TECHNICAL TRANSACTIONS | CZASOPISMO TECHNICZNE
ELECTRICAL ENGINEERING | ELEKTROTECHNIKA
3-E/2016

DOI: 10.4467/2353737XCT.16.277.6076

GRZEGORZ NOWAKOWSKI*

CONVERSION OF FUZZY QUERIES INTO STANDARD SQL QUERIES USING ORACLE 11G XE

TRANSFORMACJA ZAPYTAŃ NIEPRECYZYJNYCH NA ZAPYTANIA W STANDARDZIE SQL PRZY WYKORZYSTANIU ORACLE 11G XE

Abstract

This article presents various forms of fuzzy queries with a particular emphasis of two different approaches to the representation of fuzziness, a detailed analysis of these queries and their conversion into standard SQL queries using Oracle 11g XE. The actions discussed above point out to the methods of obtaining fuzzy information from the database that have been easy to implement. A qualitative and quantitative study about the use of fuzzy queries on relational databases has been included in this article, as well. This research takes into account the fact that obtaining this type of information is not supported by any commercial database management system.

Keywords: fuzzy logic, fuzzy queries, SQL, Oracle 11g XE

Streszczenie

W artykule przedstawiono różne formy zapytań nieprecyzyjnych do bazy danych ze szczególnym uwzględnieniem dwóch konkretnych podejść do reprezentacji nieprecyzyjności, dokonano szczegółowej analizy tych zapytań oraz ich transformacji na zapytania w standardzie SQL z zastosowaniem Oracle 11g XE. W artykule ujęto również jakościowe i ilościowe badanie dotyczące wykorzystania nieprecyzyjnych zapytań w relacyjnych bazach danych. Omawiane działania wskazują na łatwe w implementacji sposoby pozyskiwania nieprecyzyjnych informacji z bazy danych oraz uwzględniają fakt, że pozyskiwanie tego typu informacji nie jest wspierane przez żaden komercyjny system zarządzania bazami danych.

Słowa kluczowe: logika rozmyta, zapytania nieprecyzyjne, SQL, ORACLE 11g XE

* M.Sc. Eng. Grzegorz Nowakowski, Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology.

1. Introduction

Classic query languages (with SQL being the most widely applied) define the scope of data as well as conditions to be met by data throughout the entire process of searching for information in databases. These conditions should be precisely defined, as precision is the basic requirement when defining conditions in query languages. At the same time, these conditions impose limitations. For example, a customer of a car dealership looking for a cheap car has to precisely define the price range he is willing to pay. Regardless of how the limits of the range are defined, a car priced slightly above the set limit will not meet the conditions of the query. This shows that the limits in question result from the necessity of precisely defining the conditions, which were initially expressed in a natural language with imprecise terms.

The problem can be solved by the use of linguistic terms modelled and processed with fuzzy logic in queries addressed to a database. These are the so-called fuzzy queries. Linguistic terms are presented as fuzzy sets in an appropriate, usually numerical, space. Therefore, matching data to the query is no longer perceived as dichotomy: matched – unmatched. For this reason, the notion of a degree of matching the data to the query is introduced and one assumes that the value of this degree of matching corresponds (in a somewhat simplified view) to the degree to which the data belongs to the fuzzy set representing the condition of the query [1, 2].

This article discusses various forms of fuzzy queries with a particular emphasis of two different approaches to the representation of fuzziness; it also includes a detailed analysis of fuzzy queries as well as documents their conversion into SQL standard queries using Oracle 11g XE. Described actions point to easy to implement methods of obtaining fuzzy information from databases and take into account the fact that obtaining such information is not supported by any commercial database management system.

2. Fuzzy queries

A fuzzy query addressed to a relational database [1, 2] is a query including overtly used expressions of the natural language, referred to as linguistic terms (modelled with fuzzy logic), defining the following: imprecise values, imprecise comparisons and non-standard methods of aggregation of degrees of meeting partial conditions of the query.

Particular rows meet the conditions of such a query to a certain degree expressed with a number from <0,1> range, where 1 means that the query requirement is fully met, and 0 means that the query requirement is fully unmet. Therefore, the result of the query is a set of rows arranged according to the degree of meeting the conditions of the query. This way, it is easier to reflect the attempts undertaken by a given individual to match the data to the query. In the case of complex queries, which include both fuzzy linguistic terms as well as non-standard aggregation schemes, meeting conditions of the query is gradual. In such a situation, a human being naturally evaluates the data and perceives it as data, which meets his/her requirements to a greater or a lesser degree instead of distinguishing only between the data, which meets or does not meet the set requirements. The notion of the degree of meeting

conditions of the query allows us to formally present these complex queries and by the same token naturally arrange the results in such a manner, as to make sure the data, which meets the conditions of the query to the greatest degree is at the onset of the list of results.

When analysing the problem of calculating the degree to which conditions of a fuzzy query are met, one should accept a significantly simplified view of the execution of such a query by a database system. It is assumed that when a query is executed, the whole table is examined sequentially and that the degree to which the query is met is calculated for each row. Database system carries out the steps presented in Table 1.

Table 1

Calculation of the degree to which conditions of a fuzzy query are met (prepared based on [1, 2])

Step	Database system:
1	downloads a row from the table;
2	calculates partial degrees of meeting all (or only selected, depending on the structure of the query and the applied optimization) simple conditions of the query by substituting the attributes with their values form a given row;
3	aggregates partial degrees to which conditions of the query are met as calculated in the previous step up to the total degree of meeting conditions the query;
4	if the total degree of meeting conditions of the query is sufficiently high (exceeds the user-defined or default threshold value), the row is added to the reply to the query along with the degree of meeting its conditions;
5	moves to step 1;
6	if there are no more rows, STOP.

An overview of the process of calculating the degree to which conditions of a simple, fuzzy query are met can be presented on the following example. Let us assume that I am looking for a cheap car in the offer of a car dealership. Let us assume that u denotes the numerical range specifying prices of the vehicles (in kPLN) and that the term *cheap* is modelled with the fuzzy set A characterized by the following membership function:

$$\mu_A(u) = \begin{cases} 1 & u \le 200 \\ \dfrac{400-u}{200} & 200 < u \le 400 \\ 0 & u > 400 \end{cases} \tag{1}$$

Then, for a car represented by tuple t and characterized by the price *t.price*, the degree of meeting conditions of the above query *md* is calculated as follows: $md = \mu_A(t.price)$. The provided example illustrates the general rule of interpretation of fuzzy queries: the degree to which conditions of the query are met is equated with the value of the membership function of the relevant fuzzy set. In the case of complex queries, partial degrees of meeting

conditions of the query calculated in the above-mentioned manner, corresponding to particular conditions contained in the query, are aggregated with selected operators.

As it was mentioned above, the possibility of non-standard aggregation of conditions contained in the query is a significant feature of fuzzy queries. This feature significantly extends the classic scheme based predominantly on the use of conjunction and alternative as well as allows expressing often complex interrelations among partial conditions of the query. Among aggregation operators, linguistic quantifiers are particularly important as they are widely applied in the natural language and well represented in fuzzy logic.

3. Conversion of fuzzy queries into SQL standard queries using Oracle 11g XE

Construction of fuzzy queries as well as their execution and the applied grammar of fuzzy queries are usually strongly connected with the query language of a given database. This item discusses the problem of conversion of fuzzy queries into SQL standard queries using Oracle 11g XE database, in the case of which, similarly to most contemporary relational databases [6], SQL query language is used.

There are numerous fuzzy elements in queries addressed to databases:

- atomic predicates based on linguistic terms: young, tall,
- atomic predicates based on similarity of linguistic terms: information technology ≈ artificial intelligence,
- complex predicates: fuzzy sum and product operators,
- modified predicates: very, around, rather, antonyms,
- fuzzy operators: approximately, a little more, etc.,
- linguistic quantifiers,
- fuzzy combination of relations,
- fuzzy aggregate functions,
- grouping by fuzzy values.

In order to discuss the problem of approach towards the representation of imprecision, examples of imprecise elements in queries addressed to database contained in the first three items of the above list were taken into account.

Two approaches towards the representation of imprecision can be distinguished [11]:

- *based on distribution of possibilities*

The approach based on the distribution of possibilities changes the method of representation of values of attributes from numerical to linguistic. Linguistic variables (also referred to as *possibility* type variables) can be presumed for attributes with a continuous domain. Values of such a variable are linguistic terms with corresponding fuzzy sets represented in the form of distribution of possibilities.

Example 1: on the *prod_year* attribute, whose domain is originally numerical, one can introduce the following linguistic terms: *New, Average, Old*, which are defined by relevant distributions of possibilities.

Example 2: the condition: ‘new car’ will be formulated conventionally as: *prod_year is NEW*, where *NEW is* a fuzzy set corresponding to the linguistic term *NEW* for the attribute *prod_year.*

- based on similarity

Here, values of linguistic variables are also linguistic terms, yet they are interpreted as distributions of similarity (the so-called similarity matrix is defined which stores the degree of similarity for each pair of linguistic terms). Such variables can be defined exclusively for attributes with a finite domain.

From the above reasoning, it stems that the whole interpretation of fuzzy queries comes down to the following general rule: the degree of meeting conditions of the query equates with the value of the membership function of the relevant fuzzy set. Whereas in the case of complex queries, partial degrees of meeting conditions of the query calculated in the above-mentioned manner, corresponding to particular conditions contained in the query, are aggregated with selected operators discussed in item 3.3.

3.1. Exemplary data

Assuming there are already exemplary tables with data in Oracle 10g XE database. Another column, i.e. *fuzzy_degree*, with the initially set value of 1.0 for each row, which belongs to it, was added to the table labelled *tbl_cars* presented in Fig. 1. This denotes the initial state – total degree of membership of a given row to the table labelled *tbl_cars*.

```
SELECT * FROM tbl_cars;
+--------+--------------+--------+-----------+-----------------+--------+---------------+
| ID_SAM | MODEL        | TYPE   | PROD_YEAR | ENGINE_CAPACITY | PRICE  | FUZZY_DEGREE  |
+--------+--------------+--------+-----------+-----------------+--------+---------------+
|      1 | VOLVO        | SEDAN  |      2010 |             1.6 |  35500 |  1.000000000  |
|      2 | ASTON MARTIN | COUPE  |      2015 |             6.0 | 900000 |  1.000000000  |
|      3 | MAZDA        | CABRIO |      2012 |             2.0 |  43990 |  1.000000000  |
|      4 | OPEL         | COMBI  |      1993 |             1.7 |   1500 |  1.000000000  |
|      5 | VOLKSWAGEN   | SEDAN  |      2007 |             1.9 |  23900 |  1.000000000  |
|      6 | AUDI         | COUPE  |      2005 |             1.9 |  21000 |  1.000000000  |
+--------+--------------+--------+-----------+-----------------+--------+---------------+
```

Fig. 1. Exemplary data from *tbl_cars* table

Tables *tbl_possibility* and *tbl_similarity* contain data, which will be used at the stage of conversion of fuzzy queries into typical SQL queries. The *tbl_possibility* table, as presented in Fig. 2, stores linguistic variables described by the membership function.

```
SELECT * FROM tbl_possibility;
+----------+-----------+---------------+--------------------+------+------+------+------+-----------+
| username | table_name| column_name   | linguistic_variable| a    | b    | c    | d    | type      |
+----------+-----------+---------------+--------------------+------+------+------+------+-----------+
| user     | tbl_cars  | prod_year     | NEW                | 2010 | 2015 | 2050 | 2050 | trapezoid |
| user     | tbl_cars  | prod_year     | AVERAGE            | 1995 | 2000 | 2010 | 2015 | trapezoid |
| user     | tbl_cars  | prod_year     | OLD                |    0 |    0 | 1995 | 2000 | trapezoid |
| user     | tbl_cars  | engine_capacity | SMALL            |    0 |    0 |    0 |    1 | trapezoid |
| user     | tbl_cars  | engine_capacity | MIDDLE           |    0 |    1 |  1.6 |  2.5 | trapezoid |
| user     | tbl_cars  | engine_capacity | HIGH             |  1.6 |  2.5 |    5 |    8 | trapezoid |
+----------+-----------+---------------+--------------------+------+------+------+------+-----------+
```

Fig. 2. Exemplary data from *tbl_possibility* table

The *linguistic_variable* column stores the name of the linguistic variable. The *type* column contains information on the type of the membership function describing the linguistic variable (in the table in question all variables are described by the trapezoidal membership function presented in Fig. 3). Columns *a*, *b*, *c* and *d* contain parameters of the function. Columns *table_name* and *column_name* store the following information: on which

column and from which table the linguistic variable was presumed. The column labelled *username* contains information on the name of the user to which a given definition applies; this way various users can describe linguistic variables with identical names, presumed for the same columns of the same tables, in a different way.

$$\mu_A(x;a,b,c,d) = \begin{cases} 0 & x \le a \\ \dfrac{x-a}{b-a} & a < x \le b \\ 1 & b < x \le c \\ \dfrac{d-x}{d-c} & c < x \le d \\ 0 & x > d \end{cases}$$

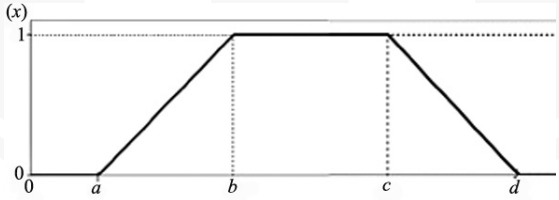

Fig. 3. Trapezoidal membership function

The table labelled *tbl_similarity* presented in Fig. 4 stores pairs of linguistic values along with the degree of their similarity. The data pertains to the column describing the type of bodywork of the car (Table 2).

```
SELECT * FROM tbl_similarity
WHERE username = 'user' AND table_name = 'tbl_cars' AND column_name = 'type'
ORDER BY value_1, value_2;
```

username	table_name	column_name	value_1	value_2	degree
user	tbl_cars	type	cabrio	cabrio	1
user	tbl_cars	type	cabrio	combi	0.2
user	tbl_cars	type	cabrio	coupe	0.2
user	tbl_cars	type	cabrio	sedan	0.2
user	tbl_cars	type	combi	cabrio	0.2
user	tbl_cars	type	combi	combi	1
user	tbl_cars	type	combi	coupe	0.6
user	tbl_cars	type	combi	sedan	0.6
user	tbl_cars	type	coupe	cabrio	0.2
user	tbl_cars	type	coupe	combi	0.6
user	tbl_cars	type	coupe	coupe	1
user	tbl_cars	type	coupe	sedan	0.8
user	tbl_cars	type	sedan	cabrio	0.2
user	tbl_cars	type	sedan	combi	0.6
user	tbl_cars	type	sedan	coupe	0.8
user	tbl_cars	type	sedan	sedan	1

Fig. 4. Exemplary data from *tbl_similarity* table

Columns *value_1*, *value_2* contain the original value and the similar value, whereas column *degree* stores the degree of similarity between the two elements. Interpretation of the remaining columns is similar as in the case of their counterparts in the table labelled *tbl_possibility*.

It needs pointing out that similarity can be presumed for columns with a discrete domain. Additionally, when defining the similarity relation one should remember that the relation must be reflexive, symmetric and transitive.

Table 2

Similarity relation for *tbl_cars* table on *type* attribute

	cabrio	combi	coupe	sedan
cabrio	1.0	0.2	0.2	0.2
combi	0.2	1.0	0.6	0.6
coupe	0.2	0.6	1.0	0.8
sedan	0.2	0.6	0.8	1.0

3.2. Fuzzy queries – fuzzy condition for possibility type variable

In a classic SQL query, the query condition can be either met or unmet. Each result row meets the conditions of the query to the same degree. In fuzzy relations, a row can appear partially, which can be interpreted as certainty or a degree of meeting conditions of the query.

A fuzzy query converted to an equivalent SQL standard query using Oracle 11g XE, as presented in Table 3, will display the offer of *new* cars. A fuzzy condition was applied in the query and a defined fuzzy set was used, which corresponds to the term *new* for the *prod_year* attribute.

Table 3

Example: display new cars for sale

Fuzzy query (conventional form)	**An equivalent query in SQL-Oracle 11g XE**
SELECT * **FROM** tbl_cars **WHERE** prod_year IS **'NEW'**;	**SELECT** c.*, FP_TRAPEZOID(prod_year,a,b,c,d) FUZZY_DEGREE **FROM** tbl_cars c, tbl_possibility **WHERE** FP_TRAPEZOID(prod_year,a,b,c,d)>0 and (linguistic_variable=**'NEW'** and column_name='prod_year');

In the discussed example, all variables are defined by the trapezoidal membership function, the definition of which (presented in Fig. 5) was formulated in PL/SQL in Oracle 10g XE database.

In Fig. 6, the column labelled *fuzzy_degree* contains the information on the degree to which a given row meets conditions of the query. In this case, it tells us to what degree the car is *new*. It is calculated as $\mu_{prod\_yearNEW}(prod\_year)$, and its definition is presented in Tab. 2.

```
create or replace FUNCTION
FP_TRAPEZOID(x IN NUMBER, a IN NUMBER, b IN NUMBER, c IN NUMBER, d IN NUMBER)
RETURN NUMBER IS
y number(6,5);
BEGIN
IF (x <= a) THEN y:= 0.0;
ELSIF (x > a AND x <= b) THEN y:= (x - a)/(b - a);
ELSIF (x > b AND x <= c) THEN y:= 1.0;
ELSIF (x > c AND x <= d) THEN y:= (d - x)/(d - c);
ELSE y:= 0.0;
END IF;
  RETURN Y;
END FP_TRAPEZOID;
```

Fig. 5. Trapezoidal membership function defined in PL/SQL in Oracle 10g XE database

```
SELECT c.*, FP_TRAPEZOID(prod_year,a,b,c,d) FUZZY_DEGREE
FROM tbl_cars, tbl_possibility
WHERE FP_TRAPEZOID(prod_year,a,b,c,d)>0 and (linguistic_variable='NEW' and column_name='prod_year');

+--------+--------------+--------+-----------+-----------------+--------+--------------+
| ID_SAM | MODEL        | TYPE   | PROD_YEAR | ENGINE_CAPACITY | PRICE  | FUZZY_DEGREE |
+--------+--------------+--------+-----------+-----------------+--------+--------------+
|      2 | ASTON MARTIN | COUPE  |      2015 |             6.0 | 900000 |  1.000000000 |
|      3 | MAZDA        | CABRIO |      2012 |             2.0 |  43990 |  0.400000000 |
+--------+--------------+--------+-----------+-----------------+--------+--------------+
```

Fig. 6. Result of the fuzzy query from Table 2 – fuzzy condition for possibility type variable

The parameters and the type of the membership function of the defined linguistic term can be easily read from *tbl_possibility* table presented in Fig. 2. This has been presented in Fig. 7.

```
SELECT linguistic_variable, a,b,c,d, type
FROM tbl_possibility
WHERE username='user'and table_name='tbl_cars' and column_name='prod_year'
ORDER BY a,b,c,d;

+--------------------+------+------+------+------+-----------+
| linguistic_variable | a    | b    | c    | d    | type      |
+--------------------+------+------+------+------+-----------+
| NEW                | 2010 | 2015 | 2050 | 2050 | trapezoid |
| AVERAGE            | 1995 | 2000 | 2010 | 2015 | trapezoid |
| OLD                |    0 |    0 | 1995 | 2000 | trapezoid |
+--------------------+------+------+------+------+-----------+
```

Fig. 7. Result of the query – parameters and type of the membership function of the defined linguistic term

As indicated by the result of the query shown in Fig. 7, the fuzzy set $PROD\_YEAR_{NEW}$=(2010, 2015, 2050, 2050) corresponds to the term *new* for the attribute *prod_year*.

3.3. Fuzzy query – combining fuzzy conditions

Fuzzy queries, similarly to classic SQL queries, can contain complex conditions resulting from combining single conditions with logical operators. In SQL, the keyword AND corresponds to the conjunction operator, the keyword OR corresponds to the alternative operator and the keyword NOT to the negation operator.

Identical keywords are applied in the case of fuzzy queries, yet they correspond, respectively, to: fuzzy conjunction, fuzzy alternative and fuzzy negation.

Nevertheless, it needs pointing out that in the theory of fuzzy sets [7], there is a number of operators carrying out intersection operation (product operation which corresponds to the logical operation AND). These are applied interchangeably depending on the problem at hand. Most of these operators meet the criteria of the so-called triangular norm *T* (*T*-norm):

$$\mu_{A\cap B}(x) = T(\mu_A(x), \mu_B(x)) \tag{2}$$

Similarly to the execution of the intersection operation, numerous operators are used for the purpose of the operation of joining (logical sum which corresponds to the logical operation OR). The most commonly applied operators meet the criteria of the so-called triangular norm *S* (*S*-norm) also referred to as *T*-conorm:

$$\mu_A \cup_B (x) = S(\mu_A(x), \mu_B(x)) \tag{3}$$

The criteria defining triangular norms consist of four fundamental conditions and due to a limited length of this article will not be presented here. These conditions were described, among others, in [1, 2, 8, 9].

The most commonly used T-norms (mapping logical operator AND) are the minimum MIN (*a*, *b*) and the product (PROD) $a \cdot b$. Whereas the most commonly used *S*-norm operators (mapping logical operator OR) are the maximum MAX (*a*, *b*) and the so-called algebraic (probabilistic) sum $a + b - a \cdot b$.

A relevant *S*-norm corresponds to each *T*-norm, provided that the following condition is met:

$$T(a, b) = 1 - S(1 - a, 1 - b) \text{ or } S(a, b) = 1 - T(1 - a, 1 - b) \tag{4}$$

Operators that meet the condition (4) form the so-called complementary (conjugate, dual) pairs. Numerous operators meeting the conditions of *T*-norms and *S*-norms have been developed and described. These operators are divided into non-adjustable, with a constant mode of operation, and adjustable (parametrized), also referred to as families of triangle norms, in the case of which the mode of operation changes depending on the accepted parameter (the degree of freedom) for the operator in question. Table 4 presents selected triangle norms forming complementary pairs.

Operators of triangle norms indicated in Table 4 execute operations only on two fuzzy (variable) sets. Operations on a greater number of sets can be executed gradually, by combining sets into pairs with the sequence of combining sets into pairs having no effect on the result (coherency quality).

From the above reasoning, it results that the degree of meeting a complex condition is calculated based on the degrees of meeting partial conditions as well as selected *T*-norm and its complementary *S*-norm.

A fuzzy query converted to an equivalent SQL standard query using Oracle 11g XE, as presented in Table 5, will display the offer of *new* cars with *high* engine cubic capacity. All variables are defined by the trapezoidal membership function, the definition of which (presented in Fig. 5) was formulated in PL/SQL in Oracle 10g XE database. In the query in question:

- a defined fuzzy set corresponding to the term *new* for the attribute *prod_year* was applied,
- a defined fuzzy set corresponding to the term *high* for the attribute *engine_capacity* was applied,
- a conjunction of both of the above-mentioned fuzzy conditions was carried out, which resulted in defining *T*-norm as the minimum operation.

Table 4

Selected non-adjustable triangle norms forming complementary pairs (prepared based on [7–10])

No.	$T(a, b)$	$S(a, b)$
1	Minimum MIN (a, b)	Maximum MAX (a, b)
2	PROD algebraic product $a \cdot b$	Algebraic sum (probabilistic product) $a + b - a \cdot b$
3	Limited difference (Łukasiewicz's) MAX (0, $a + b - 1$)	Limited sum (Łukasiewicz's) MIN (1, $a + b$)
4	Drastic product $\begin{cases} \min(a,b), & if\ \max(a,b) = 1 \\ 0, & otherwise \end{cases}$	Drastic sum $\begin{cases} \max(a,b), & if\ \min(a,b) = 0 \\ 1, & otherwise \end{cases}$
5	Hamacher product $\begin{cases} \frac{a \cdot b}{a + b - a \cdot b} \end{cases}$	Hamacher sum $\begin{cases} \frac{a + b - 2 \cdot a \cdot b}{1 - a \cdot b} \end{cases}$
6	Einstein product $\begin{cases} \frac{a \cdot b}{2 - (a + b - a \cdot b)} \end{cases}$	Einstein sum $\begin{cases} \frac{a + b}{1 + a \cdot b} \end{cases}$

Table 5

Example: display *new* cars for sale with *high* engine capacity (first way)

Fuzzy query (conventional form)	An equivalent query in SQL – Oracle 11g XE
SELECT * **FROM** tbl_cars **WHERE** prod_year IS **'NEW'** **AND** engine_capacity IS **'HIGH'**	**SELECT *** **FROM** (**SELECT** c.model, c.type, c.prod_year, c.engine_capacity, MIN(CASE column_name WHEN 'prod_year' THEN FP_TRAPEZOID(prod_year,a,b,c,d) WHEN 'engine_capacity' THEN FP_TRAPEZOID(engine_capacity,a,b,c,d) END) AS FUZZY_DEGREE **FROM** tbl_cars c, tbl_possibility **WHERE** (linguistic_variable=**'NEW'** or linguistic_variable=**'HIGH'**) AND (column_name='prod_year' or column_name='engine_capacity') group by c.model, c.type, c.prod_year, c.engine_capacity) **WHERE** FUZZY_DEGREE > 0;

Table 6

Example: display *new* cars for sale with *high* engine capacity (second way)

Fuzzy query (conventional form)	An equivalent query in SQL – Oracle 11g XE
SELECT* **FROM** tbl_cars **WHERE** prod_year IS **'NEW'** **AND** engine_capacity IS **'HIGH'**	**SELECT** c.model, c.type, c.prod_year, c.engine_capacity, MIN(CASE column_name WHEN 'prod_year' THEN FP_ TRAPEZOID(prod_year,a,b,c,d) WHEN 'engine_capacity' THEN FP_ TRAPEZOID(engine_capacity,a,b,c,d) END) AS FUZZY_DEGREE **FROM** tbl_cars c, tbl_possibility **WHERE** (linguistic_variable=**'NEW'** or linguistic_variable=**'HIGH'**) AND (column_name='prod_year' or column_name='engine_capacity') group by c.model, c.type, c.prod_year, c.engine_capacity **having** MIN(CASE column_name WHEN ,prod_year' THEN FP_ TRAPEZOID(prod_year,a,b,c,d) WHEN ,engine_capacity' THEN FP_ TRAPEZOID(engine_capacity,a,b,c,d) END) >0;

The value of the column labelled *fuzzy_degree*, the result of which was presented in Fig. 8, was set as $\min(\mu_{prod\_yearNEW}(prod\_year), \mu_{engine\_capacityHIGH}(engine\_capacity))$, the definition of which can be found in Table 5.

```
SELECT *
FROM (
  SELECT c.model, c.type, c.prod_year, c.engine_capacity,
         MIN(CASE column_name
             WHEN 'prod_year' THEN FP_TRAPEZOID(prod_year,a,b,c,d)
             WHEN 'engine_capacity' THEN FP_TRAPEZOID(engine_capacity,a,b,c,d)
             END) AS FUZZY_DEGREE

  FROM tbl_cars c, tbl_possibility
  WHERE (linguistic_variable='NEW' or linguistic_variable='HIGH') AND
        (column_name='prod_year' or column_name='engine_capacity')
  group by c.model, c.type, c.prod_year, c.engine_capacity
)
where FUZZY_DEGREE > 0;

+--------------+--------+-----------+-----------------+--------+--------------+
| MODEL        | TYPE   | PROD_YEAR | ENGINE_CAPACITY | PRICE  | fuzzy_degree |
+--------------+--------+-----------+-----------------+--------+--------------+
| ASTON MARTIN | COUPE  |      2015 |             6.0 | 900000 |  0.666666667 |
| MAZDA        | CABRIO |      2012 |             2.0 |  43990 |  0.400000000 |
+--------------+--------+-----------+-----------------+--------+--------------+
```

Fig. 8. Result of the fuzzy query from Table 5 – joining fuzzy conditions

3.4. Fuzzy queries – fuzzy condition for a similarity type variable

As it was already mentioned in item 3.1, for attributes with a discrete domain, in the case of which similarity between pairs of elements of the domain was defined, a similarity type fuzzy condition can be used. In the exemplary database, a similarity relation was defined on the *type* attribute of *tbl_cars* table (Table 2).

A fuzzy query converted into an equivalent SQL standard query using Oracle 11g XE, as presented in Table 7, will display cars with bodywork similar to coupe. Additionally, the cut-off threshold was set to 0.7.

Table 7

Example: display cars with bodywork *similar* to *coupe*. Set cut-of threshold to 0.7

Fuzzy query (conventional form)	An equivalent query in SQL-Oracle 11g XE
SELECT THRESHOLD **0.7** c.* **FROM** tbl_cars c **WHERE** type IS **'COUPE'**;	**SELECT** c.model, c.type, c.prod_year, c.engine_capacity, c.price, s.degree **FROM** tbl_cars c, tbl_similarity s **WHERE** (c.type, s.degree) in (**SELECT** value_2,degree **FROM** tbl_similarity **WHERE** value_1=**'COUPE'** and degree > **0.7**) **GROUP BY** c.model, c.type, c.prod_year, c.engine_capacity, c.price, s.degree **ORDER BY** s.degree

As a result of the query, we received four offers presented in Fig. 9. The column labelled *fuzzy_degree* contains the information on the degree to which a given row meets conditions of the query. Due to the set cut-off threshold, only values greater than 0.7 have been included.

```
SELECT c.model, c.type, c.prod_year, c.engine_capacity, c.price, s.degree fuzzy_degree
FROM tbl_cars c, tbl_similarity s WHERE (c.type, s.degree) in
(
 SELECT value_2,degree
 FROM tbl_similarity
 WHERE value_1='COUPE' and degree > 0.7
)
GROUP BY c.model, c.type, c.prod_year, c.engine_capacity, c.price, s.degree
ORDER BY s.degree
```

MODEL	TYPE	PROD_YEAR	ENGINE_CAPACITY	PRICE	fuzzy_degree
VOLKSWAGEN	SEDAN	2007	1.9	23900	0.800000000
VOLVO	SEDAN	2010	1.6	35500	0.800000000
ASTON MARTIN	COUPE	2015	6.0	900000	1.000000000
AUDI	COUPE	2005	1.9	21000	1.000000000

Fig. 9. Result of the fuzzy query from Table 7 – fuzzy condition for a similarity type variable

Needless to say, a fuzzy query can contain complex conditions resulting from joining single conditions with logical operators according to the rule presented in item 3.3.

3.5. Tests

Two different tests on the Oracle 11g XE database have been performed to analyze proposal presented in this article [19]:

– varying complexity of the query (examples of queries presented in this article),
– varying the number of tuples computed on a same query, to analyze the system scalability.

Tests have been conducted on a PC equipped with an Intel Core i5 2.80 GHz and 12 GB of memory. Queries that vary in complexity, are represented in Table 8 together with a description of them.

Table 8

Set of queries

Query ID	Fuzzy Query	Description
*Q*1	*new* cars for sale	fuzzy condition for possibility type variable
*Q*2	*new* cars for sale with *high* engine capacity (first way)	combining fuzzy conditions
*Q*3	*new* cars for sale with *high* engine capacity (second way)	combining fuzzy conditions
*Q*4	cars with bodywork *similar* to *coupe*	fuzzy condition for a similarity type variable

Table 9

Execution times (in milliseconds) varying number of tuples

Number of tuples	*Q*1	*Q*2	*Q*3	*Q*4
10	3	3	13	4
50	8	3	13	4
100	12	4	13	4
200	21	6	14	4
300	22	6	14	3
400	36	9	18	3
500	46	9	19	5
1000	54	13	22	8

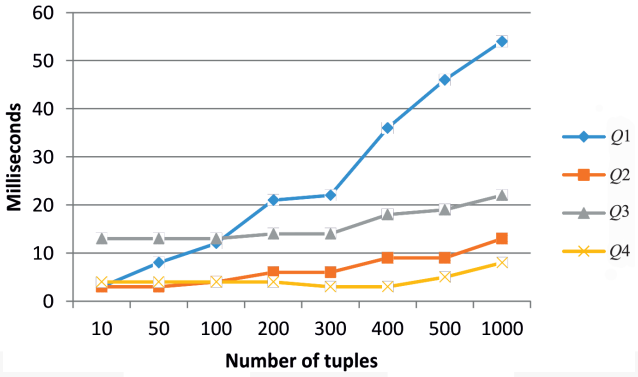

Fig. 10. Execution times in queries performed on Oracle 11g XE

Scalability was measured by varying the number of tuples in the database from 10 to 1000. To do that, examples of queries presented in this article have been executed. Execution times are shown in Table 9 and they have been illustrated in Fig. 10. In this figures, we can notice how the scalability grows accordingly with the number of computed rows in the queries. The ones that have more computational needs and/or have more complex syntax (e.g. this same example of a query (in Table 9 marked as Q2, Q3) are presented in Table 5 and Table 6 has different execution times). However, it is noticeable that varying the number of rows is not enough to distinguish the delay provoked by the changes in the number of computed rows, especially in the simplest queries [19].

4. Comparison of most relevant features in fuzzy query systems [19]

A comparison between the features of the main fuzzy relational databases in the literature [19] and proposal presented in this article is shown in Table 10. First models were mainly theoretical proposals of fuzzy relational databases. Prade H. and Testemale C. [13] have presented the original code in MACLISP on DPS8 for fuzzy query processing. Umano [12] and Fukami have presented FOOBD in SQL. The most complete implementations were provided by: Bosc P. and Pivert O. [17] called Sqlf and Kacprzyk J. and Zadrożny S. [18] called FQuery in Microsoft Access. In addition, Medina et al. [15] proposed a conceptual framework for fuzzy representation called GEFRED (Generalized Model for Fuzzy Relational Databases) and a language called FSQL (Fuzzy SQL, SQL extension) in Oracle. An implementation presented in this article is based on a Kacprzyk J. and Zadrożny S. and Medina's proposal (Table 10). It is worth noting that there are a few functionalities, that GEFRED has defined theoretically [19], that neither have been included in the implemented version, i.e. fuzzy joins and fuzzy quantifiers, nor in implementation presented in this article.

Table 10

Comparison of most relevant features in fuzzy query systems [19]

Model	Buckles B.P., Petry F.E. [12]	Prade H., Testemale C. [13]	Zemankova M., Kandel A. [14]	Medina J.M., Pons O., Vila M.A. [15]	Umano M., Hatono I., Tamura H. [16]	Bosc P., Pivert O. [17]	Kacprzyk J., Zadrożny S. [18]	Martinez-Cruz C., Noguera J.M., Vila M.A. [19]	Proposal presented in this article
Manage scalar data	x	x	x	x	x	x	x	x	x
Manage non-scalar data	x	x	x	x				x	
Similarity relationship	x		x	x				x	x
Possibility distributions		x	x	x		x	x	x	x
Degree in attributes level	x	x		x	x			x	
Degree in tuple level		x	x	x	x	x	x	x	x
Fuzzy modifiers			x			x			
Fuzzy quantifiers				x		x	x		
Fuzzy comparison operators	x	x	x	x	x	x	x	x	x
Fuzzy group by						x	x		
Fuzzy joins		x		x	x	x			
Store fuzzy data	x	x	x	x	x				
Fuzzy queries	x	x	x	x	x	x	x	x	x
Extension SQL language				x	x	x	x	x	x

5. Conclusions

Fuzzy queries enable using a natural language. Nevertheless, in order to maintain the fuzzy nature of these expressions, they are modelled with fuzzy sets. This way, fuzzy queries enable improved representation of the requirements of the user through direct expression of the same with linguistic terms and through the use of complex methods of aggregation of partial conditions.

Fuzzy queries can be applied even if the user precisely defined his/her requirements. Yet, their application is only justified in the event there is no data, which meets these requirements. When a classic query with precisely defined conditions yields an empty data set, a fuzzy query with imprecisely defined conditions may yield a data set, which is not empty. Some of the obtained results (with the highest degree of meeting conditions of the query) may be accepted by the user. This way, the user will have a better chance of learning the content of the database and consequently will have the opportunity to modify the query. A modified query may take into account the content of the database and may better reflect the actual requirements of the user. The following conclusion can be drawn: as fuzzy queries arrange results according to the degree of meeting conditions of the query, it is easier to analyze the results and the risk of obtaining an empty reply is reduced thanks to an extended interpretation of the conditions of the query.

The provided examples of conversion of fuzzy queries into SQL standard queries using Oracle 11g XE point to easy to implement methods of obtaining fuzzy information from the database and by the same token expand its functionality. Moreover, a qualitative comparison between the most relevant fuzzy query systems in the literature and proposal presented in this article has addressed the strengths and drawbacks of this contribution [19].

Acknowledgment

Presented results of the research, which was carried out under the theme No. E-3/627/2016/DS, were funded by the subsidies on science granted by Polish Ministry of Science and Higher Education.

References

[1] Zadrożny S., *Zapytania nieprecyzyjne i lingwistyczne podsumowania baz danych*, Akademicka Oficyna Wydawnicza, Warszawa 2006.

[2] Myszkorowski K., Zadrożny S., Szczepaniak P.S., *Klasyczne i rozmyte bazy danych: modele, zapytania i podsumowania*, Warszawa 2008.

[3] Zadrożny S., Kacprzyk J., *Bipolar Queries Using Various Interpretations of Logical Connectives*, Foundations of Fuzzy Logic and Soft Computing, 2007.

[4] Mesiar R., Thiele H., *On T-Quantifiers and S-Quantifiers*, [In:] Novak V., Perfilieva I. (eds.), *Discovering the World with Fuzzy Logic*, Physica-Verlag, Heidelberg 2000, 310–326.

[5] Novak V., Perfilieva I., *Fuzzy Logic on the Basis of Classical Logic*, Kacprzyk J., Krawczak M., Zadrożny S. (red.), Issues in Information Technology, EXIT, 2002.

[6] Nowakowski G., *Open source relational databases and their capabilities in constructing a web-based system designed to support the functioning of a health clinic*, Technical Transactions, vol. 1-AC/2013, 53–65.
[7] Baczyński D., Bielecki S., Parol M., Piotrowski P., Wasilewski J., *Sztuczna inteligencja w praktyce*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2008.
[8] Rutkowska D., Piliński M., Rutkowski L., *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte,* Wydawnictwo Naukowe PWN, Warszawa 1997.
[9] Yager R., Filev D., *Podstawy modelowania i sterowania rozmytego*, WNT, Warszawa 1995.
[10] Czogała E., Pedrycz W., *Elementy i metody teorii zbiorów rozmytych*, Wydawnictwo Naukowe PWN, Warszawa 1985.
[11] Rozmyte zapytania do baz danych, (online) homepage: http://www.cs.put.poznan.pl/mhapke/LR-Rozmyte%20zapytania%20do%20baz%2 0danych%20II.pdf (date of access: 2016-06-30).
[12] Buckles B.P., Petry F.E., *A fuzzy representation of data for relational databases*, Fuzzy Sets and Systems, Vol. 7, Issue 3, May 1982, 213–226.
[13] Prade H., Testemale C., *Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries*, Information Sciences, Vol. 34, Issue 2, November 1984, 115–143
[14] Zemankova M., Kandel A.: *Implementing imprecision in information systems*, Information Sciences, Volume 37, Issue 1–3, Dec. 1985, 107–141
[15] Medina J.M., Pons O., Vila M.A., *GEFRED. A Generalized Model of Fuzzy Relational Databases*, Information Sciences, vol. 76(1–2), 1994, 87–109
[16] Umano M., Hatono I., Tamura H.: *Fuzzy database systems*, Proceedings of the FUZZ-IEEE/IFES'95 Workshop on Fuzzy Database Systems and Information Retrieval, Yokohama, Japan, 1995, 53–36
[17] Bosc P., Pivert O., *SQLf: a relational database language for fuzzy querying*. IEEE transactions on Fuzzy Systems 3 (1), 1995, 1–17.
[18] Kacprzyk J., Zadrożny S.: *SQLf and FQUERY for Access*, Proceedings of the IFSA World Congress and 20th NAFIPS International Conference, vol. 4, 2001, 2464–2469.
[19] Martinez-Cruz C., Noguera J.M., Vila M.A., *Flexible queries on relational databases using fuzzy logic and ontologies*, Information Sciences, Vol. 366, 20 October 2016, 150–164.