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 a system SeeDB a system that automatically discovers statistical differences between the query results and the underlying dataset, and visualizes the differences to aid data exploration.

1. INTRODUCTION

Data scientists analyze large amounts of data to find new insights, trends and anomalies. Given the large amount of data and large bandwidth available for visual analysis, analysts often use visualizations to identify areas of interest. The general workflow, which is usually ad-hoc, can be represented as follows:

1. Step 1: Analyst selects a subset of the data they are interested in; for instance, the analysts may select patients that had costs more than twice the standard deviation of all patients, or the genes that belong to a particular group, or a set of products of a certain kind.

2. Step 2: The analyst processes and visualizes the data in various ways, e.g. via binning, aggregates, groupbys etc. and then building scatterplots, histograms etc., and studies these "views" of the data.

3. Step 3: Finally, the analyst studies all the views and identifies the interesting ones. He/she then drills down into these views or further explores the properties that show unusual trends. This can be done using sophisticated statistics or machine learning.

This iterative process is repeated for different subsets of the data.

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

1.1 Motivating Example

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:

$Q = SELECT * FROM Patients where total_cost - AVG(total_cost) > 2 * STDDEV(total_cost)$; ■ fix this query ■

Once she has identified these patients, she must 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 whether the group of patients with high cost of care are different from those with lower cost of care. Since a large number of views of the data are possible, performing these calculations repeatedly and visualizing the results is tedious and time-consuming. Moreover, since the researcher may not be able to explore all potential features, she may miss some interesting trends.

SeeDB partially automates the process by computing and visualizing various views of the data and highlighting the interesting views. Moreover, SeeDB is based on the principle that *what makes a view interesting is deviations from expected behavior*. Therefore, given the subset of data selected using a relational query, what makes the data interesting are trends or properties that are different for this subset compared to the underlying complete dataset.

Ultimately, the analyst must decide if deviations in the data are interesting but we can use deviation as a means to eliminate the laborious process of stepping through all possible views of a dataset.

1.2 Problem Description and Challenges

Formally, the problem can be described as follows: given a database and a query Q, SeeDBidentifies the most significant deviations between the subset of the data selected by the query and the underlying dataset. SeeDBdoes so by considering the space of views obtained by adding a single group-by and aggregate to the input query, and then computing the discriminating power of each of these views. Discriminating power or utility measures how different the distribution of data in the subset is compared to the underlying data. SeeDBproduces the top-k views having the highest di-

criminating power. For the utility measure, SeeDBprovides the option of using distance metrics such as earth movers distance, L2 norm etc.

While various ways of obtaining "views" of a dataset are possible, for the purpose of this work, we focus on views obtained by adding a single group-by and a single aggregate clause to the query used to select subsets of the data. This restriction also allows us to focus on a limited set of visualization types to show trends in the data.

For simplicity, we assume that a discriminating view R in $\mathcal{R}_Q$ performs a group-by on a single attribute $a \in A$, and applies an aggregation function $f \in F$ on a single measure attribute $m \in M$. A view in this class corresponds to a two-column table that shows how the value of $f(m)$ varies with values of attribute a. This table can be directly visualized using a histogram, a bar chart or a line plot. (We consider generalizations in Section 4.)

We also assume the existence of a function $U(R)$ that can characterize the utility of each view $R(Q)$ in $\mathcal{R}_Q$ (higher is better). For now, we focus on picking discriminating views that optimize $U(R)$ with latency as low as possible: we return to more general objectives in Section **??**.

Given $Q \in \mathcal{Q}$ and a positive integer K, find K discriminating views $R_i \in \mathcal{R}_Q$, such that the R_i have the largest values of $U(R_i)$ among those in $\mathcal{R}_Q$, and the total latency is minimized.

There are several technical challenges that need to be addressed:

- For a given query, n, the total number of discriminating views, (even if we restrict ourselves to views that append a group-by and an aggregation) is likely to be very large to explore exhaustively and precisely. Generating each of $R_1(Q(D))$, ..., $R_n(Q(D))$, scoring them on utility, and then picking the best one is certainly not feasible for most databases. Thus, we need mechanisms to prune the space of views and compute their utility approximately.
- Generating and scoring the discriminating views $R_i(Q(D))$ one-by-one may miss interesting optimization opportunities: First, we may share computation between discriminating views. For example, the results of two views with different aggregates but the same group-by may be computed together in one query, followed by projecting out to reveal the two individual views. Second, by evaluating the discriminating views in a deliberate order, we may be able to prune views with low utility (without evaluation) that are definitely not going to be recommended to the analyst.
- Since visualizations tend to convey approximate information, e.g., a trend in a line plot may be more important than knowing the exact coordinates of each point, we can introduce approximations as part of SeeDB. Thus, the utility of a discriminating view may be computed approximately but efficiently, and the recommended discriminating views can be populated with approximate results, based on synopses of the base data or of the query result, that can be generated much more efficiently.

1.3 State-of-the-Art Approaches

Over the past few years, there has been a significant effort from the visualization community to provide interactive tools for data analysts. In particular, tools such as ShowMe, Polaris, and Tableau [16, 10] provide a canvas for data analysts to manipulate and view data, tools such as Wrangler [7] allow data analysts to transform and clean data, and tools such as Profiler [8] allow users to visualize simple anomalies in data. However, unlike SeeDB, these tools have little automation; in effect, it is up to the analyst to generate a two-column result (like the result of the discriminating view) to be visualized. Other related areas of work include OLAP and database visualization tools. There has been some work on browsing data cubes, allowing analysts to variously find "explanations" for why two cube values were different, to find which neighboring cubes have similar properties to the cube under consideration, or get suggestions on what unexplored data cubes should be looked at next [12, 14, 13].

Fusion tables [4] allows users to create visualizations layered on top of web databases; they do not consider the problem of automatic visualization generation. Devise [9] translated user-manipulated visualizations into database queries.

In this demo paper, we describe our prototype of SeeDB, a system to automatically identify and visually highlight interesting aspects of a dataset.

2. SYSTEM ARCHITECURE

2.1 SeeDBoveriew

Figure **??** shows the architecture of our system. Currently, SeeDB is a wrapper around a database (PostgreSQL in this case). While optimization opportunities are restricted by virtue of being outside the DBMS, we believe that it allows quick iteration and permits SeeDB to be used with different backends.

Once a user issues a query Q to SeeDB the system generates potential views by rewriting Q with various group-bys and aggregates. Each such rewritten query is termed as a "view." These views are then grouped according to optimization opportunities discussed in 2.3 and sent to the DBMS. The results of these view queries are then processed by SeeDBto compute the utility of views. The top-k views with highest utility are picked, appropriate visualization techniques chosen and the top views are visualized at the SeeDBfront end. The analyst can then examine these views and perform further processing.

2.2 Basic Framework

Given a user query Q, the basic version of SeeDB computes all possible view queries by adding a single aggregate and group-by operator. Each of these view queries is executed at the backend along with an equivalent aggregate+group-by query on the complete underlying dataset. These two query results produce a "distribution" for the attribute that has been aggregated. These two distributions are compared using the chosen distance metric (default: earth movers distance, other: L2, Jensen-Shannon distance etc). All the views are ranked by the distance between the query and dataset distribution and the top k views with the largest distance are chosen. Appropriate visualizations are chosen for these distributions (heuristics in Section 2.4) and are displayed to the user for further interaction.

Utility Metric:

One of the key challenges behind SeeDB is formalizing the utility function $U(R)$ for a discriminating view R. There are many choices for U and we expect SeeDB to recommend views that score high on several metrics. As discussed previously, the proposed metric tries to capture the idea of "deviation" between distributions, i.e., a view has high utility if its contents show a trend that deviates from the corresponding

trend in the original database.

We first define some notation. For any discriminating view R_i in the class defined above, we note that $R_i(D)$ and $R_i(Q(D))$ are both two column tables. A two-column table can be represented using a weight vector. We let the weight vector $W_{a,f(m)}$ represent the result of $R_i(D) = \gamma_{a,f(m)}(D)$, i.e., distribution of the aggregate function f on the measure quantity m across various values of the attribute a.

The utility U of a discriminating view $\gamma_{a,f(m)}$ is defined to be the distance between $W^Q_{a,f(m)}$, and $W_{a,f(m)}$: $U(\gamma_{a,f(m)}) = S(W^Q_{a,f(m)}, W_{a,f(m)})$ where S is a distance metric. The higher S is, the more useful a discriminating view is. Common distance metrics used in visualization literature include K-L divergence [19], Jenson-Shannon distance [18, 17], and earth mover distance [20]. Wang [17] provides a good overview of the metrics used in scientific visualizations, while [20] provides a summary of probability-based distance metrics. As discussed earlier, we do not prescribe any specific distance metrics, instead, we plan to support a whole range of distance metrics, which can be overridden by the data analyst.

2.3 Optimizations

The fact that SeeDB evaluates a large number of possible views presents various opportunities for optimization. The strategies include:

1. *Rewrite view query*: Since similar group-by and aggregate queries are executes on the results of user query Q and the underlying dataset, we can combine these queries into one query. As shown in Figure **??**.a this achieves a speed-up of Y%.

2. *Single Group-by Multiple Aggregates*: A large number of view queries have the same group-by clause but aggregates on different attributes. A straightforward optimization 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. (Figure **??**. b).

3. *Multiple Aggregate Computation*: Similar to data cubes, SeeDB seeks to compute group-bys for a large set of attributes. One optimization is to combine queries with different group-by attributes into a single query with mulitple group-bys. For instance, consider view queries $VQ(Q, A_1, GB_1), VQ(Q, A_1, GB_2) \ldots VQ(Q, A_1, GB_n)$. Instead of executing them individually, we can rewrite them into a single view query $VQ(Q, A_1, (GB_1, GB_2 \ldots GB_n))$. While this strategy reduces query execution time, SeeDB must spend more time combining the results and obtaining separate aggregates for individual GB_i's. This is reminiscent of data cube algorithms. As shown in Figure **??**.c the speed up depends closely on the number of distinct values for each of the group-by attributes and the memory constraints of the DBMS.

4. *Sampling*: The final optimization we study for the purpose of this demo is sampling. Instead of running queries on the entire dataset, we run queries on subsets of the data at the expense of reduced accuracy. Figure **??**.d shows the effects of this optimization. ■ write about accuracy ■

   Although other optimizations are possible, particularly related to pre-computing aggregate results, we discuss them in the full paper currently in preparation.

2.4 User Interface

■ write about user interface, add pictures■

3. DEMO WALKTHROUGH

4. CONCLUSIONS

We outlined our vision for SeeDB: a system that provides analysts with visualizations highlighting interesting aspects of the query result. We defined several concrete problems in architecting SeeDB, relating to areas ranging from multi-query optimization and approximation to multi-criteria optimization. We believe that the general area of bringing visualizations closer to the DBMS is a challenging, yet, important direction for database research in the future.

5. REFERENCES

[1] K. Chakrabarti et al. Approximate query processing using wavelets. In *VLDB*, pages 111–122, 2000.

[2] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[3] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, pages 541–550, 2001.

[4] H. Gonzalez et al. Google fusion tables: web-centered data management and collaboration. In *SIGMOD Conference*, pages 1061–1066, 2010.

[5] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In J. Peckham, editor, *SIGMOD 1997*, pages 171–182. ACM Press, 1997.

[6] C. Jermaine et al. Scalable approximate query processing with the dbo engine. *ACM Trans. Database Syst.*, 33(4), 2008.

[7] S. Kandel et al. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.

[8] S. Kandel et al. Profiler: integrated statistical analysis and visualization for data quality assessment. In *AVI*, pages 547–554, 2012.

[9] M. Livny et al. Devise: Integrated querying and visualization of large datasets. In *SIGMOD Conference*, pages 301–312, 1997.

[10] J. D. Mackinlay et al. Show me: Automatic presentation for visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1137–1144, 2007.

[11] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. SeeDB: Visualizing Database Queries Efficiently. *Stanford Infolab*, 2013.

[12] S. Sarawagi. Explaining differences in multidimensional aggregates. In *VLDB*, pages 42–53, 1999.

[13] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, pages 307–316, 2000.

[14] G. Sathe and S. Sarawagi. Intelligent rollups in multidimensional olap data. In *VLDB*, pages 531–540, 2001.

[15] T. K. Sellis. Multiple-query optimization. *ACM TODS*, 13(1):23–52, 1988.

[16] C. Stolte et al. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84, 2008.

[17] C. Wang and H.-W. Shen. Information theory in scientific visualization. *Entropy*, 13(1):254–273, 2011.

[18] Wikipedia. Jensen shannon divergence — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].

[19] Wikipedia. Kullback leibler divergence — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].

[20] Wikipedia. Statistical distance — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].