Bi-temporal Query Optimization Techniques in Decision Insight

Azhar Ait Ouassarah, Nicolas Averseng, Xavier Fournet, Jean-Marc Petit, Romain Revol, Vasile-Marian Scuturici

Bi-temporal Query Optimization Techniques in Decision Insight

Azhar Ait Ouassarah[1,2]
Xavier Fournet[1]
Romain Revol[1]

[1] Axway
171 Rue Bureaux de la Colline
92210, St Cloud, France
{aaitouassarah,
naverseng,
xfournet,
rrevol}@axway.com

Nicolas Averseng[1]
Jean-Marc Petit[2]
Vasile-Marian Scuturici[2]

[2] Université de Lyon, CNRS,
INSA-Lyon, LIRIS, UMR5205
20 avenue Albert Einstein
69100, Villeurbanne, France
{azhar.ait-ouassarah,
jean-marc.petit, vasile-
marian.scuturici}@insa-
lyon.fr

ABSTRACT

La complexité de l'environnement dynamique dans lequel évoluent les entreprises requiert de la part de leurs managers de prendre des décisions pertinente en un laps de temps très court. Les systèmes de *supervision des activités métiers* devraient supporter des requêtes bitemporelles complexes qui accèdent aussi bien à des données historiques qu'à des données temps réel afin de détecter des anomalies ou bien des tendances dans l'activité de l'entreprise. Cependant, il s'avère que l'accès à ces deux types de données peut être lent, ce qui ne convient pas aux applications de supervision.
Dans ce papier, nous présentons Decision Insight, une plateforme développée par un éditeur de logiciels français pour aborder ce problème. Elle est basée sur un SGBD orienté colonnes qui redéfinit les requêtes bi-temporelles en: 1) un ensemble de requêtes continues pour gérer les données temps réel et dont les résultats sont matérialisés, et 2) une requête qui accède aussi bien aux données historiques qu'aux résultats des requêtes continues.
Nous démontrons l'intérêt de notre approche en utilisant une version adaptée du benchmark TPC-BiH qui est une extension bitemporelle du benchmark TPC-H.

Categories and Subject Descriptors

H.4 [**Information Systems**]: Data management systems—*Data model extensions,Temporal data*

General Terms

Business activity monitoring, temporal databases, temporal query optimization

BDA 29 septembre 2015, Ile de Porquerolles, France.

Keywords

Historical data, real-time data, materialized views, data reduction

1. INTRODUCTION

Companies operate in very dynamic and complex environments, requiring their managers to possess both agility and ability to make proactive operational decisions, in order to maintain or improve their business [23]. On the one hand, exploiting historical data is covered by the Business Intelligence (BI) domain [33], which provides access to past performance indicators by analyzing information stored in data warehouses. This enables users to understand what happened in the past and help them to prevent making mistakes in the future.
On the other hand, managers traditionally rely on Business Activity Monitoring (BAM) systems [23] to make operational decisions, see for instance `Splunk`[1] and `Vitria`[2]. BAM aims to provide real-time access to critical business performance indicators. Thus managers can have a deep insight into what is currently happening in their business before taking rapid and effective decisions. BAM gathers its information in real-time by analyzing data streams from multiple sources. Nevertheless these technologies are limited because they focus on real-time information, ignoring existing historical data. They do not give managers the necessary hindsight to compare the current company activity behavior with its history. Therefore, false positive decisions could be taken by analysts since real-time and historical data are not considered together.
To overcome these limitations, BAM tool capabilities could be enhanced with BI features. Managers could then use an hybrid tool to handle both real-time and historical data, allowing them to continually navigate from real-time to historical data. Such hybrid systems approach faces with two main issues: *combining real-time and historical systems* and *the performance issue*.

- *Handling Two Different Types of Systems*: Historical and real-time data are handled by two different types of systems. Historical data require well-known OLAP DataBase Management Systems (DBMS). Real-time data rely on Data Stream Management Systems (DSMS), Complex Event Processing

[1] `http://www.splunk.com/`
[2] `http://www.vitria.com/`

(CEP) or BAM systems [22]. These systems handle transient data streams and can query them thanks to *Continuous Queries*.
Handling within a unique system those two different types of data is clearly an issue.

- *Performance Issue*: BI systems are usually used to generate non interactive reports which do not have real-time (or near real-time) requirements. BAM tools however always provide managers with a graphical user interface (GUI) to monitor their business. The GUI is always interactive and offers the possibility to explore real-time data and other analyses computed over them. This means that the underlying system must guarantee fast response time of queries in charge of feeding the GUI with information. This is because GUI display lag can make the system unpractical, thus reducing its interest.
When historical data have to be queried, the main performance bottleneck is on the induced I/O cost which impacts real-time data processing, leading to unacceptable GUI display lag.

As far as we know, Chandrasekran and Franklin [9] and Reiss and al [24] were the first to address the topic of enhancing DSMS with DBMS capabilities. In [24], they define bitmap index specially designed to handle these two types of data. In [9], they present a framework using Data Reduction techniques, sampling techniques in their case, to limit I/O overhead induced by accessing historical data. Their approach enables the reduction level to be adapted according to the available resources. They also addressed the issue of when to perform data reduction. Three approaches were explored: data reduction at data arrival, at query execution or both. The framework was implemented on top of PostgreSQL.

In this paper, we focus on queries that access both historical data already existing in the system and live data that had not yet been entered into the system when the query was defined. They are temporal by nature because they access data evolving over time. Such queries support either the *valid time dimension* [28] or the *transaction time dimension* [29] or both (*bi-temporal query*). The support of the *transaction time* means that the query can access the history of data as it is modified in the DB. The *valid time* enables access to the history of data as it evolves in the modeled reality. We consider the case of bi-temporal queries. Let us consider the following example:

> "*What is the average of new revenue achieved by the company every month last year (2014), considering the DB at the instant 1/10/2015 ?*"

This simple query requires access to all items that have been ordered from 1/1/2014 to 12/31/2014. Depending on the database, this query may induce a large number of I/O operations and can be very long to complete. Whenever this query is executed only once, traditional approaches can be applied. Otherwise, if this query has to be executed several times – for example to frequently refresh some GUI – this is not acceptable.

Paper contribution.

We present Decision Insight [3], a platform that deals with historical and real-time data in a unified manner. The project was launched in 2008 under the name of Tornado by Systar, a French software editor. This latter was then acquired in 2014 by Axway, another French software editor and Tornado became Decision Insight. Axway aims to compete with major leading industries in data management worldwide and turns out be the fifth french editor according to a recent study[3]. The project mobilizes about twenty engineers and it has been marketed since 2013. This platform implements an optimization that consists in redefining complex bi-temporal queries into: 1) a set of continuous queries in charge of handling real time data streams (whose results are materialized) and 2) a query that accesses both historical and materialized results of the previous continuous queries. Thus, Decision Insight can provide analysts with timely answers through a convenient GUI [3]. Decision Insight is based on a column-store bi-temporal DBMS that handles these two types of data and implements a simple and efficient bi-temporal query optimization technique. We demonstrate the interest of our approach using an adapted version of TPC-BiH, a bi-temporal extension of the TPC-H benchmark. Extensive experiments have been conducted, pointing out the interest of Decision Insight for delivering timely information based on historical and real-time data.

Paper organization.

The remainder of this paper is structured as follows: In section 2 we introduce our query rewriting approach. Then in section 3, we address the issue of materialized continuous query computation scheduling. Then in section 4, we point out how the contribution has been implemented within the Decision Insight framework. Experiments are given in section 5 using the TPC-BiH [17] benchmark. Section 6 is devoted to related works. Then we conclude the paper in section 7.

2. QUERY REWRITING

In this section, we first describe how aggregate bi-temporal SQL queries are expressed. Then we explain how complex bi-temporal queries are decomposed as a set of materialized continuous queries (CQ) and one special bi-temporal query.

In the rest of the paper, all queries are expressed using a pseudo-SQL formalism based on SQL:2011 supporting temporal features [20].

2.1 Temporal Query Expression for Aggregate Queries

It is well known that the expressiveness of SQL:2011 is limited to defining temporal queries performing aggregations [20, 18].

2.1.1 Rhythm

To deal with this issue, we introduce the concept of *rhythm* as a partition of the valid time domain into contiguous and equal-length time intervals. A rhythm is defined by a couple $(begin, duration)$ where $begin$ is the reference time instant to be used for partitioning the valid time domain and $duration$ is the length of each interval. For example, the rhythm $(01/01/2014, 1\text{ day})$ corresponds to the following partition:

$$[01/01/2014, 01/02/2014[\cup[01/02/2014, 01/03/2014[\cup \ldots$$

Clearly, a rhythm is used to define the time range over which the aggregation is performed.

Rhythms can be implemented in SQL:2011 as a relation with two attributes representing the endpoints of each interval of that rhythm. In the sequel, all queries use a one-day rhythm $(01/01/1990, 1$ day) represented by the relation $Rhythm\_day$.

2.1.2 Temporal Data Schema of the Running Example

[3]http://www.infodsi.com/articles/155700/editeurs-francais-contre-courant-conjoncture.html?key=c9fee7a30353fd4a

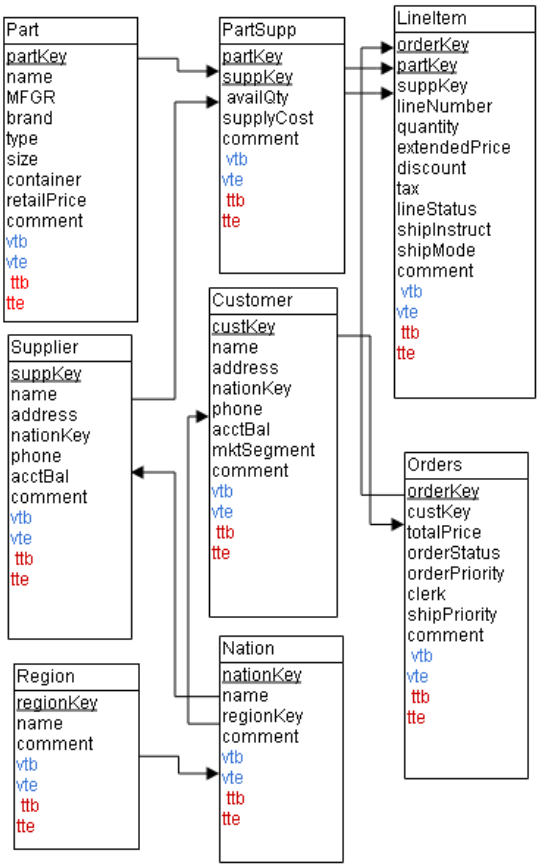

Figure 1: Temporal database schema adapted from TPC-BiH [17]

The TPC-BiH database schema [17] is a bi-temporal extension of the TPC-H database schema. It mainly describes customers, orders and line items of a fictive company. We have slightly adapted the TPC-BiH database schema to fit our needs (Fig. 1). The schema is fully bi-temporal, i.e all tables are extended with both valid time and transaction time.

2.1.3 *Temporal Aggregations*

We follow Kaufmann and al's classification of time ranges in temporal aggregations [18]. They identified four types of time ranges:

- **Instantaneous Aggregation** [12]: The aggregation is performed on all valid tuples at an instant, e.g "What is the number of orders with orderstatus='O' at the beginning of every day during the valid time interval [01/01/1994, 06/01/1994[, considering the DB at the transaction instant '10/20/2014T10:30:00.0' ?". This query is represented in Listing 1.

Listing 1: Example of an instantaneous aggregation

```
SELECT COUNT(*) as numberOpenOrders,
  Rhythm_day.vtb, Rhythm_day.vte
FROM Orders
FOR SYSTEM_TIME AS OF TIMESTAMP
'10/20/2014T10:30:00.0' ,
  Rhythm_day
WHERE
  -- filter of rhythm intervals
  (01/01/1994 <= Rhythm_day.vtb AND
  Rhythm_day.vtb < 06/01/1994) AND
  -- only orders valid at Rhythm_day.vtb
  Orders.vtb <= Rhythm_day.vtb AND
  Rhythm_day.vtb < Orders.vte  AND
  orderstatus= 'O'
GROUP BY Rhythm_day.vtb;
```

- **Tumbling Window**: the aggregation is performed on non overlapping intervals, e.g "What is the total revenue achieved by the company every day during the valid time interval [01/01/-1994, 06/01/1994[and considering the DB at tt=10/20/2014T-10:30:00.0 ?". This query is represented in Listing. 2.

Listing 2: Tumbling Window query

```
SELECT SUM(extendedPrice) as totalRevenue,
  Rhythm_day.vtb, Rhythm_day.vte
FROM LineItems
FOR SYSTEM_TIME AS OF TIMESTAMP
'10/20/2014T10:30:00.0',
  Rhythm_day
WHERE
  -- filter of rhythm intervals
  (01/01/1994 <= Rhythm_day.vtb AND
  Rhythm_day.vtb < 06/01/1994) AND
  -- filter new lineItems
  Rhythm_day.vtb <= LineItems.vtb AND
  LineItems.vtb < Rhythm_day.vte
GROUP BY Rhythm_day.vtb;
```

- **Sliding Window**: the aggregation is performed on overlapping intervals, e.g "What is the total revenue achieved by the company during the last 10 days, computed every day if we consider the valid time interval [01/01/1994, 06/01/1994[and considering the DB at tt = '10/20/2014T10:30:00.0' ? ". This query is represented in Listing 3.

Listing 3: Aggregation using a sliding window time range

```
SELECT SUM(extendedPrice) as totalRevenue,
  Rhythm_day.vtb, Rhythm_day.vte
FROM LineItems
FOR SYSTEM_TIME AS OF TIMESTAMP
'10/20/2014T10:30:00.0',
  Rhythm_day
WHERE
  -- filter of rhythm intervals
( 01/01/1994 <= Rhythm_day.vtb AND
  Rhythm_day.vtb < 06/01/1994) AND
  -- filter new lineItems
  (Rhythm_day.vtb - INTERVAL 10 days) <=
  LineItems.vtb AND
  LineItems.vtb < Rhythm_day.vtb
GROUP BY Rhythm_day.vtb;
```

- **Landmark Window**: the aggregation is performed on intervals that share the same interval begin, e.g "What is the total revenue achieved by the company to each day for the current month considering the valid time instant [01/01/1994, 01/23/1994[and the DB at tt = '10/20/2014T10:30:00.0' ?". This query is represented in 4.

Listing 4: Aggregation using a landmark window time range

```
SELECT SUM(extendedPrice) as totalRevenue,
```

```
    '01/01/1994' as vtb, Rhythm_day.vtb

FROM LineItems
FOR SYSTEM_TIME AS OF TIMESTAMP
'10/20/2014T10:30:00.0',
  Rhythm_day
WHERE
  -- filter of rhythm intervals
  (01/01/1994 <= Rhythm_day.vtb AND
  Rhythm_day.vtb < 01/23/1994) AND
  -- filter new lineItems
  01/01/1994 <= LineItems.vtb AND
  LineItems.vtb < Rhythm_day.vtb
GROUP BY Rhythm_day.vtb;
```

2.2 Query Rewriting Technique

Business Activity Monitoring systems usually provide managers with features to build so-called views to monitor their business through user-friendly GUI. Those views use underlying queries to feed them with information to display. Consequently, they are not intended to be executed only once and then deleted as is usually the case in standard applications. Indeed, they can be evaluated several times, as long as the related view needs to be updated.

In this section, we sketch the main idea of our query rewriting technique. Without loss of generality, we are concerned with the following class of bi-temporal queries [30]:

Listing 5: Initial query Q_t

```
SELECT A1, A2, ..., An, Agg1, Agg2, ..., Aggk,
  Rhythm_Table.vtb,Rhythm_Table.vte
FROM table1, table2, ..., tableJ,
  stream1, stream2, ..., streamK,
  Rhythm_Table
WHERE tc1 AND tc2 AND ... AND tcn AND
  c1 AND c2 AND ... AND cm
GROUP BY A1, A2, ..., An,
  Rhythm_Table.vtb, Rhythm_Table.vte
```

where:

- $A_1, A_2, \ldots, A_n$ are attributes or derived attributes,
- Agg_1, Agg_2, ..., Agg_k are aggregation functions, e.g., AVG, SUM, MIN.
- The WHERE clause is a conjunction of selection predicates and join predicates: tcj predicates are over temporal attributes while ci are over non temporal ones.
- $table_1, table_2, \ldots, table_J$ are tables from the accessed database (historical data).
- $stream_1$, $stream_2$, ..., $stream_K$ are data streams (live data)
- $Rhythm\_Table$ is the table defined in the previous section.

Such a query is used to feed an underlying GUI whenever it needs to be updated, e.g due to a user interaction. Whenever the amount of data to be processed exceeds some limits, GUI latency deteriorates. Therefore, to address the scalability issue, we rely on data reduction techniques. Intuitively, we compute as soon as possible some partial answers allowing to efficiently answer a query asked by decision-makers. In other words, instead of performing aggregations at query time, we propose to perform them on data arrival. Thus, when a query is executed, it simply accesses the results of the aggregations which requires fewer I/O operations. This approach ensures that the most expensive I/O costs have been performed before information is needed by a decision-maker. Hence, at query-time, the cost will be as low as possible, thus satisfying our major goal.

Given a bi-temporal query, the process is as follows:

- one or more simple *continuous queries* [4] are defined, and their results are materialized. Such queries handle large volumes of data and do not affect historical data. They are referred to as *materialized continuous queries*;
- one elaborated temporal query, referred to as an *on-demand query*, in charge of providing decision-makers with results. Such a query accesses both historical and live data, including materialized CQs.

This approach has the advantage of providing a unified way to access both real-time and historical information through temporal queries. The result of this approach is *equivalent* to the result of the initial query against the same data. The reader is referred to [19] for equivalence of continuous queries. This is not in the scope of this paper.

2.3 Materialized Continuous Queries

For each aggregation Agg_i in the initial query, we define one continuous query in charge of reducing input data into pre-computed aggregates. This query is simple and can handle a large volume of data, as in (Listing 6).

Listing 6: A materialized continuous query

```
SELECT A1,  A2, ..., An, Agg, vtb, vte
FROM stream1, stream2, ..., streamK
WHERE tc1 AND tc2 AND ... AND tcn AND
  c1 AND c2 AND ... AND cm
GROUP BY A1,  A2, ..., An
```

where:

- Agg is the aggregation operation performed by the query,
- $stream1, stream2, ..., streamK$ is the set of accessed data streams,
- vtb and vte are two time attributes representing the time interval during which the computed result is valid,
- the result of this query is stored in a table, thus becoming historical data.

Each continuous query is bound at its creation to a *rhythm*. For each interval of the rhythm, the query returns one result that is stored in the DB. The choice of the rhythm depends on the user's needs. The more accurate the expected result, the finer the rhythm's granularity, and the higher CPU cost and memory utilization.

Whenever a continuous query is created, some new attributes linked to that query are added dynamically to the database schema. This is intended to store the query results for future use. If we consider the example from the introduction, then our approach requires one continuous query (Listing. 7).

Listing 7: The continuous query sumNewRevenuePerDay

```
SELECT SUM(extendedPrice) as agg,
  [vtDay].vtb, [vtDay].vte
FROM LineItems
WHERE
  [vtDay].vtb <= vtb AND
```

```
  vtb < [vtDay].vte) AND
  Rhythm_day.vtb <= LineItems.vtb AND
  LineItems.vtb < Rhythm_day.vte;
```

2.4 On-demand Queries

An on-demand query is a bi-temporal query executed against the database whenever new information is required by decision-makers through their GUI.

Listing 8: On-demand query

```
SELECT A1, A2, ..., An
FROM table1, table2, ..., tableJ
WHERE tc1 AND tc2 AND ... AND tcn AND
  c1 AND c2 AND ... AND cm;
```

We consider two classes of *on-demand queries*: *time travel* and *time slice* queries.

- **Time Travel**: This consists in acquiring the state, or snapshot, of the DB at a specific time. Here is an example of this class of queries (listing 9): " What is the revenue achieved by the company at vt = 10/13/2014, considering the DB at tt = 10/20/2014T10:30:00.0?".

Listing 9: A time travel query example

```
SELECT aggr
FROM sumNewRevenuePerDay
FOR SYSTEM_TIME AS OF TIMESTAMP
10/20/2014T10:30:00.0
WHERE vtb <= 10/13/2014 AND 10/13/2014 < vte;
```

- **Time Slice**: This class of queries is intended to return the historical data according to one temporal dimension. We fix one time dimension at a particular time instant while the other one is fixed at an interval. Here is an example of this class of queries (listing 10): "What is the revenue achieved by the company during the interval [10/13/2014, 12/13/2014[and considering the DB at tt=10/20/2014T10:30:00.0?"

Listing 10: A time slice query example

```
SELECT aggr, vtb, vte
FROM sumNewRevenuePerDay
FOR SYSTEM_TIME AS OF TIMESTAMP
10/20/2014T10:30:00.0
WHERE 10/13/2014 <= vtb AND vtb < 12/13/2014;
```

2.5 Data Storage

A database used in BAM applications has to store both real-time data and results of the materialized continuous queries. It thus offers an unified interface to access them all. Since these data are bi-temporal, we need a database management system with bi-temporal capabilities. Decision Insight is based on a column-store DBMS [31, 7] which means that a table is stored column by column. We outline below three main reasons motivating our choice.

2.5.1 Performances in Analytical Workloads

The *oriented-column* databases are intended to perform analytical queries that analyze data and give an insight into the business activity, e.g the number of orders in pending status. The column-oriented database systems outperform row-oriented database systems on analytical workloads such as those found in business intelligence and decision support applications [1].

2.5.2 Dynamic Update of the Database Schema

Our query optimization requires adding and removing attributes dynamically. The row-oriented approach is not suitable in our case because addition or deletion of an attribute affects the whole table, with performance impacts on the modified table. However the column-oriented approach does not suffer from this issue since each attribute has its own column.

2.5.3 Dealing with Temporal Attribute Evolution

The database must have bi-temporal built-in support and is intended to handle the history of data as it evolves. In the row-oriented approach, the update of an attribute value requires adding a new tuple with the new value. This behavior causes both a storage overhead and an increase in query execution time due to data duplication. As an example, let us consider the table *Customer* (Table 1) represented using the formalism proposed by Snodgrass [27]. The update of the attribute *balance* for the customer "AWM" leads to the insertion of two new tuples in the table 2.
In a column-oriented approach, each attribute can evolve independently because it is stored in its own column.

Table 1: customer before the update

custid	name	balance	vtb	vte	ttb	tte
1	Benason	12000	10/16	∞	10/16	∞
2	AWM	2000	10/9	∞	10/10	∞
3	Vop	47800	10/13	∞	10/14	∞

Table 2: customer after the update

custid	name	balance	vtb	vte	ttb	tte
1	Benason	12000	10/16	∞	10/16	∞
2	AWM	2000	10/9	∞	10/10	10/17
2	AWM	2000	10/9	10/17	10/17	∞
2	AWM	6000	10/17	∞	10/17	∞
3	Vop	47800	10/13	∞	10/14	∞

This issue has been addressed by Jensen and Snodgrass from a logical point of view [16]. They support the idea that the introduction of temporal dimensions in the data model requires adaptation of the data model design. They propose a handbook of best practices for the design process. However, this is not in the scope of this paper. As shown above, Decision Insight is therefore based on a column-store DBMS.

2.6 Computation Scheduling of Materialized Continuous Queries

The use of materialized views requires in general to consider the scheduling strategy to compute its results. This strategy has to reconcile keeping views up-to-date as data is collected and limit the number of refresh so the computation cost overhead is contained. Actually it is unthinkable to compute a view for each single incoming update of entity. It is rather wise to refresh views periodically or by bunch of updates. In the case of a soft real-time 2TDBMS, the maintenance of materialized queries is more complex. The real-time aspect induces computation deadline constraints to ensure fresh information to managers. The bitemporality necessitates to consider the semantics of the two temporal dimensions to choose the adapted computation strategy. Indeed we need to determine the adequate instant when data is supposed to be available in the database to trigger the computation. In the general case the two dimensions are orthogonal, which means that there is no restrictions

between the valid time and the transaction time of any fact in the DB. However in many practical applications there is a restriction relationship between them. For example, if we suppose that every event that occurs in the reality is considered as valid when it is inserted in the DB which, then $vt_e = tt_e$. This topic has been addressed by Jensen and Snodgrass in bitemporal relational databases [15] under the name *temporal specialization relations*. The authors classify bitemporal relations into 15 classes of specialization.
In Decision Insight, we consider three types of events :

- *Retroactively bounded events*: It is the usual case. For each event, valid time and transaction time have the following interrelationships $vt_e < tt_e \leq \Delta t + vt_e$ with $\Delta t > 0$. In specific terms, the event occurs in reality at vt_e, then it is recorded in DB at tt_e. Δt is fixed by the user and represent the necessary time to collect it, transfer it to the DB and record it.
- *Delayed retroactive events*: It corresponds to events whose temporal attributes have the following interrelationships $\Delta t < tt_e - vt_e$. This type of events occurs in two cases:1) when there is technical issue making difficult to deliver events to the DB. 2) to correct previous events that have been recorded into the DB.
- *Predictive events*: This case corresponds to events that are recorded in the DB before they occurs in reality ($tt_e \leq vt_e$), e.g a government tax rate modification which is always announced before it is applied so that concerned people make arrangements.

In order to handle these three types of events, Decision Insight implements two different approaches: *Live Mode* and *Late Data Handler*.

- *Live Mode* : This approach is the usual mode and is in charge of handling both *retroactively bounded events* and *predictive events*. Concretely, considering a *materialized continuous query*, the condition to schedule its execution for an interval of its rhythm is that all input data are available. Thus, for a rhythm interval $[vt_{begin}, vt_{end}[$, the system supposes that at $tt = vt_{end} + \Delta t$ all input data is available and schedules the computation. Δt must smaller than $vt_{end} - vt_{begin}$. Otherwise, the computation task queue fill rate will be faster than the computation rate.
- *Late Data Handler*: This mode is dedicated to *retroactive events*. When such type of events arrives, the system determines all *materialized continuous query* and *rhythm intervals* impacted. Then it schedules a their recomputation.

In the sequel, we restrict ourselves to the *live mode*.

3. DECISION INSIGHT

Decision Insight is a comprehensive data-intensive decision support system that combines both BI and BAM capabilities. It uses a dashboard as an user interface primitive, allowing analysts to visualize activity indicators and to navigate in time to understand how they evolve. A dashboard is made up of one or more graphical elements (diagrams, charts, datagrids, ...) referred to as *pagelets* in the sequel. Each graphical element displays data returned by an underlying query. An example of a pagelet is depicted in Fig. 2, where daily revenues for a company are displayed as a curve in a particular time range.

Implementing queries using a SQL-based language can be a very difficult activity, particularly for business managers with limited technical skills. Decision Insight provides an advanced graphical interface for rapid design of the complex queries related to BAM [3].
Decision Insight allows to specify in a graphical interface the main steps of the query optimization previously defined. This query rewriting is performed intuitively via the GUI by a decision maker. The process of implementing a pagelet is divided into two phases described below.

3.1 Designing Materialized Continuous Queries

First, the manager has to define all analyses on data streams that he wants to use in his pagelet.

Let us consider the following query: "What is the revenue achieved by the company every day from 01/01/1992 to 01/01/1993, considering DB at the most recent state?". The screenshot (Fig.3) shows a part of Decision Insight's user interface used to create a new analysis corresponding to Fig. 2. (A) indicates which rhythm do we want to link to the attribute. In our case we choose a one-day rhythm as we want to know the company's total revenue per day. (B) indicates the aggregation operation used to generate the analysis. (C) indicates the time-range to consider for the aggregation. In our example we fix at the last day. Finally (D) represents data inputs used to compute the analysis, which is the attribute "extendedPrice" of the LineItem. For each of these attributes, Decision Insight creates attributes via an underlying *materialized continuous query* in a transparent way for the user.

3.2 Designing On-Demand Queries

In the second phase, the manager chooses the form and the content that will be displayed on the pagelet. Fig.4 is a screenshot of Decision Insight's user interface for implementing a pagelet. (A) indicates the type of graphical element the manager wants to display, namely a historical curve. (B) indicates the time range of information to display on the pagelet. According to the query, we choose to display the whole current month. (C) indicates the information to be displayed. Based on the provided information, Decision Insight creates a pagelet and an underlying *on-demand query* in charge of updating the pagelet content (Fig.2).

4. EXPERIMENTS

4.1 Bi-temporal DB Benchmarks

A bi-temporal benchmark can be used in our case since it offers bi-temporal data that can be used to simulate real-time data. To the best of our knowledge, the TPC-BiH [17] is the most complete bi-temporal benchmark. TPC-BiH is an extension of TPC-H [11] and measures the performances of a DBMS used by a decision support system. The benchmark also includes a data generator producing a workload based on 9 categories of business transactions (New Order, Cancel Order, Update Stock, etc).

4.2 Database Populating

From the initial data produced by the TPC-BiH data generator, we generate a stream of events $< id, data, T >$. Each event corresponds to an updating instruction addressed to the database. id is the event type, e.g "insert a new order" or "insert a new customer". $data$ is the information handled by the event and T is the timestamp when the event occurred. The events are ordered according to the attribute T, so we can simulate a real-time workload. The initial TPC-BiH dataset has a size of 400MB. The generated data stream contains 3620761 events (Table 3).

We also introduce a scaling factor "*sf*" to fix the rate of the data stream. For the initial data stream $sf = 1$. All data streams with a

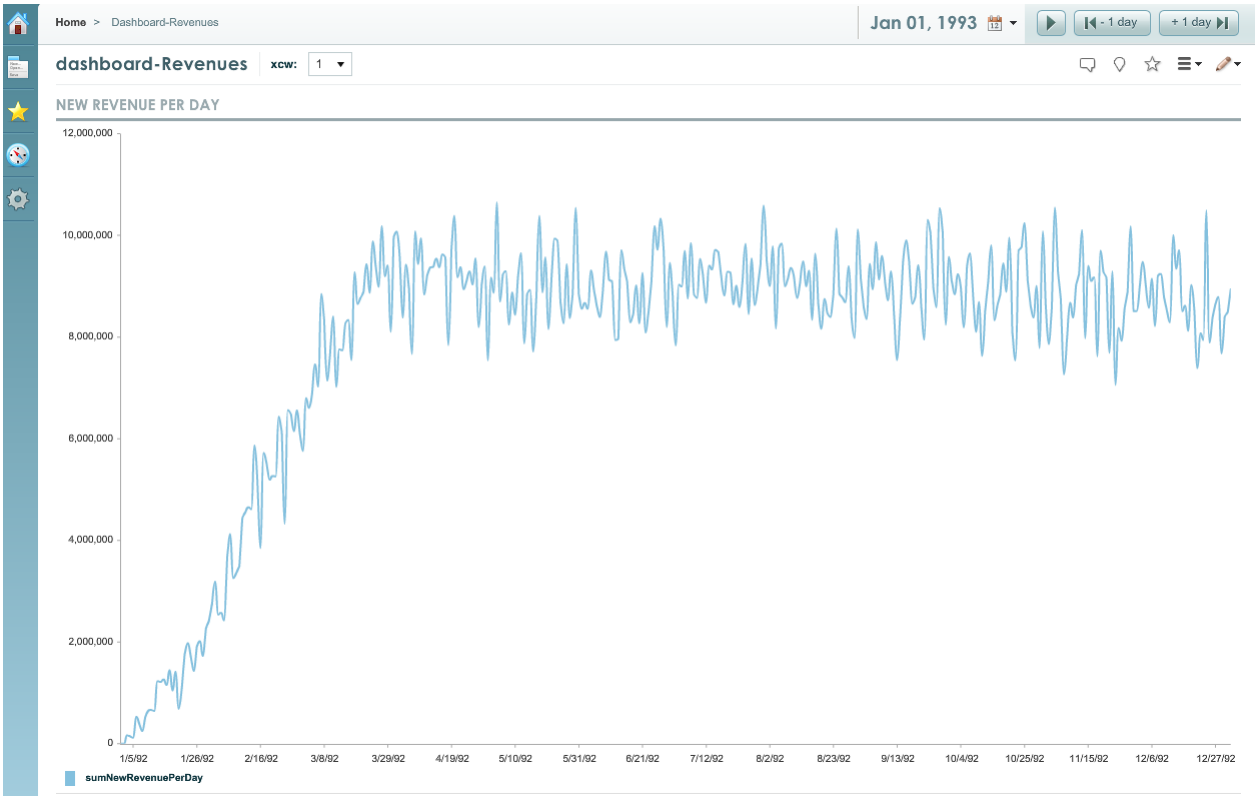

Figure 2: Snapshot of a pagelet displaying the daily evolution of revenues

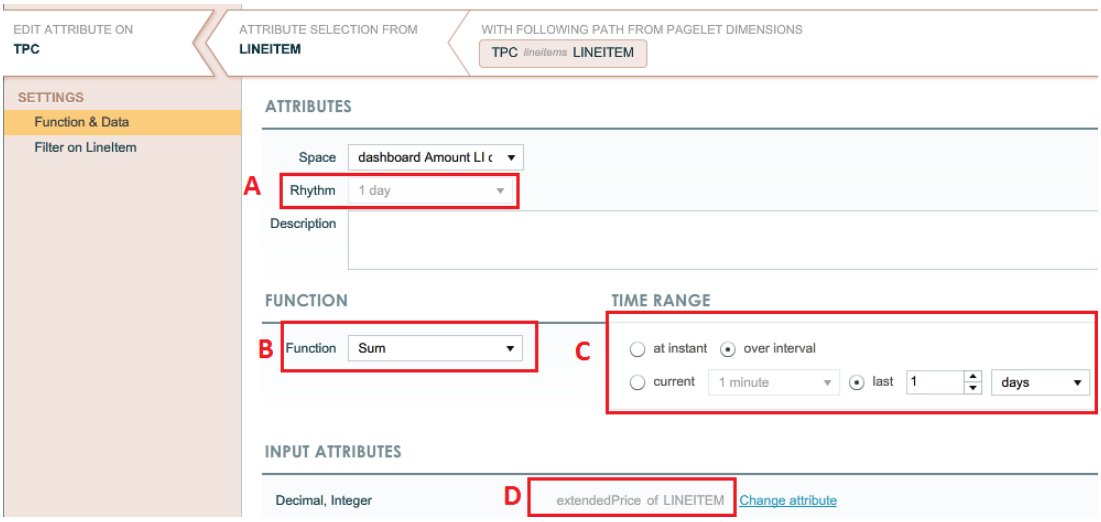

Figure 3: Decision Insight's interface to implement a continuous query

higher sf are generated by duplicating sf times each of its events.

4.3 Queries

We have implemented two examples of typical queries used in BAM. They are simple so that they can be easily expressed. Those queries are frequently executed by a GUI which requires rapid response times.

4.3.1 Query 1

This first query, (listing 11), aims at answering the following business question: "What is the sum of new revenues for the company every day from 1/1/1992 to now considering the most recent data?". We redefine this query as one *materialized continuous query* "Q1-Cont" (Listing 12) and one *on-demand query* "Q1-OnD" (Listing 13).

Listing 11: Q1: New Revenue per day

```
SELECT Ryhthm_1d.vtb as vtb, Ryhthm_1d.vte as vte,
```

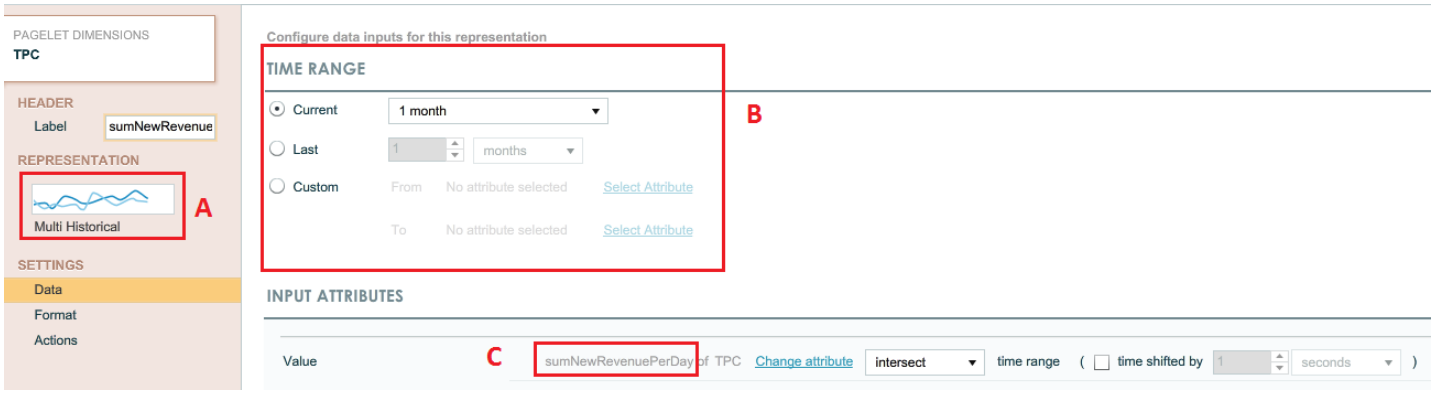

Figure 4: Decision Insight's interface for implementing an On-Demand query

Table 3: Number of operations per table

Relation	# of insertions	# of updates	# of deletions
Region	5	0	0
Nation	25	0	0
Supplier	1000	0	0
Part	20000	49861	0
Customer	164668	253430	0
Partsupp	80000	352391	0
Orders	348026	681103	8452
LineItems	939670	699310	22820

```
  SUM(extendedPrice) as totalRevenuePerDay
FROM LineItems, Ryhthm_1d
WHERE
  '01/01/1992'=<Ryhthm_1d.vtb AND
  Ryhthm_1d.vtb < '01/01/[YEAR]' AND
  Ryhthm_1d.vtb <= LineItems.vtb AND
  LineItems.vtb< Ryhthm_1d.vte
GROUP BY Ryhthm_1d.vtb;
```

Listing 12: Q1-Cont: New Revenue per day

```
SELECT SUM(extendedPrice) as totalRevenuePerDay,
  [vtInterval].begin as vtb, [vtInterval].end as vte
FROM LineItems
WHERE [vtInterval].begin <= vtb AND
  vtb < [vtInterval].end;
-- [vtInterval]: a rhythm interval
```

Listing 13: Q1-OnD: New Revenue per day

```
SELECT vtb, vte, aggr as newRevenuePerDay
FROM totalRevenuePerDay
WHERE '01/01/1992' <= vtb AND
  vtb < '01/01/[YEAR]';
```

4.3.2 *Query 2*

The query given in the (listing 14) aims at answering the following business question: "What is the number of orders per status for every day from 1/1/1990 to now considering the most recent data?". We redefine this query as one *materialized continuous query* "Q2-Cont" (Listing 15) and one *on-demand query* "Q2-OnD" (Listing 16).

Listing 14: Q2: Number orders per status and per day

```
SELECT Ryhthm_1d.vtb as vtb, Ryhthm_1d.vte as vte,
  SUM(extendedPrice) as totalRevenuePerDay
FROM Orders, Ryhthm_1d
WHERE
  '01/01/1992'=<Ryhthm_1d.vtb AND
  Ryhthm_1d.vtb < '01/01/[YEAR]' AND
  Ryhthm_1d.vtb <= order.vtb AND
  order.vtb< Ryhthm_1d.vte;
GROUP BY Ryhthm_1d.vtb, orderid;
```

Listing 15: Q2-Cont: Number of orders per status and per day

```
SELECT COUNT(*) as numberOrdersPerDay,
  [vtInterval].begin as vtb, [vtInterval].end vte
FROM Orders
WHERE [vtInterval].begin <= vtb AND
  vtb < [vtInterval].end
GROUP BY orderid;
```

Listing 16: Q2-OnD: Number of orders per status and per day

```
SELECT vtb, vte, numberOrdersPerDay
FROM Orderstatus
WHERE '01/01/1992' <= vtb AND
  vtb <'01/01/[YEAR]';
```

4.4 Experimental Results

In this section we present the results of experiments conducted to assess the performances of our approach. To do this, we compare system performances with and without using our optimization. We also present some measures of the overhead induced by our optimization. Experiments were executed on a physical machine which runs an Ubuntu 10.04, equipped with 12GB of RAM, an Intel i7 processor with 8 cores at 2.8GHz and a 4TB of RAID storage.

4.4.1 *Response Time*

In this test, we point out the interest of our approach in reducing Decision Insight's time response. We run two experiments: in the first, we evaluate the impact of the time range size on the execution time of Q1 and Q2 ($sf = 1$) while in the second, we vary the value of sf.

Fixed Scalar Factor

We inject stream concerning the period [1/1/1992, 1/1/1999[. At the beginning of each new year of the simulation period, we execute once Q1 and Q2 using a new value of the parameter "[YEAR]". We compare two versions of each query: the optimized version, using our approach based on continuous queries (Q1-OnD and Q2-OnD), and a classical version, where the result is computed whenever the

query arrives (Q1 and Q2). We collect the execution times of these queries and represent them on Fig.5.

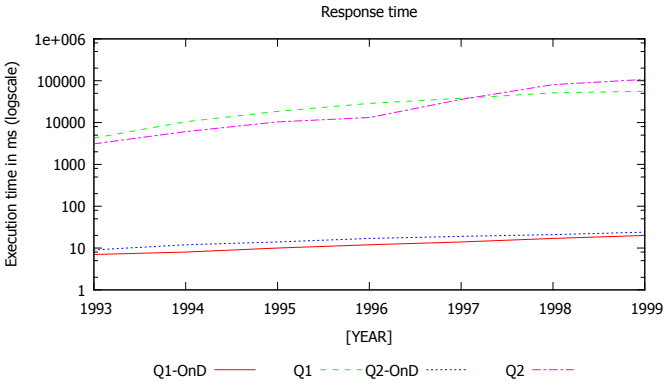

Figure 5: Query response time while varying window size

We can notice that optimized versions of queries outperform the rest by at least a factor of 100. For each day of the query interval, Q1-OnD accesses one value which is the materialized result of the underlying continuous query. Q1, however, accesses the original data, i.e about 200 items for each day.

Varying Scalar Factor

In this experiment, we assess our approach when we vary the data stream rate. The experimental conditions are similar to the previous test. We vary the value of sf from 1 to 6. For each value of sf, we inject the stream that concerns the period [1/1/1992, 1/1/1995[. Following injection, queries Q1 and Q2 are successively executed with and without optimization. The queries are executed with [YEAR]=1995.

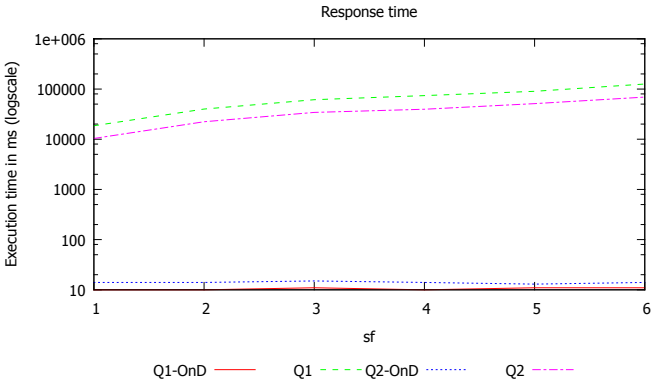

Figure 6: Query response time while varying data stream rate

When the data stream rate increases, the query execution time of the non-optimized queries increases, else it remains stable.

4.4.2 Precomputation Overhead

Previous tests demonstrate the advantage of our approach in reducing the response time of the system. However it induces a CPU and disk storage overhead.

Fixed Scalar Factor

The experimental conditions are similar to the test for response time/fixed scalar factor, except that we use only Q1. For each day of the simulated period, we collect the CPU time of Q1-Cont. We also collect the CPU time to execute Q1-OnD and Q1. Fig.7 shows the results of the experiment: one curve represents the CPU consumption of Q1, while the other is the sum of the CPU consumption of Q1-cont and Q1-OnD.

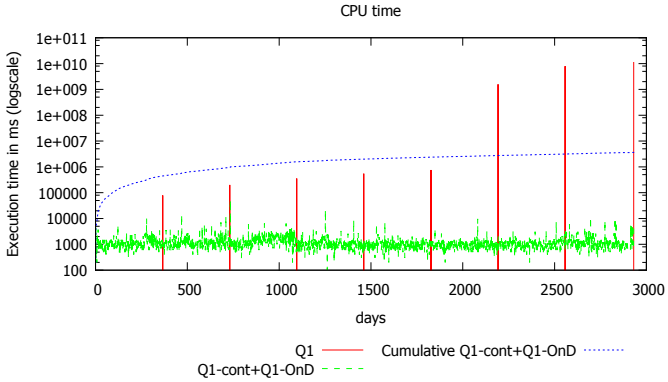

Figure 7: Continuous query computation overhead

It appears that the optimized approach requires a CPU overhead throughout the simulation time. However, it smooths the CPU consumption curve and avoids peaks at query time and thus system overload. We also notice that as from the 2000th day of simulation, the CPU Q1 cost is at least 100 times greater than the CPU required to compute Q1-cont and Q1-OnD. This means that for a query using a large time interval (6 years), the overhead induced by our approach has no impact on query processing performance.

Varying Scalar Factor

In this experiment we assess the cost of our approach as we vary the stream rate using the parameter sf. For each stream, we first inject data stream corresponding to the period [1/1/1992, 1/1/1995[, then we execute Q1 with [YEAR]= 1995. We collect the CPU time to perform Q1 and Q1-OnD. We also collect the average CPU time of Q1-cont per day and the total sum of all CPU time consumption of Q1-cont during the simulation. The results are represented in (Fig.8).

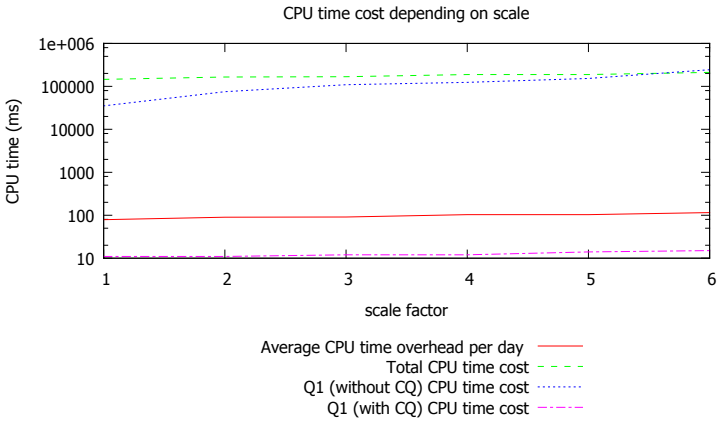

Figure 8: CPU time as a function of scale factor

Whenever $sf >= 6$, our approach does not have any CPU overhead.

4.4.3 *Concurrent Query Execution*

In this test, we simulate several users interacting with the system. We have performed two experiments: one where we vary the number of concurrent queries and another where we vary sf.

Fixed Scalar Factor

In this experiment, we use a dataset where $sf = 1$. We first populate the system with data corresponding to the period [1/1/1992, 1/1/1999[. Following system population, we execute concurrently several instances of the query Q1 with [YEAR]=1999. Then we get the CPU time required to execute them all. Fig.9 shows the results of this experiment where we varied the number of simultaneous executed queries from 1 to 20.

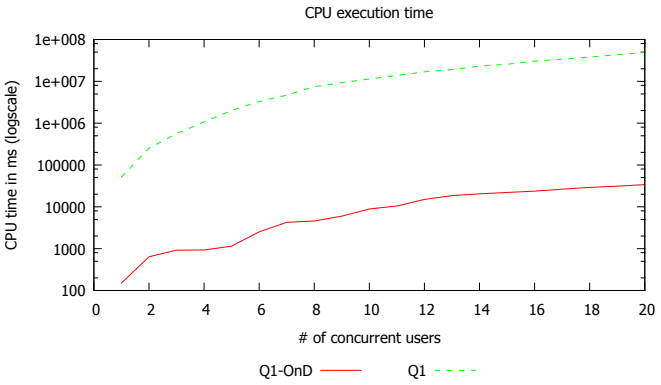

Figure 9: Concurrent query execution

As shown in Fig.9, our approach is quite adapted for execution of concurrent queries.

Varying Scalar Factor

In this experiment, we explore the impact of the data stream throughput on the execution of concurrent queries. We first populate the system with data that corresponds to the period [1/1/1992, 1/1/1995[. Then we execute 10 concurrent queries, corresponding to 10 users.

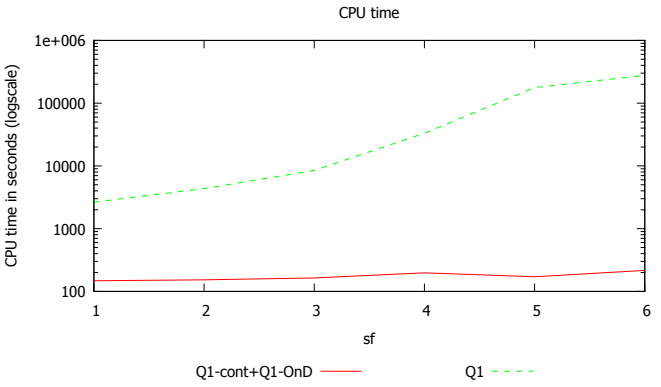

Figure 10: Concurrent query execution while varying sf

As in the previous test, we observe the advantage of the proposed optimization due to the increasing number of queries accessing the results of continuous queries as historical data.

5. RELATED WORKS

To the best of our knowledge, Chandrasekaran and Franklin were the first to address the topic of combining real-time data with historical data [9] in the academic field. They noted that the main performance issue for those systems was the I/O cost induced by gathering historical data, which decreases drastically live data stream processing performance. They proposed a framework using some data reduction techniques for historical data to limit I/O cost (see also [5]). Their framework data reduction level to be adapted with respect to current available resources. They defined three approaches to perform data reduction techniques: *OnWriteReplicate*, *OnReadModify* and *Hybdrid approach*. The first approach is based on the fact that random disk I/Os are expensive. Data reduction is performed continuously as soon as data is collected by the system. Thus at query time, the global query can access pre-computed results when needed. Nevertheless, pre-computed results can never be accessed by global queries. The second approach consists in performing the data reduction at query time only. The price to be paid can be very high for delivering timely information. The third approach combines the two previous approaches and shares the work between data arrival and query time. In this approach there is a single copy of the stream stored on disk and divided into separate batches. Each batch is divided into a fixed number of blocks. Tuples are randomly inserted in different blocks of the current batch. Once one block is filled, the entire run is flushed on disk. At query time, the system only accesses a fraction of blocks of runs according to a sampling rate.

With respect to our contribution, we use the OnWriteReplicate approach which turns out to be effective for BAM applications. Postponing query processing when the system is at a lower load is not a new idea, see for instance [10] in a BI context. Load shedding techniques have been investigated in DSMS to come up with high data throughput [32]. Such techniques are quite different from the proposition made in this paper. The application context of Business Activity Monitoring does not fit perfectly into either of those two main areas, i.e. neither the data volume is expected to be as large as in data-warehouse applications, nor the data throughput is expected to cause the system to collapse as in some DSMS applications. BAM applications lie somewhere between these two kinds of applications.

The topic of materialized views has been widely discussed in DBMS in general [21, 25] and in real-time DBMS in particular [2], since materialized views are often considered to reduce query execution time. These works mainly suppose the case of *snapshot views*, i.e views that maintain only the last state of data. This assumption is not sufficient in our case since we need to store its whole history. There were some works concerning temporal materialized views [34, 14, 13]. Yet the do not address the real-time case.

The concept of *rhythm* is close to the concept of *Granularity* that has already been defined in litterature [6] and has been implemented in some products. We can quote Teradata' DBMS that extends SQL with the key word "EXPAND ON". There is also Kx System's product KDB+ whose processing language supports such feature. Many commercial products addressing business monitoring exist, among which we quote Kx System [4], spunk and main DBMS players like DB2, Oracle and Teradata. However, as far as we know, no one covers all features offered by Decision Insight. KDB+, for example, handles both real-time and historical data and is based on a column-store database dedicated to handle time-series data. Yet, it only supports valid time dimension.

6. CONCLUSION

[4] http://kx.com/

In this paper, we have introduced Decision Insight, a comprehensive data-intensive decision support system that combines both Business Activity Monitoring (BAM) and business intelligence capabilities.
Given a bi-temporal query on historical and live data, the optimization technique of Decision Insight is based on a decomposition of the initial query into several continuous queries plus one on-demand query, which accesses the materialized data of previous queries. This technique has been implemented in Decision Insight, and experiments have been conducted on the TPC-BiH benchmark (variant of the TPC-H benchmark). Results show that Decision Insight is able to deliver very fast responses to decision-makers, which is a very strong requirement in BAM applications.

In future works, we aim to study multi-query optimization techniques to share as much as possible the processing of a set of complex bi-temporal queries [26, 8].

7. REFERENCES

[1] D. J. Abadi, S. R. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980. ACM, 2008.

[2] B. Adelberg, H. Garcia-Molina, and B. Kao. Applying update streams in a soft real-time database system. In *ACM SIGMOD Record*, volume 24, pages 245–256. ACM, 1995.

[3] A. Ait Ouassarah, N. Averseng, X. Fournet, J.-M. Petit, R. Revol, and V.-M. Scuturici. Understanding Business Trends from Data Evolution with Tornado (demo). In *Int. Conf. on Data Engineering (ICDE 2015)*. IEEE, May 2015.

[4] A. Arasu, S. Babu, and J. Widom. An abstract semantics and concrete language for continuous queries over streams and relations. 2002.

[5] D. Barbar'a, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, et al. The new jersey data reduction report. In *IEEE Data Engineering Bulletin*. Citeseer, 1997.

[6] C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, and X. S. Wang. A glossary of time granularity concepts. In *Temporal databases: Research and practice*, pages 406–413. Springer, 1998.

[7] P. A. Boncz, M. Zukowski, and N. Nes. Monetdb/x100: Hyper-pipelining query execution. In *CIDR*, volume 5, pages 225–237, 2005.

[8] G. Candea, N. Polyzotis, and R. Vingralek. A scalable, predictable join operator for highly concurrent data warehouses. *Proceedings of the VLDB Endowment*, 2(1):277–288, 2009.

[9] S. Chandrasekaran and M. Franklin. Remembrance of streams past: Overload-sensitive management of archived streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 348–359. VLDB Endowment, 2004.

[10] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[11] T. P. P. Council. Tpc-h benchmark specification. *Published at http://www.tcp.org/hspec.html*, 2008.

[12] C. Dyreson, F. Grandi, W. Käfer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J. F. Roddick, N. L. Sarda, M. R. Scalas, A. Segev, R. T. Snodgrass, M. D. Soo, A. Tansel, P. Tiberio, and G. Wiederhold. A consensus glossary of temporal database concepts. *SIGMOD Rec.*, 23(1):52–64, Mar. 1994.

[13] H. V. Jagadish, I. S. Mumick, and A. Silberschatz. View maintenance issues for the chronicle data model. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 113–124. ACM, 1995.

[14] C. S. Jensen, L. Mark, and N. Roussopoulos. Incremental implementation model for relational databases with transaction time. *Knowledge and Data Engineering, IEEE Transactions on*, 3(4):461–473, 1991.

[15] C. S. Jensen and R. Snodgrass. Temporal specialization and generalization. *Knowledge and Data Engineering, IEEE Transactions on*, 6(6):954–974, 1994.

[16] C. S. Jensen and R. T. Snodgrass. Temporally enhanced database design., 2000.

[17] M. Kaufmann, P. M. Fischer, N. May, A. Tonder, and D. Kossmann. Tpc-bih: A benchmark for bitemporal databases, 2013.

[18] M. Kaufmann, P. Vagenas, P. M. Fischer, D. Kossmann, and F. Färber. Comprehensive and interactive temporal query processing with sap hana. *Proceedings of the VLDB Endowment*, 6(12):1210–1213, 2013.

[19] J. Krämer and B. Seeger. *A temporal foundation for continuous queries over data streams*. Univ., 2004.

[20] K. Kulkarni and J.-E. Michels. Temporal features in sql: 2011. *ACM SIGMOD Record*, 41(3):34–43, 2012.

[21] P.-Å. Larson and H. Yang. *Computing queries from derived relations: Theoretical foundation*. Citeseer, 1987.

[22] D. Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002.

[23] D. W. McCoy. Business activity monitoring: Calm before the storm. *Gartner Research*, 2002.

[24] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein. Enabling real-time querying of live and historical stream data. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*, pages 28–28. IEEE, 2007.

[25] H. A. Schmid and P. A. Bernstein. A multi-level architecture for relational data base systems. In *Proceedings of the 1st International Conference on Very Large Data Bases*, pages 202–226. ACM, 1975.

[26] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

[27] R. Snodgrass. The temporal query language tquel. *ACM Transactions on Database Systems (TODS)*, 12(2):247–298, 1987.

[28] R. Snodgrass, M. Böhlen, C. Jensen, and A. Steiner. Adding valid time to sql/temporal. ansi x3h2-96-151r1, iso–ansi sql/temporal change proposal, iso. Technical report, IEC JTC1/SC21/WG3 DBL MCI-142, 1996.

[29] R. T. Snodgrass, M. H. Böhlen, C. S. Jensen, and A. Steiner. Adding transaction time to sql/temporal. *ISO-ANSI SQL/Temporal Change Proposal, ANSI X3H2-96-152r ISO/IEC JTC1/SC21/WG3 DBL*, 1101:143, 1996.

[30] R. T. Snodgrass, J. Gray, and J. Melton. *Developing time-oriented database applications in SQL*, volume 42. Morgan Kaufmann Publishers San Francisco, 2000.

[31] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very*

large data bases, pages 553–564. VLDB Endowment, 2005.

[32] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 309–320. VLDB Endowment, 2003.

[33] H. J. Watson and B. H. Wixom. The current state of business intelligence. *Computer*, 40(9):96–99, 2007.

[34] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *Advances in Database TechnologyÑEDBT'98*, pages 389–403. Springer, 1998.