An approach to the Optimization of menu-based Natural Language Interfaces to Databases

Fiaz Majeed[1], M. Shoaib[1], Fasiha Ashraf[1]

[1] Department of Computer Science & Engineering, University of Engineering and Technology Lahore, Pakistan

Abstract

Natural language interfaces to databases (NLIDB) facilitate the user to state query to database in natural language. NLIDB then interprets the natural language query into Structured Query Language (SQL) to perform action on target database. Menu-based NLIDB provides restricted set of elements on screen that are utilized to build natural language query. The latest menu-based NLIDB's use WYSIWYM interfaces that focus on automatic formation of popup menus relevant to typed word on editor. The automatic functionality has made the NLIDB more complex with heavy resource requirement to load and execute multiple processes simultaneously. This paper proposes an optimization approach to efficiently use system memory by menu-based NLIDB. It suggests the order of loading and unloading processes in memory at right time. Finally, proposed approach is evaluated on a real dataset. The application following this approach runs efficiently even on low resources system.

Keywords: *Natural Language Interfaces to Databases, Menu-based Interface, WYSIWYM, Resource Optimization, Memory Optimization*

1. Introduction

NLIDB deals with representation of user request to database in his/her native language. NLIDB then maps the user request in standard SQL to retrieve desired results from the target database. The purpose of this interface/system is to facilitate the user by hiding complexities of database query language syntax. Thus the users write their request similar to email message and submit to NLIDB system. System then understands the request and translates it in accurate database query so that the precise results can be retrieved.

A significant challenge for NLIDB is query understanding. It is the most difficult for the system to understand open natural language [12] that is why the most of NLIDB systems restrict users to write bounded queries [1, 11]. The fundamental restricted NLIDB systems include restricted natural language syntax and menu-based systems [2]. The menu-based system provides restricted set of elements from which users are constrained to choose and format their query [3, 5, 6].

In existing menu-based systems, significant attention is not given to economical use of system resources. In fact, multiple memory optimization approaches have been proposed for data-intensive applications in different domains (e.g. data streams [8], data warehouses, and algorithms tuning [14]). It is important to have optimization strategy to handle the increasing size of interfaces. In this research work, menu-based interfaces are being considered. Therefore, an optimization technique has been proposed to manage the menu-based application in memory. It will enable WYSIWYM menu-based applications to become light-weight for machine, provide remarkable performance and become scalable with new functionalities.

The objective of this research work is to propose an optimization strategy for latest WYSIWYM menu-based systems. This paper presents an approach for a key resource, the memory. It suggests accurate time and sequence of loading application processes in memory as well as the time and sequence of unloading them from memory, so that memory remains lightweight during the whole life-cycle of query transformation. The menu-based NLIDBs following this approach can efficiently manage the memory and execute processes even in the systems with lower specification (e.g. Personal Digital Assistant (PDA), Desktop machines etc.).

The rest of this paper is organized as follow: Section 2 demonstrates the related work. Resource management is explained in Section 3. Then Section 4 discusses memory optimization approach. Subsequently, experiments have been performed in Section 5. Finally, Section 6 concludes the work and provides future directions.

2. Related Work

The menu-based systems were developed to enable the user to input natural language query that can be transformed accurately in the target database query language. It provides all elements on the screen and restricts the user to select from those elements for

formatting the natural language query. In this way, system restricts the user to write query that is understandable for the system. Such systems enable the user to write error-free query by means of pointing device [2, 5, 6].

NLMENU [3, 10] is one such system that displays several menus including commands (find, delete, insert...), attributes, nouns, comparisons (between, greater than), connectors (of, and, or) and modifiers. The screen elements contextually and dynamically change by the selection from the displayed menus. For example, on selecting modifier 'that country is', a list of all countries is displayed on the screen. Along with it, system reformats query on addition of any word in it to cater a query that system can understand.

A more advanced form of menu-based interface is presented in [4]. It automatically generates semantic graph from the target database. After building semantic graph, system produces all possible queries following the semantic graph. It builds number of query frames for each node associated with other nodes. This activity is performed only once at the time of system deployment. Once queries are built, they become available in popup menus on the interface. During query building process, user selects query frame from popup menu, then according to selection next popup menus are built. The popup menus reduce the interface area, increase simplicity in visualization and provide guidance for query building.

Research has been done on memory optimization in the area of data warehousing and data streams. The architecture to process large quantity of data streams in presence of limited memory is presented [7]. Moreover, resource optimization is discussed in detail in the literature [8, 9, 14].

3. Management of Resources

The menu-based systems load semantic graph, all possible queries for pop-up menus (the queries that are generated from the semantic graph for each node and their associated nodes), query transformation module, syntactic and semantic grammars. All possible queries may be larger in number depending upon the size of database. The process that dynamically manipulates the frames in pop-up menus resides in memory. The procedure that maintains orientation of the query sentence executes on selecting any frame from the pop-up menu to adjust syntactic flow of the sentence also occupies place in memory. Syntactic rules and vocabulary as well as semantic rules and domain knowledge exists inside memory. The query parser and query transformation module convert the constructed query into SQL following semantic graph that is generated according to the query. Further, the natural language query is mapped to Directed Acyclic Graph (DAG) which is then converted to database semantic graph. Finally, semantic graph, the intermediate template is used to convert the user query into technical SQL form. Hence, the semantic graph generation procedure also resides in working memory. These all elements together overload the working memory.

A system with high memory is required to run all the processes at runtime otherwise system's performance may be badly affected. This problem can be tackled if optimization strategy is adopted. One optimization strategy can be of the following form:

- Pick those processes and data that are required at the moment.

Alongside hold other processes and data on storage disk. In this way, the usage of memory can be increased and menu-based application can run on low memory system. For example, while writing a query initially lexicon, semantic rules, queries and query orientation process are necessary for query construction. Following is the analysis of all processes with respect to optimization parameters.

3.1 Loading of Minimum Queries

Initially, only load generic queries (might be one prominent query for each node). Later, on selection of specific query, retrieve relevant query frames from disk. This method will save the memory in dominant percentage. But disk reads on runtime is expensive activity. For optimum strategy, we will have to tradeoff between memory saving and time consumption.

3.2 Loading of Semantic Rules

The semantic rules are an important component for Query Parser. They are required to format contextual and domain specific queries. It would be an expensive solution to keep the semantic grammar on disk and search appropriate rules for user query. Later, retrieve the matching rule that fulfills current information need. The semantic grammar would be needed on every insertion, deletion or updation in the query. Therefore, this solution is highly expensive and degrades the performance.

Another efficient solution is that we keep the semantic grammar in compressed form inside the memory. The interesting factor in using this technique is to use the method that search in compressed data without decompressing it [13]. The idea is the compression of search key to perform searching in the compressed text. Further, methods are available that decompress the data from any location. In this approach, it is required to decompress rules that are considered appropriate by the use of searching technique. Decompressing anywhere in the file needs to compress the file in blocks. Word-based compression is more suitable than character-based compression.

The above discussion demonstrates that the compression of semantic grammar is space and time efficient. It is important to use efficient decompression method because semantic grammar will be frequently required to use the rules.

3.3 Loading of Lexicon

The lexicon is required when user changes query without using popup menus. Such an activity will be rare because menu provides all domain knowledge within popup(s). This activity will only be committed by non-expert users. So if manual change is frequent, lexicon and semantic rules are also to be retrieved frequently. The important issue is to decide whether lexicon should be kept in memory permanently or retrieve on demand. Lexicon can be placed in permanent storage and access on demand because it will be needed rarely. But in case of its frequent access, it may be taken to the temporary storage. A threshold, say 10 continuous accesses in a specified time can be set for such decisions. These constraints can be applied using programming code, event or DB query.

3.4 Loading of Query Orientation Process

This process reconsiders query coherence in natural language with the reference of natural language grammar. It will run when user selects query frame, changes or deletes any word. Therefore, it will have to run frequently. It is not possible to reload query orientation program repeatedly. This program should remain inside memory so that it can be executed effectively on-demand. Another point to consider here is that natural language grammar is also required to be loaded to support query orientation process. Again, good solution is the compression of grammar with search key and decompress required portion of the grammar.

3.5 Loading of Semantic Graph

The semantic graph is important to be used in two situations. First, it is used to construct all possible query frames in the start when natural language interface is deployed for database representing the domain. This process runs only once. Second, when DAG is generated from user query, it is then compared with semantic graph for SQL query construction. So, semantic graph is needed at the time of SQL query formation. Once DAG is created, only the semantic graph and query transformation module is required inside the memory. The lexicon, query orientation process, semantic rules, and minimum queries are no longer required in memory. So they must be unloaded at this moment. When DAG, semantic graph and query transformation module are there in the memory, enough storage space is available due to unloading of other expired process. But remaining processes must be unloaded on submission of query and initial processes should be loaded again when returning to the query editor for new query or update previous query. This is discussed in subsequent section about loading and unloading of data and procedure categories.

3.6 Loading of Query Transformation Module

It is final process applied on DAG based semantic graph to map in SQL query. This time, only DAG based semantic graph and query transformation module are required in memory. All the other processes must be unloaded before loading them.

The above discussion clarifies that different processes execute on different stages. It means loading of modules and unloading them is a s equential process. Thus it is necessary to decide suitable time and sequence for loading and unloading each process.

4. Memory Optimization

4.1 Sequence of Loading Processes

For efficiency, next process to be executed should be available in memory and ready to be used before the completion of current running process. For example, query translation module must be ready before completion of DAG mapping with semantic graph. The sequence of loading processes is demonstrated in Fig. 1.

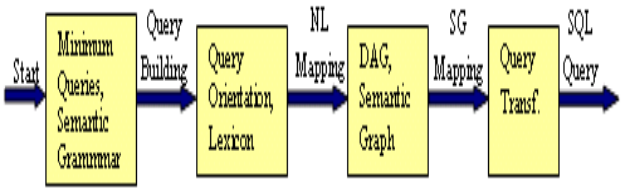

Fig. 1 Sequence of loading processes

At first (see Fig. 1) when menu-based software is executed by the user on system, it loads minimum required queries that make up pop-up menus at runtime and semantic grammar for representing the semantic rules. When user starts writing query on menu-based NLIDB editor it is required to check the query orientation in accordance to natural language (here we assume English language). To do so, lexicon and semantic grammar are desirable that have vocabulary and relevant rules respectively. At this point, query orientation process and lexicon must be taken to the memory. Note that all previous (start) and current

(query building) processes have occupied space in the memory (e.g. at the moment of natural language mapping minimum queries, Semantic Grammar, query orientation process and lexicon are already in memory).
In next section, sequence and timing will be discussed to unload these processes to optimize memory utilization at each moment. Once user completes and submits his/her query, system generates a DAG against the query. Later, DAG is mapped to the semantic graph. Therefore, semantic graph and DAG will occupy space in memory at this instant.
As semantic graph is built, final step is to map the semantic graph to SQL query. To do so, query transformation module must be loaded and ready at this moment. At last, query is ready to be executed on the underlying database.

4.2 Sequence of Unloading Processes

As processes for user's natural language query to SQL query conversion are loaded sequentially, these must be unloaded in the same sequence to reduce burden on memory and increase efficiency. Previous discussion clearly describes that all processes do not run simultaneously. Therefore, we should devise some mechanism to unload the processes that have completed their processing and are no more required for current query.

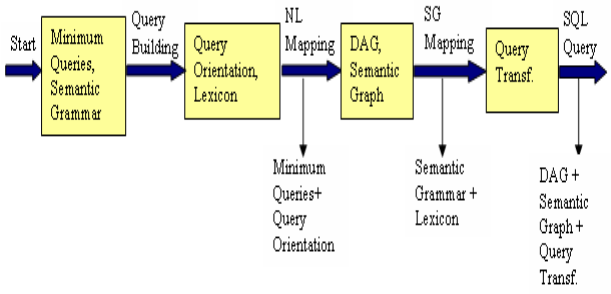

Fig. 2 Sequence of unloading processes

The sequence of unloading of processes is depicted in Fig. 2. Minimum queries, semantic graph, lexicon and query orientation process remain in memory until user submits his/her query. As user submits the query, DAG starts building by the system simultaneously. At this moment, minimum queries and query orientation process are no more required in the memory. So before DAG takes place in memory, these must be unloaded. As DAG is mapped to semantic graph, processes include Semantic Grammar and lexicon are not further required because query transformation module only use semantic graph. After building SQL query by query transformation module; DAG, semantic graph and query transformation module should be unloaded. Since, target query is built by menu-based NLIDB. Now NLIDB should be ready for new user query by loading again the minimum queries and Semantic Grammar that are initially required.

5. Experimental Results

We run menu-based system on the low resources system having temporary storage 256 MB and processing speed 1GHrtz. Different processes take different size depending upon NLIDB application domain. The size of different processes in menu application under study is given in Table 1.

Table 1: Memory consumption by NLIDB processes

NLIDB Component	Size
Minimum queries	1 (KB)
Semantic grammar	101.3 (MB)
Query orientation process	42 (MB)
Lexicon	69.1 (MB)
DAG	30.9 (MB)
Semantic graph	40 (MB)
Query transformation module	50.4 (MB)
SQL query	1 (KB)

In our experimental transportation database, there are 15 relations and each relation has link with at most 6 other relations. So automatic query generator generates at most 6 query frames for each relation.
The semantic graph is depicted in Fig. 3. If we focus on node 'geoloc' in the semantic graph, the query frames constructed by system would be:

(1) Which runaways are at [some geoloc]?
(2) Which airports exist at [some geoloc]?
(3) In which [geoloc], ships are manufactured?
(4) Which countries are located at [some geoloc]?
(5) What channels are located at [some geoloc]?
(6) Which seaports are located at [geoloc]?

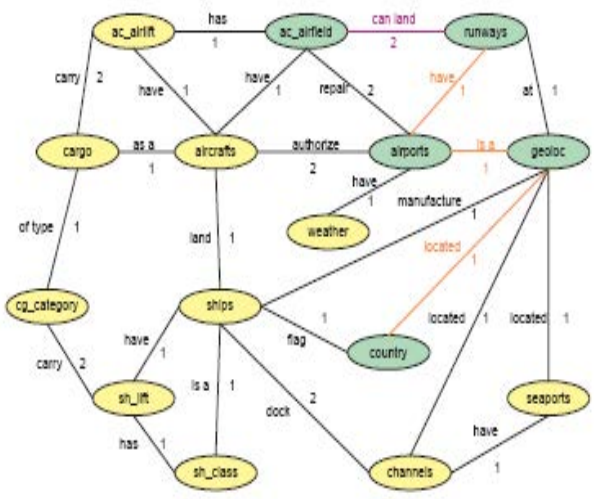

Fig. 3 Semantic graph of transportation database [12]

Therefore maximum query frames generated from transportation semantic graph are 15*6 = 90.

Initially, 15 query frames representing semantic graph nodes are loaded. Therefore, these take maximum 1KB of memory. Semantic graph and lexicon size depends on the domain, the NLIDB address. This example NLIDB consumes 101.3 MB for semantic grammar and 69.1 MB for lexicon. The query orientation process is a standalone application, supports NLIDB, occupy 42 MB of memory. Moreover, DAG and semantic sub-graph building modules take 30.9 MB and 40 MB space respectively. Finally, query transformation module consumes 50.4 MB and resulting SQL query takes 1KB of memory.

The memory consumption by menu-based NLIDB is shown in Fig. 4.

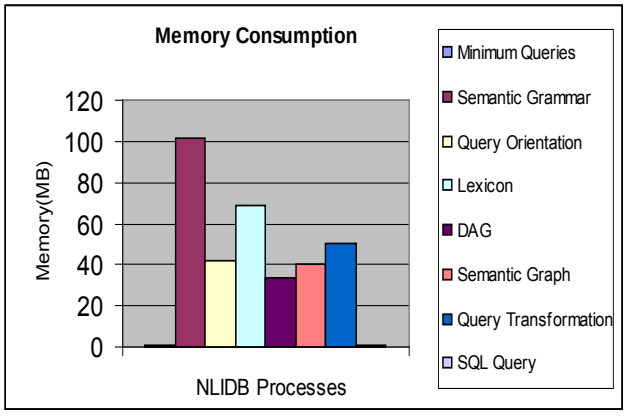

Fig. 4 Memory usage by NLIDB processes

Semantic grammar and lexicon are heavier processes on memory, reserve storage till the last process (query transformation module) of query conversion. It is important to analyze memory consumption with and without our proposed optimization strategy.

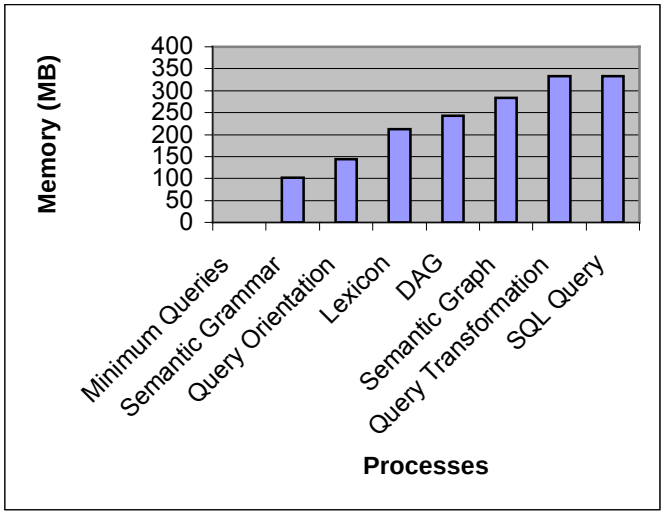

Fig. 5 Memory Usage without Optimization

Fig. 5 depicts the usage of memory without applying any optimization strategy for menu-based application. Loading of processes in sequence (taken along X-axis) and memory size in MB (taken along Y-axis) are parameters of interest in this case. The graph plotting demonstrates growth in memory load with loading of each process. Clearly, experimental system's memory 256 MB is less than the requirement of menu-based application under study (i.e. 330 MB and 2KB).

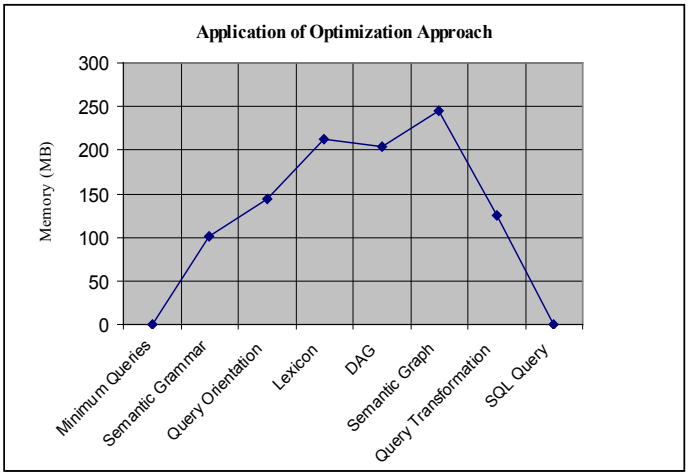

Fig. 6 Memory Usage with Proposed Optimization Strategy

Fig. 6 demonstrates the memory usage by menu-based NLIDB with the application of our proposed memory optimization strategy. NLIDB processes (taken along X-

axis) and memory size (taken along Y-axis) are parameters of interest in this case. The processes are loaded in memory in order from minimum queries to lexicon, while graph goes up straight away. Before building DAG, undesired processes start unloading from memory and memory consumption graph start moving downward. It is worth mentioning here that available memory is now enough for NLIDB application with the use of this proposed optimization strategy.

6. Conclusions

This paper addresses the issue of heavy resource requirement of current complex automated menu-based systems. An optimization strategy for memory management to keep the menu interface light-weighted throughout the life-cycle of natural language to SQL query conversion is proposed. The experiments demonstrate viability of our proposed technique. Since, it m anages application within limited memory and increase throughput of the system. Further, application can be scaled for new functionalities due to its economical usage of memory.

Acknowledgments

The authors thank Tayybah Kiren and Sumera Shaukat for their contribution in proof reading of this document.

References

[1] A. Popescu, O. Etzioni, and H. Kautz, "Towards a theory of natural language interfaces to databases", In Proceedings of the 8th international conference on Intelligent user interfaces (ICIUI), 2003, pp. 149–157.
[2] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch, "Natural Language Interfaces to Databases – An Introduction", Natural Language Engineering, Vol. I, 1995, pp. 29-81.
[3] H. R. Tennant, K. M. Ross, and C. W. Thompson, "Usable natural language interfaces through menu-based natural language understanding", In Proc. SIGCHI conference of human factors in computing systems, ACM Press, 1983, pp. 154-160.
[4] C. Hallett, "Generic Querying of Relational Databases using Natural Language Generation Techniques", In Proceedings of the Fourth International Natural Language Generation Conference (INLG '06), 2006, pp. 95–102.
[5] E. Mueckstein, "Controlled natural language interfaces: the best of three worlds", In proceedings of the 1985 A CM thirteen annual conference on c omputer science (CSC'85), ACM Press, 1985, pp. 176-178.
[6] Thompson, P. Pazandak, and H. Tennant, "Talk to your semantic web", IEEE internet computing, Vol. 9, 2005, pp. 75-78.
[7] F. Majeed, M. S. Mahmood, and M. Iqbal, "Efficient Data Streams Processing in the Real-Time Data Warehouse", In proceedings of 3rd IEEE Intl. Conference on C omputer Science and Information Technology (ICCSIT), 2010, pp. 57-61.
[8] Widomet et al., "Query Processing, approximation, and resource management in a data stream management system", In proceedings of the CIDR Conference, 2003.
[9] D. A. Grossman, S. M. Beitzel, E. C. Jensen, O. Frieder, "IIT Intranet Mediator: Bringing Data Together on a Corporate Intranet", IEEE IT PRO, 2002.
[10] H. R. Tennant, K. M. Ross, C. W. Thompson, and J. R. Miller, "Menu-based natural language understanding", In 21st annual meeting of ACL, 1983, pp. 151-158.
[11] S. Epstein, "Transportable natural language processing through simplicity- the PRE system", ACM Transactions on office information systems, Vol. 3, No. 2, 1985, pp. 107-120.
[12] G. Zhang et al., "Query formulation from high-level concepts for relational databases", In Proceedings of the UIDIS, IEEE Computer Society, 1999.
[13] R. Baeze-yates, and B. Ribeiro-neto, Modern Information Retrieval Systems, Pearson Education Inc., 2005.
[14] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N. Frantzell, "Meshing Streaming Updates with Persistent Data in an Active Data Warehouse" IEEE Transactions on Knowledge and Data Engineering, Vol. 20, No. 7, 2008, pp. 976-991.

Fiaz Majeed received MS degree from COMSATS Institute of Information Technology (CIIT) Lahore Pakistan in 2009. He is currently PhD scholar in University of Engineering and Technology Lahore Pakistan. His research interests include data warehousing, data streams and information retrieval. He has published 5 papers in refereed journals and i nternational conference proceedings in the above areas.

M. Shoaib received PhD degree from University of Engineering and Technology Lahore Pakistan in 2006. He has done hi s post doctorate from USA in 2008. He is currently associate professor in this university. His research interests include information retrieval and data mining. He has published more than 40 papers in refereed journals and international conference proceedings in the above areas.

Fasiha Ashraf received MS degree from University of Engineering and Technology (UET), Lahore Pakistan in 2010. She is currently PhD scholar in University of Engineering and Technology. Her research interests include Information retrieval, Semantic web, Cloud computing and Virtualization. She has published 1 paper in refereed journal and is working on another in the above areas.