# Chapter 2
# Fuzzy Queries

**Abstract** The goal of database queries is to separate relevant tuples from non-relevant ones. The common way to realize such a query is to formulate a logical condition. In classical queries, we use crisp conditions to describe tuples we are looking for. According to the condition, a relational database management system returns a list of records. However, user's preferences in what should be retrieved, are often vague or imprecise. These preferences can be expressed in atomic conditions and/or between them. For example, the meaning of a query: *find municipalities with small population density and altitude about 1000 m above sea level* can be understood at the first glance. The linguistic terms clearly suggest that there is a smooth transition between acceptable and unacceptable records. This chapter is focused on the construction of fuzzy sets, the aggregations functions and the issues of fuzzy logic in queries which should not be attenuated.

## 2.1 From Crisp to Fuzzy Queries

In order to work with the main topic of this chapter, brief introduction to relational databases and relational algebra is desirable. Relational databases are further examined in Chap. 5.

A relational database consists of relations (tables). We should emphasize that tables and relations are synonyms. The table is a suitable representation of relation. The relation schema has the following structure [38]:

$$R(A_1 : D_1, ..., A_n : D_n), \tag{2.1}$$

where $R$ is the name of relation, e.g. MUNICIPALITY or CUSTOMER (in order to keep notation consistent throughout the book, relations in database are written with capital letters), $A_i$ is the $i$-th attribute ($i = 1, ..., n$), often called column (e.g. *unemployment rate*) and $D_i$ is the domain of attribute $A_i$ defining a set of all possible values which could be assigned to records (e.g. interval [0, 100] for the above mentioned attribute). In case of attributes like *sales* the domain is set of positive real numbers. A record is represented by row and often called tuple. A relation instance

of a given relation schema is a set of tuples stored in a table. Hence, the term "relation instance" is abbreviated to relation or table. Each tuple $r_j (j = 1, ..., m)$ consists of the attributes' values in the following way:

$$r_j = \left\{ (d_{1j}, ..., d_{nj}) \mid (d_{1j} \in D_1, ..., d_{nj} \in D_n) \right\}, \tag{2.2}$$

where $d_{ij}$ is the value of the tuple $r_j$ corresponding to the attribute $A_i$. The letter $t$ is usually used in literature to express database tuple. In this book letter $t$ is used for t-norms. In order to avoid any misunderstanding, t-norm is marked with letter $t$ and database tuple with letter $r$ (record) throughout the book.

To clarify the understanding of relations to level required for queries, let us show one illustrative example.

*Example 2.1* The relation MUNICIPALITY(#id, name, number_of_inhabitants, area, altitude, pollution) is represented in Table 2.1.

For instance, third tuple is expressed as vector (2.2):
$r_3 = \{(3, Mun3, 810, 1030, 625, 0.20)\}.$ ☐

A query against a collection of data stored in database provides a formal description of the tuples of interest to user posing this query [30]. The Structured Query Language (SQL) is a standard query language for relational databases [13]. SQL was initially introduced in [11]. Since then, SQL has been used in many relational databases for managing data (insert, modify, delete, retrieve). The use of SQL may be regarded as one of the major reasons for the success of relational databases in the commercial world [45].

SQL has the following basic structure:

SELECT  [distinct]$\langle attributes \rangle$ FROM $\langle relations \rangle$ WHERE $\langle condition \rangle$   (2.3)

In the traditional (crisp) SQL condition a tuple can either fully satisfy the intent of a query $Q_c$, or not. Other options do not exist. In the set theory we can express set of crisp answers in the following way:

$$A_{Q_c} = \{(r, \varphi(r)) \mid r \in R \wedge \varphi(r) = 1\}, \tag{2.4}$$

where $\varphi(r) = 1$ indicates that the selected tuple $r$ meets the query criterion and $R$ states for the queried relation. Therefore, it is not necessary to write the answer as an ordered pair (tuple, matching degree).

*Example 2.2* An example of crisp SQL query is:

SELECT name
FROM municipality
WHERE altitude $< 250$ and pollution $> 20$.

This query returns two municipalities (Mun 2 and Mun 3) from the relation shown in Table 2.1.

**Table 2.1** Relation MUNICIPALITY in a database

| #Id[a] | Name | Number_of_inhab. | Area | Altitude | Pollution |
|---|---|---|---|---|---|
| 1 | Mun 1 | 1550 | 2536 | 251 | 19.93 |
| 2 | Mun 2 | 1790 | 7995 | 248 | 20.50 |
| 3 | Mun 3 | 810 | 1030 | 625 | 0.20 |
| 4 | Mun 4 | 5810 | 8030 | 126 | 60.00 |

[a]# represents the primary key, i.e. attribute(s) which unambiguously identify tuple

The result of the query is shown in graphical mode in Fig. 2.1. Values 20 and 250 delimit the space of retrieved data. Small squares stands for municipalities. From the graphical interpretation it is evident that two tuples (circled), although share almost the same values of both attributes, are separated (one is selected, whereas another is not).

□

If SQL is used for solving this problem, the relaxation would have to be done in the following way [12]:

SELECT name
FROM municipality
WHERE pollution $> 20 - l_1$ and altitude $< 250 + l_2$

where parameters $l_1$ and $l_2$ are used to expand the initial query condition in order to select records that almost meet the query condition. However, this approach has two disadvantages [12]. First, the meaning of the initial query is diluted (e.g. instead of $altitude < 250$ we have $altitude < 260$, if $l_1 = 10$) in order to capture adjacent tuples situated just beyond the border of the initial query. The meaning of the query is
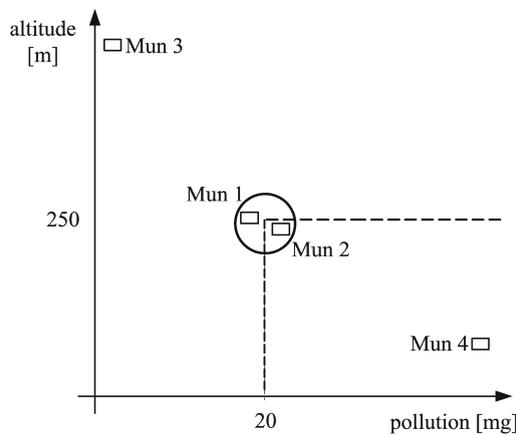


**Fig. 2.1** Crisp query condition: WHERE altitude $< 250$ and pollution $> 20$

changed and the adjacent tuples satisfy the condition in the same way as initial ones. Second, a problem arises from the question: what about tuples that are very close to satisfy the new expanded query? Should we add another relaxation of query? In this way more data from the database are selected, but the initial intent of the query is lost [24].

The other option is adjusting the *select* clause in the following way:

SELECT name, altitude, (iif altitude < 250, 1, iif(altitude > 250 and altitude < 260, (260 - altitude) / 10), 0)) as matching degree

In this way, the structure of the *select* clause is a bit complicated, even though only one attribute is relaxed. If further attributes were added, then the clause would become almost illegible, e.g.

SELECT name, altitude, (iif altitude < 250, 1, iif(altitude > 250 and altitude < 260, (260 - altitude)/10),0)) as m1, pollution, iif(pollution > 20, 1, iif(pollution > 17 and pollution < 20, (pollution - 17)/3, 0)) as m2, min(m1, m2) as matching degree

If we would like to use further possibilities of fuzzy logic such as different t-norms or non-linear fuzzy sets, then rewriting the *select* clause is a complicated task which may eventuate in higher occurrence of errors.

SQL is optimized to query relational databases. In this chapter the core of SQL remains intact and the extension is done to fuzzify the imprecise conditions. Adding flexibility to SQL meets requirements for flexible queries and increases effectiveness of the whole querying process.

The main reason for using fuzzy set theory to make querying more flexible is discussed in [15] and advocated in [27]. Firstly, fuzzy sets provide a better description of data requested by user. For example, the meaning of a query: *select customers having high turnover and short payment delay* can be understood at the first glance. Secondly, linguistic terms are clearly suggesting that there is a smooth transition between acceptable and unacceptable tuples. In fuzzy queries, some tuples definitely match the condition, some certainly not and some match to a certain degree.

Similarly to SQL query structure (2.3), the basic structure of fuzzy query is the following [5]:

SELECT [distinct]$\langle attributes \rangle$ FROM $\langle relations \rangle$ WHERE $\langle fuzzy\ condition \rangle$
$$\tag{2.5}$$

Hence, the fuzzy query returns a fuzzy relation (a subrelation of the initial database relation) consisting of set of tuples that satisfy the fuzzy condition and respective matching degrees. The set of answers to fuzzy query $A_{Q_f}$ could be written in the following way:

$$A_{Q_f} = \{(r, \mu(r)) \mid r \in R \land \mu(r) > 0\} \tag{2.6}$$

In fuzzy queries we distinguish preferences inside atomic conditions and between them. The former is expressed by constructing fuzzy sets which correspond to users' needs; the latter is realized by aggregations.

Presumably, the first attempt to fuzzify SQL-like queries is [42]. It fuzzifies the *where* clause in order to make possible to use vague terms similar to natural language. The aggregation was realized by the minimum t-norm (1.47) for the *and* connective and by the maximum s-norm (1.59) for the *or* connective. It was the inspiration for practical realizations of fuzzy queries. One of the first practical realizations of flexible queries is FQUERY, an add-in that extends the MS Access's querying capabilities with the linguistic terms inside the *where* clause [28]. SQLf [5, 6] is a more comprehensive fuzzy extension of SQL queries. SQLf extends SQL by incorporating fuzzy predicates in the *where* clause as well as supports, among others, subqueries inside the fuzzified *where* clause and fuzzy *joins*. The Fuzzy Query Language (FQL) [47] extends SQL queries with fuzzy condition inside the *where* clause in the usual way and adds other two clauses which provide additional functionality: *weight* and *threshold*.

Flexible querying is still an active field for research either in adding further flexibility such as the fuzzification of the *group by* clause [9], conversion from fuzzy to crisp queries [24], dealing with the empty and overabundant answer problems [7, 40] and user-friendly graphical interface [39]. An exhaustive source of flexible preference querying is [34].

## 2.2 Construction of Fuzzy Sets for Flexible Conditions

Generally, there are two main aspects for constructing fuzzy sets. In the first aspect, users define the parameters of each fuzzy set according to their opinion. This way gives them freedom to choose parameters $(a, b, m, m_1, m_2)$ depicted in Figures (1.5, 1.6, 1.7, 1.8, 1.9 and 1.10). Nevertheless, in this approach users are asked to set more crisp values than in classical query (e.g. two crisp values to clarify the meaning of term *small* (1.22), whereas in the classical query user assigns only one value, e.g. *attribute* $A < a$). This problem can be mitigated by adjusting fuzzy sets parameters by moving sliders (on interfaces) over the domain of attribute to set ideal and acceptable values [39] rather than filling input fields with crisp numbers, for example.

Let $D_{min}$ and $D_{max}$ be the lowest and the highest possible domain values of a numeric attribute $A$, i.e. $Dom(A) = [D_{min}, D_{max}]$ and $L$ and $H$ be the lowest and the highest values in the current content of a database, respectively. The domains of attributes should be defined during the database design process in a way that all theoretically possible values can be stored. For the attribute describing the daily frequency of a measured phenomenon during the year (e.g. *number of days with temperature below* $0\,^{\circ}C$), the domain is the [0, 365] interval of integers. In practice, the real values could be far from the $D_{min}$ and $D_{max}$ values; that is, $[L, H] \subseteq [D_{min}, D_{max}]$. This means that only part of the domain contains data. For the attribute describing the number of inhabitants, the domain is theoretically the whole set of

natural numbers ($\mathbb{N}$), but stored values are far from the infinity ($\infty$). If user is not aware of these facts, the query might easily end up as empty or overabundant. This fact should be considered in defining not only query conditions, but also inference rules and data summarizing sentences.

The second aspect deals with the dynamic modelling fuzzy sets parameters over the domains of attributes. In the first step values of $L$ and $H$ are retrieved from a database. These parameters are used to create fuzzy sets for attributes included in the query condition.

The modelling of fuzzy sets parameters depends on the type and purpose of the fuzzy query (dependable or undependable atomic conditions). In this section several approaches are examined. In the next sections suitable ways for particular tasks are discussed.

If collected data are more or less uniformly distributed in the domain, then the uniform domain covering method [44] can be applied. Otherwise, the statistical mean-based method [44] or the logarithmic transformation of respective domains [25] can be used.

An appropriate method could be chosen by users or mined from the data. In the first case, users could rely on their knowledge, common sense, attainments about examined entities (e.g. territorial units or customers), and knowledge of physical laws. Let us demonstrate this aspect on the municipal database. For attributes like the number of inhabitants, the choice may be the logarithmic transformation or statistical mean-based algorithm, because the database contains a few big municipalities and majority of smaller ones. In the case of attributes like water or gas consumption per inhabitant, users could assume that the uniform domain covering method is an appropriate option.

In the second case, per-computation (performing an initial computation before run the main task) of data summary on attributes is used to reveal the information about distribution of data. In this way the single scan of database provides relevant information about relative cardinality [40].

In the book we use the uniform domain covering method. The parameters for the linguistic terms are created by calculating the cores and slopes in the following way [44]:

$$\varepsilon = \frac{1}{8}(H - L) \tag{2.7}$$

$$\theta = \frac{1}{4}(H - L) \tag{2.8}$$

when the linguistic variable is divided into three terms. Thereafter, it is easy to calculate required parameters shown in Fig. 2.2 for a particular fuzzy set. If it is a requirement for more fuzzy sets (e.g. five sets: very small, small, medium, high, very high, Fig. 1.17) these sets can be straightforwardly constructed adjusting $\varepsilon = \frac{1}{14}(H - L)$ and $\theta = \frac{1}{7}(H - L)$.
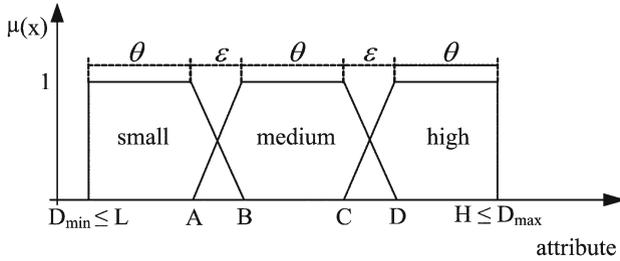
**Fig. 2.2** Linguistic variable uniformly distributed over an attribute domain by three fuzzy sets

Uncertainty of belonging to a set is expressed by $\varepsilon$. The flat segment defined by $\theta$ express clear belonging to a set. If uncertainty decreases, the value of $\varepsilon$ decreases. In the extreme situation, i.e. $\varepsilon = 0$ we got the classical partitioning of the attribute's domain.

*Example 2.3* An institution analysing usage of water is interested to find municipalities which have small water consumption per inhabitants in $cm^3$. For this task the granularity to three fuzzy sets should be sufficient.

Let a simple SQL:

SELECT max(WaterConsumpt) as L, min(WaterConsumpt) as H
FROM municipality

reveals: $L = 0.63$ and $H = 389.22$. Straightforwardly, applying (2.7) and (2.8) only parameters for the fuzzy set *small* are calculated, that is, $A = 97.7775$ and $B = 146.35125$.

Therefore, we got $m_2 = A = 97.7775$ and $b = B = 146.35125$ in order to keep the notation in accordance to the notation in Sect. 1.2.2. Afterwards, user can either modify these parameters or run a query with the suggested ones. In case of the latter, the *where* clause is:

WHERE WaterConsumpt $< 146.35125$.

This condition ensures that all tuples which fully or partially meet the condition are selected for the second step: calculating matching degrees. □

Another approach for constructing fuzzy sets is presented in [27] to support fuzzy queries such as *select municipalities where number of warm days is much lower than number of days with snow coverage*. The intervals $[L, H]$ of each database attribute are transformed into the $[-10, 10]$ interval of real numbers. In the case of condition like *altitude above sea level is small and number of inhabitants is high*, attributes' values can be also transformed into the $[-10, 10]$ interval, if the data distribution is more or less uniform. In this case, fuzzy sets *small*, *medium*, *high* can be created using $\varepsilon = 3.3$ and $\theta = 6.6$.

These linear shapes of membership functions, although very suitable for fuzzy queries, constitute only a small subset of the all possible shapes of membership functions. Discussion about other ones can be found in, e.g. [17, 32].

Both aspects of constructing fuzzy sets parameters can be merged. In the first step, parameters of fuzzy sets are calculated from the current content of database. In the second step, users are able to modify these parameters, if they are not satisfied with the suggested ones. Obviously, the user can obtain tentative information about the stored data before running a query. The querying process could become more tedious, but on the other hand, it might save computation time from running queries which return empty sets. In case when suggested parameters are far from the user's expectations, user has two options: (i) to quit the query process knowing that there is no data that satisfy the query condition; (ii) to accept or slightly modify the suggested parameters. The assistance for constructing fuzzy sets parameters should be optional for users.

The topic of construction of fuzzy sets is covered by vast literature, mainly in fuzzy systems. We should be careful when we consider constructing fuzzy sets for queries, due to the following reasons [20]:

- In fuzzy inference systems fuzzy sets should cover the whole domains of attributes in order to properly control or classify all possible occurrences of input attributes. On the contrary, queries select a subset of data which might be relevant for users.
- If the goal is to develop an easy to use and less demanding tool (a web application for the broad audience, for example), then the fuzzification process should be as simple as possible. It means that, sophisticated approaches like neural networks or genetic algorithms should be avoided.
- In queries, where answer to the first part of query influences adjusting the second part of query we have to rapidly and efficiently construct fuzzy sets for dependable attributes during the querying process.

In addition, the fuzzification step can be improved by mining parameters from the recorded history, if application keeps user's preferences. During the next attempt for similar query, the application offers parameters of fuzzy sets from the recorded user's history. The second option is recommending parameters according to the parameters used by similar users, e.g. if a 22-year-old student from the Eastern Europe searches for a hotel, then the procedure could approximately guess meanings of terms related to the price and room size from other similar students. The main drawback is keeping the user's history and building recommendations. On the other hand, this way is more tailored to users and may attract them. Anyway, finding suitable options for each application should be considered.

## 2.3   Converting Fuzzy Conditions to SQL Ones

If we wish to effectively select tuples, flexible conditions should be converted into SQL ones, because SQL is optimized for efficiently querying relational databases. This task is explained on the approach based on the Generalized Logical Condition

(GLC) [24]. This solution contains the usual steps for fuzzy querying: (i) converting fuzzy conditions to SQL ones; (ii) connecting to database, selecting all candidates (tuples which have membership degree greater than zero) and releasing a database connection; (iii) calculating satisfaction degree for each tuple to each atomic condition; (iv) calculating overall satisfaction degree (often called matching degree). The detailed explanation of this approach can be found in [23].

The GLC has the following structure:

$$where \otimes_{i=1}^{n} A_i \circ L_i, \tag{2.9}$$

where $n$ denotes the number of attributes in fuzzy condition of a query,

$\otimes = \begin{cases} and \\ or \end{cases}$ where *and* and *or* are fuzzy logical operators and

$A_i \circ L_i = \begin{cases} A_i > a, & \text{for condition } high \\ A_i < b, & \text{for condition } small \\ A_i > a \text{ and } A_i < b, & \text{for condition } medium \end{cases}$

where $A_i$ is $i$-th attribute included in the condition, parameters $a$ and $b$ delimit supports of respective fuzzy sets explained in Sect. 1.2.2.

When the compound condition contains several atomic ones connected with the *and* operator, then tuple meets partially or fully the overall condition only, if values of all attributes belong to supports of the respective fuzzy sets.

If a *where* clause contains fuzzy as well as classical conditions, classical ones could be easy added to the *where* clause (2.9) in the following way:

$$where \otimes_{i=1}^{n} A_i \circ L_i \text{ [and/or] } A_e > e \text{ [and/or] } A_f = \text{``string''} \dots \tag{2.10}$$

*Example 2.4*  In this example municipalities with altitude about 900 m above sea, high number of beds in accommodation facilities and small population density are sought.

In the first step each linguistic term is expressed by fuzzy set. Altitude about 900 m above sea is represented by the trapezoidal fuzzy set (1.20) with parameters: $a = 850$, $m_1 = 875, m_2 = 925$ and $b = 950$. High number of beds in accommodation facilities is expressed by the R fuzzy set (1.23) with parameters $a = 450$ and $m = 550$. Finally, small population density is described by the L fuzzy set (1.22) with parameters $m = 120$ and $b = 135$.

Now, we have all information for converting fuzzy into SQL query. The SQL query is:

SELECT name
FROM municipality
WHERE (altitude > 850 and altitude < 950) and beds_accommodation > 450 and population_density < 135.                                                        □

When we have selected all tuples, we can continue with calculating their respective matching degrees.

## 2.4  Calculation of Matching Degrees

Query conditions usually consist of more than one atomic condition merged by logical connectives. Generally, we can divide this topic into two main categories.

- The simpler category is aggregation of query conditions which have the same significance for users and are independent, i.e. order of the execution is irrelevant (commutative atomic conditions).
- The more complex category are queries where elementary conditions have different relevance (commutative and non-commutative queries).
  Commutative queries can be solved by fuzzy implications or Ordered Weighted Averaging Operator (OWA).
  Non-commutative queries have several structures. Bipolar queries merge constraints (have to be satisfied) and wishes (is nice if are satisfied) or negative and positive judgements, respectively. Another example is non-commutative queries containing only constraints where answer of the first atomic condition influences answer to the second one.

In the next several subsections these categories are examined.

### 2.4.1  Independent Conditions Aggregated by the "And" Operator

This is the simplest form of aggregation. The t-norm functions (Sect. 1.3.1) are used as *and* logical operator. It is well known that different t-norms produce different matching degrees. Naturally, the following question has arisen: which t-norm is the most suitable one?

Let us recall the well-known fact that only the minimum t-norm (1.47) is an idempotent one. This fact is one of the reasons for the wide use of this operator. However, the minimum t-norm has a limitation. Only minimal value of all atomic conditions is considered, that is, other atomic conditions do not influence the solution. The product t-norm (1.48) takes into account membership degrees of all elementary conditions. Therefore, this t-norm distinguishes records which have the same value of the lowest membership degree of atomic conditions and different values of the satisfaction degree of other atomic conditions, but produces lower matching than the minimum t-norm. The Łukasiewicz t-norm (1.49) produces membership degree greater than 0 only for tuples which significantly satisfy the condition, i.e.:

$$\sum_{i=1}^{n} \mu_{Ai}(r) - (n-1) > 0$$

because this t-norm is a nilpotent one. The last basic t-norm is drastic product (1.50), which is not suitable due to non-continuity and high restrictiveness.

**Table 2.2**  Matching degrees calculated by different t-norms

| Tuple | $\mu_{P_1}$ | $\mu_{P_2}$ | Min(1.47) | Prod(1.48) | Łuk(1.49) | Drast(1.50) |
|-------|-------------|-------------|-----------|------------|-----------|-------------|
| r1    | 0.21        | 0.32        | 0.21      | 0.067      | 0.00      | 0           |
| r2    | 0.20        | 0.96        | 0.20      | 0.192      | 0.16      | 0           |
| r3    | 0.95        | 0.95        | 0.95      | 0.905      | 0.9       | 0           |

Let us have three records which satisfy the first atomic condition (predicate) ($P_1$) and the second atomic condition ($P_2$) as is shown in Table 2.2.

It is obvious that tuple $r3$ is the best option. But, drastic product calculates matching equal to 0. Let us now focus on tuples $r1$ and $r2$. The minimum t-norm prefers the tuple $r1$. But, this could contradict the human reasoning when selecting the best tuple: although the tuple $r1$ is only slightly better in the first atomic condition and notably worse according to the second one, it is the preferred one. The product t-norm prefers $r2$ with the membership degree lower than the values for both atomic conditions. The Łukasiewicz t-norm calculates membership degrees greater than 0 for the tuple $r2$ because $r1$ does not significantly satisfy both atomic conditions.

On the other hand, if the decision depends on the matching degree, e.g. percentage of the financial support or discount, then using non-idempotent t-norms could contradict with the usual reasoning. If tuple satisfies all atomic conditions with degree of 0.5, then it is expectable that the percentage of support is 50 % (0.5). In case of product t-norm the matching degree significantly decreases, when number of atomic conditions increases. Although product t-norm is not a nilpotent one, it converges to 0, when large number of atomic predicates exists, that is, $\lim_{n\to\infty} \prod_{i=1}^{n} \mu_{Ai}(x) = 0$. This fact is known as the limit property of t-norms [31]. On the other hand, minimum t-norm converges to minimal value of atomic predicates.

When we want to select only records which significantly meet the compound predicate (overall query condition) and avoid issue discussed in the paragraph above, then the $\alpha$-cut may be the solution. But care should be taken, when not only minimum, but also sum of atomic conditions is relevant. The nilpotent minimum t-norm (1.54) may be the option for such tasks. Let us have two records which satisfy the first atomic condition ($P_1$) and the second atomic condition ($P_2$) as is shown in Table 2.3.

According to minimum t-norm tuple $r2$ is preferred. But, tuple $r1$ dominates in the first atomic condition and is slightly worse in the second one. In order to select

**Table 2.3**  Matching degrees calculated by minimum and nilpotent minimum t-norms

| Tuple | $\mu_{P_1}$ | $\mu_{P_2}$ | Min(1.47) | Nilpotent min(1.54) |
|-------|-------------|-------------|-----------|---------------------|
| r1    | 0.80        | 0.30        | 0.30      | 0.30                |
| r2    | 0.40        | 0.35        | 0.35      | 0                   |

only tuples which significantly meet the compound query condition, threshold can be used. Let us have threshold 0.33. By minimum t-norm $r1$ is excluded. Contrary, nilpotent minimum t-norm prefers tuple $r1$ because $\mu_{P_1} + \mu_{P_2}$ is higher than 1 and therefore meets threshold value of 1 of this t-norm. To conclude this observation, query matching degrees of tuples which pass filtre of nilpotent minimum t-norm, are calculated in the same way as by minimum t-norm.

Contrary to the crisp conjunction, the fuzzy conjunction can be expressed by variety of t-norms. This provides a benefit because we can model a large scale of users' requirements. Nevertheless, we should carefully decide which t-norm is the most suitable in order to avoid inappropriate solutions. Hence, developers of information systems and databases should be familiar with the fuzzy set and fuzzy logic theory.

To summarize, when the most restrictive matching degree of atomic condition is required, then the minimum t-norm is the appropriate solution. Furthermore, when it is desirable that the sum of atomic predicates significantly contributes to solution, then nilpotent minimum t-norm is an option. When users consider satisfaction degrees of each atomic condition, then the product t-norm is the choice. In addition, if user wishes to see in the resulting relation only tuples which significantly meet all atomic conditions, then the Łukasiewicz t-norm, or applying $\alpha$-cut on the result obtained by product t-norm are the suitable choices.

Flexible queries can be straightforwardly adjusted for searching similar entities [21] to the existing or ideal one. In this type of fuzzy query membership functions are limited to triangular ones (Fig. 1.5), because support (1.9) should be limited and membership degree equal to 1 should hold only for tuples having the same value of analysed attribute as the reference tuple. The *and* connective should not be expressed by minimum or nilpotent minimum t-norm.

If atomic conditions are merged by the *or* logical operator, the s-norms are applied. This kind of queries is not further examined, but the duality principle between t-norms and s-norms discussed in Sect. 1.3.3 helps in searching for the suitable s-norm.

Design of interfaces for commutative queries, searching for similar tuples and related discussions are in Appendix A.1.

### 2.4.2  Fuzzy Preferences Among Atomic Query Conditions

The aim of preferences among atomic conditions is to distinguish more important conditions from less important ones. In everyday tasks people rarely give the same priority to all attributes. As an example let us take the query: *select young and highly productive employees where age is more important than productivity*.

In order to solve such a query, weights $w_i \in [0, 1]$ can be associated with atomic conditions. Two types of weights can be applied [49]: static and dynamic. From the names it is obvious that the static weights are fixed, known in advance and unchangeable during query processing, whereas for dynamic weights neither their values nor association to criteria are fixed a priori.

The idea for calculating the matching degree of atomic conditions $P_i$ according to an importance weight $w_i$ by fuzzy implication has the following form [49]:

$$\mu(P_i^*, r) = (w_i \Rightarrow \mu(P_i, r)), \tag{2.11}$$

where $\Rightarrow$ represents a fuzzy implication (weight implies or influences the solution). In order to be meaningful, weights should satisfy several requirements [15]:

- if $w_i = 0$, then the result should be such as $P_i$ does not exist or does not have any influence on the solution
- if $w_i = 1$, then $P_i$ fully influences the solution
- weights should be assigned to each atomic condition, whereas at least one weight should have value of 1 for the most important attribute(s): $\max_i(w_i) = 1$, $i = 1...n$

By applying these requirements, it is easy to conclude that the regular implications (S, Q, R—Sect. 1.3.4) such as Kleene-Dienes, Gödel and Goguen match the requirements.

*Example 2.5* In this example we check whether the Kleene-Dienes implication (1.63) is suitable. Consider the overall query condition consisted of several atomic predicates connected with the *and* operator $\wedge_{i=1}^n P_i$.

For very low importance of the $P_i$ ($w_i$ is close or equal to 0), the satisfaction of elementary condition $P_i$ has a very low influence on the query satisfaction, because: $w_i \to 0 \Rightarrow \mu(P_i^*, r) \to 1$, where arrow means approaching the value of. In the extreme situation (no importance at all, $w_i = 0$), $\mu(P_i^*, r) = 1$. The value of 1 is the neutral element in the conjunction.

For maximal importance of the $P_i$, $w_i$ is equal to 1. The satisfaction of $P_i$ is essential for fulfilling the overall query, because $w_i \to 1 \Rightarrow \mu(P_i^*, r) \to \mu(P_i, r)$.

In the same way it is possible to prove that the other regular implications are suitable.  □

Using the Kleene-Dienes implication, the following query condition for the conjunction is achieved:

$$\mu(r) = \min_{i=1,...,n} (\max_{i=1,...,n}(\mu(P_i, r), 1 - w_i)) \tag{2.12}$$

if the minimum function is used as a t-norm. Other t-norms can be also used. Furthermore, the Eq. (2.12) corresponds to the definition of the weighted conjunction operator introduced in [16].

For the Gödel implication (1.67) Eq. (2.11) yields:

$$\mu(P_i^*, r) = \begin{cases} 1 & \text{for } \mu(P_i, r) \geq w_i \\ \mu(P_i, r) & \text{for } \mu(P_i, r) < w_i \end{cases} \tag{2.13}$$

The weight $w_i$ is treated as a threshold. If predicate $P_i$ is satisfied to a degree greater or equal than this threshold, then the weighted condition is considered to be

fully satisfied. Otherwise it equals to matching degree of $P_i$. Furthermore, it is easy to prove that when $w_i = 0$, regardless of value of $\mu(P_i, r)$ the answer participates in the conjunction with value 1 (neutral element).

Goguen implication (1.68) is another threshold-type interpretation, but the under-satisfaction of the condition is treated in a more continuous way:

$$\mu(P_i^*, r) = \begin{cases} 1 & \text{for } \mu(P_i, r) \geq w_i \\ \frac{\mu(P_i, r)}{w_i} & \text{for } \mu(P_i, r) < w_i \end{cases} \tag{2.14}$$

Besides these implication functions, in some applications it is common to describe implication by t-norms [18]. Let us see whether this assumption holds here. For the so-called Mamdani implication (minimum t-norm), the proof of unsuitability of this implication is simple. For no importance of $P_i$ ($\mu(P_i^*, r) = \min(w_i, \mu(P_i, r)) = 0$) the overall query satisfaction will be 0, because value of 0 annihilates truth values of other atomic conditions in conjunction. It implies that the requirement: *if $w_i = 0$, then the result should be such as if $P_i$ does not exist* or does not have influence, is not satisfied for the Mamdani implication.

This result was expected. But the goal of this short discussion was to emphasize that the "simplified implication" used in fuzzy reasoning does not work here.

*Example 2.6* Let us have two atomic predicates: $P_1$ having high importance ($w_1 = 1$) and $P_2$ having lower importance (e.g. $w_2 = 0.5$). Tuples and matching degrees are shown in Table 2.4. The second and third column shows matching degrees before applying Kleene-Dienes implication and next two after applying this implication.

Without applying preferences tuples $r1$ and $r2$ have the same matching degree regardless of used t-norm (commutativity axiom). But, when predicate $P_1$ is more preferred, then its higher membership degree should be reflected in the solution. $\square$

An example of interface adjusted to preferences is illustrated in Appendix A.1.

Another way for realization of preferences can be found in [47], where the authors suggest not only crisp values, but also fuzzy sets to describe the importance value in the additional *weight* clause. The importance weights can be crisp numbers from the [0, 1] interval or fuzzy sets defined beforehand and stored in a separated database table. The benefit for users is in the possibility to select the importance defined by linguistic terms. The linguistic variable *Preferences weight* may consist of several terms such as no importance, very low, low, medium, etc. Weights can be defined in a similar way as relative quantifiers (Sect. 1.5), because domain is the unit interval.

**Table 2.4** Preferences calculated by Kleene-Dienes implication

| Tuple | $\mu_{P1}(r)$ | $\mu_{P2}(r)$ | $\mu_{P1^*}(r)$ | $\mu_{P2^*}(r)$ | Matching degree (2.12) |
|-------|---------------|---------------|-----------------|-----------------|------------------------|
| r1 | 0.9 | 0.4 | 0.9 | 0.5 | 0.5 |
| r2 | 0.4 | 0.9 | 0.4 | 0.9 | 0.4 |

### 2.4.3 Answer to the Second Atomic Condition Depends on the Answer to the First One

It may happen in everyday tasks that the answer to the first question influences answer to the second one. This occurs in queries, where conditions are not independent, contrary to the cases explained in Sects. 2.4.1 and 2.4.2. In this case, the second atomic condition is relative to the first one.

Two gradual conditions are combined in such a way that the second condition is applied on a subset of database rows, already limited by the first one. An example of such a query is: *select companies with small number of employees* ($P_1$) *among companies with high export* ($P_2$). The condition $P_2$ (consisted of atomic or compound condition) is defined a priori and the condition $P_1$ is defined in a relative manner of satisfying the condition $P_2$. If we permute predicates, the query is: *select companies with high export* ($P_1$) *among companies with small number of employees* ($P_2$). Therefore, the result may be different.

This class of queries require focus on a limited subset of an attribute domain instead of the whole domain (or to be more precise, the subset of the current content of a database).

To solve such tasks efficiently, fuzzy aggregation operator called *among* is defined [44]:

$$\mu_{P_1 among P_2} = \min(\mu_{P_1/P_2}(x), \mu_{P_2}(x)), \tag{2.15}$$

where $\mu_{P_2}$ is the membership function defining fulfillment of the independent predicate and $\mu_{P_1/P_2}$ is the membership function of the dependent predicate relative to the independent one. The former is constructed directly from data or by user, whereas the latter represents a transformation of the initial membership function $\mu_{P_1}$ affected by the independent predicate.

The construction of the $\mu_{P_2}$ can be realized by any method mentioned in Sect. 2.2. The construction of the $\mu_{P_1/P_2}$ is realized by the transformation $f$ in the following way [44]:

$$\mu_{P_1/P_2} = f \circ \mu_{P_1}, \tag{2.16}$$

where $f$ is the transformation between initial domain $[l, h]$ and reduced one $[l', h']$

$$[l', h'] \mapsto f : [l, h] \tag{2.17}$$

$$f(x) = l + \frac{h - l}{h' - l'}(x - l') \tag{2.18}$$

Hence, the *among* operator is calculated as (2.15)–(2.18):

$$\mu_{P_1 among P_2}(x) = \min(\mu_{P_1}(l + \frac{h-l}{h'-l'}(x - l')), \mu_{P_2}(x)) \qquad (2.19)$$

Another option for constructing compressed membership function (fuzzy set) is discussed in [22]. However, this approach requires one scan and two queries. In the first step entities satisfying the condition $P_2$ are selected. Tuples selected by $P_2$ create a subrelation of all tuples. It causes the compression of initial domain, i.e. $[L_{P1-compr}, H_{P1-compr}] \subseteq [L_{P1}, H_{P1}]$ of the dependent attribute $P_1$, where $L_{P1}$ and $H_{P1}$ represent the lowest and the highest value of dependent attribute in the whole relation, respectively.

In the second step, a database scan retrieves values of $L_{P1-compr}$ and $H_{P1-compr}$. In the third step, the fuzzy set describing dependable condition is created on the sub-domain $[L_{P1-compr}, H_{P1-compr}]$ by the uniform domain covering method [44]. Even if user can define parameters for the membership function $\mu_{P_2}$ without suggestion from the current database content, defining the membership function for $\mu_{P_1/P_2}$ on the interval $[L_{P1-compr}, H_{P1-compr}]$ depends on the selected tuples in the first step. Therefore, this step should be automatized.

Finally in the last step, the overall query matching degree is calculated by (2.15). This procedure is shown in Fig. 2.3.

*Example 2.7* A small survey is conducted to find expensive books among books with small number of pages. Books from a bookshop are shown in Table 2.5. Independent predicate *number of pages is small* is expressed as L fuzzy set (Fig. 1.8) with parameters $m = 200$ and $b = 250$. This condition returns five books, which fully or partially match independent predicate, shown in Table 2.6.
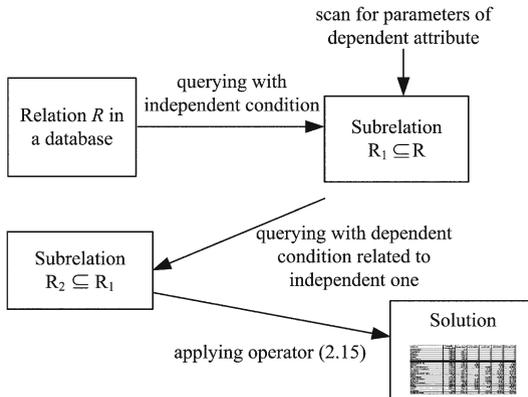


**Fig. 2.3** The procedure for calculating matching degree when dependable condition is not defined a priori but in a relative manner

**Table 2.5** Books in a bookshop

| Book | Pages | Price |
|------|-------|-------|
| Book 1 | 420 | 60 |
| Book 2 | 500 | 25 |
| Book 3 | 290 | 50 |
| Book 4 | 210 | 36 |
| Book 5 | 100 | 45 |
| Book 6 | 120 | 38 |
| Book 7 | 225 | 10 |
| Book 8 | 240 | 50 |
| Book 9 | 310 | 70 |
| Book 10 | 300 | 30 |

**Table 2.6** Selected books from Table 2.5 by independent condition

| Book | Pages | Price |
|------|-------|-------|
| Book 4 | 210 | 36 |
| Book 5 | 100 | 45 |
| Book 6 | 120 | 38 |
| Book 7 | 225 | 10 |
| Book 8 | 240 | 50 |

It is now a straightforward task to detect in the Table 2.6 the smallest value ($L_{compr} = 10$) and the highest value ($H_{compr} = 50$) in the subdomain of price attribute.

In the next step, by the uniform domain covering method parameters of fuzzy set *price is high* are calculated: $\theta = \frac{1}{4}(H_{compr} - L_{compr}) = 10$, $\varepsilon = \frac{1}{2}\theta = 5$ and then the *price is high* set is expressed as R fuzzy set with parameters $a = C = 35$ and $m_1 = D = 40$ (Fig. 2.2). Finally, by (2.15) the result is shown in Table 2.7.

Clearly, if permutation of attributes is realized, then the answer may be different. If a survey searches for small paged books among expensive ones, then the high price is analysed on the whole domain and small number of pages on the reduced subdomain. By the same procedure, the answer consists of two books: *Book 3* and *Book 8* with the matching degree of 0.33 each. □

**Table 2.7** Solution by the *among* connective

| Book | Pages | $\mu_{Pages}(r)$ | Price | $\mu_{Price/Pages}(r)$ | Among |
|------|-------|------------------|-------|------------------------|-------|
| Book 5 | 100 | 1 | 45 | 1 | 1 |
| Book 6 | 120 | 1 | 38 | 0.6 | 0.6 |
| Book 4 | 210 | 0.8 | 36 | 0.2 | 0.2 |
| Book 8 | 240 | 0.2 | 50 | 1 | 0.2 |

The commutativity and monotonicity properties are not satisfied, whereas the associativity and existence of unit element are [44]. The associativity property allows creating encapsulated dependencies, e.g. $P_1$ *among* $P_2$ *among* $P_3$. It is crucial for users to properly define parentheses. The condition could be for example, *(small polluted municipalities among high sized) among high unemployed* as well as *small polluted municipalities among (high sized among high unemployed)*.

Predicates $P_1$, $P_2$ could be atomic conditions such as in Examples 2.7 and A.4 or compound ones, for instance *select municipalities with (small altitude and high pollution)* ($P_1$) *among municipalities with (high unemployment and high population density)* ($P_2$). The same discussion related to choice of suitable conjunction in Sect. 2.4.1 holds here for calculating matching degrees inside the dependent and independent compound predicates.

The interface from cases examined in Appendix A.1 has been adjusted to this kind of tasks in Appendix A.2.

### 2.4.4   Constraints and Wishes

Not all requirements for data can be represented as constraints. When people express their requirements they could have in mind wishes as well. A suitable example is *find hotel which has low price and possibly short distance to point M*. The budget is the limitation; we cannot afford something beyond our budget. The distance is a wish. We would prefer shorter walk, if possible. These tasks can be solved either by bipolar queries (where conditions are fully independent), or by non-commutative operators keeping the wishes and constraints together. In the book the focus is put on the latter, due to simpler way for creating applications for non-expert users. In order to mention both aspects, bipolar queries are discussed as well.

We can formally write query as [48]: *find tuples satisfying N and if possible P*, where $N$ denotes negative preference and $P$ describes positive preference. Answer to a bipolar query is written in the following way:

$$A_{Q_{fbp}} = \{r \mid N(r) \ and \ possibly \ P(r)\} \tag{2.20}$$

It is evident that this kind of queries is not commutative. Query *short distance and possible low price* has a different semantic meaning (budget is not a problem but distance is).

This type of queries cannot be solved by *and* operator, weights or averaging aggregations. Let us consider the low price and short distance atomic predicates from the aforementioned example. When both predicates are either fully satisfied, or fully rejected the answer is clear. When low price is fully unsatisfied, then the answer should be zero, regardless of the satisfaction degree of short distance. But, when low price predicate is fully satisfied and short distance predicate fully unsatisfied, the answer should be lower than 1, but greater than 0. Weight attached to the short distance predicate is not able to provide solution.

### 2.4.4.1  Bipolar Queries

The first attempt to solve bipolar queries (2.20) was realized for crisp ones. In this classical approach the condition for tuple $r$ is expressed as [33]:

$$N(r) \; and \; possibly \; P(r) = N(r) \land \exists s(N(s) \land P(s)) \Rightarrow P(r) \qquad (2.21)$$

In the first step tuples satisfying $N$ are selected from a database. This step ensures that tuples which do not satisfy $N$ are not considered. If no tuple meets $N$, then answer to bipolar query is empty. In the second step tuple $r$ is preferred, if no other tuples satisfies $P$ or tuple $r$ in the best way satisfies $P$. The approach: first to select by using $N$, then to order by using $P$ cannot be directly applied when satisfaction is a matter of degree [14, 48].

The crisp structure of the bipolar query (2.21) is expressed in fuzzy terms in the following way [48]:

$$A_{Q_{(N,P,R)}} = \{(r, \mu(r)) \mid min(N(r), max(1 - max_{s \in R} min(N(s) \land P(s)), P(r)))\},$$
$$(2.22)$$

where $(N, P, R)$ means answer to $N$ and $P$ against the set of tuples $R$. This equation is the generalization of the (2.21) from crisp to fuzzy logic. Quantifier $\exists$ is modelled by the *maximum* operator. The implication is characterized by the Kleene-Dienes one (1.63). Minimum t-norm, maximum s-norm and standard negation form a triplet characteristic by the fact that minimum t-norm is dual to maximum s-norm when the standard negation is applied. Other triplets like (Łukasiewicz t-norm and s-norm, standard negation) or (product, probabilistic sum, standard negation) can be also applied, if reinforcement effect is needed (different functions provide different matching degrees). Influences of different functions for quantifier, implication, t-norm and s-norm on the solution are discussed in [48].

Formula (2.22) expresses the global interpretation of the term *and if possible*, i.e. checking whether the constraint is satisfied by at least one tuple from the dataset considered and comparing with other tuples. On the other side, in the local interpretation (Sect. 2.4.4.2) satisfying the constraint provides a benefit to the tuple, but there is no need to compare with the other tuples.

*Example 2.8* Let us have four houses satisfying $N$ (low price) and $P$ (short distance) with degrees depicted in Table 2.8. The matching degree to query condition obtained by the bipolar *and if possible* operator (2.22) is in the last column. The high satisfaction of wish is not as important as high satisfaction of constraint. Furthermore, influences of other tuples are considered. In the first step, tuples satisfying $N$ are considered. This step ensures that tuples which do not satisfy $N$ are excluded. In the second step tuple $r$ is preferred, if no other tuples better meet $P$ or tuple $r$ satisfies $P$. Hence, the best house is $Ho1$.  □

**Table 2.8** Bipolar *and if possible operator*

| House | $N(r)$ | $P(r)$ | $A_{Q_{(N,P,R)}}(r)$ (2.22) |
|-------|--------|--------|------------------------------|
| Ho 1  | 0.8    | 0.5    | 0.7                          |
| Ho 2  | 0.8    | 0.3    | 0.5                          |
| Ho 3  | 0.2    | 0.7    | 0.2                          |
| Ho 4  | 0.1    | 0.9    | 0.1                          |

The second approach for managing bipolarity is based on the possibility theory [14]. The answer is measured on bipolar scales: either on one scale, where the middle point is neutral and ends bear extreme positive or extreme negative values; or on two scales, one scale measures the positive and the other one negative preferences. The third approach for bipolar queries is based on the lexicographic ordering [3]. In this approach degrees for $N$ and $P$ are evaluated separately, i.e. no aggregation between constraint and wish is performed.

### 2.4.4.2  Non-commutative Operators

Four non-commutative operators have been introduced in [4]:

- $P_1$ *and if possible* $P_2$—relaxation of conjunction
- $P_1$ *or else* $P_2$—intensification of disjunction
- $P_1$ *all the more as* $P_2$
- $P_1$ *all the less as* $P_2$

In this book first two operators are considered. Instead of bipolarity, this approach is focused on relaxing conjunction and intensification disjunction, respectively. These operators are also called asymmetric conjunction and asymmetric disjunction correspondingly.

Bosc and Pivert [4] created the following six axioms in order to formally write *and if possible* operator:

- is less restrictive than the *and* operator ($P_1$ *and* $P_2$), i.e. $\alpha(\mu_{P_1}, \mu_{P_2}) \geq \min(\mu_{P_1}, \mu_{P_2})$
- is more drastic than only constraint ($P_1$) appears, i.e. $\alpha(\mu_{P_1}, \mu_{P_2}) \leq \mu_{P_1}$;
- is increasing in constraints argument, i.e. $a \geq b \Rightarrow \alpha(a, c) \geq \alpha(b, c)$;
- is increasing in wishes argument, i.e. $b \geq c \Rightarrow \alpha(a, b) \geq \alpha(a, c)$;
- has asymmetric behaviour, i.e. $\alpha(\mu_{P_1}, \mu_{P_2}) \neq \alpha(\mu_{P_2}, \mu_{P_1})$;
- $P_1$ *and if possible* $P_2$ is equivalent to $P_1$ *and if possible* ($P_1$ *and* $P_2$), i.e. $\alpha(\mu_{P_1}, \mu_{P_2}) = \alpha(\mu_{P_1}, \min(\mu_{P_1}, \mu_{P_2}))$.

Hence, function $h$ of the structure:

$$\alpha(\mu_{P_1}, \mu_{P_2}) = \min(\mu_{P_1}, h(\mu_{P_1}, \mu_{P_2})) \tag{2.23}$$

is sought.

From the aforementioned axioms and structure, the following operator is created [4]:

$$\alpha(\mu_{P_1}, \mu_{P_2}) = \min(\mu_{P_1}, k \cdot \mu_1 + (1 - k)\mu_{P_2}), \tag{2.24}$$

where $\mu_1$ is the satisfaction degree to a constraint, $\mu_2$ is the satisfaction degree to a wish and $k \in [0, 1]$ expresses relation between constraint and wish. If $k = 0$, the result is ordinal minimum t-norm; if $k = 1$, the result depends only on constraint $P_1$. If $k = 0.5$ is chosen, the operator is:

$$\alpha(\mu_{P_1}, \mu_{P_2}) = \min(\mu_{P_1}, \frac{\mu_{P_1} + \mu_{P_2}}{2}) \tag{2.25}$$

At the first glance, when we consider $P_1$ as $N$ and $P_2$ as $P$, then it could be the bipolar query as expressed in Sect. 2.4.4.1. However, this *and if possible* operator cannot handle bipolarity, because $\alpha$ does not keep both $P_1$ and $P_2$ separated. Anyway, this operator can efficiently solve many practical tasks where connective merges constraints and wishes.

*Example 2.9* A client of real estate agency searches for a non expensive flat and if possible near the lake. In order to focus on *and if possible* operator, steps of constructing fuzzy sets and retrieving tuples from the database are skipped. Flats and respective membership degrees are in Table 2.9. This table illustrates satisfaction of aforementioned axioms. In case of symmetric conjunction flats $Ft6$ and $Ft7$ are indistinguishable. By operator (2.24) high satisfaction of wish is not as important as high satisfaction of constraint. Furthermore, if wish is fully non-satisfied, then the matching degree to constraint is lowered; if constraint is fully non-satisfied, then the tuple's matching degree is 0. Hence, averaging operators cannot be applied. Furthermore, when minimum function is applied both attributes are constraints.

The formula (2.24) corresponds to the local interpretation of constraints and wishes, that is, each tuple is examined independently. The first row in Table 2.8 and third row in Table 2.9 have the same membership degrees to constraints and wishes, but the calculated matching degree is higher by (2.22) than by (2.25), because in Table 2.8 this tuple dominates other tuples which is reflected in the matching degree.

Clearly, result by (2.24) for $k \neq 0$ and $k \neq 1$ does not correspond with the minimum operator and using only constraint, respectively (Table 2.9). Matching degree is approaching to the result obtained by minimum t-norm (1.47), when parameter $k$ is approaching to the value of 0.

If we try to model preferences and wishes as conditions with different priorities (Sect. 2.4.2), then the solution is not the same. Although, some tuples could have the

**Table 2.9** Non expensive flats and if possible near to the lake

| Flat | $\mu_{LP}(r)$ | $\mu_{SD}(r)$ | $\alpha(r)$(2.25) | Min $(\mu_{LP}, \mu_{SD})$ |
|------|---------------|---------------|-------------------|----------------------------|
| Ft 1 | 1   | 0.7 | 0.85 | 0.7 |
| Ft 2 | 0.8 | 0.8 | 0.8  | 0.8 |
| Ft 3 | 0.8 | 0.5 | 0.65 | 0.5 |
| Ft 4 | 1   | 0   | 0.5  | 0   |
| Ft 5 | 0.8 | 0.1 | 0.45 | 0.1 |
| Ft 6 | 0.6 | 0.1 | 0.35 | 0.1 |
| Ft 7 | 0.1 | 0.6 | 0.1  | 0.1 |
| Ft 8 | 0   | 1   | 0    | 0   |
| Ft 9 | 0.9 | 0.4 | 0.65 | 0.4 |

where $\mu_{LP}(r)$ stands for membership degree to low price and $\mu_{SD}(r)$ to short distance

same matching degree calculated by (2.12) when weights of 1 and 0.5, correspondingly to constraint and wish in (2.25) are used (e.g. for matching degrees of 0.8 and 0.2), care should be taken, as the meanings of the queries is different. It is illustrated on tuples $Ft9$ in Table 2.9 and tuple $r1$ in Table 2.4. Even though these tuples have the same satisfaction degrees, the matching degree is different.  □

Analogously, the *or else* is dually defined, i.e. operator should be more drastic than the *or*, because $P_2$ is not a full alternative to $P_1$; less restrictive than using only $P_1$; must have asymmetric behaviour (the permutation of $P_1$ and $P_2$ gives different result); monotonic in both constraints and wishes; $P_1$ *orelse* $P_2$ is equivalent to $P_1$ *orelse* $(P_1$ *or* $P_2)$. From these axioms the following operator appears [4]:

$$\beta(\mu_{P_1}, \mu_{P_2}) = \max(\mu_{P_1}, k \cdot \mu_1 + (1 - k)\mu_{P_2}),  \tag{2.26}$$

where variables have the same meaning as in (2.24). If $k = 0$, the result is ordinal maximum s-norm (1.59). If $k = 1$, the result depends only on alternative $P_1$. If $k = 0.5$ is chosen, the *or else* operator yields:

$$\beta(\mu_{P_1}, \mu_{P_2}) = \max(\mu_{P_1}, \frac{\mu_{P_1} + \mu_{P_2}}{2})  \tag{2.27}$$

Construction of fuzzy sets for constraints and wishes could be modelled as independent. Construction of fuzzy sets for $N$ does not have influence on construction of fuzzy sets for $P$ (in Example 2.9 low price for constraint and short distance for wish). It depends on users, which way for construction of fuzzy sets is chosen.

An interface covering constraints and wishes and construction of fuzzy set from data is demonstrated in Appendix A.2.

Constraints $P$ and wishes $N$ could contain atomic or compound conditions. A compound condition could be of any form mentioned above, e.g. *(unemployment is high and pollution is small) and if possible (population density is high and altitude*

*is small).* As braces suggest, firstly, membership degrees inside constraint and wish are calculated. Secondly, asymmetric conjunction is applied.

### 2.4.5 Quantified Queries

This class of database queries uses linguistic quantifiers in query conditions. These conditions can be used as nested subqueries, especially for the relationship 1:N such as REGION-DISTRICT or CUSTOMER-ORDER. An example of such a query is *select districts where about half of municipalities have high altitude*.

The second application is in query relaxation tasks. A query usually consists of several atomic predicates merged by the *and* connective ($\wedge_{i=1}^{n} P_i$). The answer is empty, even if only one atomic predicate is not satisfied (whereas values of attribute for several tuples almost "touch" the space delimited by the predicate), or none of predicates is satisfied. When all atomic predicates must be satisfied, then the *and* connective is option. Otherwise, quantified query is a solution.

Users may be interested in tuples which meet majority of atomic predicates. In this way the query is of structure *select tuples where Q of* $\{P_1, P_2,..., P_n\}$ *is satisfied* [29] where $P_i$ ($i = 1, ..., n$) can be either crisp or fuzzy condition and $Q$ is a linguistic quantifier *most of*, but other quantifiers such as *about half* and *few* can be also applied.

Furthermore, constraints and wishes can be applied on quantified queries. This query is of structure $Q^C$ *of* $\{P_i\}$ *and if possible* $Q^W$ *of* $\{P_j\}$, where $Q^C$ stands for quantifier appearing in constraint part of the query and $Q^W$ stands for quantifier explaining wish part of the query. These queries can be solved by bipolar approaches or asymmetric conjunction. The former is suggested in [26] to be applied (2.22) on quantified preferences and wishes. The latter is suggested in [19] adjusting (2.24) to quantified preferences and wishes.

In addition, quantified queries mitigate empty answer problem, because these queries are less restrictive than queries created by the *and* connective. Empty answer problems are discussed in Sect. 2.5.

*Example 2.10* The task is to find suitable village for building house. The relevant predicates could be: altitude above sea level around 1500 m ($P_1$), small population density ($P_2$), medium area of village size ($P_3$), low pollution ($P_4$), high number of sunny days ($P_5$), short distance to the region capital ($P_6$) and positive reviews about village ($P_7$). It is highly presumable that none of villages meets all predicates in a query of the structure $\bigwedge_{i=1}^{7} P_i$, even though predicates have flexible boundaries.

In order to solve this problem user may say that village should be considered, if it meets majority of predicates. Let us further say that $P_1$, $P_2$, $P_3$ and $P_4$ are constraints and $P_5$, $P_6$ and $P_7$ are wishes.

This example is solved in Chap. 3, where other parts required for coping with such a query are introduced. □

Quantified queries expressed by linguistic quantifiers and fuzzy or crisp predicates are basic building blocks for linguistic summaries. In this case the answer is not a set

of tuples and their respective matching degrees, but the truth degree of the sentence. For instance, marketing department is curious to know whether *most of middle-aged customers have short payment delay*. Because linguistic summaries are separated topic in this book, Chap. 3 is dedicated to them.

### 2.4.6  Querying Changes of Attributes over Time

Particular interest is focused on analysing development of attributes over time to reveal trends, changes and to forecast future developments. Theory of time series is a mature science field capable to cope with broad variety of tasks and trends. The intent of this section is not to contribute to this field, but to show how fuzzy queries can be used for retrieving tuples of preferred or critical trends. An example of such a query is *select municipalities which have high positive change (increase) in length of roads*. Similarly, the aim of query *select customers which have almost no change in amounts of orders* is to identify customers not influenced by the marketing campaign.

In these cases, queries are not focused on attributes and their respective values, but on difference between values in target year $Y_t$ and reference (initial) year $Y_i$. Hence, the difference is theoretically value form the interval $[-\infty, \infty]$ with neutral element 0 (no change).

The usual terminology of expressing changes in fuzzy control is used in this section. The inspiration for construction of fuzzy sets has arisen from fuzzy controls of technical systems. Changes in the left half of the interval are labelled as negative, e.g. negative small, negative medium and negative high. Correspondingly, changes in the right part of the interval are labelled as positive small, positive medium and positive high. Finally, changes around value of 0 are labelled as almost zero. Linguistic variable *change* and possible definitions of its terms are plotted in Fig. 2.4.

In many fields, including municipal statistics or customer relationship management this way of naming could be weird. Lets us consider changes in pollution. If pollution significantly decreases, e.g. by value of $-75\,\%$, it can be hardly expressed as negative high change for inhabitants. On the other side, employment is an attribute for which the meaning of positive and negative changes are opposite. In this book we use terms as shown in Fig. 2.4 naming negative change for values lower than 0.

If changes are not stored in the database, then change is calculated as compound attribute for a tuple $r$ in the following way:

$$A_{cr} = \frac{A_{Y_{tr}} - A_{Y_{ir}}}{A_{Y_{ir}}} \cdot 100, \tag{2.28}$$
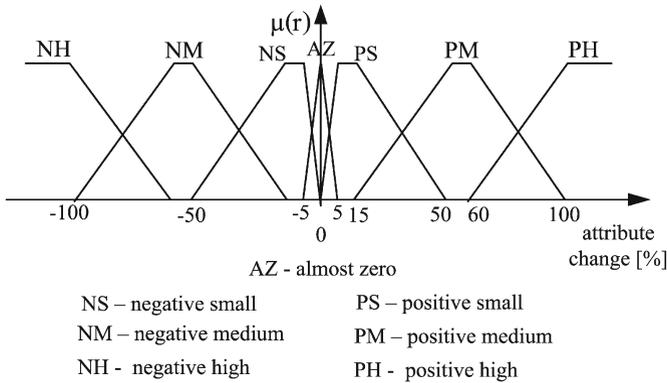
Fig. 2.4 Fuzzification of the linguistic variable *change*

where $A_{cr}$ is a change of tuple $r$ for attribute $A$, $A_{Y_{tr}}$ is a value of attribute $A$ in a target year and $A_{Y_{ir}}$ is a value of attribute $A$ in an initial or referenced year. An example of this kind of query is demonstrated in Appendix A.3.

Therefore, this is an additional computational effort, but compensated with an efficient way for identifying tuples with critical or preferred changes.

Moreover, this kind of queries can be extended with all aforementioned approaches: commutative query conditions (e.g. *select municipalities which have high positive change in length of roads and high positive change in employment*), preferences between atomic conditions, constraints and wishes, quantified queries (e.g. *select regions where most of municipalities has almost no change in gas consumption*).

## 2.5  Empty and Overabundant Answers

Queries (either crisp or fuzzy) contain a logical condition which delimits tuples we are looking for. As a result of the query two extreme situations may occur: no data or very large amount of data satisfy the query condition. The user is confronted with informativeness of the result. In some cases, these answers are informative enough, e.g. empty answer of the condition *customers in payment delay* means that all of them meet the payment deadline and therefore no reminder should be generated.

In other cases, the main objective is to solve the problem; that is, to obtain a non-empty result to inform users, why an empty answer occurred and how close are tuples to meet the query condition. The opposite holds for an overabundant answer. A survey [7] has provided a detailed insight into these two problems.

### 2.5.1 Empty Answer Problem

The empty answer problem simply means that no data match the query condition. The fuzzy query $Q_f$ (2.6) results in an empty set if $A_{Q_f} = \emptyset$. It could be useful to provide some alternative data which almost match query condition. This problem was initially recognized in [1], where linguistic modifiers were suggested as a solution. The goal of these modifiers is to slightly modify fuzzy sets in order to obtain a less restrictive variant of query condition which may return some data and remains semantically close to the initial query.

This problem appears in crisp queries more frequently due to sharp conditions (Examples 2.2 and 2.11). Making conditions flexible mitigates, but not fully eliminates this problem.

Generally, the query $Q$ (either crisp or fuzzy) is transformed into a less restrictive variant $Q_T$ by the transformation $T$ of the condition. The querying process is then repeated until the answer is not empty, or the modified query condition is semantically far from the original one.

In bipolar queries and asymmetric conjunction (*and if possible*) answer is empty, if no tuple meets the constraint part. The satisfaction of wish is not so relevant for the occurrence of empty answer. Hence, transformation should be primarily focused on the constraint part.

*Example 2.11* A simple example is selecting high players for a basketball team. The condition is *where height > 200 cm*. If query ends as empty, there are two possibilities: heights of all players are far from 200 cm or some of them almost meet the condition. Hence, query is transformed into a less restrictive one by transformation $T = 200 - 5$, i.e. relaxed query is: *where height > 195 cm*. If relaxed query returns no players, next transformation creates condition *where height > 190 cm*, and so on. The process of successive transformations should stop when some players are selected or the transformed query is far from the initial one, e.g. condition *where height > 175 cm* could be hardly considered as a selection condition for high basketball players.

The main drawback is in the satisfaction degree which is always of the value of 1 for all selected records. It is why there is no difference in membership degrees of records selected in any of transformed queries. The same problem was discussed in Sect. 2.1 in Fig. 2.1. □

In [8] the following approaches to defeat the empty answer problem are recognized: the linguistic modifier-based approach, the fuzzy relative closeness-based approach and the absolute proximity-based approach. In these approaches the query weakening process should meet the following constraints for each predicate involved in the weakening process [7]:

$$C1: \quad \forall x \in D, \ \mu_{T(P)}(x) \geq \mu_P(x), \tag{2.29}$$

where $x$ is the value of the attribute $A$ in the domain $D$. The transformation $T$ does not decrease the membership degree for any tuple in domain $D$ of attribute $A$ for

predicate $P$. Tuples in transformed query should have higher membership degree because, this query is less restrictive.

$$C2: \quad support\ (P) = \{x \mid \mu_P(x) > 0\} \subset support\ (T(P)) = \{x \mid \mu_{T(P)}(x) > 0\}$$
$$(2.30)$$

The transformation extends the support (1.9) of fuzzy set included in the condition. In the case of trapezoidal fuzzy set in Fig. 1.7, the support is the $[a, b]$ interval constructed for the predicate $P$. It means that the condition is relaxed (parameter $a$ is transformed to lower value, whereas parameter $b$ is transformed to higher value) to retrieve more tuples.

$$C3: \quad core\ (P) = \{x \mid \mu_P(x) = 1\} = core\ (T(P)) = \{x \mid \mu_{T(P)}(x) = 1\}$$
$$(2.31)$$

The transformation preserves the core (1.10) of fuzzy set. The transformed query cannot retrieve more tuples, which fully meet the relaxed condition than the initial query condition. This is a significant difference between relaxing crisp and fuzzy query. This requirement emphasizes the benefit for relaxing fuzzy queries in comparison to the relaxation of the crisp ones (Example 2.11).

An example of the original predicate $(P)$ and the transformed one $(T(P))$ according to the requirements C1–C3 (2.29–2.31) is plotted in Fig. 2.5 for the condition expressed by the triangular fuzzy set and in Fig. 2.6 for the condition expressed by the L fuzzy set. The transformation for the trapezoidal fuzzy set is shown in [7]. If a singleton fuzzy set (Fig. 1.10) is used, then it could be transformed into less restrictive triangular fuzzy set. Requirement C3 causes that the membership degree is equal to 1 only for tuples having value of attribute equal to $m$ (1.24). In this way, a sharp number is transformed into less restrictive triangular fuzzy number.

In order to ensure that the retrieved records are not semantically far from the initial query condition, the stopping criterion should be incorporated. In cases of the first and the third approaches aforementioned no intrinsic semantic limit is provided. Hence, the user has to specify a set of non-adequate data. In case of searching high basketball players the set for stopping criterion might be explained by the sharp limit *height not smaller than 185 cm*, i.e. $C_c = \{x \in D \mid x < 185\}$. Set $C_c$ is defined by its
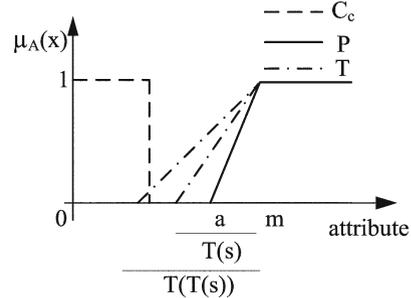


**Fig. 2.5** Weakening of the condition defined by a triangular fuzzy set
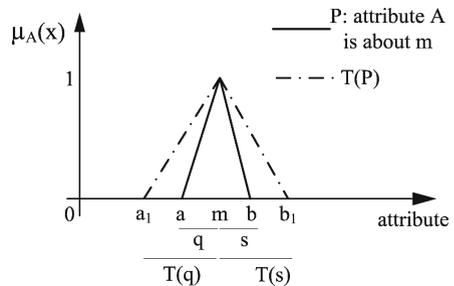
**Fig. 2.6** Weakening of the
condition defined by a L
fuzzy set
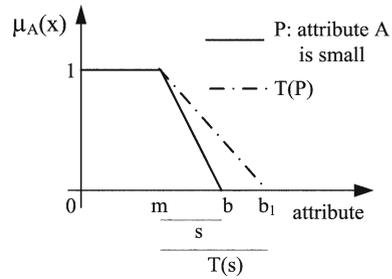


**Fig. 2.7** Requirement C4 is
activated for the $T(T(r))$



characteristic function $\varphi_c(x)$ on the domain $D$. Applied on the fuzzy condition, the
additional requirement C4 is as follows (Fig. 2.7):

$$C4: \quad \min(\mu_{T(P)}(x), 1 - \varphi_c(x)) = 0 \qquad (2.32)$$

Concerning the requirements C1–C3 (2.29–2.31), the stopping criterion says that
the weakening process stops, when the answer to modified $Q_f$ is not empty. The
constraint C4 (2.32) ensures that only tuples which are not semantically far from the
initial query are selected. When during the weakening process the transformation
$T(T(P)...)$ enters into the core of $C_c$, it causes that C4 gets value of 0 (because
$1 - \varphi_c(x) = 0$) and therefore weakening process stops. In this way the relaxation is
controlled.

Instead of the crisp set $C_c$ we can construct fuzzy set $F_c$, a set of more or less
non-adequate data.

Aforementioned approaches deal with the local weakening, that is, the basic weak-
ening transformation applies to each atomic condition.

The further approach is focused on replacing the query which caused an empty
answer by a semantically similar one which has been already processed and provided
non-empty answer [2].

### *2.5.2  Overabundant Answer Problem*

The opposite situation arises when a large amount of tuples satisfies the query condition. The query $Q_f$ results in overabundant answer if the cardinality of $A_{Q_f}$ (2.6) is too large.

To recognize the empty answer problem is a trivial task. In the case of an overabundant answer problem, it is the opposite. It is not easy to answer the question, where lies the boundary between non-overabundant and overabundant answers. It depends on the user, how many tuples he wants to obtain, and on the number of tuples in the database.

From the theoretical point of view the problems concerning empty and overabundant or plethoric answers are dual. The ways how to solve overabundant answer problems are examined in [7]. Generally, two situations might arise: too many tuples fully meet a query condition and/or too many tuples partially meet a query condition.

The first situation requires intensification of the query condition; that is, reduction of the core of fuzzy sets in order to reduce the number of tuples that fully meet the query condition. This process is dual to the weakening process expressed by requirements (2.29)–(2.32). At the first glance it is evident that this transformation is not applicable on triangular and singleton fuzzy sets.

*Example 2.12* Case 1: Large number of customers fully meet the condition *turnover = 950*. We cannot reduce the core, because it is expressed as singleton. The possible solution is adding an additional suitable atomic condition semantically close to the initial query.

Case 2: Large number of customers fully meet the condition *turnover about 950*, where the term is expressed as a trapezoidal fuzzy set. Hence, the answer can be reduced by the intensification of the flat segment in the initial query condition. □

At the first glance the second situation is easy to solve, because the theory of fuzzy sets offers several options. The first one is expressing the *and* operator with the Łukasiewicz t-norm or nilpotent minimum t-norm, because tuples which do not significantly match the query condition are excluded. The second option is the $\alpha$-cut (1.15). For example, the *threshold* clause [6, 47] could solve this problem. However, a situation when very large number of answers has the same maximal score (different from value of 1) might occur [20]. In this case the intensification of condition is not the solution. These records will have again the same, although lower membership degree to the answer. $\alpha$-cuts and the more restrictive t-norms are not applicable as well. The solution could consist in adding additional elementary condition semantically close to the initial one. An approach for adding a semantically similar attribute to the query condition in order to obtain more restrictive solution is suggested in [10]. Additional possible option are fuzzy functional dependencies [41, 46] which are useful in the process of mining related attributes. A related attribute could be connected by the *and* operator to intensify the initial query condition.

Aforementioned approaches cope with these problems mainly, when they appear: after the query realization. Another possibility is revealing information about possible

risk of empty and overabundant answers. The first approach is the pre-computation
of the summary of database in order to retrieve information about distribution of data
in domains or parts of domains limited by the created query condition, which is about
to be realized. In this way a single scan of databases provides relevant information
about fuzzy cardinality [40]. The second approach is construction of fuzzy sets for
linguistic conditions directly from data discussed in Sect. 2.2.

Anyway, from the practical point of view, the empty answer problem is more
relevant for users than the overabundant one. The empty sheet of data provides
limited information about data in databases, whereas the sheet containing a large
number of tuples could be filtered by several other methods in spreadsheet software
tools, for example.

## 2.6  Some Issues Related to Practical Realization

From the user's point of view, fuzzy queries are seen as akin to human-oriented
languages, as they offer for the user to linguistically formulate query conditions, i.e.
manage vagueness in queries for searching relevant tuples.

The purpose of an interface is to offer for non-expert users the ability to ask for data
without necessity to know the complexity of relational database and fuzzy sets and
fuzzy logic theory. There are many articles dealing with the user-friendly interfaces
at disposal, e.g. [21, 27, 37, 39, 43]. The appropriate user's interface is immanent for
the acceptance of flexible queries by variety of users. It means that topics discussed in
this chapter need to be offered for users and managed in an efficient and appropriate
way. In addition, presenting retrieved records and their matching degrees in useful
and understandable ways is also a very important task. In Appendix A interfaces
for managing several kinds of fuzzy queries for the municipal statistics database are
discussed including presentation of solutions in tables and speculation on merging
of matching degrees with polynomial areas in thematic maps.

The architecture used in this book is shown in Fig. 2.8. The mathematical back-
ground is based on the GLC (2.9), (2.10). The kernel of the application is used in
next chapters.

The operations of two-valued logic meet all axioms of Boolean algebra, namely
excluded middle, contradiction and idempotency, whereas operations of fuzzy logic
do not meet all of them [35]. At the first glance violation of the excluded mid-
dle and contradiction axioms could be considered as problematic, because user
could expect that the *and* connective between proposition and its negation should be
always 0. However, uncertainty in belonging to a proposition is reflected in belong-
ing to its negation and therefore higher value than 0 is acceptable. Although this fact
is disputable from the theoretical point of view (some solutions exist how to solve
it, e.g. interpolative realization of Boolean algebra [35, 36]), this is not a significant
problem in querying.

The fact which is more problematic, is the lack of satisfying the idempotency
axiom (1.53) for all t-norms, except the minimum t-norm (1.47). Let us look at
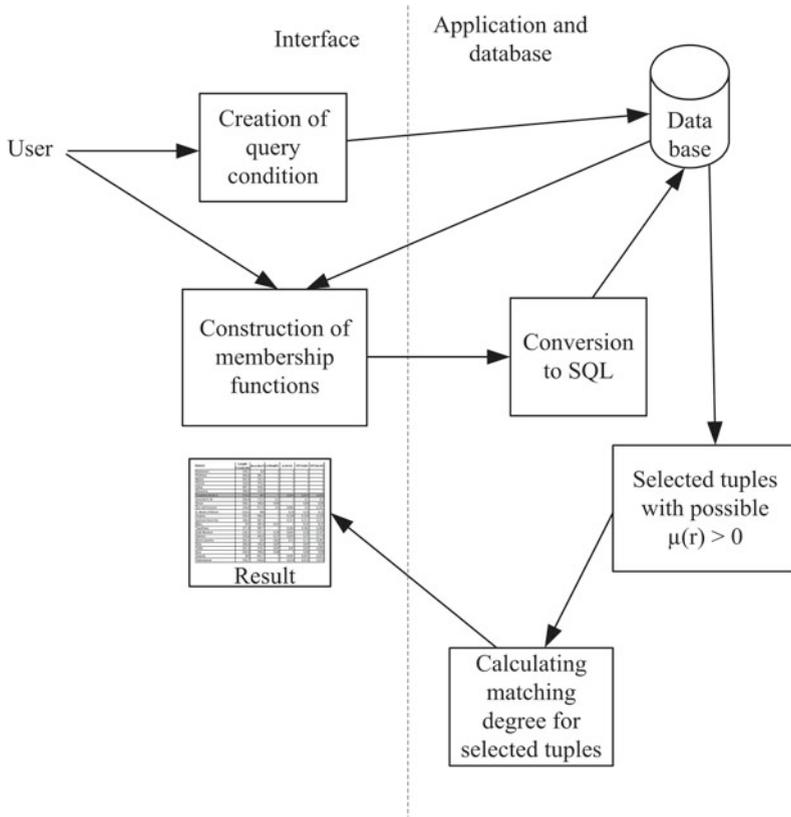
**Fig. 2.8**  A possible architecture of the flexible querying

two queries: *A is High* and *A is High and A is High*, where $\mu(r) < 1$. Some t-norms provide different answers to the first and the second query, namely the product (1.48) and Łukasiewicz (1.49) t-norms. Moreover, in the case when $\mu(r) < 0.5$ and using Łukasiewicz t-norm, the result is 0. This kind of query is not a realistic one, but if the user has the freedom to create fuzzy queries over a list of database attributes, these situations might lead (for example in the testing of a software application) to speculation of general applicability of fuzzy queries in the practice. This kind of query could appear either as a mistake during the construction of the condition or by purpose during testing application. There is a simple solution: check, before query realization, whether the user doubled the same atomic condition (semantics of query). If it is true, then the doubled atomic condition should be excluded from query. It especially holds, when other than minimum or nilpotent minimum t-norms are used. Developers of classical queries do not need to focus their attention on this problem, because classical conjunction is idempotent.

A powerful scenario for using flexible queries consists of the topics mentioned in Sects. 2.2, 2.4 and 2.5. In the first step, parameters of linguistic terms, preferences, bipolarities between elementary conditions, threshold and appropriate aggregation functions are created in cooperation between user and application. In this way occurrence of empty and overabundant answers is significantly mitigated. Even though, empty and overabundant answers might appear. In this case, software and user should cooperate to solve these problems.

# References

1. Andreasen, T., Pivert, O.: On the weakening of fuzzy relational queries. In: 8th International Symposium on Methodologies for Intelligent Systems, pp. 144–153, Charlotte (1994)
2. Bosc, P., Brando, C., Hadjali, A., Jaudoin, H., Pivert, O.: Semantic proximity between queries and the empty answer problem. In: Joint IFSA-EUSFLAT Conference, pp. 259–264, Lisbon (2009)
3. Bosc, P., Pivert, O.: On a fuzzy bipolar relational algebra. Inf. Sci. **219**, 1–16 (2013)
4. Bosc, P., Pivert, O.: On four noncommutative fuzzy connectives and their axiomatization. Fuzzy Sets Syst. **202**, 42–60 (2012)
5. Bosc, P., Pivert, O.: SQLf query functionality on top of a regular relational database management system. In: Pons, O., Vila, M.A., Kacprzyk, J. (eds.) Knowledge Management in Fuzzy Databases, pp. 171–190. Springer, Berlin, Heidelberg (2000)
6. Bosc, P., Pivert, O.: SQLf: a relational database language for fuzzy querying. IEEE Trans. Fuzzy Syst. **3**, 1–17 (1995)
7. Bosc, P., Hadjali, A., Pivert, O.: Empty versus overabundant answers to flexible relational queries. Fuzzy Sets Syst. **159**, 1450–1467 (2008)
8. Bosc, P., Hadjali, A., Pivert, O.: Weakening of fuzzy relational queries: and absolute proximity relation-based approach. Mathware Soft Comput. **14**, 35–55 (2007)
9. Bosc, P., Pivert, O., Smits, G.: On a fuzzy group-by and its use for fuzzy association rule mining. In: 14th East-European Conference on Advances in Databases and Information Systems (ADBIS'10), pp. 88–102, Novi Sad (2010)
10. Bosc, P., Hadjali, A., Pivert, O., Smits, G.: An approach based on predicate correlation to the reduction of plethoric answer sets. In: Guillet, F., Ritschard, G., Zighed, D.A. (eds.) Advances in Knowledge Discovery and Management, Studies in Computational Intelligence, vol. 398, pp. 213–233. Springer, Berlin, Heidelberg (2012)
11. Chamberlin, D.D., Boyce, R.F.: SEQUEL: A structured english query language. In: The ACM SIGMOD Workshop on Data Description, Access and Control, pp. 249–264, Ann Arbour (1974)
12. Cox, E.: Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration. Morgan Kaufman, San Francisco (2005)
13. Date, C.J., Darwen, H.: A Guide to SQL Standard, 4th edn. Addison-Wesley, Boston (1996)
14. Dubois, D., Prade, H.: Handling bipolar queries in fuzzy information processing. In: Galindo, J. (ed.) Handbook of Research on Fuzzy Information Processing in Databases, pp. 97–114. Information Science Reference, Hershey (2008)
15. Dubois, D., Prade, H.: Using fuzzy sets in flexible querying: why and how? In: Andreasen, T., Christiansen, H., Larsen, H.L. (eds.) Flexible Query Answering Systems, pp. 45–60. Kluwer Academic Publishers, Dordrecht (1997)
16. Dubois, D., Prade, H.: Weighted minimum and maximum operations in fuzzy set theory. Inf. Sci. **39**, 205–210 (1986)
17. Garibaldi, J.M., John, R.I.: Choosing membership functions of linguistic terms. In: 12th IEEE International Conference on Fuzzy Systems (FUZZ'03), pp. 578–583, St. Louis (2003)

18. Gupta, M.M., Oi, J.: Theory of t-norms and fuzzy inference methods. Fuzzy Sets Syst. **40**, 431–450 (1991)
19. Hudec, M.: Constraints and wishes in quantified queries merged by asymmetric conjunction. In: International Conference Fuzzy Management Methods (ICFM square 2016), Fribourg (2016)
20. Hudec, M., Vučetić, M.: Some issues of fuzzy querying in relational databases. Kybernetika **51**, 994–1022 (2015)
21. Hudec, M.: Improvement of data collection and dissemination by fuzzy logic. In: Joint UNECE/Eurostat/OECD Meeting on the Management of Statistical Information Systems (MSIS 2013), Paris-Bangkok (2013)
22. Hudec, M.: Dynamically modelling of fuzzy sets for flexible data retrieval. In: 46th Scientific meeting of the Italian statistical society, Rome (2012)
23. Hudec, M.: Fuzzy improvement of the SQL. Yugoslav. J. Oper. Res. **21**, 239–251 (2011)
24. Hudec, M.: An approach to fuzzy database querying, analysis and realisation. Comput. Sci. Inf. Syst. **6**, 127–140 (2009)
25. Hudec, M., Sudzina, F.: Construction of fuzzy sets and applying aggregation operators for fuzzy queries. In: 14th International Conference on Enterprise Information Systems (ICEIS 2012), Proceedings, vol. 1, pp. 253–257, Wroclaw (2012)
26. Kacprzyk, J., Zadrożny, S.: Compound bipolar queries: combining bipolar queries and queries with fuzzy linguistic quantifiers. In: 8th Conference of the European Society of Fuzzy Logic and Technology (EUSFLAT 2013), pp. 848–855, Milan (2013)
27. Kacprzyk, J., Zadrożny, S.: Computing with words in intelligent database querying: standalone and internet-based applications. Inform. Sci. **134**, 71–109 (2001)
28. Kacprzyk, J., Zadrożny, S.: FQUERY for Access: fuzzy querying for windows-based DBMS. In: Bosc, P., Kacprzyk, J. (eds.) Fuzziness in Database Management Systems, pp. 415–433. Physica-Verlag, Heidelberg (1995)
29. Kacprzyk, J. Ziółkowski, A.: Database queries with fuzzy linguistic quantifiers. IEEE Trans. Syst. Man Cyber. **SMC-16**, 474–479 (1986)
30. Kacprzyk, J., Pasi, G., Vojtáš, P., Zadrożny, S.: Fuzzy querying: issues and perspectives. Kybernetika **36**, 605–616 (2000)
31. Klement, E.P., Mesiar, R., Pap, E.: Triangular norms. Position paper I: basic analytical and algebraic properties. Fuzzy Sets Syst. **143**, 5–26 (2004)
32. Klir, G., Yuan, B.: Fuzzy Sets and Fuzzy Logic. Theory and Applications. Prentice Hall, New Jersey (1995)
33. Lacroix, M., Lavency, P.: Preferences: putting more knowledge into queries. In: 13th International Conference on Very Large Databases, pp. 217–225, Brighton (1987)
34. Pivert, O., Bosc, P.: Fuzzy Preference Queries to Relational Databases. Imperial College Press, London (2012)
35. Radojević, D.: Interpolative realization of Boolean algebra as a consistent frame for gradation and/or fuzziness. In: Nikravesh, M., Kacprzyk, J., Zadeh, L.A. (eds.) Forging new frontiers: fuzzy pioneers II, Studies in fuzziness and soft computing, pp. 295–317. Springer, Berlin, Heidelberg (2008)
36. Radojević, D.: [0, 1] Valued logic: a natural generalization of Boolean logic. Yugoslav J. Oper. Res. **10**, 185–216 (2000)
37. Ribeiro, R.A., Moreira, A.M.: Fuzzy query interface for a business database. Int. J. Human-Comput. Stud. **58**, 363–391 (2003)
38. Rosado, A., Ribeiro, R.A., Zadrożny, S., Kacprzyk, J.: Flexible query languages for relational databases: an overview. In: Bordogna, G., Psaila, G. (eds.) Flexible Databases Supporting Imprecision and Uncertainty. Studies in fuzziness and soft computing, vol. 203, pp. 3–53. Springer, Berlin, Heidelberg (2006)
39. Smits, G., Pivert, O., Girault, T.: ReqFlex: Fuzzy queries for everyone. In: 39th International Conference on Very Large Data Bases, pp. 1206–1209, Trento (2013)
40. Smits, G., Pivert, O., Hadjali, A.: Fuzzy cardinalities as a basis to cooperative answering. In: Pivert, O., Zadrożny, S. (eds.) Flexible Approaches in Data, Information and Knowledge Management. Studies in Computational Intelligence, vol. 497, pp. 261–289. Springer International Publishing Switzerland (2014)

41. Sözat, M.I., Yazici, A.: A complete axiomatization for fuzzy functional and multivalued dependencies in fuzzy database relations. Fuzzy Sets Syst. **117**, 161–181 (2001)
42. Tahani, V.: A conceptual framework for fuzzy query processing: a step toward very intelligent database systems. Inf. Process. Manag. **13**, 289–303 (1977)
43. Tudorie, C.: Intelligent interfaces for database fuzzy querying. Ann Dunarea de Jos Univ. Galati, Fascicle III, **32**(2), Galati (2009)
44. Tudorie, C.: Qualifying objects in classical relational database querying. In: Galindo, J. (ed.) Handbook of Research on Fuzzy Information Processing in Databases, pp. 218–245. Information Science Reference, Hershey (2008)
45. Urrutia, A., Pavesi, L.: Extending the capabilities of database queries using fuzzy logic. In: Collecter-LatAm conference, Santiago (2004)
46. Vučetić, M., Vujošević, M.: A literature overview of functional dependencies in fuzzy relational database models. Tech. Technol. Educat. Manag. **7**, 1593–1604 (2012)
47. Wang, T-C., Lee, H-D., Chen, C-M.: Intelligent queries based on fuzzy set theory and SQL. In: 39th Joint Conference on Information Science, pp. 1426–1432, Salt Lake City (2007)
48. Zadrożny, S., Kacprzyk, J.: Bipolar queries: a way to enhance the flexibility of database queries. In: Ras, Z.W., Dardzinska, A. (eds.) Advances in Data Management, Studies in Computational Intelligence, vol. 223, pp. 49–66. Springer, Berlin, Heidelberg (2009)
49. Zadrożny, S., de Tré, G., de Caluwe, R., Kacprzyk, J.: An overview of fuzzy approaches to flexible database querying. In: Galindo, J. (ed.) Handbook of Research on Fuzzy Information Processing in Databases, pp. 34–55. Information Science Reference, Hershey (2008)