HINDI LANGUAGE INTERFACE TO DATABASES

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Himani Jain
(Roll No. 800932009)

Under the supervision of:
Mr. Parteek Bhatia
Assistant Professor (CSED)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2011

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Hindi Language Interface to Databases*", in partial fulfillment of the requirements for the award of degree of *Master of Engineering in Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mr. Parteek Bhatia and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Himani Jain)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Parteek Bhatia)
Assistant Professor
Computer Science and Engineering Department,
Thapar University, Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

First of all, I would like to express my gratitude towards **THAPAR UNIVERSITY,** for providing me a platform to do my thesis work at such an esteemed institute.

I wish to express my deep gratitude to Mr. Parteek Bhatia, Assistant Professor, Computer Science and Engineering Department, Thapar University, Patiala for his valuable advice and guidance in carrying out my thesis.

I would like to thank Dr. Maninder Singh, Head, Computer Science and Engineering Department, Thapar University, Patiala who has been a constant source of inspiration for me throughout this work.

I am also thankful to all the staff members of the Department for their full cooperation and help.

I am Thankful to my parents, brother and all my friends for their blessing and moral support. Thanks for boosting me with their constant encouragement, support and confidence.

Last but not the least, I am thankful to God for providing me with the strength and ability to complete my work.

Himani Jain

Abstract

Database management systems have been widely used for storing and retrieving data. However, databases are often hard to use since their interface is quite rigid in cooperating with users. The analysis of existing Natural Language Interface to Databases has been carried out. Most of the systems are based on pattern-matching, syntax based, semantic based, intermediate representation languages architectures. The Employee database has been identified as a case study for developing Hindi language interface to databases. This Employee database contains Employee and Department tables. A system has been developed in which user can input query in Hindi language and can see the result in the same language. The proposed system can handle single and multiple column retrieval queries, conditional queries and join queries.

A graphical user interface has been designed for this system with the use of JAVA Swings. MYSQL has been used as database and it is connected to JAVA using *mysql-connector-java*. The proposed system can work on UNIX platform. It has been tested thoroughly on 50 Hindi queries which includes select query, update and delete queries. The results of testing are very encouraging.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The community of potential information system users is growing rapidly with advances in hardware and software. This permits computerization in a large number of application areas. Some of the areas where computerization is needed are railways, weather forecasting, library services, information management systems, *etc.* [1]. An end user normally doesn't know SQL. So, in order to interact with the system, a graphical user interface has been used. This graphical user interface still requires some basic training for using the system. It will be always better if an end user is able to interact with the system in his/her native natural language. This gives the idea of Natural Language Interface to Databases (NLIDB). It is a type of computer human interface. With the help of this system, the end user can query the system in natural language like English, Hindi, *etc*., and can see the results in the same language. Natural Language Interface to Databases is a basically a field of Natural Language Processing (NLP) which has been discussed in next section.

1.1 Natural Language Processing (NLP)

NLP is a field of computer science and linguistics concerned with the interactions between computers and human (natural) languages. In theory, NLP is a very attractive method of human-computer interaction. Natural language understanding is sometimes referred to as an AI-complete problem because it seems to require an extensive knowledge about the outside world and the ability to manipulate it. NLP has significantly overlapped with the field of computational linguistics, and is often considered a sub-field of artificial intelligence.

The foundation of NLP lies in a number of disciplines like computer and information sciences, linguistics, mathematics, electrical and electronic engineering, artificial intelligence and robotics, psychology, agriculture, weather forecasting, *etc*. Applications of NLP include a number of fields of studies, such as machine translation, natural language interface to databases, natural language text processing and summarization, user interfaces, multilingual and Cross Language Information

Retrieval (CLIR), speech recognition, artificial intelligence and expert systems, and so on [2].

1.2 Natural Language Interface to Databases

A person with no knowledge of database language may find it difficult to access database easily. Therefore, SQL Tutor was developed for analyzing the abilities of Natural Language Processing to develop products for people to interact with the database in simple English. These, these products have created a revolution in extracting information from databases. They have discarded the fuss of learning SQL and time is also saved in learning this query language.

1.3 Advantages of NLIDB system

The advantages of Natural Language Interface to Databases have been discussed below.

- **No need to learn artificial language**

  One advantage of NLIDBs is supposed to be that the user is not required to learn an artificial communication language. Formal query languages are difficult to learn and master, at least by non-computer-specialists. Graphical interfaces and form-based interfaces are easier to use by occasional users, still invoking forms, linking frames, selecting restrictions from menus, *etc*., constitute the artificial communication languages that have to be learned and mastered by the end-user. In contrast, an ideal NLIDB allows users to enter queries in their native language.

- **Easy and efficient retrieval**

  Since, people can deal with their native language in which they have good grasp, this makes NLIDB quite easy and efficient to use.

- **Easy to handle negation and quantification type of questions**

  There are some kinds of questions (*e.g.*, questions involving negation, or quantification) that can be easily expressed in natural language, but that seem difficult (or at least tedious) to express using graphical or form-based interfaces. For example, “Which department has no programmers?” (Negation), or “Which company supplies every department?” (Universal quantification), can be easily expressed in natural language, but they would be difficult to express in most graphical or form-based interfaces.

- **Easy to deal with joining of multiple tables**

  Queries that involve multiple database tables like "List the address of the employee whose department is Sales", are difficult to form in graphical user interface as compared to Natural Language Interface to Database [3].

1.4 Disadvantages of NLIDB system

The disadvantages of Natural Language Interface to Databases have been discussed below.

- **Deals with limited set of natural languages**

  Users generally don't understand the linguistic, *i.e.*, language related capability of natural languages. Presently, NLIDBs can only deal with limited subsets of natural language. Users find it difficult to understand what kinds of questions the NLIDB can or cannot deal with [4].

- **Confusion for users**

  When the NLIDB cannot understand a question, it is often not clear to the user whether the rejected question is outside the system's linguistic coverage or whether it is outside the system's conceptual coverage. Some NLIDBs attempt to solve this problem by providing diagnostic messages, showing the reason a question cannot be handled [4].

- **Wrong assumptions**

  Users assume if a system can provide the natural language access to a database, it can deduce other facts from the information facts that are not explicitly stated, but are obvious to anyone with common sense. But this is not the case [3].

1.5 Applications of using Hindi language interface to databases

Hindi is spoken mostly in northern and central India, Pakistan, Fiji, Mauritius, and Suriname. Approximately, six hundred million people speak Hindi, as either the first or the second language. It is the official language of India. Large number of e-governance applications use databases. So, to easily access databases, query given by users should be in Hindi language. For this, a system should develop which accept a sentence in Hindi, and process it to generate a SQL query and produce the desired result in Hindi only. Some of the areas have been indentified where Hindi language interface to databases can be applied.

1. **Railways: Hindi language interface to railways databases**

   Since the railways are a public transport, serving people from different regional, ethnic, and linguistic groups, the policy of the organization has been geared towards communicating with its passengers using their language and script. As Hindi is most common language used in India, so there is a need for developing Hindi language interface to railways databases. As the passengers have lots of queries related to reservation, timings, cancellation of journey, *etc*., so the queries which they want to put, should be in their own native language. Passengers give query in Hindi and get the result in Hindi itself. This is where Hindi language interface to railways databases is used.

2. **Agriculture: Hindi language interface to agriculture databases**

   Agriculture in India is the means of livelihood of almost two thirds of the work force. Government has developed many systems to help farmers solving their queries regarding irrigation and other queries. Data related to agriculture is stored in databases. The farmers are not aware how to deal with these databases because they don't understand SQL queries. So, there should be such system which is friendly with farmers. Farmers give their queries in Hindi and the result in the same language. This is all done with the help of Hindi language interface to agriculture database.

3. **Weather forecasting: Hindi language interface to weather databases**

   There is also great use of Hindi language interface to databases in weather forecasting. Weather report is necessary for almost every person. So, providing the queries related to weather in Hindi is very important. Farmers may easily get to know about the weather details by asking the query in Hindi. This is done with the help of Hindi language interface of weather database. Presently there is a vast amount of NLP-based research carried out for the development of such systems. One modern natural language system is JUPITER**.**

4. **Legal matters: Hindi language interface to legal databases**

   Lawyers also need to maintain database for their clients. They use Hindi language in their daily activities. They usually do not have any knowledge about formal query language like SQL and they don't have the enough time to learn these query languages. They just want a system which can process their

request easily. For them, Hindi language interface to legal databases can be particularly useful.

5. **Employees: Hindi language interface to employee databases**

   For employees who don't have good knowledge of query languages and English language, this Hindi language interface to employee database provides them a lot of help. They just give their queries in their Hindi language and get the result in the same language.

Chapter 2

Literature Survey

There is a large development in the field of NLIDB. Researchers started doing work on it from early seventies. Many researchers like Androutsopoulos, G.D. Ritchie and P. Thanisch has suggested various architectures for NLIDB which has been discussed in next section.

2.1 Various architectures used by researchers

Researchers have applied a number of architectures in different cases. These are pattern-matching systems, syntax based systems, semantic grammar systems, intermediate representation languages which have been discussed below.

2.1.1 Pattern-matching systems

In pattern-matching systems, some patterns, and rules are initially given. These patterns and rules are fixed. The rules states that if an input sentence or word is matched with the given pattern, an action has been taken. These actions are also mentioned in the database. Users work according to these actions. This approach is limited to specific databases and to the number and complexity of patterns [5]. The main advantage of the pattern-matching approach is its simplicity. In such systems, no parsing and modules are needed and systems are easy to implement [5]. Some pattern matching systems were able to perform impressively well in some applications but shallowness of pattern matching approach would lead to bad failures [6]. SANVY is a good example of pattern- matching systems [7].

2.1.2 Syntax-based systems

In syntax-based systems the user's question is parsed (*i.e.*, analyzed syntactically) and the resulting parse tree is directly mapped to an expression in some database query language. A good example of syntax based system is LUNAR [8]. Syntax-based systems use a grammar that describes the possible syntactic structures of the user's questions [9]. Considering an example of syntax based system, the grammar is shown in (2.1).

S → NP VP

NP → Det N

Det → “what” | “which”

N → “tree” | “mangoes” |

VP → V N

V → “contains” | “grows” ... (2.1)

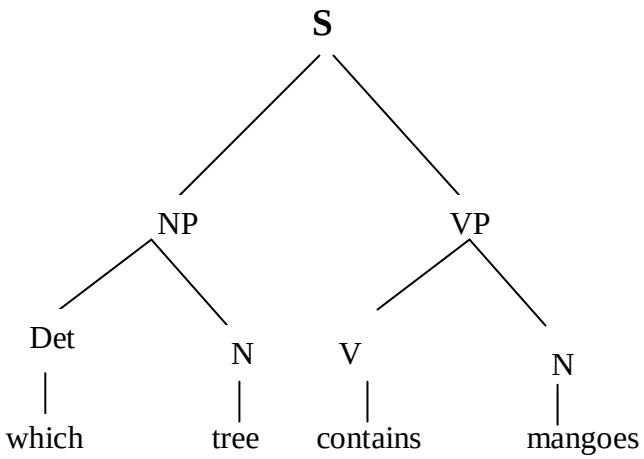

Figure 2.1: Parse tree in a syntax-based system

As shown in (2.1), a sentence (S) consists of a noun phrase (NP) followed by a verb phrase (VP). A NP is again divided into determiner (Det) followed by a noun (N). The VP is divided into Verb (V) followed by a Noun (N). Using this grammar, a NLIDB could figure out that the syntactic structure of the question “which tree contains mangoes” is shown in the parse tree of Figure 2.1. This mapping would be carried out by some rules, and would be completely based on the syntactic information of the parse tree [7].

The main advantage of using syntax based approach is that it provides detailed information about the structure of a sentence. A parse tree contains a lot of information about the sentence structure, starting from a single word and its part of speech, how words can be grouped together to form a phrase, how phrases can be grouped together to form more complex phrases, until a whole sentence is built [5].

2.1.3 Semantic grammar systems

A semantic grammar system is very similar to the syntax based system. It means that the query result is obtained by mapping the parse tree of a sentence to a database query. The basic idea of a semantic grammar system is to simplify the parse tree as much as possible, by removing unnecessary nodes or combining some nodes together. Semantic grammar systems are less ambiguous as compared to syntax based systems. The main drawback of semantic grammar approach is that it requires some prior-knowledge of the elements in the domain, therefore making it difficult to port to other domains. In addition, a parse tree in a semantic grammar system has specific structures and unique node labels, which could hardly be useful for other applications [5]. Semantic grammars are used in PLANES [10] and LADDER [11].

2.1.4 Intermediate representation languages

Most current NLIDBs first transform the natural language question into an intermediate logical query, expressed in some internal meaning representation language. The intermediate logical query expresses the meaning of the user's question in terms of high level world concepts, which are independent of the database structure. The logical query is then translated to an expression in the database's query language, and evaluated against the database. Figure 2.2, shows architecture of an intermediate representation language system.

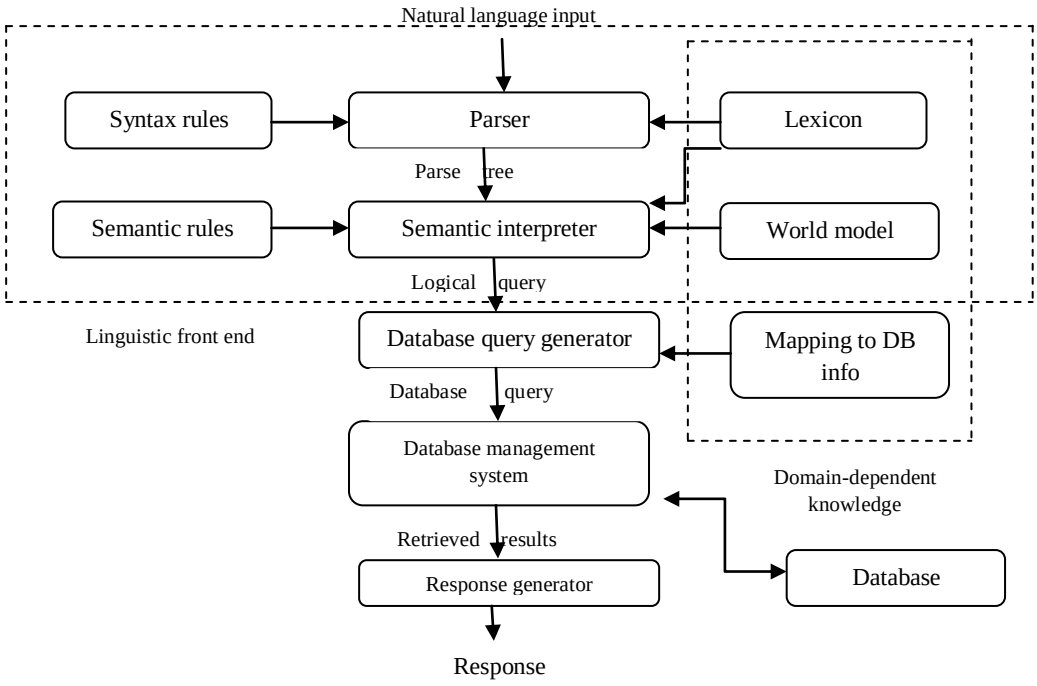

Figure 2.2: Architecture of Intermediate representation language system [7]

The natural language input is first processed syntactically by the parser. The parser consults a set of syntax rules, and generates a parse tree. The semantic interpreter subsequently transforms the parse tree to the intermediate logic query, using semantic rules. The semantics rule computes the logic expression of the constituent in the left-hand side of the syntax rule, as a function of the logic expressions of the constituents in the right-hand side of the syntax rule. The logic expressions corresponding to words are declared in the lexicon. In order to retrieve the information requested by the user, the logic query has to be transformed into a query expressed in some database query language supported by the underlying Database Management System (DBMS). This transformation is carried out by the database query generator, using the mapping to database information. The database query generated by the database query generator is passed to the underlying DBMS, which executes the query against the database, and passes the retrieved data to the response generator. The response generator gives the results to the user [7]. A good example of intermediate representation language architecture is MASQUE/SQL [7].

Various existing NLIDB systems have been discussed in next section.

2.2 Existing NLIDB systems

Prototype for NLIDB had appeared in late sixties and early seventies. The best-known NLIDB of that period is LUNAR. This system comes in early seventies (1973). LUNAR is a system that answers questions about samples of rocks brought back from the moon. The LUNAR system performance was quite impressive, it managed to handle 78% of requests without any errors and this ratio rose to 90% when dictionary errors were corrected [10][12][13].

In 1978, LADDER system was developed. It was designed as a natural language interface to a database of information about US Navy ships. LADDER system uses semantic grammar to parse questions to query a distributed database [11]. The system uses semantic grammars technique that interleaves syntactic and semantic processing. The system LADDER was implemented in LISP [7].

In early eighties, CHAT-80 was developed. CHAT-80 was implemented entirely in Prolog. It transformed English questions into Prolog expressions, which were evaluated against the Prolog database. The code of CHAT-80 was circulated widely and formed the basis of several other experimental NLIDBs [7].

In the same era, INTELLECT system was developed which was based on experience from ROBOT. ROBOT was a prototype of INTELLECT, a commercially available natural language interface to database systems which has been on market since 1981 [13].

In 1983, ASK system was developed. It allows end-users to teach the system new words and concepts at any point during the interaction. ASK was actually a complete information management system, providing its own built-in database. It has the ability to interact with multiple external databases, electronic mail programs, and other computer applications. Users stated their requests in English, and ASK transparently generated suitable requests to the appropriate underlying systems [7].

There was one another system called as Generic Interactive Natural Language Interface to Databases (GINLIDB). It was designed by the use of UML and developed using Visual Basic.NET-2005. The system was generic in nature given the appropriate database and knowledge base [14].

In December, 1993, Boris Katz at MIT's Artificial Intelligence Laboratory developed the START (SynTactic Analysis using Reversible Transformations) Natural Language System. It was world's first Web-based question answering system that has been online and continuously operating from starting [15]. It is a software system designed to answer questions that are posed to it in a natural language. START uses several language dependant functions like parsing, natural language annotation to present the appropriate information segments to the user [16].

In early 1997, JUPITER was developed. It was a conversational interface that allowed users to obtain worldwide weather forecast information over the telephone using spoken dialogue. Over a two year period since coming online in May 1997, JUPITER has received, *via*, a toll-free number in North America, over 30000 calls, mostly for naive users [17].

Intelligent Tutoring System (ITS) developed by Tanja Mitrovic in 1998 named as SQL-Tutor, which tutors students in database language SQL. SQL Tutor guides students through questions from four different databases and helps the students to create an SQL query to answer the question. Students could even ask the tutor new questions, simply by typing their question and SQL-Tutor would be able to assist them as usual [18].

The Text REtrieval Conference (TREC), co-sponsored by the National Institute of Standar]ds and Technology (NIST) and U.S. Department of Defense, was started in

1992 [19]. KUQA system was presented in TREC-9 (2000) developed by Soo-Min Kim and his colleagues. It categorized questions based on expected answer and then used NLP techniques as well as WorldNet for finding candidate answers which suit in corresponding category. It however didn't handle any linguistic phenomena [15].

In TREC-13 (2004) Michael Kaisser and Tilman Becker presented QuALiM system that used complex syntactic structure. Based on certain syntactic description question patterns were identified. Syntactic description of prospective answers was also maintained and accordingly system generated answer from the documents retrieved using Google search engine [15].

Chapter 3

Problem Statement

Query languages like SQL are very difficult to use for the general users and they find it very hard to learn. So, to make database applications easy to use for these people, Hindi language interface to databases has been developed. In the next section, problem statement is discussed.

3.1 Problem statement

We have identified EMPLOYEE database as a case study for developing Hindi language interface to databases system. The query is asked in the Hindi language for retrieving the relevant information from the database. There is also a facility of updating and deleting the table values by the user. User give the input in Hindi language and see the results same language. SQL query is also displayed on the graphical user interface.

3.2 Objectives

The main objectives of Hindi language interface to databases are discussed below.

- To design Hindi language interface for EMPLOYEE database.
- To design the graphical user interface in which an end user can input the query in Hindi language.
- To develop a system that can handle Hindi query for the extraction of single column or multiple columns from tables stored in EMPLOYEE database.
- To develop a system that can handle queries for updating single column or multiple columns of tables stored in EMPLOYEE database.
- To develop a system that can handle queries for deleting single row or multiple rows of tables stored in EMPLOYEE database.
- To develop a graphical user interface that can generate the Hindi output from data stored in database.

3.3 Methodology

To achieve the above discussed objectives, a step-by-step methodology has been followed. The details of methodology are given below.

- Study the Hindi Shallow Parser (developed by Language Technologies Research Centre (LTRC) at IIIT, Hyderabad) to use a tool for parsing the input provided in Hindi language.
- Obtain the root words and grammatical features of input Hindi words with the help of Hindi Shallow Parser.
- Create EMPLOYEE database which store the information about employees.
- Identify the nature of queries, whether it is an information retrieval, update or delete type of query.
- Extract the table, column, condition information from input Hindi sentence with the help of mapping of tokens with database values.
- Generate SQL query by mapping of input query with the values stored in database tables.
- Execute SQL query and give output to user in Hindi language.

Chapter 4

Design and Implementation of Hindi Language Interface to Databases

The design of Hindi language interface to databases includes the description of architecture of system and structure of EMPLOYEE database used in the system.

4.1 Architecture of the system

The architecture of Hindi language interface to databases is shown in Figure 4.1. Architecture of the system is divided five various phases. These phases are given below.

- To Parse and tokenize the input sentence using Hindi Shallow Parser.
- To find the English word corresponding to Hindi tokens from a dictionary.
- To map input Hindi query with database query.
- To generate SQL query with the help of database query generator and underlying DBMS.
- To execute the query and give the response to user.

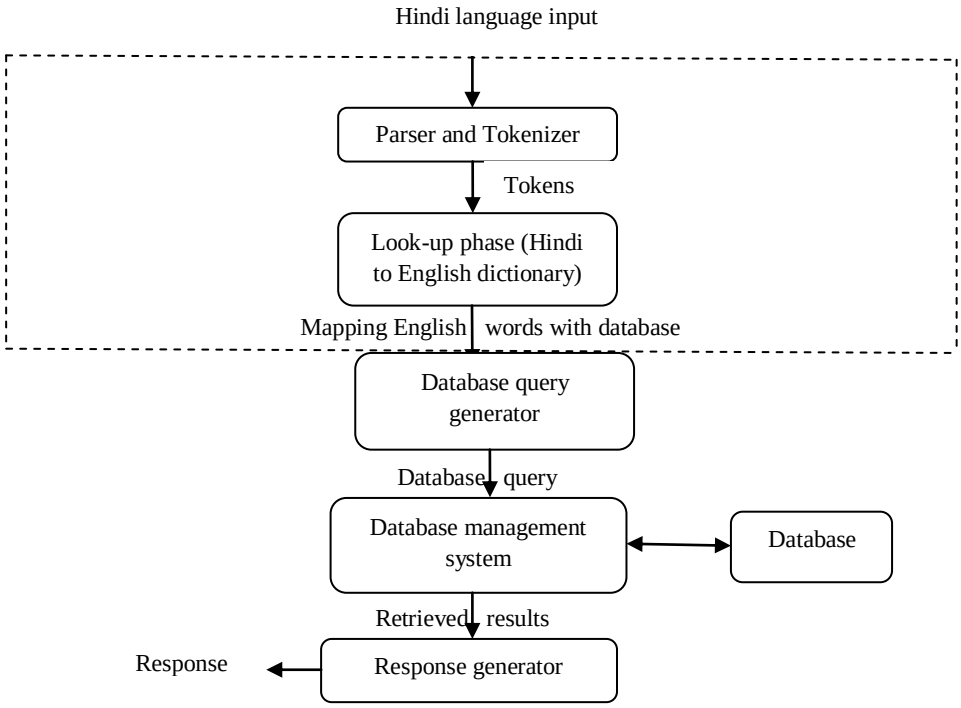

Figure 4.1: Architecture of the proposed system

The details of these phases are given below.

4.1.1 Parsing and tokenizing input sentence using Hindi Shallow Parser

In proposed system, Hindi Shallow Parser (developed by developed by Language Technologies Research Centre (LTRC) at IIIT, Hyderabad) has been used to perform parsing of a sentence given in Hindi language. It involves morphological analysis, part of speech tagging, chunking, pruning, *etc*. The overall processing task has been broken up into many modules and each one of them performs a small logical task. Hindi Shallow Parser uses Shakti Standard Format (SSF) for storing language analysis which has been described below.

Shakti Standard Format

It is a highly readable representation for storing language analysis. It is designed to be used as a common format or common representation on which all modules of a system operate [20]. The Shakti Standard Format (SSF) representation is designed to keep both rule-based as well as statistical approaches together. Two kinds of analyses are usually done in SSF. These are constituent and relational structure level analysis. Constituent analysis is used to store simple phrase level analysis (called chunking) and the relational analysis for storing relations between the simple phrases. SSF also involves one another concept of feature structures. Feature structures are used to store attribute-value pairs for phrasal node as well as for a word or a token. Outputs of many other kinds of analysis, such as grammatical relations, Tense, Aspect, Modality (TAM) computation, case computation, *etc.*, are stored using feature-structures. Though, the SSF format is fixed and it has a text representation, which makes it easy to read the output.

The working of Hindi Shallow Parser has been explained below.

Working of Hindi Shallow Parser

The working of Hindi Shallow Parser has been discussed in this section with the help of an example. The Hindi sentence that is parsed using Hindi Shallow Parser is given in (4.1).

सभी बच्चे किताब पढ़ते हैं। … (4.1)

Here, the sentence is parsed in following steps which is explained below.

- **Tokenizer**

  This module convert a sentence into word level tokens (consisting of words, punctuation marks, and other symbols) and return sentence marker for each

sentence of input text [21]. A token is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing. The result of tokenizing the input sentence given in (4.1) is shown in Figure 4.2.

<Sentence id="1">
1 सभी unk
2 बच्चे unk
3 किताब unk
4 पढ़ते unk
5 है unk
6 . unk
</Sentence>

Figure 4.2: Output of tokenizer for input sentence given in (4.1) [21]

The Hindi sentence give in (4.1) is divided into number of tokens as shown in (4.2).

सभी, बच्चे, किताब, पढ़ते, है, . … (4.2)

At this stage there is no information about the sentence like its noun, verb *etc*. So, with each token, put the symbol '*unk*' that means unknown token.

- **Morph analyzer**

The morphological analyzer identifies the root and the grammatical features of the word [21]. The output of morph analyzer for input sentence given in (4.1) is shown in Figure 4.3.

<Sentence id="1">
1 सभी unk <fs af=' सभी,n,m,sg,3,d,0,0'>|<fs af=' सभी,n,m,pl,3,d,0,0'>|
<fs af=' सभी,n,m,sg,3,o,0,0'>|<fs af=' सभी,n,m,pl,3,o,0,0'>
2 बच्चे unk <fs af=' बच्चा,n,m,pl,3,d,0,0'>|<fs af=' बच्चा,n,m,sg,3,o,0,0'>
3 किताब unk <fs af=' किताब,n,f,sg,3,d,0,0'>|<fs af=' किताब,n,f,sg,3,o,0,0'>
4 पढ़ते unk <fs af=' पढ़,v,m,pl,any,,ता,ता'>
5 है unk <fs af=' है,v,any,pl,1,,है,है'>|<fs af=' है,v,any,pl,3,,है,है'>
6 . unk <fs af='.,punc,,,,,,'>
</Sentence>

Figure 4.3: Output of morph analyzer for input sentence given in (4.1) [21]

Here, '*fs*' is the feature structure which shows the grammatical features of each word and '*af*' is a composite attribute which consists of a number of attributes which are shown in Table 4.1.

Table 4.1: Description of composite attribute of morph analyzer

Column Number	**Column Name**	**Description**
1	Root	The root is the primary lexical unit of a word, which carries the most significant aspects of semantic content and cannot be reduced into smaller constituents.
2	Lexical Category	Lexical category (also a word class, a lexical class, or in traditional grammar, a part of speech) is a linguistic category of words (or more precisely lexical items), which is generally defined by the syntactic or morphological behavior of the lexical item in question. Common linguistic categories include noun, verb, *etc*.
3	Gender	Gender can be male or female.
4	Number	Number is countable noun which is either singular or plural.
5	Person	Person is first person, second person or the third person.
6	Case	There are two cases in Hindi, direct and indirect. The indirect case is used when the noun is followed by a post-position; otherwise the direct case is used.

7	Tense	A form of a verb used to indicate the time, and sometimes the continuation or completeness, of an action in relation to the time of speaking. Tense is a method that is used in a language to refer to time- past, present and future.
	Aspect	Aspect refers to how an event or action is to be viewed with respect to time, rather than to its actual location in time. Aspect in a verb shows whether the action or state is complete or not.
	Modality	It is a part of language that indicates the degree to which an observation is possible, probable, likely, certain, permitted or prohibited.

This output of composite attribute for a token given in (4.2) is explained below.

For the token "सभी", the composite attribute consists of root word "सभी", lexical category is noun, and gender is male. The word is plural noun but this is not depicted at this phase. So, both singular and plural noun is attached in the feature structure. The word is a direct case and tense, aspect and modality is not present in this word.

- **Part of speech tagger**

  Part of speech tagging (postagger) is the process of assigning a part of speech to each word in the sentence. Identification of the parts of speech such as nouns, verbs, adjectives, adverbs for each word of the sentence helps in analyzing the role of each constituent in a sentence. The output of part of speech shows Category. The output of part of speech tagger for input sentence given in (4.1) is shown in Figure 4.4.

```
<Sentence id="1">
1       सभी     QF      <fs af=' सभी,n,m,sg,3,d,0,0'>|<fs af=' सभी,n,m,pl,3,d,0,0'>|
                        <fs af=' सभी,n,m,sg,3,o,0,0'>|<fs af=' सभी,n,m,pl,3,o,0,0'>
2       बच्चे   NN      <fs af=' बच्चा,n,m,pl,3,d,0,0'>|<fs af=' बच्चा,n,m,sg,3,o,0,0'>
3       किताब   NN      <fs af=' किताब,n,f,sg,3,d,0,0'>|<fs af=' किताब,n,f,sg,3,o,0,0'>
4       पढ़ते   VM      <fs af=' पढ़,v,m,pl,any,,ता,ता'>
5       है      VAUX    <fs af=' है,v,any,pl,1,,है,है'>|<fs af=' है,v,any,pl,3,,है,है'>
6       .       SYM     <fs af='.,punc,,,,,,'>
</Sentence>
```

Figure 4.4: Output of part of speech tagger for input sentence given in (4.1) [21]

Here, "सभी" is '*QF*' which indicates quantifier, "बच्चे" and "किताब" are '*NN*' which is singular or mass noun, "पढ़ते" is '*VM*' which indicates verb, "है" is '*VAUX*' which is an auxiliary verb. Auxiliary verbs are used together with a main verb to give grammatical information and therefore add extra meaning to a sentence, which is not given by the main verb. '.' is the '*SYM*' that indicates the symbol.

- **Chunker**

  Chunking involves identifying simple noun phrases, verb groups, adjectival phrase, and adverb phrase in a sentence. This involves identifying the boundary of chunks and the label. The output of Chunker for input sentence given in (4.1) is shown in Figure 4.5.

```
<Sentence id="1">
1       ((      NP
1.1     सभी     QF      <fs af=' सभी,n,m,sg,3,d,0,0'>|<fs af=' सभी,n,m,pl,3,d,0,0'>|
                        <fs af=' सभी,n,m,sg,3,o,0,0'>|<fs af=' सभी,n,m,pl,3,o,0,0'>
1.2     बच्चे   NN      <fs af=' बच्चा,n,m,pl,3,d,0,0'>|<fs af=' बच्चा,n,m,sg,3,o,0,0'>
        ))
2       ((      NP
2.1     किताब   NN      <fs af=' किताब,n,f,sg,3,d,0,0'>|<fs af=' किताब,n,f,sg,3,o,0,0'>
        ))
3       ((      VGF
3.1     पढ़ते   VM      <fs af=' पढ़,v,m,pl,any,,ता,ता '>
3.2     है      VAUX    <fs af=' है,v,any,pl,1,,है,है '>|<fs af=' है,v,any,pl,3,,है,है '>
3.3     .       SYM     <fs af='.,punc,,,,,,'>
        ))
</Sentence>
```

Figure 4.5: Output of chunker for input sentence given in (4.1) [21]

Here, the Hindi sentence as shown in (4.1) can be divided into a number of chunks. This is shown in (4.3).

1. सभी बच्चे

2. किताब पढ़ते

3. है. … (4.3)

- **Pruning**

  Pruning identifies the most appropriate feature structures out of different. Pruning involves two types of sub-pruning.

  Morph pruning: It takes those feature structures where the lexical category value matches with the category value. In case, if there is no feature structure whose lexical category matches with the category then all the feature structures are retained and a new attribute value pair poslcat= '*NM*' is added to every feature structure. Here, '*NM*' stands for "not matched".

  Pick one morph: It will pick only one feature structure based on selection definition given to it. By default, it will pick the first feature structure. The output of pruning for input sentence given in (4.1) is shown in Figure 4.6.

```
<Sentence id="1">
1       ((      NP
1.1     सभी     QF      <fs af=' सभी,n,m,sg,3,d,0,0' poslcat= "NM"'>
1.2     बच्चे   NN      <fs af=' बच्चा,n,m,pl,3,d,0,0'>
        ))
2       ((      NP
2.1     किताब   NN      <fs af=' किताब,n,f,sg,3,d,0,0'>
        ))
3       ((      VGF
3.1     पढ़ते   VM      <fs af=' पढ़,v,m,pl,any,,ता,ता '>
3.2     है      VAUX    <fs af=' है,v,any,pl,1,,है,है '>
3.3     .       SYM     <fs af='.,punc,,,,,,' poslcat= "NM">
        ))
</Sentence>
```

Figure 4.6: Output of pruning for input sentence given in (4.1) [21]

The lexical category of "सभी" is not matched with its part of speech category value, so a new attribute poslcat is added to it whose value is not matched. Similarly, the lexical category of "." is not matched with its part of speech

category value, so a new attribute poslcat is added to it whose value is not matched.

- **Head computation**

  This module computes the head of the chunk. A child node is identified as head of the chunk. A new feature called name is added to this child node. The attribute name is then set to an arbitrary but the unique string that sounds close to root for readability. The chunk node inherits the properties of the head child. All the features are copied from the head child to parent chunk except 'name'. A new attribute called head is added to the feature of the chunk node whose value is the name-string just assigned to the head child. Figure 4.7 shows the output of head computation for input sentence given in (4.1).

```
<Sentence id="1">
1       ((      NP      <fs af=' बच्चा,n,m,pl,3,d,0,0' head="बच्चे ">
1.1     सभी     QF      <fs af=' सभी,n,m,sg,3,d,0,0' poslcat="NM">
1.2     बच्चे   NN      <fs af=' बच्चा,n,m,pl,3,d,0,0' name="बच्चे ">
        ))
2       ((      NP      <fs af=' किताब,n,f,sg,3,d,0,0' head="किताब ">
2.1     किताब   NN      <fs af=' किताब,n,f,sg,3,d,0,0' name="किताब ">
        ))
3       ((      VGF     <fs af=' पढ़,v,m,pl,any,,ता,ता ' head="पढ़ते ">
3.1     पढ़ते   VM      <fs af=' पढ़,v,m,pl,any,,ता,ता ' name="पढ़ते ">
3.2     है      VAUX    <fs af=' है,v,any,pl,1,, है, है '>
3.3     .       SYM     <fs af='.,punc,,,,,,' poslcat="NM">
        ))
</Sentence>
```

Figure 4.7: Output of part of head computation for input sentence given in (4.1) [21]

Here, "बच्चे", "किताब", "है" acts as head node for chunk 1, 2, 3 respectively.

- **Vibhakti computation**

  The main task of vibhakti computation is to group the function words with the content words based on local information. One of the techniques to achieve this task is by using *kriya rupa* charts. These charts specify the groups to be formed out of the sequence of verbs which denote a single action. This module computes the case/tam features of noun/verb chunks and adds them to feature structure. The output after Vibhakti Computation for input sentence given in (4.1) is shown in Figure 4.8.

```
<Sentence id="1">
1       ((      NP      <fs af=' बच्चा,n,m,pl,3,d,0,0' head="बच्चे ">
1.1     सभी     QF      <fs af=' सभी,n,m,sg,3,d,0,0' poslcat="NM">
1.2     बच्चे   NN      <fs af=' बच्चा,n,m,pl,3,d,0,0' name="बच्चे ">
        ))
2       ((      NP      <fs af=' किताब,n,f,sg,3,d,0,0' head="किताब ">
2.1     किताब   NN      <fs af=' किताब,n,f,sg,3,d,0,0' name="किताब ">
        ))
3       ((      VGF     <fs af=' पढ़,v,m,pl,1,,ता_है,ता ' head="पढ़ते " vpos= “tam1_2”>
3.1     पढ़ते   VM      <fs af=' पढ़,v,m,pl,any,,ता,ता ' name="पढ़ते ">
3.2     .       SYM     <fs af='.,punc,,,,,,' poslcat="NM">
        ))
</Sentence>
```

Figure 4.8: Output of part of vibhakti computation for input sentence given in (4.1) [21]

Here, vibhakti computation for chunk 3 is tam1_2 which means the vibhakti depends upon the auxiliary verb in this case, *i.e.* “है”.

4.1.2 Hindi to English dictionary look-up phase

After getting all the tokens from Hindi Shallow Parser, their corresponding English words are found from a dictionary which contains Hindi and English words. All the words of input sentences are stored in the dictionary except proper noun.

4.1.3 Map input Hindi query with database query

After finding the English words from the dictionary, input sentence is analyzed so that system understands the type of query entered, *i.e.*, information retrieval, update or delete query. If the input sentence contains the word “करो” or “कीजिए”, system understands it as an update query. If input sentence contains the word “हटाओ”, system understands it a delete query, otherwise information retrieval query. After finding the type of query, tokens are mapped with database values for finding table name, column name and condition.

4.1.4 Generate SQL query

After mapping, SQL is generated by underlying DBMS. According to the type of Hindi sentence entered, SQL query is generated accordingly.

4.1.5 Execute query and give response to user

After SQL generation, it is executed and Hindi results are given to user accordingly.

4.2 Hindi language interface to EMPLOYEE database

The architecture shown in Figure 4.1 has been applied on EMPLOYEE database as a case study for Hindi language interface to databases. Two tables have been used in the database: EMPLOYEE table and DEPARTMENT. EMPLOYEE table contains personal information about all the employees working in the company. It includes their Employee Number (EMPNO), NAME, CITY, Date of Birth (DOB), SALARY and their corresponding Department Number (DEPTNO) in which they are working. The structure of EMPLOYEE table is shown in Table 4.2.

Table 4.2: Structure of EMPLOYEE table

Field	**Type**	**Null**	**Key**
EMPNO	Varchar (200)	No	Primary key
NAME	Varchar (200)	Yes	
CITY	Varchar (30)	Yes	
DOB	Date	Yes	
SALARY	Int (11)	Yes	
DEPTNO	Varchar (200)	Yes	Foreign key

DEPARTMENT table contains information about all the departments which has Department Number (DEPTNO) and Department Name (DEPT_NAME) as its columns. The structure of DEPARTMENT table is shown in Table 4.3

Table 4.3: Structure of DEPARTMENT table

Field	**Type**	**Null**	**Key**
DEPTNO	Varchar (200)	No	Primary key
DEPT_NAME	varchar(300)	Yes	

All the data of tables is inserted in Hindi language. This is done with the help of Google Transliteration [22].

4.2.1 Workflow of proposed system

The architecture shown in Figure 4.1 has been applied to proposed EMPLOYEE database system. The proposed system works in various steps of a workflow which has been discussed below.

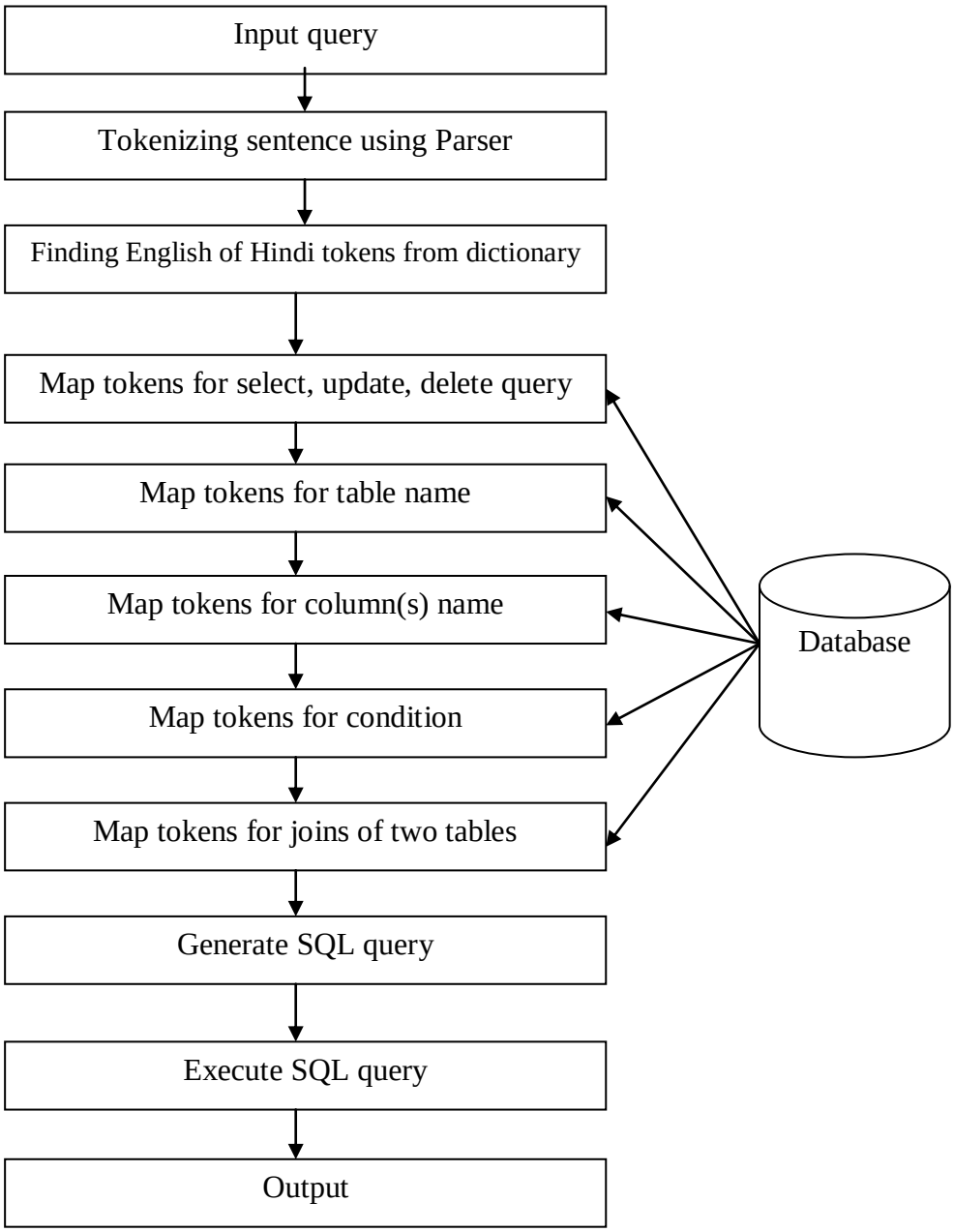

Figure 4.9: Workflow of Hindi language interface to databases

The workflow of proposed system has been described below.

- **Input query**

  The input query is a Hindi sentence entered by user. The working of system has been explained with an example sentence shown in (4.4).

  सभी कर्मचारिओं का नाम बताओ … (4.4)

- **Tokenizing**

  The Hindi Shallow Parser tokenizes the input sentence into various tokens. The root words generated by tokenizer for input sentence given in (4.4) are shown in (4.5).

  सभी, कर्मचारी, नाम, बता … (4.5)

- **Finding English of Hindi tokens from a dictionary**

  After tokenizing, the tokens are mapped to their English word with the help of a Hindi to English dictionary. The English words generated by dictionary for input sentence given in (4.4) are shown in (4.6).

  सभी-All

  कर्मचारी-Employee

  नाम-Name

  बता -Select … (4.6)

- **Map tokens for select, update and delete query**

  According to input Hindi sentence, system understands whether the query is information retrieval query, updating column (s) query or deleting row (s) query. If the Hindi sentence contains the word "करो" or "कीजिए", system understands it as an update query and "update" keyword is generated by the system. If the sentence contains the word "हटाओ", it is a delete query and "delete" keyword is generated by the system. Otherwise system understands it as an information retrieval sentence. The input sentence given in (4.4), contains the word "बता". So, the system understands it as an information retrieval query and "select" keyword is generated by the system.

- **Map tokens for table name, column(s) and condition**

  Apart from EMPLOYEE and DEPARTMENT table, system also contains some other tables like TABLE_HANDLE, COLUMN_HANDLE, CONDITION_ HANDLE and CONDITIONAL_WORD for mapping Hindi sentence into SQL query. The TABLE_HANDLE and COLUMN_HANDLE table contain data required to map table names and column names that are found in Hindi sentence into their corresponding English words respectively. TABLE_HANDLE table contains mapping for names of available tables and COLUMN_HANDLE table contains mapping for names of columns of available tables. CONDITIONAL_HANDLE table contains mapping for "where" keyword of SQL query. CONDITIONAL_WORD table contains mapping of conditional word like less than (<), greater than (>). Table 4.4, shows the TABLE_HANDLE table which describes the mapping for finding table name. It contains two columns: Token word and Mapped word. Token words are the names of database tables in Hindi, whereas mapped words are the names of database tables in English.

Table 4.4: TABLE_HANDLE table

Token Word	**Mapped Word**
कर्मचारी	Employee
विभाग	Department

COLUMN_HANDLE table is shown in Table 4.5, that explains the mapping for finding column (s) name.

Table 4.5: COLUMN_HANDLE table

Token Word	**Mapped Word**
कर्मचारीसंख्या	Empno
नाम	Name
शहर	City

जन्मतिथि	DOB
वेतन	Salary
विभागसंख्या	Deptno
विभागनाम	Dept_name

CONDITION_HANDLE table is shown in Table 4.6, which explains mapping for finding conditional word “where”.

Table 4.6: CONDITION_HANDLE Table

Token Word	**Mapped Word**
जिसका	Where
जिनका	Where
जिनकी	Where
जिसकी	Where
जिसके	Where
जिनके	Where
जो	Where
जोकि	Where

CONDITIONAL_WORD table is shown in Table 4.7, which shows the mapping for finding conditional operators.

Table 4.7: CONDITIONAL_WORD Table

Token Word	**Mapped Word**
समान	=

समान नही	!=
बराबर	=
बराबर नही	!=
बड़ा	>
ज्यादा	>
आगे	>
ऊपर	>
अधिक	>
अतिरिक्त	>
बड़ी	>
कम	<
छोटा	<
छोटी	<
नीचा	<
नीची	<
नीचे	<
पीछा	<
पीछी	<

As shown in (4.4), table name is EMPLOYEE and column name is NAME. But there is no condition present in this sentence. So, other tables of database are not processed.

- **Map tokens for joining of two tables**

  If Hindi sentence contains join of two tables, system understands it as a join query. The query is executed on DEPTNO column name as it is common column in both the tables. So, the system adds the phrase "EMPLOYEE.DEPTNO= DEPARTMENT.DEPTNO" in the output SQL query which shows joining of tables. These join queries also executes for update and delete sentences. The input sentence given in (4.4) doesn't contain any join query.

- **Generate SQL query**

  Finally, SQL query is generated which can be shown by using "Show SQL query" button. The SQL query for input sentence given in (4.4) is shown in (4.7).

  "SELECT NAME FROM EMPLOYEE" … (4.7)

- **Execute SQL query**

  The SQL query generated is executed on the database. The data retrieved from the execution is provided to the user as Hindi output. For the input sentence given in (4.4) all the names of employees are displayed which are shown in (4.8).

  विनोद
  राम
  कविता
  गौरव
  हिना
  विजय
  सोनिया
  रजनी
  सीमा
  अमन
  रिया
  साहिल
  अशोक … (4.8)

The next section discusses the steps for mapping Hindi sentence with database values for finding table name, column name and condition.

4.2.2 Steps for finding table name, column name and condition

The proposed system involves mapping of input Hindi sentence with database values for finding table name, column name and condition which has been explained below.

- **Steps to find table name from input Hindi sentence**
  - The proposed system searches the tokens first in TABLE_HANDLE table in the database. If any token matches with the value stored in this table, system maps that value with the table name and finds the table name to which that query associates.
  - If system doesn't find table name, it searches the tokens which match with the column names stored in a file. This file contains column names and their corresponding table names.
  - If any token matches with the column name value, corresponding table name retrieves from it.
  - If both the cases don't apply, then system reports an error.
- **Steps to find column name from input Hindi sentence**
  - System searches tokens generated by input sentence in COLUMN_HANDLE table. If any token matches with the values stored in the table, system maps that value with the column name and finds that column name to which that query associates.
  - System again searches in COLUMN_HANDLE table for column names that appear in condition.
  - If no token matches with column name, system executes the query "select * from table-name".
- **Steps to find condition from input Hindi sentence**
  - System searches tokens generated by input sentence in CONDITION_HANDLE table. If any token matches with the "where" keyword stored in the table, system further searches in the CONDITIONAL_WORD table for conditional operators like '>', '<'. If match occurs, query generates accordingly.

- If no token maps with “where”, system checks the sentence whether it is a conditional sentence according to its syntax. If conditional type of sentence occurs, system adds the “where” keyword in the SQL query.
- If no condition occurs, system executes the query “select column-name from table-name”.

4.2.3 Flowchart of Hindi language interface to databases

The proposed system consists of information retrieval, updating column (s), deleting row (s) type of queries. A flowchart is given in Figure 4.10, to explain the flow of steps performed in the proposed system. This flowchart has three termination points A, B, C. The remaining flowchart for Figure 4.10 is given in Figure 4.11, 4.12 and 4.13.

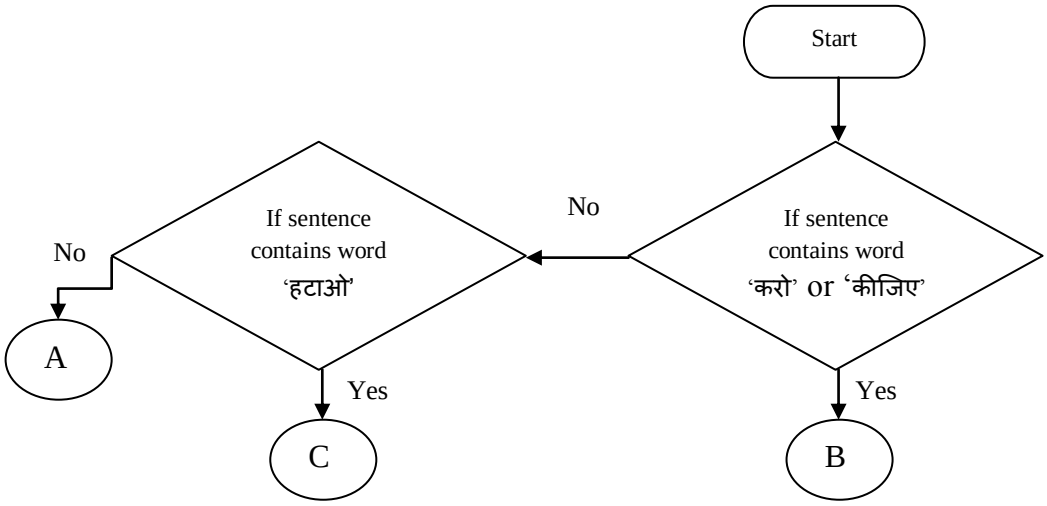

Figure 4.10: Flowchart of Hindi language interface to databases

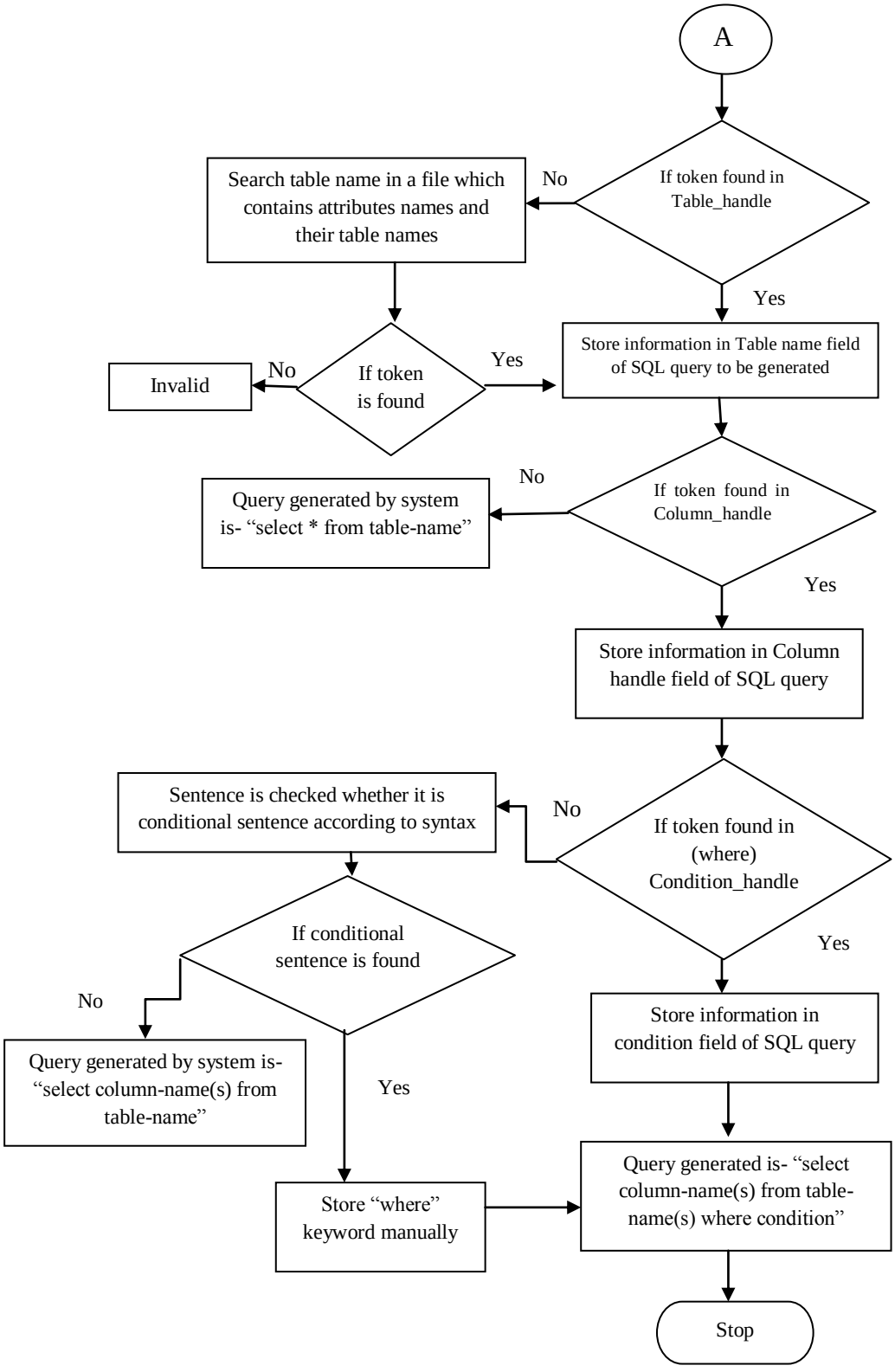

Figure 4.11: Flowchart of Hindi language interface to databases for select query

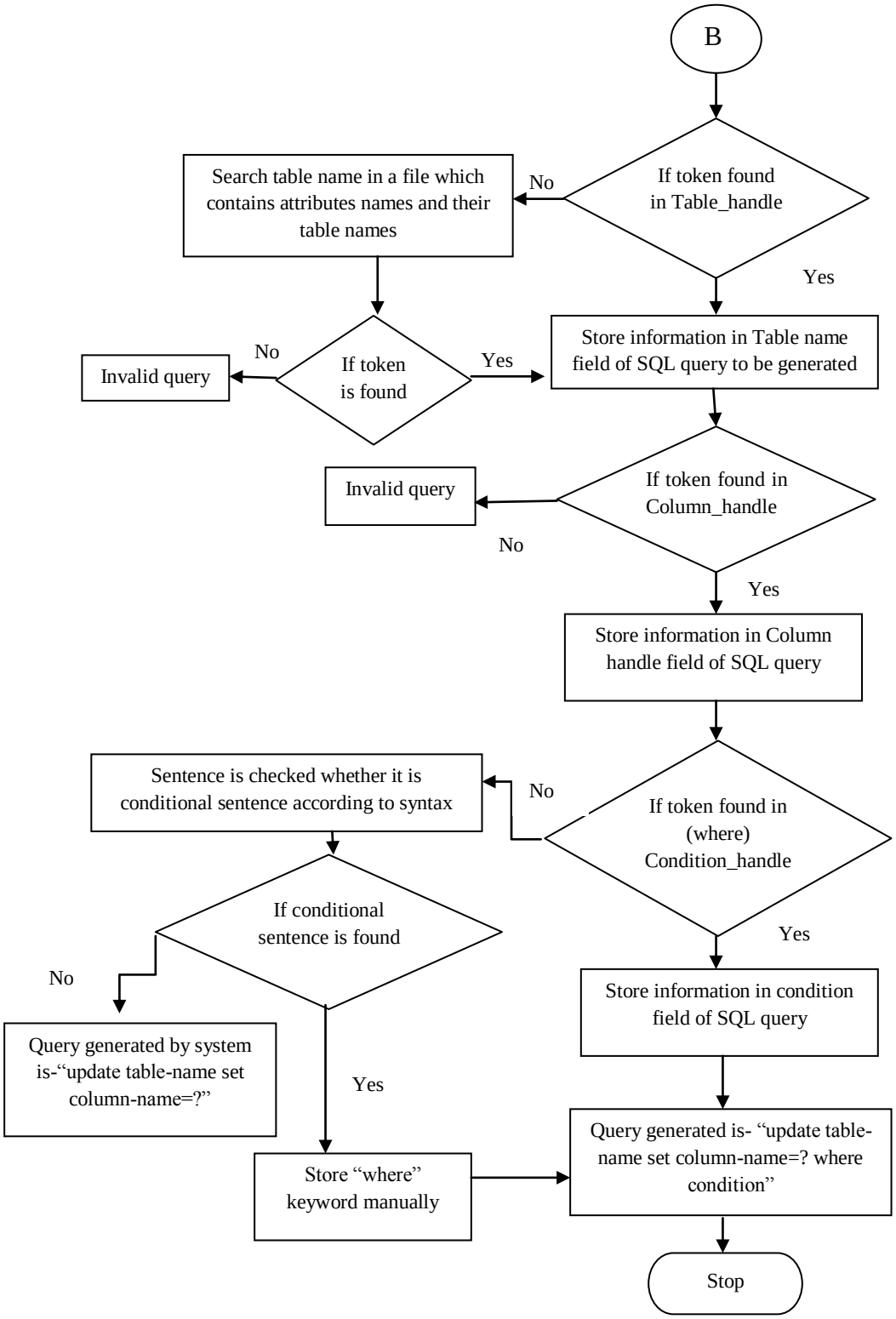

Figure 4.12: Flowchart of Hindi language interface to databases for update query

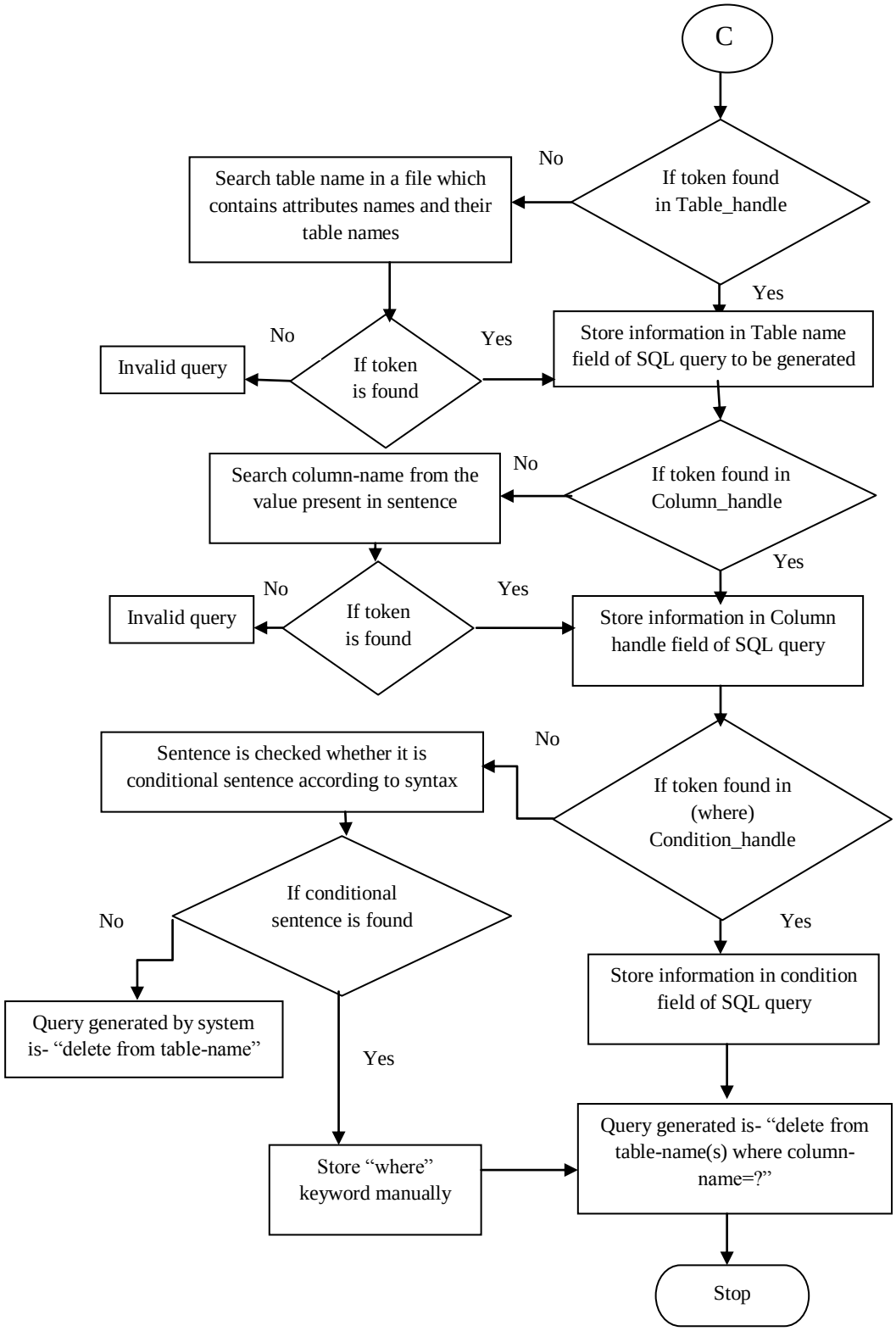

Figure 4.13: Flowchart of Hindi language interface to databases for delete query

4.3 Implementation and working of the system

The proposed system has been developed for UNIX platform. The input Hindi sentence is parsed and tokenized using Hindi Shallow Parser. It uses MYSQL as database and JAVA Swings as front end. MYSQL is connected with JAVA using *mysql-connector-java*. Unicode character set is used for providing input and output in Hindi. Data is also stored in Hindi in the database. The system handles three types of queries which have been illustrated below.

- Data Retrieval query.
- Update Column(s) query.
- Delete Row(s) query.

The user interface is shown in Figure 4.14.

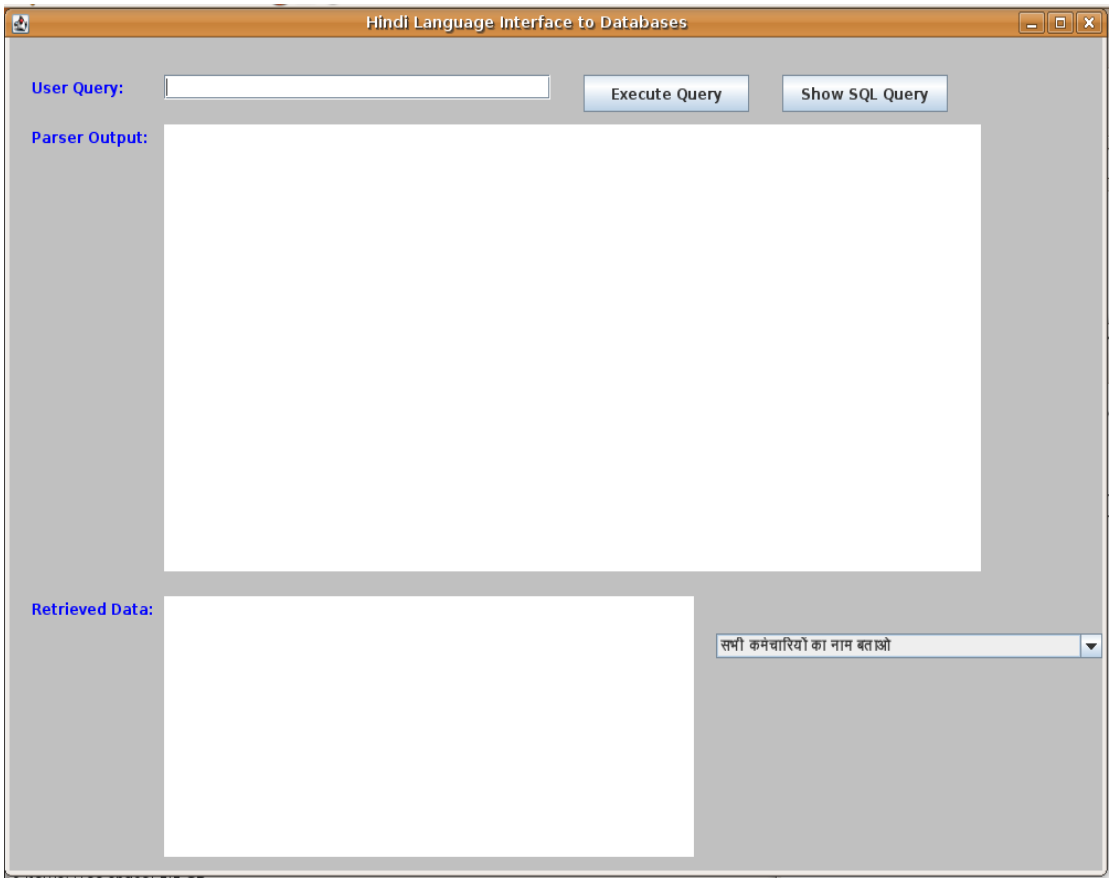

Figure 4.14: User interface of Hindi language interface to databases

Figure 4.14, describes the user interface of the proposed system. The interface includes a Combo Box which contains list of Hindi queries that user can ask from the system. The selected query is displayed in the text field provided above. User can also give his own query similar to type of queries in the combo box. The first Text Area contains the output of Hindi Shallow Parser while second text area is for query output.

“Execute Query” button is used to execute the query and “Show SQL Query” button is used to display the corresponding SQL query generated.

4.3.1 Data retrieval query

The proposed system can handle different form of queries which has been illustrated below.

- Query for selection of whole table.
- Query for selection of single or multiple columns.
- Query for selection of certain rows from certain column, *i.e.*, Conditional query.
- Query that doesn’t include table name but table name is searched from column name.
- Query which includes join of two tables.

The screenshot of query that involves selection of whole table is shown in Figure 4.15.

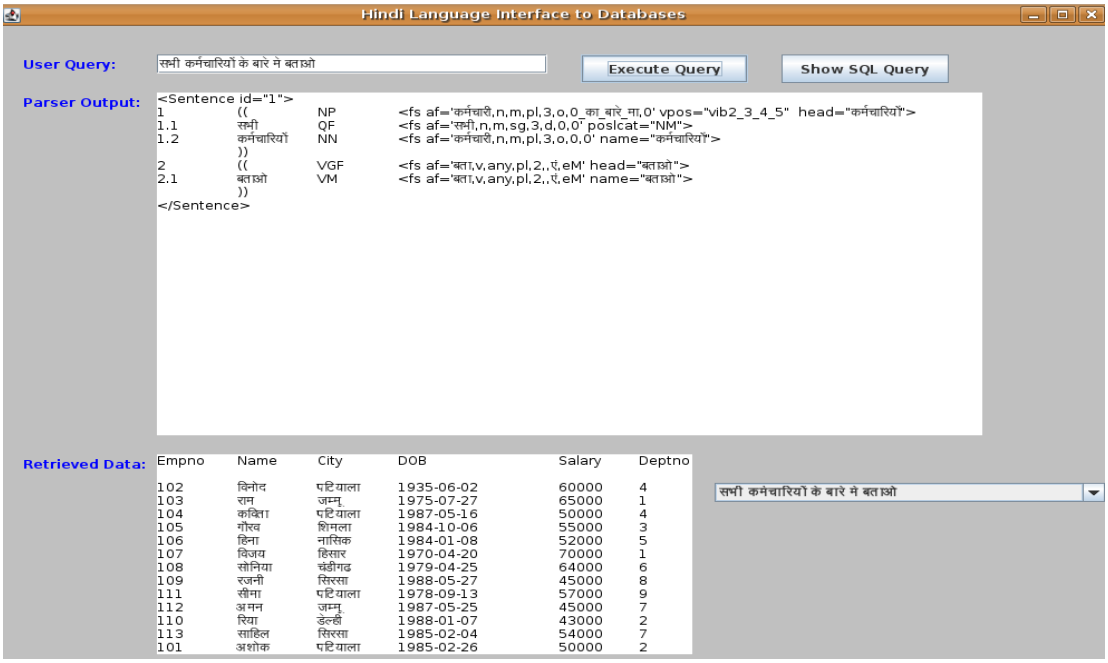

Figure 4.15: Screenshot of query that involve selection of whole table

Figure 4.15, shows when “Execute Query” button is pressed, output of parser and final results are appeared in desired text areas. If user wants to see the SQL query generated, he can press “Show SQL Query” button. The SQL query generated is shown in (4.9)

SELECT * FROM EMPLOYEE; … (4.9)

The screenshot of query that involves selection of single or multiple columns is shown in Figure 4.16.

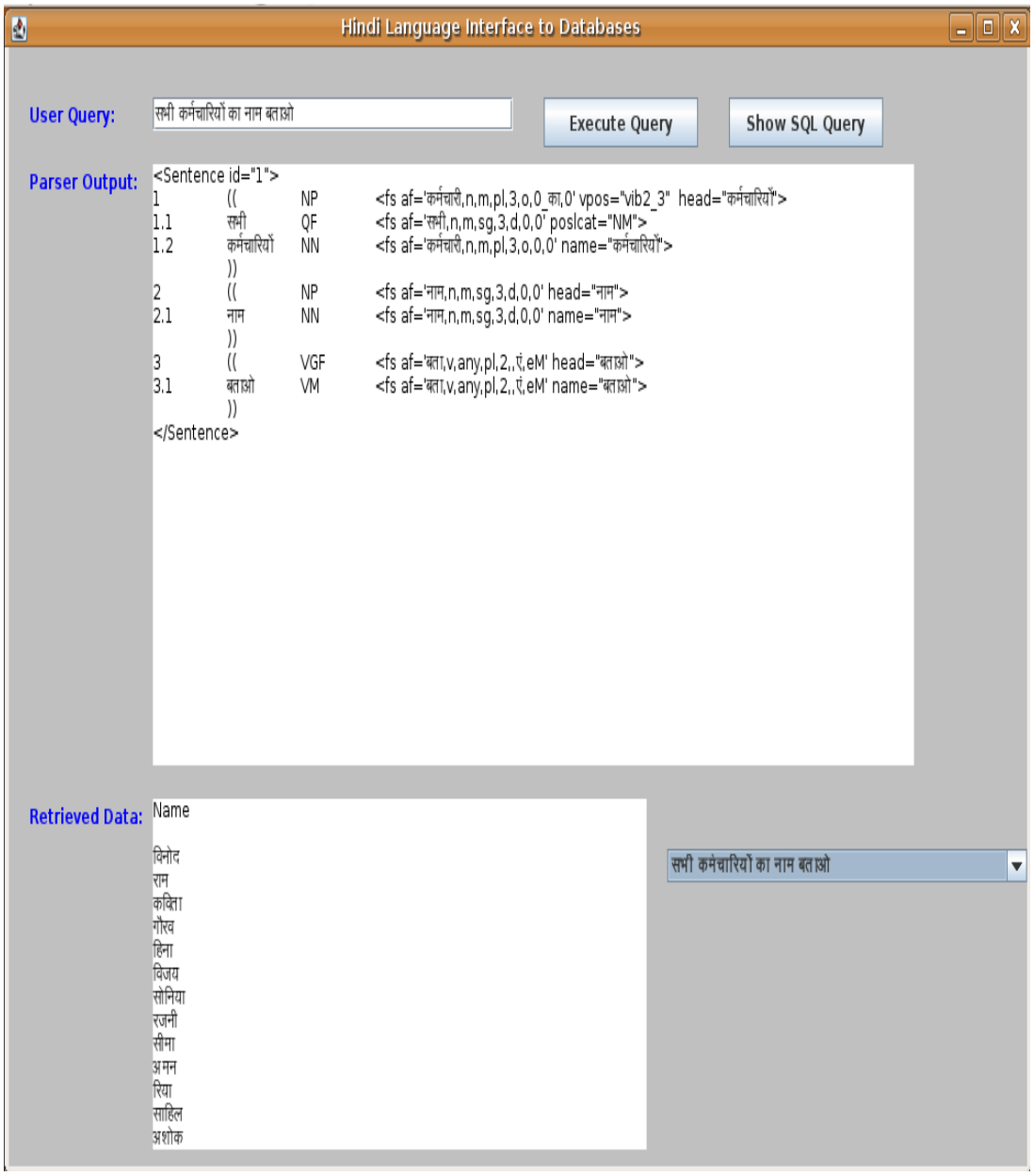

Figure 4.16: Screenshot of query that involves selection of single or multiple columns

Figure 4.16, shows when "Execute Query" button is pressed, output of parser and final results are appeared in desired text areas. If user wants to see the SQL query generated, he can press "Show SQL Query" button. The SQL query generated is shown in (4.10).

SELECT NAME FROM EMPLOYEE; … (4.10)

The screenshot of query that involves condition is shown in Figure 4.17.

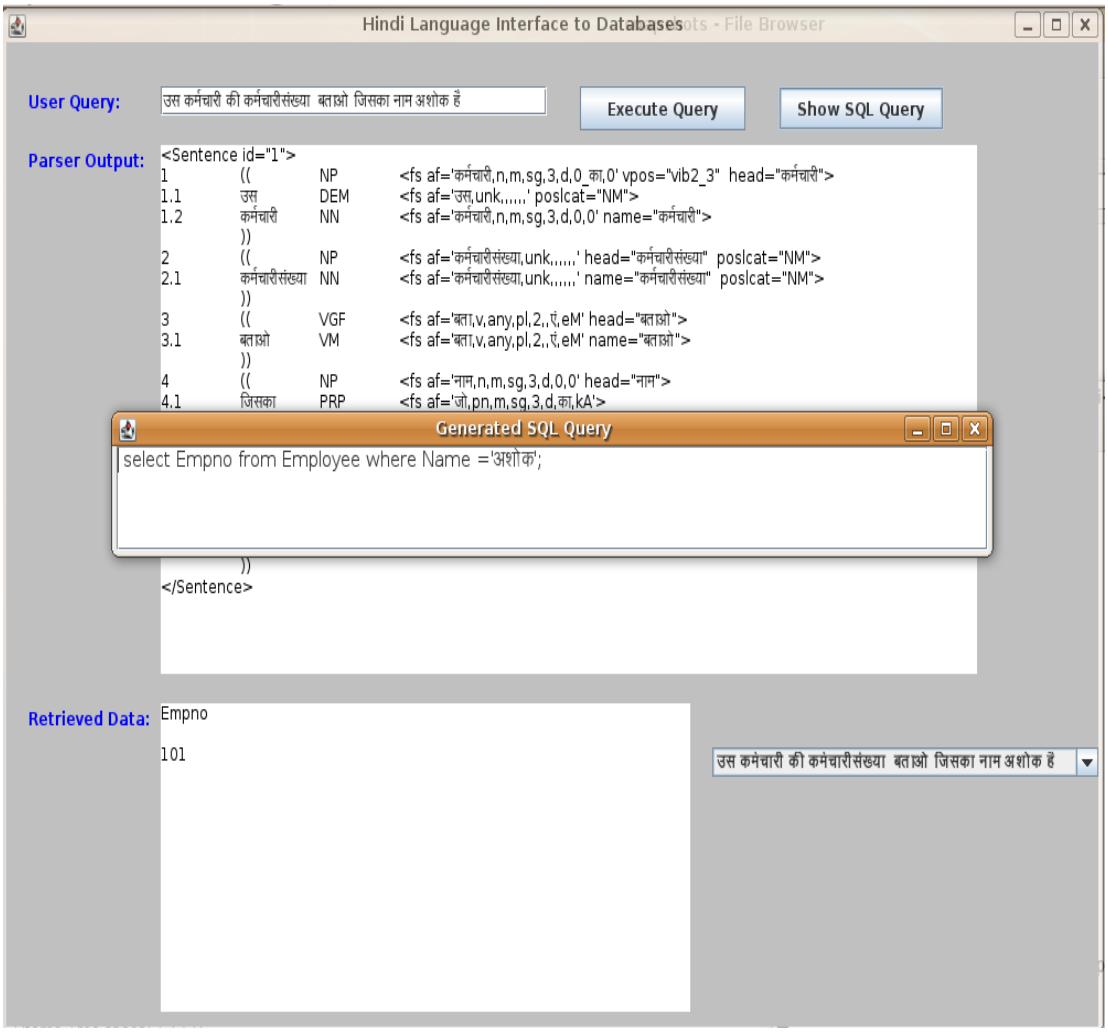

Figure 4.17: Screenshot of query that involves condition

Figure 4.17, shows the query which involves selection of employee number of that employee whose name is 'अशोक'. When "Execute Query" button is pressed, output of parser and final results are appeared in desired text areas. "Show SQL Query" button displays the SQL query generated. The SQL query generated is shown in (4.11).

SELECT EMPNO FROM EMPLOYEE WHERE NAME= 'अशोक'; … (4.11)

The screenshot of query that includes joining of two tables is shown in Figure 4.18.

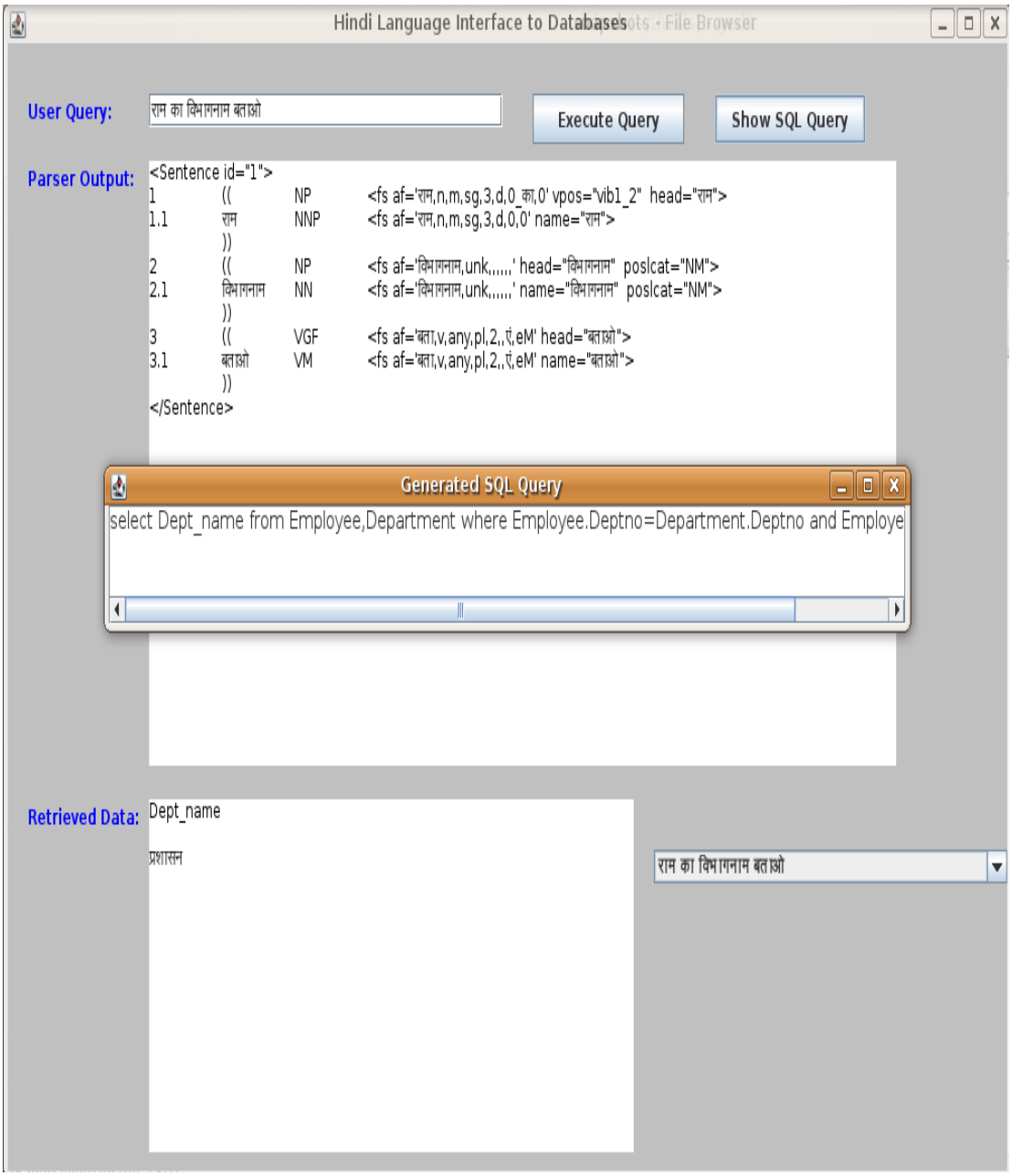

Figure 4.18: Screenshot of query that includes joining of two tables

Figure 4.18, shows when "Execute Query" button is pressed, output of parser and final results are appeared in desired text areas. If user wants to see the SQL query generated, he can press "Show SQL Query" button. The SQL query generated is shown in (4.12).

SELECT DEPT_NAME FROM EMPLOYEE, DEPARTMENT WHERE EMPLOYEE.DEPTNO= DEPARTMENT.DEPTNO AND EMPLOYEE.NAME= 'राम'; … (4.12)

4.3.2 Update column (s) query

The proposed system can handle different form of queries which has been illustrated below.

- Query that update single or multiple column(s).
- Query that update column (s) but doesn't include table name.
- Query that update column (s) by joining of two tables.

The screenshots of query that involves updating column (s) which do not include table name is shown in Figure 4.19.

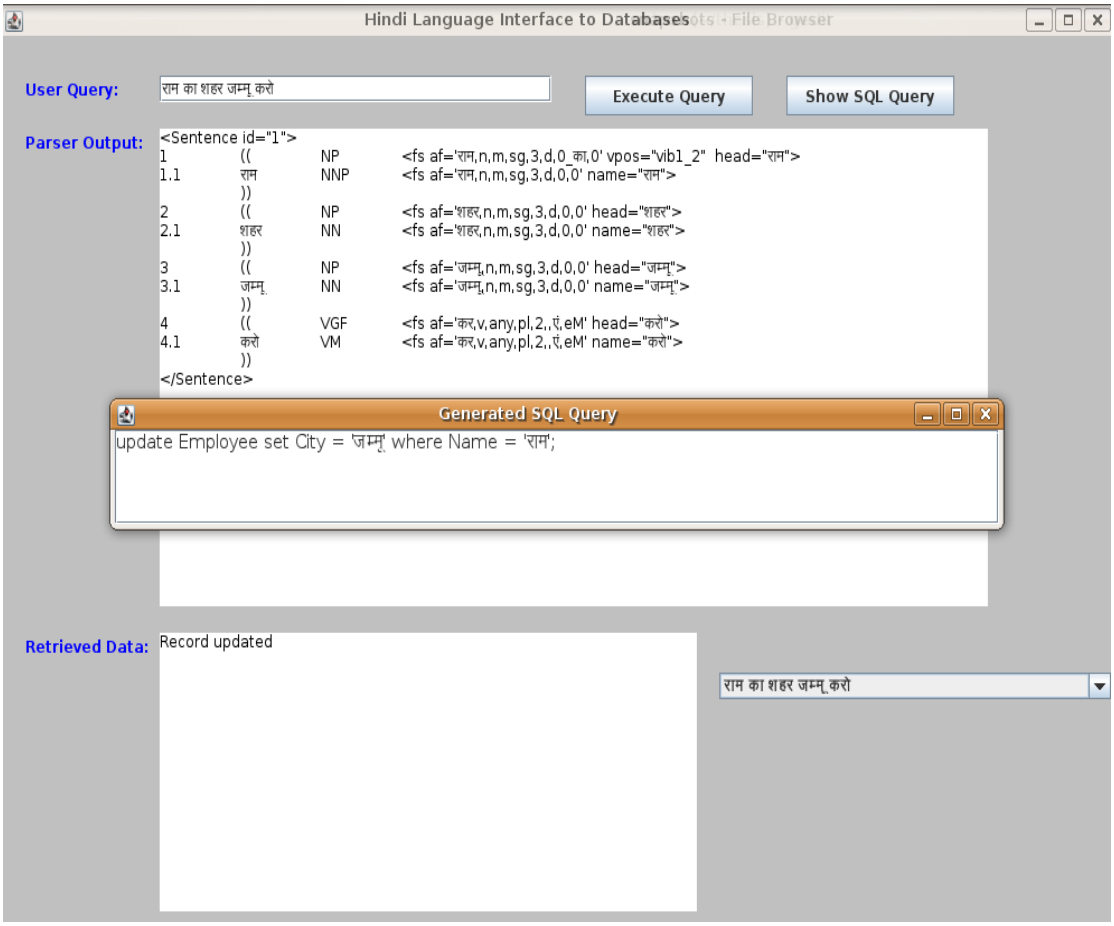

Figure 4.19: Screenshot of query that update column (s) but doesn't include table name

Figure 4.19, shows the input query is to change the city of employee 'राम' to 'जम्मू'. When "Show SQL Query" button is pressed, SQL query generated is shown in (4.13).

UPDATE EMPLOYEE SET CITY= 'जम्मू' WHERE NAME= 'राम'; … (4.13)

The screenshot of output of updated query of Figure 4.19 is shown in Figure 4.20.

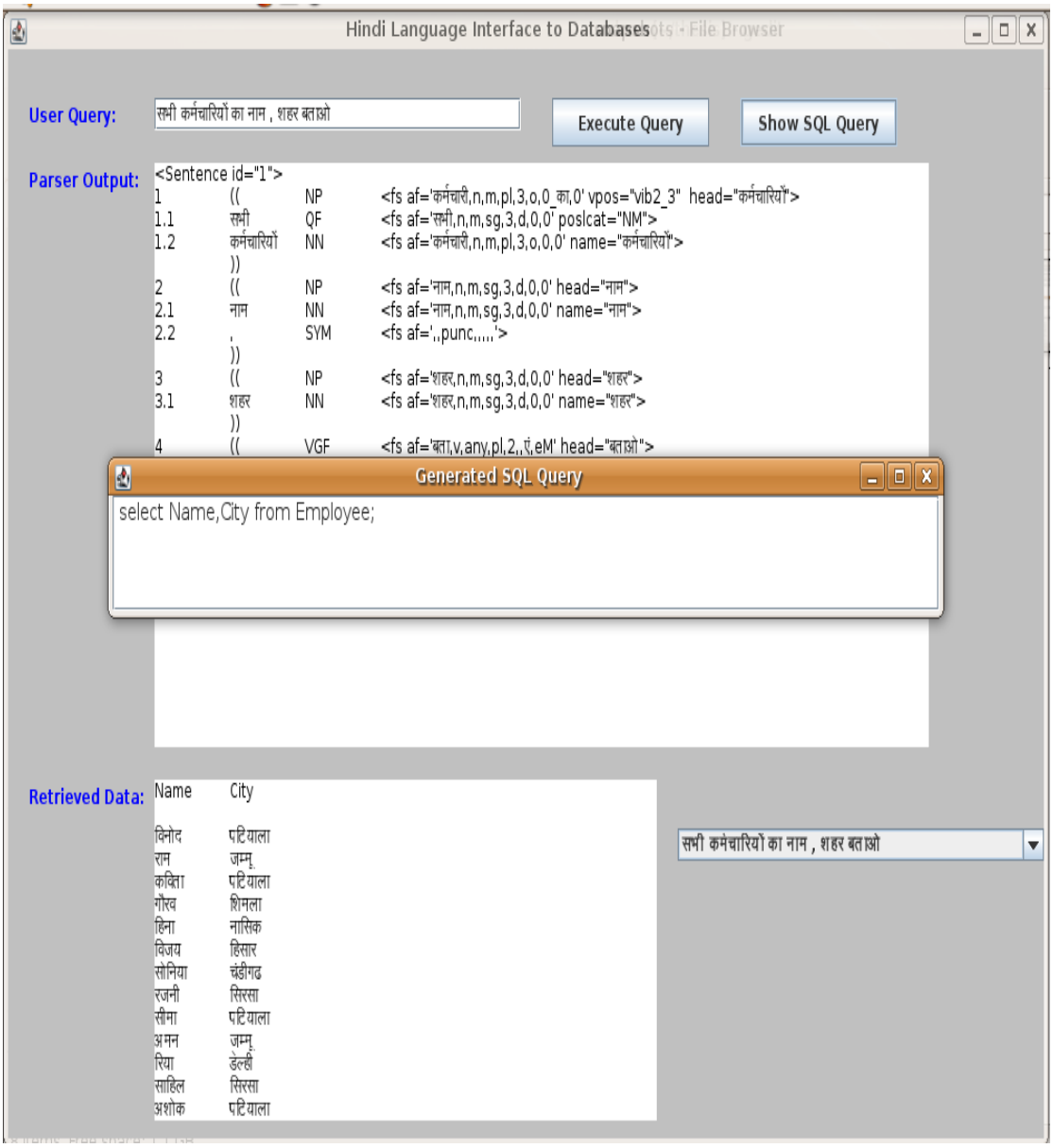

Figure 4.20: Screenshot of output of update query

The screenshots of query that involves updating column (s) by joining of two tables is shown in Figure 4.21.

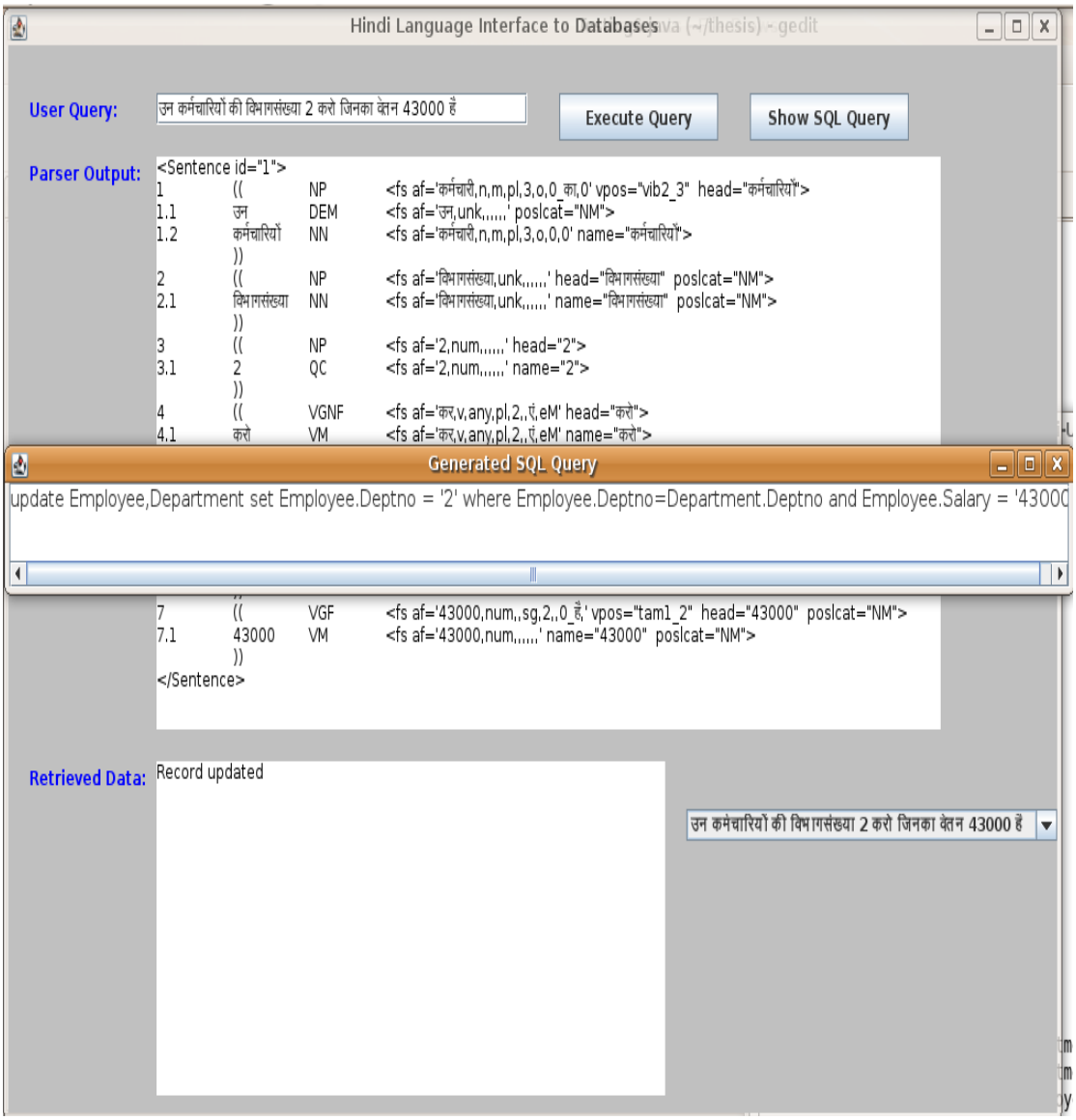

Figure 4.21: Screenshot of query that update column (s) by joining of two tables

Figure 4.21, shows the screenshot of update type of query in which a column is updated by joining of two tables, *i.e.*, EMPLOYEE and DEPARTMENT table. The input query is to change the department number of an employee to 2 whose salary is 43000. When "Show SQL Query" button is pressed, SQL query is displayed in a frame which is shown in (4.14).

UPDATE EMPLOYEE, DEPARTMENT SET EMPLOYEE.DEPTNO= '2' WHERE EMPLOYEE.DEPTNO= DEPARTMENT.DEPTNO AND EMPLOYEE.SALARY= '43000'; … (4.14)

4.3.3 Delete row (s) query

The proposed system can handle different form of queries which has been explained below.

- Delete single or multiple rows.
- Delete row (s) when table name is not present in query.
- Delete row (s) by joining two tables.

A screenshot of query that involves deleting row(s) is shown in Figure 4.22.

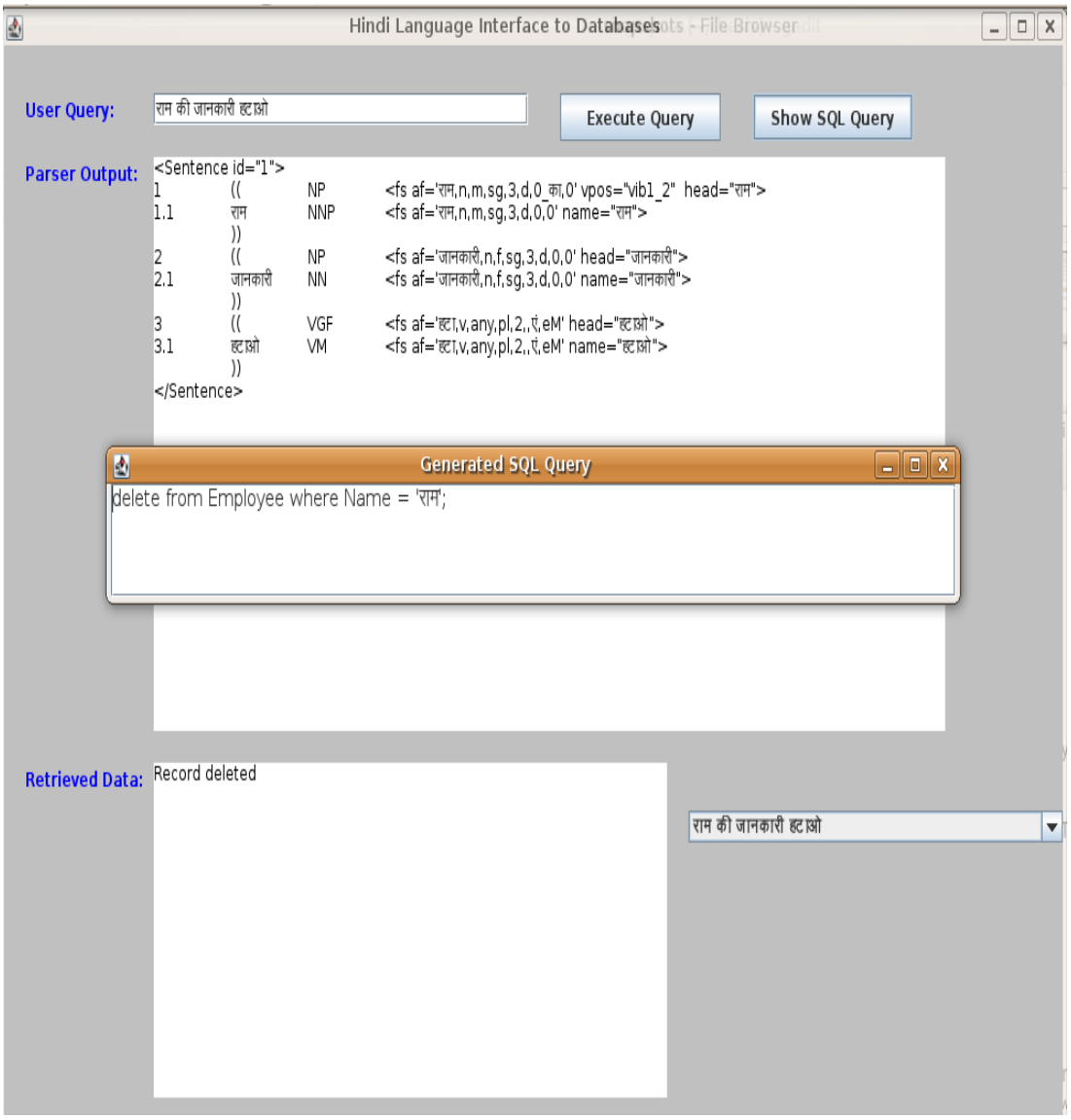

Figure 4.22: Screenshot of query that involves deleting row(s)

Figure 4.22 shows how delete query delete all the information about "राम". The generated query is shown in (4.15).

DELETE FROM EMPLOYEE WHERE NAME= 'राम'; … (4.15)

Chapter 5

Testing and Results

The Hindi language interface to databases can handles three types of queries which has been illustrated below.

- Data Retrieval query.
- Update Column(s) query.
- Delete query.

The system has been tested on 50 Hindi input sentences. The testing results performed on data retrieval queries are shown in Table 5.1.

Table 5.1: Testing results performed on data retrieval queries

Hindi Sentence	**SQL Query**
सभी कर्मचारियों का नाम बताओ	Select Name from Employee;
उस कर्मचारी की कर्मचारीसंख्या बताओ जिसका नाम अशोक है	Select Empno from Employee where Name= 'अशोक';
उस कर्मचारी का शहर बताओ जिसका नाम राम है	Select City from Employee where Name= 'राम';
रिया की जन्मतिथि बता	Select DOB from Employee where Name= 'रिया';
कर्मचारियों का वेतन दो	Select Salary from Employee;
उन कर्मचारियों का विभागनाम बताओ जिनका वेतन 50000 है	Select Dept_name from Employee, Department where Employee.Deptno= Department.Deptno and Employee. Salary= '50000';

उस कर्मचारी का नाम बताओ जिसकी कर्मचारीसंख्या 102 है	Select Name from Employee where Empno= '102';
कर्मचारियों के नाम, शहर बताओ	Select Name, City from Employee;
सिरसा के कर्मचारी का नाम बताओ	Select Name from Employee where City= 'सिरसा';
उन कर्मचारियों की कर्मचारीसंख्या, नाम बताओ जिनकी विभागसंख्या 7 है	Select Empno, Name from Employee, Department where Employee.Deptno= Department.Deptno and Employee. Deptno= '7';
विनोद का शहर बताओ	Select City from Employee where Name= 'विनोद';
राम का विभागनाम बताओ	Select Dept_name from Employee, Department where Employee.Deptno= Department.Deptno and Employee. Name= 'राम';
उन कर्मचारियों का नाम बताओ जिनकी कर्मचारीसंख्या 110 से ऊपर है	Select Name from Employee where Empno > '110';
उन कर्मचारियों का नाम बताओ जिनका वेतन 60000 से ज्यादा है	Select Name from Employee where Salary > '60000';
सभी कर्मचारियों के बारे मे बताओ	Select * from Employee;
हर विभाग की जानकारी दो	Select * from Department;

उस कर्मचारी का विभागनाम बताओ जिसकी विभागसंख्या 7 है	Select Dept_name from Employee, Department where Employee.Deptno= Department.Deptno and Employee. Deptno = '7';
उन कर्मचारियों का नाम बताओ जिनका विभागनाम खरीद है	Select Name from Employee, Department where Employee.Deptno= Department.Deptno and Department.Dept_name= 'खरीद';
उस कर्मचारी का शहर बताओ जिसका नाम हिना है	Select City from Employee where Name= 'हिना';

The testing results performed on update column (s) type queries are shown in Table 5.2.

Table 5.2: Testing results performed on update column (s) queries

Hindi Sentence	**SQL Query**
उस कर्मचारी का नाम राम करो जिसकी कर्मचारीसंख्या 107 है	Update Employee set Name= 'राम' where Empno= '107';
उस कर्मचारी का शहर पटियाला करो जिसका नाम अशोक है	Update Employee set City= 'पटियाला' where Name= 'अशोक';
उस कर्मचारी का विभागनाम प्शासन करो जिसकी विभागसंख्या 1 है	Update Department set Dept_name= 'प्शासन' where Deptno= '1';
उन कर्मचारियों की विभागसंख्या 2 करो जिनका वेतन 43000 है	Update Employee, Department set Employee. Deptno= '2' where Employee. Salary= '43000';
उस कर्मचारी का शहर पटियाला करो	Update Employee, Department set

जिसका विभागनाम खरीद है	Employee.City= 'पटियाला' where Department.Dept_name= 'खरीद';
उस कर्मचारी का शहर शिमला करो जिसकी विभागसंख्या 3 है	Update Employee, Department set Employee.City= 'शिमला' where Department.Deptno= '3';
राम का शहर जम्मू करो	Update Employee set City= 'जम्मू';

The testing results performed on delete row (s) queries are shown in Table 5.3.

Table 5.3: Testing results performed on delete row (s) queries

Hindi Sentence	**SQL Query**
उस कर्मचारी की जानकारी हटाओ जिसका नाम अशोक है	Delete from Employee where Name= 'अशोक';
उस कर्मचारी की जानकारी हटाओ जिसका विभागनाम खरीद है	Delete from Employee, Department where Department.Dept_name= 'खरीद';
कर्मचारियों की जानकारी हटाओ	Delete from Employee;
राम की जानकारी हटाओ	Delete from Employee where Name= 'राम';

Tables 5.1, 5.2, 5.3 show that the system works successfully for all type of queries.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The proposed Hindi language interface to Databases to Employee database accepts query in Hindi language, parses and tokenizes the sentence with the help of Hindi Shallow Parser and maps the Hindi root words with their corresponding English words with the help of dictionary maintained. After mapping, sentence is checked whether it is data retrieval, update or delete type of sentence. This is done by analyzing Hindi input sentence. After analyzing the Hindi sentence, table names, column names and conditions are searched in the database tables. The tokens are mapped with database values. After mapping, query is again analyzed to check whether it involve joining of multiple tables or not. SQL query is then generated and executed and result is displayed to user.

6.2 Future work

The following tasks can be performed in future:

- To extend the system to execute the queries that involves joining of more than two tables
- To design user interface for administrator where application can be tuned for new databases
- To work on queries which involve conditions like '*between*', '*like*', '*in*'
- To execute queries that include aggregate functions like sum(), max(), min()
- To add the feature to execute DDL statements like create, alter, drop type of queries
- To add the feature to execute insert type of queries
- To add the feature to execute queries that includes order by, group by and having clauses
- To add the feature to work on independent sub-queries.

References

[1] Akshar Bharti, Y. Krishna Bhargava and Rajeev Sangal, "References and Ellipsis in an Indian Languages Interface to Databases", *Journal of Computer Science of India*, vol. 23, no. 3, pp 60-82, Sep. 1993.

[2] Gobinda G. Chowdhury, "Natural Language processing", *Annual Review of Information and Science Technology*, vol. 37, no. 1, pp 51-89, 2003.

[3] Amandeep Kaur, "Punjabi Language Interface to Databases", ME Thesis, Thapar University, Jun. 2010.

[4] H.R. Tennant, K.M. Ross, M. Saenz, C.W. Thompson and J.R.Miller, "Menu-Based Natural Language Understanding", Proceedings of the *21st Annual Meeting of ACL*, Cambridge, Massachusetts, pp. 151-158, 1993.

[5] Mrs. Neelu Nihalani, Dr. Sanjay Silakari and Dr. Mahesh Motwani, "Natural Language Interface for Database: A Brief Review", *IJCSI International Journal of Computer Science Issues*, vol. 8, no. 2, pp. 600-608, Mar. 2011.

[6] T. Johnson, "Natural Language Computing-The Commercial Applications", *The Knowledge Engineering Review*, vol. 1, no. 3, pp. 11-23, 1984.

[7] Androutsopoulos, G.D. Ritchie and P. Thanisch, "Natural Language Interface to Databases-An Introduction", Department of Computer Science, University of Edinburgh, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland, U.K. , Mar. 1995.

[8] W.A. Woods, R.M. Kaplan and B.N. Webber, "The Lunar Sciences Natural Language Information System: Final Report", BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.

[9] C.R. Perrault and B.J. Grosz, "Natural Language Interfaces", Exploring Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Mateo, California, 1988, pp. 133-172.

[10] D.L. Waltz, "An English Language Question Answering System for a Large Relational Database", *Communications of the ACM*, pp. 526-539, 1978.

[11] G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Complex Data", *ACM Transactions on Database Systems*, pp. 105-147, 1978.

[12] W. Woods, "An experimental parsing system for transition network grammars in Natural Language Processing", Algorithmic Press, New York, USA, 1973.

[13] L.R. Harris, "Experience with INTELLECT: Artificial Intelligence Technology Transfer", The AI Magazine, pp. 43-50, 1984.

[14] Faraj A. El-Mouadib, Zakaria S. Zubi, Ahmed A. Almagrous and Irdess S. El-Feghi, "Generic Interactive Natural Language Interface to Databases (GINLIDB)", *International Journal of Computers*, vol. 3, no. 3, 2009.

[15] "START Natural Language Question Answering System". [Online]. Available: http://start.csail.mit.edu/

[16] M. Joshi, R. A. Akerkar, "Algorithms to improve performance of Natural Language Interface", *International Journal of Computer Science & Applications*, vol. 5, no. 2, pp. 52-68, 2008.

[17] V. Zue et al., "JUPITER: A telephone-based conversational interface for weather information", *IEEE transaction on Speech and Audio Processing*, vol. 8, no. 1, pp 85-96, Aug. 2002.

[18] Seymour Knowles and Tanja Mitrovic, "A Natural Language Interface For SQL-Tutor", Nov. 5, 1999.

[19] Rajendra Akerkar and Priti Sajja, "Natural Language Interface: Question Answering System" in Knowledge-Based Systems, Jones & Bartlett Learning, pp. 333-344, Aug. 2009.

[20] Akshar Bharati, Rajeev Sangal and Dipti Misra Sangal, "Shakti Standard Format Guide", Rep. IIIT/TR/2009/85, May 2009.

[21] "Hindi Shallo Parser". [Online]. Available: http://ltrc.iiit.ac.in/analyzer/hindi/

[22] “Google Transliteration”. [Online]. Available: http://www.google.com/transliterate/

Research Publications

Research Paper Published

- Himani Jain, Parteek Bhatia, "Hindi Language Interface to Databases" Published in Journal of Global Research in Computer Science, April 2011, vol. 2, no. 4, pp. 107-112, Available: http://www.jgrcs.info/index.php/jgrc/ article/viewFile/184/107

Research Paper Communicated

- Himani Jain, Parteek Bhatia, "Design and Development of Hindi Language Interface to Databases", communicated to IETE Journal of Research.