

**POLITECHNIKA ŚLĄSKA W GLIWICACH  
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI  
INSTYTUT INFORMATYKI**

**Rozprawa doktorska**

**Bożena Małysiak**

**Metody aproksymacyjnego  
wyszukiwania obiektów w bazach  
danych**

**Promotor  
prof. dr hab. inż. Stanisław Kozielski**

**Gliwice, 2003**

## Spis treści

Spis treści.....	III
Spis rysunków.....	VI
Spis tabel.....	VIII
1. Wprowadzenie.....	1
1.1. Wstęp.....	1
1.2. Wybrane metody aproksymacyjnego wyszukiwania.....	4
1.2.1. Pytania wektorowe.....	5
1.2.2. Wyszukiwanie w bazie według kryteriów opartych na prawdopodobieństwie... ..	10
1.3. Zastosowanie teorii zbiorów rozmytych do wyszukiwania w bazach danych.....	14
1.3.1. Zadawanie niedokładnych (rozmytych) zapytań do bazy danych.....	15
1.3.2. Zapamiętywanie rozmytych informacji w bazach danych.....	16
1.4. Cele i teza pracy.....	18
2. Bazy danych i zbiory rozmyte – podstawowe pojęcia wykorzystywane w pracy.....	19
2.1. Relacyjne bazy danych.....	19
2.1.1. Pojęcia podstawowe.....	19
2.1.2. Język SQL.....	20
2.1.3. Przekształcanie zagnieżdżonych zapytań SQL w niezagnieżdżone.....	25
2.2. Teoria zbiorów rozmytych.....	32
2.2.1. Zbiory rozmyte – pojęcia podstawowe.....	32
2.2.2. Operacje na zbiorach rozmytych.....	35
2.2.3. Arytmetyka liczb rozmytych.....	37
2.2.4. Kwantyfikatory lingwistyczne.....	45
3. Konstrukcja zapytań w języku SQL, zawierających wartości rozmyte.....	47
3.1. Interpretacja rozmytych warunków filtrujących w zapytaniach SQL.....	47
3.2. Przykłady ilustrujące proces wnioskowania przybliżonego w interpretacji pytań SQL zadawanych do bazy danych.....	53
3.3. Wprowadzenie do zapisu zapytań SQL warunku na wartość stopnia zgodności.....	67
3.4. Agregacja w pytaniach rozmytych.....	73
3.4.1. Funkcje agregujące na danych rozmytych.....	73
3.4.2. Nakładanie warunków na funkcje agregujące w pytaniach rozmytych.....	73
3.4.3. Rozmyte kwantyfikatory (rozmyte lingwistyczne funkcje agregujące).....	76
3.5. Rozmyte grupowanie danych.....	78
3.5.1. Grupowanie rozmyte dokładnych danych.....	78
3.5.2. Grupowanie rozmytych danych.....	86
3.5.3. Rozmyte grupowanie rozmytych danych.....	86
3.6. Wartości rozmyte w pytaniach zagnieżdżonych.....	88
3.6.1. Pytania nieskorelowane.....	91
3.6.2. Pytania skorelowane.....	95
3.7. Zależne kontekstowo interpretacje wyrazów rozmytych.....	98
3.8. Proces przekształcania zagnieżdżonych rozmytych zapytań SQL w niezagnieżdżone rozmyte zapytania SQL.....	100
4. Implementacja rozmytego wyszukiwania danych.....	107
4.1. Wprowadzenie.....	107

4.2. Wybór odpowiedniego środowiska SZBD .....	107
4.3. Implementacja rozmytego typu danych .....	109
4.3.1. Opis typu ftrapezium .....	109
4.3.2. Reprezentacja wewnętrzna .....	110
4.3.3. Tworzenie i modyfikacja tabel rozmytych .....	110
4.3.4. Wprowadzanie, modyfikacja i usuwanie wartości typu ftrapezium .....	111
4.3.5. Wyprowadzanie wartości typu ftrapezium .....	111
4.3.6. Szczególne przypadki typu ftrapezium .....	113
4.4. Podstawowe działania na wartościach typu ftrapezium .....	115
4.4.1. Funkcje i operatory arytmetyczne .....	115
4.4.2. Funkcje i operatory relacyjne na liczbach typu ftrapezium .....	116
4.5. Funkcje i operatory wyznaczające wartość stopnia zgodności .....	117
4.6. Realizacja funktorów iloczynu, sumy i negacji rozmytych .....	119
4.7. Wartości predefiniowane dla typu ftrapezium .....	119
4.8. Funkcje agregujące na liczbach typu ftrapezium .....	121
4.8.1. Funkcje agregujące: sum, avg, min, max .....	121
4.8.2. Rozmyte kwantyfikatory .....	122
4.9. Implementacja hierarchicznego algorytmu rozmytego grupowania danych dokładnych .....	124
4.10. Porządkowanie kolumn zawierających wartości rozmyte .....	127
4.11. Funkcje i operatory realizujące warunek wiążący pytanie zewnętrzne z wewnętrznym .....	127
4.12. Ostatecznie przyjęta forma zapisu w języku SQL pytań zawierających wartości rozmyte .....	129
5. Przykład implementacji rozmytej bazy danych .....	130
5.1. Struktura bazy danych .....	130
5.2. Wykorzystanie utworzonych elementów rozmytych w pytaniach SQL .....	137
5.3. Pomiar czasu .....	144
6. Podsumowanie i wnioski .....	148
LITERATURA .....	150
ZAŁĄCZNIK A Elementy programowe mechanizmów rozmytych w SZBD PostgreSQL .....	156
1. Wstęp .....	157
2. Tworzenie typu ftrapezium .....	157
3. Działania arytmetyczne na wartościach typu ftrapezium .....	158
3.1. Rozmyte funkcje i operatory arytmetyczne .....	158
3.2. Rozmyte funkcje i operatory porównania .....	160
4. Funkcje i operatory wyznaczające wartość stopnia zgodności .....	161
5. Funktory iloczynu, sumy i negacji rozmytych .....	162
6. Wartości predefiniowane dla typu ftrapezium .....	163
7. Funkcje konwersji typu ftrapezium do postaci alfanumerycznej .....	163
8. Funkcje agregujące na wartościach typu ftrapezium .....	163
8.1. Funkcje agregujące: sum, avg, min, max .....	163
8.2. Rozmyte kwantyfikatory .....	165
9. Funkcje i operatory realizujące rozmyte grupowanie danych .....	168
10. Funkcje i operatory rozmyte realizujące rozmyty warunek wiążący pytanie zewnętrzne z wewnętrznym .....	170
10.1. Rozszerzenie typu ftrapezium - ftrapeziumext .....	170
10.2. Porównanie wartości rozmytej z wartością dokładną .....	172

---

10.3.	Porównanie wartości dokładnej z wartością rozmytą .....	173
10.4.	Porównanie wartości rozmytej z wartością rozmytą .....	174
ZAŁĄCZNIK B Ilustracja rozmytego grupowania danych.....		176
1.	Wprowadzenie .....	177
2.	Dane wejściowe rozłożone równomiernie .....	177
3.	Dane wejściowe rozłożone w pewnych skupiskach .....	182

## Spis rysunków

Rys. 2.1 Schemat bazy danych <i>ZAKŁADY</i> .....	23
Rys. 2.2. Funkcja przynależności podzbioru rozmytego ' <i>młody człowiek</i> ' .....	33
Rys. 2.3 Funkcja przynależności zbioru rozmytego <i>około 30</i> .....	34
Rys. 2.4. Zasada rozszerzania Zadeha .....	38
Rys. 2.5. Charakterystyczne parametry dodawanych liczb rozmytych .....	41
Rys. 3.1 Przecięcie wartości rozmytej <i>około 20</i> z wartością precyzyjną 17 .....	49
Rys. 3.2 Przecięcie wartości rozmytej <i>A około 20</i> zdefiniowanej przez trójkątną funkcję przynależności z wartością rozmytą <i>X</i> (np. <i>młody</i> ) zdefiniowaną przez trapezową funkcję przynależności .....	49
Rys. 3.3 Funkcja przynależności typu krzywa Gaussa .....	54
Rys. 3.4 Trójkątna symetryczna funkcja przynależności .....	55
Rys. 3.5 Wartość rozmyta <i>wiek około 50</i> wyrażona funkcją Gaussa .....	55
Rys. 3.6 Wartość rozmyta <i>wiek około 50</i> wyrażona symetryczną funkcją trójkątną .....	56
Rys. 3.7 Wartość rozmyta <i>staż pracy około 20</i> wyrażona funkcją Gaussa .....	56
Rys. 3.8 Wartość rozmyta <i>staż pracy około 20</i> wyrażona symetryczną funkcją trójkątną .....	57
Rys. 3.9 Funkcja przynależności dla wartości lingwistycznej <i>prawie wszyscy</i> .....	77
Rys. 3.10 Stopień zgodności obliczonego odsetka (0.925) z wartością lingwistyczną <i>prawie wszystkie</i> .....	77
Rys. 3.11 Schemat blokowy zaimplementowanego algorytmu grupowania .....	85
Rys. 3.12 Trapezoidalna parametryczna funkcja przynależności .....	99
Rys. 4.1 Trapezowa funkcja przynależności .....	109
Rys. 4.2 Stopień zgodności między wartością rozmytą <i>f</i> a wartością dokładną <i>x</i> .....	117
Rys. 4.3 Przedziały zgodności nachodzą na siebie .....	117
Rys. 4.4 Lewostronne i prawostronne przedziały niepewności $f_1$ i $f_2$ przecinają się; $DG(f_1, f_2) = \mu_2$ .....	118
Rys. 4.5 Trapezy reprezentujące wartości rozmyte $f_1$ i $f_2$ są rozłączne .....	118
Rys. 4.6 Funkcje przynależności wartości lingwistycznych zmiennej <i>liczba</i> .....	120
Rys. 4.7 Funkcje przynależności wartości lingwistycznych zmiennej <i>dni</i> .....	120
Rys. 4.8 Funkcje przynależności wartości lingwistycznych zmiennej <i>sekundy</i> .....	120
Rys. 4.9 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>odsetek</i> .....	123
Rys. 4.10 Stopień zgodności obliczonego odsetka (0.92) z wartością lingwistyczną <i>prawie wszystkie</i> .....	123
Rys. 4.11 Istniejący algorytm grupowania .....	125
Rys. 5.1 Schemat bazy danych <i>ZAWODNICY</i> .....	131
Rys. 5.2 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>wiatr</i> .....	132
Rys. 5.3 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>temperatura</i> .....	132
Rys. 5.4 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>widoczność</i> .....	133

Rys. 5.5 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>senność</i>	133
Rys. 5.6 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>zmęczenie</i>	134
Rys. 5.7 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>stres</i>	134
Rys. 5.8 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>złość</i>	134
Rys. 5.9 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>wzrost mężczyzny</i>	135
Rys. 5.10 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>wzrost kobiety</i>	135
Rys. 5.11 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>wzrost człowieka</i>	135
Rys. 5.12 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>waga mężczyzny</i>	136
Rys. 5.13 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>waga kobiety</i>	136
Rys. 5.14 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>waga człowieka</i>	136
Rys. 5.15 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej <i>ocena serii</i>	137
Rys. 5.16 Wykresy czasów wykonania zapytań z wartościami rozmytymi we frazie <i>where</i> i odpowiadających im pytań dokładnych	145
Rys. 5.17 Wykresy czasów wykonania zapytań z wartościami rozmytymi we frazie <i>where</i> i <i>having</i> oraz odpowiadających im pytań dokładnych	146
Rys. 5.18 Wykresy czasów wykonania zapytań grupujących względem wartości rozmytych i odpowiadających im zapytań dokładnych	146
Rys. 5.19 Wykresy czasów wykonania rozmytych zapytań zagnieżdżonych i odpowiadających im dokładnych zapytań zagnieżdżonych	147

## Spis tabel

Tabela 1.1 Tytuły .....	7
Tabela 1.2 Dokumenty jako 18-wymiarowe wektory .....	8
Tabela 1.3 Wektor Q dla zdefiniowanego pytania .....	8
Tabela 1.4 Wektory T5 i Q .....	8
Tabela 1.5 Miary odległości i podobieństwa .....	9
Tabela 1.6 Tabela <i>Kandydaci</i> .....	12
Tabela 1.7 Cechy oraz ich prawdopodobieństwa $P(c_i s)$ i $P(c_i)$ .....	13
Tabela 3.1 Wyznaczanie stopni zgodności w instrukcji <i>SELECT</i> .....	52
Tabela 3.2 Tabela <i>Pracownicy</i> .....	53
Tabela 3.3 Tabela <i>Pracownicy</i> z obliczonymi stopniami zgodności .....	57
Tabela 3.4 Tabela <i>Pracownicy</i> z całkowitym stopniem zgodności dla każdego wiersza .....	57
Tabela 3.5 Tabela <i>Pracownicy</i> z wyliczonymi stopniami zgodności .....	58
Tabela 3.6 Tabela <i>Pracownicy</i> z całkowitym stopniem zgodności dla każdego wiersza .....	58
Tabela 3.7 Tabela <i>Zapotrzebowanie</i> .....	59
Tabela 3.8 Tabela <i>Zapotrzebowanie</i> z wyliczonymi stopniami zgodności dla każdej wartości rozmytej kolumny .....	60
Tabela 3.9 Tabela <i>Zapotrzebowanie</i> z maksymalnymi wartościami stopni zgodności kolumn rozmytych .....	60
Tabela 3.10 Tabela <i>Zapotrzebowanie</i> z obliczonym całkowitym stopniem zgodności $\tau$ dla każdego wiersza .....	61
Tabela 3.11 Tabela <i>Zapotrzebowanie</i> z wyliczonymi stopniami zgodności dla każdej wartości rozmytej kolumny .....	61
Tabela 3.12 Tabela <i>Zapotrzebowanie</i> z maksymalnymi wartościami stopni zgodności kolumny rozmytej dla każdego wiersza .....	62
Tabela 3.13 Tabela <i>Zapotrzebowanie</i> z obliczonym całkowitym stopniem zgodności $\tau$ dla każdego wiersza .....	62
Tabela 3.14 Tabela <i>Zapotrzebowanie</i> .....	63
Tabela 3.15 Tabela <i>Zapotrzebowanie</i> z wyznaczonymi stopniami zgodności .....	65
Tabela 3.16 Tabela <i>Zapotrzebowanie</i> z całkowitym stopniem zgodności .....	65
Tabela 3.17 Tabela <i>Dobrzy pracownicy</i> .....	66
Tabela 3.18 Tabela <i>Dobrzy pracownicy</i> z obliczonymi stopniami zgodności .....	66
Tabela 3.19 Tabela <i>Dobrzy pracownicy</i> z obliczonymi stopniami zgodności względem pytania .....	66
Tabela 3.20 Tabela <i>Dobrzy pracownicy</i> z obliczonym całkowitym stopniem zgodności $\tau$ ....	67
Tabela 3.21 Tabela <i>pomiary</i> .....	79
Tabela 3.22 Tabela <i>temperatura po przypisaniu</i> .....	79
Tabela 3.23 Tabela <i>wynikowa</i> .....	80
Tabela 4.1 Wiersz <i>wynikowy</i> .....	112
Tabela 4.2 Wiersze <i>wynikowe</i> .....	112
Tabela 4.3 Wartości lingwistyczne po konwersji .....	113
Tabela 4.4 Wiersze <i>wynikowe</i> ze stopniami przynależności .....	113
Tabela 4.5 Szczególne przypadki funkcji trapezowej .....	114



Tabela 5.1 Przykładowe zmienne lingwistyczne i odpowiadające im funkcje konwersji.....	137
Tabela 5.2 Rozmiary poszczególnych tabel.....	144
Tabela 5.3 Czasy wykonania zapytań (w [s]) .....	145
Tabela 2.1 Tabela <i>pracownicy</i> - eksperyment 1 .....	177
Tabela 2.2 Posortowana tabela <i>pracownicy</i> względem kolumny <i>staz_pracy</i> (eksperyment 1) .....	178
Tabela 2.3 Wynikowa tabela dla rozpiętości 3 (eksperyment 1) .....	180
Tabela 2.4 Proces sklejanie w grupy dla rozpiętości = 3 (eksperyment 1).....	180
Tabela 2.5 Wynikowa tabela dla rozpiętości 5 (eksperyment 1) .....	181
Tabela 2.6 Wynikowa tabela dla rozpiętości 10 (eksperyment 1) .....	182
Tabela 3.1 Tabela <i>pracownicy</i> - eksperyment 2 .....	182
Tabela 3.2 Posortowana tabela <i>pracownicy</i> względem kolumny <i>staz_pracy</i> (eksperyment 2) .....	183
Tabela 3.3 Wynikowa tabela dla rozpiętości 3 (eksperyment 2) .....	184
Tabela 3.4 Wynikowa tabela dla rozpiętości 5 (eksperyment 2) .....	185
Tabela 3.5 Wynikowa tabela dla rozpiętości 10 (eksperyment 2) .....	185

*Poniższa rozprawa nie powstałaby bez wsparcia świętej pamięci Profesora Adama Mrózka, który zainteresował mnie teorią zbiorów rozmytych.*

*Chciałabym również wyrazić podziękowanie mojemu Promotorowi – Profesorowi Stanisławowi Kozielskiemu za bezgraniczną cierpliwość i czas poświęcony na dyskusje, za cenne i konstruktywne uwagi, które miały istotny wpływ na ostateczną postać rozprawy.*

*Bożena Małysiak*

## 1. Wprowadzenie

### 1.1. Wstęp

W otaczającej nas rzeczywistości można wyodrębnić wiele dziedzin, w których stosowane są systemy informatyczne gromadzące duże ilości różnorodnych danych. Krytycznym problemem może być wtedy efektywne wyszukiwanie potrzebnych informacji. Zwykle, w tradycyjnych bazach danych, algorytmy wyszukiwania zapewniają znalezienie obiektów, których atrybuty ściśle spełniają warunki określone w zapytaniu. Obiekty nie spełniające precyzyjnie tych warunków nie są znajdowane.

Można zadać pytanie, czy w przypadku przeszukiwania dużych kolekcji danych byłoby korzystne zastosowanie algorytmów przybliżonego wyszukiwania obiektów i w jaki sposób realizować proces takiego wyszukiwania?

Analiza nowych kierunków rozwoju baz danych pokazuje, że obiecującym rozwiązaniem, w tradycyjnych bazach danych, jest aproksymacyjne wyszukiwanie informacji.

Analizując stopień spełnienia kryteriów podanych w pytaniu, można wyodrębnić kilka rodzajów zapytań [DcmQr00]:

- **precyzyjne**, w których dokładnie są sprecyzowane kryteria wyszukiwania, a w odpowiedzi uzyskuje się zbiór tylko tych obiektów, które spełniają podane w zapytaniu kryteria,
- **zakresowe**, w których jako warunki podane są przedziały (zakresy), a w odpowiedzi uzyskuje się tylko te obiekty, które mieszczą się w zdefiniowanym przez użytkownika przedziale,
- **aproksymacyjne**, w odpowiedzi na zadane pytania uzyskuje się zbiór obiektów spełniających w pewnym zadanym stopniu kryteria sprecyzowane w zapytaniu. Tego typu zapytania wymagają zdefiniowania tzw. funkcji charakterystycznej, która określi w jakim stopniu wyszukiwany obiekt jest zgodny z kryteriami podanymi w zapytaniu. Ze względu na rodzaj stosowanej funkcji charakterystycznej wyróżnić można następujące rodzaje zapytań aproksymacyjnych:
  - **wektorowe** (*ang. vector queries*) – funkcją charakterystyczną jest funkcja podobieństwa dwóch wektorów, z których jeden reprezentuje wyszukiwany obiekt, a drugi kryteria pytania,

- **oparte na prawdopodobieństwie** (*ang. probabilistic queries*) – funkcja charakterystyczna określona jest jako prawdopodobieństwo, z jakim obiekt związany jest z pytaniem,
- **rozmyte** (*ang. fuzzy queries*) – funkcją charakterystyczną jest funkcja określająca stopień przynależności (dopasowania) obiektu do kryteriów zapytania,
- **w języku naturalnym** (*ang. natural language queries*) – semantyka języka określa sposób interpretacji pytania.

W pierwszej części pracy przedyskutowano własności wybranych algorytmów aproksymacyjnego wyszukiwania obiektów w bazach danych, takich jak:

- algorytmy, w których obiekty (dokumenty) reprezentowane są poprzez wektory (na podstawie wartości funkcji odległości między dwoma wektorami określa się w jakim stopniu obiekty są podobne do siebie),
- algorytmy, w których stopień spełnienia kryteriów zapytania określany jest prawdopodobieństwem.

Zasadnicza część pracy dotyczy natomiast zastosowania teorii zbiorów rozmytych w wyszukiwaniu informacji w bazach danych, a szczególnie wykorzystania elementów teorii zbiorów rozmytych w zapytaniach języka SQL.

Informacja rozmyta w niektórych przypadkach jest pełniejsza i trafniej oddaje specyfikę rzeczywistości. Potrzeba stosowania logiki rozmytej rośnie wraz ze złożonością i kompleksowością baz danych.

W bazach danych istnieje kilka możliwości wykorzystania pojęcia nieostrości (rozmycia). Daje się wśród nich wyróżnić dwa problemy [Bdr99]:

- zapamiętywanie rozmytych informacji w bazie danych,
- wykorzystanie pojęcia nieostrości w językach wyszukiwania informacji w bazach danych.

Biorąc pod uwagę rodzaje informacji przechowywanych w bazie danych oraz sposoby zadawania pytań do bazy danych można wyodrębnić następujące sytuacje [Młs02a]:

1. Baza danych zawiera precyzyjne dane, tzn. wartości poszczególnych atrybutów opisujących obiekty bazy danych są określone zgodnie z ich dziedzinami, przy czym do bazy danych zadawane są precyzyjne pytania, jednoznacznie specyfikujące własności poszukiwanych obiektów.

SZBD (system zarządzania bazą danych) generuje wtedy poprawne odpowiedzi zwracające listę obiektów (lista może być także pusta), spełniających warunki wyspecyfikowane w pytaniu. Jest to klasyczny przypadek wykorzystywania baz danych, wyczerpująco zbadany, wykorzystujący różne, efektywne modele

reprezentacji danych (np. relacyjny model danych) oraz języki manipulowania danymi (SQL, QBE, QUEL i inne).

2. Baza danych zawiera precyzyjne dane (w rozumieniu jak w przypadku 1), ale przy ich wyszukiwaniu zadawane są nieprecyzyjne pytania, tzn. nie specyfikujące dokładnie własności obiektów reprezentowanych w bazie danych.

Przypadek ten ilustruje podany przykład dotyczący wyszukiwania danych w tablicy zawierającej takie dane pracowników jak: imię, nazwisko, wiek i staż pracy.

```
SELECT imie, nazwisko
FROM pracownicy
WHERE wiek jest około 50 AND staz_pracy jest około 20;
```

Gdzie wartościami atrybutów: *wiek* i *staz\_pracy* są określone wartości numeryczne.

3. Baza danych zawiera nieprecyzyjne dane np. wartości niektórych atrybutów specyfikujących jej poszczególne obiekty nie są w pełni określone lub przyjmują wartości nieprecyzyjne, np. wyrażone w języku naturalnym. Do bazy zawierającej takie dane zadawane są precyzyjne pytania. Pojawia się więc problem otrzymania zadawalających użytkownika odpowiedzi na tak postawione pytania.

Wartości atrybutów mogą być podane w postaci liczb rozmytych, przedziałów wartości lub podzbiorów rozmytych [YgrFlv95][Łchw01][Pgt99][Łsk01].

Np. wiek – około 20; między 30 a 40; 26-28; 2 albo 3; młody; średni, itd.

Do takiej bazy danych może być zadawane precyzyjne pytanie, np. o postaci:

```
SELECT imie, nazwisko
FROM pracownicy
WHERE wiek = 50 AND staz_pracy = 20
```

4. W tym przypadku zarówno baza danych zawiera nieprecyzyjne dane (w rozumieniu jak w przypadku 3) jak i zadawane są nieprecyzyjne pytania (w rozumieniu jak w przypadku 2).

Wartości atrybutów mogą być podane, tak jak w poprzednim punkcie w postaci liczb rozmytych, przedziałów wartości lub podzbiorów rozmytych.

Do takiej bazy danych zadawane jest nieprecyzyjne (rozmyte) pytanie, np. o postaci:

```
SELECT imie, nazwisko
FROM pracownicy
WHERE wiek jest około 50 AND staz_pracy jest około 20;
```

Sytuacje opisane w punktach 2 i 4 mogą wystąpić wtedy, gdy użytkownik, zainteresowany w dostępie do informacji zawartej w bazie danych, nie zna szczegółowo wartości jej atrybutów lub gdy nie potrafi sformułować precyzyjnego pytania (dla użytkownika naturalnym jest zadanie pytania przy użyciu określeń wykraczających poza ramy typowych języków zapytań). W sytuacjach tych do SZBD zostaje więc zadane

nieprecyzyjne pytanie i system ten musi je „zrozumieć”, by wyszukać w bazie danych odpowiedzi na to pytanie.

### **Układ pracy**

Część główną pracy stanowi zastosowanie teorii zbiorów rozmytych do wyszukiwania w bazach danych, ale tematyka aproksymacyjnego wyszukiwania jest szeroka i najpierw w kolejnym podrozdziale (obszernym wprowadzeniu do pracy) zostaną przedstawione inne wybrane podejścia (pytania wektorowe i wyszukiwanie w bazach danych według kryteriów opartych na prawdopodobieństwie).

Przegląd stanu wiedzy w zakresie zastosowania teorii zbiorów rozmytych w bazach danych zamieszczono w końcowej części rozdziału 1.

W kolejnej części rozprawy (rozdział 2) przytoczono formalne podstawy dotyczące wybranych pojęć z teorii baz danych i zbiorów rozmytych.

Rozdział 3 pracy przedstawia analizę interpretacji zapytań SQL zawierających wartości rozmyte. Wiele uwagi poświęcono pytaniom zagnieżdżonym, grupowaniu rozmytemu i kwantyfikatorom rozmytym.

W następnej części pracy (rozdział 4) przedstawiono proces implementacji w SZBD PostgreSQL (wykorzystując możliwości tworzenia własnych operatorów) zaproponowanych mechanizmów rozmytego wyszukiwania informacji. Następnie przeprowadzono badania eksperymentalne dla kilku przykładów zastosowań. Rozprawę kończy podsumowanie, zawierające wnioski z przeprowadzonej analizy i wykonanych badań eksperymentalnych.

W pracy pojawiają się często określenia dane (informacje, pytania) nieprecyzyjne, nieostre lub rozmyte. Przyjęto, że są to pojęcia równoważne, podobnie jak określenia dane precyzyjne, ostre, rzeczywiste.

## **1.2. Wybrane metody aproksymacyjnego wyszukiwania**

Prace dotyczące problematyki efektywnego wyszukiwania w dużych bazach danych informacji podanych w sposób nieprecyzyjny bądź niepełny, są prowadzone od kilkunastu lat w wielu ośrodkach naukowo-badawczych [WhtJn97], zwłaszcza w Stanach Zjednoczonych.

Istnieje szereg metod aproksymacyjnego wyszukiwania obiektów w bazach danych. W sytuacji, gdy w odpowiedzi mogą się znaleźć obiekty, spełniające z pewnym przybliżeniem kryteria pytania, pojawia się potrzeba wprowadzenia miar, określających w jakim stopniu wyszukiwane obiekty spełniają kryteria zadane w pytaniu.

W niniejszym podrozdziale przedstawiono metody umożliwiające formułowanie następujących rodzajów pytań przybliżonych:

- pytania wektorowe, w których obiekty reprezentowane są przez wektory cech, a miarami są funkcja odległości i funkcja podobieństwa,
- pytania oparte na prawdopodobieństwie, w których określa się prawdopodobieństwo, z jakim wybrany wiersz spełnia kryteria pytania.

### 1.2.1. Pytania wektorowe

W metodzie wykorzystującej pytania wektorowe, dla wierszy tablicy relacyjnej zawierającej  $m$  kolumn są tworzone wektory w  $n$  – wymiarowej przestrzeni cech ( $n \leq m$ ). Również na identycznej zasadzie dla pytania kierowanego do bazy danych tworzony jest wektor  $Q$  reprezentujący to pytanie. Proces poszukiwania odpowiedzi może być wtedy sprowadzony do wyszukiwania wektorów odpowiednio podobnych lub bliskich wektorowi  $Q$ .

Pytania tego typu często stosowane są w bazach danych zawierających dokumenty tekstowe zwane dalej dokumentami. Dla każdego dokumentu tworzony jest jego opis w postaci uporządkowanej listy wyrazów [DcmQr00].

Najprostszym modelem opisu dokumentu jest przedstawienie go za pomocą wektora, którego składowe przyjmują tylko wartości binarne  $\{0, 1\}$ . Sposób tworzenia wektora przedstawia przykład 1.1 [Młs02b].

#### Przykład 1.1

Przyjęto następujący zbiór słów kluczowych (słownik)  $\{\text{komputer}, \text{informatyka}, \text{dane}, \text{pytania}\}$  służący do klasyfikacji dokumentów (w momencie wprowadzania do bazy danych), a następnie do kodowania pytań. Jeśli dokument w bazie danych zawiera spośród nich tylko wyrazy *informatyka* i *pytania*, to będzie on reprezentowany przez wektor  $[0, 1, 0, 1]$ .

Podobieństwo między warunkami sformułowanymi w pytaniu a wyszukiwanym dokumentem w bazie danych można wyznaczyć na przykład jako sumę składowych, która dla rozważanego dokumentu wynosi:  $1 + 1 = 2$ . Dokument najbliższy zadanemu słownikowi charakteryzuje wartość „bliskości” równa  $1 + 1 + 1 + 1 = 4$ .

Inna nieco rozszerzona metoda opisu dokumentu opiera się na **wektorze wag**, którego składnikami są wartości wag związanych z wyrazami występującymi w dokumencie. Ilustruje to przykład 1.2.

#### Przykład 1.2

Przyjęto, że zbiór słów kluczowych (słownik) jest taki sam jak w przykładzie 1.1:  $\{\text{komputer}, \text{informatyka}, \text{dane}, \text{pytania}\}$  wraz z przypisanymi do nich wagami  $[20, 15, 40,$

25]. Niech pewien dokument w bazie danych zawiera spośród nich tylko wyrazy *informatyka* i *pytania*. W tej metodzie dokument ten będzie reprezentowany przez wektor o składowych [0, 15, 0, 25]. Podobieństwo między warunkami sformułowanymi w pytaniu a wyszukiwanym dokumentem w bazie danych można wyznaczyć na przykład jako sumę składowych, która dla rozważanego dokumentu wynosi:  $15 + 25 = 40$ . Dokument najbliższy zadanemu słownikowi charakteryzuje wartość „bliskości” równa  $20 + 15 + 40 + 25 = 100$ .

Składowe wektora reprezentującego dokument mogą być również związane z liczbą wystąpień wyrazu w dokumencie (im częściej wyraz występuje, tym ma większą wartość składowej). Użytkownik może również definiować wagi według swoich kryteriów.

W zależności od możliwości zastosowania określonej funkcji miary wiersze można porównywać ze sobą, wykorzystując [WhtJn97]:

- **miarę podobieństwa** dwóch wektorów,
- **miarę odległości** wektorów.

W pierwszym przypadku wybór lub odrzucenie pewnych wierszy z przeszukiwanego zbioru wierszy następuje na podstawie obliczonej wartości miary podobieństwa wyznaczonej zgodnie z kryteriami zawartymi w pytaniu.

Z tego punktu widzenia wynik zapytania może zawierać:

- pewną liczbę wierszy określaną przez użytkownika posortowanych według ich wartości podobieństwa,
- wszystkie wiersze, których wartości podobieństwa przekraczają pewien, ustalony przez użytkownika próg.

Jedną z miar podobieństwa dwóch wektorów jest **miara kątowa** [FltKng95][DzbŚwt85], w której im bliższe są kierunki wektorów, tym większa jest ich miara podobieństwa. Miara kątowa została omówiona na przykładzie miary **cosinusowej**.

**Miara cosinusowa**  $sim(d_1, d_2)$  – określa podobieństwo między dwoma wierszami (reprezentowanymi przez dwa wektory, o których założono, że leżą w jednej płaszczyźnie, ich kierunki nie są prostymi skośnymi), które wyznaczone jest jako cosinus kąta między dwoma niezerowymi wektorami:

$$sim(d_1, d_2) = \cos(\vec{u}_1, \vec{u}_2) = \frac{\vec{u}_1 \circ \vec{u}_2}{\|\vec{u}_1\| \cdot \|\vec{u}_2\|}, \quad (1)$$

gdzie:

$d_1, d_2$  – wiersze bazy danych o  $n$  atrybutach, reprezentowane w przestrzeni wektorowej przez wektory. Każdy atrybut wiersza przedstawiony jest jako składowa wektora,

$\vec{u}_1 = [u_{11}, u_{12}, \dots, u_{1n}]$ ,  $\vec{u}_2 = [u_{21}, u_{22}, \dots, u_{2n}]$  – wektory reprezentujące wiersze  $d_1, d_2$  w przestrzeni wektorowej,

- – iloczyn skalarny dwóch wektorów, wyrażony wzorem:



$$\vec{u}_1 \circ \vec{u}_2 = \sum_{i=1}^n u_{1i} u_{2i},$$

$\|\vec{u}\|$  – norma euklidesowa wektora (długość wektora), wyrażona wzorem:

$$\|\vec{u}\| = \sqrt{\sum_{i=1}^n u_i^2}$$

Za pomocą wzoru (1) wyznacza się wartość miary kątowej.

Określenie podobieństwa wierszy w bazie danych może być związane z wyznaczeniem **odległości** między nimi (obiekty położone blisko siebie w przestrzeni wektorowej są do siebie bardziej podobne niż pozostałe). Jedną z nich jest **odległość euklidesowa** [FltKng95], [DzbŚwt85].

Dobrze dobrana **odległość** powinna być:

- nieujemna  $O(d_1, d_2) \geq 0$  oraz  $O(d_1, d_2) = 0$  tylko wtedy gdy  $d_1 \equiv d_2$ ,
  - symetryczna  $O(d_1, d_2) = O(d_2, d_1)$
- oraz
- spełniać nierówność trójkąta, czyli  $O(d_1, d_3) \leq O(d_1, d_2) + O(d_2, d_3)$ .

**Odległość euklidesowa  $O(d_1, d_2)$**  – określa odległość dwóch wierszy  $d_1$  i  $d_2$ :

$$O(d_1, d_2) = \sqrt{\sum_{i=1}^n (u_{1i} - u_{2i})^2} \quad (2)$$

Zastosowanie przedstawionych miar ilustruje przykład 1.3.

### Przykład 1.3

Dana jest tablica zawierająca tytuły artykułów (tabela 1.1).

Tabela 1.1

Tytuły	
T1	Algorithms and Strategies for Similarity Retrieval.
T2	A Retrieval Technique for Similar Shapes.
T3	Database and Knowledge-Base Systems.
T4	Computing with Words in Intelligent Database Querying.
T5	A Fuzzy Query Language for Relational Database.
T6	A Relational Database Language for Fuzzy Querying.

Przyjęto następujący zbiór słów kluczowych (słownik) służący do klasyfikacji artykułów (w momencie wprowadzania do bazy danych), a następnie do formułowania pytań:

{Algorithms, Strategies, Similarity, Retrieval, Technique, Similar, Shapes, Computing, Words, Intelligent, Database, Querying, Knowledge-Base, Systems, Fuzzy, Query, Language, Relational}

Liczność tego zbioru wynosi: 18

Po alfabetycznym posortowaniu zbiór słów kluczowych jest następujący:

{Algorithms, Computing, Database, Fuzzy, Intelligent, Knowledge-Base, Language, Query, Querying, Relational, Retrieval, Shapes, Similar, Similarity, Strategies, Systems, Technique, Words}

Każdy wiersz z tabeli 1.1 może być przedstawiony jako 18-wymiarowy wektor, którego składowe określają liczbę wystąpień danego wyrazu w tekście. Przedstawiono to w tabeli 1.2.

Tabela 1.2

Dokumenty jako 18-wymiarowe wektory

T1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0
T3	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
T4	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1
T5	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0
T6	0	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0

Do bazy danych zadawane jest pytanie: *Znaleźć wiersze, które zawierają wyrazy: Retrieval, Fuzzy, Query, Database.*

Przyjmijmy, że pytanie jest reprezentowane przez wektor Q (tabela 1.3).

Tabela 1.3

Wektor Q dla zdefiniowanego pytania

Q	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Wartości miar podobieństwa i odległości zostaną obliczone dla wektora Q reprezentującego pytanie i wybranego wektora T5 reprezentującego wiersz w bazie danych. Wektory T5 i Q przedstawiono w tabeli 1.4.

Tabela 1.4

Wektory T5 i Q

T5	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0
Q	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0

Cosinusowa miara podobieństwa między wektorem dokumentu T5 a wektorem pytania Q, obliczane zgodnie ze wzorem (1) ma wartość:

$$\text{sim}(T5, Q) = \frac{1*1+1*1+1*1}{\sqrt{1+1+1+1+1}\sqrt{1+1+1+1}} = \frac{3}{\sqrt{20}} = \frac{3}{2\sqrt{5}} \approx 0,671,$$

a odległość między tymi wektorami obliczana zgodnie ze wzorem (2) wynosi:

$$O(T5, Q) = \sqrt{(1-1)^2 + (1-1)^2 + (1-0)^2 + (1-1)^2 + (1-0)^2 + (0-1)^2} = \sqrt{3} \approx 1,732$$

Obliczane w ten sam sposób miary podobieństwa i odległości między wektorami dokumentów (T1, ..., T6) a wektorem pytania Q przedstawione są w tabeli 1.5.

Tabela 1.5

Miary odległości  
i podobieństwa

Wektor	$Sim(T_i, Q)$	$O(T_i, Q)$
T1	0,25	2.449
T2	0,25	2.449
T3	0,289	2.236
T4	0,224	2.646
T5	0,671	1,732
T6	0,447	2,236

Z przytoczonych obliczeń wynika, że wektor T5 jest najbardziej podobny do wektora Q danego pytania i równocześnie odległość między wektorem T5 a Q jest najmniejsza.

Przedstawione przykłady dotyczyły baz danych tekstowych zawierających tytuły artykułów.

Wyszukiwanie oparte na obliczaniu podobieństwa czy odległości dwóch wektorów może być również wykorzystywane w bazach danych medycznych. W bazach tych gromadzone są informacje o pacjentach leczonych w szpitalu czy klinice i ważne jest wyszukanie pacjentów mających podobne dane do zadanych parametrów (np.: ciśnienie, wzrost, wiek, puls, poziom cukru...). Informacje dotyczące pacjenta mogą być przedstawione w postaci n-wymiarowego wektora cech, w którym każda cecha oznacza inny wymiar. Poprzez porównanie wektorów pacjentów z podanym w zapytaniu wektorem wejściowym można znaleźć wiersze (wektory) najbardziej zbliżone do wektora wejściowego. Na przykład na podstawie pozostałych informacji prześledzić, w jaki sposób przebiegało leczenie oraz jakie środki czy lekarstwa były podawane. Do takiej bazy danych można także zadawać pytania wyodrębniające grupy najbardziej pasujących do siebie wierszy (pacjentów cierpiących na podobne choroby). Odpowiedzi na tak postawione pytania są wykorzystywane w analizach statystycznych, a przedstawione w przestrzeni lub na płaszczyźnie, uwypuklają podziały na grupy (np.: przedstawienie grup chorób oraz częstości ich występowania wśród pacjentów z uwzględnieniem najbardziej zbliżonych objawów).

Istnieją także sytuacje, gdy są wyodrębnione cechy (atrybuty) obiektów, lecz jest ich bardzo dużo i wyszukiwanie tak zdefiniowanych obiektów jest trudne, żmudne i mało efektywne. Należy wtedy zredukować liczbę cech (przeprowadzić redukcję wymiaru

przestrzeni wejściowej). Problem ten jest sformułowany następująco: wiersze przedstawione są jako wektory w przestrzeni  $n$ -wymiarowej ( $n$  – określa liczbę wyodrębnionych cech, każda cecha stanowi odrębny wymiar przestrzeni). Danych jest więc  $N$  wektorów o  $n$ -składowych. Redukcja cech polega na znalezieniu  $N$  punktów w  $k$ -wymiarowej przestrzeni ( $k \leq n$ ), takich, by odległości między nimi były zachowane najlepiej jak tylko można i odzwierciedlały odległości między wierszami.

Na Uniwersytecie w Maryland [FltKng95] podjęto próby opracowania algorytmu odwzorowującego wiersze w punkty w  $k$ -wymiarowej przestrzeni cech ( $k$  – jest definiowane przez użytkownika), tak by odległości między wierszami były zachowane. W algorytmie tym zakłada się, że specjaliści z danej dziedziny dostarczają tylko funkcję odległości lub podobieństwa  $d(\_,\_)$ . Tak przygotowany zbiór wierszy z podaną funkcją odległości umożliwia znalezienie wierszy podobnych do wiersza podanego w zapytaniu, znalezienie par wierszy, które są najbardziej podobne do siebie oraz wizualizację rozkładu wierszy w odpowiednio wybranej przestrzeni. Nazwano go algorytmem FastMap [FltKng95].

### **1.2.2.     *Wyszukiwanie w bazie według kryteriów opartych na prawdopodobieństwie***

W pytaniach opartych na prawdopodobieństwie funkcją charakterystyczną jest funkcja, określająca w jakim stopniu wiersz spełnia kryteria postawione w pytaniu [DcmQr00]. W odpowiedzi na takie pytania uzyskuje się zbiór wierszy, które spełniają kryteria pytania z prawdopodobieństwem większym od progu określonego przez użytkownika.

Sposób wyszukiwania wierszy opisany w tym podrozdziale znajduje zastosowanie w przypadkach, kiedy znane są *a priori* prawdopodobieństwa występowania cech w wierszach spełniających kryterium pytania, natomiast samo wskazanie tych wierszy jest z różnych powodów niemożliwe. Pokazano tutaj matematyczne wyprowadzenie funkcji charakterystycznej pozwalającej na uzyskanie odpowiedzi na pytania typu opisanego w dwóch poniższych przykładach:

#### ***Przykład 1.4***

Do placówki firmy ubezpieczeniowej zgłaszają się nowe osoby chcące podpisać umowę ubezpieczenia samochodu. Po wprowadzeniu danych o tych osobach do systemu zadawane jest pytanie:

*„Wyszukaj nowe osoby uporządkowane malejąco wg prawdopodobieństwa wypłacenia odszkodowania (z tytułu uczestnictwa w wypadku drogowym).”*

Prawdopodobieństwo to jest tutaj funkcją charakterystyczną, od wartości której firmy ubezpieczeniowe uzależniają wysokość opłaty przy zawieraniu kontraktu. W określaniu tej funkcji uwzględniany jest między innymi wiek ubezpieczanego (ludzie młodzi powodują zazwyczaj więcej wypadków).

**Przykład 1.5**

W pewnej dzielnicy mieszkaniowej dokonano przestępstwa. Policja przyjmując za podejrzanych mieszkańców tejże dzielnicy oraz własne statystyki występowania cech osób, które popełniły już takie przestępstwo (na przykład, że 40% to recydywiści) jest w stanie uzyskać odpowiedź na następujące pytanie:

*Które osoby i w jakim stopniu są najbardziej podejrzane o dokonanie tego przestępstwa?*

W obu przykładach niemożliwe jest, z oczywistych względów, by system zwrócił wiersze spełniające kryteria – jest jednak w stanie podać prawdopodobieństwa z jakim dane wiersze je spełniają.

Punktem wyjścia określenia funkcji przynależności  $f_p$  jest definicja prawdopodobieństwa warunkowego. Na potrzeby pokazania przekształceń doprowadzających do określenia  $f_p$  przyjęto następujące założenia:

- $f_{p2}$  – wyjściowa funkcja charakterystyczna,
- $f_{p1}$  – pośrednia funkcja charakterystyczna,
- $f_p$  – końcowa funkcja charakterystyczna,
- $\Omega$  – zbiór wszystkich przeszukiwanych wierszy bazy danych,
- $W$  – zdarzenie wybrania wiersza,
- $S$  – zdarzenie wybrania wiersza spełniającego kryteria pytania,
- $c_i$  –  $i$ -ta cecha wiersza  $w$ ,
- $c$  – liczba cech, którą posiada każdy wiersz (lub inaczej: liczba kolumn).

Korzystając z klasycznej definicji prawdopodobieństwa, prawdopodobieństwo, że wiersz  $w$  zostanie wybrany jest następujące:

$$P(W) = \frac{1}{\text{card}(\Omega)}.$$

Następnie, w oparciu o definicję prawdopodobieństwa warunkowego [LtnŻkw84][Plc82], określamy wyjściową funkcję charakterystyczną  $f_{p2}$  jako prawdopodobieństwo, że losowo wybrany wiersz spełnia kryteria pytania:

$$f_{p2} = P(S | W) = \frac{P(W | S)P(S)}{P(W)},$$

Wiersz  $w$  składa się z  $c$  cech więc:

$P(W)$  - prawdopodobieństwo, wybrania wiersza spośród wszystkich wierszy znajdujących się w bazie danych,  $P(W) > 0$  wynosi:

$$P(W) = \prod_{i=1}^c P(c_i),$$

gdzie  $P(c_i)$  – prawdopodobieństwo wybrania wierszy posiadających cechę  $c_i$  spośród wszystkich wierszy (znając liczbę wierszy w bazie danych oraz liczbę wierszy, w których dana cecha występuje), gdzie  $i$  zmienia się od 1 do  $c$ .

Zakładając niezależność cech, prawdopodobieństwo wybrania wiersza spełniającego kryteria pytania wynosi:

$$P(W | S) = \prod_{i=1}^c P(c_i | S).$$

Ponieważ nie są znane wiersze spełniające kryteria pytania (przykłady 1.4 i 1.5), to niemożliwe jest także określenie wartości funkcji  $f_{p2}$ . W tym momencie wprowadza się znane *a priori* prawdopodobieństwa spełnienia kryteriów zapytania przez wiersze posiadające pewną cechę ( $P^*(c_i|S)$ ) i otrzymuje pośrednią funkcję charakterystyczną  $f_{p1}$ , wyrażoną wzorem:

$$f_{p1} \approx \frac{P^*(W | S)P(S)}{P(W)}.$$

Prawdopodobieństwo  $P(S)$  (dla konkretnego pytania) jest stałym czynnikiem. Nie ma wpływu na uporządkowanie wierszy względem stopnia spełnienia kryteriów podanych w pytaniu. Z tego powodu zrezygnowano z obliczania jego wartości we wzorze funkcji  $f_{p1}$ , otrzymując  $f_p$ :

$$f_p = \frac{P^*(W | S)}{P(W)},$$

Funkcja  $f_p$  jest ostateczną funkcją charakterystyczną pozwalającą na uzyskanie odpowiedzi na pytania klasy opisanej w przykładach 1.4 i 1.5.

Na zakończenie podrozdziału zostanie przedstawiony jeszcze jeden przykład zapytania, w którym pomocne jest użycie funkcji przynależności opartej na prawdopodobieństwie. Tym razem jednak przykład ten będzie zawierał konkretne obliczenia oparte na przytoczonych w tym rozdziale wzorach.

### **Przykład 1.6**

W pewnym mieście istnieje biuro matrymonialne, informacje o zgłaszających się samotnych osobach przechowywane są w bazie danych. Jest w niej 200 osób, w tym 120 mężczyzn. Przykładowe dane po nałożeniu warunku na płeć (płeć = M) ilustruje tabela 1.6.

Tabela 1.6

Tabela *Kandydaci*

Nr	Imię	Nazwisko	Płeć	Stan finansowy	Wzrost	Wiek	Pije	Stan cywilny	Dzieci
1	Jan	Nowak	M	pracuje	182	30	nie	Kawaler	Nie
2	Karol	Wróbel	M	pracuje	173	29	nie	Wolny	Tak
3	Wojciech	Zyga	M	pracuje	190	32	tak	Kawaler	Nie

4	Piotr	Lalik	M	nie pracuje	184	23	tak	Kawaler	Nie
5	Karol	Piątek	M	nie pracuje	170	43	nie	Wolny	Nie
6	Henryk	Ryszard	M	nie pracuje	176	21	tak	Wolny	Nie

Do biura zgłasza się młoda kobieta, szukając partnera i chce, by wyszukano jej dwóch najbardziej odpowiadających jej kandydatów. Znając swe preferencje i gust twierdzi, że wśród mężczyzn, którzy zawsze wzbudzali jej zainteresowanie jest:

- 90% posiadających pracę i niezależnych finansowo,
- 80% z nich ma powyżej 180 cm,
- 90% z nich jest w wieku między 25-35,
- 10% z nich pije alkohol,
- 70% jest kawalerami,
- 20% z nich ma dzieci.

Na podstawie podanych informacji oraz przy założeniu, że w bazie danych jest 120 mężczyzn, wśród których 80 pracuje, 40 ma powyżej 180 cm wzrostu, 70 jest w wieku 25-35, 40 z nich pije, 40 – kawalerów, 60 ma własne dzieci, można wyznaczyć prawdopodobieństwa  $P(c_i|S)$  oraz  $P(c_i)$  przedstawione w tabeli 1.7.

Tabela 1.7

Cechy oraz ich prawdopodobieństwa  $P(c_i|S)$  i  $P(c_i)$ 

Numer	Cecha	$P^*(c_i S)$	$P(c_i)$
$c_1$	Pracuje	0,9	80/120
$c_2$	Powyżej 180 cm	0,8	40/120
$c_3$	wiek 25-35	0,9	70/120
$c_4$	Pije	0,1	40/120
$c_5$	Kawaler	0,7	40/120
$c_6$	Dzieci	0,2	60/120

Dla każdego wiersza  $w_1, \dots, w_n$  należy obliczyć wartość funkcji  $f_p(w_i)$ .

*Założenie:*

Jeżeli w wierszu dana cecha nie występuje, nie jest ona brana pod uwagę, więc nie wpływa w żaden sposób na wynik obliczeń.

Przykładowo dla wiersza  $w_1$ :

$$P^*(w_1 | S) = P(c_1 | S) * P(c_2 | S) * P(c_3 | S) * P(c_5 | S) = \\ = 0.9 * 0.8 * 0.9 * 0.7 = 0.4536$$

natomiast prawdopodobieństwo wybrania wiersza  $w_1$  wynosi:

$$P(w_1) = P(c_1) * P(c_2) * P(c_3) * P(c_5) = \\ = 80/120 * 40/120 * 70/120 * 40/120 = 14/324$$

zatem wartość funkcji  $f_p(w_1)$  obliczona ze wzoru wynosi:

$$f_p(w_1) = \frac{P^*(w_1 | S)}{P(w_1)} = \frac{0.4536}{14/324} \approx 10.4976.$$

Odpowiednio dla wierszy w tabeli *kandydaci* z tabeli 2.6 wartości funkcji  $f(w_i)$  wynoszą odpowiednio:

$$f_p(w_1) = \frac{P^*(w_1 | S)}{P(w_1)} = \frac{0.4536}{14/324} \approx 10.4976$$

$$f_p(w_2) = \frac{P^*(w_2 | S)}{P(w_2)} = \frac{0.162}{14/72} \approx 0.83$$

$$f_p(w_3) = \frac{P^*(w_3 | S)}{P(w_3)} = \frac{0.04536 * 486}{7} = 3.14928$$

$$f_p(w_4) = \frac{P^*(w_4 | S)}{P(w_4)} = \frac{0.056 * 27}{1} = 1.512$$

$$f_p(w_5) = \frac{P^*(w_5 | S)}{P(w_5)} = \frac{1}{1} = 1$$

$$f_p(w_6) = \frac{P^*(w_6 | S)}{P(w_6)} = \frac{0.1 * 3}{1} = 0.3$$

Wykonane obliczenia potwierdzają, że występowanie cech niepożądanych (o małym prawdopodobieństwie wystąpienia wśród wierszy spełniających kryteria – np. cecha *pije*) wpływa w znacznym stopniu na zmniejszanie wartości  $f_p$ . Wiersz nie posiadający żadnych cech sprecyzowanych w pytaniu ma wartość funkcji większą ( $f_p(k_5) = 1$ ) niż wiersz z cechami niepożądanymi ( $k_6$  lub  $k_2$ ).

Na podstawie wartości  $f_p$  można wyznaczyć zadaną liczbę wierszy najlepiej spełniających oczekiwania pytającego.

### 1.3. Zastosowanie teorii zbiorów rozmytych do wyszukiwania w bazach danych

Twórca teorii zbiorów rozmytych Lotfi A. Zadeh uznał, że inicjującą pracą w dziedzinie wykorzystania zbiorów rozmytych w bazach danych było opracowanie jego studenta V. Tahani o wyszukiwaniu informacji rozmytej [Thn76], które ukazało się w 1976r.

Pierwsze obszerne opracowania dotyczące wykorzystania teorii zbiorów rozmytych w bazach danych pojawiły się w późnych latach siedemdziesiątych i osiemdziesiątych. W dziedzinie tej można wyodrębnić dwa zasadnicze nurty badawcze [Bdr99]:

- zadawanie niedokładnych (rozmytych) zapytań do bazy danych (*ang. fuzzy queries*),
- zapamiętywanie rozmytych informacji w bazie danych (*ang. fuzzy databases*).

Także wśród ośrodków zajmujących się rozwojem teorii zbiorów rozmytych w bazach danych można zauważyć podział ze względu na te dwa kierunki badań. Część z nich zajmuje



się rozmytymi bazami danych, a część konstruowaniem rozmytych zapytań do baz danych zawierających ostre (pewne) informacje. Wiele z tych ośrodków znajduje się w Stanach Zjednoczonych, ale również w Europie dziedzina ta jest bardzo mocno rozwijana.

### **1.3.1.      *Zadawanie niedokładnych (rozmytych) zapytań do bazy danych***

Badania w zakresie zadawania niedokładnych zapytań do baz danych rozwinęły się w latach 70'tych i 80'tych. W 1978 roku L. A. Zadeh opublikował pracę prezentującą pierwszy translator języka naturalnego wykorzystujący teorię zbiorów rozmytych – *Possibilistic Relational Universal Fuzzy (PRUF)* [Zdh78].

Obecnie dziedzina ta jest szeroko rozwijana i powstają systemy wykorzystujące teorię zbiorów rozmytych w bazach danych. W podrozdziale tym przedstawiono kilka z nich.

Yoshikane Takahashi na podstawie prac L. A. Zadeh'a, zaproponował rozmyty język zapytań dla zwykłych użytkowników (nie będących ekspertami) [Tkh91], rozszerzając język PRUF o złożone wyrażenia typu [Tsk]:

*X jest **bardzo** niski.*

*X jest mały i Y jest duży.*

***Wielu** mężczyzn jest wyższych od **większości** kobiet.*

Standardowe zapytania mogły być używane w połączeniu z rozmytymi wyrażeniami, gdyż język ten nie wymagał zmian w relacyjnych bazach danych.

#### **Fuzzy Query**

Fuzzy Query jest tworzona od połowy 1997 r. aplikacją przez Sonalysts Inc. Aplikacja ta pozwala użytkownikowi zadawać do bazy danych pytania wykorzystujące logikę rozmytą [Snst00]. Użytkownik ma możliwość: zdefiniowania wybranej funkcji przynależności poprzez wybór krzywej, określenie dziedziny tej funkcji itd.. Może również zdefiniować kryteria, zbudować zapytanie czy też wykonać je. W odpowiedzi otrzymuje się zbiór wierszy wynikowych posortowanych względem stopnia spełnienia kryteriów pytania. System Fuzzy Query znajduje zastosowanie w wielu dziedzinach życia, w których przetwarza się wiele danych i poddaje się je analizie. W Sonalysts Inc. opracowano również system Fuzzy Query dla MS Excel.

#### **Fquery**

W Polsce zastosowaniem teorii zbiorów rozmytych w bazach danych oraz semantyką niedokładnych pytań, wyrażaną w oparciu o teorię zbiorów rozmytych (niedokładność reprezentowana za pomocą zmiennych lingwistycznych), zajmuje się Instytut Badań Systemowych PAN w Warszawie (J. Kacprzyk, S. Zadrozny, A. Ziółkowski) [KcpZdr97], [KcpZlk86]. Pracownicy Instytutu opracowali narzędzie, oparte na logice rozmytej, efektywnego wyszukiwania nieprecyzyjnej informacji wyrażonej w postaci słownej

(mniejszy, większy) dla MS Access (*ang. FQUERY for Access*) [KcpZdr98]. FQUERY [KcpZdr96] umożliwia wykorzystanie elementów teorii zbiorów rozmytych takich jak: wartości lingwistyczne (np. *wysoki*), relacje lingwistyczne (np. *znacznie więcej niż*), modyfikatory lingwistyczne (np. *bardzo*) i lingwistyczne kwantyfikatory (np. *większość*) w pytaniach rozmytych zadawanych do bazy danych zawierającej dokładne informacje. FQUERY posiada interfejs graficzny, za pomocą którego można definiować rozmyte elementy (kwantyfikatory czy relacje rozmyte). Następnie, wykorzystując możliwości SZBD MS Access, użytkownik może konstruować pytanie do bazy danych zawierające zdefiniowane wcześniej elementy rozmyte.

System FQUERY może również zostać uruchomiony w sieci Internet, wykorzystując standardowe przeglądarki (NETSCAPE czy MS Internet Explorer), umożliwia zadawanie pytań przez Inetrnet. Nowsze wersje systemu FQUERY mają zaimplementowane pewne elementy SQLf opracowane przez P. Bosc'a i O. Pivert'a [BscPvr95].

Obecnie w tym ośrodku trwają prace nad wykorzystaniem teorii zbiorów rozmytych w bazach danych wykorzystywanych przez aplikacje medyczne, pozwalające na rozpoznawanie chorób, na wyodrębnianie pacjentów o objawach najbardziej odpowiadających zdefiniowanemu już wcześniej objawom choroby [KcpSzt01a],[KcpSzt01b].

We Francji od wielu lat P. Bosc i O. Pivert zajmują się rozszerzeniem języka SQL o elementy rozmyte, pozwalające na zadawanie nieprecyzyjnych zapytań do bazy danych. Są oni jednymi z twórców rozszerzonego języka SQL o nazwie **SQLf** [BscPvr95], [BscPvr94], [BscLtrPvr95], [BscDbsPrd94], [BscDbsPvrPrd]. Wraz z H. Prade określili podział informacji rozmytej na: niepewną, nieprecyzyjną, nieokreśloną i niezgodną.

W dziedzinie przetwarzania rozmytych zapytań zadawanych do bazy danych, w której przechowywane są ostre, dokładne informacje, znaczny wkład wniosły prace: G. Bordogna, G. Pasi - Hiszpania [BrdPs00a][BrdPs00b][BrdPs94], H. Larsen'a [Lrs99] i R. Yager'a [YgrLrs93][RsmYgr97].

### **1.3.2. Zapamiętywanie rozmytych informacji w bazach danych**

Zapamiętywanie rozmytych informacji w bazach danych, tworzenie rozmytych relacyjnych modeli danych, stanowi odrębny kierunek badań, choć niektóre z ośrodków naukowych zajmują się zarówno tworzeniem rozmytych baz danych jak i rozszerzaniem języków zapytań, umożliwiających zadawanie nieprecyzyjnych zapytań.

Tworzeniem rozmytych modeli danych [DbsPrd96], [YzcGrg99], [BrdPs00a], [BrdPs00b] zajmują się między innymi B. P. Buckles i F. E. Petry [BklPtr95], [IntMkd01],

M. Umamo, M. Zemankova i A. Kandel, H. Prade [DbsPrd] i C. Testemale [Prs96], D. Dubois [RdnBc91] oraz S. Sheno i A. Melton [Prs96], [RdnBc91].

Oprócz tradycyjnych wartości rozmytych w literaturze rozpatruje się możliwość przechowywania w bazach danych wartości nieostrych innej postaci, na przykład:

- wartości określonych przez rozkład możliwości [RdnBc91], [DbsPrd], [DbsPrd88], [Prs96],
- wartości scharakteryzowanych relacją podobieństwa [BklPtr95], [IntMkd01].

Podobnie na Uniwersytecie w Granadzie w Hiszpanii zespół naukowców (J. C. Cubero, M. A. Vila, K. Pons, J. M. Medina) zajmuje się również badaniami nad rozmytymi relacyjnymi modelami danych [MdnVICbrPns94]. W wyniku przeprowadzanych badań powstał rozmyty relacyjny model GEFRED [MdnPnsVI94a]. Model ten pozwala na przechowywanie w tabelach zarówno rozmytych jak i dokładnych informacji.

W oparciu o model GEFRED zaproponowano również reprezentację informacji rozmytej w bazie danych, w tym celu wyróżniono kilka typów danych. Dane opisane są trapezowymi funkcjami przynależności, których parametry zapamiętane są w specjalnej tabeli [MdnPnsVI94b]. Konsekwencją dalszych prac było zaproponowanie rozszerzenia języka SQL o rozmyte elementy. Jako system zarządzania bazą danych wybrano ORACLE<sup>TM</sup>. W tych rozszerzeniach wzorowano się na pracach P. Bosc'a i M. Galibourg'a. W 1998 roku powstał FSQ Server umożliwiający zadawanie rozmytych pytań do bazy danych opartej na modelu GEFRED [GlnMdnPnsCbr98].

Na uniwersytecie tym opublikowano wyniki wielu prac związanych z rozmytą algebrą relacji (rozmytą selekcją, projekcją i łączeniem), analizą rozmytych zależności funkcyjnych i operatorów relacyjnych [CbrMdnVI93][CbrVI94][CbrPnsVI94]. Kolejne prace dotyczące tej tematyki to prace związane z rozmytymi zależnościami funkcyjnymi [CbrMdnPnsVI99a], [CbrMdnPnsVI99b][CbrMdnPnsVI99c], rozmytą projekcją, łączeniem [CbrMdnPnsVI95], rozmytą dekompozycją bezstratną [CbrMdnPnsVI97a][CbrMdnPnsVI98] czy aksjomatami Armstronga, w wyniku których powstawały kolejne publikacje [CbrMdnPnsVI97b].

W innych ośrodkach prowadzone są także badania nad:

- wykorzystaniem możliwości sieci neuronowych w wykonywaniu rozmytych zapytań (w sztucznej inteligencji): we Francji D. Dubois i H. Prade
- wykorzystaniem teorii zbiorów rozmytych w obiektowych bazach danych: prace G. Bordogna, G. Pasi i R. De Caluwe [BrdPs00],
- modelowaniem baz danych na poziomie conceptualnym, logicznym i fizycznym, umożliwiających przechowywanie złożonych i niepewnych informacji [YzcGrg99],

- wykorzystaniem teorii zbiorów rozmytych w bazach danych systemów geograficznych, prace M.Cobb, A. Yazici, K. Akkaya i V. Robinson'a [BrdPs00].

Pomimo licznych prac dotyczących wykorzystania zbiorów rozmytych w bazach danych w dalszym ciągu do rozwiązania pozostają niektóre problemy implementacji mechanizmów rozmytych w języku SQL (np. interpretacja warunków rozmytych w pytaniach zagnieżdżonych, rozmyte grupowanie danych, niektóre kwantyfikatory rozmyte), a także problem implementacji mechanizmów rozmytych w innych niż wymienione wyżej relacyjnych systemach zarządzania bazami danych. Próbę rozwiązania tych problemów podjęto w niniejszej pracy.

#### 1.4. Cele i teza pracy

Ogólnym celem rozprawy było **zastosowanie teorii zbiorów rozmytych do rozszerzenia możliwości wyszukiwania informacji w bazach danych**. Zrealizowanie tego celu wymagało szerokiej analizy teoretycznej zagadnień zbiorów rozmytych w aspekcie baz danych, a następnie opracowanie metod i algorytmów możliwych do zastosowania w systemach zarządzania bazami danych. Dało to podstawę do sformułowania następującej tezy:

***Możliwa jest rozbudowa klasycznego systemu zarządzania relacyjną bazą danych do systemu umożliwiającego gromadzenie danych rozmytych i rozmyte wyszukiwanie informacji z zadaniem stopniem zgodności.***

**Wyróżniono też tezy cząstkowe:**

Opracowane w rozprawie metody i algorytmy :

- interpretacji rozmytych zapytań zagnieżdżonych, z uwzględnieniem warunków nakładanych na stopień zgodności z kryteriami pytania,
- rozmytego grupowania danych,
- rozmytych kwantyfikatorów,

rozszerzają znane rozwiązania w zakresie mechanizmów rozmytych w języku SQL i dają możliwość efektywnej implementacji tych mechanizmów.

Tak sformułowane tezy zostaną dowiedzione poprzez zaproponowanie metod i algorytmów wykorzystujących elementy teorii zbiorów rozmytych. Użyteczność tych algorytmów wykazano implementując je w SZBD PostgreSQL.

## 2. Bazy danych i zbiory rozmyte – podstawowe pojęcia wykorzystywane w pracy

Praca łączy dwie dziedziny poprzez wykorzystanie teorii zbiorów rozmytych w bazach danych. Rozdział ten podzielony został na dwie niezależne części. W pierwszej omówiono podstawowe pojęcia dotyczące baz danych, w drugiej podstawowe elementy teorii zbiorów rozmytych.

### 2.1. Relacyjne bazy danych

W rozdziale tym przedstawiono wybrane pojęcia z zakresu baz danych, w szczególności cechy relacyjnego modelu danych, których znajomość jest konieczna w pracy. W dalszej części przybliżono język zapytań SQL (*ang. Structured Query Language*) ze szczególnym uwzględnieniem instrukcji *SELECT*.

#### 2.1.1. Pojęcia podstawowe

**Baza danych** to zbiór wzajemnie powiązanych danych pamiętanych w sytemie komputerowym. **Dane** rozumiane są jako znane fakty, które mogą być rejestrowane i mają rozumiejące się samo przez się znaczenie [ElmNvt00].

**System zarządzania bazą danych (SZBD)** (*ang. database management system - DBMS*) to zbiór programów, które umożliwiają użytkownikom tworzenie i obsługę bazy danych (system programowy ogólnego przeznaczenia, który ułatwia użytkownikom definiowanie, konstruowanie i manipulowanie bazami danych w różnych zastosowaniach [ElmNvt00].

Jest wiele definicji pojęć **baza danych** i **system zarządzania bazą danych**, kolejne z nich można znaleźć między innymi w następujących pozycjach literaturowych: [Dt95], [UllmWdm00], [Kzl03], [DlbAdb89], [BnnDv98], [Ullm88].

**Język bazy danych** (*ang. database language*) – język bądź jego składowe, przeznaczone do definiowania struktur danych (język definiowania danych – *ang. data definition language*) i do wyszukiwania oraz aktualizowania danych (język operowania danymi – *ang. data manipulation language*).

**Relacyjny system bazy danych** (*ang. relational database system*) – system (obejmujący bazę danych oraz SZBD) oparty na relacyjnym modelu danych. W systemach tych najczęściej wykorzystywanym językiem baz danych jest język SQL.

**Relacyjny model danych**, to model opierający się na matematycznym pojęciu relacji. Twórcą relacyjnego modelu danych jest E. F. Codd, którego przełomowy artykuł<sup>1</sup> w tej dziedzinie ukazał się w 1970r [UllmWdm00].

Nazwa relacji oraz jej zbiór atrybutów stanowią **schemat relacji**.

*Nazwa ( $atr_1, atr_2, \dots, atr_n$ )*

Baza danych oparta na modelu relacyjnym składa się z jednego lub kilku schematów relacji. Zbiór schematów relacji nazywany jest **schematem relacyjnym bazy danych** [UllmWdm00], a baza danych jest zbiorem relacji [Bnch98].

Relacje są **zbiorami krotek**. W krotce każdy atrybut ma swój odpowiednik w postaci **składowej krotki**. Kolejność występowania krotek jest nieistotna.

Z każdym atrybutem relacji jest powiązana jego nazwa i **dziedzina**, określająca zbiór wartości, jakie może przyjmować atrybut. Każda składowa każdej krotki w relacji ma wartość, która należy do dziedziny odpowiedniego atrybutu.

Dla modelu relacyjnego opracowano kilka języków zapytań wysokiego poziomu. Standardem wśród nich jest język **SQL**.

W dziedzinie baz danych **relacja** utożsamiana jest z dwuwymiarową **tabelą** złożoną z **kolumn** i **wierszy**, do której wpisuje się kolejne wiersze danych [UllmWdm00].

Dalej pojęcie relacji utożsamiane będzie z tabelą.

### 2.1.2. *Język SQL*

Język SQL jest najbardziej popularnym językiem formułowania zapytań w relacyjnych systemach baz danych, przy czym pojęcie „*zapytanie*” definiuje tu zadania wyszukiwania, wprowadzania, modyfikacji lub usuwania danych, a także zarządzania danymi.

SQL jest językiem czwartej generacji, opracowywanym przez grupę badawczą IBM w San Jose przez wiele lat [Bnch98]. Pierwszy standard języka SQL zgodny z normą międzynarodowej organizacji ISO powstał w 1987 roku, drugi w 1989 roku a trzeci w 1992 roku (SQL2). Trwają prace nad kolejnymi standardami uwzględniającymi cechy obiektowe (SQL3) [Bnch98].

W języku SQL można wydzielić kilka instrukcji odpowiadających za wykonywanie pewnych operacji w bazie danych [WellSoft95]:

- wyszukiwanie informacji w bazie danych (*SELECT*),
- operowanie na danych w bazie danych – wstawianie (*INSERT*), modyfikacja (*UPDATE*) i usuwanie (*DELETE*) danych z bazy danych,

---

<sup>1</sup> Codd E. F.: A Relational Model of Data for Large Shared Data Banks, CACM 13, No. 6, 1970

- definiowanie i modyfikacja struktury bazy danych – dodawanie (usuwanie) do (z) bazy danych nowych tabel (*CREATE TABLE*, *DROP TABLE*), modyfikowanie struktury tabel (*ALTER TABLE*), tworzenie indeksów (*CREATE INDEX*), synonimów (*CREATE SYNONYM*), perspektyw (*CREATE VIEW*).
- sterowanie dostępem do danych – tworzenie użytkowników (*CREATE USER*), nadawanie uprawnień użytkownikom do wykonywania pewnych operacji w bazie danych oraz do istniejących obiektów w bazie danych (*GRANT*) oraz instrukcje związane z mechanizmami transakcji, blokad itp.

Uwzględniają fakt, że tworzone w niniejszej pracy operatory i funkcje wykorzystywane są głównie w konstrukcjach dotyczących wyszukiwania informacji w bazie danych, opartych na instrukcji *SELECT* – jej format w nieco uproszczonej formie oraz sposób wykonywania zostanie przedstawiony szerzej.

**Instrukcja *SELECT* (składnia uproszczona):**

```
SELECT [DISTINCT] ciąg nazw kolumn lub 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 wyszukiwania. 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*.

W języku SQL możliwe jest 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 z kluczem obcym. W ogólnym przypadku warunek złączenia dwóch (lub więcej) tabel może być dowolny, a połączenie między tabelami może wystąpić między dwoma dowolnymi kolumnami o tej samej dziedzinie.

Wyszukiwane dane mogą zostać przetworzone przy użyciu funkcji agregujących (*COUNT*, *AVG*, *SUM*, *MAX*, *MIN*).

W języku SQL możliwe jest dokonanie podziału wynikowych wierszy zapytania na grupy i wykonanie funkcji agregujących na wartościach należących do poszczególnych grup. Do wyodrębnienia grup służy klauzula *GROUP BY*, po której wypisuje się nazwy kolumn, których jednakowe wartości wyznaczają 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 wykorzystuje się klauzulę *HAVING*.

W języku SQL można również uzyskać wyniki w postaci sumy, przecięcia i różnicy teoriomnogościowej, łącząc relacje lub wyniki podzapytań odpowiednio spójnikami: *UNION*, *INTERSECT* lub *EXCEPT (MINUS)* [UllmWdm00].

W pracy pojęcie klauzuli utożsamiane będzie z pojęciem frazy i stosowane zamiennie.

Język SQL pozwala definiować **perspektywy**. Perspektywy, to „okna” na tabele bazy danych, umożliwiające użytkownikowi oglądanie wybranych części bazy danych. Są one również nazywane tabelami wirtualnymi, gdyż nie zawierają danych. Zawartości perspektyw same nie są przechowywane, a jedynie ich definicje [WellSoft95].

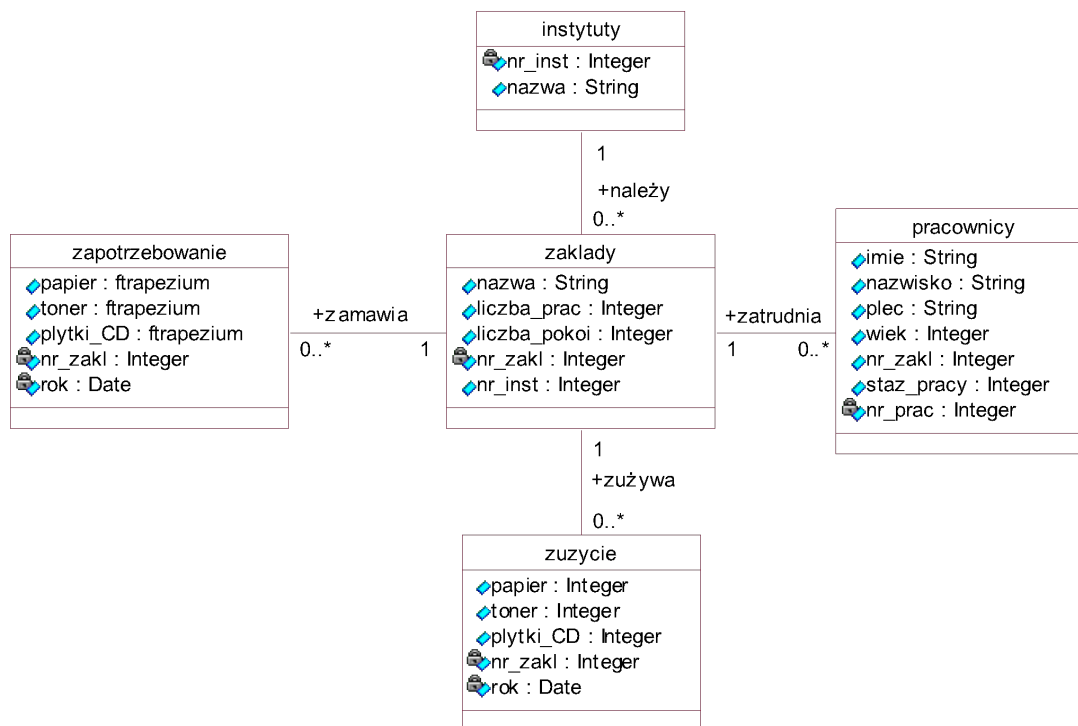
Prezentowane dalej przykłady (2.1 – 2.4) pytań w języku SQL korzystają z przykładowej bazy danych *ZAKŁADY*. Przyjmijemy, że w skład bazy danych *ZAKŁADY* wchodzi między innymi następujące tabele:

```
Instytuty (nr_inst, nazwa),  
Zaklady (nr_zakl, nazwa, liczba_prac, liczba_pokoi, nr_inst),  
Pracownicy (nr_prac, imie, nazwisko, wiek, plec, staz_pracy, nr_zakl),  
Zapotrzebowanie (nr_zakl, rok, papier, toner, plytki_CD),  
Zuzycie (nr_zakl, rok, papier, toner, plytki_CD),
```

podkreślenie wyróżnia kolumny wchodzące w skład klucza głównego tabeli.

Schemat bazy danych *ZAKŁADY* przedstawiony jest na diagramie (rys.2.1).



Rys. 2.1 Schemat bazy danych *ZAKŁADY***Przykład 2.1**

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

„Wyszukać nazwiska pracowników o stażu pracy większym niż 10 lat oraz podać nazwy zakładów, w których są zatrudnieni.”

Pytanie takie może zostać zapisane w języku SQL w następujący sposób:

```
SELECT p.nazwisko, z.nazwa
FROM pracownicy p JOIN zakłady z ON p.nr_zakl = z.nr_zakl
WHERE p.staz_pracy > 10;
```

**Przykład 2.2**

Przykładowe, nieco bardziej skomplikowane pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

„Wyszukać nazwy jednostek, w których jest zatrudnionych więcej niż 3 pracowników, którzy przekroczyli pięćdziesiąty rok życia oraz podać liczbę tych pracowników.”

Pytanie takie może zostać zapisane w języku SQL w następujący sposób:

```
SELECT z.nazwa, count(p.nr_prac)
FROM pracownicy p JOIN zakłady z ON p.nr_zakl = z.nr_zakl
WHERE p.wiek > 50
GROUP BY z.nr_zakl, z.nazwa
HAVING count(p.nr_prac) > 3;
```

### Zapytania zagnieżdżone

Wewnątrz klauzul *WHERE*, *HAVING* a także *FROM* mogą wystąpić podzapytania, mające taką samą postać jak zapytania, tylko ujęte w nawiasy.

Podzapytanie może wystąpić jako prawy argument operatorów relacyjnych:  $=$ ,  $<$ ,  $<=$ ,  $>=$ ,  $>$ ,  $<>$ , gdy w wyniku jego wykonania zwracany jest jeden wiersz. Gdy podzapytanie zwraca zbiór wartości, musi być poprzedzone operatorem *[NOT] IN* lub operatorami *ALL* i *ANY*.

Pytanie zagnieżdżone może także być związane z pytaniem nadrzędnym predykatem *[NOT] EXISTS*. Predykat *EXISTS* przyjmuje wartość prawdy (true), jeśli zbiór wierszy wybranych w zapytaniu zagnieżdżonym nie jest pusty; w przeciwnym przypadku przyjmuje wartość fałszu (false).

Zapytania zagnieżdżone dzielone są na: **zwykłe** i **skorelowane**.

W **zwykłych** pytaniach zagnieżdżonych podzapytanie wewnętrzne wykonywane jest tylko raz, zwracając w odpowiedzi wiersz lub zbiór wierszy. Następnie dla każdego analizowanego w zapytaniu zewnętrznym wiersza następuje sprawdzenie warunku łączenia z wartością lub zbiorem wartości zwracanych przez pytanie wewnętrzne. Gdy warunek jest spełniony, wiersz pojawia się w zbiorze wynikowym.

W pytaniach skorelowanych w pytaniu wewnętrznym występuje odwołanie do tablicy (i jej kolumny) zdefiniowanej w pytaniu zewnętrznym.

#### Przykład 2.3

Przykładowe pytanie zagnieżdżone, nieskorelowane kierowane do bazy danych **ZAKŁADY** może brzmieć następująco:

*„Wyszukać numery i nazwiska pracowników, którzy są starsi od pracownika o nazwisku ‘Kowalik’.”*

Pytanie takie może zostać zapisane w języku SQL w następujący sposób:

```
SELECT p.nr_prac, p.nazwisko
FROM pracownicy p
WHERE p.wiek > (SELECT wiek
                FROM pracownicy p
                WHERE p.nazwisko = 'Kowalik');
```

W zapytaniach **skorelowanych** podzapytanie wewnętrzne wykonywane jest dla każdego analizowanego wiersza zapytania zewnętrznego. I dla każdego takiego wiersza wyznaczany jest zbiór wierszy wynikowych podzapytania wewnętrznego.

#### Przykład 2.4

Przykładowe pytanie zagnieżdżone skorelowane kierowane do bazy danych **ZAKŁADY** może brzmieć następująco:

*Wyszukać nazwy zakładów, w których są zatrudnione same kobiety.*

Pytanie takie może zostać zapisane w języku SQL w następujący sposób:

```
SELECT z.nazwa
FROM zaklady z
WHERE NOT EXISTS
      (SELECT *
       FROM pracownicy p
       WHERE p.nr_zakl = z.nr_zakl AND p.plec = 'M');
```

### 2.1.3. *Przekształcanie zagnieżdżonych zapytań SQL w niezagnieżdżone*

W większości przypadków pytania zagnieżdżone da się przekształcić w niezagnieżdżone. Zagnieżdżone pytanie jest trudne do zoptymalizowania, optymalizatory i tak usiłują zagnieżdżone zapytanie doprowadzić do postaci, którą można wyrazić w postaci złączeń i wyznaczyć najlepszą kolejność wykonywania [Clk99]. Z tego powodu w literaturze często po wprowadzeniu zagnieżdżonej postaci pytania autorzy prezentują jego niezagnieżdżone formy zapisu [Dt95]. Zazwyczaj złożoność pytań niezagnieżdżonych jest mniejsza. Dlatego w punkcie tym przedyskutujemy najbardziej typowe przypadki przekształcania zapytań zagnieżdżonych w niezagnieżdżone.

Potrzeba przeprowadzenia takiej analizy wynika z faktu, iż niektóre z technik przedstawionych w tym rozdziale mogą być używane w rozmytych zapytaniach SQL [YuMng98]. Informacje zawarte w tym rozdziale będą wykorzystywane w dalszej części pracy w rozdziale 3.8.

Analiza przekształcania zagnieżdżonych zapytań w niezagnieżdżone została przeprowadzona dla pewnych charakterystycznych klas zapytań.

W pierwszej części analizie zostaną poddane pytania nieskorelowane: z warunkiem łączącym dwa podzapytania we frazie *WHERE*, następnie we frazie *HAVING* z uwzględnieniem różnic występujących przy zastosowaniu predykatów *IN*, *ALL*, *ANY*.

W drugiej części analizie zostaną poddane pytania skorelowane: z warunkiem łączącym dwa podzapytania we frazie *WHERE*, następnie we frazie *HAVING* z uwzględnieniem predykatu *EXISTS*.

## 1. Pytania nieskorelowane:

### 1.1. Warunek łączący dwa podzapytania we frazie *WHERE*

Rozpatrzmy pytanie z przykładu 3.3.

„Wyszukać numery i nazwiska pracowników, którzy są starsi od pracownika o nazwisku ‘Kowalik’.”

Przypomnijmy, że zagnieżdżona postać tego pytania zapisana w języku SQL wygląda następująco:

```
SELECT p.nr_prac, p.nazwisko
FROM pracownicy p
```

```
WHERE p.wiek > (SELECT wiek
                FROM pracownicy p
                WHERE p.nazwisko = 'Kowalik');
```

Niektóre systemy jak MS SQL Server, Oracle czy PostgreSQL oferują możliwość zapisywania pytania we frazie *FROM* jako tymczasowej tabeli. Eliminuje się wtedy kolejne kroki (nie jest konieczne tworzenie pytania tworzącego dodatkową tabelę, przechowującą na przykład wartości funkcji agregujących). Powyższe pytanie można więc również zapisać w następujący sposób:

```
SELECT p.nr_prac, p.nazwisko
FROM pracownicy p, (SELECT wiek
                    FROM pracownicy p
                    WHERE p.nazwisko = 'Kowalik') k
WHERE p.wiek > k.wiek;
```

Pytanie to może zostać zapisane w postaci pytania niezagnieżdżonego w następujący sposób:

```
SELECT p1.nr_prac, p1.nazwisko
FROM pracownicy p1, pracownicy p2
WHERE p1.wiek > p2.wiek AND p2.nazwisko = 'Kowalik';
```

Rozważmy teraz pytanie zagnieżdżone z operatorem *ANY*:

„Wyszukać nazwiska pracowników Zakładu Oprogramowanie młodszych od przynajmniej jednego z pracowników Zakładu Teorii Informatyki.”

Zagnieżdżona postać tego pytania zapisana w języku SQL jest następująca:

```
SELECT p.nazwisko
FROM pracownicy p JOIN zaklady z ON p.nr_zakl = z.nr_zakl
WHERE z.nazwa = 'Zakład Oprogramowania' AND p.wiek < ANY
      (SELECT wiek
       FROM pracownicy p JOIN zaklady z
       ON p.nr_zakl = z.nr_zakl
       WHERE z.nazwa = 'Zakład Teorii Informatyki');
```

Pytanie to może zostać zapisane w postaci pytania niezagnieżdżonego w następujący sposób:

```
SELECT distinct p1.nazwisko
FROM pracownicy p1, pracownicy p2, zaklady z1, zaklady z2
WHERE p1.wiek < p2.wiek AND p1.nr_zakl = z1.nr_zakl
      AND z1.nazwa = 'Zakład Oprogramowania'
      AND p2.nr_zakl = z2.nr_zakl
      AND z2.nazwa = 'Zakład Teorii Informatyki';
```

Na koniec tego podpunktu rozważmy pytanie zagnieżdżone z operatorem *ALL*:

„Wyszukać nazwiska pracowników Zakładu Oprogramowanie młodszych od każdego z pracowników Zakładu Teorii Informatyki.”

Zagnieżdżona postać tego pytania zapisana w języku SQL jest następująca:

```
SELECT p.nazwisko
FROM pracownicy p JOIN zaklady z ON p.nr_zakl = z.nr_zakl
WHERE z.nazwa = 'Zakład Oprogramowania' AND p.wiek < All
      (SELECT wiek
       FROM pracownicy p JOIN zaklady z
       ON p.nr_zakl = z.nr_zakl
       WHERE z.nazwa = 'Zakład Teorii Informatyki');
```

Pytanie to może zostać zapisane w postaci pytania niezagnieżdżonego w następujący sposób:

```
SELECT p1.nr_prac, p1.nazwisko
FROM pracownicy p1, pracownicy p2, zaklady z1, zaklady z2
WHERE p1.nr_zakl = z1.nr_zakl
      AND z1.nazwa = 'Zakład Oprogramowania'
      AND p2.nr_zakl = z2.nr_zakl
      AND p2.nazwa = 'Zakład Teorii Informatyki'
GROUP BY p1.nr_prac, p1.nazwisko, p1.wiek
HAVING p1.wiek < MIN (p2.wiek);
```

Jak widać analizowane powyżej pytania zagnieżdżone da się przekształcić w niezagnieżdżone. Ponieważ w odpowiadających sobie pytaniach, na tych samych danych będą wykonywane analogiczne operacje, więc każde z pytań w odpowiedzi powinno zwrócić ten sam zbiór wierszy.

## 1.2. Warunek łączący dwa podzapytania we frazie *HAVING*

Rozważmy następujące pytanie zagnieżdżone:

*„Wyszukać nazwy tych zakładów, w których średnia wieku pracowników jest wyższa od średniej wieku wszystkich pracowników.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL jest następująca:

```
SELECT z.nazwa
FROM zaklady z JOIN pracownicy p ON z.nr_zakl = p.nr_zakl
GROUP BY z.nr_zakl, z.nazwa
HAVING AVG(wiek) > (SELECT AVG(wiek) FROM pracownicy);
```

Pytanie to może zostać zapisane w postaci pytania niezagnieżdżonego w dwóch krokach w następujący sposób (składnia zgodna z MS SQL Server):

1. wyznaczenie średniej wieku wszystkich zatrudnionych pracowników:

```
SELECT AVG(wiek) AS srednia
INTO sr_wiek
FROM pracownicy;
```

2. wypisanie nazw zakładów:

```
SELECT z.nazwa FROM zaklady z
      JOIN pracownicy p ON z.nr_zakl = p.nr_zakl, sr_wiek sr
GROUP BY z.nr_zakl, z.nazwa
HAVING AVG(wiek) > sr.srednia;
```

Jak widać w przypadku, gdy w warunku łączenia pytania zewnętrznego z wewnętrznym po stronie pytania wewnętrznego występuje funkcja agregująca, analizowane pytanie zagnieżdżone da się przekształcić w niezagnieżdżone, ale w przynajmniej dwóch krokach.

Rozważmy przykładowe pytanie wykorzystujące funkcję agregującą *count*

*„Wyszukać nazwę zakładu, w którym pracuje najwięcej kobiet.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL wygląda następująco:

```
SELECT z.nazwa
FROM zaklady z JOIN pracownicy p ON z.nr_zakl = p.nr_zakl
WHERE p.plec = 'K'
GROUP BY z.nr_zakl, z.nazwa
HAVING COUNT(p.nr_prac) >= ALL
        (SELECT COUNT(p.nr_prac)
         FROM zaklady z JOIN pracownicy p
         ON z.nr_zakl = p.nr_zakl
         WHERE p.plec = 'K'
         GROUP BY z.nr_zakl, z.nazwa);
```

Pytanie to może zostać zapisane w postaci pytania niezagnieżdżonego w trzech krokach w następujący sposób:

1. wyznaczenie liczby kobiet w poszczególnych zakładach:

```
SELECT COUNT(p.nr_prac) as l_kob
INTO liczba
FROM pracownicy p
WHERE p.plec = 'K'
GROUP BY nr_zakl;
```

2. wyznaczenie maksymalnej liczby kobiet:

```
SELECT MAX(l_kob) as maxk
INTO max_kob
FROM liczba;
```

3. wypisanie zakładu z największą liczbą pracujących kobiet:

```
SELECT z.nazwa
FROM zaklady z JOIN pracownicy p
ON z.nr_zakl = p.nr_zakl, max_kob mb
WHERE p.plec = 'K'
GROUP BY z.nr_zakl, z.nazwa
HAVING COUNT(p.nr_prac) = mb.maxk;
```

Jeśli jest więcej zakładów, w których pracuje największa liczba kobiet, ich nazwy zostaną wypisane.

Przedstawione powyżej pytania zagnieżdżone i niezagnieżdżone wykorzystujące tabele tymczasowe, wykonują analogiczne operacje na tych samych danych, powinny więc zwrócić w odpowiedzi ten sam zbiór wierszy. Można więc uznać je za równoważne.

## 2. Pytania skorelowane

W pytaniach zagnieżdżonych skorelowanych dla każdego wiersza spełniającego kryteria pytania zewnętrznego sprawdzane są wszystkie wiersze spełniające kryteria pytania wewnętrznego. Jeśli jest  $n_1$  wierszy spełniających kryteria pytania zewnętrznego i  $n_2$  wierszy spełniających kryteria pytania wewnętrznego, to złożoność czasowa wykonania całe pytania będzie rzędu  $(n_1 * n_2)$  [Bnt92], [Wrt89], [YuMng98]. Podczas, gdy złożoność czasowa tego samego pytania zapisanego w postaci niezagnieżdżonej jest rzędu  $(n_1 \log_{n_1} + n_2 \log_{n_2} + n_1 + n_2)$  [Bnt92], [Wrt89] [YuMng98]. Uzasadnia to celowość poszukiwania niezagnieżdżonej formy pytań skorelowanych. Analizie w tej klasie zapytań poddane zostaną podobne grupy zapytań jak w przypadku klasy pytań nieskorelowanych:

### 2.1. Warunek łączący dwa podzapytania we frazie *WHERE*

Rozpatrzmy pytanie z przykładu 2.4:

*„Wyszukać nazwy zakładów, w których są zatrudnione same kobiety.”*

Pytanie takie może zostać zapisane w języku SQL w postaci zagnieżdżonej następujący sposób:

```
SELECT z.nazwa
FROM zaklady z
WHERE NOT EXISTS
      (SELECT *
       FROM pracownicy p
       WHERE p.nr_zakl = z.nr_zakl AND p.plec = 'M');
```

Pytanie to może zostać zapisane w postaci dwóch prostych pytań połączonych operatorem różnicy zbiorów (MINUS) w następujący sposób:

```
SELECT z.nr_zakl, z.nazwa
FROM zaklady z
MINUS
SELECT z.nr_zakl, z.nazwa
FROM pracownicy p, zaklady z
WHERE p.nr_zakl = z.nr_zakl AND p.plec = 'M');
```

Operator *MINUS* najpierw odrzuca wiersze duplikatów z pierwszego pytania, następnie usuwa wszystkie te wiersze, które zostaną zwrócone jako wynik

działania drugiego pytania [Clk99]. Oba przedstawione pytania powinny zwrócić taki sam wynik, mogą więc zostać uznane za równoważne, choć różnią się składnią.

## 2.2. Warunek łączący dwa podzapytania we frazie *HAVING*

Rozważmy następujące pytanie zagnieżdżone:

*„Wyszukać te zakłady, w których pracuje więcej kobiet niż mężczyzn.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL wygląda następująco:

```
SELECT p1.nr_zakl
FROM pracownicy p1
WHERE p1.plec = 'K'
GROUP BY p1.nr_zakl
HAVING count(p1.nr_prac) > (SELECT count(p2.nr_prac)
                             FROM pracownicy p2
                             WHERE p2.plec = 'M'
                             AND p2.nr_zakl = p1.nr_zakl);
```

Przy przekształcaniu powyższego pytania na postać niezagnieżdżoną należy uwzględnić fakt, że funkcja agregująca *count* zachowuje się inaczej niż pozostałe funkcje agregujące w przypadku występowania wartości *NULL*. I tak na przykład  $sum(NULL) = NULL$ , natomiast  $count(NULL) = 0$ . Ten fakt może rodzić problemy w przypadkach zakładów, w których pracują sami mężczyźni lub same kobiety.

Pytanie to może zostać zapisane w postaci pytania niezagnieżdżonego w trzech krokach w następujący sposób:

1. Należy wyznaczyć zakłady z liczbą kobiet w nich pracujących:

```
SELECT z.nr_zakl, count(p.nr_prac) as liczba
INTO liczbaK
FROM pracownicy p JOIN zaklady z
ON p.nr_zakl = z.nr_zakl
WHERE p.plec = 'K'
GROUP BY p.nr_zakl;
```

2. Należy wyznaczyć zakłady z liczbą mężczyzn w nich pracujących:

```
SELECT z.nr_zakl, count(p.nr_prac) as liczba
INTO liczbaM
FROM pracownicy p JOIN zaklady z
ON p.nr_zakl = z.nr_zakl
WHERE p.plec = 'M'
GROUP BY p.nr_zakl;
```



3. Pytanie końcowe - wypisujące zakłady, w których pracuje więcej kobiet niż mężczyzn:

```
SELECT k.nr_zakl
FROM liczbaK k, liczbaM m
ON k.nr_zakl = m.nr_zakl
WHERE k.liczba > m.liczba;
```

Wadą tego rozwiązania jest fakt, iż tak zapisane pytanie pominie zakłady w których nie pracują kobiety, jak również te, w których nie pracują mężczyźni.

W pracy zaproponowano wykorzystanie mechanizmu złączenia zewnętrznego, by oba pytania: zagnieżdżone i niezagnieżdżone zwracały ten sam zbiór wierszy.

1. Należy wyznaczyć zakłady z liczbą kobiet w nich pracujących, uwzględniając również takie zakłady, w których nie pracuje żadna kobieta:

```
SELECT z.nr_zakl, count(p.nr_prac) as liczba
INTO liczbaK
FROM pracownicy p RIGHT OUTER JOIN zaklady z
ON p.nr_zakl = z.nr_zakl
WHERE p.plec = 'K'
GROUP BY p.nr_zakl;
```

2. Należy wyznaczyć zakłady z liczbą mężczyzn w nich pracujących, uwzględniając również takie zakłady w których nie pracuje żaden mężczyzna:

```
SELECT z.nr_zakl, count(p.nr_prac) as liczba
INTO liczbaM
FROM pracownicy p RIGHT OUTER JOIN zaklady z
ON p.nr_zakl = z.nr_zakl
WHERE p.plec = 'M'
GROUP BY p.nr_zakl;
```

3. Pytanie końcowe - wypisujące zakłady, w których pracuje więcej kobiet niż mężczyzn:

```
SELECT k.nr_zakl
FROM liczbaK k, liczbaM m
ON k.nr_zakl = m.nr_zakl
WHERE k.liczba > m.liczba;
```

Przedstawione powyżej pytanie zagnieżdżone i pytanie niezagnieżdżone ze złączeniem zewnętrznym i tabelami tymczasowymi, operują na tych samych zbiorach danych, analizują warunki w tej samej kolejności, powinny więc dawać w odpowiedzi ten sam zbiór wierszy. Można więc je uznać za równoważne.

W kilku ostatnich przykładach zaproponowano w procesie przekształcania pytania zagnieżdżonego w niezagnieżdżone zastosowanie tabel tymczasowych, przechowujących wyniki pośrednie. Metoda ta jest bardzo efektywna, zwłaszcza w pytaniach skorelowanych, w których w pytaniu wewnętrznym we frazie *SELECT* występuje funkcja agregująca i dla których dla każdego wiersza pytania zewnętrznego należy wyliczyć wartość funkcji agregującej w pytaniu wewnętrznym.

Podsumowując można stwierdzić, że podstawowym powodem zastępowania pytań zagnieżdżonych pytaniami niezagnieżdżonymi jest skrócenie czasu ich wykonywania. Jeśli jest to możliwe to najlepiej wykonać ten proces w jednym kroku, zastępując zagnieżdżenie odpowiednim złączeniem [Clk99]. Jeśli pytania zagnieżdżonego nie da się przekształcić w niezagnieżdżone w jednym kroku, wówczas efektywną metodą jest utworzenie tabel tymczasowych przechowujących wyniki pośrednie, zwłaszcza, gdy proces dotyczy pytań skorelowanych.

Śledząc proces postępowania przy przekształcaniu pytań zagnieżdżonych w niezagnieżdżone, można stwierdzić, że dla dowolnych wierszy, jeśli warunki postawione w pytaniu są identyczne i kolejność ich wykonywania zachowana, oba rodzaje pytań powinny zwrócić ten sam wynik (zbiór wierszy) [YuMng98] [Clk99].

## 2.2. Teoria zbiorów rozmytych

W rozdziale tym przedstawiono wybrane elementy teorii zbiorów rozmytych, których zastosowanie stanowi podstawę przeprowadzonych badań prezentowanych w kolejnych rozdziałach rozprawy doktorskiej. Na początku przedstawione zostaną podstawowe pojęcia związane ze zbiorami rozmytymi. Następnie zdefiniujemy operacje wykonywane na zbiorach rozmytych. W dalszej części omówiona zostanie arytmetyka liczb rozmytych.

### 2.2.1. Zbiory rozmyte – pojęcia podstawowe

Teoria zbiorów rozmytych została wprowadzona przez L.A. Zadeh'a. Podstawowym pojęciem tej teorii jest **zbiór rozmyty**, zdefiniowany jako [Zdh65]:

zbiór par, w pewnej numerycznej przestrzeni rozważań  $X$

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

gdzie:

$\mu_A$  – **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 do zdefiniowanego zbioru lub nie, funkcja przynależności przyjmuje wtedy odpowiednio wartość 1 lub 0).

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

Funkcja przynależności może być wyrażona w postaci: diagramu (ciągłego lub dyskretnego), wzoru matematycznego, tabeli, wektora przynależności.

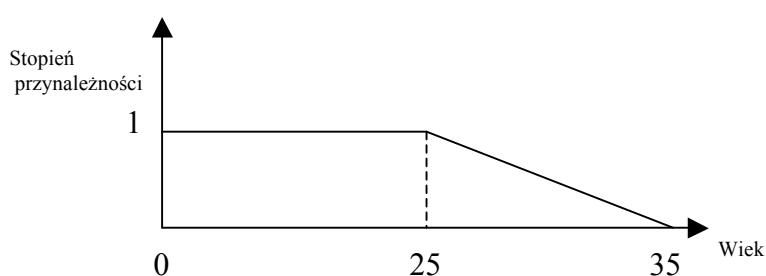
W przypadku, gdy przestrzeń  $X$  jest przestrzenią liczb rzeczywistych, można funkcję przynależności przedstawić w postaci funkcyjnej, np.:  $\mu_A(x) = 1/(1+x^2)$ , gdzie  $x \in X$ . Zbiory rozmyte są wykorzystywane do reprezentacji pojęć o niesprecyzowanych granicach.

### Przykład 2.5

Należy utworzyć podzbiór ludzi młodych, oparty na dziedzinie „*wiek ludzi*”. Przyjmijmy, że dziedziną zbioru *wiek ludzi* jest przedział  $[0, 120]$  (pod pojęciem **dziedziny** będziemy rozumieli przedział, w którym funkcja przynależności przyjmuje wartości nietrywialne ( $>0$ )).

Funkcja przynależności do podzbioru rozmytego reprezentującego wartość rozmytą *młody człowiek* może wyglądać na przykład jak na rys.2.2.

W teorii zbiorów rozmytych wprowadza się pojęcie **wartości rozmytej**. *Wartość rozmyta* jest nieprecyzyjną wartością pewnego atrybutu, np. na zbiorze *wiek* może być określona wartość rozmyta *około 30*. Każda z wartości rozmytych jest scharakteryzowana przez podzbiór rozmyty, który posiada określoną funkcję przynależności.

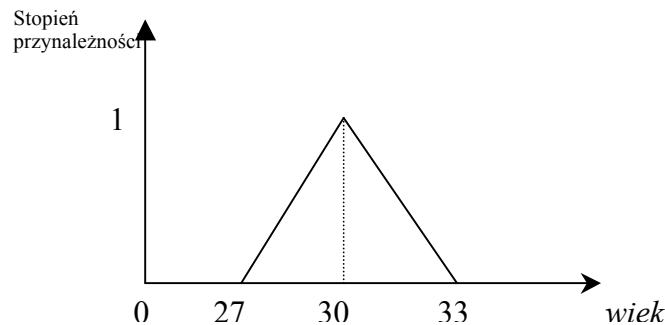


Rys. 2.2. Funkcja przynależności podzbioru rozmytego ‘młody człowiek’

Jeśli zbiór rozmyty określony jest na uniwersum rzeczywistym, to jest nazywany **liczbą rozmytą** [Łchw01].

*Przykłady liczb rozmytych to:*

- około zera, mniej więcej 5, trochę więcej niż 9, mniej więcej pomiędzy 10 i 12.
- Liczba rozmyta *wiek około 30* może mieć dziedzinę  $[27,33]$ , np. jak na rys. 2.3.

Rys. 2.3 Funkcja przynależności zbioru rozmytego *około 30*

**Zmienna lingwistyczna** to wielkość, która przyjmuje jako swe wartości, nie zwykłe wartości numeryczne, lecz **wartości lingwistyczne**[Kcp01]. Każdej wartości zmiennej lingwistycznej przyporządkować można zbiór rozmyty[ChenŁsk01].

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 [Czg97].

#### Najczęściej stosowane klasy funkcji przynależności:

– **funkcja trójkątna o parametrach a, b, c:**

$$\mu_A(x; a, b, c) = \begin{cases} 0, & x \leq a, \\ \frac{x-a}{b-a}, & a < x \leq b, \\ \frac{c-x}{c-b}, & b < x \leq c, \\ 0, & x > c \end{cases}$$

**lub w nieco prostszej postaci funkcja trójkątna symetryczna** (gdy trójkąt jest równoramienny):

$$\mu(x) = w \left( \frac{a - |x - b|}{a} \right),$$

gdzie zmienna logiczna  $w(x)$  jest określona:

$$w = \begin{cases} 1, & \text{gdy } (b-a) \leq x \leq (b+a) \\ 0, & \text{w każdym pozostałym przypadku} \end{cases}$$

– **funkcja trapezowa o parametrach a, b, c, d:**

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

W przypadku, gdy:  $b = c$  jest to funkcja trójkątna, gdy:  $a = b$  i  $c = d$  prostokątna lub przedziałowa.

– **funkcja Gaussa o parametrach m,  $\sigma > 0$ :**

$$\mu_A(x; m, \sigma) = e^{-\left(\frac{x-m}{\sigma}\right)^2}$$

– **funkcja uogólniona dzwonowa o parametrach m,  $\gamma$ ,  $\sigma > 0$ :**

$$\mu_A(x; m, \sigma, \gamma) = \frac{1}{1 + \left| \frac{x-m}{\sigma} \right|^{2\gamma}}$$

gdzie parametr m określa szerokość zbioru rozmytego, a parametry  $\sigma$  i  $\gamma$  nachylenie jego zboczy.

– **funkcja sigmoidalna o parametrach c,  $\beta$ :**

$$\mu_A(x; c, \beta) = \frac{1}{1 + \exp[-\beta(x-c)]}$$

gdzie parametr c określa punkt krzyżowania, a parametr  $\beta$  nachylenie.

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

**Proces defuzyfikacji**, zwany również **wyostrzaniem**, jest przekształceniem odwrotnym do rozmywania, czyli transformacją wartości z dziedziny zbiorów rozmytych do dziedziny liczb rzeczywistych. Przekształcenie to można zrealizować na wiele sposobów.

### 2.2.2. Operacje na zbiorach rozmytych

W klasycznej teorii zbiorów zdefiniowane są operacje sumy zbiorów, iloczynu czy dopełnienia. Wszystkie operacje na zbiorach rozmytych, będące uogólnieniem pojęć zbiorów nierozmytych, sprowadzają się do klasycznych, gdy podzbiory rozmyte mają stopień przynależności dwuwartościowy  $\{0,1\}$ .

W teorii zbiorów rozmytych należy zadać pytanie „w jakim stopniu” badany element należy do sumy lub przecięcia zbiorów, jeśli wiadomo „w jakim stopniu” należy do każdego z sumowanych lub przecinanych zbiorów [CzgPdr85].

W zbiorach rozmytych operację iloczynu, sumy czy dopełnienia rozmytego można wykonywać różnymi metodami. Najczęściej stosowanymi operatorami iloczynu (przecięcia) zbiorów są *t-normy*, a operatorami sumy *s-normy*, stanowiące różne formy realizacji tych operacji. Operatory sumy i iloczynu muszą spełniać pewne warunki, by można było je stosować w odniesieniu do zbiorów rozmytych.

Przemienność, łączność, monotoniczność i odpowiednie elementy neutralne, to cztery warunki, które muszą spełniać operatory *t-normy* i *s-normy* [YgrFlv95].

Operator *t-normy* jest funkcją  $\mathbf{T}: [0,1] \times [0,1] \rightarrow [0,1]$  modelującą operację iloczynu (*AND*) dwóch zbiorów rozmytych  $A$  i  $B$  o własnościach:

- $T(\mu_A(x), \mu_B(x)) = T(\mu_B(x), \mu_A(x))$  – przemienność,
- $T(T(\mu_A(x), \mu_B(x)), \mu_C(x)) = T(\mu_A(x), T(\mu_B(x), \mu_C(x)))$  – łączność,
- $T(\mu_A(x), \mu_B(x)) \leq T(\mu_A(x), \mu_C(x))$  dla  $\mu_B(x) \leq \mu_C(x)$  – monotoniczność,
- $T(\mu_A(x), 1) = \mu_A(x)$  – warunek brzegowy.

Operator *s-normy* jest funkcją  $\mathbf{S}: [0,1] \times [0,1] \rightarrow [0,1]$  modelującą operację sumy (*OR*) dwóch zbiorów rozmytych  $A$  i  $B$  o własnościach:

- $S(\mu_A(x), \mu_B(x)) = S(\mu_B(x), \mu_A(x))$  – przemienność,
- $S(S(\mu_A(x), \mu_B(x)), \mu_C(x)) = S(\mu_A(x), S(\mu_B(x), \mu_C(x)))$  – łączność,
- $S(\mu_A(x), \mu_B(x)) \leq S(\mu_A(x), \mu_C(x))$  dla  $\mu_B(x) \leq \mu_C(x)$  – monotoniczność,
- $S(\mu_A(x), 0) = \mu_A(x)$  – warunek brzegowy.

W przypadku, gdy zbiór rozmyty degeneruje się do zbioru ostrego, operator sumy rozmytej (iloczynu rozmytego) zachowuje się jak klasyczny operator sumy (iloczynu).

Najczęściej stosowane operatory *t-normy* to:

- Zadeha – minimum  $T(\mu_A(x), \mu_B(x)) = \min(\mu_A(x), \mu_B(x))$ , postać algebraiczna  
 $\min(\mu_A(x), \mu_B(x)) = (\mu_A(x) + \mu_B(x) - |\mu_A(x) - \mu_B(x)|)/2$
- iloczyn algebraiczny  $T(\mu_A(x), \mu_B(x)) = \mu_A(x) \cdot \mu_B(x)$ ,
- iloczyn logiczny (Łukasiewicza)  $T(\mu_A(x), \mu_B(x)) = \max(\mu_A(x) + \mu_B(x) - 1, 0)$ ,
- iloczyn drastyczny  $T(\mu_A(x), \mu_B(x)) = \min(\mu_A(x), \mu_B(x))$  dla  $\max(\mu_A(x), \mu_B(x)) = 1$ ,  
a 0 dla pozostałych.

Najczęściej stosowane operatory *s-normy* to:

- Zadeha – maksimum  $S(\mu_A(x), \mu_B(x)) = \max(\mu_A(x), \mu_B(x))$ , postać algebraiczna  
 $\max(\mu_A(x), \mu_B(x)) = (\mu_A(x) + \mu_B(x) + |\mu_A(x) - \mu_B(x)|)/2$
- suma probabilistyczna  $S(\mu_A(x), \mu_B(x)) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$ ,

- suma logiczna (Łukasiewicza)  $S(\mu_A(x), \mu_B(x)) = \min(\mu_A(x) + \mu_B(x), 1)$ ,
- suma drastyczna  $S(\mu_A(x), \mu_B(x)) = \max(\mu_A(x), \mu_B(x))$  dla  $\min(\mu_A(x), \mu_B(x)) = 0$ ,  
a 1 dla pozostałych.

**Inne rodziny norm [YgrFlv95], [Pgt99]:**

**Rodzina norm Yagera:**

- *t-norma*:  $\mu_A(x) * \mu_B(x) = 1 - \min(1, ((1 - \mu_A(x))^p + (1 - \mu_B(x))^p)^{1/p})$ , dla  $p > 0$ .
- *s-norma*:  $\mu_A(x) * \mu_B(x) = \min(1, (\mu_A(x)^p + \mu_B(x)^p)^{1/p})$ , dla  $p > 0$ ,

**Normy Hamachera:**

- *t-norma*:  $\mu_A(x) * \mu_B(x) = T_\gamma(\mu_A(x), \mu_B(x)) =$   

$$= (\mu_A(x) \cdot \mu_B(x)) / (\gamma + (1 - \gamma) \cdot (\mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x))),$$
 dla  $\gamma \geq 0$ ;  
 przy czym dla  $\gamma = 0$ ,  $\mu_A(x) = 0$  i  $\mu_B(x) = 0$  norma przyjmuje wartość 0,
- *s-norma*:  $\mu_A(x) * \mu_B(x) = S_\gamma(\mu_A(x), \mu_B(x)) =$   

$$= (\mu_A(x) \cdot \mu_B(x) \cdot (\gamma - 2) + \mu_A(x) + \mu_B(x)) / (\mu_A(x) \cdot \mu_B(x) \cdot (\gamma - 1) + 1),$$
 dla  $\gamma \geq 0$ ;  
 przy czym dla  $\gamma = 0$ ,  $\mu_A(x) = 1$  i  $\mu_B(x) = 1$  norma przyjmuje wartość 1.

**Dopełnienie (negacja) zbioru rozmytego:**

Funkcja  $N: [0, 1] \rightarrow [0, 1]$  jest funkcją modelującą operację dopełnienia dwóch zbiorów rozmytych  $A$  i  $B$  o własnościach:

- $N(0) = 1$ , (warunek brzegowy),
- $\mu_A(x) \leq \mu_B(x) \Rightarrow N(\mu_A(x)) \geq N(\mu_B(x))$ , (monotoniczność),
- $N(N(\mu_A(x))) = \mu_A(x)$  (inwolucja).

W zastosowaniach rozmytych używa się głównie dopełnienia (negacji) klasycznego (mnogościowego).

Innymi znanymi operacjami negacji są:

- $\lambda$ -dopełnienie (ang.  $\lambda$ -complement, Sugeno class complement):  

$$\mu_{\neg \lambda A}(x) = (1 - \mu_A(x)) / (1 + \lambda \mu_A(x)) \quad \text{dla } \lambda \in (-1, +\infty),$$
- $\omega$ -dopełnienie ( $\omega$ -complement, Yager class complement):  

$$\mu_{\neg \omega A}(x) = (1 - \mu^\omega(x))^{1/\omega} \quad \text{dla } \omega \in (0, +\infty).$$

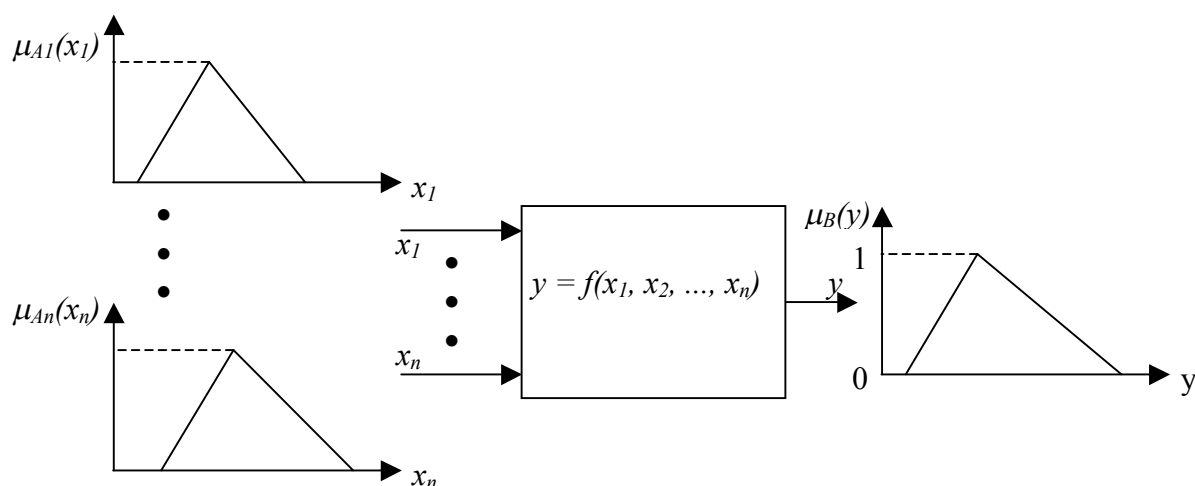
Punktem stałym negacji jest taki punkt  $\mu_A(x) \in (0, 1)$ , dla którego  $N(\mu_A(x)) = \mu_A(x)$ ; punktem stałym dla dopełnienia mnogościowego jest  $1/2$ .

### 2.2.3. Arytmetyka liczb rozmytych

**Arytmetyka konwencjonalna** określa sposób realizacji takich operacji jak dodawanie, odejmowanie, mnożenie czy dzielenie na liczbach nierozmytych.

**Arytmetyka liczb rozmytych** określa sposób realizacji tych operacji na liczbach rozmytych, np.: *około 50, mniej więcej 25, w przybliżeniu 18*. Definiuje ona podstawowe operacje matematyczne na liczbach rozmytych przez rozszerzenie ich z liczb nierozmytych. Sposób rozszerzania tych operacji wyjaśnia **zasada rozszerzania L. A. Zadeha** [Pgt99], przedstawiona na rysunku 2.4:

Wektor wejściowy  $X$  zdefiniowany jest w przestrzeni iloczynu kartezjańskiego (przestrzeni rozważań) każdego z wejść  $X_1 \times X_2 \times \dots \times X_n$ .



Rys. 2.4. Zasada rozszerzania Zadeha

Funkcja  $f$  odwzorowuje elementy zbioru z przestrzeni rozważań wektora wejściowego  $X$  w przestrzeń rozważań wyjścia  $Y$ ,  $f: X_1 \times X_2 \times \dots \times X_n \rightarrow Y$ .

Jeśli  $A_1, A_2, \dots, A_n$  są zbiorami rozmytymi zdefiniowanymi w przestrzeniach  $X_1, X_2, \dots, X_n$  poszczególnych wejść, to **zasada rozszerzania** pozwala na określenie zbioru rozmytego  $B = f(A_1, \dots, A_n)$  na wyjściu systemu, według wzoru:

$$B(y) = \vee [A_1(x_1) \wedge A_2(x_2) \wedge \dots \wedge A_n(x_n)] = \{\mu_B(y)/y \mid y = f(X), X \in X_1 \times X_2 \times \dots \times X_n\}$$

dla wszystkich  
 $X \in X_1 \times X_2 \times \dots \times X_n$   
 takich, że  $f(X) = y$

Najczęściej przestrzenie rozważań  $X_i$  oraz  $Y$  są przestrzeniami liczb rzeczywistych  $R$ .

Praktycznie operacja rozszerzania (dla wielu argumentów) sprowadza się do obliczenia funkcji przynależności wyjścia systemu:

$$\mu_B(y) = \vee_{y = f(x_1, \dots, x_n)} (\mu_{A1}(x_1) \wedge \mu_{A2}(x_2) \wedge \dots \wedge \mu_{An}(x_n)), \text{ dla każdego } x_1, x_2, \dots, x_n, y \in R,$$

gdzie:

$\vee$  - to operator połączenia zbiorów (np. *s-normy*),

$\wedge$  - to operator przecięcia zbiorów (np. *t-normy*).



### Dodawanie liczb rozmytych

Dodawanie dwóch liczb rozmytych jest odwzorowaniem wektora wejściowego  $X$  określonego w przestrzeni iloczynu kartezjańskiego  $R \times R$  na wyjście  $y$  zdefiniowane w przestrzeni rzeczywistej  $R$ .

Jeśli  $A_1$  i  $A_2$  są liczbami rozmytymi, to ich suma jest także liczbą rozmytą, obliczoną według wzoru:

$$(A_1 + A_2)(y) = \vee [A_1(x_1) \wedge A_2(x_2)], \text{ dla każdego } x_1, x_2, y \in R.$$

$$y = x_1 + x_2$$

W praktyce obliczenie sumy liczb rozmytych nie jest niczym innym jak obliczeniem funkcji przynależności  $\mu_{A_1 + A_2}(y)$ :

$$\mu_{A_1 + A_2}(y) = \vee [\mu_{A_1}(x_1) \wedge \mu_{A_2}(x_2)], \text{ dla każdego } x_1, x_2, y \in R.$$

$$y = x_1 + x_2$$

gdzie:

$\vee$  - to operator połączenia zbiorów (*np. s-normy*),

$\wedge$  - to operator przecięcia zbiorów (*np. t-normy*).

### Odejmowanie liczb rozmytych

Jeśli  $A_1$  i  $A_2$  są liczbami rozmytymi, to ich różnica jest także liczbą rozmytą, którą można obliczyć według wzoru:

$$(A_1 - A_2)(y) = \vee [A_1(x_1) \wedge A_2(x_2)], \text{ dla każdego } x_1, x_2, y \in R.$$

$$y = x_1 - x_2$$

W praktyce obliczenie różnicy liczb rozmytych jest obliczeniem funkcji przynależności  $\mu_{A_1 - A_2}(y)$ :

$$\mu_{A_1 - A_2}(y) = \vee [\mu_{A_1}(x_1) \wedge \mu_{A_2}(x_2)], \text{ dla każdego } x_1, x_2, y \in R.$$

$$y = x_1 - x_2$$

gdzie:

$\vee$  - to operator połączenia zbiorów (*np. s-normy*),

$\wedge$  - to operator przecięcia zbiorów (*np. t-normy*).

Zasada rozszerzania, choć dokładna, jest bardzo pracochłonna. Z tego powodu stosuje się uproszczone formy operacji arytmetycznych oparte na **reprezentacji L-R liczb rozmytych**.

**Liczba rozmyta typu L-R lub liczba L-R** (*ang. L-R fuzzy number*) to zbiór rozmyty  $A$  określony na uniwersum liczb rzeczywistych, którego funkcja przynależności ma postać:

$$\mu_A(x) = \begin{cases} L\left(\frac{m-x}{\alpha}\right) & \text{dla } x < m \\ 1 & \text{dla } x = m \\ R\left(\frac{x-m}{\beta}\right) & \text{dla } x > m \end{cases},$$

gdzie  $\alpha, \beta > 0$  to rozrzuty lewo- i prawostronny (*ang. left and right spreads*),

$m$  – to ustalona wartość modalna,

$L, R$  – to ustalone funkcje bazowe, zwane też funkcjami odniesienia.

Liczbę rozmytą zapisujemy za pomocą trójki  $(m, \alpha, \beta)$ .

**Przedział rozmyty typu  $L$ - $R$**  (*ang. L-R fuzzy interval*) to zbiór rozmyty  $A$  określony na uniwersum liczb rzeczywistych, którego funkcja przynależności ma postać:

$$\mu_A(x) = \begin{cases} L\left(\frac{m-x}{\alpha}\right) & \text{dla } x < m \\ 1 & \text{dla } m \leq x \leq n \\ R\left(\frac{x-n}{\beta}\right) & \text{dla } x > n \end{cases},$$

gdzie  $\alpha, \beta, m, n \in R; m < n$ ;

$\alpha, \beta > 0$  to rozrzuty lewostronny i prawostronny (*ang. left and right spreads*),

$L, R$  – to ustalone funkcje bazowe, zwane też funkcjami odniesienia,

Przedział rozmyty zapisujemy za pomocą czwórki  $(m, n, \alpha, \beta)$ .

W rozdziale tym zostaną przedstawione wybrane operacje arytmetyczne na liczbach typu  $L$ - $R$ , wykorzystywane w dalszej części pracy.

### **Dodawanie liczb typu $L$ - $R$**

Jeśli liczby rozmyte  $A_1$  i  $A_2$  przedstawione są w postaci trójek:

$$A_1 = (m_{A1}, \alpha_{A1}, \beta_{A1}), \quad A_2 = (m_{A2}, \alpha_{A2}, \beta_{A2})$$

a ich suma w postaci:

$$A_1 + A_2 = (m_{A1} + m_{A2}, \alpha_{A1} + \alpha_{A2}, \beta_{A1} + \beta_{A2}),$$

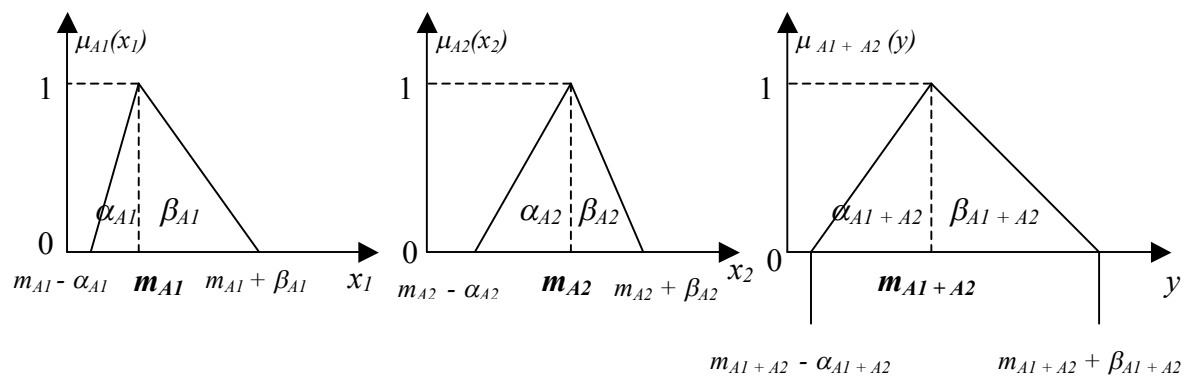
to zachodzą następujące zależności:

$$m_{A1 + A2} = m_{A1} + m_{A2},$$

$$m_{A1 + A2} - \alpha_{A1 + A2} = (m_{A1} - \alpha_{A1}) + (m_{A2} - \alpha_{A2}),$$

$$m_{A1 + A2} + \beta_{A1 + A2} = (m_{A1} + \beta_{A1}) + (m_{A2} + \beta_{A2}),$$

przedstawione na rysunku 2.5:



Rys. 2.5. Charakterystyczne parametry dodawanych liczb rozmytych

Na podstawie tych zależności można obliczyć parametry sumy liczb rozmytych  $(A_1 + A_2)$

$$\alpha_{A_1 + A_2} = \alpha_{A_1} + \alpha_{A_2},$$

$$\beta_{A_1 + A_2} = \beta_{A_1} + \beta_{A_2},$$

zatem suma w reprezentacji  $L-R$  ma postać:

$$A_1 + A_2 = (m_{A_1 + A_2}, \alpha_{A_1 + A_2}, \beta_{A_1 + A_2}) = (m_{A_1} + m_{A_2}, \alpha_{A_1} + \alpha_{A_2}, \beta_{A_1} + \beta_{A_2})$$

### Przykład 2.6

Obliczyć sumę dwóch symetrycznych liczb rozmytych  $A_1(x_1)$  = „około 25” i  $A_2(x_2)$  = „około 4”. Funkcje przynależności tych liczb zdefiniowane są następującymi wzorami:

$$\mu_{A_1}(x_1) = \frac{1}{1 + \left(\frac{x_1 - 25}{5}\right)^2}, \quad -\infty < x_1 < +\infty$$

$$\mu_{A_2}(x_2) = \frac{1}{1 + \left(\frac{x_2 - 4}{2}\right)^2}, \quad -\infty < x_2 < +\infty$$

Reprezentacja  $L-R$  tych liczb ma postać:

$$A_1 = (m_{A_1}, \alpha_{A_1}, \beta_{A_1}) = (25, 5, 5),$$

$$A_2 = (m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = (4, 2, 2),$$

$$\text{a ich suma: } A_1 + A_2 = (m_{A_1} + m_{A_2}, \alpha_{A_1} + \alpha_{A_2}, \beta_{A_1} + \beta_{A_2}) = (29, 7, 7).$$

### Odejmowanie liczb typu $L-R$

Jeśli liczby rozmyte  $A_1$  i  $A_2$  przedstawione są w postaci trójek:

$$A_1 = (m_{A_1}, \alpha_{A_1}, \beta_{A_1}), \quad A_2 = (m_{A_2}, \alpha_{A_2}, \beta_{A_2})$$

a ich różnica w postaci:

$$A_1 - A_2 = (m_{A_1 - A_2}, \alpha_{A_1 - A_2}, \beta_{A_1 - A_2}),$$

to zachodzą następujące zależności:

$$m_{A_1 - A_2} = m_{A_1} - m_{A_2},$$

$$m_{A_1 - A_2} - \alpha_{A_1 - A_2} = (m_{A_1} - \alpha_{A_1}) - (m_{A_2} + \beta_{A_2}),$$

$$m_{A_1 - A_2} + \beta_{A_1 - A_2} = (m_{A_1} + \beta_{A_1}) - (m_{A_2} - \alpha_{A_2}), \text{ z których wynika, że:}$$

$$\alpha_{A_1 - A_2} = \alpha_{A_1} + \beta_{A_2},$$

$$\beta_{A_1 - A_2} = \alpha_{A_2} + \beta_{A_1},$$

Zapis różnicy liczb rozmytych  $A_1$  i  $A_2$  w postaci  $L-R$  wygląda następująco:

$$A_1 - A_2 = (m_{A_1}, \alpha_{A_1}, \beta_{A_1}) - (m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = (m_{A_1} - m_{A_2}, \alpha_{A_1} + \beta_{A_2}, \alpha_{A_2} + \beta_{A_1}).$$

### **Przykład 2.7**

Obliczyć różnicę liczb  $A_1 = (25, 5, 7)$  i  $A_2 = (6, 3, 2)$ , stosując reprezentację  $L-R$ .

$$A_1 - A_2 = (m_{A_1} - m_{A_2}, \alpha_{A_1} + \beta_{A_2}, \alpha_{A_2} + \beta_{A_1}) = (19, 7, 10).$$

### **Mnożenie liczb typu $L-R$**

Stosując reprezentację liczb rozmytych typu  $L-R$  iloczyn dwóch liczb rozmytych  $A_1 * A_2$  można obliczyć w sposób przybliżony.

#### ***mnożenie dwóch dodatnich liczb typu $L-R$***

Zależności między parametrami dodatnich liczb rozmytych  $A_1, A_2$  oraz ich iloczynem  $A_1 * A_2$  są następujące:

$$m_{A_1 A_2} = m_{A_1} m_{A_2},$$

$$m_{A_1 A_2} - \alpha_{A_1 A_2} = (m_{A_1} - \alpha_{A_1})(m_{A_2} - \alpha_{A_2}),$$

$$m_{A_1 A_2} + \beta_{A_1 A_2} = (m_{A_1} + \beta_{A_1})(m_{A_2} + \beta_{A_2}), \text{ z których wynika, że:}$$

$$\alpha_{A_1 A_2} = m_{A_1} \alpha_{A_2} + m_{A_2} \alpha_{A_1} - \alpha_{A_1} \alpha_{A_2},$$

$$\beta_{A_1 A_2} = m_{A_1} \beta_{A_2} + m_{A_2} \beta_{A_1} + \beta_{A_1} \beta_{A_2},$$

Zapis iloczynu dwóch dodatnich liczb rozmytych  $A_1$  i  $A_2$  w postaci  $L-R$  jest następujący:

$$A_1 * A_2 = (m_{A_1}, \alpha_{A_1}, \beta_{A_1})(m_{A_2}, \alpha_{A_2}, \beta_{A_2}) =$$

$$(m_{A_1} m_{A_2}, m_{A_1} \alpha_{A_2} + m_{A_2} \alpha_{A_1} - \alpha_{A_1} \alpha_{A_2}, m_{A_1} \beta_{A_2} + m_{A_2} \beta_{A_1} + \beta_{A_1} \beta_{A_2}), \text{ dla } A_1, A_2 > 0.$$

### **Przykład 2.8**

Obliczyć iloczyn liczb  $A_1 = (25, 5, 7)$  i  $A_2 = (6, 3, 2)$ , stosując reprezentację  $L-R$ .

$$\begin{aligned} A_1 * A_2 &= (m_{A_1} m_{A_2}, m_{A_1} \alpha_{A_2} + m_{A_2} \alpha_{A_1} - \alpha_{A_1} \alpha_{A_2}, m_{A_1} \beta_{A_2} + m_{A_2} \beta_{A_1} + \beta_{A_1} \beta_{A_2}) = \\ &= (150, 90, 94). \end{aligned}$$

#### ***mnożenie dodatniej i ujemnej liczby typu $L-R$***

Przybliżony iloczyn dodatniej i ujemnej liczby typu  $L-R$  oblicza się w następujący sposób:

– parametry liczb  $A_1$  i  $A_2$  powiązane są zależnościami:

$$m_{A_1 A_2} = m_{A_1} m_{A_2},$$

$$m_{A_1 A_2} - \alpha_{A_1 A_2} = (m_{A_1} + \beta_{A_1})(m_{A_2} - \alpha_{A_2}),$$

$$m_{A_1 A_2} + \beta_{A_1 A_2} = (m_{A_1} - \alpha_{A_1})(m_{A_2} + \beta_{A_2}), \text{ z których wynika, że:}$$

$$\alpha_{A_1 A_2} = m_{A_1} \alpha_{A_2} - m_{A_2} \beta_{A_1} + \alpha_{A_2} \beta_{A_1},$$

$$\beta_{A_1A_2} = m_{A_1}\beta_{A_2} - m_{A_2}\alpha_{A_1} - \alpha_{A_1}\beta_{A_2},$$

– przybliżony iloczyn dodatniej  $A_1 > 0$  i ujemnej  $A_2 < 0$  liczby rozmytej  $L-R$  jest wyznaczany następująco:

$$A_1 * A_2 \approx (m_{A_1}, \alpha_{A_1}, \beta_{A_1})(m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = \\ (m_{A_1}m_{A_2}, m_{A_1}\alpha_{A_2} - m_{A_2}\beta_{A_1} + \alpha_{A_2}\beta_{A_1}, m_{A_1}\beta_{A_2} - m_{A_2}\alpha_{A_1} - \alpha_{A_1}\beta_{A_2}).$$

#### **mnożenie ujemnej i dodatniej liczby typu L-R**

Dla liczb rozmytych typu  $L-R$  ujemnej  $A_1 < 0$  i dodatniej  $A_2 > 0$  ich iloczyn oblicza się według następującego wzoru:

$$A_1 * A_2 \approx (m_{A_1}, \alpha_{A_1}, \beta_{A_1})(m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = \\ (m_{A_1}m_{A_2}, -m_{A_1}\beta_{A_2} + m_{A_2}\alpha_{A_1} + \beta_{A_2}\alpha_{A_1}, -m_{A_1}\alpha_{A_2} + m_{A_2}\beta_{A_1} - \beta_{A_1}\alpha_{A_2}).$$

#### **mnożenie ujemnych liczb typu L-R**

Dla ujemnych liczb rozmytych typu  $L-R$   $A_1 < 0$  i  $A_2 < 0$  iloczyn oblicza się według następującego wzoru:

$$A_1 * A_2 \approx (m_{A_1}, \alpha_{A_1}, \beta_{A_1})(m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = \\ (m_{A_1}m_{A_2}, -m_{A_1}\beta_{A_2} - m_{A_2}\beta_{A_1} + \beta_{A_2}\beta_{A_1}, -m_{A_1}\alpha_{A_2} - m_{A_2}\alpha_{A_1} + \alpha_{A_1}\alpha_{A_2}).$$

#### **mnożenie zer rozmytych typu L-R**

Jeśli liczby  $A_1$  i  $A_2$  są liczbami rozmytymi typu „około zero”, to parametry ich iloczynu  $A_1 * A_2$  powiązane są zależnościami:

$$m_{A_1A_2} = m_{A_1}m_{A_2} = 0, \\ \alpha_{A_1A_2} = \min(\alpha_{A_1}\beta_{A_2}, \alpha_{A_2}\beta_{A_1}), \\ \beta_{A_1A_2} = \max(\alpha_{A_1}\alpha_{A_2}, \beta_{A_1}\beta_{A_2}),$$

stąd przybliżony iloczyn zer rozmytych wyznaczamy następująco:

$$A_1 * A_2 \approx (0, \alpha_{A_1}, \beta_{A_1})(0, \alpha_{A_2}, \beta_{A_2}) = \\ (0, \min(\alpha_{A_1}\beta_{A_2}, \alpha_{A_2}\beta_{A_1}), \max(\alpha_{A_1}\alpha_{A_2}, \beta_{A_1}\beta_{A_2})).$$

#### **mnożenie liczb typu L-R o dowolnym znaku**

Uproszczony iloczyn dwóch liczb rozmytych o nieokreślonym znaku może być obliczony w oparciu o reprezentację liczb  $L-R$ .

Parametry liczb  $A_1$  i  $A_2$  powiązane są zależnościami:

$$m_{A_1A_2} = m_{A_1}m_{A_2}, \\ \alpha_{A_1A_2} = m_{A_1}m_{A_2} - \min[(m_{A_1} - \alpha_{A_1})(m_{A_2} - \alpha_{A_2}), (m_{A_1} - \alpha_{A_1})(m_{A_2} + \beta_{A_2}), \\ (m_{A_1} + \beta_{A_1})(m_{A_2} - \alpha_{A_2}), (m_{A_1} + \beta_{A_1})(m_{A_2} + \beta_{A_2})], \\ \beta_{A_1A_2} = \max[(m_{A_1} - \alpha_{A_1})(m_{A_2} - \alpha_{A_2}), (m_{A_1} - \alpha_{A_1})(m_{A_2} + \beta_{A_2}), \\ (m_{A_1} + \beta_{A_1})(m_{A_2} - \alpha_{A_2}), (m_{A_1} + \beta_{A_1})(m_{A_2} + \beta_{A_2})] - m_{A_1}m_{A_2},$$

**Dzielenie liczb typu L-R**

Stosując reprezentację liczb rozmytych typu L-R, iloraz dwóch liczb rozmytych  $A_1 / A_2$  można obliczyć w sposób przybliżony.

***dzielenie dwóch dodatnich liczb typu L-R***

Zależności między parametrami dodatnich liczb rozmytych  $A_1, A_2$  oraz ich ilorazem  $A_1 / A_2$  są następujące:

$$m_{A_1/A_2} = m_{A_1}/m_{A_2},$$

$$m_{A_1/A_2} - \alpha_{A_1/A_2} = (m_{A_1} - \alpha_{A_1}) / (m_{A_2} - \alpha_{A_2}),$$

$$m_{A_1/A_2} + \beta_{A_1/A_2} = (m_{A_1} + \beta_{A_1}) / (m_{A_2} + \beta_{A_2}), \text{ z których wynika, że:}$$

$$\alpha_{A_1/A_2} = (m_{A_1}\beta_{A_2} + m_{A_2}\alpha_{A_1}) / m_{A_2}(m_{A_2} + \beta_{A_2}), m_{A_2} \neq 0$$

$$\beta_{A_1/A_2} = (m_{A_1}\alpha_{A_2} + m_{A_2}\beta_{A_1}) / m_{A_2}(m_{A_2} - \alpha_{A_2}), (m_{A_2} - \alpha_{A_2}) \neq 0.$$

Zapis ilorazu dwóch dodatnich liczb rozmytych  $A_1$  i  $A_2$  w postaci L-R jest następujący:

$$\begin{aligned} A_1 / A_2 &= (m_{A_1}, \alpha_{A_1}, \beta_{A_1}) / (m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = \\ &= (m_{A_1}/m_{A_2}, (m_{A_1}\beta_{A_2} + m_{A_2}\alpha_{A_1})/m_{A_2}(m_{A_2} + \beta_{A_2}), (m_{A_1}\alpha_{A_2} + m_{A_2}\beta_{A_1})/m_{A_2}(m_{A_2} - \alpha_{A_2})), \\ &\text{dla } A_1, A_2 > 0. \end{aligned}$$

***dzielenie dodatniej liczby L-R przez ujemną liczbę L-R***

Zapis ilorazu liczb rozmytych  $A_1 > 0$  i  $A_2 < 0, A_2 \neq 0$  w reprezentacji L-R jest następujący:

$$\begin{aligned} A_1 / A_2 &= (m_{A_1}, \alpha_{A_1}, \beta_{A_1}) / (m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = \\ &= (m_{A_1}/m_{A_2}, (m_{A_1}\beta_{A_2} - m_{A_2}\beta_{A_1})/m_{A_2}(m_{A_2} + \beta_{A_2}), (m_{A_1}\alpha_{A_2} - m_{A_2}\alpha_{A_1})/m_{A_2}(m_{A_2} - \alpha_{A_2})). \end{aligned}$$

***dzielenie ujemnej liczby L-R przez dodatnią liczbę L-R***

Zapis ilorazu liczb rozmytych  $A_1 < 0$  i  $A_2 > 0, A_2 \neq 0$  w reprezentacji L-R jest następujący:

$$\begin{aligned} A_1 / A_2 &= (m_{A_1}, \alpha_{A_1}, \beta_{A_1}) / (m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = \\ &= (m_{A_1}/m_{A_2}, (-m_{A_1}\alpha_{A_2} + m_{A_2}\alpha_{A_1})/m_{A_2}(m_{A_2} - \alpha_{A_2}), (-m_{A_1}\beta_{A_2} + m_{A_2}\beta_{A_1})/m_{A_2}(m_{A_2} + \beta_{A_2})). \end{aligned}$$

***dzielenie dwóch liczb ujemnych typu L-R***

Zapis ilorazu liczb rozmytych  $A_1 < 0$  i  $A_2 < 0, A_2 \neq 0$  w reprezentacji L-R jest następujący:

$$\begin{aligned} A_1 / A_2 &= (m_{A_1}, \alpha_{A_1}, \beta_{A_1}) / (m_{A_2}, \alpha_{A_2}, \beta_{A_2}) = \\ &= (m_{A_1}/m_{A_2}, (-m_{A_1}\alpha_{A_2} - m_{A_2}\beta_{A_1})/m_{A_2}(m_{A_2} - \alpha_{A_2}), (-m_{A_2}\alpha_{A_1} - m_{A_1}\beta_{A_2})/m_{A_2}(m_{A_2} + \beta_{A_2})). \end{aligned}$$

***dzielenie liczb typu L-R o dowolnym znaku***

Uproszczony iloraz dwóch liczb rozmytych o nieokreślonym znaku  $A_1, A_2, A_2 \neq 0$  może być obliczony w oparciu o reprezentację liczb L-R.

Parametry liczb  $A_1$  i  $A_2$  powiązane są zależnościami:

$$m_{A_1/A_2} = m_{A_1}/m_{A_2},$$

$$\alpha_{A_1/A_2} = m_{A_1}/m_{A_2} - \text{MIN}[(m_{A_1} - \alpha_{A_1})/(m_{A_2} - \alpha_{A_2}), (m_{A_1} - \alpha_{A_1})/(m_{A_2} + \beta_{A_2}),$$

$$\begin{aligned} & (m_{A1} + \beta_{A1})/(m_{A2} - \alpha_{A2}), (m_{A1} + \beta_{A1})/(m_{A2} + \beta_{A2}), \\ \beta_{A1/A2} = & \text{MAX}[(m_{A1} - \alpha_{A1})/(m_{A2} - \alpha_{A2}), (m_{A1} - \alpha_{A1})/(m_{A2} + \beta_{A2}), \\ & (m_{A1} + \beta_{A1})/(m_{A2} - \alpha_{A2}), (m_{A1} + \beta_{A1})/(m_{A2} + \beta_{A2})] - m_{A1}/m_{A2}, \end{aligned}$$

### Minimum i maksimum dwóch liczb rozmytych

Niech liczby rozmyte  $A_1$  i  $A_2$  przedstawione są w postaci trójek:

$$A_1 = (m_{A1}, \alpha_{A1}, \beta_{A1}), \quad A_2 = (m_{A2}, \alpha_{A2}, \beta_{A2}), \text{ dla liczb odległych od siebie:}$$

- jeśli  $m_{A1}$  jest dużo większe od  $m_{A2}$ , to  $\min(A_1, A_2) \approx A_2$ ,
- jeśli  $m_{A1}$  jest dużo większe od  $m_{A2}$ , to  $\max(A_1, A_2) \approx A_1$ .

Wzory te są dokładne jeśli funkcje przynależności tych liczb przecinają się w co najwyżej jednym punkcie.

Dla liczb bliskich sobie, gdy  $m_{A1} \approx m_{A2}$ :

- $\min(A_1, A_2) \approx (\min(m_{A1}, m_{A2}), \max(\alpha_{A1}, \alpha_{A2}), \min(\beta_{A1}, \beta_{A2}))$ ,
- $\max(A_1, A_2) \approx (\max(m_{A1}, m_{A2}), \min(\alpha_{A1}, \alpha_{A2}), \max(\beta_{A1}, \beta_{A2}))$ , przy czym wzory te są dokładne, jeżeli  $m_{A1} = m_{A2}$  i funkcje przynależności tych liczb nie mają punktów przecięcia poza wspólnym szczytem.

Wszystkie wzory dla liczb rozmytych  $LR$  można prosto przepisać dla przedziałów rozmytych  $L-R$ . Na przykład dla liczb rozmytych  $A_1$  i  $A_2$  przedstawionych w postaci czwórek parametrów:

$$A_1 = (m_{A1}, n_{A1}, \alpha_{A1}, \beta_{A1}), \quad A_2 = (m_{A2}, n_{A2}, \alpha_{A2}, \beta_{A2})$$

dodawanie przedziałów rozmytych będzie określone wzorem:

$$\begin{aligned} A_1 + A_2 &= (m_{A1}, n_{A1}, \alpha_{A1}, \beta_{A1}) + (m_{A2}, n_{A2}, \alpha_{A2}, \beta_{A2}) = \\ &= (m_{A1} + m_{A2}, n_{A1} + n_{A2}, \alpha_{A1} + \alpha_{A2}, \beta_{A1} + \beta_{A2}). \end{aligned}$$

### 2.2.4. Kwantyfikatory lingwistyczne

W języku naturalnym pojawia się klasa pojęć, które wyrażają agregację zawierającą zarówno przypadki skrajne jak i wszystkie pośrednie, Zadeh nazwał te pojęcia **kwantyfikatorami lingwistycznymi**. Przykłady pojęć pośrednich: *prawie wszystko*, *kilka*, *około*, *większość*, *prawie połowa*, *co najmniej 20%*. Pojęcia skrajne to na przykład: *wszystko*, *istnieje*. Ponieważ słowa te wyrażają informacje o proporcjach, Zadeh zaproponował, że można je przedstawić jako podzbiory rozmyte przedziału jednostkowego.

Jeśli  $Q$  jest kwantyfikatorem logicznym, takim jak na przykład *większość*, to można go przedstawić jako podzbiór rozmyty  $Q$  przedziału  $I$ , przy czym dla każdego  $r \in I$ ,  $Q(r)$  – oznacza, w jakim stopniu proporcja  $r$  odpowiada pojęciu oznaczonemu literą  $Q$  [YgrFlv95].

Jeśli  $Q$  oznacza *większość*, to jeśli  $Q(0,9) = 1$ , oznacza to, że 0,9 zupełnie (całkowicie) odpowiada kwantyfikatorowi *większość*; a  $Q(0,6) = 0,7$  oznacza, że 0,6 jest w 70% zgodne z kwantyfikatorem *większość*.

Jeśli chce się przedstawiać kwantyfikatory logiczne w terminach zbiorów rozmytych, trzeba scharakteryzować kilka specjalnych klas kwantyfikatorów:

1. Kwantyfikator logiczny  $Q$  jest **regularny niemalejący**, gdy:

- a.  $Q(0) = 0$ ,
- b.  $Q(1) = 1$ ,
- c. jeśli  $r_1 > r_2$ , to  $Q(r_1) \geq Q(r_2)$ .

Kwantyfikatory te stosuje się w sytuacji, gdy zgodność pojęcia kwantyfikatora nie maleje, gdy rośnie proporcja. Do takich kwantyfikatorów należą pojęcia takie jak: *co najmniej*, *większość*, *wszystko*.

2. Kwantyfikator logiczny  $Q$  jest **regularny nierosnący**, gdy:

- a.  $Q(0) = 1$ ,
- b.  $Q(1) = 0$ ,
- c. jeśli  $r_1 < r_2$ , to  $Q(r_1) \geq Q(r_2)$ .

Kwantyfikatory te, stosuje się w sytuacji, gdy zgodność pojęcia kwantyfikatora maleje, gdy rośnie proporcja. Do takich kwantyfikatorów należą pojęcia takie jak: *co najmniej  $\alpha$*  i *kilka*.

3. Kwantyfikator logiczny  $Q$  jest **regularny unimodalny**, gdy:

- a.  $Q(0) = Q(1) = 0$ ,
- b.  $Q(r) = 1$ , dla  $a \leq r \leq b$
- c. jeśli  $r_2 \geq r_1 \geq a$ , to  $Q(r_1) \geq Q(r_2)$ ,
- d. jeśli  $r_2 \geq r_1 \geq b$ , to  $Q(r_2) \geq Q(r_1)$ .

Do takich kwantyfikatorów należą pojęcia takie jak: *około  $\alpha$* .



### **3. Konstrukcja zapytań w języku SQL, zawierających wartości rozmyte**

Można przyjąć, że pytania, które użytkownik kieruje do bazy danych, są przez niego w pierwotnej postaci formułowane w języku naturalnym. Należy się wtedy spodziewać w sformułowaniu takich pytań określeń nieprecyzyjnych lub nie mających swoich odpowiedników w języku SQL, stanowiącym obecnie standardowy język zapytań w bazach danych. Rozszerzenie języka SQL, przez wprowadzenie elementów teorii zbiorów rozmytych, umożliwi użytkownikowi formułowanie zapytań zawierających nieprecyzyjnie podane warunki.

Treść tego rozdziału stanowi analiza i konstrukcja zapytań w języku SQL zawierających wartości rozmyte i zastosowanie w tym podejściu teorii zbiorów rozmytych. Przedstawiono w nim problemy, jakie się w związku z tym pojawiają i zaproponowano autorskie rozwiązania tych problemów. Należą do nich m.in.:

1. Konstrukcja i interpretacja rozmytych warunków filtrujących w zapytaniach SQL.
2. Wprowadzenie do zapisu zapytań wartości progowej warunku na stopień zgodności.
3. Rozmyta agregacja danych w zapytaniach SQL.
4. Rozmyte grupowanie danych w zapytaniach SQL.
5. Kwantyfikatory rozmyte.
6. Rozmyte warunki wiążące w pytaniach zagnieżdżonych.
7. Zależna kontekstowo interpretacja wyrazów rozmytych.
8. Proces przekształcania rozmytych zagnieżdżonych pytań w ich niezagnieżdżone odpowiedniki.

#### **3.1. Interpretacja rozmytych warunków filtrujących w zapytaniach SQL**

W klasycznych pytaniach zadawanych do bazy danych wiersz tablicy, do której odnosi się pytanie spełnia kryteria podane w zapytaniu lub nie, a funkcja charakterystyczna przyjmuje dwie wartości: 1 – gdy wiersz spełnia warunki pytania i należy do zbioru wynikowego odpowiedzi na pytanie i 0 w przeciwnym przypadku.

W pytaniach rozmytych stosujemy elementy teorii zbiorów rozmytych, a funkcją charakterystyczną określającą w jakim stopniu wiersz w bazie danych spełnia kryteria wyszukiwania podane w pytaniu jest funkcja przynależności, która może przyjmować wartości z przedziału  $[0, 1]$ .

Rozważmy proste zapytanie w języku SQL, o postaci:

```
SELECT A1, . . . , Ak  
FROM T  
WHERE W;
```

gdzie  $W$  oznacza warunek filtrujący postaci:  $X \text{ jest } A$  (gdzie  $X$  – kolumna tablicy  $T$  bazy danych,  $A$  – zadana wartość).

Zauważmy, że proces wyszukiwania wierszy spełniających warunek zadania można potraktować jak proces wnioskowania: dla każdego wiersza tablicy  $T$ , na podstawie porównania wartości kolumny  $X$  zadaną wartością  $A$ , wnioskujemy o zaliczeniu wiersza do zbioru wynikowego lub nie.

Ze względu na rodzaj wartości  $X$  i  $A$  warunki można podzielić na:

1. klasyczne, gdzie  $X$  i  $A$  przyjmują wartości ostre (dokładne),
2.  $X$  przyjmuje wartości ostre a  $A$  wartości rozmyte – reprezentowane przez funkcje przynależności,
3.  $X$  przyjmuje wartości rozmyte a  $A$  wartości ostre,
4.  $X$  i  $A$  przyjmują wartości rozmyte (reprezentowane przez funkcje przynależności).

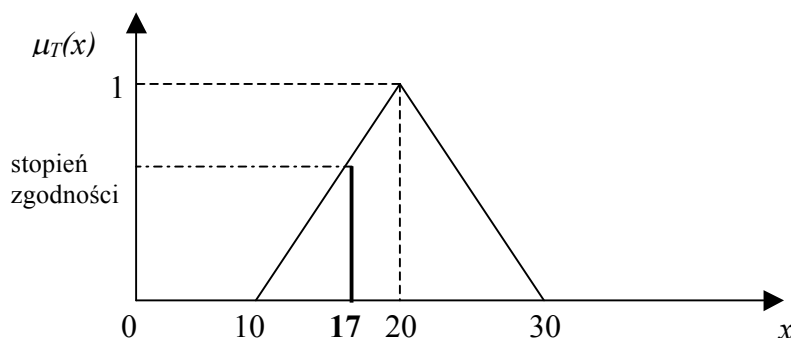
Analiza każdego przypadku przebiega nieco inaczej, ale wynik ma zawsze taką samą ogólną postać: dla każdego wiersza (określającego wartość  $X$ ) należy wyznaczyć stopień zgodności wartości  $X$  z wartością  $A$  (lub ogólniej stopień zgodności, z jakim wartość  $X$  spełnia zadany warunek).

ad 1)

W przypadku, kiedy wartości  $X$  oraz  $A$  są nierozmyte (warunek dokładny) wyznaczana jest wartość logiczna *True* (1) lub *False* (0) wyrażenia frazy *WHERE* i proces wnioskowania jest prosty: dla wartości *True* wiersz jest zaliczany do zbioru wynikowego, natomiast dla wartości *False* – nie jest.

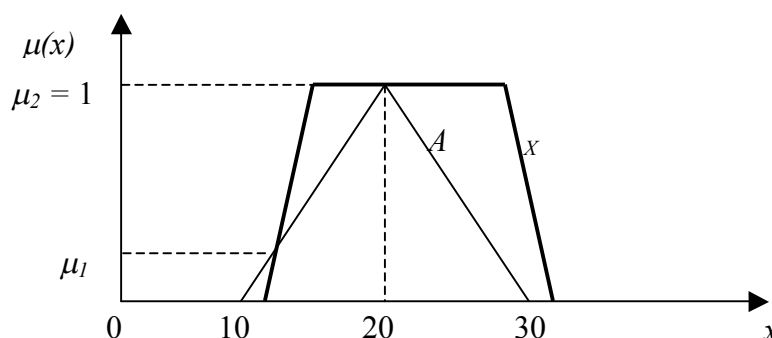
ad 2 i 3)

W przypadku, gdy jedna z wartości  $X$  lub  $A$  jest precyzyjna, a druga rozmyta, element rozmyty może być reprezentowany jako podzbiór rozmyty, dla którego zdefiniowana jest funkcja przynależności, a wartość dokładna reprezentowana jest przez linię pionową. Punkt przecięcia funkcji przynależności i linii pionowej wyznacza stopień zgodności wiersza z kryteriami pytania (rys.3.1).

Rys. 3.1 Przecięcie wartości rozmytej *około 20* z wartością precyzyjną 17

ad 4)

a) W przypadku, gdy obie wartości  $X$  i  $A$  są rozmyte, (reprezentowane przez dwa podzbiory rozmyte, dla których określone są funkcje przynależności), wtedy stopień zgodności wiersza z kryteriami pytania wyznaczany jest jako przecięcie dwóch funkcji przynależności. Gdy występuje wiele punktów przecięcia, brany jest pod uwagę ten o największej wartości [YgrFlv95]. Ilustruje to rys. 3.2. Pod uwagę brany jest stopień zgodności  $\mu_2$ , gdyż jego wartość jest większa niż  $\mu_1$ .

Rys. 3.2 Przecięcie wartości rozmytej  $A$  *około 20* zdefiniowanej przez trójkątną funkcję przynależności z wartością rozmytą  $X$  (np. *młody*) zdefiniowaną przez trapezową funkcję przynależności.

b) Jeżeli wartość kolumny  $X$  bieżącego wiersza wyrażona jest w postaci przedziału, a wartość  $A$  wyrażona jest za pomocą funkcji przynależności, to dla każdej wartości kolumny należy wyznaczyć stopień zgodności, a następnie wybrać ten o największej wartości [YgrFlv95].

Wymienione przypadki dotyczą prostego warunku porównującego jedną wartość z drugą. Klauzula *WHERE* w klasycznej instrukcji *SQL* może jednak zawierać wiele prostych warunków porównania połączonych poprzez operatory koniunkcji *AND* (*iloczyn logiczny*),

dysjunkcji *OR* (*suma logiczna*), warunki mogą również być poprzedzone operatorem negacji *NOT*.

Szczegółową analizę wybranych przypadków przedstawimy poniżej, natomiast ogólnie można powiedzieć, że w takiej klauzuli *WHERE* wartości iloczynu lub sumy rozmytej mogą być obliczane na różne sposoby, najprostszym jest zastosowanie *norm Zadeha* (*t-norma* dla operacji iloczynu polega na wyznaczeniu wartości minimalnej porównywanych stopni zgodności, *s-norma* dla operacji sumy polega na wyznaczeniu wartości maksymalnej). Można również zastosować inne operatory *s-normy* i *t-normy*, zostały one dokładnie omówione w podrozdziale 2.2.2. Natomiast, gdy warunek poprzedzony jest operatorem negacji *NOT*, wartość stopnia przynależności wyznaczana jest na przykład poprzez odjęcie od 1 początkowej wartości stopnia przynależności; można zastosować również inne metody, przedstawione w rozdziale 2.2.2 pracy.

Ze względu na rodzaj spójników łączących warunki filtrujące, warunki złożone można podzielić na:

1. Warunki koniunkcyjne:

$$W_1 \text{ AND } W_2 \text{ AND } W_3 \dots W_N,$$

Można tu wyodrębnić następujące przypadki:

- wszystkie  $W_i$  dają wartość *True* lub *False* (warunki dokładne), wtedy spójnik *AND* pełni rolę iloczynu logicznego,
- jeden z warunków  $W_i$  jest rozmyty:

Proces filtracji przebiega w tym przypadku następująco: w pierwszym kroku filtracja przebiega względem warunków dokładnych. Jeśli analizowany wiersz nie spełnia któregoś z warunków dokładnych, to nie jest on brany pod uwagę i dalsza analiza nie jest prowadzona. Jeśli analizowany wiersz je spełnia, wyznaczany jest stopień zgodności wiersza z warunkiem rozmytym  $W_i$ . Wyznaczona wartość stanowi całkowity stopień zgodności wiersza z warunkiem rozmytym.

- dwa (lub więcej) warunki  $W_i$  są rozmyte:

Wtedy, wykorzystując własność przemienności iloczynu, zgrupujemy warunki rozmyte razem, np.:

$$(W_k \text{ AND } W_m) \text{ AND } W_n \dots$$

Proces filtracji przebiega w tym przypadku następująco: w pierwszym kroku filtracja przebiega względem warunków dokładnych. Jeśli analizowany wiersz je spełnia, wyznaczany jest całkowity stopień zgodności tego wiersza z warunkami rozmytymi  $W_k$  i  $W_m$ .

Całkowity stopień zgodności dla tak połączonych warunków rozmytych wyznacza się poprzez obliczenie iloczynu rozmytego wyznaczonych stopni zgodności dla

poszczególnych warunków rozmytych  $W_k$  i  $W_m$  (w tym celu można zastosować wybrany operator *t-normy*).

2. Warunki dysjunkcyjne:

$W_1 \text{ OR } W_2 \text{ OR } W_3 \dots W_N$ ,

Można tu wyodrębnić następujące przypadki:

- wszystkie  $W_i$  są dokładne tzn. dają wartość *True* lub *False* (prawdy lub fałszu), wtedy spójnik *OR* pełni rolę sumy logicznej,

- jeden z warunków  $W_i$  jest rozmyty:

Proces filtracji przebiega w tym przypadku następująco: w pierwszym kroku filtracja przebiega względem warunków dokładnych. Jeśli którykolwiek z nich jest spełniony, to całkowity stopień zgodności wiersza jest zawsze równy 1 i wiersz jest zaliczany do zbioru wynikowego. Jeśli analizowany wiersz nie spełnia żadnego z nich, wyznaczany jest stopień zgodności tego wiersza z warunkiem rozmytym.

- dwa (lub więcej) warunki  $W_i$  są rozmyte:

Wtedy, wykorzystując własność przemienności sumy, zgrupujemy warunki rozmyte razem, np.:

$(W_k \text{ OR } W_m) \text{ OR } W_n \dots$

Proces filtracji przebiega w tym przypadku następująco: w pierwszym kroku filtracja przebiega względem warunków dokładnych. W przypadku, gdy ani jeden warunek dokładny nie jest spełniony, całkowity stopień zgodności wiersza jest równy całkowitemu stopniowi zgodności z warunkami rozmytymi. Całkowity stopień zgodności dla tak połączonych warunków rozmytych wyznacza się poprzez obliczenie sumy rozmytej stopni zgodności każdego z warunków (w tym celu można zastosować wybrany operator *s-normy*).

3. Warunki mieszane:

Jest to przypadek najbardziej ogólny. Należy wtedy odrębnie pogrupować warunki dokładne i warunki rozmyte. Dla pierwszej grupy należy wyznaczyć końcową wartość logiczną (*True* lub *False*) warunku, dla drugiej natomiast należy wyznaczyć całkowity stopień zgodności z warunkami rozmytymi. Na końcu w zależności od rodzaju spójników łączących warunki rozmyte z warunkami dokładnymi należy wyznaczyć całkowity stopień zgodności wiersza z kryteriami pytania.

Wyznaczanie stopni zgodności dla warunków rozmytych w instrukcji *SELECT*, przedstawia tabela 3.1 [YuMng98]:

Tabela 3.1

Wyznaczanie stopni zgodności w instrukcji *SELECT*

frazą <i>WHERE</i>	<b>AND</b>	<i>t-norma</i>
	<b>OR</b>	<i>s-norma</i>
	<b>NOT</b>	<i>1-<math>\mu</math> lub inne</i>

Fragment gramatyki definiujący w sposób formalny zapis warunku rozmytego ma następującą postać:

```

...
<war_rozmyty_złożony> ::= <war_rozmyty_prosty> |
                        <war_rozmyty_złożony><funktor><war_rozmyty_prosty>
<war_rozmyty_prosty> ::= <argument><operator_relacyjny><argument>
                        | <argument><operator_rozmyty><argument>
<argument> ::= <wartość_dokładna> | <wartość_rozmyta>
<operator_relacyjny> ::= > | < | <= | >= | <>
<operator_rozmyty> ::= jest | mniej_wiecej
<funktor> ::= AND | OR | AND NOT | OR NOT
...

```

W pewnych sytuacjach wiersz może należeć do tabeli ze zdefiniowanym już wcześniej dodatkowo stopniem zgodności  $\mu$ , a ponadto można wyznaczyć wartość stopnia z jakim wiersz spełnia warunki rozmyte  $\mu_i$  zadanego pytania. W tym przypadku całkowity stopień przynależności wiersza określony jest jako wartość  $\min \{\mu, \mu_i\}$ . Przypadek ten zostanie zilustrowany w przykładzie 3.4.

W wyniku wyznaczenia stopni zgodności dla frazy *WHERE*, powstaje zbiór wierszy wynikowych. Zbiór ten może zawierać wszystkie wiersze tabeli (jeśli w pytaniu nie było ostrych warunków filtrujących nałożonych na ostre dane), a zawarte w nim wiersze mogą w różnym stopniu spełniać kryteria pytania. Może więc powstać potrzeba wybrania tylko części wierszy ze zbioru wynikowego w celu udostępnienia ich użytkownikowi

Istnieją różne kryteria takiego wyboru [Pgt99], zwanego procesem wyostrzania. Najprostszym jest wybór wiersza o maksymalnym stopniu zgodności. Do innych należy na przykład wybór wierszy o stopniu zgodności przekraczającym ustaloną wartość progową. Problem ten zostanie szczegółowo rozpatrzony w podrozdziale 3.3 pracy.

### 3.2. Przykłady ilustrujące proces wnioskowania przybliżonego w interpretacji pytań SQL zadawanych do bazy danych

W rozdziale tym przedstawione zostaną przykłady zapytań SQL zawierające rozmyte warunki filtrujące zadawane do bazy danych zawierającej dokładne, jak również rozmyte dane [Młs02a].

#### Przykład 3.1

Załóżmy, że w systemie istnieje baza danych *ZAKŁADY* (przedstawiona w rozdziale 2.1.2) w tym tabela *Pracownicy*, zawierająca informacje o pracownikach. Kolumny tej tabeli zawierają precyzyjne dane (tabela 3.2). Użytkownik chce uzyskać informacje o pracownikach w wieku **około 50 lat**, mających staż pracy **około 20 lat**. Przykładowe pytanie wyrażone w języku SQL może mieć następującą postać:

```
SELECT imię, nazwisko
FROM pracownicy
WHERE wiek jest około 50 AND staz_pracy jest około 20;
```

Tabela 3.2

Tabela *Pracownicy*

Nr	Imię	Nazwisko	Nr_zakl	Wiek	Staz_pracy	Plec	Adres
1	Jan	Kowalski	1	48	19	M	Zabrze
2	Kasia	Nowak	2	38	10	K	Chorzów
3	Marcin	Sowa	3	21	1	M	Gliwice
4	Jakub	Sroka	2	53	22	M	Kraków
5	Anna	Maj	1	47	8	K	Katowice

Rozważmy (dla porównania) dwie różne postacie funkcji przynależności dla wartości rozmytych **około 50** i **około 20**:

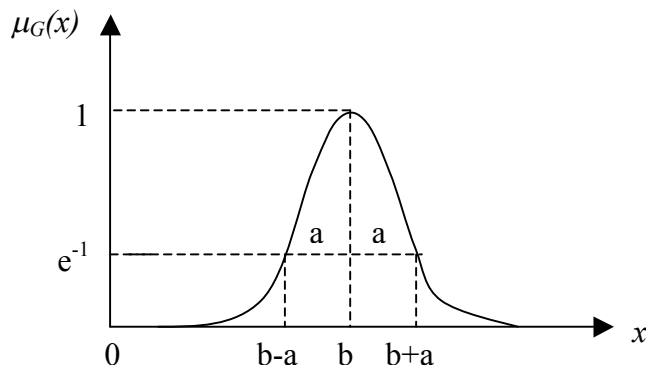
#### 1. Funkcja Gaussa opisana wzorem [Pgt99]:

$$\mu_G(x) = e^{-\left(\frac{x-b}{a}\right)^2} \quad (1)$$

Przebieg funkcji określony jest przez dwa parametry  $a$ ,  $b$ , gdzie odpowiednio:

$b$  – wartość modalna funkcji (najbardziej typowa wartość  $x$  dla zbioru rozmytego),

$a$  – szerokość tej funkcji (funkcja Gaussa ma szerokość  $2a$  dla  $\mu(x) = e^{-1} \approx 0,36788$ ).



Rys. 3.3 Funkcja przynależności typu krzywa Gaussa

Wartość modalną funkcji (parametr  $b$ ) można wyznaczyć poprzez postawienie pytania o najbardziej typową wartość  $x$  dla danego zbioru rozmytego. W celu wyznaczenia parametru  $a$ , można posłużyć się pojęciem **punktu krytycznego**  $x_k$  funkcji przynależności. Jest to taki punkt, dla którego wartość funkcji przynależności wynosi 0,5. Krzywa Gaussa posiada dwa takie punkty [Pgt99].

$$\mu_G(x_k) = e^{-\left(\frac{x_k - b}{a}\right)^2} = 0,5 \quad (2)$$

Stąd, dla zadanych  $b$  i  $x_k$ , wartość parametru  $a$  określana jest następująco:

$$a = \frac{|x_k - b|}{\sqrt{\ln 2}} \quad (3)$$

**2. Symetryczna funkcja trójkątna** opisana wzorem [Pgt99]:

$$\mu_T(x) = w \left( \frac{a - |x - b|}{a} \right), \quad (4)$$

gdzie parametry  $a$  i  $b$  oznaczają odpowiednio:

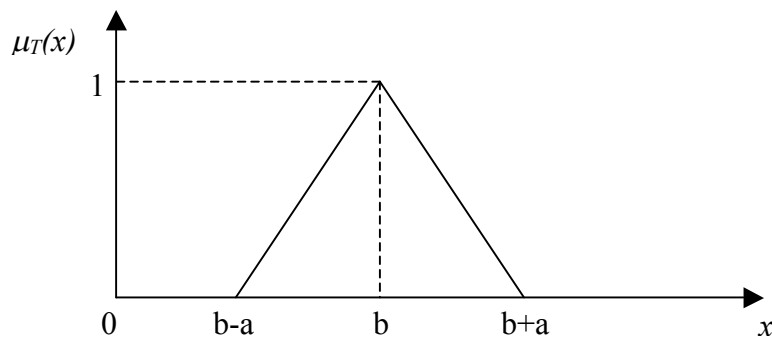
$a$  – wartość modalna funkcji przynależności,

$b$  - szerokość funkcji,

a współczynnik  $w$  wynosi:

$$w = \begin{cases} 1 & \text{gdy } (b - a) \leq x < (b + a), \\ 0 & \text{dla } x \text{ poza tym zakresem.} \end{cases} \quad (5)$$





Rys. 3.4 Trójkątna symetryczna funkcja przynależności

Przyjmijmy, że funkcje przynależności określone dla poszczególnych wyrazów rozmytych rozpatrywanego przykładu mają postać:

**wiek około 50**

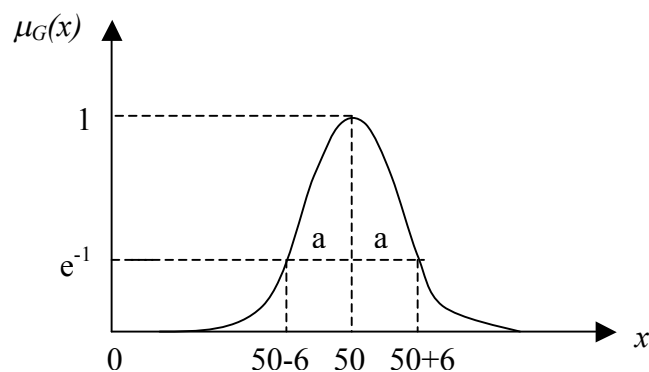
–  $\mu_G(x) = e^{-\left(\frac{x-50}{6}\right)^2}$ , gdzie  $x$  – wiek, jako  $b$  przyjęto 50, a jako punkt krytyczny  $x_k = 45$ .

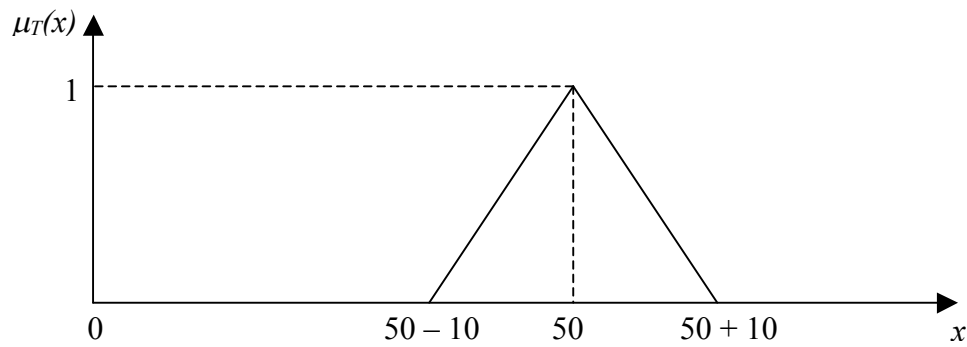
Z równania (3) wyznaczono parametr  $a = 6$ .

–  $\mu_T(x) = w \left( \frac{10 - |x - 50|}{10} \right)$ , gdzie  $x$  – wiek, a jako parametry  $b$  i  $a$  w równaniu (4) przyjęto

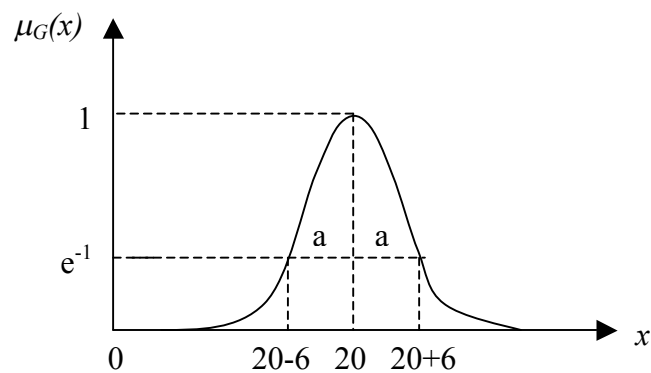
odpowiednio 50 i 10. Współczynnik  $w$  z równania (5) wynosi:

$$w = \begin{cases} 1, & \text{gdy } 40 \leq x < 60, \\ 0, & \text{dla } x \text{ poza tym zakresem.} \end{cases}$$

Rys. 3.5 Wartość rozmyta *wiek około 50* wyrażona funkcją Gaussa

Rys. 3.6 Wartość rozmyta *wiek około 50* wyrażona symetryczną funkcją trójkątną***staż pracy około 20***

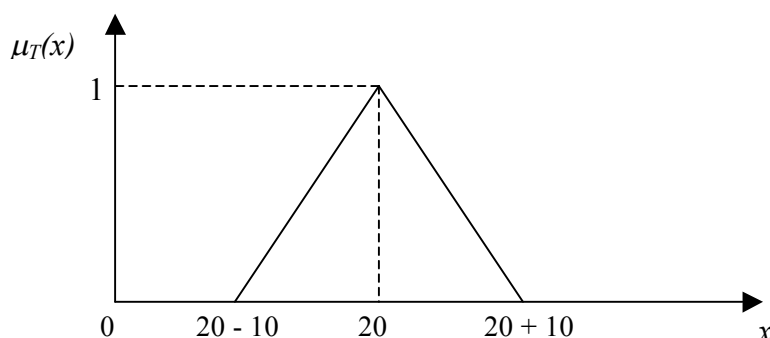
- $\mu_G(x) = e^{-\left(\frac{x-20}{6}\right)^2}$ , gdzie  $x$  – *staż pracy*, jako  $b$  przyjęto 20, a jako punkt krytyczny  $x_k = 15$ . Z równania (3) wyznaczono parametr  $a = 6$ .

Rys. 3.7 Wartość rozmyta *staż pracy około 20* wyrażona funkcją Gaussa

- $\mu_T(x) = w \left( \frac{10 - |x - 20|}{10} \right)$ , gdzie  $x$  – *staż pracy*, a jako parametry  $b$  i  $a$  z równania (4)

przyjęto odpowiednio 20 i 10. Współczynnik  $w$  z równania (5) wynosi:

$$w = \begin{cases} 1, & \text{gdy } 10 \leq x < 30, \\ 0, & \text{poza tym zakresem.} \end{cases}$$

Rys. 3.8 Wartość rozmyta *staż pracy około 20* wyrażona symetryczną funkcją trójkątną

a) Interpretacja zapytania SQL-owego w przykładzie 3.1, przy uwzględnieniu różnych funkcji przynależności wartości rozmytych, będzie przebiegać następująco: Dla wartości rozmytych *wiek około 50* i *staż pracy około 20*, których funkcje przynależności zdefiniowano za pomocą funkcji Gaussa:

- wyznaczono stopnie zgodności rozpatrywanych kolumn w każdym wierszu tabeli *Pracownicy* (tabela 3.3).

Tabela 3.3

Tabela *Pracownicy* z obliczonymi stopniami zgodności

Nr	Imie	Nazwisko	Wiek	$\mu_{50}(\text{Wiek})$	Staz_pracy	$\mu_{20}(\text{Staz\_pracy})$	Plec	Adres
1	Jan	Kowalski	48	0,895	19	0,973	M	Zabrze
2	Kasia	Nowak	38	0,018	10	0,062	K	Chorzów
3	Marcin	Sowa	21	0,001	1	0,001	M	Gliwice
4	Jakub	Sroka	53	0,779	22	0,895	M	Kraków
5	Anna	Maj	47	0,779	8	0,018	K	Katowice

- dla każdego wiersza wyznaczono całkowity stopień zgodności z kryteriami zadawanego pytania (stosując jeden z operatorów iloczynu rozmytego – *t-normę* Zadeh’a, wyznaczającą wartość minimalną spośród porównywanych stopni zgodności):

$$\tau = \text{MIN} (\mu_{50}(\text{Wiek}), \mu_{20}(\text{Staz\_pracy})),$$

Wyniki po wykonaniu tej operacji przedstawiono w tabeli 3.4.

Tabela 3.4

Tabela *Pracownicy* z całkowitym stopniem zgodności dla każdego wiersza

Nr	Imie	Nazwisko	Wiek	$\tau$	Staz_pracy	Plec	Adres
1	Jan	Kowalski	48	0,895	19	M	Zabrze
2	Kasia	Nowak	38	0,018	10	K	Chorzów
3	Marcin	Sowa	21	0,001	1	M	Gliwice

4	Jakub	Sroka	53	0,779	22	M	Kraków
5	Anna	Maj	47	0,018	8	K	Katowice

Otrzymano zbiór wyjściowy zawierający wiersze z odpowiadającymi im stopniami zgodności do zadanego pytania. Na tym etapie, w zależności od wymagań użytkownika, należy zastosować odpowiednią metodę wyboru wierszy spełniających kryteria pytania.

W tablicy 3.4 wyróżniono 2 wiersze o największym stopniu zgodności. Podobna reguła będzie stosowana w następnych przykładach, zaś pełniejsza analiza problemu tego wyboru będzie przedstawiona w rozdziale 3.3.

*b)* Odpowiednio dla wartości rozmytych *wiek około 50* i *staż\_pracy około 20* zdefiniowanych za pomocą funkcji trójkątnej, po wykonaniu opisanych wcześniej operacji, otrzymano wyniki przedstawione w tabeli 3.5:

Tabela 3.5

Tabela *Pracownicy* z wyliczonymi stopniami zgodności

Nr	Imie	Nazwisko	Wiek	$\mu_{50}(\text{Wiek})$	Staż_pracy	$\mu_{20}(\text{Staż_pracy})$	Plec	Adres
1	Jan	Kowalski	48	0,8	19	0,9	M	Zabrze
2	Kasia	Nowak	38	0,0	10	0,0	K	Chorzów
3	Marcin	Sowa	21	0,0	1	0,0	M	Gliwice
4	Jakub	Sroka	53	0,7	22	0,8	M	Kraków
5	Anna	Maj	47	0,7	8	0,0	K	Katowice

– dla każdego wiersza wyznaczono całkowity stopień zgodności (stosując *t-normę* Zadeh’a):

$$\tau = \text{MIN} (\mu_{50}(\text{Wiek}), \mu_{20}(\text{Staż\_pracy}))$$

Wyniki przedstawiono w tabeli 3.6.

Tabela 3.6

Tabela *Pracownicy* z całkowitym stopniem zgodności dla każdego wiersza

Nr	Imie	Nazwisko	Wiek	$\tau$	Staż_pracy	Plec	Adres
1	Jan	Kowalski	48	0,8	19	M	Zabrze
2	Kasia	Nowak	38	0,0	10	K	Chorzów
3	Marcin	Sowa	21	0,0	1	M	Gliwice
4	Jakub	Sroka	53	0,7	22	M	Kraków
5	Anna	Maj	47	0,0	8	K	Katowice

Otrzymany zbiór wyjściowy to wiersze z odpowiadającymi im stopniami zgodności z warunkami zadanego pytania. Podobnie jak w przykładzie poprzednim można ograniczyć zbiór wynikowy, zgodnie z wymaganiami użytkownika, wykorzystując odpowiednie metody wyboru wierszy spełniających kryteria pytania.

**Przykład 3.2**

Rozpatrzmy tabelę *Zapotrzebowanie* należącą do bazy danych *ZAKŁADY* (przedstawioną w rozdziale 2.1.2). Tabela ta zawiera szacunkowe informacje o rocznych potrzebach zakładów w zakresie papieru, tonerów oraz płytek CD. Przyjmijmy, że wartości w tych kolumnach określone są w postaci przedziałów (tabela 3.7). Załóżmy, że szukamy jednostek, których roczne zapotrzebowanie wynosi **około 20 ryz papieru** i **około 5 tonerów**.

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE papier jest około 20 AND toner jest około 5;
```

Tabela 3.7

Tabela *Zapotrzebowanie*

Nr_zakl	Płytki_CD	Papier	Toner	Rok
1	40 – 50	19 – 21	3 – 5	2003
2	30 – 40	10 – 12	2 – 4	2003
3	20 – 30	1 – 3	1 – 2	2003
4	50 – 60	22 – 23	5 – 6	2003
5	40 – 50	7 – 10	2 – 5	2003

W przykładzie tym dla wartości rozmytej **około 20 ryz papieru** funkcje przynależności określone są za pomocą funkcji Gaussa i trójkątnej, tak samo jak w przykładzie 3.1, a dla wartości rozmytej **około 5 tonerów**, przyjmujemy że funkcje przynależności określone są następującymi wzorami:

– **funkcja Gaussa:**

$$\mu_G(x) = e^{-\left(\frac{x-5}{1,5}\right)^2},$$

gdzie  $x$  – liczba tonerów, jako parametry  $b$  i  $a$  z równania (1) przyjęto odpowiednio 5 i 1,5 (dla przyjętego punktu krytycznego  $x_k = 4$ ).

– **funkcja trójkątna:**

$$\mu_T(x) = \begin{cases} 0 & \text{dla } x \leq 3 \\ \frac{2-|x-5|}{2} & \text{dla } 3 < x \leq 7, \\ 0 & \text{dla } x > 7 \end{cases}$$

gdzie  $x$  – oznacza liczbę tonerów, a jako parametry  $b$  i  $a$  z równania (4) przyjęto odpowiednio 5 i 2.

a) Przyjmijmy, że wartości rozmyte **około 5 tonerów** i **około 20 ryz papieru** określają funkcje przynależności zdefiniowane za pomocą funkcji Gaussa. Proces poszukiwania odpowiedzi na zadane przez użytkownika pytanie przebiega następująco:

Dla każdej rozmytej kolumny tabeli *Zapotrzebowanie*, na którą nałożony jest warunek w pytaniu, wyznacza się stopnie zgodności. Jeśli w kolumnie tej występuje wartość określona przedziałem, to stopień zgodności wyznaczany jest dla każdej wartości z tego przedziału (podrozdział 3.1 punkt 4b).

Po wykonaniu niezbędnych obliczeń dla wartości rozmytych określonych za pomocą funkcji Gaussa, otrzymano wyniki przedstawione w tabeli 3.8.

Tabela 3.8

Tabela *Zapotrzebowanie* z wyliczonymi stopniami zgodności dla każdej wartości rozmytej kolumny

Nr_zakl	Płytki_CD	Papier	$\mu_{20}(\text{papier})$	Toner	$\mu_5(\text{toner})$	Rok
1	40 – 50	{19, 20, 21}	{0,973, <b>1</b> , 0,973}	{3, 4, 5}	{0,169, 0,641, <b>1</b> }	2003
2	30 – 40	{10, 11, 12}	{0,062, 0,105, <b>0,169</b> }	{2, 3, 4}	{0,018, 0,169, <b>0,641</b> }	2003
3	20 – 30	{1, 2, 3}	{0,001, 0,001, 0,001}	{1, 2}	{0,000, <b>0,018</b> }	2003
4	50 – 60	{22, 23}	{ <b>0,895</b> , 0,779}	{5, 6}	{ <b>1</b> , 0,641}	2003
5	40 – 50	{7, 8, 9, 10}	{0,009, 0,018, 0,032, <b>0,062</b> }	{2, 3, 4, 5}	{0,018, 0,169, 0,641, <b>1</b> }	2003

Spośród wyznaczonych stopni zgodności poszczególnych wartości kolumn rozmytych wyznacza się maksymalny stopień zgodności dla każdej kolumny w wierszu (podrozdział 3.1 punkt 4b).

Po wykonaniu tej operacji, wiersze z wyznaczonymi stopniami zgodności dla kolumny przedstawione są w tabeli 3.9.

Tabela 3.9

Tabela *Zapotrzebowanie* z maksymalnymi wartościami stopni zgodności kolumn rozmytych

Nr_zakl	Płytki_CD	Papier	$\mu_{20}(\text{Papier})$	Toner	$\mu_5(\text{Toner})$	Rok
1	40 – 50	20	<b>1</b>	5	<b>1</b>	2003
2	30 – 40	12	<b>0,169</b>	4	<b>0,641</b>	2003
3	20 – 30	3	<b>0,001</b>	2	<b>0,018</b>	2003
4	50 – 60	22	<b>0,895</b>	5	<b>1</b>	2003
5	40 – 50	10	<b>0,062</b>	5	<b>1</b>	2003

Dla każdego wiersza wyznaczono całkowity stopień zgodności z warunkami zadawanego pytania. Ponieważ w pytaniu wystąpił spójnik *AND*, zastosowano operator iloczynu rozmytego *t-normę* Zadeh'a:

$$\tau = \text{MIN}(\mu_{20}(\text{Papier}), (\mu_5(\text{Toner})),$$

Po wykonaniu tej operacji wiersze z wyznaczonymi stopniami zgodności przedstawione są w tabeli 3.10.

Tabela 3.10

Tabela *Zapotrzebowanie* z obliczonym całkowitym stopniem zgodności  $\tau$  dla każdego wiersza

Nr_zakl	Płytki_CD	$\tau$	Papier	toner	rok
1	40 – 50	<b>1</b>	<b>20</b>	<b>5</b>	2003
2	30 – 40	0,169	12	4	2003
3	20 – 30	0,001	3	2	2003
4	50 – 60	<b>0,895</b>	<b>22</b>	<b>5</b>	2003
5	40 – 50	0,062	10	5	2003

Otrzymany zbiór zawiera wiersze z odpowiadającymi im stopniami zgodności z warunkami zadanego pytania. Na tym etapie, w zależności od wymagań użytkownika, można zastosować odpowiednią metodę wyboru wierszy wyjściowych.

**b)** Odpowiednio dla wartości rozmytych *około 5 tonerów* i *około 20 ryz papieru*, których funkcje przynależności zdefiniowano za pomocą funkcji trójkątnej, wykonano kolejno takie same operacje jak w przykładzie 3.2a. Otrzymane wyniki przedstawione są w tabelach 3.11, 3.12 i 3.13.

Tabela 3.11

Tabela *Zapotrzebowanie* z wyliczonymi stopniami zgodności dla każdej wartości rozmytej kolumny

Nr_zakl	Płytki_CD	Papier	$\mu_{20}(\text{Papier})$	Toner	$\mu_5(\text{Toner})$	Rok
1	40 – 50	{19, 20, 21}	{0,9, <b>1</b> , 0,9}	{3, 4, 5}	{0, 0,5, <b>1</b> }	2003
2	30 – 40	{10, 11, 12}	{0,0, 0,1, <b>0,2</b> }	{2, 3, 4}	{0, 0, <b>0,5</b> }	2003
3	20 – 30	{1, 2, 3}	{0,0, 0,0, 0,0}	{1, 2}	{0, 0}	2003
4	50 – 60	{22, 23}	{ <b>0,8</b> , 0,7}	{5, 6}	{ <b>1</b> , 0,5}	2003

5	40 – 50	{7, 8, 9, 10}	{0,0, 0,0, 0,0, 0,0}	{2, 3, 4, 5}	{0, 0, 0,5, 1}	2003
---	---------	---------------	-------------------------------	--------------	----------------	------

Tabela 3.12

Tabela *Zapotrzebowanie* z maksymalnymi wartościami stopni zgodności kolumny rozmytej dla każdego wiersza

Nr_zakl	Płytki_CD	Papier	$\mu_{20}(\text{Papier})$	Toner	$\mu_5(\text{Toner})$	Rok
1	40 – 50	20	<b>1</b>	5	<b>1</b>	2003
2	30 – 40	12	<b>0,2</b>	4	<b>0,5</b>	2003
3	20 – 30	3	0,0	2	0,0	2003
4	50 – 60	22	<b>0,8</b>	5	<b>1</b>	2003
5	40 – 50	10	0,0	5	<b>1</b>	2003

Tabela 3.13

Tabela *Zapotrzebowanie* z obliczonym całkowitym stopniem zgodności  $\tau$  dla każdego wiersza

Nr_zakl	Płytki_CD	Papier	$\tau$	Toner	Rok
<b>1</b>	<b>40 – 50</b>	<b>20</b>	<b>1</b>	<b>5</b>	<b>2003</b>
2	30 – 40	12	0,2	4	2003
3	20 – 30	3	0,0	2	2003
<b>4</b>	<b>50 – 60</b>	<b>22</b>	<b>0,8</b>	<b>5</b>	<b>2003</b>
5	40 – 50	10	0,0	5	2003

Interpretacja zawartości tabeli 3.12 może być przeprowadzona podobnie jak tabeli 3.10 z podpunktu *a* tego rozdziału.

### Przykład 3.3

Rozważmy teraz przypadek najbardziej ogólny, gdzie baza danych zawiera nieprecyzyjne (rozmyte) dane i do bazy danych zadawane jest również nieprecyzyjne (rozmyte) pytanie. W poprzednim przykładzie analizie poddano przypadek, gdy kolumny przyjmowały wartości dane przedziałami. Tym razem kolumny zawierać będą wartości rozmyte zdefiniowane odpowiednimi funkcjami przynależności.

W systemie istnieje baza danych *ZAKŁADY* (przedstawiona w rozdziale 2.1.2) oraz tabela *Zapotrzebowanie* (tabela 3.14), zawierająca szacunkowe informacje o rocznych potrzebach zakładów w zakresie papieru, tonerów oraz płytek CD. W kolumnach *papier*, *toner* oraz *płytki\_CD* przechowywane są wartości rozmyte zdefiniowane za pomocą odpowiednich funkcji przynależności.

Niech kierowane do bazy danych pytanie ma postać: *Wyszukać zakłady, które złożyły zapotrzebowanie na dość dużo tonerów i jednocześnie niewiele papieru.*



Przykładowe dane zapisane we fragmencie tabeli *Zapotrzebowanie* zawiera tabela 3.14.

Tabela 3.14

Tabela *Zapotrzebowanie*

Nr_zakl	toner	papier
1	około 7	około 4
2	około 6	około 25
3	około 3	około 30
4	około 4	około 10
5	około 5	około 15

W pytaniu tym obie wartości, zarówno ta określona w kryteriach zapytania jak i wartość kolumny w tabeli są wartościami rozmytymi.

W języku SQL powyższe pytanie może być zapisane w następujący sposób:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE toner jest dosc_duzo AND papier jest niewiele;
```

Założmy, że wartości rozmyte *dosc\_duzo tonerów* i *niewiele papieru*, występujące w warunkach filtrujących pytania, opisane są trapezowymi funkcjami przynależności.

Przyjmijmy, że funkcja przynależności dla podzbioru *dosc\_duzo tonerów* opisana jest wzorem:

$$\mu(x) = \begin{cases} 0 & \text{dla } x \leq 4 \\ \frac{x-4}{2} & \text{dla } 4 < x \leq 6 \\ 1 & \text{dla } x > 6 \end{cases}$$

Przyjmijmy, że funkcja przynależności do podzbioru *niewiele papieru* jest opisana wzorem:

$$\mu(x) = \begin{cases} 1 & \text{dla } x \leq 10 \\ \frac{20-x}{10} & \text{dla } 10 < x \leq 20 \\ 0 & \text{dla } x > 20 \end{cases}$$

Dla każdej z wartości kolumn rozmytych (*toner*, *papier*) również zdefiniowany jest zbiór rozmyty, reprezentowany przez odpowiednią funkcję przynależności. Dla danych z tabeli 3.14 dla wartości kolumny *toner* zdefiniowane są odpowiednio następujące trójkątne symetryczne funkcje przynależności:

$$- \mu(x)_{\text{około } 7} = \begin{cases} 0 & \text{dla } x \leq 5 \\ \frac{2-|x-7|}{2} & \text{dla } 5 < x \leq 9, \\ 0 & \text{dla } x > 9 \end{cases} \quad \text{gdzie } x \in N,$$

$$\begin{aligned}
- \mu(x)_{okolo\ 6} &= \begin{cases} 0 & dla \quad x \leq 4 \\ \frac{2-|x-6|}{2} & dla \quad 4 < x \leq 8, \\ 0 & dla \quad x > 8 \end{cases} & gdzie \quad x \in N, \\
- \mu(x)_{okolo\ 3} &= \begin{cases} 0 & dla \quad x \leq 1 \\ \frac{2-|x-3|}{2} & dla \quad 1 < x \leq 5, \\ 0 & dla \quad x > 5 \end{cases} & gdzie \quad x \in N, \\
- \mu(x)_{okolo\ 4} &= \begin{cases} 0 & dla \quad x \leq 2 \\ \frac{2-|x-4|}{2} & dla \quad 2 < x \leq 6, \\ 0 & dla \quad x > 6 \end{cases} & gdzie \quad x \in N, \\
- \mu(x)_{okolo\ 5} &= \begin{cases} 0 & dla \quad x \leq 3 \\ \frac{2-|x-5|}{2} & dla \quad 3 < x \leq 7 \\ 0 & dla \quad x > 7 \end{cases} & gdzie \quad x \in N.
\end{aligned}$$

Dla danych z tabeli 3.14 dla wartości kolumny *papier* zdefiniowane są odpowiednio następujące trójkątne symetryczne funkcje przynależności:

$$\begin{aligned}
- \mu(x)_{okolo\ 4} &= \begin{cases} 0 & dla \quad x \leq 2 \\ \frac{2-|x-4|}{2} & dla \quad 2 < x \leq 6, \\ 0 & dla \quad x > 6 \end{cases} & gdzie \quad x \in N, \\
- \mu(x)_{okolo\ 25} &= \begin{cases} 0 & dla \quad x \leq 22 \\ \frac{3-|x-25|}{3} & dla \quad 22 < x \leq 28, \\ 0 & dla \quad x > 28 \end{cases} & gdzie \quad x \in N, \\
- \mu(x)_{okolo\ 30} &= \begin{cases} 0 & dla \quad x \leq 25 \\ \frac{5-|x-30|}{5} & dla \quad 25 < x \leq 35, \\ 0 & dla \quad x > 35 \end{cases} & gdzie \quad x \in N, \\
- \mu(x)_{okolo\ 10} &= \begin{cases} 0 & dla \quad x \leq 7 \\ \frac{3-|x-10|}{3} & dla \quad 7 < x \leq 13, \\ 0 & dla \quad x > 13 \end{cases} & gdzie \quad x \in N, \\
- \mu(x)_{okolo\ 15} &= \begin{cases} 0 & dla \quad x \leq 12 \\ \frac{3-|x-15|}{3} & dla \quad 12 < x \leq 18, \\ 0 & dla \quad x > 18 \end{cases} & gdzie \quad x \in N.
\end{aligned}$$

Stopień zgodności między kryterium pytania a wartością kolumny (reprezentowanymi przez dwa zbiory rozmyte, dla których określone są funkcje przynależności) wyznaczany jest jako przecięcie dwóch funkcji przynależności. Jeśli w wyniku wykonania tej operacji jest

wiele punktów przecięcia, brany jest pod uwagę ten o największym stopniu zgodności [YgrFlv95], [Pgt99].

Na przykład w wyniku porównania wartości rozmytych *niewiele papieru* z liczbą rozmytą *około 15*, otrzymujemy dwa stopnie przynależności:  $2/7$  i  $8/13$ . Zgodnie z algorytmem całkowity stopień przynależności wyznaczany jest jako wartość maksymalna spośród nich, tzn. wartość  $8/13$ .

Ponieważ w pytaniu są dwa warunki, proces ten jest powtarzany dla każdego z nich. Otrzymane wartości stopni zgodności przedstawione są w tabeli 3.15.

Tabela 3.15

Tabela *Zapotrzebowanie* z wyznaczonymi stopniami  
zgodności

Nr_zakl	toner	$\mu_{\text{toner}}$	papier	$\mu_{\text{papier}}$
1	około 7	1	około 4	1
2	około 6	1	około 25	0
3	około 3	$\frac{1}{4}$	około 30	0
4	około 4	$\frac{1}{2}$	około 10	1
5	około 5	$\frac{3}{4}$	około 15	$8/13$

Następnie rozważa się spójnik jaki występuje w klauzuli *WHERE* i odpowiednio wyznacza całkowity stopień zgodności wiersza z kryteriami zapytania. Po wykonaniu przedstawionych operacji zostaje wyznaczony całkowity stopień zgodności z jakim każdy wiersz spełnia warunki pytania. Wiersze z całkowitym stopniem zgodności, wyznaczonym jako wartość minimalna (dla spójnika *AND*) przedstawione są w tabeli 3.16.

Tabela 3.16

Tabela *Zapotrzebowanie* z całkowitym  
stopniem zgodności

Nr_zakl	toner	$\tau$	papier
1	około 7	1	około 4
2	około 6	0	około 25
3	około 3	0	około 30
4	około 4	$\frac{1}{2}$	około 10
5	około 5	$8/13$	około 15

W zależności od potrzeb użytkownika i zastosowania należy wyznaczyć najbardziej reprezentatywne wiersze tabeli, stosując odpowiednie metody wyboru wierszy.

#### **Przykład 3.4**

Przyjęty w pracy sposób interpretacji zapytań dopuszcza istnienie **kolumn, których zawartość tworzą stopnie zgodności** z pewnym kryterium. Rozpatrzmy na przykład bazę danych *ZAKŁADY*, która została rozbudowana, poprzez utworzenie dodatkowej tabeli *Dobrzy*

*pracownicy*. Tabela ta powstała na podstawie istniejącej już tabeli *Pracownicy*, poprzez dodanie nowej kolumny *Dobry*. Wartości w tej kolumnie utworzono przez określenie dla każdego wiersza tej tabeli stopnia zgodności z kryterium *dobrzy pracownicy*. Przykładowe dane przedstawione zostały w tabeli 3.17.

Tabela 3.17

Tabela *Dobrzy pracownicy*

Nr	Imie	Nazwisko	Wiek	Staz_pracy	Plec	Adres	Dobry
1	Jan	Kowalski	48	19	M	Zabrze	0,8
2	Kasia	Nowak	38	10	K	Chorzów	0,7
3	Marcin	Sowa	21	1	M	Gliwice	0,6
4	Jakub	Sroka	53	22	M	Kraków	0,3
5	Anna	Maj	47	8	K	Katowice	0,9

Do bazy danych zadawane jest następujące pytanie:

*Wyszukaj **dobrych** pracowników w wieku **około 50 lat** lub dobrych mających staż pracy **około 20 lat**.*

Dla wartości rozmytych *wiek około 50* i *staz\_pracy około 20* określono odpowiednie funkcje przynależności (wykorzystując funkcję Gaussa określoną w poprzednim rozdziale).

```
SELECT imie, nazwisko
FROM dobry_pracownicy
WHERE (wiek jest około 50 OR staz_pracy ~= około 20) AND dobry;
```

Obliczone stopnie zgodności przedstawione są w tabeli 3.18.

Tabela 3.18

Tabela *Dobrzy pracownicy z obliczonymi stopniami zgodności*

Nr	Imie	Nazwisko	Wiek	$\mu_{50}(\text{Wiek})$	Staz_pracy	$\mu_{20}(\text{Staz\_pracy})$	Plec	Adres	Dobry
1	Jan	Kowalski	48	0,895	19	0,973	M	Zabrze	0,8
2	Kasia	Nowak	38	0,018	10	0,062	K	Chorzów	0,7
3	Marcin	Sowa	21	0,001	1	0,001	M	Gliwice	0,6
4	Jakub	Sroka	53	0,779	22	0,895	M	Kraków	0,3
5	Anna	Maj	47	0,779	8	0,018	K	Katowice	0,9

We frazie *WHERE* jako spójnik występuje operator *OR*, zastosujemy dla niego *s-normę* Zadeh'a. Stopień zgodności wiersza wyznaczony będzie jako wartość maksymalna spośród obliczonych cząstkowych stopni zgodności, ilustruje to tabela 3.19.

Tabela 3.19

Tabela *Dobrzy pracownicy z obliczonymi stopniami zgodności względem pytania*

Nr	Imie	Nazwisko	Wiek	$\mu$	Staz_pracy	Plec	Adres	Dobry
1	Jan	Kowalski	48	0,973	19	M	Zabrze	0,8
2	Kasia	Nowak	38	0,062	10	K	Chorzów	0,7
3	Marcin	Sowa	21	0,001	1	M	Gliwice	0,6

4	Jakub	Sroka	53	0,895	22	M	Kraków	0,3
5	Anna	Maj	47	0,779	8	K	Katowice	0,9

W kolejnym kroku określa się całkowity stopień zgodności wiersza, poprzez wybranie wartości minimalnej spośród: obliczonego stopnia zgodności wiersza a wprowadzonej wartości kryterialnej (stopnia zgodności z kryterium *dobrzy pracownicy*). Ponieważ między tymi stopniami zgodności występuje operator *AND*, zastosowano *t-normę* Zadeh'a. Całkowity stopień zgodności wiersza wyznaczony będzie jako wartość minimalna obu stopni zgodności. Wiersze z wyznaczonym całkowitym stopniem przynależności przedstawione są w tabeli 3.20.

Tabela 3.20

Tabela *Dobrzy pracownicy* z obliczonym całkowitym stopniem zgodności  $\tau$ 

Nr	Imie	Nazwisko	Wiek	$\tau$	Staz_pracy	Plec	Adres
1	Jan	Kowalski	48	0,8	19	M	Zabrze
2	Kasia	Nowak	38	0,062	10	K	Chorzów
3	Marcin	Sowa	21	0,001	1	M	Gliwice
4	Jakub	Sroka	53	0,3	22	M	Kraków
5	Anna	Maj	47	0,779	8	K	Katowice

### 3.3. Wprowadzenie do zapisu zapytań SQL warunku na wartość stopnia zgodności

Wprowadzenie do pytań SQL-owych warunków rozmytych prowadzi w procesie interpretacji pytań do wyznaczenia stopnia zgodności wszystkich wyszukiwanych wierszy z kryterium pytania, co przedstawiono w podrozdziałach 3.1 i 3.2. Powstaje wtedy problem końcowego wyboru wierszy, które mają się znaleźć w odpowiedzi. Można tu wymienić kilka metod:

- wybranie wierszy o maksymalnym stopniu zgodności z warunkami pytania,
- wybranie wierszy o stopniu zgodności przekraczającym średni stopień,
- wybranie wierszy o stopniu zgodności przekraczającym pewną zadaną wartość progową.

W niniejszym rozdziale rozważymy najciekawszy, trzeci z wymienionych przypadków. Wymaga on rozwiązania sposobu przekazania takiej wartości progowej interpreterowi zapytań.

Wartość progowa stopnia zgodności może zostać wprowadzona do systemu poprzez:

- a) wstępne przyjęcie stałej wartości progowej,
- b) jawne ustalenie warunku na stopień zgodności w pytaniu SQL.

Wstępne przyjęcie wartości progowej jest prostsze w implementacji, ale ogranicza w pewnym stopniu elastyczność konstruowania zapytań. Można bowiem rozważać przypadki złożonych zapytań SQL-owych, zwłaszcza zagnieżdżonych, gdzie warunki rozmyte pojawiają się w różnych frazach zapytań i zajdzie potrzeba uwzględnienia dla różnych fraz, różnych stopni zgodności z kryteriami pytania [ChnChn02]. Dlatego w niniejszym rozdziale przedyskutujemy możliwości wprowadzenia do zapytań SQL jawnego zapisu warunku na wartość stopnia zgodności. Rozważmy trzy, arbitralnie przyjęte, formy zapisu warunku na stopień zgodności:

- forma dołączana do warunku rozmytego,
- forma funkcyjna,
- forma operatorowa.

Formy te zilustrujemy przykładem następującego pytania:

*„Wyszukaj pracowników, którzy mają około 50 lat. W odpowiedzi należy uwzględnić wiersze ze stopniem zgodności z kryterium pytania większym niż 0.7*

Dotychczasowy zapis tego pytania bez uwzględniania warunku na stopień zgodności przedstawiał się następująco:

```
SELECT nazwisko
FROM pracownicy
WHERE wiek jest okolo 50;
```

- Zapis warunku na stopień zgodności w formie dołączanej do warunku rozmytego  
Do zapisu warunku na stopień zgodności w tej formie wprowadzimy predefiniowaną zmienną *DG* (ang. *degree*). Warunek nakładany na stopień zgodności jest wtedy dołączany (poprzez koniunkcję) do zapisu warunku rozmytego we frazie *WHERE*.

```
WHERE wiek jest okolo 50 AND DG > 0.7;
```

Rozwiązanie to wymaga jednak kontekstowej analizy całego warunku, co jest trudne w implementacji.

Formalny zapis warunku rozmytego w tej formie jest zdefiniowany następująco:

```
...
//wykorzystamy definicję złożonego warunku rozmytego przedstawioną
w podrozdziale 3.1
<war_rozszerzony> ::= <war_rozmyty_zlozony> AND <war_progowy>
<war_progowy> ::= DG<operator_rel><wartosc_progowa>
<operator_rel> ::= = | < | > | <= | >= | <>
<wartosc_progowa> ::= <liczba dziesiętna z przedziału [0, 1]>
...
```

- Zapis warunku na stopień zgodności w formie funkcyjnej

Dla powiązania warunku na stopień zgodności z warunkiem rozmytym zdefiniowano funkcję DG, której użycie w rozważanym przykładzie ma następującą postać:

```
WHERE DG(wiek jest około 50) > 0.7
```

Rozpatrywano też alternatywną formę, którą ilustruje zapis:

```
DG(p.wiek, około 50) > 0.7;
```

Formalny zapis rozszerzonej pierwszej z przytoczonych postaci warunku filtrującego z wykorzystaniem tej funkcji przedstawia się następująco:

```
...
//wykorzystamy definicję złożonego warunku rozmytego przedstawioną
w podrozdziale 3.1
<war_rozszerzony> ::= DG(<war_rozmyty_złożony>)<war_progowy>
<war_progowy> ::= <operator_rel><wartość_progowa>
<operator_rel> ::= = | < | > | <= | >= | <>
<wartość_progowa> ::= <liczba dziesiętna z przedziału [0, 1]>
...
```

- Zapis warunku na stopień zgodności w formie operatorowej

Trzecia z rozpatrzonych form zapisu jest uproszczeniem formy funkcyjnej. Wykorzystanie jej do rozpatrywanego przykładu wygląda następująco:

```
WHERE (wiek jest około 50) > 0.7;
```

Interpretacja warunku rozmytego w powyższym zapisie prowadzi do wyznaczenia stopnia zgodności, który jest natychmiast porównywany z wartością progową 0.7. Formalny zapis rozszerzonej postaci warunku filtrującego dla tego przypadku jest następujący:

```
...
//wykorzystamy definicję złożonego warunku rozmytego przedstawioną
w podrozdziale 3.1
<war_rozszerzony> ::= (<war_rozmyty_złożony>)<war_progowy>
<war_progowy> ::= <operator_rel><wartość_progowa>
<operator_rel> ::= = | < | > | <= | >= | <>
<wartość_progowa> ::= liczba dziesiętna z przedziału <0, 1>
...
```

W pracy zaimplementowano formę funkcyjną i formę operatorową. Obie te formy będą używane w rozważanych dalej przykładach. Czasami dla skrócenia zapisu będziemy używać krótszego sformułowania: *pytanie w formie operatorowej*, bądź *pytanie w formie funkcyjnej*.

Celem tych przykładów jest pokazanie różnorodności zapisu warunków z uwzględnieniem wartości progowej stopnia zgodności. Taka analiza była podstawą

rozbudowy algorytmu interpretacji zapytań SQL-owych, w części dotyczącej uwzględnienia warunku na stopień zgodności.

Przykłady zapytań w języku SQL oparte są na bazie danych *ZAKŁADY*. Wykorzystywany fragment struktury bazy danych został już omówiony w rozdziale 2.1.2, lecz ze względu na czytelność zostanie w tym miejscu przedstawiony jeszcze raz. W skład bazy danych *ZAKŁADY* wchodzi między innymi następujące tabele:

```
Instytuty (nr_inst, nazwa)
Zaklady (nr_zakl, nazwa, liczba_prac, liczba_pokoi, nr_inst)
Pracownicy (nr_prac, nr_zakl, nazwisko, wiek, plec, staz_pracy)
Zapotrzebowanie (nr_zakl, rok, papier, toner, plytki CD)
Zuzycie (nr_zakl, rok, papier, toner, plytki CD)
```

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

Przechowywane w tabeli *zapotrzebowanie* dane dotyczą szacowanych (planowanych) potrzeb na dany rok.

W przedstawionych przykładach rozpatrywane warunki rozmyte będą dla wyróżnienia pogrubione.

Rozważmy na początku pytanie zawierające jeden rozmyty warunek filtrujący umieszczony we frazie *WHERE*:

```
WHERE <warunek filtrujący>;
```

a) Warunek nałożony na kolumnę zawierającą wartości dokładne

Przykładowe pytanie tej postaci zadawane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, w których pracuje około 30 pracowników. W odpowiedzi powinny się znaleźć wiersze ze stopniem zgodności z kryterium pytania równym lub większym niż 0.8.”*

Pytanie takie może zostać zapisane w języku SQL z funkcyjną formą zapisu warunku na stopień zgodności w następujący sposób:

```
SELECT nr_zakl
FROM zaklady
WHERE DG(liczba_prac jest okolo 30) >= 0.8;
```

Postać tego pytania z operatorową formą zapisu warunku na stopień zgodności jest następująca:

```
SELECT nr_zakl
FROM zaklady
WHERE (liczba_prac jest okolo 30) >= 0.8;
```



Realizację tego typu pytań bez warunku na stopień zgodności omówiono w podrozdziale 3.2. W powyższym pytaniu we frazie *WHERE* występuje wartość rozmyta *około 30*. Ogólnie można stwierdzić, że dla każdego wiersza tabeli wyznaczany jest stopień zgodności z warunkiem *liczba\_prac jest około 30*. Następnie wybierane są wiersze, których stopień zgodności jest równy lub większy niż 0.8.

b) Warunek nałożony na kolumnę zawierającą wartości rozmyte

*WHERE <warunek filtrujący>;*

Przykładowe pytanie skierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, których zapotrzebowanie na papier wynosi około 45 ryz, zaś stopień zgodności z tym warunkiem nie jest mniejszy od 0.6”*

Pytanie takie może zostać zapisane w języku SQL w formie operatorowej w następujący sposób:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE (papier jest około 45) >= 0.6;
```

We frazie *WHERE* tego pytania występuje porównanie wartości rozmytej *około 45* z wartościami rozmytymi kolumny *papier* i wyznaczenie stopnia zgodności. Każda z tych wartości reprezentowana jest przez zdefiniowaną wcześniej funkcję przynależności.

Na kolumnę rozmytą może być również nakładany dokładny warunek filtrujący. Wyznaczanie stopnia przynależności dla takiego pytania przebiega w ten sam sposób jak w przypadku nakładania warunku rozmytego na kolumnę zawierającą dokładne dane, proces ten został opisany w punkcie 1a tego podrozdziału.

Przy nakładaniu warunków filtrujących na kolumny zawierające wartości rozmyte można wykorzystywać także inne operatory relacyjne, takie jak: *<*, *>*, *>=*, *<=*. W tym przypadku wartości rozmyte należy potraktować jako liczby rozmyte (najwygodniej przedstawione w postaci *L-R*, omówione w rozdziale 2.2.3). Określenie, która z liczb jest większa, może polegać na przykład na porównaniu ich wartości modalnych, a w przypadku przedziałów rozmytych na przykład na porównaniu wartości największych, dla których funkcje przynależności przyjmują jeszcze wartość 1.

c) Nakładanie różnej wartości progowej dla każdego warunku oddzielnie

Jeśli we frazie *WHERE* wystąpi kilka warunków filtrujących, można dla każdego z nich wprowadzić oddzielny warunek na stopień zgodności z inną wartością

progową. Wtedy dla każdego warunku oddzielnie należy określić, czy jego stopień zgodności jest równy lub przekracza wartość progową czy nie. W tym przypadku warunki filtrujące mogą być połączone spójnikami logicznymi (*AND*, *OR*).

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, w których pracuje około 25 pracowników (ze stopniem zgodności nie mniejszym od 0.5) i których zapotrzebowanie na papier wyniosło 15 ryz (ze stopniem zgodności nie mniejszym od 0.8) lub 4 tonery (ze stopniem zgodności przekraczającym 0.75).”*

Pytanie to zapisane w języku SQL w formie operatorowej może mieć następującą postać [Młs03]:

```
SELECT nr_zakl
FROM zaklady z JOIN zapotrzebowanie z
ON z.nr_zakl = z.nr_zakl
WHERE (liczba_prac jest około 25) >= 0.5 AND
      ((z.papier jest 15) = 0.8 OR (z.toner jest 4) > 0.75);
```

- d) Przypadek najbardziej ogólny – warunek filtrujący złożony z kilku warunków dokładnych i rozmytych połączonych różnymi funktorami.

Przykładowe pytanie zadawane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj te zakłady, które zatrudniają około 20 pracowników i których zeszłoroczne zapotrzebowanie na tonery lub papier mniej więcej pokryło się ze zużyciem. Wyświetl te wiersze, których stopień zgodności przekracza 0.8.”*

Postać tego pytania w języku SQL może być wyrażona następująco:

```
SELECT nr_zakl
FROM zaklady z JOIN zapotrzebowanie zp
ON z.nr_zakl = zp.nr_zakl
JOIN zużycie zu ON zu.nr_zakl = zp.nr_zakl
WHERE (liczba_prac jest około 20 AND
      zu.rok = '2002' AND z.rok = '2002' AND
      (zu.toner jest zp.toner) OR (zu.papier jest zp.papier)) > 0.8;
```

Realizacja tego typu pytań wykorzystuje te same reguły, które były podane w poprzednich podrozdziałach. Analogicznie do punktów 1, 2 i 3 z rozdziału 3.1 oddzielamy warunki dokładne i rozmyte. Adekwatnie do zastosowanego funktora wyznaczamy stopień zgodności. Na końcu sprawdzany jest warunek na stopień zgodności.

### 3.4. Agregacja w pytaniach rozmytych

#### 3.4.1. Funkcje agregujące na danych rozmytych

Język SQL umożliwia między innymi wyznaczanie wartości zagregowanych (np. suma, wartość średnia i inne). Wyliczenie wartości funkcji agregujących *sum*, *average*, *min*, *max* operujących na danych rozmytych wymaga znajomości arytmetyki liczb rozmytych czyli operacji takich jak dodawanie, wyznaczanie wartości najmniejszej czy największej liczb rozmytych. W pracy założono, że liczby rozmyte są typu *L-R*. Arytmetyka liczb rozmytych i reprezentacja typu *L-R* zostały omówione w podrozdziale 2.2.3 rozprawy. Funkcja agregująca *count* zawsze zwraca liczbę analizowanych wartości, więc wyznaczona wartość jest zawsze liczbą ostrą.

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:  
„Wyznacz wartość średniego zapotrzebowania na tonery w roku 2002.”

Postać tego pytania zapisana w języku SQL jest następująca:

```
SELECT AVG(toner)
FROM zapotrzebowanie
WHERE rok = '2002';
```

Zaimplementowane w pracy funkcje agregujące działające na rozmytych danych mają takie same nazwy jak klasyczne funkcje agregujące, różnią się natomiast typem argumentu i na tej podstawie uruchamiana jest odpowiednia autorska funkcja agregująca wykonująca operacje na rozmytych lub klasycznych danych.

#### 3.4.2. Nakładanie warunków na funkcje agregujące w pytaniach rozmytych

Wartości rozmyte mogą występować również w zapisie warunków filtrujących we frazie *HAVING*, gdzie warunki filtrujące nakładane są na funkcje agregujące i ten przypadek jest szczególnie dokładnie analizowany w pracy.

W pytaniach rozmytych można wyróżnić kilka rodzajów agregacji danych [Młs03]:

- agregacja wartości ostrych, a nakładany warunek jest rozmyty,
- agregacja wartości rozmytych, a nakładany warunek jest ostry,
- agregacja wartości rozmytych i nakładany warunek jest rozmyty,
- rozmyte kwantyfikatory operujące na grupie wierszy.

Nowym problemem jest agregacja wykonywana dla kolumn zawierających rozmyte dane.

Proces wyznaczania stopnia zgodności z kryteriami warunku przebiega tak samo jak w przypadku warunku we frazie *WHERE*.

Rozpatrzmy następujące przypadki:

### 1. Agregacja wartości ostrych - nakładany warunek rozmyty

Wszystkie prowadzone dotąd rozważania dotyczące wyznaczania stopni zgodności między wartością dokładną a wartością rozmytą, dotyczące frazy *WHERE* (w rozdziale 3.3), we frazie *HAVING* są podobne.

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj te zakłady, w których jest zatrudnionych około 10 kobiet, przy czym stopień zgodności z tym warunkiem powinien być większy od 0.65.”*

Pytanie to zapisane w języku SQL w formie operatorowej może mieć następującą postać:

```
SELECT nr_zakl
FROM pracownicy
WHERE plec = 'K'
GROUP BY nr_zakl
HAVING (count(nr_prac) jest około 10) > 0.65;
```

Proces wyboru grup do wyniku wykonywany jest w typowy sposób (np. jak w przykładach z podrozdziału 3.3).

### 2. Agregacja wartości rozmytych

Zadanie wyszukiwania w takich przypadkach realizowane jest w ten sam sposób jak w poprzednim punkcie oraz punktach 1b i 2b rozdziału 3.3, z tą różnicą, że w pytaniach, tych niezbędne jest wyznaczenie wartości funkcji agregujących: *sum*, *average*, *min*, *max* operujących na wartościach rozmytych.

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj te instytuty, których sumaryczne roczne zapotrzebowanie na papier wynosiło około 1000 ryz. W odpowiedzi powinny znaleźć się wiersze o stopniu zgodności co najmniej 0.7.”*

Pytanie to zapisane w języku SQL w formie operatorowej może mieć następującą postać:

```
SELECT nr_zakl
FROM zapotrzebowanie z JOIN instytuty i
  ON z.nr_inst = i.nr_inst
GROUP BY nr_inst, rok
HAVING (sum(papier) jest około 1000) >= 0.7;
```

W przedstawionym przykładzie zastosowano funkcję agregującą *sum*, której argumentem są wartości typu rozmytego. Proces wyboru grup do wyniku wykonywany jest w typowy sposób (przykłady z podrozdziału 3.3).

### 3. Rozmyty warunek filtrujący we frazie *WHERE* i agregacja we frazie *HAVING*

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 w tego typu pytaniach, po wybraniu wierszy spełniających warunki filtrujące we frazie *WHERE*, wyodrębniane są grupy wierszy, na które nakładane są warunki wyspecyfikowane we frazie *HAVING*.

W tego typu pytaniach dwa razy wyznaczane są stopnie zgodności, najpierw dla poszczególnych wierszy, później dla grup zawierających już tylko te wiersze, które zostały wyodrębnione w pierwszym procesie wyboru wierszy.

Rozpatrzmy następujące pytanie, kierowane do bazy danych *ZAKŁADY*:

*„Wyszukaj nazwy tych zakładów, które w roku 2003 złożyły zapotrzebowanie na około 100 płytek CD oraz ich maksymalne zużycie płytek CD wynosiło około 120. W odpowiedzi powinny się znaleźć wiersze, które spełniają podane kryteria ze stopniem zgodności nie mniejszym niż 0.8.”*

Pytanie to zapisane w języku SQL w formie operatorowej może mieć następującą postać:

```
SELECT z.nr_zakl, nazwa
FROM zaklady z JOIN zapotrzebowanie zp
  ON z.nr_zakl = zp.nr_zakl
  JOIN zuzycie zu ON zp.nr_zakl = zu.nr.zakl
WHERE zp.rok = '2003' AND (zp.plytki_CD jest okolo 100) >= 0.8
GROUP BY nr_zakl, nazwa
HAVING (max(zu.plytki_CD) jest okolo 120) >= 0.8;
```

W procesie interpretacji powyższego pytania zwrócimy uwagę na następujące etapy:

- Najpierw na etapie filtracji (wykonanie frazy *WHERE*) wybierane są wiersze spełniające warunek dokładny (dotyczący roku 2003) i dla tych wierszy wyznaczany jest stopień zgodności dla warunku rozmytego:  
(*zp.plytki\_CD jest okolo 100*),
- Następnie wiersze, dla których stopień zgodności nie jest mniejszy niż 0.8 podlegają grupowaniu,

- Kolejno dla grup we frazie *HAVING* wyznaczana jest wartość maksymalna i po raz drugi wyliczany jest stopień zgodności, tym razem dotyczący tej wartości.

Z grup, dla których ten stopień zgodności spełnia warunek (nie mniejszy niż 0.8), tworzony jest wynik.

### 3.4.3. Rozmyte kwantyfikatory (rozmyte lingwistyczne funkcje agregujące)

W języku naturalnym często używa się sformułowań: *prawie wszystkie*, *prawie żaden*, *około połowa* itd., dopuszczających niepewność. Wszystkie te stwierdzenia odnoszą się do pewnej analizowanej grupy obiektów.

Określenia te mogą pełnić rolę rozmytych kwantyfikatorów, co umożliwia formułowanie zapytań typu: „Wyszukaj dane o zakładach, w których **prawie wszyscy pracownicy to mężczyźni**.” lub „Wyszukaj zakłady, w których **około połowa pracowników ma mniej niż 35 lat**.”

Zauważmy, że bezwzględne wartości (liczby pracowników) spełniające kryteria pytania są w każdym przypadku inne. Można jednak w prosty sposób unormować takie wyniki wyznaczając iloraz:

$$o = \frac{\text{Card}(G(\text{warunek}))}{\text{Card}(G)},$$

gdzie:

$G$  - oznacza grupę wierszy  $G$  o licznosci  $\text{card}(G)$  (np. pracowników zakładu  $k$ ).

$G(\text{warunek})$  - oznacza wiersze grupy  $G$  spełniające warunek (np. pracownicy zakładu  $k$  młodszy od 35 lat).

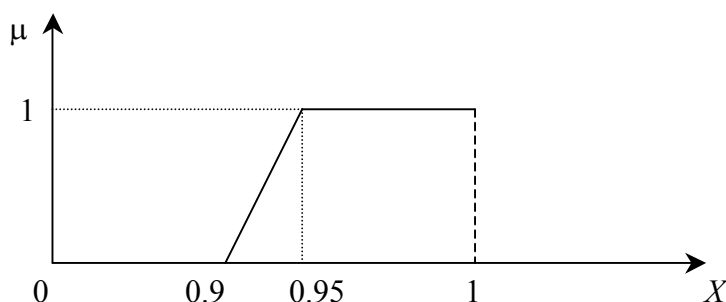
Powyższy iloraz oznacza więc *odsetek* wierszy w grupie, spełniających warunki pytania. Zauważmy, że zachodzi  $0 \leq o \leq 1$ .

Zwróćmy następnie uwagę, że określenia „*prawie wszyscy*”, „*około połowa*”, „*prawie żaden*” możemy potraktować jako tzw. wartości lingwistyczne, tzn. wartości rozmyte.

Powstaje wtedy problem zdefiniowania funkcji przynależności dla takich wartości. Jeśli argumentem funkcji przynależności byłyby liczby bezwzględne (np. liczba mężczyzn, czy liczba pracowników młodszych od 35 lat), to funkcja ta miałaby zastosowanie w jednym konkretnym pytaniu, zaś ogólnie byłaby bezużyteczna.

Zauważmy jednak, że zdefiniowanie takiej funkcji na przedziale  $[0, 1]$  (lub inaczej w procentach) pozwala skojarzyć (porównać) taką funkcję przynależności z funkcją *odsetek*. Możemy wtedy wyznaczyć stopień „*spełnienia*”, danego kwantyfikatora (np. *prawie wszyscy*), czyli inaczej – stopień zgodności warunku z kryteriami pytania.

Przyjmijmy na przykład funkcję przynależności dla wartości lingwistycznej *prawie wszyscy* jak na rysunku 3.9.



Rys. 3.9 Funkcja przynależności dla wartości lingwistycznej *prawie wszyscy*

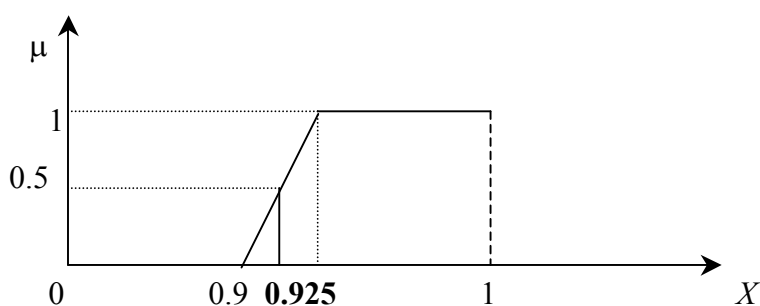
Rozważmy ponownie pytanie:

„Wyszukaj te zakłady, w których *prawie wszyscy* pracownicy to mężczyźni. W odpowiedzi powinny znaleźć się wiersze o stopniu zgodności co najmniej 0.7.”

Postać tego pytania zapisana w języku SQL może być wyrażona następująco:

```
SELECT nr_zakl, prawie_wszystkie(plec = 'M')
FROM pracownicy
GROUP BY nr_zakl
HAVING prawie_wszystkie(plec = 'M') >= 0.7;
```

Założmy, że dla zakładu numer 1 wartość funkcji odsetek wyniosła 0.925. Wtedy z wykresu funkcji przynależności odczytujemy (rysunek 3.10), że wartość stopnia zgodności wynosi w tym przypadku 0.5. Tak więc zakład numer 1 nie spełnia warunku na stopień zgodności.



Rys. 3.10 Stopień zgodności obliczonego odsetka (0.925) z wartością lingwistyczną *prawie\_wszystkie*

Zauważmy na koniec, że wartość lingwistyczna *prawie wszyscy*, którą na początku tego rozdziału określiliśmy jako kwantyfikator rozmyty, zajmuje w zapisie pytania w języku SQL miejsce funkcji agregującej. Stąd też wartości tego typu (również *okolo\_polowa*, *prawie\_zaden*) można określać jako lingwistyczne funkcje agregujące.

### 3.5. Rozmyte grupowanie danych

Grupowanie jest realizowane w języku SQL za pomocą frazy:

```
GROUP BY <lista kolumn>
```

i prowadzi ono do wyróżnienia grup wierszy o identycznych wartościach we wskazanej liście kolumn.

W przypadku, gdy rozważamy grupowanie danych w systemie, w którym możliwe jest przechowywanie danych rozmytych, rysuje się co najmniej kilka różnych podejść w zależności od potrzeb użytkownika i rodzaju przechowywanych danych. Już sam proces grupowania można podzielić na:

1. grupowanie rozmyte dokładnych danych,
2. grupowanie rozmytych danych,
3. grupowanie rozmyte rozmytych danych.

#### 3.5.1. *Grupowanie rozmyte dokładnych danych*

W klasycznym grupowaniu dokładnych danych nawet najmniejsza różnica wartości w kolumnie (kolumnach), według których następuje grupowanie, prowadzi do wyodrębnienia oddzielnych grup. Jeśli w rozmytej bazie danych wprowadzamy mechanizmy wyszukiwania danych podobnych, to naturalnym się staje oczekiwanie możliwości łączenia w grupy danych o podobnych wartościach.

W pierwszym punkcie rozważmy sytuację, gdy chcemy przeprowadzić grupowanie rozmyte dokładnych danych i chcemy znaleźć podobne dane, by umieścić je w jednej grupie.

Rozważmy w tym celu następujące algorytmy grupowania rozmytego:

1. grupowanie względem wartości lingwistycznych (określonych dla dziedziny atrybutu),
2. grupowanie według arbitralnego podziału dziedziny atrybutu,
3. grupowanie rozmyte z zastosowaniem metody hierarchicznej,
4. grupowanie rozmyte według autorskiego, zaimplementowanego algorytmu.

##### ad 1) **Grupowanie względem wartości lingwistycznych**

Grupowanie tego typu można zastosować, jeśli dziedzinę kolumny (kolumn), według której grupujemy, można opisać wartościami lingwistycznymi. Wartości te mogą być zdefiniowane na przykład za pomocą funkcji przynależności.



**Przykład 3.5**

Założmy, że w bazie danych istnieje tablica *Pomiary* z kolumną *temperatura* zawierająca wartości temperatury w poszczególnych dniach wyrażone w stopniach C (tabela 3.21) oraz, że utworzyliśmy zbiór wartości lingwistycznych, opisujących występujące temperatury, np. {bardzo zimno, zimno, chłodno, ciepło, bardzo ciepło itd}. Każda z tych wartości lingwistycznych opisana jest przez odpowiednią trapezową funkcję przynależności (podobnie jak w rozdziale 5).

Tabela 3.21  
Tabela *pomiary*

data	temperatura
02.07.03	17
08.07.03	18
11.07.03	14
14.07.03	25
21.07.03	29
25.07.03	30
31.07.03	15

W tym przypadku można każdej wartości z kolumny *temperatura* przyporządkować najbardziej odpowiadającą jej wartość lingwistyczną (mającą największą wartość stopnia zgodności). Jeśli jest kilka takich wartości, arbitralnie przypisujemy wybraną wartość lingwistyczną. Po wykonaniu takiej operacji otrzymujemy w kolumnie *temperatura* wartości lingwistyczne najbardziej pasujące do rzeczywistych danych (tabela 3.22).

Tabela 3.22  
Tabela *temperatura po przypisaniu*

data	temperatura
02.07.03	dosc_cieplo
08.07.03	cieplo
11.07.03	dosc_cieplo
14.07.03	bardzo_cieplo
21.07.03	bardzo_cieplo
25.07.03	bardzo_cieplo
31.07.03	dosc_cieplo

W następnym kroku przeprowadzamy klasyczne już grupowanie względem przyporządkowanych wartości lingwistycznych, tworząc tyle grup, ile wystąpiło różnych wartości lingwistycznych.

Założmy, że szukamy liczby dni o podobnych temperaturach, określonych wartościami lingwistycznymi.

Forma zapisu pytania wykorzystującego taki rodzaj grupowania może mieć następującą postać:

```
SELECT temperatura_to_lingw(temperatura) as temperatura,
      COUNT(w.data) as liczba_dni
FROM pomiary w
GROUP BY temperatura_to_lingw(w.temperatura);
```

gdzie funkcja `temperatura_to_lingw` przekształca rzeczywistą wartość z kolumny `temperatura` w najbardziej podobną do niej wartość lingwistyczną zmiennej `temperatura`. Odpowiedź na tak sformułowane zapytanie przedstawiono w tabeli 3.23.

Tabela 3.23

Tabela wynikowa

temperatura	liczba_dni
dosc_cieplo	3
cieplo	1
bardzo_cieplo	3

### Przykład 3.6

Kolejny przykład dotyczy badań wzrostu dzieci w poszczególnych klasach szkoły podstawowej. Należy w wyniku przeprowadzonych badań wzrostu pogrupować dzieci w klasie na bardzo niskie, niskie, średnie, wysokie i bardzo wysokie. Przyjmujemy, że w kolumnie `wzrost` tabeli opisującej uczniów przechowywane są informacje o wzroście wyrażone w centymetrach. Założmy, że zdefiniowano zbiór wartości lingwistycznych:  $\{bardzo\ ni\text{ski}, ni\text{ski}, \text{średni}, wysoki\ i\ bardz\o\ wysoki\}$  wraz z ich funkcjami przynależności. Analogicznie jak w poprzednim przykładzie możemy wtedy przeprowadzić grupowanie według kolumny `wzrost`, w oparciu o przedstawione wartości lingwistyczne, np. w celu wyznaczenia liczby dzieci w poszczególnych grupach.

Założmy w tym celu, że w bazie danych istnieje tabela:

*Uczniowie* (`nr_ucznia`, `imię`, `nazwisko`, `klasa`, `wzrost`, `waga`).

Forma zapisu pytania wykorzystującego taki rodzaj grupowania może mieć następującą postać:

```
SELECT COUNT(u.nr_ucznia), wzrost_to_lingw(u.wzrost)
FROM uczniowie u
GROUP BY wzrost_to_lingw(u.wzrost);
```

gdzie funkcja `wzrost_to_lingw` przekształca rzeczywistą wartość z kolumny `wzrost` w najbardziej podobną do niej wartość lingwistyczną zmiennej `wzrost`.

## ad 2) Grupowanie według arbitralnego podziału dziedziny atrybutu

W pewnych zadaniach wymagających grupowania wystarczającym dla użytkownika rozwiązaniem może być wyznaczenie grup według arbitralnie przyjętego, zwykle regularnego, podziału dziedziny kolumny, według której następuje grupowanie.

W większości baz danych istnieją dane, dla których można arbitralnie przyjąć pewien podział przedziałów ich wartości.

### *Przykład 3.7*

Rozważmy bazę danych pracowników pewnego zakładu pracy. Pracowników tego zakładu można podzielić na grupy względem ich stażu pracy. Można przyjąć podział według stażu pracy na: około 5 lat, około 10 lat, około 15 lat itd. Dla tych wartości można byłoby teraz zdefiniować odpowiednie funkcje przynależności. Następnie każdej z rzeczywistych wartości z kolumny trzeba przypisać najbardziej odpowiadającą jej wartość: *około 5, około 10 itd.*

Takie podejście byłoby identyczne z przedstawionym w poprzednim punkcie. Zauważmy jednak, że przedstawiona klasyfikacja stażu pracy (*około 5 lat, około 10 lat, około 15 lat itd.*) jest równoważna wyróżnieniu następujących przedziałów:

[2.5, 7.5), [7.5, 12.5), [12.5, 17.5) ... itd.

Grupowanie według stażu pracy będzie wtedy polegało na zaliczeniu danego pracownika do jednego z powyższych przedziałów.

Załóżmy, że w bazie danych istnieje tabela

*Pracownicy* (*nr\_prac*, *imię*, *nazwisko*, *staz\_pracy*).

Szukamy liczby pracowników o stażu pracy w wyróżnionych grupach.

Forma zapisu pytania wykorzystującego taki rodzaj grupowania może mieć następującą postać:

```
SELECT COUNT(p.nr_prac), przedzial_staz (p.staz_pracy)
FROM pracownicy p
GROUP BY przedzial_staz (p.staz_pracy) ;
```

gdzie funkcja `przedzial_staz` ustala przedział, do którego należy określona wartość z kolumny `staz_pracy`.

Zauważmy na koniec, że tak prowadzone grupowanie ma cechy podobne do tworzenia histogramów.

### ad 3) Grupowanie wykorzystujące metodę hierarchiczną

Obie omówione dotychczas metody grupowania charakteryzował arbitralny, narzucony z góry wybór grup. W wielu zadaniach związanych z grupowaniem interesujące może być łączenie podobnych wartości w grupy, których postać nie jest z góry ograniczona.

Rozpatrywane podejście jest podobne do problemu wyodrębniania skupisk w algorytmach klastrowania [Klt01].

Dla przedstawienia klasycznego algorytmu klastrowania metodą hierarchiczną [Wjc92] przyjmijmy, że klastrowaniu podlega  $N$  danych  $\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N$ , zaś celem jest uzyskanie  $M$  klastrów. Algorytm będzie wtedy obejmował następujące kroki:

1. Przyjmujemy na wstępie, że każda dana tworzy oddzielny klaster  $K_i = \{\underline{x}_i\}$ ,  $i = 1 \dots N$ ; liczba klastrów  $l := N$
2. Znajdujemy dwa najbliższe sobie klastry  $K_k$  i  $K_j$
3. Łączymy klastry  $K_k$  i  $K_j$  tworząc nowy klaster  $K_k$ ; Usuwamy klaster  $K_j$ ;  $l := l - 1$ ;
4. Jeśli  $l > M$ , przechodzimy do 2.

Zastosowanie przedstawionego algorytmu do baz danych wymaga sprecyzowania pojęcia danych  $\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N$ . W tym celu rozważmy frazę zapewniającą grupowanie w instrukcji SQL w postaci:

GROUP BY  $\langle A_1, A_2, \dots, A_p \rangle$ ,

gdzie  $A_1, A_2, \dots, A_p$  są atrybutami (nazwami kolumn) pochodzącymi z tablicy  $T$ , na której wykonywane jest zapytanie z operacją grupowania.

Zauważmy, że przy przyjętych oznaczeniach zestaw wartości atrybutów  $A_1, A_2, \dots, A_p$ , występujących w jednym wierszu tablicy  $T$  tworzy pojedynczą daną  $\underline{x}$ . Tak więc dane  $\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N$  odpowiadają zestawom wartości atrybutów, według których odbywa się grupowanie, w kolejnych wierszach tablicy  $T$ , przy czym  $N$  oznacza liczbę grupowanych wierszy.

Wykorzystanie przedstawionego algorytmu wymaga przyjęcia założenia, że w wielowymiarowej przestrzeni, którą wyznaczają dane  $\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N$  możliwe jest zdefiniowanie funkcji odległości pomiędzy  $\underline{x}_i$  oraz  $\underline{x}_j$  (i ogólniej – między klastrami).

Odległość ta może być rozumiana na przykład jako [CzgŁsk00]:

- najmniejsza odległość pomiędzy dowolną daną klastra  $A$  i daną klastra  $B$ :

$$d_{\min}(A, B) = \min_{\underline{x}_A \in A, \underline{x}_B \in B} |\underline{x}_A - \underline{x}_B|,$$

- największa odległość pomiędzy dowolną daną klastra  $A$  i daną klastra  $B$ :

$$d_{\max}(A, B) = \max_{\underline{x}_A \in A, \underline{x}_B \in B} |\underline{x}_A - \underline{x}_B|,$$

- średnia arytmetyczna wszystkich odległości pomiędzy wszystkimi danymi klastrów A i B:

$$d_{sr}(A, B) = \frac{1}{card(A)card(B)} \sum_{x_A \in A} \sum_{x_B \in B} |x_A - x_B|,$$

- odległość pomiędzy punktem centralnym (wartością średnią)  $m_A$  klastra A i punktem centralnym (wartością średnią)  $m_B$  klastra B:

$$d_{mean}(A, B) = |m_A - m_B|.$$

Wykorzystanie przedstawionego algorytmu wymaga określenia z góry liczby grup, co w wielu przypadkach może być niewygodne lub nieakceptowalne. Dlatego interesująca może być modyfikacja tego algorytmu, poprzez wprowadzenie innego warunku jego zakończenia, nie związanego z liczbą grup. Takim warunkiem może być ograniczenie na maksymalny rozmiar grupy, rozumiany jako maksymalna odległość między dwiema dowolnymi danymi w grupie.

W zmodyfikowanym w ten sposób algorytmie (w zapisie algorytmu pojęcie „klaster” zastąpiono już pojęciem „grupa”) przyjmujemy, że grupowaniu podlega (jak poprzednio)  $N$  danych  $x_1, x_2, \dots, x_N$ , zaś celem jest uzyskanie grup, których rozmiar nie przekracza zadanej wartości  $maxd$ .

### Algorytm hierarchicznego grupowania

1. Przyjmijmy na wstępie, że każda dana  $x_i$  tworzy oddzielną grupę  $G_i = \{x_i\}$ ,  $i = 1..N$ ,  $N > 1$ .  
Tworzymy zbiór wszystkich grup  $G = \{G_1, G_2, \dots, G_N\}$ .
2. W zbiorze  $G$  znajdujemy dwie najbliższe sobie grupy  $G_k$  oraz  $G_j$ .  
Łączymy wstępnie obie grupy tworząc grupę  $G_k'$ . Wyznaczamy rozmiar  $d$  takiej grupy, tzn. maksymalną odległość między dwiema dowolnymi danymi w grupie.  
Jeśli  $d > maxd$  rezygnujemy z grupy  $G_k'$  i przechodzimy do punktu 4.
3. Zastępujemy grupę  $G_k$  grupą  $G_k'$ . Usuwamy ze zbioru grup  $G$  grupę  $G_j$ .  
Wracamy do punktu 2.
4. Koniec grupowania. – Zbiór  $G$  zawiera utworzone grupy.

Rozważać można również inne warunki zakończenia algorytmu grupowania, bez narzucania liczby grup. Ponieważ w procesie łączenia grup, punkty centralne (środki) grup stopniowo oddalają się od siebie, można zakończyć grupowanie, kiedy odległość dwu najbliższych sobie grup przekroczy zadaną wielkość.

**ad 4) Zaimplementowany algorytm grupowania rozmytego wg metody hierarchicznej**

Dyskutując różne warianty algorytmu grupowania trzeba na koniec wziąć również pod uwagę możliwości zaimplementowania takiego algorytmu w strukturze instrukcji *SELECT* wybranego systemu zarządzania bazą danych PostgreSQL.

Ze względu na ograniczenia narzucane przez ten system udało się zrealizować nieco zmodyfikowaną wersję algorytmu hierarchicznego. Ponieważ okazało się, że w trakcie realizacji instrukcji *SELECT* po rozpoczęciu interpretacji frazy *GROUP BY* dane udostępnione do grupowania zostały już posortowane, wykorzystano ten fakt w zaimplementowanym, przedstawionym dalej algorytmie.

Opis algorytmu zostanie przedstawiony dla jednego wymiaru. Proces realizacji dla kolejnych wymiarów będzie przebiegał analogicznie.

**Zaimplementowany algorytm grupowania:**

Przyjmujemy na wstępie, że grupowaniu podlega uporządkowany rosnąco ciąg danych  $F = (\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N)$ ,  $N > 1$ . Działanie algorytmu zilustrowane zostało schematem blokowym na rysunku 3.11.

Dla uzyskania wyników przewidywanych przez powyższy algorytm opracowana została funkcja *otoczenie*. Jej implementacja szczegółowo zostanie przedstawiona w podrozdziale 4.10, natomiast sposób jej wykorzystania przedstawia poniższy przykład.

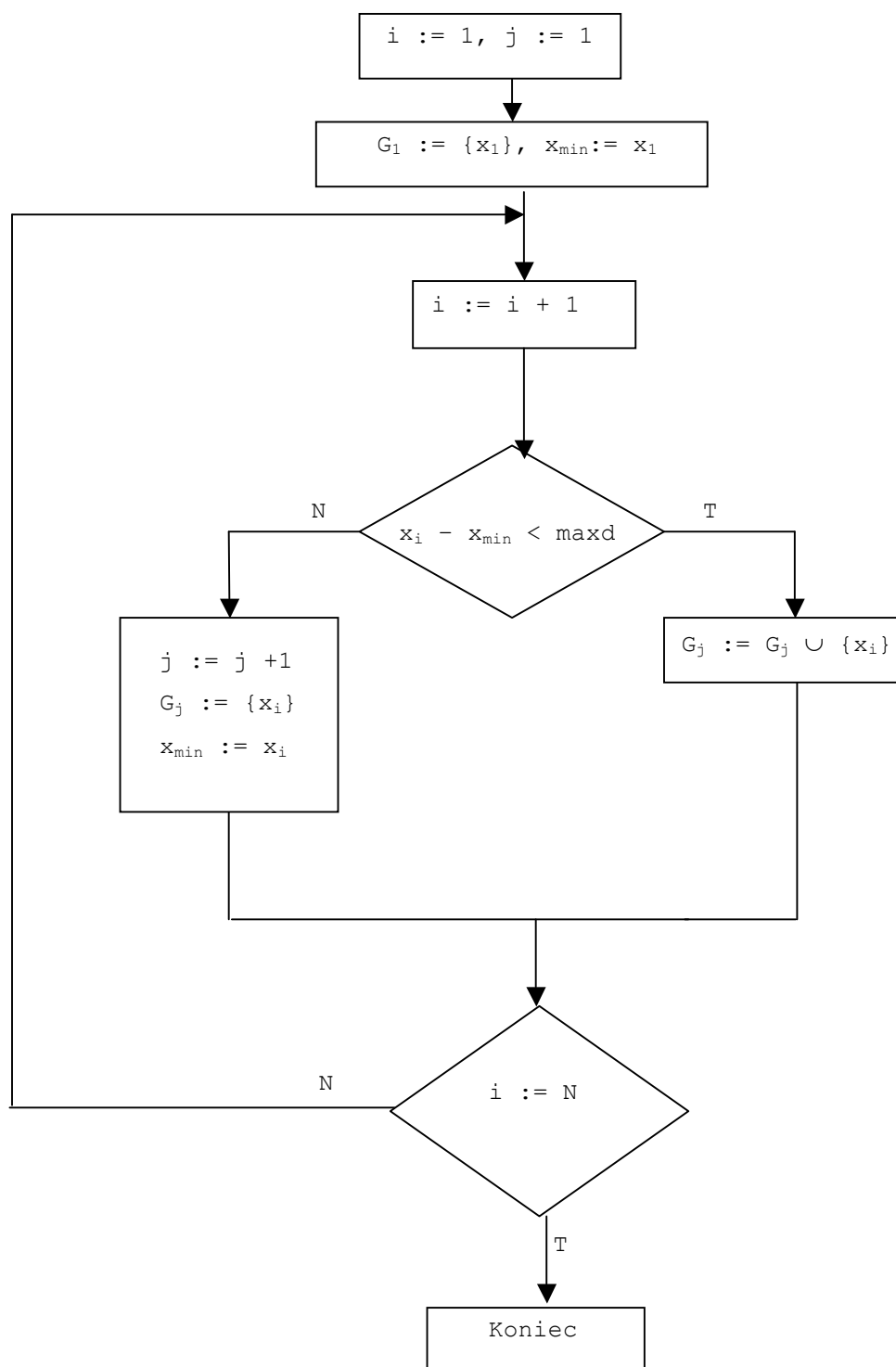
Założmy, że w bazie danych istnieje tabela

*Uczniowie* (*nr\_ucznia*, *imię*, *nazwisko*, *klasa*, *wzrost*, *waga*).

Szukamy liczby uczniów w poszczególnych klasach o podobnej wadze (zakładamy, że waga podobna to waga różniąca się nie więcej niż o 5 kg w ramach grupy).

Forma zapisu pytania realizującego taki rodzaj grupowania będzie miała następującą postać:

```
SELECT klasa, waga, COUNT(nr_ucznia)
FROM uczniowie
GROUP BY klasa, otoczenie(waga, 5);
```



Rys. 3.11 Schemat blokowy zaimplementowanego algorytmu grupowania

### 3.5.2. *Grupowanie rozmytych danych*

Analizując możliwości grupowania rozmytych danych (a ściślej grupowania według kolumn zawierających rozmyte dane) rozpatrzmy najpierw podejście podobne do klasycznego grupowania ostrych (dokładnych) danych. Jako podstawę grupowania przyjmiemy opis rozmytych danych, zakładając, że wartości rozmyte są przedstawione w reprezentacji  $L$ - $R$ .

- Najbardziej rygorystycznym rozwiązaniem jest uwzględnienie w grupowaniu wszystkich parametrów opisujących wartości rozmyte i włączanie takich wartości do jednej grupy tylko wtedy, gdy wszystkie ich parametry są takie same.
- Bardziej elastycznym podejściem jest grupowanie według wartości modalnych (tzn. wartości, dla których funkcja przynależności przyjmuje wartość 1). Należy tu rozróżnić dwa przypadki:
  - grupowane są liczby rozmyte typu  $L$ - $R$ : wszystkie liczby tej postaci o jednakowej wartości modalnej będą stanowiły grupę,
  - grupowane są przedziały rozmyte typu  $L$ - $R$ : wszystkie wartości rozmyte o jednakowym przedziale wartości modalnych będą stanowiły grupę.

Zaimplementowano pierwszą z wyżej wymienionych metod. Sposób jej wykorzystania przedstawia poniższy przykład.

Pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj liczby zakładów składających zapotrzebowanie na podobne ilości papieru w poszczególnych latach”.*

Pytanie to zapisane w języku SQL może mieć następującą postać:

```
SELECT count(distinct nr_zakl)
FROM zapotrzebowanie
GROUP BY papier, rok;
```

Innym podejściem do grupowania rozmytych danych może być wstępne ustalenie zbioru grup, a następnie kwalifikowanie rozpatrywanych danych do określonych grup. Jeśli przyjmiemy rozmyty opis takich grup, to otrzymamy analogię do metod opisanych w podrozdziale 3.5.1. Podejście to przedstawimy w kolejnym podrozdziale.

### 3.5.3. *Rozmyte grupowanie rozmytych danych*

Zakładając, że dane rozmyte zapisane są w reprezentacji liczb  $L$ - $R$  wszystkie rozważane podejścia przedstawione w punkcie 3.5.1 mogą być również stosowane, gdy kolumna względem której wykonuje się grupowanie zawiera rozmyte dane.



- W przypadku pierwszego z omawianych podejść (grupowania względem wartości lingwistycznych), zakładamy, że istnieje zbiór wartości lingwistycznych opisujących dziedzinę kolumny, względem której grupujemy.

Proces kwalifikowania rozmytej danej do najbardziej odpowiadającej jej wartości lingwistycznej bazuje na wyznaczeniu stopni zgodności rozmytej danej ze zdefiniowanymi wartościami lingwistycznymi (proces wyznaczania stopni zgodności między dwiema wartościami rozmytymi omówiono w podrozdziale 3.1.). Następnie analizowanej, rozmytej danej przyporządkowuje się wartość lingwistyczną o największym stopniu zgodności.

Sam proces grupowania, polega, podobnie jak dla dokładnych danych, na scalaniu w jedną grupę danych o tych samych wartościach lingwistycznych.

- Proces realizacji drugiej z omawianych metod (grupowania według arbitralnego podziału zadanego przedziału wartości) przebiega w podobny sposób.
- W przedstawionych powyżej metodach znany i narzucony jest podział na grupy. Algorytmy, w których nie ma ścisłego, wyodrębnionego podziału na grupy można również stosować w grupowaniu rozmytych danych, o ile zostanie zdefiniowana funkcja odległości między dwiema wartościami rozmytymi. Wtedy proces wyznaczania grup można realizować analogicznie do metod 3 i 4 przedstawionych w podrozdziale 3.5.1.

Na koniec rozważmy specyficzny, możliwy do zrealizowania przypadek grupowania, nie mieszczący się w wyróżnionych kategoriach.

Wartości rozmyte mogą się również pojawić we frazie *GROUP BY* w pośredniej formie, na przykład, gdy chcemy grupować względem stopnia zgodności. Problem ten zilustrujemy następującym przykładem:

Do bazy danych *ZAKŁADY* jest kierowane przykładowe pytanie:

*„Wyszukaj liczby zakładów o jednakowych stopniach zgodności z warunkiem pytania: liczba zatrudnionych pracowników około 25.”*

Postać tego pytania zapisana w formie funkcyjnej w języku SQL może być wyrażona następująco:

```
SELECT count(nr_zakl), DG(liczba_prac jest okolo 25)
FROM zaklady
GROUP BY DG(liczba_prac jest okolo 25);
```

Postać tego pytania w formie operatorowej w języku SQL jest następująca:

```
SELECT count(nr_zakl), liczba_prac jest okolo 25
FROM zaklady
GROUP BY liczba_prac jest okolo 25;
```

W przedstawionym przykładzie grupowanie odbywa się względem tych samych wartości stopni zgodności dla warunku (`liczba_prac jest okolo 25`), a nie jak w poprzednich przypadkach względem wartości kolumny. Grupę w tym przypadku stanowią te wiersze, dla których warunek podany we frazie *GROUP BY* przyjmuje tę samą wartość stopnia zgodności.

Przedstawione propozycje nie wyczerpują możliwych do zastosowania metod grupowania. Na przykład nawiązując do wskazanej analogii między grupowaniem i klastrowaniem należy podkreślić, że w literaturze [Klt01] rozpatrywanych jest wiele metod klastrowania, a więc podobnie można by rozważać wiele metod grupowania danych.

### 3.6. Wartości rozmyte w pytaniach zagnieżdżonych

W poprzednich punktach tego rozdziału przeprowadzono analizę procesu interpretacji niezagnieżdżonych zapytań SQL-owych zawierających wartości rozmyte. W niniejszym rozdziale przeprowadzimy taką analizę dla pytań zagnieżdżonych.

Wartości rozmyte w zapytaniach zagnieżdżonych mogą pojawić się w kilku miejscach:

- we wszystkich frazach podzapytania wewnętrznego,
- we wszystkich frazach podzapytania zewnętrznego,
- lub w warunku wiążącym podzapytanie zewnętrzne z wewnętrznym.

W pytaniach zagnieżdżonych proces wyszukiwania wierszy spełniających rozmyte warunki filtrujące zarówno we frazie *WHERE* jak i we frazie *HAVING* jest realizowany podobnie jak w pytaniach niezagnieżdżonych. Są jednak pewne różnice:

- pytania zagnieżdżone przyjmują często bardzo skomplikowaną formę,
- pojawia się także problem z ulokowaniem warunku na stopień zgodności w przypadku rozmytego warunku wiążącego dwa podzapytania.

Z tych powodów celowe jest szczegółowe rozpatrzenie możliwości umiejscowienia wartości rozmytych w pytaniach zagnieżdżonych.

Ze względu na występowanie wartości rozmytych w warunkach wiążących pytań zagnieżdżonych, wyodrębniono następujący podział:

1. porównanie wartości dokładnej podzapytania zewnętrznego z wartością rozmytą (lub zbiorem takich wartości) zwracaną w wyniku wykonania pytania wewnętrznego,
2. porównanie wartości rozmytej podzapytania zewnętrznego z wartością dokładną (lub zbiorem takich wartości) zwracaną w wyniku wykonania pytania wewnętrznego,
3. porównanie wartości rozmytej podzapytania zewnętrznego z wartością rozmytą (lub zbiorem takich wartości) zwracaną w wyniku wykonania pytania wewnętrznego,

4. umiejscowienie wartości rozmytych w warunku korelacji wiążącym skorelowane pytania zagnieżdżone.

Istotnym rozwiązaniem w pracy problemem jest sposób zapisu wartości progowej w warunku dotyczącym stopnia zgodności dla rozmytego warunku wiążącego dwa podzapytania.

Także w tym przypadku można zaproponować trzy uprzednio wprowadzone formy takiego zapisu. Formalny zapis warunku wiążącego dwa podzapytania, w tych formach, jest zdefiniowany następująco:

– Forma dołączana:

```
...
<forma_dołączana_w_w> ::= <wyrażenie_pyt_zewn><op>(<pytanie_wewn>)
                        AND DG<warunek_progowy>
    <op> ::= <operator_rel>| <operator_rel><operator_zb>| IN| NOT IN
<warunek_progowy> ::= <operator_rel><wartość_progowa>
<operator_rel> ::= = | < | > | <= | >= | <>
<wartość_progowa> ::= <liczba dziesiętna z przedziału [0, 1]>
<operator_zb> ::= ANY| ALL
...
```

– Forma funkcyjna:

```
...
<forma_funkcyjna_w_w> ::=
                        DG(<wyrażenie_pyt_zewn><op>(<pytanie_wewn>))
                        <warunek_progowy>
    <op> ::= <operator_rel>| <operator_rel><operator_zb>| IN| NOT IN
<warunek_progowy> ::= <operator_rel><wartość_progowa>
<operator_rel> ::= = | < | > | <= | >= | <>
<wartość_progowa> ::= <liczba dziesiętna z przedziału [0, 1]>
<operator_zb> ::= ANY| ALL
...
```

– Forma operatorowa:

```
...
<forma_operatorowa_w_w> ::= <wyrażenie_rozmyte_rozsz><op>(<pytanie_wewn>)
                        <wyrażenie_pyt_zewn><op_r>(<pytanie_wewn>)<warunek_progowy> |
                        <wyrażenie_pyt_zewn><op>(<pytanie_wewn_rozsz_r>) |
<pytanie_wewn_rozsz_r> ::=
SELECT <wyrażenie><op_rozsz><wartość_progowa><kolejne_frazy_pytania_wewn>
<wyrażenie_rozmyta_rozsz> ::= <wartość_rozmyta><op_rozsz><wartość_progowa>
<op_rozsz> ::= *= | *> | *>= | *< | *<= | *<
<op> ::= <op_rel>| <op_rel><op_zb>| IN| NOT IN
<op_r> ::= jest | mniej więcej
```

```

<warunek_progowy> ::= <op_rel><wartość_progowa>
<op_rel> ::= = | < | > | <= | >= | <>
<wartość_progowa> ::= <liczba dziesiętna z przedziału [0, 1]>
<operator_zb> ::= ANY | ALL
...

```

gdzie:

<wyrażenie\_pyt\_zewn> oznacza wyrażenie występujące jako lewa strona warunku filtrującego w pytaniu zewnętrznym. Zazwyczaj będzie to nazwa kolumny,

<pytanie\_wewn> oznacza zapis pytania wewnętrznego.

Celowość rozpatrywania trzech różnych form zapisu uzasadnić można następująco:

1. Forma dołączana jest formą najprostszą, jednak jej zastosowanie jest ograniczone do prostych pytań niezagnieżdżonych.
2. Forma funkcyjna jest bardzo elastyczna i daje przejrzysty zapis nawet dla pytań zagnieżdżonych. Jednak ograniczenia implementacyjne powodują, że dla pewnych specyficznych pytań zagnieżdżonych (przedstawionych w dalszych przykładach) jest ona niewystarczająca.
3. Forma operatorowa jest najbardziej uniwersalna, daje bardzo zwarty zapis, choć nie zawsze tak czytelny jak dwie pierwsze formy.

Sposób zapisu warunków z wartością progową (przy użyciu formy funkcyjnej i operatorowej) zostanie przedyskutowany w analizowanych dalej przykładach.

Czasami dla skrócenia zapisu, zamiast *forma operatorowa zapisu warunku na stopień zgodności* będziemy używać krótszego sformułowania *pytanie w formie operatorowej*, podobnie dla formy funkcyjnej.

Dla ułatwienia analizy zaproponowanych pytań przypomnijmy fragment bazy danych *ZAKŁADY*:

```

Instytuty (nr_inst, nazwa)
Zakłady (nr_zakl, nazwa, liczba_prac, liczba_pokoi, nr_inst)
Pracownicy (nr_prac, nr_zakl, nazwisko, wiek, plec, staz_pracy)
Zapotrzebowanie (nr_zakl, rok, papier, toner, płytki CD)
Zużycie (nr_zakl, rok, papier, toner, płytki CD)

```

Stosując wyodrębniony podział ze względu na występowanie wartości rozmytych w warunkach wiążących w pytaniach zagnieżdżonych, rozpatrzmy następujące przypadki:

### 3.6.1. Pytania nieskorelowane

- a) Porównanie wartości dokładnej pytania zewnętrznego z wartością dokładną (lub zbiorem wartości dokładnych) zwracaną przez pytanie wewnętrzne

W tym przypadku warunek wiążący podzapytania ma klasyczną postać, jednakże dopuszczamy występowanie wartości rozmytych zarówno „wewnątrz” pytania zewnętrznego jak i wewnętrznego.

Realizacja tego typu pytań jest taka sama jak pytań w klasycznym języku SQL. Po wyszukaniu wierszy spełniających warunki filtrujące występujące we frazie *WHERE* i *HAVING* w pytaniu zewnętrznym następuje, dla każdego analizowanego wiersza, porównanie wartości kolumny z wartością (lub zbiorem wartości) otrzymaną w wyniku wykonania pytania wewnętrznego. Wiersze spełniające warunki filtrujące pytania zewnętrznego i warunek łączenia tworzą zbiór wynikowy pytania.

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj te zakłady, które zatrudniają około 30 pracowników i zajmują mniej pokoi niż Zakład Teorii Informatyki. Wyświetl te wiersze, których stopień zgodności wynosi co najmniej 0.55.”*

Postać tego pytania zapisana w języku SQL z formą funkcyjną zapisu warunku na stopień zgodności może być wyrażona następująco:

```
SELECT nr_zakl, nazwa
FROM zaklady
WHERE liczba_pokoi < (SELECT liczba_pokoi FROM zaklady
                      WHERE nazwa = 'Zakład Teorii Informatyki')
AND DG(liczba_prac jest okolo 30) >= 0.55;
```

Postać tego pytania zapisana w języku SQL z formą operatorową zapisu warunku na stopień zgodności może być wyrażona następująco:

```
SELECT nr_zakl, nazwa
FROM zaklady
WHERE liczba_pokoi < (SELECT liczba_pokoi FROM zaklady
                      WHERE nazwa = 'Zakład Teorii Informatyki')
AND (liczba_prac jest okolo 30) >= 0.55;
```

W podzapytaniu zewnętrznym prezentowanego pytania występuje warunek rozmyty. Po wyznaczeniu stopnia zgodności wartości kolumny analizowanego wiersza z warunkiem pytania oraz porównania go z wartością progową, proces wyszukiwania wierszy spełniających warunki zadane w pytaniu zagnieżdżonym

realizowany jest w ten sam sposób jak w klasycznych pytaniach zagnieżdżonych języka SQL.

W pytaniach, w których w jawny sposób porównuje się stopnie zgodności, wartości rozmyte mogą pośrednio występować w warunku wiążącym podzapytania, na przykład w warunku wyznaczającym stopień zgodności. W tych przypadkach także porównujemy ze sobą wartości dokładne (wartości jakie przyjmują stopnie zgodności).

Pytanie tego typu może brzmieć następująco:

*„Wyszukaj zakłady, w których zapotrzebowanie na rok 2003 na płytki CD oszacowano na około 100 sztuk. W odpowiedzi powinien znaleźć się wiersz najlepiej spełniający kryteria pytania.”*

Pytanie to zapisane w języku SQL w formie funkcyjnej może mieć następującą postać:

```
SELECT nr_zakl, nazwa, DG(plytki_CD jest okolo 100) as stopien
FROM zaklady z JOIN zapotrzebowanie zp
  ON z.nr_zakl = zp.nr_zakl
WHERE zp.rok = '2003' AND DG(plytki_CD jest okolo 100) >= ALL
                        (SELECT DG(plytki_CD jest okolo 100)
                        FROM zapotrzebowanie zp
                        WHERE zp.rok = '2003');
```

Forma operatorowa tego pytania zapisana w języku SQL jest następująca:

```
SELECT nr_zakl, nazwa, (plytki_CD jest okolo 100) as stopien
FROM zaklady z JOIN zapotrzebowanie zp
  ON z.nr_zakl = zp.nr_zakl
WHERE z.rok = '2003' AND (plytki_CD jest okolo 100) >= ALL
                        (SELECT plytki_CD jest okolo 100
                        FROM zapotrzebowanie z
                        WHERE z.rok = '2003');
```

W tym pytaniu dla każdego analizowanego wiersza pytania zewnętrznego, realizowane jest porównanie wartości stopnia zgodności wyznaczonego w wyniku porównania wartości *plytki\_CD* z wartością rozmytą *okolo 100* ze zbiorem stopni zwróconych w wyniku wykonania pytania wewnętrznego.

- b) Porównanie wartości dokładnej pytania zewnętrznego z wartością rozmytą (lub zbiorem wartości rozmytych) zwracaną przez pytanie wewnętrzne

W tym przypadku głównym problemem jaki należało rozwiązać było:

*Jak rozbudować warunek wiążący pytanie zewnętrzne z wewnętrznym o zapis warunku na stopień zgodności?*

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, które w roku 2002 zużyły mniej więcej tyle papieru, ile wynosi średnie oszacowane zapotrzebowanie we wszystkich zakładach. W odpowiedzi powinny znaleźć się wiersze o stopniu zgodności co najmniej 0.45.”*

Postać tego pytania zapisana w języku SQL w formie funkcyjnej może być następująca:

```
SELECT nr_zakl
FROM zuzycie
WHERE rok = '2002' AND DG(papier mniej_wiecej (SELECT AVG(papier)
FROM zapotrzebowanie
WHERE rok = '2002')) >= 0.45;
```

Funkcja **DG** wyznacza wartość stopnia zgodności wartości kolumny analizowanego wiersza z pytania zewnętrznego z wartością rozmytą zwracaną w wyniku wykonania pytania wewnętrznego.

W pytaniu tym występuje również rozmyta funkcja agregująca *avg()*, której argumentem są wartości rozmyte kolumny *papier*.

Pytanie to zapisane w języku SQL w formie operatorowej ma następującą postać:

```
SELECT nr_zakl
FROM zuzycie
WHERE rok = '2002' AND (papier mniej_wiecej (SELECT AVG(papier)
FROM zapotrzebowanie z
WHERE rok = '2002')) >= 0.45;
```

W przypadku, gdy w pytaniu konieczne okaże się użycie jednego z operatorów:  $<$ ,  $<=$ ,  $>$ ,  $>=$  w warunku wiążącym dwa podzapytania, należy zastosować operacje zgodne z arytmetyką liczb rozmytych, tworząc nowe operatory, działające na liczbach rozmytych.

Jednym ze sposobów określenia, czy wartość dokładna jest w relacji mniejszości lub w relacji większości z wartością rozmytą, jest wykonanie wcześniej rozmycia wartości dokładnej i dopiero wtedy porównanie ze sobą obu wartości. Zaproponowane metody określania, która z wartości rozmytych jest większa czy mniejsza przedstawiono w rozdziale 4.

Należy także zauważyć, że wśród różnych operatorów porównania występujących w warunkach wiążących pytanie zewnętrzne z wewnętrznym postać rozmytą może przyjmować tylko operator równości wyrażony na przykład w postaci operatorów:  $\sim=$ , *mniej więcej*, *w przybliżeniu* itd.

- c) Porównanie wartości kolumny rozmytej z pytania zewnętrznego z wartością dokładną (lub zbiorem wartości dokładnych) zwracaną przez pytanie wewnętrzne

Proces realizacji takiego pytania przebiega podobnie jak w przypadku przedstawionym w podpunkcie b.

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, które w roku 2002 złożyły zapotrzebowanie na liczbę ryz papieru mniej więcej równą średniemu zużyciu we wszystkich jednostkach. W odpowiedzi powinny się znaleźć tylko wiersze ze stopniem zgodności co najmniej 0.7.”*

Pytanie to zapisane w języku SQL w postaci zagnieżdżonej w formie funkcyjnej może być wyrażone następująco:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE rok = '2002' AND DG(papier mniej więcej (SELECT AVG(papier)
                                                FROM zuzycie
                                                WHERE rok = '2002')) >= 0.7;
```

Pytanie to zapisane w języku SQL w postaci zagnieżdżonej w formie operatorowej ma następującą postać:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE rok = '2002' AND (papier *>= 0.7) mniej więcej
                        (SELECT AVG(papier)
                         FROM zuzycie
                         WHERE rok = '2002');
```

lub wykorzystując nieco inną bardziej intuicyjną w zapisie formę:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE rok = '2002' AND (papier mniej więcej
                        (SELECT AVG(papier)
                         FROM zuzycie
                         WHERE rok = '2002')) >= 0.7;
```

- d) Porównanie wartości kolumny rozmytej z pytania zewnętrznego z wartością rozmytą (lub zbiorem wartości) zwracaną przez pytanie wewnętrzne

Proces porównania ze sobą dwóch wartości rozmytych został omówiony w rozdziale 3.1 pracy. W tym przypadku proces wyznaczania stopnia zgodności i porównywania go z wartością progową realizowany jest tak samo, jak w podpunkcie b lub c.



Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, które złożyły w roku 2003 zapotrzebowanie na mniej więcej taką liczbę płytek CD jak zakład Oprogramowanie. Wyświetl te wiersze, których stopień zgodności wynosi przynajmniej 0.45.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL w formie funkcyjnej może być wyrażona następująco:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE rok = '2003' AND DG(plytki_CD mniej_wiecej
                        (SELECT plytki_CD
                         FROM zapotrzebowanie zp JOIN zaklady z
                         ON zp.nr_zakl = z.nr_zakl
                         WHERE z.nazwa = 'Zakład Oprogramowania'
                         AND zp.rok = '2003')) >= 0.45;
```

Zagnieżdżona postać tego pytania zapisana w języku SQL w formie operatorowej może być wyrażona następująco:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE rok = '2003' AND (plytki_CD *>= 0.45) mniej_wiecej
                        (SELECT plytki_CD
                         FROM zapotrzebowanie zp JOIN zaklady z
                         ON z.nr_zakl = zp.nr_zakl
                         WHERE nazwa = 'Zakład Oprogramowania' AND z.rok = '2003');
```

lub wykorzystując nieco inną bardziej intuicyjną w zapisie formę:

```
SELECT nr_zakl
FROM zapotrzebowanie
WHERE rok = '2003' AND (plytki_CD mniej_wiecej
                        (SELECT plytki_CD
                         FROM zapotrzebowanie zp JOIN zaklady z
                         ON zp.nr_zakl = z.nr_zakl
                         WHERE nazwa = 'Zakład Oprogramowania' AND z.rok = '2003')) >= 0.45;
```

### 3.6.2. Pytania skorelowane

W pytaniach skorelowanych wartość rozmyta oprócz warunku wiążącego podzapytanie zewnętrzne z wewnętrznym (przypadki te rozważono w poprzednim punkcie tego rozdziału) może jeszcze pojawić się w warunku korelacji między kolumną aktualnie analizowanego wiersza z pytania zewnętrznego a wartością kolumny analizowanej w pytaniu wewnętrznym.

Taka struktura rozmytych zapytań skorelowanych wymagała przeprowadzenia analizy występowania wartości rozmytych w warunku korelacji oraz właściwej ich interpretacji. W wyniku przeprowadzonej analizy w pracy uznano, że rozwiązanie problemu rozmytej korelacji sprowadzi się do zastosowania metod przedstawionych w rozdziałach 3.1 i 3.3 (rozmyta korelacja może być traktowana jak rozmyty warunek filtrujący we frazie *WHERE*), a inna struktura pytań wymusza tylko odpowiednią ich interpretację.

Dla rozpatrywanego warunku korelacji mogą się więc w nim pojawić następujące rodzaje połączeń między wartościami dokładnymi a rozmytymi:

- a) korelacja wartości dokładnej z pytania zewnętrznego z wartością dokładną pytania wewnętrznego,
- b) korelacja wartości dokładnej z pytania zewnętrznego z wartością dokładną pytania wewnętrznego (korelacja względem stopnia zgodności),
- c) korelacja wartości dokładnej z pytania zewnętrznego z wartością rozmytą pytania wewnętrznego lub na odwrót,
- d) korelacja wartości rozmytej z pytania zewnętrznego z wartością rozmytą pytania wewnętrznego.

ad a) Korelacja wartości dokładnej z pytania zewnętrznego z wartością dokładną pytania wewnętrznego, w pytaniu tym występuje dodatkowo rozmyty warunek wiążący dwa podzapytania.

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, które zużyły w roku 2002 mniej więcej tyle papieru, ile oszacowały w zapotrzebowaniu. Wyświetl te wiersze, których stopień zgodności przekracza 0.8.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL w formie operatorowej może być wyrażona następująco:

```
SELECT nr_zakl
FROM zuzycie zu
WHERE rok = '2002' AND (papier mniej_wiecej
                        (SELECT papier
                         FROM zapotrzebowanie z
                         WHERE rok = '2002'
                         AND z.nr_zakl = zu.nr_zakl)) > 0.8;
```

W zapisie tym warunek korelacji został wyróżniony pogrubioną czcionką.

ad b) Korelacja wartości dokładnej z pytania zewnętrznego z wartością dokładną pytania wewnętrznego - korelacja względem stopni zgodności

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj te zakłady, których zapotrzebowanie na papier w roku 2003 w tym samym stopniu bliskie jest liczbie około 45.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL w formie operatorowej może być wyrażona następująco:

```
SELECT DISTINCT z1.nr_zakl
FROM zapotrzebowanie z1
WHERE z1.rok = '2003' AND EXISTS
      (SELECT * FROM zapotrzebowanie z2
       WHERE z1.nr_zakl <> z2.nr_zakl
        AND z2.rok = '2003' AND
          (z1.papier jest okolo 45) = (z1.papier jest okolo 45));
```

W pytaniu tym w warunku korelacji pośrednio występują wartości rozmyte, gdyż sam warunek operuje już na dokładnych wartościach wyznaczonych stopni zgodności.

ad c) Korelacja wartości dokładnej z pytania zewnętrznego z wartością rozmytą pytania wewnętrznego lub na odwrót.

Analizowane w podpunkcie a pytanie może zostać zapisane w innej postaci:

*„Wyszukaj zakłady, które zużyły w roku 2002 mniej więcej tyle papieru, ile oszacowały w zapotrzebowaniu. W odpowiedzi powinny pojawić się te wiersze, których stopień zgodności wynosi co najmniej 0.6.”*

Zapis tego pytania w formie funkcyjnej w języku SQL jest następujący:

```
SELECT nr_zakl
FROM zuzycie zu
WHERE rok = '2002' AND nr_zakl IN
      (SELECT nr_zakl
       FROM zapotrzebowanie z
       WHERE rok = '2002'
        AND DG(z.papier jest zu.papier) >= 0.6);
```

Zapis tego pytania w formie operatorowej w języku SQL może być wyrażony w następujący sposób:

```
SELECT nr_zakl
FROM zuzycie zu
WHERE rok = '2002' AND nr_zakl IN
      (SELECT nr_zakl
       FROM zapotrzebowanie z
       WHERE rok = '2002'
        AND (z.papier jest zu.papier) >= 0.6);
```

ad d) Korelacja wartości rozmytej z pytania zewnętrznego z wartością rozmytą pytania wewnętrznego

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady, dla których istnieje chociaż jeden inny zakład, który złożył zapotrzebowanie w roku 2003 na podobną ilość tonerów. W odpowiedzi powinny znaleźć się wiersze ze stopniem zgodności co najmniej 0.9.”*

Zagnieżdżone pytanie zapisane w języku SQL w formie funkcyjnej może mieć następującą postać:

```
SELECT nr_zakl
FROM zapotrzebowanie z1
WHERE z1.rok = '2003' AND EXISTS
      (SELECT * FROM zapotrzebowanie z2
       WHERE z2.nr_zakl <> z1.nr_zakl AND
             z2.rok = '2003' AND
             DG(z2.toner jest z1.toner) >= 0.9);
```

Zapis tego pytania w języku SQL w formie operatorowej jest następujący:

```
SELECT nr_zakl
FROM zapotrzebowanie z1
WHERE z1.rok = '2003' AND EXISTS
      (SELECT * FROM zapotrzebowanie z2
       WHERE z2.nr_zakl <> z1.nr_zakl AND
             z2.rok = '2003' AND
             (z2.papier jest z1.papier) >= 0.9);
```

Wszystkie te trzy przypadki, choć w nieco innym aspekcie, zostały rozpatrzone w rozdziale 3.1 i 3.3 podczas analizy rozmytych warunków filtrujących występujących we frazie *WHERE*. Proces porównywania wartości ze sobą, zarówno w przypadku warunku filtrującego jak i korelacji realizowany jest w ten sam sposób.

### 3.7. Zależne kontekstowo interpretacje wyrazów rozmytych

W poprzednich rozdziałach omówione zostały sytuacje, w których każdą wartość rozmytą charakteryzowała dokładnie jedna konkretna funkcja przynależności.

Niektóre przypadki mogą być jednak bardziej złożone i wtedy takie założenie może być zbyt dużym uproszczeniem.

Założmy, że w pewnej bazie danych istnieją tabele:

```
Oceny (nr_studenta, nr_przedmiotu, oceny)
Studenci (nr_studenta, nr_kierunku, nazwisko),
```

Rozpatrzmy następujące zapytanie:

*„Wyszukać studentów na poszczególnych kierunkach, którzy uzyskali wysoką średnią”*

Pytanie to w języku SQL może mieć następującą postać:

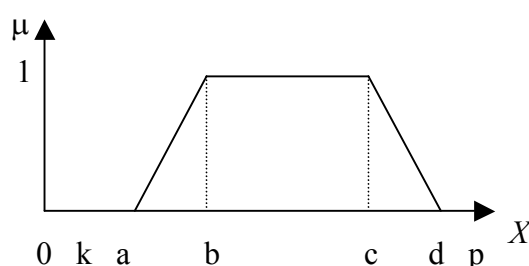
```

SELECT nazwisko
FROM studenci s JOIN oceny o
  ON s.nr_studenta = o.nr_studenta
GROUP BY nr_kierunku, nazwisko
HAVING AVG(oceny) jest wysoka;

```

Realizacja pytania wymaga wcześniejszego zdefiniowania zbioru rozmytego *wysoka średnia* a więc określenia odpowiedniej funkcji przynależności.

Zakładanie istnienia tylko jednej funkcji może być nadmiernym uproszczeniem, gdyż na różnych kierunkach definicja *wysokiej średniej* może być inna.



Rys. 3.12 Trapezoidalna parametryczna funkcja przynależności

Wartość rozmyta *wysoka średnia* może tu być reprezentowana przez funkcję przynależności w postaci funkcji trapezowej (rysunek 3.12), której dziedziną jest przedział  $[k, p]$ , a trapez zdefiniowany jest przez punkty  $(x=a, y=0)$ ,  $(x=b, y=1)$ ,  $(x=c, y=1)$ ,  $(x=d, y=0)$  [YuMng98]. Punkty  $a, b, c, d$  mogą być wyznaczane przez poszczególne dziekanaty.

Postać tej funkcji dla każdego kierunku określona będzie innym zestawem parametrów.

Wróćmy do pytania postawionego na początku rozdziału, rozszerzonego następująco:

„Wyszukaj studentów na poszczególnych kierunkach, którzy uzyskali wysoką średnią.  
Wyświetl wiersze o stopniu zgodności większym niż 0.8.”

Założmy, że poprzednią bazę danych rozszerzymy o tabelę *średnie*:

```

Srednie (nr_kierunku, wysoka_srednia),
Oceny (nr_studenta, nr_przedmiotu, oceny)
Studenci (nr_studenta, nr_kierunku, nazwisko),

```

gdzie w kolumnie *wysoka\_srednia* tabeli *Srednie* przechowywane są parametry funkcji przynależności do zbiorów *wysokie\_srednie* dla poszczególnych kierunków. W pracy proponuje się następujące rozwiązanie tego typu pytań:

```

SELECT nazwisko
FROM studenci st JOIN oceny o ON st.nr_studenta = o.nr_studenta
GROUP BY st.nr_studenta, nazwisko
HAVING AVG (o.oceny) = (SELECT (wysoka_srednia)*>0.8
                        FROM Srednie s
                        WHERE s.nr_kierunku = st.nr_kierunku);

```

W powyższym pytaniu parametryczna funkcja przynależności będzie transponowana w bieżącą funkcję przynależności, opartą na warunkach wyspecyfikowanych w pytaniu. Właściwe parametry funkcji przynależności wybierane są według numeru kierunku.

Jak wynika z przytoczonego przykładu, może się zdarzyć, że wartości lingwistyczne używane w pytaniach (np.: dobry, duży, wysoki itp.) będą różnie interpretowane w kontekście danych występujących w bazie danych. Wtedy konieczne będzie rozszerzenie bazy danych o tabelę, która będzie opisywała taką zależność. Pytanie w języku SQL musi wtedy jawnie korzystać z tej tabeli.

Na przykład w pytaniu wyszukującym wysokich studentów na uczelni, ważnym atrybutem rozróżniającym funkcję przynależności będzie atrybut *pleć*, gdyż zwykle mężczyźni są wyżsi od kobiet i parametry *wysokiego wzrostu* dla mężczyzn są inne niż dla kobiet.

Rozszerzmy tabelę *studenci* o kolumny *plec* i *wzrost*:

```
Studenci (nr_stud, nr_kier, nazwisko, plec, wzrost),
```

i bazę danych o tabelę *wzrost*, o strukturze:

```
Wzrost (plec, wysoki_wzrost),
```

W pracy proponuje się następujące rozwiązanie tego typu pytań w formie operatorowej, zakładając, że w odpowiedzi chcemy nazwiska studentów, dla których stopień zgodności z warunkiem pytania przekracza 0.8.

```
SELECT nazwisko
FROM studenci s JOIN Wzrost w ON s.plec = w.plec
WHERE (s.wzrost jest w.wysoki_wzrost) > 0.8
```

### 3.8. Proces przekształcania zagnieżdżonych rozmytych zapytań SQL w niezagnieżdżone rozmyte zapytania SQL

W rozdziale 2.1.3 przeprowadzono dyskusję na temat przekształcania zagnieżdżonych pytań SQL w niezagnieżdżone. Zazwyczaj złożoność czasowa pytań niezagnieżdżonych jest mniejsza. Dlatego celowym wydaje się poszukiwanie podobnych rozwiązań dla pytań rozmytych.

Rozmyte pytania zagnieżdżone formułowane do bazy danych omówiono w rozdziale 3.6, w tym rozdziale zostanie przeprowadzona analiza sposobu przekształcania pewnych charakterystycznych klas rozmytych zapytań zagnieżdżonych w ich niezagnieżdżone odpowiedniki.

Rozpatrzmy klasy pytań biorąc pod uwagę warunek łączący podzapytania oraz korelację dwóch podpytań:

## 1. Warunek łączący podzapytania:

- a) warunek łączący dwa podzapytania typu: **wartość dokładna z wartością dokładną (względem stopni zgodności)**:

Rozważmy pytanie analizowane w rozdziale 3.6 w punkcie 1a, o treści „Wyszukaj zakłady, w których w roku bieżącym złożono zapotrzebowanie na około 100 płytek CD. W odpowiedzi powinien się znaleźć zakład najlepiej spełniający kryteria pytania.”

Postać tego pytania zapisana w języku SQL w formie funkcyjnej może być wyrażona następująco:

```
SELECT nr_zakl, nazwa, (plytki_CD jest okolo 100) as stopien
FROM zaklady z JOIN zapotrzebowanie zp
ON z.nr_zakl = zp.nr_zakl
WHERE zp.rok = '2003' AND DG(plytki_CD jest okolo 100) >= ALL
      (SELECT DG(plytki_CD jest okolo 100)
       FROM zapotrzebowanie zp
       WHERE zp.rok = '2003');
```

Postać tego pytania zapisana w języku SQL w formie operatorowej jest następująca:

```
SELECT nr_zakl, nazwa, (plytki_CD jest okolo 100) as stopien
FROM zaklady z JOIN zapotrzebowanie zp
ON z.nr_zakl = zp.nr_zakl
WHERE zp.rok = '2003' AND (plytki_CD jest okolo 100) >= ALL
      (SELECT plytki_CD jest okolo 100
       FROM zapotrzebowanie zp
       WHERE zp.rok = '2003');
```

W pytaniu tym interesujący jest fakt, że porównywane wartości rzeczywiste otrzymane są w wyniku wyznaczenia wartości stopnia zgodności. W rezultacie porównujemy dwa pytania względem stopni zgodności. Pytanie to może zostać zapisane również w inny sposób, wykorzystując funkcję agregującą MAX, która wyznaczy wartość maksymalną spośród stopni zgodności należących do zbioru wynikowego podzapytania wewnętrznego:

```
SELECT nr_zakl, nazwa, (plytki_CD jest okolo 100) as stopien
FROM zaklady z JOIN zapotrzebowanie zp
ON z.nr_zakl = zp.nr_zakl
WHERE zp.rok = '2003' AND (plytki_CD jest okolo 100) =
      (SELECT Max(plytki_CD jest okolo 100)
       FROM zapotrzebowanie zp
       WHERE zp.rok = '2003');
```

W przypadku, gdy w warunku łączenia pytania zewnętrznego z wewnętrznym po stronie pytania wewnętrznego występuje funkcja agregująca (na przykład

*MAX*), analizowane pytanie zagnieżdżone da się przekształcić w niezagnieżdżone, ale w przynajmniej dwóch krokach.

Niezagnieżdżona postać analizowanego pytania wymaga następujących kroków:

1. utworzenie dodatkowej tabeli przechowującej wartość maksymalną stopnia zgodności z warunkiem: **plytki\_CD jest okolo 100**

```
SELECT Max(plytki_CD jest okolo 100) as max_st
INTO max_stopien
FROM zapotrzebowanie z
WHERE z.rok = '2003';
```

2. realizacja pytania podstawowego, stosując utworzoną tabelę **max\_stopien**

```
SELECT z.nr_zakl, z.nazwa,
      (plytki_CD jest okolo 100) as stopien
FROM zaklady z JOIN zapotrzebowanie zp
      ON z.nr_zakl = zp.nr_zakl, max_stopien ms
WHERE zp.rok = '2003'
      AND (plytki_CD jest okolo 100) = ms.max_st;
```

Pytanie to może zostać również zostać zapisane następująco:

```
SELECT nr_zakl, nazwa, (plytki_CD jest okolo 100) as stopien
FROM zaklady z JOIN zapotrzebowanie zp
      ON z.nr_zakl = zp.nr_zakl,
      (SELECT Max(plytki_CD jest okolo 100) as max_st
      FROM zapotrzebowanie zp WHERE zp.rok = '2003') Max_stopien
WHERE (plytki_CD jest okolo 100) = Max_stopien.max_st
      AND zp.rok = '2003';
```

- b) warunek łączący dwa podzapytania typu: **wartość dokładna z rozmytą**:

Rozpatrzmy następujące pytanie:

*„Podaj zakłady, w których zeszłoroczna zmiana (2002 rok) w zużyciu ryz papieru była mniej więcej taka jak przynajmniej jeden z przyrostów w szacowanym zapotrzebowaniu z ostatnich 3 lat. Zeszłoroczna zmiana zużycia jest to różnica między zużyciem za rok  $n$  a zużyciem w roku  $n-1$ . Na potrzeby zapytania założmy, że w bazie danych uwzględniono informacje z co najmniej 4 ostatnich lat. W odpowiedzi powinny się znaleźć wiersze ze stopniem zgodności większym lub równym 0.9”.*

Pytanie to w postaci zagnieżdżonej w formie operatorowej może być wyrażone następująco:

```
SELECT nazwa
FROM zaklady zk JOIN zuzycie z
      ON zk.nr_zakl = z.nr_zakl
      JOIN zuzycie z1 ON z.nr_zakl = z1.nr_zakl
```



```

WHERE z.rok = '2002' AND z1.rok = '2001'
and abs(z.papier - z1.papier) mniej_wiecej
ANY (SELECT (abs(zp.papier - zpl.papier) * >= 0.9)
FROM zapotrzebowanie zp JOIN zapotrzebowanie zpl
ON zp.nr_zakl = zpl.nr_zakl
WHERE zp.rok > '2000' AND z.nr_zakl = zp.nr_zakl
AND zpl.rok = 'zp.rok - 1');

```

Pytanie to może zostać zapisane w niezagnieżdżonej postaci w następujący sposób:

```

SELECT DISTINCT nazwa
FROM zaklady zk JOIN zuzycie z
ON zk.nr_zakl = z.nr_zakl
JOIN zuzycie z1 ON z.nr_zakl = z1.nr_zakl
JOIN zapotrzebowanie zp ON z1.nr_zakl = zp.nr_zakl
JOIN zapotrzebowanie zpl ON zp.nr_zakl = zpl.nr_zakl
WHERE z.rok = '2002' AND z1.rok = '2001' AND
zp.rok > '2000' AND zpl.rok = zp.rok - 1 AND
(abs(z.papier - z1.papier) jest abs(zp.papier - zpl.papier)) >= 0.9;

```

Jak widać analizowane powyżej pytania zagnieżdżone da się przekształcić w niezagnieżdżone w jednym kroku.

c) warunek łączący dwa podzapytania typu: **wartość rozmyta z wartością dokładną**:

Weźmy pod uwagę następujące pytanie:

*„Wyszukaj tegoroczne (rok 2003) szacowane zapotrzebowania na toner, które są o co najmniej połowę większe od każdego średniego zużycia we wszystkich zakładach we wszystkich poprzednich latach”.*

Zagnieżdżone pytanie w języku SQL może być wyrażone następująco:

```

SELECT nazwa, z.toner
FROM zaklady zk JOIN zapotrzebowanie z
ON zk.nr_zakl = z.nr_zakl
WHERE z.rok = '2003' AND (z.toner * 2/3) >
(SELECT AVG(zu.toner)
FROM zuzycie zu);

```

W pytaniu tym wykonywane są także operacje arytmetyczne na liczbach rozmytych (mnożenie wartości rozmytej przez stałą: **z.toner \* 2/3**), stosując arytmetykę liczb rozmytych *L-R*.

Pytania to może zostać przekształcone do postaci niezagnieżdżonej w kilku krokach w następujący sposób:

1. Wyznaczamy średnie zużycie toneru:

```

SELECT AVG(zu.toner) as srednia
INTO sr_zuzycie
FROM zuzycie zu;

```

## 2. Wyświetlamy nazwy jednostek:

```
SELECT nazwa, z.toner
FROM zaklady zk JOIN zapotrzebowanie z
ON zk.nr_zakl = z.nr_zakl, sr_zuzycie sr
WHERE z.rok = '2003' AND (z.toner * 2/3) > sr.srednia;
```

d) warunek łączący dwa podzapytania typu: **wartość rozmyta z wartością rozmytą**:

Rozpatrzmy następujące pytanie zadawane do bazy danych *ZAKŁADY*:

*„Wyszukaj nazwy zakładów, które złożyły zapotrzebowanie na największą liczbę płytek CD w roku bieżącym (2003).”*

Zagnieżdżona postać tego pytania w języku SQL w formie operatorowej może być wyrażona następująco:

```
SELECT nazwa
FROM zaklady zk JOIN zapotrzebowanie z
ON zk.nr_zakl = z.nr_zakl
WHERE z.rok = '2003' AND z.plytki_CD >= ALL
    (SELECT z.plytki_CD
     FROM zapotrzebowanie z
     WHERE z.rok = '2003');
```

Ponieważ predykat  $\geq ALL$  można zastąpić wyrażeniem  $\geq SELECT \dots MAX$ , niezagnieżdżona postać tego pytania może być następująca:

```
SELECT nazwa
FROM zaklady z JOIN zapotrzebowanie zp
ON z.nr_zakl = zp.nr_zakl,
(SELECT MAX(zp.plytki_CD) as mx_plytki_CD
 FROM zapotrzebowanie z WHERE z.rok = '2003') mxp
WHERE zp.rok = '2003' AND zp.plytki_CD >= mxp.mx_plytki_CD);
```

e) warunek łączący dwa podzapytania we **frazie HAVING**:

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj lata, w których zużycie papieru przez wszystkie zakłady było mniej więcej równe szacowanemu zapotrzebowaniu za ten rok. W odpowiedzi powinny znaleźć się wiersze ze stopniem zgodności przekraczającym 0.75.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL w formie operatorowej może być wyrażona następująco:

```
SELECT zu.rok
FROM zuzycie zu
GROUP BY zu.rok
HAVING SUM(zu.papier) mniej_wiecej (SELECT SUM(z.papier)
    FROM zapotrzebowanie z
    WHERE z.rok = zu.rok) > 0.75
```

Niezagnieżdżona postać tego pytania w formie operatorowej może być zapisana w dwóch krokach:

1. wyznaczenie szacowanych sumarycznych zapotrzebowań w danym roku

```
SELECT SUM(z.papier) as suma, z.rok
INTO sum_zap
FROM zapotrzebowanie z
GROUP BY z.rok
```

2. realizacja pytania podstawowego, wyszukującego odpowiednie lata:

```
SELECT zu.rok
FROM zuzycie zu JOIN sum_zap sz ON zu.rok = sz.rok
GROUP BY zu.rok
HAVING (SUM(zu.papier) jest sz.suma) > 0.75;
```

## 2. Korelacja dwóch podzapytań:

- a) korelacja typu: **wartość dokładna z wartością dokładną (względem stopni zgodności)**

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj te zakłady, których zapotrzebowanie w roku bieżącym na papier jest w tym samym stopniu bliskie liczbie około 45.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL może być wyrażona następująco:

```
SELECT DISTINCT z1.nr_zakl
FROM zapotrzebowanie z1
WHERE z1.rok = '2003' AND EXISTS
    (SELECT *
     FROM zapotrzebowanie z2
     WHERE z1.nr_zakl <> z2.nr_zakl AND
          (z1.papier jest okolo 45) = (z2.papier jest okolo 45));
```

Niezagnieżdżona postać tego pytania może być następująca:

```
SELECT DISTINCT z1.nr_zakl
FROM zapotrzebowanie z1, zapotrzebowanie z2
WHERE z1.nr_zakl <> z2.nr_zakl AND
      (z1.papier jest okolo 45) = (z2.papier jest okolo 45);
```

- b) korelacja typu: **wartość dokładna z wartością rozmytą lub na odwrót:**

Rozważmy pytanie analizowane w rozdziale 3.6 w punkcie 2.b. Treść tego pytania brzmi następująco:

*„Wyszukaj zakłady, które zużyły w roku 2002 mniej więcej tyle papieru ile oszacowały w zapotrzebowaniu. W odpowiedzi powinny pojawić się te wiersze, których stopień zgodności wynosi co najmniej 0.6.”*

Zagnieżdżona postać tego pytania zapisana w języku SQL może być wyrażona następująco:

```
SELECT nr_zakl
FROM zuzycie zu
WHERE rok = '2002' AND nr_zakl IN
      (SELECT nr_zakl
       FROM zapotrzebowanie z
       WHERE rok = '2002' AND z.nr_zakl = zu.nr_zakl
       AND (z.papier jest zu.papier) >= 0.6);
```

Niezagnieżdżona postać tego pytania jest następująca:

```
SELECT nr_zakl
FROM zuzycie zu JOIN zapotrzebowanie z
      ON zu.nr_zakl = z.nr_zakl AND z.rok = zu.rok
WHERE (z.papier jest zu.papier) >= 0.6) AND z.rok = '2002';
```

c) korelacja typu: **wartość rozmyta z wartością rozmytą:**

Rozważmy pytanie analizowane w rozdziale 3.6 w punkcie 2.d:

*„Wyszukaj zakłady, dla których istnieje chociaż jeden inny zakład, który złożył zapotrzebowanie w roku 2003 na tę samą ilość tonerów. W odpowiedzi powinny znaleźć się wiersze ze stopniem zgodności co najmniej 0.9.”*

Pytanie to w zagnieżdżonej postaci zapisane w języku SQL może być wyrażone następująco:

```
SELECT nr_zakl
FROM zapotrzebowanie z1
WHERE z1.rok = '2003' AND EXISTS
      (SELECT * FROM zapotrzebowanie z2
       WHERE z2.nr_zakl <> z1.nr_zakl AND z2.rok = '2003'
       AND (z2.papier jest z1.papier) >= 0.9);
```

Niezagnieżdżona postać tego pytania może wyglądać następująco:

```
SELECT z1.nr_zakl
FROM zapotrzebowanie z1, zapotrzebowanie z2
WHERE z2.nr_zakl <> z1.nr_zakl AND z1.rok = '2003' AND
      z2.rok = '2003' AND (z2.papier jest z1.papier) >= 0.9)
```

Podstawowym powodem zastępowania pytań zagnieżdżonych pytaniami niezagnieżdżonymi jest skrócenie czasu ich wykonywania, dotyczy to także pytań zawierających wartości rozmyte.

Przy przekształcaniu rozmytych pytań zagnieżdżonych w niezagnieżdżone, należy pamiętać o tym, by warunki filtrujące występujące w pytaniach były identyczne, by stosować te same operatory rozmyte.

## **4. Implementacja rozmytego wyszukiwania danych**

### **4.1. Wprowadzenie**

W pierwszej części tego rozdziału zostanie uzasadniony wybór środowiska pracy oraz odpowiedniego systemu zarządzania bazą danych (SZBD) dla implementacji mechanizmów rozmytych.

Implementacja ta wymagała następnie opracowania definicji i algorytmów dla między innymi następujących elementów programowych:

- typu rozmytego,
- rozmytych funkcji i operatorów arytmetycznych,
- rozmytych funkcji i operatorów relacyjnych,
- rozmytych funkcji i operatorów wyliczających stopień zgodności dla podanego warunku,
- rozmytych funkcji i operatorów iloczynu, sumy i negacji warunków rozmytych,
- rozmytych funkcji agregujących,
- rozmytych kwantyfikatorów,
- funkcji i operatorów realizujących rozmyte grupowanie danych,
- funkcji konwersji wartości rozmytych w ich lingwistyczne odpowiedniki,
- funkcji i operatorów realizujących rozmyty warunek wiążący pytanie zewnętrzne z wewnętrznym.

Proces realizacji tych elementów zostanie omówiony w kolejnych częściach tego rozdziału.

### **4.2. Wybór odpowiedniego środowiska SZBD**

W wyniku analizy różnych systemów zarządzania bazami danych oraz ich środowisk pracy SZBD - wybrano środowisko Unix (Linux), a jako system zarządzania bazą danych PostgreSQL [Dbk01].

Autorami PostgreSQL 1.01 są Andrew Yu i Jolly Chen. W pisaniu, testowaniu i udoskonalaniu kodu uczestniczyło również wiele innych osób. Oryginalny kod Postgres'a, na podstawie którego opracowano PostgreSQL, był efektem prac wielu absolwentów, techników i studentów pracujących w zespole profesora Michaela Stonebrakera

w Uniwersytecie Kalifornijskim w Berkeley [Pstgr7.2UG]. Oryginalną nazwą stosowaną w Berkeley do 1995 roku był Postgres. W 1995 roku dodano obsługę języka SQL i wtedy nazwa została zmieniona na Postgres95. PostgreSQL jest rozwinięciem SZBD Postgres95.

PostgreSQL zawiera model danych i wiele typów danych Postgres'a, przy czym język zapytań PostQuel został zastąpiony rozszerzonym podzbiorem języka SQL. Zaimplementowany język SQL w PostgreSQL jest zgodny ze standardem ANSI, co ułatwia przenoszenie zapytań z innych SZBD [Pstgr7.2PG].

PostgreSQL jest bazą danych, która wchodzi w skład popularnych dystrybucji RedHat Linux, jednak może również działać na innych platformach, na przykład: Solaris, SunOS, HP-UX, AIX, Digital Unix, SCO unix, Unixware i całej gamie innych systemów unixowych. Najnowsze wersje PostgreSQL można również uruchamiać na Windows NT, Windows 2000 oraz na Windows 9x, po zainstalowaniu emulatora środowiska Unixowego, jakim jest biblioteka CYGWIN [Pstgr7.2RM].

Pomimo iż komercyjne bazy danych są wyposażone w wiele ciekawszych możliwości, Postgres jest bazą o dużej popularności. W konkursie Linux World Editor's Choice Awards 1999, w konkurencji baz danych, PostgreSQL wyprzedził produkt Oracle 8i.

Przewagę nad innymi darmowymi serwerami stanowi możliwość obsługi transakcji oraz możliwość obsługi niestandardowych typów danych. PostgreSQL może obsługiwać znacznie więcej typów danych niż tradycyjne bazy danych. Oprócz trzech podstawowych typów danych: znakowych, numerycznych oraz data i czas, można tworzyć definiowane przez użytkownika typy danych. Zaletą SZBD PostgreSQL jest również możliwość tworzenia i implementowania własnych funkcji i operatorów wykorzystywanych później w instrukcjach języka SQL [Pstgr7.2PG]. PostgreSQL ma własne środowisko programistyczne – język pg/plsql, który w swej roli podobny jest do języka PL/SQL w SZBD Oracle. Cechą odróżniającą PostgreSQL od innych bezpłatnych serwerów baz danych (np. My SQL 3.5) jest obsługa zapytań zagnieżdżonych. Wadą jest niestety słabsza wydajność.

Po przeanalizowaniu cech SZBD PostgreSQL oraz licznych jego zalet: **możliwości definiowania nowych typów, funkcji i operatorów, tworzenia podzapytań – system ten okazał się najwygodniejszym narzędziem do implementacji algorytmów aproksymacyjnego wyszukiwania**. Pozwala on na zdefiniowanie typów rozmytych, funkcji i operatorów umożliwiających wykorzystanie elementów teorii zbiorów rozmytych w języku SQL do formułowania rozmytych pytań, jak również wyszukiwania informacji w bazie danych zawierającej kolumny, których wartości mogą być rozmyte.

W pracy przyjęto, że kod źródłowy danych fraz nie zostanie zmodyfikowany, a ingerencje zostaną ograniczone do zdefiniowania odpowiednich funkcji i operatorów.

Analizie poddane zostały także takie SZBD jak MS SQL Server 2000 i Oracle 8i, jednak ze względu na duże koszty tych systemów oraz znacznie większe wymagania sprzętowe nie zostały one wybrane.

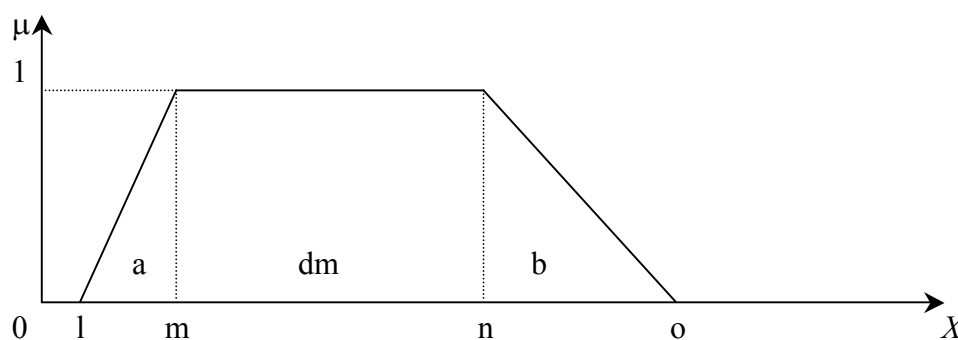
### 4.3. Implementacja rozmytego typu danych

Podstawowym problemem dla implementacji rozmytego typu danych był wybór rodzaju funkcji przynależności, która będzie opisywać taki typ. W rozdziale 2.2 oraz 3.2 przedstawiono kilka klas takich funkcji, w szczególności funkcję Gaussa i trójkątną. Biorąc pod uwagę ogólność, popularność, a także prostotę obliczeń zdecydowano się na wybór funkcji trapezowej.

#### 4.3.1. Opis typu *ftrapezium*

W niniejszym rozdziale zostanie przedstawiona implementacja typu rozmytego zdefiniowanego przez trapezową funkcję przynależności. Będzie on dalej nazywany typem *ftrapezium*. Implementacja ta może być traktowana jako wzorzec implementacji innych typów rozmytych opartych na dowolnej funkcji przynależności.

Typ *ftrapezium* służy do przechowywania informacji rozmytej zdefiniowanej za pomocą trapezowej funkcji przynależności, zilustrowanej na rysunku 4.1.



Rys. 4.1 Trapezowa funkcja przynależności

Trapezową funkcję przynależności charakteryzuje:

- lewostronny przedział niepewności o długości  $a$  (od punktu  $l$  do  $m$ ),
- przedział pewności o długości  $dm$  (od punktu  $m$  do  $n$ ),
- prawostronny przedział niepewności o długości  $b$  (od punktu  $n$  do  $o$ ).

Parametry  $l, m, n, o$  służą do zewnętrznej reprezentacji wartości typu *ftrapezium* opisanej szerzej w podrozdziale 4.3.4., natomiast parametry  $a, m, dm, b$  stosowane są w reprezentacji wewnętrznej opisanej szerzej w podrozdziale 4.3.2.

### 4.3.2. Reprezentacja wewnętrzna

W pracy przyjęto, że użytkownik wprowadza wartości rozmyte w postaci łańcucha  $l/m \sim n \backslash o$  lub funkcji  $okolo(l, m, n, o)$  (podrozdział 4.3.4.). Następnie postać ta, dla zoptymalizowania obliczeń, jest konwertowana do reprezentacji wewnętrznej, opartej na parametrach  $a, m, dm, b$ . Taka postać reprezentacji wewnętrznej jest wskazana, gdyż podczas wykonywania obliczeń na typie *ftrapezium* częściej używane są wartości  $a, m, dm, b$  (np. w obliczeniach, w których stosuje się wzory arytmetyki liczb rozmytych typu  $L-R$ ). Parametry te wyznaczane są przez funkcję *ftrapezium\_in* (opisaną w punkcie 2 załącznika A) na podstawie parametrów  $l, m, n, o$  w następujący sposób:

$$a = m - l$$

$$m = m$$

$$dm = n - m$$

$$b = o - n$$

Następnie  $a, m, dm$  oraz  $b$  umieszczane są w strukturze *ftrapezium* zaimplementowanej w języku C o postaci:

```
typedef struct {
    float8 a, m, dm, b;
} ftrapezium ;
```

Wszystkie cztery parametry  $a, m, dm, b$  (czyli struktura *ftrapezium*) pamiętane są w jednym polu tabeli (podobnie jak np. typ *datetime*, zawierający datę i czas).

### 4.3.3. Tworzenie i modyfikacja tabel rozmytych

Rozmyte tabele (zawierające rozmyte kolumny) tworzy się, podobnie jak tabele klasyczne, poleceniem *CREATE TABLE*, natomiast modyfikuje poleceniem *ALTER TABLE*.

Typ kolumny, która ma zawierać rozmyte dane należy podać **ftrapezium**.

#### Przykład 4.1

Instrukcja *CREATE TABLE (ALTER TABLE)* w języku *SQL* pozwalająca na utworzenie tabeli *Zapotrzebowanie* ma następującą postać:

```
CREATE TABLE Zapotrzebowanie (
    nr_zakl integer
    , rok integer
    , papier ftrapezium
    , toner ftrapezium
    , plytki_CD ftrapezium
    , PRIMARY KEY ( nr_zakl, rok )
) ;

ALTER TABLE Zapotrzebowanie ADD COLUMN dyskiетки ftrapezium ;
ALTER TABLE Zapotrzebowanie DROP COLUMN dyskiетки ;
```



#### 4.3.4. Wprowadzanie, modyfikacja i usuwanie wartości typu *fttrapezium*

Dla potrzeb wprowadzania danych typu **fttrapezium**, opisanych parametrami *l*, *m*, *n*, *o* (rys. 4.1), zaproponowano tzw. reprezentację zewnętrzną.

Reprezentacja zewnętrzna jest ciągiem alfanumerycznym o ogólnej postaci '*l/m~n\o*' gdzie parametry *l*, *m*, *n* i *o* należą do liczb rzeczywistych.

Wartości rozmyte typu *fttrapezium* (oraz wyrażenia je zawierające) uzyskujemy na jeden z czterech sposobów:

1. w postaci alfanumerycznej '*l/m~n\o*',

```
INSERT INTO zapotrzebowanie( nr_zakl, rok, papier, toner, plytki_CD)
values( 1, '2002', '22/25~25\\27', '4/5~6\\7', '120/150~170\\190') ;
```

Konieczność zapisu podwójnego znaku '\\' wynika z ograniczeń wybranego SZBD, w SZBD PostgreSQL znak '\' jest znakiem specjalnym i należy poprzedzić go dodatkowym '\\.

2. przez użycie funkcji *fttrapezium* ('*l/m~n\o*'),

```
SELECT * FROM pracownicy p
WHERE p.wiek jest fttrapezium('20/23~25\\27');
```

3. przez użycie konstrukcji ('*l/m~n\o*') :: *fttrapezium*,

```
SELECT * FROM pracownicy p
WHERE p.wiek jest ('20/23~25\\27')::fttrapezium;
```

4. przez użycie funkcji *okolo*(*l*, *m*, *n*, *o*) – jeśli liczbę rozmytą wprowadzamy do wyrażen, stosując zdefiniowaną funkcję *okolo* (punkt 2 załącznika A).

```
SELECT * FROM pracownicy p
WHERE p.wiek jest okolo(20, 23, 25, 27);
```

Postacie 2, 3 oraz 4 są sobie równoważne, a ich użycie zależy wyłącznie od preferencji użytkownika; można je także stosować zamiast sposobu 1 przy wprowadzaniu wartości rozmytych poleceniem *INSERT*.

Polecenia modyfikujące *UPDATE* oraz usuwające *DELETE* niczym nie różnią się od klasycznego użycia:

```
UPDATE zapotrzebowanie SET papier = '20/23~24\\27' ;
DELETE FROM zapotrzebowanie WHERE papier jest okolo(20, 23, 24, 27);
```

#### 4.3.5. Wyprowadzanie wartości typu *fttrapezium*

W tym podrozdziale przedstawiony zostanie problem prezentowania (wyświetlania) wyszukiwanych wartości typu rozmytego *fttrapezium*. W kilku analizowanych przykładach również we frazie *select* instrukcji *SELECT* występowały wartości rozmyte. We frazie *select* podajemy te kolumny, wyrażenia czy funkcje agregujące, których wartości mają znaleźć się w zbiorze wynikowym.

W tej frazie również mogą pojawić się wartości rozmyte zarówno same, jak i w wyrażeniach, funkcjach agregujących czy funkcjach konwertujących. (sposób ich implementacji przedstawiony jest w punkcie 8.1 i 8.2 załącznika A)

Rozważmy najpierw przykłady wyświetlenia wyszukanych wartości rozmytych w formie alfanumerycznej.

Do bazy danych *ZAKŁADY* może zostać skierowane przykładowe pytanie:

*„Wyszukaj wartość zapotrzebowania na papier Zakładu Oprogramowania w roku 2003.”*

```
SELECT papier, nazwa
FROM zapotrzebowanie zp JOIN zakłady z
      ON zp.nr_zakl = z.nr_zakl
WHERE zp.rok = '2003' AND z.nazwa = 'Zakład Oprogramowania';
```

W powyższym pytaniu kolumna **papier** zawiera wartości rozmyte.

Postać przykładowego wiersza wynikowego ilustruje tabela 4.1:

Tabela 4.1

Wiersz wynikowy	
papier	nazwa
12/14~16\18	Zakład Oprogramowania

Wartości rozmyte mogą się również pojawić jako wynik zapytania wykonującego np. agregację wartości rozmytych, co jest tematem następującego pytania:

*„Wyszukaj sumaryczne zamówienia na papier w poszczególnych latach”*

```
SELECT sum(papier), rok
FROM zapotrzebowanie zp
GROUP BY rok;
```

Przykładowe wiersze wynikowe ilustruje tabela 4.2:

Tabela 4.2

Wiersze wynikowe	
sum(papier)	Rok
45/48~50\53	Zakład Baz Danych
23/25~28\32	Zakład ZMiTAC
25/29~32\35	Zakład Oprogramowania

Celem kolejnego przykładu jest zilustrowanie użycia funkcji konwertującej wartość rozmytą do wartości lingwistycznej.

*„Wyszukaj zamówienia na liczbę tonerów w poszczególnych zakładach w roku 2003, Wartości rozmyte zastąp odpowiadającymi im wartościami lingwistycznymi.”*

```
SELECT toner_to_lingw(toner), nazwa
FROM zapotrzebowanie zp JOIN zakłady z
      ON zp.nr_zakl = z.nr_zakl
WHERE zp.rok = '2003';
```

W zapisie pytania zastosowano funkcję konwertującą `toner_to_lingw(toner)` przypisującą liczbom rozmytym odpowiednie wartości lingwistyczne.

W tabeli 4.3 przedstawiono przykładowe wiersze wynikowe.

Tabela 4.3  
Wartości lingwistyczne po konwersji

toner	nazwa
dużo	Zakład Bazy Danych
średnio	Zakład ZMiTAC
średnio	Zakład Oprogramowania

Kolejne pytanie kierowane do bazy danych *ZAKŁADY* wyszukujące wartości stopni zgodności z warunkami pytania ma następującą treść [Młs03]:

*„Wyszukaj 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 w formie funkcyjnej może być wyrażona następująco:

```
SELECT nazwisko, wiek, DG(wiek jest okolo 50) as stopien_przynaloznosci_1,
      staz_pracy, DG(staz_pracy jest okolo 20) as stopien_przynaloznosci_2
FROM pracownicy;
```

W powyższym pytaniu wyznaczona jest wartość stopnia przynależności, przy zastosowaniu funkcji DG, w której obliczany jest stopień zgodności wartości analizowanego wiersza z zadaniem kryterium. Przykładowe wiersze wynikowe wygenerowane w wyniku wykonania tego pytania przedstawione są w tabeli 4.4.

Tabela 4.4  
Wiersze wynikowe ze stopniami przynależności

nazwisko	wiek	stopien_ przynaloznosci 1	staz_pracy	stopien_ przynaloznosci 2
Kowalik	48	0,8	19	0,9
Sarna	21	0,0	1	0,0
Nowak	38	0,0	10	0,0
Sroka	53	0,7	22	0,8

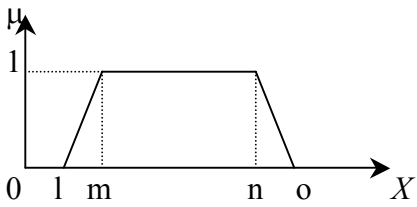
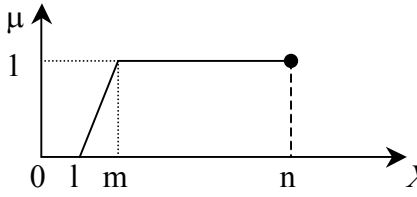
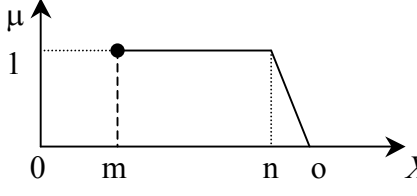
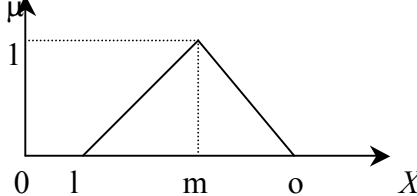
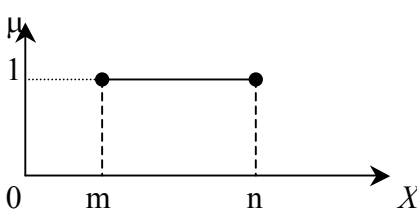
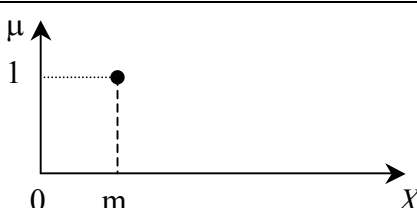
#### 4.3.6. Szczególne przypadki typu *ftrapezium*

Dla przedstawionej na rysunku 4.1 ogólnej postaci typu *ftrapezium* można wyróżnić przypadki szczególne, przedstawione w tabeli 4.1, gdy wprowadzane wartości nie posiadają jednej lub więcej charakterystycznych cech wymienionych w podrozdziale 4.3.1. (tzn. nie posiadają lewostronnego przedziału niepewności i/lub przedziału pewności i/lub prawostronnego przedziału niepewności).

W takich przypadkach w reprezentacji zewnętrznej można stosować uproszczony zapis przedstawiony w tabeli 4.5.

Tabela 4.5

## Szczególne przypadki funkcji trapezowej

Postać graficzna	Reprezentacja zewnętrzna	Opis
	$l/m\sim n\backslash o$	przedział rozmyty
	$l/m\sim n$	przedział lewostronnie rozmyty (brak prawostronnego przedziału niepewności)
	$m\sim n\backslash o$	przedział prawostronnie rozmyty (brak lewostronnego przedziału niepewności)
	$l/m\backslash o$	liczba rozmyta (brak przedziału pewności)
	$m\sim n$	przedział dokładny (brak obu przedziałów niepewności)
	$m$	wartość dokładna (brak obu przedziałów niepewności i przedziału pewności)

Do konwersji reprezentacji wewnętrznej typu **ftrapezium** do postaci zewnętrznej wyrażonej w postaci alfanumerycznej opracowano funkcję *ftrapezium\_out* (opisaną

w punkcie 2 załącznika A). Funkcja ta jest zupełnie nieistotna z punktu widzenia użytkownika, gdyż SZBD wywołuje ją niejawnie w celu wyświetlenia wartości ftrapezium.

Analogicznie można tworzyć inne typy rozmyte o dowolnych funkcjach przynależności oraz elementy rozmyte z nimi związane (na przykład funkcji Gaussa).

#### 4.4. Podstawowe działania na wartościach typu ftrapezium

W podrozdziale tym zostaną omówione zaimplementowane działania na liczbach rozmytych typu ftrapezium. W pierwszej części będą przedstawione działania arytmetyczne, w następnej logiczne.

##### 4.4.1. Funkcje i operatory arytmetyczne

PostgreSQL umożliwia definiowanie operatorów lewostronnych, prawostronnych, jak również obustronnych. Można definiować operatory przeciążone, czyli takie, w których ta sama nazwa używana jest dla różnych operatorów mających różną liczbę i typ argumentów.

Utworzenie operatora w PostgreSQL jest realizowane w kilku krokach [Pstgr7.2PG]:

- utworzenie funkcji, którą będzie wywoływał operator (PostgreSQL umożliwia dodawanie własnych funkcji na dwa sposoby: w języku pg/plsql oraz w innych językach programowania np.: w języku C).
- rejestracja tej funkcji w PostgreSQL, jeśli nie została napisana w pg/plsql,
- zdefiniowanie operatora w PostgreSQL, wykorzystującego tę funkcję..

W pracy zaimplementowano podstawowe operacje na liczbach typu *ftrapezium*, wykorzystując arytmetykę liczb rozmytych, opisaną w rozdziale 2.2.3.

Wymagało to oddzielnego rozpatrzenia następujących operacji:

- wyznaczanie liczby przeciwnej do podanej „-”,
- dodawanie do liczby rozmytej wartości dokładnej „+”,
- dodawanie do liczby dokładnej wartości rozmytej „+”,
- odejmowanie od liczby rozmytej wartości dokładnej „+”,
- odejmowanie od liczby dokładnej wartości rozmytej „-”,
- dodawanie dwóch liczb rozmytych „+”,
- odejmowanie dwóch liczb rozmytych „-”,
- dzielenie liczby rozmytej przez stałą „/”,
- dzielenie stałej przez liczbę rozmytą „/”,
- mnożenie liczby rozmytej przez stałą „\*”,

- mnożenie stałej przez liczbę rozmytą „\*”,
- dzielenie dwóch liczb rozmytych „/”,
- mnożenie dwóch liczb rozmytych „\*”,
- wyznaczanie wartości mniejszej spośród dwóch liczb rozmytych,
- wyznaczanie wartości większej spośród dwóch liczb rozmytych.

Nagłówki zaimplementowanych w języku C funkcji i ich definicje w SZBD PostgreSQL oraz operatory rozmyte umieszczono w punkcie 3.1 załącznika A.

Przykładowy schemat wykonania operacji wyznaczania liczby przeciwnej do liczby rozmytej jest następujący:

1. reprezentacją wewnętrzną liczby (4.3.2.) jest struktura o parametrach (a, m, dm, b),
2. jeśli liczba ma wartość NULL zwróć wartość NULL i zakończ,
3. jeśli nie to :
  - zmiennej tymczasowej  $b2$  przypisz wartość parametru  $a$ ,
  - parametrowi  $m$  przypisz wartość wyrażenia:  $-(m + dm)$ ,
  - parametr  $dm$  pozostaje bez zmian,
  - parametrowi  $a$  przypisz wartość parametru  $b$ ,
  - parametrowi  $b$  przypisz wartość zmiennej tymczasowej  $b2$ ,
4. Obliczone nowe wartości parametrów zwróć jako wynik funkcji i zakończ,
5. Koniec.

#### 4.4.2. Funkcje i operatory relacyjne na liczbach typu *ftrapezium*

W przeciwieństwie do funkcji i operatorów arytmetycznych, które są jednoznacznie zdefiniowane, w przypadku funkcji i operatorów relacyjnych istnieje kilka alternatywnych do siebie definicji [Kcp01].

Ze względu na to, że systemy zarządzania bazami danych wymuszają, by operatory relacyjne w wyniku działania zwracały prawdę (true) lub fałsz (false), w pracy zdefiniowano je w następujący sposób:

Przyjmijmy, że  $f_1, f_2$  to wartości rozmyte

1. operator równości „=”

$$f_1 = f_2 \Leftrightarrow (l_1 = l_2) \wedge (m_1 = m_2) \wedge (n_1 = n_2) \wedge (o_1 = o_2)$$

2. operator różności „<>”

$$f_1 <> f_2 \Leftrightarrow (l_1 <> l_2) \vee (m_1 <> m_2) \vee (n_1 <> n_2) \vee (o_1 <> o_2)$$

3. operator mniejszości „<”

$$f_1 < f_2 \Leftrightarrow (m_1 < m_2)$$

4. operator mniejszy lub równy „ $\leq$ ”

$$f_1 \leq f_2 \Leftrightarrow (f_1 < f_2) \vee (f_1 = f_2)$$

5. operator większości „ $>$ ”

$$f_1 > f_2 \Leftrightarrow (n_1 > n_2)$$

6. operator większy lub równy „ $\geq$ ”

$$f_1 \geq f_2 \Leftrightarrow (f_1 > f_2) \vee (f_1 = f_2)$$

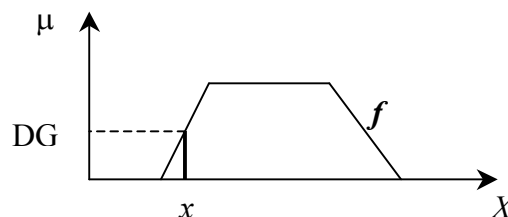
Nagłówki odpowiednich funkcji zaimplementowanych w języku C i definicje funkcji i operatorów w SZBD PostgreSQL umieszczono w punkcie 3.2 załącznika A.

#### 4.5. Funkcje i operatory wyznaczające wartość stopnia zgodności

W rozdziale tym przedstawiono zaimplementowane funkcje, pozwalające na wyznaczenie wartości stopnia zgodności (sposób wyznaczania opisano w rozdziale 3.1) w następujących przypadkach:

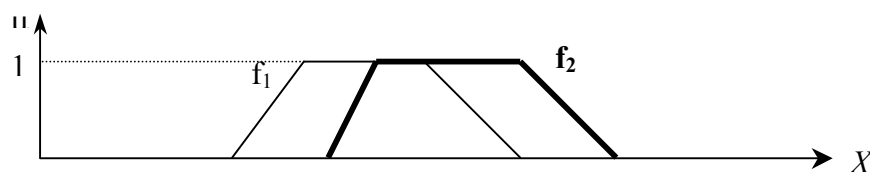
1. Stopień zgodności między wartością rozmytą  $f$  typu *ftrapezium* a wartością dokładną  $x$ ; Jest on obliczany na podstawie wzoru:

$$DG(x, f) = \begin{cases} 0, & x \leq l, \\ \frac{x-l}{m-l}, & l < x \leq m, \\ 1, & m < x \leq n, \\ \frac{o-x}{o-n}, & n < x \leq o, \\ 0, & x > o. \end{cases}$$



Rys. 4.2 Stopień zgodności między wartością rozmytą  $f$  a wartością dokładną  $x$

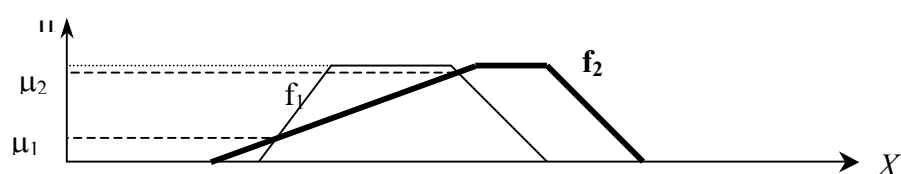
2. Stopień zgodności między dwiema wartościami rozmytymi  $f_1, f_2$ ; Można tu wyróżnić trzy przypadki:
  - a. przedziały zgodności nachodzą na siebie (mają co najmniej jeden punkt wspólny) – wtedy stopień zgodności wynosi 1, tzn.  $DG(f_1, f_2) = 1$



Rys. 4.3 Przedziały zgodności nachodzą na siebie

- b. odcinek reprezentujący jeden z przedziałów niepewności  $f_1$  przecina się (ma punkt wspólny) z odcinkiem reprezentującym jeden z dwóch przedziałów niepewności  $f_2$ . Pojawiają się tu cztery możliwości:

- lewostronny przedział niepewności  $f_1$  z lewostronnym przedziałem niepewności  $f_2$ ,
- lewostronny przedział niepewności  $f_1$  z prawostronnym przedziałem niepewności  $f_2$ ,
- prawostronny przedział niepewności  $f_1$  z lewostronnym przedziałem niepewności  $f_2$ ,
- prawostronny przedział niepewności  $f_1$  z prawostronnym przedziałem niepewności  $f_2$ .

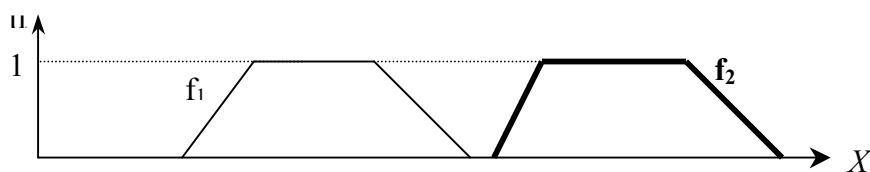


Rys. 4.4 Lewostronne i prawostronne przedziały niepewności  $f_1$  i  $f_2$  przecinają się;  $DG(f_1, f_2) = \mu_2$

Za stopień zgodności przyjmuje się odciętą tego punktu, a jeśli istnieje więcej takich punktów, za stopień zgodności przyjmuje się maksymalną z nich.

Punkty przecięcia dwóch odcinków wyznacza się na podstawie zależności z geometrii analitycznej opisujących funkcje  $f_1$  i  $f_2$ . Z uwagi na założenia, odrzuca się punkty o odciętej nie mieszczącej się w przedziale  $[0,1]$ . Sytuacje takie są rozpatrywane w pozostałych dwóch przypadkach.

- c. przedziały pewności nie zachodzą na siebie i nie istnieją punkty, o których mowa w poprzednim podpunkcie (trapezy reprezentujące wartości rozmyte  $f_1$  i  $f_2$  są rozłączne). Wtedy stopień zgodności wynosi 0, tzn.  $DG(f_1, f_2) = 0$ .



Rys. 4.5 Trapezy reprezentujące wartości rozmyte  $f_1$  i  $f_2$  są rozłączne

Nagłówki funkcji zaimplementowanych w języku C i ich definicji w SZBD PostgreSQL oraz operatorów rozmytych zostały dołączone w punkcie 4 załącznika A.



#### 4.6. Realizacja funktorów iloczynu, sumy i negacji rozmytych

W tym podrozdziale zostaną omówione te zrealizowane elementy, które nie są związane z konkretnym typem funkcji przynależności. Należą do nich operatory iloczynu rozmytego &&& i sumy rozmytej ||| oraz operator negacji ~.

Operatory &&& i ||| zastępują operacje logiczne *AND* i *OR* dla wyrażeń rozmytych, na przykład wiążą warunki rozmyte we frazie *WHERE*. Operator negacji ~ umożliwia uzyskanie dopełnienia stopnia zgodności.

*SZBD PostgreSQL nie umożliwia tworzenia operatorów o nazwach literowych, dlatego przyjęto nazwy &&& i ||| oraz ~.*

Szczegóły implementacji tych operatorów są następujące:

- operator &&& realizuje operację iloczynu rozmytego wykorzystując *t-normę* Zadeh’a obliczaną jako:

$$T(\mu_A(x), \mu_B(x)) = \min(\mu_A(x), \mu_B(x)),$$

- operator ||| realizuje operację sumy rozmytej wykorzystując *s-normę* Zadeh’a obliczaną jako:

$$S(\mu_A(x), \mu_B(x)) = \max(\mu_A(x), \mu_B(x)),$$

- natomiast operator ~ realizuje operację negacji (dopełnienia).

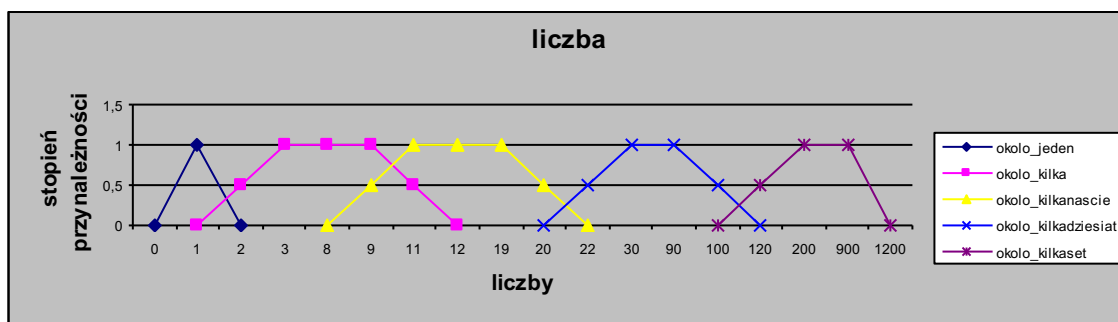
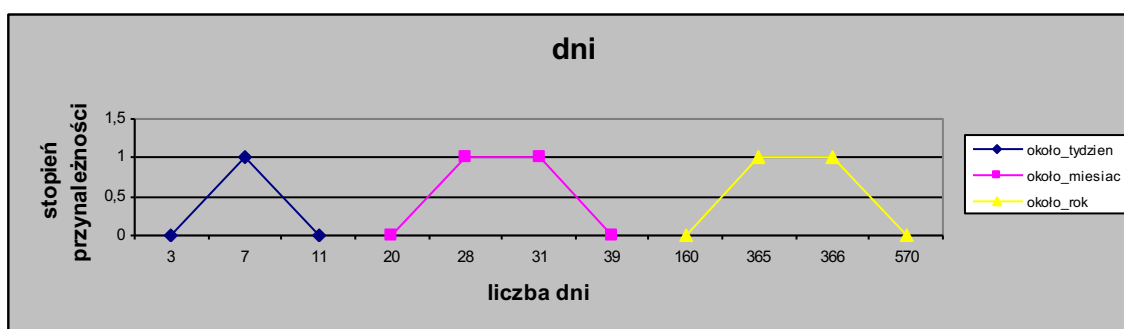
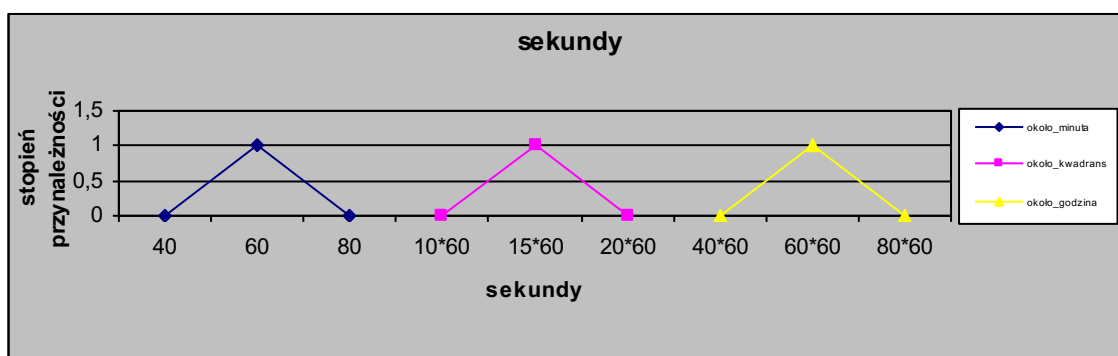
Nagłówki odpowiednich funkcji i definicje operatorów przedstawiono w punkcie 5 załącznika A.

#### 4.7. Wartości predefiniowane dla typu **fttrapezium**

W praktycznym stosowaniu wartości typu **fttrapezium**, należy ustrzec się przed pewnymi nieprawidłowościami (błędami), których rodzaj ilustruje następujący przykład:

Dla uniknięcia indywidualnej, a przez to subiektywnej interpretacji liczbowej często spotykanych określeń: *kilka*, *kilkanaście*, *około kwadrans*, *około godzina*, *około połowa* itd. w pracy zdefiniowano odpowiednie wartości lingwistyczne wraz z ich funkcjami przynależności.

Rysunki 4.6, 4.7 i 4.8 przedstawiają funkcje przynależności dla przykładowych uniwersalnych zmiennych lingwistycznych wykorzystywanych także w dalszej części pracy.

Rys. 4.6 Funkcje przynależności wartości lingwistycznych zmiennej *liczba*Rys. 4.7 Funkcje przynależności wartości lingwistycznych zmiennej *dni*Rys. 4.8 Funkcje przynależności wartości lingwistycznych zmiennej *sekundy*

Przyjęcie takich, a nie innych wartości jest mniej lub bardziej subiektywne – jednak projektant bazy danych, w porozumieniu z ekspertem z danej dziedziny, może sparametryzować własne funkcje w dowolny, uznany przez niego za najbardziej odpowiedni, sposób.

Pamiętając o funkcjach operujących na typie ftrapezium rozwiązanie to można poszerzyć, łącząc tego rodzaju funkcje w wyrażenia rozmyte typu:

*okolo\_kilka()* \* *okolo\_tydzien()* co odpowiada wartości *okolo\_kilka\_tygodni()*

*1,5* \* *okolo\_godzina()* co odpowiada wartości *okolo\_1,5\_godziny()*

*okolo\_kilka()* \* *okolo\_miesiac()* co odpowiada wartości *okolo\_kilka\_miesiecy()*

To bardziej zaawansowane użycie nie wymaga żadnych czynności od strony tworzącego bazę danych i może być wykorzystywane w dowolnych, potrzebnych użytkownikowi kombinacjach. Sposób implementacji przedstawionych funkcji znajduje się w załączniku A w punkcie 6.

#### 4.8. Funkcje agregujące na liczbach typu *ftrapezium*

W pracy zdefiniowano i zaimplementowano funkcje agregujące operujące na liczbach rozmytych typu *ftrapezium*. Są one następujące:

- ***sum*** – sumująca wszystkie liczby rozmyte w grupie wierszy,
- ***avg*** – wyznaczająca wartość średniej arytmetycznej spośród wszystkich liczb rozmytych w grupie,
- ***min*** – wyznaczająca wartość minimalną spośród wszystkich liczb rozmytych w grupie,
- ***max*** – wyznaczająca wartość maksymalną spośród wszystkich liczb rozmytych w grupie,
- ***rozmyte kwantyfikatory*** – funkcje agregujące rozmywające funkcję agregującą *odsetek*.

##### 4.8.1. Funkcje agregujące: *sum, avg, min, max*

SZBD PostgreSQL umożliwia przeciążanie funkcji, tzn. na podstawie typów argumentów rozpoznaje odpowiednią funkcję. Fakt ten pozwolił na zaimplementowanie rozmytych funkcji agregujących (operujących na danych typu *ftrapezium*) o takich samych nazwach jak nazwy klasycznych funkcji agregujących, operujących na dokładnych danych.

Nagłówki funkcji zaimplementowanych w języku C oraz ich definicje w SZBD PostgreSQL zamieszczono w punkcie 8.1 załącznika A. W obliczeniach wykorzystano arytmetykę liczb rozmytych typu *LR* (rozdział 2.2.3).

Realizacja funkcji agregujących w SZBD przebiega według ogólnego schematu:

1. *{blok inicjalizujący}* - wpisz do zmiennej *stan* wartość inicjującą.
2. Jeśli zagregowano już wszystkie wartości w grupie wykonaj krok 6.
3. Pobierz wartość kolejną z grupy jako *wartość bieżącą*.
4. *{blok aktualizujący}* – uaktualnij zmienną *i* na podstawie zmiennej *stan* i *wartości bieżącej*.
5. Przejdź do punktu 2.
6. *{blok finalizujący}* - wykonaj obliczenia kończące agregację i zwróć je jako wartość funkcji agregującej.
7. Koniec.

Dla przykładowej funkcji agregującej **avg(ftrapezium)** poszczególne elementy schematu oznaczają:

zmienna *stan* jest strukturą o polach:

*licznik* (liczba całkowita) – przechowujące liczbę rekordów,

*suma* (wartość typu *ftrapezium*) – przechowująca sumę dotychczas zagregowanych wartości,

*blok inicjujący* zawiera przypisania

*stan.licznik* := 0 ;

*stan.suma* := '0/0~0\0'

*blok aktualizujący* zawiera przypisania:

*stan.licznik* := *stan.licznik* + 1;

*stan.suma* := *stan.suma* + bieżąca wartość

*blok finalizujący* oblicza wartość funkcji agregującej następująco:

jeśli *stan.licznik* = zero wtedy wartość agregatu := NULL

w przeciwnym razie wartość agregatu := *stan.suma*/*stan.licznik*

#### 4.8.2. Rozmyte kwantyfikatory

W nawiązaniu do rozdziału 4.9.1 opisany zostanie sposób realizacji funkcji agregującej *odsetek*, co umożliwi zrozumienie realizacji rozmytych kwantyfikatorów typu: prawie wszystkie, prawie żaden, około połowa, przedstawionych później.

*Odsetek* jest funkcją agregującą zwracającą stosunek liczby wierszy spełniających zadane kryterium do liczby wszystkich wierszy w grupie.

Proces obliczania funkcji *odsetek* realizowany jest następująco:

1. Zmienna *stan* jest strukturą o polach:

– *wszystkie* (liczba całkowita) – przechowuje liczbę wierszy poddanych agregacji,

– *spełniające* (liczba całkowita) – przechowuje liczbę wierszy spełniających warunek agregacji.

2. Blok inicjujący zawiera przypisania:

– *stan.wszystkie* = 0;

– *stan.spełniające* = 0;

3. Blok aktualizujący zawiera przypisania:

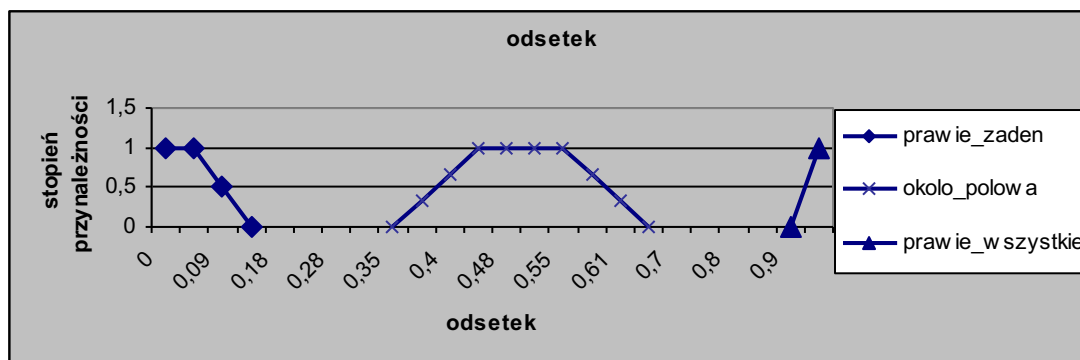
– *stan.wszystkie* = *stan.wszystkie* + 1 ;

– jeśli *bieżąca wartość* równa się *true* wtedy *stan.spełniające* = *stan.spełniające* + 1;

4. W bloku finalizującym następuje zwrócenie wyniku funkcji *odsetek*:

- Jeśli *stan.wszystkie* jest równy zero zwróć jako wynik wartość zero, w przeciwnym przypadku zwróć jako wynik iloraz *stan.spelniajace* / *stan.wszystkie*.

W rozdziale 3.4.3 wprowadzono pojęcie kwantyfikatora rozmytego i wyjaśniono jego związek z funkcją *odsetek*. Funkcje przynależności dla przykładowych kwantyfikatorów przedstawia rysunek 4.9. Można ewentualnie rozpatrywać inne kwantyfikatory, np.: około jedna trzecia, około trzy czwarte itd.

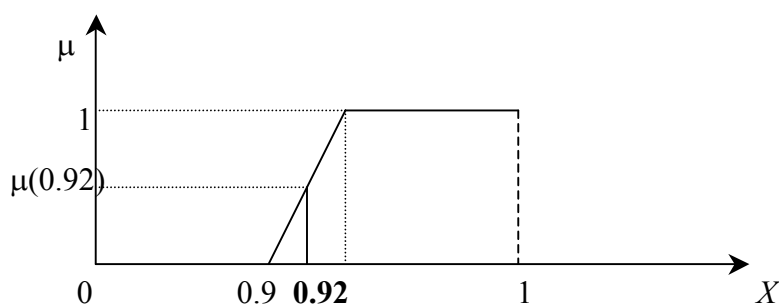


Rys. 4.9 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *odsetek*

Sposób realizacji kwantyfikatora rozmytego różni się jedynie, w stosunku do funkcji *odsetek*, rozbudowaniem bloku finalizującego.

W bloku *finalizującym* oblicza się stopień zgodności *odsetka* z odpowiednią wartością rozmytą (rysunek 4.10). Obliczony stopień zgodności jest zwracany jako wartość kwantyfikatora rozmytego (funkcji agregującej).

Implementacja wszystkich przedstawionych funkcji znajduje się w załączniku A w punkcie 8.2.



Rys. 4.10 Stopień zgodności obliczonego odsetka (0.92) z wartością lingwistyczną *prawie\_wszystkie*

#### 4.9. Implementacja hierarchicznego algorytmu rozmytego grupowania danych dokładnych

Wstępna ocena możliwości zaimplementowania hierarchicznego algorytmu rozmytego grupowania wskazywała na konieczność zastąpienia w istniejącym systemie całego fragmentu kodu odpowiedzialnego za interpretację frazy *GROUP BY*. Byłaby to bardzo poważna zmiana, odbiegająca od wszystkich pozostałych rozwiązań zastosowanych w pracy, a sprowadzających się do implementacji własnych (autorskich) funkcji i operatorów. Ostatecznie znaleziono jednak rozwiązanie ograniczone do tych dwóch elementów. Jego wyjaśnienie wymaga dość szczegółowego przeanalizowania dotychczasowego sposobu wykonywania frazy *GROUP BY X* w systemie PostgreSQL.

Pierwszym etapem tego procesu jest sortowanie wartości kolumny *X*. Załóżmy, że w jego wyniku otrzymujemy ciąg rosnących wartości  $x_1, x_2, \dots, x_N$ .

Drugi etap, właściwego grupowania, realizowany jest według algorytmu (rysunek 4.11), który można przedstawić w sposób bardzo podobny do rysunku 3.11.

Po tych spostrzeżeniach można sformułować następujące pytanie, prowadzące do rozwiązania postawionego na wstępie problemu: *Jak zmienić w powyższym algorytmie wyrażenie warunku porównania  $x_i = x_{min}$  na wyrażenie  $x_i - x_{min} \leq maxd$ , gdzie  $maxd$  jest dopuszczalną rozpiętością grupy (odległością między skrajnymi elementami grupy)?*

Taka modyfikacja zmieniałaby bowiem istniejący algorytm grupowania w docelowy algorytm przedstawiony na rysunku 3.11.

Okazało się to możliwe poprzez zdefiniowanie własnego operatora równości (=), który powinien zastąpić istniejący operator we wspomnianym wyrażeniu.

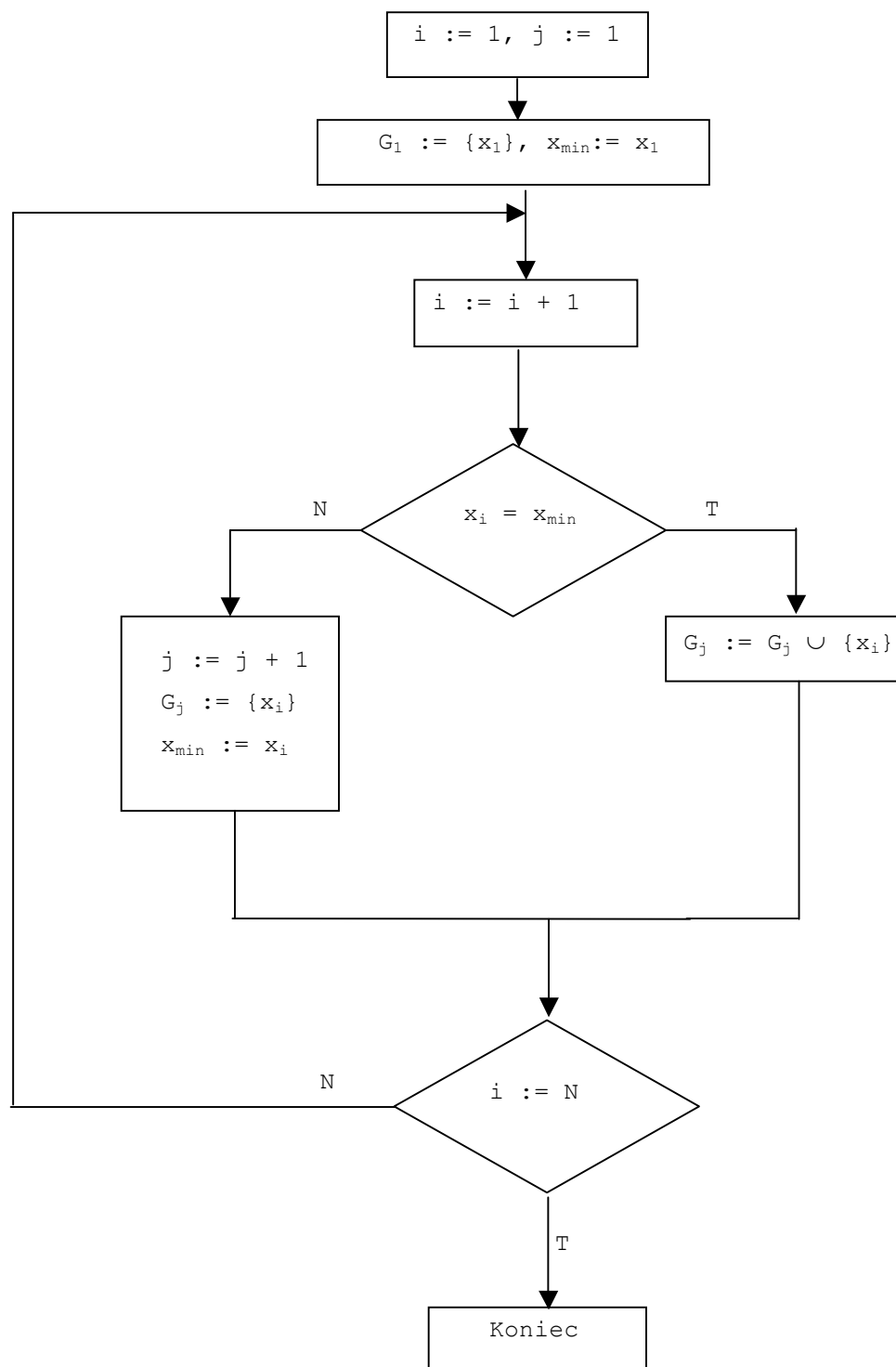
System PostgreSQL pozwala definiować własne operatory, jednak zastąpienie operatora standardowego operatorem nowo zdefiniowanym trzeba w tym systemie wymusić. Formą takiego wymuszenia może być wprowadzenie do analizowanego wyrażenia danych o niestandardowym typie. System wywołuje wtedy funkcję obsługującą nowy typ danych. I właśnie taką technikę zastosowano w pracy. Całość opracowanego rozwiązania składa się z następujących etapów:

1. Przekształcenie danych z kolumny, według której wykonywane jest grupowanie, za pomocą zaimplementowanej funkcji *otoczenie*. Funkcja ta wywoływana jest następująco:

```
GROUP BY otoczenie (<kolumna>, <maxd>);
```

i przekształca każdą wartość rzeczywistą z kolumny w dwuelementową strukturę typu *otoczenie\_r* o polach:

- wartość rzeczywista,
- rozpiętość ( $maxd$ ) grupy elementów typu rzeczywistego.



Rys. 4.11 Istniejący algorytm grupowania

Całość tworzy nowy typ danych, wymagający własnej funkcji dla realizacji wspomnianego operatora równości „=”, ale również własnej funkcji dla operatora mniejszości „<” wykorzystywanego wcześniej do uporządkowania danych nowego typu.

2. Zdefiniowanie wspomnianego operatora mniejszości „<” działającego na dwóch wartościach typu *otoczenie\_r*. Porównuje on dwie wartości typu *otoczenie\_r* wyłącznie na podstawie ich pól *wartość*. Operator ten jest konieczny, ponieważ SZBD wykorzystuje go niejawnie do sortowania grupowanych wartości, które odbywa się przed ostatecznym podziałem na grupy.
3. Zmodyfikowanie operatora równości „=” działającego tym razem na dwóch wartościach typu *otoczenie\_r*. Operator ten zwraca wartość *true* jeśli:
  - $otoczenie\_r1.wartosc - otoczenie\_r2.wartosc \leq otoczenie\_r1.rozpiętosc$
  - w przeciwnym wypadku zwraca wartość *false*.

Wartość *true* tego operatora oznacza, że porównywane wartości znajdują się w jednej grupie. W przeciwnym razie (*false*) algorytm SZBD utworzy nową grupę.

Aby zastosować grupowanie względem kolumny innego typu niż rzeczywisty należy, korzystając z przedstawionego tutaj sposobu utworzyć nową (przeciążoną) funkcję *otoczenie*, oraz nowe (przeciążone) operatory mniejszości „<” i równości „=” operujące na odpowiednich typach.

Omówiona tutaj metoda grupowania rozmytego według jednej kolumny może rozszerzać się na grupowanie według większej liczby kolumn, ponieważ SZBD używa utworzonych funkcji niezależnie dla każdej z kolumn względem których chcemy grupować.

W pracy zaimplementowano odpowiednie funkcje i operatory umożliwiające takie grupowanie dla wartości typu float8, a ich implementację można znaleźć w punkcie 9 załącznika A.

W pracy zaimplementowano również inne metody grupowania danych:

- omówioną powyżej metodę rozszerzono i dostosowano, tak by można było grupować z jej użyciem dane typu *ftapezium* (wymagało to modyfikacji operatorów „<” i „=”),
- najbardziej restrykcyjną metodę grupowania rozmytych danych (opartą na równości wszystkich parametrów opisujących funkcję przynależności danej wartości rozmytej).



#### 4.10. Porządkowanie kolumn zawierających wartości rozmyte

W przypadku, gdy zapytanie wymaga uporządkowania kolumny zawierającej wartości rozmyte, pojawia się problem określenia, która z wartości rozmytych jest większa, a która mniejsza. W pracy zaproponowano rozwiązanie tego problemu w następujący sposób [Młs03]:

- W przypadku liczb rozmytych, można porównywać ich wartości modalne.
- W przypadku przedziałów rozmytych, można 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 wartość większą, porównujemy największe wartości modalne przedziałów.

Przykładowe pytanie kierowane do bazy danych *ZAKŁADY* może brzmieć następująco:

*„Wyszukaj zakłady posortowane względem zapotrzebowania na ryzy papieru”.*

Pytanie to zapisane w języku SQL może mieć następującą postać:

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

#### 4.11. Funkcje i operatory realizujące warunek wiążący pytanie zewnętrzne z wewnętrznym

Szczególnie trudnym problemem jest uwzględnienie w tej realizacji warunku nakładanego na stopień zgodności, towarzyszącego warunkowi rozmytemu.

Na wstępie przypomnijmy, że powiązanie pytania zewnętrznego z wewnętrznym tworzy warunek filtrujący, który może być zapisany w języku SQL na kilka sposobów:

1. <argument> <op> (<pytanie wewnętrzne>),
2. <argument> <op> {ANY|ALL} (<pytanie wewnętrzne>),
3. <argument> [NOT] IN (<pytanie wewnętrzne>),
4. [NOT] EXISTS (<pytanie wewnętrzne >).

Pierwszym problemem, który należało tu rozwiązać, to interpretacja takiego warunku, jeśli jeden lub oba argumenty są wartościami rozmytymi.

Stosunkowo prosto problem ten da się rozwiązać w pierwszym przypadku. Po wykryciu przez interpreter niestandardowych typów danych (rozmytych) następuje bowiem wywołanie

zewnętrznie zdefiniowanej funkcji obsługującej te typy. Funkcja realizująca operator <op> została tak skonstruowana, że pozwala na wyznaczenie stopnia zgodności rozmytych argumentów. Uwzględnienie warunku narzuconego na stopień zgodności, tzn. porównanie jego wartości z wartością progową, wykonywane jest wtedy automatycznie przez interpreter PostgreSQL'a. Ilustracją tego przypadku może pytanie przedstawione już w rozdziale 3.1:

*„Wyszukać nazwy zakładów, które złożyły maksymalne zapotrzebowanie na papier w roku 2003. W rozwiązaniu uwzględnić wiersze, dla których stopień zgodności przekracza 0.9.”*

Zapis tego pytania w opracowanym rozmytym języku SQL ma postać:

```
SELECT nazwa
FROM zaklady z JOIN zapotrzebowanie zp
ON z.nr_zakl = zp.nr_zakl
WHERE zp.rok = '2003' AND (zp.papier ~= (SELECT max(papier)
                                         FROM zapotrzebowanie
                                         WHERE rok = '2003')) > 0.9;
```

Niestety dla wariantów 2) i 3) system PostgreSQL nie dopuszcza do żadnej modyfikacji zapisu operatorów <op>{ANY| ALL} i [NOT] IN. W rezultacie interpreter zapytań sam inicjuje porównanie lewego argumentu z kolejnymi argumentami zbioru wynikowego pytania wewnętrznego. W trakcie tego porównania niedostępna byłaby wartość progowa (wraz z operatorem) umieszczona w ten sposób jak w powyższym przykładzie. Dlatego w tym przypadku zastosowano rozwiązanie nie tak eleganckie jak poprzednio, ale skuteczne, polegające na dołączeniu do prawego lub lewego argumentu rozmytego, wartości progowej wraz z poprzedzającym ją operatorem porównania. Wracając do analizy zainicjowanego porównania należy wyjaśnić, że interpreter PostgreSQL'a, po stwierdzeniu, że typy argumentów nie są standardowymi typami PostgreSQL, szuka zdefiniowanej przez użytkownika funkcji realizującej operację porównania argumentów i przekazuje jej sterowanie. Funkcja ta operuje na dwóch argumentach rozmytych: jednym typowym (ftrapezium) i drugim rozbudowanym, nazwanym ftrapeziumext o następujących polach:

- *wartosc* – typu ftrapezium,
- *operator* – alfanumeryczna reprezentacja operatora relacyjnego,
- *wartoscProgowaStopniaZgodnosci* – typu rzeczywistego.

Dane te pozwalają omawianej funkcji wyznaczyć stopień zgodności rozmytych argumentów, a następnie porównać stopień zgodności z wartością progową.

Na koniec należy jeszcze wyjaśnić jak dołączyć wartość progową (wraz z operatorem porównania) do jednego z argumentów rozmytych?

Wykorzystano tu wspomnianą już możliwość, którą stwarza PostgreSQL, a mianowicie możliwość definiowania własnych operatorów i funkcji je realizujących. Zdefiniowano w tym celu zestaw operatorów o postaci \*<op>, którym towarzyszy wartość progowa,

tzn.  $\ast < op > < \text{wartość progowa} > np. \ast > 0.9$ .

Zastosowanie takiej konstrukcji przedstawia przykład zapytania:

„Wyszukać nazwy zakładów, które złożyły maksymalne zapotrzebowanie na papier w roku 2003. W rozwiązaniu uwzględnić wiersze, dla których stopień zgodności przekracza 0.9.”

Postać tego pytania, zapisana bez używania funkcji agregującej *max*, w rozbudowanym, rozmytym języku SQL jest następująca:

```
SELECT nazwa
FROM zaklady z JOIN zapotrzebowanie zp
ON z.nr_zakl = zp.nr_zakl
WHERE zp.rok = '2003' AND zp.papier >= ALL (SELECT (papier) * >= 0.9
                                           FROM zapotrzebowanie
                                           WHERE rok = '2003');
```

Warunek progowy  $\geq 0.9$  jest tu dołączany do wyszukanych w pytaniu wewnętrznym wartości rozmytych atrybutu *papier*.

Warunek progowy powinien występować w zapisie pytania po wartości rozmytej. W przypadku, gdy w warunku wiążącym po jednej i po drugiej stronie występują wartości rozmyte, położenie warunku progowego jest dowolne.

Nagłówki funkcji zaimplementowanych w języku C i definicje operatorów w SZBD PostgreSQL znajdują się w punkcie 10 załącznika A.

#### 4.12. Ostatecznie przyjęta forma zapisu w języku SQL pytań zawierających wartości rozmyte

W procesie implementacji pojawiły się pewne trudności związane z formą zapisu pytań stosowaną w rozdziale 3. Wynikały one z faktu, iż niektórych stosowanych w rozdziale 3 operatorów np. *jest* nie dało się zaimplementować. Dlatego w ich miejsce zaproponowano inne zapisy symboliczne, które je zastępują. I tak:

- operator *jest* został zaimplementowany w postaci symboli  $\sim=$ ,
- operator *mniej więcej* został zaimplementowany w postaci takich samych symboli  $\sim=$ ,
- operatory *AND* i *OR* zostały zaimplementowane w postaci  $\&\&\&$  i  $\|$ , szerzej zostało to opisane w podrozdziale 4.6.

Format zapisu pytań z użyciem tych operatorów można prześledzić w rozdziale 5, gdzie stosuje się postać ostatecznie przyjętą.

## 5. Przykład implementacji rozmytej bazy danych

Rozdział ten przedstawia zastosowanie elementów rozmytych w przykładowej bazie danych. Stanowi on dodatkowe, praktyczne dopełnienie teoretycznych rozważań przedstawionych w poprzednich rozdziałach pracy. Przy większości elementów rozmytych niniejszego rozdziału, umieszczono informację o odpowiadającym im rozdziale teoretycznym.

Rozdział ten zawiera:

- Opis słowny oraz schemat relacyjny bazy danych *ZAWODNICY*, opracowanej dla potrzeb tego rozdziału,
- Opis wybranych zmiennych lingwistycznych i ich wartości używanych w tym rozdziale,
- Szereg zapytań SQL, o wzrastającym stopniu trudności, wykorzystujących coraz to bardziej zaawansowane elementy rozmyte przedstawione w rozprawie, (w zapisie pytań stosowano ściśle składnię wynikającą z ograniczeń implementacji),
- Pomiar czasu realizacji wybranych zapytań SQL (dla porównania czasu realizacji zapytań rozmytych z czasem realizacji podobnych im zapytań klasycznych (dokładnych)).

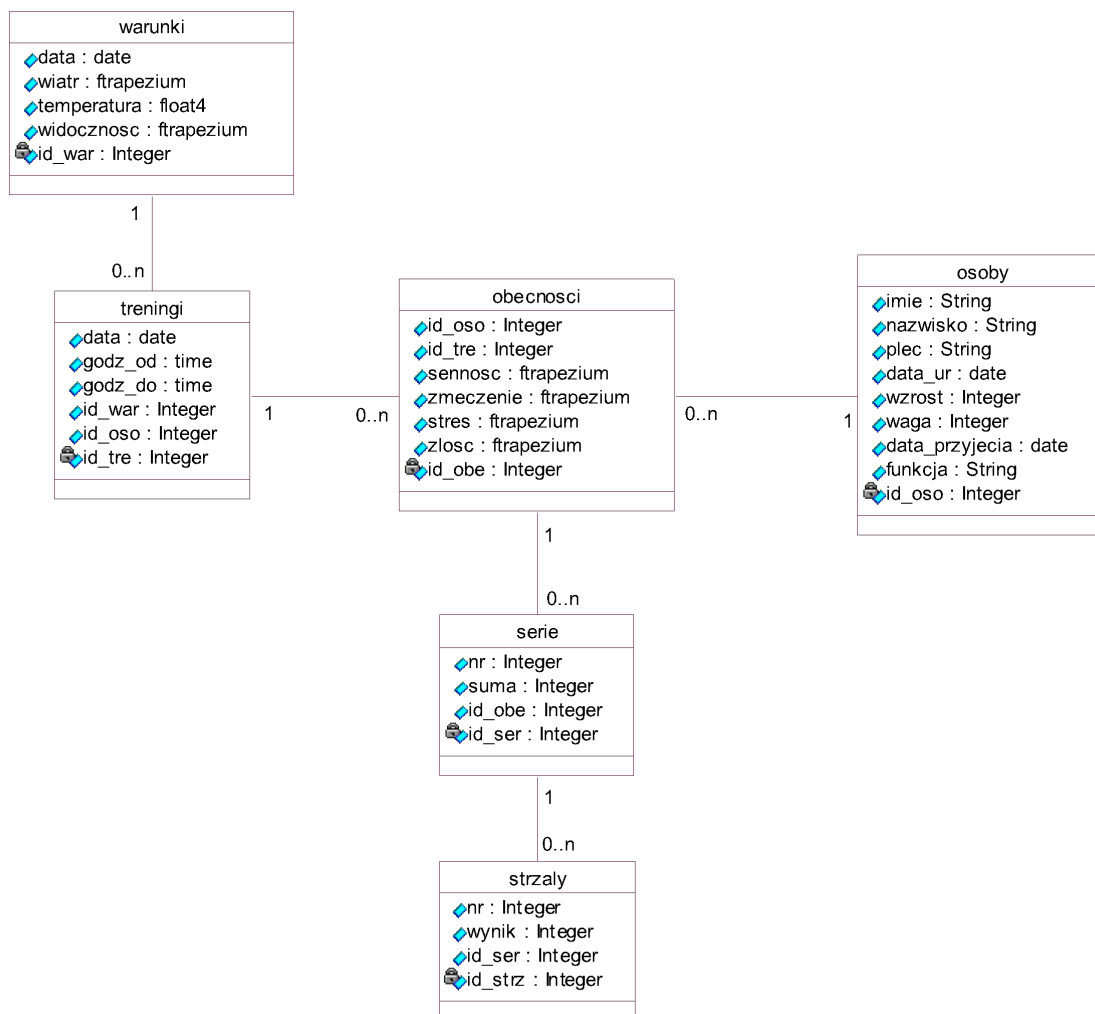
### 5.1. Struktura bazy danych

Opracowana baza danych o nazwie *ZAWODNICY* zawiera dane o sekcji łucznictwa sportowego. I tak odpowiednio tabela:

- **osoby** – przechowuje informacje dotyczące zawodników i trenerów,
- **treningi** – zawiera informacje o treningach,
- **warunki** – zawiera informacje o warunkach pogodowych, jakie były na treningu: temperaturze, wietrze, widoczności. Dane te zawierają wartości rozmyte.
- **obecności** – zawiera informacje o obecnościach trenerów i zawodników na treningach, a także ich dyspozycyjności fizycznej i psychicznej (kolumny: *sennosc*, *zmeczenie*, *stres*, *zlosc*), kolumny te zawierają wartości rozmyte,
- **serie** – przechowuje wyniki serii strzałów zawodników obecnych na treningach. Założono, że na jednym treningu zawodnik może wykonać 10 serii.

- **strzały** – zawiera informacje o strzałach oddanych w ramach serii. Założono, że zawodnicy w ramach jednej serii oddają 5 strzałów, punktowanych od 0 do 10.

Struktura bazy danych *ZAWODNICY* przedstawiona jest na diagramie 5.1.



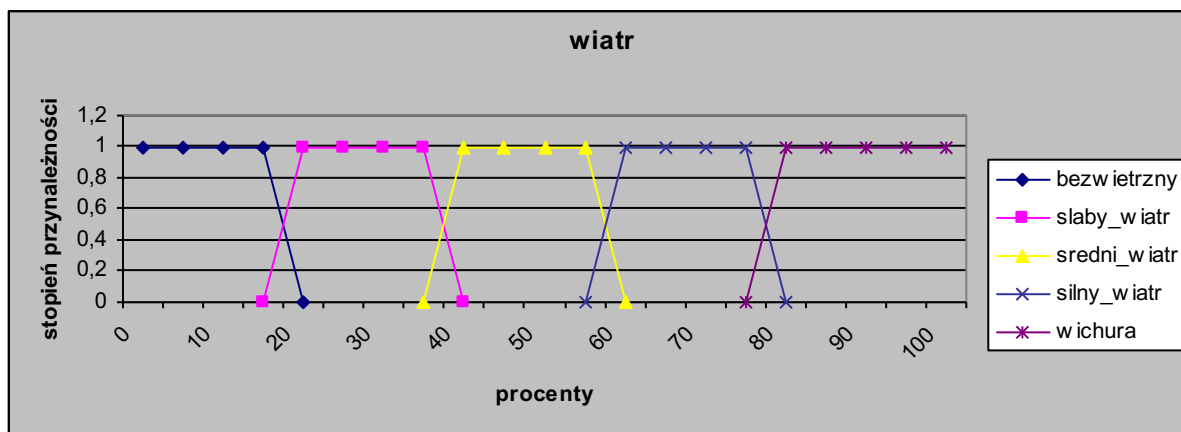
Rys. 5.1 Schemat bazy danych *ZAWODNICY*

Schemat relacyjnej bazy danych *ZAWODNICY* składa się z następujących schematów relacji:

Osoby (id\_oso, imie, nazwisko, plec, data\_ur, wzrost, waga, data\_przyjecia, funkcja),  
 Obecności (id\_obe, id\_tre, id\_oso, **sennosc**, **zmeczenie**, **stres**, **zlosc**);  
 Serie (id\_ser, id\_obe, nr, suma),  
 Strzały (id\_strz, id\_ser, nr, wynik),  
 Treningi (id\_tre, data, godz\_od, godz\_do, id\_oso, id\_war)  
 Warunki (id\_war, data, temperatura, **wiatr**, , **widocznosc**)

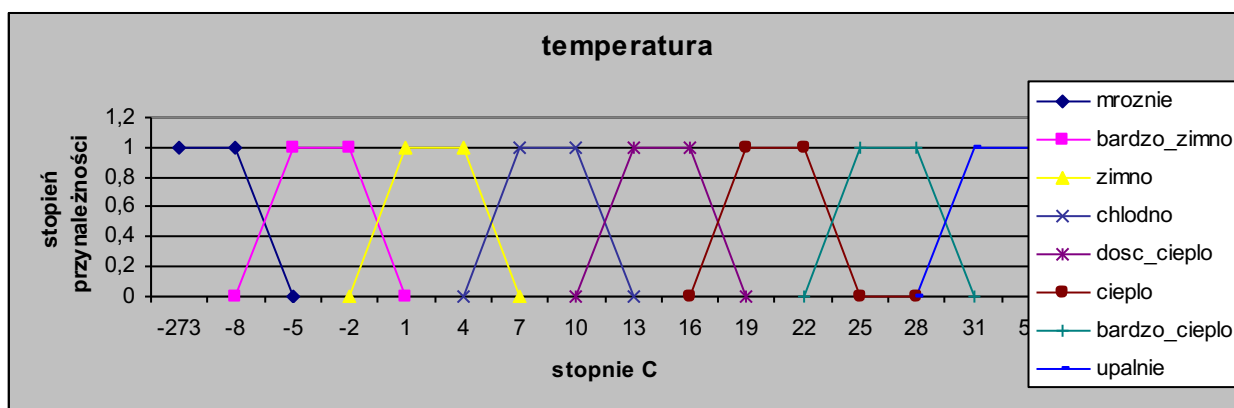
Nazwy kolumn zawierających informacje typu rozmytego (*ftrapezium*) zostały pogrubione.

Kolumny typu rozmytego potraktowane mogą być również jako zmienne lingwistyczne, dla których można zdefiniować wartości lingwistyczne. Przykładowe zmienne lingwistyczne dla bazy danych *ZAWODNICY* oraz zdefiniowane dla nich wartości lingwistyczne przedstawione zostały na wykresach rys. 5.2, 5.3 i 5.4. Dotyczą one warunków atmosferycznych, panujących na treningach.

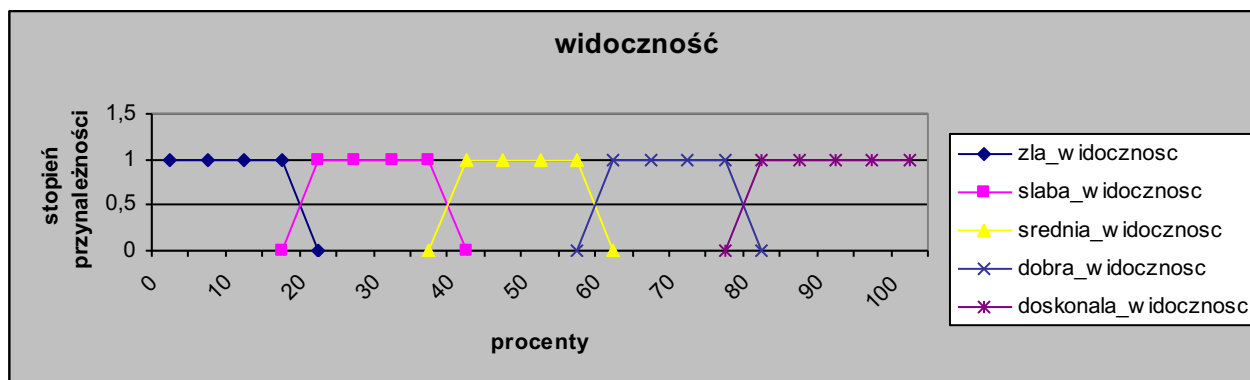


Rys. 5.2 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *wiatr*

Dziedziną zmiennej lingwistycznej *wiatr* jest w tym przypadku siła wiatru wyrażona w procentach, przykładowo w dzień bezwietrzny siła wiatru waha się od 0 do 20 procent maksymalnej siły wiatru.



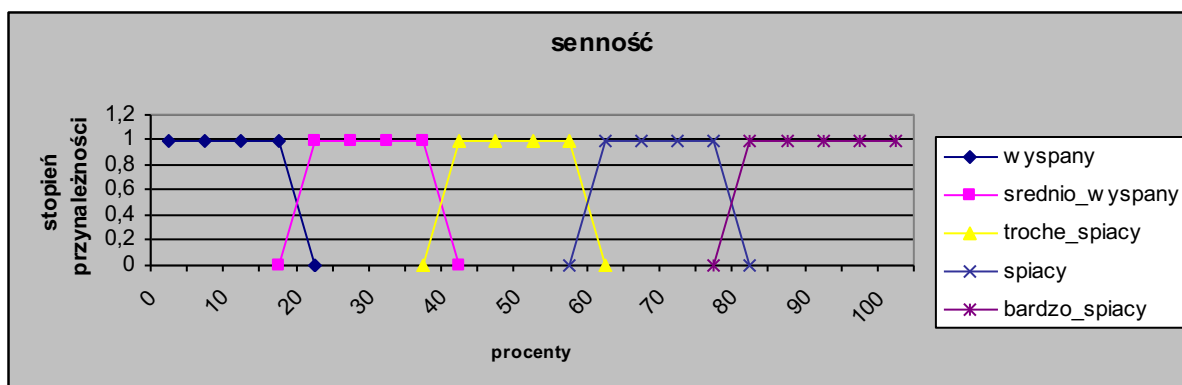
Rys. 5.3 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *temperatura*



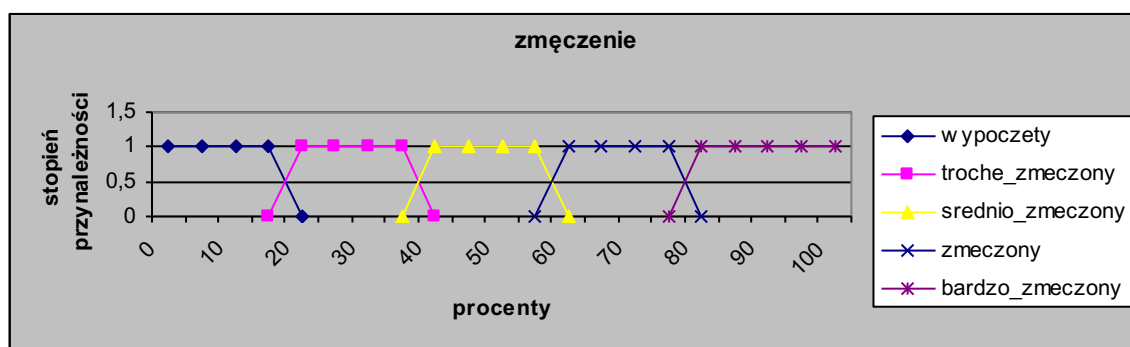
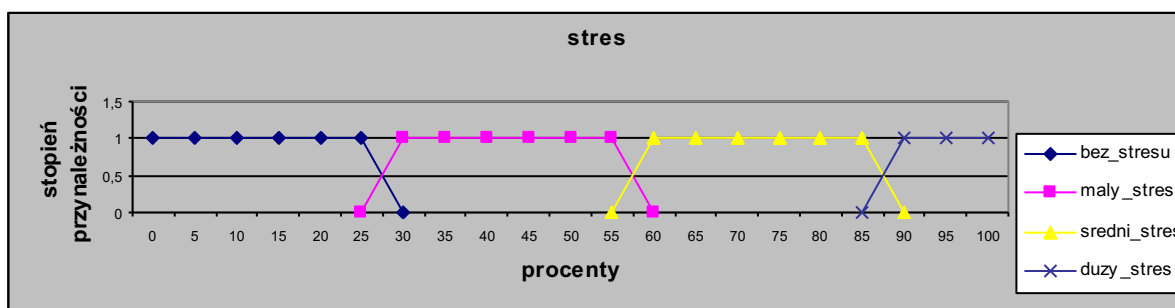
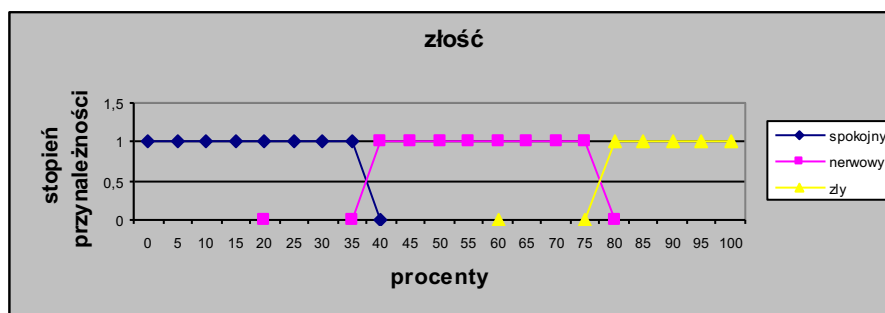
Rys. 5.4 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *widoczność*

Dziedziną zmiennej lingwistycznej *widoczność* jest widoczność wyrażona w procentach, i tak na przykład *doskonała widoczność* to widoczność w ponad 80 procentach.

Zmienne lingwistyczne wraz ze zdefiniowanymi wartościami lingwistycznymi dotyczące cech psychomotorycznych zawodników na poszczególnych treningach ilustrują rysunki: 5.5, 5.6, 5.7 i 5.8. Dziedzina tych zmiennych także wyrażona jest w procentach. Na przykład *bardzo śpiący* to śpiący w ponad 80 procentach, *bardzo zmęczony* to zmęczony w ponad 80 procentach, *przeżywający duży stres*, to stres w ponad 85 procentach itd.

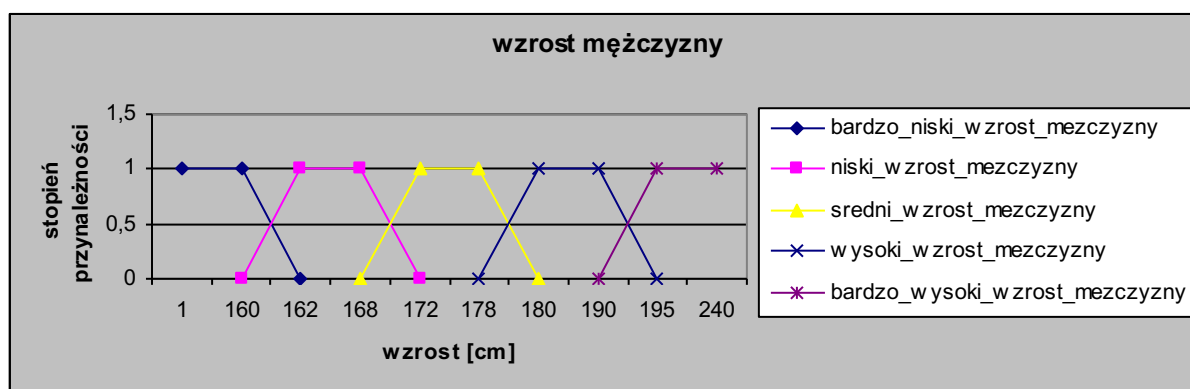


Rys. 5.5 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *senność*

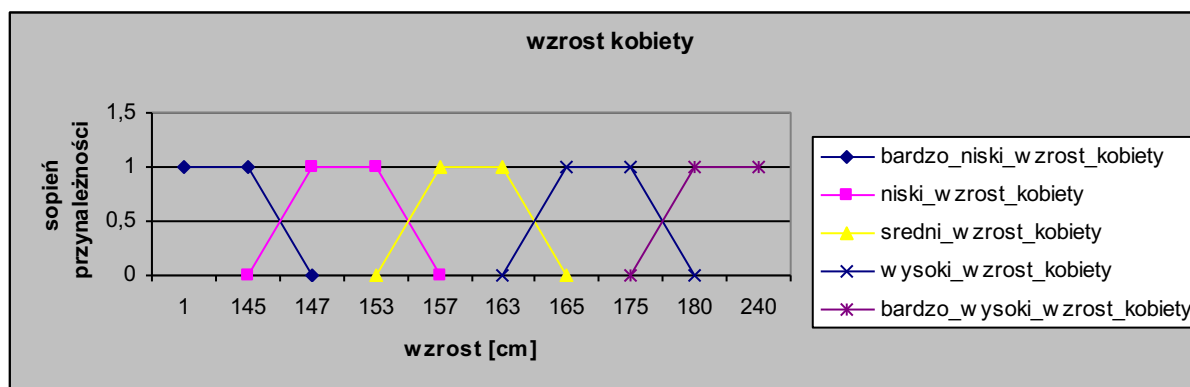
Rys. 5.6 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *zmęczenie*Rys. 5.7 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *stres*Rys. 5.8 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *złość*

Zmienne lingwistyczne wraz ze zdefiniowanymi wartościami lingwistycznymi służące do realizacji porównań z wartościami dokładnymi określającymi wagę i wzrost zawodników ilustrują rysunki: 5.9, 5.10, 5.11, 5.12, 5.13 i 5.14.

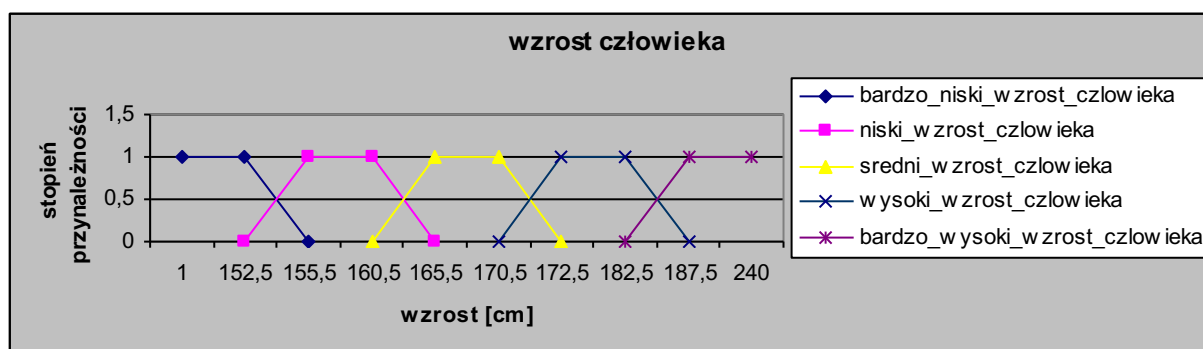




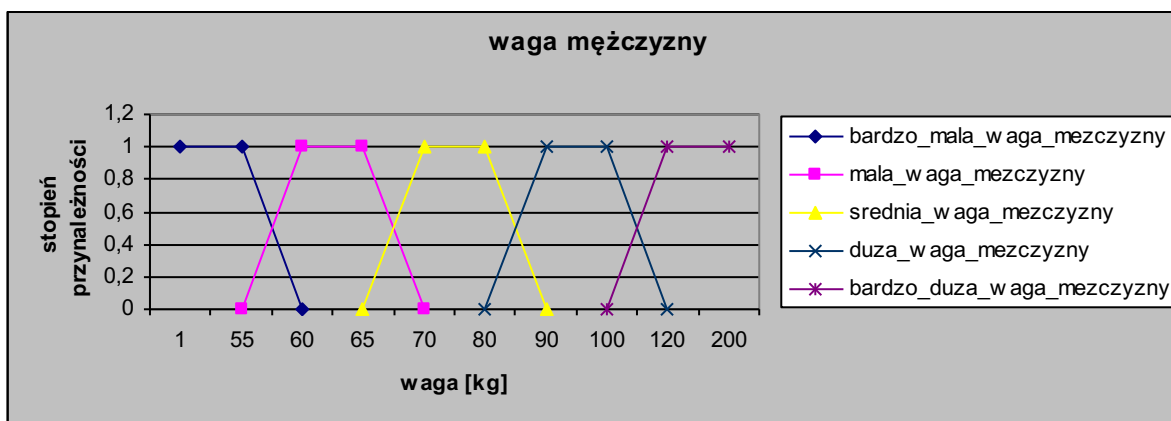
Rys. 5.9 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *wzrost mężczyzny*



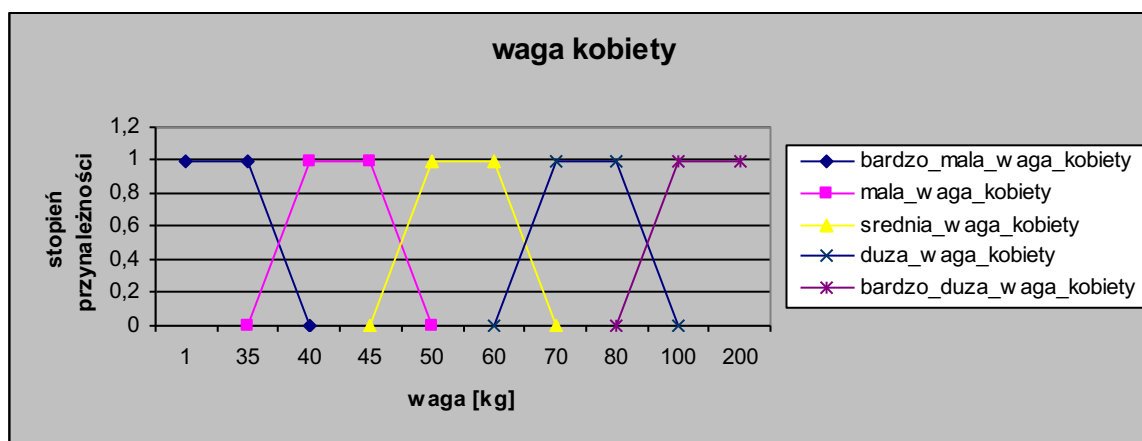
Rys. 5.10 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *wzrost kobiety*



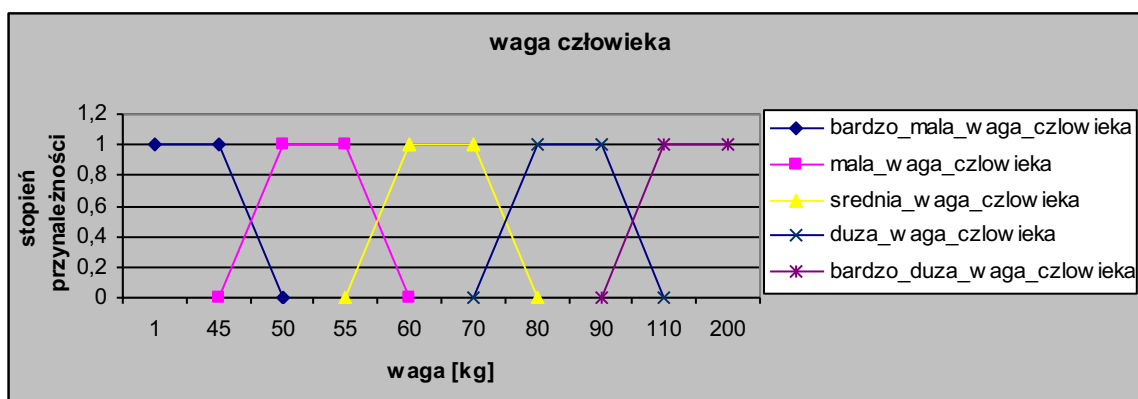
Rys. 5.11 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *wzrost człowieka*



Rys. 5.12 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *waga mężczyzny*

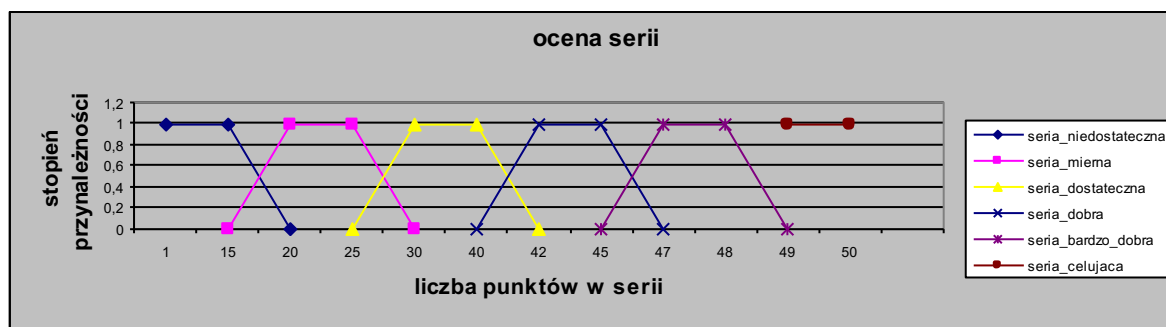


Rys. 5.13 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *waga kobiety*



Rys. 5.14 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *waga człowieka*

Dodatkowo zdefiniowano zmienną lingwistyczną *ocena serii*, a odpowiadające jej wartości lingwistyczne przedstawiono na rysunku rys. 5.15. Zawodnicy na treningu oddają 10 serii po 5 strzałów. Za każdy strzał można zdobyć od 0 do 10 punktów. Zatem w serii można maksymalnie uzyskać 50 punktów.



Rys. 5.15 Funkcje przynależności przykładowych wartości lingwistycznych zmiennej *ocena serii*

Dodatkowo zaimplementowano funkcje konwersji wartości typu ftrapezium do typu alfanumerycznego, w sposób opisany w rozdziale 4. Nazwy przykładowych zmiennych lingwistycznych oraz odpowiednich funkcji konwersji przedstawiono w tabeli 5.1.

Tabela 5.1

Przykładowe zmienne lingwistyczne i odpowiadające im funkcje konwersji

zmienna lingwistyczna	funkcja konwersji
wiatr	wiatr_to_lingw(ftrapezium)
temperatura	temperatura_to_lingw(ftrapezium)
widocznosc	widocznosc_to_lingw(ftrapezium)
sennosc	sennosc_to_lingw(ftrapezium)
zmeczenie	zmeczenie_to_lingw(ftrapezium)
stress	stress_to_lingw(ftrapezium)
zlosc	zlosc_to_lingw(ftrapezium)
wzrost_kobiety	wzrost_kobiety_to_lingw(ftrapezium)
wzrost_mezczyzny	wzrost_mezczyzny_to_lingw(ftrapezium)
wzrost_czlowieka	wzrost_czlowieka_to_lingw(ftrapezium)
waga_kobiety	waga_kobiety_to_lingw(ftrapezium)
waga_mezczyzny	waga_mezczyzny_to_lingw(ftrapezium)
waga_czlowieka	waga_czlowieka_to_lingw(ftrapezium)

## 5.2. Wykorzystanie utworzonych elementów rozmytych w pytaniach SQL

W tym podrozdziale przedstawione zostaną pytania rozmyte, wykorzystujące zaprezentowane elementy rozmyte, kierowane do bazy danych *ZAWODNICY*. Pytania te ze względu na fakt, że będą wykorzystywane w pomiarach czasowych, będą bardziej złożone.

Pytania te będą formułowane według klasyfikacji przedstawionej w rozdziale 3.3.

## 1. Wartości rozmyte we frazie *WHERE*

Do bazy danych *ZAWODNICY* kierowane jest następujące pytanie:

1. „Wyszukaj nazwiska wysokich zawodników średniej wagi, którzy przy słabej widoczności choć raz trafili w dziesiątkę. W odpowiedzi powinny znaleźć się wszystkie wiersze spełniające ze stopniem zgodności większym niż 0 kryteria pytania.”

Postać tego pytania z operatorową formą zapisu warunku na stopień zgodności w języku SQL jest następująca:

```
SELECT DISTINCT os.id_oso, os.imie, os.nazwisko
FROM osoby os
JOIN obecności ob ON os.id_oso = ob.id_oso
JOIN treningi t ON ob.id_tre = t.id_tre
JOIN warunki w ON t.id_war = w.id_war
JOIN serie s ON s.id_obe = ob.id_obe
JOIN strzaly st ON s.id_ser = st.id_ser
WHERE ((os.wzrost ~= wysoki_wzrost_mezczyzny()) &&&
      (os.waga ~= srednia_waga_mezczyzny()) &&&
      (w.widocznosc ~= slaba_widocznosc())) > 0.0
      AND st.wynik = 10;
```

W tym podrozdziale dla skrócenia zapisu używać będziemy zamiast sformułowania *pytanie z operatorową formą zapisu warunku na stopień zgodności* krótszego sformułowania *pytanie w formie operatorowej*.

Kolejne pytanie kierowane do bazy danych *ZAWODNICY* może brzmieć następująco:

2. „Wyszukaj daty treningów wraz z liczbą zawodników, którzy przystępując do treningu byli wyspani lub wypoczęci, zaś w czasie treningów nie denerwowali się (nie odczuwali stresu). W odpowiedzi powinny znaleźć się wszystkie wiersze spełniające ze stopniem zgodności większym niż 0.5 kryteria pytania.”

Pytanie to zapisane w języku SQL w formie operatorowej ma następującą postać:

```
SELECT t.id_tre, count(ob.id_oso)
FROM obecności ob
JOIN treningi t ON ob.id_tre = t.id_tre
WHERE (((sennosc ~= wyspany()) ||| (zmeczenie ~= wypoczety())) &&&
      (stres ~= bez_stresu())) > 0.5
GROUP BY t.id_tre;
```

Dyskusja tego typu pytań została przedstawiona w rozdziale 3.3 a ich implementacja w rozdziale 4.

## 2. Wartości rozmyte we frazie *WHERE* i *HAVING*

Do bazy danych *ZAWODNICY* kierowane jest następujące pytanie:

3. „Wyszukaj zawodników, którzy odbywali dwa pierwsze treningi z małym stresem i mieli na nich chociaż jedną bardzo dobrą serię. W odpowiedzi powinny się znaleźć wiersze spełniające kryteria pytania ze stopniem większym niż 0.”

Pytanie to może zostać wyrażone w języku SQL w formie operatorowej w następujący sposób:

```
SELECT t.id_tre, os.id_oso, os.imie, os.nazwisko, t.data,
       st.id_ser, (sum(st.wynik) ~= seria_bardzo_dobra()),
       sum(st.wynik), (stres ~= maly_stres())
FROM treningi t
      JOIN obecności ob ON ob.id_tre = t.id_tre
      JOIN osoby os ON ob.id_oso = os.id_oso
      JOIN serie se ON ob.id_obe = se.id_obe
      JOIN strzaly st ON se.id_ser = st.id_ser
WHERE t.id_tre in (1,2) AND (stres ~= maly_stres()) > 0.0
GROUP BY t.id_tre, st.id_ser, os.nazwisko, stres, os.id_oso
HAVING (sum(st.wynik) ~= seria_bardzo_dobra()) > 0.0;
```

W powyższym pytaniu oprócz danych zawodnika w zbiorze wynikowym znajdują się również takie informacje jak: identyfikator serii, suma punktów w serii oraz stopnie zgodności z kryteriami *seria\_bardzo\_dobra()* i *maly\_stres()*.

Rozpatrzmy następujące pytanie kierowane do bazy danych *ZAWODNICY*:

4. „Wyszukaj warunki pogodowe panujące w czasie tych treningów, w trakcie których prawie wszyscy zawodnicy byli wyspani ( stopień zgodności z warunkiem rozmytym wyspany powinien być nie mniejszy niż 0.8, a własność prawie wszystkie winna być spełniona ze stopniem zgodności nie mniejszym niż 0.7).”

Postać tego pytania zapisana w języku SQL w formie operatorowej jest następująca:

```
SELECT w.id_war, t.id_tre, temperatura, wiatr, widocznosc
FROM warunki w
      JOIN treningi t ON w.id_war = t.id_war
      JOIN obecności o ON t.id_tre = o.id_tre
GROUP BY w.id_war, t.id_tre, temperatura, wiatr, widocznosc
HAVING (prawie_wszystkie( sennosc ~= wyspany() ) >= 0.8 ) >= 0.7 ) ;
```

W powyższych pytaniach elementy rozmyte występują we frazie *WHERE* i we frazie *HAVING*. W pytaniach tych występują także rozmyte kwantyfikatory (*prawie wszyscy*), oraz funkcje lingwistyczne (*maly\_stres()*, *wyspany()*), jak również wartości rozmyte we frazie *GROUP BY*, choć temat ten jest rozwijany w następnym punkcie.

### 3. Grupowanie względem wartości rozmytych

Do bazy danych *ZAWODNICY* kierowane jest następujące pytanie:

5. „Wyszukaj średnią liczbę punktów uzyskanych we wszystkich seriach dla poszczególnych zdefiniowanych przedziałów wartości siły wiatru.”

Postać tego pytania wyrażona w języku SQL w formie operatorowej może być następująca:

```
SELECT avg(suma), wiatr_to_lingw(w.wiatr)
FROM serie se
      JOIN obecności ob ON se.id_obe = ob.id_obe
      JOIN treningi t ON ob.id_tre = t.id_tre
      JOIN warunki w ON t.id_war = w.id_war
GROUP BY w.wiatr ;
```

Inne pytanie tego typu kierowane do bazy danych *ZAWODNICY* może brzmieć następująco:

6. „Określ wpływ zmęczenia zawodników na celność ich strzałów. (Wyznacz średnią liczbę punktów uzyskaną dla poszczególnych stanów zmęczenia).”

Pytanie to zapisane w języku SQL w formie operatorowej ma następującą postać:

```
SELECT id_oso, avg(st.wynik), zmeczenie_to_lingw(ob.zmeczenie)
FROM obecności ob
      JOIN serie se ON ob.id_obe = se.id_obe
      JOIN strzaly st ON se.id_ser = st.id_ser
GROUP BY ob.zmeczenie, id_oso, id_tre ;
```

Zastosowanie elementów rozmytych występujących w pytaniach tej grupy zostało przedstawione w rozdziale 3, a implementacja w rozdziale 4.

W przedstawionych powyżej pytaniach zastosowano grupowanie rozmytych danych, omówione w podrozdziale 3.5.2 (najbardziej restrykcyjne, oparte na równości wszystkich parametrów charakteryzujących wartość rozmytą).

### 4. Wartości rozmyte w pytaniach zagnieżdżonych

W punkcie tym przedstawione zostaną rozmyte pytania zagnieżdżone.

#### – Wartości rozmyte w podzapytaniach

Do bazy danych *ZAWODNICY* kierowane jest pytanie:

7. „Wyszukaj nazwiska zawodników, którzy w poszczególnych treningach mieli najwięcej dobrych serii. W odpowiedzi powinny znaleźć się wiersze ze stopniem zgodności przekraczającym 0.”

Pytanie to zapisane w postaci zagnieżdżonej w formie operatorowej może mieć następującą postać:

```
SELECT obl.id_tre, imie, nazwisko, count(*)
FROM obecności obl JOIN serie sel ON obl.id_obe = sel.id_obe
      JOIN osoby os1 ON obl.id_oso = os1.id_oso
```

```

WHERE (se1.suma ~= seria_dobra()) > 0.0
GROUP BY ob1.id_tre, ob1.id_obe, os1.id_oso, imie, nazwisko
HAVING count (*) = (SELECT max(p.liczba) FROM
                    (SELECT ob3.id_tre as tre, count(*) AS liczba
                     FROM obecności ob3
                     JOIN serie se3 ON ob3.id_obe = se3.id_obe
                     WHERE (se3.suma ~= seria_dobra()) > 0.0
                     GROUP BY ob3.id_tre, ob3.id_oso ) AS p
                    WHERE p.tre = ob1.id_tre);

```

Do bazy danych *ZAWODNICZY* sformułowane jest następujące pytanie:

8. „Wyszukaj treningi, w których liczba bardzo dobrych serii była większa niż dobrych. W rozwiązaniu należy uwzględnić wiersze ze stopniem zgodności przekraczającym 0.8.”

Pytanie to zapisane w języku SQL w formie operatorowej ma następującą postać:

```

SELECT b.id_tre , count(*) FROM
  (SELECT t.id_tre, sum(st.wynik)
   FROM treningi t
   JOIN obecności ob ON ob.id_tre = t.id_tre
   JOIN osoby os ON ob.id_oso = os.id_oso
   JOIN serie se ON ob.id_obe = se.id_obe
   JOIN strzaly st ON se.id_ser = st.id_ser
   GROUP BY t.id_tre, se.id_ser
   HAVING (sum(st.wynik) ~= seria_bardzo_dobra()) > 0.8) b
GROUP BY b.id_tre
HAVING count(*) >
  (SELECT count(*) from
   (SELECT t.id_tre, sum(st.wynik)
    FROM treningi t
    JOIN obecności ob ON ob.id_tre = t.id_tre
    JOIN osoby os ON ob.id_oso = os.id_oso
    JOIN serie se ON ob.id_obe = se.id_obe
    JOIN strzaly st ON se.id_ser = st.id_ser
    GROUP BY t.id_tre, se.id_ser
    HAVING (sum(st.wynik) ~= seria_dobra()) > 0.8) a
   WHERE a.id_tre = b.id_tre) ;

```

W przedstawionych pytaniach elementy rozmyte występują we frazach *WHERE* i *HAVING* pytania zewnętrznego i wewnętrznego (rozdział 3.6).

#### – Wartości rozmyte w warunku łączącym

Rozważmy następujące pytanie:

9. „Wyszukaj treningi przeprowadzone w najslabszej widoczności, na których choć jeden zawodnik trafił dziesiątkę.”

Zagnieżdżona postać tego pytania wyrażona w języku SQL w formie operatorowej jest następująca:

```
SELECT DISTINCT t.id_tre, t.data, widocznosc_to_lingw(widocznosc)
FROM treningi t JOIN warunki w ON t.id_war = w.id_war
      JOIN obecności ob ON ob.id_tre = t.id_tre
      JOIN serie se ON ob.id_obe = se.id_obe
      JOIN strzaly st ON st.id_ser = se.id_ser
WHERE st.wynik = 10 AND widocznosc <= all
      (SELECT widocznosc
       FROM treningi t
        JOIN warunki w ON t.id_war = w.id_war) ;
```

W pytaniu tym w warunku wiążącym występuje element rozmyty (rozdział 3.6).

Inne pytanie tej klasy może brzmieć następująco:

*10. „Wyszukaj wyniki i cechy psychomotoryczne (senność, zmęczenie, stres) uczestników treningów, w czasie których wiatr był taki jak wiatr na najmniej udanym treningu. W odpowiedzi powinny się znaleźć wiersze spełniające kryteria pytania ze stopniem zgodności przekraczającym 0.5. ”*

Zagnieżdżona postać tego pytania w formie operatorowej jest następująca:

```
SELECT DISTINCT os.id_oso, os.imie, sennosc_to_lingw(sennosc),
      zmeczenie_to_lingw(zmeczenie), stres_to_lingw(stres)
FROM osoby os
      JOIN obecności ob ON os.id_oso = ob.id_oso
      JOIN treningi t ON ob.id_tre = t.id_tre
      JOIN warunki w ON t.id_war = w.id_war
WHERE w.wiatr ~= ( SELECT w.wiatr FROM obecności ob
                  JOIN treningi t ON ob.id_tre = t.id_tre
                  JOIN warunki w ON t.id_war = w.id_war
                  JOIN serie s ON s.id_obe = ob.id_obe
                  JOIN strzaly st ON s.id_ser = st.id_ser
                  GROUP BY t.id_tre, w.wiatr
                  HAVING sum(st.wynik) <= all
                      (SELECT sum(st.wynik) FROM obecności ob
                       JOIN serie s ON s.id_obe = ob.id_obe
                       JOIN strzaly st ON s.id_ser = st.id_ser
                       GROUP BY ob.id_tre) ) > 0.5 ;
```

Rozważmy następujące pytanie:

*11. „Wyszukaj daty tych treningów, na których odsetek wyspanych zawodników jest największy.”*



W pytaniu tym wyznaczamy odsetek wyspanych zawodników, pośrednio występują w warunku także wartości rozmyte.

Zagnieżdżona postać tego pytania zapisana w języku SQL w formie operatorowej może być wyrażona w następujący sposób:

```
SELECT t.id_tre, odsetek((sennosc ~= wyspany())> 0.5)
FROM treningi t
      JOIN obecności o ON t.id_tre = o.id_tre
GROUP BY t.id_tre
HAVING odsetek((sennosc ~= wyspany())> 0.5) >= all
      (SELECT odsetek((sennosc ~= wyspany())> 0.5)
       FROM obecności o
       GROUP BY o.id_tre);
```

#### – wartości rozmyte w warunku korelacji

Do bazy danych *ZAWODNICY* kierowane jest następujące pytanie:

12., „Wyszukaj odsetek zawodników mających indywidualne (różne od innych) zmęczenie na treningu.”

Zagnieżdżona postać tego pytania jest następująca:

```
SELECT
  (SELECT count(*) from
    (SELECT DISTINCT id_oso FROM obecności ob
     WHERE NOT EXISTS (SELECT * FROM obecności ob1
                       WHERE ob.id_obe <> ob1.id_obe
                       AND ob.id_tre = ob1.id_tre AND
                       (ob.zmęczenie ~= ob1.zmęczenie) > 0.9)) ala)
  /float8(count(*))
FROM osoby ;
```

Konstrukcja tego pytania jest dość nietypowa. Pytanie, którego wynik wykonania nazwano *ala* zwraca w odpowiedzi zbiór numerów zawodników mających różne od innych zmęczenie na treningu. Kolejne pytanie wylicza liczbę tych zawodników, najbardziej zewnętrzne wyznacza stosunek (odsetek) liczby zawodników, którzy mają indywidualne zmęczenie do wszystkich zawodników.

Podsumowując należy podkreślić, że zaprezentowane pytania zostały zapisane w postaci realizowanej w rozbudowanym, rozmytym SZBD PostgreSQL, a więc z uwzględnieniem wszystkich ograniczeń implementacyjnych. Nie wydaje się, aby te ograniczenia zmniejszały czytelność zapisu pytań w języku SQL.

Przedstawione w tym rozdziale pytania zostały wykorzystane w pomiarach czasu wykonania zapytań rozmytych. Wyniki przeprowadzonych badań przedstawione są w następnym podrozdziale.

### 5.3. Pomiary czasowe

W tym podrozdziale przedstawimy czasy wykonania zapytań zamieszczonych w rozdziale 5.2. Dla porównania zmierzono również czasy wykonania takich samych dokładnych zapytań.

W tym celu utworzono alternatywną bazę danych o tej samej strukturze, w której wartości rozmyte zastąpiono wartościami dokładnymi, odpowiadającymi wartości modalnej liczby rozmytej. Zmodyfikowano również pytania, zastępując operatory rozmyte, operatorami dokładnymi. W rezultacie, wykonanie pytań dokładnych, prowadziło do uzyskania takich samych zbiorów wierszy jak dla pytań rozmytych.

Rozmiar tabel bazy danych *ZAWODNICY*, na której oparte były zapytania przedstawiony jest w tabeli 5.2.

Tabela 5.2

Rozmiary poszczególnych tabel

nazwa tabeli	liczba wierszy
Osoby	12
Obecności	100
Serie	1000
Strzały	5000
Treningi	10
Warunki	10

Celem tych pomiarów było wyznaczenie czasów wykonania zapytań rozmytych i odpowiadających im zapytań dokładnych.

Dla zagwarantowania miary średniej pomiary były wykonywane w następujący sposób:

- dwudziestokrotnie zmierzono czas wykonywania zapytania,
- uśredniono pomiary, stosując średnią arytmetyczną, uzyskane wyniki przedstawione są w tabeli 5.3.

Numery wierszy odpowiadają numerom pytań w rozdziale 5.2. Wykorzystane skróty oznaczają odpowiednio:

- w&w** – warunki rozmyte we frazie where,
- w&h** – warunki rozmyte we frazie where & having,
- g** – grupowanie według wartości rozmytych,
- pz** – pytania zagnieżdżone.

Tabela 5.3

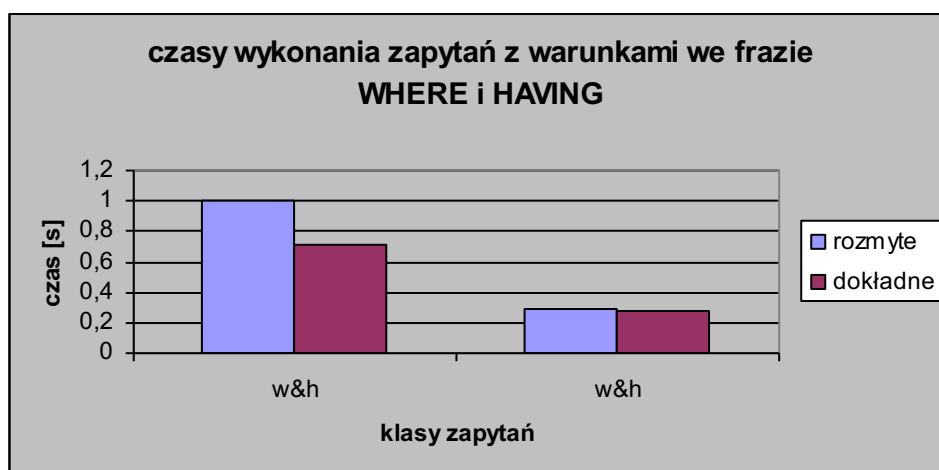
Czasy wykonania zapytań (w [s])

lp	grupa pytań	pytanie rozmyte	pytanie dokładne
1	w&w	1.794	0.316
2	w&w	0.264	0.151
3	w&h	1.003	0.711
4	w&h	0.294	0.279
5	g	1.206	1.124
6	g	3.502	3.124
7	pz	5.491	3.693
8	pz	5.331	5.364
9	pz	0.951	0.767
10	pz	0.202	0.258
11	pz	6.137	5.892
12	pz	0.299	0.272

Uzyskane wyniki zostały zilustrowane wykresami, które podzielono również według wyodrębnionych grup zapytań.



Rys. 5.16 Wykresy czasów wykonania zapytań z wartościami rozmytymi we frazie *where* i odpowiadających im pytań dokładnych



Rys. 5.17 Wykresy czasów wykonania zapytań z wartościami rozmytymi we frazie *where* i *having* oraz odpowiadających im pytań dokładnych



Rys. 5.18 Wykresy czasów wykonania zapytań grupujących względem wartości rozmytych i odpowiadających im zapytań dokładnych



Rys. 5.19 Wykresy czasów wykonania rozmytych zapytań zagnieżdżonych i odpowiadających im dokładnych zapytań zagnieżdżonych

Jak widać z zamieszczonych wykresów, rozmyte pytania SQL wykonują się nieco dłużej niż pytania dokładne. Wyniki te są zgodne z oczekiwaniami autorki, gdyż w czasie porównywania wartości rozmytych wykonywanych jest znacznie więcej niezbędnych operacji matematycznych, pozwalających na wyznaczenie właściwej odpowiedzi.

Różnice te nie są jednak zbyt znaczące, największe da się zauważyć w tych pytaniach, gdzie wartości rozmyte występują we frazie *WHERE* i *HAVING*. Zwłaszcza w tym, w którym jest używanych kilka wartości rozmytych, połączonych spójnikiem rozmytym *&&&*. W przykładzie tym dla każdego warunku rozmytego wyznaczany jest stopień zgodności, później wyznaczany całkowity stopień zgodności wiersza, na końcu wybierane są wiersze o całkowitym stopniu zgodności większym niż zadana wartość progowa. W tym przypadku czas wykonania zapytania rozmytego jest dłuższy o czas wykonywanych działań matematycznych.

## 6. Podsumowanie i wnioski

Celem rozprawy było zastosowanie teorii zbiorów rozmytych do rozszerzenia możliwości relacyjnych baz danych.

W pierwszej części pracy zwrócono jednak uwagę również na inne metody aproksymacyjnego wyszukiwania: metody wyszukiwania wektorowego oraz wyszukiwania uwzględniającego prawdopodobieństwo spełnienia kryteriów zapytania.

**Główną uwagę w badaniach przedstawionych w pracy skupiono na wykorzystaniu teorii zbiorów rozmytych w wyszukiwaniu informacji w bazach danych, a szczególnie na wykorzystaniu elementów teorii zbiorów rozmytych w zapytaniach języka SQL.**

W rozdziale 2 przedstawiono podstawowe pojęcia dotyczące relacyjnych baz danych i zbiorów rozmytych.

Główne rezultaty uzyskane w pracy przedstawiono w rozdziałach 3 i 4.

Rozdział 3 otwiera analiza istoty wnioskowania rozmytego w procesie interpretacji warunków filtrujących w zapytaniach formułowanych w języku SQL. Następnie przedstawiono propozycje sposobu wprowadzenia do zapisu pytań SQL-owych warunku na stopień zgodności wiersza z kryteriami zapytania. Kolejne rozwiązane problemy to analiza możliwości i propozycje nowych metod rozmytego grupowania danych. Ostatnimi analizowanymi w rozdziale 3 zagadnieniami są kwantyfikatory rozmyte oraz interpretacja rozmytych zapytań zagnieżdżonych z uwzględnieniem problemu zapisu w tych pytaniach warunku na stopień zgodności.

Rozdział 4 poświęcono implementacji mechanizmów rozmytych w wybranym systemie zarządzania bazą danych. Pierwszym, podstawowym dla całości zagadnienia była implementacja rozmytego typu danych. Kolejno przedstawiono implementację wszystkich rozwiązań przedstawionych w rozdziale 3. Wszystkie mechanizmy udało się zaimplementować nie zmieniając kodu źródłowego systemu PostgreSQL, a jedynie wprowadzając własne typy danych wraz z obsługującymi je funkcjami i operatorami.

W końcowym rozdziale 5 przedstawiono zestaw kilkunastu złożonych zadań rozmytego wyszukiwania danych i zaprezentowano zapis tych wszystkich rozmytych pytań przy wykorzystaniu mechanizmów zaproponowanych w pracy. Przeprowadzono też eksperyment z pomiarem i porównaniem czasu wykonania zapytań rozmytych i dokładnych.

W załącznikach pracy przedstawiono wyniki eksperymentu z grupowaniem danych, ale przede wszystkim najważniejsze elementy dokumentacji oprogramowania implementującego mechanizmy rozmyte w SZBD PostgreSQL.

Do autorskich, najbardziej interesujących rozwiązań przedstawionych w rozprawie należą:

- opracowanie własnych algorytmów interpretacji pytań języka SQL, w szczególności zagnieżdżonych z warunkami rozmytymi, przy uwzględnieniu warunków nakładanych na stopień zgodności,
- opracowanie własnych metod rozmytego grupowania danych,
- wprowadzenie kwantyfikatorów rozmytych, działających na grupach wierszy,
- implementacja mechanizmów rozmytych w systemie zarządzania bazą danych PostgreSQL.

Implementacja wyszukiwania rozmytego w systemie PostgreSQL pozwoliła pozytywnie zweryfikować wyniki analizy teoretycznej wraz z proponowanymi mechanizmami i algorytmami.

Opracowany system umożliwia tworzenie rozmytych baz danych (przechowujących dane typów rozmytych), wyszukiwanie rozmytej informacji, jak też formułowanie rozmytych zapytań odnoszących się do klasycznych (nierozmytych) typów danych.

Przedstawione wnioski pozwalają na stwierdzenie, że cel pracy został osiągnięty, a teza pracy udowodniona.

Tworzone w języku C typy i funkcje są uniwersalne, a definiowanie funkcji w języku SQL można w prosty sposób przenieść do innego SZBD (np. MS SQL Server czy Oracle). System ten tworzy platformę do dalszych badań nad rozmytymi bazami danych (planowana jest między innymi kontynuacja prac nad algorytmami grupowania rozmytego). Jest dość uniwersalnym narzędziem, które można nadal rozwijać wzbogacając je o nowe elementy.

## LITERATURA

- [Bdr99] Badurek Z.: *Logika rozmyta w bazach danych*. Informatyka. Styczeń 1999.
- [BklPtr95] Buckles B.P., Petry F.E.: *Fuzzy databases in the New Era*. ACM 1995.
- [Bnch98] Banachowski L.: *Bazy Danych – Tworzenie aplikacji*, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1998r.
- [BnnDv98] Beynon-Davies P.: *Systemy baz danych*. Wydawnictwa Naukowo-Techniczne. Warszawa 1998.
- [Bnt92] Bentley J.: *Perelki oprogramowania*, Wydawnictwa Naukowo-Techniczne, Warszawa 1992.
- [BrdPs00a] Bordogna G., Pasi G.: *Recent Issues on Fuzzy Databases*. A Springer-Verlag Company. Niemcy Heidelberg 2000r.
- [BrdPs00b] Bordogna G., Pasi G.: *Modeling Vagueness in Information Retrieval*. Springer-Verlag Berlin Heidelberg 2000.
- [BrdPs94] Bordogna G., Pasi G.: *A fuzzy query language with a linguistic hierarchical aggregator*. ACM 1994.
- [BscDbsPrd94] Bosc P., Dubois D., Prade H.: *Fuzzy Functional Dependencies An Overview And A Critical Discussion* Flexible Query-Answering Systems. 1994.
- [BscDbsPvrPrd] Bosc P., Dubois. D, Pivert O., Prade H.: *Flexible queries in relational databases – the example of the division operator*.
- [BscLtrPvr95] Bosc P., Lietard L., Pivert O.: *Quantified Statements in a flexible relational query language*. Proceedings of the 1995 ACM symposium on Applied computing, February 26-28, 1995, Nashville, TN, USA. ACM 1995. p.488-492.
- [BscPvr94] Bosc P., Pivert O.: *Fuzzy queries and Relational Databases* Proceedings of the 1994 ACM Symposium on Applied Computing, March 6-8, 1994, Phoenix, AZ, USA. ACM 1994. 170-174.
- [BscPvr95] Bosc P., Pivert O.: *SQLf: A Relational Database Language for Fuzzy Querying*. IEEE Transactions on Fuzzy Systems. Vol. 3, Nr



- 1, luty 1995.
- [CbrMdnPnsVl95] Cubero J. C., Medina J. M., Pons O., Vila M. A.: *Rules discovery in fuzzy relational databases*. NAFIPS'95. Maryland (USA). IEEE computer Society Press. 414-419, 1995.
- [CbrMdnPnsVl97a] Cubero J. C., Medina J. M., Pons O., Vila M.A.: *Data summarization with linguistic labels: a loss less decomposition approach*. Proc. of IFSA'97 V. II, 186-191. Praga. 1997.
- [CbrMdnPnsVl97b] Cubero J. C., Medina J. M., Pons O., Vila M. A.: *Extensions of a resemblance relation*. Fuzzy Sets and Systems. 86(2). 1997.
- [CbrMdnPnsVl98] Cubero J. C., Medina J. M., Pons O., Vila M. A.: *Fuzzy loss less decompositions in databases*. Fuzzy Sets and Systems. V.97-2 145-167, Holanda. 1998.
- [CbrMdnPnsVl99a] Cubero J. C., Medina J. M., M.A., Pons O., Vila M.A.: *Non Transitive Fuzzy Dependencies (I)*. Fuzzy Sets and Systems V.106 401-431, Holanda. 1999.
- [CbrMdnPnsVl99b] Cubero J. C., Medina J. M., Pons O., Vila M.A.: *Transitive Fuzzy Dependencies(II)*. Fuzzy Sets and Systems V.106 pp.433-438, Holanda. 1999.
- [CbrMdnPnsVl99c] Cubero J. C., Medina J. M., Pons O., Vila M.A.: *Data Summarization in Relational Databases through Fuzzy Dependencies*. Information Sciences V. 121 pp.233-270. 1999
- [CbrMdnVl93] Cubero J. C., Medina J. M., Vila M. A.: *Influence of Granularity Level in Fuzzy Functional Dependencies*. Lectures Notes in Computer Sciences, 747. Springer Verlag, pp.73-78.1993.
- [CbrPnsVl94] Cubero J. C., Pons O., Vila M. A.: *Weak and strong Resemblance in Fuzzy Functional Dependencies*. IEEE'94 International Conference, Florida, 1994.
- [CbrVl94] Cubero J. C., Vila M. A.: *A new definition of fuzzy functional dependency in fuzzy relational databases*. International Journal of Intelligent Systems, 9(5),.441-448, 1994.
- [ChcnŁsk01] Pod redakcją Chojcana J., Łęskiego J.: *Zbiory rozmyte i ich zastosowanie. Praca dedykowana Profesorowi E. Czogale*. Wydawnictwo Politechniki Śląskiej. Gliwice 2001.
- [ChnChn02] Chen S.-M., Chen H.-H. *Fuzzy query processing in the distributed relational databases environment*. Database and Data Communication Network Systems, Vol.1, Elsevier Science (USA) 2002.

- [Clk99] Celko J.: *SQL zaawansowane techniki programowania*, MIKOM, Warszawa, listopad 1999.
- [Czg97] <http://kalejdoskop.iele.polsl.gliwice.pl/1997/czogała/>
- [CzgŁsk00] Czogała E., Łęski J.: *Fuzzy and neuro-fuzzy intelligent systems*. Heiderbrg: New York: Phisical-Verlag, 2000.
- [CzgPdr85] Czogała E., Pedrycz W.: *Elementy i metody teorii zbiorów rozmytych*. Wydawnictwa Naukowo-Techniczne. Warszawa 1985.
- [Dbk01] Dybikowski Z.: *PostgreSQL*, Helion. 2001.
- [DbsPrd] Dubois. D, Prade H.: *Valid or Complete information in databases – a possibility Theory-based analysis*.
- [DbsPrd88] Dubois D., Prade H.: *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. Plenum Press: New York, 1988.
- [DbsPrd96] Dubois D., Prade H.: *Using fuzzy sets in flexible querying: Why and how?* Query-Answering Systems. 1996.
- [DcmQr00] *Document and Queries*. Adres internetowy:  
<http://www.coe.uncc.edu/~eelkwae/CSC12170Spring2000>
- [DlbAdb89] Delobel C., Adiba M.: *Relacyjne bazy danych*. WNT, 1989.
- [Dt95] Date C. J.: *An Introduction to Database Systems*. Addison-Wesley Publishing Company 1995.
- [Dt95] Date C. J.: *An Introduction to Database Systems*. Addison-Wesley Publishing Company 1995.
- [ElmNvt00] Elmasri R., Navathe S. B.: *Fundamentals of Database Sytems*. Addison-Wesley Publishing Company, World Student Series. 2000
- [FltKng95] Faloutsos C., King-Ip L.: *FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualisation of Traditional and Multimedia Datasets*. Technical Report Univercity of Maryland, 1995, TR-520.
- [GlnMdnPnsCbr98] Galindo J., Medina J. M., Pons O., Cubero J. C.: *A Server for Fuzzy SQL Queries*. Springer-Verlag Berlin Heiderberg, 1998.
- [IntMkd01] Intan R., Mukaidono M.: *Conditional probability relations in fuzzy relational database*. Springer-Verlag, Berlin Heidelberg. 2001.
- [Kcp01] Kacprzyk J.: *Wieloetapowe sterowanie rozmyte*. Wydawnictwa Naukowo Techniczne. Warszawa. 2001.
- [KcpSzt01a] Kacprzyk J., Szmidt E.: *Intiutionistic Fuzzy Sets in Some Medical Applications*. Springer-Verlag Berlin Heidelberg, 2001.
- [KcpSzt01b] Kacprzyk J., Szmidt E.: *Intiutionistic Fuzzy Sets in Intelligent Data*

- Analysis for Medical Diagnosis*. Springer-Verlag Berlin Heidelberg, 2001.
- [KcpZdr96] Kacprzyk J., Zadrozny S.: *FQUERY for Access: Towards human consistent querying user interface*. ACM 0-89791-820-7, 1996.
- [KcpZdr97] Kacprzyk J., Zadrozny S.: *Computing with words in intelligent database querying: standalone and Internet-based applications*. 1997.
- [KcpZdr98] Kacprzyk J., Zadrozny S.: *On summarization of large data sets via a fuzzy-logic-based querying add-on to Microsoft Access*. Intelligent Information Systems VII. Proceedings of the Workshop. Malbork czerwiec 1998r.
- [KcpZlk86] Kacprzyk J., Ziolkowski A.: *Database Queries with Fuzzy Linguistic Quantifiers*. IEEE Transactions of Systems, Man, and Cybernetics. Vol. SMC-16, Nr 3, maj/czerwiec 1986.
- [Klt01] Kolatch E.: *Clustering Algorithms for Spatial Databases: a Survey*. Dept. of Computer Science. University of Maryland. College Park. 2001.
- [Kzl03] Kozielski S.: *Wykłady z baz danych*. Gliwice. 2003.
- [Lrs99] Larsen H.: *An Approach to flexible information access systems using soft computing*. International Conference on System Sciences, Hawaje 1999.
- [LtnŻkw84] Leitner R., Żakowski W.: *Matematyka dla kandydatów na wyższe uczelnie techniczne*, Wydawnictwa Naukowo-Techniczne, Warszawa 1984
- [Łchw01] Łachwa A.: *Rozmyty świat zbiorów, liczb, relacji, faktów, regul i decyzji*. Akademicka Oficyna Wydawnicza Exit, Warszawa 2001.
- [Łsk01] Wykłady prof. Łęski J.: *Teoria zbiorów rozmytych*. Gliwice 2001.
- [MdnPnsVl94a] Medina J. M., Pons O., Vila M. A.: *GEFRED. A Generalized Model of Fuzzy Relational Data Bases. Ver. 1.1*. Information Sciences, 76, 1-2, 87-109, 1994.
- [MdnPnsVl94b] Medina J. M., Pons O., Vila M. A.: *An Elemental Processor of Fuzzy SQL*. Mathware and Soft Computing 3, 285-295, 1994.
- [MdnVlCbrPns94] Medina J. M., Vila M.A, Cubero J. C., Pons O.: *Fuzzy Knowledge Representation in Relational Databases*. Information Sciences, 1994.
- [Mls02a] Małysiak B.: *Mechanizmy wnioskowania przybliżonego w bazach danych*. Studia Informatica, Vol. 23, nr 4(51). Gliwice

- 2002.
- [Mls02b] Małysiak B.: *Aproksymacyjne zapytania do baz danych*. Studia Informatica, Vol. 23, nr 4 (51). Gliwice 2002.
- [Mls03] Małysiak B.: *Wartości rozmyte w pytaniach sql do baz danych*. Studia Informatica, Vol. 24, nr 2A (53) Szczyrk 2003.
- [Pgt99] Piegat A.: *Modelowanie i sterowanie rozmyte*. Akademicka Oficyna Wydawnicza Exit, Warszawa 1999.
- [Plc82] Plucińska A., Pluciński E. *Zadania z rachunku prawdopodobieństwa i statystyki matematycznej dla studentów politechnik*. PWN. Warszawa 1982.
- [Prs96] Parsons S. *Current approaches to handling imperfect information in data and knowledge bases*. IEEE Transactions on Knowledge and Data Engineering, 8(3). 1996.
- [Pstgr7.2PG] *PostgreSQL 7.2 Programmer's Guide*. The PostgreSQL Global Development Group
- [Pstgr7.2RM] *PostgreSQL 7.2 Reference Manual*. The PostgreSQL Global Development Group
- [Pstgr7.2UG] *PostgreSQL 7.2 User's Guide*. The PostgreSQL Global Development Group.
- [RdnBc91] Rundensteiner A., Bic L.: *Evaluating Aggregates in possibilistic relational databases*. . Date&Knowledge Engineering, Vol.7, p. 239-267, 1991.
- [RsmYgr97] Rasmussen D., Yager R.: *Summary SQL – A Fuzzy Tool For Data Mining*. Intelligent Data Analysis 1. 1997
- [Snst00] Strona internetowa <http://fuzzy.sonalysts.com>, 2000.
- [Thn76] Tahani V.: *A Conceptual Framework for Fuzzy query processing: a step toward very intelligent data systems*. Inf. Proc. and Management 13. 1976.
- [Tkh91] Takahashi Y. *A fuzzy query language for relational databases*. IEEE Transactions on Systems, Man, and Cybernetics. Vol. 21, No.6, November/December 1991.
- [Tsk] Teska K.: *Fuzzy Logic Query Processing*
- [Ullm88] Ullman J. D.: *Database and knowledge-base systems*. Computer Science Press, 1988.
- [UllmWdm00] Ullman J. D., Widom Z.: *Podstawowy wykład z systemów baz danych*. Wydawnictwa Naukowo-Techniczne. Warszawa 2000
- [WellSoft95] *SQL Język Relacyjnych Baz Danych*. Wellesley Software. Wydawnictwa Naukowo-Techniczne, Warszawa 1995r.
- [WhtJn97] White D. A., Jain R.: *Algorithms and Strategies for Similarity*

- Retrieval*. Visual Computing Laboratory, Uniwersytet w Kaliforni, raport wewnętrzny. 1997. Adres internetowy <http://vision.ucsd.edu/papers/simret/journ1>
- [Wjc92] *Laboratorium podstaw przetwarzania i rozpoznawania obrazów*. Pod redakcją Wojciechowskiego K. Skrypty uczelniane nr 1662, Gliwice 1992.
- [Wrt89] Wirth N.: *Algorytmy + Struktury Danych = Programy*. Wydawnictwa Naukowo-Techniczne, Warszawa 1989.
- [YgrFlv95] Yager R, Filev D.: *Podstawy modelowania i sterowania rozmytego*. Wydawnictwa Naukowo-Techniczne, Wiley. Warszawa 1995.
- [YgrLrs93] Yager R., Larsen H.: *Retrieving information by fuzzyfication of queries*. Journal of Intelligent Information Systems. 1993.
- [YuMng98] Yu C.T., Meng W.: *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann Publishers, Inc., 1998.
- [YzcGrg99] Yazici A., George R.: *Fuzzy Database Modeling*. A Springer-Verlag Company. Niemcy Heidelberg 1999r.
- [Zdh65] Zadeh L. A.: *Fuzzy sets*, Information and Control 8, 338-353, 1965.
- [Zdh78] Zadeh L. A.: *PRUF – a meaning representation language for natural languages*. International Journal Man-Machine Studies, VOL. 10, pp. 395-460, 1978.

## **ZAŁĄCZNIK A**

### **Elementy programowe mechanizmów rozmytych w SZBD PostgreSQL**

## 1. Wstęp

Załącznik ten zawiera elementy programowe mechanizmów rozmytych, tzn. nagłówki funkcji w języku C oraz definicje typów, funkcji i operatorów w SZBD PostgreSQL.

## 2. Tworzenie typu *ftrapezium*

Struktura typu *ftrapezium* zaimplementowana w języku C ma następującą postać:

```
typedef struct {  
    float8 a, m, dm, b;  
} ftrapezium ;
```

Nagłówki funkcji konwertujących wartości typu *ftrapezium* mają następującą postać:

```
ftrapezium * ftrapezium_in( char *lancuch )  
char * ftrapezium_out( ftrapezium *f )
```

Po zaimplementowaniu w języku C powyższych funkcji należy je zarejestrować w PostgreSQL i utworzyć typ *ftrapezium*.

```
CREATE OR REPLACE FUNCTION ftrapezium_in( opaque ) RETURNS ftrapezium  
AS '/fuzzy/ftrapezium/ftrapezium.so' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION ftrapezium_out( ftrapezium ) RETURNS opaque  
AS '/fuzzy/ftrapezium/ftrapezium.so' LANGUAGE 'c';
```

```
CREATE TYPE ftrapezium(  
    internallength = 32,  
    input = ftrapezium_in,  
    output = ftrapezium_out  
) ;
```

W celu łatwiejszego wprowadzania wartości typu *ftrapezium* zaimplementowano funkcję *okolo*. Nagłówek tej funkcji zaimplementowanej w języku C jest następujący:

```
ftrapezium* about( float8 *l, float8 *m, float8 *n, float8 *o )
```

Definicja funkcji *okolo* w PostgreSQL ma następującą postać:

```
CREATE OR REPLACE FUNCTION okolo( float8, float8, float8, float8 )  
RETURNS ftrapezium  
AS '/fuzzy/ftrapezium/ftrapezium.so', 'about' LANGUAGE 'c' ;
```

### 3. Działania arytmetyczne na wartościach typu **ftrapezium**

#### 3.1. Rozmyte funkcje i operatory arytmetyczne

Nagłówki funkcji, wykorzystywanych przez operatory arytmetyczne, zaimplementowanych w języku C są następujące:

```
ftrapezium* minus_f ( ftrapezium* f )
ftrapezium* f_sum_r( ftrapezium* f, float8* r )
ftrapezium* r_sum_f( float8* r, ftrapezium* f )
ftrapezium* f_sub_r( ftrapezium* f, float8* r )
ftrapezium* r_sub_f( float8* r, ftrapezium* f )
ftrapezium* f_sum_f( ftrapezium *x, ftrapezium *y)
ftrapezium* f_sub_f( ftrapezium *x, ftrapezium *y)
ftrapezium* f_div_r( float8* r, ftrapezium *f )
ftrapezium* r_div_f( ftrapezium *f, float8* r )
ftrapezium* f_mul_r( ftrapezium *f, float8* r )
ftrapezium* r_mul_f( float8* r, ftrapezium* f )
ftrapezium* f_div_f( ftrapezium* a, ftrapezium* b )
ftrapezium* f_mul_f( ftrapezium* a, ftrapezium* b )
ftrapezium* f_min( ftrapezium* f1, ftrapezium* f2 )
ftrapezium* f_max( ftrapezium* f1, ftrapezium* f2 )
```

Funkcje te zarejestrowano w PostgreSQL w następujący sposób:

```
CREATE OR REPLACE FUNCTION minus( ftrapezium ) RETURNS ftrapezium
AS '/fuzzy/ftrapezium/ftrapezium.so', 'minus_f' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION sum( ftrapezium, float8 ) RETURNS ftrapezium
AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_sum_r' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION sum( float8, ftrapezium ) RETURNS ftrapezium
AS '/fuzzy/ftrapezium/ftrapezium.so', 'r_sum_f' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION sum( ftrapezium, ftrapezium )
RETURNS ftrapezium AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_sum_f'
LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION sub( ftrapezium, float8 ) RETURNS ftrapezium
AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_sub_r' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION sub( float8, ftrapezium ) RETURNS ftrapezium
AS '/fuzzy/ftrapezium/ftrapezium.so', 'r_sub_f' LANGUAGE 'c' ;
```



```

CREATE OR REPLACE FUNCTION sub( ftrapezium, ftrapezium )
  RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_sub_f' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION div( ftrapezium, float8 ) RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_div_r' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION div( float8, ftrapezium ) RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'r_div_f' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION mul( ftrapezium, float8 ) RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_mul_r' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION mul( float8, ftrapezium ) RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'r_mul_f' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION div( ftrapezium, ftrapezium )
  RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_div_f' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION mul( ftrapezium, ftrapezium )
  RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_mul_f' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION max( ftrapezium, ftrapezium )
  RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_max' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION min( ftrapezium, ftrapezium )
  RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'f_min' LANGUAGE 'c' ;

```

Dla większości z wymienionych operacji utworzono także arytmetyczne operatory rozmyte w SZBD PostgreSQL:

```

CREATE OPERATOR - ( rightarg = ftrapezium, procedure = minus );
CREATE OPERATOR + ( leftarg = ftrapezium, rightarg = float8,
  procedure = sum );
CREATE OPERATOR + ( leftarg = float8, rightarg = ftrapezium,
  procedure = sum );
CREATE OPERATOR + ( leftarg = ftrapezium, rightarg = ftrapezium,
  procedure = sum);
CREATE OPERATOR - ( leftarg = ftrapezium, rightarg = float8,
  procedure = sub );
CREATE OPERATOR - ( leftarg = float8, rightarg = ftrapezium,
  procedure = sub );

```

```

CREATE OPERATOR - ( leftarg = ftrapezium, rightarg = ftrapezium,
    procedure = sub );
CREATE OPERATOR / ( leftarg = ftrapezium, rightarg = float8,
    procedure = div );
CREATE OPERATOR * ( leftarg = ftrapezium, rightarg = float8,
    procedure = mul );
CREATE OPERATOR * ( leftarg = float8, rightarg = ftrapezium,
    procedure = mul );
CREATE OPERATOR / ( leftarg = ftrapezium, rightarg = ftrapezium,
    procedure = div );
CREATE OPERATOR * ( leftarg = ftrapezium, rightarg = ftrapezium,
    procedure = mul );

```

### 3.2. Rozmyte funkcje i operatory porównania

Nagłówki odpowiednich funkcji, wykorzystywanych przez operatory porównania, zaimplementowanych w języku C są następujące:

```

bool is_equal( ftrapezium* ft1, ftrapezium* ft2 )
bool not_equal( ftrapezium* ft1, ftrapezium* ft2 )
bool is_lower( ftrapezium* ft1, ftrapezium* ft2 )
bool is_lower_equal( ftrapezium* ft1, ftrapezium* ft2 )
bool is_greater( ftrapezium* ft1, ftrapezium* ft2 )
bool is_greater_equal( ftrapezium* ft1, ftrapezium* ft2 )

```

Funkcje te zdefiniowane w PostgreSQL mają następującą postać:

```

CREATE OR REPLACE FUNCTION is_equal( ftrapezium, ftrapezium )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_equal' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION not_equal( ftrapezium, ftrapezium )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'not_equal' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION is_lower( ftrapezium, ftrapezium )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_lower' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION is_lower_equal( ftrapezium, ftrapezium )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_lower_equal' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION is_greater( ftrapezium, ftrapezium )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_greater' LANGUAGE 'c' ;

```

```
CREATE OR REPLACE FUNCTION is_greater_equal( ftrapezium, ftrapezium )
  RETURNS bool
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_greater_equal' LANGUAGE 'c';
```

Dla tak zdefiniowanych funkcji utworzono w SZBD PostgreSQL operatory porównania, są one następujące:

```
CREATE OPERATOR = ( leftarg = ftrapezium, rightarg = ftrapezium,
  procedure = is_equal ) ;
CREATE OPERATOR <> ( leftarg = ftrapezium, rightarg = ftrapezium,
  procedure = not_equal ) ;
CREATE OPERATOR < ( leftarg = ftrapezium, rightarg = ftrapezium,
  procedure = is_lower ) ;
CREATE OPERATOR <= ( leftarg = ftrapezium, rightarg = ftrapezium,
  procedure = is_lower_equal ) ;
CREATE OPERATOR > ( leftarg = ftrapezium, rightarg = ftrapezium,
  procedure = is_greater ) ;
CREATE OPERATOR >= ( leftarg = ftrapezium, rightarg = ftrapezium,
  procedure = is_greater_equal ) ;
```

#### 4. Funkcje i operatory wyznaczające wartość stopnia zgodności

Nagłówki zaimplementowanych w języku C funkcji mają postać:

```
float8* degreeerf( float8* r, ftrapezium* f )
float8* degreefr( ftrapezium* f, float8* r )
float8* degreeoftrapezium ( ftrapezium *x, ftrapezium *y )
```

Każda z powyższych funkcji została zdefiniowana w PostgreSQL następująco:

```
CREATE OR REPLACE FUNCTION degree( float8, ftrapezium ) RETURNS float8
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'degreeerf' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION degree( ftrapezium, float8 ) RETURNS float8
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'degreefr' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION degreeoftrapezium( ftrapezium, ftrapezium )
  RETURNS float8
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'degreeoftrapezium' LANGUAGE 'c';
```

Dla każdego z przypadków utworzono również w PostgreSQL odpowiednie operatory:

```
CREATE OPERATOR ~= ( leftarg = float8, rightarg = ftrapezium,
  procedure = degree ) ;
CREATE OPERATOR ~= ( leftarg = ftrapezium, rightarg = float8,
  procedure = degree ) ;
```

```
CREATE OPERATOR ~= ( leftarg = ftrapezium, rightarg = ftrapezium,
    procedure = degreeoftrapezium ) ;
```

## 5. Funktory iloczynu, sumy i negacji rozmytych

Nagłówki funkcji, z których korzystają operatory &&&, ||| i ~ są następujące:

```
float8* min( float8* x, float8* y )
float8* max( float8* x, float8* y )
float8* neg_dm( float8 *x )
```

Deklaracje tych funkcji w PostgreSQL są następujące:

```
CREATE OR REPLACE FUNCTION min( float8, float8 ) RETURNS float8
AS '/fuzzy/fstandard/fstandard.so', 'min' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION max( float8, float8 ) RETURNS float8
AS '/fuzzy/fstandard/fstandard.so', 'max' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION negdm ( float8 ) RETURNS float8
AS '/fuzzy/fstandard/fstandard.so', 'neg_dm' LANGUAGE 'c' ;
```

Stosując przedstawione wcześniej funkcje utworzono w PostgreSQL operatory:

&&&, ||| i ~:

```
CREATE OPERATOR &&& (
    LEFTARG = float8,
    RIGHTARG = float8,
    PROCEDURE = min
)
```

```
CREATE OPERATOR ||| (
    LEFTARG = float8,
    RIGHTARG = float8,
    PROCEDURE = max
)
```

```
CREATE OPERATOR ~ (
    RIGHTARG = float8,
    PROCEDURE = negdm
)
```

## 6. Wartości predefiniowane dla typu *fttrapezium*

Przykładowe funkcje lingwistycznych w PostgreSQL są następujące:

```
CREATE OR REPLACE FUNCTION prawie_zaden() RETURNS fttrapezium
AS '/fuzzy/fttrapezium/fttrapezium.so', 'prawie_zaden' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION prawie_wszystkie() RETURNS fttrapezium
AS '/fuzzy/fttrapezium/fttrapezium.so', 'prawie_wszystkie' LANGUAGE 'c';

CREATE OR REPLACE FUNCTION okolo_jeden() RETURNS fttrapezium
AS '/fuzzy/fttrapezium/fttrapezium.so', 'okolo_jeden' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION okolo_kilka() RETURNS fttrapezium
AS '/fuzzy/fttrapezium/fttrapezium.so', 'kilka' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION okolo_kilkanascie() RETURNS fttrapezium
AS '/fuzzy/fttrapezium/fttrapezium.so', 'kilkanascie' LANGUAGE 'c';
```

## 7. Funkcje konwersji typu *fttrapezium* do postaci alfanumerycznej

Nagłówki przykładowych funkcji zaimplementowanych w języku C są następujące:

```
text* sekundy_to_lingw( fttrapezium* sekundy )
text* dni_to_lingw( fttrapezium* dni )
```

Funkcje te w PostgreSQL mają następującą postać:

```
CREATE OR REPLACE FUNCTION sekundy_to_lingw( fttrapezium ) RETURNS TEXT
AS '/fuzzy/zawodnicy/lingw.so', 'sekundy_to_lingw' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION dni_to_lingw( fttrapezium ) RETURNS TEXT
AS '/fuzzy/zawodnicy/lingw.so', 'dni_to_lingw' LANGUAGE 'c' ;
```

## 8. Funkcje agregujące na wartościach typu *fttrapezium*

### 8.1. Funkcje agregujące: *sum*, *avg*, *min*, *max*

Utworzenie funkcji agregującej w PostgreSQL wymaga wcześniejszego określenia dwóch funkcji tzw. *sfunc* i *final\_func* [Pstgr7.2PG]. Funkcje te zdefiniowano w języku C.

W przypadku funkcji agregującej *sum* jako funkcję *sfunc* wykorzystano istniejącą już w systemie funkcję *sum* dodającą do siebie liczby rozmyte (rozdział 4).

Nagłówki zaimplementowanych w języku C funkcji: *sfunc* i *final\_func* dla funkcji agregujących *avg*, *min*, *max* są następujące:

```

ftrapeziumext* stateavg( ftrapeziumext* state, ftrapezium* nextdate )
ftrapezium* finalavg( ftrapeziumext* x )
ftrapeziumext* statemin( ftrapeziumext* state, ftrapezium* next_data )
ftrapeziumext* statemax( ftrapeziumext* state, ftrapezium* next_data )
ftrapezium* ftrapeziumext_to_ftrapezium( ftrapeziumext* last_state)

```

Kolejnym krokiem jest zadeklarowanie tych funkcji w SZBD PostgreSQL w następujący sposób:

```

CREATE OR REPLACE FUNCTION stateavg( ftrapeziumext, ftrapezium )
  RETURNS ftrapeziumext
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'stateavg' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION finalavg( ftrapeziumext ) RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'finalavg' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION statemin( ftrapeziumext, ftrapezium )
  RETURNS ftrapeziumext
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'statemin' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION statemax( ftrapeziumext, ftrapezium )
  RETURNS ftrapeziumext
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'statemax' LANGUAGE 'c' ;

CREATE OR REPLACE FUNCTION toftrapezium( ftrapeziumext )
  RETURNS ftrapezium
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'ftrapeziumext_to_ftrapezium'
  LANGUAGE 'c';

```

Funkcje agregujące w PostgreSQL mają następującą postać:

```

CREATE AGGREGATE sum( basetype = ftrapezium, stype = ftrapezium,
  sfunc = sum, initcond = '0/0~0\\0' ) ;

CREATE AGGREGATE avg( basetype = ftrapezium, stype = ftrapeziumext,
  sfunc = stateavg, finalfunc = finalavg, initcond = '0|0/0~0\\0' ) ;

CREATE AGGREGATE min( basetype = ftrapezium, stype = ftrapeziumext,
  sfunc = statemin, finalfunc = toftrapezium, initcond = '0|0' ) ;

CREATE AGGREGATE max( basetype = ftrapezium, stype = ftrapeziumext,
  sfunc = statemax, finalfunc = toftrapezium, initcond = '0|0' ) ;

```

W powyższych funkcjach agregujących został wykorzystany rozszerzony typ *ftrapezium* o nazwie *ftrapeziumext*.

## 8.2. Rozmyte kwantyfikatory

Funkcje umożliwiające zdefiniowanie procentowych udziałów podanych wartości kolumn w zbiorze wierszy wymagają wykonania pewnych operacji na grupach wierszy.

Do swej realizacji wymagają one podania liczby wierszy spełniających kryterium oraz liczby wszystkich wierszy. W tym celu utworzono w języku C i w PostgreSQL typ pozwalający na przechowywanie dwóch wartości całkowitych o nazwie *twoint*.

Definicja typu (struktury) *twoint* w języku C:

```
typedef struct {  
    int8 bool_count, count ;  
} twoint ;
```

Zdefiniowanie typu w PostgreSQL wymaga dodatkowo określenia funkcji konwersji. Nagłówki tych funkcji dla typu *twoint* są następujące:

```
twoint* twoint_in( char *lancuch )  
char* twoint_out( twoint* t)
```

Funkcja *twoint\_out* konwertuje reprezentację wewnętrzną typu, w tym przypadku strukturę *twoint*, do postaci zewnętrznej wyrażonej w postaci alfanumerycznej. Funkcja *twoint\_in* realizuje konwersję odwrotną.

Nagłówki tych funkcji są następujące:

```
twoint* odsetek_sfunc( twoint * state, bool next_data )  
float8* odsetek_final_func( twoint* last_state)
```

### Rejestracja typu *twoint*

Po zaimplementowaniu w języku C typu oraz funkcji konwersji (operacji we/wy) należy je zarejestrować w SZBD PostgreSQL.

```
CREATE OR REPLACE FUNCTION twoint_in( opaque ) RETURNS twoint  
AS '/fuzzy/fstandard/fstandard.so' LANGUAGE 'c';  
  
CREATE OR REPLACE FUNCTION twoint_out( opaque ) RETURNS opaque  
AS '/fuzzy/fstandard/fstandard.so' LANGUAGE 'c';  
  
CREATE TYPE twoint(  
    internallength = 16,  
    input = twoint_in,  
    output = twoint_out  
) ;
```

### Definowanie funkcji *odsetek*

Kolejnym krokiem jest zadeklarowanie tych funkcji w SZBD PostgreSQL:

```
CREATE OR REPLACE FUNCTION odsetek_sfunc ( twoint, bool ) RETURNS twoint
AS '/fuzzy/fstandard/fstandard.so', 'odsetek_sfunc' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION odsetek_final_func ( twoint ) RETURNS float8
AS '/fuzzy/fstandard/fstandard.so', 'odsetek_final_func' LANGUAGE 'c';
```

Definicja funkcji agregującej w PostgreSQL jest następująca:

```
CREATE AGGREGATE odsetek (
    BASETYPE = bool,
    SFUNC = odsetek_sfunc,
    STYPE = twoint,
    FINALFUNC = odsetek_final_func,
    INITCOND = '0,0'
) ;
```

### Rozmyte kwantyfikatory

Nagłówki przykładowych funkcji zaimplementowanych w języku C są następujące:

```
float8* odsetek_prawie_zaden( twoint* last_state )
float8* odsetek_prawie_wszystkie( twoint* last_state )
float8* odsetek_okolo_jedna_czwarta( twoint* last_state )
float8* odsetek_okolo_jedna_trzecia( twoint* last_state )
float8* odsetek_okolo_polowa( twoint* last_state )
float8* odsetek_okolo_dwie_trzecie( twoint* last_state )
float8* odsetek_okolo_trzy_czwarte( twoint* last_state )
```

Zdefiniowane funkcje w PostgreSQL mają następującą postać:

```
CREATE OR REPLACE FUNCTION odsetek_prawie_zaden( twoint ) RETURNS float8
AS '/fuzzy/ftrapezium/ftrapezium.so', 'odsetek_prawie_zaden' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION odsetek_prawie_wszystkie( twoint )
RETURNS float8 AS '/fuzzy/ftrapezium/ftrapezium.so',
'odsetek_prawie_wszystkie' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION odsetek_okolo_jedna_czwarta( twoint )
RETURNS float8 AS '/fuzzy/ftrapezium/ftrapezium.so',
'odsetek_okolo_jedna_czwarta' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION odsetek_okolo_jedna_trzecia( twoint )
RETURNS float8 AS '/fuzzy/ftrapezium/ftrapezium.so',
'odsetek_okolo_jedna_trzecia' LANGUAGE 'c';
```



```
CREATE OR REPLACE FUNCTION odsetek_okolo_polowa( twoint ) RETURNS float8
AS '/fuzzy/ftrapezium/ftrapezium.so', 'odsetek_okolo_polowa' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION odsetek_okolo_dwie_trzecie( twoint )
RETURNS float8 AS '/fuzzy/ftrapezium/ftrapezium.so',
'odsetek_okolo_dwie_trzecie' LANGUAGE 'c';
```

```
CREATE OR REPLACE FUNCTION odsetek_okolo_trzy_czwarte( twoint )
RETURNS float8 AS '/fuzzy/ftrapezium/ftrapezium.so',
'odsetek_okolo_trzy_czwarte' LANGUAGE 'c' ;
```

Przykładowe rozmyte kwantyfikatory - funkcje agregujące rozmywające funkcje agregującą *odsetek* zapisane w PostgreSQL mają następującą postać:

```
CREATE AGGREGATE prawie_zaden( BASETYPE = bool, SFUNC = odsetek_sfunc,
    STYPE = twoint, FINALFUNC = odsetek_prawie_zaden, INITCOND = '0,0' ) ;
```

```
CREATE AGGREGATE prawie_wszystkie( BASETYPE = bool,
    SFUNC = odsetek_sfunc, STYPE = twoint,
    FINALFUNC = odsetek_prawie_wszystkie, INITCOND = '0,0' ) ;
```

```
CREATE AGGREGATE okolo_jedna_czwarta( BASETYPE = bool,
    SFUNC = odsetek_sfunc, STYPE = twoint,
    FINALFUNC = odsetek_okolo_jedna_czwarta, INITCOND = '0,0' ) ;
```

```
CREATE AGGREGATE okolo_jedna_trzecia( BASETYPE = bool,
    SFUNC = odsetek_sfunc, STYPE = twoint,
    FINALFUNC = odsetek_okolo_jedna_trzecia, INITCOND = '0,0' ) ;
```

```
CREATE AGGREGATE okolo_polowa( BASETYPE = bool, SFUNC = odsetek_sfunc,
    STYPE = twoint, FINALFUNC = odsetek_okolo_polowa, INITCOND = '0,0' ) ;
```

```
CREATE AGGREGATE okolo_dwie_trzecie( BASETYPE = bool,
    SFUNC = odsetek_sfunc, STYPE = twoint,
    FINALFUNC = odsetek_okolo_dwie_trzecie, INITCOND = '0,0' ) ;
```

```
CREATE AGGREGATE okolo_trzy_czwarte( BASETYPE = bool,
    SFUNC = odsetek_sfunc, STYPE = twoint,
    FINALFUNC = odsetek_okolo_trzy_czwarte, INITCOND = '0,0' ) ;
```

## 9. Funkcje i operatory realizujące rozmyte grupowanie danych

Definicje typów rozszerzających typ *ftrapezium* oraz typ rzeczywisty *float8* o rozpiętość grupy *epsilon* oraz nagłówki funkcji konwertujących zaimplementowanych w języku C są następujące:

```
typedef struct{
    ftrapezium epsilon ;
    ftrapezium dane ;
} otoczenie ;

otoczenie* otoczenie_in( char *lancuch )
char* otoczenie_out( otoczenie *f )

typedef struct {
    float8 epsilon ;
    float8 dane ;
} otoczenie_float8 ;

otoczenie_float8* otoczenie_float8_in( char *lancuch )
char* otoczenie_float8_out( otoczenie_float8 *f )
```

Definicje powyższych funkcji i typów w SZBD PostgreSQL mają następującą postać:

```
-- tworzenie typu otoczenia ftrapezium

CREATE OR REPLACE FUNCTION otoczenie_in( opaque ) RETURNS otoczenie
AS '/fuzzy/ftrapezium/ftrapezium.so' LANGUAGE 'c';

CREATE OR REPLACE FUNCTION otoczenie_out( otoczenie ) RETURNS opaque
AS '/fuzzy/ftrapezium/ftrapezium.so' LANGUAGE 'c';

CREATE TYPE otoczenie( internallength = 64, input = otoczenie_in,
    output = otoczenie_out ) ;

-- tworzenie typu otoczenia float8 (numeryczne)

CREATE OR REPLACE FUNCTION otoczenie_float8_in( opaque )
RETURNS otoczenie_float8 AS
'/fuzzy/ftrapezium/ftrapezium.so', 'otoczenie_float8_in' LANGUAGE 'c';

CREATE OR REPLACE FUNCTION otoczenie_float8_out( otoczenie_float8 )
RETURNS opaque AS
'/fuzzy/ftrapezium/ftrapezium.so', 'otoczenie_float8_out' LANGUAGE 'c';
```

```
CREATE TYPE otoczenie_float8( internallength = 16,
    input = otoczenie_float8_in, output = otoczenie_float8_out ) ;
```

Nagłówki funkcji zaimplementowanych w języku C, modyfikujące działanie operatorów mniejszości i równości wykorzystywanych w procesie grupowania danych są następujące:

```
-- dla wartości typu ftrapezium
bool otoczenie_is_equal( otoczenie* ot1, otoczenie* ot2 )
bool otoczenie_is_lower( otoczenie* ot1, otoczenie* ot2 )
otoczenie* otoczenie_ftrapezium( ftrapezium* f, ftrapezium* epsilon )

-- dla wartości typu float8
bool otoczenie_float8_is_equal( otoczenie_float8* ot1,
                                otoczenie_float8* ot2 )
bool otoczenie_float8_is_lower( otoczenie_float8* ot1,
                                otoczenie_float8* ot2 )
otoczenie_float8* make_otoczenie_float8( float8* r, float8* epsilon )
```

Definicje tych funkcji oraz operatorów w SZBD PostgreSQL mają następującą postać:

```
// dla wartości typu ftrapezium
-- liczby rowne
CREATE OR REPLACE FUNCTION is_equal( otoczenie, otoczenie )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'otoczenie_is_equal' LANGUAGE 'c' ;
CREATE OPERATOR = ( leftarg = otoczenie, rightarg = otoczenie,
    procedure = is_equal ) ;

-- pierwsza mniejsza
CREATE OR REPLACE FUNCTION is_lower( otoczenie, otoczenie )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'otoczenie_is_lower' LANGUAGE 'c' ;
CREATE OPERATOR < ( leftarg = otoczenie, rightarg = otoczenie, procedure
= is_lower ) ;

CREATE OR REPLACE FUNCTION otoczenie( ftrapezium, ftrapezium )
    RETURNS otoczenie AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'otoczenie_ftrapezium' LANGUAGE 'c';
CREATE OPERATOR <<< ( leftarg = ftrapezium, rightarg = ftrapezium,
    procedure = otoczenie ) ;

//dla wartości typu float8
-- liczby rowne
CREATE OR REPLACE FUNCTION is_equal( otoczenie_float8, otoczenie_float8 )
    RETURNS bool AS '/fuzzy/ftrapezium/ftrapezium.so',
    'otoczenie_float8_is_equal' LANGUAGE 'c';
CREATE OPERATOR = ( leftarg = otoczenie_float8,
```

```

    rightarg = otoczenie_float8, procedure = is_equal ) ;

-- pierwsza mniejsza
CREATE OR REPLACE FUNCTION is_lower( otoczenie_float8, otoczenie_float8 )
    RETURNS bool AS '/fuzzy/fttrapezium/fttrapezium.so',
    'otoczenie_float8_is_lower' LANGUAGE 'c';
CREATE OPERATOR < ( leftarg = otoczenie_float8, rightarg =
    otoczenie_float8, procedure = is_lower ) ;

-- pierwsza mniejsza
CREATE OR REPLACE FUNCTION make_otoczenie_float8( float8, float8 )
    RETURNS otoczenie_float8 AS '/fuzzy/fttrapezium/fttrapezium.so',
    'make_otoczenie_float8' LANGUAGE 'c' ;
CREATE OPERATOR <<< ( leftarg = float8, rightarg = float8,
    procedure = make_otoczenie_float8 ) ;

```

## 10. Funkcje i operatory rozmyte realizujące rozmyty warunek wiążący pytanie zewnętrzne z wewnętrznym

### 10.1. Rozszerzenie typu *fttrapezium* - *fttrapeziumext*

Reprezentację wewnętrzną typu *fttrapeziumext* stanowi struktura zaimplementowana w języku C o postaci:

```

typedef struct {
    float8 ext ;
    fttrapezium f ;
    char op [ 4 ] ;
} fttrapeziumext ;

```

Do konwersji reprezentacji wewnętrznej typu czyli w tym przypadku struktury *fttrapeziumext* do postaci zewnętrznej wyrażonej w postaci alfanumerycznej służy funkcja *fttrapeziumext\_out*, konwersję odwrotną realizuje funkcja *fttrapeziumext\_in*. Nagłówki tych funkcji są następujące:

```

fttrapeziumext * fttrapeziumext_in( char *lancuch )
char * fttrapeziumext_out(fttrapeziumext *x)

```

Po zaimplementowaniu w języku C powyższych funkcji należy je zarejestrować w PostgreSQL i utworzyć typ *fttrapeziumext*.

#### Funkcje konwersji dla typu *fttrapeziumext*

```

CREATE OR REPLACE FUNCTION fttrapeziumext_in( opaque )
    RETURNS fttrapeziumext
    AS '/fuzzy/fttrapezium/fttrapezium.so' LANGUAGE 'c';

```

```
CREATE OR REPLACE FUNCTION ftrapeziumext_out( ftrapeziumext )
  RETURNS opaque
  AS '/fuzzy/ftrapezium/ftrapezium.so' LANGUAGE 'c';
```

```
CREATE TYPE ftrapeziumext(
  internallength = 44,
  input = ftrapeziumext_in,
  output = ftrapeziumext_out
) ;
```

Poniżej przedstawione zostaną uniwersalne funkcje rozszerzające typ *ftrapezium* w *ftrapeziumext*:

Nagłówek funkcji rozszerzającej typ *ftrapezium* o próg defuzyfikacji jest następujący:

```
ftrapeziumext* to_fext( ftrapezium* ft, float8* ext, char op[ 4 ] ).
```

Nagłówki funkcji wywołujących funkcję *to\_fext* i wprowadzające odpowiedni operator relacyjny do wartości typu *ftrapeziumext* są następujące.

```
ftrapeziumext* to_fext_equal( ftrapezium* ft, float8* ext )
ftrapeziumext* to_fext_not_equal( ftrapezium* ft, float8* ext )
ftrapeziumext* to_fext_greater( ftrapezium* ft, float8* ext )
ftrapeziumext* to_fext_greater_equal( ftrapezium* ft, float8* ext )
ftrapeziumext* to_fext_lower( ftrapezium* ft, float8* ext )
ftrapeziumext* to_fext_lower_equal( ftrapezium* ft, float8* ext )
```

Definicje powyższych funkcji w PostgreSQL są następujące:

```
CREATE OR REPLACE FUNCTION to_fext_equal( ftrapezium, float8)
  RETURNS ftapeziumext
  AS '/fuzzy/ftrapezium/ftapezium.so', 'to_fext_equal' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION to_fext_not_equal( ftapezium, float8)
  RETURNS ftapeziumext
  AS '/fuzzy/ftapezium/ftapezium.so', 'to_fext_not_equal' LANGUAGE 'c'
;
```

```
CREATE OR REPLACE FUNCTION to_fext_greater( ftapezium, float8)
  RETURNS ftapeziumext
  AS '/fuzzy/ftapezium/ftapezium.so', 'to_fext_greater' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION to_fext_greater_equal( ftapezium, float8)
  RETURNS ftapeziumext AS '/fuzzy/ftapezium/ftapezium.so',
  'to_fext_greater_equal' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION to_fext_lower( ftapezium, float8)
  RETURNS ftapeziumext
  AS '/fuzzy/ftapezium/ftapezium.so', 'to_fext_lower' LANGUAGE 'c' ;
```

```
CREATE OR REPLACE FUNCTION to_fext_lower_equal( ftrapezium, float8)
  RETURNS ftrapeziumext AS '/fuzzy/ftrapezium/ftrapezium.so',
  'to_fext_lower_equal' LANGUAGE 'c' ;
```

Kolejnym krokiem jest utworzenie operatorów wpisujących podany nazwą operator relacyjny do typu *ftrapeziumext*.

```
CREATE OPERATOR *= ( leftarg = ftrapezium, rightarg = float8,
  procedure = to_fext_equal ) ;
CREATE OPERATOR *<> ( leftarg = ftrapezium, rightarg = float8,
  procedure = to_fext_not_equal ) ;
CREATE OPERATOR *> ( leftarg = ftrapezium, rightarg = float8,
  procedure = to_fext_greater ) ;
CREATE OPERATOR *>= ( leftarg = ftrapezium, rightarg = float8,
  procedure = to_fext_greater_equal ) ;
CREATE OPERATOR *< ( leftarg = ftrapezium, rightarg = float8,
  procedure = to_fext_lower ) ;
CREATE OPERATOR *<= ( leftarg = ftrapezium, rightarg = float8,
  procedure = to_fext_lower_equal ) ;
```

Nagłówek ogólnej funkcji realizującej warunek progowy zgodnie z podanym operatorem relacyjnym oraz wartością progową ma następującą postać:

```
bool defuzzy( float8 degree, char op[ 4 ], float8 threshold )
```

## 10.2. Porównanie wartości rozmytej z wartością dokładną

W tym przypadku wartość rozmyta jest rozszerzana o zadaną wartość progową oraz odpowiedni operator relacyjny (=, <>, <, <=, >, >=).

Nagłówki funkcji zaimplementowanych w języku C realizujących warunek progowy wartości typu *ftrapeziumext* z wartością dokładną są następujące:

```
bool is_equal_fx_r( ftrapeziumext* fext, float8* r )
bool is_not_equal_fx_r( ftrapeziumext* fext, float8* r )
bool is_lower_fx_r( ftrapeziumext* fx, float8* r )
bool is_lower_equal_fx_r( ftrapeziumext* fx, float8* r )
bool is_greater_fx_r( ftrapeziumext* fx, float8* r )
bool is_greater_equal_fx_r( ftrapeziumext* fx, float8* r )
```

Kolejnym krokiem jest rejestracja zaimplementowanych funkcji oraz operatorów porównania, których utworzenie jest niezbędne dla poprawnego działania operatora *IN* i *NOT IN* oraz *ANY* i *ALL* w systemie PostgreSQL:

```
CREATE OR REPLACE FUNCTION is_equal( ftrapeziumext, float8 )
  RETURNS bool
  AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_equal_fx_r' LANGUAGE 'c' ;
```

```

CREATE OPERATOR = ( leftarg = ftrapeziumext, rightarg = float8,
procedure = is_equal ) ;
CREATE OR REPLACE FUNCTION is_not_equal( ftrapeziumext, float8 )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'is_not_equal_fx_f' LANGUAGE 'c';
CREATE OPERATOR <> ( leftarg = ftrapeziumext, rightarg = float8,
procedure = is_not_equal ) ;

CREATE OR REPLACE FUNCTION is_lower_equal( ftrapeziumext, float8 )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'is_lower_equal_fx_r' LANGUAGE 'c';
CREATE OPERATOR <= ( leftarg = ftrapeziumext, rightarg = float8,
procedure = is_lower_equal ) ;

CREATE OR REPLACE FUNCTION is_greater_equal( ftrapeziumext, float8 )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'is_greater_equal_fx_r' LANGUAGE 'c';
CREATE OPERATOR >= ( leftarg = ftrapeziumext, rightarg = float8,
procedure = is_greater_equal ) ;

```

### 10.3. Porównanie wartości dokładnej z wartością rozmytą

W tym przypadku również wartość rozmyta jest rozszerzana o zadaną wartość progową defuzyfikacji oraz odpowiedni operator relacyjny (=, <>, <, <=, >, >=).

Nagłówki funkcji zaimplementowanych w języku C realizujących proces defuzyfikacji wartości dokładnej z wartością typu *ftrapeziumext* są następujące:

```

bool is_equal_r_fx(float8* data, ftrapeziumext* fext )
bool is_not_equal_r_fx(float8* data, ftrapeziumext* fext )
bool is_lower_r_fx( float8* r, ftrapeziumext* fx )
bool is_lower_equal_r_fx( float8* r, ftrapeziumext* fx )
bool is_greater_r_fx( float8* r, ftrapeziumext* fx )
bool is_greater_equal_r_fx( float8* r, ftrapeziumext* fx )

```

Kolejnym krokiem jest rejestracja zaimplementowanych funkcji w systemie PostgreSQL:

```

CREATE OR REPLACE FUNCTION is_equal( float8, ftrapeziumext )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_equal_r_fx' LANGUAGE 'c' ;
CREATE OPERATOR = ( leftarg = float8 , rightarg = ftrapeziumext,
procedure = is_equal ) ;

CREATE OR REPLACE FUNCTION is_not_equal( float8, ftrapeziumext )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'is_not_equal_r_fx' LANGUAGE 'c';

```

```

CREATE OPERATOR <> ( leftarg = float8, rightarg = ftrapeziumext,
procedure = is_not_equal ) ;

CREATE OR REPLACE FUNCTION is_lower_equal( float8, ftrapeziumext )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'is_lower_equal_r_fx' LANGUAGE 'c' ;
CREATE OPERATOR <= ( leftarg = float8, rightarg = ftrapeziumext,
procedure = is_lower_equal ) ;

CREATE OR REPLACE FUNCTION is_greater_equal( float8, ftrapeziumext )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'is_greater_equal_r_fx' LANGUAGE 'c';
CREATE OPERATOR >= ( leftarg = float8, rightarg = ftrapeziumext,
procedure = is_greater_equal ) ;

```

#### 10.4. Porównanie wartości rozmytej z wartością rozmytą

W tym przypadku dowolna z wartości rozmytych jest rozszerzana o zadaną wartość progową oraz odpowiedni operator porównania (=, <>, <, <=, >, >=).

Nagłówki funkcji zaimplementowanych w języku C realizujących proces defuzyfikacji wartości typu *ftrapeziumext* z wartością rozmytą oraz wartości rozmytej z wartością typu *ftrapeziumext* są następujące:

```

bool is_equal_fx_f( ftrapeziumext* fext, ftrapezium* ft )
bool is_not_equal_fx_f( ftrapeziumext* fext, ftrapezium* ft )
bool is_lower_fx_f( ftrapeziumext* fx, ftrapezium* ft)
bool is_lower_equal_fx_f( ftrapeziumext* fx, ftrapezium* ft )
bool is_greater_fx_f( ftrapeziumext* fx, ftrapezium* ft )
bool is_greater_equal_fx_f( ftrapeziumext* fx, ftrapezium* ft )

bool is_equal_f_fx(ftrapezium* ft, ftrapeziumext* fext )
bool is_not_equal_f_fx( ftrapezium* ft, ftrapeziumext* fext )
bool is_lower_f_fx( ftrapezium* f, ftrapeziumext* fext )
bool is_lower_equal_f_fx( ftrapezium* f, ftrapeziumext* fext )
bool is_greater_f_fx( ftrapezium* f, ftrapeziumext* fext )
bool is_greater_equal_f_fx( ftrapezium* f, ftrapeziumext* fext )

```

Kolejnym krokiem jest rejestracja zaimplementowanych funkcji w systemie PostgreSQL:

```

CREATE OR REPLACE FUNCTION is_equal( ftrapeziumext, ftrapezium )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_equal_fx_f' LANGUAGE 'c' ;
CREATE OPERATOR = ( leftarg = ftrapeziumext, rightarg = ftrapezium,
procedure = is_equal ) ;

CREATE OR REPLACE FUNCTION is_not_equal( ftrapeziumext, ftrapezium )

```



```
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_not_equal_fx_f' LANGUAGE 'c';
CREATE OPERATOR <> ( leftarg = ftrapeziumext, rightarg = ftrapezium,
procedure = is_not_equal ) ;

CREATE OR REPLACE FUNCTION is_lower_equal( ftrapeziumext, ftrapezium )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftrapezium.so', 'is_lower_equal_fx_f' LANGUAGE 'c';
CREATE OPERATOR <= ( leftarg = ftrapeziumext, rightarg = ftrapezium,
procedure = is_lower_equal ) ;

CREATE OR REPLACE FUNCTION is_greater_equal( ftrapeziumext, ftrapezium )
    RETURNS bool AS
    '/fuzzy/ftrapezium/ftrapezium.so', 'is_greater_equal_fx_f' LANGUAGE 'c';
CREATE OPERATOR >= ( leftarg = ftrapeziumext, rightarg = ftrapezium,
procedure = is_greater_equal ) ;

CREATE OR REPLACE FUNCTION is_equal( ftrapezium, ftapeziumext )
    RETURNS bool
    AS '/fuzzy/ftrapezium/ftapezium.so', 'is_equal_f_fx' LANGUAGE 'c' ;
CREATE OPERATOR = ( leftarg = ftapezium, rightarg = ftapeziumext,
procedure = is_equal ) ;

CREATE OR REPLACE FUNCTION is_not_equal( ftapezium, ftapeziumext )
    RETURNS bool AS
    '/fuzzy/ftapezium/ftapezium.so', 'is_not_equal_f_fx' LANGUAGE 'c';
CREATE OPERATOR <> ( leftarg = ftapezium, rightarg = ftapeziumext,
procedure = is_not_equal ) ;

CREATE OR REPLACE FUNCTION is_lower_equal( ftapezium, ftapeziumext )
    RETURNS bool AS
    '/fuzzy/ftapezium/ftapezium.so', 'is_lower_equal_f_fx' LANGUAGE 'c' ;
CREATE OPERATOR <= ( leftarg = ftapezium, rightarg = ftapeziumext,
procedure = is_lower_equal ) ;

CREATE OR REPLACE FUNCTION is_greater_equal( ftapezium, ftapeziumext )
    RETURNS bool AS
    '/fuzzy/ftapezium/ftapezium.so', 'is_greater_equal_f_fx' LANGUAGE 'c';
CREATE OPERATOR >= ( leftarg = ftapezium, rightarg =ftapeziumext ,
procedure = is_greater_equal ) ;
```

## **ZAŁĄCZNIK B**

### **Ilustracja rozmytego grupowania danych**

## 1. Wprowadzenie

Rozpatrzmy tabelę *Pracownicy* wchodzącą w skład analizowanej w poprzednich rozdziałach bazy danych *ZAKŁADY*.

Niech do bazy danych kierowane będzie następujące pytanie:

„Wyszukaj liczby pracowników o podobnym stażu pracy.”

Postać tego pytania z zastosowaniem grupowania rozmytego realizowanego wg zaimplementowanego algorytmu (podrozdział 3.5.1) jest następująca:

```
SELECT min(staz_pracy), count(*)
FROM pracownicy
GROUP BY otoczenie(staz_pracy, maxd) ;
```

gdzie *maxd* jest zadawaną rozpiętością grupy.

Na przykładzie tego pytania pokażemy jak dane wejściowe zostaną podzielone na grupy w zależności od ich rozkładu i wielkości rozpiętości grupy.

Rozważmy następujący podział danych wejściowych:

- dane wejściowe dla eksperymentu 1 - rozłożone równomiernie,
- dane wejściowe dla eksperymentu 2 - rozłożone w pewnych skupiskach.

Dla każdego z wymienionych przypadków rozpatrzmy trzy różne rozpiętości: 3, 5 i 10.

## 2. Dane wejściowe rozłożone równomiernie

Dane wejściowe ilustruje tabela 2.1:

Tabela 2.1

Tabela *pracownicy* - eksperyment 1

nr_prac	Nazwisko	staz_pracy
1	Adamczak	35
2	Aksamit	5
3	Bakanowski	36
4	Czartyszyn	41
5	Czupryna	36
6	Dyduch	10
7	Filipek	37
8	Jablonka	24
9	Jagoda	34
10	Jakubczyk	14
11	Janowski	15
12	Jelonek	19
13	Jemiola	15
14	Kamyk	32
15	Kocot	31

16	Kosiński	24
17	Kowalik	26
18	Kowalski	22
19	Kropla	8
20	Kroplewski	2
21	Kukulik	19
22	Kupczyk	7
23	Leśniak	27
24	Lis	3
25	Lisecka	5
26	Lizak	31
27	Majewski	10
28	Motylkiewicz	44
29	Niezgoda	12
30	Nowak	4
31	Nowakowski	2
32	Okocimski	38
33	Palik	14
34	Pawluszyn	40
35	Pebianowska	38
36	Pękała	43
37	Powoli	42
38	Ramek	32
39	Raptus	22
40	Romanowski	39
41	Smolecka	41
42	Sowa	19
43	Widuch	8
44	Zajęc	12
45	Zalewska	8
46	Zawoja	15
47	Zebra	34
48	Zientara	29
49	Zioło	27
50	Zyzak	30

Przed samym grupowaniem, dane wejściowe zostają posortowane, dane po posortowaniu przedstawione są w tabeli 2.2.

Tabela 2.2  
Posortowana tabela *pracownicy*  
względem kolumny *staz\_pracy*  
(eksperyment 1)

nr_prac	nazwisko	staz_pracy
20	Kroplewski	2
31	Nowakowski	2
24	Lis	3
30	Nowak	4
2	Aksamit	5
25	Lisecka	5
22	Kupczyk	7
19	Kropla	8
43	Widuch	8

45	Zalewska	8
6	Dyduch	10
27	Majewski	10
29	Niezgoda	12
44	Zając	12
10	Jakubczyk	14
33	Palik	14
11	Janowski	15
13	Jemioła	15
46	Zawoja	15
12	Jelonek	19
21	Kukulik	19
42	Sowa	19
18	Kowalski	22
39	Raptus	22
8	Jabłonka	24
16	Kosiński	24
17	Kowalik	26
23	Leśniak	27
49	Zioło	27
48	Zientara	29
50	Zyzak	30
15	Kocot	31
26	Lizak	31
14	Kamyk	32
38	Ramek	32
9	Jagoda	34
47	Zebra	34
1	Adameczak	35
3	Bakanowski	36
5	Czupryna	36
7	Filipek	37
32	Okocimski	38
35	Pebianowska	38
40	Romanowski	39
34	Pawluszyn	40
4	Czartyszyn	41
41	Smolecka	41
37	Powoli	42
36	Pękała	43
28	Motylkiewicz	44

Kolejny etap, to proces tworzenia grup w zależności od podanej rozpiętości:

### Rozpiętość 3

Zapis powyższego pytania dla rozpiętości 3 jest następujący:

```
SELECT min(staz_pracy), count(*)  
FROM pracownicy  
GROUP BY otoczenie(staz_pracy, 3) ;
```

Tabelę wynikową z licznnością wyodrębnionych grup przedstawia tabela 2.3:

Tabela 2.3

Wynikowa tabela dla rozpiętości 3  
(eksperyment 1)

nr	min(staz_pracy)	liczba
1	2	6
2	7	6
3	12	7
4	19	5
5	24	5
6	29	6
7	34	6
8	38	6
9	42	3

W powyższej tabeli kolumna *min(staz\_pracy)* zawiera dolną granicę grupy.

Proces sklejania wierszy w grupy zilustrujemy w tabeli 2.4:

Tabela 2.4

Proces sklejania w grupy dla rozpiętości = 3  
(eksperyment 1)

nr_prac	nazwisko	staz_pracy	podzial grup
20	Kroplewski	2	grupa 1
31	Nowakowski	2	
24	Lis	3	
30	Nowak	4	
2	Aksamit	5	
25	Lisecka	5	
22	Kupeczyk	7	grupa 2
19	Kropla	8	
43	Widuch	8	
45	Zalewska	8	
6	Dyduch	10	
27	Majewski	10	
29	Niezgoda	12	grupa 3
44	Zajac	12	
10	Jakubczyk	14	
33	Palik	14	
11	Janowski	15	
13	Jemioła	15	
46	Zawoja	15	
12	Jelonek	19	grupa 4
21	Kukulik	19	
42	Sowa	19	
18	Kowalski	22	
39	Raptus	22	
8	Jablonka	24	grupa 5
16	Kosiński	24	
17	Kowalik	26	
23	Leśniak	27	

49	Zioło	27	
48	Zientara	29	grupa 6
50	Zyzak	30	
15	Kocot	31	
26	Lizak	31	
14	Kamyk	32	
38	Ramek	32	
9	Jagoda	34	grupa 7
47	Zebra	34	
1	Adamczak	35	
3	Bakanowski	36	
5	Czupryna	36	
7	Filipek	37	
32	Okocimski	38	grupa 8
35	Pebianowska	38	
40	Romanowski	39	
34	Pawluszyn	40	
4	Czartyszyn	41	
41	Smolecka	41	
37	Powoli	42	grupa 9
36	Pękała	43	
28	Motylkiewicz	44	

### Rozpiętość 5

Zapis powyższego pytania dla rozpiętości 5 jest następujący:

```
SELECT min(staz_pracy), count(*)
FROM pracownicy
GROUP BY otoczenie(staz_pracy, 5) ;
```

Tabelę wynikową z licznosciami wyodrębnionych grup (podobnie jak dla rozpiętości 3) przedstawia tabela 2.5:

Tabela 2.5

Wynikowa tabela dla rozpiętości 5  
(eksperyment 1)

nr	min(staz_pracy)	liczba
1	2	7
2	8	7
3	14	8
4	22	7
5	29	8
6	35	8
7	41	5

Dla rozpiętości grupy równej 5, proces sklejania wierszy przebiega podobnie jak przedstawiono to powyżej dla rozpiętości 3.

**Rozpiętość 10**

Zapis powyższego pytania dla rozpiętości 10 jest następujący:

```
SELECT min(staz_pracy), count(*)
FROM pracownicy
GROUP BY otoczenie(staz_pracy, 10) ;
```

Tabelę wynikową z licznoscią wyodrębnionych grup przedstawia tabela 2.6:

Tabela 2.6

Wynikowa tabela dla rozpiętości 10  
(eksperyment 1)

nr	min(staz_pracy)	liczba
1	2	14
2	12	12
3	26	14
4	37	10

**3. Dane wejściowe rozłożone w pewnych skupiskach**

Dane wejściowe ilustruje tabela 3.1:

Tabela 3.1

Tabela *pracownicy* - eksperyment 2

1	Adamczak	23
2	Aksamit	2
3	Bakanowski	23
4	Czartyszyn	31
5	Czupryna	24
6	Dyduch	3
7	Filipek	24
8	Jablonka	11
9	Jagoda	21
10	Jakubczyk	4
11	Janowski	4
12	Jelonek	6
13	Jemioła	6
14	Kamyk	21
15	Kocot	20
16	Kosiński	12
17	Kowalik	12
18	Kowalski	11
19	Kropla	2
20	Kroplewski	1
21	Kukulik	10
22	Kupeczyk	2
23	Leśniak	13
24	Lis	1
25	Lisecka	2
26	Lizak	20
27	Majewski	3
28	Motylkiewicz	33



29	Niezgoda	3
30	Nowak	1
31	Nowakowski	1
32	Okocimski	25
33	Palik	4
34	Pawluszyn	31
35	Pebianowska	30
36	Pękała	33
37	Powoli	33
38	Ramek	21
39	Raptus	11
40	Romanowski	30
41	Smolecka	32
42	Sowa	10
43	Widuch	3
44	Zajęc	4
45	Zalewska	3
46	Zawoja	6
47	Zebra	21
48	Zientara	14
49	Zioło	14
50	Zyzak	20

Przed samym grupowaniem, dane wejściowe zostają posortowane, dane po posortowaniu przedstawione są w tabeli 3.2.

Tabela 3.2  
Posortowana tabela *pracownicy*  
względem kolumny *staz\_pracy*  
(eksperyment 2)

nr_prac	nazwisko	staz_pracy
20	Kroplewski	1
31	Nowakowski	1
24	Lis	1
30	Nowak	1
2	Aksamit	2
25	Lisecka	2
22	Kupczyk	2
19	Kropla	2
43	Widuch	3
45	Zalewska	3
6	Dyduch	3
27	Majewski	3
29	Niezgoda	3
44	Zajęc	4
10	Jakubczyk	4
33	Palik	4
11	Janowski	4
13	Jemiola	6
46	Zawoja	6
12	Jelonek	6
21	Kukulik	10
42	Sowa	10

18	Kowalski	11
39	Raptus	11
8	Jabłonka	11
16	Kosiński	12
17	Kowalik	12
23	Leśniak	13
49	Zioło	14
48	Zientara	14
50	Zyzak	20
15	Kocot	20
26	Lizak	20
14	Kamyk	21
38	Ramek	21
9	Jagoda	21
47	Zebra	21
1	Adamczak	23
3	Bakanowski	23
5	Czupryna	24
7	Filipek	24
32	Okocimski	25
35	Pebianowska	30
40	Romanowski	30
34	Pawluszyn	31
4	Czartyszyn	31
41	Smolecka	32
37	Powoli	33
36	Pękała	33
28	Motylkiewicz	33

Kolejny etap, to proces tworzenia grup w zależności od podanej rozpiętości:

### Rozpiętość 3

Zapis powyższego pytania dla rozpiętości 3 jest następujący:

```
SELECT min(staz_pracy), count(*)
FROM pracownicy
GROUP BY otoczenie(staz_pracy, 3) ;
```

Tabelę wynikową z licznnością wyodrębnionych grup przedstawia tabela 3.3:

Tabela 3.3

Wynikowa tabela dla rozpiętości 3  
(eksperyment 2)

nr	min(staz_pracy)	liczba
1	1	17
2	6	3
3	10	8
4	14	2
5	20	9
6	24	3
7	30	8

W powyższej tabeli kolumna *min(staz\_pracy)* zawiera dolną granicę grupy.

Proces sklejania wierszy w grupy przebiega podobnie jak zilustrowano to w poprzednim rozdziale.

### Rozpiętość 5

Zapis powyższego pytania dla rozpiętości 5 jest następujący:

```
SELECT min(staz_pracy), count(*)
FROM pracownicy
GROUP BY otoczenie(staz_pracy, 5) ;
```

Tabelę wynikową z licznosciami wyodrębnionych grup (podobnie jak dla rozpiętości 5) przedstawia tabela 3.4:

Tabela 3.4

Wynikowa tabela dla rozpiętości 5  
(eksperyment 2)

nr	min(staz_pracy)	Liczba
1	2	20
2	10	10
3	20	12
4	30	8

### Rozpiętość 10

Zapis powyższego pytania dla rozpiętości 10 jest następujący:

```
SELECT min(staz_pracy), count(*)
FROM pracownicy
GROUP BY otoczenie(staz_pracy, 10) ;
```

Tabelę wynikową z licznosciami wyodrębnionych grup przedstawia tabela 3.5:

Tabela 3.5

Wynikowa tabela dla rozpiętości 10  
(eksperyment 2)

nr	min(staz_pracy)	liczba
1	1	22
2	11	8
3	21	12
4	32	8