A Web Based Tool for Accessing Distributed Relational Databases through Multilingual Fuzzy Interface

Saroj Kaushik
saroj@cse.iitd.ernet.in

Department of Computer Science & Engineering,
Indian Institute of Technology Delhi, New Delhi- 110016
INDIA

Shivendra Prasad Tiwari
shivendratiwari81@yahoo.co.in

Abstract

Most of the relational database systems use SQL, which has strict syntax and semantics defined precisely to retrieve data. Sometimes, user may not be aware of SQL syntax and concepts and wants to use databases without any technical effort. In recent years, an increasing number of people have begun to realize the need for a technology to reach beyond the barriers of SQL. Some of papers have given solutions regarding this problem in 1999 [1, 3]. We have developed a fuzzy query system for multiple human languages. The implementation is an interface between the user on the web and the databases spread over distributed database servers. It gives the flexibility to query the database in any natural language. It provides a facility to make a fuzzy query and it also corrects wrongly spelled words in the query. If the query is in natural language, the system parses it and converts it into SQL after resolving various tables, attributes and fuzzy terms used in the query. The system also accepts SQL query straightway. The interface is generic enough to work on all Relational Databases and is a generalized tool which can be customized for any organization and any natural language. Thus its use has reduced a need of trained professionals in organizations.

Keywords: Artificial Intelligence, Multilingual Fuzzy Language Query (MFLQ), Fuzzy Natural Language Query (FNLQ), Relational databases, fuzzy interface, fuzzy SQL interface, Super User, Remote access, Customized GUI.

1. Introduction

We have used the phrase "Multilingual Systems" to refer to computer programs, which permit user interaction with the computer in one or more languages, where the language can be selected dynamically, either at the time of invocation of the system or subsequently during its execution. Typically, a Multilingual System will permit users to interact with computers in their own mother tongue. Such a system will have far reaching consequences in our country (India) where English is not spoken or understood by the majority of the people living in areas away from urban environments. Multilingual Systems are not only feasible at the level of an application program but can also present a truly localized environment for a user desiring to interact with computers in regional languages. The phenomenal growth of the Internet has created the need for inter nationalization of the web where, using the right software tools, one can present information in the form in which it would be received best. However, technical challenges faced in implementing such Multilingual Systems and the lack of standards has retarded the development, especially in respect of the languages of India.

Another important couch used is “Fuzzy Interface” to refer to an interface which can interpret the vague instructions from the user. Zadeh introduced Fuzzy set in 1965 [zadeh] to represent data and information processing, non-statistical uncertainty. In this paper, fuzzy logic is used to fuzzify the values of attributes of any entity in the system, which may cause an indistinct query. For example, in a result of an examination all the students scored between 90% and 95%, one can easily say that the student scoring 95% is “excellent”, and the student scoring 90% is relatively poor among all the appeared students whereas if the score range is between 40% to 99%; then the student securing between 90% to 95% are “excellent”. Human is intelligent and can take decisions according to the data and situation. Nevertheless, machine works on Boolean theory, therefore vague instructions can not be handled through simple if -- then …else logic. To solve this ambiguity, we have used fuzzification methods which are able to handle the blurred queries at runtime.

SQL has been responsible for the success of the relational database technology. However, SQL has a major drawback. It has a particular syntax and hence users need to learn the language to access databases. There is a need to bridge this gap between human and machines by introducing a natural language interface between the user and SQL engine. Lot of work has been done in past to build such interfaces [1, 3].
The proposed system is generic and significantly more advanced than most of the available systems, since we are able to handle the “fuzziness” that is inherently embedded into natural languages. It provides facility to configure the system for any language. Currently the system has been tested for three languages i.e. English, Hindi, and Telugu (See Appendix). One can query distributed relational databases in any language and queries can be fuzzy in nature. For example, one can ask system to “get rich people of Delhi”, “find out very bright student of college” etc. It converts input query (ordinary or fuzzy natural language query) to equivalent SQL statements that are fired on the databases and RDBMS returns the retrieved result set as output.

2. Overall Design of the Proposed Tool

The proposed system has been decomposed in three major functional layers. The User Interface is the top layer which user interacts most of the time with. It consists of various graphs, forms, lists, tables and command buttons to offer a rich look and feel. Language selection, database administration, authentication management, and querying are the major services given by the user interface.

Fig. 1: Layered Functional Decomposition

The application engine is the second and the most important layer of the system. It consists of the Query Processing, Fuzzification Engine, Query Formulation, and Query Execution modules, which are responsible to produce the result for the user’s vague query. The third layer is the Database Server module, which is not necessarily in a single physical location. Database Server is supposed to be distributed.

Fig.2 explains how the proposed system MFLQ works. MFLQ consists of 8 major steps to answer a vague query. The figure consists of Language processing module, GUI module, Query formulation and execution modules. Each module is described in the following sub sections.

The language-processing component takes text from keyboard and extracts the necessary information for formulating a database query. The Query Formulator takes this information and generates a database query using the Semantic Graph and Fuzzy Function Library. The SQL query is then fired on the database and answers are returned and displayed to the user.

2.1. A Web-Based Multilingual Fuzzy Query Interface (GUI)

Fuzzy Query interface is a user friendly GUI which provides facility to configure system for different languages. The first page of GUI contains four options viz., “General Query User”, “Domain Sub-administrator”, “Domain Administrator” and “Super Administrator”. General Query interface can be used for triggering vague queries and other modules are useful for system administration activities. Since the major objective of the proposed system is processing fuzzy queries intelligently, GUI explanation has been shortened.

2.2. Language Translation Module

In this module, the query is interpreted and semantics of the query is captured. The query need not be syntactically correct but should convey intended meaning. If it unable to understand or resolve the query word(s), then it will show a spelling error and offers some words as suggestion to replace or ignore such un-resolvable words using LCS algorithm. It makes use of meta-dictionary stored in the system. One can modify its query if there are some incorrect words in the query. This is a good example of the technique of dynamic programming. This module tags the query words such as attributes, table name, fuzzy word, numbers, condition, operators etc.

The system is not only limited for English language but the concept is for language independence. The system follows a very simple strategy to convert any language into English. The interesting thing is that the system has no restriction of syntax but instead it has a word to word mapping between one language words to corresponding English word as English is the base language of the system. Therefore, the concept is that any administrator can add its own language words and their corresponding English words for the domain of database.
2.3. Conditional Fuzzy Words and Hedge Handling

The system has a fuzzy function library, which is updated offline time to time. Administrator can define fuzzy membership functions; fuzzy variables, hedges (very, almost, extremely etc.) and the system automatically calculate their corresponding conditions. At run time, there is no need to calculate fuzzy condition but it automatically detects the fuzzy attributes and table names and just fetches fuzzy condition from the fuzzy function library. Fuzzy Function definition is explained in detail in section 4.

2.4. Query Formulation Module

Based on tagged query, the system formulates the SQL query. The keywords are selected from the query as attributes. If some attribute comes with some value or connector or operator then this fact is taken into account while formulating a condition. Query formulation is explained in sub section 5.1.

For Example:

**Input:** “I would like to see the excellent students of CSE department who are residing in hostel or age>20”

**Output:**
```
SELECT 'academic'.'name' WHERE ('academic'.'marks'>=95 and 'academic'.'marks' <= 100) AND 'academic'.'department' like 'cse' AND 'personal'.'address' like 'hostel' OR 'academic'.'age'>=20
```

2.5. Query Execution Module

Execution module is an important part of the system. It takes the SQL query from query formulation module and directly fires SQL query to MySql and the result set is returned by MySql which is sent to GUI to make it visualize to the user in the tabular form.

2.6. Direct SQL

The proposed system is not only a tool for multilingual system and natural language queries but it is also able to run direct queries from the same GUI. So query could be in SQL or in natural language, the system returns same results for both the queries if these are semantically same.

2.7. Join Operation in required tables of databases

Joining two or more tables of data is a powerful feature found in the relational model. SQL provides facility of joining tables. The proposed system also helps user to query information available in different tables by using primary key.

3. Database Administration tool

Database administration is a difficult task and it needs technical knowledge of database SQL queries. This system provides Look and Feel GUI where administrator can maintain databases with very few efforts. Super Administrator is able to authenticate some domain administrator, update, or create some new databases. There are three level of administration hierarchy in the admin module of the system.

3.1. Database Administrator Hierarchy in various levels of authentication

The system must be secure system with user access control features. All users will have user names and passwords to log on to the system. The tool provides two types of users operating on the system – System Administrator and Normal Users. System Administrators are Domain Experts and must be able to install new languages, manage semantic graphs for registered databases and also build and manage fuzzy function library. System Administrator must also be able to manage users, manage system logs and handle system configurations. Normal users should be able to choose language and use the query interface.

**Super User:**
Super user can access all the databases in hierarchy and can authenticate / register new users. Super user is a top level authorized person who takes care of the things that which domain administrators have to allow customizing this tool for their organization.

**Domain Administrator:**
Domain Administrator is a chief and full-authenticated person for one particular domain. He can register new sub-administrators called Domain Sub-Administrators, and can manage all the databases related to that domain i.e. creation of new databases, deletion etc. as shown in Fig. 3.

![Database Administrators Hierarchy](image.png)

**Domain Sub-Administrator:**
This is also a powerful person who is authenticated for database administration in some particular domain. However, Sub-administrator cannot authenticate other users to update databases. It means that the user information table cannot be accessed by Domain-Sub administrators.

3.2. Database operations through Look & Feel GUI

This tool has all database management capabilities. One can manage database using powerful GUI through which following operations can be performed:
- Create New Database
- Delete Databases
- Create New Table
- Insert data
- Delete Data
- Update data
- Delete Tables and many other operations

3.1.1 Remote Database Administration

The database can be of distributed nature in our system. The system is able to query distributed databases, i.e. databases not residing on a single machine. The user is able to select the database that he wants to work with. Since MFLQ is a multi-tier architecture oriented web-based system, it has the facility to manage database remotely by sitting on a single desktop where databases are stored at different sites. One can remotely manage database of all these domains if he/she is an authenticated personal and having assigned user id and password.

3.1.2 Secure DB Admin through Password protection

Each database administration facility is properly protected from unauthorized user with user id and password. One can access database from anywhere in Internet if he has assigned password and is authorized.

4. Fuzzy Logic & Membership Function Derivation

4.1. Fuzzy Set Theory

The basis for fuzzy logic is the basis for human communication. Natural language, which is used by ordinary people on a daily basis, has been shaped by thousands of years of human history to be convenient and efficient. Sentences written in ordinary language represent a triumph of efficient communication. We are generally unaware of this because ordinary language is, of course, something we use every day. Since fuzzy logic is built atop the structures of qualitative description used in everyday language, fuzzy logic is easy to use. The truth-values in fuzzy logic are indicated by a value in the range \([0.0, \ldots, 1.0]\), with 0.0 implying absolute falseness and 1.0 indicating absolute truth. For example, in a query “Get me all GOOD students from IIT Delhi”, GOOD is a Fuzzy word, which is not absolute but depends on the environment and situation that can be defined by using appropriate fuzzy membership function.

4.2. Membership function Generation

There are various methods for membership function generation. One of the good methods is automatic membership function generation through Entropy minimization [4]. Using the entropy equations with the estimates, we calculate the entropy for the entire \(x\)'s. The value of \(x\) that yields the minimum entropy is taken to be the threshold value \((X)\) of the two partitions. We call this first threshold and indicate by \(X1\). This threshold value is calculated in the range of \(X_{\text{min}}\) and \(X_{\text{max}}\). If we replace the variables \(X_{\text{min}}\) and \(X_{\text{max}}\) by \(X_{\text{01}}\) and \(X_{\text{02}}\), respectively,
then we can compute $X_{11}$ by $\text{Smin}(X_{01}, X_{02})$. With only one threshold value, there can be two non-overlapping fuzzy terms with rectangular-shaped membership functions:

- $[X_{01}, X_{11}]$: NG (negative)
- $[X_{11}, X_{02}]$: PO (positive)

To generate the more fuzzy terms we have to make more partitions. We need more levels of thresholds. In Fig. 4, there are seven partitions. We label them, from the left, NB, NM, NS, ZE, PS, PM, and PB.

To generate the "universal" seven fuzzy terms, we always go down to the third threshold level. However, for well-separated samples, three levels of threshold calculation may cause an over-partition of the sample space. With our algorithmic generation of membership functions, it is possible for two thresholds to share the same value. In this case, samples are well classified with the primary and secondary thresholds alone and further clustering is redundant for separation purposes. However, this situation does not cause a problem, as illustrated.

4.3. MF derivation in our System

In MFLQ a very simple and user friendly approach is used for fuzzy membership function generation. There is a simple interface which asks for the database name, table name, attribute name, linguistic variables, fuzzy hedges, and threshold value (Alpha-cut). Fuzzy Membership function to be used for each fuzzy variable and then calculates the membership value for each variable in selected domain. According to fuzzy membership value, fuzzy condition is generated and it is stored in fuzzy function library corresponding to fuzzy variables. The advantage of this approach is its speed i.e. the system does not calculate fuzzy condition when a fuzzy variable is detected. It just fetches completely fuzzy information from fuzzy function library at runtime. Fuzzy membership generator toll is offline system.

5. Implementation

One of the key features of the system is that it has to be deployed in a web-based environment, and it should be designed with the Client/Server model of interaction in view. The system must be able to handle concurrent queries. Here we are using PHP scripts along with HTML tags to design GUI and handle connectivity with MySql Server.
5.1.2 Tagging

Every word is recognized with the help of Meta databases. Once the tokenized query is passed to the tagging module, every word is tagged with the information whether it is an attribute of some table, table of selected database, operator, connector, fuzzy words, hedges or a value. Every word is tagged with all the relevant information required for SQL formation. The tagged query is used by SQL formulation module as shown in Fig. 4. For example:


**Output:** token[0][fuzzy] = "excellent"; token[1][table] = "students"; token[2][value] = "CSE"; token[3][attribute] = "department"; ... token[4][attribute] = "residing"; token[4][value] = "hostel"; token[5][connector] = "or"; token[6][attribute] = "age"; token[7][where] = ">"; token[8][value] = "20"

![Diagram](image)

Fig. 4: Tagging & Query Formulation

5.1.3 Fuzzy word related Condition Handling

Once the query is tagged, the fuzzy words are removed with their corresponding fuzzy conditions that are stored offline in the fuzzy function library. In fact, there is a fuzzy and hedge Meta table which has records of all the fuzzy words and hedges of the linguistic variable.

5.1.4 SQL Formulation

The SQL query is formulated with the help of a Semantic Graph of the database. It describes the objects and relationships between objects modeled in the database. The Semantic Graph consists of three elements: nodes, attributes, and links. The nodes are the objects in the world, the attributes are the features of the nodes, and the links describe how the nodes are related to each other. The following operations are made for SQL formulation by tagged query.

**Search for attributes i.e. select list**

The ‘select list’ is dependent on what the user wants to return as an answer. Our system handles two types of inputs: ones with explicit select lists and other with implicit select lists. Inputs with explicit select lists are those in which the user explicitly specifies which attributes are to be returned by the system. Examples of such inputs include "what is the name and address of good students of CSE department?". Here name and address are explicitly mentioned. For inputs with implicit select lists, the system uses a heuristic for determining what attributes belong in the select list. For example “Get me excellent students with their performance”. Here performance is the implicit attribute. The system keeps track of all such attributes. Attributes those do not participate in the query constraints are automatically put into the select list. The assumption is that attributes used in the query constraints already have values, and those not in the query constraints need their values retrieved.

**Search for tables**

Search for tables is a step where the system searches for the entire table name in the selected database, which is needed to specify scope of the attribute names and to join two or more tables.

**Search for connectors**

Connector is a symbol or a word, which is used to connect two or more conditions. If a connector comes among select list then it makes no affect but if it comes among conditions then it is placed as it is between two conditions. There are mainly two connectors i.e. “AND” and “OR”.

**Search for Operators**

Operator is a word or a symbol, which comes between two operands. Here <=, >=, <, >, =, and != are some of the operators which are used to make a condition.

**Search for Values**

If some token of symbol is not found in system’s meta dictionary, then it is considered as a value. Every value has should be associated with some attribute through an operator to formulate a condition. System searches nearest attributes of an operator to make a condition. For example:

**Input:** “I would like to see the excellent students of CSE department who are residing in hostel or age>20”

**Output:** SELECT ‘academic’.‘name’ WHERE (‘academic’.‘marks’>=95 and ‘academic’.‘marks’<=100)
AND `academic`.`department` like 'cse' AND
`personal`.`address` like 'hostel' OR
`academic`.`age`>=20

6. System capabilities & limitations

6.1. Users Query Interface

- Direct SQL Execution
- Natural Language and Fuzzy query Handling
- Spelling Checking & Corrections through LCS
- Execute unstructured queries
- There is no need of syntactically correct sentences but should have correct intended meaning.
- Tabular and well-organized Results

6.2. Database Management Interface

- Super User interface
- Domain admin and Sub-admin Administration interface
- Insertion, deletion, updating, creation of Database through user friendly GUI
- Easy interface to define Membership Function and store definitions in Fuzzy Function Library

6.3. Help and Documentation

- Fuzzy Set theory Help
- How to manage database
- How to develop and define a membership function
- Properly Documented PHP script
- Easy and understood HTML tags

6.4. Limitations

Presently, all the informations such as Semantic Graph, Fuzzy Function Library etc., are fed into the system by a domain expert. We can introduce learning into the system so that the system may learn this information in an evolutionary manner.

7. Conclusion

We have designed, developed and implemented a fuzzy language query tool, based on a semantic model, which takes semantic information from the database and uses a special SQL formulation methodology to formulate SQL statement from a fuzzy natural query. We use keywords to approximate the meaning of the user's input. The keywords are natural language labels of nodes and attributes of the semantic graph, the values contained in the database and the operators. The system uses no formal grammars or world knowledge but the most of the knowledge utilized by the system can be obtained from the databases itself. This system is most suitable for applications where the user input is short and simple, and it is necessary to support domains with a limited ontology. The proposed system is a generalized tool which can be customized for any organization and any natural language. Currently the system has been tested for three languages i.e. English & Hindi and Telugu (examples are shown in appendix). One can query distributed relational databases in any. It has been implemented using PHP scripts along with HTML tags to design GUI and handle connectivity with MySql Server.

Acknowledgement

I would like to acknowledge MHRD (Ministry of Human Resource Development, INDIA) who funded this project under thrust area scheme. I would also like to thank all those who contributed in this project during this project execution phase.

References

2. C. J. Kim An Algorithmic Approach for Fuzzy Inference
7. Fuzzy Sets and Fuzzy Logic: Theory and Applications by George J. Klir and Bo Yuan
8. Fuzzy Logic Toolbox Help MATLAB 7.0

APPENDIX
1. RESULTS (Sample English Queries and corresponding SQL).

- Find those students along with cgpa who have cgpa greater than 9.
  ```sql
  SELECT `academic`.'name', `academic`.'cgpa'
  FROM `personal`, `academic`
  WHERE (`academic`.'cgpa'>9) AND ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- Get me names and cgpa of civil dept students.
  ```sql
  SELECT `academic`.'name', `academic`.'cgpa'
  FROM `personal`, `academic`
  WHERE (`academic`.'department' like '%civil%')
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- List the name and address of the students who reside in lucknow.
  ```sql
  SELECT `academic`.'name', `personal`.'address'
  FROM `personal`, `academic`
  WHERE (`academic`.'address' like '%lucknow%')
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- What is the address of student whose date of birth is above 1980 and sno is above 8?
  ```sql
  SELECT `academic`.'name', `personal`.'address'
  FROM `personal`, `academic`
  WHERE (`academic`.'dob'>1980) AND ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- List the name and address of the students who are belonging to lucknow.
  ```sql
  SELECT `academic`.'name', `personal`.'address'
  FROM `personal`, `academic`
  WHERE (`academic`.'address' like '%lucknow%')
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- I want to see the good students in cse department along with their address and cgpa.
  ```sql
  SELECT `academic`.'name', `academic`.'cgpa',
  `personal`.'address' FROM `personal`, `academic`
  WHERE (`academic`.'department' like '%cse%')
  and ( `personal`.'sno' = `academic`.'sno' )
  and ( `academic`.'marks'>=67 and `academic`.'marks'<=78)
- Get me excellent students who reside in Delhi
  ```sql
  SELECT `academic`.'name'
  FROM `personal`, `academic`
  WHERE (`personal`.'address' like '%delhi%')
  and ( `academic`.'marks'>90)
  and (1)
```

2. RESULTS (Sample Hindi queries and Corresponding SQL)

- कृपया विवादियों का नाम और पता दियाओ।
  ```sql
  SELECT `academic`.'name' , `personal`.'address'
  FROM `personal`, `academic`
  WHERE (`academic`.'name' like '%please%')
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- मैं ऐसे विवादियों के नाम और पते देना चाहता हूँ।
  ```sql
  SELECT `academic`.'name' , `academic`.'marks'
  FROM `personal`, `academic`
  WHERE (1)
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- कृपया ऐसे विवादियों का नाम और पते देना चाहता हूँ।
  ```sql
  SELECT `academic`.'name' , `academic`.'cgpa'
  FROM `personal`, `academic`
  WHERE ( `academic`.'marks'>90 )
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- ऐसे विवादियों के नाम और पते देना चाहता हूँ।
  ```sql
  SELECT `academic`.'name' , `academic`.'marks'
  FROM `personal`, `academic`
  WHERE (`academic`.'marks'>80 and `academic`.'marks'<90)
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- मैं ऐसे विवादियों का नाम और पते देना चाहता हूँ।
  ```sql
  SELECT `academic`.'name' , `personal`.'father'
  FROM `personal`, `academic`
  WHERE (1)
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
- मैं ऐसे विवादियों का नाम और पते देना चाहता हूँ।
  ```sql
  SELECT `academic`.'name' , `personal`.'address'
  FROM `personal`, `academic`
  WHERE (1)
  and ( `personal`.'sno' = `academic`.'sno' )
  and (1)
like '%%cse%%' ) and ( `personal`.'sno' = `academic`.'sno' ) and (1)

- ᓯ ᔑ ᛘ GCBO ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᒔ ᐃe ᐃn ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃente ᕰ ᔑ ᕰ ᔑ ᛘ内科 ᐃentes (Sample Telugu queries and Corresponding SQL)

- 3. RESULTS (Sample Telugu queries and Corresponding SQL)

  SELECT `academic`.'name' , `academic`.'cgpa' FROM `personal`,`academic` WHERE ( `academic`.'cgpa'>9 ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'cgpa'>9 ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' , `academic`.'cgpa' FROM `personal`,`academic` WHERE ( `academic`.'cgpa'>9.2 ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'cgpa'>9.2 ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'department' like 'cse' ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' , `personal`.'address' FROM `personal`,`academic` WHERE ( `personal`.'address' like 'meerut') and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'cgpa'>8 ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT * FROM `personal`,`academic` WHERE ( `academic`.'name' like 'raman')

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'name' like 'raman')

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'name' like 'raman')

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'name' like 'raman')

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'name' like 'raman')

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'name' like 'raman')

  SELECT `academic`.'name' , `academic`.'cgpa' FROM `personal`,`academic` WHERE ( `academic`.'cgpa'>9 ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' , `academic`.'cgpa' FROM `personal`,`academic` WHERE ( `academic`.'cgpa'>9 ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'department' like 'cse' ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'department' like 'cse' ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)

  SELECT `academic`.'name' FROM `personal`,`academic` WHERE ( `academic`.'department' like 'cse' ) and ( `personal`.'sno' = 'academic'.'sno' ) and (1)