SeeDB: What's interesting about my query?

ABSTRACT

Data scientists analyzing large amounts of data often rely on visualizations to identify interesting trends and gain insights. However, picking the right visualization is a manual and tedious task. Given a query, the analyst would like to know what makes the results of a query "interesting" compared to the underlying dataset. In this paper, we demonstrate a prototype of SEEDB a system that automatically discovers differences in trends between a query and the underlying dataset, and visualizes the differences to help data analysis.

1. INTRODUCTION

Data scientists analyze large amounts of data to find interesting trends and gain novel insights. Given the large scale of data and relative ease of visual analysis, analysts often use visualizations as a means to identify properties of interest. The general analysis workflow is usually ad-hoc and can be represented as follows: (1) The analyst selects a subset of the data they are interested in; for instance, in a medical use-case (Section 1.1), the analysts may select patients that had expenses more than twice the standard deviation of all patients. (2) Next, the analyst processes the data in various ways, e.g. via binning, aggregates, group-bys etc. to generate a large number of "views" of the data and builds visualizations such as scatterplots, histograms, pie charts etc. for each view of the data. (3) The analyst then studies all the views and identifies the *interesting* ones. (4) Once he/she has identified the interesting views, he/she then drills down into these views or further explores the properties that show unusual trends.

Of the four steps in the above process, only two (Steps 1 and 4) require creative thinking and analyst input. Steps 2 and 3 are tedious and time-consuming and ideal candidates for automation. Specifically, given the data that the analyst is interested in (result of Step 1), we can automatically generate various views of that data, evaluate each one for "interesting"-ness and only surface the most promising views. SEEDB automates these labor-intensive parts of data analysis.

1.1 Motivating Example I

Proceedings of the VLDB Endowment, Vol. 7, No. 4

Consider a medical researcher studying the cost of care for cancer patients. Her research involves the analysis of a set of 1M electronic medical records (EMRs). To analyze this data, the researcher identifies patients that cost significantly more than the average: specifically, she selects patients whose cost of care is greater than the average cost by two standard deviations. In terms of SQL, she runs the following query:

```
Q = SELECT * FROM Patients where total_cost –
(SELECT AVG(total_cost) from Patients) as avg_cost
> 2 * (SELECT STDDEV(total_cost) from Patients);
```

Once she has identified these patients, she would like to study various aspects of their care to determine the reason why the patients have large cost of care. For instance, she may study length of treatment, survival rate, severity of disease etc. For each of these parameters, she is interested in determining how the group of patients with high cost of care are different from the overall group of patients. As a result, she may construct various views of the data that compare various metrics between the high cost patients and the overall patient population. For instance, she may compare the distribution of length of treatment for the two populations, the average severity of the disease etc. Since there are a large number of metrics that may be responsible for high cost of care, the analyst must construct, visualize and examine a large number of views to identify interesting trends. For more than 5 metrics, this process quickly becomes tedious and time-consuming. We can significantly simplify and speed up the analysis process if we can automate the creation and evaluation of views.

1.2 Motivating Example II

Consider a store owner who wants to figure out why a certain category of products (say staplers), is not selling well.

Dataset: For this example, we assume that we are working with the canonical SuperStore dataset [**?**] commonly used in Tableau [**?**]. This dataset has all sales data for a store selling 50 categories of products over XXX years. Stored as a star-schema, the dataset has separate tables for Geography, Client information, Shipping details, Dates etc. connected to the main Sales table via foreign-key constraints. We show a subset of the schema in Table **??**.

Suppose the store owner runs the following SQL query to select the relevant stapler sale records.

```
Q = SELECT * FROM sales join products where
products.name = "Staplers" and sales.year = 2013
```

To understand why staplers sold poorly in 2013, the owner may want to compare how staplers did compared to all the products in the store (or another type of office supply product). Since the number of records is too large to view individually, the owner will aggregate sales along various dimensions such as month, geography,

customer type etc. and compare the distribution of stapler sales to overall sales distribution. Any significant differences in two distributions can point to potential reasons why stapler are going poorly (e.g. no staplers were shipped to Argentina which is otherwise a major purchaser of office supplies). We will use this example and the associated dataset as our running example in the paper.

1.3 Summary

In this paper, we describe our system called SEEDB [**?**] that partially automates the data analysis process. Given a query posed by the user, SEEDB can compute a large number of views based on that query, determine the "interesting"-ness or utility of each of the view and show to the user only those views that it deems most interesting. The researcher can then focus only on the important trends in the data. SEEDB is based on the principle that it is the **deviations from expected behavior that make a view interesting**. For instance, in the above example, the researcher would be interested in the fact that high-cost patients actually visit a specific set of doctors compared to the entire patient population. Distribution of doctors across patients would not be interesting if the distribution was similar for the high cost patients and the overall population. SEEDB therefore assigns higher utility to views that show divergent trends in the query and the underlying dataset.

In the process of automatically producing an interesting set of views for any query, SEEDB must address a few challenges: (a) the size of the space of potential views increases exponentially with the number of attributes in a table, as a result, SEEDB must intelligently explore this space; (b) computing each view and its utility independently is expensive and wasteful, and hence SEEDB must share computation between queries; and (c) since visual analysis must happen in real-time, SEEDB must tradeoff accuracy of views for reduced latency. In Section **??**, we describe how SEEDB addresses these challenges.

Our contributions are as follows XXX:

- We present a practical implementation of the SEEDB system based on and by extending the original vision paper [**?**].
- We discuss and evaluate various optimizations required to build such a system with realistic response rates. We also discuss how to make interactivity a part of the system.
- We show results of our user study demonstrating the utility of such a tool and the associated performance studies.

2. PROBLEM STATEMENT

In this paper, we describe our prototype of SEEDB, a system to automatically identify and visually highlight interesting aspects of a dataset. Given a database D and a query Q, SEEDB finds and visualizes the most interesting aspects of Q with respect to the underlying dataset.

To identify the most interesting aspects of Q, SEEDB considers various "views" of the query, where a view can be some way of slicing or aggregating the data so that it may be visualized in terms of histograms, time series charts etc. Currently SEEDB limits its views to those that can be generated by adding one aggregate and one group-by clause. To illustrate, consider the query Q for "Stapler" product sales in 2013 from the SuperStore dataset. A possible "view" of the input query can be constructed by *aggregating the sales data by region*. The corresponding SQL query, which we call the *input view query*, is shown below.

```
Q = SELECT Geography.Region, AVG(Sales.sales) FROM
Sales JOIN Geography JOIN Product WHERE
Products.name = "Staplers" AND Sales.year = 2013
GROUP BY Geography.Region
```

The above query produces the result table shown in Figure **??**.a that can then be visualized as a histogram shown in Figure **??**.b. To determine if this view is interesting or useful, SEEDB applies the same view, i.e., the same aggregate and group-by, to the entire dataset, D. We call this query the *data view query* (shown below). This query generates the result table shown in Figure **??**.c and histogram from Figure **??**.d.

```
Q = SELECT Geography.Region, AVG(Sales.sales) FROM
Sales JOIN Geography JOIN Product
GROUP BY Geography.Region
```

Results of *input view query* and *data view query* are comparable since they have the same values in column 1. SEEDB computes the utility of this view as the difference in the distribution of results (after normalization) of these two view queries. Using one potential metric, the Earth-Mover-Distance, we can compute the difference between these two queries to be XXX. To get the most interesting views, SEEDB must compute and evaluate the utility of a large number of such views and pick the views with largest utility.

2.1 Definitions

We classify table attributes into two types for use by SEEDB.

- **Dimension Attribute**: An attribute is considered a dimension attribute if the attribute has categorical or ordinal values. Dimension attributes are those that users can insert into a group-by clause.
- **Measure Attribute**: An attribute is considered a measure attribute if the attribute is numeric and can be aggregated. Measure attributes are usually metrics that the user cares about (e.g. sales, cost etc.) and can be inserted into aggregate clauses.

We denote the subset of data selected by query Q as $D(Q)$ and a view $\mathcal{V}$ with group-by attribute d_i and aggregate attribute m_j as $V(Q, D, d_i, m_j)$. The normalized results of the *input view query* are denoted by $P_{\mathcal{V}}(Q)$ and those of the *data view query* by $P_{\mathcal{V}}D$ (the results are normalized to values $\in [0, 1]$ so that the sum of the aggregates = 1).

2.2 Utility Function

SEEDB uses a utility function or distance metric, $\mathcal{U} : \mathcal{R}^n \times \mathcal{R}^n \rightarrow R$, to determine the difference between the two results, $P_{\mathcal{V}}(Q)$ and $P_{\mathcal{V}}(D)$. Higher the difference, larger the utility. SEEDB can use a variety of existing metrics, a few of which are discussed below.

- **Earth Movers Distance**: Commonly used to measure differences between color histograms from images, EMD is a popular metric for comparing discrete distributions.
- **Euclidean Distance**: The L2 norm or Euclidean distance considers the two distributions are points in a high dimensional space and measures the distance between them.
- **Kullback-Leibler Divergence**: K-L divergence measures the information lost when one probability distribution is used to approximate another.
- **Jenson-Shannon Distance**: Based on the K-L divergence, this distance measures the similarity between two probability distributions.

2.3 Formal Definition

Given an input query Q, a dataset D with dimension attributes $\mathcal{D} = \{d_1, d_2 \ldots d_m\}$ and measure attributes $\mathcal{M} = \{m_1, m_2 \ldots m_n\}$, a utility function $\mathcal{U} : \mathcal{R}^n \times \mathcal{R}^n \rightarrow R$, and a positive integer k, SEEDB computes and returns the top-k views of query Q generated by adding a group-by dimension attribute $d_i \in \mathcal{D}$ and an aggregate measure attribute $m_j \in \mathcal{M}$ that have the highest utility as defined by utility function $\mathcal{U}$.

2.4 Extensions

There are two important variations of the problem that can be addressed exactly by the same techniques discussed in the paper. These are:

- **Group-by clauses with multiple dimension attributes**: It is straightforward to extend the SEEDB techniques to group-by clauses with multiple attributes. However, for ease of exposition and visualization, we limit the number of attributes in the group-by clause to one.
- **Comparison of two-queries**: Instead of comparing the views of the input query to the entire underlying dataset, it may be more appropriate to compare them to another subset of the data (e.g. sales of "Staplers" vs. sales of "Printers"). This simply involves replacing the dataset parameter D with a second query Q'. This variation is important since it can help users find interesting differences in data. Our techniques apply to this variation unchanged and we show experimental results for this variation.

3. SEEDB SYSTEM ARCHITECTURE

Figure 1 shows the architecture of our system. Currently, SEEDB is a wrapper around a database. While optimization opportunities are restricted by virtue of being outside the DBMS, we believe that this design allows quick iteration and permits SEEDB to be used with a variety of databases.

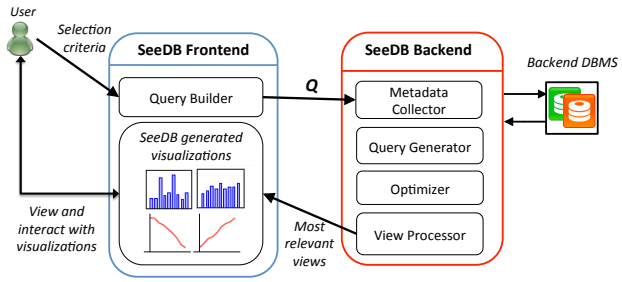

Figure 1: SeeDB Architecture

The SEEDB front end allows the user to make queries to the SEEDB backend through several means including raw SQL, via a query-builder and via pre-formulated queries (more in Section **??**). Once user queries are sent to the backend, the Metadata Collector queries the relevant tables for information such as table sizes, column types, table statistics etc. This metadata is essential for the subsequent processing performed by SEEDBṪhe user queries along with metadata are then sent to the Query Generator. The purpose of the Query Generator is two-fold: first, it uses the metadata and user query to knock down the search space of possible view queries, and second, it generates the remaining view queries that must be run against the database. The Optimizer is responsible for determining the best ways to combine view queries so that total query execution time is minimized. We discuss the optimizations performed by the Query Generator and Optimizer in Section 4. With the optimized queries in hand, SEEDB asks the DBMS to execute queries. Query results are returned to the View Processor that ingests combined query results in a streaming fashion and produces results for individual queries. The View Processor is also responsible for selecting the most interesting views and returning them to the user.

4. SEEDB BACKEND

$T(C_i)$	Data type for column C_i
$\|C_i\|$	Number of distinct values in C_i
$Var(C_i)$	Variance of values in C_i
$Corr(C_i, C_j)$	Correlation measure for all pairs of columns
$\mathcal{H}_{i\ldots k}$	Hierarchies between columns C_i to C_k
f_{C_i}, f_{C_i,C_j}	Frequency of access for each column and column pair

Table 1: Statistics and Table Metadata

One of the chief challenges in SEEDB is producing the most interesting views of the data in the minimum time. For the set of dimension attributes $\mathcal{D}$ and measure attributes $\mathcal{M}$, the number of potential views for any query is $O(|\mathcal{D}| * |\mathcal{M}|)$. Even for a modest dataset of 1M rows, 10 dimension attributes and 10 measure attributes, the time for computing all potential views is too large to permit realistic response times. XXX: Add table or chart.

As a result, SEEDB must aggressively prune views and optimize query execution. Pruning of the view space is performed in the Query Generator and optimization of queries is performed by the Optimizer module. We first describe the basic SEEDB algorithm and then discuss our optimization in detail.

4.1 Basic Framework

Given a user query Q, the basic version of SEEDB obtains the list of dimension and measure attributes, and computes all possible view queries by adding a single aggregate and a single group-by clause to Q. Each of the view queries is then executed independently at the backend along with an equivalent aggregate+group-by query on the complete underlying dataset. The two resulting distributions are compared using the chosen distance metric (Section 2) and the top k views with the largest utility are chosen.

4.2 View Space Pruning

To minimize the SEEDB response time, we aggressively prune view queries that are unlikely to generate interesting views. This pruning, done in the Query Generator, is based on prior knowledge about data statistics as well as access patterns for the table, if available. Specifically, no expensive scans of the underlying tables are performed. In addition, we order the execution of view queries so that higher utility views can be computed before those with lower utility, thus permitting early stopping. For each table in the DBMS, we assume that statistics from Table 1 are available or can be computed cheaply. The data type for each column is numeric, categorical, ordinal, geographic, or date_or_time. The data type for column C_i, $T(C_i)$, along with the number of distinct values $|C_i|$ is used to determine whether the column will be treated as a *dimension* attribute or a *measure* attribute (Section 2.1). As before, we denote the set of dimension attributes by $\mathcal{D}$ and measure by $\mathcal{M}$.

We employ the following heuristics for pruning and ordering views based on the statistics above.

4.2.1 *No variance rule*

If a dimension attribute has 0 variance, we remove it from $\mathcal{D}$. We cannot prune a measure attribute with 0 variance.

4.2.2 *Correlation-based clustering*

We cluster dimension attributes and measure attributes separately based on their correlation, $Corr(C_i, C_j)$. As a result, we identify groups of attributes that are highly correlated (correlation coefficient > threshold). Highly correlated attributes will produce similar views, and therefore we pick only a single representative attribute from each set and prune the rest from $\mathcal{D}$ and $\mathcal{M}$. Correlations between columns are also used by the optimizer while combining queries **??**.

4.2.3 *Bottom-up hierarchy traversal*

Name	dim attr	measure attr	$D * M$	nrows	size (GB)
XS	5	2	10	10M	0.5
S	50	5	250	10M	3.8
M	100	10	1000	10M	7.7
L	500	20	10000	10M	26

Table 2: Micro-benchmark Datasets

We observe that for a set of dimension attribute with a hierarchial structure, $H_{C_{i\ldots k}}$, if a view V at hierarchy level h has utility u, then views at hierarchy level $h-1$ will have utility $\leq u$. XXX: is this true for all utility functions?

4.2.4 Order views by frequency of attribute access

In order to surface the most interesting views of the dataset, we can use access traces for the queried tables. Specifically, a view $V_i = V(Q, D, d_i, m_i)$ is ordered before view $V_j = V(Q, d_j, m_j)$ if frequency of access of attributes in V_i is greater than that of V_j, i.e., $f_{d_i,m_i} > f_{d_j,m_j}$ or $max(f_{d_i}, f_{m_i}) > max(f_{d_j}, f_{m_j})$.

It is possible to collect the above statistics at the dataset level too, as opposed to the entire table level. The advantage of table level statistics is that they have to be computed only once per table; however, dataset-level statistics are more accurate since they only consider the specific parts of the table. XXX: we use dataset-level statistics with table statistics do not result in aggressive pruning.

These rules are applied by the Query Generator to generate a reduced space of views ordered by their potential utility.

4.3 View Query Optimizations

Although the Query Generator knocks down the view query space, the resulting view queries are very similar (differing only in the group-by and aggregates). As a result, we can optimize these queries to execute them more efficiently. For each of the optimization strategies described below, we ran micro-benchmarks to measure the effect of each optimization independently. We ran the benchmarks on four datasets with properties shown in Table 2.

4.3.1 Rewrite view query

Since SEEDB executes the same group-by+aggregate query twice for each view (once for the query and once for the dataset), one straighforward optimization is to rewrite these two queries as one query. Our microbenchmarks test the effect of this optimization for varying selectivities of the input query Q. We note the presence of specific indexes may make this optimization less useful (e.g. when we are comparing two datasets which are highly selective and have indexes optimized for the specific selection). However, we do not explore these cases further since they are heavily dependent on the type of selection predicate and presence of indexes. Our experiments show this simple optimization to achieve a speed-up of Y% **??** when we compare against the entire underlying dataset.

4.3.2 Single Group-by Multiple Aggregates

A large number of view queries have the same group-by clause but aggregates on different attributes. Therefore, SEEDB combines all view queries with the same group-by clause into a single view query. This rewriting provides a speed up linear in the number of aggregate attributes.

4.3.3 Multiple Aggregate Computation

Similar to data cubes ([]), SEEDB seeks to compute a large number of group-bys. As a result, we can combine queries with different group-by attributes into a single query with multiple group-bys attributes. For instance, consider view queries $V(Q, A_1, G_1)$, $V(Q, A_1, G_2) \ldots V(Q, A_1, G_n)$. Instead of executing them individually, we can rewrite them into a single view query $V(Q, A_1, (G_1, G_2 \ldots G_n))$. While this optimization can reduce the number of queries executed, the number of group by attributes we can include in a single query depends on the correlation between values of the various attributes (this affects number of distinct groups) and the working memory. We can compute the optimal combinations of group-bys by modeling the problem as a variant of bin-packing. The correlation between group-by attributes can be used to approximate the number of distinct groups that will be produced by a group-by set. Note that the grouping-set functionality, if available can be used to make this process efficient. The bin-packing formulation for grouping-sets is as follows.

4.3.4 Sampling

The optimization that can have the most impact in terms of efficiency is to reduce the number of tuples examined by constructing a data sample and running all queries against the sample. As expected, the sampling technique and size of the sample can affect the accuracy of the generated views.

4.3.5 Parallel Query Execution

Finally, we take advantage of the ability of DBMSs to run queries in parallel and potentially share scans of the underlying table. We find that although the execution time of a single query increasing while running in parallel (potentially due to some amount of thrashing), the overall time required to execute a set of queries decreases.