# Optimizing Data Warehouse Loading Procedures for Enabling Useful-Time Data Warehousing

Ricardo Jorge Santos[1] and Jorge Bernardino[1,2]

[1] CISUC – Centre of Informatics and Systems of the University of Coimbra – Portugal
[2] ISEC – Superior Institute of Engineering of the Polytechnic Institute of Coimbra – Portugal
lionsoftware.ricardo@gmail.com, jorge@isec.pt

**Abstract.** The purpose of a data warehouse is to aid decision making. As the real-time enterprise evolves, synchronism between transactional data and data warehouses is redefined. To cope with real-time requirements, the data warehouses must be able to enable continuous data integration, in order to deal with the most recent business data. Traditional data warehouses are unable to support any dynamics in structure and content while they are available for OLAP. Their data is periodically updated because they are unprepared for continuous data integration. For real-time enterprises with needs in decision support while the transactions are occurring, (near) real-time data warehousing seem very promising. In this paper we present a survey on testing today's most used loading techniques and analyze which are the best data loading methods, presenting a methodology for efficiently supporting continuous data integration for data warehouses. To accomplish this, we use techniques such as table structure replication with minimum content and query predicate restrictions for selecting data, to enable loading data in the data warehouse continuously, with minimum impact in query execution time. We demonstrate the efficiency of the method using benchmark TPC-H and executing query workloads while simultaneously performing continuous data integration.

## 1 Introduction

Essentially, operational systems are transactional systems, which support the daily business processes and store the whole complete and detailed information for each business transaction while they occur, i.e, in real-time. These systems use *on-line transactional processing* (OLTP). Since they need to insure their availability for supporting each current business transaction, OLTP databases frequently store only the current daily information which is considered as essential for those transactions. They usually process small, atomic and isolated transactions, which typically read or/and write a very small amount of data. These features allow minimizing data processing efforts, in order to minimize processing response time for each transaction, so that it can handle a high volume of simultaneous transactions, therefore maximizing performance and availability for their users.

The main purpose of a data warehouse (DW) is to aid decision making. In order to accomplish this, it collects data from multiple heterogeneous operational source systems and stores summarized integrated business data in a central repository used by analytical applications with different user requirements. The databases of a DW usually store the complete history of the business. Because of this, they frequently have a huge number of rows and grow to gigabytes, terabytes and even petabytes of storage size. The common process for obtaining decision making information from DWs is based on executing *ad-hoc* queries and tools which perform *on-line analytical processing* (OLAP) [7]. Due to the dimension of the tables within the DW, decision support OLAP queries usually access huge amounts of data, making OLAP performance one of the most important issues in data warehousing.

Traditionally, it has been well accepted that DW databases are updated periodically, typically in a daily, weekly or even monthly basis [28]. This update policy was defined as common practice due to two major assumptions: 1) Since the DW has huge sized tables, performing data integration in a continuous fashion, similar to transactional operational systems, would create immense bottlenecks in OLAP processing and data updating procedures, due to the time and resource consumption implied in loading data in such tables and updating the data structures (such as indexes, partitions and materialized views, among others) used for performance optimization; and 2) Business decisions were made on a daily, weekly or monthly basis, dispensing near real-time business data. This type of update policy meant that the DW's data was never up-to-date, because transactional records saved between those updates were not included in its databases. This implied that the most recent operational records were excluded from results supplied by OLAP tools.

Until recently, using periodically updated data for obtaining decision support information was not a crucial issue. However, with enterprises such as e-business, stock brokering, online telecommunications, and health systems, for instance, relevant information needs to be delivered as fast as possible to knowledge workers or decision systems which rely on it to react in a near real-time manner, according to the new and most recent data captured by the organization's information system [8]. *Active Data Warehousing* refers to a new trend where DWs are updated as frequently as possible, due to the high demands of users for fresh data. The term is also designated as *Real-Time Data Warehousing* (RTDW) for that reason in [24]. Today, business models and business processes take advantage of the latest technology in informatics and telecommunications, bringing the capability and the necessity of obtaining decision making information while the business transactions themselves are taking place. Nowadays, IT managers are facing crucial challenges deciding whether to build a real-time data warehouse instead of a conventional one, and whether their existing DW has become inadequate and needs to be converted into a real-time DW in order to remain competitive. In some specific cases, data update delays greater than a few seconds or minutes may jeopardize the usefulness of the whole system. This makes supporting RTDW a critical issue for such applications.

The term *Real-Time* in RTDW is a relative concept, because it is impossible to accomplish updating both operational source systems and OLAP databases at exactly the same moment in time. Usually, what is referred to as RTDW is the effort in trying to enable updating OLAP databases as soon as possible after business transactions are committed, minimizing the time gap between their actual occurrence and their propagation into the DW. This allows DW users or tools to take decisions and react actively to a set of business transactions which might have occurred in a very recent past, contrarily to what they could do if they were using traditional DWs. In our opinion, the main goal for changing the data update policy of a DW from periodical batch load updates towards a continuous data integration fashion is to deliver decision making information to OLAP and/or business intelligence tools, enabling them to react while the transaction itself is occurring. If this can be accomplished, we consider that the DW is capable of delivering decision making information in *Useful-Time*. Therefore, the main issue in useful-time data warehousing is: how can relevant information for aiding decision making be delivered to OLAP end users and tools in order to enable bringing business intelligence to the transaction itself?

The DW's data area is updated by Extraction, Transformation and Loading (ETL) tools. ETL tools are responsible for identifying and extracting relevant data from the operational source systems, customizing and integrating it into a common format, cleaning and conforming it into an adequate format for updating the DW data area and, finally, loading the final formatted data into its database. The efficiency and feasibility of the methods used in this last step of ETL are crucial for enabling useful-time data warehousing.

In a nutshell, accomplishing near zero latency between OLTP and OLAP systems consists in insuring data integration in a near continuous fashion from the former type of systems into the last. In order to make this feasible, several issues need to be taken under consideration: (1) Operational OLTP systems are designed to meet well-specified (short) response time requirements aiming for maximum system availability, which means that a RTDW scenario would have to cope with this in the overhead implied in OLTP systems; (2) The tables existing in a data warehouse's database directly related with transactional records (commonly named as fact tables) are usually huge in size, and therefore, the addition of new data and consequent procedures such as index updating or referential integrity checks would certainly have impact in OLAP systems' performance and data availability. Our work is focused on the DW perspective, for that reason we present an efficient methodology for continuous data integration, performing the ETL loading process.

This paper presents a solution which enables efficient continuous data integration in data warehouses, while allowing OLAP execution simultaneously, with minimum decrease of performance. With this, we seek to minimize the delay between the recording of transactional information and its reflected update on the decision support database. If the operational data extraction and transformation are able of performing

without significant delay during the transaction itself, this solution will load all the decision support information needed in the DW in useful time, allowing an answer while the transaction is still occurring. This is the general concept of useful-time data warehousing. The issues focused in this paper concern the DW end of the system, referring how to perform *loading* processes of ETL procedures and the DW's data area usage. The items concerning *extracting* and *transforming* of operational (OLTP) source systems data are not the focus of this paper. Therefore, we shall always be referring the data warehouse point of view, in the remainder of the paper.

The remainder of this paper is organized as follows. In section 2, we refer the requirements for useful-time data warehousing. Section 3 presents background and related work in real-time data warehousing. Section 4 presents the best data loading methods for actual data warehouses. Section 5 explains our continuous data integration methodology, along with its experimental evaluation, demonstrating its functionality. The final section contains concluding remarks and future work.

## 2 Requirements for Useful-Time Data Warehousing

Nowadays, organizations generate large amounts of data, at a rate which can easily reach several megabytes or gigabytes per day. Therefore, as time goes by, a business DW can easily grow to terabytes or even petabytes in size. The size of DWs implies that each decision support query which is executed usually accesses large amounts of records, performing actions such as joins, sorting, aggregating and calculation functions. To optimize these accesses – and, consequently, their OLAP performance – the DW uses predefined data structures (such as indexes or partitions, for instance), which are also large in size and have a considerably high level of complexity. These facts imply that it is very difficult to efficiently update the DW's data area in useful-time, for the propagation of data near real-time would most likely overload the server, given its update frequency and volume; it would involve immense complex operations on the DW's data structures and dramatically degrade OLAP performance.

Our aim is transforming a standard DW using batch loading during update windows (during which OLAP is not allowed) into a near zero latency analytical environment providing current data. The business requirements for this kind of analytical environment introduce a set of service level agreements that go beyond what is typical in a traditional DW. The major issue is how to enable continuous fashion data integration assuring that it minimizes negative impact in several main features of the system, such as availability and response time of both OLTP and OLAP systems.

An in-depth discussion of these features from the analytical point of view (to enable timely consistent analysis) is given in [6]. Combining highly available systems with active decision engines allows near real-time information dissemination for DWs. Cumulatively, this is the basis for zero latency analytical

environments [6]. The useful-time data warehouse provides access to an accurate, integrated, consolidated view of the organization's information and helps to deliver near real-time information to its users. This requires efficient ETL techniques enabling continuous fashion data integration, the focus of this paper.

By adopting useful-time data warehousing, it becomes necessary to cope with at least two radical data state changes. First, it is necessary to perform continuous data update actions, due to the continuous data integration, which should mostly concern row insertions. Second, these update actions must be performed in parallel with the execution of OLAP, which – due to its new near real-time nature – will probably be solicited more often. Therefore, the main contributions of this paper are threefold:

- Maximizing the freshness of data by efficiently and rapidly integrating the most recent decision making data, obtained from OLTP, into the data warehouse in useful-time;
- Minimizing OLAP response time while simultaneously integrating data in a continuous fashion;
- Maximizing the data warehouse's availability by reducing its update time window, in which users and OLAP applications are off-line.

## 3 Related Work

So far, research has mostly dealt with the problem of maintaining the DW in its traditional periodical update setup [14, 27]. Related literature presents tools and algorithms to populate the DW in an off-line fashion. In a different line of research, data streams [1, 2, 15, 20, 31, 32] could possibly appear as a potential solution. However, research in data streams has focused on topics concerning the front-end, such as on-the-fly computation of queries without a systematic treatment of the issues raised at the back-end of a DW [10]. Much of the recent work dedicated to RTDW is also focused on conceptual ETL modelling [4, 5, 19, 23, 33, 34], lacking the presentation of concrete specific extraction, transformation and loading algorithms along with their consequent OLTP and OLAP performance issues. Data warehouse operational processes are normally composed by a labor-intensive workflow, involving data extraction, transformation, integration, cleaning and transportation. To deal with this workflow, specialized tools are available today in the market [36, 37, 38, 39], under the general title Extraction – Transformation – Loading (ETL) tools. To give a general idea of these tools' functionality we refer their most prominent tasks, which include (a) the identification of relevant information at the source, (b) the extraction of that information, (c) the customization and integration of the information coming from multiple sources into a common format, (d) the cleaning of the resulting data set on the basis of database and business rules, and (e) the propagation of the data to the DW and/or data marts [35].

Temporal data warehouses address the issue of supporting temporal information efficiently in data warehousing systems [25]. In [27], the authors present efficient techniques (e.g. temporal view self-

maintenance) for maintaining data warehouses without disturbing source operations. A related challenge is supporting large-scale temporal aggregation operations in DWs [26]. In [4], the authors describe an approach which clearly separates the DW refreshment process from its traditional handling as a view maintenance or bulk loading process. They provide a conceptual model of the process, which is treated as a composite workflow, but they do not describe how to efficiently propagate the date. Theodoratus et al. discuss in [21] data currency quality factors in DWs and propose a DW design that considers these factors.

An important issue for near real-time data integration is the accommodation of delays, which has been investigated for (business) transactions in temporal active databases [18]. The conclusion is that temporal faithfulness for transactions has to be provided, which preserves the serialization order of a set of business transactions. Although possibly lagging behind real-time, a system that behaves in a temporally faithful manner guarantees the expected serialization order.

In [23], the authors describe the ARKTOS ETL tool, capable of modeling and executing practical ETL scenarios by providing explicit primitives for capturing common tasks (such as data cleaning, scheduling and data transformations) using a declarative language. ARKTOS offers graphical and declarative features for defining DW transformations and tries to optimize the execution of complex sequences of transformation and cleansing tasks.

In [13] is described a zero-delay DW with Gong, which assists in providing confidence in the data available to every branch of the organization. Gong is a Tecco product [3] that offers uni or bi-directional replication of data between homogeneous and/or heterogeneous distributed databases. Gong's database replication enables zero-delay business to assist in the daily running and decision making of the organization.

But not all transactional information needs to be immediately dealt with in real-time decision making requirements. We can define which groups of data are more important to include rapidly in the DW and other groups of data which can be updated in latter time. Recently, in [9], the authors present an interesting architecture on how to define the types of update and time priorities (immediate, at specific time intervals or only on DW offline updates) and respective synchronization for each group of transactional data items.

## 4 Searching for the Most Efficient Data Loading Procedures

The maturity of today's DBMS shows the evolution of all kinds of data management routines. Along with hardware evolution, we can manage greater amounts of data, increasingly faster. Their performance leads us to assume that today they may be able to deal with continuous fashion data integration in DWs. One of the primary features to consider is searching for the most efficient data loading procedures, whether

OLAP is performed or not. In this section, we focus on determining the actual fastest and securest way of getting data into the data warehouse, without jeopardizing data quality, demonstrating which are the most efficient data loading methods. To do this, we will consider the set of most common and available data loading techniques for most of the used DBMS, such as Oracle, MySQL, Postgres, SQL Server, etc.

Nowadays, DBMS supply standard bulk loading utilities for batch loading data. Oracle has the SQL*Loader [16] for importing data directly from outside the database itself. This approach is useful in traditional periodically updated data warehouses, in which the data to load usually represents a large number of rows. In these cases, OLTP data is usually extracted, cleaned, transformed and saved into a flat text-file format which will be loaded into the DW later on by its DBMS bulk loader, such as SQL*Loader. Another option is loading data using standard SQL statements or stored procedures through middleware drivers, such as ODBC. They can execute with higher or lower frequency, according to the needed update frequency in the DW. We also need to consider if the OLTP systems use the same DBMS as the DWs.

Therefore, we aim to test which of the data loading methods is the most efficient:

1) DBMS Bulk data loaders such as SQL*Loader, using flat-text files;
2) Insertion from outside the DBMS using SQL statements and/or stored procedures executed through middleware drivers, such as ODBC; or
3) Insertion from inside the DBMS using SQL statements and/or stored procedures without needing middleware drivers.

To perform these tests, we used the TPC-H decision support benchmark [22] and created a 25 Gigabyte sized TPC-H database with the Oracle 10g DBMS [16], on an Intel Pentium IV 2.8GHz CPU with 1 Gigabyte of 400 MHz RAM memory and a IDE UDMA-133 7200rpm 180 Gigabyte hard drive. The purpose of this database is to support business referring to customer orders for diverse articles. There are two tables that mainly support every new business transaction: table `Orders`, which saves every new customer order; and table `LineItem`, which saves all ordered items for each customer order. To test data integration in this database, we simulate a scenario which considers that it must be able to deal with loading 1.000.000 transactions, in which each transaction is composed by 1 row for the `Orders` table and an average of 4 rows for the `LineItem` table. We assume that this amount of data, characterized in Table 1, represents 1 day (24 hours) of the transactional operational systems work.

**Table 1.** Characterization of the data used for the experimental testing

| Table | Number of Rows to load | Data Size to load |
|---|---|---|
| Orders | 1.000.000 | 120 Mbytes |
| LineItem | 4.000.000 | 435 Mbytes |
| Total | 5.000.000 | 555 Mbytes |

### 4.1 Loading Data as a One-Step Batch Load Procedure

As we have mentioned before, the integration of new data in DWs can be done using one of two possible data update policies: executing periodical data updates, in a traditional DW fashion; or in a continuously near real-time data integration fashion, for a useful-time data warehousing mode. In the first case, the new decision support data to integrate in the DW is usually a considerable amount of data, easily reaching dozens or hundreds of megabytes or more, concerning all operational transactions occurred since the last DW update was performed. This update is performed while the DW is offline to OLAP users and tools, meaning that the data must be fully integrated as soon as possible, in order to minimize this time window. All the data structures in the DW which are used for performance optimization, such as indexes, partitions, materialized views, etc, are typically rebuilt when this update is executed, regaining their maximum efficiency according to the data's nature and features. This allows the database to regain maximum processing speed once more.

In the continuous data integration policy used in useful-time data warehousing, there is no foresight on how to maintain efficiency of the performance optimization data structures, because the nature and features of the new data which is to be integrated is unknown. Therefore, it is obvious that the functionality of the databases is affected, implying a growing decrease of performance. Sooner or later, after a certain number of insertions, the performance becomes too poor to consider as acceptable. To regain performance it is necessary to rebuild performance optimization data structures, similar to what is done for the traditional data warehouse. Therefore, optimization of data loading when the DW databases are offline to its end users and OLAP tools is important for both traditional and useful-time data warehousing. Table 2 shows the results for inserting 1.000.000 complete transactions as soon as possible in the 25 Gbyte TPC-H database. The Oracle bulk loader, SQL*Loader, was used with optimized parameters according to know-how expertise referred in published articles by experts in this field [29, 30].

**Table 2.** Time (HH:MM:SS) spent inserting a load of 1.000.000 TPC-H transactions with each loading method

| | Table Orders | Table LineItem | Total |
|---|---|---|---|
| **SQL*Loader** | 00:03:01 | 00:23:14 | 00:26:15 |
| **ODBC SQL Insert** | 00:21:11 | 01:26:32 | 01:47:43 |
| **DBMS SQL Insert** | 00:02:16 | 00:08:19 | 00:10:35 |

Many authors [8, 12, 29] mention that using bulk loaders is the fastest way of getting data into the DW. Observing Table 2, we can state that this is true, if the data comes from outside the DBMS. However, if the data already lies within the DBMS, the standard SQL Insert statement outperforms the bulk loader. This is an important conclusion, which should be considered when designing useful-time DW ETL tools.
Since multi-tasking is available and more than one parallel instances of each data loading method can be executed at the same time, we also tested loading data with simultaneous execution of several data loading

instances. The experiments showed that executing several instances of bulk loaders at the same time for integrating data in the same tables is not efficient. Contrarily to what could be expected at first, every extra running instance of the bulk loader degraded the performance approximately 30%. This happens because SQL*Loader is trying to insert data in the same tablespace segments, causing immense data block locking problems and consequent database waits. Therefore, only one instance of the bulk loader should be used at a time, at least for each table to update. Tests with the other data loading methods revealed the same results, although with performance degrading approximately 10%.

### 4.2 Loading Data in a Continuous Data Integration Fashion

To integrate data in a continuous near real-time manner into the DW database, instead of having a considerably large amount of data to insert in a one-step batch load, we need to consider that the data arrives in small amounts, representing a business transaction, which needs to be rapidly stored. In the test scenario mentioned earlier, we want to load 1.000.000 complete transactions in 24 hours. This means that a complete transaction must be integrated and committed into the TPC-H database every 0,0864 seconds, during 24 hours. This is very different from loading a large amount of data in a bulk manner and committing the changes when all is completed, as shown in the prior sub-section. When integrating small amounts of new data, the new rows of each loaded transaction are committed in their respective tables. This means that the number of commits that needs to be executed and the frequency at which data arrives make it a new challenge.

Since the new data comes from the OLTP systems, which operate outside the DW database, it has been formerly extracted, cleaned and transformed. When it is ready to be integrated into the DW database, data loading can be done using one of the two following methods which were mentioned before: using a bulk loader recurring to a flat-text file containing the new data; or using ODBC SQL Insert procedures.

Regarding to Oracle SQL*Loader, the bulk loader needs to perform the following actions in order to complete a data loading procedure: create and open a new log file for writing the status of the loading procedure; access and open the flat-text source file which has the new data to load; read the data in the flat-text source file; write the data in the corresponding database table(s); commit the changes in the corresponding database table(s); close the flat-text source file; and write the results of the data load procedure in the log file and close it. As we mentioned before, loading 1.000.000 complete transactions in 24 hours imply the insertion of one complete transaction every 0,0864 seconds. After experimenting, we concluded that every execution of SQL*Loader took between 1,5 and 1,9 seconds for executing the actions mentioned above. For update frequencies lower than 2 seconds, the bulk loader is not efficient, for the log files show us that many records are not loaded into their tables because the operating system file operations cannot keep up with the time execution frequency. Therefore, to use SQL*Loader efficiently, it

needs to execute once every 2 seconds for loading 24 transactions each time, to load 1.000.000 complete transactions a day.

When using ODBC middleware for loading and committing each complete TPC-H transaction, we need to determine if it can assure integrating of 1.000.000 transactions within the 24 hour limit. To do this, we tested inserting the data with a various number of simultaneous loaders. The results of these tests are shown in Figure 1.

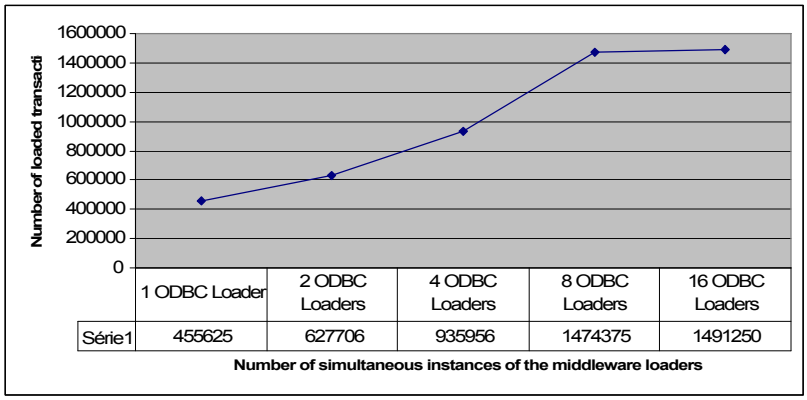



**Fig. 1.** ODBC insertion volume for running simultaneous data loading middleware instances

As seen in Figure 1, only one ODBC loader cannot insert 1.000.000 transactions in 24 hours. Using 8 simultaneous instances of the ODBC loader seems to be the most efficient method, because 16 simultaneous ODBC loaders increases the number of loaded transactions in an insignificant manner. Based on this, we can state that to accomplish loading 1.000.000 transactions in a day means that we can integrate a set of 8 complete transactions every 691 milliseconds using 8 simultaneous instances.

In conclusion, if we use the SQL*Loader bulk loader, we can integrate new data every 2 seconds, 24 complete transactions each time. If we use the ODBC middleware loader, we can integrate 8 new complete transactions each time, every 0,691 seconds. This is the closest to continuous data integration both the loading methods can get.

### 4.3 Loading Data in Large Tables vs. Loading Data in Small Tables

Since data is stored physically in tablespaces, the resources involved in the management of their segments can also interfere with data loading performance. The number of tablespace segments which needs to be managed for each table is proportional to the amount of data that lies in the table. Therefore, we also tested inserting 1.000.000 transactions into the original tables `Orders` and `LineItem` against inserting that same data into empty replicas of these tables, using the same procedures as shown is sub-section 4.1.

**Table 3.** Time (HH:MM:SS) inserting a single complete load of TPC-H 1.000.000 transactions using empty tables

| | Empty Orders Table Replica | Empty LineItem Table Replica | Total | Improvement |
|---|---|---|---|---|
| **SQL*Loader** | 00:01:13 | 00:06:05 | 00:07:18 | 72% less time |
| **ODBC SQL Insert** | 00:10:02 | 00:42:09 | 00:52:11 | 52% less time |
| **DBMS SQL Insert** | 00:00:55 | 00:04:55 | 00:05:50 | 45% less time |

Observing Table 3 and comparing the results with Table 2, we can state that loading new data into an empty table is significantly faster than loading it into a table which already has a considerable size. This should also be taken under account for designing useful-time data warehousing ETL. We shall now present our continuous data integration methodology, based on the conclusions illustrated in this section.

## 5. Useful-Time Data Warehouse Loading Methodology

Our methodology is focused on four major areas: (1) data warehouse schema adaptation; (2) ETL loading procedures; (3) OLAP query adaptation; and (4) DW database packing and reoptimization. It is mainly based on a very simple principle: new row insertion procedures in tables with few (or no) contents are performed much faster than in big size tables, as shown in the previous section of this paper. It is obvious and undeniable that data handling in small sized tables are much less complex and much faster than in large sized tables. As a matter of fact, this is mainly the reason why OLTP data sources are maintained with the fewer amount possible of necessary records, in order to maximize its availability. The proposed continuous data warehouse loading methodology is presented in Figure 2.

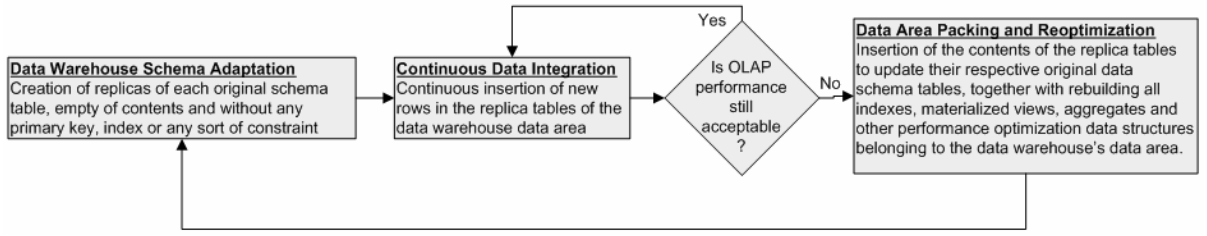



**Fig. 2.** General architecture of the proposed continuous data warehouse loading methodology

### 5.1 Adapting the Data Warehouse Schema

Suppose a very simple sales data warehouse with the schema illustrated in Figure 2, having two dimensional tables (`Store` and `Customer`, representing business descriptor entities) and one fact table (`Sales`, storing business measures aggregated from transactions). To simplify the figure, the Date dimension is not shown. This DW allows storing the sales value per store, per customer, per day. The primary keys are represented in bold, while the referential integrity constraints with foreign keys are represented in italic. The factual attribute `S_Value` is additive. This property in facts is very important for our methodology, as we shall demonstrate further on.

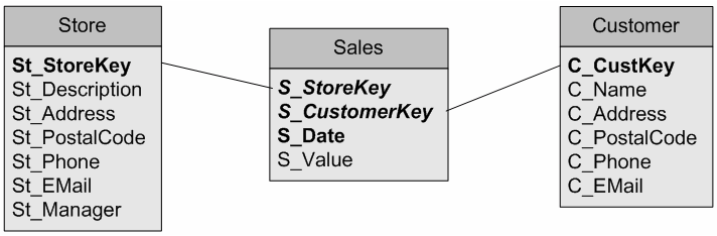



**Fig. 3.** Sample sales data warehouse schema

For the area concerning data warehouse schema, we adopt the following method:

**Data warehouse schema adaptation for supporting useful-time data warehousing:** Creation of an exact structural replica of all the tables of the data warehouse that could eventually receive new data. These tables (referred also as temporary tables) are to be created empty of contents, with no defined indexes, primary key, or constraints of any kind, including referential integrity. For each table, an extra attribute must be created, for storing a unique sequential identifier related to the insertion of each row within the temporary tables.

The modified sub-schema for supporting RTDW is illustrated in Figure 4. The unique sequential identifier attribute in each temporary table should record the sequence in which each row is appended in the database. This allows identifying the exact sequence of arrival for each new inserted row. This is useful for restoring prior data states in disaster recovery procedures, and also for discarding dimensional rows which have more recent updates. For instance, if the same customer has had two updates in the OLTP systems which, consequently, lead to the insertion of two new rows in the temporary table `CustomerTmp`, only the most recent one is relevant. This can be defined by considering as most recent the row with highest `CTmp_Counter` for that same customer (`CTmp_CustKey`).

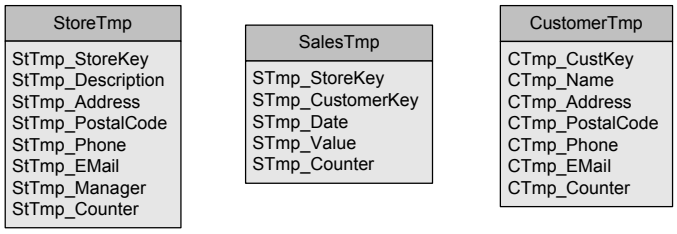



**Fig. 4.** Sample data warehouse sub-schema for supporting useful-time data warehousing

The authors of the ARKTOS tool [23] refer that their own experience, as well as the most recent literature, suggests that the main problems of ETL tools do not consist only in performance problems (as normally would be expected), but also in aspects such as complexity, practicability and price. By performing only record insertion procedures inherent to continuous data integration using empty or small sized tables without any kind of constraint or attached physical file related to it, we guarantee the simplest and fastest logical and physical support for achieving our goals [12].

The fact that the only significant change in the logical and physical structure of the data warehouse's schema is the simple adaptation shown in Figure 4, allows implementing ETL procedures in a manner to maximize its operationability. Data loading may be done by simple standard SQL instructions or DBMS batch loading software such as SQL*Loader [16], with a minimum of complexity. There is no need for developing complex routines for updating the data area, in which the needed data for is easily accessible, independently from the used ETL tools.

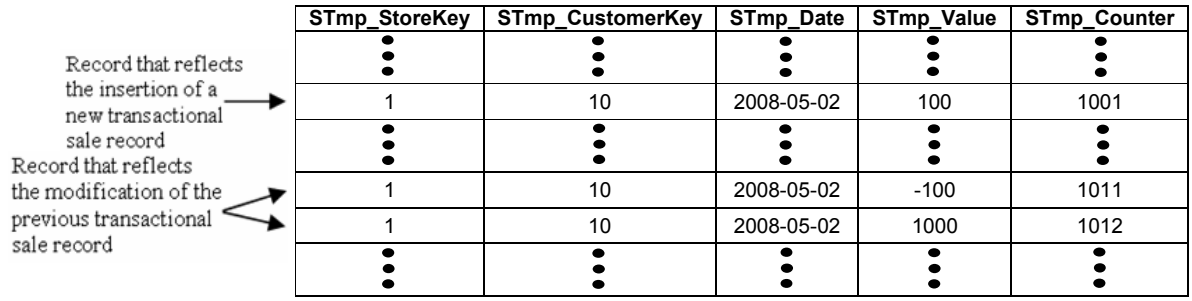
### 5.2 ETL Loading Procedures

To refresh the DW, once the ETL application has extracted and transformed the OLTP data into the correct format for loading the data area, it immediately proceeds in inserting that record as a new row in the correspondent temporary table, filling the unique sequential identifier attribute with the autoincremented sequential number. This number should start at 1 for the first record to insert in the DW after executing the packing and reoptimizing technique (explained in section 4.4 of this paper), and then be autoincremented by one unit for each record insertion. The algorithm for accomplishing continuous data integration by the ETL tool may be similar to:

```
Trigger for each new record in OLTP system (after it is commited)
    Extract new record from OLTP system
    Clean and transform the OLTP data, shaping it into the DW destination table's format
    Increment record insertion unique counter
    Create a new record in the data warehouse temporary destination table
    Insert the data in the temporary destination table's new record, along with the value of the record insertion
      unique counter
End_Trigger
```

Following, we demonstrate a practical example for explaining situations regarding updating the data warehouse shown in Figure 4. Figure 5 presents the insertion of a row in the data warehouse temporary fact table for the recording of a sales transaction of value 100 which took place at 2008-05-02 in store with `St_StoreKey` = 1 related to customer with `C_CustKey` = 10, identified by `STmp_Counter` = 1001. Meanwhile, other transactions occurred, and the organization's OLTP system recorded that instead of a value of 100 for the mentioned transaction, it should be 1000. The rows in the temporary fact table with `STmp_Counter` = 1011 and `STmp_Counter` = 1012 reflect this modification of values. The first eliminates the value of the initial transactional row and the second has the new real value, due to the additivity of the `STmp_Value` attribute. The definition of which attributes are additive and which are not should be the responsibility of the Database Administrator. According to [11], the most useful facts in a data warehouse are numeric and additive.

The method for data loading uses the most simple method for writing data: appending new records. Any other type of writing method needs the execution of more time consuming and complex tasks.




| STmp_StoreKey | STmp_CustomerKey | STmp_Date | STmp_Value | STmp_Counter |
|---|---|---|---|---|
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 1 | 10 | 2008-05-02 | 100 | 1001 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 1 | 10 | 2008-05-02 | -100 | 1011 |
| 1 | 10 | 2008-05-02 | 1000 | 1012 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

**Fig. 5.** Partial contents of temporary fact table SalesTmp with exemplification record insertions

### 5.3 OLAP Query Adaptation

Consider the following query, calculating the total revenue per store in the last 7 days:

```
SELECT S_StoreKey, Sum(S_Value) AS Last7DaysSaleVal
FROM Sales WHERE S_Date>=SystemDate()-7 GROUP BY S_StoreKey
```

To take advantage of method and include the most recent data in the OLAP query response, queries should be rewritten taking under consideration the following rule: *the FROM clause should join all rows from the required original and temporary tables with relevant data, excluding all fixed restriction predicate values from the WHERE clause whenever possible.* The modification for the prior instruction is illustrated below. It can be seen that the relevant rows from both issue tables are joined for supplying the OLAP query answer, filtering the rows used in the resulting dataset according to its restrictions in the original instruction.

```
SELECT S_StoreKey, Sum(S_Value) AS Last7DaysSaleVal
FROM (SELECT S_StoreKey, S_Value FROM Sales
             WHERE S_Date>=SystemDate()-7)
      UNION ALL
     (SELECT STmp_StoreKey, STmp_Value FROM SalesTmp
             WHERE STmp_Date>=SystemDate()-7)
GROUP BY S_StoreKey
```

An interesting and relevant aspect of the proposed methodology is that if OLAP users wish to query only the most recent information, they only need to do so against the temporary replicated tables. For instance, if the temporary tables are meant to be filled with data for each business day before they are recreated, and we want to know the sales value of the current day, per store, the adequate response could be obtained from the following SQL instruction:

```
SELECT STmp_StoreKey, Sum(STmp_Value) AS TodaysValue
FROM SalesTmp
WHERE STmp_Date=SystemDate()
GROUP BY STmp_StoreKey
```

This way, our method aids the processing of the data warehouse's most recent data, for this kind of data is stored within the temporary replica tables, which are presumed to be small in size. This minimizes CPU, memory and I/O costs involved in most recent data query processing. Theoretically, this would make it possible to deliver the most recent decision making information while the business transaction itself occurs.

### 5.4 Packing and Reoptimizing the Data Warehouse

Since the data is integrated within tables without optimization of any kind that could speed up querying, such as indexes, for instance, its functionality is affected, implying a decrease of performance. After a certain number of insertions the performance becomes too poor to consider as acceptable. To regain performance it is necessary to execute a pack routine which will update the original DW schema tables using the records in the temporary tables, and recreate them empty of contents, along with rebuilding the original tables' indexes and materialized views, so that maximum processing speed is obtained once more. For updating the original DW tables, the rows in the temporary tables should be aggregated according to the original tables' primary keys, maintaining the rows with highest unique counter attribute value for possible duplicate values in non-additive attributes, for they represent the most recent records. The time needed for executing these procedures represents the only period of time in which the DW in unavailable to OLAP tools and end users, for they need to be executed exclusively. The appropriate moment for doing this can be determined by the Database Administrator, or automatically, taking under consideration parameters such as a determined number of records in the temporary tables, the amount of physically occupied space, or yet a predefined period of time. The determination of this moment should consist on the best possible compromise related to its frequency of execution and the amount of time it takes away from all user availability, which depends on the physical, logical and practical characteristics inherent to each specific DW implementation itself and is not object of discussion in this paper.

### 5.5 Experimental Evaluation of the Methodology

In order to evaluate our methodology, we tested the execution of a set of TPC-H queries { Q1, Q2, Q4, Q6, Q11, Q12, Q13, Q14, Q16, Q22 } while continuously integrating data, aiming for the insertion of 1.000.000 transactions coming from outside the database within 24 hours, representing 555 Mbytes of data. Both data loading methods were experimented. Following what was mentioned in section 4 of this paper, we used one instance of SQL*Loader for loading 24 transactions each time, every 2 seconds. We also tested ODBC middleware data loading, inserting 8 simultaneous transactions at a time, every 0,691

seconds. We compare these results to the standard execution time of the mentioned TPC-H query workload, without performing continuous data integration. These results are shown in Table 4.

**Table 4.** Experimental evaluation of the continuous data integration methodology

| Standard OLAP exec. time (seconds) | OLAP exec. time in new schema *without data integration* | % of increase in exec. time | OLAP exec. time in new schema *using ODBC data loading* | % of increase in exec. time | OLAP exec. time in new schema *using bulk data loading* | % of increase in exec. time |
|---|---|---|---|---|---|---|
| 3842 | 4012 | 4,4 % | 10244 | 166,6 % | 4848 | 26,2 % |

As seen in Table 4, the extra time needed for executing the query workload when accessing and joining the temporary tables with the original ones led to an increase of 4,4 % of standard execution time (4012 – 3842 = 170 seconds). This represents an almost insignificant impact in OLAP response time. In what concerns the data loading methods, using ODBC middleware allows inserting data at a higher frequency than bulk loading, but the impact in OLAP response time for the first is much higher than for the last. The extra time needed for executing the query workload using the bulk loading method is 1006 seconds (4848 – 3842), representing an increase of 26,2 % of standard execution time, against 6402 seconds and increase of 166,6% of standard execution time for ODBC data loading, which is much worse. Therefore, using frequent bulk loading of a small set of complete transactions with predefined time intervals between executions which assure data quality loading seems to be the best method.

### 5.6 Final Remarks on Our Methodology

Notice that only record insertions are used for updating the DW for all related transactions in the OLTP source systems. Since this type of procedure does not require any record locking in the tables (except for the appended record itself) nor search operations for previously stored data before writing data (like in update or delete instructions), the time necessary to accomplish this is minimal. The issue of record locking is strongly enforced by the fact that the referred tables do not have any indexes or primary keys, implying absolutely no record locking, except for the appended record itself. Furthermore, since they do not have constraints of any sort, including referential integrity and primary keys, there is no need to execute time consuming tasks such as index updating or referential integrity cross checks. Kimball refers in [12] that many ETL tools use a *UPDATE ELSE INSERT* function in loading data, considering this as a performance killer. With our method, any appending, updating or eliminating data tasks on OLTP systems reflect themselves as only new record insertions in the data warehouse, which allows minimizing row, block and table locks and other concurrent data access problems. Physical database tablespace fragmentation is also avoided, once there is now deletion of data, only sequential increments. This allows us to state that the data update time window for our methods is minimal for the insertion of new data,

maximizing the availability of that data, and consequently contributing to effectively increase the data warehouses' global availability and minimize any negative impact in its performance.

Since we want to obtain data near real-time, the time gap between recording OLTP transactions and their extraction by ETL processes is minimal, occurring nearly at the same time, which somewhat reduces error probability. We can also assume that the amount of intermediate "information buckets" which the data passes through in the ETL Area is also minimal, for temporary storage is almost not needed. Furthermore, instead of extracting a considerable amount of OLTP data, which is what happens in "traditional" bulk loading, the volume of information extracted and transformed in real-time is extremely reduced (representing commonly few dozen bytes), since it consists of only one transaction per execution cycle. All this allows assuming that the extraction and transformation phase will be cleaner and more time efficient.

As a limitation to our methodology, DW contexts in which additive attributes are difficult or even not possible to define for their fact tables may invalidate its practice. It also does not consider the optimization of materialized views.

## 6. Conclusions and Future Work

This paper refers the necessary requirements for useful-time data warehousing, which imply the capability to deal with integrating data in the data warehouse in a continuous fashion. We demonstrate which are the best DW loading methods for each context and present a methodology for achieving useful-time data warehousing by enabling continuous data integration, while minimizing impact in query execution on the user end of the DW. This is achieved by data structure replication and adapting query instructions in order to take advantage of the new schemas, while executing the best previously determined continuous data integration methods. We have shown its functionality, recurring to a simulation using the TPC-H benchmark, performing continuous data integration against the execution of query workloads, for each used data warehouse loading method. All scenarios show that it is possible to achieve UTDW performance in exchange for an average increase of query execution time. This should be considered the price to pay for real-time capability within the DW.

As future work we intend to develop an ETL tool which will integrate this methodology with extraction and transformation routines for the OLTP systems. There is also room for optimizing the query instructions used for our methods, as well as including the problems referring to updating materialized views.

## References


1. D. J. Abadi, D. Carney, et al.: *"Aurora: A New Model and Architecture for Data Stream Management"*, The VLDB Journal, 12(2), pp. 120-139, 2003.
2. S. Babu, and J. Widom: *"Continuous Queries Over Data Streams"*, SIGMOD Record 30(3), pp. 109-120.
3. T. Binder: *Gong User Manual*, Tecco Software Entwicklung AG, 2003.
4. M. Bouzeghoub, F. Fabret, and M. Matulovic: *"Modeling Data Warehouse Refreshment Process as a Workflow Application"*, Int. Workshop on Design and Management of DW (DMDW), 1999.
5. R. M. Bruckner, B. List, and J. Schiefer: *"Striving Towards Near Real-Time Data Integration for Data Warehouses"*, Int. Conf. Data Warehousing and Knowledge Discovery (DAWAK), 2002.
6. R. M. Bruckner, and A. M. Tjoa: *"Capturing Delays and Valid Times in Data Warehouses – Towards Timely Consistent Analyses"*. Journal of Intelligent Inf. Systems (JIIS), 19:2, pp. 169-190, 2002.
7. S. Chaudhuri, and U. Dayal: *"An Overview of Data Warehousing and OLAP Technology"*, SIGMOD Record, Volume 26, Number 1, pp. 65-74, 1997.
8. W. H. Inmon, R. H. Terdeman, J. Norris-Montanari, and D. Meers: *Data Warehousing for E-Business*, J. Wiley & Sons, 2001.
9. C. Italiano, and J. E. Ferreira: *"Synchronization Options for Data Warehouse Designs"*, IEEE Computer Magazine, 2006.
10. Karakasidis, P. Vassiliadis, and E. Pitoura: *"ETL Queues for Active Data Warehousing"*, IQIS'05, 2005.
11. R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite: *The Data Warehouse Lifecycle Toolkit – Expert Methods for Designing, Developing and Deploying Data Warehouses*, Wiley Computer Pub, 1998.
12. R. Kimball, and J. Caserta: *The Data Warehouse ETL Toolkit*, Wiley Computer Publishing, 2004.
13. E. Kuhn: *"The Zero-Delay Data Warehouse: Mobilizing Heterogeneous Databases"*, International Conference on Very Large Data Bases (VLDB), 2003.
14. W. Labio, J. Yang, Y. Cui, H. Garcia-Molina, and J. Widom: *"Performance Issues in Incremental Warehouse Maintenance"*, Int. Conf. on Very Large Data Bases (VLDB), 2000.
15. D. Lomet, and J. Gehrke: *Special Issue on Data Stream Processing*, IEEE Data Eng. Bulletin, 26(1), 2003.
16. *Oracle Corporation*, 2005. www.oracle.com
17. T. B. Pedersen: *"How is BI Used in Industry?"*, Int. Conf. on Data Warehousing and Knowledge Discovery (DAWAK), 2004.
18. J. F. Roddick, and M. Schrefl: *"Towards an Accommodation of Delay in Temporal Active Databases"*, 11th Australasian Database Conference (ADC), 2000.
19. Simitsis, P. Vassiliadis and T. Sellis: *"Optimizing ETL Processes in Data Warehouses"*, Int. Conference on Data Engineering (ICDE), 2005.
20. U. Srivastava, and J. Widom: *"Flexible Time Management in Data Stream Systems"*, Int. Conf. on Principles of Database Systems (PODS), 2004.
21. D. Theodoratus, and M. Bouzeghoub: *"Data Currency Quality Factors in Data Warehouse Design"*, Int. Workshop on Design and Management of Data Warehouses (DMDW), 1999.
22. *TPC-H decision support benchmark*, Transaction Processing Council, www.tpc.com.
23. P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis: *"ARKTOS: Towards the Modelling, Design, Control and Execution of ETL Processes"*, Inf. Systems, Vol. 26(8), 2001.
24. White: *"Intelligent Business Strategies: Real-Time Data Warehousing Heats Up"*, DM Preview, www.dmreview.com/article_sub_cfm?articleId=5570, 2002.
25. J. Yang: *"Temporal Data Warehousing"*, Ph.D. Thesis, Dp. Computer Science, Stanford Univ, 2001.
26. J. Yang, and J. Widom: *"Incremental Computation and Maintenance of Temporal Aggregates"*, 17th Intern. Conference on Data Engineering (ICDE), 2001.
27. J. Yang, and J. Widom: *"Temporal View Self-Maintenance"*, 7th Int. Conf. Extending Database Technology (EDBT), 2001.
28. T. Zurek, and K. Kreplin: *"SAP Business Information Warehouse – From Data Warehousing to an E-Business Platform"*, 17th Int. Conf. on Data Engineering (ICDE), 2001.
29. D. Burleson: *Oracle data load (import, SQL*Loader) speed tips*, Burleson Consulting, http://www.dba-oracle.com/oracle_tips_load_speed.htm, 2006.
30. *Oracle Corporation*: http://www.oracle.com/technology/products/database/utilities/index.html, 2008.
31. N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, S. B. Zdonik: *"Towards a Streaming SQL Standard"*, Int. Conf. Very Large Data Bases (VLDB), 1(2): 1379-1390, 2008.

32. N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, N. Frantzell: *"Meshing Streaming Updates with Persistent Data in an Active Data Warehouse"*, IEEE Transactions on Knowl. Data Eng, 20(7): 976-991, 2008.
33. H. Agrawal, G. Chafle, S. Goyal, S. Mittal, S. Mukherjea: *"An Enhanced Extract-Transform-Load System for Migrating Data in Telecom Billing"*, Int. Conference on Data Engineering (ICDE), pp. 1277-1286, 2008.
34. A. Simitsis, P. Vassiliadis: *"A method for the mapping of conceptual designs to logical blueprints for ETL processes"*, Decision Support Systems (45) 22–40, 2008.
35. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, S. Skiadopoulos: *"A generic and customizable framework for the design of ETL scenarios"*, Information Systems, 30(7): 492-525, 2005.
36. *IBM*, IBM Data warehouse manager, available at http://www-3.ibm.com/software/data/db2/datawarehouse/.
37. *Informatica*, Power Center, available at http://www.informatica.com/products/data+integration/powercenter/default.htm.
38. *Microsoft*, Data transformation services, available at http://www.microsoft.com.
39. *Oracle Corporation*, "Oracle warehouse builder product page", available at http://otn.oracle.com/products/warehouse/content.html.