QueryScope: Visualizing Queries for Repeatable Database Tuning*

Ling Hu
Northeastern University
Boston, MA, USA
audnyhu@ccs.neu.edu

Yuan-Chi Chang Christian A. Lang
IBM T.J. Watson Research Center
Hawthorne, NY, USA
{yuanchi, langc}@us.ibm.com

Kenneth A. Ross
Columbia University
NYC, NY, USA
kar@cs.columbia.edu

Donghui Zhang
Northeastern University
Boston, MA, USA
donghui@ccs.neu.edu

"To understand is to perceive patterns." — Isaiah Berlin

ABSTRACT

Reading and perceiving complex SQL queries has been a time consuming task in traditional database applications for decades. When it comes to decision support systems with automatically generated and sometimes highly nested SQL queries, human understanding or tuning of these workloads becomes even more challenging. This demonstration explores visualization methods to represent queries as graphs. We developed the *QueryScope* tool to help visualize and understand critical elements of a query, thereby cutting down the learning curve. We show how the tool allows the user to drill down on particular queries or to find similarly structured queries that may exhibit similar tuning opportunities. The queries shown in the demonstration are taken from real tuning engagements.

1. INTRODUCTION

The market of decision support systems and business intelligence is growing every year. Building data warehouses to support such systems is a serious undertaking and involves a variety of tools for modeling, ETL, and tuning. With business requirements getting more and more complex, so do the supporting tools, resulting in sometimes very long and convoluted queries being issued to the underlying database system. This is further magnified by the use of model-driven technology that, while enabling easier reuse and high-level understanding, also leads to increased nesting in queries and extensive table and column aliasing. Compared with a scenario in which all SQL queries are concise and written by hand, a bigger burden is placed on tuning the physical schema to achieve the required performance.

Autonomic tuning tools [1, 2, 6, 10, 9, 4, 3] are helpful, but they do not eliminate the need for physical tuning. Autonomic tuning does not scale well with query complexity. For example, the view selection problem has a lower bound complexity that is exponential in the query size [5]. For queries of even moderate length, only a tiny fraction of the search space of possible tuning decisions can be explored. Shasha and Bonnet [7] state the challenge well: "Tuning rests on a foundation of informed common sense. This makes it both easy and hard." The informed common sense is accumulated through learning and practising on real projects, something that is hard to capture in an autonomic tool.

Query Visualization

To address the problems discussed above, we have designed and implemented a prototype query visualization system we call *QueryScope*. The goals of this system are:

- To communicate the essence of a query (or a collection of queries) pictorially through a controlled visual semantics.
- To provide a variety of visualization options so that a user can focus on the aspects of queries of relevance.
- To visualize queries in the context of a physical schema, so that schema-specific information such as table sizes and indexes are incorporated into the query visualization.
- To facilitate searches for similar queries, from both current and prior engagements. A variety of similarity notions are supported.
- To make the tuning process productive and repeatable.

We show how the visualization allows a user to quickly identify important features of queries that would take much longer if one was working from the underlying SQL text. We also provide examples of how the visualization techniques were helpful in practice on real data and schema.

To our knowledge, we are the first to propose a systematic approach to visualize and search collections of large complex

*This paper describes a user interface that utilizes color in an essential way. We therefore request that the reviewers either read an electronic version of the paper, or print the document using a color printer.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-306-8/08/08

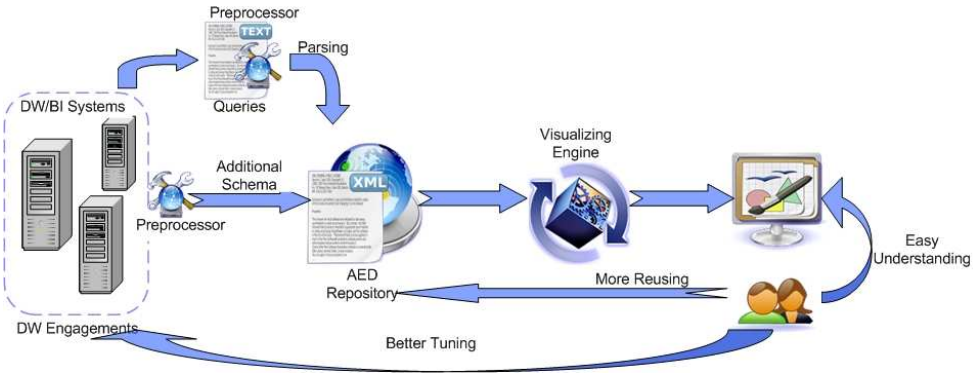

Figure 1: *QueryScope* Overview.

queries. *QueryScope* is an important step toward a discipline of repeatable data warehouse engagements.

2. QUERY TRANSFORMATION AND VISUALIZATION

This section presents the query ingestion and visualization techniques in *QueryScope*. The information flow for our tool is shown in Figure 1. A preprocessor reads and parses SQL queries occurring in the data warehouse workload and generates an XML representation of the queries' structure. The XML representation of the queries forms one part of the so-called *Abstract Engagement Descriptor* (AED). The AED also stores schema information, hardware characteristics, and data statistics; in short, everything that is needed to specify a tuning engagement. The visualization engine then reads the AED, and generates images based on the engagement information and user interaction.

2.1 The Abstract Engagement Descriptor

A set of SQL queries from one engagement is parsed by the preprocessor and transformed into XML format for storage in the AED. The preprocessor assumes that the queries have already been successfully compiled against the target database and it does not need to do extensive error checking. Important elements of the query are extracted and represented in a structured way. Tables, subqueries and joins between table columns are represented as XML elements with subquery nodes containing again other tables, subqueries, and joins. The schema information obtained from the underlying database system and other configuration information, such as hardware characteristics, database configuration parameters, data refresh rate and query execution frequency are also valuable information for tuning purposes. We do not record all of these in the AED for now, but plan to incorporate them in the future.

2.2 Visualization Primitives and Design

In this section we show how the AED is visualized. Our visualization focuses on three basic entities:

- *Tables*, including base tables, views, and materialized views.
- *Subqueries*, at arbitrary levels of nesting.
- *Links*, connecting combinations of tables and subqueries.

Tables are visualized using colored discs, with the area of the disc proportional to the table cardinality. Queries and subqueries are visualized using black circles, with the structure of the (sub)query shown within the circle. Links are visualized as lines between tables and/or subqueries. Figure 2 shows the visualization graphs of two queries, a simple one (left) and a more complex one (right).

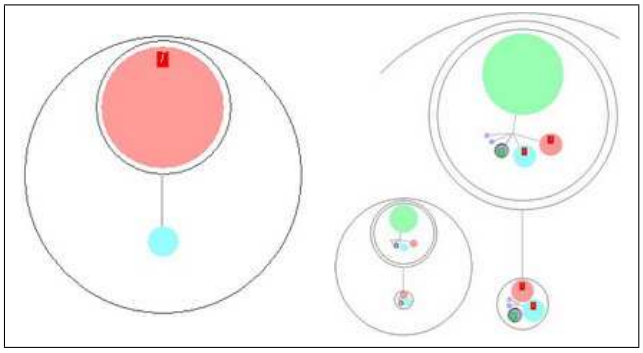

Figure 2: Visualization of Two Queries.

The left query graph in Figure 2 is a simple query example. The cyan disc represents one table and the pink disc represents another. The black circle around the pink disc means that the second table is referenced within a subquery. The grey line between the cyan disc and the subquery represents a join conditions between the table and the subquery. The small red icon in the pink disc stands for index(es) on that table. This icon is red if joins on that table cannot use the index column(s) and green if the index is used in joins.

The query graph on the right of Figure 2 comes from a real deployment and is therefore more complex. The full SQL query is 192 lines and is omitted due to space limitations. There are many table/column aliases, functions, join conditions and predicates in the original query. The textual information is overwhelming to readers. The smaller circle on the bottom left corner is a bird's-eye view of the whole query graph. The rest of the right figure is a zoomed-in view to show more detail. The tool supports a zoom in/out function so a user can enlarge any part of a graph to view details. Figure 2 gives the following initial clues to a user about the right query graph:

- There are two subqueries inside the query. The top subquery is more complicated; it has two nested subqueries with six tables inside the inner subquery.
- The biggest table (the topmost disc) in green has no index while the other three smaller tables have indexes.

- There are several red index icons, indicating that indexes may not be well chosen for the given joins.
- The subquery on the bottom has five tables and the three bigger tables have indexes while the two smaller tables do not.
- From the colors of the tables in each subquery, it appears that (apart from the big green disc in the top subquery) the tables being joined are the same (this can be verified by moving the mouse pointer over the tables as discussed below).

Some dynamic features of the visualization are not captured by the static representation of Figure 2. When a user moves the mouse over an entity, a tooltip will be displayed. The information displayed depends on the entity:

- Names and local selection conditions for tables or subqueries.
- A description of the index type (which columns, primary or secondary, etc.) for an index.
- The join conditions for a link.

We display this information only when a user's attention is focused on the entity. That way, the cognitive load required to absorb the overall query structure is reduced: no reading is required.

The main design elements that can be used to convey information in *QueryScope* are color, size, and placement. These are elements that the human visual system is very good at perceiving [8]. Hence we want them to convey the most important aspects of our query. Color is used to denote *table identity*. The size of a table (or view) corresponds to its row cadinality and subquery circles are scaled to accommodate the included elements. The placement of tables and subqueries is tricky and it is easy to end up with a graph with lots of overlaps and intersections. We therefore place the tables and subqueries in a circular and size-descreasing sequence in clock-wise order. Overlaps of tables/subqueries are carefully avoided by computing bounding outlines. The circular placement is scale-independent, and can be used at an arbitrarily level of nesting, while still efficiently using the available screen space. The circular orientation by size is a simple rule-of-thumb that users can apply in order to navigate the query. Finally, ordering the tables by size means that repeated patterns (say a join of 3 tables) still look the same in different places. The placement of links also requires some thought. To avoid lines intersection and overlap with other tables or subqueries, we chose to visualize links as a set of lines radiating outward from the center of the circle of tables/subqueries to the border of the discs/circles. Thereby, both line intersection and overlapping with discs/circles are avoided.

2.3 Workload Visualization

Queries in one database system are oftentimes interrelated, especially in analytics systems. To represent connections between queries, the *QueryScope* system uses a consistent color for the same table when it occurs in different queries. The layout of tables and subqueries in different query graphs conforms to the same principle. And the scaling factor will make sure the ratio between any pair of tables stays the same. The consistent layout and color assignment can ensure that if one table occurs in different queries, or pairs of tables join together in multiple queries, they can be identified as repeated patterns by the human eye.

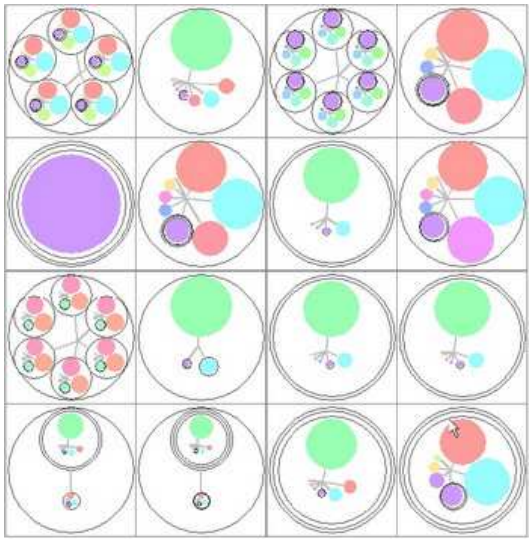

Figure 3: Workload Visualization.

Showing a collection of queries in one picture can help a user identify commonalities between different queries. Figure 3 shows such a collection as displayed by *QueryScope*. In this visualization, we can quickly recognize that there are some join patterns shared by multiple queries (in this example, there are four types of join patterns).

3. MINING AND MATCHING EXAMPLES

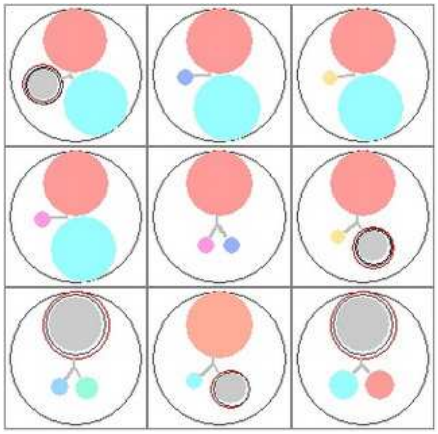

Figure 4: Top 9 3-way Join Patterns.

Similar join patterns are easy to perceive when the number of queries is reasonably small (say, less than 100). If there are thousands of queries stored in the repository, it is better to employ machine-based mining, similarity searching and ranking in order to reduce the number of candidates to a manageable quantity before presenting it to a user. *QueryScope* employees novel query graph mining and matching algorithms that try to emulate the human perceptual system but can scale up to thousands of queries. Our system currently provides two types of query mining algorithms: (1) common join pattern discovery, and (2) overall query similarity search.

The first algorithm (join pattern discovery) is useful for finding and eliminating performance bottlenecks shared by many queries. For example, if two or three tables are often joined together, the execution time may be reduced for all

involved queries by turning the common join pattern into a materialized view or by creating indexes on the columns involved in the common join. Figure 4 and Figure 5 show two steps in this process using *QueryScope*: first, the most common join patterns are discovered automatically (Figure 4) and then the user can pick one of them to explore all queries that contain that pattern (Figure 4). By exploring individual queries, maybe missing indexes are discovered and added, benefitting all the queries shown in Figure 5.

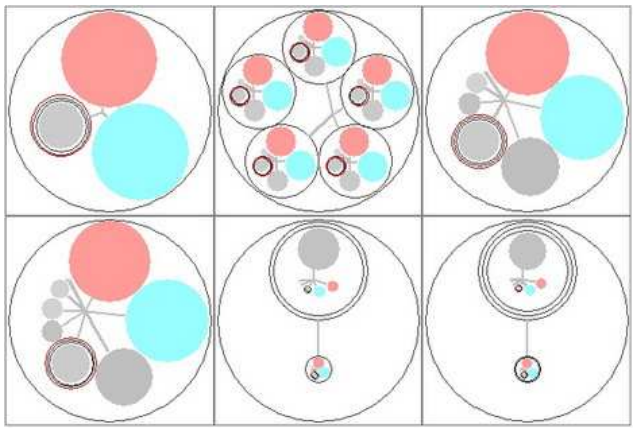

Figure 5: Queries Containing the 3-way Join Pattern in the Upper Left.

The second algorithm (query similarity search) is useful for carrying tuning knowledge forward from older engagements. For example, in real data warehouse engagements, DBAs often have a specific query that they wish to optimize due to the importance of the query or some unmet response time constraints. By using the query similarity search in *QueryScope*, the DBA can find the top-k most similar queries from the whole repository which contains thousands of queries from previous engagements. Figure 6 shows an example: given the query on the left, *QueryScope* returns the query on the right (from an older engagement) as the most similar query. It can be seen that there are additional indexes in the right query graph. Adding similar indexes to the current engagement may be a first step in improving its performance. The similarity measures used for finding similar queries are based on a combination of query tree structure similarity, participating table similarities, join expression similarities, and query cost similarities.

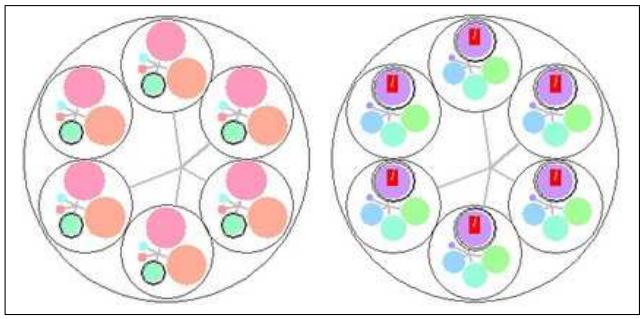

Figure 6: A Current Query Graph and A Query Graph in the Past.

4. DEMONSTRATION

QueryScope was written entirely in JAVA. It also employs the Apache Xalan library for XML parsing. In the demonstration, we will show a gallery of SQL queries obtained from real tuning engagements. By comparing the original query text with the visualization, we will show how the visualization captures the essence of a query and helps users to quickly understand the query. We will then show how users can quickly detect common query patterns by eyeballing the query collection. The demo will also walk through how the join pattern discovery and query similarity search can be employed to discover useful tuning opportunities. Finally, we will share some anecdotes on how the tool helped to discover and eliminate suboptimal query patterns in an automatically generated workload from a real engagement.

5. CONCLUSION

We propose *QueryScope*, a novel query workload visualization and exploration system. *QueryScope* uses compact visual semantics to capture key elements of queries. Enhanced with common query pattern mining and similarity search, it enables database consultants to look up queries captured in related data warehouse projects and examine opportunities for performance tuning. Conceived after observing and experiencing the ordeal of multi-page queries and hours of tuning, our work revisits the old challenges with a fresh angle.

While we have successfully used *QueryScope* in some of our engagements, a thorough assessment of the value of *QueryScope* would require putting it in the hands of many practitioners for use in real customer projects to judge tuning knowledge accumulation and repeatability of the tuning advice. We intend to pursue this avenue in the future and to release the tool for public trial. The VLDB demonstration will hopefully allow us to recruit participants for such a trial.

6. REFERENCES

[1] R. Ahuja. Self-tuning Memory in DB2 Version 9. *http://www.ibm.com/developerworks/db2/library /techarticle/dm-0606ahuja/*, 2006.

[2] N. Bruno and S. Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. In *SIGMOD*, pages 227–238, 2005.

[3] N. Bruno and S. Chaudhuri. To Tune or not to Tune? A Lightweight Physical Design Alerter. In *VLDB*, pages 499–510, 2006.

[4] S. Chaudhuri and V. R. Narasayya. Self-Tuning Database Systems: A Decade of Progress. In *VLDB*, pages 3–14, 2007.

[5] R. Chirkova, A. Y. Halevy, and D. Suciu. A Formal Perspective on the View Selection Problem. In *VLDB*, pages 59–69, 2001.

[6] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood. Automatic Performance Diagnosis and Tuning in Oracle. In *CIDR*, pages 84–94, 2005.

[7] D. Shasha and P. Bonnet. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*. Morgan Kaufmann, 2002.

[8] E. R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[9] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *VLDB*, pages 1087–1097, 2004.

[10] D. C. Zilio, C. Zuzarte, S. Lightstone, and W. M. et al. Recommending Materialized Views and Indexes With the IBM DB2 Design Advisor. In *International Conference on Autonomic Computing*, 2004.