STUDIA INFORMATICA 2003
Volume 24 Number 2A (53)

Bożena MAŁYSIAK
Politechnika Śląska, Instytut Informatyki

WARTOŚCI ROZMYTE W PYTANIACH SQL DO BAZ DANYCH

Streszczenie. W artykule przedyskutowano możliwe miejsca występowania wartości rozmytych w pytaniach SQL kierowanych do bazy danych. Każda fraza instrukcji *SELECT* została dokładnie przeanalizowana. Wystąpienie wartości rozmytych zostało zilustrowane konkretnym przykładem.

Słowa kluczowe: wartości rozmyte, teoria zbiorów rozmytych, bazy danych, pytania SQL.

FUZZY VALUES IN SQL QUERIES OF THE DATABASES

Summary. This article discussed possible places of fuzzy values appearance in the SQL queries directed to the databases. Each phrase of instruction *SELECT* has been carefully analyzed fuzzy values, fuzzy sets theory, databases, SQL queries. The appearance of fuzzy values has been shown by detailed example.

Keywords: fuzzy values, fuzzy sets theory, databases, SQL queries.

1. Wprowadzenie

Pytania, które użytkownik kieruje do bazy danych, są przez niego w pierwotnej postaci formułowane w języku naturalnym. W sformułowaniu takich pytań należy się spodziewać określeń nieprecyzyjnych lub nie mających swoich odpowiedników w języku SQL. W niniejszym artykule przedstawiono analizę struktury zapytań w języku SQL, w kontekście umiejscowienia w nich wartości rozmytych.

Warunki w pytaniach formułowanych przez użytkownika w języku naturalnym przenoszą się zwykle na warunki filtrujące, wobec czego właśnie te należy poddać szczegółowej analizie.

Rozpatrując warunki filtrujące można podzielić je na:

- występujące we frazie *WHERE* (nakładane na poszczególne wiersze),
- występujące we frazie *HAVING* (nakładane na wyodrębnione grupy wierszy).

Wartości rozmyte mogą wystąpić również we frazie *GROUP BY* i *ORDER BY*.

2. Pojęcia podstawowe

2.1. Składnia instrukcji SELECT

Operatory i funkcje rozmyte wykorzystywane są głównie w konstrukcjach dotyczących wyszukiwania informacji w bazie danych, opartych na instrukcji *SELECT*.

Instrukcja *SELECT* posiada składnię (wersja uproszczona):

```
SELECT [DISTINCT] ciąg nazw kolumn, [ciąg wyrażeń] | *
FROM ciąg nazw tabel, [podzapytanie]
[WHERE warunek]
[ORDER BY ciąg nazw kolumn [ASC| DESC]]
[GROUP BY ciąg nazw kolumn]
[HAVING warunek]
```

W klauzuli *SELECT* określa się nazwy kolumn, których wartości są wyszukiwane z wierszy spełniających warunki filtrujące. Gdy w klauzuli *SELECT* występuje symbol „*", w odpowiedzi zostaje umieszczony cały wiersz. W klauzuli *SELECT* mogą się również pojawić zapisy wyrażeń – wtedy w odpowiedzi wyznaczane są wartości tych wyrażeń. Powtarzające się wiersze nie są automatycznie eliminowane i usuwane z odpowiedzi na zapytanie. Można to zapewnić słowem kluczowym *DISTINCT*.

Klauzula *FROM* służy do określania tabel wykorzystywanych w zapytaniu. W klauzuli *WHERE* formułuje się warunki, które określają ograniczenia, jakie mają spełniać wiersze, by zostały wybrane w danym zapytaniu. Jeśli w klauzuli *WHERE* występuje więcej niż jeden warunek, warunki te są połączone ze sobą za pomocą operatorów logicznych *AND* lub *OR*.

Język SQL umożliwia porządkowanie wyników wyszukiwania przez zastosowanie klauzuli *ORDER BY*, po której wypisuje się nazwy kolumn, względem których ma przebiegać porządkowanie.

Gdy informacje wyszukiwane w bazie danych dotyczą kilku tabel, tabele te łączone są ze sobą, zwykle przy występowaniu równości klucza głównego jednej tabeli z kluczem obcym innej. W ogólnym przypadku warunek złączenia dwóch (lub więcej) tabel może być zupełnie dowolny, a połączenie między tabelami może wystąpić między dwoma dowolnymi kolumnami o tej samej dziedzinie.

Wyszukiwane dane mogą zostać przetworzone przy użyciu jednej z dostępnych funkcji agregujących (*COUNT, AVG, SUM, MAX, MIN*). W języku SQL możliwe jest dokonanie podziału wynikowych wierszy zapytania na grupy i wykonania funkcji agregujących na wartościach należących do poszczególnych grup. Do wyodrębnienia grup służy klauzula *GROUP BY*, po której wypisuje się nazwy kolumn, których jednakowe wartości stanowią kolejne grupy wierszy.

Tak jak na pojedyncze wiersze w tabeli można nałożyć warunki we frazie *WHERE*, tak i na wyodrębnione grupy można również nakładać warunki, w tym celu wykorzystując klauzulę *HAVING*.

2.2. Zbiory rozmyte

Teoria zbiorów rozmytych została wprowadzona przez L.A. Zadeha. Podstawowym pojęciem tej teorii jest zbiór rozmyty, zdefiniowany jako [7] zbiór par, w pewnej numerycznej przestrzeni rozważań X

$$A = \{(\mu_A(x), x)\}, \text{ dla każdego } x \in X,$$

gdzie:

μ_A – **funkcja przynależności zbioru rozmytego A**, która każdemu elementowi zbioru $x \in X$ przypisuje **stopień jego przynależności** $\mu_A(x)$ do zbioru A (przy czym: $\mu_A(x) \in [0,1]$).

Zbiory rozmyte dopuszczają częściową przynależność obiektów do zbioru (w klasycznych zbiorach – element zbioru należy lub nie do zdefiniowanego zbioru, funkcja charakterystyczna przyjmuje wtedy odpowiednio wartość 1 lub 0).

W teorii zbiorów rozmytych funkcja charakterystyczna została zastąpiona funkcją przynależności, która określa, czy dany element zbioru na pewno do niego należy ($\mu_A(x) = 1$), czy być może należy w pewnym stopniu ($\mu_A(x) > 0$) lub z całą pewnością nie należy do zdefiniowanego zbioru ($\mu_A(x) = 0$).

Liczba rozmyta to zbiór rozmyty określony na uniwersum rzeczywistym. Jest to zbiór normalny, wypukły o funkcji przynależności przedziałami ciągłej [3].

Przykłady liczb rozmytych to: około zera, mniej więcej 5, trochę więcej niż 9, mniej więcej pomiędzy 10 i 12.

Liczba rozmyta typu *LR* – reprezentowana jest przez trzy parametry [4]:

$L_l = (m_l, a_l, b_l)$, gdzie m_i – oznacza wartość modalną liczby (wartość, dla której funkcja przynależności przyjmuje wartość 1), a i b – określają rozrzuty liczby: lewostronny i prawostronny.

Przedział rozmyty typu *LR* – wartość rozmyta reprezentowana przez cztery parametry: $P_i = (m_i, n_i, a, b)$, gdzie m_i i n_i – oznaczają przedział wartości modalnych, a i b – określają rozrzuty przedziału: lewostronny i prawostronny.

Zmienna lingwistyczna to wielkość, która przyjmuje jako swe wartości nie zwykłe wartości numeryczne, lecz **wartości lingwistyczne** [5]. Każdej wartości zmiennej lingwistycznej przyporządkować można zbiór rozmyty [6].

Przykładem zmiennej lingwistycznej może być wielkość o nazwie *szybkość*. Zmienna ta może przyjmować różne wartości lingwistyczne, takie jak: *bardzo mała, mała, średnia, duża, bardzo duża*. Wartości lingwistyczne można opisać numerycznie za pomocą funkcji przynależności.

Wartość dokładna – pojęcie ogólne, określające wartości nierozmyte przechowywane w bazie danych, jak również dokładne (nierozmyte) warunki kryteriów podawane w zapytaniu; wartością dokładną może być liczba, cyfra, tekst itd.

Wartość rozmyta – pojęcie ogólne określające zarówno liczbę rozmytą, jak również wartość lingwistyczną, a także zbiór rozmyty; zarówno wartość lingwistyczna, jak i liczba rozmyta reprezentowane są przez zbiory rozmyte. O ile wartość lingwistyczna może być zdefiniowana zarówno na zbiorach elementów będących liczbami rzeczywistymi (*wysoki wzrost*), jak i nie będących liczbami (*dobre samopoczucie*), to liczba rozmyta zdefiniowana jest zawsze w zbiorach liczb rzeczywistych [2]. Zbiory podstawowe mogą zawierać zarówno wartości lingwistyczne, jak i liczby rozmyte, dlatego często nie rozróżnia się tych pojęć, stosując pojęcie zbioru rozmytego jako najbardziej ogólne. W tym artykule pojęcie zbioru rozmytego utożsamione zostało z pojęciem wartości rozmytej.

Stopień przynależności – w pracy będzie rozumiany jako stopień, z jakim wiersz spełnia kryteria podane w zapytaniu.

Rozmyty system relacyjny – to dla uproszczenia zapisu relacyjny system baz danych, w którym dopuszcza się występowanie rozmytych danych i/lub rozmytych zapytań, w odróżnieniu od nierozmytych, czyli klasycznych systemów relacyjnych,

Proces fuzzyfikacji - określany także mianem **rozmywania**, polega na transformacji wartości z dziedziny liczb rzeczywistych na wartość z dziedziny zbiorów rozmytych.

Proces defuzzyfikacji, zwany również **wyostrzaniem**, jest przekształceniem odwrotnym do rozmywania, czyli transformacją wartości z dziedziny zbiorów rozmytych do dziedziny liczb rzeczywistych.

W artykule **proces defuzzyfikacji** będzie rozumiany jako proces wyboru konkretnych wierszy w odpowiedzi na niedokładne (rozmyte) pytanie; inaczej mówiąc, proces uzyskiwania precyzyjnej odpowiedzi na nieprecyzyjne (rozmyte) pytanie.

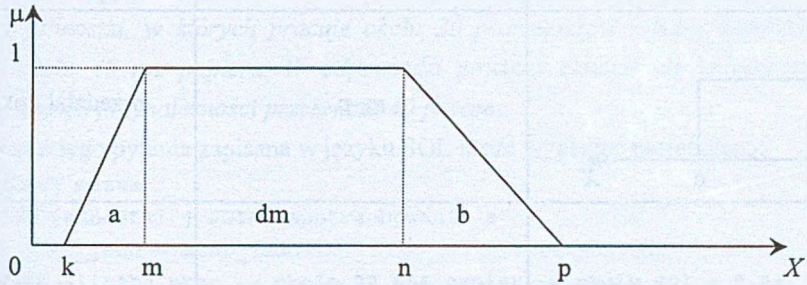
3. Struktura bazy danych

Przedstawione przykłady zapytań dotyczą utworzonej w SZBD PostgreSQL bazy danych *ZAKŁADY*, w skład której wchodzą między innymi następujące tabele:

```
Jednostki (nr_jedn, nazwa, liczba_prac, liczba_pokoi)
Pracownicy (nr_prac, nr_jedn, nazwisko, wiek, płeć, staż_pracy)
Zapotrzebowanie (nr_jedn, rok, papier, toner, płytki CD, skoroszyty)
Zużycie (nr_jedn, rok, papier, toner, płytki CD, skoroszyty)
```

Nazwy kolumn wchodzących w skład kluczy głównych w tabelach zostały podkreślone, natomiast nazwy kolumn zawierających wartości rozmyte zostały pogrubione.

W systemie został utworzony nowy typ danych o nazwie *ftrapezium*, służący do przechowywania informacji rozmytej zdefiniowanej za pomocą trapezowej funkcji przynależności. Postać graficzną typu *ftrapezium* przedstawia rys.1.



Rys. 1. Trapezowa funkcja przynależności
Fig. 1. Trapezoidal membership function

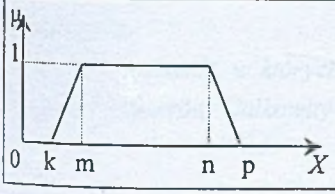
Gdzie: k, m, n, p – wartości należące do dziedziny *X* liczby;

a = m – k, dm = n – m, b = p – n.

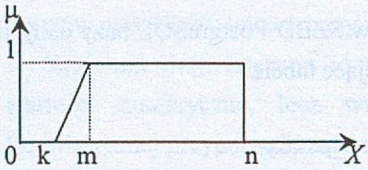
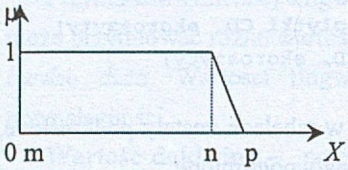
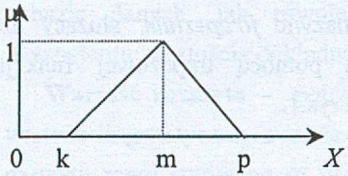
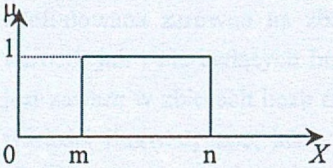
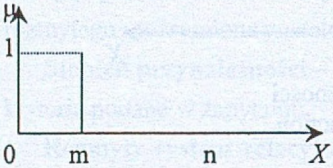
Typ ten może reprezentować także wszystkie pochodne funkcji trapezowej, jak przedstawiono w tabeli 1.

Tabela 1

Pochodne funkcji trapezowej

Postać graficzna	Reprezentacja zewnętrzna	Opis
1	2	3
	k/m~n\p	przedział rozmyty

cd. tabeli 1

1	2	3
	k/m~n	przedział rozmyty
	m~n\p	przedział rozmyty
	k/m\p	liczba rozmyta
	m~n	przedział dokładny
	m	wartość dokładna

4. Wartości rozmyte w pytaniach SQL

Wartości rozmyte mogą pojawić się w każdej frazie instrukcji *SELECT:* jako warunki filtrujące we frazach *WHERE* i *HAVING*, jako elementy, po których grupujemy we frazie *GROUP BY*, czy jako elementy, względem których porządkuje się wiersze wynikowe we frazie *ORDER BY*.

Wreszcie elementy rozmyte mogą wystąpić we frazie *SELECT*, gdy chce się wyświetlić kolumny zawierające informacje rozmyte czy stopnie przynależności wierszy do kryterium pytania.

4.1. Warunki filtrujące we frazie WHERE

Podobnie jak w klasycznych pytaniach SQL, w pytaniach rozmytych we frazie *WHERE* może wystąpić jeden warunek lub kilka warunków filtrujących, połączonych ze sobą spójnikami. W klasycznym języku SQL warunki łączone są spójnikami *AND* (iloczyn logiczny) lub *OR* (suma logiczna). W przypadku rozmytych warunków filtrujących są to odpowiednio operatory iloczynu i sumy rozmytej. W rozważanych przykładach wykorzystano zaimplementowany operator iloczynu rozmytego &&& (realizujący operację wyznaczania wartości minimalnej dwóch stopni przynależności, stosując S-normę Zadeha) i sumy rozmytej ||| (realizujący operację wyznaczania wartości maksymalnej dwóch stopni przynależności, stosując T-normę Zadeha).

Przykładowe pytanie kierowane do bazy danych może brzmieć następująco:

„Wyświetl jednostki, w których pracuje około 20 pracowników i które zamówiły w roku bieżącym około 40 ryz papieru. W odpowiedzi powinny znaleźć się te wiersze, których całkowity stopień przynależności przekracza 40 procent."

Postać tego pytania zapisana w języku SQL może wyglądać następująco:

```
SELECT nazwa
FROM jednostki j JOIN zapotrzebowanie z
ON j.nr_jedn = z.nr_jedn
WHERE (liczba_prac ~= okolo 20 &&& papier ~= okolo 40) > 0.4;
```

Ogólnie można stwierdzić, że dla każdego analizowanego wiersza wyznaczane są stopnie przynależności każdej kolumny wyspecyfikowanej w pytaniu do odpowiednich zbiorów rozmytych. Operacja ta realizowana jest przy zastosowaniu operatora ~=, wyznaczającego stopień przynależności wartości kolumny do zbioru rozmytego, podanego w pytaniu. Następnie wyznaczany jest całkowity stopień przynależności wiersza do warunków pytania. Stopień ten w przypadku spójnika &&& wyznaczany jest jako iloczyn rozmyty. Na końcu w celu otrzymania zbioru wynikowego trzeba zastosować wybraną metodę defuzzyfikacji (w tym przypadku wyświetlamy wiersze o całkowitym stopniu przynależności większym niż 0.4).

Przykładowe pytanie kierowane do bazy danych wykorzystujące spójnik ||| może brzmieć następująco:

„Wyświetl jednostki, w których pracuje około 20 pracowników lub które złożyły zamówienie na około 5 tonerów. Całkowity stopień przynależności nie może przekraczać 70 procent."

Postać tego pytania zapisana w języku SQL może wyglądać następująco:

```
SELECT nazwa
FROM jednostki j JOIN zapotrzebowanie z
ON j.nr_jedn = z.nr_jedn
WHERE (liczba_prac ~= okolo 20 &&& toner ~= okolo 5) <= 0.7;
```

Całkowity stopień przynależności wiersza do warunków pytania w przypadku spójnika ||| wyznaczany jest jako suma rozmyta. Na końcu w celu otrzymania zbioru wynikowego trzeba zastosować wybraną metodę defuzzyfikacji (na przykład wyświetlić wiersze, ze stopniem przynależności mniejszym bądź równym 0.7).

Przypadki te można rozszerzyć na dowolną liczbę warunków.

Realizacja defuzzyfikacji dla każdego warunku filtrującego oddzielnie

Jeśli we frazie *WHERE* wystąpi kilka warunków filtrujących, istnieje możliwość wykonania defuzzyfikacji dla każdego z nich oddzielnie. Sprowadza się to do wyznaczenia stopnia przynależności dla każdego warunku (przy zastosowaniu operatora ~=, wyznaczającego stopień przynależności wartości rozmytej kolumny do warunku pytania), a następnie wykonania procesu defuzzyfikacji, który przekształci stopień przynależności w wartości logiczne (*TRUE* - spełnia, *FALSE* – nie spełnia). Warunki filtrujące mogą być połączone spójnikami logicznymi (*AND, OR*).

Przykładowe pytanie kierowane do bazy danych może brzmieć następująco:

„Wyświetl jednostki, w których pracuje około 25 pracowników (warunek powinien być spełniony w co najmniej 50 procentach), które złożyły zamówienie na 15 ryz papieru (warunek powinien być spełniony w dokładnie 80 procentach) lub 4 tonery (warunek powinien być spełniony w ponad 75 procentach)."

Postać tego pytania zapisana w języku SQL może wyglądać następująco:

```
SELECT nazwa
FROM jednostki j JOIN zapotrzebowanie z
ON j.nr_jedn = z.nr_jedn
WHERE (liczba_prac ~= okolo 25) >= 0.5 AND
((z.papier ~= 15) = 0.8 OR (z.toner ~= 4) > 0.75);
```

4.2. Warunki filtrujące we frazie *HAVING*

Wyznaczanie stopni przynależności między wartością dokładną a wartością rozmytą, dotyczące frazy *WHERE*, we frazie *HAVING* są podobne.

W przypadku gdy warunki filtrujące nakładane są na kolumny zarówno we frazie *WHERE*, jak i we frazie *HAVING*, zadanie wyszukiwania realizowane jest w ten sam sposób,

jak w poprzednich przykładach, z tą różnicą, że po wybraniu wierszy spełniających warunki filtrujące we frazie *WHERE*, wyodrębniane są grupy wierszy, na które nakładane są warunki wyspecyfikowane we frazie *HAVING*.

W pytaniach z frazą *HAVING* dwa razy wyznaczane są stopnie przynależności, najpierw dla poszczególnych wierszy, później dla grup zawierających już tylko te wiersze, które zostały wyodrębnione w pierwszym procesie defuzzyfikacji.

Jeśli we frazie *HAVING* nakładane są warunki na kolumny zawierające dane typu rozmytego, zadanie wyszukiwania realizowane jest również w ten sam sposób, co w poprzednim punkcie, z tą różnicą, że w pytaniach, w których konieczne jest wyznaczenie wartości funkcji agregujących: *SUM, AVERAGE, MIN, MAX* operujących na wartościach rozmytych, konieczna jest znajomość arytmetyki liczb rozmytych, czyli operacji takich jak dodawanie, wyznaczanie wartości najmniejszej czy największej liczb rozmytych. W artykule założono, że liczby rozmyte są typu *L-R*.

Przykładowe pytanie kierowane do bazy danych może brzmieć następująco:

„Wyświetl te jednostki, których największe zamówienie na papier wynosiło około 100 ryz. W odpowiedzi powinny znaleźć się wiersze o stopniu przynależności co najmniej 70 procent."

Postać tego pytania zapisana w języku SQL może wyglądać następująco:

```
SELECT nr_jedn
FROM zapotrzebowanie
GROUP BY nr_jedn
HAVING (max(papier)~= okolo 100) >= 0.7;
```

4.3. Grupowanie według kolumn zawierających wartości rozmyte *GROUP BY*

Można rozpatrzyć kilka rozwiązań realizacji frazy *GROUP BY*.

Jednym z możliwych rozwiązań, gdy wartości rozmyte reprezentowane są w postaci liczb rozmytych, jest grupowanie ich względem wartości modalnej. Wartości rozmyte o jednakowej wartości modalnej będą stanowiły grupę.

Innym, nieco bardziej naturalnym rozwiązaniem mogłoby być ustalenie przedziału możliwych wartości modalnych stanowiących jedną grupę (np. liczby rozmyte *około 20* i *około 22* stanowią grupę, a już *około 25* należy do innej). Rozwiązanie to dopuszcza jednak, by jedna wartość pojawiła się w dwóch różnych grupach. Jest to niezgodne z definicją grupowania. Wystąpienie wartości w dwóch różnych grupach mogłoby generować błędne wyniki.

Gdy wartości rozmyte reprezentowane są nie jako liczby rozmyte, lecz przedziały rozmyte, można brać pod uwagę przedział wartości modalnych, wartości rozmyte o jednakowym przedziale wartości modalnych stanowią w tym przypadku grupę.

Najbardziej ścisłym rozwiązaniem jest wzięcie pod uwagę wszystkich parametrów liczby czy przedziału rozmytego i włączanie wiersza do grupy tylko wtedy, gdy wszystkie wartości parametrów są takie same.

Przykładowe pytanie kierowane do bazy danych może brzmieć następująco:

„Wyświetl liczby jednostek zamawiających podobne ilości papieru".

Postać tego pytania zapisana w języku SQL może wyglądać następująco:

```
SELECT count(distinct nr_jedn)
FROM zapotrzebowanie
GROUP BY papier;
```

Wartości rozmyte mogą się również pojawić we frazie *GROUP BY* w pośredniej formie, na przykład, gdy chcemy grupować względem stopnia przynależności.

Przykładowe pytanie kierowane do bazy danych może brzmieć następująco:

„Wyświetl liczby jednostek o jednakowych stopniach przynależności do warunku pytania: liczba zatrudnionych pracowników około 25. Nie uwzględniać wierszy o zerowym stopniu przynależności."

Postać tego pytania zapisana w języku SQL może wyglądać następująco:

```
SELECT count(nr_jedn), liczba_prac ~= okolo 25
FROM jednostki
WHERE (liczba_prac ~= okolo 25) > 0.0
GROUP BY (liczba_prac ~= okolo 25);
```

4.4. Porządkowanie według kolumn zawierających wartości rozmyte *ORDER BY*

W przypadku gdy zapytanie wymaga posortowania względem kolumny zawierającej wartości rozmyte, podobnie jak we frazie *GROUP BY* pojawia się problem określenia, która z wartości rozmytych jest większa, która mniejsza itd. Można zaproponować rozwiązanie tego problemu w następujący sposób: w przypadku liczb rozmytych można porównywać ich wartości modalne. W przypadku przedziałów rozmytych wygodnie jest porównywać krańcowe wartości przedziału, dla którego funkcja przynależności przyjmuje wartość 1. W przypadku gdy chcemy określić, która z wartości jest mniejsza – porównujemy najmniejsze wartości modalne przedziałów, w przypadku gdy chcemy określić większą wartość porównujemy największe wartości modalne przedziałów.

Przykładowe pytanie kierowane do bazy danych może brzmieć następująco:

„Wyświetl jednostki posortowane względem liczby zamawianych ryz papieru".

Postać tego pytania zapisana w języku SQL może wyglądać następująco:

```
SELECT nr_jedn
FROM zapotrzebowanie
ORDER BY papier;
```

4.5. Wartości rozmyte we frazie *SELECT*

Fraza *SELECT* umożliwia pokazanie wartości wyliczanych. W tej frazie również mogą się pojawić wartości rozmyte: zarówno same, jak i w wyrażeniach czy funkcjach agregujących.

Do bazy danych może zostać skierowane przykładowe pytanie:

„Wyświetl nazwiska wszystkich pracowników wraz z ich stopniami przynależności do zbiorów: wiek około pięćdziesiąt lat i staż pracy około 20 lat."

Postać tego pytania w języku SQL może być wyrażona następująco:

```
SELECT nazwisko, (wiek ~= okolo 50) AS stopien_przynaleznosci_1,
       (staż_pracy ~= okolo 20) AS stopien_przynaleznosci_2
FROM pracownicy;
```

5. Podsumowanie

Rozszerzenie języka SQL przez wprowadzenie elementów teorii zbiorów rozmytych, umożliwia formułowanie zapytań w języku naturalnym zawierających nieprecyzyjnie podane warunki.

Artykuł jest efektem prowadzonych badań nad miejscami występowania wartości rozmytych w pytaniach SQL kierowanych do bazy danych zawierającej dokładne jak i rozmyte dane.

Rozważane w pracy problemy dotyczą miejsc występowania wartości rozmytych w pytaniach niezagnieżdżonych. Rozpoczęte badania są nadal kontynuowane i dotyczą występowania wartości rozmytych w zagnieżdżonych pytaniach SQL.

LITERATURA

1. Wellesley Software. SQL Język Relacyjnych Baz Danych. Wydawnictwa Naukowo-Techniczne, Warszawa 1995.
2. Yager R, Filev D.: Podstawy modelowania i sterowania rozmytego. Wydawnictwa Naukowo-Techniczne, Wiley. Warszawa 1995.
3. Łachwa A.: Rozmyty świat zbiorów, liczb, relacji, faktów, reguł i decyzji. Akademicka Oficyna Wydawnicza Exit, Warszawa 2001.
4. Piegat A.: Modelowanie i sterowanie rozmyte. Akademicka Oficyna Wydawnicza Exit, Warszawa 1999.
5. Kacprzyk J.: Wieloetapowe sterowanie rozmyte. Wydawnictwa Naukowo-Techniczne. Warszawa 2001.

6. Chojcan J., Łeski J.: Zbiory rozmyte i ich zastosowanie. Praca dedykowana Profesorowi E. Czogale. Wydawnictwo Politechniki Śląskiej, Gliwice 2001.
7. Zadeh L. A.: Fuzzy sets, Information and Control 8, 338-353, 1965.

Recenzent: Dr hab. inż. Stanisław Wołek Prof. Pol. Rzeszowskiej

Wpłynęło do Redakcji 9 kwietnia 2003 r.

Abstract

This article discussed possible places of fuzzy value appearance in the SQL queries directed to the databases. Each phrase of instruction *SELECT* has been carefully analyzed.

The appearance of fuzzy values has been shown by detailed example.

In the introduction author discussed the places in which fuzzy values can appear.

In the second chapter basic terms used in this work are presented. The first prat of this chapter includes short description of all phrases of *SELECT* command in classical SQL language. In the second part elements of fuzzy sets theory, used in this article, are defined.

In the third chapter author presents structure of the fuzzy database and fuzzy type, which are used in examples of SQL queries.

In the fourth chapter author presents places of appearance fuzzy values in SQL queries.

Fuzzy value in each phrase is described and illustrated by detailed examples.

The last chapter is the summary of realised reasearch.

Adres

Bożena MAŁYSIAK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, bozena@ivp.iinf.polsl.gliwice.pl .