Applying fuzzy databases and FSQL to the management of rural accommodation

José Galindo\textsuperscript{a,b}, M. Carmen Aranda\textsuperscript{a,b}, José L. Caro\textsuperscript{a,c,*}, Antonio Guevara\textsuperscript{a,d}, Andrés Aguayo\textsuperscript{a,c}

\textsuperscript{a}Departamento Lenguajes y Ciencias de la Computación, Málaga University, Málaga, Spain
\textsuperscript{b}EU Politecnica, Malaga University, 29071 Malaga, Spain
\textsuperscript{c}EU Turismo, Malaga University, 29071 Malaga, Spain
\textsuperscript{d}EU Informática, Malaga University, 29071 Malaga, Spain

Received 12 June 2002; accepted 24 December 2002

Abstract

Database management systems used in information systems for the tourism industry and based on SQL are flexible insofar as they allow queries regarding the data stored in them. However, the SQL language, in spite of its flexibility, cannot deal with the vagueness of the queries clients present to travel agencies. Questions like “I want an inexpensive trip to an Italian city not too far from Rome” cannot be directly answered by the system. Rather, the agent has to make different queries about available trips and then has to summarise and adapt them to the needs of the clients. A solution that gives an answer to these requirements is found in fuzzy logic sets applied to databases.

Currently, we have implemented a fuzzy SQL server (FSQL) in information systems as a result of our previous work in the field of Information Systems (IS). This server is available for Oracle databases and allows queries in traditional relational databases, as well as in fuzzy databases, by using the FSQL language (fuzzy SQL). The FSQL language is an extension of the SQL language that allows vague or fuzzy queries to databases by using fuzzy conditions (with fuzzy comparators), fulfilment thresholds, fuzzy constants, etc.

In this work we show an application of the FSQL language to the tourism sector, in particular, to the IS database of a travel agency which includes attributes with fuzzy values. That is, the database allows the storing of fuzzy values and queries with conditions involving such values. This allows the management of all the company’s products to be carried out in a more efficient way and offers clients the products that best fit their requirements. Also, the system makes it possible to obtain the degree of match for each product in relation to the set obtained in the fuzzy query.

Keywords: Flexible or fuzzy management applications; Travel agencies; Fuzzy SQL; Fuzzy query languages; Fuzzy relational databases (FRDB); Information systems (IS)

1. Introduction

The application of Database Management Systems (BDMS) and database theory to problems of "management" is fundamental, as the large number of applications for this market shows. Applications for the administration or management of all kinds of business and institutions can be found, from a video club to a garage, supermarket, hospital, hotel, school or university. In general, business management applications, independently of their size or field, are based on queries and the updating of a database with a specific piece of software that carries out processing of the data (Caro, Guevara, Aguayo, & Gálvez, 1999; Guevara, Aguayo, Falgueras, & Triguero, 1995; Guevara, Aguayo, Caro, & González, 1997).

In the last few years, fuzzy relational databases (FRDB) have been developed using different models, among which the Prade–Testemale model (Prade & Testemale, 1987), the Umano–Fukami model (1982, 1994), the Buckles–Petry model (Buckles & Petry, 1984), the Buckles–Petry model (Buckles & Petry, 1984), the Buckles–Petry model (Buckles & Petry, 1984), and the Zemankova–Kaendel model (Zemankova-Leech &
Kandel, 1985) and the GEFRED model of Medina, Pons, and Vila (1994) are worth mentioning. This last model represents an eclectic synthesis of the previous models dealing with the problem of the representation and management of fuzzy information in relational databases. One of its main advantages is that it works as a general abstraction and this enables us to handle different approaches and models, even if they seem to be very different.

FRDBs can be used in all classical database applications because they enhance databases by making it possible to store and treat imprecise data, something which cannot be done using only traditional models of databases. Imprecise information is a common phenomenon in any context given that it is not unusual to receive information in an incomplete or inexact way. In traditional databases, if there is other than precise information about something, the value NULL is stored, preventing the storage any information that is unavailable because it is not a “precise” fact. For example, in traditional databases, if a man is “tall”, but his exact height is not known, the value NULL will be stored, whereas in an FRDB it is possible to store the information given by the concept “tall”.

By using the deductive power of the fuzzy SQL (FSQL) language properly, very useful results can be obtained. For example, in a hospital, one could make queries like, “Give me a list of young patients suffering from hepatitis and who were admitted approximately more than 5 weeks ago”. In a school, queries such as, “Give me a list of the students who have passed Maths with good marks and have passed Physics with average marks” can be made. In a supermarket, it would be useful to know the answer to requests for such things as “a listing of the products that have sold very well, but which we have spent little on regarding their publicity”.

The list of management applications and useful queries that could be done in this way is endless. This paper illustrates how FRDBs and the FSQL language can contribute to the management of a travel agency (Galindo & Aranda, 1999). In more specific terms, the concepts will be applied to the management of rural accommodation. First there is a description of the main advantages that the FSQL language adds to the command SELECT to express fuzzy queries. A more detailed description of this and other FSQL commands, as well as an explanation of the FSQL server architecture can be found in Galindo, Medina, Pons, and Cubero (1998, 1999). Also, the type of information that can be stored in an FRDB is shown. This article ends with the practical application of the FRDB to the management of rural accommodation in a travel agency and some conclusions.

2. Flexible queries with FSQL

The FSQL language (Galindo et al., 1998; Galindo, Medina, Vila, & Cubero, 1998; Galindo, 1999) is an extension of the SQL which allows the use of vague, flexible or fuzzy commands. The FSQL server is a program that allows the execution of SQL commands and is programmed in PL/SQL for Oracle DBMS (DataBase Management System). The FSQL server has been designed to work with both fuzzy databases as well as traditional ones, and so it is very fast and easy to implement the advantages of flexible queries in already existing traditional databases.

The FSQL language extends the command SELECT to express fuzzy queries. However, given its complex format, this article will only provide a summary of the main extensions added to this command:

- **Linguistic labels:** If an attribute is amenable to fuzzy treatment, then linguistic labels are defined that are preceded by the $ symbol to distinguish them easily. Each label has a possibility distribution associated with it or it is value scaled with a similarity relationship, as illustrated later. The possibility distributions are based on Zadeh’s fuzzy sets (Zadeh, 1978). Fig.1 shows an example of the distribution of a trapezoidal possibility given by four values that define the fuzzy value “tall” of the attribute height, for example, a relationship (or table) of people. Possibility distributions can take any form, but the trapezoidal form simplifies calculations a good deal without losing expressive power when dealing with fuzzy concepts. One drawback, rather difficult to solve, is the influence of subjectivity in the definition of the fuzzy concepts included in each specific application (see Fig. 1).

- **Fuzzy comparators:** Besides the common comparators (=, >, etc.), FSQL includes the fuzzy comparators shown in Table 1. Their definition can be found in Galindo (1999) and Carrasco et al. (1999). As in SQL, the FSQL fuzzy comparators are able to compare a column (or attribute) to a constant or two columns of the same type. Necessity comparators are more restrictive than possibility comparators, and so their degree of fulfilment is always smaller than the degree

![Fig. 1. Trapezoidal possibility distribution for “Tall”](image-url)
of fulfilment obtained by their corresponding possibility comparator. In general, necessity comparators demand that the condition is fulfilled to a certain degree, even if not completely, whereas the possibility comparators measure to what extent (or degree) it is possible for the condition to be fulfilled.

- **Fulfilment threshold** ($\gamma$): A minimum fulfilment threshold (1 by default) can be established for each simple condition, with the following general format: $<\text{condition}> \text{THOLD} \gamma$, indicating that the condition must be fulfilled to a minimum degree of $\gamma \in [0,1]$. The reserved word THOLD is optional and can be substituted by a traditional comparator ($=$, $\leq$, etc.), thus modifying the meaning of the query. The word THOLD is equivalent to the comparator $\geq$.

- **CDEG Function** ($<\text{attribute}>$): This function shows a column with the degrees of fulfilment of the query condition for the attribute specified in parenthesis as the argument of the function. If there are logical operators in the condition, the calculation of the compatibility degree is done by using the minimum (triangular norm) and the maximum (triangular conorm), although the user can easily change these functions (Galindo, 1999). If the CDEG argument is an attribute, then the function only uses simple conditions where this attribute is used. It could be more useful to use CDEG(*) to obtain the degree of fulfilment of each tuple in the whole condition (considering all its attributes, not just one).

- **Wild card %**: This is similar to the wild card * in SQL, but this also includes columns for the attributes’ degrees of fulfilment (using the CDEG function), that are used in the condition.

- **Fuzzy constants**: All constants shown in Table 2 can be used in FSQL. The first three can be used as in Umano (1982) and will be explained later. Value $\#n$ is represented as a trapezoid where $\beta = \gamma$, and $\beta - \alpha = \delta - \gamma = \text{margin}$, these values being stored in the Fuzzy Meta-knowledge Base, FMB, as shown in the following section.

- **IS condition**: This is used with the same format as in standard SQL, but includes the first three types of fuzzy constants shown in Table 2:

\[ \langle \text{Fuzzy.Attribute} \rangle \]

\[ \text{IS [NOT] (UNKNOWN|UNDEFINED|NULL)} \]

If the attribute is not fuzzy and the constant is NULL, then the meaning of such a constant is different, taking the meaning given by the DBMS.

### 3. Data in FRDBs

“Data” is understood as any type of information stored in the database. In our model, data is classified in two categories:

1. **Traditional database**: This includes traditional data, stored as relationships, but with a special format to represent the fuzzy attributes included (those admitting “vagueness”). Fuzzy attributes can be of three types:
   - **Type 1**: These attributes are totally crisp (i.e., classic attributes lacking vagueness), but they accept flexible queries when used under fuzzy conditions (with fuzzy comparators, fuzzy constants, fulfilment thresholds, etc.). Also, one can define linguistic labels in their domain, which will be associated with some specific fuzzy value (see Fig. 1). In query conditions, all the fuzzy constants included in Table 2 with these attributes can be used. These types of fuzzy attributes can be used to exploit the advantages of FSQL in traditional databases. In this way, the already existing attributes in the database can be declared Type 1 fuzzy attributes, and so it is possible to define labels and apply fuzzy comparators to them.
   - **Type 2**: These attributes admit both crisp and fuzzy data as possibility distributions on an underlying ordered domain. Also, with these attributes, one will be able to store and use all the fuzzy constants shown in Table 2.
   - **Type 3**: These attributes are defined in an underlying non-ordered domain, such as the colour of

| Possibility | Necessity | Meaning: possibly/necessarily...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FEQ</td>
<td>NFEQ</td>
<td>Fuzzy equal</td>
</tr>
<tr>
<td>FGT</td>
<td>NFGT</td>
<td>Fuzzy greater than</td>
</tr>
<tr>
<td>FGEQ</td>
<td>NFGEQ</td>
<td>Fuzzy greater or equal</td>
</tr>
<tr>
<td>FLT</td>
<td>NFLT</td>
<td>Fuzzy lesser than</td>
</tr>
<tr>
<td>FLEQ</td>
<td>NFLEQ</td>
<td>Fuzzy lesser or equal</td>
</tr>
<tr>
<td>MGT</td>
<td>NMGT</td>
<td>Much greater than</td>
</tr>
<tr>
<td>MLT</td>
<td>NMLT</td>
<td>Much lesser than</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuzzy const.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN</td>
<td>Unknown value, but attribute applicable</td>
</tr>
<tr>
<td>UNDEFINED</td>
<td>Non-applicable value</td>
</tr>
<tr>
<td>NULL</td>
<td>Totally unknown: it is not known whether is applicable or not</td>
</tr>
</tbody>
</table>

### Table 1

The 14 fuzzy comparators of FSQL

<table>
<thead>
<tr>
<th>Fuzzy comparators</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDEG</td>
<td>Fuzzy</td>
</tr>
<tr>
<td>FGT</td>
<td>Fuzzy greater than</td>
</tr>
<tr>
<td>FGEQ</td>
<td>Fuzzy greater or equal</td>
</tr>
<tr>
<td>FLT</td>
<td>Fuzzy lesser than</td>
</tr>
<tr>
<td>FLEQ</td>
<td>Fuzzy lesser or equal</td>
</tr>
<tr>
<td>MGT</td>
<td>Much greater than</td>
</tr>
<tr>
<td>MLT</td>
<td>Much lesser than</td>
</tr>
</tbody>
</table>

### Table 2

Fuzzy constants that can be used in FSQL fuzzy comparisons

<table>
<thead>
<tr>
<th>Fuzzy const.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exists x, \beta, \gamma, \delta$</td>
<td>Possibilistic trapezoid (con $\alpha \leq \beta \leq \gamma \leq \delta$): example in Fig. 1</td>
</tr>
<tr>
<td>$[n, m]$</td>
<td>Range: “between $n$ and $m$”</td>
</tr>
<tr>
<td>$#n$</td>
<td>Value “approximately $n$” (triangle): $n \pm \text{margin}$</td>
</tr>
</tbody>
</table>
the hair. Some labels are defined for these attributes, which are scaled with a similarity relationship defined between them, indicating to what degree a pair of labels match each other. Given that there is no order established in their domain, we will only be able to use the fuzzy possibilistic equality comparator (FEQ) with these attributes. Obviously, the trapezoid, range, or “approximate value” type constant values shown in Table 2 will not be accepted.

2. Fuzzy meta-knowledge base (FMB): Here information about the FRDB and its fuzzy attributes is stored in a relational format. Fuzzy attributes and their type need to be stored, and depending on the type, the information stored will be different for each of them: (a) Type 1: In order to use crisp attributes in fuzzy conditions for flexible queries, they have to be declared as Type 1 fuzzy attributes and store the following data in the FMB: trapezoid linguistic labels, a margin value for the approximate values (see Table 2), and the minimum distance for two values to be considered very far (employed in \( \leq \)/\( \geq \) and \( \leq \)/\( \geq \) fuzzy comparators); (b) Type 2: The same data as for Type 1 fuzzy attributes is stored in the FMB; (c) Type 3: Linguistic labels and the similarity relationships between them are stored.

4. FSQL for the management of accommodation in a travel agency

The database design of any business must not only include fuzzy attributes, but also traditional attributes, such as client name, telephone number, etc. Generally speaking, to give more versatility to the system, more Type 2 than Type 1 fuzzy attributes would be used. However, it must be taken into account that Type 2 attributes require more storage space and processing time. Therefore, a balance must be reached between flexibility (in fuzzy representation and treatment) and efficiency (in storage space and CPU time).

With the development of rural tourism in Spain, accommodation in rural areas has experienced a great boom. This kind of lodging offers clients some special features related to its rural location and the use of the place. These characteristics are stored in the database so that the agency can search for the accommodation most suitable for the client’s needs (Caro et al., 2000a, b).

By using an FRDB, the agency will have all the capabilities of traditional databases and the added advantage of having the option of storing attributes with vague values and carrying out flexible queries on both fuzzy and classic attributes. For example, the user of an FRDB can make a query such as: “List all cheap lodgings in mountain settings, where it is not too cold and is near a city”.

For this kind of application, the following attributes will be considered as being Type 1: the number of rooms, range of services, number of floors, etc. Generally speaking, these values for such attributes are defined without vagueness. However, one can make queries by imposing fuzzy conditions on them.

In this example, most attributes would be Type 2, so as to make the database as flexible as possible. Among these attributes can be included the size of the garden and the parking lot (if applicable) and, more importantly, the following:

- **Price**: Very often, the price of rural accommodations is not fixed but “negotiable”, so the owners establish an approximate or a fixed price, but tell the agency that they would be willing to negotiate the final price.
- **Surface area (m\(^2\)**: Very often it is difficult to have access to the deeds of the house or make exact measurements of its surface area. However, the FRDB allows the storage of approximate values of the type “approximately 100 m\(^2\)”.
- **Age**: For some tenants, it is very important to know the approximate age of the property they are going to spend their holidays in, but it is usually quite difficult and unnecessary to know the exact date. The FRDB allows the storage of fuzzy values such as “new”, “semi-new”, “old”, “very old”, “approximately 15 years old”, etc.
- **Facilities**: Even more important than having information about age is for the clients to know about the facilities and general state of the property. Some labels defined for this attribute are in a domain from 0 to 10, which indicate a mark or evaluation on the general state of the place and the facilities and services offered. Alternatively, these labels are “Super Luxury”, “Luxury”, “Good”, “Average”, “Poor” and “Very Poor”.

In this case, the attributes given below are Type 3 fuzzy attributes. Each one has a set of scaling labels, and there is a similarity relationship established between them: light (amount of sun), noise, views, and the quality of furniture. The following are also Type 3 attributes:

- **Location**: This attribute has been defined as length 3, indicating that the property can be located between three areas, with a different degree (between 0 and 1) for each one. For example, several rural houses and a camping site are located within a given natural area; the value \{0.5/Camp, 1/Mountain, 0.7/Lake\} indicates that the property is located in the mountains, closer to the lake than to the campsite.
- **Type of accommodation**: This attribute distinguishes between apartment, chalet, farmhouse, flat, rural house, and semi-detached house. For example, a chalet is defined as similar to a semi-detached house...
with a 0.8 degree, so that if a request is made for accommodation similar to a chalet with a minimum degree of 0.8 (or smaller), the result of this query will also list semi-detached houses. In general, the degree of global fulfilment of a query for semi-detached houses will be smaller than the one for chalets. This is a very useful feature, because a client who wants to rent a chalet is also potentially interested in renting a semi-detached house. When the accommodations selected are shown to the client, the agent does it in a descending order of degree of match, so that the client gets to see all the places the agency has which might be to his/her interest.

- **Tourist area:** This attribute indicates the kind of habitat or the ambience the property is located in and can take values such as: “beach”, “mountain”, “village”, “city”, “cultural”, “nature”, “mountain”, “solitary”, “lively”, etc. In this case, the same property can take values with several labels and there is an independent degree of possibility for each of them.

The following values can also be stored: **UNKNOWN**, if nothing is known about the value of a given attribute. The value **UNDEFINED** is used if a given value is not applicable or meaningless (for example, the size of the garden in a house with no garden). The value **NULL** is used if nothing is known about the attribute; that is, it is not known whether the attribute is applicable or not (for example, it is not known whether there is a garden or not).

With this kind of database, many different types of queries can be made. For example, one can compare the list of available accommodation and the list of accommodation requested. The first list stores all available accommodation not booked yet and their characteristics. The request lists store the general characteristics of the accommodations requested by certain clients. Both listings can be compared for each client, and there can be obtained a list of possibilities with a degree (between 0 and 1) that measures to what extent the place fulfills the characteristics of the property this client is looking for. Such comparisons can first be made with necessity comparators (Table 1) and thresholds strictly greater than zero. If the query retrieves too many properties, the fulfillment threshold is increased, and, if the query retrieves too few, the necessity comparators can be swapped for their corresponding possibility comparators, which are less restrictive.

Another possibility is to query the FRDB just at the moment clients indicate their preferences, so that offers can be presented to them. For example, if we assume that a client tells the agent that she is looking for a large house or chalet in a mountain village with at least eight bedrooms. In this case, the next FSQL query retrieves properties that fulfill those conditions in descending order regarding the condition's degree of fulfillment. Possibility comparators are used to retrieve a larger amount of tuples, because the many elementary conditions might result in insufficient retrievals:

```
SELECT CDEG(*), Accommodation.*
FROM Accommodation
WHERE Type FEQ $Chalet .5
AND Surface FGEQ $Large .5
AND Room FGEQ #8 .5
AND Area FEQ $Village .7
AND Area FEQ $Mountain .7
ORDER BY 1 DESC;
```

In this query, detached and semi-detached houses will also be retrieved if the type of property has a degree of similarity higher or equal to 0.5 regarding the type chalet.

In this way, the clients are offered properties with a higher degree of fulfilment, that is, those closer to their preferences. It is easy to see that the number of possible queries and the usefulness and versatility of the FSQL language are very large.

Fuzzy queries reduce the risk of obtaining blank answers, since they allow the use of a finer discrimination scale: the range [0,1] instead of the set {0,1}. That is, they enable the recovery of tuples that in a crisp query would not have drawn any results. However, it can sometimes happen that there are no elements that would fulfill the result of a query. FSQL queries are especially flexible for solving this possible problem, because the requisites in each simple condition can be relaxed by acting on the following query parameters:

- **Fuzzy comparators** (Table 1): There are many comparators and alternating between possibility and necessity, comparators can be especially useful.
- **Thresholds**: By modifying this value, the degree of relevance of the tuples sought for can be determined in order to retrieve only the most important ones.
- **Using classic comparators instead of the word THOLD**: In this way, meanings of queries can be modified. For example, one can retrieve the less important elements using the comparator < (or <=). It is possible to also retrieve just those elements that fulfil the condition within a certain degree using the comparator =.
- **Fuzzy constants** (Table 2): If the right-hand part of a simple condition is a fuzzy constant, it can be modified to make the query more flexible and so better achieve our objective.
- **Logical operators**: Some simple conditions that are not very important can be eliminated, or less important conditions can be linked with the logical operator OR instead of AND and parentheses used appropriately.
5. Conclusions

The system presented in this article seeks to show some of the advantages that FRDB can offer to general management applications that use traditional databases. This system is more than a theoretical discussion, because it has been implemented under the DBMS Oracle. Currently, the authors have an FSQL language server for Oracle, which is both easy to install and use. The FSQL language has a syntax very similar to the popular SQL, which makes it easy to learn and use. On the other hand, it is easy to implement Client FSQL programs that use the language and the FSQL server in a transparent way to users, without them having to become familiarised with the syntax of FSQL. At the moment, the authors are working on the creation of a Visual FSQL Client that could be executed via Internet (in Java).

The system presented here focuses on typical activities in a travel agency, but it can be transferred to any other business activity. Obviously, this flexible query system does not guarantee closing the deal, but it does ensure finding just the information needed at any given moment. It must be borne in mind that when people look for a place to spend their holidays, they hardly ever have a fixed and established idea, but rather they look for something in a more or less vague way, with certain initial characteristics, and these usually change when the agent lists their available products. This system allows the travel agency to maintain a large database without having to remember the general characteristics of all the properties and packages available, which would make the management of a large database quite impossible.

The power of the FSQL language, the large number of fuzzy comparators implemented, the flexibility regarding establishing fulfillment thresholds, the possibility of installation and use in a traditional DBMS and other characteristics, makes clear the many advantages of FRDBs. This is an important step forward in the transfer of results from the research field to the managerial world in general.

The incorporation of FSQL in a tourist website is an example of real applications. The tourist website “Ronda: The enchanted land” (Aguayo, Caro, Guevara, & Galvez, 2001) is the result of an R&D project carried out by a group of professors from the University of Malaga in collaboration with the firm Turismo de Ronda S.A. This website contains a relational database that includes tourist resources (including rural accommodations) of the surrounding area (La Serrania). The FSQL will be incorporated in the database improving the flexibility and accessibility to the tourist.

References


José Galindo is professor of Computer Science in the School of Engineering at Málaga University. Research interest: Fuzzy logic, Databases. He is a member of FrontDB research group.

M. Carmen Aranda is professor of Computer Science in the School of Engineering at Málaga University. Research interest: Image processing, pattern recognition and artificial intelligence.

José L. Caro is a professor of “Information Systems applied to Tourism” in the University School of Tourism and “Databases” in the Computer Science Studies and Academic Secretary at the Center. He is a SICUMA research group member. Research interest: Workflow management systems and information systems applications in tourism.

Antonio Guevara is a professor of “Information Systems” in the Computer Science department and “Information Systems applied to Tourism” and “Information Technology in Tourism” and Subdirector at the Center. He is the director of a SICUMA computer science research group centered in information systems methodology and development. Research Interest: Information systems, tourism information systems.

Andrés Aguayo is a professor of “Information Systems applied to Tourism” and “Information Technology in Tourism” in the University School of Tourism and “Operating Systems” in Computer Science and the coordinator of web services at School of Tourism and also a SICUMA member. Research Interest: Internet technologies and information technologies in tourism.