Real-Time Data Warehouse Loading Methodology

Ricardo Jorge Santos
CISUC – Centre of Informatics and Systems
DEI – FCT – University of Coimbra
Coimbra, Portugal
lionsoftware.ricardo@gmail.com

Jorge Bernardino
CISUC, IPC – Polytechnic Institute of Coimbra
ISEC – Superior Institute of Engineering of Coimbra
Coimbra, Portugal
jorge@isec.pt

ABSTRACT
A data warehouse provides information for analytical processing, decision making and data mining tools. As the concept of real-time enterprise evolves, the synchronism between transactional data and data warehouses, statically implemented, has been redefined. Traditional data warehouse systems have static structures of their schemas and relationships between data, and therefore are not able to support any dynamics in their structure and content. Their data is only periodically updated because they are not prepared for continuous data integration. For real-time enterprises with needs in decision support purposes, real-time data warehouses seem to be very promising. In this paper we present a methodology on how to adapt data warehouse schemas and user-end OLAP queries for efficiently supporting real-time data integration. To accomplish this, we use techniques such as table structure replication and query predicate restrictions for selecting data, to enable continuously loading data in the data warehouse with minimum impact in query execution time. We demonstrate the efficiency of the method by analyzing its impact in query performance using benchmark TPC-H executing query workloads while simultaneously performing continuous data integration at various insertion time rates.

Keywords
real-time and active data warehousing, continuous data integration for data warehousing, data warehouse refreshment loading process.

1. INTRODUCTION
A data warehouse (DW) provides information for analytical processing, decision making and data mining tools. A DW collects data from multiple heterogeneous operational source systems (OLTP – On-Line Transaction Processing) and stores summarized integrated business data in a central repository used by analytical applications (OLAP – On-Line Analytical Processing) with different user requirements. The data area of a data warehouse usually stores the complete history of a business. The common process for obtaining decision making information is based on using OLAP tools [7]. These tools have their data source based on the DW data area, in which records are updated by ETL (Extraction, Transformation and Loading) tools. The ETL processes are responsible for identifying and extracting the relevant data from the OLTP source systems, customizing and integrating this data into a common format, cleaning the data and conforming it into an adequate integrated format for updating the data area of the DW and, finally, loading the final formatted data into its database.

Traditionally, it has been well accepted that data warehouse databases are updated periodically – typically in a daily, weekly or even monthly basis [28] – implying that its data is never up-to-date, for OLTP records saved between those updates are not included the data area. This implies that the most recent operational records are not included into the data area, thus getting excluded from the results supplied by OLAP tools. Until recently, using periodically updated data 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 who rely on it to react in a near real-time manner, according to the new and most recent data captured by an organization's information system [8]. This makes supporting near real-time data warehousing (RTDW) a critical issue for such applications.

The demand for fresh data in data warehouses has always been a strong desideratum. Data warehouse refreshment (integration of new data) is traditionally performed in an off-line fashion. This means that while processes for updating the data area are executed, OLAP users and applications cannot access any data. This set of activities usually takes place in a preset loading time window, to avoid overloading the operational OLTP source systems with the extra workload of this workflow. Still, users are pushing for higher levels of freshness, since more and more enterprises operate in a business time schedule of 24x7. *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* for that reason in [24]. The conclusions presented by T. B. Pedersen in a report from a knowledge exchange network formed by several major technological partners in Denmark [17] refer that all partners agree real-time enterprise and continuous data availability is considered a short term priority for many business and general data-based enterprises. 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 data warehouse is going out of style and needs to be converted into a real-time data warehouse to remain competitive. In some specific cases, data update delays larger than a few seconds or minutes may jeopardise the usefulness of the whole system.

IDEAS'08, September 10–12, 2008, Coimbra, Portugal
Editor: Bipin C. DESAI

In a nutshell, accomplishing near zero latency between OLTP and OLAP systems consists in insuring continuous data integration 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 those 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 data warehouse in useful time, allowing an answer while the transaction still is occurring. This is the general concept of real 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 for efficiently supporting continuous data integration. 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.

Based on the existing schema(s) of the DW's database, our methodology consists on creating a replica for each of its tables, empty of contents and without defining any type of indexes, primary keys or any other kind of restrictions or constraints. In this new schema, the replicated tables will receive and record the data from the staging area, which will be continuously loaded. The fact that they are initially created empty of contents and not having any sort of constraints allows to considerably minimize the consumption of time and resources which are necessary in the procedures inherent to data integration. As the data integration is performed, the database's performance which includes the new most recent data deteriorates, due to the lack of usual data structures that could optimize it (such as indexes, for example), in the replicated tables. When the data warehouse's performance is considered as unacceptable by its users or administrators, the existing data within the replicated tables should serve for updating the original schema. After performing this update, the replicated tables are to be recreated empty of contents, regaining maximum performance. We also demonstrate how to adapt OLAP queries in the new schema, to take advantage of the most recent data which is integrated in real time.

The experiments which are presented were performed using a standard benchmark for decision support systems, benchmark TPC-H, from the Transaction Processing Council [22]. Several configurations were tested, varying items such as the database's physical size, the number of simultaneous users executing OLAP queries, the amount of available RAM memory and the time rates between transactions.

The remainder of this paper is organized as follows. In section 2, we refer the requirements for real-time data warehousing. Section 3 presents background and related work in real-time data warehousing. Section 4 explains our methodology, and in section 5 we present the experimental evaluation of our methods and demonstrate its functionality. The final section contains concluding remarks and future work

2. REQUIREMENTS FOR REAL-TIME DATA WAREHSOUSING

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 data warehouse can easily grow to terabytes or even petabytes. The size of data warehouses imply that each query which is executed against its data area usually accesses large amounts of records, also performing actions such as joins, sorting, grouping and calculation functions. To optimize these accesses, the data warehouse uses predefined internal data structures (such as indexes or partitions, for instance), which are also large in size and have a very considerable level of complexity. These facts imply that it is very difficult to efficiently update the data warehouse's data area in real-time, for the propagation of transactional data in real-time would most likely overload the server, given its update frequency and volume; it would involve immense complex operations on the data warehouse's data structures and dramatically degrade OLAP performance.

In a nutshell, real-time data warehouses aim for decreasing the time it takes to make decisions and try to attain zero latency between cause and effect for that decision, closing the gap between intelligent reactive systems and systems processes. Our aim is transforming a standard DW using batch loading during update windows (during which analytical access is not allowed) into near zero latency analytical environment providing current data, in order to enable (near) real-time dissemination of new information across an organization. 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 data integration and 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 data warehouses. Cumulatively, this is the basis for zero latency analytical environments [6]. The real-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 data integration, the focus of this paper.

By adopting real-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 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 most recent OLTP data into the data warehouse;
- Minimizing OLAP response time while simultaneously performing continuous data integration;
- 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 warehouse in its traditional periodical update setup [14, 27]. Related literature presents tools and algorithms to populate the warehouse in an off-line fashion. In a different line of research, data streams [1, 2, 15, 20] 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 data warehouse [10]. Much of the recent work dedicated to RTDW is also focused on conceptual ETL modelling [4, 5, 19, 23], lacking the presentation of concrete specific extraction, transformation and loading algorithms along with their consequent OLTP and OLAP performance issues.

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 data warehouses [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 data warehouses 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 in order to assist in the daily running and decision making of the organization.

But not all transactional informations need to be immediately dealt with in real-time decision making requirements. We can define which groups of data is more important to include rapidly in the data warehouse 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 data warehouse offline updates) and respective synchronization for each group of transactional data items.

Our methodology overcomes some of the mentioned drawbacks and is presented in the next section.

4. CONTINUOUS DATA WAREHOUSE LOADING METHODOLOGY

From the DW side, updating huge tables and related structures (such as indexes, materialized views and other integrated components) makes executing OLAP query workloads simultaneously with continuous data integration a very difficult task. Our methodology minimizes the processing time and workload required for these update processes. It also facilitates the DW off-line update (see section 4.4), because the data already lies within the data area and all OLTP data extraction and/or transformation routines have been executed during the continuous data integration. Furthermore, the data structure of the replicated tables is exactly the same as the original DW schema. This minimizes the time window for packing the data area of the DW, since its update represents a one step process by resuming itself as a cut-and-paste action from the temporary tables to the original ones, as we shall demonstrate further.

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. 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 1.

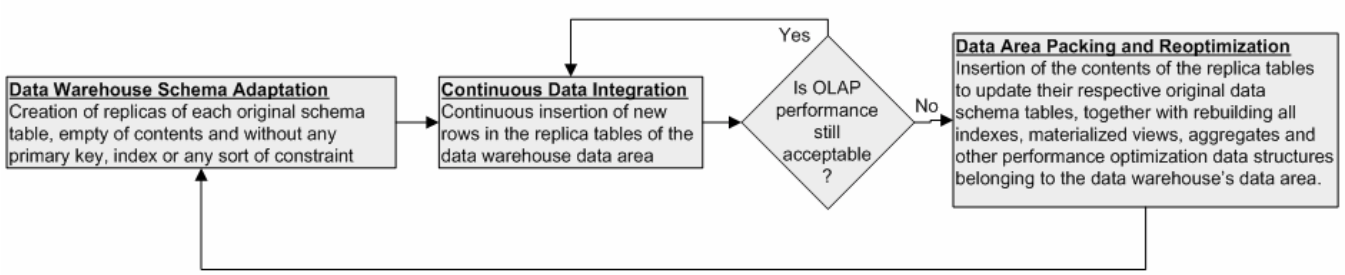


Figure 1. General architecture of the proposed continuous data warehouse loading methodology.

4.1 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.

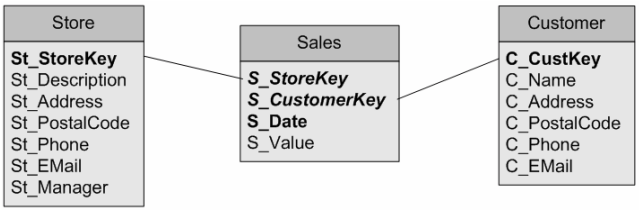


Figure 2. Sample sales data warehouse schema.

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

> **Data warehouse schema adaptation method for supporting real-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 schema for supporting RTDW based on our methodology is illustrated in Figure 3. The unique sequential identifier attribute present in each temporary table should record the sequence in which each row is appended in the Data Area. This will allow 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`).

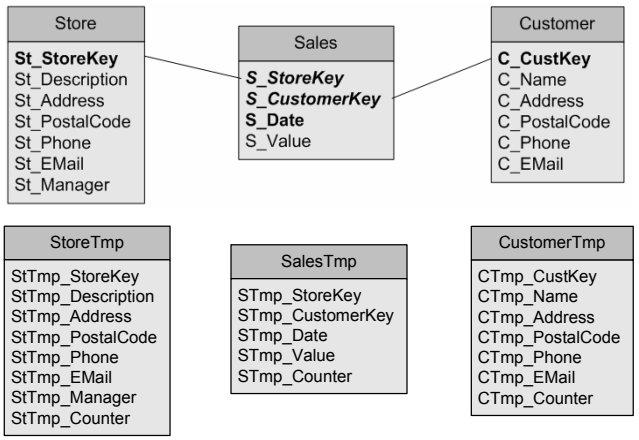


Figure 3. Sample sales data warehouse schema modified for supporting real-time data warehousing.

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, practibility 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 3, allows that the implementation of the necessary ETL procedures can be made 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.

4.2 ETL Loading Procedures

To refresh the data warehouse, once the ETL application has extracted and transformed the OLTP data into the correct format for loading the data area of the DW, it shall proceed immediately 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 data warehouse 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 3. Figure 4 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
	⋮	⋮	⋮	⋮	⋮
Record that reflects the insertion of a new transactional sale record →	1	10	2008-05-02	100	1001
	⋮	⋮	⋮	⋮	⋮
Record that reflects the modification of the previous transactional sale record →	1	10	2008-05-02	-100	1011
	1	10	2008-05-02	1000	1012
	⋮	⋮	⋮	⋮	⋮

Figure 4. Partial contents of the temporary fact table SalesTmp with exemplification of record insertions.

4.3 OLAP Query Adaptation

Consider the following OLAP query, for calculating the total revenue per store in the last seven days.

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

To take advantage of our schema modification 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, with respect to our methodology. 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.

4.4 Packing and Reoptimizing the Data Warehouse

Since the data is being integrated within tables that do not have access optimization of any kind that could speed up querying, such as indexes, for instance, it is obvious that its functionality is affected, implying a decrease of performance. Due to the size of physically occupied space, after a certain number of insertions the performance becomes too poor to consider as acceptable. To regain performance in the DW 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 these temporary tables 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 update 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 may 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.

4.5 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.

As mentioned earlier, the extracted data usually needs to be corrected and transformed before updating the data warehouse's data area. Since we pretend to obtain this 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 the "traditional" bulk loading data warehousing, 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, data warehouse contexts in which additive attributes are difficult or even not possible to define for their fact tables may invalidate its practice.

5. EXPERIMENTAL EVALUATION

Recurring to TPC-H decision support benchmark (TPC-H) we tested our methodology, creating 5GB, 10GB and 20GB sized DWs for continuous data integration at several time rates, in ORACLE 10g DBMS [16]. We used an Intel Celeron 2.8GHz with variations of 512MB, 1GB and 2GB of SDRAM and a 7200rpm 160GB hard disk. The modified TPC-H schema according to our methodology can be seen in Figure 5. Tables `Region` and `Nation` are not included as temporary tables because they are fixed-size tables in the TPC-H benchmark, and therefore do not receive new data. For the query workloads we selected TPC-H queries 1, 8, 12 and 20 (TPC-H), executed in random order for each simultaneous user in the tests, for up to 8 hours of testing for each scenario.

The summarized results which are presented in this section correspond to a total amount of approximately 4000 hours of testing.

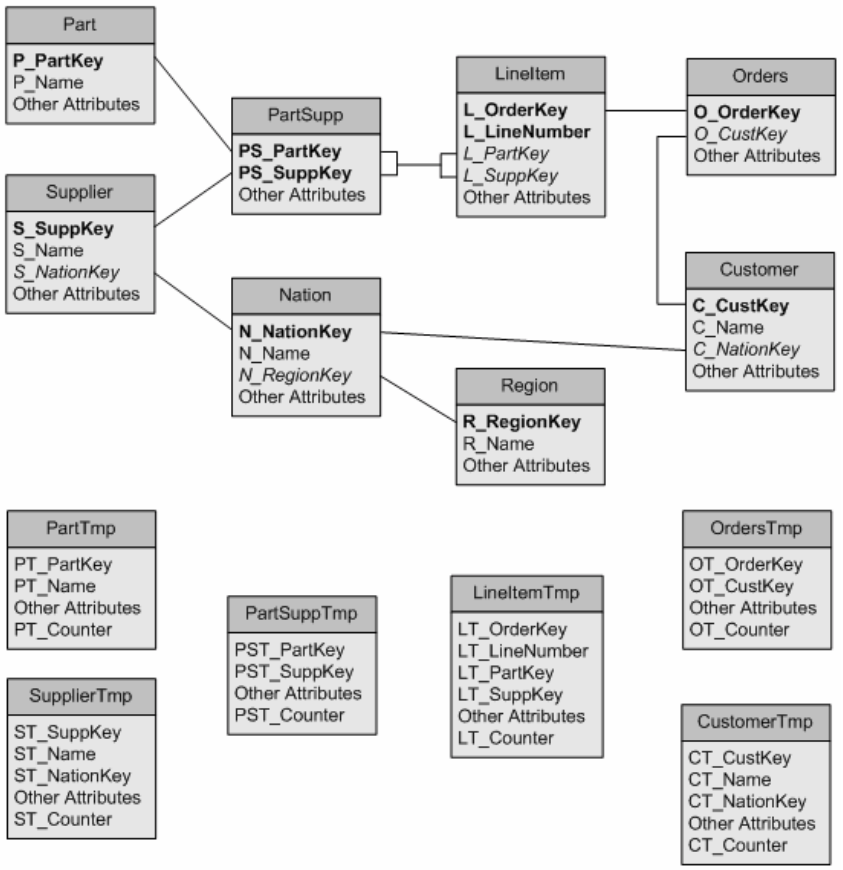


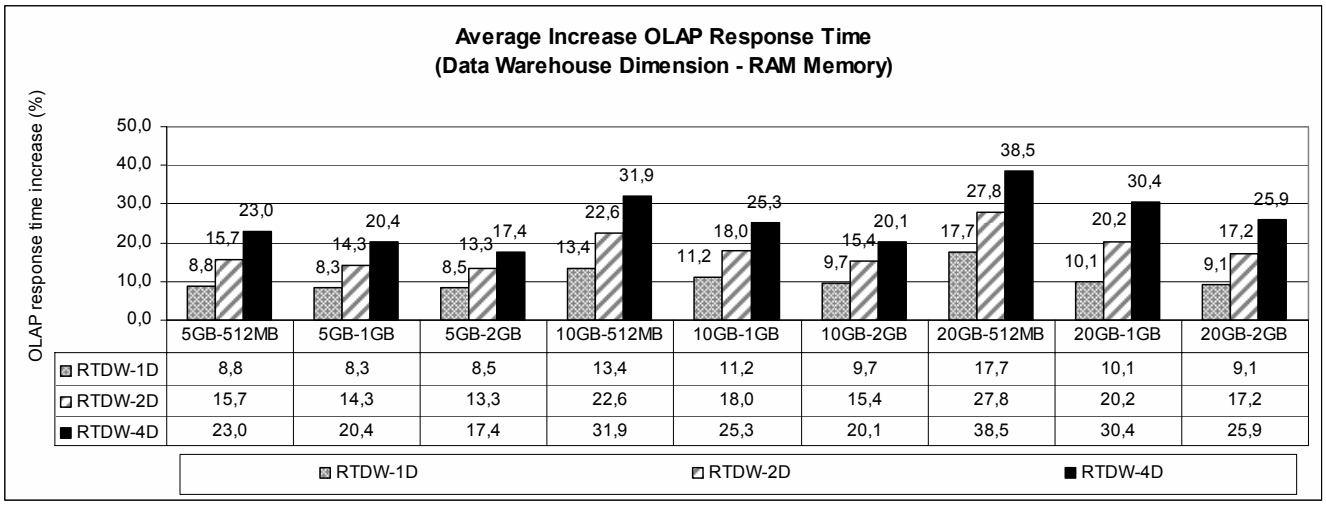
Figure 5. TPC-H data warehouse schema modified for supporting real-time data warehousing.

The definition of the number of transactions, average time interval between transactions, number of records and data size to be inserted for each scenario number of records are illustrated in Table 1. Each new transaction represents the insertion of an average of four records in `LineItemTmp` and one row in each of the other temporary tables, continuously being integrated for a period of 8 hours. For each database size, we also tested three different transaction insertion time interval frequencies, referenced as 1D, 2D and 4D, each corresponding to a different amount of records to be inserted throughout the tests.

The impact in OLAP processing while performing the continuous data integration is shown in Figures 6 and 7. Table 2 resumes the results for each scenario. All results show that the system is significantly dependable on the transaction rates, performing better if more RAM is used. They present a minimal cost of around 8% of query response time for practicing our methodology, independently from the amount of RAM, for a transactional frequency of inserting 1 transaction every 9,38 seconds. Apparently, this is the lowest price to pay for having real time data warehousing with our methodology, for the tested benchmark. At the highest transactional frequency, 1 every 0,57 seconds, the increase of OLAP response time climbs up to a maximum of 38,5% with 512MB of RAM memory. The results also show that the methodology is also relatively scalable in what concerns database size.

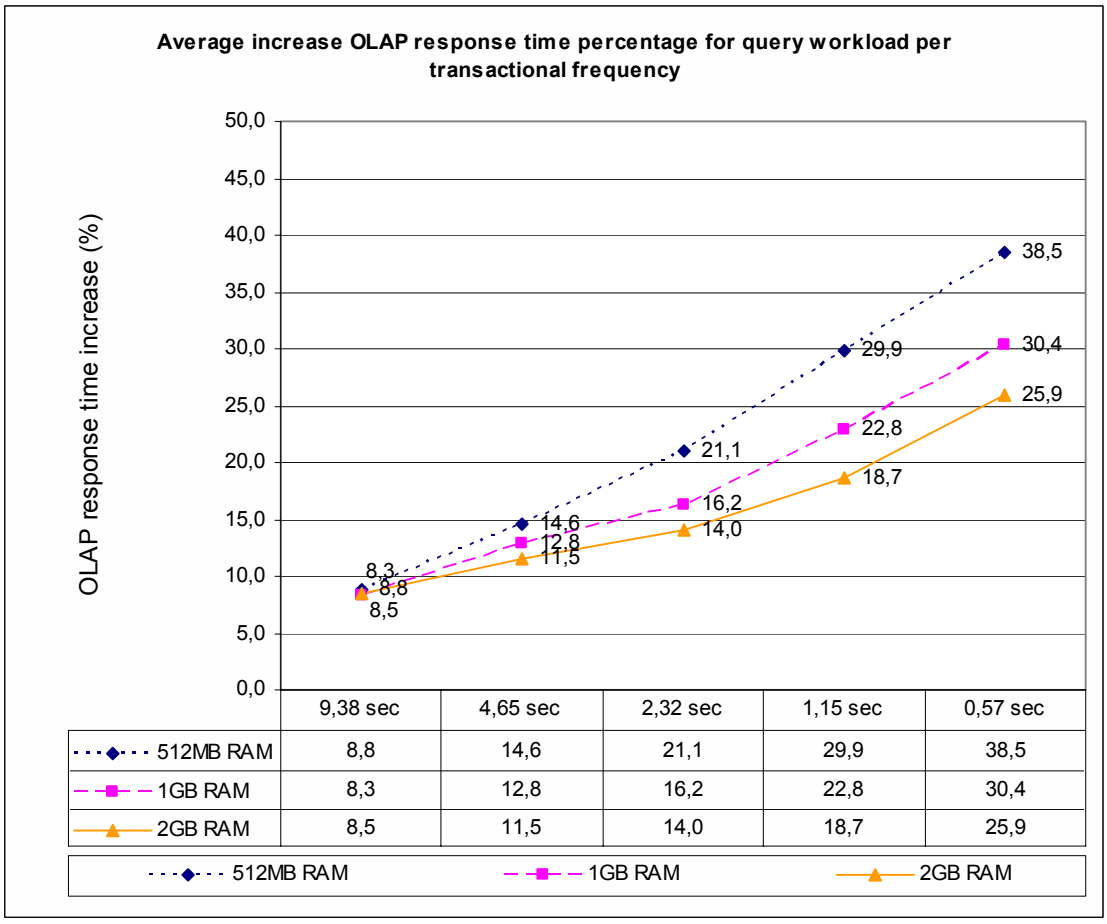
Table 1. Characterization of the experimental evaluation scenarios.

Scenario	Nr. Transactions to load in 8 Hours	Average Time between Transactions	N.º Records to integrate in 8 Hours	Total Size of RTDW Transactions
DW5GB-1D	3.072	9,38 seconds	54.040	4,59 MBytes
DW5GB-2D	6.205	4,64 seconds	106.302	9,28 MBytes
DW5GB-4D	12.453	2,31 seconds	204.904	18,62 MBytes
DW10GB-1D	6.192	4,65 seconds	109.464	9,26 MBytes
DW10GB-2D	12.592	2,29 seconds	213.912	18,83 MBytes
DW10GB-4D	25.067	1,15 seconds	410.764	37,48 MBytes
DW20GB-1D	12.416	2,32 seconds	218.634	18,57 MBytes
DW20GB-2D	25.062	1,15 seconds	429.214	37,48 MBytes
DW20GB-4D	50.237	0,57 seconds	825.242	75,12 MBytes



	5GB-512MB	5GB-1GB	5GB-2GB	10GB-512MB	10GB-1GB	10GB-2GB	20GB-512MB	20GB-1GB	20GB-2GB
RTDW-1D	8,8	8,3	8,5	13,4	11,2	9,7	17,7	10,1	9,1
RTDW-2D	15,7	14,3	13,3	22,6	18,0	15,4	27,8	20,2	17,2
RTDW-4D	23,0	20,4	17,4	31,9	25,3	20,1	38,5	30,4	25,9

Figure 6. Average increase of the OLAP response time (Data Warehouse dimension – RAM memory).



	9,38 sec	4,65 sec	2,32 sec	1,15 sec	0,57 sec
512MB RAM	8,8	14,6	21,1	29,9	38,5
1GB RAM	8,3	12,8	16,2	22,8	30,4
2GB RAM	8,5	11,5	14,0	18,7	25,9

Figure 7. Average increase of the OLAP response time per transactional frequency.

Table 2. Characterization of the average increase of OLAP response times for each experimental scenario.

	Data Warehouse 5 GBytes			Data Warehouse 10 GBytes			Data Warehouse 20 GBytes		
	512 MBytes	1 GBytes	2 GBytes	512 MBytes	1 Gbyte	2 GBytes	512 MBytes	1 GByte	2 GBytes
9,38 seconds	8,8%	8,3%	8,5%	nt	nt	nt	nt	nt	nt
4,65 seconds	15,7%	14,3%	13,3%	13,4%	11,2%	9,7%	nt	nt	nt
2,32 seconds	23,0%	20,4%	17,4%	22,6%	18,0%	15,4%	17,7%	10,1%	9,1%
1,15 seconds	nt	nt	nt	31,9%	25,3%	20,1%	27,8%	20,2%	17,2%
0,57 seconds	nt	nt	nt	nt	nt	nt	38,5%	30,4%	25,9%

nt – not tested

6. CONCLUSIONS AND FUTURE WORK

This paper refers the necessary requirements for RTDW and presents a methodology for supporting the implementation of RTDW 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 real time data warehousing schemas.

We have shown its functionality, recurring to a simulation using the TPC-H benchmark, performing continuous data integration at various time rates against the execution of various simultaneous query workloads, for data warehouses with different scale sizes. All scenarios show that it is possible to achieve real-time data warehousing 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 data warehouse.

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.

7. REFERENCES

[1] D. J. Abadi, D. Carney, et al., 2003. *"Aurora: A New Model and Architecture for Data Stream Management"*, The VLDB Journal, 12(2), pp. 120-139.

[2] S. Babu, and J. Widom, 2001. *"Continuous Queries Over Data Streams"*, SIGMOD Record 30(3), pp. 109-120.

[3] T. Binder, 2003. *Gong User Manual*, Tecco Software Entwicklung AG.

[4] M. Bouzeghoub, F. Fabret, and M. Matulovic, 1999. *"Modeling Data Warehouse Refreshment Process as a Workflow Application"*, Intern. Workshop on Design and Management of Data Warehouses (DMDW).

[5] R. M. Bruckner, B. List, and J. Schiefer, 2002. *"Striving Towards Near Real-Time Data Integration for Data Warehouses"*, International Conference on Data Warehousing and Knowledge Discovery (DAWAK).

[6] R. M. Bruckner, and A. M. Tjoa, 2002. *"Capturing Delays and Valid Times in Data Warehouses – Towards Timely Consistent Analyses"*. Journal of Intelligent Information Systems (JIIS), 19:2, pp. 169-190.

[7] S. Chaudhuri, and U. Dayal, 1997. *"An Overview of Data Warehousing and OLAP Technology"*, SIGMOD Record, Volume 26, Number 1, pp. 65-74.

[8] W. H. Inmon, R. H. Terdeman, J. Norris-Montanari, and D. Meers, 2001. *Data Warehousing for E-Business*, J. Wiley & Sons.

[9] I. C. Italiano, and J. E. Ferreira, 2006. *"Synchronization Options for Data Warehouse Designs"*, IEEE Computer Magazine.

[10] A. Karakasidis, P. Vassiliadis, and E. Pitoura, 2005. *"ETL Queues for Active Data Warehousing"*, IQIS'05.

[11] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite, 1998. *The Data Warehouse Lifecycle Toolkit – Expert Methods for Designing, Developing and Deploying Data Warehouses*, Wiley Computer Publishing.

[12] R. Kimball, and J. Caserta, 2004. *The Data Warehouse ETL Toolkit*, Wiley Computer Publishing.

[13] E. Kuhn, 2003. *"The Zero-Delay Data Warehouse: Mobilizing Heterogeneous Databases"*, International Conference on Very Large Data Bases (VLDB).

[14] W. Labio, J. Yang, Y. Cui, H. Garcia-Molina, and J. Widom, 2000. *"Performance Issues in Incremental Warehouse Maintenance"*, International Conference on Very Large Data Bases (VLDB).

[15] D. Lomet, and J. Gehrke, 2003. *Special Issue on Data Stream Processing*, IEEE Data Eng. Bulletin, 26(1).

[16] *Oracle Corporation*, 2005. www.oracle.com

[17] T. B. Pedersen, 2004. *"How is BI Used in Industry?"*, Int. Conf. on Data Warehousing and Knowledge Discovery (DAWAK).

[18] J. F. Roddick, and M. Schrefl, 2000. *"Towards an Accommodation of Delay in Temporal Active Databases"*, 11th ADC.

[19] A. Simitsis, P. Vassiliadis and T. Sellis, 2005. *"Optimizing ETL Processes in Data Warehouses"*, Inte. Conference on Data Engineering (ICDE).

[20] U. Srivastava, and J. Widom, 2004. *"Flexible Time Management in Data Stream Systems"*, PODS.

[21] D. Theodoratus, and M. Bouzeghoub, 1999. *"Data Currency Quality Factors in Data Warehouse Design"*, International Workshop on the Design and Management of Data Warehouses (DMDW).

[22] *TPC-H decision support benchmark*, Transaction Processing Council, www.tpc.com.

[23] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis, 2001. *"ARKTOS: Towards the Modelling, Design, Control and Execution of ETL Processes"*, Information Systems, Vol. 26(8).

[24] C. White, 2002. *"Intelligent Business Strategies: Real-Time Data Warehousing Heats Up"*, DM Preview, www.dmreview.com/article_sub_cfm?articleId=5570.

[25] J. Yang, 2001. *"Temporal Data Warehousing"*, Ph.D. Thesis, Dpt. Computer Science, Stanford University.

[26] J. Yang, and J. Widom, 2001. *"Incremental Computation and Maintenance of Temporal Aggregates"*, 17th Intern. Conference on Data Engineering (ICDE).

[27] J. Yang, and J. Widom, 2001. *"Temporal View Self-Maintenance"*, 7th Int. Conf. Extending Database Technology (EDBT).

[28] T. Zurek, and K. Kreplin, 2001. *"SAP Business Information Warehouse – From Data Warehousing to an E-Business Platform"*, 17th International Conference on Data Engineering (ICDE).