# Real Time Loading of Enterprise Data Using Fragmentation of Data Warehouses

Diego Pereira[1], Leonardo Guerreiro Azevedo[1, 2], Asterio Tanaka[1]

[1] Applied Informatics Department, Federal University of the State of Rio de Janeiro
[2] NP2Tec – Research and Practice Group in Information Technology
Rio de Janeiro, RJ, Brasil
**{diego.pereira, azevedo, tanaka}@uniriotec.br**

**Abstract.** Real-time ETL (Extraction, Transformation and Loading) of enterprise data is one of the foremost features of next generation Business Intelligence (BI 2.0). This paper presents a proposal for loading operational data in real time using a Data Warehouse (DW) architecture with faster processing time than current approaches. Distributed database techniques, like derived horizontal fragmentation in shared nothing architecture, are used to create fragments that are specialized in most current data and optimized to achieve continuous insertions. Using this approach, the DW can be updated near-line from operational data sources. As a result, DW queries are executed over real time data or very close to that. Moreover, continuous loadings do not impact queries response time. In addition, we extended the Star Schema Benchmark to address loading operational data in real time. This benchmark, including the new feature, is used to validate and demonstrate the efficiency of our approach compared to other ones.

Keywords: Real Time Data Warehouse, Business Intelligence 2.0, Database Distribution, Database Fragmentation

## 1. INTRODUCTION

Business Intelligence (BI) is a term coined in the early 90s by the Gartner Group [Watson and Wixom 2007], and it represents a collection of technologies, tools and practices for data warehousing, data mining, analytical processing, reporting and queries. The goal of BI is to collect, integrate, cleanse and mine business information to assist experts making strategic decisions based on historical data [Chaudhuri *et al*. 2001] [Dayal *et al*. 2009].

With the emergence of new business models, the BI architecture needs to undergo some changes in what has been called BI 2.0. Access to operational data as close as to real time is one of the innovations outlined in this scenario [Stodder 2007].

Accessing real-time data refers to operational information being available for analysis by BI tools as soon as possible, not just at the end of the day, as in traditional BI scenario where DW updates are performed out of business hours. This feature is fundamental to decision making in businesses such as e-commerce, stock exchanges and telecommunications. Moreover, with the globalization and the intensive use of Internet, critical business needs its data sources available 24 hours a day, 7 days a week for decision making process. Thus, the time window for loading a DW is getting smaller. In this way, a major requirement for BI 2.0 success is the evolution of the DW and the ETL process to support a continuous flow of data, avoiding or at least decreasing any downtime.

Among the areas identified as potential problems in the scenario of real time data loading [Inmon et al. 2008] [Kimball and Caserta 2004], this paper focuses on solving problems related to the storage of real-time data in Data Warehouses. Gathering and processing problems are abstracted by using the Star Schema Benchmark extension proposed in this paper that simulates the data insertion step. We propose a physical DW architecture that uses distribution techniques and fragmentation in the DW facts table to address the real time data loading problem. The goal is to allow data insertion in

acceptable time without impacting queries execution performance. This work also extends the Star Schema Benchmark (SSB) [O'Neil et al. 2007] to measure Real Time DW performance.

This article is organized in five sections, the first being this introduction. In section 2, related works are revised. In section 3, the proposed architecture is described. In section 4, experiments that validate the proposal are presented. Finally, section 5 presents the conclusions and final remarks.

## 2. RELATED WORKS

Several proposals have been presented to address the continuous load problem in Data Warehouses. Inmon *et al*. [2008] presented the main innovations that a DW needs to address in order to fit into the BI 2.0 scenario. The ability to perform online updates, to store unstructured data and to define data life cycle are some of their proposals. Online updates should happen on a real-time environment called interactive sector, in which data is stored following application layer structure, in order to avoid transformations to allow lower insertion time. Later, when it is desirable to execute more complex queries over the real-time data, it is integrated and moved to the integrated sector. Solutions were developed in the literature following those ideas, separating real-time data from historical ones. Thus, materialized views and indexes updates become less striking, as they occur in smaller amounts of data.

Thomsen *et al*. [2008] present a middleware proposal where data coming from operational sources is stored in main memory. It allows queries executing on historical data and real-time data. When the amount of data reaches a threshold, an ETL process is executed to load main memory data into the DW. The proposal resulted in faster insertions than loading data directly on DW, however to query main memory structures was slower than querying data on a common database. Also, this proposal uses a specific JDBC driver to allow users transparency, which was not developed to consider a distributed scenario.

Santos and Bernardino [2008] propose an adaptation to the DW model. New tables must be created with the same structure as those from the original DW, but without query optimizations mechanisms, such as indexes. These tables are added to the traditional model and become the destination of real time operational data. When real time data access performance becomes unacceptable, because of the lack of optimization mechanisms, they are extracted and loaded on the conventional DW tables. The drawback in this approach is that queries used by analytical processing tools need to be changed to consider both DW and new tables. This result in reconfiguration of dashboards, scorecards and cubes of business environments. These reconfigurations are very difficult to handle and error prone.

Therefore, an approach following Inmon *et al*. [2008] proposal requires not only an acceptable performance during continuous insertions in the interactive sector, but also needs to guarantee query and update transparency.

## 3. FRAGMENTATION FOR REAL TIME LOADING

This paper proposes the use of data fragmentation and distribution techniques to allow real-time loading without impacting queries response time. Also, query and update transparency is provided to user´s data accesses through the use of a Distributed Database Management System (DDBMS) [Risch 2009].

Using fragmentation and distribution techniques in data warehouses is historically related to database tuning. However, the distribution of the fragments usually takes into account only the query performance [Furtado 2004], generating scenarios where loading time optimization is ignored. With the need for continuous loads execution to allow real time Data Warehousing, the behavior of these insertions becomes a key issue in the whole performance of the DW and their applications.

DW usually uses database optimization mechanisms to decrease queries response time, such as indexes and materialized views. These mechanisms need to be refreshed whenever an insertion happens on the DW, impacting the performance of insertion operations. We propose to execute these insertions in a table with few tuples where those mechanisms are unnecessary, while still working with previously inserted historical data in traditional fact tables. Hence queries can be executed with low response times, and insertions can be executed without the cost of refreshing the whole fact tables.

To achieve this goal, this proposal uses fragmentation techniques to create a fragment, called Real Time Fragment, where every insertion into the DW is executed. The time dimension is used as a fragmentation key, performing a derived horizontal fragmentation over the fact table.

Each tuple inserted into a fact table represents one business transaction, which usually has a date column that indicates transaction time. When these operations perform in real time, this column point to the current date. As in Star Schema model, the time dimension stores date values, while the fact table only has a foreign key of it. Fact table fragments are created using values from the time dimension. So, Real Time Fragment is created using current date value, and other fragments are created using different criteria, such as hash, round-robin or another date ranges. This proposal implements an balanced distribution of historical data between historical fragments, but this technique could also be used to create distinct date ranges to the historical fragments implementing Inmon *et al.*[2008] concept of data life cycle.

After being created, these fragments are distributed over computer cluster nodes in the same way as traditionally fragment allocation is done. Shared nothing distributed architecture [Özsu and Valduriez 1999] is commonly used over such fragmentation scenarios. In this architecture each node has its own database, disk and memory areas.

Fig. 1 illustrates the proposed distribution technique. The first node is responsible for real time insertions, while the others have all historical data distributed among themselves. All insertions are routed by the middleware pgpool-II[1] to node 1, as it is the Real Time Fragment, while queries are partitioned by pgpool-II and forwarded to all nodes. Thus this middleware features query and update transparency to user's data access, allowing this proposal to bypass user's data location awareness problem that happens among related works.

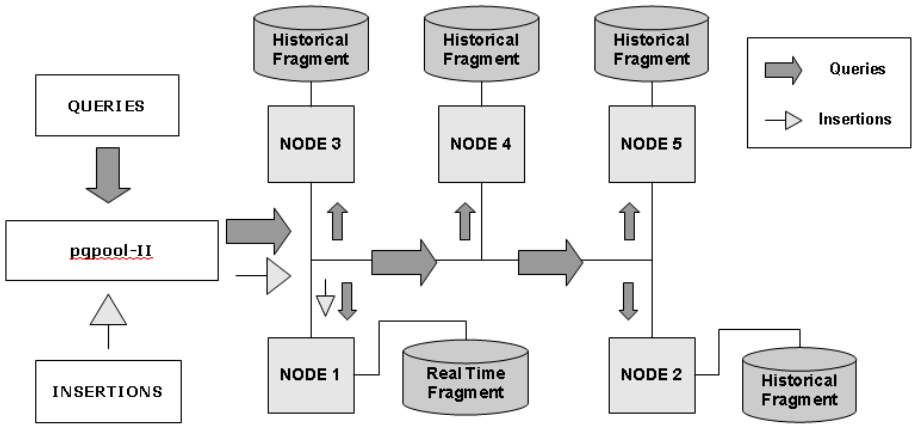


Fig. 1. Proposed Distribution Architecture

[1] http://pgpool.projects.postgresql.org/

As more insertions occur into the Real Time Fragment, its size continuously grows and its performance decreases over time. When a threshold is reached, data in this fragment is redistributed to the historical nodes. This operation is faster than an ETL process gathering data from operational data sources, because all data transformations has already been done in the Real Time Fragment. Thus the downtime for data redistribution of the DW is quite lower.

Santos and Bernardino [2008] propose the following metrics to choose the right time to redistribute real time data among the DW: number of rows, size of the database, every *t* time units, when some query exceeds *t* time units or even when the database administrator (DBA) simply choose to do it. This work implements query response time check approach. The DBA chooses a query *q*, usually the most executed one, and periodically executes it on historical data and on the Real Time Fragment. If the second response time is greater than the first one, it means that the Real Time Fragment is delaying query execution, and redistribution is executed to avoid this.

## 4. EXPERIMENTAL EVALUATION

Experimental tests were executed to compare our proposal against a traditional scenario. Those tests include Star Schema Benchmark queries response times, load times, Real Time Fragment redistribution frequency and comparisons about data location awareness problems.

### 4.1 Star Schema Benchmark

The Star Schema Benchmark (SSB) [O'Neil et al. 2007] is an adaptation of the TPC-H[2] benchmark for decision support systems where the DW schema is converted to the dimensional model and the queries are adapted accordingly. Its data model has a fact table, called LineOrder, and four dimension tables: customer; supplier; part, and date. This benchmark is composed of thirteen queries grouped into four categories.

The SSB also adapted the TPC-H data generator, called DBGEN, to work with the dimensional model. However this benchmark does not offer any kind of analysis about Real Time DW behavior. We extended SSB to include this feature.

Original DBGEN considers fact rows with date between '01/01/1992' and '12/31/1998'. It was changed to include '01/01/1999' day representing the current date, which is used for real time loading. In this way, DBGEN generates original loading files using its regular date range and also new fact tables files with tuples using only '01/01/1999' date.

Regarding data size, DBGEN has a parameter called scale factor which allows users to choose how many data it will generate. When a scale factor of 1 is used, DBGEN generates about 6,000,000 fact table rows, if scale factor is 10, then DBGEN generates 60,000,000 fact table rows, and so on. In the experimental scenarios, the new fact table files for real time loading has about 0.1% of fact table rows. This percentage was derived from the TPC-H refresh functions size.

This newly generated data can be loaded according to several strategies of Real Time Loading, such as Micro Batch, ETL Capture, Transform and Flow (CTF), Enterprise Information Integration (EII) and Enterprise Application Integration (EAI). Which strategy to use depends entirely on the load characteristics of the simulated scenario. Kimball and Caserta [2004] present a comparison between these strategies, classifying them by its reporting capabilities, such as historical data support, integration complexity and data freshness. They argue that Micro Batch ETL and EAI solutions are suitable for organizations facing data integration challenges, while CTF, EII and EAI are suitable for organizations expecting low latency reporting. This work chooses to use Micro Batch ETL strategy

[2] http://www.tpc.org/tpch/default.asp

during the experiments, mainly because it allows the same level of integration and transformation as a regular ETL tool. This means that real time data reports have the same level of complexity as the historical ones.

Besides, a new set of queries (Fig. 2) was added into SSB to allow measuring response time on real time data. Such queries were derived from each of the four SSB query groups, in order to represent the same level of analysis being performed on the most current data. So, every one of them has a where clause d_datekey=19990101 as this date correspond to the current date in the benchmark.

Q5.1

```
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey and d_datekey = 19990101
and lo_discount between 1 and 3 and lo_quantity < 25;
```

Q5.2

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
and lo_partkey = p_partkey
and lo_suppkey = s_suppkey and p_category = 'MFGR#12'
and s_region = 'AMERICA' and d_datekey = 19990101
group by d_year, p_brand1 order by d_year, p_brand1;
```

Q5.3

```
select c_nation, s_nation, d_year,
sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey and c_region = 'ASIA'
and s_region = 'ASIA' and d_datekey = 19990101
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;
```

Q5.4

```
select d_year, c_nation,
sum(lo_revenue - lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey
and lo_partkey = p_partkey and lo_orderdate = d_datekey
and c_region = 'AMERICA' and s_region = 'AMERICA'
and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
and d_datekey = 19990101
group by d_year, c_nation order by d_year, c_nation;
```

Fig. 2. Real Time SSB Queries

## 4.2 Comparison Procedures

First, DBGEN was executed using a scale factor of 100, resulting on 600,000,000 fact table rows. This size was large enough to allow fragmentation impact analysis. Micro Batch ETL strategy was used to execute real-time insertions. The performance results using our proposal of Real Time Data Fragmentation (section 3 - Fig. 1) were compared against traditional approach using Furtado [2004] proposal.

Furtado [2004] proposed data fragmentation using a hashing algorithm applied to the primary keys of facts table. Data is randomly distributed and balanced among fragments, such that each node has approximately 20% of total data. The insertion of new data occurs in the same way.

The comparison procedure was performed as follows: (i) Databases are loaded with data generated by DBGEN; (ii) Integrity constraints are activated; (iii) Database memory flush; (iv) SSB queries are executed without loads; (v) Database memory flush; (vi) Real-time loads are simulated using files generated by DBGEN; (vii) SSB queries are executed simultaneously with load simulations.

An analysis about Real Time Fragment degradation over time was also made. This assess whether our proposal needs an excessive number of redistributions. At last, a comparison was made regarding the data location awareness problem of related works, presenting benefits that query and update transparency provides to our proposal. This is achieved comparing how Santos and Bernardino [2008] proposal accesses data over its Real Time DW.

All tests were performed on the cluster of the CGOLAP[3] project: five computers with Intel Core 2 Duo E6750 (2.66 GHz) and 2GB of RAM. All computers run OpenSUSE 10.3 operating system and are interconnected through a Gigabit network. The databases are stored in PostgreSQL 8.4 clustered through pgpool-II middleware set in parallel mode.

### 4.3 Results

The first test considers the execution of SSB queries 1.1 to 4.3 [O'Neil et al. 2007]. Tests without loads simulate cases where loads do not occur with analytical queries executions. Tests with loads consider query execution during continuous loading. The chart presented at Fig. 3 shows results where traditional fragmentation has higher performance than the proposed fragmentation when loads are not executed together with queries. However, as soon as loading starts, query response time grows a lot in traditional fragmentation, while the proposed scenario remains almost unchanged.

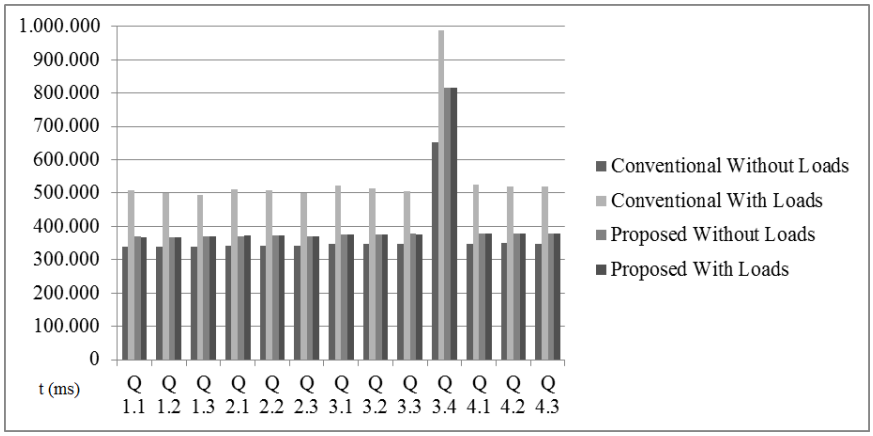


Fig. 3. Time Comparison Over Fragmentation Scenarios

The time required to execute loadings was also assessed. The longer is the loading time, users will wait longer to analyze fresh data. The loading time of our proposal was 22m 27s, while traditional approach required 04h 55m 58s, which means that our approach consumes only approximately 8% of traditional approach loading time.

Another important analysis is real time fragment degradation over time. This analysis was executed running benchmark's loads and evaluating queries responses. The chosen queries were the new set of SSB queries proposed in Section 4.1 - Fig. 2, which access only real time data. Fig. 4 presents response times after every new load. As each benchmark loads represents an increment of 0.1% in fact table rows, after forty loads, the real time fragment has a number of rows equal to 4% of the original inserted fact table rows. Considering this volume of data, queries response time on real time fragment was about 70 seconds. This is still lower than the average response time presented on Fig. 3, which is more than 300 seconds. This demonstrates that the redistribution process does not need to occur frequently, which avoids impacts that this process causes on the DW.

Each load time was also measured in order to analyze whether the performance decreases when real time fragment size grows. Fig. 5 shows that the load time was not affected by the size of the real time fragment, as it varied between 1290s (21m:30s) and 1440s (24m:00s) independently of the size.

[3] http://www.uniriotec.br/~cgolap/index.html

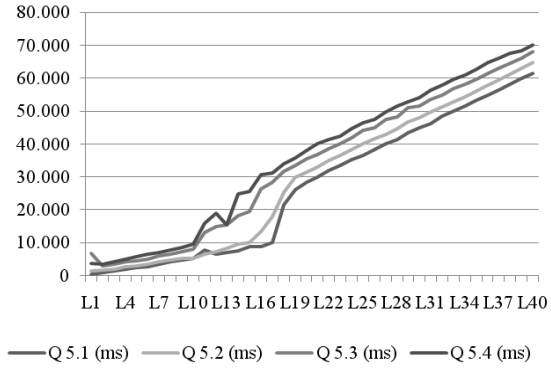


Fig. 4. Queries Degradation on Real Time Fragment

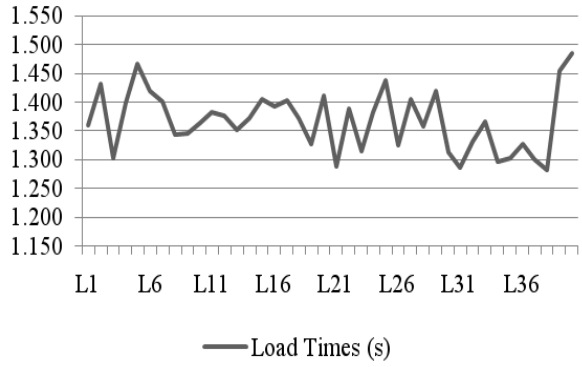


Fig. 5. Insertion Degradation into Real Time Fragment

Due to the use of a DDBMS, a consequence of this proposal is transparency in writing SQL statements. In other words, the user writes statements considering only one database, and the DDBMS is responsible for re-writting them to execute in the database cluster. To demonstrate this characteristic, we compared query Q1.1 from SSB against the same query when using Santos and Bernardino [2008] proposal. Fig 6 presents the original query and its corresponding execution plan generated from PostgreSQL while Fig. 7 presents the same query and its execution plan considering Santos and Bernadino approach. It's clear that query complexity and cost increases significantly when employing Santos and Bernardino approach.

ORIGINAL QUERY

```
select sum(lo_extendedprice*lo_discount) as revenue
    from lineorder, date
    where lo_orderdate = d_datekey
        and d_year = 1993
        and lo_discount between 1 and 3
        and lo_quantity < 25
```

ORIGINAL QUERY PLAN

```
Aggregate  (cost=4856615.10..4856615.11 rows=1 width=14)
  ->  Hash Join  (cost=86.51..4850599.51 rows=2406233 width=14)
        Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
        ->  Seq Scan on lineorder  (cost=0.00..4766970.56 rows=15861362 width=23)
              Filter: ((lo_discount >= 1::numeric) AND (lo_discount <= 3::numeric)
              AND (lo_quantity < 25::numeric))
        ->  Hash  (cost=81.95..81.95 rows=365 width=9)
              ->  Seq Scan on date  (cost=0.00..81.95 rows=365 width=9)
                    Filter: (d_year = 1993::numeric)
```

Fig. 6. SSB Query Q1.1

SANTOS AND BERNADINO QUERY

```
select sum(lo_extendedprice*lo_discount) as revenue
    from ( select lo_extendedprice, lo_discount
              from lineorder, date
              where lo_orderdate = d_datekey
                  and d_year = 1993
                  and lo_discount between 1 and 3
                  and lo_quantity < 25
            UNION ALL
            select lo_extendedprice, lo_discount
              from lineorderTmp, dateTmp
              where lo_orderdate = d_datekey
                  and d_year = 1993
                  and lo_discount between 1 and 3
                  and lo_quantity < 25 ) as result
```

SANTOS AND BERNADINO QUERY PLAN

```
Aggregate  (cost=5139137.31..5139137.33 rows=1 width=64)
  ->  Append  (cost=86.51..5131120.21 rows=3206840 width=64)
        ->  Hash Join  (cost=86.51..4850599.51 rows=2406233 width=14)
              Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
              ->  Seq Scan on lineorder  (cost=0.00..4766970.56 rows=15861362 width=23)
                    Filter: ((lo_discount >= 1::numeric) AND (lo_discount <= 3::numeric)
                    AND (lo_quantity < 25::numeric))
              ->  Hash  (cost=81.95..81.95 rows=365 width=9)
                    ->  Seq Scan on date  (cost=0.00..81.95 rows=365 width=9)
                          Filter: (d_year = 1993::numeric)
        ->  Nested Loop  (cost=0.00..248452.30 rows=800607 width=14)
              Join Filter: (lineordertmp.lo_orderdate = datetmp.d_datekey)
              ->  Seq Scan on datetmp  (cost=0.00..11.88 rows=1 width=32)
                    Filter: (d_year = 1993::numeric)
              ->  Seq Scan on lineordertmp  (cost=0.00..238432.84 rows=800607 width=23)
                    Filter: ((lineordertmp.lo_discount >= 1::numeric)
                    AND (lineordertmp.lo_discount <= 3::numeric)
                    AND (lineordertmp.lo_quantity < 25::numeric))
```

Fig. 7. SSB Query 1.1 Adapted to Santos and Bernardino [2008] Proposal

## 5. CONCLUSIONS

The broader use of BI tools has brought new challenges for their architecture. A relevant one is the use of operational data in real time, as addressed in this proposal. Continuous loading in DW has been the target of many proposals. Most of them focused on separating insertions from queries on historical data.

In this paper, we propose to use fragmentation and distribution to deal with this issue. Through the use of those database design techniques, this approach aims to solve the problem of loading data in real time, as well as to minimize the impact of continuous loading in the BI environment as a whole. An extension to the Star Schema Benchmark was made to allow the comparison of the impact of continuous loading over several DW architectures.

Results using the extended benchmark confirmed that using those well known techniques in a simple distributed DW architecture allows executing continuous loadings having 20% lower query response time and a huge loading time gain against a traditional DW.

As future work we propose: to experiment on others distributions techniques; to use our proposal on a real life scenario to evaluate its behavior; and, to adapt other approaches on distributed environments to to compare with our approach. The last proposal of future work is to check different trade-offs of fragment distribution scenarios. One of these could be allocating more than one fragment on each node, which would allow us to verify whether having nodes with both real-time and historical data fragments would be better than having specialized nodes. Another scenario would consider more than one specialized node for insertions, and evaluate the time required for real-time insertions versus historical queries.

REFERENCES


CHAUDHURI, S.; DAYAL, U.; GANTI, V. **Database technology for decision support systems.** Computer, v. 34, n. 12, p. 48-55, 2001.

DAYAL, U.; CASTELLANOS, M.; SIMITSIS, A.; WILKINSON, K. **Data integration flows for business intelligence.** In: Proceedings of the 12th Intl. Conference on Extending Database Technology: Advances in Database Technology, p. 1-11, 2009.

FURTADO, P. **Experimental Evidence on Partitioning in Parallel Data Warehouses.** In: Proceedings of the 7th *DOLAP*, p. 23-30, 2004.

INMON, W.; STRAUSS, D.; NEUSHLOSS, G. **DW 2.0 - Architecture for the Next Generation of Data Warehousing**. Morgan Kauffman, 2008.

KIMBALL, R.; CASERTA, J. **The Data Warehouse ETL Toolkit**. John Wiley & Sons Inc, 2004.

O'NEIL, P.; O'NEIL, B.; CHEN, X. **The Star Schema Benchmark (SSB).** http://www.cs.umb.edu/~poneil/StarSchemaB.pdf , 2007.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems**. 2nd ed. Prentice Hall, 1999.

SANTOS, R. J.; BERNARDINO, J. **Real-time data warehouse loading methodology.** In: Proceedings of the 2008 International Symposium on Database Engineering & Applications. p.49-58. Coimbra, Portugal, 2008.

STODDER, D. **Good BI, Cruel World?** Network Computing, v. 18, p. 56-66, 2007.

THOMSEN, C.; PEDERSEN, T.; LEHNER, W. **RiTE: Providing On-Demand Data for Right-Time Data Warehousing.** In: 24th International Conference Data Engineering. p.456-465, 2008.

WATSON, H.; WIXOM, B. **The Current State of Business Intelligence**. Computer, v. 40, n. 9, p. 96-99, 2007

RISCH, T. Distributed Architecture. In: Encyclopedia of Database Systems, Springer, pp.875-879, 2009.