An Approach for Alert Raising in Real-Time Data Warehouses

Maximiliano Ariel López*, Sergi Nadal**,
Mahfoud Djedaini***, Patrick Marcel***, Verónika Peralta***, Pedro Furtado****

*École Centrale Paris, maxilopez@economicas.uba.ar,
**Universitat Politècnica de Catalunya, snadal@essi.upc.edu
***Université de Tours, firstname.lastname@univ-tours.fr
****University of Coimbra, pnf@dei.uc.pt

Abstract. This work proposes an approach for alert raising within a real-time data warehouse environment. It is based on the calculation of confidence intervals for measures from historical facts. As new facts arrive to the data warehouse on a real-time basis, they are systematically compared with their appropriate confidence intervals and alerts are raised when anomalies are detected. The interest of this approach is illustrated using the particular real world use case of technical analysis of stock data.

1 Introduction

Traditional data warehousing architectures assume offline periods in order to run the costly ETL processes that move and transform data coming from operational sources. Currently, many organisations have the requirement of analysing their information in a real-time manner. For instance, in the stock markets domain, technical analysis allows modelling market behaviour and predicting tendencies. Monitoring markets and quickly detecting deviations from the expected behaviour allow analysts to face abrupt changes. Other domains where real-time analysis is desirable include energy production, traffic and network monitoring.

In most cases, the aforementioned requirement cannot be satisfied with the classic data warehouse and ETL architectures. To enable near real-time analysis based on the most recent information, data warehouse architectures have been extended or adapted (Ferreira and Furtado, 2013; Ferreira et al., 2013; Waas et al., 2013; Jörg and Dessloch, 2009; Santos and Bernardino, 2008). In such systems, how data is loaded and the frequency in which this process is executed change. In (Ferreira and Furtado, 2013), an integrated architecture is proposed, which implements a real-time data warehouse without data duplication. It is composed of three main components: the Dynamic Data Warehouse (D-DW), the Static Data Warehouse (S-DW) and the Merger, with additional components providing real-time ETL. The main idea is to load new data into the D-DW, an in-memory database that holds the most recent information and provides fast integration. On the other side, the largest volume of historical data is stored in the S-DW. Integration between D-DW and S-DW is done through classical offline procedures. After this integration is done, the D-DW is emptied. Queries are handled by the Merger component which checks whether they have to be dispatched to the D-DW, S-DW or both. In the latter case, it also combines the results of both components into a final answer.

In this paper, we present an approach for alert raising in a real-time data warehouse that assumes the architecture proposed by (Ferreira and Furtado, 2013). The key idea involves leveraging user traces (past queries recorded in a query log) to build an in-memory summary of the S-DW, and then checking this summary against the data in the D-DW to raise alerts. More precisely, we assume that user traces express sets of facts that need to be monitored. In an offline phase, for each of these queries, we construct an in-memory structure (called *baseline* from now onwards) recording a confidence interval for the facts contributing to each cell (i.e., aggregate) of the query result over the S-DW. Confidence intervals are built using the bootstrap method (Efron and Tibshirani, 1986). This method is particularly well adapted to a real-time context, in the sense that it allows estimating population parameters based on a sample. In our case, the unknown population is linked to the complete answer of the query (not yet known since new data arrives continuously in the D-DW), and the sample is the current answer to this query. In the online phase of our approach, new data loaded into the D-DW are compared to the appropriate baselines. This comparison is used to raise alerts.

If the main idea of our anomaly detection approach is quite general, it is particularly well adapted to a data warehousing context, for the following reasons. Firstly, our approach leverages a specific real-time data warehouse architecture. Secondly, it is analyst-tailored in the sense that anomalies are checked using an analyst navigation history. Finally, it uses a specificity of the multidimensional model since baselines are basically results of multidimensional queries, recording at a particular granularity level confidence intervals for primary facts.

The outline of the paper is the following. Next section motivates the approach with an example based on a prototype data warehouse storing stock exchange market data that we implemented. Section 3 gives basics definitions for the formal description of the approach exposed in Section 4. Results of our experiments are presented in Section 5 while Section 6 discusses related work. Finally, Section 7 concludes and draws perspectives.

2 Motivating Example

Our approach is inspired in technical analysis of stock exchange markets due to its dynamic nature. It is worth mentioning that our aim is not to provide a replacement to other adapted solutions like high-frequency trading systems, but to illustrate and test the dynamics, features and problems of a real-time data warehouse. We think that this case poses an interesting scenario of streaming data with a remarkable volume.

Therefore, we consider a data warehouse allowing the analysis of security trading transactions (i.e. quotes). The schema of the data warehouse is depicted in Figure 1 using DFM notation (Golfarelli and Rizzi, 2009).

Quotes are associated with the following dimensions: markets (geographical information), securities (issuing organization, currencies, type of activity), date and time information. As far as measures are concerned, five items are considered: *open* (i.e. the price at the beginning of the period), *high* (i.e. the maximum price seen), *low* (i.e. the minimum price seen), *close* (i.e. the price at the end of the period) and *volume* (i.e. the quantity of stock traded so far).

We take into account a business analyst interested in monitoring some companies and markets. Let us suppose the user devises queries for answering the following requirements: Euro-

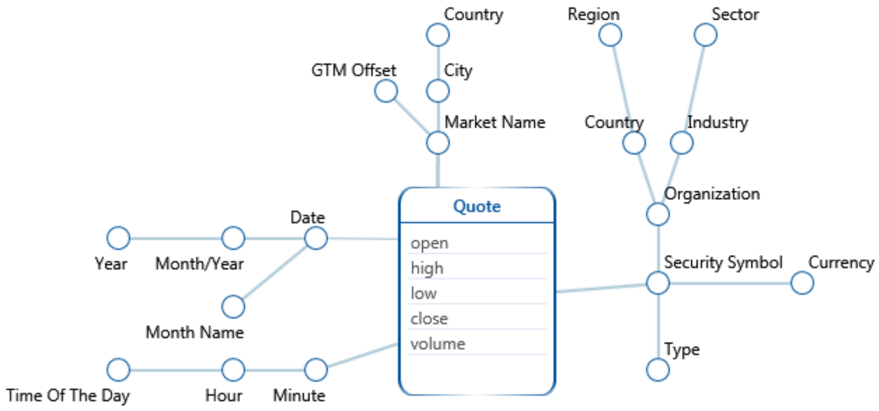

FIG. 1 – *Data warehouse schema for the illustrative example*

pean Health-Care Companies (Q_1), US Health-Care Companies (Q_2), Semiconductors firms in NASDAQ by Year (Q_3), and Water Supply firms by Year (Q_4). These pieces of information would appear in a query log, as depicted in the following table, using MDX notation (for illustrative purposes we assume only the measure *close* is used in each query):

Query	Group By Set	Filters
Q_1	[Security.Geography].[Organisation] [Market.Geography].[Market Name]	[Security.Geography].[Region].[Europe] [Security.Activity].[Sector].[Health Care]
Q_2	[Security.Geography].[Organisation] [Market.Geography].[Market Name]	[Security.Activity].[Sector].[Health Care] [Security.Geography].[Country].[USA]
Q_3	[Security.Activity].[Security Symbol] [Date.DateMonthYear].[Year]	[Market.Geography].[Market Name].[NASDAQ] [Sector.Activity].[Industry].[Semiconductors]
Q_4	[Security.Activity].[Security Symbol] [Date.DateMonthYear].[Year]	[Sector.Activity].[Industry].[Water Supply]

In order to check for abnormal values and raise alerts, we first build baselines of historical data and then, when fresh data arrive, we compare them to the baselines. By replaying the queries against the S-DW (the historical component), we compute the baseline related to each query, i.e. confidence intervals for the values of measures of primary facts. For instance, Q_1's baseline would store, for each combination of European Healthcare organisations and market names, the confidence interval corresponding to each measure of the concerned primary fact. A sample of this baseline for *close* measure might be:

	[Market.Geography].[Market Name]. [NASDAQ]
[Security.Geography].[Organisation].[Shire plc]	[143.87 - 166.317]
[Security.Geography].[Organisation].[Grifols, S.A.]	[29.414 - 31.765]

As new data arrive into the D-DW (the real-time component), the system automatically checks individual facts against the in-memory baselines. The following fact inserted into D-DW might then trigger an alert as its closing price lies outside of the confidence interval:

date_id	minute_id	market_name	security_symbol	open	high	low	close	volume
...	...	NASDAQ	Shire plc	149.10	149.10	142.50	143.70	206026

3 Preliminaries

In this section, we introduce the theoretical basis that later will be used in our approach. Firstly, in Section 3.1 we introduce basic definitions such as multidimensional schema, queries and baselines, the in-memory data structure whose goal is to provide an overview of the confidence intervals in the data warehouse. In Section 3.2 we briefly present bootstrapping, a statistical method based on data re-sampling that can provide estimations of the sampling distribution. The key part is that, this estimation is akin to the real distribution of the entire population.

3.1 Multidimensional Schema, Queries, and Baselines

We now introduce our formal framework. To keep the formalism simple, we consider cubes under a ROLAP perspective, described by a star schema (Kimball, 1996).

We consider the classic definition of hierarchies, where a hierarchy h_i is a set $Lev(h_i) = \{l_0, \ldots, l_d\}$ of levels together with a *roll-up* total order $\succeq_{h_i}$ of $Lev(h_i)$.

A *multidimensional schema* (or, briefly, a *schema*) is a triple $\mathcal{M} = \langle A, H, M \rangle$ where:

- $A = \{l_1, \ldots, l_m\}$ is a finite set of *levels*, whose domains are assumed pairwise disjoint,
- $H = \{h_1, \ldots, h_n\}$ is a finite set of *hierarchies*, such that each level l_i of A belongs to al least one hierarchy of H.
- $M = \{m_1, \ldots, m_p\}$ is a finite set of *measure* attributes.

Given a schema $\mathcal{M} = \langle A, H, M \rangle$, let $g \in Lev(h_1) \times \ldots \times Lev(h_n)$ be a *group-by set* of $\mathcal{M}$. In what follows we will use the classical partial order on group by sets, defined by: $g \succeq g'$, if $g = \langle l_1, \ldots, l_n \rangle, g' = \langle l'_1, \ldots, l'_n \rangle$ and $\forall i \in [1, n]$ it is $l_i \succeq_{h_i} l'_i$. We note g_0 the most specific (finest) group by set. A coordinate $coord_i$ in a hierarchy h_i is a pair $\langle l, m \rangle$ where l is a level and m is a member of l. Given a group by set $g = \langle l_1, \ldots, l_n \rangle$, let $Dom(g) = Dom(l_1) \times \ldots \times Dom(l_n)$ be the set of all combinations of members at granularity g. An element of $Dom(g)$ is called a reference. For this schema, a fact of granularity g is a pair $\langle ref, meas \rangle$, where ref is a reference and $meas$ is a set of measure values. The facts of granularity g_0 are called primary facts.

Example 3.1 *In the example of Section 2, the schema $\mathcal{M} = \langle A, H, M \rangle$ is:*

- $A = \{MarketName, City, Country, Currency, Date, \ldots\}$
- $H = \{(\{Year, Month/Year, Date\}, \succeq_{Date}), \ldots\}$
- $M = \{open, high, close, low, volume\}$

The finest group by set of $\mathcal{M}$ is $g_0 = \{Minute, Date, MarketName, SecuritySymbol\}$. A coordinate is $(MarketName, NASDAQ)$. A reference is $\{Minute, Date, MarketName, SecuritySymbol\}$.

A *query* over schema $\mathcal{M} = \langle A, H, M \rangle$ is a triple $q = \langle G, P, Meas \rangle$ where:

1. G is the query group-by set;
2. $P = \{p_1, \ldots, p_n\}$ is a set of Boolean predicates, one for each hierarchy, whose conjunction defines the *selection predicate* for q; they are of the form $p_i = (l_i \in V_i)$, with l_i a level and V_i a set of members.
3. $Meas \subseteq M$ is the measure set whose values are returned by q.

The answer to a query $q = \langle G, P, Meas \rangle$ over a schema $\mathcal{M}$ the a set of facts of granularity G whose references are in $V_1 \times \ldots \times V_n$. A log L is a finite set of queries, noted $L = \{q_1, \ldots, q_p\}$. A monitoring query set $MQ = \langle q_1, \ldots, q_k \rangle$ is a subset of L.

In our approach, we assume the existence of two data cubes $\mathcal{D}_{RT}$ and $\mathcal{D}_{HIST}$, namely the real-time component and the historical component, that both share the same schema $\mathcal{M}$. Information in $\mathcal{D}_{RT}$ is stored in the D-DW component, as described in (Ferreira and Furtado, 2013) it only contains the most recent data. On the other hand $\mathcal{D}_{HIST}$ is stored in the S-DW component, which contains all historical data.

In what follows, for a given query q, we use $q(\mathcal{D}_{RT})$ (resp. $q(\mathcal{D}_{HIST})$) to denote the set of facts being the answer to query q over $\mathcal{D}_{RT}$ (resp. over $\mathcal{D}_{HIST}$). The log L considered in our case only applies to the queries over the historical cube $\mathcal{D}_{HIST}$. Finally, we assume the existence of a set of baselines $\mathcal{B} = \{b_1, \ldots, b_n\}$ to record properties of measure values for a given granularity level.

Definition 3.1 (Baseline) *Given a schema $\mathcal{M} = \langle A, H, M \rangle$ and a query $q = \langle G, P, Meas \rangle$ over $\mathcal{M}$, with $P = \{(l_1 \in V_1), \ldots, (l_n \in V_n)\}$, a baseline for q over $\mathcal{M}$ is a set of facts of granularity G, each of the form $\langle ref, cell \rangle$, where ref is a reference of $V_1 \times \ldots \times V_n$ and cell is a set of triplets of the form $(meas, \hat{\theta_1}, \hat{\theta_2})$, $meas$ being a measure of $Meas$ and $(\hat{\theta_1}, \hat{\theta_2})$ being the confidence interval for the measure.*

Example 3.2 *Retaking the previous example, the query Q_1 is represented as (forgetting about the All levels):*

- $G = \{Organisation, MarketName\}$
- $P = \{(Region \in \{Europe\}), (Sector \in \{HealthCare\})\}$
- $Meas = \{close\}$

After the inclusion of Q_1 in MQ, its baseline can be computed. One of its facts is $(\{$Shire plc,NASDAQ$\}, \langle close, 143.87, 166.31 \rangle)$.

3.2 The Bootstrap in a Nutshell

So far, we have been assuming the existence of a statistical method on which the calculation of confidence intervals $[\hat{\theta_1}, \hat{\theta_2}]$ is based on. This specific method is called bootstrap (Efron and Tibshirani, 1986), a computationally-intense non-parametrical algorithm that provided a sample $x = (x_1, \ldots, x_n)$ from an unknown probability distribution F, gives an estimation parameter of some population parameter $\theta = t(F)$ from x. This estimation, $\hat{\theta}$, is also known as the *plug-in estimator*, defined as $t(\hat{F})$. For this paper, we are interested in the *plug-in estimator* of the mean, which is defined by $\hat{\mu} = \int x d\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} X_i = \overline{X}_n$. Many different applications can be derived from this method including confidence intervals, which are of our interest in this paper. Bootstrap is a proven valid method with its grounds on the prominent Central Limit Theorem (CLT), (Efron and Tibshirani, 1994).

The main idea behind bootstrap is to repeatedly calculate the *plug-in estimator* a certain number of times b, where this value b lays in the order of magnitude of hundreds or thousands. Each pass on which this estimator is calculated is a sample x^*, i.e. a portion, of the real population, usually the 2.5% - 5% . One can later order this results $T_1, \ldots, T_b$ in order to get, for instance, the aforementioned confidence intervals $[\hat{\theta_1}, \hat{\theta_2}]$.

For example, let us assume that we have a dataset sample x consisting of 100 elements comprised within the range $[1, 10]$ for which we do not know in advance their distribution. The first step of the bootstrapping algorithm consists of performing a high number of passes through the dataset. In each pass we will be taking a random sample with replacement (every element $x[i]$ has a probability $\frac{1}{100}$ to be selected) of 5% from the total, hence samples of size 5.

For each of the b passes, the *plug-in estimator* T_i for the mean will be calculated, this is $\hat{\mu_i}$.

- $x_1 = 3, 5, 4, 3, 8 \rightarrow \hat{\mu}_1 = 4.6$
- $x_2 = 9, 3, 4, 2, 1 \rightarrow \hat{\mu}_2 = 3.8$
- $\ldots$
- $x_{b-1} = 4, 3, 1, 2, 1 \rightarrow \hat{\mu}_{b-1} = 2.2$
- $x_b = 2, 2, 1, 4, 6 \rightarrow \hat{\mu}_b = 3$

In order to retrieve the confidence interval from the set $\{\hat{\mu}_1, \ldots, \hat{\mu}_b\}$, we just need to order it in an ascending manner: $[2.2, 3, \ldots, 3.8, 4.6]$ to further retrieve the 5% and 95% percentiles as the bounds for a confidence interval with a 90% confidence. Hence, the resulting confidence interval $[\hat{\theta_1}, \hat{\theta_2}]$ is $[3 - 3.8]$.

4 Approach

This section details our approach for automatically detecting, in a real-time DW, anomalies that could be of interest for analysts. Our approach is user tailored in the sense that it takes into consideration user queries to decide whether an alert has to be raised or not.

4.1 General Architecture

Let us recall that, following the real-time architecture of (Ferreira and Furtado, 2013), historical data is stored in the S-DW (on disk), while most recent data is stored in D-DW (in memory). Following the underlying bipartite physical architecture, our approach is mainly made of two phases, as illustrated by Figure 2. Even though we use the previously described architecture, it is not the purpose of this paper. Our contribution is not about how to design a RT-DW, but rather how to design and implement an anomaly detection algorithm based on an already available RT-DW architecture.

Coming back to the two phases, a first phase, referred to as the offline phase, consists of computing baselines over S-DW, by means of the bootstrap method. For each query selected from the query log, a confidence interval is obtained for every reference and measure of the baseline. To some extent, that interval stands for a normality range for all the primary facts that have been considered for computing the answer to the monitored query. This offline phase is processed periodically; refreshment frequency is a function of the number of new facts, the

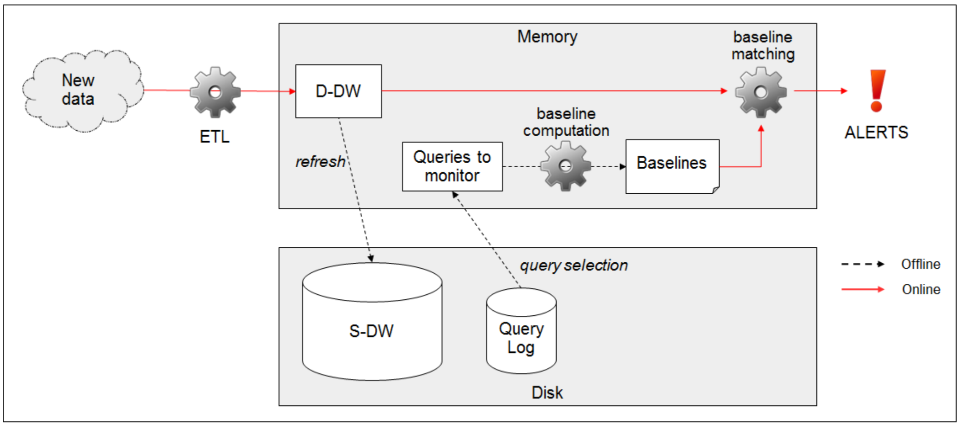

FIG. 2 – *Overall architecture*

number of primary facts contributing to query result and the size of query result. Note that the set of queries to monitor can be automatically selected from the query log (e.g. based on query frequency or user interaction) or can be explicitly stated by the user. Our approach is independent of the way the queries are selected.

The second phase, referred to as the online phase, consists of analysing each fresh fact as it arrives in the D-DW, to look for anomalies. To do so, we identify the baselines that cover this fact, i.e., the baselines to which this fact would have contributed to if it was available at the baseline computation time. The intuition of anomaly detection is that, a fact is flagged as abnormal for a given measure, regarding a given baseline, if the value of the measure for this fact is out of the bounds of the confidence interval located in the corresponding baseline cell.

We distinguish three levels of anomaly detection: baseline cell, baseline, and overall group of baselines. At baseline cell level, we count the number of new facts covered by that cell that are out of the confidence interval of at least one of the measures. We also maintain a counter for each baseline, that is incremented whenever a fact is detected as abnormal regarding any of the baseline cells. Finally, a global counter counts the number of anomalies within the whole group of baselines. Alerts are raised at a given level when the ratio of anomalies on a given period of time reaches a user defined threshold.

As depicted in Figure 2, the online steps of our algorithm mainly act upon components located in memory. Indeed, the baseline matching operation only involves most recent data and baselines, that are both of relatively small size and are stored in memory. These two aspects make this operation very fast in terms of computation time. Another important point is that baselines are a kind of summary of the S-DW, and they are not storage consuming. This fact is important and supports the feasibility of our approach, as it does not require unrealistic amounts of memory.

4.2 Computing Baselines

The first, and only offline phase of our approach, is to compute baselines from the S-DW, as shown in Algorithm 1. The idea of the process is, for each query to monitor (line 2), to

Algorithm 1 Computing Baselines

```
INPUT $MQ$: monitoring query set, $\mathcal{D}_{HIST}$: historical cube, $DWMeas$: DW measure set,
    $b$: number of bootstrap replications, $s$: sampling percentage, $f$: bootstrap estimator, $\alpha$:
    confidence interval width
OUTPUT $\mathcal{B}$ set of baselines
1: $\mathcal{B} = \emptyset$
2: for $q = \langle g, \{(l_1 \in V_1), \ldots, (l_n \in V_n)\}, QMeas \rangle \in MQ$ do
3:     for $m \in DWMeas$ do
4:         for $ref \in V_1 \times \ldots \times V_n$ do
5:             $q' = \langle g_0, \{(l_1 \in \{ref(1)\}), \ldots, (l_n \in \{ref(n)\})\}, m \rangle$
6:             $S = S \cup \{(ref,$ Bootstrapped Confidence Intervals$(q', \mathcal{D}_{HIST}, b, s, f, \alpha))\}$
7:         $\mathcal{B} = \mathcal{B} \cup S$
8: return $\mathcal{B}$
```

Algorithm 2 Bootstrapped Confidence Intervals

```
INPUT $q$: query, $\mathcal{D}_{HIST}$: historical cube, $b$: number of bootstrap replications, $s$: sampling
    percentage, $f$: bootstrap estimator, $\alpha$: confidence interval width
OUTPUT $S = \langle meas, \hat{\theta_1}, \hat{\theta_2} \rangle$
1: EstimatorList = $\langle\rangle$
2: $x^* = q(\mathcal{D}_{HIST})$
3: for $i = 1 \rightarrow b$ do
4:     $CurrentSample =$ randomSample$(x^*, s)$
5:     $EstimatorList = EstimatorList \cup f(CurrentSample)$
6: $Sort(EstimatorList)$
7: return $\langle q.m, EstimatorList[b \times (1 - \alpha)], EstimatorList[b \times \alpha] \rangle$
```

compute the confidence interval for each measure present in the cube (line 3) and for each reference (line 4) of the query result. The calculation is done by using the boostrap method (line 6) using the same predicate filters as the query at hand and the finest group-by set (line 5).

The calculation of each confidence interval is done by Algorithm 2. It is an adaptation to a multidimensional case from (Pol and Jermaine, 2005), where bootstrap is applied in a relational context. The underlying idea is that in each iteration in the number of bootstrap replications, a sample of the query answer is taken (line 4) and its plug-in estimator, here the mean, is obtained (line 5). That information is collected in a list that is later sorted to return the appropriate confidence interval.

Note that the bootstrapping is performed for each measure, for each cell of the result returned by query q (lines 2-4 of Algorithm 1). In algorithm 2, the query q' provided as argument only concerns one particular measure and one particular cell of q. So, given a cube with a number m of measures, and a query q whose result contains c cells, we perform $m \times c$ bootstrappings to record in each cell c a confidence interval. For each cell c, the bootstrap algorithm is applied by considering as bootstrap sample the primary facts (at the most detailed level) covered by c (line 2 of Algorithm 2).

Baseline recomputation can be naively done at each refresh. However, the recomputation of a baseline is only necessary when too many fresh facts covered by that baseline have been propagated into S-DW. The bigger this number of facts is, the less relevant the confidence intervals of the baseline are. We use this observation to improve our algorithm, by assigning to a baseline the probability that a given fact is used for one of its cell computation. This probability is $\frac{|DDW_Q|}{|DDW_Q|+|SDW_Q|} \times (1-(1-s)^b) \times (1-(1-\frac{1}{|Q|})^{|DDW_Q|+|SDW_Q|})$, where Q is the query from which the baseline is derived, DDW_Q (resp. SDW_Q) is the set of facts of $\mathcal{D}_{RT}$ (resp. $\mathcal{D}_{HIST}$) covered by Q, s the bootstrap sampling percentage, and b the number of bootstrap replications. In this formula, $\frac{|DDW_Q|}{|DDW_Q|+|SDW_Q|}$ is the probability that the fact comes from DDW_Q, and $(1-(1-s)^b)$ is the probability that a fact is chosen for the bootstrap computation. Finally, $(1-(1-\frac{1}{|Q|})^{|DDW_Q|+|SDW_Q|})$ is the probability that a cell of the baseline covers at least a given primary fact, which is derived from the Cardenas formula, widespread in data warehousing (Shukla et al., 1996). A given baseline is recomputed if this probability exceeds a threshold. In any case, we still need to refresh $|SDW_Q|$, by incrementing it by the total number of comparisons performed so far at baseline level and for this baseline.

4.3 Anomaly Detection

The core of the approach is depicted in Algorithm 3, a process which continuously runs in background by performing comparisons of the just arrived real-time information with the relevant, and already calculated baselines. In this algorithm, we compare a given fact to each of the relevant baselines (line 2) that have been gathered from Algorithm 5 (line 1). Basically, Algorithm 5 extracts relevant baselines from the whole set of baselines. So, for each of them, Algorithm 4 checks for anomalies.

Algorithm 3 Real-time Anomaly Detection

```
INPUT $fact$: Fact $\in \mathcal{D}_{RT}$, $\mathcal{B}$ : set of baselines
1: let $RB =$ relevantBaselines($fact, \mathcal{B}$)
2: for $b \in RB$ do
3:     $checkForAnnomalies(fact, b)$
```

Algorithm 4 Check for Anomalies

```
INPUT $fact = \langle factRef, factMeas \rangle$: Fact $\in \mathcal{D}_{RT}$, $b$ : baseline
1: let $bCell = \langle ref, \{(m_1, \hat{\theta_1^1}, \hat{\theta_2^1}), \ldots, (m_n, \hat{\theta_1^n}, \hat{\theta_2^n})\} \rangle \in b$ such that $ref \preceq factRef$
2: let $counters = getCounters(b, bCell)$
3: let $flag = \varnothing$
4: for $m_i \in \{m_1 ... m_n\}$ do
5:     if $\neg(\hat{\theta_1^i} \leq m_i \leq \hat{\theta_2^i})$ then
6:         $flag = increaseCounters(counters, b, bCell)$
7:         if $flag > 0$ then
8:             $raiseAlert(flag, b, bCell)$
```

Algorithm 4 starts by identifying the appropriate baseline cell to which the fact contributes (line 1). Then, for each mesure of the fact (line 3), we check if the value of the measure is out of the confidence interval (line 4). If so, the $increaseCounters$ function (line 5) increments the appropriate counters at the three alerting levels (baseline cell, baseline and overall set of baselines) and compares them againt the appropriate thresholds. It returns a flag indicating if some alerts have to be raised and some alerting levels. The $raiseAlert$ function raises the corresponding alerts (line 6). Counters and thresholds are global, we gather them thanks to the $getCounters$ function (line 2), that returns, for the three alerting levels, the set of anomaly counters, the set of processed fact counters and the set of thresholds.

Algorithm 5 Relevant Baselines

```
INPUT $Fact = \langle FactRef, FactMeas\rangle$: fact $\in \mathcal{D}_{RT}$, $\mathcal{B}$ set of baselines
OUTPUT $S$: set of relevant baselines
1: $S = \emptyset$
2: for $b \in \mathcal{B}$ do
3:     for $\langle ref, cell\rangle \in b$ do
4:         if $ref \preceq FactRef$ then
5:             $S = S \cup \{b\}$
6: return $S$
```

The cube given in Section 2 contains five measures, that are open, high, low, close and volume. We assume that Q_1 returns one member [NASDAQ] for level [Market.Geography].[Market Name] and two members [Shire plc] and [Grifols, S.A.] for level [Security.Geography]. [Organisation]. When Q_1 is monitored, q in line 2 of Algorithm 1 stands for Q_1. Following the algorithm convention, we have $Q_1 = \langle\{[Security.Geography].[Organisation], [Market. Geography]. [MarketName]\}, \{([Security.Geography].[Region] \in \{Europe\}), ([Security.Activity]. [Sector] \in \{[HealthCare]\})\}, close\rangle$.

For each measure of the cube (line 3), for each reference extracted from the result of Q_1 (each cell), we derive a query Q' that returns the facts related to this reference, at the finest level of details. In this precise case, we derive two queries for each measure m in $\{open, high, low, close, volume\}$, that are:

- $Q'_1 = \langle g_0, \{([Market.Geography].[MarketName] \in \{[NASDAQ]\}),$
  $([Security.Geography] \in [Shire\ plc])\}, m\rangle$
- $Q'_2 = \langle g_0, \{([Market.Geography].[MarketName] \in \{[NASDAQ]\}),$
  $([Security.Geography] \in \{[Grifols, S.A.]\})\}, m\rangle$

Bootstrapping is run separately for each derived query (Algorithm 2), and the corresponding cell in the baseline is populated by the confidence interval obtained for measure m. In our case, each baseline cell contains five confidence intervals, one for each measure.

When a new fact comes, for instance ⟨[Market Name].[NASDAQ], [Security Symbol].[Shire plc], close = 143.70, ... ⟩, relevant baselines are extracted (algorithm 5). Then, for each baseline (algorithm 4) the right cell is extracted (line 1), and the counters are increased according to the result of the comparison (line 6). The example baseline is extracted as it matches the fact. This is more precisely the first cell of this baseline that contains the fact, so the fact is compared to the confidence intervals in this cell for each measure. It appears that for measure $close$, value 143.70 is out of the confidence interval bounds [143.87 - 166.317], so an anomaly counter is incremented for this precise cell, another for the baseline that contains the cell, and another global counter.

5 Evaluation

In this section, we first describe a prototype system which implements the previously described features, and proofs the feasibility and correctness of the approach. Next, we present our experimental findings using the prototype and the running example described in Section 2. In order to evaluate the quality of the approach, we make use of two real-world cases based on the data model described in Section 2.
Baseline computations and the assessment of new facts are transparent and non-disruptive from a user interaction perspective. The former may take place in a batch mode during low user interaction periods -e.g. nightly or weekly- and the latter may be performed as an asynchronous background thread. Despite this, we considered important to assess the efficiency of these processes and identify potential improvement opportunities.

5.1 Prototype

In order to provide a proof of concept we designed a prototype implementation of the alerting system working on top of a real-time data warehouse. With a simple, but yet powerful interface, it allows the user to run queries to the different cubes, while storing the queries in the query log and baseline calculation. On the background, as new data arrives into the real-time data warehouse, the monitoring component executes Algorithm 3 in order to raise alerts.

OLAP schema and cubes are handled by Mondrian, an open source OLAP server (Pentaho, 2009). $\mathcal{D}_{HIST}$ and $\mathcal{D}_{RT}$ are stored as two independent cubes with their corresponding real-time or historical data sources, in a *MySQL* database using respectively disk and in-memory engines, which implements the data model introduced in Section 2. The Data Acquisition Module (DAM) is made up by two Python scripts and a number of batch scripts that crawl quotes from Yahoo Finance API and push data into the real-time data warehouse. As far as markets are concerned, the prototype contains data from the following institutions: MERVAL, IPC, NASDAQ, NYSE and BOVESPA. In terms of securities, we have chosen the top 1000 companies with the greatest market capitalisations in NASDAQ and NYSE markets. On top of that, an additional number of 50 securities from the Argentine, Brazilian and Mexican markets were added. Last but not least, around 10 major currency exchange rates were included. The time horizon is meant to date back to 50 years ago. Nevertheless, we noticed that the data volume greatly varies from the distant past (1 record every quarter up to 50 years ago) to the recent past (400 records for up to 1 day ago).

There is an important trade-off to be considered in Algorithm 2, that is whether to perform random sample selection and plug-in estimator calculation at database level or in the application logic. Current implementation considers the latter, hence it only needs a single access to $\mathcal{D}_{HIST}$ per query, this entails less encumber to the database and possibility to benefit from concurrency techniques like multithreading. On the other hand, by moving those operations down to database level, we can take advantage of optimised sampling and aggregation, by making use of indexing structures, for instance.

5.2 Experiments

Experiments have been performed using the query log presented in Section 2. Regarding parameters, we used what we considered standard values, specifically this is 100 replications for bootstrapping with samples of 1% of relevant records. For comparison purposes, we used a number of 3 standard deviations. In turn, the anomalies threshold was set to 0.1%.

5.2.1 Case 1: A Black Day for Markets

We chose October 10th, 2014 as an interesting point. As NASDAQ Composite Index plummeted by 2.33% that day, we wanted to check whether the system would have early warned users so that they were able to make better business decisions in a timely fashion (e.g. selling stock whose price was falling). $\mathcal{D}_{HIST}$ contained data from 4/Jan/1965 to 10/Oct/2014 at 13:29 GMT (1,974,462 rows). In turn, $\mathcal{D}_{RT}$ contained data for 10/Oct/2014 between 13:30 and 13:35 GMT (854 rows).

Baseline computation We computed 4 baselines (B_1, B_2, B_3, B_4) for the 4 monitoring queries (Q_1, Q_2, Q_3, Q_4) described in Section 2. The table below shows the experiment results in terms of output cells, time and storage. We can see that computation time is more sensitive to the number of input facts than to the number of output cells. In particular, the computation time increases proportionally to the number of input facts.

	Input		**Results**		
	Input Facts	**Coordinate Groups**	**Output Cells**	**Time (min)**	**Storage (KB)**
Q_1	18.263	10	50	8	9
Q_2	152.063	80	400	56	74
Q_3	34.868	406	2.030	9	378
Q_4	33518	20	100	1	19
Totals	209.072	516	2.580	74	480

Assessment of new facts Only 90 out of the 854 facts present in $\mathcal{D}_{RT}$ were relevant for the available baselines (8 facts were linked to B_1, 62 to B_2, 18 to B_3 and 2 to B_4). Considering that the cube has 5 measures, those 90 facts demanded 450 comparisons. All of them were assessed in about **627 seconds**, which represents averages of **1.39 seconds/measure/fact**.

Out of the 450 comparisons performed, the baseline B_3 detected **6 anomalies**. The cells involved were linked to security symbols TXN (High and Close measures) and MCHP (Open, High, Low and Close measures). As the threshold of 0.1% we had set was exceeded at baseline level (6 out of 90), at baseline cell level (1 out 1 in 6 cells) and at general level (6 out of 450), alerts were issued in the three of them.

By performing an ex-post analysis of what happened with stock TXN, we notice that five minutes after the alert, the price kept on falling. A business user would have valued a timely alert, not only because this would have allowed him or her to close positions on that stock but also because that would have represented an early warning of what was going to happen in NASDAQ that day. Going into details, the Closing price of TXN at 13:34 (42.674 USD) fell 2.42% compared to the Opening price of the immediately subsequent transaction that was registered at 13:39 (41.64 USD).

At variance with TXN, if we check the case of MCHP, we see that price went up just after the alert. Then, there was a declining cycle, followed by an ascending cycle and then there was another declining cycle. At the end of the day, price turned out to be higher than the one detected by the alert. However, if we analyse this from a broader perspective, the alert would have enforced the warning on the general behaviour of NASDAQ market that day.

5.2.2 Case 2: An Apparently Quiet Day

November 13th, 2014 has been apparently a quiet day for NASDAQ market as a whole. NASDAQ composite showed an overall slight increase of almost 0.11%. Particularly, $\mathcal{D}_{HIST}$ contained data from 4/Jan/1965 until 13/Nov/2014 at 13:29 GMT (3,221,378 rows). In turn, $\mathcal{D}_{RT}$ contained data for 13/Nov/2014 between 13:30 and 14:34 GMT (1386 rows).

Baseline computation Results below confirm the relationship between the number of input facts and the amount of time that the baseline computation takes. Compared to Case 1, the number of input facts increased around a 62% and so did the baseline computation time.

	Input		**Results**		
	Input Facts	**Coordinate Groups**	**Output Cells**	**Time (min)**	**Storage (KB)**
Q_1	29.576	10	50	10	9
Q_2	247.432	80	400	91	74
Q_3	57.186	406	2.030	15	378
Q_4	5.880	20	100	2	19
Totals	340.074	516	2.580	118	480

Assessment of new facts In this case, we had 1386 facts in $\mathcal{D}_{RT}$ but only 110 of them where relevant for baselines (10 facts were linked to "European Health-Care Companies", 80 to "US Health-Care Companies", 18 to "Semiconductors firms in NASDAQ by Year" and 2 to "Water Supply firms by Year"). Considering the number of measures, those 110 facts which yielded 550 comparisons. All of them were assessed in 384 seconds, representing averages of **0.7 seconds/measure/fact**, which is lower than the figure obtained in Case 1. No anomalies were detected in any of the four baselines, which makes sense since overall figures show that it was a quiet day for the market.

6 Related Work

Real-Time warehousing has become a reality in companies for some time now and the real-time keyword is part of many of the characteristics described by vendors of data warehousing products like Oracle or Vertica. Real-time solutions typically rely on a smaller repository (ideally an in-memory one) holding the most recent data, to allow faster loading and refreshing, without affecting the performance of querying activity (Ferreira and Furtado, 2013; Cuzzocrea et al., 2014). While products and approaches claiming to offer real-time data warehousing should encompass all parts of the architecture (ETL, materialising, refreshing, etc.), earlier real-time warehousing approaches were limited in the way they handled the simultaneity between online querying and continuous data loading (Vassiliadis and Simitsis, 2009; Zuters, 2011; Jain et al., 2012). Those limitations led to the proposal in (Ferreira and Furtado, 2013), where queries and data loading occur simultaneously with minimum performance degradation.

At variance with real-time data warehouses, which aim at capturing business activity data as it occurs, active data warehouses (Thalhammer et al., 2001) were designed to support automatic decision-making when faced with routine decision tasks. Inspired by active database systems, active data warehouses implement rules that are used to trigger predetermined OLAP queries in response to events detected on the data sources or that occur periodically (Thalhammer et al., 2001; Zwick et al., 2006). Our proposal can be seen as a form of real-time active data warehouses, where events are the anomalies detected and the predetermined queries correspond to baselines.

Finally, we note that our approach borrows from techniques proposing models of data cubes. For instance, approximate query answering in OLAP pursues the idea of compressing the data cube in order to obtain approximate answers to efficiently processed OLAP queries, whose approximation error is tolerable. Modeling the cube can be achieved either from base data (e.g., by using clustering (Yu and Wang, 2002)) or from aggregated data (e.g., using information entropy (Palpanas et al., 2005)).

7 Conclusion

This paper introduces an approach for alert raising in a real-time data warehouse architecture, where fresh data are in memory while historical data are stored on disk. We assume that items in the query log express the sets of facts that users would like to monitor. In an offline phase, for each of these queries we build an in-memory structure, called baseline, recording a confidence interval for the facts contributing to each cell of the query result. Then, fresh data are compared to the appropriate baselines and used to raise alerts. We implemented the approach and illustrated its interest in the domain of technical analysis of stock markets.

As future work, we will first address the optimisation of baseline computation, which might be seen as the bottleneck of our approach. We will particularly study strategies for an iterative computation of baselines, using a combination of application logic and database features. We also plan to test our approach in a more realistic data warehouse situation, where anomaly detection competes with regular analytical queries. Particular attention needs to be paid to the trade-off between storage space needed for baselines and the number of alerts triggered. Finally, a longer term goal would be including our approach in a full active data warehouse setting, where complex analyses are triggered in response to the alerts raised.

References

Cuzzocrea, A., N. Ferreira, and P. Furtado (2014). Real-time data warehousing: A rewrite/merge approach. In *DaWaK*, pp. 78–88.

Efron, B. and R. Tibshirani (1986). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, 54–75.

Efron, B. and R. J. Tibshirani (1994). *An introduction to the bootstrap*, Volume 57. CRC press.

Ferreira, N. and P. Furtado (2013). Real-time data warehouse: a solution and evaluation. *IJBIDM 8*(3), 244–263.

Ferreira, N., P. Martins, and P. Furtado (2013). Near real-time with traditional data warehouse architectures: factors and how-to. In *IDEAS*, pp. 68–75.

Golfarelli, M. and S. Rizzi (2009). *Data warehouse design: Modern principles and methodologies*. McGraw-Hill, Inc.

Jain, T., R. S, and S. Saluja (2012). Article: Refreshing datawarehouse in near real-time. *International Journal of Computer Applications 46*(18), 24–29.

Jörg, T. and S. Dessloch (2009). Near real-time data warehousing using state-of-the-art etl tools. In *BIRTE*, pp. 100–117.

Kimball, R. (1996). *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley.

Palpanas, T., N. Koudas, and A. O. Mendelzon (2005). Using datacube aggregates for approximate querying and deviation detection. *IEEE Trans. Knowl. Data Eng. 17*(11), 1465–1477.

Pentaho (2009). Mondrian open source olap engine.

Pol, A. and C. Jermaine (2005). Relational confidence bounds are easy with the bootstrap. In *SIGMOD*, pp. 587–598.

Santos, R. J. and J. Bernardino (2008). Real-time data warehouse loading methodology. In *IDEAS*, pp. 49–58.

Shukla, A., P. Deshpande, J. F. Naughton, and K. Ramasamy (1996). Storage estimation for multidimensional aggregates in the presence of hierarchies. In *VLDB*, pp. 522–531.

Thalhammer, T., M. Schrefl, and M. K. Mohania (2001). Active data warehouses: complementing OLAP with analysis rules. *Data Knowl. Eng. 39*(3), 241–269.

Vassiliadis, P. and A. Simitsis (2009). Near real time ETL. In *New Trends in Data Warehousing and Data Analysis*, pp. 1–31.

Waas, F., R. Wrembel, T. Freudenreich, M. Thiele, C. Koncilia, and P. Furtado (2013). On-demand elt architecture for right-time bi: Extending the vision. *IJDWM 9*(2), 21–38.

Yu, F. and S. Wang (2002). Compressed data cube for approximate OLAP query processing. *J. Comput. Sci. Technol. 17*(5), 625–635.

Zuters, J. (2011). Near real-time data warehousing with multi-stage trickle and flip. In *BIR*, pp. 73–82.

Zwick, M., C. Lettner, and C. Hawel (2006). Implementing automated analyses in an active data warehouse environment using workflow technology. In *TEAA*, pp. 341–354.

Résumé

Ce travail propose une approche pour la levée des alertes dans le cadre d'un entrepôt de données temps-réel. Il est basé sur le calcul d'intervalles de confiance pour les mesures des faits historiques. Des nouvelles données qui arrivent dans l'entrepôt en temps-réel sont systématiquement comparés à des intervalles respectifs et des alertes sont levées lorsque des anomalies sont détectées. L'intérêt de l'approche est illustré sur un entrepôt permettant l'analyse technique de données boursiéres.