CHAPTER 4
FUZZY DATABASE QUERYING

4.1 Introduction

Database querying process by the two valued realization of Boolean algebra is not adequate and offers solution based on the fuzzy logic because the fuzzy logic is an approach to computing based on "degrees of truth" rather than the usual "true or false" logic. The fuzzy logic deals with reasoning that is approximate rather than precise to solve problems in a way that more resembles human logic. This area of research is not new one but there are still many possibilities for the improvement of existing approaches and for creating new approaches. Fuzzy queries have emerged in the last 25 years to deal with the necessity to soften the two-valued Boolean logic in relational databases. A fuzzy query system is an interface to users to get information from database using (quasi) natural language sentences. Many fuzzy query implementations have been proposed, resulting in slightly different languages. Although there are some variations according to the particularities of different implementations, the answer to a fuzzy query sentence is generally a list of records, ranked by the degree of matching [9].

Organizations work with very large data collections mainly stored in relational databases. Linguistic expressions are interesting for data extraction, analysis, dissemination and decision making. In case of statistical data they are often collected with some errors and vagueness. If two-valued logic in selection process is used then the small error in data values or cases when user can not unambiguously define the criterion by crisps boundaries may involve some inadequate selected or non-selected data. To avoid this, the wanted scenario for the user is to determine what kind of data he wants to select by linguistic expressions and degrees of truth. These linguistic
expressions have the logical meaning for user and describe a data selection process in the natural language. The conversion from human language into computer language and all mathematical operations are implemented behind the user interface and user obtains the final solution of a task. Issues and perspectives of fuzzy querying can be found in [33]. The fuzzy query approach based on the fuzzy Generalized Logical Condition (GLC) is presented and analyzed here. The GLC formula capable to capture linguistic expressions into the WHERE part of the SQL was initially created and presented in [29]. The implementation of the GLC for statistical databases is explained in [30]. Although fuzzy approach has significant advantages beyond the two-valued approach, there are situations when some inconsistency may occur. It is possible to avoid this problem by several ways [42, 46].

4.2 SQL and its limitation

Users search databases in order to obtain data needed for analysis, decision making or to satisfy their curiosity. The SQL is a standard query language for relational databases. The simply SQL query is as follows:

```
select attribute_1,…,attribute_n
from T
where attribute_p > P and attribute_r < R
```

The result of the query is shown in graphical mode in figure 1. Values P and R delimit the space of interesting data. Small squares in the graph show database records. In the graph it is obviously shown that three records are very close to meet the query criterion. These records could be potential customers and direct marketing could attract them or municipalities which almost meet the criterion for some financial support for example.
The SQL uses the crisp logic in querying process that causes crisp selection. It means that the record would have not been selected even if it is extremely close to the intent of the query criterion. As the criterion becomes more and more complex, the set of records selected by the WHERE statement becomes more and more crisp. If the classical SQL is used for solving this problem, the SQL relaxation would have to be used in the following way:

```sql
select attribute_1,…,attribute_n
from T
where attribute_p > P-p and attribute_r < R+r
```

where p and r are used to expand the initial query criteria to select records that almost meet the query criteria. This approach has two disadvantages [15]. First, the meaning of the initial query is diluted in order to capture adjacent records. The meaning of a query: “where attribute_p is more than P” is changed and adjacent records satisfy a query in the same way as initial ones. More precisely, the difference between original and adjacent data (caught records along the “edge” of interesting space) does not exist. Secondly problem rises from the question: what about records that are very close to satisfy the new expanded query and it is useful to make another expanding of a query.
In this way more data from the database is selected, but the user has lost the accuracy of his query. Many applications have created uncountable accesses to wide variety of data. The data and the classical access to data are simply not enough in many cases. In cases when the user can not unambiguously separate interesting data from not interesting by sharp boundaries or when the user wants to obtain data that is very close to meet the query criterion and to know the index of distance to full query satisfaction, it is necessary to adapt the SQL to these requirements. The SQL was initially developed in [12]. Since then the SQL has been used in many relational databases and information systems for data selection. The use of SQL may be regarded as one of the mayor reasons for the success of relational databases in the commercial world [50]. In this research the core of SQL remains intact and the extension is done to improve the selection process. Adding some flexibility to the SQL meets above mentioned requirements and increases effectiveness and comprehensibility of the whole querying process.

4.3 Queries based on fuzzy logic

The GLC for the WHERE part of the SQL based on linguistic expressions has been created in [29]. For the further reading it is important to define the Query Compatibility Index (QCI). The QCI is used to indicate how the selected record satisfies a query criterion. The QCI has values from the [0, 1] interval with the following meaning: 0 - record does not satisfy a query, 1 - record fully satisfies the query, interval (0, 1) - record partially satisfies a query with the distance to the full query satisfaction.

The GLC has the following structure:

\[ \text{Where } \bigotimes_{i=1}^{n} \left( a_i \circ L_{xi} \right) \]
Here n denotes the number of attributes with fuzzy constraints in a WHERE clause of a query,

$$\bigotimes = \begin{cases} 
\text{and} \\
\text{or}
\end{cases}$$

where and and or are fuzzy logical operators, and

$$a_i \circ L_{\psi} = \begin{cases} 
a_i > L_{di} , & a_i \text{ is Big} \\
a_i < L_{gi} , & a_i \text{ is Small} \\
a_i > L_{di} \text{ and } a_i < L_{gi} & a_i \text{ is About}
\end{cases}$$

Where \( a_i \) is a database attribute, \( L_{di} \) is the lower bound and \( L_{gi} \) is upper bound of a linguistic expression described by fuzzy sets shown in figure 2. Conditions in queries contain these comparison operators: >, <, =, \( \neq \) and between when numerical attributes are used. These crisp logical comparison operators are adapted for fuzzy queries in the following way: operator > (greater than) was improved with fuzzy set “Big value” (fig. 4.2a), operator < (less than) was improved with fuzzy set “Small value” (fig. 4.2b) and operator = (equal) was improved with fuzzy set “About value” (fig. 4.2c). Operator \( \neq \) is the negation of the operator = so this operator is not further analysed. Analogous statement is valid for the operator between because it is similar to the operator = from the fuzzy point of view. Other types of fuzzy sets could be added in the future to catch other linguistic expressions.
In the fig. 4.2 two often used fuzzy sets are shown and implemented in the fuzzy query approach. It is possible to add more types of fuzzy sets for each of linguistic expressions. A useful theoretical overview about fuzzy sets and fuzzy operators can be found in [34] and a useful practical overview can be found in [15].

If a query in the WHERE clause contains fuzzy as well as classical constraints, these classical constraints could be easy added to the WHERE clause (3) in the following way:

\[ \text{where } \bigwedge_{i=1}^{n}(a_i \circ L_{ai}) \bigwedge_{i} (\text{attribute} \_ \text{mLIKE"*String"}) [\text{and/or}] \ldots \]

Where LIKE is the SQL comparison operator and String is an arbitrary string variable. The querying process consists of the two steps. In the first step lower and/or upper bounds of linguistic expressions (fuzzy sets) are used as parameters for database queries. It means that all records that have QCI greater than zero are selected only. In the second step the chosen analytical form of the fuzzy set is used to calculate the
membership degree of each selected record to appropriate fuzzy set. Finally, appropriate t-norms or t-conorms are used to calculate QCI values for all retrieved records.

In case when two fuzzy constraints are in a WHERE clause all t-norms or t-conorms can be used as an aggregation function. However, for fuzzy SQL it is required to use more than two attributes in the WHERE clause of a query. From the associative rule of t-norms and t-conorms, the following functions can be easy aggregated for cases when more than two attributes are used [46]:

**t-norms for and operator:**

- min:

  \[
  QCI = \min(\mu_i(a_i)) \, , \, i=1,2,\ldots,n
  \]

- Product:

  \[
  QCI = \prod_{i=1}^{n} (\mu_i(a_i)) ,
  \]

- Bounded Difference(BD):

  \[
  QCI = \max(0, \sum_{i=1}^{n} \mu_i(a_i) - n+1)
  \]

**t-conorm for or operator**

- max:

  \[
  QCI = \max(\mu_i(a_i)) \, , \, i=1,2,\ldots,n
  \]

- Bounded Sum(BS):

  \[
  QCI = \min(0, \sum_{i=1}^{n} \mu_i(a_i))
  \]

Where \( \mu_i(a_i) \) denotes the membership degree of the attribute \( a_i \) to the i-th fuzzy set.
These functions are implemented into the query language and cover usual expected cases. Min t-norm takes into account the lowest value of membership degrees to fuzzy sets only; product t-norm takes into account all membership degrees and balances the query truth membership value across each of the membership degrees. In case when the correlation is unknown or does not exist, these two norms are used. In other cases, the BD t-norm is used. From above mentioned t-norms, only the BD t-norm satisfies the non-contradiction and excluded middle laws. The similar discussion is valid for t-conorms.

To implement this a fuzzy query interpreter, which transforms fuzzy queries to the classical SQL structure, was developed. In this way, queries based on linguistic expressions on client side are supported and are accessing relational databases in the same way as with the classical SQL. Figure 3 shows this model. The first step of querying process (to select records that have QCI>0) is situated in parts 1, 2 and 3. Lower and/or upper limits of linguistic expressions are calculated and converted into SQL query in the part 1. Thus created SQL query selects data from database (part 2) and saves it into the temporary table (part 3). The second step uses data from part 3 for further calculations. Firstly, the chosen analytical form of the fuzzy set (from part 1) is used to determine the membership degree of each selected record to appropriate fuzzy set. Secondly, t-norm and/or t-conorm function also defined in the part 1 are used to calculate the QCI value. These calculations are situated in the part 4. Finally the result of fuzzy query is displayed in part 5.
4.3.1 Principle of Functioning

The principle of functioning is as follows:

Begin

Cut a DDL instruction in several lines containing each one an attribute.

Study every DDL line:

If it contains a classic attribute then

Copy this line in a first file (result1.sql)

Else

Make a very specific treatment to every fuzzy attribute. This treatment divides in two under treatments:

a) Treatment in DB: translate the command that concerns the DB and copy it in the file (result1.sql)

b) Treatment in FMB: provide a script writes in the file (result2.sql), containing the formation in the FMB concerning the fuzzy attributes and the objects defined on them.

End
As result of these two treatments, it get two files. A file containing the DDL part of the DB and a second one containing the modification to be done in the FMB. The two treatments in a same file can also be regrouped since they will be executed in the DBMS.

4.3.2 The GLC and the Boolean algebra

The GLC is developed on axioms of fuzzy logic. Fuzzy logic is based on same principle as classical logic, the principle of truth-functionality. Logic is truth functional if the truth value of a compound sentence depends only on the truth values of the constituent atomic sentences [54], not on their meaning or structure [42]. In the two-valued logic the mentioned principle is enough for all axioms. In case of all many-valued logics, including fuzzy logic, this principle is not sufficient and as a consequence these logics are not in the Boolean frame. More precisely, fuzzy logic is a precise many-valued logic where axioms of non-contradiction and excluded middle are not satisfied. It is obvious on following example: “WHERE attribute_p >5 and attribute_p <=5” (contradiction). In classical query it is obvious that criterion retrieves no record from database. In case of fuzzy query when the “WHERE attribute_p is Big and attribute_p is not Big” criterion is used it could be expected that no record is retrieved because of non-contradiction axiom existence but min t-norm (5) retrieves some records with QCI\(\leq 0.5\). This is the consequence of not satisfied non-contradiction axiom.

First way how to use the gradation in mathematics is to leave these axioms as non-adequate and accept the principle of truth functionality with all consequences. When the first way is chosen, it is possible to avoid this problem by selecting adequate t-norm or t-conorm function for each query. A new t-norm \(T^\wedge\) and a new t-conorm \(C^\wedge\),
which depend on a parameter $r$ in $[-1, 1]$ or the correlation between the truth values of the operands are explained in [46].

The second way is to go to the source of Boolean algebra and find the principle for gradation to be in the frame of the Boolean algebra. New approach to treating fuzziness or gradation in logic is based on the Interpolative Realization of Boolean Algebra (IBA) [42]. The IBA ensures that the whole selection process will be in the frame of Boolean algebra and avoids theoretically possible situations when inappropriate functions are chosen.

### 4.4 Proposed Model

The comparison between SQL and fuzzy query performance cannot be unambiguously determined because of different nature of these two querying concepts. SQL has faster performance because of non existence of additional calculation of lower bounds of fuzzy sets, membership degrees and QCIs for selected records as in the fuzzy counterpart. On the other side fuzzy query provides more information than classical one and gives the user more freedom for creating of a selection task. In cases when user does not have ambiguities and uncertainties concerning data, the SQL solves all user needs and requirement for fuzzy queries does not exist.

The WHERE clause of a query usually contains comparison operators with crisp values such as:

```sql
where unemployment_rate > 5 and population_density > 200 and migration_level < 1.
```

The meaning and purpose of the previous query is understandable for the user who creates and uses it. In case of reuse of the same query in e.g. different time period or by different user the meaning and purpose of query is not obviously clear at the first glance. Fuzzy query contains logical conditions defined by linguistic expressions
whereby the query becomes easy understandable and applicable. The meaning of a query remains the same, only the parameters and shapes of fuzzy sets are changeable to allow the query adaptation to new situations or requirements.

A disseminated number without meaning (without explanatory metadata) does not tell very much to user. A number in a WHERE clause does not explain the purpose of a query in many situations. As metadata are used to explain the meaning of the figures, the linguistic expressions are used to explain the meaning of a query. Fuzzy query enables also simplified and easy to use distance measurement of records around selected value. For this purposes normalised (fuzzy set is normalised if \( \exists x, \mu_A(x) = 1 \) ) and symmetric fuzzy sets shown in figure 4 are used. The criterion “WHERE attribute is approximately 5” can be described with “about” fuzzy set. The query retrieves all records that have attribute value equal to 5 and all records that have QCI>0 and QCI value is a distance value of each selected record.

![Fig. 4.4 Approximate or About Value Fuzzy Sets](image)

**Fig. 4.4 Approximate or About Value Fuzzy Sets**

Case-based Reasoning (CBR) is very interesting field for the research and applications. Using relational databases for storing and representing cases in CBR reduces effort needed to develop a CBR system. The approximate reasoning is not possible with the SQL. A proposal where the case retrieval is implemented by using a fuzzy version of SQL is presented in [40].

Fuzzy query implementation FSQIL in [25] deals with fuzzy databases. The FSQIL’s syntax permits to use fuzzy comparisons with fuzzy and non-fuzzy database attributes. The FSQIL supports possibility and necessity comparators and supports fuzzy hedges,
thresholds, distributions, quantifiers in WHERE and HAVING clause of a query. This is a complex solution for including fuzzy concepts into database queries. However, many statistical databases are developed in Relational Database Management Systems (RDBMS) and this trend dominates in development of new statistical information systems and databases. The fuzzy SQL for traditional relational databases has the significant perspective for usage in many areas including statistics. The approach presented here is a way how to create an easy to use fuzzy querying tool for end users. Further improvement of the proposed fuzzy query approach could be in avoiding the inconsistency of fuzzy logic caused by the principle of truth-functionality.

The goal of fuzzy querying is not to obtain more data but better data for users. Data selection depends on degree of satisfaction that implies that a fuzzy query retrieves some data in cases when traditional SQL retrieves no data. Additional information for users could be as follows: although no records fully satisfy the intent of a query, there are some records that are very close to meet a query criterion and it may be interesting to analyse these records.

4.5 Implementation of Proposed Model

For the proposed model prototype, a database of a school is used to implement the concept of flexible querying. There are three tables namely Teacher, Student and Class. On certain fields (age, salary etc.) of Teacher and Student table, fuzzy conditions are applied. Fuzzy membership functions used for applying fuzzy conditions are explained in the following section.

4.5.1 Membership Functions Used

Fuzzy membership functions to express fuzziness in the query are used. There are a series of membership functions that could be classified into two groups: those made up of straight lines, or “linear,” and Gaussian forms, or “curved.” In this work linear
Trapezoid function are used. For implementing fuzzy conditions following membership functions have been used in this research work which describes small values, about values and big values respectively.

1. **Small values** mean initial range of values of any attribute used in our database. Like for age field of children of our table Student, range of toddlers (age between 0-6) is fuzzified with the help of membership function shown in Fig. 4.5. Likewise the low values of salary field of Teacher table of our database are fuzzified with the same membership function.

2. **About values** mean values in the middle range of values of any attribute used in our database. Like for age field of children of our table Student, range of young and teenagers (age between 3-14 and 11-20 respectively) is fuzzified with the help of membership function shown in Fig. 4.6. Likewise the medium values of salary field of Teacher table of our database are fuzzified with the same membership function.

![Fig. 4.5 Membership Function for Small Values](image_url)
Fig. 4.6 Membership Function for About Value

The function shown in fig. 4.6 is Trapezoidal fuzzy set. It is defined by its lower limit $a$, its upper limit $d$, and the lower and upper limits of its nucleus or kernel, $b$ and $c$, respectively:

$$T(x) = \begin{cases} 
0 & \text{if } (x \leq a) \text{ or } (x \geq d) \\
(x-a)/(b-a) & \text{if } x \in (a, b) \\
1 & \text{if } x \in [b, c] \\
(d-x)/(d-c) & \text{if } x \in (c, d) 
\end{cases}$$

Fig. 4.7 Membership Function for Big Value

3. **Big values** mean values in the ending range of values of any attribute used in our database. Like for age field of children of our table Student, range of senior students (age between 16-20 and onwards) is fuzzified with the help of membership function shown in fig. 4.7. Likewise the high values of salary field
of Teacher table of our database are fuzzified with the same membership
function.

4.6 Developing Environment

Following development environment is being used.

Integrated Development Environment/Framework - .net 4.0

Language – VC#.net (Visual Studio 2010 Express)

Application – Windows Application

Back End – SQL Server 2005

4.6.1 Methods

Following methods are being used in said prototype.

1. FuzzyForm()
   
   It is a constructor to initialize the form and components on it.

2. FuzzyForm_Load(object sender, EventArgs e)
   
   This is a form Load method of windows application. This will be the first
   one to get executed.

3. cmbTableNames_SelectedIndexChanged(object sender, EventArgs e)
   
   It is used to choose different tables from the drop down. This method will
   populate the respective fields and where condition drop downs

4. btnGenerateSQLFromFuzzy_Click(object sender, EventArgs e)
   
   After the selections have been done, user will click on the button
   GenerateSQLFromFuzzy. This method will implement the algorithm to
   determine fuzzy values and display them in the grid.

5. StringBuilder ApplyFuzzy(StringBuilder FuzzySQL)
This method will take SQL and then apply fuzzy algorithm and return the changed SQL as a StringBuilder value.

6. cmbWhereField_SelectedIndexChanged(object sender, EventArgs e)

This is the method to populate the fuzzy titles on selection of where fields.

7. cmbFuzzyValues_SelectedIndexChanged(object sender, EventArgs e)

On the fuzzy titles changed, this method will populate the respective range values.

8. btnShowSQLDetails_Click(object sender, EventArgs e)

This method will show the generated SQL details in a message box.

Table details, coding and screen shots are shown in Appendix 1, Appendix 2 and Appendix 3 respectively.