

2016-01-18

Integrating Flexibility and Fuzziness into a Question Driven Query Model

Sarhan, Abdullah

Sarhan, A. (2016). Integrating Flexibility and Fuzziness into a Question Driven Query Model (Master's thesis, University of Calgary, Calgary, Canada). Retrieved from <https://prism.ucalgary.ca>. doi:10.11575/PRISM/26577
<http://hdl.handle.net/11023/2760>

Downloaded from PRISM Repository, University of Calgary

UNIVERSITY OF CALGARY

Integrating Flexibility and Fuzziness into a Question Driven Query Model

by

Abdullah Sarhan

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

January, 2016

© Abdullah Sarhan 2016

Abstract

Data plays an important role in our daily life. Thus, data collection, storage, maintenance and processing continue to attract considerable attention. Data may exist in various formats, ranging from unstructured to structured as the two extremes. Traditionally, researchers and practitioners cooperated and developed various data models which form the main foundation for existing database management systems. The relational data model is still dominating despite the rapid development in the techniques used for data collection, storage and processing. Further, a relational database management system supports a structured query language (SQL) for data processing, and it is not possible to access and retrieve data from a relational database without knowing how to use SQL. However, the wide usage of relational databases motivated researchers to develop more user friendly interfaces which would allow a larger population of users to access relational databases. Such interfaces range from visual to natural language based.

This thesis contributes a question driven query model which falls under the natural language based category. The target is to make databases reachable by a larger population, especially after the Internet increased database availability. The proposed model supports fuzziness where every user is given the freedom to define his/her own understanding of fuzzy terms. The developed system absorbs the fuzzy understanding of each user to utilize it while deciding on the result to be communicated back as answer to the raised question. Data mining techniques are employed to guide users in defining their fuzzy understanding. The developed model is intended to help users to retrieve the data they want from a relational database without expecting them to know SQL. In the current version only questions written in English are allowed. The system handles different types of questions, such as (1) simple questions, (2) complex questions with inner joins and where conditions, (3) questions that involves the usage of aggregate functions (e.g., min, max, etc.), and (4) questions with fuzzy

terms. The reported test results demonstrate the effectiveness of the developed system in handling various types of questions raised by a heterogeneous set of users ranging from professionals to naive.

Keywords: Query Model; Fuzzy Model; User Interface; SQL; Relational Database; Structured Database.

Table of Contents

Abstract	i
Table of Contents	iii
List of Tables	vi
List of Figures	viii
List of Symbols	x
1 Introduction	1
1.1 Relational Databases	2
1.2 Problem Definition and Motivation	6
1.3 The Proposed Solution	8
1.3.1 Database Configuration	9
1.3.2 Validating User Input	9
1.3.3 Analyzing User Request	10
1.3.4 Displaying the Results	11
1.3.5 User Feedback	11
1.4 Thesis Structure	11
2 Background and Related Work	13
2.1 Background	13
2.1.1 Data Mining	13
Classification	14
Fuzziness	18
2.1.2 Natural Language Processing(NLP)	20
Dependency parser	20
Name Entity Recognition	21
Dates Phrase Extractor	21
2.2 Related Work	22
3 Proposed Solution and Methodology	26
3.1 Development Tools	27
3.2 User Requirements	27
3.3 Tools Used	27
3.3.1 Stanford NLP Parser	28
SNLP Relation Extractor	29
SNLP Name Entity Recognizer	30
SNLP Time Expression Extractor	31
3.3.2 Hunspell Spell Checker	32
3.3.3 The Pluralization Library	33
3.4 Project Configuration	33
3.4.1 Creating and Loading Project	34
3.4.2 Database Configuration	34
3.4.3 Create Fuzzy Domain and Ranges Dictionary	36
3.4.4 Create Word Synonyms Dictionary	36
3.4.5 Create Database Prefix and Suffix Dictionary	36
3.5 System Architecture	37

3.5.1	User Input	37
3.5.2	Input Refinement	39
	Spell Checking	39
	Changing Text Based Numbers to Numeric Values	39
	Fixing Negation	40
	Replacing Date Expressions with the Actual Date	41
3.5.3	Get Word Dependencies	42
3.5.4	Eliminate Unnecessary Words and Punctuations	43
3.5.5	Map Words to the Database Schema	44
3.5.6	Tables and Columns Selection	45
3.5.7	Question's Return Type	46
3.5.8	Building the SQL Statement	47
	Constructing the Select Clause	47
	Constructing the From Clause	49
	Constructing the "Where" Clause	50
3.6	Fuzziness	52
3.6.1	Fully Automated	53
3.6.2	Semi Automated	56
3.6.3	Manual Specification of the Fuzzy Domains and their Ranges	57
3.7	Display the Result and Receiving the Feedback	57
3.8	Summary	58
4	Experimental Analysis	62
4.1	Testing Environment	62
4.2	Data Set	63
4.3	Example Queries	64
	4.3.1 Simple Questions	65
	4.3.2 Complex Questions	65
	4.3.3 Fuzzy Questions	66
4.4	Performance Analysis	66
4.5	System Accuracy	67
4.6	Summary	68
5	Conclusion and Future Work	74
5.1	Summary	74
5.2	Limitations	75
5.3	Future Work	75
	Bibliography	77
A	System Documentation	85
A.1	System Overview	85
A.2	Query Examples and Results	86
A.3	Saving Data	87
A.4	Modules being Used	88
A.5	System Snapshots	89
B	Data Set Snapshots	103
B.1	Bulk Insert Application	103
B.2	Database Tables	104

B.2.1	People Table Snapshot	104
B.2.2	Services Table Snapshot	105
B.2.3	Category Look Up Table Snapshot	110
B.2.4	Service Reviews Table Snapshot	111
B.2.5	Friends Table Snapshot	116

List of Tables

1.1	Ranking of some Database Systems [2]	3
2.1	Confidence Table Used to Measure the Accuracy of a Classifier	15
2.2	Example on a Classifier Results	15
2.3	Comparing our Approach with the Approach of Li et al. [42]	23
2.4	Comparing our Approach with the Approach of Giordani et al. [28]	24
2.5	Comparing Our Approach with the Approach of Agrawal et al. [3]	25
3.1	Word POS Tagger Example	30
3.2	The Relation between Words in a Sentence	31
3.3	Date Extraction Example	32
3.4	General Database Schema Prefixes and Suffixes	37
3.5	Examples of Text Based Numbers Converted to Numeric Values	40
3.6	Examples on Transforming Abbreviated Negated Words to Full Word Representation	42
3.7	Examples on Transforming Date Expressions to Actual Dates	42
3.8	List of Unnecessary Words that can be Removed from a Sentence	44
3.9	Examples of WH Phrases and their Expected Results	47
3.10	Examples on WH Phrases and their Expected Parsing	48
3.11	Example on Conditions and their Corresponding Expression in the “Where” Clause	51
3.12	Methods for Handling Fuzziness	53
3.13	Fee Column Values	53
3.14	Fully Automated Fuzzy Ranges	54
3.15	Calculated Ranges for the Predefined Five Fuzzy Domains	56
4.1	Characteristics of the Computer Used in Testing the Developed System	63
4.2	Databse Tables’ Columns and their Type	69
4.3	Expected Input for each Column Type	70
4.4	Number of Records in the Populated Tables	70
4.5	Sample of Simple Questions	70
4.6	Sample of Complex Questions with Possible Corresponding SQL Statements	71
4.7	Sample of Fuzzy Questions and Possible Corresponding SQL Queries	72
4.8	Number of Questions Executed to Measure System’s Performance	72
4.9	Average Time Needed for each Query Type	72
4.10	Average Time Needed by each Analysis Component (ms)	72
4.11	Number of Wrongly/Correctly Analyzed Queries	73
A.1	List of External Modules Used by each Feature	89
B.1	List of People Available in the Database	105
B.2	A Snapshot of Data Stored in Person_Services Table	105
B.3	A Snapshot of Data Stored in LKUP_Category Table	110

B.4	A Snapshot of some Sample Reviews Maintained in Service_Reviews Table .	111
B.5	A Snapshot of Data Stored in Person_Friends Table	116

List of Figures and Illustrations

1.1	An example of a Relational Database Diagram	4
1.2	SQL Syntax	5
1.3	SQL Query for Getting city name for all Canadian people	6
1.4	Example on Fuzzy SQL Query	8
1.5	Block diagram of the Proposed Query Model	9
2.1	Mapping of Neural Network Nodes	17
2.2	Triangular Membership Function	19
2.3	Example on Word Dependencies Generated by Stanford Dependency Parser .	21
3.1	The communication between the System Components	28
3.2	Features of the Stanford NLP Tool	29
3.3	Suggestions Produced by the Spelling Checker	33
3.4	Database Configuration Helper Form	35
3.5	The general Flow of Control in the System for Analyzing User Input	59
3.6	An Example SQL Query Using more than 2 Inner Joins	60
3.7	An Example of SQL Query using 2 inner joins	60
3.8	The Five Fuzzy Domains of the “Fee” Column as Discussed in the Example that Illustrates Semi-Automated Specification of the Fuzzy Domains	60
3.9	Visualization of the Relationship between a User’s Question and the Database Tables and Columns	61
4.1	Schematic Diagram of the Example Persons-Services Database	64
A.1	System’s Main Page after a Successful Connection to a Database	86
A.2	A Question to Demonstrate the Usage of an Aggregate Function	87
A.3	A question which Involves the Usage of Fuzziness	87
A.4	An Example Complex Question	88
A.5	System’s Splash Screen	89
A.6	The Form Used for Project Selection or Creation	90
A.7	The form Used to Enter Database Information	90
A.8	Helper Form for Displaying a List of Available Databases	91
A.9	Helper Form before Displaying the List of Available Databases	92
A.10	Displaying Tables and Columns for the User	93
A.11	Brief Explanation of Various Database Schema Elements	94
A.12	Suggestions form for the Misspelled Words	95
A.13	The form that Manages Tables’ Synonyms Defined by the User	96
A.14	The form that Manages Columns’ Synonyms Defined by the User	97
A.15	The form that Manages Tables’ Suffix and Prefix	98
A.16	A list of Unnecessary Words that can be Modified by the User	98
A.17	Modifying Unnecessary Words during analysis	99
A.18	Asking for User’s Choice when Ambiguity Occurs	100
A.19	List of Mapped Words Obtained during the Analysis	101

A.20	The form Responsible for Managing Fuzzy Domains and Ranges	101
A.21	A list of all Questions Made by the User	102
A.22	Providing Users the Ability to Save their Data	102
B.1	A snapshot of the Bulk Insert Application	104

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
C#	C# Programming Language
RDBMS	Relational Database Management System
SNLP	Stanford Natural Language Processing Tool
NLI	Natural Language Interface
NLP	Natural Language Processing
XML	EXtensible Markup Language
SQL	Structured Query Language
K-NN	K-Nearest Neighbors Algorithm
VIREX	VIual RElational to XML
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
ANN	Artificial Neural Network
CRF	Conditional Random Field
TMF	Triangular Membership Function
POS	Part Of Speech
OS	Operating System
IDE	Integrated Development Environment
RAM	Random Access Memory
CPU	Central Processing Unit
GB	Gigabyte

Chapter 1

Introduction

It is not possible to think of how people could function without the existence of data which is a great resource of knowledge for decision making. Over history people realized the need to collect and store data in various forms. Researchers developed data models to help in structuring data and hence provide a systematic access to data by a variety of users. For instance, books are structured into chapters which are composed of sections, subsections, paragraphs, etc. A table of content or an index allow readers to seek to a specific page to read. As far as databases are concerned, network and hierarchical models dominated in 1960s and 1970s until academia and industry adopted the relational data model developed by E.F. Codd at IBM research centers in 1970s. Since then the relational model is dominating in the database area. It is so strong that it has resisted to all efforts to replace it by other models, including object-oriented data model and XML data model, among others. The relational model survived after absorbing all the additional features that were thought of as advantages of the other models.

The wide spread of mobile and web applications may be seen as the main reason behind the current increase in the amount of data collected. Though the relational database management system (RDBMS) has shown a great success in managing the storage and processing of structured data, unfortunately not everyone is capable of retrieving data stored in a RDBMS. Access to a RDBMS is restricted to those who know SQL (structured query language), which is the standard query language of RDBMS. SQL may be seen as a high level database programming language. Even experts might face some difficulties when coding complex queries using SQL. Thus, it is hard for naive users who lack computing background and programming experience to learn SQL. We argue that naive users should have an al-

ternative way to retrieve the data they want without the need to learn SQL or depend on database experts to extract the reports they need.

Naive users should have direct access to RDBMS to retrieve the data they want for their work and not to be dependent on reports developed by database experts. This may become possible by a flexible user-friendly interface for RDBMSs. This thesis proposes a question-driven user interface that can be equally well be used by expert and naive users to retrieve data from a RDBMS. The proposed system should retrieve expected results by analyzing users' requests expressed as structured English questions. As imprecision is part of natural language, we decided to extend the model to cover fuzziness in queries [65, 4, 39]. Using Natural Language Processing (NLP) combined with some data mining techniques will allow us to analyze fuzzy queries. The current system has been configured to work with Microsoft SQL Server and questions can be expressed in English. However, it can be smoothly expanded to cover other relational database platforms and languages other than English. This have been left as future work.

This chapter is organized as follows. Section 1.1 introduces relational databases. Section 1.2 identifies the problem tackled in this thesis along with a description of target users. Section 1.3 is an overview of the proposed solution. Section 1.4 shows the structure of the remainder of the thesis.

1.1 Relational Databases

The relational database model was first proposed in 1970 by E.F. Codd [17], since then researchers have been working extensively on improving this model by absorbing aspects which have been deemed necessary and hence motivated developing other models which were developed over the past three decades, e.g., object-oriented model. Thus, many relational database management systems have been developed [47, 63, 55] by a number of vendors, including Oracle, IBM, TeraData, etc. However, the recent tremendous increase in the

amount of data collected and the rapid increase in the number of users accessing the collected data has enforced the need for more flexible data and query models which do not require all data to be mapped into structured relational databases only accessible by SQL. This motivated a new trend called the not only SQL (NoSQL) databases [31, 11] which allow a wide range of users to access data in various formats, including structured and unstructured. Indeed, collecting data at high speed and from a wide variety of sources encouraging adapting NoSQL databases especially for online and real-time data analysis where it is not affordable to spent time on matching data to an existing structure to make it understandable and accessible by a given interface.

However, NoSQL has not been well adopted yet due to immaturity and maintenance difficulties [52, 41]. Many undergoing improvements of the available RDBMS target expanding their capabilities to smoothly handle big data with all its characteristics, including volume, velocity, variety, etc. Table 1.1 shows a ranking of a number of database systems which are mostly relational; it is obvious that most relational databases rank top in terms of usage depending on a scoring system that involves the six different criteria explained in [2].

Table 1.1: Ranking of some Database Systems [2]

Rank			DBMS	Database Model	Score		
Oct 2015	Sep 2015	Oct 2014			Oct 2015	Sep 2015	Oct 2014
1.	1.	1.	Oracle	Relational DBMS	1466.95	+3.58	-4.95
2.	2.	2.	MySQL	Relational DBMS	1278.96	+1.21	+15.99
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1123.23	+25.40	-96.37
4.	4.	↑ 5.	MongoDB 📈	Document store	293.27	-7.30	+52.86
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	282.13	-4.05	+24.41
6.	6.	6.	DB2	Relational DBMS	206.81	-2.33	-0.86
7.	7.	7.	Microsoft Access	Relational DBMS	141.83	-4.17	+0.19
8.	8.	↑ 10.	Cassandra 📈	Wide column store	129.01	+1.41	+43.30
9.	9.	↓ 8.	SQLite	Relational DBMS	102.67	-4.99	+7.71
10.	10.	↑ 12.	Redis 📈	Key-value store	98.80	-1.86	+19.42

One of the main attractions of relational databases is the way they manage the storage of data. The model was inspired from the set theory and hence has a strong mathematical

foundations. Data is stored in tables, where each table contains data that describes a specific entity type or a table may act as a connector between interrelated tables. A table has a list of columns representing various characteristics of the corresponding entity type. Further, each table must have a unique primary key which consists of a minimum set of at least one attribute. An example database schema is shown in Figure 1.1.

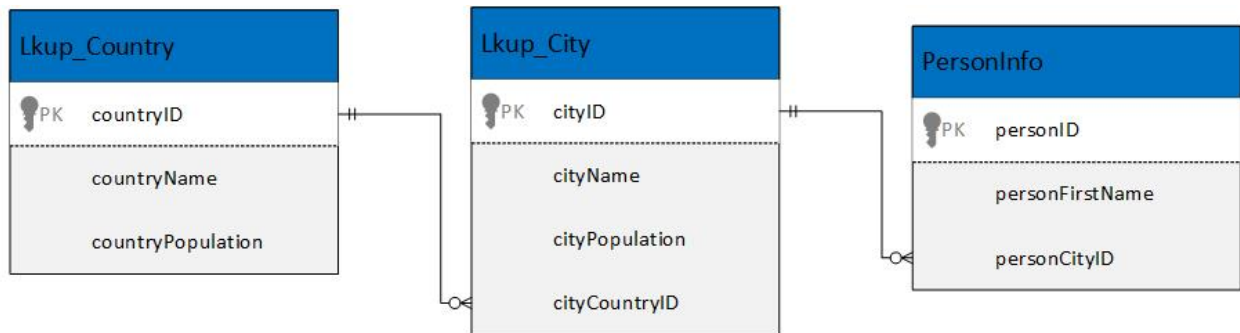


Figure 1.1: An example of a Relational Database Diagram

The database schema shown in Figure 1.1 includes three tables, each represents an entity. The relationships between the three tables are realized using foreign keys as indicated by the links between tables. It is recommended to assign meaningful names to tables and attributes. For instance, the table named PersonInfo is intended to contain information about people. It has three attributes where each attribute has a corresponding column. Each column has a type which indicates expected domain of the values to be stored in the column, e.g., numeric, date, text, boolean, etc. This kind of information helps in building conditions that can be used to filter the retrieved data. Each table in a database should have a primary key which is a minimum nonempty set of attributes that uniquely identify each row (interchangeably called tuple). For instance, the table PersonInfo shown in Figure 1.1 has personID as primary key.

Another type of key directly related to primary key is called *foreign key*, which is a representative of a primary key to indicate a link that connects two entities. In other words, Primary-Foreign key concept is important to build relationships between tables. For example, the column named PersonCityID in PersonInfo table is linked to cityID column

which is the primary key in lkup-City table. Here, it is worth mentioning that only primary keys of tables that represent entities may have corresponding foreign keys; that is, tables that represent relationships from the entity-relationship diagram will not have any primary key with corresponding foreign key(s). Finally, more explanation about the relational database schema, tables, attributes, keys, etc. can be found in [22].

Retrieving and updating data in a relational database is one of the key features provided by RDBMS. Users can update, add, delete or retrieve data from a RDBMS. In general, a prior knowledge of SQL is needed to perform such tasks. The retrieval statement of SQL consists of three main clauses which are “SELECT”, “FROM” and “WHERE” as in the example query shown in Figure 1.2; other clauses may be added (e.g., having, group by, order by, etc.) for coding more sophisticated queries that involve aggregation, sorting, etc. [22].

```
SELECT  column1,columns2 .....
FROM  Tablename...
WHERE column1 <operator> value. columns2 <operator> value...
```

Figure 1.2: SQL Syntax

The “Select” clause is used to specify attributes which should be retrieved from the nonempty set of tables mentioned in the “From” clause. The asterisk (*) may be used to retrieve all attributes from the tables expressed in the “From” clause. The “From” clause is used to specify tables to be used in constructing the result. Listing multiple tables in the “From” clause means the query will consider their cross product. The “Where” clause is used to filter the outcome from the “From” clause. A standard “Select”, “From”, “Where” query corresponds to the cross-product of tables specified in the “From” clause; then filtering the result by applying on the result of cross-product a select statement with the condition specified in the “where” clause; after that, the result is processed further by applying a projection operation to include from the outcome of the selection operation only columns mentioned in the “Select” clause. Finally, it is possible to explicitly use all types of the join

operation in the “From” clause as demonstrated in Figure 1.3.

An example query with the three main clauses is shown in Figure 1.3. A rich set of comparison operators is allowed in the “Where” clause, including =, !=, >=, <=, <, >, like, between, etc. The query given in Figure 1.3 retrieves names and cities of Canadians from the related tables in the database. More examples queries can be found in any classical database textbook, e.g., the book by Elmasri and Navathe [22].

```
SELECT  PersonInfo.PersonFirstname,lkup_City.cityName
FROM    personInfo
        inner join    lkup_City on    PersonInfo.personCityID= lkup_City=cityID
        inner join lkup_Country on lkup_City.cityCountryID=lkup_Country.countryID
WHERE   lkup_country.countryName='Canada'
```

Figure 1.3: SQL Query for Getting city name for all Canadian people

To wrap up, the relational database model is one the most popular database models for handling and organizing data. Recent developments in industry and academia clearly identified the need for some enhancements to RDBMS to be capable of handling big data; this is one of the main targets RDBMS companies are currently working on. This means that RDBMS will continue to successfully absorb emerging characteristics distinguishing their competitors and hence will continue to lead the market as it was the case following the development of object-oriented databases. The latter disappeared after RDBMS vendors expanded their coverage to subsume the attractive characteristics of object-oriented databases.

1.2 Problem Definition and Motivation

Though a RDBMS is attractive as a data repository due to the solid underlying mathematical foundations, accessing data stored in a relational database is mostly restricted to people who know SQL. In other words, SQL stands as a barrier between domain experts who need data

from database repositories. The database community realized this serious issue at early stage and considerable efforts have been dedicated to provide user-friendly interfaces to RDBMS's with the target to increase number of users who can retrieve data from a RDBMS without any need to know SQL. One trend in this direction is providing natural language interfaces to RDBMS. This will allow domain experts who are in general not database experts to access and retrieve data without feeling the need to consult a SQL expert [32].

This thesis provides a possible solution for this problem that naive users are usually facing. It builds on previous achievements of our research group where VIREX was developed. VIREX is an attractive user-friendly visual interface that allows users to express their queries by ticking boxes next to the names of tables and attributes to appear in the result [34, 35, 44, 46, 45]. VIREX also allow users to specify a filtering prediction by using a specific box that provides a listing of all attributes that exist in the database with opportunity to use a rich set of comparative operators, including fuzziness. Further, VIREX could connect to RDBMS and XML databases in the back-end. In this thesis, the target is to extend the power of the user interface by providing a natural language style that allows to express queries as questions that follow a specific structure. We argue that deciding on a natural language extension to the user interface is important to avoid neglecting imprecision which is a main component of natural language expressions. This is covered in the developed system by having fuzziness as part of the query model.

SQL suffers from serious limitations when it comes to handling imprecision in queries. SQL is built on standards that are based on boolean interpretations [3] which prevent database experts from making fuzzy requests. For example, assume we want to retrieve cities with high population. Using the database structure in Figure 1.1, the query will be coded as shown in Figure 1.4. However, executing this SQL query will return an error because it is not possible to compare text ("high") with the values in column ("Population") which is of type numeric. Handling user's imprecision is a feature that SQL lacks. However,

having such feature will add flexibility to users who are interested in expressing their queries with imprecision and still get back accurate results. This is what we try to handle in the proposed system.

```
SELECT    cityPopulation
FROM lkup_City
WHERE lkup_City.cityPopulation='high'
```

Figure 1.4: Example on Fuzzy SQL Query

The need for users to express their queries with imprecision was realized by researchers since 1980s [27, 26, 51, 58, 62]. However, the efforts of various groups who were involved in this research trend were not well received by industry simply because the main trend was to incorporate fuzziness in database design. This means forcing one fuzzy understanding of imprecision and imposing such understanding on all database users. Such a solution will lead to confusion on users' side because each user (group of users) has his/her (have their) own understanding of imprecision when expressed in natural language. For instance, "hot" weather has different interpretations by various groups depending on climate of the region they live in, and hence it is not acceptable to have one understanding of "hot" pinned in the database.

This thesis describes the development of a question-driven query interface to RDBMS's. It allows users to: (1) express queries as spoken English questions, and (2) incorporate imprecision in their queries to better express their needs.

1.3 The Proposed Solution

The main contribution of this thesis is a flexible question-driven query model that allows end-users to express their queries without feeling any need to learn a database specific query language. Users are given the freedom to state their queries as questions that adhere to a certain structure without sacrificing the opportunity to incorporate imprecision. The block

diagram of the proposed query model is shown in Figure 1.1 . Main components of the system may be enumerated as follows: (1) configuring the database, (2) validating user input, (3) analyzing user’s request, (4) displaying the result and (5) getting feedback. These components are briefly described in this section.

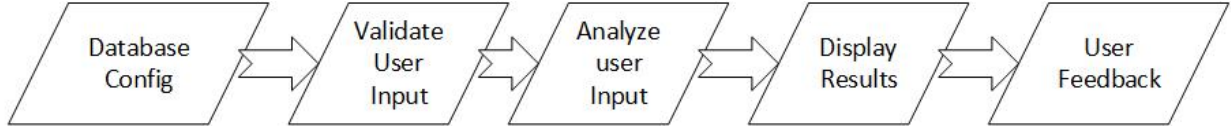


Figure 1.5: Block diagram of the Proposed Query Model

1.3.1 Database Configuration

We developed the query model as a plug-in which connects to a given relational database in order to facilitate question-driven queries. Thus, it is required to set the configuration by defining the server type and credentials. The collected information is stored in an XML file to avoid bothering users for entering the same credentials every time they use the system. XML is a platform independent extensible markup language used to store data, it is a tree based structure which is different from first normal form based relational database. The reason behind using XML is the availability of a built in parser in .NET that is fast in loading and parsing data. The same XML file used for saving the configuration is also used to partially save database schema information, including tables name, columns name and tables relations. Chapter 3 describes in details the information stored in the mentioned XML file.

1.3.2 Validating User Input

The main feature of the proposed system is its simplicity, we tried our best to minimize user involvement in decision making during the analysis; and at the same time we should return to the user the expected accurate results. Users should only enter their queries in a question format and the system will analyze the questions and return the results to users. Limiting users involvement to entering questions is one way to keep track of the structure

and content of the user input. Otherwise, if we allow users to specify their queries as free text then it will be hard to analyze the text properly without involving a very sophisticated natural language processor. Such an approach is outside the scope of this thesis. Actually, the database community concentrate only on providing specific yet powerful approaches for expressing user queries. A high level query language like SQL has been proven effective and attractive to professional users and a visual user-interface or question-driven model would be sufficient and satisfactory for a naive user. Finally, it is worth mentioning that users can still code their queries in SQL and run them using our system.

1.3.3 Analyzing User Request

Questions input by users should be refined before starting the analysis. The refinement is necessary to make sure that the questions do not suffer from any structure and/or spelling mistakes which might effect the accuracy of the results. We used Stanford natural language processing (NLP) tool for dependency extraction, entity name tagger, POS tagging and Data expression detection. Chapter 3 includes more details about these aspects.

The mapping process between user input words and database schema requires using multiple techniques. These include pluralization and singularization of word, removing unneeded words, and usage of word synonyms. A scoring system was built to indicate tables to be used for the mapped words. In addition, we have a very powerful algorithm that helps in detecting multiple conditions in questions.

In case of fuzziness, the system analyzes existing data to derive some initial ranges for the fuzzy sets where triangular shape has been adopted in this study. Users have the option to modify the suggested ranges or they may add more modules to the analysis. We keep track of the ranges selected by users for later usage. We also keep track of users' options in case of ambiguity during the analysis in order to avoid frustrating/offending users by asking them to re-input the same information.

The proposed system works perfectly on different types of queries; this will be shown in

details in Chapter 5. However, if ambiguous cases exist, users are approached to get the best options that match their intended requests.

1.3.4 Displaying the Results

For reporting back the results to users, we tried to make this as simple as possible by employing visualization. We used data grid to display the retrieved results. In addition, we display for users the translated SQL query that corresponds to their questions. Two benefits of this approach are: (1) users will trust the output more by reviewing the actual SQL query, and (2) users will get the opportunity to learn how to code queries in SQL by watching the SQL code equivalent to their questions.

Another feature added to the system allow users to visualize how their input is connected to tables and columns in the relational database. For this, we use D3JS which is a java script based tool that takes data as input and displays how they are connected [1]. Using D3JS it is possible for users to interactively visualize the output.

1.3.5 User Feedback

In all systems, feedback is one of the most important features that help in evaluating system's performance. In our system, we added this feature to measure users' satisfaction with the results retrieved. This will help in deciding on improvements to be done to the system. In other words, system improvements will be decided and conducted based on the feedback provided by users.

1.4 Thesis Structure

The remaining chapters of this thesis include details related to the proposed approach briefly described in this chapter. Chapter 2 covers the necessary background required to understand the techniques used in this research. Chapter 2 also includes an overview of the related work

by emphasizing how it is different from our work. Chapter 3 describes the proposed solution and the methodology, including the techniques used to parse users' questions and how they are mapped to the database. Chapter 4 reports the results and the time needed to complete a query. Also reported in this chapter is the result of the analysis of the feedback that was collected from users of our system. Chapter 5 includes a summary of the thesis, some conclusions and future research directions. Finally, Appendices A and B are added at the end of the thesis to show some snapshots of the developed system and the sample database used in the testing, respectively.

Chapter 2

Background and Related Work

The research described in this thesis is based on NLP and data mining techniques. These techniques will be briefly discussed in this chapter. We will also discuss the work already completed and published by different scholars. We will emphasize how our approach is different.

2.1 Background

Data mining may be seen as the process of automated learning and knowledge discovery by utilizing a set of algorithms and techniques capable of extracting hidden information from data [7]. In our research, we used NLP combined with data mining techniques to extract necessary information from the questions posed by users who are interested in retrieving some content from a given RDBMS. This section will briefly cover the main data mining and NLP techniques used in our research.

2.1.1 Data Mining

Data mining is a sub field of computer science which makes use of machine learning techniques to analyze and discover hidden information in data. It is mainly used to return implicit information from data whose the returned knowledge cannot be easily discovered otherwise [33]. Data mining employs sophisticated analysis techniques to be able to get hidden information from data. Various techniques have been developed mostly in the past three decades to assist in the data mining process.

Learning based data mining techniques are mainly categorized in two main groups, namely supervised and unsupervised learning algorithms. Supervised learning requires users

to predefine domains and then decide on how to map objects into domains by considering characteristics of some already known objects. The derived mapping policy will be applied to decide on the domain of any new object; it is the domain that fits the object's characteristics the most. Classification is considered to be a supervised learning technique. On the other hand, a learning technique is said to be unsupervised when domains are not defined in advance, but they are derived from the data. Clustering is considered to be unsupervised learning technique [30].

In this research, we apply classification to extract useful information that can be used to map user's request to the database schema. In addition, we use clustering to decide on potential fuzzy sets for the fuzziness model employed in this thesis. We used triangular fuzzy membership function as described in Section 3.6.

Classification

Classification is one of the most famous techniques in data mining. The classification process involves two main tasks which are accomplished based on some data; this is the case for all data mining tasks which are by definition data centric. In other words, it is not possible to talk about data mining techniques without specifying the application domain and the particular data to be considered in the mining process.

For the classification process, it is required to have an initial data which consists of a set of objects each has its own values for some specific characteristics. Also, there should be some known classes or categories for objects such that each object belongs to one class based on the combined values of its characteristics relevant to the classification. First the data is split into two disjoint subsets, one called training data and the other constitutes testing data. It is important to have all classes somehow represented in the training data which is used to construct a classification model. The quality of the model is judged by evaluating its power in finding the right classes for objects in the test set. The accuracy of the model is measured as the percentage of objects which are correctly classified.

A number of classification techniques are described in the literature, such as Neural Network Classifier, Naive Bayes Classifier, Decision Tree Classifier, K-Nearest Neighbor Classifier, Support Vector Machine, among others [30, 56].

To interpret the words constituting questions, we used classification rather than clustering since we have some predefined domains that every word should be mapped to. In other words, for named entity recognition (NER) process, we used four domains to map items which are person, organization, Location and miscellaneous; any new item should be mapped to one of these four domains. In English, we have some predefined clauses, such as auxiliary, conjunction, etc., and any clause type should go under one of the specified four categories.

Table 2.1: Confidence Table Used to Measure the Accuracy of a Classifier

		Predicted	
		Class 1	Class 2
Actual	Class 1	True Positive	False Negative
	Class 2	False Positive	True Negative

Table 2.2: Example on a Classifier Results

		Predicted		
		Canadians	Non-Canadians	Total
Actual	Canadians	10	5	15
	Non-Canadians	2	15	17
	Total	12	20	32

The quality of a classifier is measured by using the confusion matrix or contingency table, e.g., the two-class matrix shown in Table 2.1. For example, consider a classifier which has been set to classify some given objects into one of two classes, say Canadians and Non-Canadians. Assume there are 32 persons who are actually classified as 15 Canadians and 17 Non-Canadians. The Canadians row in Table 2.2 shows that 5 out of the 15 Canadians have been predicted as Non-Canadians; and the Non-Canadians row in Table 2.2 shows that 2 out of the 17 Non-Canadians have been predicted as Canadians. In other words, the result from

the classifier employed to classify the 32 persons into Canadians and Non-Canadians reported 15 true positives, 5 false negatives, 2 false positives and 15 true negatives. These four values are very crucial to study characteristics of the classifier, including recall, precision, accuracy, etc.

Equations 2.1, 2.2 and 2.3 show the formulas for calculating sensitivity (recall), precision, and accuracy, respectively. For the classification results reported in Table 2.2, sensitivity is 0.4, precision is 0.83, and accuracy is 0.78125.

$$\frac{TP}{(TP + FN)} \quad (2.1)$$

The Formula for Calculating Sensitivity of a Classifier

$$\frac{TP}{(TP + FP)} \quad (2.2)$$

The Formula for Calculating Precision of a Classifier

$$\frac{TP + TN}{(TN + FN + TP + FP)} \quad (2.3)$$

The Formula for Calculating Accuracy of a Classifier

Two classification techniques are used in our research. The first is called Artificial Neural network Classifier and the second is Conditional Random Filed Classifier. The former is used to extract dependencies and the latter is used for name entity recognition. We will elaborate on the reasons behind using these two kinds of classifiers in the rest of this section.

Artificial Neural Networks (ANN) form one type of computing techniques that are capable of performing some interesting analysis on non linear data. ANN have proven their efficiency in dependency extraction (see Section 2.1.2) which we will use in our study [5, 29]. They are characterized by having high accuracy in feature extraction, optimization, noise immunity and categorization which are among the main requirements for our system. A

neural network finds the relationship between the first and the last words in a sentence; unlike other dependency detection algorithms which decide on word type based only on preceding and following words, and hence might result in misleading information.

Neural Networks tend to find hidden links between the input. Finding such kind of hidden information will help us in getting real dependency between words in a sentence, and hence will lead to more accurate results.

Figure 2.1 shows an example which demonstrates how a neural network works. The input and the hidden links are mapped based on the trained models which have weighted nodes. In other words, ANN use observation and weights to generate the best output. Many algorithms are commonly used in Neural Networks. In our research, we used the cube algorithm which proves its accuracy over the other presented algorithms [15].

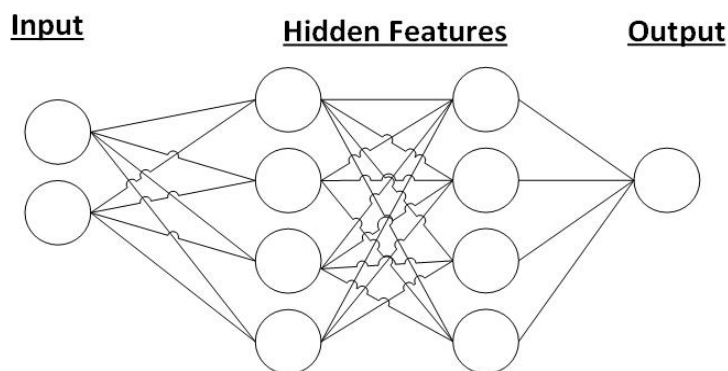


Figure 2.1: Mapping of Neural Network Nodes

Conditional Random Field (CRF) is a statistical model mainly used in machine learning and pattern recognition. CRF was first introduced by Lafferty [40]. Since then it started to receive considerable interest and researcher used it in many different domains, e.g., [66, 25, 64, 60]. Some classifiers usually label an object without taking into consideration the influence of its neighbors. Neighbors of a given object have a great effect on its type. For example, Washington can be the name of a person or a place. But, we can't know its exact meaning unless we take into consideration its neighbors to understand the context.

CRF effectively uses neighbors to best predict the type of a word. Other models have been

proposed for sequential processing, such as Hidden Markov Model (HMM) and Maximum Entropy Markov Model (MEMM). These models suffer from the label bias problem which might result in an inaccurate mapping. However, it has been proved that CRF is more efficient in entity recognition [40, 48] compared to HMM and MEMM. In our research, CRF classifier is used to identify the type of unmapped words and for extracting date expressions.

Fuzziness

Fuzziness is the act of having vague words in users' requests. People usually find it easy to express what they need imprecisely, i.e., in a fuzzy way. In other words, fuzzy words in general have imprecise meanings and hence it is not possible to retrieve a result based on fuzzy terms without further analysis [19, 67]. For example, suppose a database user wants to know people who have high service fee. The word "high" in this sentence is considered to be vague. It is not possible to know how much high a user meant, is it the highest, close to high, etc. To handle such requests effectively, the research conducted for this thesis covers the ability to handle fuzziness in users' queries. The limitations that we faced while handling fuzziness will be described in Chapter 5.

Discrete values have one interpretation, e.g., an object either exists or not, which means the value of existence is either one or zero. On the other hand, fuzzy values allow for smooth transition in the $[0,1]$ range to express the degree of existence in the domain with 0 and 1, respectively, refer to 100% does not exist and 100% exists, while all other values between 0 and 1 refer to partial existence. The degree of membership may be either specified in discrete form by deciding on degree of membership for each given value or computed using a function, including triangular, trapezoidal, Gaussian, etc. In other words, various membership functions have been defined by researchers for handling the fuzzy concept [37, 67]; in our system we used Triangular Membership Function (TMF) due to its simplicity [36, 21, 50]. To use TMF for calculating membership degree for each given value, we should apply the formula in Equation 2.4, where a is lowest bound value in a specific domain, b is

centroid (midpoint) of the domain and c is highest value in the domain.

$$\mu F(x, a, b, c) \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b \\ \frac{c-x}{c-b} & \text{if } b \leq x \leq c \\ 0 & \text{if } c < x \end{cases} \quad (2.4)$$

Triangular Fuzzy Membership Function

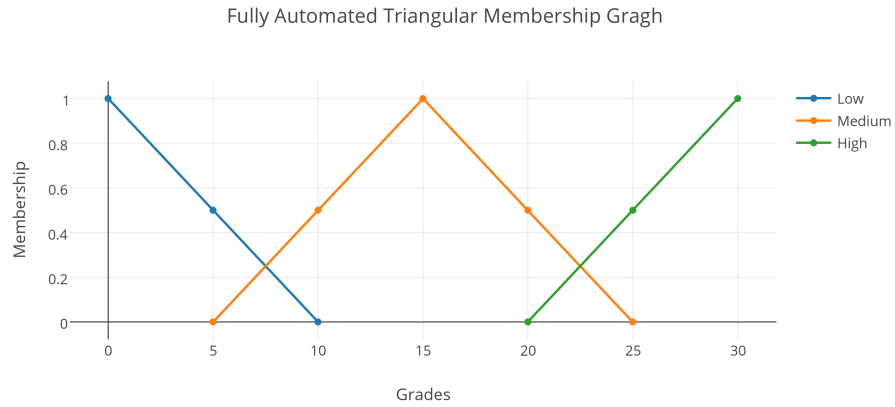


Figure 2.2: Triangular Membership Function

To illustrate the computation of membership function values, consider the fuzzy domains shown in Figure 2.2 which specifies three fuzzy sets for values of a variable called “Grades”, and assume we want to calculate membership degree when “Grade” takes the value 2. We first check under which domain the value 2 is located. Using the information in Figure 2.2, it can be easily seen that the value 2 is located under the domain “Low”. By applying the formula in Equation 2.4, the membership degree of 2 will be $\frac{(10-2)}{(10-0)} = 0.8$, this means when the value of “Grade” is 2 its membership degree in the “Low” domain is 0.8.

Finally, it is worth mentioning that fuzziness refers to possibility which is different from probability. Though both are expected to take a value in the range $[0,1]$, the former is measured by a membership score while the latter is computed as the percentage of possible choices with respect to all existing choices. For example, assume the probability of Chris

to have a high grade is 80% and the membership of Chris in the "High" domain is also 80%. The former value shows that the chance for Chris to have a high grade is 80% and hence he has 20% chance to be in the other categories of grade. On the other hand, 80% as membership value indicates that Chris grade is 80% close to be the highest and this has nothing to do with the membership degree of the grade of Chris in the other fuzzy domains as each fuzzy domain has its own membership function which should be applied to compute corresponding degrees of membership.

2.1.2 Natural Language Processing(NLP)

Natural Language processing is a hot topic in computer science and especially the artificial intelligence domain. The main usage of NLP is to analyze text in order to get meaningful information. Over the past decades, many researchers have been working in this domain leading to the development of efficient algorithms used for text analysis [10]. We used NLP in our system to analyze users requests in order to find relevant words and map them to database tables and columns.

Two main things were used for the mapping process, the first is dependency between words and the second is word type which is known as name entity recognition (NER). NER is applied to know the types of some words in a sentence. In the rest of this section, we will talk briefly about dependencies and NER.

Dependency parser

Knowing the dependency between words is very important in our research. Dependencies allow us to discover hidden information in a given text; such information cannot be found using normal analysis techniques [49].

Figure 2.3 shows some example dependencies, where each POS and Relation tag has a specific meaning which can be found in the work described in [18]. Many approaches have been proposed to extract dependencies from a sentence but to the best of our knowledge

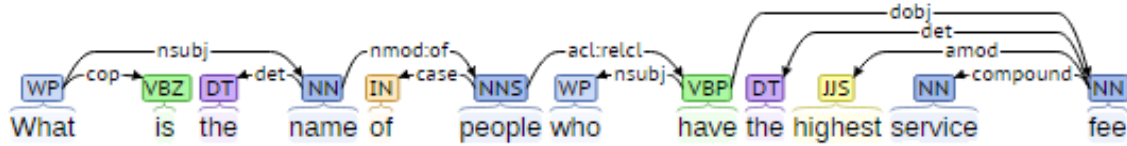


Figure 2.3: Example on Word Dependencies Generated by Stanford Dependency Parser

Stanford Dependency Parser is the best among the available parsers [15, 38]. However, the results presented in [12] are related to the old Stanford dependency parser; the new dependency parser of Stanford uses ANN which has improved its efficiency.

Name Entity Recognition

Name Entity recognition (NER) is one of the subfield in NLP. The main usage of NLP is to find the type of a specific word in a given sentence [24]. NER can be used in many domains ranging from health to business, where each has its specific domain reference that can be used [24, 20]. For example, the type of the word “Abed” is person. We used NER to identify unmapped words for better accuracy. Many algorithms have been used to detect words types, such as Hidden Markov Model, Maximum Entropy markup Model and Conditional Random Field. CRF has demonstrated its high accuracy and efficiency for detecting words types [24]. Section 2.1.1 demonstrated the effectiveness of CRF compared to other available algorithms.

Dates Phrase Extractor

Time expression extractor is another field in NLP. Many algorithms were used to extract time expressions from text. Consider this sentence as an example to illustrate the process: “last year Chris went to Peru”. Here, the combination of two words “Last year” is considered as a time expression. However, in text this kind of time expression should be replaced by the actual date to be used in the mapping.

Several parsers have been proposed to extract date expressions from sentences. However, none of them works perfectly. The result reported in [23] and [14] show the effectiveness of

SUTime parser over other date expression extractors. SUTIME is based on regular expressions that help in detecting date text phrases. Stanford NLP tool uses SUTime library for extracting time expressions from text [13]. Since we decided to use Stanford NLP tool in our research and since SUTime has proved its effectiveness, we decided to use SUTime library for date phrase extraction. Some modification were added to SUTime library which we will describe in details in Chapter 3.

2.2 Related Work

Natural language processing is used to analyze text with the target to extract useful information. Researchers have been working extensively on developing algorithms that can handle this task. Moreover, many approaches have been proposed to include fuzziness in SQL, e.g., [32, 4, 8, 39, 6, 9]. The work described in this thesis focuses on translating users' requests into valid SQL queries, regardless whether the requests are fuzzy or not. In this section, we focus on approaches that utilize NLP and fuzziness while translating users' requests to SQL queries; we also discuss how our work is different from other approaches described in the literature.

Several approaches have been proposed to handle the mapping of users' requests to relational databases using NLP and data mining techniques. For instance, the work proposed by Li et al. [42] discusses an approach developed to analyze users requests, not necessarily expressed in question format. Stanford Dependency parser and similarity functions were used to map words into database tables and columns. However, in our approach we introduce the use of NER which help in mapping unidentified words without the need to bother users. In addition, we use synonyms, pluralization and singularization for word mapping. We also handle fuzzy requests and aggregation functions which are features that the approach of Li et al. [42] lacks. Table 2.3 includes a detailed comparison between our approach and that of Li et al. [42].

Table 2.3: Comparing our Approach with the Approach of Li et al. [42]

features	[42]	Our Approach
Used Questions	No	YES
Stanford Dependency Parser	YES	YES
Similarity	YES	YES
Synonyms	NO	YES
Prefix and Suffix	NO	YES
NER	NO	YES
Fuzziness	NO	YES
Singularization and Pluralization	NO	YES
Aggregation Functions	NO	YES
Date Expression Detector	NO	YES
Transform Numbers in Words to Numeric Values	NO	YES
Spell Checking	NO	YES
Handling Ambiguity	YES	YES

Another approach that analyzes user questions to retrieve data from relational databases was proposed by Giordani et al.[28]. In this approach, the authors used Stanford dependency parser and they applied direct mapping with the database schema. They reported 81% accuracy. However, our system differs from their solution by the way we map words to the database and the kind of queries we handle. Table 2.4 shows a detailed comparison between our approach and the approach of Giordani et al. [28]. Only one question was given as proof of their parsing capability which is considered to be not enough to demonstrate the power of their work.

Many researchers have been working on building effective systems for handling user queries. However, not all of them are capable of outputting accurate results. In other words, their mapping process is different and lacks some features which could help in improving accuracy. For instance, the work done by Nihalami et al. [54] has some limitations, such as number of inner joins it can handle. In addition, the accuracy reported in their paper is low and the mapping process depends on direct matching. Another system was proposed by Samsonova et al. [59]; further, Little et al. [43] is another example that uses NLP to parse and map users request to SQL. The system proposed by Little et al. [43] provides

Table 2.4: Comparing our Approach with the Approach of Giordani et al. [28]

features	[28]	Our Approach
Used Questions	YES	YES
Stanford Dependency Parser	YES	YES
Similarity	YES	YES
Synonyms	YES	YES
Prefix and Suffix	NO	YES
NER	NO	YES
Fuzziness	NO	YES
Singularization and Pluralization	NO	YES
Aggregation Functions	YES	YES
Date Expression Detector	NO	YES
Transform Numbers in Words to Numeric Values	NO	YES
Spell Checking	NO	YES
Handling Ambiguity	NO	YES

suggestions to users when writing their queries. Our system doesn't support such type of feature. However, fuzziness is not handled in the system proposed by Little et al. [43]. Also their mapping process is different from the one that we are using in this thesis, since they only rely on direct matching. Other systems have been implemented such as those described in [61, 54, 53, 57]. But they also don't handle fuzziness, and they depend on direct mapping without adding the extra checking we are applying in our system.

The approach of Agrawal et al. [3] is very close to ours. Stanford Parser was used to get dependencies between words along with their POS tags. They also worked on intractable tokens and replaced them with meaningful date/time that can be used in the mapping, e.g., the word "midnight" can be replaced by an actual time which will be more meaningful in the mapping process. However, Agrawal et al. [3] didn't use NER and synonyms, though both might help in the mapping process; also they didn't handle imprecision in user questions. Table 2.5 shows a detailed comparison between our approach and the one proposed by Agrawal et al. [3].

To the best of our knowledge, there is no system described in the literature that translate users' requests the way done in this thesis. Our mapping process includes various techniques,

Table 2.5: Comparing Our Approach with the Approach of Agrawal et al. [3]

features	[3]	Our Approach
Used Questions	YES	YES
Stanford Dependency Parser	YES	YES
Similarity	YES	YES
Synonyms	YES	YES
Prefix and Suffix	NO	YES
NER	NO	YES
Fuzziness	NO	YES
Singularization and Pluralization	NO	YES
Aggregation Functions	NO	YES
Date Expression Detector	NO	YES
Transform Numbers in Words to Numeric Values	NO	YES
Spell Checking	NO	YES

such as dependencies, word refinement, synonyms and NER. The combination of these techniques will help in better mapping the results. We also handle nested queries which most systems don't do. In addition, we handle questions that include aggregation functions, such as maximum, average, count, etc. All these features integrated in our system will help in handling more diverse queries which cannot be handled by other systems described in the literature. We don't tight users' hands to a limited number of conditions in a query. We also avoid limiting the number of tables that users can refer to in their questions, and hence will be used for data retrieval. Chapter 3 describes the proposed solution in details and highlights the steps for analyzing users' requests to retrieve the intended results from the database. The accuracy of our results and performance measures are reported in Chapter 4.

Chapter 3

Proposed Solution and Methodology

Data is collected every day by different types of devices and applications. The collected data may be analyzed to discover some hidden information which may help in decision making. For example, a company might use data related to sales to find out when its products are bought the most. Such type of analysis will help companies to improve their marketing plans.

Despite the growing interest in NoSQL, the main data repositories are still RDBMS based and hence require knowing SQL for data access and retrieval. However, the number of users interested in retrieving data from existing data repositories is rapidly increasing. Moreover, the user group is characterized by diversity in background and capabilities ranging from naive users to professionals. Forcing all user groups to learn and master SQL may be classified as a harsh expectation. Instead, it is more reasonable and practical to develop something that could allow all types of users to easily and smoothly access existing data repositories, including RDBMS's. In other words, in most organizations data is usually stored in RDBMS's which can be only accessed by knowledgeable database experts who know SQL. For instance, if a manager wants to have a specific report then he/she should wait for the database expert to write the specified query, design the report and send it to the manager. All this process is time consuming and expert dependent. This issue and the like motivated us to build a system that will help naive users to retrieve from RDBMS's the data they want without the need to learn SQL.

This chapter will discuss the proposed system in details. The tools used in realizing the system are described. We also describe the functionality of every component in the system. In Chapter 4, we elaborate on the efficiency of our system.

3.1 Development Tools

Visual studio 2013 and C# programming language were used in developing the proposed system.. C# is a very powerful and widely used programming language. We built the natural language interface (NLI) as a desktop application, i.e., it runs on a single machine, unlike Web applications which can be accessed from anywhere using Internet or Intranet.

3.2 User Requirements

The purpose of this research is to develop a system that will allow naive users to retrieve data from a relational database. We tried our best to make the system as simple as possible. However, a user is expected to acquire some minimum computer literacy before trying our system. Fortunately, it is assumed that every person do have such knowledge by default especially in this era where most people have mobile devices.

Users might need some help for configuring database credentials in our system. However, we think users with basic computer literacy will be able to set the configuration on their own using the automated helper that accompanies our system. The project configuration is described in Section 3.4.2. Figure 3.1 depicts the interaction between users and the application as well as between the application and the database server.

3.3 Tools Used

Research groups have developed several tools for text analysis. We decided to use an existing tool for the text analysis component of the developed system. Actually, this is a very common practice in the research community. The main motivation for using existing tools is that they have been well tested and hence adopting an existing tool will save time and effort.

Three existing tools have been integrated into the developed system to help in analyzing users' requests. These are: Stanford NLP parser, Hunspell spell checker, and the pluraliza-

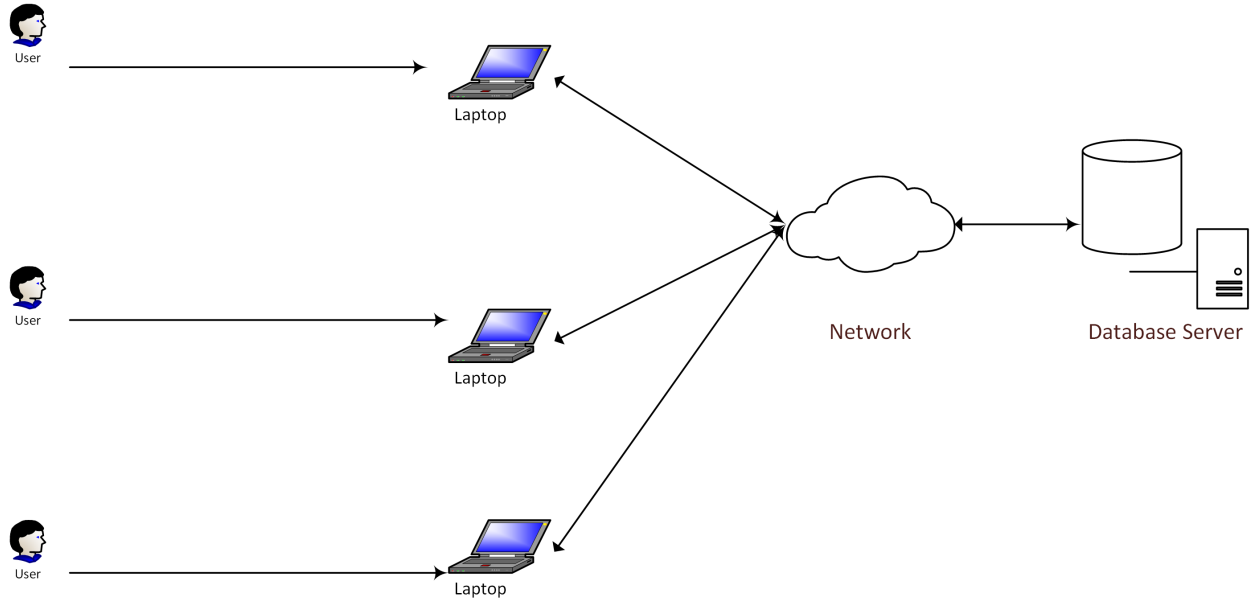


Figure 3.1: The communication between the System Components

tion. They are briefly described in this section.

3.3.1 Stanford NLP Parser

Stanford NLP parser is one of the best parsers available for dependency extraction and name entity recognition. SNLP is an open source tool under GNU General public license. It can be used for research purposes. SNLP have many features which are shown in Figure 3.2. However we will not use all these features in our research.

Our analysis mainly uses three features that are provided by SNLP. These are dependency parser, name entity recognizer (NER) and temporary tagger which can help in detecting date expressions. We already discussed the efficiency of the selected tools in Chapter 2. We used models to train our data in order to get better results. These models are mainly provided by Stanford and they are case insensitive. All models are loaded when the developed system starts; this is way better than loading the models every time a user runs a query.

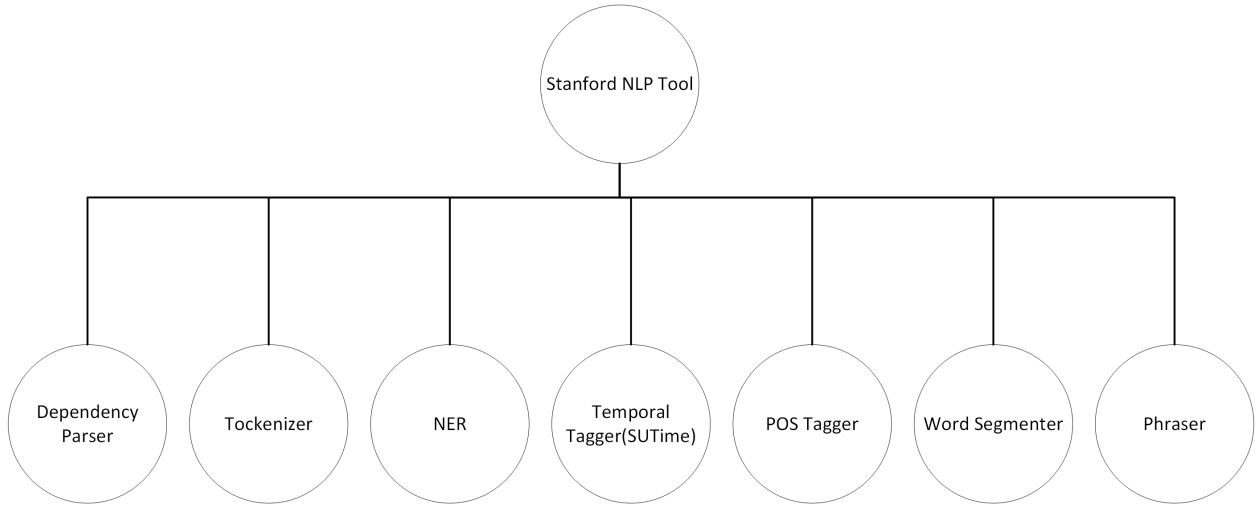


Figure 3.2: Features of the Stanford NLP Tool

SNLP Relation Extractor

The main task for SNLP Relation Extractor is to find the relationship between words in a given sentence. It identifies the relationship between words and detects POS tags for each word in a sentence. Detecting POS in a sentence will help in getting better accuracy when extracting dependencies. Each relation has a type which is also detected by SNLP. For example, assume it is required to get the relationship between words in the following sentence: “What are names of people who have highest service fee and are living in New York”, each word will have a POS tagger as shown in Table 3.1.

Each word in Table 3.1 has an ID which helps in differentiating it from other similar words in the sentence. Moreover, each relationship has a type which is reported in Table 3.2. Word ID in Table 3.2 is mapped to “ID” column in Table 3.1. Relation types can be used to determine if the relationship between two words is compliment, negation, etc. This type of relationship is helpful while building a query. The full meaning of POS Tags and Relation types can be found in [18].

The dependency parser is based on a neural network classifier which should be trained

first in order to return accurate results. We used the model developed by Stanford; it is case insensitive. The name of Stanford model is “englishPCFG.caseless”. Stanford NLP tool supports multiple languages, including Chinese and Arabic, but only English is used in this thesis.

Table 3.1: Word POS Tagger Example

ID	Word	Start position	End Position	POS	NER
1	what	0	4	WP	o
2	are	5	7	VBZ	o
3	the	8	11	DT	o
4	names	12	16	NN	o
5	of	17	19	IN	o
6	people	20	26	NNS	o
7	who	27	30	WP	o
8	have	31	35	VBP	o
9	the	36	39	DT	o
10	highest	40	47	JJS	o
11	service	48	55	NN	o
12	fee	56	59	NN	o
13	and	60	63	CC	o
14	are	64	67	VBP	o
15	living	68	74	VBG	o
16	in	75	77	IN	o
17	New	78	81	NNP	Location
18	York	82	86	NNP	Location

SNLP Name Entity Recognizer

Name entity recognition is another feature we used to analyze users’ requests. After excluding stop words, unmapped words are those which cannot be mapped to and matched with existing tables or columns in the database. A detailed explanation about the mapping process is included in Section 3.5.5. If a type is identified for any of the unmapped words then we check if there is any table or column that matches the specified type.

For our research, we used four types of classes. These classes are: Location, Organization, Person, and Date. The Date type is used only for detecting date expression in a sentence. Adding more classes has been left as a future work. The model used for SNLP NER is

Table 3.2: The Relation between Words in a Sentence

First Word	First Word ID	Second Word	Second Word ID	Relation Type
Root	0	what	1	root
what	1	is	2	COP
name	4	the	3	DET
what	1	name	4	nsubj
people	6	of	5	case
name	4	people	6	nmod:of
have	8	who	7	nsubj
living	15	who	7	nsubj
people	6	have	8	acl:relcl
fee	12	the	9	det
fee	12	highest	10	amod
fee	12	service	11	compound
have	8	fee	12	dobj
have	8	and	13	cc
living	15	are	14	aux
people	6	living	15	acl:relcl
have	8	living	15	conj:and
york	18	IN	16	case
york	18	new	17	compound
living	15	york	18	nnmod:in

called “english.conll.4class.caseless.distsim”. This model is case insensitive, i.e., it recognizes “Chris” and “chris” as the same word. Using the sentence “What are names of people who have highest service fee and live in New York” with its analysis result reported in Table 3.1, shows that the last two words, namely “New York” were recognized as constituting name of a city instead of treating them as two separate words. A word is neglected if it does not fall in any of the four predefined classes.

SNLP Time Expression Extractor

SNLP Tool contains another important feature that recognizes a date expressed in words. This feature helps us in detecting and replacing dates with corresponding numerical expressions. To illustrate the process, consider the sentence “What are reviews that were completed last winter”. Here, there is an implicit date expressed as “last winter”. Table 3.3 shows how this date expression was detected. Based on the outcome, the phrase will be replaced with

Table 3.3: Date Extraction Example

ID	Word	Start Position	End Position	NER
1	What	0	4	o
2	are	5	8	o
3	the	9	12	o
4	reviews	20	23	o
5	that	21	25	o
6	were	26	30	o
7	made	31	35	o
8	in	36	38	o
9	the	39	42	o
10	last	43	47	Date
11	Winter	48	54	Date

an approximate actual date. Some phrases are detected as date expressions but Stanford tool does not return an actual corresponding date for them. We extended the coverage of our implementation to have these cases handled by the developed system. Multiple files were used for this purpose, in addition to “english-bidirectional-distsim.tagger” tagger.

3.3.2 Hunspell Spell Checker

As the developed system is intended to expand the user group to include naive users, it is expected to have some typing errors in the questions submitted by the users. This type of mistakes might negatively affect the analysis. However, we cover this by integrating Hunspell Spell Checker tool into the developed system. It is a very effective tool for detecting spelling mistakes in a given text. It also has its own model that should be loaded in advance. However, the suggestions given by Hunspell spell checker are not always accurate. For instance, sometimes users might intentionally write some words which may not exist in the list covered by Hunspell. To handle this, a text-box has been added to allow users to confirm the words they want to keep untouched despite having them not recognized by the checker. These additional words will be saved for later usage. Figure 3.3 shows an example which demonstrates how the spell checker displays suggestions for users as well as the text-box that allows users to confirm extra words.

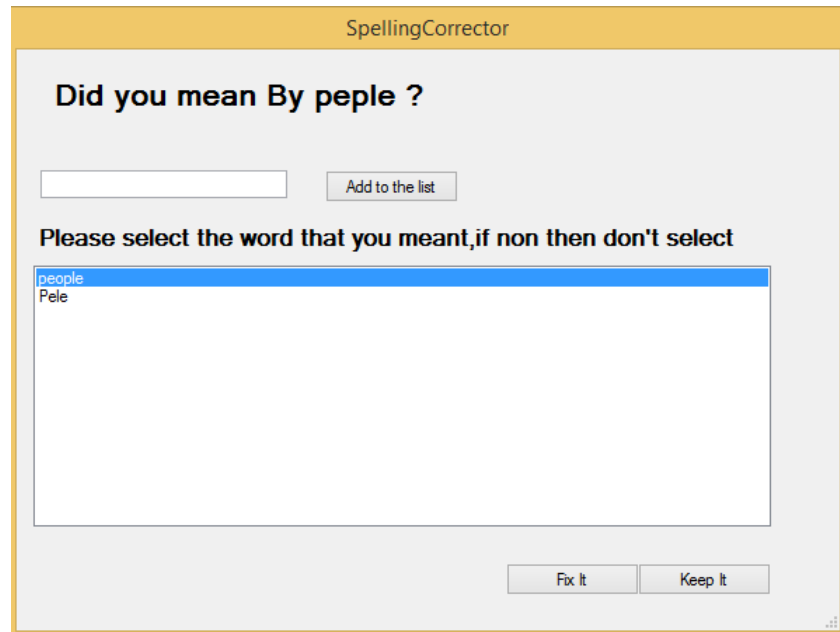


Figure 3.3: Suggestions Produced by the Spelling Checker

3.3.3 The Pluralization Library

The pluralization library is a .Net library which is used to check if a given word is in plural or singular format. It is also possible to use the tool to get pluralization or singularization of a specific word. This will help in mapping words to database tables or columns by not just checking for exact match. For instance, it is possible to have in the database a table named “person”, while a user may use the word ”people” in his/her question; the two words are matched after referring to the library. This tool has its own dictionaries and hence there is no need for any model to be loaded. Actually, working with pluralization and singularization of a word is a straight forward concept.

3.4 Project Configuration

In general, a software system should be configured before it is used. We tried to make the configuration process of the developed system as simple as possible, keeping in mind that a large number of users of the system are expected to lack advanced computer literacy. The

configuration process involves creating a project, fuzzy domains and ranges, word synonyms, and prefix-suffix words that are used in mapping words to tables or columns.

Various requirements of the configuration process are saved in an XML file on user's local machine. There are two alternatives for saving data in a file: (1) using XML, and (2) using Json parsers. The reason behind selecting XML over JSon is the availability of .NET XML parser which parses and loads objects in a faster way compared to Json parers. We decided to store configuration data in a file rather than the database in order to avoid modifying the database schema. Thus, the database schema is not modified during the analysis. All the information expected to be needed in the future will be saved in XML file.

3.4.1 Creating and Loading Project

Each created project corresponds to a database server. Users might need to create for the same database multiple projects. Each created project has its own information, including fuzzy domains, database schema, etc. Further, users can utilize the developed system to connect to multiple relational database servers such as Microsoft SQL Server, Oracle, MySql, etc. They can always save their work in XML file and next time they start the application they can just load the XML file and proceed to retrieve information from the database.

3.4.2 Database Configuration

Creating a project requires a user to know the database server he/she is connecting to. Users with enough expertise can just insert the connection string directly. Other users should refer to the helper form shown in Figure 3.3. The main purpose of the helper form is to show users the information needed to be able to connect to a database. After a user enters server type (e.g., Microsoft SQL,Oracle or MySQL), server IP, username and password (if available), the database will be auto populated from the selected server. A user should select the database that he/she should connect to. After database settings are entered successfully the system will get the database schema of the selected database.

Figure 3.4: Database Configuration Helper Form

The database schema includes all information about tables in the given database, including their columns information. Each column has a type. To remove ambiguity for users, we replace SQL columns types with more general yet easier to understand ones. For example, instead of putting `nvarchar(50)` or `varchar(50)` we just indicate column type as text. We show to users the set of tables along with their columns information because we think this will help them in building their questions.

Relationships between tables are determined by Foreign-Primary keys connections. Each table has a predefined primary key which is a minimal set of attributes that uniquely identifies each record in the table. On the other hand, a foreign key represents a specific primary key to simulate a predefined relationship. A primary key may have $n \geq 0$ corresponding foreign keys. A foreign key may exist in the same table with its corresponding primary key and this simulates a self reference type of relationship. For instance, consider two tables, namely persons and services and assume ID is the primary key of persons table, a self reference may be added to “persons” table to show the children of each person; this is simulated by duplicating ID in “persons” table to have the original primary key and its representative foreign key. Further, assuming that each person should have a set of services would lead to have person ID added to “services” table as a foreign key. This kind of information can be

retrieved from the database schema.

3.4.3 Create Fuzzy Domain and Ranges Dictionary

One of the main key features in the proposed system is the ability to handle fuzziness in users' questions. Section 3.6 talks in details about the fuzzy scenarios built in our system. Generally, users can specify fuzzy domains and their ranges for each column. For example, the column "Fee" in table "Services" would have 4 fuzzy domains; these are: Expensive, normal, average and low. Users have the option to define ranges for each domain. Once these fuzzy domains are defined, every condition in the user question will be checked to see if it contains a fuzzy domain or not. In our system, we allow users to specify domains for numeric columns only.

3.4.4 Create Word Synonyms Dictionary

Handling word synonyms is another feature supported by our system to improve accuracy. Users can define a set of synonyms for each table or column. When a synonym word is indicated in the text, the word will be mapped to its corresponding table or column. For instance, the column "fee" might have the following synonyms "Service fee", "Price" and "payment", so that when one of these is detected in a sentence we directly use its associated column and assign to it a score of 100. All this information will be saved in XML file to allow users to update or modify them any time.

3.4.5 Create Database Prefix and Suffix Dictionary

Tables in a database may have prefixes and/or suffixes. Suffixes and prefixes might lead to mismatching between tables or columns on one side and words on the other side. These words should be removed during the mapping process in order to get more accurate matching with database tables or columns. We created our own list of prefixes and suffixes, but we also give users the option to modify the list. Table 3.4 shows a general list of words that can

be used as suffix or prefix for tables and columns in the database.

Table 3.4: General Database Schema Prefixes and Suffixes

Suffix/prefix	Meaning
Tbl	table
New	New Table
LKUP	Look Up Table
Tx	Text field
Dt	Date Type
API	Application Program Interface
ASPnet	represents the ASP.Net tool
Tmp	Temporary table
TLKP	Look Up Table

3.5 System Architecture

A number of predefined steps should be applied to ensure high quality in the analysis of users requests. These steps are given in Figure 3.5 which shows the flowchart that depicts the flow of control in the system from receiving a user question to delivering the result back to the user. A given request (question) submitted by a user goes through the following steps: refining the request, mapping keywords to tables/columns, building SQL statement, resolving fuzziness, and seeking feedback from users. Here it is worth mentioning that loading a project is not considered part of the flowchart shown in Figure 3.5. Instead it is assumed that a user configures database connection and the related information once before initiating a session which may include submitting a number of requests. Finally, the various steps shown in Figure 3.5 are described in details in the rest of this section.

3.5.1 User Input

In general, it is not possible to keep track of and evaluate the background of the wide range of users, the level of their computer literacy, and their ability to write SQL queries. This has been the main motivation to have commercial DBMS's apply optimization on every SQL

query submitted by a user. The query passes through a number of steps that produce what is expected to be an equivalent optimized query. Such a process may be just a validation for queries written by professionals as their original queries are expected to be similar to the optimized query because they are expected to keep in mind the optimization of time and space utilization related to their queries.

The developed system has been set in a way such that it accepts from users queries expressed in SQL or as questions which are written in English and follow a certain format. For questions, the system takes responsibility of producing the corresponding SQL statement; and providing this facility will eliminate the risk of having some users submitting inaccurate SQL statements in case they do not have any other choice. Further, the current version of the developed system is capable of handling only WH (Which, What and How many) types of questions. Each WH question may target the retrieval of a different type of data. We will talk about each type in Section 3.5.7. As other researchers who have tackled the same problem did, we avoid giving users full freedom in constructing their questions because it will be hard to parse a sentence that does not follow a predefined structure, and hence it will not be possible to produce results that satisfy users. Losing users' trust in the system will diminish its acceptability and hence the main target of providing the new interface will not be satisfied.

Though it is preferred to have queries coded as questions written in English, the system still provides the option to submit queries in SQL. The latter option is mostly used by database experts who are even given opportunity to build fuzzy SQL queries which are smoothly handled and transferred by our system. Coding fuzzy SQL queries has been made possible because as depicted in Figure 3.5, fuzziness is handled after the SQL statement becomes available regardless whether it is submitted by a user or derived by the system from a user's question.

Regular Expressions are used to check whether a user input is an SQL query or not. We

check if the user input has the Select and the From clauses then it is interpreted as an SQL statement, otherwise it is considered as an invalid SQL query. For the latter case, i.e., for queries which have not been identified as SQL statements, we check if the user input is in question format that can be processed; otherwise the system asks the user to enter a valid question. We mainly allow three types of WH questions; these are: "Which", "What" and "How many". More detailed information about these types of questions will be covered in Section 3.5.7.

3.5.2 Input Refinement

Some refinement should be applied on the user input before starting the analysis. This includes spell check, replacing numbers expressed in words by corresponding numeric values, identifying dates expressed in words in order to replace them with corresponding actual date, and fixing negation in a sentence. All these should be done before getting the dependency between words so that we can get a better dependency information.

Spell Checking

Spell checking is used to make sure that all words in a question are what the user really meant and no typing mistakes exist. We are using Hunspell tool (Section 3.3.2) to check spelling mistakes and to give suggestions for corrections. However, Hunspell tool does not always give correct suggestions. Thus, we give users the opportunity to add correct words in case they were not found in the list as discussed in Section 3.3.

After completing the spell checking process, the sentence will be updated with the correct words and new values (in case they were determined); then the user will be able to see the modified sentence.

Changing Text Based Numbers to Numeric Values

Numeric values can be expressed using two formats, either text based format such as "Five" or as digits such as "5". Using the first format will cause some difficulties while getting the

Table 3.5: Examples of Text Based Numbers Converted to Numeric Values

Text Based Numeric WordsWord	Numeric Value
five	5
one hundred and five	105
two millions and three hundred	2000300
one million, one hundred and one thousand, two hundred and thirty-one	1101231
one million, one hundred and one thousand, 5 hundred and thirty-one	1101531
one hundred and one thousand	101000

dependency and building the query. For example, consider the following sentence: “What is the number of people who have a service fee greater than five hundred and twelve”. When getting the relation between words, SNLP will detect the expression “Five hundred and twelve” as number but it will deal with each word alone; each word will be mapped to the corresponding digit(s). However, including in the sentence the numeric value ”512” instead of the expression “Five hundred and twelve” will lead to a different outcome. Namely, the value ”512” will be mapped to the word ”fee” directly without splitting. Another reason for converting text based numbers to numeric values is that when building the “where” condition in SQL statement we should use with the condition operator numeric values instead of text base values.

We convert all text base numbers to numeric values. Table 3.5 shows a list of text based numbers and their representation as numeric values. Algorithm 1 is invoked for replacing text based numbers with numeric values. The input parameters are a list of constituents of text based number and the original sentence in which text based number will be replaced with its numeric value. Extracting the expression of the text based number from a sentence is done by another function.

Fixing Negation

Fixing negation in sentences is another refinement done on the user input. This kind of refinement is not displayed for the user. It is rather intended for internal usage by the system. Fixing negation includes replacement of negation words with corresponding full

Algorithm 1 Parsing Text Based Numbers to Numeric Values

Require: $a! = ""$ and $numExpressions.length > 0$

```
1:  $numDictionary = ["one" : 1, \dots "nine" : 9, "ten" : 10, \dots "nineteen" : 19, "twenty" : 20, \dots "ninety" : 90]$ 
2:  $uniteDictionary = ["hundred" : 100, \dots]$ 
3: for  $sentence \in numExpressions$  do
4:    $words = sentence.split('')$ 
5:    $numVal = 0$ 
6:   for  $w \in words$  do
7:     if  $w$  is numeric and  $w$  not in  $uniteDictionary$  then
8:        $numVal = numVal + w$ 
9:     else if  $w$  is numeric and  $w$  in  $uniteDictionary$  then
10:       $numVal = numVal * w$ 
11:     else if  $w$  is not numeric and  $w$  in  $numDictionary$  then
12:       $numVal = numVal + numDictionary[w]$ 
13:     else
14:       $numVal = numVal * numDictionary[w]$ 
15:     end if
16:   end for
17:    $a = \text{replace } w \text{ in } a \text{ with } numVal$ 
18: end for
19: return  $a$ 
```

words without abbreviation. For example, the word "don't" is transformed to "do not". Doing this will help us when building the "where" condition. Here it is important to mention that the developed system does not support types of negation that will lead to set difference. We rather support negation at the record level, i.e., to filter records that do not satisfy a given condition. The former has been left as future work. Finally, Table 3.6 includes some examples of words that are used to express negation in sentences.

Replacing Date Expressions with the Actual Date

For this refinement, we get help of Stanford SU library. The main task of this library is to detect dates expression in a sentences and replace them with corresponding actual dates. However, this library doesn't transform all the detected date related expressions to actual dates. In other words, Stanford SU library is capable of handling mostly straight forward conversions and it skips some complicated cases like "Winter". Thus, we had to expand

Table 3.6: Examples on Transforming Abbreviated Negated Words to Full Word Representation

Negated Word	Full Word
Don't	Do not
Doesn't	Does not
Wasn't	Was not
Didn't	Did not
Weren't	Were not
Haven't	Have not

the coverage of our system to deal with complex cases unconverted by Stanford SU library. Explicitly speaking, we developed an algorithm for handling expressions for which Stanford SU library doesn't produce actual dates. Table 3.7 gives a list of date expressions, how they were parsed, and whether by Stanford SU library (Stanford for short) or by our algorithm; we assumed current data is 11/08/2015 (mm/dd/yyyy).

Table 3.7: Examples on Transforming Date Expressions to Actual Dates

Date Expression	Actual date	Notes
Last Monday	11/02/2015	Parsed by Stanford
Last Year	1/1/2014	Parsed by Stanford
Last 4 winters	12/01/2011	This is detected by Stanford but parsed by our system
Coming 4 winters	12/01/2019	This is detected by Stanford but parsed by our system
Last 4 days	11/04/2015	This is detected by Stanford but parsed by our system
Last 6 months	05/08/2015	Parsed by Stanford
Coming 4 Days	11/12/2015	This is detected by Stanford but parsed by our system

3.5.3 Get Word Dependencies

After completing all refinements for the user input, we get relationships between words. This will help in detecting which values are related to which columns while building the “where” condition in SQL statement; in addition this will help in figuring out columns that should be retrieved and in case they were specified. The latter columns are part of the “select” clause of the SQL statement.

Stanford NLP tool is used for getting the dependencies. This tool helps in getting re-

relationships between words along with the relationship type. We get dependencies in a tree format; we then parse them into a dictionary by grouping each word with words that depend on it. This will help us in faster checking for words that are related to each others.

3.5.4 Eliminate Unnecessary Words and Punctuations

Punctuation and unnecessary words are removed from the given sentence before starting to match words with database tables and columns. This is not done before getting dependencies as it might affect the output of Stanford Dependency Parser. Also unnecessary words should be removed from the sentence. Keeping unnecessary words in the sentence will result in consuming extra time for the mapping process.

We usually have a predefined list of unnecessary words that we usually remove from a sentence. However, this list might change depending on the database application from which the data is to be retrieved because some words may be relevant for one application and irrelevant for another. For example, the word “From” is considered to be unnecessary word for a student database, but it might be very relevant for an airlines reservation database where “From” may be used as a column name. For the latter case and the like, we don’t remove from the sentence words which are expected to be relevant due to the investigated database application. Users also have the option to expand the list of unnecessary words. Table 3.8 contains a list of words that are usually removed from a sentence before starting the mapping process.

Moreover, we also remove duplicate words in a sentence. We don’t need the same word to be mapped twice as this might consume more time without affecting the output. All these modifications are not made to the original sentence; instead changes are done on a copy of the original sentence.

Table 3.8: List of Unnecessary Words that can be Removed from a Sentence

This	The	Equal
That	Which	And
New	Previous	Best
With	Coming	Do
Not	Last	Does
Between	Like	Of
By	To	Often
Made	Too	You

3.5.5 Map Words to the Database Schema

Here, we concentrate on identifying the main words to be mapped to the database schema. Numbers and dates are ignored in the mapping process because these should be handled in the condition. Each word in the modified sentence is compared with database tables. The mapping involves the following:

- pluralization or singularization of a word and checking if there is a match.
- checking if the word is a synonym of a specific table or column.
- removing prefixes and suffixes from the names of tables and columns and checking if there is a match.
- checking if the word was already mapped to a table or not.

If one of the specified conditions returns true then the word will be added to a dictionary and its relationship score with the corresponding table will be 100. Partial scoring is given to words that don't have exact match with existing tables. For example, assume there are two tables one named "Service_Reviews" and another called " Services", then the word "Services" will have a score of 100 with the second table and its score is 50 with the first table. The scoring system we created is a straight forward process that will help us in filtering tables we need to get data from.

Mapped words are also removed from the sentence to avoid any trial to map them again. Then the remaining words will be mapped to columns of existing tables by applying the same process described above to handle tables. Each word mapped to a column is removed from further consideration. In case of any remaining words, other than dates and numbers, the user will be consulted to specify whether these words are necessary or not, if the user decides that the remaining words should be included in the analysis then the extra step described in Section 3.5.6 will be executed.

3.5.6 Tables and Columns Selection

The process of mapping words to tables and columns results in two lists, the first is a dictionary of words with their mapped tables and associated scores for each relationship and the other is a dictionary of words with corresponding columns and associated relationship scores. These lists should be filtered to find: (1) the tables to be included in the “From” clause of the SQL statement (these are in general tables from which the target data is to be retrieved) and (2) the columns that should appear in the “Select” and “Where” clauses of the SQL statement.

We iterate over all the keys in a dictionary. These should be the mapped to the words identified by the process described above. We first select tables that have a relationship score of 100. Each word is expected to have at most one corresponding table of score 100. However, it is possible to have some words which do not have a corresponding table associated with a score of 100. Each such word is associated with a corresponding table that has the highest score. A number of scenarios could be applied to narrow down the association to a single table in case the latter check identifies more than one candidate table sharing the same highest score. The first scenario gives priority to the table which has direct connection with the largest number of tables identified with a high score, preferably 100. The second choice will be to consider the table which has some of its columns already identified as part of the query with a 100 score. The last chance is to consult the user who is expect to know his

target result and hence should be able to select a single table from the set of all candidates identified by the system. The same process is repeated for matching columns. However, the system considers only columns which have a 100 score or which are linked to one of the tables filtered by the process described above.

At the end of the matching process, it is possible to have remaining unmapped words. These words are handled by first prompting the user who is expected to suggest whether these words are necessary for the analysis. Accordingly, each necessary word is linked to the query by applying the following two steps in order:

- Use NER to check the word type; if a type was returned then check which table matches with the type
- Check for each table's neighbors (directly related tables) whether any of their columns of type text contains the candidate word as a value

The two tests are applied in order. In case one of them retrieves a table then the table will be added to the filtered list to be used in forming the SQL query. Here it is important to make sure that the returned table has a relationship with other tables that exist in the filtered list. We know it is not very practical to apply the last step which requires checking the content of specific columns. However, we are forced to go this way to avoid omitting possibly important word(s). We improve the performance by maintaining inverted lists of the key words available in the content of each column. Any words which are left unmatched after applying all the test scenarios will be reported to the user as not considered in constructing the SQL query to be executed.

3.5.7 Question's Return Type

Users may ask questions in different ways so that they can get the information they need. Some questions may involve the retrieval of a single numeric value while others may involve retrieving a set of records. For instance, some questions submitted by users may involve the

usage of aggregation functions while others may target the retrieval of a list of records. In our research, we benefit from the advantage of knowing the question structure which would lead to the question type, and hence will reveal what should be retrieved to the inquiring user. Table 3.9 shows a set of WH phrases and their expected results. Implementing other question types has been left as a future work.

Table 3.9: Examples of WH Phrases and their Expected Results

WH Phrase	Expected Results
What is the Number	Single numeric value that represents the count per a retrieved result
What is the count	Single numeric value that represents the count per a retrieved result
What is the average	Single numeric value that represents the average value of a numeric column
What is the lowest	Single numeric value that represents the lowest value of a numeric column
What is the highest	Single numeric value that represents the highest value of a numeric column
How many	Single numeric value that represents the count per a retrieved result
Which <table or column name>	A list of records
What is the <table/column name>	A list of records

3.5.8 Building the SQL Statement

After filtering the selected tables and columns, it is time to build the SQL query, which consists of three main clauses and some other optional clauses as mentioned in Chapter 1. These clauses should follow a certain structure. We should make sure that each clause we build is valid and error free. In this section, we describe how our system builds the Select, From and Where clauses. First each of the three clauses is separately formed and then they are combined together into SQL statement to be executed. Finally, this section includes also some examples that show how the developed system handles nested conditions.

Constructing the Select Clause

In general, the Select clause includes a list of columns that should be retrieved. Since we have multiple types of questions, it is necessary to first check the type of the question submitted by the user. Knowing question type will guide the construction of the “Select”

clause. For instance, some types of questions might involve the selection of columns and other types might require computing certain values from one or more columns. The latter process involves applying some aggregation functions which are predefined in SQL environment.

Table 3.10 lists some of the aggregation functions used in SQL such as Count, Min, Max and Average. Some of these aggregation functions don't require a column name in general, such as Count where we can use "*" which means count all rows that satisfy the query. However, other aggregation functions require explicit specification of a numeric column; then values in the specified column will be used to calculate or determine the target value, e.g., Min, Max, Average, etc. Table 3.10 lists some WH phrases together with possible corresponding content of the "Select" clause.

Table 3.10: Examples on WH Phrases and their Expected Parsing

WH Phrase	Select Query
What is the number	Select Count(*)
What is the count	Select Count(*)
What is the average	Select AVG(columnName)
What is the lowest	Select Min(columnName)
What is the highest	Select max(columnName)
How many	Select Count(*)
What is the <columnName>	Select <columnName>

Some questions might target the selection of multiple columns. This can be determined from the words that were mapped to columns and their specific locations in the question. When dealing with multiple tables in a query, it is possible for one or more target columns to exist in more than one of the tables considered in the query. This is resolved by adding aliases to SQL queries to be constructed. As an alternative, it is possible to write the name of a table followed by "." (period) as prefix for the name(s) of its column(s) used in queries. For example, when name of a person is a target column from the person table, the column could appear in the query as "person.name".

Columns in the filtered list are checked to determine the ones that should be in the Select clause. These are mostly columns specified towards the beginning of the question. Having

only a table name specified towards the head of the question would be interpreted as no specific column is the target. Accordingly, all columns of the specified table will be retrieved and this is expressed by having only the asterisk (“*”) in the “Select” clause.

However, having in a given question some aggregate functions without corresponding column(s) explicitly specified is considered as an error. Users will be advised to revise the question by addressing the specific error message. Finally, it is important to note that columns used in the filtering condition (i.e., in the “Where” clause) will not appear in the “Select” clause unless they are explicitly specified as target columns by listing them close to the head of the question. In other words, it is possible to have the same column in both the “Select” and the “Where” clauses. Further, we run a final refinement process on the “Select” clause to avoid listing the same column multiple times.

Constructing the From Clause

In general, the “From” clause is used to specify tables to be used in the query. At least one table should be listed in the “From” clause. When more than one table are listed in the “From” clause, the RDBMS takes their cross product. However, it is possible to have various versions of the join operation used in the “From” clause. Shown in Figure 1.3 is an example SQL query which illustrates the usage of the inner join in the “From” clause. It is important that all columns used in the select statement have their corresponding tables specified in the “from” clause. The developed system does this check while filtering the tables and columns as described in Section 3.5.6.

Join is one of the most interesting operations in the relational model. It is derived by a selection operation applied on the outcome from a cross-product operation. It matches and combines records from two tables. There are various versions of the join operation, including theta join, equi-join, natural join, semi-join, etc. In our research, we used the inner join as the default join. Integrating other versions of the join has been left as future work. Our system is capable of realizing multiple inner joins in a user’s question. For example, consider

the database diagram shown in Figure 1.1 and the query: “What is the population of the country where Chris lives”; the question will be converted by the developed system into the SQL query shown in Figure 3.6.

Ambiguity may occur while analyzing user’s input. When it is not possible to resolve the ambiguity using the steps described in Section 3.5.6, the system will ask the user to select the table/column he/she meant. The information supplied by the user will be stored for later usage. Algorithm 2 is responsible for constructing the “From” clause in the SQL query under construction by the system.

Algorithm 2 Building the “From” Clause for SQL

Require: *tables* which is the list of tables to be used

```

1: relations = listoftuples(string, string)
2: query = ""
3: for table ∈ tables do
4:   for t ∈ tables do
5:     if table ≠ t then
6:       if there is relation between table and t then
7:         if relations don't have table and t then
8:           relations.Add(table, t)
9:           query = query + "innerjoin" + t + "on" table.primaryKey = t.foreignKey
10:        end if
11:      end if
12:    end if
13:  end for
14: end for
15: return query

```

Constructing the “Where” Clause

Building the where clause is one of the most important steps in coding a SQL query. Users might have multiple conditions, each with a different operator and complexity. Users might target to retrieve records based on a specific range, date, etc. or they might need to get records based on a specific instance in the database.

Table 3.11 shows some example conditions and how they are reflected in the “Where” clause. In these examples, we assumed there are two tables, namely “Person” and “Services”,

and there is a relation between the two tables where every person has a list of services provided.

Table 3.11: Example on Conditions and their Corresponding Expression in the “Where” Clause

Condition	SQL translated
Service fee between 5 and 10	Where services.fee between 5 and 10
Service fee is the highest	Where services.fee=(select max(service.fee) from services)
Service fee is higher than the average or service fee between 5 and 10	Where services.fee >(select min(services.fee) from services) or services.fee between 5 and 10
Service fee is higher than the average and service fee between 5 and 10	Where services.fee >(select min(services.fee) from services) and services.fee between 5 and 10
Service fee is equal to the average service fee provided by abed	Where services.fee=(select avg(services.fee) from services inner join person on services.pid=person.pid Where person.name=”abed”)
Services made by abed	Where services.serviceID=(select services.serviceID from services inner join person on services.pid=person.pid Where person.name=”abed”)
Service fee is the lowest	Where services.fee=(select min(service.fee) from services)
Services created after 1/1/2015 and have a service fee greater than 5	Where services.creationDate >1/1/2015 and services.fee >5

One advantage of the developed system is its capability to handle multiple conditions. On the other hand, many other systems can handle only a limit number of conditions. Our system is also capable of considering and differentiating between “and” and “or” connectors in compound conditions.

Conditions are split using a separate function. In general, a user question consists of a main clause and condition clause. The condition clause might have multiple conditions. We built our own algorithm to detect these multiple conditions based on different predefined tokens. During the process, we make sure that when we split a condition in a sentence we do not map values to other conditions, especially when there are ranges. Fuzzy conditions are kept aside; they are added to a list that will be checked later on because fuzziness requires a special treatment as explained in Section 3.6.

Each condition is expected to have at least one column that should be used in the coding. This is accomplished based on the dependency between the operator and the value from the filtered columns list. We check the type of each column to make sure that the correct

mapping is done using the correct value. We also check whether the relation between a column and an operator is negation. Some conditions might cover tables, such as “What are names of people who have services”; here “Services” is a table name. Such cases are successfully handled by our system. However, there are cases which our system cannot handle as mentioned in Chapter 5; these have been left as future work.

3.6 Fuzziness

Fuzziness is handled as the last step before submitting the constructed SQL for execution. Recall that the developed system supports fuzziness both in the two types of allowed queries: (1) questions may include fuzzy terms, and these are captured as part of the condition in the “Where” clause, and (2) a user who decides to submit an SQL statement is allowed to have incorporate fuzziness as part of the condition in the “Where” clause. The second case is possible because the developed system pre-process every submitted SQL statement to validate its content and to resolve fuzziness, if exists. In other words, when a user inputs the query as a question, the system checks resolves the fuzziness while building the “Where” condition; alternatively, the system checks for and resolves fuzziness that may exist in the “where” clause of any actual SQL query submitted by a user.

The result from a given query is post processed before it is delivered to the user. The post processing step is needed to deal with values related to a fuzzy term in the input query, if any. For each returned row, the system calculates the membership degree to show the closeness of the retrieved record to the domain centroid related to the fuzzy term. Finally, in case none of the records in the database matches the condition in the “where” clause (including the fuzzy condition), a message is delivered to the user to indicate that the system could not find any result that matches the submitted query.

We have adopted from a previous work by our group [34] the three methods shown in Table 3.12 for specifying and handling the fuzzy sets associated with fuzzy terms allowed in

Table 3.12: Methods for Handling Fuzziness

Possible Method	Specifying Fuzzy Domains	Deciding on Fuzzy Ranges
Manual	By User	By User
Semi Automated	By User	By System
Automated	By System	By System

queries. The details related to each of these three alternatives are covered in this section.

As a running example for the material covered in this section, assume there is a column called “Fee” which contains the values listed in Figure 3.13. Here it is important to note that we have adopted triangular fuzzy membership functions for this study. Further, a user got the opportunity to modify the fuzzy membership functions at any time. Finally, the membership degree for each given value is computed using Equation 2.4.

Table 3.13: Fee Column Values

Fee	10	15	0	12	25	28.5	16.5	4	20	2	3	5.5	11	18	30	55	26
-----	----	----	---	----	----	------	------	---	----	---	---	-----	----	----	----	----	----

3.6.1 Fully Automated

The users may decide to leave it up to the system to take full control for deciding on the fuzzy sets and their ranges. In this case, the system decides on having three fuzzy domains/sets by default. These are named as “High”, “Medium”, and “Low”. The ranges for these three fuzzy domains and their centroids are determined by analyzing the values present in the corresponding column which is intended to be queried based on a fuzzy term. The centroids of the “High” and “Low” domains take the maximum and minimum values in the analyzed column, respectively. By default, the ranges for these two fuzzy domains will have one side with slope= 1. However, the range may be adjusted by the system based on the distribution of the values present in the column. For the “Medium” fuzzy domain, its centroid is normally taken as the middle point between the maximum and minimum values present in the column. The range of the “Medium” fuzzy domain is specified between two lines each with slope= 1 on the two sides of the centroid. As it is the case with the other two fuzzy domains, it is

also possible to adjust and tune-up the range of the “Medium” fuzzy domain by considering the values in the column.

To illustrate the fully automated specification of the fuzzy domains, assume it is required to evaluate the following query: “What are names of people who have high service fee?”. Here, “High” is determined as a fuzzy term. The system checks if already the “Fee” column has corresponding fuzzy domains defined for the specific user who submitted the query, or in general otherwise. The general case means the system has already or will split the range of the values present in the “Fee” column into three domains as described above; these ranges are shown in Figure 3.14. Since the user specified “High” in the query, the system will retrieve all services that have an associated fee between 55 and 34.5, inclusively. Then, Equation 2.4 will be used to compute the corresponding membership values.

Table 3.14: Fully Automated Fuzzy Ranges

Fuzzy Domain	Ranges
High	36.67-55
Medium	9.17-45.83
Low	0-18.33

We developed our own algorithm for computing the ranges of the fuzzy domains. The default number of domains is 3. Our algorithm has been built to work on triangular shaped fuzzy domains which is the shape we have adopted for the study in this thesis. Choosing any other shape for the fuzzy domains will not have major effect on the study; only some tuning is required in the developed algorithms. We will refer to Figure 3.7 to explain how the triangles are determined. To demonstrate how the algorithm works, we used three domains in Figure 3.7, namely “Low”, “Medium”, and “High”. The bases of the three triangles extend along the x-axis with some overlap. The segment “ac” forms the base for “Low”, the segment “bf” forms the base for “Medium”, and the segment “eg” forms the base for “High”.

Assume the length of the segment “ab” is x . The range of the values along the base is

divided into six equal segments, namely, “ab”, “bc”, “cd”, “de”, “ef”, and “fg”. The length of each of these segments is x . The length of the segment separating two consecutive centroids is $3x$. For instance, “a” is the centroid of “Low” and “d” is the centroid of “Medium”; the length of the segment “ad” is —”ab”—+—”bc”—+—”cd”—= $3x$. This way, we know the number of segments needed for constructing the bases of the fuzzy domains along the x-axis. The base of each of the left-most and right-most triangles should be $2x$, while the base of each intermediate triangle should be $4x$. Further, each two adjacent triangles overlap by a segment of length x . All of these can be easily realized by considering the example fuzzy domains shown in Figure 3.7.

Equation 3.1 is used to calculate the membership values for n fuzzy domains. In Equation 3.1, HighValue and MinValue, respectively, stand for the maximum and minimum values in the column for which the fuzzy domains are to be derived. The start and the End point for each intermediate range (other than the left-most and right-most ranges) are calculated using Equations 3.2 and 3.3 respectively.

3.1: Calculating the value of x

$$x = \frac{maxValue - MinValue}{3 \cdot (n - 1)} \quad (3.1)$$

3.2: Calculating the start point for each intermediate range

$$startPoint = endPointOfPreviousRange - x \quad (3.2)$$

3.3: Calculating the end point for each intermediate range

$$endpoint = startPoint + 4 \times x \quad (3.3)$$

3.6.2 Semi Automated

Semi automated fuzziness means the user is expected to decide on the domains but the system will work out the range to be covered by each domain. For instance, a user may decide on having five domains, and accordingly the system has to analyze the data to find the range of each domain. The system first finds the maximum and minimum values of the column for which the fuzzy domains are to be determined. Second, the system divides the area between the minimum and maximum values along the x-axis by considering the analysis done above for the fully automated case. This way, the system will find the base of each fuzzy domain.

To illustrate the semi-automated specification of the fuzzy domains, assume a user decided to have five fuzzy domains for “Fee” column. These five fuzzy domains will be named: “High”, “MidHigh”, “Medium”, “MidLow”, and “Low”. By considering the values in “Fee” column as listed in Table 3.13, it is possible to find the maximum and the minimum values as 0 and 55, respectively. This range along the x-axis is to be divided into segments of equal size x as explained in the previous section. The number of segments depends on the number of fuzzy domains. For the five fuzzy domains in the illustrating examples, $3 \times (5 - 1) = 12$ consecutive segments are needed. Accordingly, the length of each segment is computed as $\frac{(55-0)}{3 \times (5-1)} = 4.583$. Finally, the computed length of the five bases for the five fuzzy domains are listed in Table 3.15. These five triangular fuzzy domains are shown in Figure 3.8.

Table 3.15: Calculated Ranges for the Predefined Five Fuzzy Domains

Domain	Range
High	45.833 – 55
MidHigh	32.083 – 50.417
Medium	18.333 – 36.667
MidLow	4.583 – 22.917
Low	0 – 9.167

3.6.3 Manual Specification of the Fuzzy Domains and their Ranges

To avoid imposing certain specification on the users, the developed system provides the opportunity to the users to decide on both the fuzzy domains and their ranges. Then based on the values entered by the user, the system determines the slope of the lines surrounding each fuzzy domain. For the two extremes, i.e., the leftmost and the rightmost domains, only one line is considered for each of them. But for each intermediate fuzzy domain, the system determines the slopes of the two lines the domain, one on the left side and one on the right side.

The system stores all the fuzzy domains regardless how they are specified, using any of the three strategies described above. As the content of the database may change, it is expected that the fuzzy domains may need some tune up to keep the good coverage of the data. This can be done either by the system or the users may adjust the ranges as they wish.

3.7 Display the Result and Receiving the Feedback

After successfully completing all the analysis needed, the query is executed and the result is returned to the user in the form of a grid which contains all the information requested by the user and retrieved from the database. In case fuzziness is detected for the first time in link to a particular user, the user has the option to save the specified ranges for later usage, regardless whether the ranges were determined by the system or decided manually by the user. In addition, users have the option to complete a report that reflects their degree of satisfaction with the results. The system also displays for the users the translated queries so that they can see how their queries were transformed into SQL and how the fuzziness was handled. Moreover, each parsed question will be saved in the XML file to save the processing time in case the user will decide to execute the same query again in the future.

For the visualization, we used D3JS which is a JavaScript library. This library is used to show the users how their questions are linked to the database schema. We believe this kind

of visualization will increase the confidence of the users in the accuracy and completeness of the retrieved data. Figure 3.9 illustrates the visualization of the links between users' questions and the database tables and columns. Here it is worth mentioning that this is an interactive visualization where a user can hover on each node to see its name. The orange nodes represent the words in the question (sentence), the dark blue nodes are names of the tables and the light blue nodes refer to names of the columns. The arrows between nodes indicate the existence of a mapping between each directly linked two nodes.

3.8 Summary

This chapter covered in details how the developed system deals with users' queries expressed either as questions or as normal SQL queries with the opportunity to incorporate fuzziness. All steps of the process are described from the submission of the query to the delivery of the result to the user who submitted the query. We believe that the proposed approach will lead to high accuracy in retrieving data by database users who are not experts in coding SQL statements. While developing the system we tried to minimize the interference of the users and also at the same time retrieve the most accurate result possible. However, our system doesn't always report 100% accuracy due to the difficulty in completely understanding and executing a query submitted by a naive user. In other words, the source of the lower accuracy may be due to having users unable to fully express their intention and target from the database. Still we admit that there are some limitations in the system and these will be discussed in Chapter 5. Finally, the next chapter describes the database used in testing the developed system; also covered in the next chapter are the test scenarios used to check the accuracy of the results retrieved by the system. We used multiple types of questions with different structures to check the efficiency of the developed system.

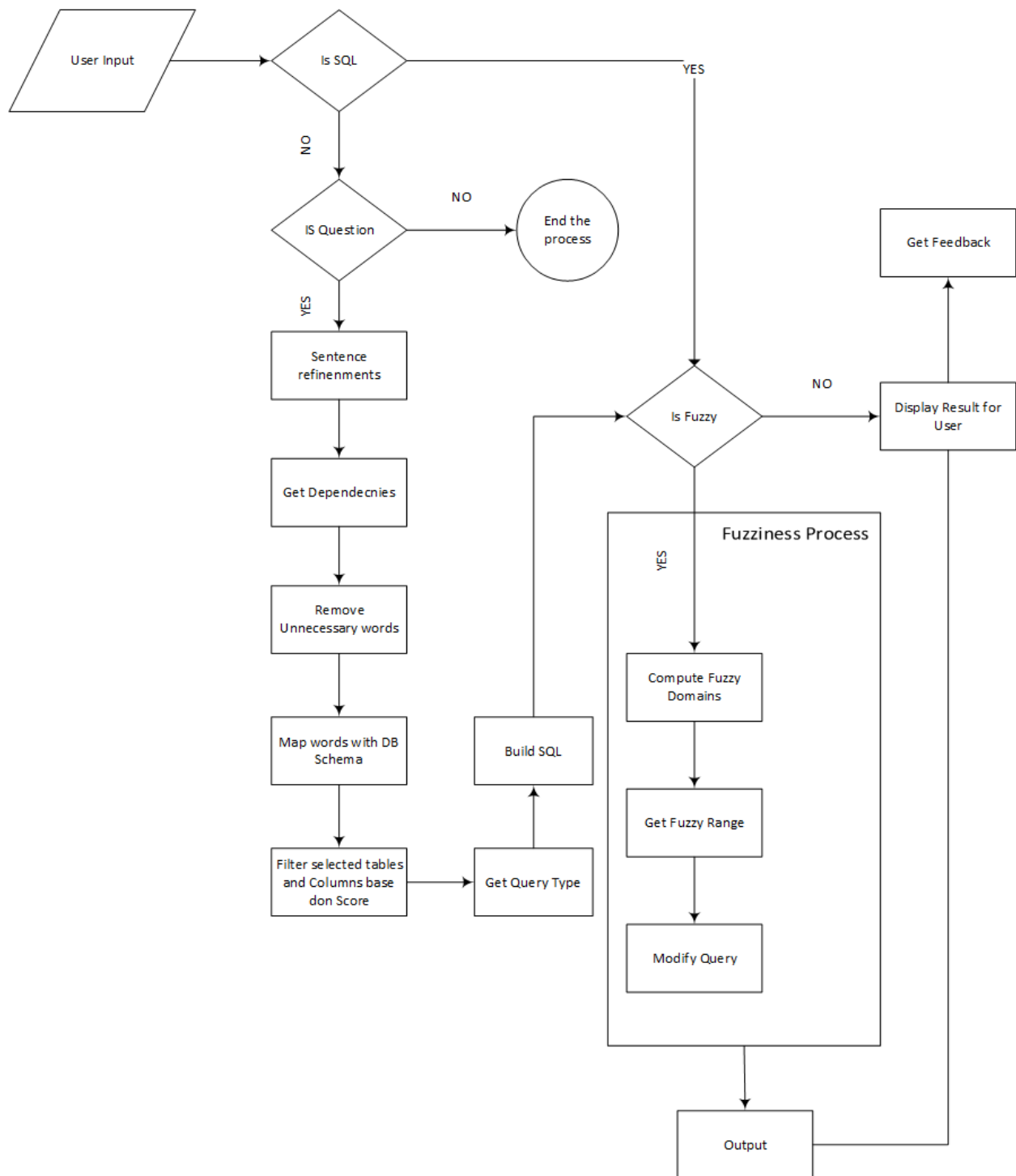


Figure 3.5: The general Flow of Control in the System for Analyzing User Input

```

SELECT  lkup_country.countryPopulation
FROM  personInfo
      inner join  lkup_City on  PersonInfo.personCityID= lkup_City=cityID
      inner join lkup_Country on lkup_City.cityCountryID=lkup_Country.countryID
WHERE PersonInfo.personFirstName='Chris'

```

Figure 3.6: An Example SQL Query Using more than 2 Inner Joins

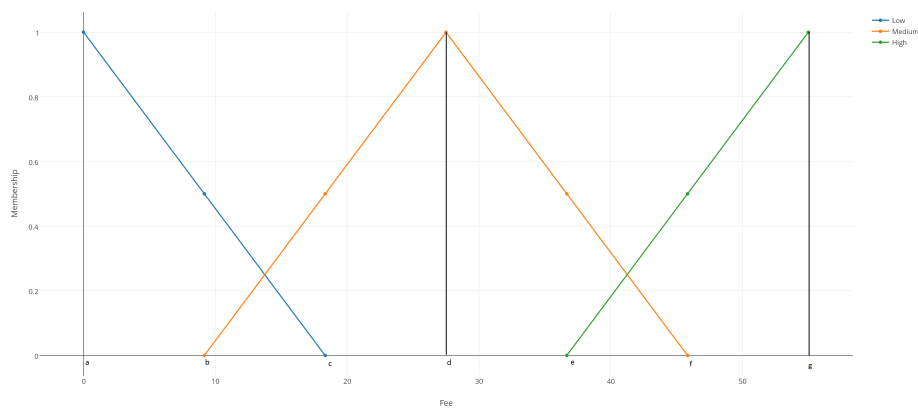


Figure 3.7: An Example of SQL Query using 2 inner joins

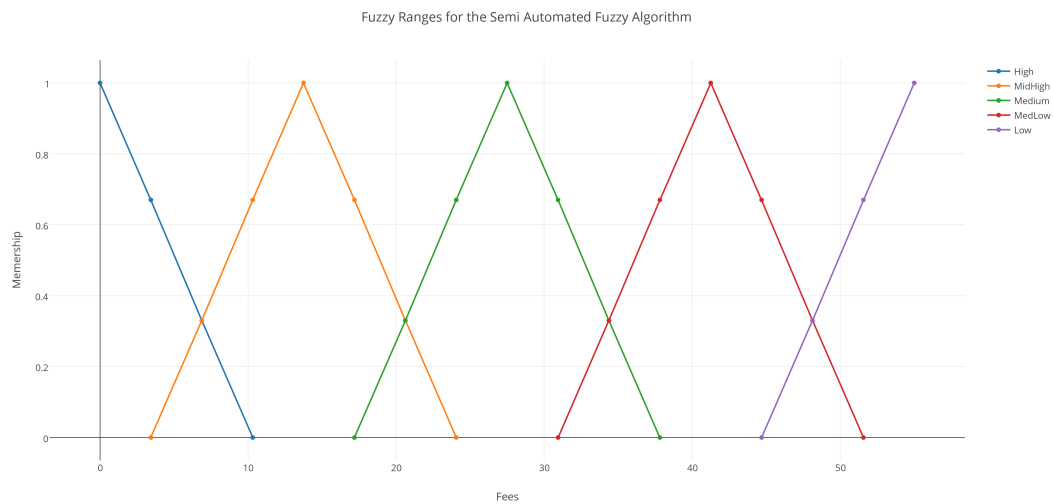


Figure 3.8: The Five Fuzzy Domains of the “Fee” Column as Discussed in the Example that Illustrates Semi-Automated Specification of the Fuzzy Domains

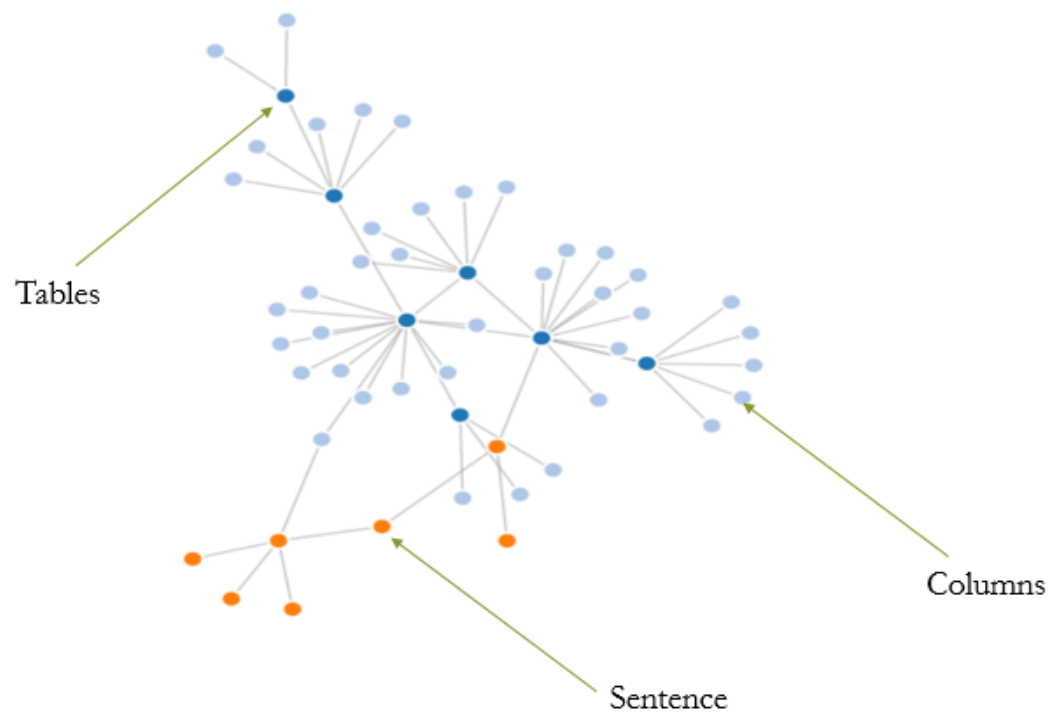


Figure 3.9: Visualization of the Relationship between a User's Question and the Database Tables and Columns

Chapter 4

Experimental Analysis

Though the proposed system has been thoroughly covered in Chapter 3, we understand the need to support that by conducting some experiments to demonstrate its various aspects. Accordingly, this chapter is dedicated to describe the test environment, the conducted experiments and to report the results together with the necessary validation. The database described in Section 4.2 will be used in the experiments. The created test cases are very close to real life scenarios. A variety of illustrative questions are used to show how the feedback process works.

4.1 Testing Environment

The developed question driven query system can be used as front-end to any relational database. This is true because the whole process depends on finding keywords in a given question and then matching them with names of tables/columns in the given database. We used Microsoft SQL Server 2014 in the testing. Here it is worth mentioning that neither the test scenarios nor the outcome will be affected in case another DBMS is used, e.g., MySQL, Oracle, etc. The only reason for using Microsoft SQL server 2014 is the available free license for University of Calgary students as part of dreams park provided by Microsoft.

The SQL server was installed on a desktop computer in the database lab at the University of Calgary. Characteristics of the computer are listed in Table 4.1. Though it is a powerful machine, the minimum requirements to run the developed system are Windows OS 7 and 2 GB RAM.

Table 4.1: Characteristics of the Computer Used in Testing the Developed System

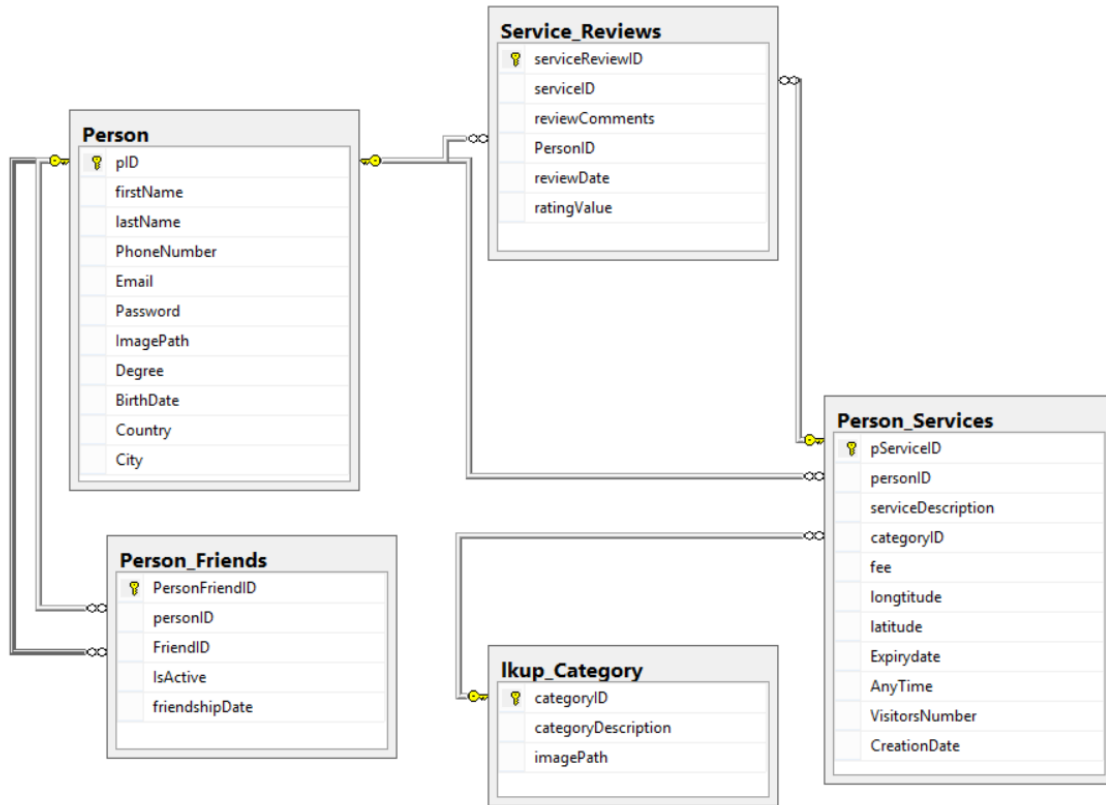
Feature	Description
CPU	Intel Core i5-2400
RAM	8GB RAM
Hard disk Capacity	500 GB
Operating System	Windows 8 - 64bit

4.2 Data Set

A variety of techniques exist for collecting data which may be stored in a number of possible formats, including structured, semi-structured, plain text, etc. However, since the relational data model was development in 1970s, most enterprise data is stored in structured relational databases. Further, data repositories are currently characterized by increased availability and reachability by a wide variety of users from naive to professionals. Thus, question driven queries are gaining popularity where all types of users could enjoy running their own queries without any barriers like the need to learn SQL.

For the experiments, we defined and populated an example relational database which includes data related to persons and the services they provide to people living in their neighborhood. In other words, the example database structure is based on a test scenario where people are providing services such that each service has a category and reviews. The problem definition underlying the database could be briefly articulated as follows. Persons may register their services in the given relational database and neighbors may seek specific services by running their own queries. It is also possible to add a review and rate a given service. Further, the example database keeps the friendship relationship between persons. A person may refer others to use services provided by his/her friends; he gets credit in case the service receives good review and rating. The database diagram is shown in Figure 4.1, and the columns associated with each table in the example database are listed in Table 4.2. Expected values for the various types are given in Table 4.3.

The primary key of each table is indicated in Figure 4.1. Further, the relationships



inner joins and conditions are involved in a user's question, the more time will be need to analyze the question. In this section, we will describe each type of query which can be executed using the developed system.

4.3.1 Simple Questions

In general, a simple question involves a simple select statement from a single table. These question are attractive because they can be executed almost instantly without consuming many resources. However, there are some cases where a simple question may turn into a nightmare when it contains some words which could be directly linked to tables/columns in the database. Recall that such case requires analyzing all values in specific columns to find out whether it is possible to report some matching.

Table 4.5 shows a sample set of simple questions and their corresponding SQL statements.

4.3.2 Complex Questions

In general, complex queries involve more than one table and hence may be coded either as traditional cross product with associated conditions or as inner joins. Further, it is possible for a complex query to have a number of sophisticated conditions which include a mixture of disjunctive and conjunctive conditions, i.e., connected by “and” or “or”. Recall that a complex condition is split and recognized as explained in Chapter 3. This type of queries is expected to consume more time and resources to be processed by the system.

In addition to having columns used in conditions on either or both sides of an operator, some conditions might contain nested queries or aggregation functions. Some conditions might even need description with other records in the database, and this will require several steps to be completed. Table 4.6 shows some example complex queries with possible corresponding parsed SQL statements.

One attraction of the developed system is the variety of queries it is capable of handling.

Indeed, allowing arbitrary numbers of conditions per question and arbitrary number of inner loops per query, as well as having different types of questions that involve various types of conditions are very important feature of our system that other existing systems lack. In the testing, we tried to focus on the variety of question that are expected to be commonly asked by users.

To test the developed system we used multiple queries similar to the examples shown in Table 4.6 and Table 4.5. This testing style adds some credit to our system and removes the bias. The cost of complex questions will be discussed in Section 4.4. Finally, Appendix B shows results for some of the questions shown in Table 4.6 and Table 4.5.

4.3.3 Fuzzy Questions

Fuzziness is another feature that distinguishes our system. We implemented a set of fuzzy questions to test the effectiveness of our system. Table 4.7 shows some fuzzy questions. Some limitations were faced when dealing with fuzziness, i.e., the developed system still needs some tuneup to be capable of handling fuzziness more comprehensively. The deeper the fuzziness is in users' queries the less accurate will be the results; we plan to improve this as future work.

Finally, the three approaches described earlier (namely, manual, semi-automated and fully automated) were employed to decide on number of fuzzy domains and their ranges for the fuzzy terms used in the example queries listed in Table 4.7.

4.4 Performance Analysis

Measuring system performance is an integral part to determine the quality of the developed system, and to decide whether it meets the expectations. In this section, we will describe how to measure the time needed for each query type. For this purpose, we used the same computer with the specifications described in Table 4.1. We run visual studio in a release

mode and we used a built-in time function in visual studio to measure the time needed.

Some users were asked to provide some questions to measure system's performance. Users submitted 95 questions which follow the distribution shown in Table 4.9.

For effective analysis we measured the average execution time of the queries in Table 4.8. We separated queries that require user interaction from those that were analyzed without any user interference. The obtained results are shown in Table 4.9. Recall that users are expected to interfere just in case of ambiguity or in case there was remaining unmapped words.

The results shown in Table 4.9 demonstrate that complex queries usually take double the time needed by a simple query. The measured time includes retrieval time, i.e., the time needed to retrieve data from the database. The number of conditions used in the complex questions executed in the experiments ranges from one to three. The fuzzy queries executed contain only one fuzzy condition and at most two inner joins.

We also measured the time needed for the steps of the analysis. To be able to know which analysis step is consuming the most time we should record the average time needed by each category from the 95 queries. We didn't consider in this test user interaction because we are measuring only analysis time of any interaction. The results are shown in Table 4.10.

As reported in Table 4.10, fuzzy questions are the most time consuming because data should be retrieved before deciding on the various parameters related to fuzziness.

4.5 System Accuracy

System's effectiveness is directly related to its accuracy level. In this section, we describe the queries used for measuring system's effectiveness and we elaborate on some queries for which our system failed.

Accuracy was checked using the same queries that were used to measure system's performance. Table 4.8 includes the number of queries used for each type. Each query is tested

alone and it is given a score equal to one if it retrieved the correct result. Table 4.11 shows the number of questions that were analyzed correctly and those which were wrongly analyzed for each query type. It is not surprising to realize that all simple queries were correctly analyzed since they are straightforward. However, our system failed to correctly analyze some complex questions and some fuzzy questions. In Chapter 5, we elaborate on these types of queries.

4.6 Summary

The time needed by our system to analyze users' requests ranges between 1 to 3 seconds per request which is acceptable. Also, we shouldn't forget that there is some dependency on the database size and the amount of information to be retrieved. Of course, the time is directly proportional to the database size.

Our system reported around 90% accuracy which proves its efficiency in handling user requests in a reasonable amount of time. Some enhancements should be done to handle more sophisticated questions which will be discussing in Chapter 5.

Table 4.2: Database Tables' Columns and their Type

Table Name	Column Name	Column Type
Person	firstName	bigint
Person	lastName	nvarchar(50)
Person	phoneNumber	nvarchar(50)
Person	Email	nvarchar(50)
Person	Password	nvarchar(50)
Person	imagePath	nvarchar(50)
Person	Degree	nvarchar(50)
Person	birthdate	datetime
Person	Country	nvarchar(50)
Person	City	nvarchar(50)
Lkup_category	categoryId	bigint
Lkup_category	categoryDescription	nvarchar(50)
Lkup_category	imagePath	nvarchar(50)
Person_Friends	personFriendId	bigint
Person_Friends	personId	bigint
Person_Friends	friendId	bigint
Person_Friends	isActive	bit
Person_Friends	friendshipDate	datetime
Person_Services	personServiceId	bigint
Person_Services	personId	bigint
Person_Services	serviceDescription	nvarchar(max)
Person_Services	categoryId	bigint
Person_Services	Fee	money
Person_Services	longitude	decimal (18,10)
Person_Services	Latitude	decimal (18,10)
Person_Services	expiryDate	date
Person_Services	anyTime	bit
Person_Services	visistorsNumber	bigint
Person_Services	creationDate	datetime
Service_Reviews	serviceReviewId	bigint
Service_Reviews	serviceId	bigint
Service_Reviews	reviewComments	nvarchar(max)
Service_Reviews	personId	bigint
Service_Reviews	reviewDate	datetime
Service_Reviews	ratingValue	int

Table 4.3: Expected Input for each Column Type

ColumnType	Expected Input
Datetime	Date value(mm/dd/yy)
Nvarchar(50) or Nvarchar(max)	Text
Bit	0 or 1
Bigint	Number without decimals
Money	Number with decimals
Int	Number without decimals
Decimal	Number with decimals

Table 4.4: Number of Records in the Populated Tables

Table Name	Number of Records
person	11
Service_Reviews	4253
Person_Services	2996
person_friends	30
Lkup_category	13

Table 4.5: Sample of Simple Questions

Question	Parsed Query
What is the highest service fee	Select max(Person_Services.fee) from Person_Services
What is the lowest service fee	Select min(Person_Services.fee) from Person_Services
what is the average service fee	Select AVG(Person_Services.fee) from Person_Services
What are the names of the providers	Select name from people
How many service reviews are in the database	Select count(*) from Service_Reviews
What are the services stored in the database	Select * from Person_Services
How many categories are in the database	Select * from Lkup_Category
What are the descriptions of the categories	Select categoryDescription from Lkup_Category

Table 4.6: Sample of Complex Questions with Possible Corresponding SQL Statements

Question	Corresponding Query
What services were made by first user	Select * from Person_Services inner join Person on Person_Services.personID=Person.personID where LOWER(LTRIM(RTRIM(Person.firstName)))='first' and LOWER(LTRIM(RTRIM(Person.lastName)))='user'
What is the average fee for services made by first user	Select avg(Person_Services.fee) from Person_Services inner join Person on Person_Services.personID=Person.personID where LOWER(LTRIM(RTRIM(Person.firstName)))='first' and LOWER(LTRIM(RTRIM(Person.lastName)))='user'
What servers were created after 10/10/2015	Select * from Person_Services where Person_Services.creationdate > '10/10/2015'
What is the number of people who have a service fee greater than 5	select count(*) from People inner join Person_Services on Person.personID=Person_Services.personID where Person_Services.fee > 5
How many service reviews exist in the database	select count(*) from Service_Reviews
How many people have a service fee greater than the average	select count(*) from People inner join Person_Services on Person.personID=Person_Services.personID where Person_Services.fee > (Select avg(Person_Services.fee) from Person_Services)
What are the names of people who have the highest number of services	select Person.firstName, Person.lastName from Person_Services inner join Person on Person.personID=Person_Services.personID where Person.personID in (2)
What are services which don't have the highest number of reviews	select * from Person_Services inner join Service_Reviews on Person_Services.pServiceId=Service_Reviews.serviceId where Person_Services.pServiceId not in (16662)
What are the names of people who have friends	select Person.firstName, Person.lastName from Person inner join Person_Friends on Person.personID=Person_Friends.personId
What are the average ratings for services	select Avg(Service_reviews.ratingValue) from Person_Services inner join Service_Reviews on Person_Services.pServiceId=Service_reviews.serviceId

Table 4.7: Sample of Fuzzy Questions and Possible Corresponding SQL Queries

Fuzzy Question	Translated Query
What are names of people who have a high service fee and more than 5 service visitors	select Person.firstName, Person.lastName,(Person.Services.fee-4)10 as HowFarFromBeinglow_Person.Services.fee from Person.Services inner join Person on Person.personID=Person.Services.personId inner join Service_Reviews on Person.Services.pServiceId=Service_Reviews.serviceId where Service_Reviews.ratingValue 5 and Person.Services.fee between 4 and 9999.5
Which reviews have moderate rating	select *,(Service_Reviews.ratingValue-2)10 as HowFarFromBeingmedium_Service_Reviews_ratingValue from Service_Reviews where Service_Reviews.ratingValue between 1 and 3
Which services have the best service fee	select *,(Person.Services.fee-19995)10 as HowFarFromBeinghigh_Person.Services.fee from Person.Services where Person.Services.fee between 9999.5 and 19995
Which services have a low service fee and the highest number of reviews	select *,(Person.Services.fee-4)10 as HowFarFromBeinglow_Person.Services.fee from Person.Services inner join Service_Reviews on Person.Services.pServiceId=Service_Reviews.serviceId where Person.Services.pServiceId in (16662) and Person.Services.fee between 4 and 9999.5
What are names of people who have the highest number of service reviews and normal number of service visitors	select Person.firstName, Person.lastName,(Person.Services.visitorsNumber-12484.5)10 as HowFarFromBeingmedium_Person.Services.visitorsNumber from Person.Services inner join Person on Person.personIDPerson.Services.personId inner join Service_Reviews on Person.Services.pServiceIdService_Reviews.serviceId where Person.personID in (10) and Person.Services.visitorsNumber between 6245.75 and 18723.25
What are names of people who have high service fee or low number of service visitors and high number of service reviews rating	select Person.firstName, Person.lastName,(Person.Services.fee-19995)10 as HowFarFromBeinghigh_Person.Services.fee,(Person.Services.visitorsNumber-7)10 as HowFarFromBeinglow_Person.Services.visitorsNumber,(Service_Reviews.ratingValue-4)10 as HowFarFromBeinghigh_Service_Reviews_ratingValue from Person.Services inner join Person on Person.personIDPerson.Services.personId inner join Service_Reviews on Person.Services.pServiceIdService_Reviews.serviceId and Person.Services.fee between 9999.5 and 19995 or Person.Services.visitorsNumber between 7 and 12484.5 and Service_Reviews.ratingValue between 2 and 4

Table 4.8: Number of Questions Executed to Measure System's Performance

Query Type	Number of Executed Query
Simple Queries	20
Complex Queries	55
Fuzzy Queries	20

Table 4.9: Average Time Needed for each Query Type

Query Type	With Interaction	Without Interaction
Simple	2	1.2
Complex	3.4	2.5
Fuzzy	4.5	3.8

Table 4.10: Average Time Needed by each Analysis Component (ms)

Query Type	Refinement Process	mapping Process	SQL Formation Process	Fuzziness Process	Data Retrieval	Total
Simple	86.70	333.12	6.13	0	864.55	1290.5
Complex	253.08	578.29	14.56	0	1662.04	2507.9
Fuzzy	39.18	1216.73	1.19	29.774	2547	3834

Table 4.11: Number of Wrongly/Correctly Analyzed Queries

Query Type	Correct	Wrong	Total
Simple	20	0	20
Complex	51	4	55
Fuzzy	15	5	20

Chapter 5

Conclusion and Future Work

In this chapter, we will summarize the outcome of this thesis, discuss limitations of the proposed framework, and list several directions for future work.

5.1 Summary

This thesis propose a new system as a friendly visual interface between naive users and RDBMS. The proposed system shows efficiency in analyzing user queries in reasonable amount of time with a very high accuracy. Users will be able to use this system to retrieve data from a RDBMS by writing some questions that follow a predefined structure. Users are allowed to use fuzziness in their queries.

The basic features provided by the developed system can be summarized as follows:

- It allow all types of users (naive to professionals) to connect to and query a RDBMS.
- It allow users to define specific fuzzy domains for each numeric column intended to be queried using fuzzy terms. Ranges of the various fuzzy domains may be specified by users or the system may derive them by data analysis.
- It allows the system to learn and build decisions automatically by analyzing previous users' decisions
- It allow users to view their previously executed queries and run them again without any need to repeat the conversion into SQL.
- It refines user's input that might effect system's accuracy

- It detects date and number phrases expressed in words and parses them into their corresponding actual values.
- It allow users to see database structure to help them in building their queries.
- It allow users to write questions that involve fuzziness and multiple conditions from different tables.

The reported accuracy and performance analysis prove system's efficiency in handling users' request. To the best of our knowledge there is no system that can handle user questions with fuzziness and complex conditions as our system does.

5.2 Limitations

Some limitations have been faced during the analysis. These limitations are mostly faced when there is imprecision in user's questions. In other words, some user's questions may involve more rule based fuzziness which is a case not handled in our system. This kind of analysis has been left as future work. In addition, when mapping components of users' questions to database schema elements we didn't take into consideration the grammatical relation between words. Also this has been left as future work.

5.3 Future Work

The developed system can be used and enhanced in many different ways. Multiple components can be enhanced to positively affect the analysis process for better accuracy and accessibility. These changes can be summarized as follows:

- Change the system into a web based application where everyone will be able to use regardless of their operating system. This kind of improvement will lead to better learning system that may be used by users who have Internet

connection and are capable of connecting to the system remotely regardless of their locations.

- Add speech recognition as an alternative user input. This way users may avoid typing their questions and depend on the voice recognizer instead.
- Enhance the mapping process using grammatical relations.
- Develop more sophisticated algorithm to enhance the handling fuzziness which might involve multiple columns to be used in a set of rules.
- Provide the ability to handle prediction in users' questions so that when users ask about some information that might happen in the future then the answer will be based on some intelligent data analysis.
- Provide the ability to handle multiple languages

All these extra features and enhancement are on our to do list. We will start to work on them shortly. However, following the completion of these extensions some more test scenarios should be run to check the influence on the system as a whole and to find out whether new subsequent extension may become feasible.

Bibliography

- [1] D3js tool used for visualization. <http://d3js.org/>. Accessed: 2015-11-10.
- [2] Relational database ranking. <http://db-engines.com/en/ranking>. Accessed: 2015-10-21.
- [3] Rajeev Agrawal, Amogh Chakkarwar, Prateek Choudhary, Usha Jogalekar, Deepa H Kulkarni, et al. Dbqisan intelligent system for querying and mining databases using nlp. In *Information Systems and Computer Networks (ISCON), 2014 International Conference on*, pages 39–44. IEEE, 2014.
- [4] Amira Aloui and Amel Grissa. A new approach for flexible queries using fuzzy ontologies. In *Computational Intelligence Applications in Modeling and Control*, pages 315–342. Springer, 2015.
- [5] Robert Andrews, Joachim Diederich, and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389, 1995.
- [6] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2):191–203, 1984.
- [7] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.
- [8] Livia Borjas, Josué Ramírez, Rosseline Rodríguez, and Leonid Tineo. Automated system for tests preparation and configuration using fuzzy queries. In *Computational Intelligence*, pages 199–212. Springer, 2015.
- [9] Patrick Bosc and Olivier Pivert. Sqlf: a relational database language for fuzzy querying. *Fuzzy Systems, IEEE Transactions on*, 3(1):1–17, 1995.

- [10] Erik Cambria and Bruce White. Jumping nlp curves: a review of natural language processing research [review article]. *Computational Intelligence Magazine, IEEE*, 9(2):48–57, 2014.
- [11] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.
- [12] Daniel M Cer, Marie-Catherine De Marneffe, Daniel Jurafsky, and Christopher D Manning. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *LREC*, 2010.
- [13] Angel X Chang and Christopher D Manning. Sutime: A library for recognizing and normalizing time expressions. In *LREC*, pages 3735–3740, 2012.
- [14] Angel X Chang and Christopher D Manning. Sutime: Evaluation in tempeval-3. *Atlanta, Georgia, USA*, page 78, 2013.
- [15] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 740–750, 2014.
- [16] Philipp Cimiano, Peter Haase, Jörg Heizmann, Matthias Mantel, and Rudi Studer. Towards portable natural language interfaces to knowledge bases—the case of the orakel system. *Data & Knowledge Engineering*, 65(2):325–354, 2008.
- [17] Edgar F Codd. Further normalization of the data base relational model. *Data base systems*, pages 33–64, 1972.
- [18] Marie-Catherine De Marneffe and Christopher D Manning. Stanford typed dependencies manual. Technical report, Technical report, Stanford University, 2008.
- [19] Richard Dietz and Sebastiano Moruzzi. *Cuts and clouds: Vagueness, its nature and its logic*. Oxford University Press, 2010.

- [20] Shipra Dingare, Malvina Nissim, Jenny Finkel, Christopher Manning, and Claire Grover. A system for identifying named entities in biomedical text: How results from two evaluations reflect on both the system and the evaluations. *Comparative and Functional Genomics*, 6(1-2):77–85, 2005.
- [21] Dimitar Driankov, Hans Hellendoorn, and Michael Reinfrank. *An introduction to fuzzy control*. Springer Science & Business Media, 2013.
- [22] Ramez Elmasri and Shamkant B Navathe. *Fundamentals of database systems*. Pearson, 2014.
- [23] Anton Fagerberg. Temporal information extraction using regular expressions. 2014.
- [24] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.
- [25] Jenny Rose Finkel, Alex Kleeman, and Christopher D Manning. Efficient, feature-based, conditional random field parsing. In *ACL*, volume 46, pages 959–967, 2008.
- [26] J. Galindo, J. M. Medina, and M. C. Aranda. Querying fuzzy relational databases through fuzzy domain calculus. *International Journal of Intelligent Systems*, 14(4):375–41, 1999.
- [27] J. Galindo, M. Medina, O. Pons, and J. C. Cubero. A server for fuzzy sql queries. In *In T. Andreassen, H. Christiansen, & H. L. Larsen (Eds.), Lecture Notes in Artificial Intelligence (Vol. 1495): Flexible query answering systems*, pages 164–174. Springer, 1998.
- [28] Alessandra Giordani and Alessandro Moschitti. Generating sql queries using natural language syntactic dependencies and metadata. In *Natural Language Processing and*

- Information Systems*, pages 164–170. Springer, 2012.
- [29] Martin T Hagan, Howard B Demuth, Mark H Beale, et al. *Neural network design*. Pws Pub. Boston, 1996.
 - [30] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.
 - [31] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.
 - [32] Janusz Kacprzyk, Sławomir Zadrozny, and Guy De Tré. Fuzziness in database management systems: Half a century of developments and future prospects. *Fuzzy Sets and Systems*, 2015.
 - [33] Micheline Kamber, Lara Winstone, Wan Gong, Shan Cheng, and Jiawei Han. Generalization and decision tree induction: efficient classification in data mining. In *Research Issues in Data Engineering, 1997. Proceedings. Seventh International Workshop on*, pages 111–120. IEEE, 1997.
 - [34] Keivan Kianmehr, Negar Koochakzadeh, and Reda Alhajj. Askfuzzy: Attractive visual fuzzy query builder. In *IEEE ICDE 2012*, 2012.
 - [35] Keivan Kianmehr, Tansel Özyer, Mehmet Kaya, and Reda Alhajj. Wrapping vrxquery with self-adaptive fuzzy capabilities. *Web Intelli. and Agent Sys.*, 7(3):269–279, 2009.
 - [36] Keivan Kianmehr, Tansel Özyer, Mehmet Kaya, and Reda Alhajj. Wrapping vrxquery with self-adaptive fuzzy capabilities. *Web Intelligence and Agent Systems*, 7(3):269–279, 2009.
 - [37] George Klir and Bo Yuan. *Fuzzy sets and fuzzy logic*, volume 4. Prentice Hall New Jersey, 1995.

- [38] Lingpeng Kong and Noah A Smith. An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*, 2014.
- [39] Donald H Kraft, Erin Colvin, Gloria Bordogna, and Gabriella Pasi. Fuzzy information retrieval systems: A historical perspective. In *Fifty Years of Fuzzy Logic and its Applications*, pages 267–296. Springer, 2015.
- [40] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.
- [41] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010.
- [42] Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.
- [43] Jim Little, Michael de Ga, Tansel Özyer, and Reda Alhajj. Query builder: A natural language interface for structured databases. In *Computer and Information Sciences-ISCIS 2004*, pages 470–479. Springer, 2004.
- [44] Anthony Lo, Reda Alhajj, and Keivan Barker. Virex: Visual relational to xml conversion tool. *Journal of Visual Languages and Computing*, 17(1):25–45, 2006.
- [45] Anthony Lo, Tansel Özyer, Keivan Kianmehr, and Reda Alhajj. Virex and vrxquery: Interactive approach for visual querying of relational databases to produce xml. *Journal of Intelligent Information Systems*, 35(1):21–49, 2010.
- [46] Anthony Lo, Tansel Özyer, Radwan Tahboob, Keivan Kianmehr, Jamal Jida, and Reda Alhajj. Xml materialized views and schema evolution in virex. *Information Sciences*, 180(24):4940–4957, 2010.
- [47] David Maier. *The theory of relational databases*, volume 11. Computer science press Rockville, 1983.

- [48] Gideon S Mann and Andrew McCallum. Generalized expectation criteria for semi-supervised learning of conditional random fields. 2008.
- [49] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.
- [50] Claudio Moraga and Rodrigo Salas. A new aspect for the optimization of fuzzy if-then rules. In *null*, pages 160–165. IEEE, 2005.
- [51] H. Nakajima, T. Sogoh, and M. Arao. Fuzzy database language and library: Fuzzy extension to sql. In *Proceedings of the Second International Conference on Fuzzy Systems (FUZZ-IEEE93)*, pages 477–482, 1993.
- [52] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19, 2013.
- [53] Anh Kim Nguyen and Phuong Hong Nguyen. An intelligent natural language interface to relational databases. In *6th international conference on information technology and applications, ICITA*, 2009.
- [54] Mrs Neelu Nihalani, Sanjay Silakari, and Mahesh Motwani. Natural language interface for database: A brief review. 2011.
- [55] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.
- [56] Thair Nu Phyu. Survey of classification techniques in data mining. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 18–20, 2009.

- [57] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, page 141. Association for Computational Linguistics, 2004.
- [58] L. Portinale and A. Verrua. Exploiting fuzzy-sql in case-based reasoning. In *Fourteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS), AAAI Press*, pages 103–107, 2001.
- [59] Maria Samsonova, Andrei Pisarev, and Maxim Blagov. Processing of natural language queries to a relational database. *Bioinformatics*, 19(suppl 1):i241–i249, 2003.
- [60] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, pages 93–128, 2006.
- [61] Valentin Tablan, Danica Damjanovic, and Kalina Bontcheva. *A natural language query interface to structured information*. Springer, 2008.
- [62] V. Tahani. A conceptual framework for fuzzy query processing: a step toward very intelligent database systems. *Information Processing and Management*, 13:289–303, 1977.
- [63] Jeffrey D Ullman. *Principles of database systems*. Galgotia publications, 1984.
- [64] Mustafa Gokhan Uzunbas, Chao Chen, and Dimitris Metaxas. An efficient conditional random field approach for automatic and interactive neuron segmentation. *Medical image analysis*, 2015.
- [65] Peter Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 124(3):361–370, 2001.
- [66] Yang Wang, Kia-Fock Loe, and Jian-Kang Wu. A dynamic conditional random field model for foreground and shadow segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(2):279–289, 2006.

- [67] Lotfi A Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy sets and systems*, 90(2):111–127, 1997.

Appendix A

System Documentation

This appendix is dedicated to reflect on various characteristics of the developed system. It covers system documentation, some snapshots, data saving and some illustrative examples.

A.1 System Overview

Many features have been implemented in the developed system to achieve more accurate mapping while analyzing user queries. In this section, we will just show the main page in our system and its functionality; other features will be discussed in the following sections.

To view the system main page, some modules should be loaded and and connection to a specific database should be established. Users shouldn't bother about the modules since the system will detect their location automatically. However, users should manage connecting to a database which will be used in answering their questions.

Figure A.1 shows the main page of our system. The "loadProject" and "OpenProject" are used in case users want to switch to another project. The "Save" button is used to save the loaded information into an XML file. The system prompts the user to specify the location where the XML file should be stored. It is also possible to save the current data to a separate file instead of modifying the original loaded file. The "Database Dictionaries" button at the top of the page is used to define the dictionaries needed for the system. These include dictionaries related to synonyms, prefixes and suffixes, unnecessary words and fuzzy domains and ranges. Finally, the various forms supported by the system are covered in the rest of this chapter.

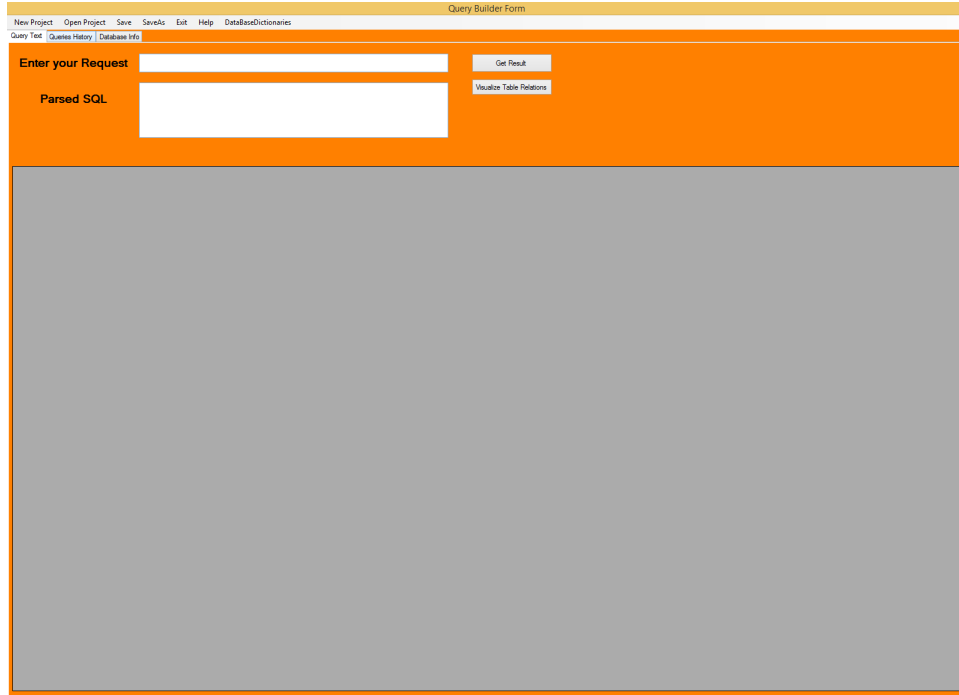


Figure A.1: System’s Main Page after a Successful Connection to a Database

A.2 Query Examples and Results

Recall that the developed system can handle three main types of questions, namely simple, complex and fuzzy. All valid questions are expected to return a result which may be either a set of records or a single value. A number of examples are presented in this section to demonstrate the output produced by different types of questions.

Figure A.2 shows an example of a question that involves the usage of an aggregate function. As previously discussed, aggregation functions involve some calculations, such as sum, count, average, etc. A single value is returned as a result when only an aggregate function appears in the “Select” clause of a SQL statement.

Figure A.3 and Figure A.4, respectively, show examples on fuzzy and complex query types. Complex questions lead to SQL queries that involve a compound condition in the “Where” clause and/or multiple tables in the “From” clause. It is also possible for a complex question to involve fuzziness which will lead to at least one fuzzy sub-condition in the “Where” clause.

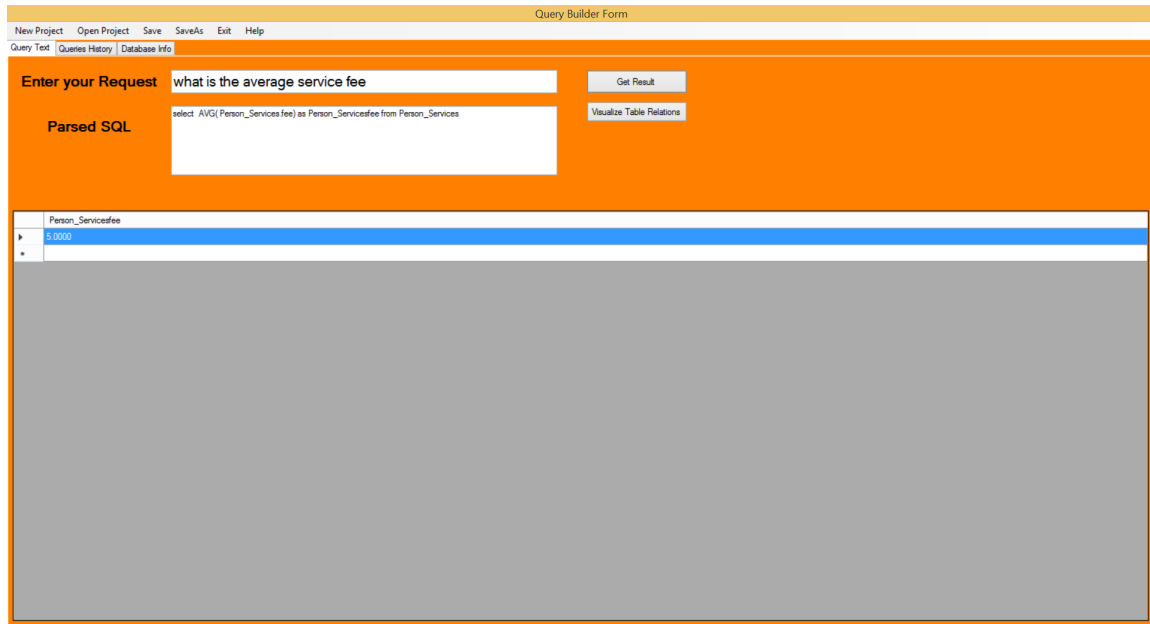


Figure A.2: A Question to Demonstrate the Usage of an Aggregate Function

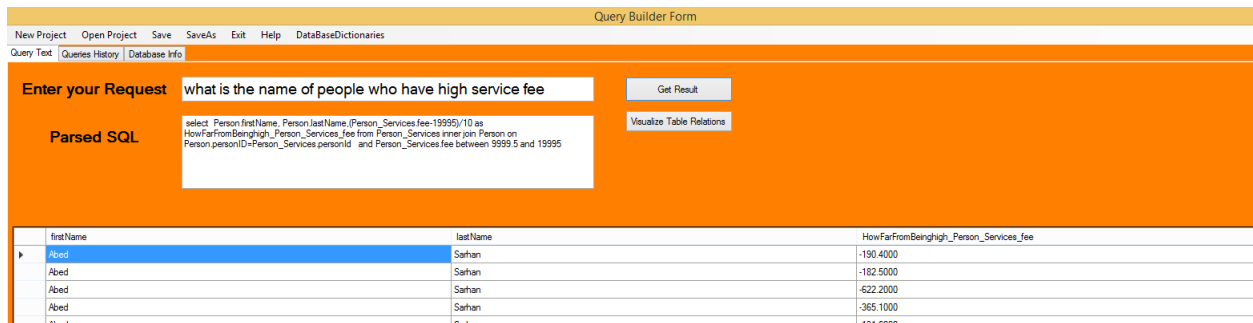


Figure A.3: A question which Involves the Usage of Fuzziness

A.3 Saving Data

Data is loaded and saved in XML format. We decided to use XML due to the availability of a built in .Net parser which performs faster than other Json parsers. A XML File is used to store the following information:

- Database Connection String.
- Database Schema information, such as names of tables and columns, types of columns, and primary keys.

New ProjectOpen ProjectSaveSaveAsExitHelpDatabaseDictionaries

Query TextQueries HistoryDatabase Info

Enter your Request

what are the services that have the highest number of

Get Result

Parsed SQL

select * from Person_Services inner join Service_Reviews on Person_Services.pServiceId=Service_Reviews.serviceId where Person_Services.pServiceId in (16662)

Visualize Table Relations

	pServiceId	personId	serviceDescripto	categoryId	fee	Longitude	Latitude	expiryDate	anyTime	visitorsNumber	creationDate	serviceReviewId	serviceId	reviewComments	personId1	reviewDate	ratingValue
▶	16662	8	I have many c...	10	7594.0000	-114.1740683	51.0883785	2/27/2016	<input checked="" type="checkbox"/>	10266	3/16/2015 3:3...	703	16662	good but there ...	10	11/5/2015 3:3...	2
	16662	8	I have many c...	10	7594.0000	-114.1740683	51.0883785	2/27/2016	<input checked="" type="checkbox"/>	10266	3/16/2015 3:3...	706	16662	Great Work	12	6/18/2015 3:3...	3
	16662	8	I have many c...	10	7594.0000	-114.1740683	51.0883785	2/27/2016	<input checked="" type="checkbox"/>	10266	3/16/2015 3:3...	712	16662	Advice Every o...	2	12/22/2015 3:3...	4
	16662	8	I have many c...	10	7594.0000	-114.1740683	51.0883785	2/27/2016	<input checked="" type="checkbox"/>	10266	3/16/2015 3:3...	718	16662	not that good	6	7/2/2015 3:39 ...	2

Figure A.4: An Example Complex Question

- Relationships between tables.
- Fuzzy domains and corresponding ranges
- Unnecessary Words to be eliminated in the analysis
- Synonyms of columns and tables
- Suffixes and prefixes
- Mapped Values which can be used for later analysis to remove ambiguity.

All these dictionaries are updated and saved upon users' need. These are saved to avoid asking users to enter the same information every time they login. This information is also useful for automated learning. The system learns from historical preferences to resolve any possible ambiguity and to serve users better.

A.4 Modules being Used

Multiple modules have been used in our system. modules are usually used as training data to help in making a specific prediction about a new item. Table A.1 shows a list of modules used for each feature in our system. These modules are loaded when users run the system, i.e., before loading or creating any project.

Table A.1: List of External Modules Used by each Feature

Feature	module name
Dependency parser	englishPCFG.caseless.ser.gz
Spelling Corrector	en_us_dic
Date Expression Detector	english-bidirectional-distsim.tagger defs.sutime.txt english.holidays.sutime.txt english.sutime.txt
Name Entity Recognizer	english.conll.4class.caseless.distsim.crf.ser.gz

A.5 System Snapshots

In this section, we will discuss the implemented feature. The main task of the splash screen shown in Figure A.5 is to show the user the progress of loading the modules. For the computing environment used in the testing, all modules need around 4 minutes load. Of course this depends on the available RAM and CPU power. The memory needed for loading these modules is around 350 MB. A caption under the progress bars shows modules that were successfully loaded.

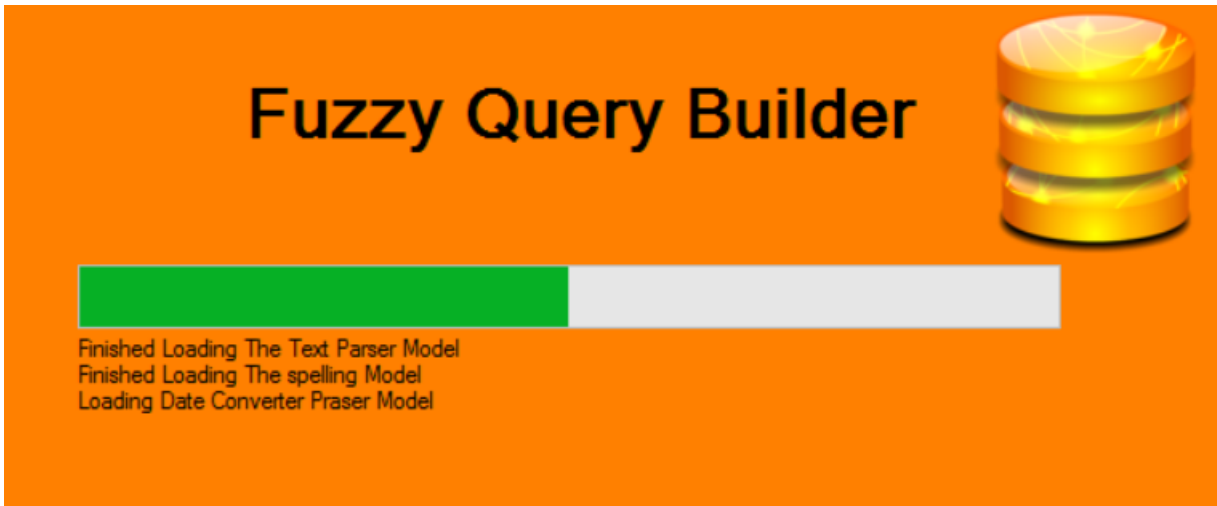


Figure A.5: System's Splash Screen

Once all modules are successfully loaded, users have the option either to load or open a project as shown in Figure A.6. If users select the option to load a file then a windows will

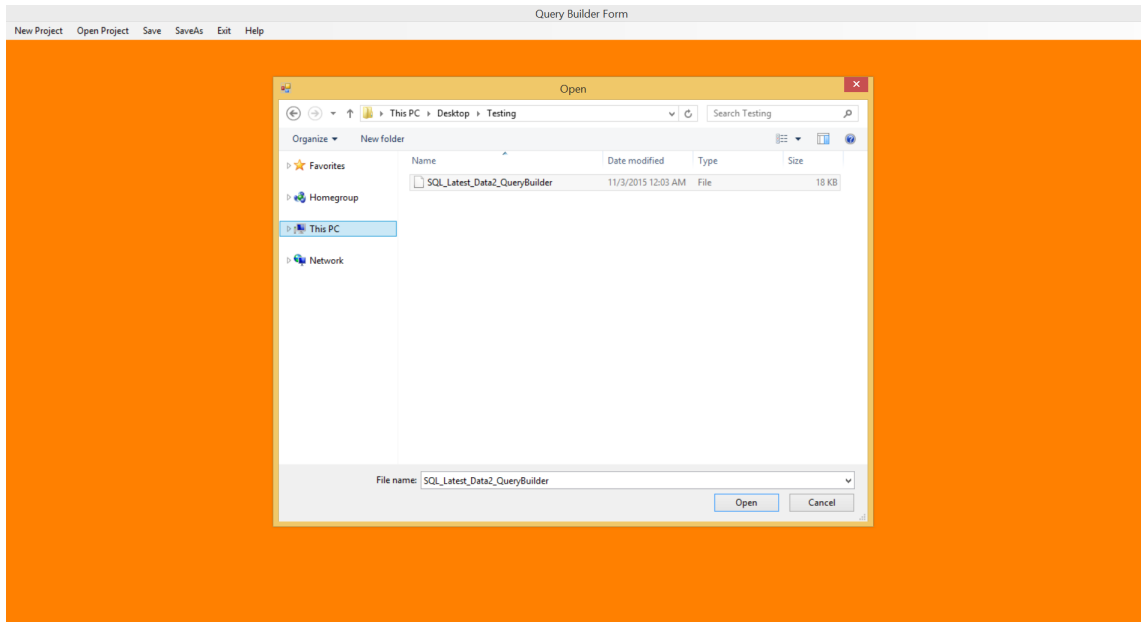


Figure A.6: The Form Used for Project Selection or Creation

pop up to inquire about the location of the XML file to be loaded. Otherwise, a connection form will appear asking for credentials of the database to connect to, as shown in Figure A.7.

A helper form is created for non-expert users to help them in building connection string. Here a user is expected to enter server credentials and the system will display a list of available databases for the user to select the database he/she wants to connect to. Figure A.6 and Figure A.8 show the helper form before and after displaying available databases, respectively.

Figure A.7: The form Used to Enter Database Information

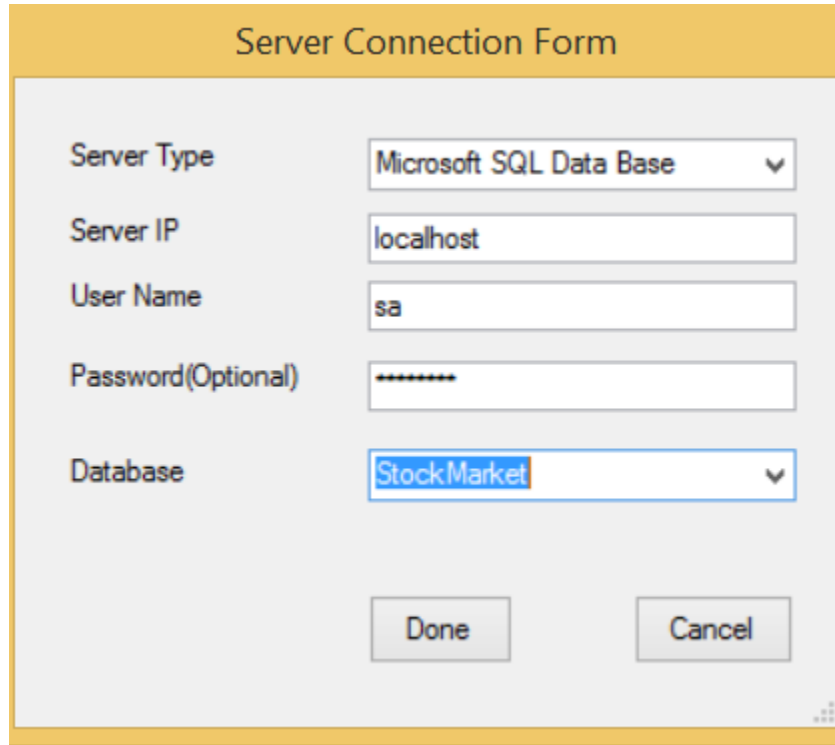
A screenshot of a 'Server Connection Form' with a yellow header. The form contains five input fields: 'Server Type' (dropdown menu showing 'Microsoft SQL Data Base'), 'Server IP' (text box with 'localhost'), 'User Name' (text box with 'sa'), 'Password(Optional)' (password box with masked characters), and 'Database' (dropdown menu showing 'StockMarket'). At the bottom are 'Done' and 'Cancel' buttons.

Figure A.8: Helper Form for Displaying a List of Available Databases

After database credentials are all specified, the system will upload and display the database schema as shown in Figure A.10. Users are given the additional opportunity to request further explanation regarding any part of the database schema (Figure A.11). This will help users in articulating their questions in a more informative way and hence will reduce the risk of errors during the parsing and validation phase. Finally, users have to click on a specific button to reflect any possible database schema updates to the corresponding XML file.

Users' input passes through multiple sentence refinements to ensure more satisfactory results. Users may get involved in the refinement process to deal with spelling mistakes. Figure A.12 shows the spell checker form used in our system to display the suggestions related to a misspelled word. However, users are not restricted to choose from the displayed list. Instead, a user may choose to either keep the current word as spelled or type a new word to replace a mistaken word. All words typed by users as replacements will be added

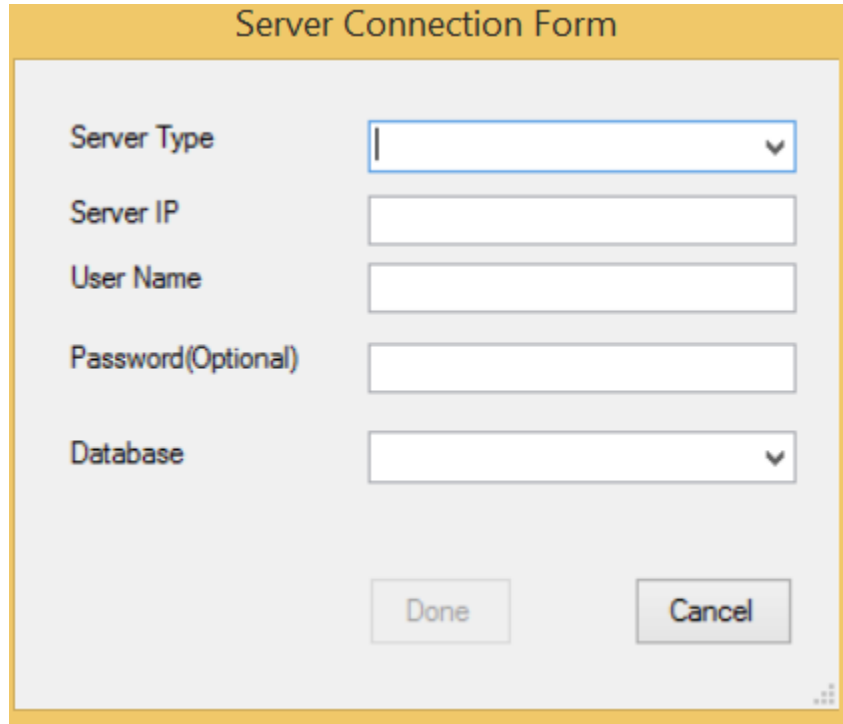
A screenshot of a 'Server Connection Form' dialog box. The form has a yellow title bar and a light gray background. It contains five input fields: 'Server Type' (a dropdown menu), 'Server IP' (a text box), 'User Name' (a text box), 'Password(Optional)' (a text box), and 'Database' (a dropdown menu). At the bottom, there are two buttons: 'Done' and 'Cancel'. The 'Server Type' dropdown is currently open, showing a list of options, though the options themselves are not clearly visible. The 'Database' dropdown is also open, showing a list of options, though the options themselves are not clearly visible.

Figure A.9: Helper Form before Displaying the List of Available Databases

to the existing list and will be used as part of future suggestions.

Multiple dictionaries should be defined for a more accurate mapping process while analyzing users' requests, such as synonyms, fuzzy domains, unnecessary words, etc. We will start with the synonym forms shown in Figure A.13 and Figure A.14, which are responsible for defining the synonyms for tables and columns, respectively. Duplicates are not allowed in the lists maintained by the system, e.g., list of synonyms, fuzzy domains, unnecessary words, etc.

Names of tables and columns might have some prefixes or suffixes, which are not necessary in the mapping process, and hence removing them will lead to a more accurate mapping. To manage this we allow users to enter their own prefixes and suffixes that were used for names of tables or columns. These are removed during the mapping. Figure A.15 shows the form for managing prefixes and suffixes; the same form allows users to add their own prefixes and suffixes, as well as remove existing ones.

Another feature supported by the system is receiving feedback from users regarding

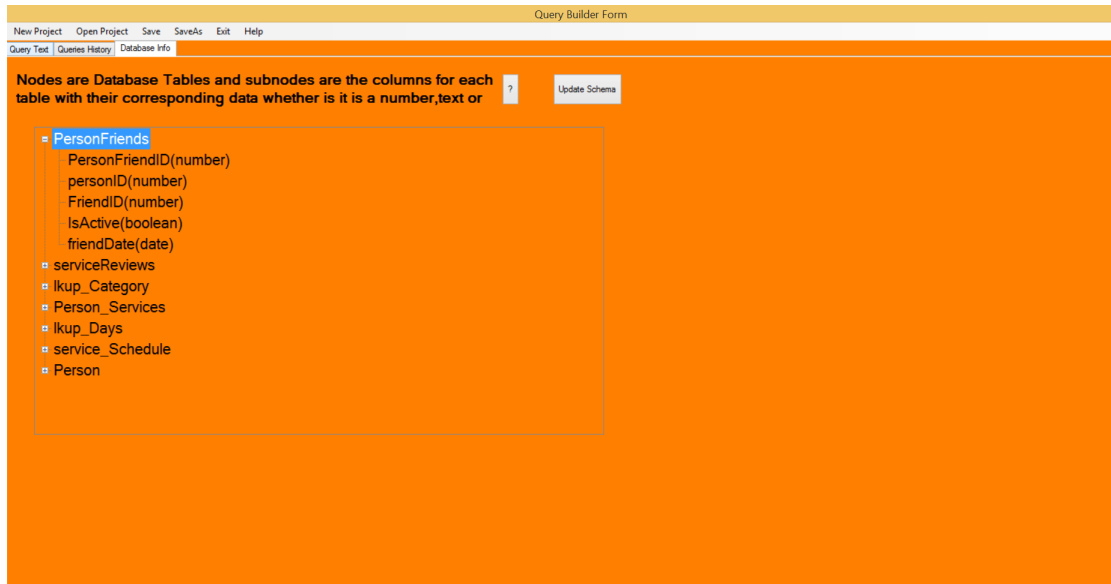


Figure A.10: Displaying Tables and Columns for the User

words deemed unnecessary. Indeed, it is not acceptable in a database environment to keep a static list of unnecessary words because relevance of words is directly related to the specific domain covered by the database. For instance, the word “From” may be the name of a table or a column in an airlines reservation database. Also, users are consulted to help in deciding whether any words that remain unmapped during the analysis should be identified as unnecessary.

Figure A.16 shows the form used to add or remove unnecessary words from a list. The form shown in Figure A.17 is used to indicate if a specific word should be eliminated from a sentence during the analysis.

Figure A.18 shows the form used for getting users’ decisions when ambiguity occurs. For instance, users should be consulted to help in resolving the ambiguity when a word has the potential to be mapped to two or more tables or columns. Users decide on the most fitting column/table for each word. The system keeps track of users’ decisions for later use, i.e., users’ choices are added to the list of mapped words. Figure A.19 shows a list of words that were discovered and mapped during the analysis; users still have the option to remove these words.

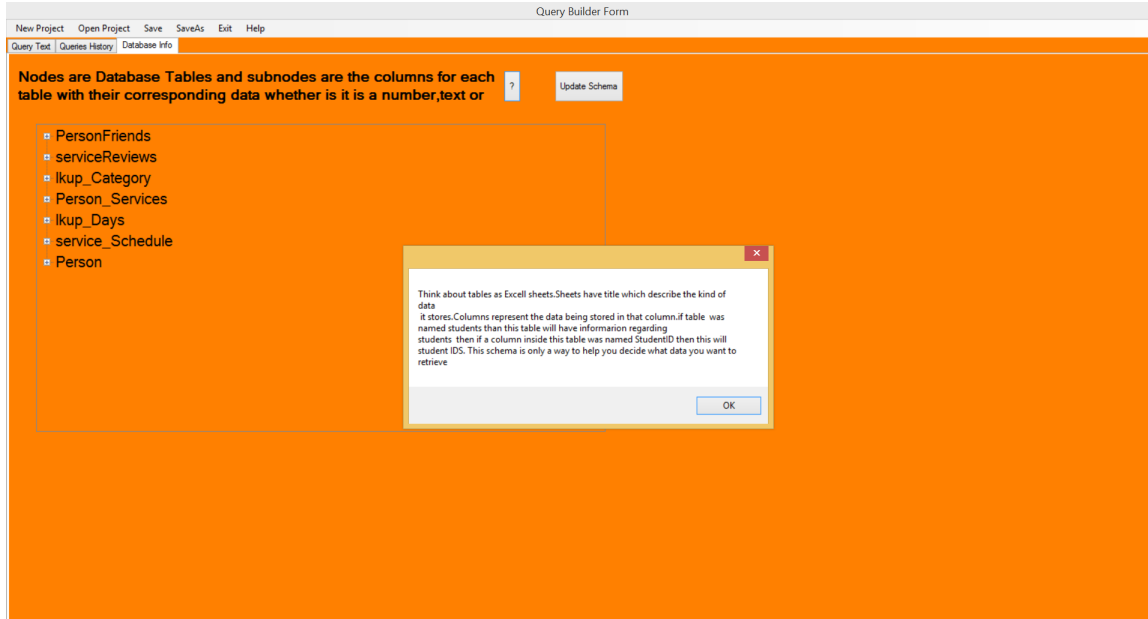


Figure A.11: Brief Explanation of Various Database Schema Elements

Fuzziness is one of the main features in our system. The form shown in Figure A.20 is used to allow users to add their own fuzzy domains and ranges. Users have the option to define their fuzzy domains and may depend on the system to automatically decide on corresponding ranges. Domains are generally specified/decided only for columns that contain numeric values.

Users can initiate the process of defining fuzzy domains by selecting a specific table with at least one column with numeric values. Table name should be entered in the combo box shown in Figure A.20. In case there exist in the table more than one column with numeric values, users are expected to choose column(s) for which fuzzy domains are to be defined. A user who decides to decide on his/her own fuzzy domains may do this by defining at least two fuzzy domains per fuzzy column. These domains can be saved by clicking the 'Save' button.

We also keep track of all requests submitted by users and successfully analyzed by the system. Figure A.21 shows some illustrative examples of past questions maintained by the system. Maintaining such record would save time and effort which were to be consumed

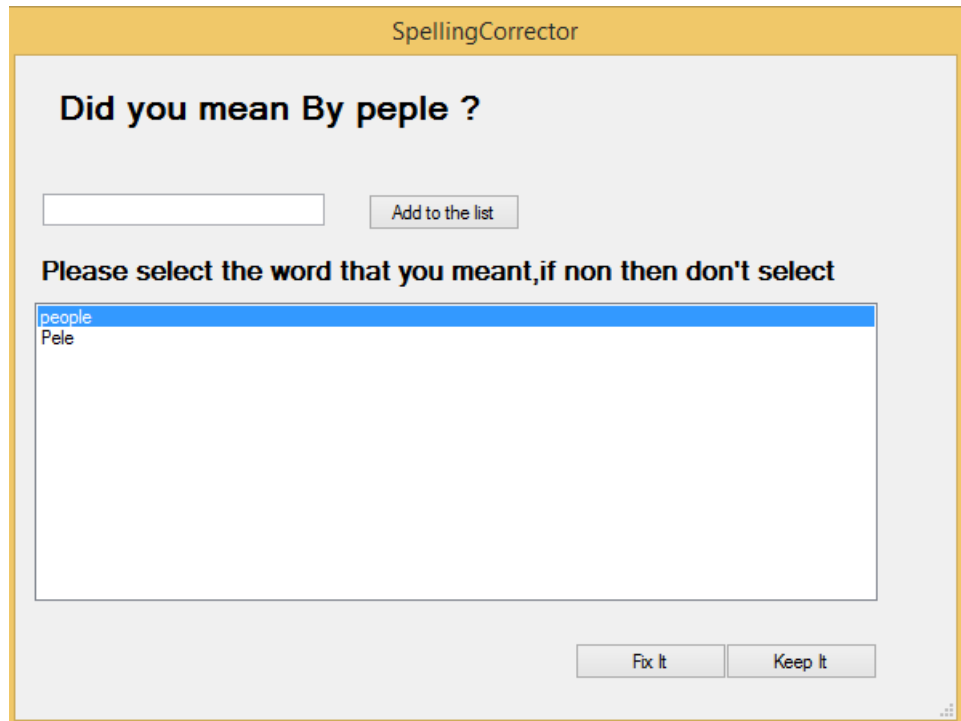


Figure A.12: Suggestions form for the Misspelled Words

in case users decide to resubmit some already processed questions. Users are expected to help the system in deciding on questions for which a record is to be maintained for future usage. This is done using the screen shown in Figure A.22. On the other hand, the system decides to keep track of questions which may help in trend analysis. The latter record should be clean from duplicates which may occupy more space without extra contribution to the analysis.

Table Synonyms Management Form

Table Name

Synonym

Add Synonym

Below is a list of tables with their Corresponding synonyms. Click on the table you want in the left listbox and it synonyms will appear in the right listbox. To delete a synonym just make a double click to be deleted.

Tables

Ikup_Category
Person_Services
Service_Reviews
Person_Friends
Person

Synonyms

Done

Figure A.13: The form that Manages Tables' Synonyms Defined by the User

96

Columns Synonym Form

Table Name

Column Name

Synonym

Add Synonym

Below is a list of columns with their corresponding synonyms. Once you select a column in the column's combo box its corresponding synonyms will appear in the listbox. Double click on a synonym in the list box to be Deleted. To add a new synonyms select the column you need to add it for and press the ADD button

Done

Figure A.14: The form that Manages Columns' Synonyms Defined by the User

Suffix Prefix Management Form

Word

☐ Prefix
 ☐ Suffix
 ☐ Prefix and Suffix

Below is a list of suffixes and prefixes used for table name or columns. These suffixes and prefixes will be eliminated from column or table name during our analysis. To delete a specific or suffix double click on it in its corresponding box and it will be deleted. Use the Add Word button

Prefixes

lkup
tbl
ASP

Suffixes

Figure A.15: The form that Manages Tables' Suffix and Prefix

Unnecessary Words Management Form

Below is a list of unnecessary words that we eliminate from a sentence when analyzing it. To delete a word double click on it and it will be deleted. User the Add Word button to add new words

Word

what
where
all
equal
when
how
number
to
many
from
we
have
in
on
for
that
there
their
he
at
she
it

Figure A.16: A list of Unnecessary Words that can be Modified by the User

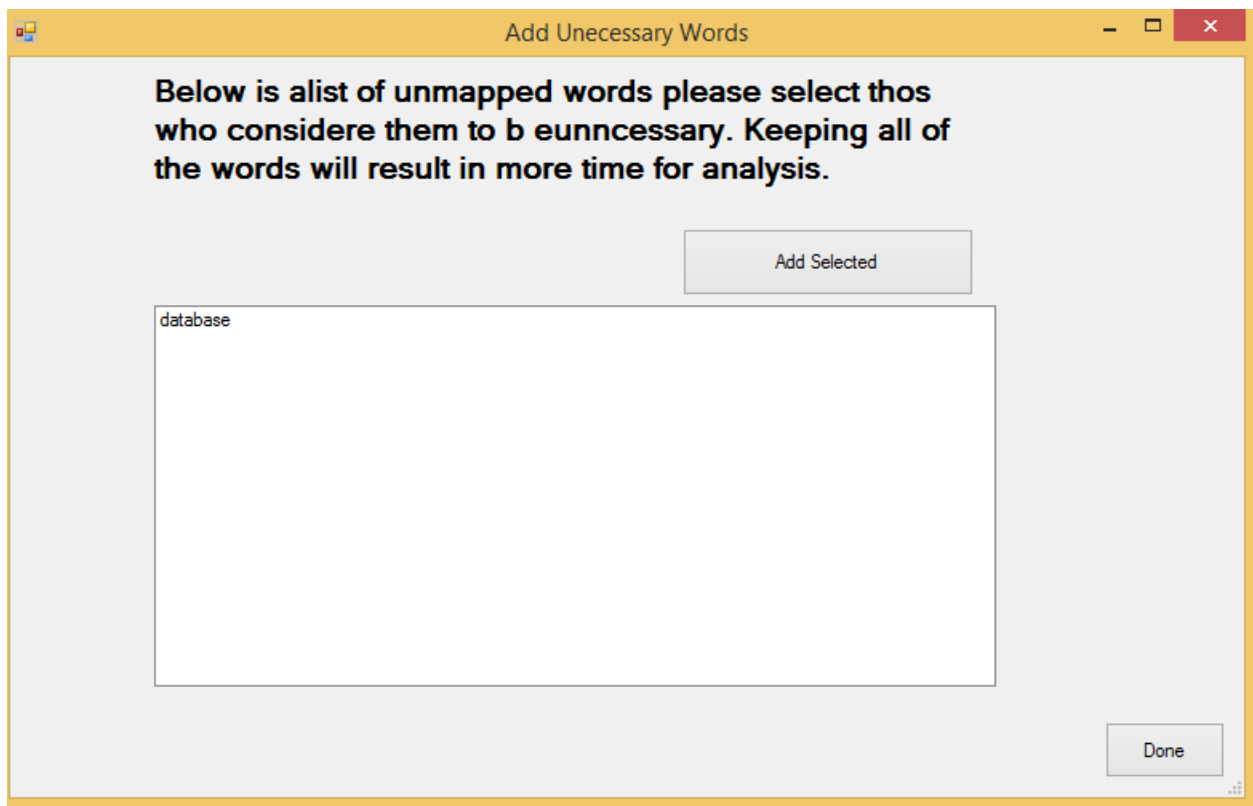


Figure A.17: Modifying Unnecessary Words during analysis

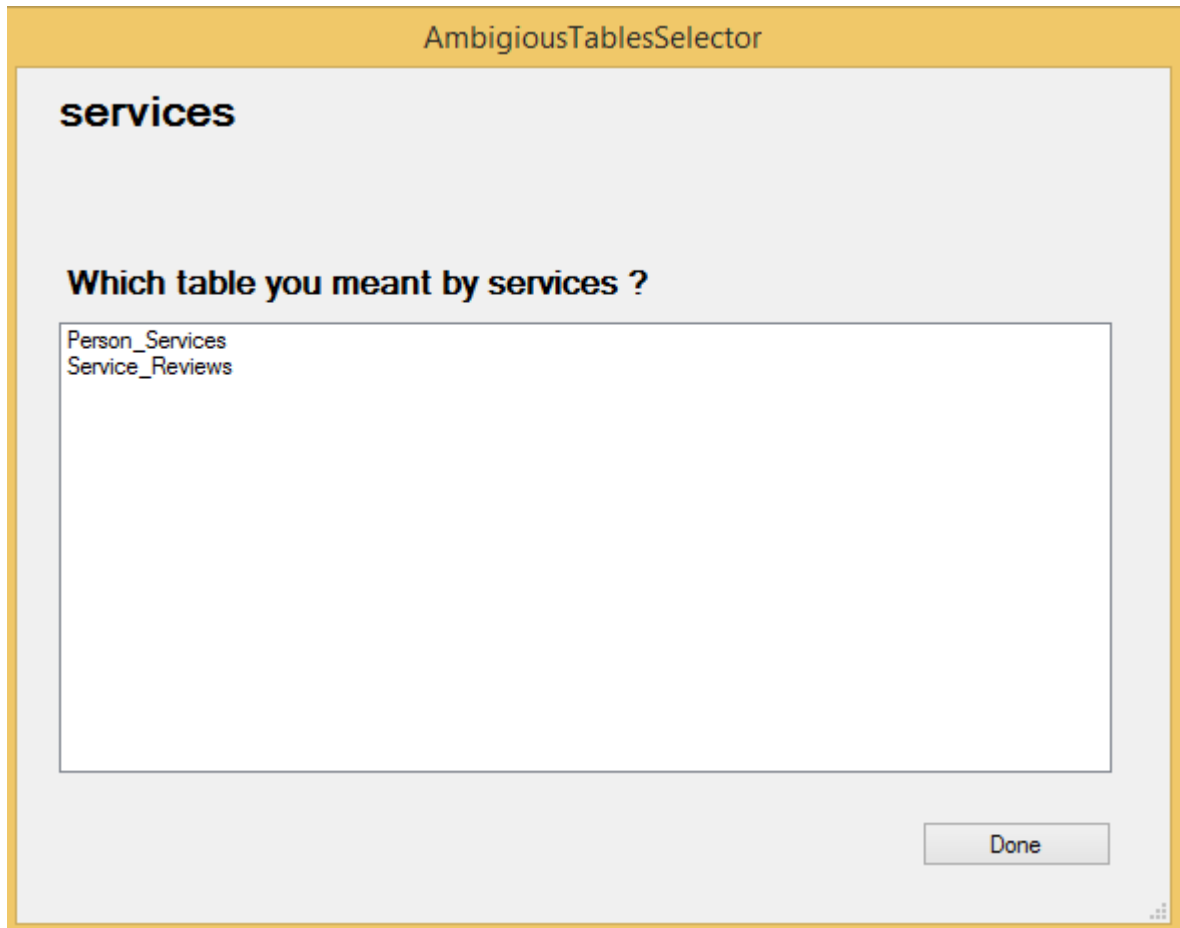
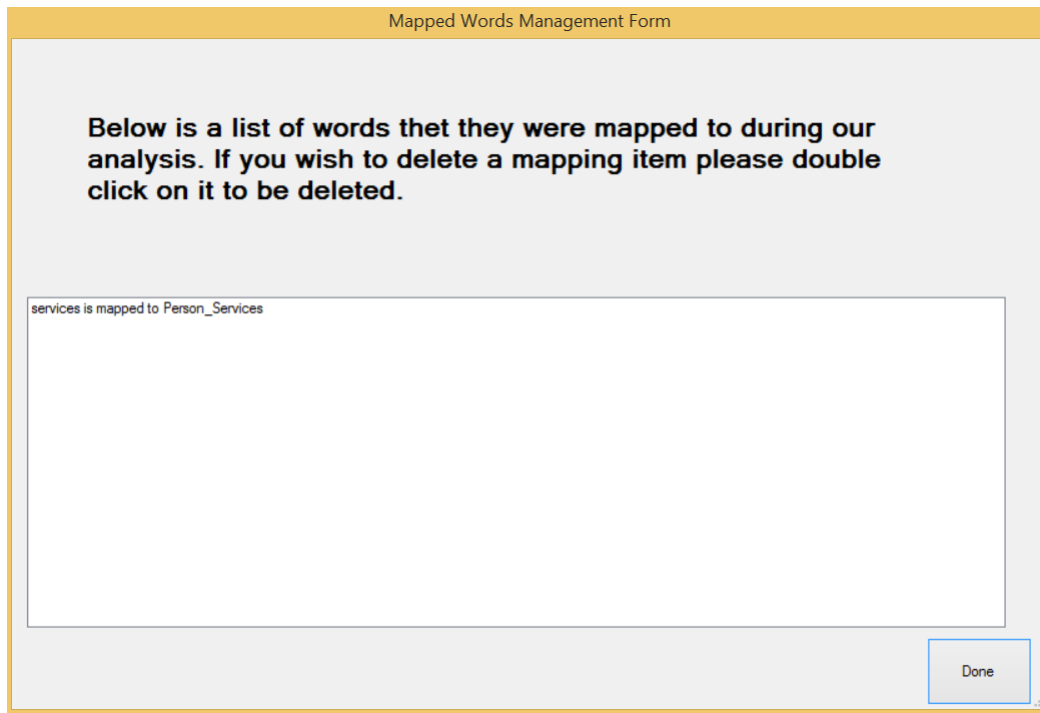


Figure A.18: Asking for User's Choice when Ambiguity Occurs



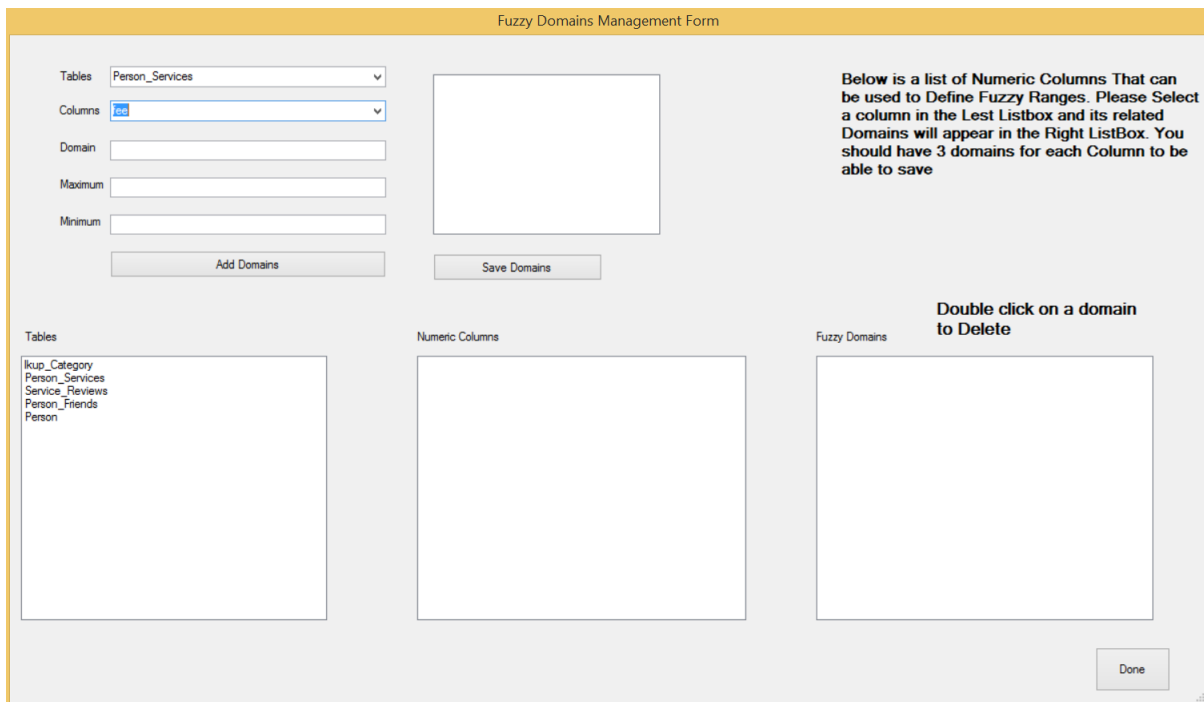
Mapped Words Management Form

Below is a list of words that they were mapped to during our analysis. If you wish to delete a mapping item please double click on it to be deleted.

services is mapped to Person_Services

Done

Figure A.19: List of Mapped Words Obtained during the Analysis



Fuzzy Domains Management Form

Tables: Person_Services

Columns: id

Domain:

Maximum:

Minimum:

Add Domains

Save Domains

Below is a list of Numeric Columns That can be used to Define Fuzzy Ranges. Please Select a column in the Left Listbox and its related Domains will appear in the Right ListBox. You should have 3 domains for each Column to be able to save

Double click on a domain to Delete

Tables

- Ikup_Category
- Person_Services
- Service_Reviews
- Person_Friends
- Person

Numeric Columns

Fuzzy Domains

Done

Figure A.20: The form Responsible for Managing Fuzzy Domains and Ranges

Query Builder Form			
New Project Open Project Save SaveAs Exit Help			
Query Test Queries History Database Info			
RequestText	RequestType	RequestSql/Information	IsUserSatisfiedBy/Result
what are the services we have	Object	select * from Person_Services	
what are the services we have with the highest service fee	Object	select * from Person_Services where Person_Services fee=(select max(Person_Ser...	
what are the people we have with the highest service fee	Object	select * from Person inner join Person_Services on Person.pID=Person_Services.p...	
what are the names of persons who do not provide any of the services provi...	Object	select Person_Services.name, Person_Services Name from Person_Services	
what are the names of persons who do not provide any of the services made...	Object	select Person_Services.name, Person_Services Name, Person.pName, Person.p...	
what are the names of persons who provide any of the services made by abed	Object	select Person_Services.name, Person_Services Name, Person.pName, Person.p...	
what are the names of persons who have friends same as abed	Object	select Person_Services.name, Person_Services Name, Person.pName, Person.p...	
which people have same friends as abed	Possession	select * from Person inner join PersonFriends on Person.pID=PersonFriends.person...	
what is the schedule of services made by abed	Object	select * from Person_Services inner join service_Schedule on Person_Services.pS...	
what is the name of people with a high service fee	Object	select Person_Services Name, Person.pName, Person.pName(Person_Services...	
what is the name of people with a low service fee	Object	select Person_Services Name, Person.pName, Person.pName(Person_Services...	

Figure A.21: A list of all Questions Made by the User

Query Builder Form	
New Project Open Project Save SaveAs Exit Help	
Query Test Queries History Database Info	
<p>Nodes are Database Tables and subnodes are the columns for each table with their corresponding data whether it is a number, text or ?</p> <div> <div> <div>PersonFriends</div> <div>serviceReviews</div> <div>lkup_Category</div> <div>Person_Services</div> <div>lkup_Days</div> <div>service_Schedule</div> <div>Person</div> </div> <div> <div>?</div> <div>Update Schema</div> </div> </div>	
<div> <div>Do You Want to Save Data?</div> <div> <div>Yes</div> <div>No</div> <div>Cancel</div> </div> </div>	

Figure A.22: Providing Users the Ability to Save their Data

Appendix B

Data Set Snapshots

As discussed in Chapter 4, the data used in testing the developed system is larger than what could fit in couple pages. Thus, we will not be able to view all database content as part of this thesis document. In this appendix, we will show snapshots that contain some illustrative records from various tables in the database. We will also describe the application dedicated for bulk insertions in the database.

B.1 Bulk Insert Application

The bulk insert application populates the database with dummy data which is intended to be used in the analysis. Figure B.1 shows a snapshot of the bulk insert application. The List box on the left-side of the screen shows a list of users; we made them unclear for privacy concerns. Three text boxes are used for inserting bulk data into the database. We first select a specific user for whom we want to add data; then we decide on the number of records we need to populate. For example, if we need to add 10 services for the first user then we put 10 in the text box beside the "Add Services" button, and then we click the button.

The bulk insert application locates and eliminates duplicates. Further, the application keeps track of time to impose some ordering on certain insert operations, e.g., to avoid generating reviews for data that does not exist. Also it validates dates to make sure that expiry date of a service does not precede its generation date.

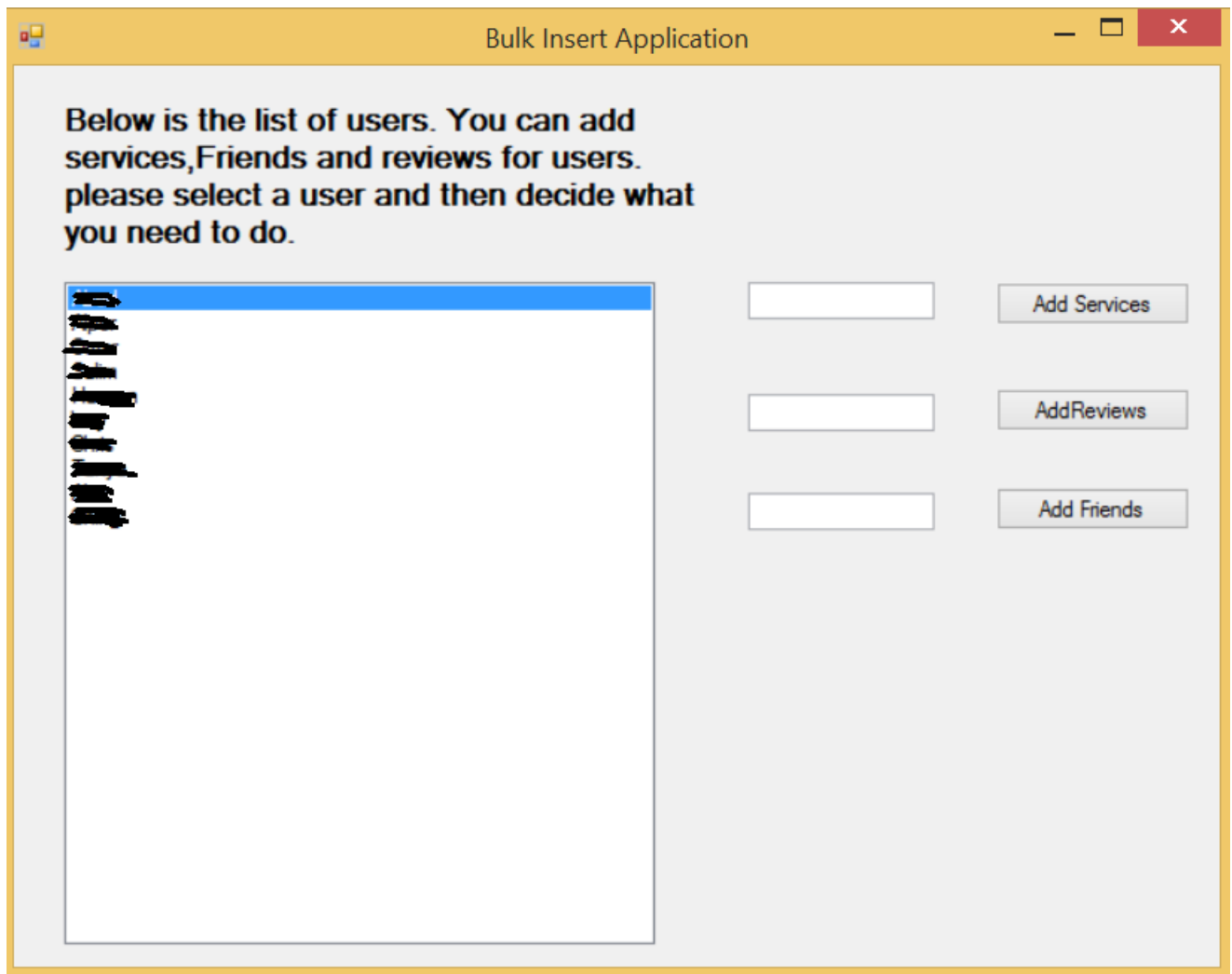


Figure B.1: A snapshot of the Bulk Insert Application

B.2 Database Tables

B.2.1 People Table Snapshot

“Person” table keeps information about people, including first name, last name, phone, address, birth date, phone number, email, etc. These attributes help in identifying a better service when needed. Each person has a unique ID which acts as primary of “Person” table. Table B.1 shows a list of records from “Person” table. For privacy reasons, we replaced actual names in the database by “First User”, “Second User”, etc.

Table B.1: List of People Available in the Database

personID	firstName	lastName	phoneNumber	Email	Password	imagePath	Degree	BirthDate	Country	City
2	First	User	██████	██████	1234	../img/UserImages/2.png	Developer	1980-12-12	Canada	Calgary
3	Second	User	██████	██████	123	../img/profileImage.png	Developer	1990-10-18	Canada	Toronto
4	Third	User	██████	██████	1234	../img/pImage.png	Developer	1995-05-05	France	Nice
6	Fourth	User	██████	██████	123456	../img/UserImages/6.jpg	Tutor	1981-10-15	Italy	Rome
8	Fifth	User	██████	██████	123@123	../img/p1.png	Developer	1975-06-06	Italy	Milan
9	Sixth	user	██████	██████	12	../img/UserImages/9.png	Developer	1993-12-23	France	Paris
10	Eighth	user	██████	██████	1234	../img/UserImages/9.png	Technician	1989-04-04 00:00:00.000	Lebanon	Beirut
11	Ninth	User	██████	██████	1234	../img/UserImages/9.png	Baby Sitter	1997-03-22	Canada	Calgary
12	Tenth	User	██████	██████	1234	../img/UserImages/9.png	Electricien	1985-07-07	Canada	Calgary
13	Eleventh	User	██████	██████	1234	../img/UserImages/9.png	Plumber	1991-01-01	China	HongKong

B.2.2 Services Table Snapshot

“Services” table stores services provided by each person in the database. The information stored for each service includes, date added, expiry date, category id, associated fee, availability, number of persons who viewed the service page, longitude and latitude. “Services” table is linked to “Person” via “personID” column and to “lkup_Category” table via “categoryId” column. A snapshot of the data stored in “Person_Services” table is shown in Table B.2, where only most important columns are viewed.

Table B.2: A Snapshot of Data Stored in Person_Services Table

personId	categoryId	fee	expiryDate	visitorsNumber	creationDate
2	3	18091.00	2016-09-13	3320	2016-03-02
2	13	18170.00	2015-10-18	1050	2014-09-26
2	4	9219.00	2016-10-20	1931	2015-07-06
2	6	13773.00	2015-05-23	24220	2014-05-17
2	1	9298.00	2015-11-23	24661	2015-06-12
2	10	4822.00	2016-05-25	101	2015-02-24
2	8	4901.00	2015-06-29	22831	2015-02-01

Continued on next page

Table B.2 – *Continued from previous page*

personId	categoryId	fee	expiryDate	visitorsNumber	creationDate
2	4	4980.00	2014-08-01	20560	2013-08-27
2	11	583.00	2016-12-01	18730	2016-01-13
2	5	741.00	2015-02-07	14188	2014-07-15
2	11	16344.00	2014-09-12	12358	2014-03-05 1
2	2	7394.00	2015-09-15	13240	2014-12-13
2	10	7551.00	2016-08-17	8699	2016-03-11
2	8	7630.00	2015-09-21	6428	2014-10-05
2	11	18679.00	2016-09-22	7310	2015-07-15
2	9	18758.00	2015-10-27	5039	2015-06-22
2	5	18837.00	2014-11-30	2768	2014-01-15
2	3	18916.00	2016-09-28	497	2016-09-18
2	4	10044.00	2016-11-04	24108	2016-01-22
2	2	10123.00	2015-12-08	21837	2014-08-16
2	8	5726.00	2015-07-14	20007	2014-04-07
2	5	5805.00	2014-08-16	17737	2014-03-15
2	13	1329.00	2015-02-16	18177	2013-11-26
2	2	5884.00	2016-06-15	15466	2015-07-05
2	11	1408.00	2016-12-16	15907	2016-07-31
2	3	17012.00	2016-07-21	14077	2016-03-21
2	1	17090.00	2015-08-25	11806	2014-10-15
2	6	12694.00	2015-03-30	9976	2014-06-05
2	2	8218.00	2015-09-30	10417	2015-07-01
2	4	12773.00	2017-01-27	7705	2015-09-25

Continued on next page

Table B.2 – *Continued from previous page*

personId	categoryId	fee	expiryDate	visistorsNumber	creationDate
2	8	3822.00	2015-05-05	8587	2015-02-19
2	5	3901.00	2017-03-04	6316	2016-06-11
2	11	19504.00	2016-10-07	4486	2016-01-31
2	13	4058.00	2015-05-11	1775	2014-12-12
2	9	19583.00	2015-11-11	2215	2014-08-26
2	12	15265.00	2014-07-20	23115	2014-03-24
2	5	10869.00	2016-11-19	21285	2016-08-09
2	13	11026.00	2015-01-26	16743	2013-09-28
2	9	6551.00	2015-07-29	17184	2014-10-24
2	5	6630.00	2014-08-31	14913	2013-05-19
2	6	17758.00	2014-10-07	13524	2014-02-03
2	2	17915.00	2015-09-09	8983	2015-05-03
2	5	8964.00	2016-09-10	9864	2016-02-09
2	13	9122.00	2014-11-18	5323	2014-08-13
2	11	9201.00	2016-09-16	3052	2015-12-02
2	1	250.00	2014-12-24	3934	2013-12-15
2	12	329.00	2016-10-23	1663	2016-08-19
2	4	15932.00	2016-05-28	24833	2016-04-08
2	13	16090.00	2014-08-04	20291	2013-05-28
2	5	11693.00	2016-12-04	18461	2015-10-14
2	6	2821.00	2017-01-09	17072	2016-06-30
2	4	2900.00	2016-02-13	14802	2015-01-24
2	6	7454.00	2014-09-15	12090	2013-12-05

Continued on next page

Table B.2 – *Continued from previous page*

personId	categoryId	fee	expiryDate	visistorsNumber	creationDate
2	1	2979.00	2015-03-18	12531	2014-12-31
2	10	18504.00	2015-09-18	12972	2014-09-14
2	12	3058.00	2017-01-15	10260	2016-04-22
2	8	18582.00	2014-10-22	10701	2014-08-22
2	2	14107.00	2015-04-23	11142	2014-05-05
2	4	18661.00	2016-08-20	8430	2015-12-12
2	13	14186.00	2017-02-20	8871	2017-01-06
2	2	18740.00	2015-09-24	6159	2014-07-07
2	11	15090.00	2016-04-10	3777	2016-02-18
2	6	10614.00	2016-10-11	4218	2015-11-02
2	9	15168.00	2015-05-14	1506	2014-09-13
2	3	10693.00	2015-11-14	1947	2015-10-09
2	12	6218.00	2016-05-16	2388	2015-06-23
2	1	10772.00	2014-12-18	24676	2014-05-05
2	10	6296.00	2015-06-20	117	2015-05-31
2	5	1821.00	2015-12-20	558	2015-02-11
2	8	6375.00	2014-07-23	22846	2013-12-24
2	2	1900.00	2015-01-23	23287	2015-01-19
2	4	6454.00	2016-05-22	20575	2015-04-15
2	13	1979.00	2016-11-22	21016	2016-05-11
2	9	17503.00	2014-08-28	21457	2013-04-27
2	11	2057.00	2015-12-26	18746	2014-12-04
2	5	17582.00	2016-06-27	19186	2015-12-30

Continued on next page

Table B.2 – *Continued from previous page*

personId	categoryId	fee	expiryDate	visitorsNumber	creationDate
2	1	13107.00	2016-12-28	19627	2015-09-13
2	3	17661.00	2015-08-01	16916	2014-07-26
2	12	13185.00	2016-02-01	17356	2015-08-21
2	1	17740.00	2014-09-03	14645	2014-07-02
2	9	13264.00	2015-03-06	15086	2014-03-16
2	4	8789.00	2015-09-06	15527	2015-04-11
2	6	13343.00	2017-01-03	12815	2016-11-17
2	2	8868.00	2014-10-10	13256	2013-11-05
2	10	4392.00	2015-04-11	13697	2014-11-30
2	12	8946.00	2016-08-08	10985	2016-07-08
2	8	4471.00	2017-02-08	11426	2016-03-22
2	10	9025.00	2015-09-12	8714	2015-02-01
2	5	4550.00	2016-03-14	9155	2016-02-28
2	13	74.00	2016-09-13	9596	2015-11-10
2	2	4629.00	2015-04-17	6884	2014-09-22
2	11	153.00	2015-10-18	7325	2014-06-06
2	6	15678.00	2016-04-19	7766	2015-07-02
2	9	232.00	2014-11-21	5054	2014-05-14
2	3	15757.00	2015-05-24	5495	2014-01-26
2	5	311.00	2016-09-19	2783	2015-09-03
2	1	15836.00	2017-03-22	3224	2016-09-28
2	10	11360.00	2014-12-27	3665	2013-09-16
2	12	15914.00	2016-04-25	954	2015-04-24

Continued on next page

Table B.2 – *Continued from previous page*

personId	categoryId	fee	expiryDate	visitorsNumber	creationDate
2	6	11439.00	2016-10-26	1394	2016-05-20
2	2	6964.00	2014-08-01	1835	2013-05-07
2	4	11518.00	2015-11-29	24124	2014-12-13
2	13	7042.00	2016-05-31	24565	2016-01-09
2	2	11597.00	2015-01-02	21853	2014-11-21
2	10	7121.00	2015-07-05	22294	2014-08-04

B.2.3 Category Look Up Table Snapshot

Each service in the database is linked to a category which describes its kind the most. Users can view group services based on a specific category they are interested in. Table B.3 shows all categories defined in the database. For the analysis, the word “LKup” was defined as a prefix in order to help in conducting a more accurate mapping.

Table B.3: A Snapshot of Data Stored in LKUP_Category Table

categoryId	categoryDescription	imagePath
1	Electrician	../img/icon/Electrician.png
2	Tutor	../img/icon/Tutor.png
3	Health Care	../img/icon/HealthCare.png
4	Child Care	../img/icon/ChildCare.png
5	Plumber	../img/icon/Plumber.png
6	Cleaning	../img/icon/Cleaning.png
8	Constructor	../img/icon/HealthCare.png
9	Gardener	../img/icon/Gardner.png
10	General	../img/icon/General.png

Continued on next page

Table B.3 – *Continued from previous page*

categoryId	categoryDescription	imagePath
11	Scientist	../img/icon/General.png
12	Technician	../img/icon/General.png
13	Translator	../img/icon/General.png
14	lawyer	../img/icon/General.png

B.2.4 Service Reviews Table Snapshot

It has become a common practice to add a feedback option to systems that provide some services to users, regardless whether a service is free or has an associated fee. Such option allows users to share their experience with others including providers who may depend heavily on feedback to improve their service and in their marketing campaigns.

The developed system provides the feedback option only to registered users, i.e., only to users recorded in the database. Users are allowed to provide a rating as part of the services related feedback captured by the system. Some sample reviews maintained by the system are shown in Table B.4. Each review has a comment expressed as free text, associated date, and rating value.

Table B.4: A Snapshot of some Sample Reviews Maintained in Service_Reviews Table

serviceId	reviewComments	personId	reviewDate	ratingValue
15714	good but there are better people	4	8/6/2016	1
15252	good but there are better people	8	9/14/2015	0
17807	Perfect	8	11/25/2015	3
15062	Great Work	3	10/5/2015	4
14965	Advice Every one to Work with him	6	10/13/2015	4
16266	not that good	8	8/14/2013	2

Continued on next page

Table B.4 – *Continued from previous page*

serviceId	reviewComments	personId	reviewDate	ratingValue
15371	not that good	4	8/10/2015	0
15569	not that good	4	7/12/2015	0
15399	perfect Service	11	8/28/2016	4
16049	good but there are better people	11	11/26/2015	2
16115	Great Work	3	5/26/2015	4
15398	perfect Service	3	9/1/2015	4
16181	not that good	11	1/10/2014	2
16209	not that good	6	2/8/2015	2
17626	not that good	11	6/13/2016	0
15319	Perfect	9	11/25/2014	3
14962	Great Work	10	8/12/2013	3
15311	Great Work	10	1/16/2016	3
15494	perfect Service	6	10/5/2014	4
15751	not that good	3	9/16/2016	1
15555	not that good	11	8/14/2015	1
17752	Advice Every one to Work with him	12	1/9/2016	4
16082	Advice Every one to Work with him	9	8/25/2015	4
15058	not that good	9	3/27/2014	1
15065	good but there are better people	3	8/1/2015	1
15826	perfect Service	11	4/18/2015	3
15300	perfect Service	3	6/25/2015	3
15385	not that good	3	5/26/2017	2
15437	Great Work	12	3/9/2014	4

Continued on next page

Table B.4 – *Continued from previous page*

serviceId	reviewComments	personId	reviewDate	ratingValue
16249	not that good	9	4/30/2014	0
17863	good but there are better people	8	3/20/2017	1
15545	not that good	6	6/18/2015	0
15130	good but there are better people	8	6/2/2015	2
15888	Great Work	6	12/28/2015	4
15022	perfect Service	10	11/11/2015	4
15249	Advice Every one to Work with him	8	10/7/2015	3
17901	good but there are better people	10	9/8/2014	2
16209	good but there are better people	9	3/10/2015	1
17761	perfect Service	12	3/24/2014	4
17669	Great Work	3	1/3/2015	4
15615	Advice Every one to Work with him	12	7/16/2014	3
15309	good but there are better people	11	6/14/2014	2
17809	good but there are better people	4	9/15/2014	1
15596	good but there are better people	10	5/11/2015	2
17806	Advice Every one to Work with him	11	7/20/2016	3
15080	good but there are better people	12	6/23/2016	1
15167	perfect Service	11	2/19/2016	3
15029	Advice Every one to Work with him	10	1/2/2016	4
17717	Advice Every one to Work with him	11	6/11/2015	3
16246	perfect Service	9	5/12/2014	4
15297	good but there are better people	6	3/2/2016	0
16006	Advice Every one to Work with him	9	4/8/2016	4

Continued on next page

Table B.4 – *Continued from previous page*

serviceId	reviewComments	personId	reviewDate	ratingValue
15382	perfect Service	6	11/18/2014	3
15039	perfect Service	12	10/30/2016	3
16174	good but there are better people	11	6/5/2016	0
15104	perfect Service	8	1/17/2017	4
15887	not that good	6	9/18/2014	0
16290	Great Work	10	4/23/2014	4
15684	good but there are better people	12	6/4/2016	1
15869	Great Work	9	10/13/2015	4
16329	perfect Service	6	4/13/2014	4
15521	Perfect	3	4/23/2014	3
15260	Advice Every one to Work with him	12	8/8/2016	4
17744	Perfect	3	9/15/2014	4
15625	good but there are better people	6	9/22/2013	1
16177	Great Work	8	11/29/2015	3
15857	not that good	11	1/18/2014	1
15292	Perfect	9	4/22/2015	4
16108	not that good	6	3/15/2014	1
15830	perfect Service	3	7/13/2014	3
15678	Create Work	3	12/19/2013	3
15744	Create Work	12	10/17/2015	4
17847	Create Work	10	8/26/2014	4
15567	good but there are better people	6	8/7/2016	2
15466	good but there are better people	8	2/27/2017	2

Continued on next page

Table B.4 – *Continued from previous page*

serviceId	reviewComments	personId	reviewDate	ratingValue
15038	Perfect	11	7/7/2015	4
16046	good but there are better people	4	4/5/2016	1
15788	perfect Service	10	2/17/2015	3
15390	good but there are better people	12	2/23/2015	1
17894	not that good	10	7/9/2014	1
15373	Great Work	6	8/10/2015	3
14953	not that good	4	7/28/2013	1
16160	Perfect	3	3/20/2016	3
16231	good but there are better people	3	10/20/2015	0
16131	Great Work	12	11/22/2014	4
15674	perfect Service	12	10/31/2014	3
15550	Perfect	4	11/23/2015	3
15489	perfect Service	4	5/28/2016	4
15452	perfect Service	3	3/28/2016	3
15387	not that good	10	10/9/2015	0
17762	Great Work	8	7/5/2016	3
15530	Advice Every one to Work with him	9	4/6/2016	4
15025	perfect Service	10	1/21/2015	3
15764	good but there are better people	10	7/7/2015	2
15932	Advice Every one to Work with him	8	6/30/2015	4
16045	not that good	11	8/19/2016	2
16205	Great Work	12	7/10/2014	4
17715	Perfect	4	2/3/2015	4

Continued on next page

Table B.4 – *Continued from previous page*

serviceId	reviewComments	personId	reviewDate	ratingValue
16242	Perfect	9	9/25/2017	3

B.2.5 Friends Table Snapshot

“Friends” table keeps information about who is a friend of whom. This kind of relationship may increase the trust in services which have been positively reviewed by friends. Only first level of direct friendship is maintained and explicitly stored. All other levels of friendship (e.g., friend of a friend, etc.) may be extracted from by running some queries that simulate recursion which is not explicitly supported as part of SQL. A snapshot of the “Friends” table is shown in Table B.5.

Table B.5: A Snapshot of Data Stored in Person_Friends Table

personId	friendId	isActive	friendshipdate
2	11	true	4/5/2015
2	9	true	3/27/2015
2	12	true	3/28/2015
2	6	true	8/20/2014
2	3	true	9/17/2015
2	4	true	9/17/2014
4	10	true	10/3/2014
4	3	true	12/16/2014
6	9	true	4/20/2015
6	10	true	7/30/2014
6	3	true	7/28/2014
13	11	true	4/13/2015

Continued on next page

Table B.5 – *Continued from previous page*

personId	friendId	isActive	friendshipdate
13	6	true	3/2/2015
13	2	true	8/22/2015
13	3	true	3/14/2015
10	9	true	2/20/2015
10	3	true	1/3/2015
10	8	true	2/1/2015
10	11	true	12/25/2014
10	2	true	1/25/2015
10	12	true	7/23/2014
12	11	true	2/17/2015
12	4	true	2/13/2015
12	6	true	10/2/2014
12	9	true	5/7/2015
12	3	true	11/3/2014
8	3	true	9/21/2015
8	2	true	7/17/2014
8	6	true	5/25/2015
8	9	true	5/6/2015