Design and Development of Natural Language Query Interface for Relational Databases

A Presentation
For
The Degree

Doctor of Philosophy
(Computer Science)

To

Sardar Patel University
Vallabh Vidyanagar

Research Guide
Dr. Paresh Virparia

Research Scholar
Amisha Shingala
Preamble

• Everyday most of the people are interacting with information stored in database, usually, through the specialized and predefined programs.

• The limitation of such programs is that it restricts the interaction between user and database to predefined set of queries.

• Only few people who have knowledge of database structure and formal database language (such as Structured Query Language (SQL)) can retrieve the desired information from database.
Preamble

• A novice user having no knowledge of database structure and formal database query language cannot retrieve desired information if it is not supported by well thought application.

• Hence, it was a need to improve human computer interface that allows people to interact with the database in their natural language (such as English).

• The effective Natural Language Interface to Database (NLIDB) system has high potential to mimic computer systems as conversational system thereby making it easy to use by large mass of people enhancing its utility value.
Research Objective

• To provide more efficient and user friendly Natural-English Language Query Interface to Database (N-ELIDB) system

• To allow non-technical users, who are not aware of formal database language (such as SQL) by interacting with database using their own English language
Research Challenges

• To derive the users’ intended meaning from limited (sometimes, only few words) number of query words.

• To map Natural language to Structured database language for extracting the relevant output from database.

For example,

In user query, direct column or field name not provided, still it retrieves the data from the database.

<table>
<thead>
<tr>
<th>User Query</th>
<th>SQL Statement Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Give phone number of student amit</td>
<td>Select contact_no from student where (studnm like ‘%amit%’)</td>
</tr>
</tbody>
</table>
Research Challenges (Contd..)

For example,

Here the token “state” can be an attribute or a stop word.

<table>
<thead>
<tr>
<th>User Query</th>
<th>SQL Statement Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>state the number of students who live in gujarat state</td>
<td>select count(*) from student where (state like '%gujarat%')</td>
</tr>
</tbody>
</table>

The word “branch” can be a attribute name and also a table name where join condition apply.

<table>
<thead>
<tr>
<th>User Query</th>
<th>SQL Statement Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many female stud from mca branch?</td>
<td>select count(*) as &quot;Total&quot; from stud NATURAL JOIN branch where (gender like 'female') and (branchnm like '%mca%')</td>
</tr>
</tbody>
</table>
Research Benefits

• Provides means of accessing information in the database independent of its structure and encoding knowledge

• Easy to understand and retrieve data for naïve users

• Flexible and generalized framework for all domains
Literature Survey on NLIDB

SQ-HAL – (Runvanpura)

- *Uses NLQ and converts into SQL.*
- Platform independent
- Multi-user support.
- Written in Perl
- Uses top down parser methodology.
Literature Survey on NLIDB (Contd..)

SQ-HAL system Limitations:
• String values need to be quoted,
• Date value needs hashes (#) around the values,
• Limited thesaurus,
• User has to manually entered the relationships,
• No direct method of retrieving column name,
• System cannot determine synonym for table name and column names,
• User has to manually enter synonym words,
• Does not support Microsoft Access.
Literature Survey on NLIDB (Contd..)

**CINDI** - Concordia Virtual Library System - (Niculae Stratica).

- Uses natural language input
- Gives structured representation of the answer in the form of SQL.
- Uses Link Parser to semantically parse the query.
- Uses WordNet to build the conceptual knowledge base from the database schema.

**CINDI system - Limitations:**

- Values should be in double quotes,
- Table name and attribute name should be specified,
- Template should be specified.
Neelu Nihalani discussed NLI to database using semantic matching.

- The system was tested for Northwind Database and compared with MS English Query product.
- The main functionality is based on semantics and rules.
- It is composed of two modules: a pre-processor and run time processor.

Limitations:
- Does not deal with genitive relations of the sentences.
  For example (i) “get details of an employee” – Generates SQL, (ii) “get employee details” – Does not Generate SQL.
- It does not give answers to elliptical questions.
GINLIDB - Generic Interactive Natural Language Interface to Database (Faraj).

The system dealt only limited domain and answered a small set of queries and was tested using VB.NET 2005.

Limitations:

• System depends on the size and content of the system's knowledge base.
• When the query is not available as per Augmented Transition Network Parser rule, then the query is rejected.
• Besides, in the input query, user has to explicitly specify the attribute name.

For example, user asks “display employee location”, - system does not recognize this query and user asks “display employee address”, the system recognized this query.
Literature Survey on NLIDB (Contd..)

Himani Jain - Hindi Language Interface to Database.
- Uses Shakti Standard Format - parsing a sentence.
- Developed in Java with MySQL, employee database is used.

Limitations:
- System does not deal with Linguistic Components.
- Directly maps user keywords to database entity names.
- Does not deal with complex queries, range queries, order by queries, group queries, and temporal queries.
- Does not support DDL statements.
Literature Survey - Conclusion

Looking at various limitations of existing NLIDB systems, Knowledge gap exists in this field, particularly:

• Lack of User Friendliness
• Lack of effective utilization
• Only limited domain knowledge
• Only limited set of semantic words used, etc.

Hence, we designed and developed a Natural-English Language Interface to Database (N-ELIDB).
Silent Features of N-ELIDB

- User can input the query in natural language like English
- Allows relaxation in grammatical rules while entering query
- String value not necessary to be in quotes
- Can enter the sentence in free form
- Direct mapping to column name with user words are not necessary
- It supports spell checker
- It supports context resolution
- It provides multiple data format
- Semantic of words are considered
- Knowledge can be preserved for reusability
- Lexicon analysis is done in detailed manner
- Not only SELECT statement but SQL of DDL and DML are also generated by the system, etc.
Architecture of N-ELIDB
Architecture of N-ELIDB

It has two major components:
(a) Linguistic Component, and (b) Database Component.

• The **Linguistic Component** translates the natural language input to an expression of Intermediate Query Representation (IQR), which is subsequently passed to **Database Component** for generation of Structured Query Language (SQL) statement. The resulting SQL statement is then executed by relational database management system.

• The Linguistic Component consists of morphological analysis, query pre-processing & context resolution, lexical analysis, syntactical analysis and semantic analysis; and

• Database Component consists of SQL query generation and SQL query execution.
Linguistic Component
Morphological Analysis

Morphology is the study of the way words are built up from smaller meaning-bearing units called morphemes. It is also known as Lemmatization, which is a process of analyzing the token morphologically in order to find their basic forms.

<table>
<thead>
<tr>
<th>Sample Tokens</th>
<th>Stem Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students</td>
<td>Student</td>
</tr>
<tr>
<td>Staying, Stays, Stayed</td>
<td>Stay</td>
</tr>
<tr>
<td>Locating, Location</td>
<td>Locate</td>
</tr>
<tr>
<td>Studying, Studies</td>
<td>Study</td>
</tr>
<tr>
<td>Branches</td>
<td>Branch</td>
</tr>
<tr>
<td>Details</td>
<td>Detail</td>
</tr>
<tr>
<td>Youngest, Younger</td>
<td>Young</td>
</tr>
</tbody>
</table>
Linguistic Component
Query Pre-Processing

The pre-processing of the input query includes:

- Stop words removal
- Word based n-gram generation and its conversion into base words
- Spelling check
- Identifying domain
- Knowledge reuse.
Query Pre-processing
Stop word Removal

Stop words are non context bearing words, also known as noisy words which are to be excluded from the input sentence to speed up the process.

<table>
<thead>
<tr>
<th>StopWords</th>
</tr>
</thead>
<tbody>
<tr>
<td>again, already, amongst, any, about, against, also, anyhow, around, although, anyone, across, available, always, a, an, anything, alone, am, anyway, afterwards, along, among, another, anywhere, back, be, became, because, been, behind, beside, beyond, but, call, connect, carry, do, done, etc, either, else, everyone, everything, fill, further, here, my, nobody, nothing, other, please, he, she, somehow, it, the, thereafter, therefore, through, together, towards, whenever, your, yourself, etc.</td>
</tr>
</tbody>
</table>
Query Pre-processing
Word based N-gram conversion

The n-gram is a contiguous sequence of n items from a given sequence of text.

<table>
<thead>
<tr>
<th>Sample Sequence</th>
<th>Unigram</th>
<th>Bi-gram</th>
<th>Tri-gram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master of</td>
<td>Master, of,</td>
<td>Master of,</td>
<td>Master of</td>
</tr>
<tr>
<td>Computer</td>
<td>Computer,</td>
<td>Computer</td>
<td>Computer, of</td>
</tr>
<tr>
<td>Applications</td>
<td>Applications</td>
<td>Applications,</td>
<td>Computer</td>
</tr>
<tr>
<td></td>
<td>of Computers</td>
<td>of Computers</td>
<td>Applications</td>
</tr>
</tbody>
</table>
Query Pre processing
Spelling Check Method

Three most popular method:

• **Insertion**: mistyping *the* as *ther*
• **Deletion**: mistyping *the* as *th*
• **Substitution**: mistyping *the* as *thw*

<table>
<thead>
<tr>
<th>Correct Token</th>
<th>Error Token</th>
<th>Spell Check Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS Student</td>
<td>Dstudent, Studant</td>
<td>Substitution</td>
</tr>
<tr>
<td></td>
<td>Studnt</td>
<td>Insertion</td>
</tr>
<tr>
<td></td>
<td>Studeent</td>
<td>Deletion</td>
</tr>
</tbody>
</table>
Query Preprocessing

Context Resolution

The context can be resolved by identifying the domain class in form of `<database name> <domain name> <key terms> <attribute terms>`

<table>
<thead>
<tr>
<th>Database name</th>
<th>Domain name</th>
<th>Key terms</th>
<th>Attribute terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Student</td>
<td>roll no, name, villa, residence, phone no, mobile no, department, etc.</td>
<td>stud_id, contact_no, address, birthdate, studnm, city, state, sem_no, per, branchid, gender, etc.</td>
</tr>
<tr>
<td>Exam</td>
<td>Student</td>
<td>exam, percentage, semester, branch</td>
<td>examid, examnm, sem, mks, etc.</td>
</tr>
<tr>
<td>Library</td>
<td>Student, Faculty</td>
<td>books, return, issue, library, author, edition, title, publisher, etc.</td>
<td>bookid, booknm, isbnno, authornm, publisherid, etc.</td>
</tr>
<tr>
<td>Branch</td>
<td>Student</td>
<td>department, course</td>
<td>branchid, branchnm</td>
</tr>
</tbody>
</table>
Query Pre-processing
Knowledge Reuse

- Query stored in the knowledge base- can be used as a template from GUI, can modify template and resubmit the query.
- Keeping user search history and providing a query template preserves user’s prior search effort, and gives quick starting point when he/she needs to create new queries.
Linguistic Component
Lexical Analysis

Each identified tokens can be represented as **attribute token, value token, core token, multi-token, continuous token, etc.**

- Attribute token- using metadata
- Core Token-first, all capital letters
- Numeric Token-digits, digits separated by decimal point
- Sentence Ending Markers- (. ? !)
- Value Token- (M.C.A, “mca”, ‘mca’)
- Continuous Token – (‘@’, apostrophe (‘) , ‘$’)
- Multi-token- emp_no or e-no
- Abbreviated Token- CE for Computer Engineering
Linguistic Component
Syntactic Analysis

- Syntactic Analysis is also called Hierarchical analysis/Parsing, used to recognize a sentence, to allocate token groups into grammatical phrases and to assign a syntactic structure to it.

- The Stanford Parser is a probabilistic parser.

- The Stanford dependencies provide a representation of grammatical relations between words in a sentence.

- Using Stanford Parser, we developed a Multi-Liaison algorithm that takes input in the form of template <subject> <verb> <object>.

- It is also useful for researchers who are doing research in Natural Language Processing and Text Mining.
Syntactic Analysis
Multi-Liaison Algorithm

• The Multi-Liaison Algorithm would display English statement with connecting verbs or predicates.

• The sentence is parsed with the help of the Stanford parser and output of the parser is used as an input to our algorithm for finding all the subjects, objects and the predicates.

• The algorithm is written in JAVA using Net Beans IDE 6.5 and is tested with a variety of inputs.
Syntactic Analysis
Multi-Liaison Algorithm

Function: CONVERT_SENTENCE (Input_Str)
Returns: Output_Str

Input_Str: Sentence to be parsed
[Run the Stanford parser with Input_Str as input]
Output_Str ← i) POS of each word
          ii) The parse tree generated
          iii) The typed dependencies

Function: MULTI_LIAISON (Input_Str)
Returns: Multiple liaisons or error message

Input_Str: Output_Str
T = Call Function GET_TRIPLETS (Output_Str)
R = Call Function GET_RELATIONSHIP (Output_Str)
Return multiple liaisons T and R
Example 1: The Puja and Reema work together for Dissertation work in their college

The Stanford Parser output:

Tagging:
The/DT Puja/NNP and/CC Reema/NNP work/VBP together/RB for/IN Dissertation/JJ work/NN in/IN their/PRP$ college/NN

Parse Tree:
(ROOT
  (S
    (NP (DT The) (NNP Puja)
      (CC and)
      (NNP Reema))
    (VP (VBP work)
      (ADVP (RB together))
      (PP (IN for)
        (NP
          (NP (JJ Dissertation) (NN work))
          (PP (IN in)
            (NP (PRP$ their) (NN college))))))))

Typed Dependencies:
det(Puja-2, The-1)
nsubj(work-5, Puja-2)
conj_and(Puja-2, Reema-4)
nsubj(work-5, Reema-4)
advmod(work-5, together-6)
amod(work-9, Dissertation-8)
prep_for(work-5, work-9)
poss(college-12, work-9)
prep_in(work-9, college-12)
The Puja and Reema work together for dissertation work in their college.
Output of Multi-Liaison Algorithm

The Multi-Liaison Output

Subject: 2

NNP Puja  DT The

NNP Reema

Predicate: 1

VBP work

Object: 2

NN work  JJ Dissertation

NN college  PRP$ their

Relationship:

Puja - work - Dissertation - work - college

Reema - work - Dissertation - work - college
Linguistic Component
Semantic Analysis

We have used WordNet to know the semantics of particular token.

Example 1: List of students staying in gujarat
Input: Token = Gujarat

The WordNet output:
Jun 26, 2014 4:29:50 PM net.didion.jwnl.util.MessageLog doLogINFO:
Installing dictionary net.didion.jwnl.dictionary.FileBackedDictionary@9df354

Senses of the word 'gujarat':
First char is capital : 1,  Hyponym : 1,  Sense no : 1

hypernymy in wordnet : entity,  physical_entity
hypernymy in wordnet : object,  physical_object
hypernymy in wordnet : location

value of t in wordnet == 3
Semantic Analysis
Lexicon Semantic Representation

It is another form of representation for user tokens and user input symbols in the form of semantic word.

<table>
<thead>
<tr>
<th>User tokens</th>
<th>Semantic Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>phone number, mobile number, contact number, cell number, telephone no, residential number, office phone no, college phone no, dial no</td>
<td>contact-no</td>
</tr>
<tr>
<td>percentage above 70</td>
<td>distinction</td>
</tr>
<tr>
<td>address, place, stay, living, situated, residence, dwelling, venue, locus point, domicile</td>
<td>location</td>
</tr>
</tbody>
</table>
Database Component

- Database component consists of SQL Generation and SQL Execution.

- The SQL Generation component takes an Intermediate Query (IQ) as an input from Linguistic component and generates an equivalent SQL query as output.

- The SQL Execution component executes the generated SQL query and displays appropriate query output or message on GUI to user.
Database Component
SQL Generation - Grammar

<select query> ::= <select> [ALL] [DISTINCT] <select query>
<select> ::= SELECT <select list> <from clause> [ <where clause> ]
<select list> ::= ‘*’ | <column element> [ { ‘,’ <column element> }... ]
<column element> ::= <column> | [<aggregate function>] ‘(’ <column> ‘)’
<from clause> ::= FROM <table reference>[{‘,’<table reference>}...]
<table name> ::= tablename
<where clause> ::= WHERE <search conditions>
<search condition> ::= <condition> with logical predicate (AND/OR/NOT)
<condition> ::= Expression in form of $X_RY$
<group by > ::=<column element>
$order by > ::= <column element> [desc]

Note: X and Y are the set of values representing column, constant and R can be operator like {<,>,<=, >=, !=,==}
Database Component
SQL Generation - Grammar

<Insert Query> := <insert> INTO <tablename> VALUES <values>
   <insert> := INSERT [<column element>]
   <tablename> := tablename
   <column element> := <column>
   <values> := set of values representing columns or constant value

<Update Query> := <Update> <table name> SET <column element> = <expression>
   [WHERE <condition>]
   <Update> := UPDATE
   <tablename> := tablename
   <column element> := <column>
   <expression> := constant values or expression in form of $X_R Y$

<Delete Query> := <delete> FROM <table name> [WHERE <condition>]
   <delete> := DELETE
   <tablename> := tablename
   <condition> := Expression in form of $X_R Y$

Note: X and Y are the set of values representing column, constant and R can be operator like {<,=, >, !<=, ==}
Database Component
SQL Generation - Grammar

<Create Statement> ::= <create statement> <table reference> <column element> <datatype> <constraint>]

<create statement> ::= Create
<table reference> ::= Table tablename
<column element> ::= <column>
<datatype> ::= int/char/varchar2/float/date
<constraint> ::= primary key/reference key/unique key/not null/check

<Drop statement> ::= <drop statement> <table reference>
<drop statement> ::= DROP
<table reference> ::= TABLE tablename
Example 1: list name, address and phone no of student who live in Surat city

**DOMAIN**: stud  
**DATABASE NAME**: stud  
**Base str**: give studnm, address and contact_no of stud who location in Surat city  
**POS tagging**: words: give/VB studnm/NN address/NN and/CC contact_no/NN of/IN stud/NN who/WP location/VBN in/INSurat/NNP city/NN

**Intermediate representation:**
- iobj(give-1, studnm-2)
- dobj(give-1, address-3)
- nsubj(location-9, address-3)
- dobj(give-1, contact_no-5)
- conj_and(address-3, contact_no-5)
- prep_of(address-3, stud-7)
- rcmmod(address-3, location-9)
- nn(city-12, Surat-11)
- prep_in(location-9, city-12)

**Entity**: stud;  
**Attribute**: studnm; address; contact_no;  
**Filtering condition**: (city like '%surat%');  
**Generated SQL** = `select studnm, address, contact_no from stud where (city like '%surat%')`
Database Component
Function - Insert

Function Insert (User Query UQ)
Returns: Generated Insert Statement CT

Input : User Query
// add the data into table t1 with value 11, ami, 89000
Tokenize the UQ with keyword ‘table’ and ‘value’ as delimiter and store it into TB (table) and VB (values)

// generate Insert statement
CT= Insert into TB values VB
Return  CT
Database Component  
Function - Update

Function Update (User Query UQ)  
Returns: Generated Update Statement CT

Input: User Query UQ  
// edit the table t1 with value sal = sal * 0.5 where empno = 1

Tokenize the UQ with keyword “table”, “value” and “where” clause and store it into TB(table), VB(values) and WH(where condition).

//Generate Update statement  
CT = Update TB set VB = expression where WH  
Return CT
Database Component

Function – Delete

Function Delete (User Query UQ)
Returns: Generated Delete Statement CT

Input: User Query UQ // delete all the data from emp table or delete the data from emp table where id = 1

Tokenize the UQ with keyword “table”, “where” and store into TB(table) and WH(where clause).

//Generate Delete statement
If “where” clause not present then
   CT= Delete from TB
Else
   CT= Delete from TB where WH.
End If
Return CT
Database Component
Function-CreateTable

Function CreateTable (User Query UQ)
Returns: Generated Create Statement CT
Input: User Query
// make table t1 with field empno as int, empnm as varchar, sal as float
   Tokenize the UQ with keyword ‘table ’, ‘field’ and ‘as’ as delimiter and store it into TB, FT, DT
[ Change the datatype of user specified with MySQL datatype]

// Generate SQL statement
CT= Create table TB( FT DT, FT DT)
Return CT
Database Component
Function-DropTable

Function Drop (User Query UQ)
Returns: Generated Drop Statement CT
Input: User Query // remove table emp

// Tokenize the UQ with keyword “table “and store it as TB

// generate Drop statement
CT= Drop table TB
Return CT
Function Retrieve (User Query UQ)

Returns: CT

Input : User Query UQ

// select from table emp

    Tokenize the UQ with keyword “table” and store it into TB

//Generate Select statement

CT= Select * from TB

Return CT
Database Component
SQL Query Executor

• The retrieved database tuples contain answers and are stored in file rather than displaying directly in grid format as specified by the SQL database output.

• When the system may not able to search the particular tuples, then the appropriate response is sent to the user. For example, no record found.

• When the system may not understand the question, then no tuples will be retrieved and system may send appropriate response to the user. For example, the query cannot be generated or no domain found.
Database Component
Any2MySQL Tool
Database Component
General Algorithm for Data Conversion

Step 1: Select text file (.csv or .txt file), Excel file (.xls file), MS-Access file (.mdb file), Oracle file, SQL Server file, Extensible Markup Language file (.xml) as an input to the system through Interface.
Step 2: For Oracle selected file, Input Server name, port number, sid, username and password through Interface.
Step 3: Create a connection link to selected source.
Step 4: After connection – Get Metadata- table name, attribute names, data types through system.
Step 5: Create connection link -MySQL database.
Step 6: Compare the data types of source and destination files.
Step 7: Formulate a CREATE query of each table and Execute.
Step 8: Formulate an Insert query of each record for each table and Execute.
Step 9: Refresh query browser of MySQL and view the tables and records.
Database Component
Any2MySQL Tool

Select Your DB/File Type:
- Notepad File (.txt tabbed file)
- Notepad File (.txt tabbed file)
- Comma Separated File (.csv)
- Excel (.xls)
- My Access (.mdb), (.accdb)
- XML (.xml)
- Oracle Database

Look In:
- My Documents

File Name:

Files of Type: All Files

Convert DB/File  Cancel
Database Component
Any2MySQL Tool
Database Component

Any2MySQL Tool

<table>
<thead>
<tr>
<th>faculty_id</th>
<th>name</th>
<th>Design</th>
<th>Exp_Teach</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD09</td>
<td>Ms. Anisha Shingala</td>
<td>Lecturer</td>
<td>12 yrs</td>
</tr>
<tr>
<td>MD10</td>
<td>Dr. Sandeep Vij</td>
<td>Director</td>
<td>10 yrs</td>
</tr>
<tr>
<td>MD11</td>
<td>Mr. Prasad Gupta</td>
<td>Lecturer</td>
<td>7 yrs</td>
</tr>
<tr>
<td>MD12</td>
<td>Dr. Rohan Gupta</td>
<td>Lecturer</td>
<td>11 yrs</td>
</tr>
<tr>
<td>MD13</td>
<td>Mr. Prabhat Patil</td>
<td>Lab. Technician</td>
<td>10 yrs</td>
</tr>
<tr>
<td>MD14</td>
<td>Mr. Shubham Singh</td>
<td>Lab. Technician</td>
<td>10 yrs</td>
</tr>
<tr>
<td>MD15</td>
<td>Mr. Manoj Patel</td>
<td>Lecturer</td>
<td>3 yrs</td>
</tr>
<tr>
<td>MD16</td>
<td>Mr. Ayush Singh</td>
<td>Lecturer</td>
<td>5 yrs</td>
</tr>
<tr>
<td>MD17</td>
<td>Mr. N. Prasad</td>
<td>Lecturer</td>
<td>3 yrs</td>
</tr>
<tr>
<td>MD18</td>
<td>Mr. Virendra Singh</td>
<td>Lecturer</td>
<td>5 yrs</td>
</tr>
<tr>
<td>MD19</td>
<td>Ms. Palna Patil</td>
<td>Lecturer</td>
<td>3 yrs</td>
</tr>
</tbody>
</table>
Tools and Technology

- We have used Stanford Parser since it contains a Java implementation of probabilistic Natural Language (English) Parsers and gives visualization of the parse tree using its GUI.
- We have used WordNet tool (English) to match lexically and semantically words.
- We have used Java Programming Language, NetBeans 6.5 IDE (with JDK 1.6).
- We have used MySQL as Relational Database and its GUI as SQL-YOG.
- The following Java APIs are used in development of the prototype:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Java APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Java.awt</td>
<td>Contains all the classes for creating user interfaces and for painting graphics and images.</td>
</tr>
<tr>
<td>2</td>
<td>Java.beans</td>
<td>Contains classes related to developing beans</td>
</tr>
<tr>
<td>3</td>
<td>Java.io</td>
<td>Provides for system input and output through data stream, serialization and file systems</td>
</tr>
<tr>
<td>4</td>
<td>Java.sql</td>
<td>Provides the API for accessing and processing data stored in a data source (usually a relational database) using Java programming language.</td>
</tr>
<tr>
<td>5</td>
<td>Java.util</td>
<td>Contains the collection framework, legacy collection classes, event model, date and time facilities, internationalization and miscellaneous utility classes.</td>
</tr>
<tr>
<td>6</td>
<td>Java.util.logging</td>
<td>Provides the classes and interfaces of Java platform core logging facilities.</td>
</tr>
<tr>
<td>7</td>
<td>Java.lang</td>
<td>Provides classes that are fundamental to the design of the Java programming language.</td>
</tr>
<tr>
<td>8</td>
<td>Javax.swing</td>
<td>Provides a set of lightweight components that, to the maximum degree possible, work the same way on all platforms.</td>
</tr>
</tbody>
</table>
Tools and Technology (Contd..)

JAVA JAR (Java Archives) is a package format typically used to aggregate many Java class files and associated metadata and resources such as text, images, etc.

We have used the following JAR files in the prototype implementation:

- Mysql-connector-java-5.1.18-bin.jar
- Stanford-parser-2010-02-26.jar
- Junit-4.1.jar
- Jwnl.jar
- Commons-logging.jar
- JDK1.6 (Default)
# Databases Used

## Student Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data type &amp; Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stud_id</td>
<td>Varchar2(7)</td>
<td>Student identification number</td>
</tr>
<tr>
<td>studnm</td>
<td>Varchar2(50)</td>
<td>Name of student</td>
</tr>
<tr>
<td>address</td>
<td>Varchar2(255)</td>
<td>Address of student</td>
</tr>
<tr>
<td>state</td>
<td>Varchar2(10)</td>
<td>State of student</td>
</tr>
<tr>
<td>city</td>
<td>Varchar2(50)</td>
<td>City of student</td>
</tr>
<tr>
<td>contact_no</td>
<td>Varchar2(30)</td>
<td>Phone number of student</td>
</tr>
<tr>
<td>gender</td>
<td>Varchar2(6)</td>
<td>Gender (M/F) of student</td>
</tr>
<tr>
<td>birthdate</td>
<td>Datetime</td>
<td>Birthdate of student</td>
</tr>
<tr>
<td>sem_no</td>
<td>Integer</td>
<td>Semester of student</td>
</tr>
<tr>
<td>branch_id</td>
<td>Integer</td>
<td>Branch identification number</td>
</tr>
<tr>
<td>percentage</td>
<td>Integer</td>
<td>Percentage of student</td>
</tr>
</tbody>
</table>
## Databases Used (Contd..)

### Branch Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data type &amp; Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch_id</td>
<td>Integer</td>
<td>Branch identification number</td>
</tr>
<tr>
<td>branchnm</td>
<td>Varchar2(10)</td>
<td>Branch name</td>
</tr>
</tbody>
</table>

### Knowledge Base

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data type &amp; Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>que</td>
<td>Varchar2(250)</td>
<td>User question/query</td>
</tr>
<tr>
<td>sqlquery</td>
<td>Varchar2(250)</td>
<td>Generated SQL query</td>
</tr>
<tr>
<td>status</td>
<td>Integer</td>
<td>Status 1 for successfully generated else 0</td>
</tr>
<tr>
<td>domain</td>
<td>Varchar2(10)</td>
<td>Domain of question category</td>
</tr>
<tr>
<td>cnt</td>
<td>Integer</td>
<td>Count of same question asked</td>
</tr>
<tr>
<td>last_dt</td>
<td>Datetime</td>
<td>Last date and time of question input</td>
</tr>
</tbody>
</table>
### Categories of the Natural Language Queries supported by N-ELIDB

<table>
<thead>
<tr>
<th>Category of Query</th>
<th>Description</th>
<th>Category of Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>List Queries</td>
<td>E</td>
<td>Time Oriented Queries</td>
</tr>
<tr>
<td>B1</td>
<td>Single Condition Queries</td>
<td>F</td>
<td>Range Queries</td>
</tr>
<tr>
<td>B2</td>
<td>Composite Condition Queries</td>
<td>G</td>
<td>Negation Queries</td>
</tr>
<tr>
<td>C</td>
<td>Group Queries</td>
<td>H</td>
<td>Join Queries</td>
</tr>
<tr>
<td>D</td>
<td>Order Queries</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A : List Queries

List queries that includes the single table query without condition (all rows will be retrieved) and columns can be in form of \( c_1, c_2 \ldots c_n \) or \( \text{count}(*) \) or \( c_1 \) and \( c_2 \ldots c_n \) or different \( c_1 \) etc.

The query may be paraphrase in different form such as:

- List student details,
- List of students,
- Give me list of students,
- Student list,
- Give address of stud,
- Name of all students,
- Show me address and name of students,
- Show student list, etc.
<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>list student details</td>
<td>select * from stud</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>tell me data of all students</td>
<td>select * from stud</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>give name, address and phone no of all students</td>
<td>select studnm, address, contact_no from stud</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>count total number of students</td>
<td>select count(*) as &quot;Total&quot; from stud</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>display name of student and its birthdate</td>
<td>select studnm, birthdate from stud</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>list bloodgr of stud</td>
<td>select * from stud where (studnm like '%bloodgr%' or state like '%bloodgr%' or city like '%bloodgr%') ;</td>
<td>I</td>
</tr>
<tr>
<td>7</td>
<td>List all student name and unique percentage</td>
<td>select distinct(percentage), studnm from stud</td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>show me rollno and semester of students</td>
<td>select stud_id,sem_no from stud</td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>get student record</td>
<td>select * from stud</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>List unique name, phone no and sem of student</td>
<td>select distinct(studnm), contact_no,sem_no from stud</td>
<td>C</td>
</tr>
</tbody>
</table>
A: LIST QUERIES-User Interface
A: List Queries

System Performance for Category A Query

Number of Queries

Category A Query

Correct
Partial
Incorrect
## B1: Single Condition Queries

These types of queries require a information to be listed from a single table based on specified condition(single).

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>count total number of stud live in anand city</td>
<td><code>select count(*) as &quot;Total&quot; from stud where (city like '%anand%' ) and (city like '%anand%' ) ;</code></td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>list of students who lives in city baroda</td>
<td>`select * from stud where (city like '%vadodara%' ) ;</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>list of stud who secured more than 60%,</td>
<td>`select * from stud where percentage &gt;60 ;</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>list of students whose birthdate is in month of february</td>
<td>`select * from stud where month(birthdate)=02 ;</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>In which semester student mona study?</td>
<td>`select sem_no from stud ;</td>
<td>P</td>
</tr>
<tr>
<td>6</td>
<td>detail of student from maharashtra state</td>
<td>`select * from stud where (state like 'maharashtra%' ) ;</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>Mention the student detail chirag</td>
<td>`select * from stud where (studnm like '%chirag%' ) ;</td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>Where does student rinku stay?</td>
<td>Select address from stud ;</td>
<td>I</td>
</tr>
<tr>
<td>9</td>
<td>who is 09mcg20 student?</td>
<td>`select * from stud where (stud_id like '%09mcg20%' ) ;</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>give phone number of student whose name starts with ami</td>
<td>`select contact_no,studnm from stud where studnm like 'ami%' ;</td>
<td>C</td>
</tr>
</tbody>
</table>
B1: Single Condition Queries

System Performance for Category B1 Query

Number of Queries

Correct
Partial
Incorrect

Category B1 query
### B2: Composite Condition Query

*Composite condition queries* include the rows and columns to be retrieved based on more than one condition for a single table.

<table>
<thead>
<tr>
<th>Sr.No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>List address of female student who are available in sem III</td>
<td><code>select address from stud where gender='female' and sem_no=03</code></td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>List of students who lives in city baroda studying in sem 4</td>
<td><code>select * from stud where (city like '%vadodara%') and (city like '%'vadodara%' ) and sem_no=04</code></td>
<td>P</td>
</tr>
<tr>
<td>3</td>
<td>List of boy student who lives in gujarat state</td>
<td><code>select * from stud where gender='male' and (state like '%gujarat%' )</code></td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>Find 3rd highest percentage in sem 6 whose gender male</td>
<td><code>select percentage from stud where sem_no=06 and (gender like 'male' )</code></td>
<td>P</td>
</tr>
<tr>
<td>5</td>
<td>List of stud whose per greater than 50 or less than 55</td>
<td><code>Select * from student where percentage &lt; 55 and percentage &gt; 50</code></td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>How many female students from anand and baroda city</td>
<td><code>select count(*) as &quot;Total&quot; from stud where (gender like 'female' ) and (city like '%vadodara%' ) and (studnm like '%'Anand%' or state like '%Anand%' or city like '%Anand%')</code></td>
<td>P</td>
</tr>
<tr>
<td>7</td>
<td>List student detail whose percentage is greater than 70 and lives in surat city</td>
<td><code>select * from stud where percentage &gt;70 and (city like '%surat%' ) and (city like '%surat%')</code></td>
<td>P</td>
</tr>
<tr>
<td>8</td>
<td>List name of student who score more than 50 per in sem 5</td>
<td><code>select studnm from stud where percentage &gt;50 and sem_no=05</code></td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>Number of students whose percentage less than 70 in sem 5</td>
<td><code>select * from stud where percentage &lt; 70 and sem_no=05</code></td>
<td>P</td>
</tr>
<tr>
<td>10</td>
<td>Details of girl student who secured first rank in sem 4</td>
<td><code>select * from stud where percentage= ( select max(percentage) from stud ) and sem_no=04</code></td>
<td>C</td>
</tr>
</tbody>
</table>
B2: Composite Condition Query - User Interface
B2: Composite Condition Query

System Performance for Category B2 Query

- Correct: 5
- Partial: 5
- Incorrect: 0

Category B2 Query

Number of Queries
C: Group Query

*Group query* that includes the group or category wise display of record. It does not include the condition.

The paraphrases of such queries can be in the form:

- count total number of student semwise,
- percentagewise count of students,
- total number of student genderwise,
- branchwise total number of students, etc.
<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Give details of stud group by sem</td>
<td><code>select * from stud group by sem_no</code></td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>percentagewise count of students</td>
<td><code>select percentage, count(*) as &quot;Total&quot; from stud group by percentage</code></td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>Branchwise total number of students</td>
<td><code>select branchid, count(*) as &quot;Total&quot; from stud group by branchid</code></td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>total no of stud genderwise</td>
<td><code>select gender, count(*) as &quot;Total&quot; from stud group by gender</code></td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>list of students group by city</td>
<td><code>select city, count(*) as &quot;Total&quot; from stud group by city</code></td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>Number of stud group by rollno</td>
<td><code>select stud_id, count(*) as &quot;Total&quot; from stud group by stud_id</code></td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>list of female students group by sem</td>
<td><code>select sem_no, count(*) as &quot;Total&quot; from stud group by sem_no</code></td>
<td>P</td>
</tr>
<tr>
<td>8</td>
<td>List student details based on percentage</td>
<td><code>select percentage, count(*) as &quot;Total&quot; from stud group by percentage</code></td>
<td>P</td>
</tr>
<tr>
<td>9</td>
<td>Genderwise total number of stud</td>
<td><code>select gender, count(*) as &quot;Total&quot; from stud group by gender</code></td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>Count of Stud records Group by semester</td>
<td><code>select sem_no, count(*) as &quot;Total&quot; from stud group by sem_no</code></td>
<td>C</td>
</tr>
</tbody>
</table>
C: Group Query - User Interface

Enter Query: total number of students branchwise

Dynamic List:

SQL Query:

select branchid, count(*) as "Total" from student group by branchid ;

Output:

- branchid: 1
  Total: 58
- branchid: 2
  Total: 75
- branchid: 3
  Total: 23
C: Group Query

System Performance for Category C Query

- Correct
- Partial
- Incorrect

Number of Queries

Category C Query
D: Order Query

*Order queries* include the rows to be retrieved based on ascending or descending order. It includes single or composite conditions.

The paraphrases of such queries are:

- *semwise list of students*,
- *list of students genderwise*,
- *list female students percentagewise*,
- *show me list of students branchwise*,
- *list of students order by percentage*,
- *list of boy student order by semester*,
- *list of students in descending order of semester*, etc.
## D: Order Queries

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Give me the sorted list of students by name</td>
<td>select * from stud</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>show me address of female stud citywise</td>
<td>select address from stud where gender='female' order by city</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>list of stud in descending order of their name</td>
<td>select * from stud order by studnm desc ;</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>Detail of stud branchwise</td>
<td>select * from stud order by branchid</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>list name of stud in asc order of their city</td>
<td>select studnm from stud order by city</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>List the student name by state</td>
<td>select studnm, state from stud</td>
<td>I</td>
</tr>
<tr>
<td>7</td>
<td>List address of stud genderwise</td>
<td>select address from stud order by gender</td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>List of female students citywise</td>
<td>select * from stud where gender='female' order by city</td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>give name of student who study in sem 3 citywise</td>
<td>select studnm,sem_no from stud order by city ;</td>
<td>P</td>
</tr>
<tr>
<td>10</td>
<td>list name of girl student order by their semester</td>
<td>select studnm from stud where (gender like 'female') order by sem_no</td>
<td>C</td>
</tr>
</tbody>
</table>
D: Order Query – User Interface

Enter Query: citywise list of students

Dynamic List:

SQL Query:
```
SELECT * FROM STUD ORDER BY CITY;
```

Output:
```
student_id: DTMC006
student: RAVI KUMAR SHAH
H:\naddress: 718/37 KANAPALI'S NEW CHAL, INR. SBI, KABIRCHOWK, SABARMATI
state: GUJARAT
city: Ahmedabad
contact_no: 079 27502888, 322403 316891
```
D: Order Queries

System Performance for Category D Query

Number of Queries

Category D Query

- Correct
- Partial
- Incorrect
E: Time Oriented Query

*Time oriented queries* includes the temporal queries based on birthdate, day, month, year, today, age.

The different paraphrases of such queries are:
- *list of students whose age is below 25,*
- *give student details whose age greater than 25,*
- *list name of student whose birthday is on april month,*
- *show me phone number of student who born in year 1987*
- *list of student whose birthday is today,*
- *list youngest student,*
- *list name, branch and semester of oldest student,* etc.
<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>list of students whose age is below 25</td>
<td>select * from stud where (round(year(now())))-year(birthdate) &lt; 25</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>list name of student whose birthday is on april month</td>
<td>select studnm,birthdate from stud where month(birthdate)=04</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>List youngest student</td>
<td>select * from stud where birthdate=(select max(birthdate) from stud where birthdate&gt;0)</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>list name and semester of oldest student</td>
<td>select studnm, sem_no from stud where birthdate=(select min(birthdate) from stud where birthdate&gt;0)</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>list of stud whose birthday is today</td>
<td>select * from stud where (day(birthdate) = round(day(now()))))</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>list name and semester of student who born in april or june month</td>
<td>select studnm,sem_no from stud where month(birthdate)=04 and month(birthdate)=06</td>
<td>P</td>
</tr>
<tr>
<td>7</td>
<td>List of students whose birthday is in year 1978</td>
<td>select * from stud where year(birthdate)=1978</td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>List the students whose birthday comes in month of may</td>
<td>select * from stud where month(birthdate)=05</td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>List detail of student who born in november month</td>
<td>select * from stud where month(birthdate)=11</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>List of stud whose birthday is today</td>
<td>select * from stud where (day(birthdate) = round(day(now()))))</td>
<td>C</td>
</tr>
</tbody>
</table>
E: Time Oriented Query- User Interface

Enter Query: show me semester and phone number of student who born in month of March

Dynamic List:

SQL Query: select sem_no, contact_no from stud where month(birthdate)=03;

Output:

<table>
<thead>
<tr>
<th>sem_no</th>
<th>contact_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0265-254451, 9878675680</td>
</tr>
<tr>
<td>5</td>
<td>0265-267814, 9027968995</td>
</tr>
<tr>
<td>8</td>
<td>0265-278241, 9825708863</td>
</tr>
</tbody>
</table>
E: Time Oriented Query

System Performance for Category E Query

Number of Queries

Category E Query

Correct
Partial
Incorrect
F: Range Query

Range queries include the display of records in a given range in terms of ‘between’ keyword.

The different paraphrases of such queries are:

- list of students whose percentage between 50 and 60,
- give female students who study in sem 3 having percentage between 50 and 60,
- how many boy students who have percentage between 78 and 84, etc.
## F: Range Query

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>list of stud who born between month march to month may</td>
<td>select * from stud where month(birthdate)=03 and month(birthdate)=05</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>give me list of stud who got per between 100 and 90</td>
<td>select * from stud where percentage&gt;=100 and percentage&lt;=90</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>how many stud studying in sem 5,6</td>
<td>select count(*) as &quot;Total&quot; from stud where sem_no=05 and sem_no=06</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>list of stud studying in sem 2 to 4</td>
<td>select * from stud where (sem_no in (04,02))</td>
<td>P</td>
</tr>
<tr>
<td>5</td>
<td>how many stud who have per between 50 and 60</td>
<td>select count(*) as &quot;Total&quot; from stud where percentage&gt;=50 and percentage&lt;=60</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>list students who score between 60 and 90%</td>
<td>select * from stud where percentage&gt;=60 and percentage&lt;=90</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>give phone no, birthdate of stud who study in sem 3, 4 and 5</td>
<td>select contact_no, birthdate,sem_no from stud where (sem_no in (04,03))</td>
<td>P</td>
</tr>
<tr>
<td>8</td>
<td>list of stud whose study in sem 4 having percentage between 65 and 76</td>
<td>select * from stud where sem_no=04 and percentage&gt;=65 and percentage&lt;=76</td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>find male students who got percentage between 70 and 80</td>
<td>select * from stud where gender='male' and percentage&gt;=70 and percentage&lt;=80</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>show me the semester of student having per between 100 and 10</td>
<td>select sem_no from stud where percentage&gt;=100 and percentage&lt;=10</td>
<td>C</td>
</tr>
</tbody>
</table>
F: Range Query

System Performance for Category F Query

- Correct: 7
- Partial: 3
- Incorrect: 0

Number of Queries: 0 to 10
G:Negation Queries

These types of queries include keyword “not” in a sentence which indicates a negative result.

The different paraphrases of negation queries are:

- list name of student who do not live in city baroda,
- list name and address of student not born in month of april,
- give phone number of student who do not live in anand city, etc.
## G: Negation Query

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>list name of student who do not live in city surat</td>
<td>select studnm, city from stud where (city not like '%surat%')</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>how many students who do not stay in gujarat state?</td>
<td>select count(*) as &quot;Total&quot; from stud where (state not like '%gujarat%')</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>list mobile no of students who do not born in month of march</td>
<td>select contact_no from stud where month(birthdate) &lt;&gt; 03</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>how many students who do not born in year 1997</td>
<td>select count(*) as &quot;Total&quot; from stud where year(birthdate) &lt;&gt; 1997</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>list of stud who do not study in sem 3</td>
<td>select * from stud where sem_no=03</td>
<td>I</td>
</tr>
<tr>
<td>6</td>
<td>List student detail who does not live in valsad city</td>
<td>select * from stud where (city not like '%valsad%') and (city not like '%city%')</td>
<td>I</td>
</tr>
<tr>
<td>7</td>
<td>List student details who do not live in Gujarat</td>
<td>select * from stud where (state like '%gujarat%' or city like '%gujarat%')</td>
<td>P</td>
</tr>
<tr>
<td>8</td>
<td>list of stud who do not score 50 per</td>
<td>Select * from stud</td>
<td>I</td>
</tr>
<tr>
<td>9</td>
<td>how many stud who do not born in month of july</td>
<td>select count(*) as &quot;Total&quot; from stud where month(birthdate) &lt;&gt; 07</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>how many stud who do not live in valsad and baroda city</td>
<td>select count(*) as &quot;Total&quot; from stud where (city not in ('valsad', 'vadodara'))</td>
<td>C</td>
</tr>
</tbody>
</table>
G: Negation Query- User Interface

Enter Query:
list name and phone number of student who do not live in city baroda

Dynamic List:

SQL Query:
select studnm, contact_no, city from stud where (city not like '%krodara%');

Output:

studnm: KHATRI HIRAL
contact_no: 0265 5571008
city: bombay

studnm: RAVI PRASHANT
contact_no: 0773 27502983, 692240 31891
city: Ahmedabad
G: Negation Query

System Performance for Category G Query

- Correct
- Partial
- Incorrect

Number of Queries

Category G Query
H: Join Query

Join queries includes joining of two tables.

The different paraphrases of such queries are:

- list of students studying in mca department,
- list student who are in computer branch,
- list name and phone no of student who study in mca dept in semester 5,
- list of female student who study in computer branch, etc.
<table>
<thead>
<tr>
<th>Sr. No</th>
<th>User Input Query</th>
<th>System Generated SQL Queries</th>
<th>C/I/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Total male students in mca dept</td>
<td><code>select count(*) as &quot;Total&quot; from stud NATURAL JOIN branch where gender = 'm' and (branchnm like '%mca%')</code></td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>List of stud who is from computer branch</td>
<td><code>select * from stud NATURAL JOIN branch where (branchnm like '%ce%')</code></td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>List of students who top in computer dept</td>
<td><code>select * from stud NATURAL JOIN branch where percentage= (select max(percentage) from stud )</code></td>
<td>P</td>
</tr>
<tr>
<td>4</td>
<td>How many female stud from mca branch?</td>
<td><code>select count(*) as &quot;Total&quot; from stud NATURAL JOIN branch where (gender like 'female') and (branchnm like '%mca%')</code></td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>detail of student who study in sem 3 of computer dept</td>
<td><code>select * from stud NATURAL JOIN branch where sem_no = 03 and (branchnm like '%ce%')</code></td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>give address of stud bindi who study in sem 5 of mca dept</td>
<td><code>select address from stud NATURAL JOIN branch where (studnm like '%bindi%') and sem_no=05 and (branchnm like '%mca%')</code></td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>give detail of female stud from mca dept who study in sem 6 and lives in baroda city</td>
<td><code>select * from stud NATURAL JOIN branch where (gender like 'female') and (branchnm like '%mca%') and sem_no=06 and (city like '%vadodara%')</code></td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>list of male student from anand and surat city who study in mca dept</td>
<td><code>select * from stud NATURAL JOIN branch where gender='male' and (city in ('ANAND', 'SURAT')) and (branchnm like '%mca%')</code></td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>List of girl students who live in baroda city studying in sem 4 of mca dept</td>
<td><code>select * from stud NATURAL JOIN branch where gender='female' and (city like '%vadodara%') and (city like '%vadodara%') and sem_no=04 and (branchnm like '%mca%')</code></td>
<td>P</td>
</tr>
<tr>
<td>10</td>
<td>List of girl student who live in anand city studying in sem 3 of mca dept having percentage &gt;50</td>
<td><code>select * from stud NATURAL JOIN branch where gender='female' and (city like '%anand%') and (city like '%anand%') and sem_no=03 and (branchnm like '%mca%') and percentage &gt;50</code></td>
<td>P</td>
</tr>
</tbody>
</table>
H:Join Query- User Interface

Enter Query: list of students studying in MCA department

Dynamic List:

SQL Query:
```
SELECT * FROM stud NATURAL JOIN branch WHERE (branchNm like "%mca%");
```

Output:
```
<table>
<thead>
<tr>
<th>Address</th>
<th>Class</th>
<th>City</th>
<th>Contact_no</th>
<th>Gender</th>
<th>Birthdate</th>
<th>Sem_no</th>
<th>BranchId</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Address)</td>
<td>(Class)</td>
<td>(City)</td>
<td>(Contact_no)</td>
<td>(Gender)</td>
<td>(Birthdate)</td>
<td>(Sem_no)</td>
<td>(BranchId)</td>
</tr>
</tbody>
</table>
```
H: Join Query

System Performance for Category H Query

Number of Queries

Category H Query

Correct
Partial
Incorrect
Categorywise Accuracy

System Performance for all Categories of Query

Accuracy in Percentage

Different Categories of Query

- A
- B1
- B2
- C
- D
- E
- F
- G
- H
Analysis according to User Category

We have tested our prototype system taking different users who are not aware of database concepts and also the users who are aware of database concepts.

<table>
<thead>
<tr>
<th>User Category</th>
<th>Correct</th>
<th>Incorrect</th>
<th>Partial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non Database Users</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Database Users</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

System Performance according to User Group

Non Database Users (nondb) and Database Users (db)
Conclusion

- The proposed work Natural-English Language Interface to Database is successfully demonstrated the integration of various techniques that can enhance the accuracy of the (Natural to Formal language) translation process.

- It also tries to improve user interface so that user can enter the query in free form without any restriction of template or grammar.

- The algorithm can be applied to different domains provided that it is supported with appropriate external resources.

- Though, this work demonstrates how different NLP techniques and methodologies can be used in a specific domain using domain knowledge and other external resources.

- It also shows a method to assist user in database interaction by suggesting/providing the previously used queries from the knowledge base.
Future Work

• To accept queries in vernacular languages. Any natural language such as English can have several dialects whose word does not match with standard dictionary words. To include support for these words further processing is required along with appropriate external resources.

• To include more fuzzy questions using fuzzy terms like bad, very bad, not good, not very bad, not very good, etc. Here the fuzzy terms needs to be scale. For example, 1-10 measurement (1-very bad and 10 – very good).

• To include question based on prediction in case of Information Retrieval system. For example, user can ask question like: “when the student puja will complete the final year of her studies?”, “what will happen if a student fail”, etc.

• To support multimedia data such as image, sound and graphics can be attempted.
Future Work (Contd...)

• To include anaphora resolution concept. The Anaphora denotes the act of referring (in broad sense). That is, an expression refers to another contextual entity. For example, in sentence “Akash helped Ram, he was happy”, here, the term “he” refers to the “Akash”. It is a challenge to process anaphora (pronoun) since reference identification may be difficult in certain sequence of text. It requires understanding of how discourse is constructed.

• To include computational phonology and text-to-speech. The phonology refers to the study of sound pertaining to the system of language. The phonology component may allow user to interact with database using speech rather than through text input.

• To include discourse structure. The system can include discourse analysis component to deal with discourse structure defined in terms of coherent sequences of sentences, propositions and speech.
Publications by Candidate

**International Journals**

- Ms. Amisha H. Shingala and Dr. Paresh V. Virparia, published Research paper on *Enhancing the relevance of Information Retrieval by Querying the Database in National Form*, Published in IEEE Xplore Journal, ISSN no: 978-81-909376-6-5 ©2013IEEE, June 2013. This paper is also presented in International Conference on Intelligent Systems and Signal Processing - ISSP-2013, GCET, Vallabh Vidyanagar.


- Ms. Amisha H. Shingala, Ms. Rinku Chavda and Dr. Paresh V. Virparia, Research paper on *Any2MySQL- Effective tool for data conversion* published in International journal of computational intelligence & communication technology, ISSN no: 2278-6732 Volume 1 issue 1, June 2012 published by DIT- IJCICT, Noida.


- Ms. Amisha H. Shingala and Dr. Paresh V. Virparia, Research paper on *Survey of Natural Language Interface* published in International Journal of Information and Computing Technology, Research@ ICT, ISSN no: 0976-5999 , volume 1, Issue 2, May 2011 published by Institute of Science and Technology for Advanced Studies and Research (ISTAR), Vallabh Vidyanagar.
Publications by Candidate

National Journals

• Ms. Amisha H. Shingala and Dr. Paresh V. Virparia, Research paper on *Intelligent Natural Language Processor*, (under press) in National Journal of Systems and Information Technology (NJSIT), ISSN:0974-3308.


Conference

• Ms. Amisha H. Shingala and Dr. Paresh V. Virparia, presented and published Research paper on *Design and Development of Natural Language Query Interface for Relational Databases*, symposium on Innovations in Science- SPU Research Scholar meet, Jan 2012 at Sardar Patel University.
Bibliography
Thank you!