insertion and Deletion Workflow process moreover one task dependency between each massive delete task and massive insert task with the same target table. That dependency is from the delete task to the insert task. Furthermore no any task dependency is needed*[3].

Before the demonstration we must note that the hypothesis says that with the dependencies proposed is sufficient to perform a full load, but not necessarily the dependencies will be those ones.

That is, can exist other processes (with exactly the same tasks) that can get the same result, and no necessarily the proposed dependencies belongs to the dependencies of those others processes.

Before (at Insertion and Deletion process definition), we say that there are no others dependencies (except by un-benefit dependencies). If the process has not the dependencies defined then the process cannot get the same result.

Demonstration: By definition the Insertion and Deletion are Workflow processes, even if we stop here we already have a Workflow process defined by both but not with the properties needed.

Now take two tasks A and B, where A belongs to the Insertion process, B belongs to the Deletion process and both have the same target table. There are three possibilities, the dependency over those tasks is from A to B, B to A or it not exist. There are not other

[3] That is a sufficient but not necessary condition as is explained in the following paragraph.

possibilities because not exists other dependencies inside the Insertion or Deletion process, and the dependencies between task with different target tables are not considered.

By definition is obvious that the option to chose is the dependency from B to A, because we need to delete the target table before and then put the new data.

Because those dependencies are made over all data warehouse tables (there are one insert and delete task for each table in the data warehouse by definition of Insertion and Deletion process), all the tables are clear and the fill with the new data and that is what we ask.

3. Conclusions and Future Work

In our point of view, this paper present that a line of work, which follows the load and refreshment process in the workflow way, can be analyzed and develop automatic basic processes from existing information.
Those processes can be used by the data warehouse designer as start point, then the problem is not take from the scratch. We think that is a contribution for the designer.

We do not expect that the present ideas are totally new, but it sounds estrange that the simply idea which is presented here, was not already in a basic bibliography, may be as a little comment. “If you must design a load process you must take into account the foreign keys”, that is really a basic idea. Some months ago when I was designing a load process and I spent a lot of hours developing this I stopped and think: “This is not right I must do something”, this was the origin of this work.

In the future, based on the experience of develop this work, we plain to extend the propose to manage refreshment processes, cleaning processes and other issues of the design process. The main intention is, if the designer do not spend time in these things, that every time do, then a contribution can be made.

In other way, if anything can be done, I will happy to read a work that explain why.

4. References

[GFSS00] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon. Declaratively cleaning your data using AJAX, 2000.

[LWMG] Wilburt Juan Labio, Janet L. Wiener, Hector Garcia-Molina, Vlad Gorelik. Efficient Resumption of Interrupted Warehouses Loads.

[MFM99] Mokrane Bouzeghoub, Françoise Fabert, Maja Matulovic-Broqué. Modeling Data Warehouse Refreshment Process as a Workflow Application. Proceedings of the International Workshop on Design and Management of Data Warehouses, Heidelberg, Germany, 14-15.6.1999.