International Journal of Information Technology and Knowledge Management
Vol. 5 No. 1 (January-June, 2019)

Received: 17th January 2019 *Revised: 26th February 2019* *Accepted: 31ststst May 2019*

Corresponding Reviewer: Dilek Onkal

FSQL: A Systematic Framework for Habitual and Fuzzy SQL Commands

Khalid Al-Radaideh** **and Mohammed Akour***

**Yarmouk University, P. O. Box 566, Irbid 21163, Jordan*
***Qassim University, P. O. Box 6633, Buridah, Saudi Arabia 514521*

Authors emails
mohammed.akour@yu.edu.jo
Khm.radaideh@qu.edu.sa

ABSTRACT

A great number of database application systems have been developed using relational database systems and written in SQL language which is an international standard. These resources cannot be overlooked in developing fuzzy database applications. Accordingly, practical utilization requires relational model based fuzzy database systems and SQL based fuzzy database languages.Fuzzy database has been introduced to deal with uncertain or incomplete information in many applications. In this paper, we proposed a new method to draw an association diagram for a university subsystem (Teachers, Students, Courses), that is needed when using Structure Query Language (SQL) or Fuzzy Structure Query Language (FSQL) queries. The representation of fuzzy data has been proposed is entirely based on pointers and queue data structures. FSQL model has been designed for treating the term 'fuzzy' as an important key not allowing the habitual SQL but permitting the crisp commands in fuzzy context and the fuzzy commands together. We write the needed lexical analyzer for the fuzzy commands, designs the syntax analysis (using finite automata) and write the parser for each commands. This automaton is decomposed into many others like fuzzy tables, fuzzy expression and fuzzy condition. The implementation shows the validity of the proposed algorithms.

Keywords: SQL commands,fuzzy queries, fuzzy generalized logical condition

INTRODUCTION

Fuzzy logic (FL) is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth- truth values between "completely true" and "completely false". As its name suggests, it is the logic underlying modes of reasoning which are approximate rather than exact (Hrudaya, Tripathy, Das & Khadanga 2009). Fuzzy SQL (FSQL) is an extension of the standard SQL to deal with crisp and fuzzy values in the fuzzy database and it allows us to write flexible conditions in our queries. It supports queries based on linguistic expressions. Fuzzy improving of SQL queries has advantages in cases when the user can not unambiguously separate data he is interested in from data he is not interested in by sharp boundaries or when the user wants to obtain data that are very close to satisfy queries. In other cases, classical SQL fulfills the requirements for data (Hudec2009). Fuzzy databases and the FSQL language have many applications, and the deductive power is very important. For example, in a hospital, one could make queries such as the following: "Give me a list of young patients suffering from hepatitis who were admitted approximately more than 5 weeks ago." In a supermarket, it would be useful to know the answer to a request such as the following: "Give me a listing of the products that have sold very well, but on which we have spent little for publicity." The list of management applications and useful queries that can be done in this way is endless.

However, the applications of FSQL are not limited to management applications.

FSQL can be used for deductive processes in the so-called Fuzzy Deductive Relational Databases

(Blanco, Cubero, Pons, & Vila 2002) &(Galindo, Aranda, Guevara, Caro, & Aguayo 2002) and for data-mining applications (Carrasco, Vila, & Galindo 2002) Several problems related to fuzzy data bases may be considered and addressed, such as representing fuzzy tables in memory, writing syntax fuzzy commands like CREATE, INSERT etc., and corresponding lexical and parser for these fuzzy commands, calculating the number of low, middle, high, between small and very young, …, totals, average, max, min,…. sorting the database according to fuzzy attributes; fuzzy attributes may have a crisp value like 20, 35,… and fuzzy value like young, old,.. , using the proposed fuzzy SQL command to retrieve, insert, delete and update data themselves in the fuzzy database.

In this work, we propose a representation of fuzzy data which is entirely based on pointers (inverted index) and queue data structures. Then we design an FSQL model treating the term 'fuzzy' as an important key not allowing the habitual SQL but permitting the crisp commands in fuzzy context and the fuzzy commands together. We draw the automata for the fuzzy commands and write the corresponding lexical analyzer. This automaton is decomposed into many others like fuzzy tables, fuzzy expression and fuzzy condition.

THE METHODOLOGY

In this section we describe the proposed method to draw an association diagram for a university subsystem (Teachers, Students, Courses) as a case study, that is needed when using SQL or FSQL queries; provide a representation of the fuzzy data in memory; and extension of SQL commands to FSQL like: CREATE, UPDATE, DELETE, INSERT and SELECT; finally we draw automata and write the algorithms for these fuzzy commands.

Drawing an Efficient Association Diagram (Case Study)

Association diagram is a graphical representation of entities and their relationships to each other. As a case study we attempt to draw an efficient association diagram for the University Database system. When the teacher takes a job at the university, the university makes a printed employment document containing a code, the date of the document and the teacher data. The teacher data may be used in different services (stuff service, finance, registration …). So to avoid redundancy and applying normalization rules, this printed document is converted into an electronic document containing the code (CodeEmplDoc) which is the primary key and date of the document and only the teacher code as a foreign key and hence creating a relation between the documents file and the teacher file. If we suppose that the teacher contract is renewed yearly, and since every teacher code appears in at least one employment document (one per year), and every employment document is related to just one teacher, the cardinalities between teachers and employment documents are shown Figure 1.

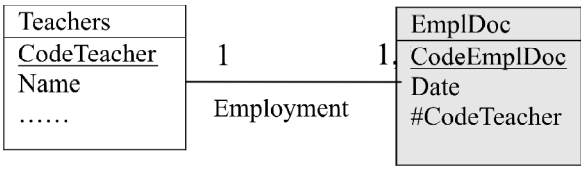

Figure 1: The Relation between Teachers and Employment Document.

All the relations deduced from the documents are assembled to get the following association diagram Figure 2.

Representing Fuzzy Database

The problem is representing the fuzzy data in the memory. Indeed, suppose we have an integer attribute like age, so that one must have to represent " young", "very young", and so on as integers . All values in one field have same type, so since the age 'young' is a string and the age 20 is integer then we use a pointer to the corresponding data. If we have an SQL that can deal with fuzzy and crisp database then we may get precise decisions and inferences and a high value database.

The first step is opening FSQL software. Then the program creates a head pointing to a tail as shown in Figure 3.

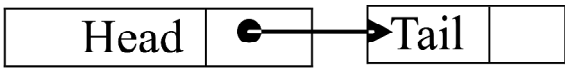

Figure 3: the list after opening FSQL

When typing the following command:

```
FSQL>  FUZZY CREATE  TABLE
STUDENTS
(CodeStudent  crisp  integer;
CodeCollege  crisp  integer;
Name  crisp  varchar (30);
```

The table names are inserted in a queue of a linked list. When typing the string: "fuzzy create table students (", the head students node will be initialized. And then "Head" points to the "Head students" node that points to the "tail" node, and another field in head students points to NULL that is waiting for another creating

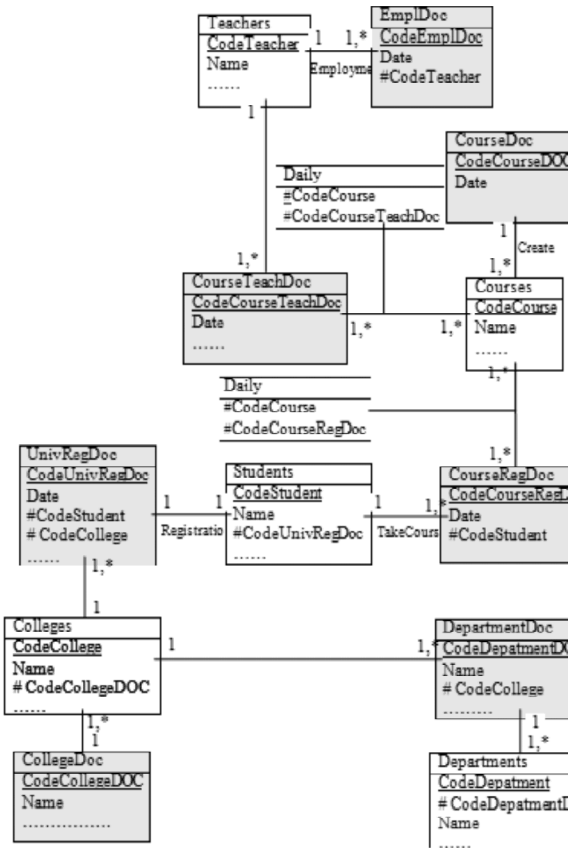

Figure 2: The Association Diagram for the University Database system

command to be entered to point to another table (Figure 4). Tail points to NULL and waits for the first insert command to point to the first record in the fuzzy table.

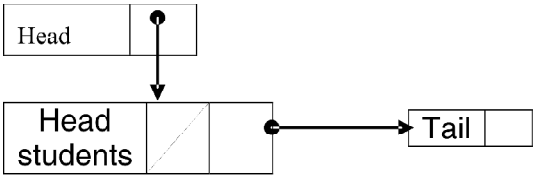

Figure 4: The list after entering "fuzzy create table students

When another fuzzy table is created, we must ensure that there is no other table with the same name in the list. Then a constantcurves names table that contains the curve code, curve name and number of parameters for the fuzzy attributes will be created as shown in Table 1.

Table 1: Constant curves names table

curvecode	*curve name*	*number of parameters*
1	Sigmoidz	3
2	Trapezoid	5
3	Triangle	4
4	Sigmoids	3
5	Crisp	/

After adding the fields of the fuzzy table students, the fields are inserted in a queue after checking that there is no other field with the same name (Figures 5, 6 illustrate the operation of adding the first two fields to the fuzzy table students), and so on.

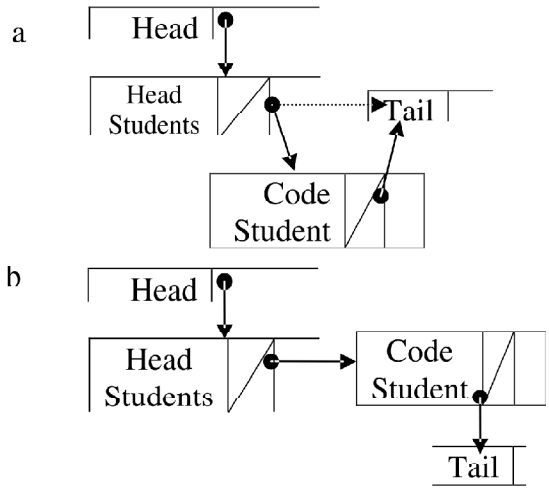

Figure 5: A graphical representation of adding the first field in the fuzzy table Students.

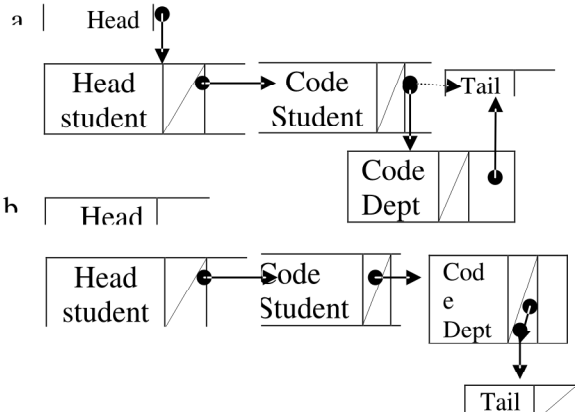

Figure 6: A graphical representation of adding the second field in the fuzzy table student.

Each table node points to other queue nodes that contain the names of fields for that table, all of which are created in the fuzzy create phase (creating the tables to be represented in the memory), that is: enter the name of the first table. Then enter the fields for that table (the first field, second field,…etc).

Each field node has the following specifications:

Field name.

Field type (NULL if Crisp, or a pointer (inverted index) to the linguistic term table for the fuzzy attribute if not crisp Figure 7.

Pointer to the next field in the table or pointer to the tail if there is no field to be added

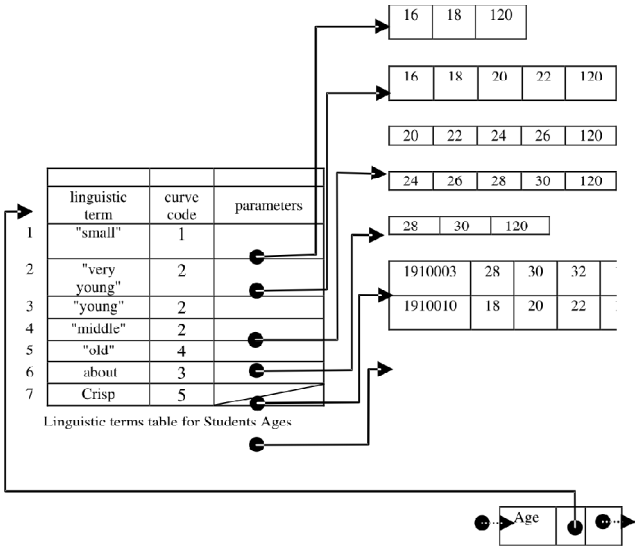

Figure 7: The fuzzy attribute age points to the linguistic term table.

The tail of the field's list points to the list of values to be inserted in the fuzzy insert phase.

When we insert any field value, there is a program to check that we enter an acceptable value (the same type of the field). The last node in the list of values contains EOF mark to tell the user that he/she has reached the last value in the last record in the table.

As shown in Figure 8 the fuzzy field points to a table which contains the following:

1) Linguistic term of the fuzzy field like ('small', 'young', …, crisp) as shown in Figure 7.

2) Curve type which is a pointer (inverted index) to another table which has the following:

a) Curve Code (1, 2, ….,5). A foreign key for the curve type in the linguistic table.

b) Curve name (sigmoidz, trapezoid, triangle, sigmoids, and crisp).

c) Number of parameters: contains the number of parameters to be used in the fuzzy function.

3) Parameters: these are pointers to a list which has the values to be used when evaluating the membership functions.

The complete fuzzy structure is shown in Appendix A.

THE EXPERIMENTAL WORK

In this chapter, to simplify the way of writing the algorithms for lexical and parser of the FSQL commands, we drew the tree structure for the main program and for each method needed for executing the FSQL commands, next we saw the program execution for them.

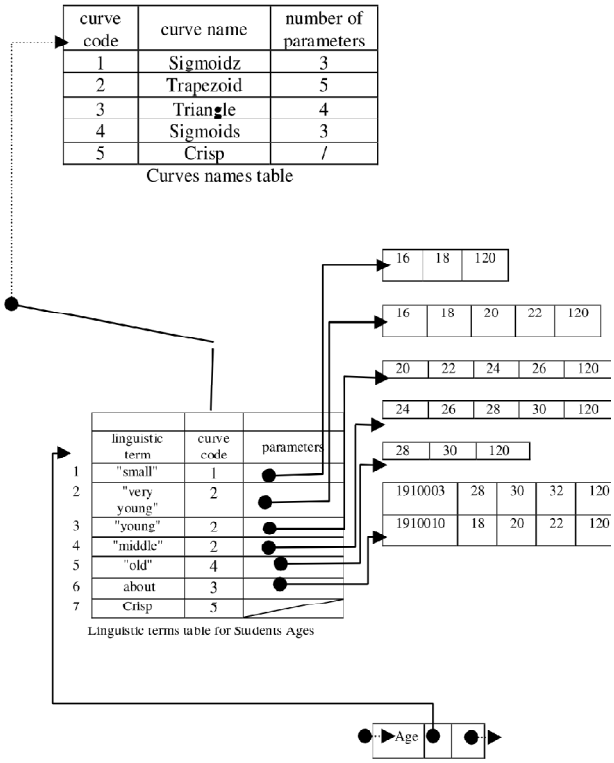

Figure 8: Graphical structure of the fuzzy field age.

Executionof the main program

The structure of the whole automate is represented by the following tree (Figure 9). Each path Root - Leaf yields in either an error which stops the analyzer or a procedure executing a specified action.

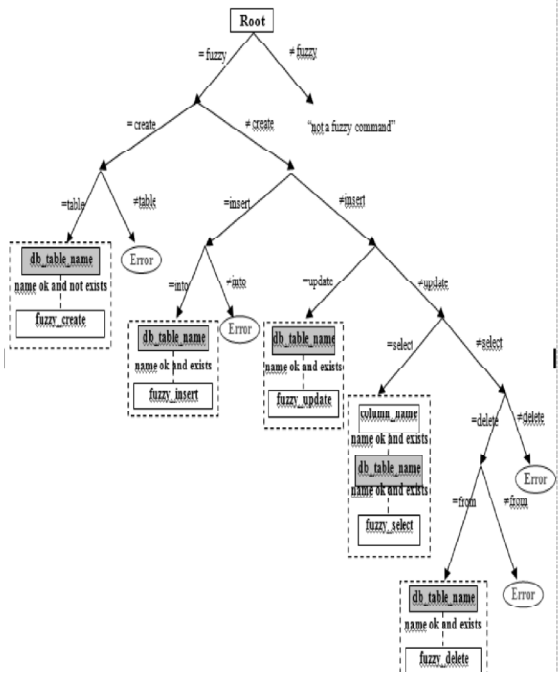

Figure 9: The whole automate for the FSQL commands.

Execution of a command

The following trees (Figures 10, 11) represent a detailed graphical representation of the execution of the program. Initially in the main program, after the user inputs a command, the checking is done to determine if the command begins with the term "fuzzy" or not. If the entered command does not begin with the term "fuzzy", the execution stops after printing a message indicating that the entering command is not fuzzy, otherwise, other checking statements are executed to determine the appropriate procedure or procedures to execute. For example, the "fuzzy_create" procedures are called (Figures 10,11) whenever they entered command begins with "fuzzy create table".

Execution of a fuzzy_create command

As shown in the fuzzy_create trees (Figures 10, 11), the name of the table must be entered, so the db_table_name (Figure 10) procedure is called for checking the table name acceptation: the name is accepted if (a) it is an identifier, (b) it is not a reserved word, and (c) there is no other table with the same name. If there is any error, a specified error message is printed and stops the execution.

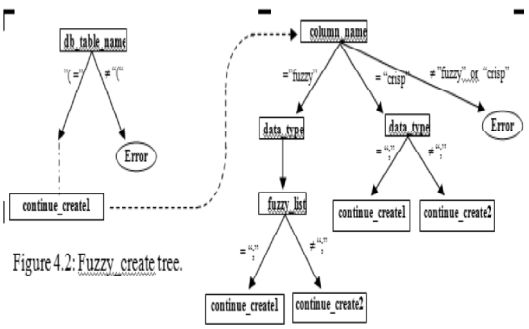

Figure 10: Continue_create1 tree

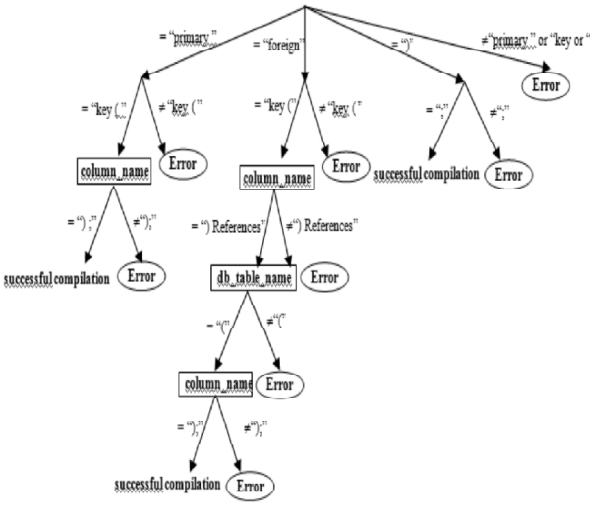

Figure 11: db_table_name tree

ET = 0 means that the entered table name exists. This allows us to insert, update or delete records in the table. ET = 1 means that the entered name does not exist and one can create a table with this name.

Another procedure called by the fuzzy_create procedure is column_name (Figure 12). It determines the acceptation of the entered column inthe command. The column name is accepted if (a) it is an identifier, (b) it is not a reserved word and (c) there is no other column name in the table with the same name.

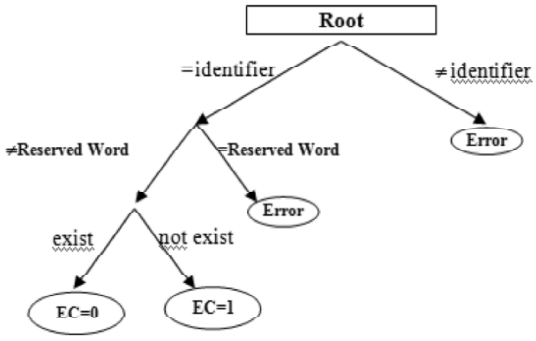

Figure 12: Column_name tree.

EC = 0 means that the entered column name exists and is acceptable, while EC = 1 means that the entered name does not exist.

The data_type procedure is then called to determine if the type of the column field is acceptable or not. Inside all of these procedures an error message is printed whenever the execution encounters any error in the entered command and then stops execution.

Fuzzy create table execution

The following "fuzzy create table" command is correct, so when it is entered, the "successful compilation of create fuzzy command" will appear (Screen 1):

```
FSQL > 1: FUZZY CREATE TABLE STUDENTS
      2:  (CodeStudent crisp integer;
      3:  CodeCollege crisp integer;
      4:  Name crisp varchar (30);
      5:  Age fuzzy integer
      6:  (limit 120;
      7:  crisp;
      8:  'small' sigmoidz(16 , 18 , 120);
      9:  'very young' trapezoid (16, 18, 20, 22, 120);
      10: 'young ' trapezoid (20, 22, 24, 26, 120);
      11: 'middle ' trapezoid (24, 26,28,30,120);
      12: 'old' sigmoids (28, 30, 120);
      13: 'about' triangle (– 2, 2, 120);
      14:     );
      15: );
```

```
"D:\          \Debug\FCOMPILER.exe"
Enter the command.
Enter the EOF ( ^Z ) character to end input.

fuzzy create table students
(codestudent crisp integer;
 codecollege crisp integer;
 name crisp varchar(30);
 age fuzzy integer
 (limit 120;
 crisp ;
 'small'       sigmoidz (16,18,120);
 'very young'  trapezoid (16,18,20,22,120);
 'young'      trapezoid (20,22,24,26,120);
 'middle'     trapezoid (24,26,28,30,120);
 'old'         sigmoids (28,30,120);
 'about 30'    triangle(-5,+5 ,120);
 );
);
^Z
successful  compilation of fuzzy create  command
 String is empty

Press any key to continue
```

Screen 1

In line 1 if the term "fuzzy" is entered as "fuzy" the following result appears (Screen 2):

```
"D:\          \Debug\FCOMPILER.exe"
Enter the command.
Enter the EOF ( ^Z ) character to end input.

fuzy create table students
(codestudent crisp integer;
 codecollege crisp integer;
 name crisp varchar(30);
 age fuzzy integer
 (limit 120;
 crisp ;
 'small'       sigmoidz (16,18,120);
 'very young' trapezoid (16,18,20,22,120);
 'young'      trapezoid (20,22,24,26,120);
 'middle'     trapezoid (24,26,28,30,120);
 'old'         sigmoids (28,30,120);
 'about 30'    triangle(-5,+5 ,120);
 );
);
^Z
not a fuzzy command
the string
 create table students ( codestudent crisp integer ; codecollege crisp integ
 name crisp varchar ( 30 ) ; age fuzzy integer ( limit 120 ; crisp ; ' small
igmoidz ( 16 , 18 , 120 ) ; ' very young ' trapezoid ( 16 , 18 , 20 , 22 , 1
 ; ' young ' trapezoid ( 20 , 22 , 24 , 26 , 120 ) ; ' middle ' trapezoid (
 26 , 28 , 30 , 120 ) ; ' old ' sigmoids ( 28 , 30 , 120 ) ; ' about 30 ' tr
le ( - 5 , + 5 , 120 ) ; ) ; ) ; is ignored
Press any key to continue
```

Screen 2

The following result appears if the term "STUDENTS" is missed in line 1 (Screen 3):

```
"D:\          \Debug\FCOMPILER.exe"
Enter the command.
Enter the EOF ( ^Z ) character to end input.

fuzzy create table
(codestudent crisp integer;
 codecollege crisp integer;
 name crisp varchar(30);
 age fuzzy integer
 (limit 120;
 crisp ;
 'small'       sigmoidz (16,18,120);
 'very young' trapezoid (16,18,20,22,120);
 'young'      trapezoid (20,22,24,26,120);
 'middle'     trapezoid (24,26,28,30,120);
 'old'         sigmoids (28,30,120);
 'about 30'    triangle(-5,+5 ,120);
 );
);
^Z
error: table name expected
error compilation of fuzzy create command
the string
 codestudent crisp integer ; codecollege crisp integer ; name crisp varchar
 ) ; age fuzzy integer ( limit 120 ; crisp ; ' small ' sigmoidz ( 16 , 18 ,
) ; ' very young ' trapezoid ( 16 , 18 , 20 , 22 , 120 ) ; ' young ' trapezo
 20 , 22 , 24 , 26 , 120 ) ; ' middle ' trapezoid ( 24 , 26 , 28 , 30 , 120
' old ' sigmoids ( 28 , 30 , 120 ) ; ' about 30 ' triangle ( - 5 , + 5 , 120
 ) ; ) ; is ignored
Press any key to continue
```

Screen 3

Execution of a Fuzzy_insert command

The Fuzzy_insert procedure (Figures 13) calls the db_table_name, column_name fuzzy_expression procedures. The db_table_name procedure is called for determining if the table name is accepted or not. The column_name is called to determine if the column entered in the command is accepted or not, and the fuzzy_expression procedure is called to ensure that the expression in the command is an acceptable fuzzy expression. During execution of any previous procedures an error message is printed whenever the execution encounters any error.

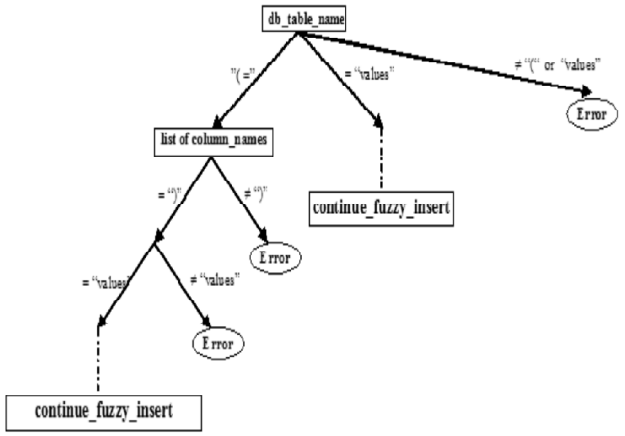

Figure 13: fuzzy_insert tree

The following "fuzzy insert" command is correct, so when it is entered, the "successful compilation of fuzzy insert command" will appear (Screen 4):

FSQL > 1: fuzzy insert into students

2: values (1910001, 101, 'Ali', 'young', 'about 70', 'moh@yahoo.com');

```
"D:\          \Debug\FCOMPILER.exe"
Enter the command.
Enter the EOF ( ^Z ) character to end input.

fuzzy insert into students
values (1910001, 101, 'Mohammad', 'young', 'about 70', 'moh@yahoo.com');
^Z
successful compilation of fuzzy insert command
 String is empty

Press any key to continue
```

Screen 4

In line 2 when the symbol "(" is missed, the following window is shown (Screen 5):

```
"D:\          \Debug\FCOMPILER.exe"
Enter the command.
Enter the EOF ( ^Z ) character to end input.

fuzzy insert into students
values 1910001, 101, 'Mohammad', 'young', 'about 70', 'moh@yahoo.com');
^Z
error: '(' expected
error compilation of  fuzzy insert command
the string
 , 101 , ' Mohammad ' , ' young ' , ' about 70 ' , ' moh @ yahoo . com ' ) ; is
ignored
Press any key to continue
```

Screen 5

Execution of a Fuzzy_update command

The Fuzzy_update procedure (Figure 14) calls the db_table_name, fuzzy_expression and fuzzy_condition procedures. The fuzzy_condition procedure is called to determine if the condition in the command is a fuzzy expression or not, and a printed message must appear when encountering any error before stopping the execution.

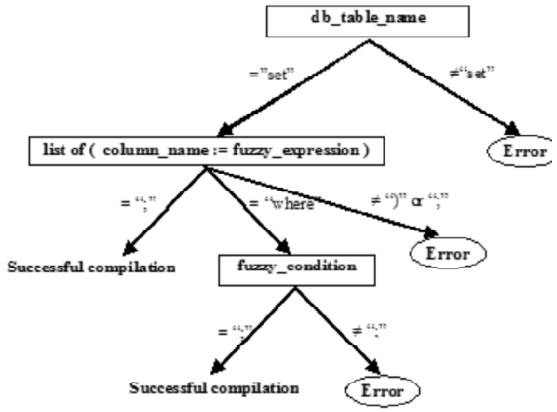

Figure 14: fuzzy_update tree.

The following "fuzzy update" command is correct, so when it is entered, the "successful compilation of fuzzy update command" will appear (Screen 6):

FSQL > 1: fuzzy update students
2: set age := 'very young'
3: where age > = 18 and age < = 20;

```
"D:\          Debug\FCOMPILER.exe"
Enter the command.
Enter the EOF ( ^Z ) character to end input.

fuzzy update students
set   age := 'very young'
where age >=18 and age <=20;
^Z
successful compilation of  fuzzy update command
 String is empty

Press any key to continue
```

Screen 6

Execution of a Fuzzy_delete command

The Fuzzy_delete procedure (Figure 15) calls the db_table_name and fuzzy_condition procedures.

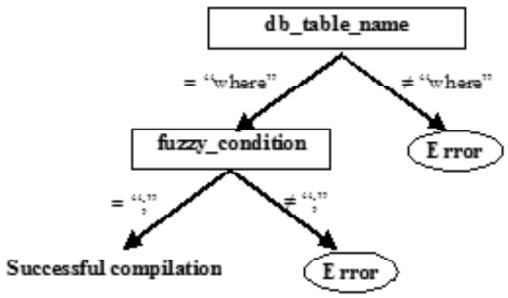

Figure 15: Fuzzy_delete tree

The following "fuzzy delete" command is correct, so when it is entered, the "successful compilation of fuzzy delete command" will appear (Screen 7):

FSQL > 1: fuzzy delete
2: from students
3: where age = 'about 30';

```
"D:\          Debug\FCOMPILER.exe"
Enter the command.
Enter the EOF ( ^Z ) character to end input.

fuzzy delete
from  students
where age ='about 30';
^Z
successful compilation of  fuzzy delete command
 String is empty
```

Screen 7

RELATEDWORK

Fuzzy databases and the FSQL have many applications, and the deductive power is very important. For example, in a hospital, one can make queries such as the following: "Give me a list of young patients suffering from hepatitis who were admitted approximately more than 5 weeks ago." In a supermarket, it would be useful to know the answer to a request such as the following: "Give me a list of the products that have sold very well, but on which we have spent for publicity." (Galindo, Urrutia, & Piattini 2006).

However, real life applications involve a great deal of imprecise and inexact data, which stem from personal traits and judgments, data may be fuzzy i.e. they are known with uncertainty, like "The salary of Salah is low" or else unknown, like "We do not know the salary of Yaser". Thus, the general case is that a database may contain crisp, fuzzy and unknown values. If we use standard SQL we will fail in making a correct decision and conclusion on fuzzy database. So we try to find a way to represent fuzzy values in the database and to extend the standard SQL to deal with crisp and fuzzy values in the database.

There are a lot of fuzzy approaches that have been studied in order to develop fuzzy databases. Here are some of these approaches. (Zvieli and Chen 1986) offered the first approach in fuzzy ER modeling by introducing three levels of fuzziness. (Nakajima, Sogoh, & Arao1993) presented the specification of the Fuzzy SQL language and the implementation aspects of Fuzzy Database Library - FDL2. They concluded that the Fuzzy SQL language and the implementation aspects of Fuzzy Database Library - FDL2 were used to develop the Fuzzy SQL processor.

(Yang, Zhang, Liu, Wu, Yu, Nakajima, & Rishe 2001) extended the unnesting techniques to process several

types of nested fuzzy queries.By using extended merge-join to evaluate the unnested fuzzy queries, (Ben Hassine, Ounelli, Touzi, & Galindo 2007)presented a new approach for the migration from relational DB (RDB) towards fuzzy relational DB (FRDB). The goal of this migration is to permit an easy mapping of the existing data, schemas and programs, while integrating the different fuzzy concepts.

(Hrudaya, Tripathy, Das & Khadanga2009) divided the process into the following steps. At the beginning there is normal relation, which querying through unfocused conditions become fuzzy semi-relation. Doing this, each tuples in result gets classification level in connection with fuzzy conditions. Through fuzzy set operators, semi-relation becomes definite fuzzy relation, which is translated into normal relation by result definition.

Fuzzy conditions can include words like "very", "very much", "little", "few or more" or negation. (Umano, Hatono, & Tamura 1995) developed a fuzzy database system of possibility-distribution-fuzzy-relational model proposed by authors that covers three kinds of fuzziness, (a) data themselves, e.g., " Ali is young" , (b) association between data, e.g., " 'Ali is young' is more or less true " and (c) words in queries, e.g., "Get names of students who are young and excellent" . And they developed two data manipulation languages based on fuzzy relational algebra and SQL, which are standard in ordinary database systems for theoretical issue and practical use, respectively. There are two feasible ways to incorporate fuzziness in DBMS (Liberios, Anton, & Norbert 2006): One is making fuzzy queries to the classical databases and the other is adding fuzzy information to the system. Among several data models, relational data model is the most useful and powerful model (Abraham, Henry, & Sudarshan 2002)

CONCLUSION AND FUTURE WORK

In this paper we have first designed a database by drawing an association diagram allowing requests with ensured responses. We used printed documents (containing all the data) and electronic documents (containing only the codes) in order to get the primary keys and foreign keys according to the normalization rules.

In the second phase we designed and executed automates for the fuzzy commands FUZZY CREATE, FUZZY UPDATE, FUZZY DELETE, FUZZY INSERT and FUZZY SELECT. We introduced simple fuzzy conditions and simple fuzzy expressions.

We proposed a representation of fuzzy data which is entirely based on pointers (inverted index) and queue data structures. Then we tried to design an FSQL model. We discussed the syntax analysis for the fuzzy commands and wrote the corresponding lexical analyzer. During the execution of the proposed fuzzy commands, the proposed commands worked very well and the parser could check the entered command and determined any error correctly.

To overcome the problem of the memory capacity, we suggest checking the inserted record from errors. If the record to be inserted is correct then it will be stored in a data file in a secondary storage device, otherwise the field with error will be rejected.

This work contributes to the database concepts, especially in age of information where the society depends entirely on the databases that contain different formats of data and unstructured text. We could continue this work by completing and enhancing the preceding automates and searching new features related to using fuzzy data. The developed methods may be further investigated and developed in the following ways:

It is possible to study the complex fuzzy conditions (conditions containing AND, OR, NOT operators, ….), and complex fuzzy expressions. It is possible to develop the nested commands like NESTED SELECT to deal with more complex queries and give a good performance for the Fuzzy database.

To make the design more reliable and efficient we can store all the programs related to membership functions needed for ENTERING data and SAVING them in files and UPDATING, DELETING, INSERTING and RETRIEVING data from the database. Designing and implementing new methods to calculate functions like MIN, MAX, and AVERAGE…etc.

REFERENCES

[1] Abraham, S., Henry, F., Sudarshan, S., (2002), Database System Concepts, Mcgraw Hill Education. 4th edition.

[2] Ben Hassine, M., Ounelli, H. , Touzi, A., Galindo, J. (2007), A Migration Approach from Crisp Databases to Fuzzy Databases ", Fuzzy Systems Conference,. FUZZY-IEEE 2007. IEEE International P.P 1-8.

[3] Blanco, I., Cubero, J. C., Pons, O., & Vila, M. A. (2000), An implementation for fuzzy deductive relational databases. In G. Bordogna & G. Pasi (Eds.), Recent issues on fuzzy databases (pp. 183-207). Physica-Verlag.

[4] Carrasco, R. A., Vila, M. A., & Galindo, J. (2002), FSQL: A flexible query language for data mining. In M. Piattini, J. Filipe, & J. Braz (Eds.), Enterprise information systems IV (pp. 68-74). Kluwer Academic.

[5] Galindo, J., Aranda, M. C., Guevara, A., Caro, J. L., and Aguayo, A. (2002), Applying fuzzy databases and FSQL to the management of rural accommodation, *Tourist Management Journal*, 23(6), 623-629

[6] Galindo, J. , Urrutia, A. and Piattini, M., (2006), Fuzzy Databases: Modeling, Design and Implementation. Eds. Idea Group Publishing Hershey, USA.

[7] Hrudaya, K., Tripathy, B., Das, P. and Khadanga, P., (2008), Application of Parallelism SQL in Fuzzy Relational Databases. ICCSIT.

[8] Hudec, M., (2009), An Approach to Fuzzy Database Querying, Analysis and Realisation. Computer Science and Information Systems, Vol. 6, No. 2, 127-140.

[9] Liberios, V., Anton, B., Norbert, A., (2006), Parallelism in fuzzy databases, Teachmedia, 5th Edition.

[10] Nakajima, H., Sogoh, T., Arao, M., (1993), Fuzzy Database Language and Library - Fuzzy Extension to SQL – Second IEEE International Conference on Volume, Issue, p.p: 477 - 482 vol. 1.

[11] Umano, M., Hatono, I., Tamura, H., (1995), FUZZY DATABASE SYSTEMS" Proceedings of 1995 IEEE International Conference on Fuzzy Systems, vol. 5 , 35-36.

[12] Yang, Q., Zhang, W., Liu, C., Wu, J., Yu, C., Nakajima, H.,. Rishe, N. D., (2001), Efficient Processing of Nested Fuzzy SQL Queries in a Fuzzy Database, IEEE Transactions on.

[13] Zvieli, A., and Chen, P. (1986), ER modeling and fuzzy databases. In Proceedings of the Second International Conference on Data Engineering (pp. 320-327).