Semantic and Fuzzy Integration: A New Approach to Efficient and Flexible Querying of Relational Databases

Rachid Mama, Mustapha Machkour
Ibn Zohr University-Faculty of Sciences, Information Systems and Vision Laboratory, Agadir, PB 810 Morocco

***Abstract*—Data are "gold mines" that must be processed and interpreted quickly and efficiently to be useful. Thus, flexible queries continue to attract considerable attention. Several works have been proposed that allow users to perform flexible queries on relational databases. Most are related to fuzzy logic, which showed its performance in handling fuzziness in scalar values, but non-scalar values are still a more complex task. To solve this drawback of fuzzy logic, we propose using ontologies to establish the semantic relationships between the domain elements of a queried attribute. Moreover, we present the architecture of a new system that combines both techniques to allow users to write and execute queries in a flexible way where the criteria are not only exact but can also be fuzzy or semantic, and they may also include accomplishment degrees. Furthermore, the proposed system uses a new fast methodology for handling fuzzy queries, which has shown its great efficiency in accelerating the execution of fuzzy queries. Data mining techniques are used to assist users in defining their fuzzy understanding. The developed system has a user-friendly interface to assist users in managing their fuzzy preferences and semantic preferences. Finally, we have proven the performance of our system by conducting a set of experiments in different areas. We have also provided a qualitative and quantitative comparison with flexible query systems, which are documented in the literature.**

***Keywords*—Relational databases; fuzzy logic; ontologies; flexible queries; user interface**

I. Introduction

In this paper, we suggest a new proposal for flexible querying relational databases where answer results are not limited to the searched object but also similar information to this object. It is based on fuzzy logic and ontology. Numerous approaches have been proposed in the literature to enable flexible querying in relational databases, with most relying on fuzzy logic as an effective tool for managing imprecision. In terms of scalar values, we can simply define any fuzzy set according to the domain of the queried attribute, for example, we can use a trapezoid function to represent the "young people" fuzzy set as shown in Fig. 1, to compute the search in the query "Return young people". However, handling non-scalar attributes using fuzzy logic, such as "attitude", "qualification", or "hair color", which are closer to natural language representations, is a complex task, as it requires explicitly defining relationships between each pair of domain values to establish a similar relationship among them. This is the case, for example, when querying: "Return blond people". In such situations, similarity relationships among all domain values of the hair_color attribute must be defined during the design phase, as illustrated in Table I. Such definitions are a difficult task because they depend on the subjective perception of the designer who evaluates this degree of similarity, as well as the application domain of the non-scalar attribute. To solve this drawback of fuzzy logic, we propose the use of a new technology that has appeared in the past decade to make semantic queries. It consists of using ontologies to define the semantic relationships among the domain elements of a queried attribute.

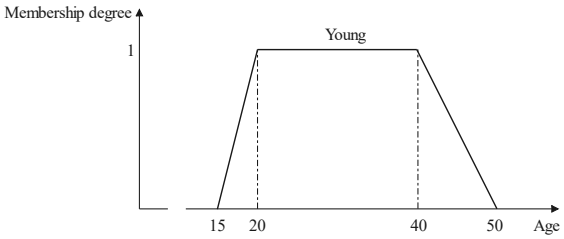


Fig. 1. Fuzzy set represents the human age.

TABLE I. A Human Hair Color Similarity

Hair colour	*Red*	*Brown*	*Blond*
Dark	0.2	0.7	0.1
Red		0.6	0.5
Brown			0.3

Despite significant progress in developing flexible query systems, most existing systems struggle to efficiently support the combination of fuzzy predicates (which handle imprecise or vague information) with ontology-based predicates (which leverage semantic reasoning). Current approaches often treat these paradigms in isolation, leading to inefficiencies, limited expressiveness, or the inability to process hybrid queries involving fuzzy logic and ontological knowledge. This paper addresses this gap by proposing a novel query processing framework that seamlessly integrates fuzzy and ontology predicates within a unified and flexible query system.

The main contribution of our proposal is to provide a solution for answering flexible queries, such as the following: "return all the new action movies" where the concept action is not included in any database, and the term "new" is fuzzy, representing recent films. Or, like this, "return all vintage books about life sciences" that is, old books on Biology, Botany, Zoology, etc. Generally, any query contains fuzzy terms, semantic terms, or both, with preferences if specified.

In this approach, when executing a flexible query, the

system evaluates the query, identifies conditions, and handles each non-Boolean condition based on the domain of the queried attribute. If the domain is scalar, the system applies fuzzy logic techniques; however, if the domain is non-scalar, it employs semantic techniques. We have described the system architecture and the development process for processing flexible queries. In addition, as proof of the behavior and performance of the system, we conducted three experiments, including a detailed example of the behavior of our system as a proof of concept.

The remainder of the paper is organized as follows. Section II surveys and reviews related works, classifying them into two taxonomies. Section III details the system architecture and development details. In Section IV, the results of experiments are presented, and the main features of this proposal are compared with existing approaches. Finally, Section V concludes the paper.

II. Related Work

Several research efforts have focused on addressing fuzziness in database systems, with most of them remaining largely theoretical [1], [2], [3], [4], [5], [6]. The integration of fuzziness into database systems has been a longstanding research focus [7], [8]. Fuzzy logic, designed to manage uncertainty and imprecision, has been explored to enhance traditional database systems. While theoretical work has laid the foundation for understanding and representing fuzzy information in databases, practical implementations have been limited. To highlight the important proposed flexible query systems, we propose two taxonomies: the first focuses on flexible query systems for crisp relational databases, while the second addresses flexible query systems for fuzzy relational databases.

In crisp relational databases, various models incorporating fuzzy terms have been explored to enable flexible queries, with SQLf standing out as a notable achievement [9], [10]. Based on their architecture, these proposed systems can be categorized into weakly integrated, semi-integrated, and strongly integrated. Each category offers different approaches to incorporating fuzzy logic into traditional database systems, e.g. VAGUE [11], SQLf3 [12], iSQLf [13], SQLf-pl [14], ReqFlex [15], and PostgreSQLf [16].

Different models have been proposed for fuzzy relational databases to build a database that can involve imprecision and vagueness represented by fuzzy or possibilistic elements and to support the handling of imprecise queries. Noteworthy models include the GEFRED model, which stands out as the most generalized model of fuzzy relational databases, it constitutes an eclectic synthesis of the various published models aimed at addressing the representation and treatment of fuzzy information for relational databases [17], [18]. Additionally, fuzzy database management systems, such as FREEDOM-O and a fuzzy interface called FIRST, have been developed to address these challenges [19], [20].

However, in the above works, the proposed solutions to similarity management in non-scalar values are still inappropriate and have several disadvantages. Such a definition of an explicit relationship between each pair of the domain of a non-scalar attribute is a hard task because it depends on the application domain and the subjective perception of the designer. Similarly, the attribute domain is limited to the initial domain data set.

To solve these issues, we propose an alternative that consists of using ontologies to formally represent the semantics of a domain. The basic goal of our research is to develop a complete system that allows users to write and execute flexible queries in conventional databases where criteria can be classical, fuzzy, or semantic.

III. Description of the System

In our proposal, two technologies—fuzzy logic and ontology—are integrated into a single system to enable flexible querying of the relational database. On the one hand, fuzzy logic has been used to process fuzzy queries by defining fuzzy sets (linguistic variables) and associating them with selected attributes, according to the same strategy presented in our recently published proposal [21]. Fuzzy sets can be defined using any kind of membership function, e.g. triangular, trapezoidal, or Gaussian. On the other hand, semantic queries rely on ontology-based semantic similarity to return tuples containing information that is semantically similar to the concepts mentioned in the query.

Fuzzy query processing is done with the same methodology that we have presented in [21]. We employed fuzzy views to manage the satisfaction degrees associated with user-defined fuzzy predicates. This simple and intelligent technique allows us to write and execute fuzzy queries as classical SQL, which induces an important verified performance.

Semantic query processing is performed by utilizing a chosen ontology to define the semantic relationships among terms within the same domain of the queried attribute. A semantic query returns an ordered result dataset by comparing the contents of the database using a selected similarity measure to the ontology that represents the domain of the queried attribute.

In the following subsections, we present the architecture of the proposed system and the development process that enables flexible querying of relational databases.

A. System Architecture

The proposed system architecture, shown in Fig. 2, has two principal parts:

- A database that stores data and the attribute domains, which can be defined by fuzzy sets or ontologies, and will be stored in the database catalog.
- A three-module functional part that defines the system's behavior:
    - Flexible query process module (FQM): this module is divided into two sub-modules, and it is responsible for performing three basic operations (see Fig. 3):
        - The input processing stage extracts query conditions and classifies them into three types. Each condition is then routed to the appropriate process module based on the

domain of the query attributes, with the exception of Boolean conditions, which are retained as they are directly supported by the DBMS.

- Replaces each non-exact condition with its corresponding Boolean one in the original query.
- Building an ordinary query and sending it to the database and getting results.

- Fuzzy condition process module (FzzCM): The aim of this module is only to change the syntax of the fuzzy condition to an SQL syntax that we proposed in [21] and will be explained in detail in the next section. Therefore the fuzzy condition becomes a classical one that applies to the virtual column of the fuzzy view associated with the concerned table.
- Semantic condition process module (SemCM): This process executes an algorithm that returns a set of database terms semantically similar to the searched term in the input semantic condition, by assessing the similarity degree between the database content and the searched term, along with their relationships in a selected ontology. Finally, this process generates an SQL IN condition by using this result set.

A list of attributes and modules that can be involved in each kind of condition is shown in Table II. It is worth noting that all the conditions involve the flexible query module to analyze the input.

TABLE II. TYPE OF ATTRIBUTES AND THEIR CORRESPONDING PROCESS MODULES

Kinds of attributes	*Module*	*Accomplishment degree*
Ordinary attribute	Ordinary DB	1 or 0
Fuzzy attribute	FzzCM	Membership degree
Semantic attribute	SemCM	Similarity degree

B. Development

A Database Server running an Oracle instance and a Java-based client application are required for the system architecture. The database server provides data services and manages fuzzy queries. Also, it provides the informational needs (metadata and data) of semantic queries, fuzzy queries, and I/O processing. It is worth noting that the database must be prepared thanks to the implementation of a set of stored procedures in PL/SQL, which allows for example the management of fuzzy query metadata (DFC), semantic query metadata (DSC), and the generation of fuzzy views.

A Java web-based application has been developed to implement the principal operational functionalities described in the previous section. It features a user-friendly interface that simplifies tasks for users, such as preparing fuzzy and semantic queries. Additionally, it performs operations not directly supported by the database, including ontology-based semantic similarity computation.

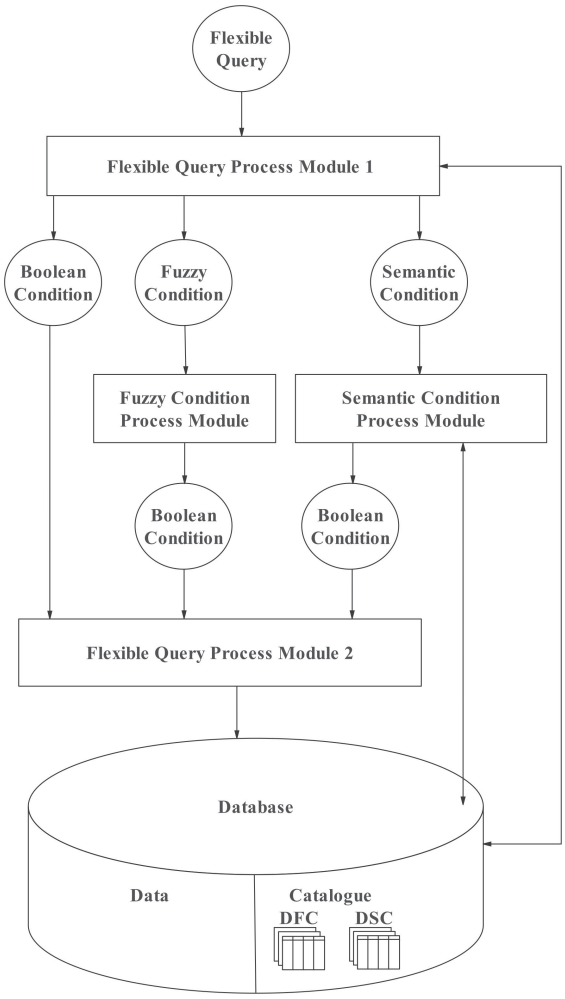


Fig. 2. System architecture.

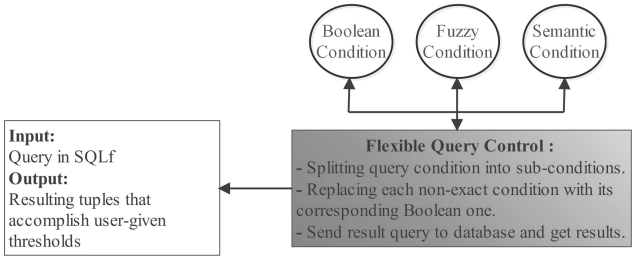


Fig. 3. Flexible query process module behavior description.

We used data mining techniques to assist users in expressing and refining their fuzzy understanding of complex datasets. The web version of this application has been designed for universal accessibility, irrespective of the user's operating system. This decision ensures a flexible system that accommodates users with internet connectivity, enabling remote access from any location. For those interested, a working version of our system and its description can be found on the following website: https://github.com/mathmama/FQSFO.

This section provides an analysis and description of the system's behavior and implementation details.

1) Fuzzy queries: To manage fuzzy queries that contain fuzzy conditions, we have used our published approach [21], which extends the SQL language to allow us to write fuzzy conditions in our queries without the need for a translation/parser. His main principle is to use views (called fuzzy views) to manipulate the membership degrees related to user-defined fuzzy predicates rather than calculating them at runtime using user functions built into the query. As a result, the response time for executing a fuzzy query will be reduced.

When the database is intended to be fuzzily queried, it must be prepared and initialized by user preferences. For example, if we are querying: "Return young employee", we have to associate the Age attribute to the linguistic variable Age that contains a fuzzy set (linguistic value) called Young as part of its domain as shown in Fig. 1, in this time we have defined a fuzzy predicate. Then, a fuzzy view will be generated automatically that contains a virtual column named "age.young" to manipulate the membership degrees. Consequently, the above fuzzy query can be expressed with SQL as:

$$\begin{aligned} &\textbf{select} \;\; emp \;\; \textbf{from} \;\; FEMP \\ &\textbf{where} \;\; \text{“}age.young\text{”} > 0 \end{aligned} \tag{1}$$

where FEMP is the fuzzy view generated from the initial table (see the example given in Table III, which is generated based on the definition of the term "young" identified in Fig. 1). Note that this new strategy allows the user can also define thresholds on his fuzzy conditions and easily integrate most fuzzy query characters such as fuzzy quantifiers, fuzzy modifiers, fuzzy joins, etc. [21].

TABLE III. GENERATED FUZZY VIEW FEMP

emp	e-name	age	age.young
12	Martin	29	1
10	Smith	46	0.4
20	Marc	38	1
30	Aline	18	0.9
5	Jean	56	0

In this proposal, we have changed the syntax of fuzzy queries to be easier for the user. These changes concern the syntax of fuzzy conditions and the name of the queried table, with no requirement for specifying the fuzzy view's name. The new syntax is:

$$\begin{aligned} &\textbf{select} \;\; \textbf{[distinct]} \;\; < attributes > \\ &\textbf{from} \;\; < tables > \\ &\textbf{where} \;\; (attribut\_name =' fuzzy\ term') \\ &\qquad\qquad comparison\_operator \;\; threshold \end{aligned} \tag{2}$$

For example, the previous query will be expressed as:

$$\begin{aligned} &\textbf{select} \;\; emp \;\; \textbf{from} \;\; EMP \\ &\textbf{where} \;\; (age =' young') > 0 \end{aligned} \tag{3}$$

So, due to these changes, the fuzzy query will not be processed directly by the DB. This is the role of the fuzzy conditions processing module; this involves changing the syntax of the fuzzy conditions to the supported one, as well as the name of the queried table.

2) Semantic queries: To perform semantic queries, a semantic predicate must be created. This involves associating an attribute in the database with an ontology that represents its domain, similar to the preparation of fuzzy queries.

It is worth mentioning that the chosen ontology must contain the semantic term that will be searched in the database, but not all database values for this attribute. For example, suppose we want to search semantically for "Action movies" in the content of the movie_genre attribute. In that case, it is not necessary to include all movie genres in the ontology, but only those we are interested in.

a) Semantic condition process module: The semantic condition process module consists of converting a semantic condition into a Boolean one by executing an algorithm that searches the database content the terms that are semantically similar to the searched term with the desired threshold based on the domain ontology associated with the queried attribute and chosen semantic similarity measure. And then return a list of these terms that will be used to build a simple SQL IN condition.

In our system, we opted to use a recent ontology-based semantic similarity measure library, HESML V1R5 [22], [23] to calculate the semantic similarity measure between two terms. Is a scalable and real-time semantic measures library that includes several of the most significant semantic similarity measures, supports importing ontology file formats such as OWL or RDF files, and implements the most significant biomedical ontologies, such as MeSH, SNOMED-CT, and GO (for more detail, and support, we refer to the HESML web site[1]).

Via a user-friendly interface, a user can easily create a reference on an existing ontology (WordNet, YAGO, Go, etc.) or on an ontology file (OWL, RDF(S)), then create his semantic predicate by associating this reference with an attribute of a selected table and a chosen ontology-based semantic similarity measure. All this information will be organized and stored in the Semantic Metaknowledge Base (DFC), which is envisaged as an extension of the catalog of the system (Data Dictionary).

To evaluate the database content with the searched term and its relationship in the ontology, Algorithm 1 is implemented in the semantic condition process module to return a set of found terms. This algorithm takes as parameters this searched term, threshold, a comparison operator, and the information contained in the DFC concerning this queried attribute.

3) Flexible queries: The flexible queries module is responsible for receiving and analyzing a query, extracting query conditions, and sending each non-exact condition to the corresponding module as depicted in Fig. 4. This module is implemented in Java and establishes a database connection to access the catalog of the DB system.

[1] http://hesml.lsi.uned.es/

Algorithm 1 Semantic Similarity Search

Input: $tableId, attributeId, evalvalue, [threshold], [cndOpr], ontologyType, ontologyRef, SMeasureId$
Output: $ResSet : set$
Data: $ResSet : set(value)$, $AttrValSet : set(value)$, $SMC : SMComp$,
$RestSet \leftarrow$ ""
AttrValSet ← get_Attr_Vals(tableId, attributeId)
SMC ← SMComp(ontologyType, ontologyRef, SMeasureId)
for $value \in AttrValSet$ **do**
SM_degree ← SMC.getSM_degree(evalvalue, value)
if $Comparison(cnOpr, sm\_degree, threshold)$ **then**
$add(ResSet, value)$
end if
end for

Also, this module is responsible for building an ordinary query after replacing each non-exact condition with its corresponding Boolean one in the original query and getting execution results. The input processing of this module identifies inexact conditions just from its syntax, which will be presented in the next section. However, to distinguish between semantic and fuzzy conditions, a search in the database catalog is required.

When a query involves multiple conditions, the calculation of the final degree of satisfaction is performed only if the user specifies it in the query. Each returned row includes the calculated degree of accomplishment using Zadeh's set of operations, which involves Union (Max) and Intersection (Min):

- Union: $\mu_{A \cup B}(t) = max(\mu_A(t), \mu_B(t))$
- Intersection: $\mu_{A \cap B}(t) = min(\mu_A(t), \mu_B(t))$

Note that, after processing, each condition attribute is assigned a membership or similarity degree based on its type, as indicated in Table II.

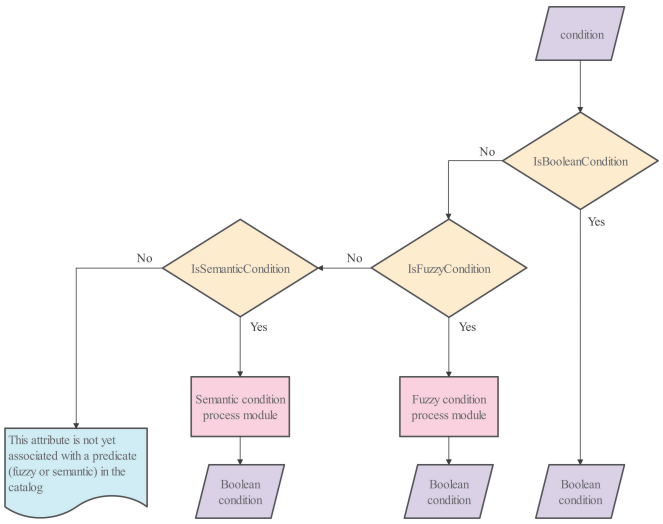


Fig. 4. The General flow of control in the flexible query module.

4) SQLf extension: The SQLf syntax has been slightly modified. We have changed the syntax of fuzzy queries to make it more user-friendly and to support new features. The new syntax is as follows:

$$\begin{aligned} &\textbf{select} \quad \textbf{[distinct]} \quad < attributes > \\ &\textbf{from} \quad < tables > \\ &\textbf{where} \quad (attribut\_name =' fuzzy\ or\ semantic\ term') \\ &\qquad\qquad comparison\_operator\ threshold \end{aligned} \tag{4}$$

The semantic query syntax is the same as that of the fuzzy query. If a query is performed on semantic attributes, such as "Return action movies", this query would be expressed as:

$$\begin{aligned} &\textbf{select} \quad movieid \quad \textbf{from} \quad movie \\ &\textbf{where} \quad (genre =' Action') > 0 \end{aligned} \tag{5}$$

where genre attribute has associated a movie ontology that includes the term "Action" in it. We mention that the user can use a comparison operator, e.g. $=, >, <, \geq, or \leq$.

The latter must be enclosed within parentheses to accelerate input processing time and facilitate clear differentiation between Boolean and non-Boolean conditions. For example, the query "Return the new action movies whose budget is over 30 million $" would be expressed as:

$$\begin{aligned} &\textbf{select} \quad movieid \quad \textbf{from} \quad movie \\ &\textbf{where} \quad (ReleaseDate =' new') > 0 \\ &\textbf{and} \ (genre =' Action') > 0 \ \textbf{and} \ budget > 30 \end{aligned} \tag{6}$$

IV. Experimental Results and Discussion

Though the proposed system has been previously presented, we understand the need to support that by showing the system's behavior and its performance through some experiments in different datasets. Accordingly, this section describes the test environment, the experiments conducted, the results reported, and the necessary validation. The first experiment is proposed to analyze in detail the system behavior, and to measure the efficiency and the scalability of the system, two additional experiments were carried out that vary according to the ontology type and size. Moreover, quantitative comparison and qualitative one are included to highlight the strengths and drawbacks of our contribution.

A. Testing Environment

The developed flexible relational database query system can be used as a front-end to any relational database, provided it is prepared beforehand. This is true because all the developed procedures can be adapted to other DBMSs. Also, semantic query processing is implemented in application clients, so it is independent of DBMS. We used Oracle Database 19c Enterprise as a database server in the testing. The reason to use Oracle is that it is the most widely used commercial relational database management system. It processes data faster and takes up less space [24].

A Java web application has been developed to facilitate tasks for users and to compute all the operations that are not provided by the database. Technical details are shown in Table IV.

TABLE IV. TECHNICAL DETAILS OF THE DEVELOPED SYSTEM

Database Server	*Flexible Queries Application*
Intel(R) Core(TM) i7-4790k CPU @ 4GHz	intel(R) Core(TM) i5-2.40Ghz
Mem 16GB	Mem 8GB
CentOS release 7	Windows 10 Pro
Oracle Database 19c	Java 8 (JEE)

B. Experimentation #1: Movie Ontology

In this experimentation, we employed a small movie database to thoroughly analyze the system's behavior when executing a flexible query. An example of data is shown in Table V. The flexible query "Return all the new action movies" requires the definition of two attribute domains in the database:

- ReleaseDate: A virtual column named "ElapsedPeriod" has been calculated and added to the table, representing the elapsed years since the movie was released. This attribute has been associated with a fuzzy set named "New", represented by the membership function shown in Fig. 5.
- Genre: This attribute represents the genre of the movie, and we have associated it with an ontology about movie genres[2].

TABLE V. EXAMPLE OF DATA

MovieID	Title	Genre	ReleaseDate
1	Avatar	SciFi_and_Fantasy	17/12/2009
2	Titanic	Love	18/11/1997
3	The jungle Book	Family	7/04/2016
4	The Hobbit	Kids	14/09/2012
5	APOLLO 11	Documentarial_Information	01/03/2019
6	Furious 7	Entertainment	02/04/2015
7	Spectre	Action	26/10/2015
8	The Tomorrow War	War	02/07/2021
9	Monsters University	Animation	20/06/2013
10	Cinderella	Romance	12/12/2015
11	Alone	Zombie	16/09/2020
...	...	...	...

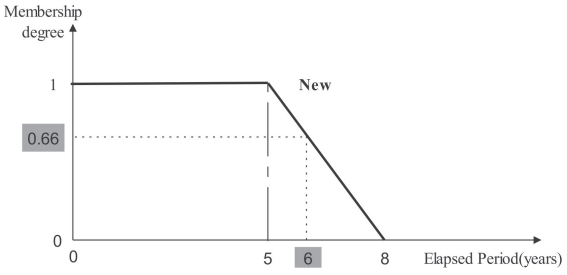


Fig. 5. Calculation process of "New Movies" membership degree.

The query introduced in the system will be expressed in SQLf as:

$$\begin{aligned} &\textbf{select} \ * \ \textbf{from} \ movie \\ &\textbf{where} \ \ (ReleaseDate =' new') > 0 \\ &\textbf{and} \ (Genre =' Action') > 0 \end{aligned} \tag{7}$$

[2]https://www.kaggle.com/datasets/hijest/genre-classification-dataset-imdb

TABLE VI. A PREVIEW OF FMOVIE FUZZY VIEW

MovieID	...	ReleaseDate	...	ReleaseDate.new
1	...	17/12/2009	...	0
2	...	18/11/1997	...	0
3	...	7/04/2016	...	0.66
4	...	14/09/2012	...	0
5	...	01/03/2019	..	1
6	...	02/04/2015	...	0.33
7	...	26/10/2015	...	0.33
8	...	02/07/2021	...	1
9	...	20/06/2013	...	0
10	...	12/12/2015	...	0.33
11	...	16/09/2020	...	1
...	...	...	...	...

TABLE VII. SOME CALCULATED SEMANTIC SIMILARITY DEGREES WITH THE "ACTION" TERM

MovieID	Genre	Similarity degree	MovieID	Genre	Similarity degree
1	SciFi_and_Fantasy	0.43	7	Action	1
2	Love	0.42	8	War	0.9
3	Family	0.41	9	Animation	0.41
4	Kids	0.42	10	Romance	0.40
5	Documentarial_Information	0.26	11	Zombie	0
6	Entertainment	0.58	...	...	...

The database searching process starts when the query is analyzed and converted to SQL. The FQ process module analyses this query and extracts query conditions to send each one of them to the corresponding process module. There are two conditions in this query:

- The first one is a fuzzy condition that will be sent to the FzzCm module to just change their syntax to "$ReleaseDate.new$" > 0, which is a Boolean condition that will be applied to a virtual column named "ReleaseDate.new" of a previously generated fuzzy view associated with the movie table. This virtual column contains the membership degree of each release date to the "New Movie" fuzzy set shown in Fig. 5. A preview of the generated fuzzy view is represented in Table VI.
- The second one is a semantic condition that is sent to the SemCM module. The semantic value "Action" will be compared semantically with all values of the Genre column using a pre-selected semantic similarity measure. In this experimentation, we have used the semantic similarity measure proposed by Sanchez et al. [25]. That relies on the exploitation of taxonomical features. This measure is efficient, easy to calculate, and can be used in a variety of ontologies.

The Table VII shows some calculated semantic similarity degrees. The obtained similarity degree is equal to 1 only if the searched value "Action" matches with a database register and 0 if the searched value is not in the ontology (e.g. Movie 11 (Zombie)). Otherwise, the degrees of similarity decrease according to the distance between the searched value and the race of the movie found in the queried column content. It should be noted that for better results, the threshold must be initialized. Based on these obtained results, SemCM generates an SQL condition like:

genre **IN** ('SciFi_and_Fantasy','Love','Family', 'Kids', 'Documentarial_Information','Entertainment',

'Action','War', 'Animation','Romance','Adventure', 'News','Biography')

As you can see, we have obtained all the movie genres except those not included in the ontology, as we haven't set a threshold in this semantic condition.

Both original conditions will be replaced by their corresponding Boolean equivalents in the original query. Consequently, the resulting flexible query will be expressed in SQL as:

select * **from** movie
where "ReleaseDate.new"¿0
and genre **IN** ('SciFi_and_Fantasy','Love','Family', 'Kids', 'Documentarial_Information','Entertainment', 'Action','War', 'Animation','Romance','Adventure', 'News','Biography')

Resulting data are shown in Table VIII.

TABLE VIII. RESULTS OF QUERY: "RETURN ALL THE NEW ACTION MOVIES"

MovieID	Title	Genre	ReleaseDate	Accomp. deg.	
3	The jungle Book	Family	7/04/2016	0.41	
5	APOLLO 11	Documentarial_Information	01/03/2019	0.26	
6	Furious 7	Entertainment	02/04/2015	0.33	
7	Spectre	Action	26/10/2015	0.33	
8	The Tomorrow War	War	02/07/2021	0.9	
10	Cinderella	Romance	12/12/2015	0.33	
...	...	...	...	...	...

C. Experimentation #2: Book Ontology

Various tests to measure the system's efficiency were conducted on a database of books. A partial example of it, with fuzzy data definitions, is shown in Fig. 6. For brevity, we will describe only the attributes utilized in our experimentation:

- Bookage: This is a NUMBER-type attribute that characterizes the age of the book. It is calculated from the book's publish date and expressed in years. The associated fuzzy sets ([new, classic, vintage]) are illustrated in Fig. 6.
- Genre: is a VARCHAR type attribute that describes the book genre. His semantic definition is represented by a Book ontology[3] employed by the Canadian Writing Research Collaboratory to assign genres to different types of cultural objects. The similarity is estimated using Lin's measure [26].

We have used a dataset containing 25 variables and 52478 records that have been collected in the frame of the Prac1 of the subject Topology and Data Life in the Universitat Oberta of Catalunya.[4] To analyze the efficiency of our system, we have varied two parameters: query complexity from simple to more complex (see Table IX), and to analyze the system scalability we have varied the number of tuples computed on the same query (see Table X). The results have been illustrated in Fig. 7.

[3]https://sparql.cwrc.ca/ontologies/genre-2020-07-14.html
[4]https://zenodo.org/record/4265096

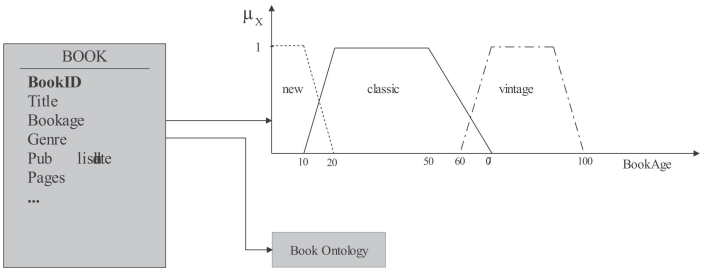
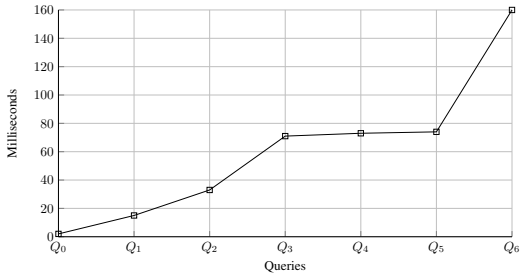
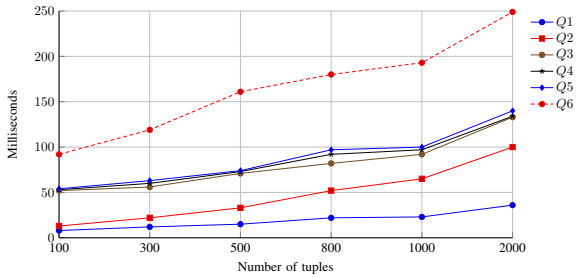


Fig. 6. Partial example of book database with fuzzy data definitions.



(a) Execution times (500 tuples)



(b) Escalability in execution times

Fig. 7. Execution times of the performed queries.

D. Experimentation #3: Medical Subject Headings Ontology (MeSH)

In this experiment, we propose an example of the use of our system in the medical field. A medical database about characteristics of patients likely to be infected by the coronavirus has been flexibly queried to select patients with symptoms of a type of coronavirus.We have used MeSH[5] ontology version 2024 to define a semantic definition of virus type. MeSH is a thesaurus produced by the National Library of Medicine that is used for indexing, cataloging, and searching for biomedical and health-related information. Table XI presents a set of different queries that have been designed to measure the performance of the system.

E. Discussion

To evaluate the system's efficiency and demonstrate its adaptability, we conducted a series of tests by varying the complexity of the queries, changing the number of tuples computed on the same query, and varying the ontology.

[5]https://www.nlm.nih.gov/databases/download/mesh.html

TABLE IX. SET OF FLEXIBLE QUERIES

ID	*Query*	*SQLf*	*Rows*	*Time(ms)*
1	Find the new books	**select** bookid **from** book **where** (bookage="new")> 0.5	72	15
2	Find new or vintage books	**select** bookid **from** book **where** (bookage="new")> 0.5 **or** (bookage="vintage")> 0.5	228	33
3	Find the historical books	**select** bookid **from** book **where** (genre="historical")> 0.5	194	71
4	Find the fictional books	**select** bookid **from** book **where** (genre="fictional")> 0.5	177	73
5	Find the historical vintage books	**select** bookid **from** book **where** (bookage="vintage")> 0.5 **and** (genre="historical")> 0.5	63	74
6	Find the historical or fictional books	**select** bookid **from** book **where** (genre="historical")> 0.5 **or** (genre="fictional")> 0.5	371	161
0	Reference query to measure network latency	**select** bookid **from** book	500	2

TABLE X. EXECUTION TIMES VARYING NUMBER OF TUPLES

N. tuples	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
100	8	13	52	50	52	92
300	12	22	56	60	63	119
500	15	33	71	73	74	161
800	22	52	82	92	97	180
1000	23	65	92	97	100	193
2000	36	100	133	132	140	249

We can see in Fig. 7a in experimentation 2 how the execution time increases as the number of attributes or the query complexity increases. Also, we can see in Fig. 7b how the scalability increases depending on the number of computed rows in complex queries. However, varying the number of rows is insufficient to distinguish the delay caused by network latency or changes in the number of rows computed.

In addition, we can see from these experiments how the execution time increases when we use a large ontology as in experimentation 3 in which we used the MeSH thesaurus that contains several concepts. Consequently, the system performance degrades depending on the size of the chosen ontology. So, since the user is the one who chooses the ontology, then it does not have to be very large to get good results.

F. Comparison to the Other Approaches

A qualitative comparison between the most relevant characteristics in fuzzy query systems of relational databases in the literature and our proposal is shown in Table XII. We conclude that none of the proposals in the literature is complete. Most of them give a partial version of the representation and processing of imprecise information. Our approach appears to be most complete for fuzzy querying classical relational databases because it supports most features.

Many query systems use ontologies to perform flexible queries on relational databases, cf., e.g. [33], [34], [35]. Most of these systems need a preprocessing stage that executes a mapping process between the ontology and the RDB, contrary to our system, which only needs to establish the relationship between the ontology and the queried attribute. However, this strategy could lead us to decrease the performance when the chosen ontology is large. The ontology size must be small to get good results. In Table XIII, we consider the systems that combine fuzzy and semantic queries. In this context, we refer to a recent system proposed by Martinez [29] that performs flexible queries on a fuzzy relational database using fuzzy logic and ontology. This proposal does not support all types of ontologies and implements only one semantic similarity measure, unlike our system, which implements several semantic similarity measures and supports most types of ontologies.

A quantitative comparison between our system and the system proposed by Martinez is shown in Fig. 8. We considered experimentation 2 with a dataset of 2000 tuples. The graph clearly shows that our system outperforms the Martinez system in terms of response speed by a substantial margin. This is due to his principle that consists of splitting the flexible query into subqueries and includes an aggregation phase that needs to store the temporal results of each process module to combine them at last, which induces an important overhead. Moreover, this proposal is a modification of Medina's one [17] that requires browsing a large fuzzy catalog to translate fuzzy query.

Our system implements the SQLf language, utilizing fuzzy views to manage the satisfaction degrees of user-defined fuzzy predicates, rather than calculating them at runtime through user functions embedded in the query. Therefore, it doesn't need an analyzer or a translator because all fuzzy queries generated will be compatible with standard SQL, which leads to significant performance.

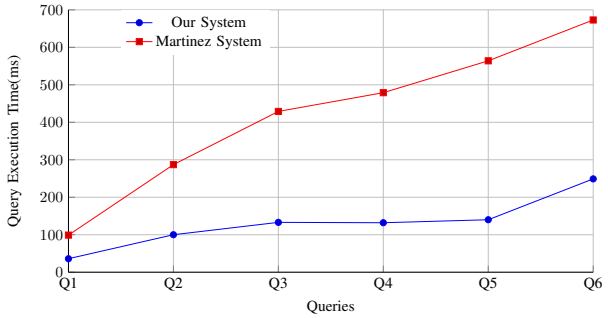


Fig. 8. Execution times comparison between our and Martinez's proposed system.

TABLE XI. SET OF FLEXIBLE QUERIES

ID	*Query*	*SQLf*	*Rows*	*Time* (ms)
1	Find the old patients	**select** patientid **from** patient **where** (age="old")> 0.5	72	17
2	find the old patients that have high respiratory rate	**select** patientid **from** patient **where** (age="old")> 0.5 **and** (respiratory_rate="high")> 0.5	17	10
3	Find the patients that are probably affected by SARS virus	**select** patientid **from** patient **where** (symptom="D045473")> 0.5	500	857
4	find the old patients that have high respiratory rate and are probably affected by SARS virus	**select** patientid **from** patient **where** (age="old")> 0.5 **and** (respiratory_rate="high")> 0.5 **and** (symptom="D045473")> 0.5	17	404
5	Find the patients that have Corona virus symptoms	**select** patientid **from** patient **where** (symptom="D017934")> 0.5	500	655
6	Find the tired patients that have high respiratory rate, dry cough, loss of taste, loss of smell, and are probably affected by the Corona virus	**select** patientid **from** patient **where** (tiredness="tired")> 0.5 **and** (respiratory_rate="high")> 0.5 **and** (dry_cough="yes")> 0.5 **and** (loss_of_taste="yes")> 0.5 **and** (loss_of_smell="yes")> 0.5 **and** (symptom="D017934")> 0.5	72	441
0	Reference query to measure network latency	**select** patientid **from** patient	500	4

TABLE XII. COMPARISON OF MOST RELEVANT CHARACTERISTIC IN FUZZY QUERY SYSTEMS

Model	Medina [17]	Bosc [9]	Zemankova [27]	Prade [28]	Martinez [29]	Umano [30]	Kacprzky [31]	Buckles [32]	Our proposal
Manage scalar data	✓	✓	✓	✓	✓	✓	✓	✓	✓
Manage non-scalar data	✓		✓	✓	✓			✓	✓
Similarity relationship	✓		✓		✓			✓	✓
Possibility distributions	✓	✓	✓	✓	✓		✓		✓
Degree in tuple level	✓	✓	✓	✓	✓	✓	✓		✓
Fuzzy modifiers		✓	✓						✓
Fuzzy quantifiers	✓	✓							✓
Fuzzy comparison operators	✓	✓	✓	✓	✓	✓	✓	✓	✓
Fuzzy group by		✓					✓		
Fuzzy joins	✓	✓		✓		✓			✓
Nesting	✓	✓							✓
Store fuzzy data	✓		✓	✓		✓		✓	
Fuzzy queries	✓	✓	✓	✓	✓	✓	✓	✓	✓
Extension SQL	✓	✓		✓	✓	✓			✓
Portability									✓

TABLE XIII. COMPARISON OF MOST RELEVANT FEATURES IN RECENT SEMANTIC QUERY SYSTEMS

Model	Martinez [29]	Our proposal
Query language	✓	✓
Manage fuzziness	✓	✓
Degree of similarity	✓	✓
Enriched query	✓	✓
semantic similarity measures	1	26
Ontology		
MeSH		✓
SNOMED		✓
WordNet		✓
OBO file format		✓
Gene Ontology		✓
OWL file format	✓	✓
RDF triples files		✓

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a flexible relational database query system based on fuzzy logic and ontology that provides a mechanism to perform flexible queries where the criteria are not only exact but also can be fuzzy or semantic, and they may also include an accomplishment degree. One of the main goals of this proposal is to solve the fuzzy logic drawback of handling non-scalar data. We have presented the architecture of this novel system and a detailed description of all methods and algorithms involved in the handling process of flexible queries. As a proof of concept, the proposal has been tested on three different databases and three ontologies with a quantitative study on the behavior and efficiency of the system. We showed that the execution time increases according to the number of tuples, the query's complexity, and the chosen ontology's size. In addition, we have addressed the strengths and drawbacks of our system through a quantitative and qualitative comparison of the most relevant features of flexible query systems in the literature.

Finally, since this approach supports all ontology types and provides a rich set of semantic similarity measures, it can be used in many other fields such as geography, biology, health, and genomics. Most importantly, this approach laid the basis for implementing in an alternative database model a less rigid query frame.

As forthcoming activities, we plan to enrich the flexibility of our system by extending it to support bipolar queries that can accommodate the users' intentions and preferences involving some sort of required and desired, mandatory and optional, etc. conditions. Additionally, our goal is to exploit our approach to develop a natural language interface for relational databases. This interface will allow users to flexibly query and manipulate databases using everyday language rather than requiring them to use formal query languages like SQL.

REFERENCES

[1] T. Andreasen, G. Bordogna, G. De Tré, J. Kacprzyk, H. L. Larsen, and S. Zadrożny, "The power and potentials of flexible query answering systems: A critical and comprehensive analysis," *Data & Knowledge Engineering*, vol. 149, p. 102246, 2024.

[2] P. Córdoba-Hidalgo, N. Marín, and D. Sánchez, "Rl-instances: An alternative to conjunctive fuzzy sets of tuples for flexible querying in relational databases," *Fuzzy Sets and Systems*, vol. 445, pp. 184–206, 2022.

[3] R. Mama, M. Machkour, K. Ahkouk, and K. Majhadi, "Towards a flexible relational database query system," in *Proceedings of the 4th International Conference on Networking, Information Systems & Security*, 2021, pp. 1–5.

[4] R. Mama and M. Machkour, "A study of fuzzy query systems for relational databases," in *Proceedings of the 4th International Conference on Smart City Applications*, 2019, pp. 1–5.

[5] R. Mama, M. Machkour, M. Ennaji, and K. Ahkouk, "Flexible query systems for relational databases," in *The Proceedings of the Third International Conference on Smart City Applications*. Springer, 2020, pp. 1015–1029.

[6] R. Mama and M. Machkour, "Fuzzy questions for relational systems," in *The Proceedings of the Third International Conference on Smart City Applications*. Springer, 2019, pp. 104–114.

[7] J. Kacprzyk, S. Zadrożny, and G. De Tré, "Fuzziness in database management systems: Half a century of developments and future prospects," *Fuzzy Sets and Systems*, vol. 281, pp. 300–307, Dec. 2015.

[8] K. Min, H. Jananthan, and J. Kepner, "Fuzzy relational databases via associative arrays," *2023 IEEE MIT Undergraduate Research Technology Conference (URTC)*, pp. 1–5, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:265043040

[9] P. Bosc and O. Pivert, "Sqlf: a relational database language for fuzzy querying," *IEEE transactions on Fuzzy Systems*, vol. 3, no. 1, pp. 1–17, 1995.

[10] M. Goncalves and L. Tineo, "Sqlf flexible querying language extension by means of the norm sql2," in *10th IEEE International Conference on Fuzzy Systems.(Cat. No. 01CH37297)*, vol. 1. IEEE, 2001, pp. 473–476.

[11] A. Motro, "Vague: A user interface to relational databases that permits vague queries," *ACM Transactions on Information Systems (TOIS)*, vol. 6, no. 3, pp. 187–214, 1988.

[12] J. Eduardo, M. Goncalves, and L. Tineo, "A fuzzy querying system based on sqlf2 and sqlf3," in *The XXX Latin-American Conference on Informatics*, 2004.

[13] B. Samuel, "Interrogation floue de bases de données: extension de isqlf," *Rapport de projet, laboratoire lannionais d'informatique, ENSSAT LANNION*, 2005.

[14] E. González, R. Rodríguez, and L. Tineo, "Prototipo experimental para consultas difusas," 2012.

[15] G. Smits, O. Pivert, and T. Girault, "Reqflex: fuzzy queries for everyone," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1206–1209, 2013.

[16] A. Aguilera, J. T. Cadenas, and L. Tineo, "Fuzzy querying capability at core of a rdbms," in *Advanced Database Query Systems: Techniques, Applications and Technologies*. IGI Global, 2011, pp. 160–184.

[17] J. M. Medina, O. Pons, and M. A. Vila, "Gefred: A generalized model of fuzzy relational databases," *Information sciences*, vol. 76, no. 1-2, pp. 87–109, 1994.

[18] J. Medina, O. Pons, and A. Vila, "First. a fuzzy interface for relational systems," in *VI International Fuzzy Systems Association World Congress (IFSA 1995). Sao Paulo (Brasil)*, 1995.

[19] M. Umano, "Freedom-0: a fuzzy database system," in *Readings in Fuzzy Sets for Intelligent Systems*. Elsevier, 1993, pp. 667–675.

[20] J. Galindo, J. M. Medina, O. Pons, and J. C. Cubero, "A server for fuzzy sql queries," in *International Conference on Flexible Query Answering Systems*. Springer, 1998, pp. 164–174.

[21] R. Mama and M. Machkour, "Fuzzy querying with sql: Fuzzy view-based approach," *Journal of Intelligent & Fuzzy Systems*, no. Preprint, pp. 1–12, 2021.

[22] J. J. Lastra-Díaz, A. García-Serrano, M. Batet, M. Fernández, and F. Chirigati, "Hesml: A scalable ontology-based semantic similarity measures library with a set of reproducible experiments and a replication dataset," *Information Systems*, vol. 66, pp. 97–118, 2017.

[23] J. J. Lastra-Díaz, A. Lara-Clares, and A. Garcia-Serrano, "Hesml: a real-time semantic measures library for the biomedical domain with a reproducible survey," *BMC bioinformatics*, vol. 23, no. 1, pp. 1–31, 2022.

[24] "Relational database ranking." [Online]. Available: http://db-engines.com/en/ranking

[25] D. Sánchez, M. Batet, D. Isern, and A. Valls, "Ontology-based semantic similarity: A new feature-based approach," *Expert systems with applications*, vol. 39, no. 9, pp. 7718–7728, 2012.

[26] D. Lin *et al.*, "An information-theoretic definition of similarity." in *Icml*, vol. 98, no. 1998, 1998, pp. 296–304.

[27] M. Zemankova and A. Kandel, "Implementing imprecision in information systems," *Information Sciences*, vol. 37, no. 1-3, pp. 107–141, 1985.

[28] H. Prade and C. Testemale, "Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries," *Information sciences*, vol. 34, no. 2, pp. 115–143, 1984.

[29] C. Martínez-Cruz, J. M. Noguera, and M. A. Vila, "Flexible queries on relational databases using fuzzy logic and ontologies," *Information Sciences*, vol. 366, pp. 150–164, Oct. 2016.

[30] M. Umano, S. Fukami, M. Mizumoto, and K. Tanaka, "Retrieval processing from fuzzy databases," *Preprints of Working Group of IEICE of Japan*, vol. 80, no. 204, pp. 45–54, 1980.

[31] J. Kacprzyk and S. Zadrozny, "Sqlf and fquery for access," in *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, vol. 4. IEEE, 2001, pp. 2464–2469.

[32] B. P. Buckles and F. E. Petry, "A fuzzy representation of data for relational databases," *Fuzzy sets and Systems*, vol. 7, no. 3, pp. 213–226, 1982.

[33] J. Zhang, Z. Peng, S. Wang, and H. Nie, "Si-seeker: Ontology-based semantic search over databases," in *International Conference on Knowledge Science, Engineering and Management*. Springer, 2006, pp. 599–611.

[34] N. Konstantinou, D.-E. Spanos, M. Chalas, E. Solidakis, and N. Mitrou, "Visavis: An approach to an intermediate layer between ontologies and relational database contents." *WISM*, vol. 239, 2006.

[35] C. B. Necib and J.-C. Freytag, "Ontology based query processing in database management systems," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2003, pp. 839–857.