About the Authors

Todd Deshane
Graduate Student
Division of Mathematics and Computer Science
Clarkson University

Patty Jablonski
Graduate Student
Engineering Sciences Program
Clarkson University

Kevin Roberts
Undergraduate Student
Division of Mathematics and Computer Science
Clarkson University

Dr. Jeanna Matthews
Assistant Professor
Department of Computer Science
Clarkson University

Contact Details

Dr. Jeanna Matthews, Todd Deshane, Patty Jablonski, Kevin Roberts
Department of Computer Science
Clarkson University
8 Clarkson Avenue, MS 5815
Potsdam, New York 13699 USA

{jnm, deshantm, jablonpa, robertkn}@clarkson.edu

Phone: 315-268-6288; Fax 315-268-2371

Web-Based Collaborative Exploration and Characterization of Large Databases

Main Description

Groups of people in many diverse fields face the challenge of characterizing or mining information from a large database. Typically, this exploration and characterization is done via individual long-running SQL queries with little support for collaboration among those issuing the queries. In this paper, we present a generic system for collaborative exploration of large SQL databases. Our system encourages collaboration by enabling users to reuse and build upon the queries issued by others. In addition to supporting collaboration among people, it also uses limited computing resources more efficiently by avoiding the repetition of commonly used, long-running queries by caching results. The system chooses which results to cache based on the run time of the query, the size of the result, and the user rating of the query. For queries that are already cached, it considers the number of times they have been accessed, the aggregate user rating of the query, and the degree to which the underlying tables consulted for the query have changed since the cached result was generated. Our system supports each user in their transition from new user to expert user. We introduce a categorization of users along this spectrum and offer benefits to each category of user to support collaboration. Our system is generic and can be used to explore any collection of data tables. The data tables can be inserted into our web-based system and with only minimal configuration, collaborative exploration can begin.

Short Description

We present the motivation for and design of a web-based tool for collaborative exploration and characterization of large relational databases.

Keywords

Collaborative Exploration

Database Characterization

1. Introduction

Groups of people in many diverse fields face the challenge of characterizing or mining information from a large data set. In science, the data set might contain astronomy or human genome data. In business, the data set might contain customer purchase or supply chain data. In sociology, the data set may contain census or demographic data. Companies like FedEx, UPS and Wal-Mart and scientific projects like the NASA space missions and the Human Genome Project all have terabytes of data being examined by hundreds of people.

Wherever large databases exist, regardless of their exact contents, there are groups of people seeking to understand that data. Often, this is done with individual SQL queries that summarize certain aspects of the data with little support for collaboration between people. People may work together on forming a query to submit to the system, but the system offers no direct support to help identify and exploit possible collaboration. This lack of collaboration has three main drawbacks.

- Running a query over a large data set can take hours or even days and users often run many of the same basic SQL queries to summarize basic aspects of the data set. With support for caching, users could benefit from answers obtained by other users.
- As users move beyond basic queries, it can often take several tries to write a query that accomplishes the intended purpose. With support for collaboration, users could learn from each other's mistakes by modifying previously run successful queries. It is easier to modify a working query than to write a completely new query from scratch.
- Users may be characterizing a similar aspect of the data, but would be unable to help one another modify their queries. Without support for collaboration, they would be wasting resources by running similar queries and would be unable to build on each other's ideas.

To address these problems, we present the design of a generic system for collaborative exploration of large data sets. Our system has the following characteristics:

1) It caches the results of commonly executed queries to avoid repeating expensive, long-running queries.
2) It automatically caches the results of basic data characterization queries like a histogram of values for each row element.
3) It allows new users to build on the successful queries of others, making the system easier for new users. This could reduce the number of expensive, long-running queries that are caused by user error and return undesired results.
4) It allows expert users to export polished query sets that answer important questions and benefit others.

5) It allows users to export collaborative query sets that invite others to refine queries in real time.
6) It provides a generic interface that will work across any data set that can be represented in a relational database. In other words, the system need not be customized based on the category of data being explored.

In this paper, we describe the design of our system and present the implementation of a prototype system that demonstrates many aspects of this design. We also tell of our experience using our prototype to support collaborative exploration of a large collection of Border Gateway Protocol (BGP) data.

In Section 2, we describe the use of query sets in our generic collaborative data exploration system. In Section 3, we present a categorization of users from new to expert and discuss the support provided by the system to each class of user. In Section 4, we describe our query caching system. In Section 5, we describe the classes of tables in the design of our system. In Section 6, we describe the implementation of the system and in Section 7 we describe how we used the system to explore a large set of publicly available Border Gateway Protocol data. In Section 8, we consider related work. Finally, in the remaining sections we conclude with contributions and future work, followed by our references.

2. Query Sets

A query set is simply a collection of queries that together characterize a certain aspect of a database. Query sets can be accompanied by a description of the query set as a whole and a description of each query. The system automatically generates a set of basic summary query sets that are useful regardless of the type of data stored in the system. Users can also export query sets that they have developed to answer a particular question. Finally, collaborative query sets can be used as a workspace for collaborative exploration.

2.1 Automated Query Sets

Regardless of the type of data stored in a large database, users begin with the same basic questions. They want to know how many tables are in the database and what the attributes of the rows in those tables are. They want to see a few sample rows from each table and know the distribution of values in each row element (e.g. maximum, minimum, average, histogram or other depending on the element type). Some of these queries like listing the tables or returning a few sample rows from a table can be run quickly. Others like a histogram of values for each row element would require long-running queries that pass over all data in the database.

These basic summary queries can be automatically generated given some basic information about the structure of the database. Also, the results of these

queries can be cached so that system resources need not be wasted on common queries that many users will want to execute. Finally, the system can determine when the cached results become invalid due to updates in the underlying tables. The system administrator and/or the users can decided whether to re-run the queries whenever the underlying tables change or whether to run them at preset intervals and to return slightly out-of-date answers in between.

A basic summary query set is just one type of automated query set. There are other automated query sets that apply to any relational database regardless of the database contents. For example, one automated query set is the set of most popular queries. Another is the set of the most recently run queries. These types of automated queries build a sense of community by providing users with a sense of one another's activity.

2.2 Polished Query Sets

To fully explore a large data set, users must move beyond automated queries to custom-crafted query sets that illustrate the types of interesting questions that can be answered from the data.

Our system allows advanced users to export or publish polished query sets to other users. These query sets consist of a set of queries designed to answer a particular set of specific questions from the data. In addition to the queries, the author of the query set can write an introduction describing the purpose of the query set and can write text explaining the purpose of each query. Having experts teach new users is one type of collaboration.

The benefits of these query sets to new users are clear. Without this opportunity, inexperienced users typically write many incorrect queries before they successfully write a query that returns data of interest. Also, without this opportunity, inexperienced users may have difficulty imagining all the ways in which the data can be used.

However, the benefits to advanced users deserve some additional explanation. First, the results from these query sets can be cached so that new users do not tax the system resources by repeating the queries. Second, without this help, new users would tax the system many times over with incorrect and inefficient versions of queries when attempting to answer similar questions. Third, our system maintains statistics about the use of each polished query set. In this way, advanced users can document the impact of their work on others. Fourth, system administrators could decide that users who have exported query sets should be given a higher priority when running their queries based on the popularity of their exported query sets.

When users examine a polished query set, they are given the opportunity to run modified versions of the queries in the set. Modifications are not incorporated into the finished query set, but the modified query is run on the database. By modifying existing queries, users have the potential of discovering new information without the hurdle of writing the original query from scratch. (Modifying a working example is always easier than starting from scratch!). Our

system also allows users to provide feedback on polished query sets to the original authors. Thus, those authors who are advanced users may receive some new insight that they can incorporate into the polished query set.

2.3 Collaborative Query Sets

In polished query sets, all modifications to the query set must go through the author. However, our design also includes another type of query set, the collaborative query set, which allows users to modify the queries in place. The users who open the collaborative query set are called the "query set owners" and have the opportunity to include a description of the problem they are trying to solve. Other users can see the set of queries that have been tried. With each query tried, users can enter an explanation of the results – in what way they represent an aspect of a solution to the problem and in what way they are lacking. Users can also choose not to save a query in the query set if they don't believe it is worthy of inclusion.

The query set owner can set limits on who can join the collaborative query set – either by identifying the participant by username or limiting participation based on the expertise of the user. Query set owners can also finalize the query set when they are satisfied with the solution. In this way, the collaborative query set can become a polished query set containing all the queries tried. Query set owners can also save a modified version of their query set containing only the final polished queries if desired. Alternatively, other users may decide to use those queries as a basis for other problems and branch off in a new direction.

3. Supporting Different User Groups

System support for collaboration takes on a different form for new users than for expert users. One of the primary design goals for our system is for it to support users in their transitions from new user to expert user in a seamless and effective way.

Most users will follow the same steps during this transition:

1) Get basic statistics and information about the database.
2) View previously run queries and their results.
3) Run new queries by modifying existing queries or by writing their own.
4) Assemble sets of useful queries for their own benefit and the benefit of others.

3.1 User Classification

Here we identify four specific classes of users based on their level of expertise with the system.

New users – New users are those users that are unfamiliar with this data set. New users may be experts in the field related to this data set – for example an astronomist characterizing a set of astronomy data – but they are new to this particular set of data, both its contents and the best way to write queries for the particular table structure in this database. These users will be directed to a set of basic queries that summarize the contents of the database – its table structure, number of rows in each table, range of values in each row, etc.

Interested users – Interested users are those users who have explored the database and can characterize its basic contents. By continuing to use the system, they have demonstrated that this set of data has the potential to help them. These users will be directed to a set of instructional and cached queries that illustrate interesting questions that can be answered using this data. In addition to suggesting interesting queries, these queries demonstrate important techniques for querying this particular data set like the join criteria needed to combine multiple tables or how to express a date range correctly.

Active collaborators – Active collaborators are those users who have exhausted the usefulness of the cached queries and are now ready to create queries of their own. These users can modify existing cached queries. They could also be directed to a query builder to aid in the formulation of a new query. They can now begin to test their own theories and hypotheses. They may consider participating in a collaborative query set with other active collaborators.

Expert collaborators – Expert collaborators are those users who have written successful queries of their own. These users have successfully characterized some portion of the data and can explain through a set of queries what results they have obtained. Thus, they are ready to export query sets for other users. Their results will allow others to learn and have something to build upon to explore and characterize the data better.

3.2 Moving Through the Classes of Users

Users are supported through each of these levels by the dynamic content of the page that they see when they login to the system. When new users first register with the system, they will be presented with unambiguous links to basic summary queries, along with an explanation of these queries and their results. After completing this task, users will be given a set of links to some of the most popular instructional query sets as well as an opportunity to browse or search all cached queries. Users could also be given a link to a query builder tool. When viewing the results of a cached query, users are given the opportunity to modify the query. Interested users are presented with an interface similar to that of the new users, except that the system suggests new query sets at each login.

As soon as users create their first original query (either by modifying a query or by using the query builder), they become active collaborators and a new feature is added to their initial page upon login – namely a list of recent queries that they have written. They can see whether their queries are cached and, if so, they can also view how many other users have accessed the cached results.

Active collaborators are presented with the option of building a query set for export. As soon as users export their first query set, they become expert collaborators and another new feature is added to their initial page upon login – namely a list of their exported query sets. They can then view the access statistics and see feedback given by other users regarding the query sets.

4. Query Caching System

In our discussion of users and query sets, we have mentioned query caching at a high level. In this section, we describe the design of our query caching system in detail.

At the highest level, the query caching system stores the results of an SQL query in a file so that they can be returned without re-running the query. For large data sets, like the ones we are dealing with, returning a cached result can save hours or days of intense computation. However, no single system has the space to cache the results of all queries. So we must choose which results to cache.

To decide whether to cache the result of a query, our query caching system considers the following five primary factors:

(1) **The user's judgment of the query (rating,R)**
In our collaborative data exploration system, we expect that many queries run on the system will represent experiments rather than polished queries. As users experiment with writing queries, they will often realize when they see the result that the query was not quite right. In this case, caching the result would not be helpful. Therefore, the most important factor in caching a query is to ask users whether this query answered their question or whether modifications are required. In addition, we ask users whether they think the query results should be cached for other users to be able to view. Even if the query did answer their question correctly, it might not be a result that they expect to be of general interest.

(2) The time taken to produce the result (time, T)
The system also considers the running time of the query when deciding whether to cache the results. The faster a query runs, the less important it is to save the results. We do not save any results that are generated under a parameterizable threshold. For example, in our prototype system, we do not cache the results of any query that takes under 3 seconds to run as we consider that a reasonable response time.

(3) The size of the result (size, S)
The system also considers the size of the result. The larger the result, the less efficient it is for the system to store them. The administrator allocates a fixed amount of space for query caching. The system cannot store any result that exceeds this fixed amount. In addition, the system will not save any results

that are a parameterizable fraction of this fixed space. For example, in the prototype system, we do not cache any result that occupies more than 10% of the query cache.

(4) The value of keeping an existing cached query as opposed to replacing it with the new or current one (hits, H)
An initial caching value can be computed by dividing the running time by the size of the result. The units on this result are sec/KB. We also multiply by a user-supplied "interest rating." For existing results, the user-supplied "interest rating" is replaced by a rating that reflects how often users have accessed the query results and how helpful they rated those results to be.

(5)Whether the result is currently accurate (changed percentage, C)
Then, once a query result is saved based on the above criteria, the system also saves the last modification time for each table accessed during the query. These times can be consulted to determine if there is any reason to re-run the query to update the result. Even if the underlying table has changed, a result can still be useful. For example, the average value over terabytes of data is unlikely to be changed by the addition of a few new rows. Administrators of the system can designate certain queries, like the basic summary query set, for example, to be re-run when the database changes.

We combine these factors in the following formula: $\frac{R * T * H}{S * C}$

where R is rating, T is time, H is hits, S is size and C is the changed percentage.

5. Use with Any Relational Database

A key design goal of our system is to be able to explore any type of data contained in a set of tables in a relational database. In other words, we did not want to require that our collaborative exploration system be modified based on the type of data being explored.

We accomplish this by dividing the tables in our database into two basic classes – infrastructure tables and domain tables. Infrastructure tables are those tables specified by our system to support the collaborative exploration functionality. Domain tables, on the other hand, contain the actual data to be explored. There can be any number of domain tables of any size and structure depending on the data set.

Some infrastructure tables are populated automatically as the system is used. For example, these include tables of users, tables of cached queries, and tables of query sets. Other infrastructure tables are populated by a system administrator that is familiar with the basic structure of the domain tables. For example, one infrastructure table lists all the domain tables and elements of each row in those tables.

Administrators also have the opportunity to set certain parameters such as the amount of space dedicated to the query cache, the maximum percentage of the query cache for any one result, and the minimum time for a cached query.

In the rest of this section, we describe some of the key infrastructure tables.

5.1 Administrative Infrastructure Tables

The infrastructure tables that are populated by an administrator at installation are the ProjectInfo and ,DomainElements. The ProjectInfo table contains parameters that allow administrators to configure project-specific details like the size of the query cache, the maximum size of any one cached query result and even the filename of logo picture to customize the look of the system's web pages. The DomainElements table is used to allow administrators to provide descriptive overviews of the data being explored. They can enter textual descriptions of each database, each table within a database and each field within a table.

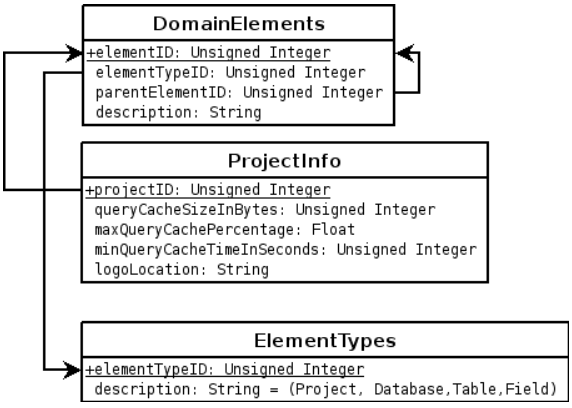

Figure 1: Relationships between the administrative infrastructure tables

The DomainElements table also captures the hierarchical relationship of projects, databases, tables and field. For example, each domain database in the project has the projectId of that project as its parentElementId. Similarly, each table within a database has the elementId of that database as its parentElementId. In this way, our system can use these tables to provide a project explorer functionality. New users can quickly gain an overview of all the domain specific data by viewing a project with its administrator provided description, then each of its databases with their descriptions, etc. later, we will discuss how this browser is integrated with the cached results of automated query sets to provide a quick sense of the type of data stored in each field of each table.

Care is taken to minimize the administrative set up tasks. All tasks could be automated if necessary. For example, the list of domain specific tables and their structures could be populated when the domain tables are created. The administrative set-up tasks simply give administrators a chance to enter textual

descriptions of each table and row element as well as to group domain specific tables into named projects. Similarly, administrators need not change the system defaults for parameters such as the maximum percentage of the query cache for any one result.

Since these tables are populated by an administrator, or someone who is familiar with the domain-specific data, who may not know SQL, we will provide web-based entry forms to allow easy population of these infrastructure tables

Note that the database containing the infrastructure tables is stored separate from the domain-specific databases. The logical link is made by populating these infrastructure tables with the information that describes the domain-specific data sets.

5.2 Query Infrastructure Tables

Another set of tables are used to track the queries that are executed in the system and the cached results of those queries if applicable. The tables used to store all of this information are the DistinctQueries, ExecutedQueries, CachedQueries, TableModifications, and CacheQueryTableModifications tables. Figure 2 shows the relationships between these tables.

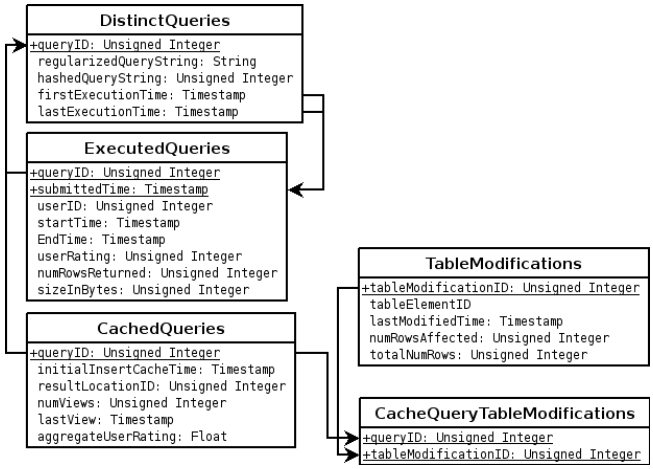

Figure 2: Relationships between the Query Infrastructure Tables

When a query is executed, the system first determines if this query has been run previously by searching for equivalent queries in the DistinctQueries tables for matching queries. If we find a match, we then look in the cached queries table to see if the result is already cached. Even if there is no direct match, we examine the cached queries to see if their results either contain the answer to the current query or can be used to return the answer to the current query more efficiently. We currently use a set of simple tests for query equivalence, but future versions could take advantage of more advanced techniques for determining query

equivalence and containment as described in the literature [AH01] [GL01] [SC05].

In all cases, an entry is started in the ExecutedQuery table listing the distinctQueryId for this query, the time the query was submitted and the userId for the user issuing the query. When the query actually begins to execute, the startTime will be update and when the query completes the endTime will be updated as well as information about the size of the result.

When a query completes, we decided whether to cache the result as described in Section 4. If the decision is made to cache the query, then the results will be stored in a location implied by the resultLocationID. Currently, we store it in a file with a name based on this resultLocationID, but we could also have another table mapping resultLocationID to other location information.

Also when a result is cached, we note the last modify time for each table consulted in the query. Each time a table is modified (i.e rows added, deleted or updated), we add a entry in the TableModification table with the time of the modification and the number of row affected by the change. The last modify time of a table is simply the latest time of all rows referring to that table. For each table consulted by a query, a row is added to the CacheQueryTableModification linking the queryID to the last modified row of the TableModification table.

We assumed that all users of our system are not updating the data directly, They are characterizing data that may be changing as the result of periodic updates from another source.

Rows in the TableModification table are used to determine whether a cached result is still accurate and if not to compute its change percentage, C, as described in Section 4. Rows in the CacheQueryTableModification table can be deleted when the cached query they refer to is updated or removed from the cache. Similarly, rows in the TableModification table can be removed when no rows in the CacheQueryTableModification table refer to them.

Note that we do not cache multiple copies of the results from any one distinct query. We keep only the most up-to-date version. We have considered that it might be nice to see how the result of a query changes over time as a table is modified but this is currently not accommodated in the design.

Each time a query is executed – regardless of whether the result is generated directly or a cached result is returned – the user is asked to rate the usefulness of the query. This feeds is used to decide whether to cache a result in the first place and also feeds into an aggregateUserRating of each cached query.

We can use these tables to return a variety of important information to users. For example, ExecutedQueries is used to generate lists of the most recent queries and the most popular queries. The ExecutedQueries table can also be used to generate lists of queries that are currently in progress and lists of queries that are queued for execution. The CachedQueries table is used to allow users to browse cached results. Even if the results of a query are no longer cached, the ExecutedQueries table can be used to report how long it has taken to run in the past and how large the result was.

For each user, we can generate a list of all the queries they have executed and for each one we can determine if the results are currently cached and if so if the cached results are still acute.

Rows in the ExecutedQueries table can be truncated by time to free up space. For example, the system may choose to store only the last month of executed queries. Similarly, rows in the CachedQueries table can be removed when the result is evicted from the cache. Rows in the distinct query table can also be removed as long as they are no longer pointed to by any other tables (CachedQueries, ExecutedQueries, or QuerySets).

5.3 User and Query Set Infrastructure Tables

Another set of tables are used to track users and the query sets they create. The tables used to store users and their interactions with query sets are in the Users, UserLevels, QuerySets, QuerySetElements, and QuerySetPermissions tables. Figure 3 shows the relationships between these tables

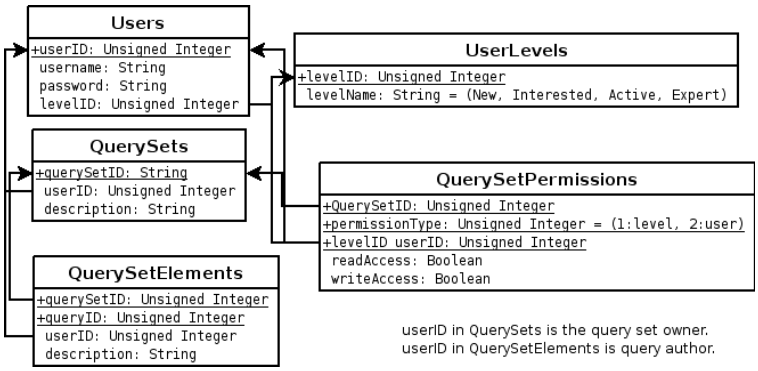

Figure 3: Relationships between the User and Query Set Infrastructure Tables

The Users table contains a unique userID, a username, a password or other authentication information, and a levelID which corresponds to the levels defined in UserLevels: New, Interested, Active, and Expert. The system could also store any number of other user attributes like email address, physical address, when they became a user, etc.

The QuerySets table contains a unique querySetID, a userID of the query set owner, and a description of the query set. The actual queried contained the query set are stored in the QuerySetElements table. This table contains the id of the parent query set, the queryID, and the userID of the query author, and a description of the query as it pertains to this query set.

Finally, QuerySetPermissions is used to control who has access to other queries. Read and/or write access can be granted to other users either by specifying them individually or by specifying a certain minimum level of users. Users with write access may add queries to the query set in which case they would be listed as the query author but not the owner of the query set. Users with read

access may see all the queries in the query set. Only the query set owner may delete the query set.

We can use these tables to return a variety of important information to users. For example, each user can see every query set they own and its contents. They can also see queries they have authored in other query sets. For all of these query sets and queries, they can see what results are still cached and an aggregate user rating of these queries.

5.4 Integrating the Infrastructure Tables Into Our Implementation

There are many open source database administration tools, particularly for MySQL. Before beginning the implementation of our prototype, we reviewed many of these systems. However, none of these tools provided the query caching and support for collaboration we wanted. The closest tool to ours is phpMyAdmin. [PHA05]. It does provide functionality like a database browser and query builder. However, they were insufficient for our purposes. In this section, we describe how we integrated our infrastructure tables into the implementation of these pieces. We also describe some functionality that we have integrated from existing open source tools.

We implemented a new database explorer that is integrated both with the DomainElement table, the QuerySet table and the CachedQuery table. It uses the DomainElement table to drive the hierarchy of elements displayed as user traverses the domain data in a project. We also use it to display the descriptions entered by the administrator.

We use the QuerySet table to hold the automatically generated query sets that summarize the number of rows in each table and the range of values for each field. These automatic query sets belong to a special user, the system user, denoted by a special userID. These queries are readable by all other users but are not modifiable.

Finally, we use the CachedQueries table to store the results of these automated queries. Users can then browse these results to get an general feel for the data without issuing long running queries to determine the average value in a field or a histogram of the most common values in a field. If a domain table is modified, the system can either rerun all automated queried that reference that table or it can wait to run them on a regular schedule or in idle time if the cached results are still relatively accurate.

Another piece of functionality that exists in other systems is the query builder. We modified our query builder in several ways. First, we found many query builders overly complicated. We decided to build a very easy-to-user builder that supported only simple queries. We decided that if a user need help building more complicated queries that we would help them in another way. Specifically, we provide them with previously issued queries that they can modify in the form of polished query sets.

Implementing our own query builder also allows us to integrate support for finding equivalent queries through matches in our DistinctQueries, searching

through the CachedQueries to find cached results and querying the user for their rating of the usefulness of the results they receive.

Finally, we needed to provide some communication links for the users of our system. For this we decided to use the popular phpBB forums. These forums allow for a lot of flexibility and customizability, in terms of administration, general use and look-and-feel. The phpBB forums are also mature and stable software. The choice of this forum software was based on availability and easy integration. Other forms of communication would also be reasonable to integrate if desired.

6 Experience With Prototype

We downloaded the publicly available Border Gateway Protocol (BGP) data that was collected by the Internet Performance Measurement and Analysis (IPMA) project [IPM01]. In the following sections, we describe where the data comes from, what the data means, and how we transformed the data into something that could be stored efficiently in a relational database. Further, we discuss the BGP domain-specific tables and how they interact with the collaborative data explorer. Finally, we show how some of the different classes of users interact with this data.

6.1 Border Gateway Protocol Data

The IPMA project collected the BGP messages sent between peer routers from several Internet Exchange Points (IXP) located across the United States from January of 1999 to December of 2002. We downloaded the data from that project and have imported it into SQL databases. The raw gzipped BGP data is about 16 gigabytes of data. Once it has been extracted, transformed, and imported into SQL databases, its size is about 170 gigabytes. Information about the exchange points is listed in Table 4.1. The Ameritech Advanced Data Services (AADS) exchange point is located in Chicago, IL. MCI's Metropolitan Area Exchanges (MAE) East and West are located in Washington, D.C. and Silicon Valley, CA respectively. The Palo Alto Internet Exchange (PAIX) is located in Palo Alto, CA. The SBC's Pacific Bell (PB) Internet exchange is located in San Francisco Bay area, CA. The largest amount of raw data was collected from MAE-East followed by MAE-West, PB, AADS, and PAIX.

6.2 Transforming the Data

Note that even if data is not originally stored in a relational database, it is often possible to convert that data into a format suitable to be imported into a relational database. We did this with some Border Gateway Protocol (BGP) data for our prototype.

The raw data that we downloaded is in a binary format that had to be parsed by a tool called route_btoa, which is part of the Muti-threaded Routing Toolkit

(MRT) developed by the Merit project at the University of Michigan. [MRT04] [MER04]. An example of the human readable format produced by route_btoa can be seen in Figure 4.1(a). This particular message, in the figure below, was collected from the MAE-East exchange point and was a message sent on October 5th 1998 at 11:01pm. It was sent from a peer router with the IP address 144.228.107.1 in the Autonomous System numbered 1239 to another peer router with the IP address 192.41.177.169 in the Autonomous System numbered 2885. The message is a withdrawal message since it withdraws Internet prefixes that are no longer available. Specifically, these prefixes are 192.195.250.0/24, 193.40.0.0/24, 193.40.1.0/24, 193.40.2.0/24, 193.40.5.0/24 … etc as seen in the figure. The format of the output is easy to read for a human, but not fast to parse by a computer program. Fortunately, the route_btoa tool supports a –m flag that outputs the data into a machine readable format shown in Figure 4.1(b). In this format it is easy for a computer program to parse into the different pieces. It is also easier to break apart the various prefixes that are withdrawn. The format is:

Protocol | Time | Type | PeerIP | PeerAS | Prefix | <update dependent info>

Protocol is the language version the routers will use, this is typically a version of BGP. Time is the unix timestamp (in seconds since Jan 1, 1970). The type is usually either A or W for "announcement" and "withdrawal" messages, respectively. Another less common type is S for "state" messages. The peer IP and peer AS are for the origin of the message. The prefix is the beginning of the route being announced or withdrawn. Finally, the update dependent information only exists if the message is an "announcement" message.

Once we understood the data that was output by the command route_btoa –m, we wrote a script to parse the data and import it into an SQL database. The tables used to store it are described in section 4.3.

6.3 Loaded DB stats

Table 1 shows the size of the raw data that was collected and the size it takes in the database.

Table 1: Raw data sizes and size in database

Location	Raw (MB)	DB sizes(GB)
Mae-East	3072	115
Mae-West	1126.4	33
Pb	519	18
Aads	345	11
Paix	183	1.3

Domain Size	Characterize Size	Time to cache
1.3GB	5.5KB	15.9 hours

For one of the databases in our domain-specific data (BGP Data) we were able to characterize 1.3 GB of data and store the characterization results in 5.5KB of cache files. This give us histograms, and minimum and maximum values as shown in the screenshot in the appendix (Section 12).

7. Related Work

In this chapter we discuss related work. There are several areas of work that related to collaborative exploration and characterization of large data sets. These include data exploration, collaborative systems, recommender systems, information retrieval, and community and collaborative filtering systems. In the following sections, we discuss the most relevant, collaborative information retrieval and data exploration, in more detail.

Twidale and Nichols designed an application, Adriane, to support collaborative information retrieval [TN98]. Their project focuses on visualization of the search process and collaborative browsing. Most of their work has dealt with using library resources. They also realized that this activity is typically a solitary one and that these systems fail to support the process of learning how to locate information and the sharing of information. Chau et. al. [CZ+02] developed a multi-agent collaborative web mining system. Their system was designed for web searches in which users can annotate their sessions for use by future users. They found that for the application of web searches, the search productivity of users only increased when there were enough previous sessions for them to see.

A number of online sites allow data exploration including online libraries and encyclopedias, e-commerce and multimedia sites, and search engines and portals. Database content and use can be seen across all disciplines, from geological sampling to business data mining. Many of the web interfaces to these databases allow some collaboration between end users. A prime example of user-database interaction is when customers search for products or information on sites such as Amazon.com, eBay.com, Google.com and IMDb.com. However, these sites do not allow users a high degree of interaction with other users that are seeking or researching a specific topic or product. This type of interaction is characteristic of recommender systems.

Data exploration in general has been aided by advances in parallel and distributed computing technologies. In particular, grid computing has been embraced as the platform for the creation, processing and management of petabytes of high-energy physics data [LF04]. In [BAK99], S. Bakin points out that the shear size and complexity of large data sets can be the limiting factor that does not allow for practical analysis of the data. For large and complex data sets, it can be difficult to find useful information in a timely manner. Szalay et al. discussed the Sloan Digital Sky Survey (SDSS) in which they seek to mine multi-terabyte astronomy data. Their system is collaborative in the sense that data is collected across the world and the calculation and results are done in a distributed manner.

9. Conclusion

We have presented a generic system for collaborative exploration of large SQL databases. Our system allows system resource to be used more efficiently. Specifically, the query cache avoids long running queries by caching the results of previously executed queries according to a formula that takes into account the user's own rating of the query, the running time of the query and size of the result. It compares the value of caching new results against the value of existing results which also take into account the number of times the cached results have been viewed, the average user rating of the results and how accurate the cached result is.

Our system also allows users to learn from each others queries. It helps users transition from new to expert user. It provides a series of automatically cached queries that allow new users to get a quick overview of the type of data housed in the database. It also allows new users to learn from expert users through polished query sets. This also benefits expert users by reducing the system resources consumed by new user queries that are more likely to be incorrect without assistance. Finally, it enables users to learn by modifying the queries of others. This allows users to focus on the novel aspects of their queries rather than the logistical details.

The contributions of this project are the design of a collaborative system in which users can work together to explore and characterize an SQL database, an interface for cached queries and their results, a user-driven query cache, and a prototype system that begins to test the usefulness of such a system. Our system is not customized to the data being explored and can be used with any set of domain specific tables.

We note that the benefits of collaboration are multi-faceted. Collaboration can help free up system resources by reducing the number of duplicate and incorrect queries. It helps users free up "mental" resources by also allowing users to focus on the novel aspects of their query rather the logistical details. For both these reasons, our collaborative data exploration system can help users understand large, complex data sets more effectively.

Bibliography

[AH01] A. Halevy. Answering queries using views: A survey. The VLDB Journal — The International Journal on Very Large Data Bases , Volume 10 , Issue 4, pp. 270 - 294 , December 2001.

[BAK99] Bakin, S. Adaptive regression and model selection in data mining problems. PhD thesis. Australian National University. 1999.

[CZ+02] Chau, M. and D. Zeng, H. Chen, M. Huang, D. Hendriawan. Design and evaluation of a multi-agent collaborative web mining system. Decision Support Systems 2002.

[GL01] J. Goldstein and P. Larson. Optimizing Queries using Materialized Views: A Practical, Scalable Solution. ACM SIGMOD 2001, May 2001.

[IPM01] University of Michigan Department of Electrical Engineering and Computer Science. Merit Network. Internet Performance Measurement and Analysis (IPMA) Project. May 2003.
http://www.merit.edu/~ipma/.

[LF04] Liu, David T. and Franklin, Michael J. GridDB: a data centric overlay for scientific grids. VLDB 2004.

[MER04] Merit Network Inc. May 2004.
http://www.merit.edu/.

[PHPA] PhpMyAdmin - MySQL Database Administration Tool. February 2005.
http://www.phpmyadmin.net/home_page/.

[SC05] S. Cohen. Containment of Aggregate Queries. SIGMOD Record, Vol. 34, No. 1, March 2005.

[SK+00] Szalay, A. and P. Z. Kunszt, A. Thakar, J. Gray, and D. R. Slut. Designing and mining multi-terabyte astronomy archives: the sloan digital sky survey. ACM SIGMOD 2000.

[TN98] Twidale, M. B. and D. Nichols. Designing interfaces to support collaboration in information retrieval. Interacting with Computers 1998.

[TR04] Thor, Andreas and Erhard Rahm. AWESOME – a data warehouse-based system for adaptive website recommendations. VLDB 2004.

Appendix (Screen shot)