ON

Relevant Searching in Structured Database

By

Rohan Agarwal - 1PI13CS124
Srinivas Akhil Mallela - 1PI13CS164
Anirudh Agarwal - 1PI13CS199

Guide

Prof. Channa Bankapur

PESIT-CSE
Bangalore

January 2016 – May 2016

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
PES INSTITUTE OF TECHNOLOGY
(an autonomous institute under VTU)
100 FEET RING ROAD, BANASHANKARI III STAGE,
BANGALORE-560085

Certificate

Certified that the sixth semester mini project work under the topic “Relevancy Search in Structured Database (Information Retrieval)” is a bonafide work carried out by

Rohan Agarwal	1PI13CS124
Srinivas Akhil Mallela	1PI13CS164
Anirudh Agarwal	1PI13CS199

in partial fulfillment of 2 credit course for the award of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum during the academic semester January 2016 – May 2016.

Signature of the Guide
(Prof. Channa Bankapur)

Signature of the HOD
(Prof. Nitin V. Pujari)

External Viva :

Names -

Date -

ACKNOWLEDGEMENT

We take this opportunity to express my profound gratitude and deep regards to our guide Prof. Channa Bankapur, for his exemplary guidance, monitoring and constant encouragement throughout the course of the project. The blessing, help and guidance given by him from time to time has been very productive to the outcomes of the project work undertook by us.

We would like to express our heartfelt thanks to our chairman Dr. M R Doreswamy, our CEO Prof. D Jawahar and our Principal Dr. K S Shridhar for providing us with a learning environment to carry out our academic projects.

We feel privileged to offer our sincere thanks to Mr. Nitin V Pujari (HOD) for expressing his confidence in us and giving his liberal encouragement not only during the project, but also throughout the year.

INDEX

Abstract

The project is about developing a software that allows users to provide simple statements (of language expression) rather than SQL querying constructs to query a database. Thus on the user interface front a general search box allows users to enter their query statement. The system does a search on the query terms as soon as they are entered by the user thus providing an overview of the most relevant results that match the query received thus far. The ranking of the query results is based on a number of factors which have been discussed at length in this report.

1. Introduction

Smart Database Retriever is a general database search engine for people without any knowledge of databases and querying norms. The results are returned in the order of relevancy which is unlike any other contemporary database management softwares handling structured data. The user needs to attach database as a parsable file, like .csv, .xls etc, and the system automatically starts operating on it.

A general search engine interface is where a user can enter query and the results will be prompted for each term. This client interface communicates with a running server that indexes user's database for the purpose of relevancy based fast searches. The indexing is done using an amalgam of Information Retrieval techniques. The results returned are based on the ranking functions which have been profiled and the best blend of these ranking functions is used for ranking the results so that none of the ranking functions are overpowering. The top-k results, that is, the k tuples with the highest scores after ranking, from the database are returned back to the client interface based on the user supplied query.

This software can be used to handle the following two scenarios that are not gracefully handled by a SQL system:

Empty answers: When the query is too selective, the answer may be empty. In that case, it is desirable to have a ranked list of approximately matching tuples.

Many answers: When the query is not too selective, too many tuples may be in the answer. In such a case, it will be desirable to have the option of ordering the matches automatically that ranks more "globally important" results higher and returning only the best matches.

2. Problem Definition

A general application program interface for providing relevant and quick search in structured database.

- Automated ranking of relevant results from a database acting on a user-supplied query.
- Returning the Top-K results by using various Information Retrieval techniques and ranking functions to identify the most significant aspects of the data to the query.
- Handle “Many answers” and “Empty answers” problems when it comes to the results of a Query.

3. Literature Survey

Literature survey involved a thorough analysis of 'Automated Ranking of Database Query Results', a paper by Sanjay Agrawal, Surajit Chaudhuri, Gautam Das (Microsoft Research) and Aristides Gionis (Computer Science Dept, Stanford University).

The major inspiration behind the project was a deep study of above mentioned paper. Other sources involve some other papers discussing 'relevancy tuple search in structured and unstructured data', a popular paradigm in Information retrieval. Sources namely include 'Keyword searching and browsing system over relational database' from University of Computer Studies Yangon, 'In-RDBMS inverted indexes' from University of Wisconsin Madison, 'A Ranking algorithm based on contents and non-key attributes for object level keyword search over relational databases' from Harbin Institute of Technology.

Wikipedia pages for Indexing, Query-Frequency, Inverse Document Frequency and c-plusplus official site, for the purpose of implementation, are also among our sources for literature survey. Apart from above mentioned material, we got insightful guidance and direction from our guide in this project.

4. Project Requirement Definition

The project requirements were thought out meticulously as they are critical for development of all the other downstream software components of the projects.

Requirements elicitation or gathering the requirements for a project such as ours was in large part done through the problem definition provided to us. However a significant number of requirements and many important ones were understood by extensive use of similar tools which are already in the market by the team members and fellow colleagues. Thus the problem definition was modified and the requirements of the project were dynamic as new features and specifications were added through the course of development. Still the most basic requirements were documented and prioritized. These and other newly added requirements were realized by broadly segregating them into two domains - Functional Requirements and Nonfunctional Requirements.

Functional requirements primarily document the functionalities and services that the software is expected to provided to the end user. They describe the general behavior of the system in particular situations with given inputs.
The functional requirements satisfied by the said software system are -

- **Provision of a generic search facility to the user.** The generic property of the search enables a user to enter search terms without specifying any columns, tables or any other metadata about the database or the search query.

- **Ability to associate any database or CSV format file to the software.** The software provides a clean interface to the client to associate any file (which is the source of the data) to the software on which the searches are to be made.

- **Relevancy based search results on the given query terms.** This is the primary requirement of the system. The results of a query (which are the tuples in the database) are ranked based on their relevance to the query and on history of previous related searches by other users of the system.
- **User Interface.** A web application is provided to satisfy the requirement of a user interface through which the user can enter the query. The first page of the interface must provide the user a prompt based on the query that has been entered thus far. When the user clicks on a button (DB Search), he must be directed to another page where all the tuples are displayed in a

ranked order. Beneath every tuple should be metadata about the tuple - Tuple Identifier and Relevancy score of the tuple. This page also must include certain statistical information such as the Response Time of the system, the number of tuples returned and the number of tuples in the database that were considered while processing the search.

The Non Functional requirements satisfied are -

- **Quick response times.** The preliminary aim during development has been to reduce the Runtime of the system as much as possible, enhancing the user experience significantly. This is the period after entering the query and before the results start to appear, also referred to as the Response Time of the system. The choice of the programming language, the algorithms and the data structures used in the implementation all have been selected meticulously with an overall effect of optimal time complexity of the system leading to reduced response times.

- **Ability to handle large volumes of data.** The software has been developed such that files of large sizes can be attached to it for search operations. The algorithms and their implementation is irrespective of the volume of data that is being sifted through. Although the size of the data files would have a proportionate effect on the response time of the system.

- **Interoperability of the software system.** The software can be executed on any computer system and source code of the program can be compiled on any system with a C++ compiler. Using the user interface provided with the software package, the user can query a database while the back end could be executed on any other system independent of the client's system.

5. System Requirements Definition

Some of the system software requirements are:

- Unistd.h library to provide access to the POSIX Operating System API.
- Algorithm c++ library to support a collection of functions especially designed to be used on ranges of elements.
- iostream,fstream, sstream libraries to support file and terminal operations and fcntl.h for file control operations.
- If the software has to compiled on a device, the GNU C++ compiler and related packages have to be installed.
- Javascript has to be enabled on the user's browser(Client Side). One of the primary purposes is to make Ajax calls and retrieve critical information from the server.
- CGI : Common Gateway Protocol is used on the server to interface with executable C++ programs installed on the server that generate web pages dynamically.
- Named pipes are used for Inter Process Communication between the CGI scripts and the server process. The server has to support the concept of named pipes.

Hardware requirements-

- Operating System (Linux, Windows & MAC OS)
- Processor Speed (1.5 Ghz or above)
- RAM (512 MB or Above)
- Hard Disk Space (80 GB or more)

6. System Design

System design basically involves a decoupled approach of normalizing interfaces and implementation at each and every sub-component level, thus making them independent. Starting from highest level, the User Interface comprises of a CGI script entertaining ajax GET requests from an interface screen. The script acts as a client process while a server process is continuously active in the background. Both client and server processes interact via named pipes, blocking and waiting for each other to fill the pipe in a synchronized manner.

Coming to server, the primary data structure behind most of the indexes is 'hashmap'. The algorithm used is 'pattern matching' that basically hashes the key to fetch information stored as the value. The front part of server uses index_data interface for the creation of column and cell indexes and index_workload interface for workload index creation.

The main server has been divided into sub-systems namely 'index_data' and 'index_workload' files for the purpose of index creation based on user's database, 'idf_ranking' for taking in user query, computing relevancy score for various tuples using idf_ranking algorithm and filling a fixed user defined k-box set, 'qfidf_ranking' for the purpose of breaking relevancy ties in obtained tuples with the aid of user query trends in workload, thus calculating aggregate score using 'query frequency' algorithm and a 'client' component to control these components.

Each sub-component, except 'client', has an interface and an implementation file distribution. Within each sub-component, the main service is subdivided and organized as routines complying with modular programming paradigm.
A common header file is shared across the system for the purpose of dynamic linking of class libraries, thus making the system compact. Only Interface files are included in other components, where they are needed, and implementation files are separately defined and mapped using scope resolution. The abstraction is public as different components require access to both data and members methods of other sub-components. Even modules call other modules to use their functionalities.

Apart from general design, the system also follows language specific design, by including basic functionalities such as copy and move assignment and

constructor modules which are prerequisites for a user-defined type in c-plusplus.

7. Pseudo code

The project is divided several subcomponents doing index creation, query processing, relevancy score calculation and tuple fetching.

IDF_RANKING :

- Can define k_box size in define.
- Calculates idf_score of tuples having user queried words.

Member Variables :-

- *Index index_obj* -> Index Object to use cell_index for IDF.
- *map<unsigned int,double> idf_calc* -> Stores tupleno VS their score in the final k_box.
- *unsigned int tot_tups* -> Stores total no of tuples in database for IDF calculation.
- *vector<unsigned int> k_box* -> Stores tuple no belonging to k_box.
- *map<string,double> k_box_tuples* -> Stores fetched tuples from data file VS their IDF score.
- *string data_file* -> Stores name of data file to fetch tuple from.

Member Methods :

- *hash_terms()* :-

-Takes in user query , splits in words and calculates IDF = log(total tuples / freq of that term) for each and every word one by one. Also sends tuple collection and IDF score of each word in query to `calc_idf`.

- *calc_idf() :-*

-Fills idf_calc by finding union of tuplenos in all the words in query and accumulating idf_score of words into respective tuple no. At the end of this , idf_calc would have all tuples having any combination of user query words occurring in them, along with tuple idf_scores.

- *tokenize*() :-

- Sent a vector reference and a line. At the end stores words in vector from line passed splitted by spaces.

- *min_box_element() :-*

- Returns tupleno with minimum idf_score in the filled k_box at a certain point of time.

- *create_box() :-*

- Fills k_box from idf_calc with n tuples with highest idf_scores. Box is filled and then it is checked if coming tuple score is greater than minimum score in k_box. If no, iteration continues, if yes, the minimum element is replaced with this entry and new minimum of k_box is found.

- *fetch_tuples():-*

- Calls `hash_terms` and `create_box`. After obtaining k_box, it uses tuple_index of `Index` class to fetch complete tuples from data file. On getting it , tuples along with their respective scores are stored in map - `k_box_tuples`.

- *disp_info :-*

  - Displays both `idf_calc` i.e tupleno Vs IDF_score and `k_box_tuples` i.e tuples VS IDF_score.

QFIDF_RANKING :

- Calculates qf score of terms in k_box tuples, based on workload, to break ties of relevancy score as new score = idf + qf.

Member Variables :-

- *map<string,double> k_box_tuples* -> Stores fetched tuples from data file VS their IDF score.
- *map<double,string> final_result* ->To store tie-broken tuples with score as key and tuple as value so as to print in descending order of score relevancy.
- *workload window_obj* -> workload object so as to use workload column_cum_cell index to calculate QF score of each term of tuples in k_box to break ties.
- *map<int,unsigned int> max_freq*->To store RQFmax of all columns in workload's column_cell_index to use for QF = RQF/RQFmax.

Member Methods :

- *start_qfidf() :-*

-Iterates through each tuple in `k_box_tuple` and passes each tuple, one by one, to `tup_calc()` which gives QF score for passed tuple.This new score is added to old IDF score and then the whole `k_box_tuple` is copied to `final_result` with score as key.

- *calc_max_fq() :-*

-Iterates through workload's column_cell_index and fills `max_freq` with RQFmax for each column. Also can print this RQFmax list.

- *tokenize*() :-

- Sent a vector reference and a line and a delimiter (,). At the end stores words in vector from line passed splitted by the passes delimiter. Used to split tuples of `k_box_tuple` into separate columns.

- *tup_calc()* :-

- Receives each tuple of box one by one, breaks each tuple into columns, iterates through them and sends each column to `index_n_score`, receives QF score for each column and accumulates them as cumulative score for each tuple to be sent back.

- *index_n_score() :-*

- Receives each column value and column no.,uses column no. to index from workload's column_cell_index and find RQF of column value that was passed. If column value has multiple words separated by spaces , they are broken into simpler words before indexing.

- *disp_results() :-*

- Displays `k_box_tuple` which has tuple VS (qf+idf) score.Also it displays the `final_result` having the tie-broken tuples to be presented to users as the result.The score is the key, tuple as value and since map automatically arranges in ascending order of key,function uses a reverse iterator to print the result in reverse, having most relevant result at the top.

INDEX_WORKLOAD :

Default Constructor and a Parameterized constructor of workload are defined here.

The Parameterized Constructor takes the file name of the workload and the Index object instantiated. The index object will have the column index, tuple index, cell index and col_cell index already created.

Member Variables:-

- outer_map Workload_col_index:
- typedef map<string,vector<unsigned int>> inner_map;
- typedef map<int,inner_map> outer_map;
  Stores column number as key and stores the strings and query id's in inner map. A String can have multiple query Id's.

Member Methods:-

- get_data_file()

Return type: Void

Basic Functionality: Opens the workload file passed and delimits each query on "," using the getline function (Also removes double quotes for each element as it is a necessity because of the format of our CSV File) to get that specific element and passes each element to identify_column_insert_map function along with the workload_tid which increments by one for each each query.

As the function name identify_column_insert_map suggests, it identifies the column the element belongs to. If the element is null, it is ignored.

- Identify_column_insert_map

Return type: void

Input: Inputs the element and user query ID.(I.e., workload_tid).

Basic Functionality: Splits the element one by one if it has spaces, and finds out where each and every part of the full element belongs to which column(i.e.., The column which has the highest frequency of this sub-element) and puts this into a map by using the insert_in_map function. For example,if "Highlander 4w 2c" is the element, it breaks it up into three parts i.e.., Highlander, 4w and 2c and finds out individually which column this sub-element belongs to and adds the info into a map using insert_in_map function.

- Insert_in_map:

Inserts into a map of the type outer_map the column no, String(Sub element), and tid(query id). Workload col-cell index is being created.

- Display_workload_column_index:

Displays the Workload col-cell-index using the outer_map object.

INDEX_DATA:-

- *Basic Functionality*
  - The file instantiates an object of Index class. It includes the file "header.h".
  - The Index class includes index variables which contain the mapping of the data in the database according to various methods of indexing.

- The class provides indexing methods to create the above mentioned indexes.

Member Variables :-

- Map objects -
  - Col_index - a map where, key is column numbers(int) , value is another map(inner_map). Inner_map - a map where, key is the values of that column(whole column is taken as a value, delimited only by a comma no delimiting on space character.)(strings) , value is a vector of tuple IDs(vector of ints) corresponding to the tuples where the particular value appears.
  - Col_cell_index- similar to col_index, however the key in the second map(inner_map) is each individual word in that column, delimiting using space character for every column value(i.e. If a column contains only one value eg.nissan, then nissan is the key. If the column is made up of more than one values, eg. highlander 4WD 4C, then highlander, 4WD and 4C are all separate keys.)
  - Cell_index-a map where key is every word in the database(every value of every column delimited on both spaces and comma) and value is a vector of tuples denoting where that key appeared.
  - Tup_index - a map where key is the tuple ID, value is a pair of unsigned int and int, First value of pair is the offset(position in the file where the tuple begins), second value is the length(length of that tuple's characters.)
  - Total_size - total number of tuples in the database.

Member Methods :-

- Parameterized Constructor
  - Input - filename of the database csv file
  - Retrieves tuples as a single line from the database and passes to the methods-create_column_index,create_cell_index,create_column_cell_index and create_tuple_index. While loop runs until there are any more tuples in the database.
- Copy Assignment
- Create_tuple_index

- Input - offset(position of first character of tuple in file), length(length of the tuple in the file), tid(an identifier given to the tuple).
- Output - creates the tuple index.

- Create_cell_index
  - Input - line(the entire tuple, i.e. row of the database file) , TID
  - Output - creates cell_index.
- Create_column_index
  - Input - line(entire tuple), TID
  - Output - creates column index.
- Create_column_cell_index
  - Input - line , TID
  - Output - creates col_cell_index
- Tokenize
  - Input - vector of strings v - contains the delimited output of the string(line) passed as argument to tokenize. String delim - contains the characters to be used for the delimiting.
  - Description - final is a temporary string. The given string is traversed char by char. If the chat is not found in delim, it is appended to the string final. When there are no more chars in the string, in the else part, final is added to the vector of delimited strings. If two consecutive comma chars are found, the string '_E_M_P_T_Y_' is appended to vector v.
- Insert_in_map - used by function create_col_index
  - Input - column no., the word to be inserted in the column, TID, the index in which the word has to be inserted.
  - Description - the function insert_in_map is called every time for each word taken out from the column index. Either a new column is created, a new word is added to an existing column or another entry is made for an already existing word in a column.
- Display_column_index
  - Prints the column index to a file specified in the function definition.
- Display_cell_index
- Display_tuple_index - similarly

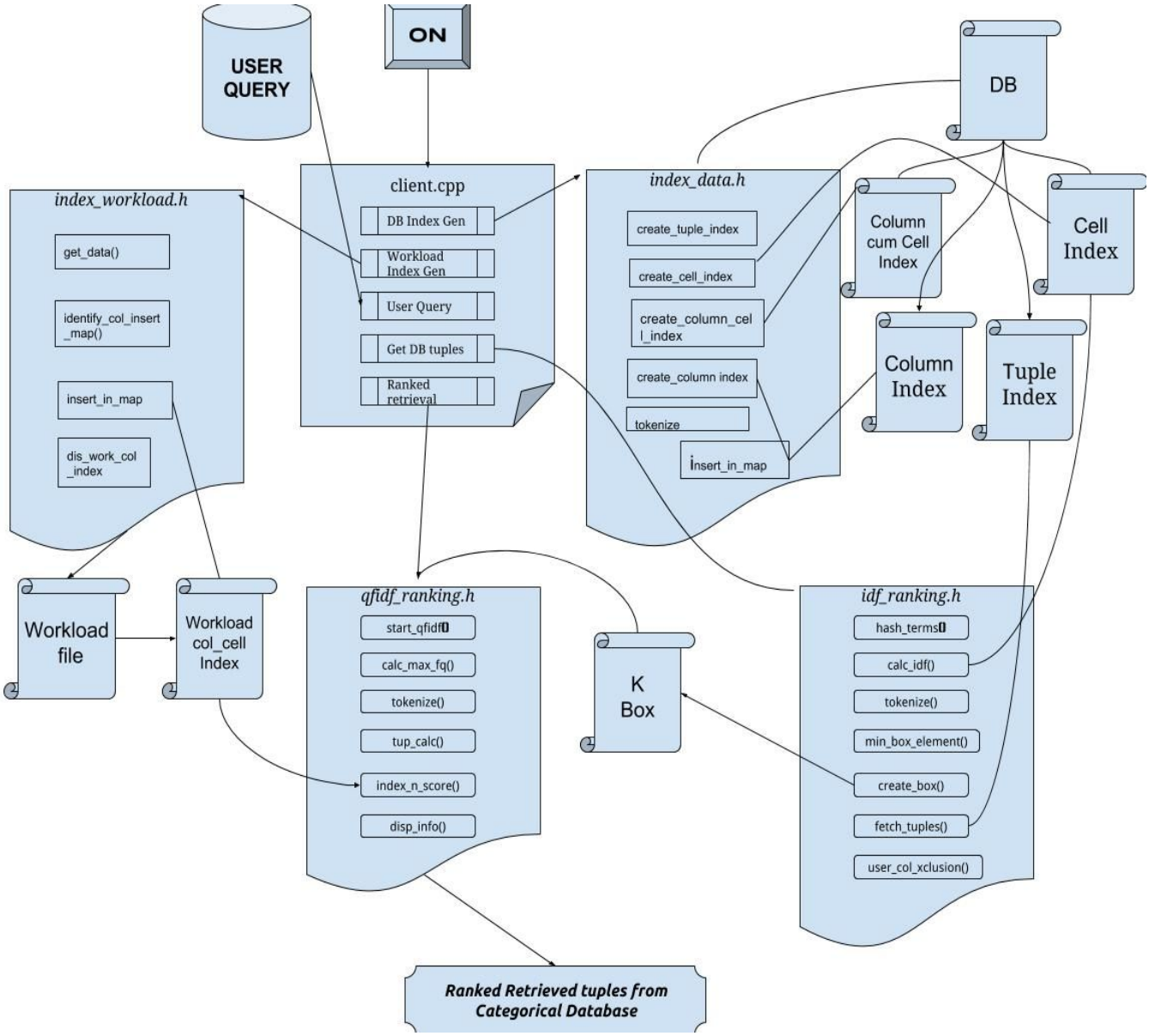

Fig 1 - Detailed Dataflow Diagram of Smart Retrieval

Fig 2(a) - Client query box interface for general search

Processing time : 0.550240 **Tuples Processed : 48707** **K-Box Size : 10**

Results :

73181,2/10/2010,2005,5,SUZUKI,RENO,EX,4D SEDAN,BLACK,AUTO,Alloy,OTHER ASIAN,COMPACT,6582,30212,GA
Tuple No : 167 ; Relevancy : 8.005665

73106,1/20/2010,2006,4,FORD,FIVE HUNDRED,SEL,4D SEDAN SEL,BLACK,AUTO,Alloy,AMERICAN,LARGE,10712,30212,GA
Tuple No : 92 ; Relevancy : 7.489792

73148,1/27/2010,2007,3,PONTIAC,G6 V6,Bas,4D SEDAN SE1,BLACK,AUTO,Alloy,AMERICAN,MEDIUM,12590,30212,GA
Tuple No : 134 ; Relevancy : 7.460007

73072,12/30/2009,2004,5,CHEVROLET,IMPALA,SS,4D SEDAN SS,BLACK,AUTO,Alloy,AMERICAN,LARGE,9754,30212,GA
Tuple No : 58 ; Relevancy : 7.311361

73076,12/30/2009,2006,3,CHEVROLET,COBALT,LS,4D SEDAN LS,BLACK,AUTO,Alloy,AMERICAN,COMPACT,7664,30212,GA
Tuple No : 62 ; Relevancy : 7.289139

73171,2/3/2010,2005,5,HYUNDAI,ELANTRA,GLS,4D SEDAN,BLACK,AUTO,Alloy,OTHER ASIAN,MEDIUM,7206,30212,GA
Tuple No : 157 ; Relevancy : 7.260567

73219,2/24/2010,2004,6,PONTIAC,GRAND PRIX,GT,4D SEDAN GT,BLACK,AUTO,Alloy,AMERICAN,LARGE,7374,30212,GA
Tuple No : 205 ; Relevancy : 7.091753

73162,2/3/2010,2005,5,SUZUKI,RENO,EX,4D SEDAN,BLACK,AUTO,NULL,OTHER ASIAN,COMPACT,6773,30212,GA
Tuple No : 148 ; Relevancy : 7.005665

73039,12/16/2009,2005,4,DODGE,NEON,SXT,4D SEDAN,BLACK,AUTO,Alloy,AMERICAN,COMPACT,6622,30212,GA
Tuple No : 25 ; Relevancy : 6.917430

73226,2/24/2010,2006,4,CHEVROLET,IMPALA,LS,4D SEDAN LS,BLACK,AUTO,Covers,AMERICAN,LARGE,10715,30212,GA
Tuple No : 212 ; Relevancy : 6.711361

Fig 2(b) - Detailed result statistics for user query

8. Results Discussion

The results or the output of the execution of the program is a list of tuples from the database. This list is a ranked list of tuples, where the ranking depends on the query terms entered by the user.

The results were tested for correctness and consistencies after several executions of the program on different inputs. We manually went through several iterations of the program, verifying the viability of the resulting tuples.

The forms of the input queries can be broadly classified as belonging to two types – many answers (query terms which are expected to produce a lot of tuples in the result) and empty answer (query terms which are expected to produce very little or no results from the data).

Many answer queries contain terms which occur frequently in the database. In this case the algorithms rank the many tuples in the order of most relevant to least relevant to the user.
Empty answer queries are ones that are too selective and thus do not match to a lot of tuples (no tuples in few cases) in the database. In such a case the software gives a ranked list of approximately matching tuples rather than giving no output at all. The software attempts to find any terms of the query in the database rather than trying to find all the terms of the query collectively in such a case.

We also timed the results for different types of queries and concluded that the average time taken by the system was always under half a second, with the exact figure lying around 0.33 seconds. On contemplating these results against those produced by the MySQL database language they were around 10 times slower than those of MySQL. This is in large part due to the provision of enhanced user convenience in searching and other extra functionalities added to the overall software.

9. Conclusion

To conclude 'Smart Database Retriever', standing true to its name, is a general lightweight database management system for any kind of database stored in the form of a parsable file. The querying is general and the results obtained are ordered on the basis of their relevancy in user attached database and query generated workload.

The project aims at exploring the concept of relevancy, that is an instrument for unstructured data searching, and applying it to leverage search results obtained in structured data. In relevancy search, the project amalgamates the best of inverse document frequency and query frequency algorithms to refine the relevancy scores, thus making computation more accurate.In terms of indexes, the system again joins column and cell indexing data structures to obtain better results. The system is implemented using c-plusplus to achieve speed and precision.

In comparison to other contemporary softwares providing same functionality, our project stands out in-terms of 'Usability', as querying done in simple language after attaching database file, 'Interoperability', as can be deployed on any platform supporting c-plusplus, 'Efficiency',as the result computation and processing happens in milliseconds, 'Flexibility', as can support any form of databases and 'Uniqueness', as no other database management system handling structured data does relevancy search.

Thus, the initial problem statement was to built an API to support relevancy searching on attached structures data and the objective was successfully achieved by the end of the term.

10. Further Enhancements

- *Data Type identification to do IDF for non-categorical data:*
  IDF on a whole works differently for categorical and Numerical data. Identifying and applying IDF for numerical data is definitely something we wish to do in the future.

- *Prefix match:*
  Matching a whole word based only on the first few characters. Trie Trees can be used to solve prefix matching.

- *Make Workload analysis faster:*
  Making the workload analysis faster on a whole by improving the efficiency of the code.

- *IN queries:*
  Works on basis of workload patterns of 'IN' queries relating unrelated data in a column.

- *Improving the User Interface:*
  Enhancing the aesthetics and responsiveness of the interface.

- *Serialization of index objects into hard-disk*:
  Storing the index objects in the hard disk (Serialization) and having the ability to retrieve the index objects on will can ensure that the index objects don't have to be created regularly. Saves time.

11. Bibliography

- Automated Ranking of Database Query Results, Sanjay Agrawal - Microsoft Research, Surajit Chaudhuri - Microsoft Research, Gautam Das - Microsoft Research, Aristides Gionis - Computer Science Dept, Stanford University

- Data mining support in database management systems. Tadeusz Morzy- Poznan university of Technology, Poland

- Mining of Massive datasets - Jure Leskovec - Stanford University, Anand Rajaraman - Milliway Labs, Jeffrey D. Ullman - Stanford University

- Providing Ranked Relevant Results for Web Database Queries - Ullas Nambiar, Subbarao Kambhampati - Arizona State University