Materializing Baseline Views for Deviation Detection Exploratory OLAP[*]

Pedro Furtado[1], Sergi Nadal[3], Veronika Peralta[2], Mahfoud Djedaini[2], Nicolas Labroche[2], and Patrick Marcel[2]

[1] University of Coimbra, Portugal, `pnf@dei.uc.pt`
[2] University of Tours, France, `firstname.lastname@univ-tours.fr`
[3] Universitat Politècnica de Catalunya, Spain, `snadal@essi.upc.edu`

Abstract. Alert-raising and deviation detection in OLAP and exploratory search concerns calling the user's attention to variations and non-uniform data distributions, or directing the user to the most interesting exploration of the data. In this paper, we are interested in the ability of a data warehouse to monitor continuously new data, and to update accordingly a particular type of materialized views recording statistics, called baselines. It should be possible to detect deviations at various levels of aggregation, and baselines should be fully integrated into the database. We propose Multi-level Baseline Materialized Views (BMV), including the mechanisms to build, refresh and detect deviations. We also propose an incremental approach and formula for refreshing baselines efficiently. An experimental setup proves the concept and shows its efficiency.

Keywords: OLAP, deviation detection, materialized views

1 Introduction

Data warehouses are used to support analysis over large amounts of data and decision making based on the results of those analysis. Online analytic processing (OLAP) tools enable analysis of multidimensional data interactively using multiple perspectives, roll-up, drill-down, slicing and dicing. That multitude of analysis perspectives contributes to information overload. While a huge amount of information is available and browsable in many different ways, it is commonly agreed that mechanisms to ease the job of the analyst are of key importance.

Deviation detection is a useful mechanism to identify variations in data distributions that can be relevant to explore, and alert users to look at those uncommon patterns. In order to do this, it is necessary to build, refresh and monitor baselines. Baselines are statistical artifacts that maintain information about the normal distribution of some data. When new values are outside that normality users are alerted, or the variations are used to guide navigation. In this work we restrict deviation detection to identifying if values are outside a usual band, on future work we intend to explore more complex models.

[*] This work was done while Pedro Furtado was visiting University of Tours.

Baselines should be integrated into databases as “first-class citizens”. They have many similarities to materialized views in terms of their life-cycle. In this paper we propose Multi-level Baseline Materialized Views (BMV). The approach assumes baselines can be created and monitored at different levels of aggregation, such as daily and monthly totals. BMVs are integrated into the data warehouse, managed and refreshed as a kind of materialized view, at multiple levels of aggregation. The proposed architecture should be efficient, therefore we design the physical mechanisms for baseline monitoring and refreshing taking efficiency into account. The resulting architecture is evaluated using a proof of concept and the dataset of the Star Schema Benchmark (SSB).

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 motivates Multi-level Baseline Materialized Views with an example and possible uses. Section 4 defines the concept formally. Section 5 proposes the mechanisms for managing BMVs (creation, refreshing and detection). Section 6 presents experimental results to validate our proposal and Section 7 concludes.

2 Related work

For a long time now, researchers have been interested in finding out interesting patterns in datasets [4,6,11,12], and interestingness measures have been studied as well [5]. Some previous works have dealt with the more specific issue of deviation detection [2,4,11,12].

In [2], the authors adapt attribute focusing algorithms to multidimensional hierarchical data. Attribute focusing discovers interesting attribute values, in the sense that the observed frequency of those values deviate from their expected frequency. In order to make attribute focusing useful for OLAP, they have adapted the method to hierarchical dimensions often found in data cubes.

In [11], the authors introduce an operator that summarizes reasons for drops or increases observed at an aggregated level. They developed an information theoretic formulation and designed a dynamic programming algorithm to find the reasons. This work is then also evolved in [12], where the authors propose automation through the *iDiff* operator that in a single step returns summarized reasons for drops or increases observed at an aggregated level. Comparing to our proposal, this approach does not materialize baselines, therefore it is very computationally intensive during navigation, since it requires searching for deviations on the fly. In [4], a more general framework is proposed for measuring differences in data characteristics based in quantifying the deviation between two datasets in terms of the models they induce. The framework covers a wide variety of models, including frequent itemsets, decision tree classifiers, and clusters. The authors show how statistical techniques can be applied to the deviation measure to assess whether the difference between two models is significant. In [3] a set of approaches was designed for summarizing multidimensional cubes, including quantization, multidimensional histograms and an efficient approach for computation and maintenance of confidence intervals.

Our previous work on the subject [8], for raising alerts in real-time data warehouses, introduced a first definition of baselines, an approach to compute them and to detect deviation in real-time data warehouses. We now evolve the concept in some directions. We generalize the concept of baseline to detect deviations at any level of aggregation. Multi-level Baseline Materialized Views materialize baselines, and are fully integrated in the data warehouse life-cycle. The mechanisms for building, refreshing and monitoring deviations are also different and achieve integration into the data warehouse architecture. BMVs share many of the mechanisms of materialized views, but with specific approaches for refreshing the baseline. We also propose a mechanism for holding and computing data normality intervals incrementally within the baseline.

3 Using Baseline Materialized Views

In this section we make use of examples to better motivate and clarify the concept of BMV, its relevance in the framework of exploratory search and query recommendation, and how it could be used in such context.

A Baseline Materialized View defines two levels of aggregation: (i) a detailed level (DL), at which deviations are detected, and (ii) a more aggregated level (AL), at which statistical information is summarized to the user. For each cell of the AL, a set of cells of the DL allow to study deviations.

As an example, consider a data warehouse that records each individual sale of products. A sales manager is interested in knowing when the sales of some products are above or below its typical sales values. He creates a baseline that alerts when the weekly sales of a product are abnormal when compared to the average weekly sales of the product category along the year. The baseline monitors weekly sales of products (DL: $product \times week$) and provides statistics by year and category (AL: $category \times year$). If the user wanted the baseline for a specific subset of the cube, he could add the appropriate filters (e.g. country="Canada").

A straightforward use of BMV is alerting. This means that the baseline will compare the current week with the remaining weeks of the same year or of the previous year (if there are not enough weeks in the current year yet to compare to). In order for this to work, ($product \times week$) must be a hierarchical detail of ($category \times year$). Weekly product sales will be automatically integrated into a BMV after each week ends (because of the week field in the definition of the baseline). The system writes into an alert log when the value to be integrated is out of a typical interval, which we define as $[AVG(sales){-}CI, AVG(sales){+}CI]$, where CI is a confidence interval.

In addition, the concept of BMVs is extremely useful in exploratory OLAP. In the case of a user exploring cubes from some perspective, deviation detection with BMVs can be used to efficiently direct him to drill-down from the current perspective directly into some interesting facts that the BMV maintains.

As another example, consider the drill-down scenario (shown in Figure 1). A user initially compares sales per nation for *1997* and *1998*. He then focuses on *Canada*, comparing sales per city. Finding that *Canada 5* city has an abnormal volume of sales, he drills-down to product category, and finds out that category *MFGR#42* has an enormous volume of sales. Analyzing the monthly evolution of sales of this category, he discovers that it has had an abnormal amount of orders in *February 1998*. The length of this simple navigation is 4 queries. This compares with 2 queries if using a baseline with AL: $nation \times year$ and DL: $city \times month \times category$. When the user is navigating at the more aggregated level, the baseline immediately indicates the abnormal increase in sales of category *MFGR#42*. But, even more important, while the user navigating through the data may never find the interesting data, the interesting data is suggested to him when using baselines. In the experimental section we revisit this example, and show the improvement also in execution time.

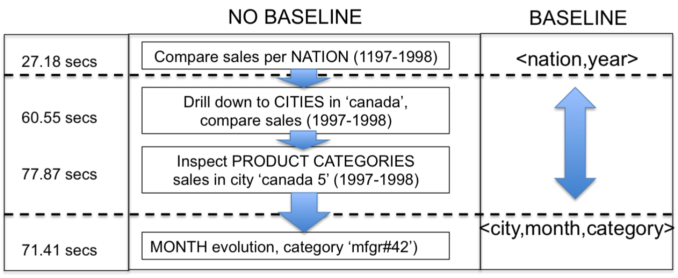

NO BASELINE
BASELINE
27.18 secs
Compare sales per NATION (1197-1998)
<nation,year>
60.55 secs
Drill down to CITIES in 'canada', compare sales (1997-1998)
77.87 secs
Inspect PRODUCT CATEGORIES sales in city 'canada 5' (1997-1998)
<city,month,category>
71.41 secs
MONTH evolution, category 'mfgr#42')

Fig. 1. Drill-down scenario. Time on the left refers to experiments in Section 6.

BMVs are compatible with, and complement ordinary materialized views (MV) in two ways. First, while an ordinary MV provides one-level aggregation of finest granularity data (i.e. primary fact events), a BMV analyzes data at some level of detail (DL), not necessarily the finest granularity, and describes the distribution of those data at a more aggregated level (AL). Second, the most relevant innovation in BMVs is that they are associated with automatic deviation checks at each BMV refresh instant, generating a log of detected deviations and/or alerts.

4 Formal Framework

In this section we define baseline materialized views and illustrate them using simple examples. To keep the formalism simple, we consider read-only cubes under a ROLAP perspective, described by a star schema. 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*, with pairwise disjoint domains,

– $H = \{h_1, \dots, h_n\}$ is a finite set of *hierarchies*, such that each level l_i of A belongs to at least one hierarchy of H.
– $M = \{m_1, \dots, m_p\}$ is a finite set of *measure* attributes.

Given a schema $\mathcal{M} = \langle A, H, M\rangle$, let $Dom(H) = Lev(h_1) \times \ldots \times Lev(h_n)$; each $g \in Dom(H)$ being a *group-by set* of $\mathcal{M}$. We use the classical partial order on group-by sets, defined by: $g \succeq g'$, if $g = \langle l_1, \dots, l_n\rangle$, $g' = \langle l'_1, \dots, l'_n\rangle$ 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.

Definition 1 (Baseline).
A baseline *over schema* $\mathcal{M} = \langle A, H, M\rangle$ *is a 5-tuple* $B = \langle G, Gd, P, Me, Stats\rangle$ *where:*

1. $G \in Dom(H)$ *is the baseline group-by set;*
2. $G_d \in Dom(H)$*, with* $G \succeq G_d$ *(*G *is a rollup of* G_d*), is the baseline deviation detection group-by set;*
3. $P = \{p_1, \dots, p_n\}$ *is a set of Boolean predicates, whose conjunction defines the* selection predicate *for* q*; they are of the form* $l = v$*, or* $l \in V$*, with* l *a level,* v *a value,* V *a set of values. Conventionally, if no selection on* h_i *is made in* B *(all values being selected) nothing appears in* P *and we write* $P = \emptyset$ *if no selection is made at all;*
4. $Me \subseteq M$ *is a subset of measures;*
5. $Stats = \{f_1, \dots, f_k\}$ *over* Me *is a set of aggregation functions, one per measures in* Me*. In this paper, these functions compute the average and confidence interval for each measure in* Me*.*

A baseline can be viewed as a query, that specifies how to aggregate (using functions in $Stats$) a set of facts (defined by group by set G_d and predicates in P) at a given aggregation level (group by set G). A baseline materialized view (BMV) is a baseline materialized as a stored data set and refreshed automatically using the procedure described in section 5.

Example 1. Assume a baseline by Month and Brand, describing the statistics (in terms of a confidence interval) for the average sales by Day and Product. This baseline is:
$\langle\{month, brand\}, \{day, product\}, \{nation = \text{“}France''\}, \{sales\}, \{f_{CI}\}\rangle$[4],

where f_{CI} computes the confidence interval of a set of numerical values. The baseline will update the sales total, the average sales and the confidence interval when products are added to the baseline, by addition to the (month $\times$ brand) cell to which the product and current month belong. The baseline will also allow detecting deviations, by verifying if the new values being inserted are within or outside of the confidence interval. In some cases this verification needs to be done with the previous period. For instance, when a new month starts, there is no sale yet for that month, therefore new sales should be compared with the values of the previous month, until there are enough values in the current month to compare with it.

[4] For the sake of readability, baseline expressions are simplified in the examples: the top levels are dropped in the group-by sets.

5 Baseline Materialized Views life-cycle

The BMV life-cycle is composed by a set of mechanisms that includes building the BMV initially, refreshing it, detecting deviations and selecting the best BMVs to materialize.

5.1 Updating Baseline Materialized Views

In a data warehouse, materialized views (MVs) are refreshed periodically, either as soon as "new data arrives", or in deferred mode. Conceptually, the update is based on a merge operation, whereby the MV is merged with new data.

The approach for refreshing BMVs in a star schema of a ROLAP data warehouse shares a lot of similarities with refreshing MVs. An MV would have the following elements: $MV = \langle G, P, Me, agg \rangle$. In this section we describe the general approach to refresh these structures and how to merge the measures (Me). The operation that merges the statistics ($Stats$) of BMVs is described in 5.2.

The refresh is done by first running the query associated with the deviation detection group set (G_d in BMV) or with the group-set (G in MV) against the new data items. The result is an aggregation of the data items with granularity G_d or G respectively. Now, new values are merged with those already in the BMV/MV using additive formulas. Examples of such formulas to merge measures of MVs or BMVs are:

$$sum_{\text{new}} = sum_{\text{old}} + SUM(\text{new items}) \tag{1}$$

$$count_{\text{new}} = count_{\text{old}} + COUNT(\text{new items}) \tag{2}$$

$$min_{\text{new}} = MIN(min_{\text{old}}, MIN(\text{new items})) \tag{3}$$

$$max_{\text{new}} = MAX(max_{\text{old}}, MAX(\text{new items})) \tag{4}$$

$$avg_{\text{new}} = \frac{sum_{\text{old}} + SUM(\text{new items})}{count_{\text{old}} + COUNT(\text{new items})} \tag{5}$$

Note that this is similar to the standard approach used for incremental maintenance of data cubes and summary tables in a Warehouse [9].

Example 2. Consider the following examples of, respectively, BMV1, BMV2 and BMV3:
$\langle\{product, year\}, \{product, day\}, \{nation = France\}, \{sales\}, \{sum(sales)\}\rangle$
$\langle\{product, year\}, \{product, week\}, \{nation = France\}, \{sales\}, \{sum(sales)\}\rangle$
$\langle\{brand, year\}, \{product, week\}, \emptyset, \{sales, \{sum(sales)\}\rangle$

Consider a new data item to be added with the following fields:

product	**brand**	**date**	**nation**	**sales**
P101	Philips	01-01-2015	France	...

It is integrated into BMV_1 at the end of the day, as it needs to be aggregated into *(product,day)*. It is integrated into BMV_2 and BMV_3 at the end of the week, as it is pre-aggregated into *(product,week)* to represent sales of each product for a week. The measure *sum(sales)* for product *P101* will be updated by summing all sales of that product in the period.

5.2 Statistics Structure and Merging Operation

The process of refreshing BMVs is incremental, with the addition of new data to the existing data. New data needs to refresh the statistics describing it.

Consider the structure of BMVs, $BMV = \langle G, Gd, P, Me, Stats\rangle$. G_d is the deviation detection group-by set. The BMV is refreshed when there is complete data at that level. For instance, if we are considering daily sales per product, the deviation detection group-by set is daily sales per product and we can refresh the baseline after each day has passed (or later, if deferred). Now we describe how to refresh the *Stats* element. Note that the statistics to be considered must have an incremental property to guarantee the usability and the efficiency of the approach.

According to the *Central Limit Theorem* it is possible to estimate a confidence interval around the mean of a population with the hypothesis of an underlying *Normal* distribution if the number of samples is 30 or more. The mean μ and the standard deviation σ of a population $X = \{x_i\}_{i=1}^n$ can be computed incrementally as follows if we keep in memory the tuple $Stats = (n, \sum_{i=1}^n x_i, \sum_{i=1}^n x_i^2)$:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma = \left(\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^n x_i\right)^2}{n}\right)^{\frac{1}{2}} \tag{6}$$

The confidence interval CI is then defined as follows:

$$CI = \mu \pm Z_{(1-\rho)} \cdot \frac{\sigma}{\sqrt{n}} \tag{7}$$

where $Z_{(1-\rho)}$ is a parameter that is read from the *Normal* distribution table and that indicates how many standard deviations covers the interval when the confidence is in the interval is $1-\rho$, or in other words, when the probability that the true mean is outside the confidence interval is ρ.

Finally, we define the interval I as being the sum of the confidence interval of the estimator of the mean CI, that reflects the uncertainty in the estimator, and a value that can be expressed as either a fraction f of the mean itself or a constant value c or both (defined by hand). Both f and c should be positive real values or zero.

Example 3. The following is the information in a BMV, from which the average and the confidence intervals are promptly computed using (5), (6) and (7). The confidence interval is configured for $1 - \rho = 0.90$ which corresponds to a parameter $Z_{0.90} = 1.645$.

$\langle\{month, brand\}, \{day, product\}, \emptyset, \{sum(sales)\},$
$\{((count(sales), sum(sales), sum(sales * sales)), 1.645\}\rangle$

5.3 Materialization Algorithm

Before introducing the algorithm, we motivate it showing the unfeasibility of materializing the complete set of BMVs. Consider a roll-up lattice, where each node n_i represents a group-by set G, hence the number of BMVs that can be generated from the node, is the total number of available nodes from n_i to the top node n_0. That represents all possible perspectives G that can be observed from G_d Figure 2 depicts a simple lattice structure with 3 dimensions, where each node shows the number of possible BMVs. Thus, the number of BMVs it can contain is 22. On top of that, that number will be increased by the possible combinations of selection predicates P at each level n_i.

In [8] we introduced the concept of monitoring query set MQ, that contains a set of queries q which reflect the user's interest for deviation detection. Here we extend the definition of MQ in order to incorporate the elements of a BMV, therefore $MQ = \{m_1, \ldots, m_n\}$ where $m_i = \langle G, G_d, P, Me, Stats \rangle$. MQ can be obtained by mining the query log, with techniques such as [1,12], tailored to the user's interest, or following a cost-based approach based on empirical measures as the well-known greedy algorithm [7].

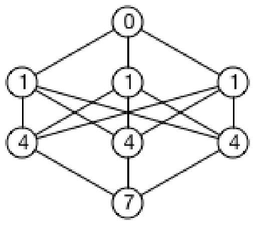


Fig. 2. A 3-dimensional lattice

Algorithm 1 depicts the materialization phase, valid for both generation and refreshment of BMVs. For each monitoring query, the related BMV is materialized using the formulas presented in 5.1 and 5.2. We assume the existence of a *CDC* (Change Data Capture) table containing the new incoming items. Operation `isMaterialized` checks if a set of BMVs contains the BMV in the second parameter. On the other hand, `mergeStatistics` given a set of BMVs, a BMV and a sample of items applies eqs. (1) to (5) from the sample to the BMV and includes it in the set. Likewise, `refreshConfidenceInterval` applies eqs. (6) and (7).

Algorithm 1 Materializing BMVs

```
Input $MQ$: monitoring query set, $BMV$: set of baseline materialized views, $CDC$
    table containing new items
Output $BMV$ refreshed set of baseline materialized views
1: for $q = \langle G, G_d, P, Me, Stats \rangle \in MQ$ do
2:     $sample = q(CDC)$                                  ▷ Run query against CDC table
3:     if isMaterialized($BMV, q$) then
4:         mergeStatistics($BMV, q, sample$)
5:     refreshConfidenceInterval($BMV, q, sample$)
6: return $BMV$
```

5.4 Detection and Alerting

Detection refers to the system detecting deviations in data. The baselines contain reference statistics, so detection is based on verifying whether a new value is within the interval. For instance,

```
if (newSalesValue > avgSales + IC + f · IC + c or
    newSalesValue < avgSales − IC − f · IC − c) {
        dump alert to log
}
```

where the information dumped to the log has the structure $\langle product, day, month, year, new\ sales, average\ sales, confidence\ interval\ limits\rangle$.

Deviations are tested immediately prior to integrating the new data into the baseline. If a deviation is detected, information is dumped into a deviation log. Only cases with more than a pre-specified minimum number of samples in the baseline will be dumped. Then there are different possible courses of action. It is possible to raise alerts for the user to look at the data, but the log can also be used to direct the user to the most interesting data during navigation. Besides the deviation log, there can be a "new cases" log, which records new cases that are not yet in the baseline when they appear (e.g. a new product).

At the start of a new period, there are no values registered yet in the BMV for that period, so it is not possible to compare new values with the same period. If there is at least a minimum number of samples, the value is compared within the period, otherwise it is compared against the previous period instead of the current one.

6 Proof of concept

We illustrate the interest of our approach on the Star Schema Benchmark [10], recording sales of products (parts) over time (the date dimension), customers and suppliers. We modified the generated data to add skewness/deviations (an abnormal amount of sales of specific products or brands in specific time intervals). SSB generates sales data from 1992 to 1999. The sales of product 1 and brand MFGR#241 where increased to 5% of total sales in the first week of 1998. 1GB of data was generated (SF=1).

The set of BMVs created is listed in Table 2. We apply the approach to detect weekly abnormalities in the first week of 1998 (parameters f and c were set to 0), show sales evolution and detection results, and evaluate the following efficiency metrics: a) time to build the BMV; b) time to refresh the BMV; c) time to detect the deviations (generate the alert log). The time to build the BMVs and MVs (materialized views) is compared. All the experiments were run on a Toshiba Qosmio machine, on a multicore Intel i7 CPU q720 (1.6 GHz), 8 GB of RAM, 1 TB 7400 RPM disk. The DBMS was MySQL version 5.5.41 bundled with MAMP out-of-the-box with no tuning except for the creation of indexes.

Figure 3 shows the evolution of sales (sum of quantity) in 52 weeks of 1997 and the first week of 1998 of two particular brands (MFGR#111 and MFGR#421).

The injected increase in sales can be seen in the evolution of sales of brand MFGR#421, with a steep rise in the first weeks of 1998. We can also see the deviation interval limits for the same brand (MFGR#421). Figure 1 shows the output log with deviations detected for the first week of 1998. It correctly identifies brand MFGR#421 as having sales of 19,945, a value well above the mean (5,457) and outside the interval limits also identified (2,601 to 8,313). This was the only row in the log, indeed correct since we only injected variations in MFGR#421.

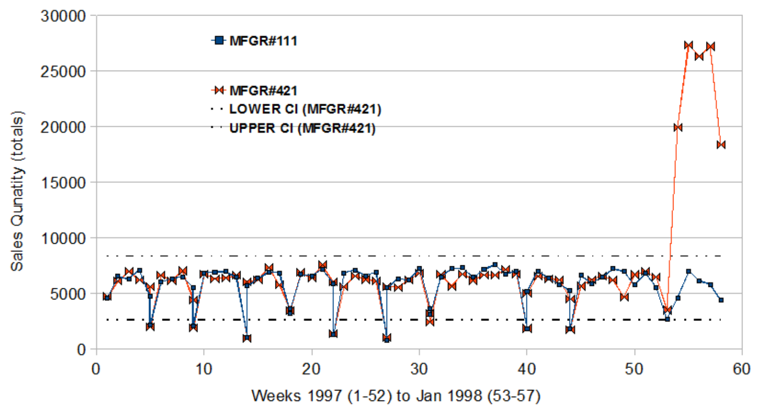


Fig. 3. Charting Brand by Week 1997 and Jan 1998

Brand	Week	Year	Sales	Num of Sales	Avg Sales	Upper Limit	Lower Limit
MFGR#412	1	1998	19,945	62	5457.47	8313.62	2601.32

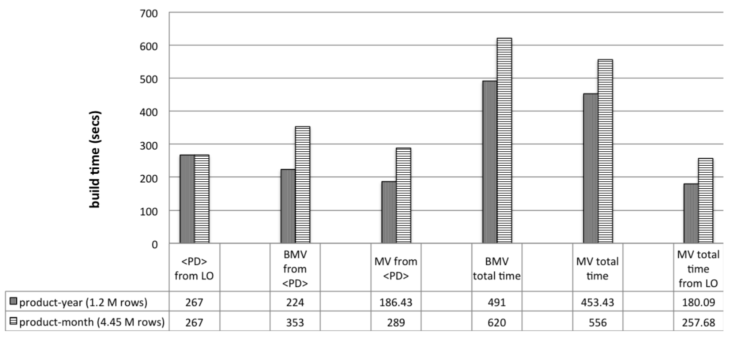
Table 1. Content of Alert Log for Jan 1998

Table 2 depicts the times taken to build, refresh and detect deviations in BMVs, and the size of the BMVs. The build time is incurred only once and depends on the size of the dataset. It was between 60 and 620 seconds, depending on the amount of data and aggregation level. Figure 4 details the build times for product BMVs, and compare with the time taken to build MVs of the same aggregation level. The same test was done for the brand BMVs but is not reported due to lack of space. The total times to build a BMV, an MV following the same pre-aggregation and an MV directly from the base data were 620, 556 and 258 seconds respectively. Creation of an MV is significantly faster if directly from the base data, but times to build BMVs are still acceptable and offer the advantage of having deviation detection.

The refresh times and time taken to generate the deviation detection logs are shown in table 2. Refreshing was very quick for BMVs with few rows and took between 25 and 125 seconds for huge BMVs (updating sales statistics for 200,000 products). The time taken to test and generate deviation logs was insignificant in all cases.

BMV	name	initial build time	size(rows)	refresh time	time to generate alert log
PY-PD	prod-year, prod-day	367	1.2M	29.36	1.07
PM-PD	prod-month, prod-day	620	4.45 M	124.41	0.96
BY-BD	brand-year,brand-day	80	384	0.11	0.12
BM-BD	brand-month,brand-day	79.85	4608	0.09	0.46
PY-PW	prod-year,prod-week	484	1.2M	24.9	3.6
BY-BW	brand-year,brand-week	59.35	384	0.1	0.09

Table 2. Time to Operate Baseline Materialized Views



	<PD> from LO	BMV from <PD>	MV from <PD>	BMV total time	MV total time	MV total time from LO
■ product-year (1.2 M rows)	267	224	186.43	491	453.43	180.09
▤ product-month (4.45 M rows)	267	353	289	620	556	257.68

Fig. 4. Detailed Timing to Build Products BMVs

Finally, we tested the drill-down scenario detailed in section 3. Without using BMVs, the user had to submit **4 different queries**, for a cost of **237 secs**. When using right BVMs, which provided him assistance, he only had to submit **2 queries** with a cost of **98.5 secs**.

7 Conclusions and Future Work

In this paper we have proposed Multi-Level Baseline Materialized Views (BMV) as a way of integrating deviation detection functionality into the data warehouse, to efficiently detect abnormal values. We have defined baseline materialized views, described their structure and the mechanisms for building, refreshing, monitoring and alerting on various levels of aggregation based on those baselines. In order to prove that the approach works, we created an experimental setup with the Star Schema Benchmark and tested it over sales data. The results of the experiments show that the approach works well and does not introduce significant overheads. It provides an effective way to integrate baselines, deviation detection and alerting into databases.

An interesting avenue for future work is to relax the uniform distribution assumption underlying the approach, and to devise baseline materialization algorithms by looking at the navigation sessions of a user, to decide automatically to build a BMV based on her knowledge of the data. In other words, there should be a BMV selection algorithm for choosing the best set to materialize. Additionally, more complex methods can be used to detect anomalies by comparing data distributions, such as the KullbackLeibler divergence. We would like to explore real-time capabilities further and to apply these concepts to help in exploratory OLAP, since the abnormalities detected in data are crucial clues to the most interesting paths to follow during analysis and exploration. Finally, we also need to deal with the information overload that results from too many alerts.

References

1. J. Aligon, E. Gallinucci, M. Golfarelli, P. Marcel, and S. Rizzi. A collaborative filtering approach for recommending OLAP sessions. *Decision Support Systems*, 69:20–30, 2015.
2. C. C. Fabris and A. A. Freitas. Incorporating deviation-detection functionality into the OLAP paradigm. In *XVI Simpósio Brasileiro de Banco de Dados, 1-3 Outubro 2001, Rio de Janeiro, Brasil, Anais/Proceedings.*, pages 274–285, 2001.
3. P. Furtado. Reduced representations of multidimensional datasets, phd thesis, u. coimbra, December 2000.
4. V. Ganti, J. Gehrke, R. Ramakrishnan, and W. Loh. A framework for measuring differences in data characteristics. *J. Comput. Syst. Sci.*, 64(3):542–578, 2002.
5. L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.
6. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.
7. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. *SIGMOD Rec.*, 25(2):205–216, June 1996.
8. M. Lòpez, S. Nadal, M. Djedaini, P. Marcel, V. Peralta, and P. Furtado. An approach for raising alert in real-time data warehouses. *Journes francophones sur les Entrepts de Donnes et lAnalyse en ligne Bruxelles, Belgique, 2-3 avril 2015*, 2015(1):55–86, 2015.
9. I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA.*, pages 100–111, 1997.
10. P. E. O'Neil, E. J. O'Neil, X. Chen, and S. Revilak. The star schema benchmark and augmented fact table indexing. In R. O. Nambiar and M. Poess, editors, *TPCTC*, volume 5895 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2009.
11. S. Sarawagi. Explaining differences in multidimensional aggregates. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 42–53, 1999.
12. S. Sarawagi. idiff: Informative summarization of differences in multidimensional aggregates. *Data Min. Knowl. Discov.*, 5(4):255–276, 2001.