

POLITECHNIKA ŚLĄSKA w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki
Instytut Informatyki

Praca Dyplomowa Magisterska

*„Występowanie elementów rozmytych w prostych
i złożonych pytaniach SQL”*

PROMOTOR:
Dr inż. Bożena Małysiak

DYPLOMANT:
Bartosz Dziedzic

GLIWICE 2005

SPIS TREŚCI

Spis rysunków	4
Spis tabel	6
1. Wstęp	7
1.1 Plan Pracy	7
2. Analiza wymagań	8
3. Język SQL	8
3.1 DML	9
3.2 DCL	12
3.3 DDL	14
4. Podstawy logiki rozmytej	16
4.1 Teoria zbiorów rozmytych	16
4.2 Logika rozmyta	17
5. Zapytania rozmyte (Fuzzy SQL)	18
5.1 Logika rozmyta w systemach Baz Danych	19
5.2. Budowa zapytań rozmytych	20
6. Istniejące rozwiązania	23
7. Wybór narzędzi programowych	27
8. Projekt systemu	29
8.1 Ogólny zarys systemu	29
8.2 Architektura systemu	30
8.3 Funkcjonalność systemu	33
9. Specyfikacja wewnętrzna systemu	33
9.1 Wymagania sprzętowe i programowe	33
9.2 System FuzzyQ – VS .NET 2003	34
9.2.1 Przestrzenie nazw .NET	35
9.2.2 Klasy przestrzeni nazw „myconnect”	36
9.2.2.1 Klasa „connect”	36
9.2.2.2 Klasa „InteractiveMovie”	37
9.2.2.3 Klasa „query”	37
9.2.3 Pliki wej/wyj	38
9.3 System FuzzyQ – Flash MX 2004	39
9.4 Analiza ciekawszych zagadnień implementacyjnych	40
9.4.1 Komunikacja C# i Flash	40
9.4.2 Przetwarzanie komunikatów Flash w systemie napisanym w języku C#	43
9.5 Wybrane problemy realizacji systemu	46
10. Interfejs systemu FuzzyQ	47
10.1 Okno główne systemu	48
10.1.1 Górne menu	48
10.1.2 Pasek narzędzi	50
10.1.3 Zakładka „Zapytanie”	51
10.1.4 Zakładki: „Dane wyjściowe”, „Komunikaty”, „Historia”	52
10.2 Okno parametrów połączenia z BD	55

10.3 Kreator zapytań.....	58
11. Uruchamianie i testowanie systemu FuzzyQ.....	64
11.1 Przebieg testowania i uruchamiania na etapie tworzenia systemu FuzzyQ.....	64
11.2 Przebieg testowania i uruchamiania systemu FuzzyQ na etapie wersji „alpha”.....	66
11.3 Wnioski końcowe z testowania i uruchamiania.....	71
12. Podsumowanie.....	72
Załączniki:.....	73
Nr 1: Instalacja systemu FuzzyQ	73
Nr 2: Deinstalacja systemu FuzzyQ	78

Spis rysunków

- Rys. 1 Składnia polecenia insert
- Rys. 2 Przykłady zastosowania polecenia insert
- Rys. 3 Składnia polecenia update
- Rys. 4 Przykłady zastosowania polecenia update
- Rys. 5 Przykład zastosowania polecenia update
- Rys. 6 Składnia polecenia delete
- Rys. 7 Przykłady zastosowania polecenia delete
- Rys. 8 Składnia polecenia select
- Rys. 9 Przykład zastosowania polecenia select
- Rys. 10 Składnia wyrażeń systemowych DCL
- Rys. 11 Składnia polecenia grant
- Rys. 12 Składnia polecenia deny
- Rys. 13 Składnia polecenia revoke
- Rys. 14 Przykłady użycia polecenia grant
- Rys. 15 Przykłady użycia poleceń create table i drop table
- Rys. 16 Przykłady użycia poleceń create index i drop index
- Rys. 17 Przykłady użycia poleceń create table\index i drop table\index
- Rys. 18 Definicja zbioru rozmytego
- Rys. 19 Wykres spalania benzyny w litrach na 100 km
- Rys. 20 Wykres klasyfikacji wzrostu ludzkiego - logika klasyczna
- Rys. 21 Wykres klasyfikacji wzrostu ludzkiego - logika rozmyta
- Rys. 22 Możliwe połączenia w zależności od typu zapytania i typu bazy danych
- Rys. 23 Przykład występowania warunku rozmytego we frazie select
- Rys. 24 Przykłady występowania warunku rozmytego we frazie where
- Rys. 25 Przykład występowania elementu rozmytego we frazie having
- Rys. 26 Przykład występowania elementu rozmytego we frazie group by
- Rys. 27 Przykład występowania wartości rozmytej we frazie order by
- Rys. 28 Definiowanie „dobrej oceny” w programie FuzzyQuery
- Rys. 29 Definiowanie „dobrej frekwencji” w programie FuzzyQuery
- Rys. 30 Prezentacja wyników zapytania w programie FuzzyQuery
- Rys. 31 Architektura systemu - podstawowe funkcje i moduły
- Rys. 32 Diagram przypadków użycia programu FuzzyQ

Rys. 33 Przykładowa struktura danych zapisanych w formacie xml

Rys. 34 Przykładowy skrypt w języku Action Script 2.0 w środowisku Macromedia Flash MX 2004

Rys. 35 Schemat blokowy algorytmu zaszytego w funkcji analizuj

Rys. 36 Widok wszystkich formatek z zaznaczeniem miejsc wywołania z okna głównego

Rys. 37 Okno główne systemu FuzzyQ

Rys. 38 Pasek narzędzi okna głównego aplikacji FuzzyQ

Rys. 39 Zakładka „zapytanie” okna głównego systemu FuzzyQ

Rys. 40 Zakładka „dane wyjściowe” okna głównego systemu FuzzyQ

Rys. 41 Zakładka „komunikaty” okna głównego systemu FuzzyQ

Rys. 42 Zakładka „historia” okna głównego systemu FuzzyQ

Rys. 43 Widok okna parametrów połączenia

Rys. 44 Okno systemu Windows umożliwiające dodanie sterownika ODBC

Rys. 45 Widok okna kreatora nowego źródła danych.

Rys. 46 Widok okna parametrów połączenia ze źródłem danych PostgreSQL

Rys. 47 Okno „Kreatora zapytań” – przykładowy wybór

Rys. 48 Okno „Kreatora zapytań” – wybór opcji normal

Rys. 49 Okno „Kreatora zapytań” – wybór opcji fuzzy

Rys. 50 Okno „Kreatora zapytań” – przykładowe miejsce występowania przycisku end

Rys. 51 Okno „Kreatora zapytań” – przykładowe zapytanie rozmyte zakończone słowem kluczowym order by

Rys. 52 Okno „Kreatora zapytań” – tworzenie zapytania złożonego

Rys. 53 Okno „Kreatora zapytań” – komunikat zakończenia budowy zapytania głównego

Rys. 54 Okno „Kreatora zapytań” – komunikat zakończenia budowy podzapytania

Rys. 55 Schemat bazy danych ZAKŁADY

Rys. 56 Schemat bazy danych STRZELCY

Rys. 57 Kreator zapytań dla przykładu nr.1

Rys. 58 Wynik zapytania z przykładu nr. 1

Dla dodatku nr 1:

Rys. 1 Instalacja systemu FuzzyQ – komunikat rozpoczęcia instalacji

Rys. 2 Instalacja systemu FuzzyQ – komunikat o prawach autorskich

Rys. 3 Instalacja systemu FuzzyQ – wybór folderu instalacji

Rys. 4 Instalacja systemu FuzzyQ – informacje o wolnym miejscu na wszystkich dyskach logicznych

Rys. 5 Instalacja systemu FuzzyQ – potwierdzenie instalacji programu FuzzyQ

Rys. 6 Instalacja systemu FuzzyQ – postęp instalacji programu FuzzyQ

Rys. 7 Instalacja programu FuzzyQ – komunikat zakończenia instalacji programu FuzzyQ

Rys. 8 Ikona programu FuzzyQ

Rys. 9 Widok plików programu FuzzyQ po udanej instalacji w systemie

Rys. 10 Widok plików sterownika ODBC dla bazy PostgreSQL

Dla dodatku nr 2:

Rys. 1 Deinstalacja systemu FuzzyQ – okno wyboru czynności

Rys. 2 Deinstalacja systemu FuzzyQ – Postęp deinstalacji systemu FuzzyQ

Rys. 3 Deinstalacja systemu FuzzyQ – komunikat końcowy

Rys. 4 Deinstalacja systemu FuzzyQ – deinstalacja za pomocą narzędzi systemowych Windows

Spis tabel

Tab. 1 Dodatkowe operatory rozmyte aplikacji FSQL Server

Tab. 2 Tabela podstawowych przestrzeni nazw użytych w projekcie FuzzyQ

Tab. 3 Tabela skrótów klawiszowych systemu FuzzyQ

1.Wstęp

Celem pracy dyplomowej o temacie: „Występowanie elementów rozmytych w prostych i złożonych pytaniach SQL” jest projekt i realizacja systemu dydaktycznego pozwalającego na zapoznanie się z miejscami występowania elementów rozmytych w pytaniach formułowanych w języku SQL. Zapytania rozmyte, których system ten dotyczy, są rozszerzeniem składni bardzo popularnego języka SQL, który jest najbardziej znanym i rozpowszechnionym językiem zapytań do Bazy Danych. Rozbudowanie SQL o dodatkowe instrukcje pozwalające na wprowadzenie elementów rozmytych (nieprecyzyjnych), zwiększa jego możliwości i jest dodatkowym atutem. Z uwagi na coraz większą popularność rozmytych zapytań oraz baz danych, jakie rozmyte SQL, wnosi do wyszukiwania informacji w bazach danych, powstały plany włączenia tego tematu do programu nauczania. System ma umożliwić nauczanie zapytań rozmytych oraz powinien być pomocny w przeprowadzaniu praktycznych zajęć laboratoryjnych z przedmiotu Bazy Danych na Politechnice Śląskiej w Gliwicach.

1.1 Plan Pracy

Praca została podzielona na dwanaście rozdziałów. W rozdziale pierwszym, czyli we wstępie, określono zarys dziedziny której dotyczyć będzie praca oraz przedstawiono krótki opis każdego z jej rozdziałów. W rozdziale drugim przeprowadzona została analiza wymagań aplikacji. Rozdział trzeci to opis języka SQL, opatrzonego licznymi przykładami. Rozdział czwarty jest wprowadzeniem w teorię zbiorów rozmytych oraz przedstawieniem podstaw logiki rozmytej. Wiadomości zawarte w rozdziale czwartym ułatwiają czytelnikowi zrozumienie mechanizmów tworzenia oraz miejsc występowania elementów rozmytych opisanych w rozdziale piątym. Rozdział szósty to przegląd rynku aplikacji i modułów o podobnym działaniu lub zastosowaniu do projektowanego systemu w ramach tej pracy dyplomowej. W rozdziale siódmym znajduje się opis i uzasadnienie wyboru narzędzi wykorzystanych do realizowania systemu FuzzyQ. W rozdziale ósmym zaprezentowano projekt systemu, jego architekturę oraz funkcjonalność. Rozdział dziewiąty przedstawia strukturę wewnętrzną systemu. Zawiera on opis podstawowych klas programu FuzzyQ, przedstawia ciekawsze implementacje oraz wybrane problemy realizacji systemu. Dziesiąty rozdział zawiera opis możliwości systemu, jego współpracę z użytkownikiem, a także opis interfejsu

aplikacji. W rozdziale tym przedstawiono przykładowe scenariusze korzystania z systemu. W rozdziale jedenastym znajduje się opis oraz wnioski wynikające z procesu testowania i uruchamiania systemu FuzzyQ. Dwunasty, ostatni z rozdziałów, zawiera podsumowanie całej pracy, perspektywy oraz możliwości rozwijania projektu w przyszłości. Praca zawiera również dodatki. Oprócz bibliografii są to: instrukcja instalacji i deinstalacji systemu FuzzyQ oraz spis rysunków, tabel, wykresów i przykładów.

2. Analiza wymagań

W trakcie realizacji systemu założono istnienie bazy danych, w której zaimplementowane są dodatkowe funkcje i operatory umożliwiające realizację zapytań rozmytych (Fuzzy SQL). Głównym zadaniem projektowanego systemu jest nauka języka SQL wzbogaconego o nowe instrukcje. Jego głównym celem jest zobrazowanie miejsc występowania elementów rozmytych w składni języka SQL. Dodatkowo program ma wspomóc użytkownika w zbudowaniu szkieletu takiego zapytania, umożliwić przesłanie go do bazy danych oraz odpowiednio zaprezentować zbiór wyników.

Zakładamy, że użytkownik korzystający w przyszłości z tej aplikacji posiadać będzie ogólną znajomość składni języka SQL oraz minimalną wiedzę na temat logiki rozmytej i zapytań rozmytych w postaci znajomości dodatkowych funkcji, operatorów, etc. wynikających z rozszerzenia standardów SQL o FuzzySQL^[2]. Program powinien mieć możliwość łączenia się z każdą bazą danych, jednak, aby korzystanie z niego miało sens, wymagana jest baza danych udostępniająca funkcje oraz operatory rozmyte. Do testowania projektu autor zastosuje skrypty napisane w języku C oraz SQL umożliwiające bazie danych PostgreSQL interpretację zapytań rozmytych (stanowiące fragment rozprawy doktorskiej dr inż. Bożeny Małysiak).

3. Język SQL

Język SQL zaprojektowany i rozwijany przez naukowców firmy IBM stał się w obecnych czasach standardem języków zapytań. Jest wbudowany w większość systemów relacyjnych baz danych. Jest to język, którego konstrukcja opiera się na słowach kluczowych zaczerpniętych z języka angielskiego i jest zbliżona do zdań

budowanych w tym języku. Dzięki temu nawet początkujący użytkownik może w krótkim czasie opanować podstawy SQL umożliwiające mu dostęp do danych oraz do instrukcji sterujących bazą danych. Jednocześnie jego możliwości są na tyle duże, że zaawansowani użytkownicy mogą wykonywać bardzo skomplikowane operacje na bazie danych. Dostęp do baz danych za pomocą języka SQL jest niezależny od platform sprzętowych. Dzięki tym zaletom język SQL szybko stał się standardem w swojej dziedzinie^[3]. Jego polecenia można podzielić na 3 grupy:

- DML (ang. Data Manipulation Language),
- DCL (ang. Data Control Language) ,
- DDL (ang. Data Definition Language) .

3.1 DML

DML to język umożliwiający wszystkim użytkownikom wykonywanie na danych operacji dodawania, usuwania, selekcjonowania oraz edytowania. Odpowiadają za to instrukcje: INSERT, DELETE, SELECT, UPDATE. Powyższe instrukcje można podzielić na dwie zasadnicze grupy. Pierwsza z nich to polecenia odpowiedzialne za modyfikowanie danych, należą do niej instrukcje: INSERT, UPDATE i DELETE. Druga grupa to polecenie SELECT służące wyłącznie wyszukiwaniu danych.

- a) INSERT – służy do wstawiania wierszy do tabeli. Składnia tego polecenia jest następująca:

INSERT [INTO] {nazwa_tabeli|nazwa_widoku} [(lista_kolumn)] **VALUES** lista_wartości

Rys. 1 Składnia polecenia insert

Słowo kluczowe INTO jest opcjonalne, zależy od bazy danych, jednakże ANSI SQL wymaga używania go. Lista kolumn nie jest konieczna, jeżeli podaje się wartości dla każdej z nich. Dane wprowadzane są po kolei do każdej kolumny. Jeżeli liczba podawanych wartości nie odpowiada liczbie kolumn danej tabeli, należy wypisać nazwy kolumn i w takiej samej kolejności będą do tabeli wprowadzane dane wymienione po słowie kluczowym VALUES. Brak wartości dla jakiegokolwiek kolumny spowoduje automatycznie próbę wprowadzenia wartości

NULL. Należy zwrócić uwagę, czy pozostałe kolumny pozwalają na wprowadzenie takiej wartości^[1]. Przykłady użycia polecenia INSERT:

```
INSERT INTO persons VALUES('name1','family name2','country3',34);
INSERT persons VALUES('name1','family name2',NULL,34);
INSERT persons VALUES('name1','family name2',DEFAULT,34);
INSERT persons (name_,age) VALUES('name1',34)
INSERT INTO persons (name_,family_name,country,age)
VALUES(NULL,'family name2',DEFAULT,34)
```

Rys. 2 Przykłady zastosowania polecenia insert

- b) UPDATE – służy do modyfikowania istniejących danych. Składnia polecenia jest następująca:

```
UPDATE {nazwa_tabeli|nazwa_widoku}
SET nazwa_kolumny1= {wyrażenie1 | NULL | DEFAULT | (SELECT) }
[, nazwa_kolumny2= {wyrażenie2 | NULL | DEFAULT | (SELECT) }
[...n]
WHERE {warunki_wyszukiwania}
```

Rys. 3 Składnia polecenia update

Polecenie UPDATE zawiera klauzulę WHERE, która filtruje i określa, czy zmiany mają być przeprowadzone dla wybranych wierszy, czy dla wszystkich. Jeżeli nie występuje słowo kluczowe WHERE, aktualizacja przeprowadzana jest dla wszystkich wierszy^[1]. Proste przykłady użycia funkcji UPDATE:

```
UPDATE persons SET family_name='Kowalski' WHERE id_per='GFYIVG789';
UPDATE cargo SET price=price* 1.22;
UPDATE persons SET prefix=DEFAULT WHERE state='ALASKA';
```

Rys. 4 Przykłady zastosowania polecenia update

Aktualizacje mogą być przeprowadzone nie tylko dla większej liczby wierszy, ale także dla kolumn:

```
UPDATE persons SET name1='Alfred',family_name='Becon',age='18'  
WHERE id_per='1234'
```

Rys. 5 Przykład zastosowania polecenia update

- c) DELETE – służy do usuwania wierszy z tabeli. Ogólna postać instrukcji DELETE jest następująca:

```
DELETE [FROM] {nazwa_tabeli | nazwa_widoku} WHERE {warunki_wyszukiwania}
```

Rys. 6 Składnia polecenia delete

Po zatwierdzeniu tej operacji nie można jej wycofać, chyba że instrukcja zostanie ujęta w blok BEGIN TRAN / COMMIT TRAN. Instrukcja nie usuwa kolumn, jedynie wiersze. Podobnie jak przy instrukcji UPDATE klauzula WHERE określa liczbę wierszy, których dotyczyć ma polecenie. Jej brak powoduje, że z tabeli zostaną usunięte wszystkie wiersze^[1]. Przykłady użycia funkcji DELETE:

```
DELETE FROM persons;  
DELETE FROM persons WHERE wiek > 21
```

Rys. 7 Przykłady zastosowania polecenia delete

- d) SELECT – służy do przeszukiwania danych w celu wyodrębniania pewnej interesującej nas grupy z całości dostępnych danych. Składnia polecenia prezentuje się następująco:

```
SELECT [DISTINCT] <kolumny>  
FROM <tabale>  
[ WHERE ] <warunek>  
[ GROUP BY ] <kolumny>  
[ HAVING ] <warunek>  
[ ORDER BY ] <kolumny>
```

Rys. 8 Składnia polecenia select

Instrukcja SELECT jest najczęściej używanym i jednym z najważniejszych poleceń dostępnych w języku SQL. Najprostsze zapytanie zawiera jedynie słowa kluczowe SELECT oraz FROM i zwraca ono wszystkie wiersze tabeli. Gdy chcemy ograniczyć w jakiś sposób wyselekcjonowane dane, nakładamy na zapytanie warunki filtrujące po słowie kluczowym WHERE, pozostałe słowa kluczowe służą do grupowania wyników oraz odpowiedniej ich prezentacji^[1]. Proste przykładowe zapytanie SELECT wygląda następująco:

```
SELECT lastname,firstname,nationalid,count(skierid) AS il_bad  
FROM patient p,exam e,adresy a WHERE p.id=e.patid AND p.adressid=a.adresid  
AND a.miasto='katowice' AND p.dateob>'1950-01-01' AND p.dateob<'1970-12-12'  
GROUP BY p.lastname,p.firstname,p.nationalid,skierid,patid;
```

Rys. 9 Przykład zastosowania polecenia select

Każde zapytanie budowane za pomocą instrukcji INSERT, UPDATE, DELETE oraz SELECT może być zapytaniem złożonym, a więc w swojej konstrukcji może zawierać podzapytanie. Takie podzapytanie może być samodzielnym zapytaniem pojedynczym albo może zawierać kolejne podzapytanie. Zapytanie może być użyte w innym podzapytaniu np. po instrukcji WHERE lub HAVING, gdzie może wystąpić porównanie w odniesieniu do wyników zwracanych przez podzapytanie. Taka składnia stwarza użytkownikom ogromne możliwości i pozwala na bardzo precyzyjne operacje na bazie danych.

3.2 DCL

DCL to język umożliwiający administratorom bazy danych zarządzanie prawami dostępu zarówno pojedynczych użytkowników, jak i całych grup. Każdy użytkownik posiada swoje prawa dostępu do różnych obiektów znajdujących się w bazie danych. Pozwala to na zapewnienie bezpieczeństwa danym znajdującym się w bazie. Prawa można nadawać wyrażeniom systemowym takim jak: CREATE DATABASE, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW, itp. lub obiektom^[3]. Ogólna postać dla wyrażień systemowych prezentuje się następująco:

GRANT | **DENY** | **REVOKE** {**ALL** | wyrażenie_sys1,wyrażenie_sys2,... }
TO nazwa_konta1,nazwa_konta2....

Rys. 10 Składnia wyrażień systemowych DCL

Natomiast ogólna składnia dla obiektów wygląda nieco inaczej:

- Dla polecenia GRANT:

GRANT { { **SELECT** | **INSERT** | **UPDATE** | **DELETE** }
[,...] | **ALL** [przywileje] }
ON [**TABLE**]nazwa_tabeli [, ...]
TO { nazwa_uzytkownika | **GROUP** nazwa_grupy | **PUBLIC** } [, ...]
[**WITH GRANT OPTION**]
[**AS** {Group | Role}]

Rys. 11 Składnia polecenia grant

- Dla polecenia DENY:

DENY { { **SELECT** | **INSERT** | **UPDATE** | **DELETE** }
[,...] | **ALL** [przywileje] }
ON [**TABLE**]nazwa_tabeli [, ...]
TO { nazwa_uzytkownika | **GROUP** nazwa_grupy | **PUBLIC** } [, ...]
[**CASCADE**]

Rys. 12 Składnia polecenia deny

- Dla polecenia REVOKE:

REVOKE { { **SELECT** | **INSERT** | **UPDATE** | **DELETE** }
[,...] | **ALL** [przywileje] }
ON [**TABLE**]nazwa_tabeli [, ...]
TO { nazwa_uzytkownika | **GROUP** nazwa_grupy | **PUBLIC** } [, ...]
[**CASCADE**]
[**AS** {Group | Role}]

Rys. 13 Składnia polecenia revoke

Polecenie GRANT nadaje przywileje, REVOKE zabiera je im, a DENY zabrania dostępu do wymienionych poleceń. Przywileje nazywają się tak samo jak polecenia DML, bo odnoszą się bezpośrednio do tych poleceń. Przywileje nadawane są dla

konkretnych grup, dla użytkowników lub dla wszystkich za pomocą słowa kluczowego PUBLIC^[1]. Przykłady użycia polecenia GRANT:

```
GRANT insert,update,select,delete ON pacjenci TO lekarze;  
GRANT select ON terminy TO public;  
GRANT CREATE DATABASE, CREATE TABLE  
TO Rysiek, Ania;  
GRANT insert ON terminy TO Joasia WITH GRANT OPTION;
```

Rys. 14 Przykłady użycia polecenia grant

Słowo kluczowe WITH GRANT OPTION występuje jedynie po poleceniu GRANT. Jeżeli jest użyte, pozwala na przekazanie praw nadawania przywilejów do zasobów wymienionych w instrukcji.

3.3 DDL

DDL to język, który pozwala użytkownikom posiadającym odpowiednie uprawnienia na zarządzanie obiektami w bazie danych. Za jego pomocą użytkownik może np. tworzyć i usuwać tabele oraz indeksy. Odpowiednie polecenia służące temu to: CREATE TABLE, DROP TABLE, CREATE INDEX, DROP INDEX. Ogólna postać polecenia wygląda następująco:

Dla tabel:

```
CREATE TABLE nazwa_tabeli { nazwa_kolumny1 typ_danych, n  
nazwa_kolumny2 typ_danych, [, ... ] } [ INHERITS ( tabela_głowna [, ... ] ) ];  
DROP TABLE nazwa1 [, ...] [ CASCADE | RESTRICT ];
```

Rys. 15 Przykłady użycia poleceń create table i drop table

Dla indeksów:

```
CREATE [ UNIQUE ] INDEX nazwa ON tabela { kolumna1,kolumna2,...} ;  
DROP INDEX nazwa1 [, ...] [ CASCADE | RESTRICT ];
```

Rys.16 Przykłady użycia poleceń create index i drop index

Polecenia CREATE tworzą tabele lub indeksy, a DROP usuwają je. Indeksy służą do zoptymalizowania i zwiększenia wydajności dostępu do danych oraz do ich modyfi-

kacji^[1]. Szczególnie często przydatne są przy hurtowniach danych. Konkretnie przykłady korzystania z poleceń CREATE, DROP podane zostały poniżej:

```
CREATE TABLE pacjenci (  
    id                int(4)          primary key,  
    imie              varchar(20)     not null,  
    nazwisko          varchar(30)     not null,  
    data_ur           date            not null,  
    data_odwiedzin    date            default sysdate not null );
```

```
DROP TABLE pacjenci;
```

```
CREATE INDEX index_pacjenci ON pacjenci(id);
```

```
DROP INDEX index_pacjenci;
```

Rys. 17 Przykłady użycia poleceń create table\index i drop table\index

Systemy zarządzania bazami danych (SZBD) korzystające z języka SQL to m.in.:

- DB2,
- Firebird,
- InterBase SQL,
- MS Access,
- MS SQL Server,
- MySQL,
- Oracle,
- PostgreSQL,
- Sybase.

4. Podstawy logiki rozmytej

4.1 Teoria zbiorów rozmytych

W roku 1965 Lotfi Asker Zadeh, znakomity automatyk azerski, profesor uniwersytetu Berkeley w Kalifornii, który otrzymał wiele doktoratów honoris causa, w tym od Politechniki Śląskiej w Gliwicach, wprowadził pojęcie zbiorów rozmytych. Zbiór rozmyty to zbiór par: obiektu i przypisanej mu funkcji przynależności (μ_A)^[4]. Definicję zbioru rozmytego można przedstawić następująco:

zbiorem rozmytym A w przestrzeni X jest zbiór uporządkowanych par:

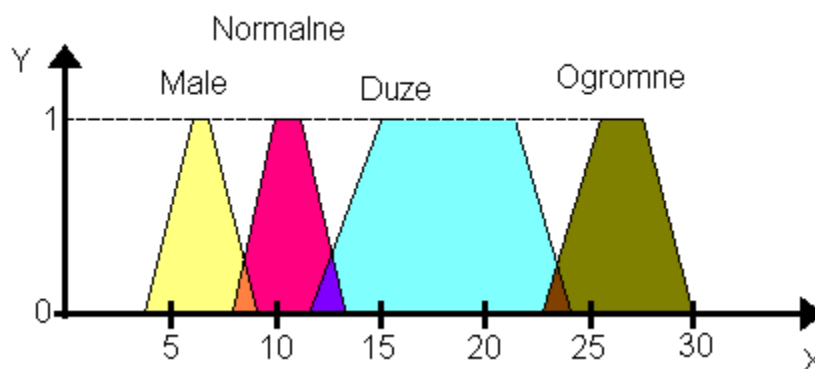
$$A = \{(x, \mu_A(x)) | x \in X\}$$

gdzie: $\mu_A : X \rightarrow [0, 1]$.

Rys. 18 Definicja zbioru rozmytego

Definicja zbioru rozmytego różni się od definicji zbioru klasycznego tym, że funkcja przynależności (μ_A) zbioru rozmytego przybiera wartości z ciągłego przedziału $<0,1>$, a nie, jak to jest w klasycznej definicji, jedną z wartości dwuelementowego zbioru $\{0,1\}$. Innymi słowy, zbiór rozmyty dopuszcza częściową przynależność do zbioru, a to, w jakim stopniu element należy lub nie, określa funkcja przynależności (μ_A). Jeżeli wartość funkcji wynosi 0, to znaczy, że dany element nie należy do tego zbioru, jeżeli wynosi ona 1, to dany element całkowicie do niego należy, jeśli natomiast jest większa od 0, ale mniejsza niż 1, to element taki należy do tego zbioru, ale w pewnym stopniu^[5].

Przykładem zbioru rozmytego może być zbiór samochodów z wyróżnioną cechą „spalanie benzyny w litrach na 100km”. Funkcje przynależności kilku wyróżnionych podzbiorów rozmytych, takich jak: „małe spalanie benzyny”, „średnie spalanie benzyny”, „duże spalanie benzyny” oraz „ogromne spalanie benzyny”, prezentuje następujący wykres:



Rys. 19 Wykres spalania benzyny w litrach na 100 km

Na osi odciętych zaprezentowano spalanie benzyny w litrach na 100km, a na osi rzędnych stopień przynależności. Określenia: „Małe”, „Normalne”, „Duże” oraz „Ogromne” to oceny (wartości) lingwistyczne, zmiennej lingwistycznej „spalanie benzyny”. Zmienna lingwistyczna jest to wielkość, do określenia której stosuje się wartości lingwistyczne^[5]. Jest to więc zmienna, która przyjmuje słowa, jako wartości. Słowa te określane są jako wartości zmiennej lingwistycznej, są to podzbiory rozmyte zbioru głównego. Na rysunku (rys. 19) zbiorem głównym jest: „spalanie benzyny”. Wartości lingwistyczne jakie przyjmuje to właśnie słowa: „Małe”, „Normalne”, „Duże” i „Ogromne”. Podobnych przykładów w życiu codziennym możemy wskazać mnóstwo, zmienne lingwistyczne to np.: waga, wzrost, zużycie materiału, itp. Odpowiadające im oceny lingwistyczne to: niedowaga, nadwaga, otyłość, niski, wysoki, małe, duże, ogromne, itp.

4.2 Logika rozmyta

Osiem lat po wprowadzeniu teorii zbiorów rozmytych, Lotfi Zadeh wprowadził pojęcie logiki rozmytej^[3]. Jest to logika wielowartościowa stanowiąca uogólnienie logiki dwuwartościowej. Logika klasyczna dopuszcza istnienie dwóch wartości 1 (prawda) lub 0 (fałsz). Logika rozmyta wprowadza dodatkowe wartości pośrednie, czyli cały przedział liczb rzeczywistych pomiędzy zerem a jedynką^[2]. Logika Zadeha jest bliższa rozumowaniu rzeczywistemu, w którym pojawia się bardzo dużo niepewności, czyli tzw. wartości środka^[4]. Przykładem może być klasyfikacja ludzi ze względu na ich wysokość, czyli po prostu wzrost. W klasycznej logice musimy określić sztywne granice, a więc np. od 0 do 1,2m będą to ludzie karłowaci, od 1,2m do 1,5m

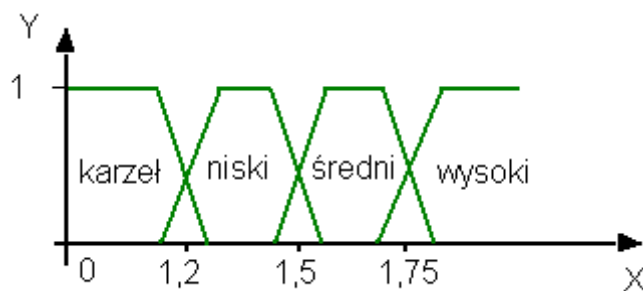
- ludzie niski, od 1,5m do 1,75m - ludzie średniego wzrostu i powyżej 1,75m - ludzie wysocy.



Rys. 20 Wykres klasyfikacji wzrostu ludzkiego - logika klasyczna

Na osi OX naniesiony został wiek, a na osi OY stopień przynależności.

Jednak gdybyśmy przeprowadzili badania i kazali grupie osób zakwalifikować wzrost 1,7m do jednej z grup: karzeł, niski, średni, wysoki zdania byłyby podzielone. Część osób przydzieliłaby go do osób średnich, a część do wysokich.



Rys. 21 Wykres klasyfikacji wzrostu ludzkiego - logika rozmyta

W tym przypadku granice uległy rozmyciu. Jak widać na tym przykładzie logika rozmyta jest bardziej naturalna i zbliżona do ludzkiego pojmowania rzeczywistości niż logika klasyczna.

5. Zapytania rozmyte (Fuzzy SQL^[2])

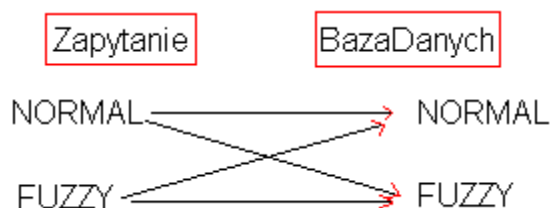
Język SQL rozszerzony o możliwość tworzenia warunków rozmytych oraz stosowania operatorów rozmytych pozwala na interpretację oraz wykorzystywanie zapytań rozmytych. Język SQL jest uważany za bardzo naturalny, ponieważ swoją składnią zbliżony jest do języka angielskiego, jednak w swojej postaci uwzględnia

wyłącznie konkretne wartości. W rzeczywistości, chcąc wyszukać jakieś dane, np. pracowników ze stażem pracy około 25 lat, musimy w warunku nałożonym na frazę „SELECT” użyć konkretnej wartości określającej staż pracy wynoszący 25 lat, mimo iż pracownicy ze stażem pracy 24 oraz 26 lat również pozostają w swerze naszych zainteresowań. Takie i inne możliwości dają nam właśnie funkcje i operatory rozmyte, o które poszerzany jest język SQL.

5.1 Logika rozmyta w systemach Baz Danych

Rozszerzony język SQL pozwala stosować elementy zarówno „ostre”, jak i „rozmyte”. To samo dotyczy baz danych, do których zapytania rozmyte są kierowane - mogą one zawierać wartości „ostre”, jak również „rozmyte”.

W związku z tym wyróżniamy cztery możliwości odpytywania bazy danych:



Rys. 22 Możliwe połączenia w zależności od typu zapytania i typu bazy danych

a) Zapytanie „ostre” wysłane do bazy danych zawierającej „ostre” dane (NORMAL-NORMAL):

Jest to przypadek standardowy. Zapytanie zawiera dokładne wartości atrybutów określających poszukiwane dane i baza danych zawiera również dokładne dane^[5].

b) Zapytanie „ostre” wysłane do bazy danych zawierającej niedokładne (rozmyte) dane (NORMAL-FUZZY):

Niedokładne dane w bazie danych to np. wartości wyrażone w języku naturalnym czyli, zakładając, że w bazie istnieje kolumna „wzrost”, wartości, jakie się w niej pojawiają, to np.: „niski”, „wysoki”, „bardzo niski”, „średni”, itp. Wartości atrybutów mogą również być określone jako przedziały albo zmienne

rozmyte, czyli w kolumnie „wzrost” mogą się pojawić także inne wartości, takie jak: „około 180”, „189-192”, itp^[5].

c) Zapytanie „rozmyte” wysyłane do bazy zawierającej „ostre” dane (FUZZY-NORMAL):

Baza danych zawiera dokładne dane, ale zapytanie wysłane do bazy danych jest nieprecyzyjne (zawiera rozmyte warunki). Sytuacja taka ma miejsce, gdy nie chce się otrzymać konkretnych wyników, natomiast interesują nas np. wartości oscylujące w jakimś punkcie^[5].

d) Zapytanie „rozmyte” wysyłane jest do bazy danych zawierającej „rozmyte” dane (FUZZY-FUZZY):

Baza danych zawiera niedokładne (rozmyte) dane oraz pytanie kierowane do bazy danych jest nieprecyzyjne (zawiera rozmyte warunki). W wyniku otrzymujemy jednak konkretny zestaw danych, tak jak w pozostałych trzech przypadkach^[5].

5.2. Budowa zapytań rozmytych

Wartości rozmyte mogą pojawić się w zapytaniu SQL po słowach kluczowych: SELECT, w warunkach filtrujących po frazach WHERE oraz HAVING, a także we frazie grupującej GROUP BY, czy porządkującej ORDER BY^[6]. Rozszerzenie języka SQL o elementy rozmyte ułatwiło przenoszenie zadań sformułowanych w języku naturalnym na postać zapytań języka SQL. Zadania w swej pierwotnej postaci mogą zawierać wartości nieprecyzyjne oraz określenia, które wcześniej musiały być poddawane korektom, aby dostosować je do możliwości standardowego SQL-a. Autor przedstawił formę zapisu oraz operatory i funkcje rozmyte zdefiniowane przez dr inż. Bożenę Małysiak^[5].

a) Wartości rozmyte we frazie SELECT:

W zapytaniu po słowie kluczowym SELECT mogą wystąpić wartości rozmyte w postaci jawnych wyrażeń, jak również ukryte w funkcjach agregujących. Przykład elementu rozmytego we frazie SELECT^[2]:

```
SELECT (spalanie_benzyny~=około 20) AS st_przynależności FROM samochody WHERE marka='Renault';
```

Rys. 23 Przykład występowania warunku rozmytego we frazie select

b) Wartości rozmyte w warunku filtrującym WHERE:

We frazie WHERE może wystąpić kilka warunków połączonych spójnikami logicznymi OR i AND. Proces wyznaczania stopnia przynależności może przebiegać dla każdego z nich oddzielnie lub z wykorzystaniem sumy i iloczynu rozmytego. Iloczyn rozmyty wyznaczy wartość minimalną dwóch stopni przynależności (S-norma Zadeha), a suma rozmyta wyznaczy nam wartość maksymalną dwóch stopni przynależności (T-norma Zadeha)^[6]. Przykłady występowania elementów rozmytych we frazie WHERE^[2]:

```
SELECT imie,nazwisko FROM osoby  
WHERE ( ( wiek~=około15 ) &&& ( wzrost~=około170 ) ) >0.8;
```

```
SELECT imie,nazwisko FROM osoby  
WHERE ( ( wiek~=około15 ) ||| ( wzrost~=około170 ) ) >0.8;
```

```
SELECT imie,nazwisko FROM osoby  
WHERE ( ( wiek~=około15 ) >0.6 ) AND ( ( wzrost~=około170 ) >0.8 );
```

Rys. 24 Przykłady występowania warunku rozmytego we frazie where

c) Wartości rozmyte w warunku filtrującym HAVING:

Proces wprowadzania i wyznaczania warunków rozmytych we frazie HAVING przebiega analogicznie jak we frazie WHERE. Jeżeli wartości rozmyte występują w obu frazach jednocześnie w jednym zapytaniu, najpierw wyznaczane są dane spełniające warunki podane we frazie WHERE, następnie na wyselekcjonowaną grupę wyodrębnionych danych nakładane są warunki

z frazy HAVING^[6], wcześniej może wystąpić grupowanie. Przykład zapytania z warunkiem rozmytym we frazie HAVING^[2]:

```
SELECT imie,nazwisko,wiek,wzrost FROM osoby  
WHERE (wiek~=okolo15) HAVING (wzrost~=okolo170);
```

Rys. 25 Przykład występowania elementu rozmytego we frazie having

d) Wartości rozmyte w warunku grupującym GROUP BY:

Grupowanie wyników według kolumn zawierających wartości rozmyte może odbywać się na podstawie wartości modalnych liczb rozmytych, jeśli liczby te przedstawione są w reprezentacji LR, lub przedziału wartości modalnych, gdy wartości rozmyte reprezentowane są jako przedziały, a nie jako liczby^[6]. Przykład występowania wartości rozmytej we frazie GROUP BY może być następujący^[2]:

```
SELECT imie, nazwisko, (wiek~=okolo20) AS st_p FROM osoby  
WHERE (wiek~=okolo20) >= 0.7 GROUP BY (wiek~=okolo20);
```

Rys. 26 Przykład występowania elementu rozmytego we frazie group by

e) Wartości rozmyte w warunku porządkującym ORDER BY:

Sortowanie wyników zawierających rozmyte dane może odbywać się w podobny sposób jak przy grupowaniu. Jeżeli dane sortowane zawierają liczby rozmyte, wtedy porównywane są wartości modalne liczb, natomiast w przypadku przedziałów rozmytych porównujemy górne granice, gdy szukamy największej, albo dolne, gdy najmniejszej^[6]. Przykład występowania wartości rozmytej we frazie ORDER BY^[2]:

```
SELECT imie,nazwisko, (wiek~=okolo20) AS st_p  
FROM osoby ORDER BY (wiek~=okolo20)
```

Rys. 27 Przykład występowania wartości rozmytej we frazie order by

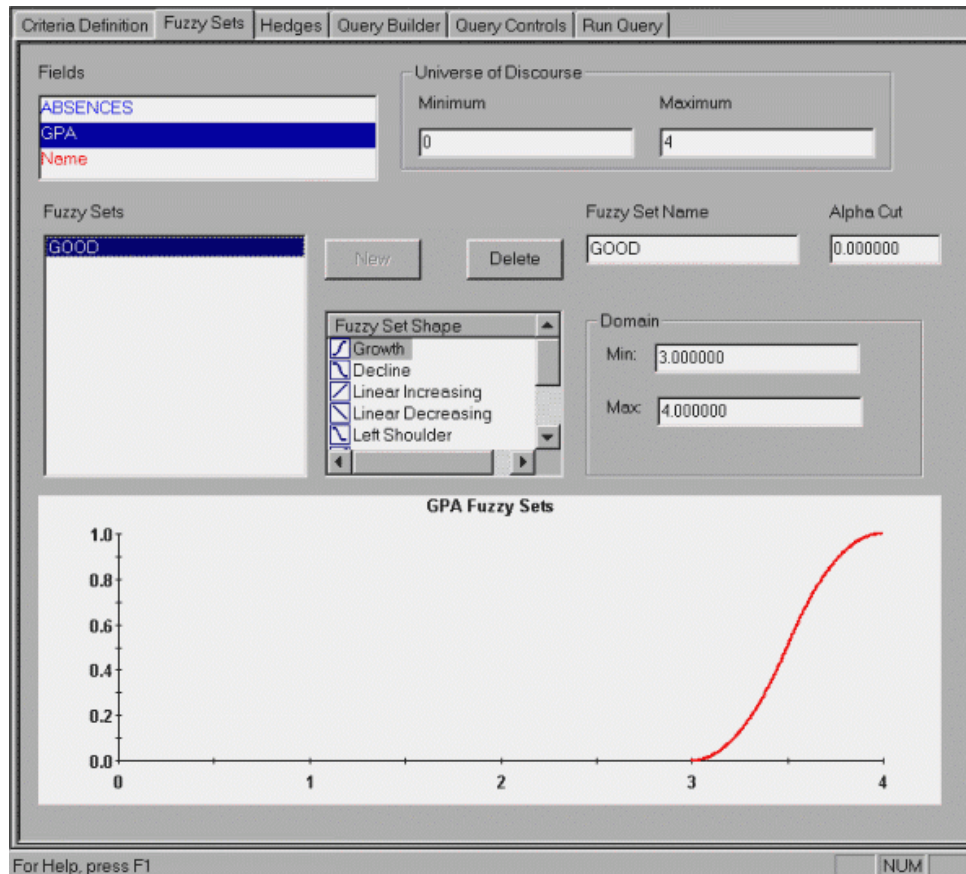
Rozszerzenie języka SQL o elementy logiki rozmytej pozwala nam na formułowanie zapytań w języku naturalnym bez dokładnego precyzowania warunków.

6. Istniejące rozwiązania

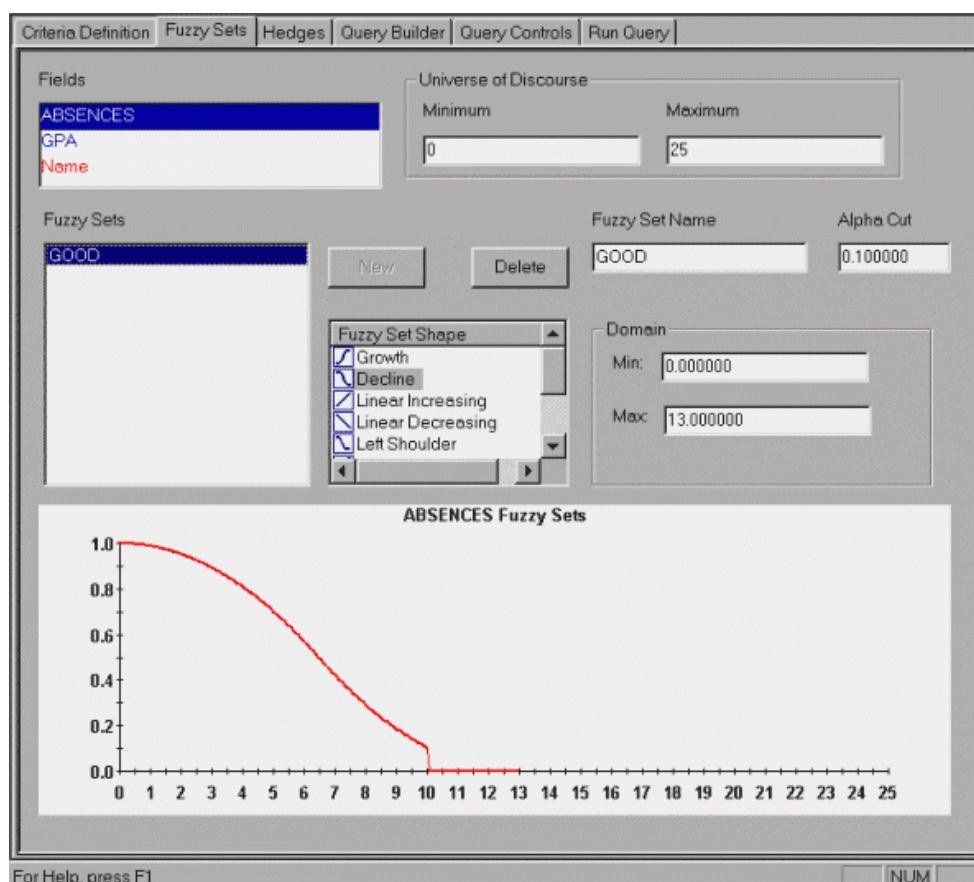
Analizując pod kątem dydaktycznym rynek darmowych i komercyjnych programów, nie udało mi się znaleźć żadnej aplikacji, która wskazywałaby na miejsca występowania elementów rozmytych i ułatwiałaby opanowanie rozszerzonego języka SQL. Jest wysoce prawdopodobne, że takie aplikacje istnieją, jednak działają w obrębie ośrodków naukowych i nie są przez nie rozpowszechniane. Po bezowocnych poszukiwaniach autor zmienił obiekt poszukiwań na aplikacje uczące języka SQL. Tym razem wynik był znacznie lepszy, choć daleki od oczekiwań. Spodziewano się, że aplikacje wspomagające budowanie zapytań formułowanych w języku SQL będzie bardzo dużo, biorąc pod uwagę popularność (miliony użytkowników na całym świecie), funkcjonalność oraz niepodważalną bezkonkurencyjność tego języka w zastosowaniu z relacyjnymi bazami danych. Na wyróżnienie zasługuje projekt polskiego wydawnictwa „Wydawnictwa Szkolne i Pedagogiczne”, który utworzył aplikację umożliwiającą odbycie kursów on-line. Niestety, jest to projekt komercyjny i dostępna jest jedynie wersja demo produktu, aby można było poznać wady i zalety programu, zanim zdecydujemy się na jego kupno. Kurs ten stał się podstawą do utworzenia Polskiego Uniwersytetu Wirtualnego, co świadczy o wysokiej jakości dydaktycznej. Być może w przyszłości treść kursu poszerzona zostanie o temat występowania elementów rozmytych w pytaniach SQL. Pomimo, że nie ma wiele programów dydaktycznych uczących języka SQL, w sieci Internet można znaleźć bardzo dużo materiałów teoretycznych popartych przykładami.

Druga część analizy dotyczyła systemów korzystających z zapytań rozmytych. Jedną z kilku firm tworzących oprogramowanie systemowe w oparciu o logikę rozmytą jest firma Sonalysts Inc. Dział Fuzzy System Solutions zajmuje się rozwijaniem i wdrażaniem oprogramowania przetwarzającego dane oparte na zapytaniach rozmytych. Program nosi nazwę Fuzzy Query i kosztuje 60 dolarów, co nie jest ceną wysoką, biorąc pod uwagę amerykańskie realia. Daje on nam duże możliwości ingerencji w parametry wejściowe systemu. W zamieszczonym w pracy przykładzie zadaniem systemu było znalezienie najlepszych uczniów, którzy osiągnęli dobre wyniki w nauce oraz dobrą frekwencję na zajęciach. Na rysunkach przedstawiona jest zakładka FuzzySet (rys. 28 i rys. 29), umożliwiająca definiowanie „dobrych osiągnięć” (rys. 28) oraz „dobrej frekwencji” (rys. 29). Kolejny rysunek prezentuje dane wynikowe zwrócone przez bazę danych i przefiltrowane przez system FuzzyQuery (rys. 30).

W zakładce FuzzySet podajemy parametry, które mają na celu określić funkcję przynależności, aby przy analizie wyników zwróconych przez zapytanie wyróżnić dokładnie te, które są interesujące.



Rys. 28 Definiowanie „dobrej oceny” w programie FuzzyQuery



Rys. 29 Definiowanie „dobrej frekwencji” w programie FuzzyQuery

W tabeli wyników (rys. 30) ostatnia kolumna [QCIX] prezentuje wypadkową obu czynników (dobra ocena + dobra frekwencja). Im bliższa jest ona wartości 1 tym lepszy uczeń.

Criteria Definition Fuzzy Sets Hedges Query Builder Query Controls Run Query				
Run Query				
Name of results table: FzyQryAnswer				
[Name]	[GPA]	[ABSENCES]	[QCIX]	
Mike Salisbury	4.000000	1.000000	0.994492	
Barry Allen	3.900000	2.000000	0.966476	
Becky Springston	4.000000	3.000000	0.946884	
Kevin Costner	3.750000	1.000000	0.931992	
Shelly Whitman	3.750000	4.000000	0.844666	
Leroy Jefferson	3.750000	4.500000	0.819336	
Alice Cook	3.490000	0.000000	0.738419	
John Conner	4.000000	7.000000	0.716080	
Bobby Smith	3.450000	1.000000	0.696289	
Billy Kidd	4.000000	9.750000	0.562500	
Orville Wright	4.000000	10.000000	0.554932	
Tim Evans	3.990000	10.000000	0.554794	
Adrian Barbough	3.400000	5.000000	0.512207	
Jeff Williams	3.500000	9.750000	0.312500	
Sherlock Holmes	3.210000	9.500000	0.072647	

Rys. 30 Prezentacja wyników zapytania w programie FuzzyQuery

Na stronie firmy Sonalysts inc. „<http://fuzzy.sonalysts.com/fuzzysql1.htm>” znajduje się opis działania języka SQL poszerzonego o elementy rozmyte, z którego korzysta aplikacja Fuzzy Query. Można dowiedzieć się z niego na czym polega i czym się różni zapytanie rozmyte, od zwykłego zapytania SQL. Istnieje kilka firm, które podobnie jak Sonalysts Inc. zajmują się sprzedażą i wdrażaniem programów, które opierają się na zapytaniach rozmytych. Są to jednak rozwiązania komercyjne, a więc odpłatne.

Z tego względu wartym poświęcenia uwagi jest rozwiązanie udostępniane przez dr Jose Galino Gomeza pracownika uniwersytetu w Maladze. Jest ono ogólnie dostępne na stronie „<http://www.lcc.uma.es/~ppgg/FSQL.html>”. Składa się z dwóch części: pierwsza to FSQL Server, udostępniający dodatkowe funkcje i operatory rozmyte dla bazy danych firmy ORACLE, natomiast druga to aplikacja klienta pozwalająca na zastosowanie rozmytych zapytań. W poniższej tabeli przedstawiono niektóre z nowych operatorów języka SQL^[12]:

<u>OPERATOR Fuzzy</u>	<u>OPERATOR Normalny</u>	<u>OPIS</u>
FEQ	=	FuzzyRówność
FGT	>	FuzzyWiększość
FGEQ	=>	FuzzyWiększość-równość
FLT	<	FuzzyMniejszość
FLEQ	<=	FuzzyMniejszość-równość
MGT	>>	FuzzyWieleWięcejNiż
MLT	<<	FuzzyWieleMniejNiż

Tab. 1 Dodatkowe operatory rozmyte aplikacji FSQL Server

Przykładowy problem wraz z rozwiązaniem korzystającym z powyższych operatorów:

Zadanie: „Wyszukaj wszystkie osoby z kręconymi (\$Curly) włosami ze stopniem przynależności większym niż 0.5 oraz wzroście określonym jako wysoki (\$Tall) ze stopniem przynależności większym niż 0.8”

Rozwiązanie: *SELECT * FROM Person WHERE (Hair FEQ \$Curly THOLD 0.5) AND (Height FGT \$Tall THOLD 0.8);*

W powyższym rozwiązaniu znaczek \$ oznacza, że słowo następujące po nim jest wartością zmiennej lingwistycznej^[12]. Operator serwera FSQL opisujący równość dwóch wartości rozmytych przedstawiony jest jako „FEQ”. Hiszpański FSQL posiada specjalną funkcję THOLD służącą do badania stopnia przynależności do zbioru.

Jak widać mechanizmy tworzenia zapytań rozmytych są tematem wielu badań naukowych na całym świecie, dlatego kształcenie studentów w tej dziedzinie może być dla nich przydatnym doświadczeniem. Do tego właśnie ma przyczynić się ta praca dyplomowa.

7. Wybór narzędzi programowych

W dzisiejszych czasach istnieje ogromny wybór języków programowania oraz środowisk programistycznych. Wybierając narzędzie programistyczne, autor kierował się tym, by możliwe było szybkie utworzenie zaawansowanej aplikacji. Zwracano uwagę na ilość i dostępność dokumentacji dotyczącej tak samego środowiska, jak i języka programowania. Zdecydowano się napisać system w środowisku Visual Studio .Net 2003, ponieważ spełnia wszystkie założone przez autora warunki i znacznie przewyższa oczekiwania. Jedyną wadą tego środowiska jest to, iż nie jest darmowe. Owszem, istnieją takowe, jednak zdaniem autora nie dorównują Visual Studio .Net 2003 swoją funkcjonalnością oraz możliwościami.

Firma Microsoft, która jest autorem tego środowiska, zbudowała je od podstaw, łącząc w sobie wszystkie zalety kilku innych, istniejących wcześniej na rynku narzędzi programistycznych. Przy jednoczesnym odrzuceniu największych wad oraz utrudnień jakie one posiadały, powstało narzędzie wprost idealne dla programistów, warte w 100% swojej ceny. Dodatkowym atutem środowiska Visual Studio .Net jest wspieranie go w ramach projektu Microsoft Developer Network. MSDN jest zbiorem usług dostępnych m.in. przez Internet, które powstały, aby wspomagać zarówno pojedynczych programistów, jak i duże fabryki oprogramowania w zdobywaniu wiedzy na temat produktów firmy Microsoft. MSDN to bogata dokumentacja oraz pomoc ekspertów, która bardzo ułatwia zrozumienie i wdrożenie nowych, bardzo dobrych technologii. Ogólny dostęp do tak dużej wiedzy sprawia, iż jest to ogromny atut Visual Studio .Net. Korzystając z możliwości użycia wielu kreatorów, kontrolek i innych pomocy projektowych, możemy uzyskać znaczne zwiększenie efektywności projektowania i tworzenia aplikacji^[9].

W wyżej opisanym środowisku mamy dostęp do wielu popularnych języków programowania, m.in. takich jak:

- C++,
- Java ,
- Visual Basic.

Środowisko Visual Studio .NET 2003 posiada również nowy, w pełni obiektowy język programowania, będący niejako hybrydą dwóch bardzo popularnych języków: Java i C++. Nowy język programowania to „C sharp”, oznaczany jako C#. Łączy on w sobie zalety obu wcześniej wymienionych języków, a w połączeniu z nowym, bardzo dobrym środowiskiem Visual Studio .Net 2003 zdaniem autora jest bezkonkurencyjny. Mając dostęp do tak potężnego narzędzia, zdecydowano się napisać system właśnie w tym języku.

Składnia języka C# oparta jest na składni popularnego języka C++ ,a więc obejmująca przeciążanie operatorów, typy wyliczeniowe, rozróżnianie wielkich i małych liter. Dodatkowo wzbogacona została cechami obiektowymi takimi, jak atrybuty, właściwości czy zdarzenia. Programiści korzystający z języka C# mają do dyspozycji składnię z dodatkowymi zaletami w stosunku do języka C++ takimi, jak większa wydajność, zarządzalność, a także bezpieczeństwo kodu^[9].

Mając na uwadze dydaktyczny charakter aplikacji, zdecydowano się część odpowiedzialną za przedstawienie miejsc występowania wartości rozmytych zrealizować w języku Flash 7.0. Filmy interaktywne napisane w tym języku doskonale nadają się do prezentacji i w bardzo przejrzysty sposób pomagają wyjaśniać nawet bardzo złożone zagadnienia, praktycznie z każdej dziedziny. Animacje tworzone w języku Flash opierają się na grafice wektorowej, dzięki czemu są bardzo łatwo skalowalne. Pliki te są również bardzo małe, co jest dodatkowym atutem. Język Flash w wersji 6.0 lub wyższej został wzbogacony o możliwość programowania w języku skryptowym zwanym ActionScript. Znacznie zwiększyło to jego możliwości, gdyż wprowadziło do tego języka elementy obiektowe.

Obecnie na rynku istnieje kilka środowisk umożliwiających programowanie w języku Flash 6.0. Są to środowiska zarówno komercyjne, jak i zupełnie darmowe, możliwe do ściągnięcia z sieci Internet. Ze względu na możliwości, najlepszym narzędziem do programowania w języku Flash 6.0 jest Macromedia Flash MX 2004. Zdecydowano się skorzystać właśnie z tego środowiska, ponieważ uznano je za najlepsze z wszystkich dostępnych na rynku. Macromedia jest firmą, która przejęła li-

cencję pierwszej wersji środowiska i zainwestowała w jego rozwój, dzięki czemu ewoluował on do obecnej wersji. Na całym świecie firma ta jest bardzo popularna, a technologie Flash wykorzystuje się m.in. do tworzenia:

- Stron WWW,
- Sieciowych gier,
- Animowanych kreskówek,
- Wideoklipów muzycznych,
- Interaktywnych animacji,
- Prezentacji multimedialnych.

Odtwarzacz plików Flash jest bardzo popularny, darmowy i powszechnie dostępny poprzez witrynę WWW firmy Macromedia.

Jednym z założeń realizowanego systemu jest połączenie go z bazą danych, przesyłanie zapytań do bazy i odbieranie oraz prezentacja wyników zapytania. System będzie wykorzystywał interfejs ODBC, a więc będzie możliwość połączenia go z każdą bazą danych udostępniającą ten interfejs. Spośród narzędzi potrzebnych do uruchomienia oraz administracji bazą danych wybrano PostgreSQL w wersji 8.0.1 na platformę Windows oraz pgAdmin III. Na wybór ten zasadniczy wpływ miał fakt, że na tej właśnie bazie danych powstała praca doktorska dr inż. Bożeny Małysiak, udostępniająca dodatkowe instrukcje SQL, pozwalające na tworzenie zapytań rozmytych. Podsumowując uznano język C# oraz środowisko Visual Studio .Net za najlepsze i najbardziej odpowiednie do realizacji systemu przedstawionego w pracy dyplomowej. Dodatkowo autor zamierza skorzystać z możliwości tworzenia bogatych, interaktywnych prezentacji w języku flash 6.0 za pomocą narzędzi Macromedia Flash MX i dołączyć je do systemu. Na pewno pomoże to zobrazować miejsca występowania wartości rozmytych w pytaniach SQL, co jest głównym celem tego projektu.

8. Projekt systemu

8.1 Ogólny zarys systemu

Głównym zadaniem systemu ma być nauka języka FuzzySQL^[2]. Aby to zrealizować, system musi zwrócić uwagę użytkownika na miejsca występowania i sposób budowy warunków rozmytych w pytaniach SQL. Zapytanie utworzone przez użytkownika po-

winno być wysłane do bazy danych z zaimplementowanymi funkcjami interpretującymi polecenia i operatory rozmyte. Taka baza danych nie jest przedmiotem tej pracy, jednak jest dla systemu niezbędna z logicznego punktu widzenia. System, którego zadaniem jest nauka konstruowania zapytań rozmytych bez możliwości sprawdzenia poprawności i praktycznego wykorzystania zapytań rozmytych nie ma większego sensu. Z tego powodu od początku głównym założeniem jest współpraca systemu z bazą danych umożliwiającą realizację zapytań rozmytych. Możliwości połączeń z bazą danych jest wiele, najodpowiedniejszym dla tego systemu jest uniwersalny sterownik ODBC. Pozwala on na absolutną dowolność w doborze bazy danych, ponieważ wszystkie znane bazy danych posiadają interfejs ODBC.

8.2 Architektura systemu

Budowę systemu, który będzie nosił nazwę FuzzyQ, można podzielić na dwie części. Pierwsza z nich to analizator zapytań, który obejmować będzie narzędzie służące do komunikacji z bazą danych. Będzie to część systemu napisana za pomocą narzędzia Visual Studio .NET w języku C#. Ma ona realizować następujące funkcje:

- Połączenie z bazą danych,
- Przesłanie zapytania w języku SQL/FuzzySQL^[2] do bazy danych,
- Odbiór wyniku przesłanego zapytania,
- Prezentację danych zwróconych przez zapytanie,
- Zapis wyników do pliku,
- Współpracę z kreatorem zapytań.

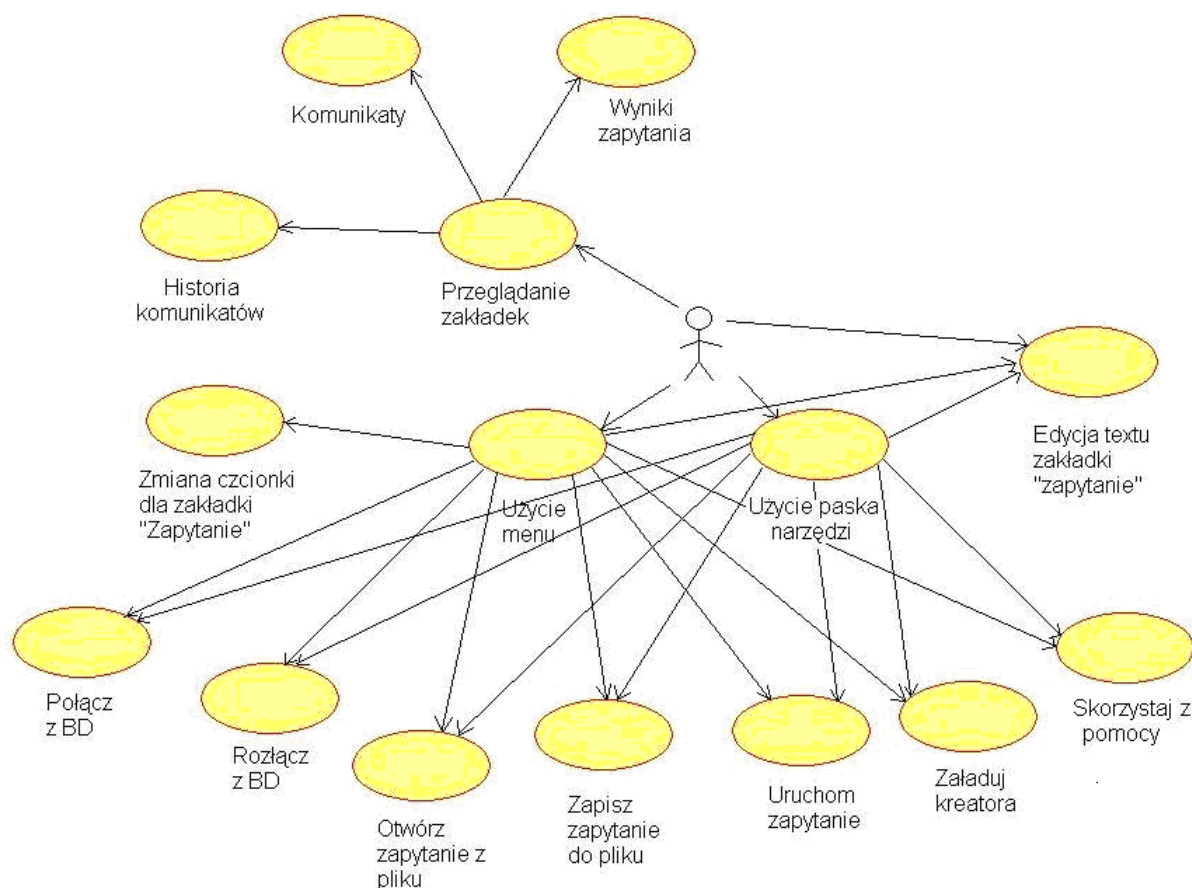
Druga część systemu to część dydaktyczna. Będzie to „kreator zapytań”; narzędzie, które pomoże w budowie zapytania rozmytego. Głównym zadaniem tej części systemu będzie oddziaływanie na użytkownika tak, aby w krótkim czasie potrafił wskazać miejsca występowania elementów rozmytych w prostych i złożonych pytaniach SQL oraz samodzielnie potrafił takie zapytanie skonstruować. Część ta powinna współpracować z „analizatorem zapytań”, aby użytkownik mógł na bieżąco sprawdzać działanie poszczególnych zapytań. Zapytanie zbudowane za pomocą kreatora ma być jedynie szkieletem i prezentować postać ogólną zapytania. Zbudowane zdanie w języku FuzzySQL^[2] zostanie w momencie zakończenia budowy przekazane do modułu analizującego zapytania.

Przed wysłaniem zapytania do bazy danych niezbędna będzie ingerencja użytkownika, który dostosuje je do własnych potrzeb albo po zapoznaniu się ze schematami tworzenia takich zapytań, zawartych w kreatorze, zbuduje je samodzielnie od podstaw. Dostosowanie zapytania będzie uzależnione od bazy danych, tabel i wartości w nich zawartych oraz od treści zadania. Po drobnych przeróbkach i dostosowaniu zapytania do poprawnej postaci użytkownik będzie mógł przetestować je i zobaczyć wynik, który zostanie zwrócony w wyniku wykonania zapytania. Gdyby, mimo wszelkich starań, zapytanie zawierało jakiegokolwiek błędy, zostaną one zakomunikowane użytkownikowi, aby ten mógł poprawić je i spróbować przesłać ponownie zapytanie do bazy danych.

Graficzne przedstawienie podstawowych funkcji i modułów systemu przedstawia rysunek 31.

8.3 Funkcjonalność systemu

Funkcje, które system ma realizować - widziane z poziomu użytkownika zaprezentowano na diagramie (rys.32).



Rys. 32 Diagram przypadków użycia programu FuzzyQ

9. Specyfikacja wewnętrzna systemu

9.1 Wymagania sprzętowe i programowe

Program FuzzyQ został napisany na platformę systemową Windows. Można go zainstalować zarówno w systemie Windows 98 jak i w systemach z rodziny NT (Windows 2000 ,Windows XP).

Ze względu na to, że aplikacja została utworzona w technologii .NET, do jej poprawnego działania wymagany jest zainstalowany w systemie .NET framework w wersji 1.1. Instalator systemu FuzzyQ wykrywa jego nieobecność i proponuje przed zainstalowaniem systemu instalację frameworka. W systemie Windows wymagany jest również zainstalowany moduł Microsoft Data Access Komponent w wersji co najmniej 2.6. MDAC jest instalowany razem z wieloma popularnymi programami firmy Microsoft np. z pakietem Office. Można zainstalować go ręcznie pobierając odpowiednią wersję ze strony firmy Microsoft. System Windows XP posiada wersję 2.7 komponentu i instalacja MDAC jest zbędna do uruchomienia systemu FuzzyQ na tej platformie.

Moduł „kreatora zapytań” został napisany w technologii Flash, a więc do poprawnego działania aplikacji niezbędny jest Macromedia FlashPlayer w wersji 7.0. Jest on darmowy i można go ściągnąć z witryny WWW firmy Macromedia.

Aby wykorzystać pełne możliwości systemu FuzzyQ, należy posiadać dostęp do bazy danych z zaimplementowanymi funkcjami i operatorami rozmytymi. Aby połączenie z bazą danych aplikacji FuzzyQ było możliwe, w systemie musi być zainstalowany sterownik ODBC dla bazy danych. W przypadku, gdy baza danych istnieje na odległym serwerze, wymagany jest również dostęp do sieci, czyli po prostu fizyczne połączenie z bazą danych.

Do zainstalowania programu wymagane jest około 3MB przestrzeni dyskowej na sam program FuzzyQ oraz 37-40MB (w zależności od wersji językowej) na zainstalowanie .NET framework 1.1. Zalecane wymagania sprzętowe to komputer klasy Pentium ~800Mhz i 128MB RAM.

9.2 System FuzzyQ – VS .NET 2003

Cały system, z wyjątkiem „kreatora zapytań”, zrealizowany został w graficznym środowisku Visual Studio 2003, jako projekt „Windows Application”. W trakcie tworzenia systemu pojawiła się wersja Beta nowego, lepszego środowiska Visual Studio 2005. Ponieważ jednak nie jest to jeszcze wersja stabilna autor kontynuował pisanie w poprzedniej wersji środowiska.

9.2.1 Przestrzenie nazw .NET

Biblioteki klas dostępne w .NET Framework stanowią podstawę tworzenia kontrolek, aplikacji i komponentów .NET. Za pomocą tych bibliotek można uzyskać dostęp do wielu funkcji i zasobów systemowych. Są one pogrupowane w hierarchiczną strukturę przestrzeni nazw. Biblioteka nazw składa się z klas, interfejsów i stałych. Dwie klasy o identycznej nazwie mogą istnieć równolegle pod warunkiem, że są przypisane do różnych przestrzeni nazw. Jest to możliwe, ponieważ przestrzenie nazw zawężają zakresy obowiązywania typów. Aby odwoływać się do interesujących elementów czy funkcji, należy poruszać się po przestrzeniach nazw za pomocą znaku kropki. Bazowe przestrzenie nazw to System oraz Microsoft. Aby w programie skrócić odwołania do obiektów często występujących, możemy na samym początku programu wskazać kompilatorowi, które z przestrzeni nazw chcemy uznawać za domyślne. Wskazanie to odbywa się za pomocą słowa kluczowego Using^[9].

W systemie FuzzyQ zostało wykorzystane kilka funkcji i typów zawartych w przestrzeni nazw. Ich nazwy oraz krótki opis znajdują się w tabeli Tab.2.

Przestrzeń Nazw	Opis
<code>using System;</code>	Przestrzeń nazw System zawiera podstawowe klasy bazowe, które udostępniają procedury, funkcje, typy, zdarzenia, interfejsy oraz atrybuty służące m.in. do: nadzorowania kodu zarządzanego i niezarządzanego, czyszczenia pamięci, obliczeń matematycznych i wielu, wielu innych.
<code>using System.Collections;</code>	Przestrzeń ta zawiera interfejsy i klasy pojemnikowe: listy, kolejki, tablice bitów, tablice rozproszone (hashtable), słowniki, wektory, itp.
<code>using System.ComponentModel;</code>	Przestrzeń ta zawiera klasy i interfejsy pomocne w tworzeniu komponentów i kontrolek.
<code>using System.Windows.Forms;</code>	Klasy i metody umożliwiające tworzenie interfejsów graficznych.

<code>using System.Data;</code>	Przestrzeń ta zawiera klasy składające się na architekturę dostępu do danych ADO.NET.
<code>using System.Data.Odbc;</code>	Przestrzeń ta zawiera klasy odpowiedzialne za komunikację ze źródłem danych ODBC.
<code>using System.IO;</code>	Przestrzeń ta udostępnia nam typy i funkcje umożliwiające synchroniczny i asynchroniczny odczyt i zapis strumieni danych i plików.
<code>using System.Drawing;</code>	Przestrzeń ta pozwala na dostęp do funkcji graficznych biblioteki systemowej GDI+. Biblioteka ta udostępnia zaawansowane mechanizmy tworzenia grafiki.

Tab. 2 Tabela podstawowych przestrzeni nazw użytych w projekcie FuzzyQ

9.2.2 Klasy przestrzeni nazw „myconnect”

Przestrzeń nazw „myconnect” obejmuje klasy będące zestawem narzędzi służącym do połączenia z bazą danych, komunikacji z bazą danych czy interpretacją komunikatów przychodzących z kreatora zapytań, który jest napisany w języku Flash. Można wyróżnić następujące klasy:

9.2.2.1 Klasa „connect”

```
public class connect : System.Windows.Forms.Form
{
    [...]
}
```

Jest to klasa, która tworzy wygląd formatki „Connect”. Klasa ta odpowiada za nawiązanie połączenia z bazą danych za pomocą interfejsu ODBC. Odpowiednie parametry połączenia przekazuje do niej użytkownik za pomocą formatki „Connect”. Dostęp do uzyskanego połączenia odbywa się za pomocą metody „Connection()”.

9.2.2.2 Klasa „InteractiveMovie”

```
public class InteractiveMovie : System.Windows.Forms.Form
{
    [...]
}
```

Jest to klasa, której zadaniem jest załadowanie „kreatora zapytań”, czyli interaktywnego filmu napisanego w języku Flash. Klasa odpowiada również za komunikację z filmem oraz za interpretację komunikatów i danych zwróconych przez „kreatora zapytań”. Ważniejsze metody tej klasy to m.in.:

- InteractiveMovie (string sciezka) - czyli konstruktor tej klasy, odpowiada on za załadowanie filmu w języku flash z domyślnego katalogu,
- axShockwaveFlash1_FSCommand (object sender, AxShockwaveFlashObjects._IShockwaveFlashEvents_FSCommandEvent e) – metoda ta odpowiada za nasłuchiwanie i odbiór wiadomości przekazywanych z filmu w języku flash do aplikacji głównej. Wiadomości te są zapisane w treści zdarzeń (events) tworzonych z poziomu filmu. Funkcja ta po otrzymaniu wiadomości, poddaje ją analizie,
- Analizuj() – funkcja służąca do analizy wiadomości otrzymanej z „kreatora zapytań”. Funkcja ta odpowiedzialna jest za rozpoznanie czy zapytanie utworzone w kreatorze jest zapytaniem rozmytym złożonym czy pojedynczym. Jeżeli jest to zapytanie złożone odbiera kolejne wiadomości i składa je w jedno zapytanie złożone. Wynik analizy zapisany jest w zmiennej zapytanie_koncowe,
- Wynik() – metoda umożliwiająca dostęp obiektom z zewnątrz do pola zapytanie_koncowe.

9.2.2.3 Klasa „query”

```
public class query : System.Windows.Forms.Form
{
    [...]
}
```

Jest to klasa, której konstruktor tworzy wygląd głównego okna. Jest ona odpowiedzialna za przesyłanie zapytań i prezentację wyników zwróconych przez bazę danych. Zawiera ona m.in. metody takie jak:

- `Polacz()` – odpowiada za ustanowienie połączenia z bazą danych. W metodzie tej tworzony jest obiekt klasy „connect”, który zwraca za pomocą metody `Connection` nawiązane połączenie jako obiekt typu `ODBCConnection`. Jest ono wykorzystywane w innych metodach jako parametr niezbędny do komunikacji z bazą danych,
- `Rozlacz()` – zamyka przekazane wcześniej połączenie,
- `Wykonaj_zapytanie()` – funkcja tworzy obiekt klasy `ODBCDataAdapter` i z pomocą metod, które ta klasa udostępnia przekazuje zapytanie do bazy danych oraz wypełnia obiekt klasy `DataSet` zwróconym wynikiem. Parametrem wymaganym przez `ODBCDataAdapter` jest obiekt klasy `ODBCConnection` skojarzony z bazą danych,
- `Loadmovie()` - funkcja ta tworzy obiekt klasy `InteractiveMovie` przekazując mu sterowanie. W momencie zakończenia działania poprzez zamknięcie okna obiektu `InteractiveMovie` lub zakończeniu tworzenia szkieletu zapytania okno jest zamykane, a zapytanie przekazane do okna głównego aplikacji FuzzyQ.

9.2.3 Pliki wej/wyj

System FuzzyQ po wybraniu opcji połączenia z bazą danych pobiera dane z pliku „polacz.ini”, o ile taki plik istnieje. Przykładowa struktura pliku „polacz.ini” wygląda następująco:

PostgreSQL#83.22.90.123#5432#postgres#test

Słowa kluczowe są oddzielone znakiem „#”. Aplikacja szuka tego pliku w katalogu, w którym jest zainstalowana. Plik konfiguracyjny można utworzyć ręcznie, pamiętając o odpowiedniej kolejności podawania parametrów:

nazwa_źródła_danychODBC#IP#port#nazwa_użytkownika#nazwa_bazy_danych

Aplikacja może zapisać w pliku testowym historie komunikatów, a także treść zapytania.

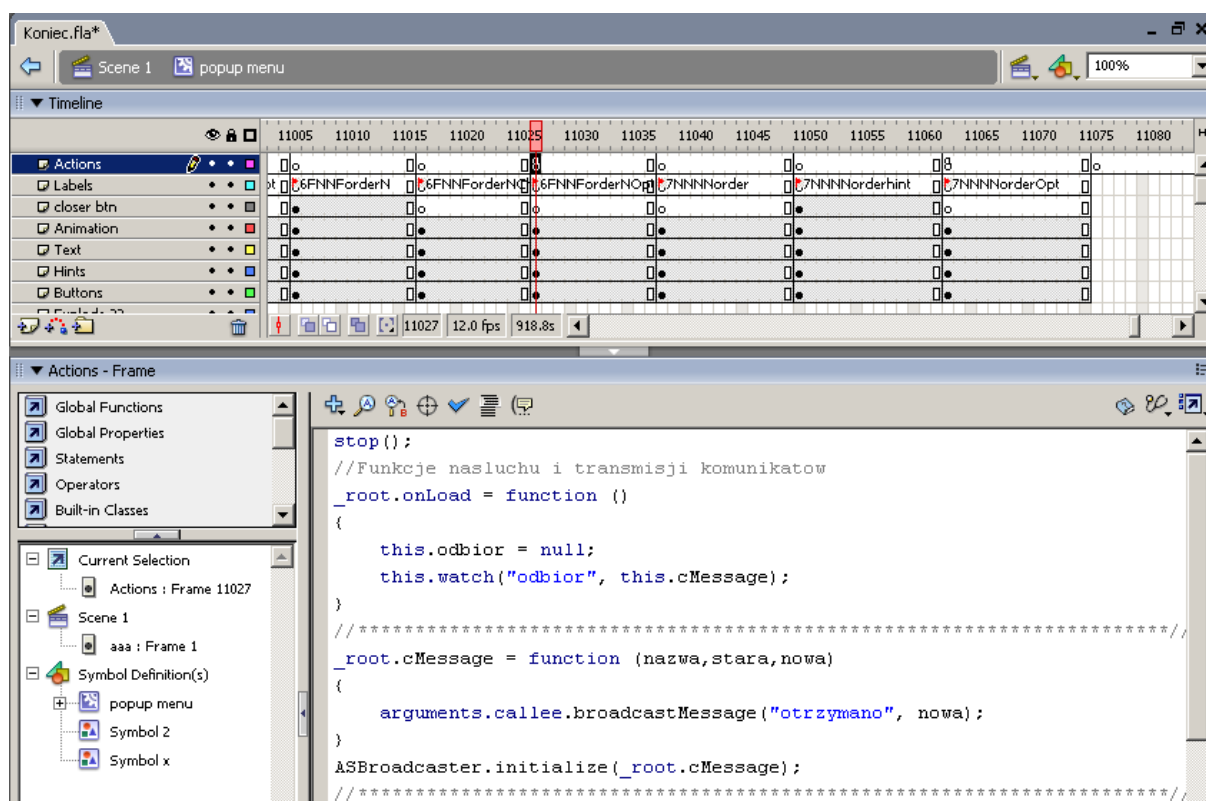
Istnieje możliwość zachowania danych wyjściowych wyświetlonych po uruchomieniu zapytania w pliku *.xml. Dane te prezentują się w następujący sposób (rys. 33):

```
<Wynik_x0020_Zapytania_x0020_>
+<Table></Table>
-<Table>
    <imie>Grzegorz          </imie>
    <nazwisko>Walendziak    </nazwisko>
    <id>4 </id>
    <data_ur>1996-03-22T00:00:00.0000000+01:00 </data_ur>
</Table>
+<Table></Table>
</Wynik_x0020_Zapytania_x0020_>
```

Rys. 33 Przykładowa struktura danych zapisanych w formacie xml

9.3 System FuzzyQ – Flash MX 2004

W środowisku Macromedia Flash MX 2004 powstał tzw „kreator zapytań”, czyli film utworzony w technologii flash, mający na celu budowę szkieletu zapytania rozmytego. Film interaktywny jest wynikiem kompilacji projektu zbudowanego za pomocą Studia MX 2004. Środowisko to składa się z trzech głównych paneli: Timeline, Actions oraz Preferences. Końcowy projekt poddany ostatecznej kompilacji zawierał ponad 10 tys klatek zwykłych oraz około 700 klatek kluczowych w sekcji Timeline. Kluczowe klatki uruchamiają w momencie wywołania przypisane im skrypty. Skrypty te powstały w języku Action Script , który w swojej nowej wersji 2.0 pozwala pisać programy obiektowo-zorientowane. Na rysunku (rys. 34) przedstawiono przykładowy skrypt tworzony dla jednej z klatek kluczowych:



Rys. 34 Przykładowy skrypt w języku Action Script 2.0 w środowisku Macromedia Flash MX 2004

Nie bez przyczyny firma Macromedia inwestuje w rozwój tego języka skryptowego, ponieważ poszerza on w znacznym stopniu możliwości całego środowiska. Ważniejsze części kodu skryptów napisanych w celu komunikacji z oknem głównym aplikacji, zostaną omówione w następnym podrozdziale.

9.4 Analiza ciekawszych zagadnień implementacyjnych

9.4.1 Komunikacja C# i Flash

Ze wszystkich zagadnień implementacyjnych tego projektu na szczególne wyróżnienie zasługuje komunikacja między modulem napisanym w języku Flash oraz modulem aplikacji napisanym w języku C#. Poszukując w sieci Internet materiałów dotyczących tego problemu, autor poznał wielu ludzi z całego świata, którzy w różnym celu poszukiwali materiałów, wskazówek czy gotowych rozwiązań dotyczących tego zagadnienia. Ponieważ pytań było mnóstwo, a odpowiedzi niewiele autor wnioskuje, iż temat jest nietypowy i wart zwrócenia uwagi.

Aby wstawić element napisany w języku Flash do aplikacji C# i zapewnić komunikację między nimi należy:

1. W projekcie w Visual Studio dodać do projektu obiekt COM o nazwie Shockwave Flash Object. Jeżeli nie występuje on na liście dostępnych obiektów COM należy upewnić się czy poprawnie zainstalowano w systemie kontrolkę ActiveX Macromedia Flash Player^[11].
2. W projekcie w Visual Studio, film w języku Flash, załadować do powyższej kontrolki ActiveX za pomocą polecenia^[10]:

```
axShockwave.LoadMovie ( klatka_startowa , sciezka do pliku *.swf )
```

3. Aby wysłać coś z aplikacji C# do filmu w języku Flash, należy skorzystać z funkcji:

```
axShockwave.SetVariable( nazwa_zmiennej , wartość )
```

4. W projekcie Flash należy umieścić kod odpowiedzialny za odbieranie wiadomości z aplikacji C#:

```
Funkcje nasluchu i transmisji komunikatow
_root.onLoad = function ()
{
    this.odbior = null;
    this.watch("odbior", this.cMessage);
}
_root.cMessage = function (nazwa,stara,nowa)
{
    arguments.callee.broadcastMessage("otrzymano", nowa);
}
ASBroadcaster.initialize(_root.cMessage);
```

Odbieranie wiadomości od Visual C#:

```
obiekt = new Object();
obiekt.otrzymano = function (tresc)
{
    //test
    _root.flashoutput_txt.text += tresc + "\n";
}
_root.cMessage.addListener(obiekt);
```

5. Aby przesłać informację z filmu w języku Flash do aplikacji w C# należy skorzystać z poniższej funkcji:

```
_level0.select.buttonSend_mc2.onRelease = function ()
{
    fscommand("flashMessage", "SELECT kol1,kol2 FROM
tab1,tab2");
    gotoAndStop("Scene 1", 1);
}
```

6. Aby w aplikacji C# odbierać komunikaty wysłane przez film w języku Flash potrzebny nam będzie poniższy fragment kodu^[10]:

```
this.axShockwaveFlash1.FSCommand +=
new AxShockwaveFlashObjects._IShockwaveFlashEvents
_FSCCommandEventHandler (axShockwaveFlash1_FSCCommand);
```

oraz ciało funkcji:

```
private void axShockwaveFlash1_FSCCommand(object sender, AxShockwave-
FlashObjects._IShockwaveFlashEvents_FSCCommandEvent e)
{
    int licznik=0;
    if (e.command == "flashMessage") //tylko gdy dostaniemy
    //odpowiednia wiadomość
    {
        if(licznik==0)
        {
            message=e.args.ToString();
            Analizuj(message);
            wynik();
        }
    }
}
```

9.4.2 Przetwarzanie komunikatów Flash w aplikacji napisanej w języku C#

Kreator zapytań pozwala zarówno na budowę szkieletu zapytania prostego jak i złożonego. Każde ukończone zapytanie jest wysyłane do aplikacji napisanej w C# i tam zostaje poddane przetworzeniu, ponieważ otrzymany szkielet może być samodzielnym zapytaniem pojedynczym lub zapytaniem głównym zapytania złożonego, albo podzapytaniem zapytania złożonego. Aby rozpoznać, z którym przypadkiem mamy do czynienia w momencie, gdy użytkownik zdecyduje o utworzeniu podzapytania, „kreator zapytań” wyśle dotychczas utworzoną część z oznaczeniem „#” na końcu zapytania, np.:

```
SELECT kol1,kol2 FROM tab1,tab2 WHERE kol3=#
```

Następnie przeprowadzi budowę zapytania do końca i prześle całkowity, skończony szkielet główny do aplikacji. Aplikacja przetworzy oba ciągi i powstanie jeden, który w miejscu wystąpienia podzapytania będzie miał znaczek „#”, np.:

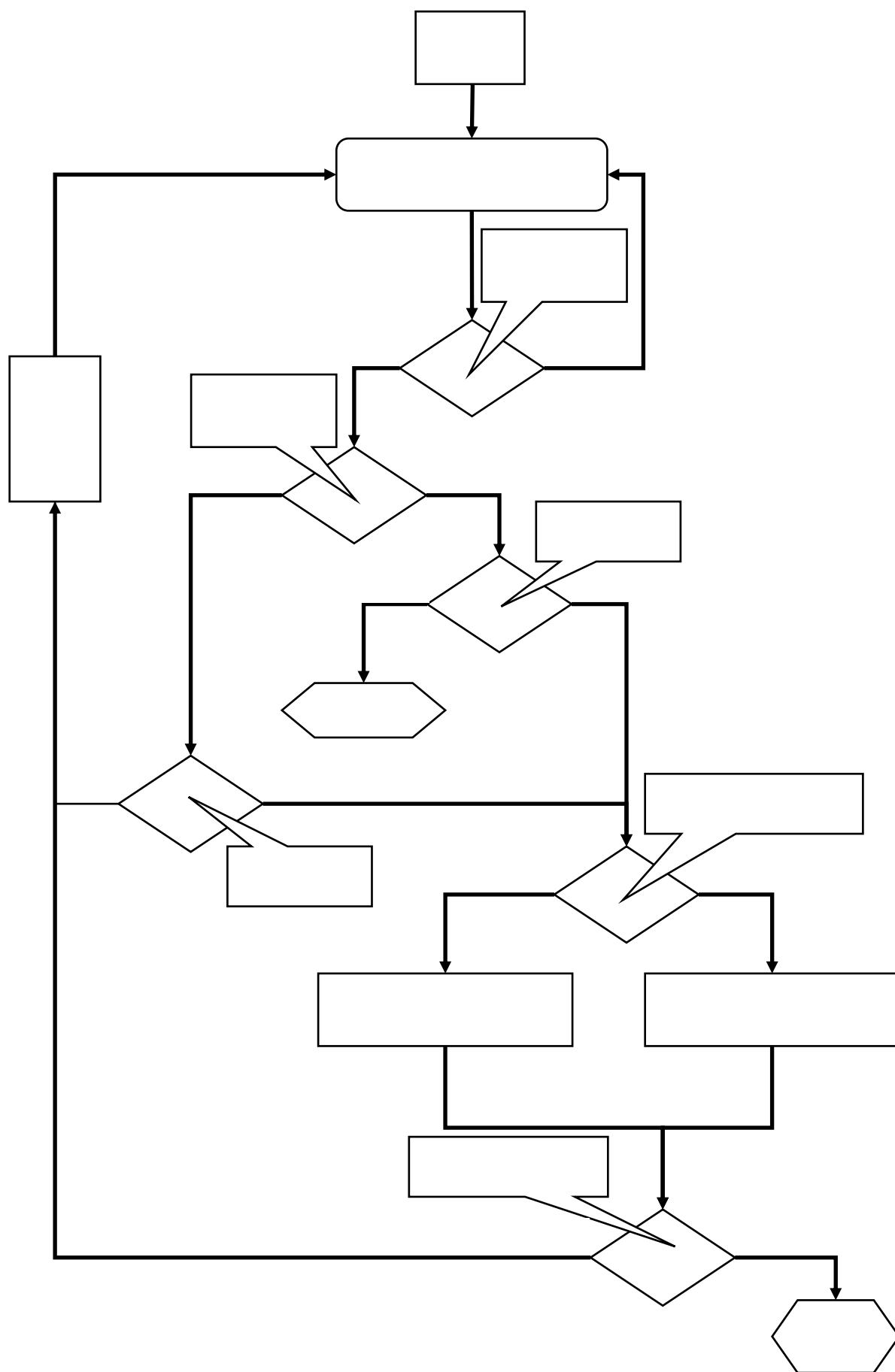
```
SELECT kol1,kol2 FROM tab1,tab2 WHERE kol3=# ORDER BY kol1;
```

Jednocześnie ustawiona zostanie flaga informująca program, że kolejne przysyłane zapytania są podzapytaniem i należy je podstawiać kolejno w miejsce znaczników „#”, np.:

```
SELECT kol1,kol2 FROM tab1,tab2 WHERE kol3=
  (SELECT kol1~=okolo123,kol2 FROM tab1,tab2 ORDER BY kol1~=okolo123)
ORDER BY kol1;
```

Gdy w poskładanym zapytaniu nie występuje znaczek „#”, zwalniana jest flaga zapytania złożonego i gotowe zapytanie wysyłane jest do okna głównego aplikacji.

Algorytm przetwarzający komunikaty z kreatora zapytań zawarty jest w funkcji: Analizuj(string zmienna). Ogólny schemat blokowy algorytmu przedstawiono na diagramie (rys. 35).



Rys. 35 Schemat blokowy algorytmu zaszytego w funkcji analizuj

W kodzie algorytmu możemy wyróżnić 4 główne części, które wynikają z obsługi czterech różnych sytuacji nadejścia wiadomości:

- a) Przychodzi komunikat nie zawierający znacznika „#”, a bufor wiadomości jest pusty,
- b) Przychodzi komunikat nie zawierający znacznika „#”, a bufor wiadomości jest pełny,
- c) Przychodzi komunikat zawierający znacznik „#”, a bufor wiadomości jest pusty,
- d) Przychodzi komunikat zawierający znacznik „#”, a bufor wiadomości jest pełny,

Nadchodzący komunikat zostaje zakwalifikowany do jednej z powyższych możliwości i w zależności od jego rodzaju obsługa jest inna w każdej z opcji.

Pierwszy przypadek „a)” – jeżeli przychodzi komunikat bez znacznika # to znaczy że jest to zapytanie proste, a nie złożone. Jednocześnie spełniony jest warunek, że bufor zapytania końcowego jest pusty, a więc komunikat ten jest jednocześnie końcowym zapytaniem zwracany przez kreatora. Na powyższym diagramie jest to przypadek kolejnych wyborów: TAK, NIE, TAK

Drugi przypadek „b)” – przychodzący komunikat jest bez znacznika #, a więc jest to zapytanie proste, ale tym razem bufor zawiera już pierwszą część zapytania. Jeżeli komunikat zostaje tu zakwalifikowany mamy pewność, że zwrócone zapytanie będzie zapytaniem złożonym. Do bufora wpisywane jest zapytanie posiadające co najmniej jeden znacznik #. Następnym elementem w tej części będzie określenie, do którego znacznika zostanie przypisane podzapytanie, które przyszło w aktualnym komunikacie. Rozpatrzmy skrajną możliwość zawartości bufora. Jeżeli komunikat przychodzący był prostym zapytaniem postaci np. *”SELECT kol1,kol2 FROM tab1,tab2”*, a bufor prezentował się następująco: *„SELECT kol1,(kol2~=okolo 20) AS st_przynależności FROM tab1,tab2 WHERE kol3=(SELECT kol1,kol2 FROM tab1,tab2 WHERE kol3=(¹) ORDER BY kol1) HAVING kol3=(²) ORDER BY kol2~=okolo20”* to algorytm wstawi go w miejsce znacznika „#” o numerze 2, ponieważ w pierwszej kolejności uzupełnianie są znaczniki poziomu najwyższego. Znacznik # o numerze 1 jest znacznikiem, który jest w podzapytaniu zapytania głównego, a więc w zapytaniu o jeden poziom niższym. Po wstawieniu całość: *„SELECT kol1,(kol2~=okolo 20) AS*

st przynależności FROM tab1,tab2 WHERE kol3=(SELECT kol1,kol2 FROM tab1,tab2 WHERE kol3=(#¹:) ORDER BY kol1) HAVING kol3=(SELECT kol1,kol2 FROM tab1,tab2) ORDER BY kol2~=okolo20 ” jest zapisywana do bufora. Ostatnim krokiem jest sprawdzenie czy zapytanie w buforze posiada jeszcze miejsca do wstawienia podzapytań, a więc znaczniki #. Jeżeli nie to zapytanie jest gotowe, jeżeli tak to czekamy na kolejny komunikat. Na powyższym diagramie jest to przypadek kolejnych wyborów: TAK, NIE, NIE, TAK, TAK.

Trzeci przypadek „c)”- przychodzący komunikat zawiera znacznik # ale bufor jest pusty, a więc jest to główny poziom zapytania złożonego. Komunikat jest przepisywany do bufora i algorytm oczekuje kolejnych, które będą podzapytaniami zapytania głównego. Na powyższym diagramie jest to przypadek kolejnych wyborów: TAK, TAK, TAK.

Ostatni przypadek „d)”- przychodzący komunikat zawiera znacznik # i bufor zawiera już treść wcześniejszych komunikatów, a więc jest to podzapytanie zapytania znajdującego się już w buforze. Ponieważ zawiera ono znacznik # czyli rozszerza ono nasz szkielet zapytania o kolejny niższy poziom. Wstawianie komunikatu w zapytanie główne z bufora odbywa się na analogicznych zasadach jak opisane w punkcie b). Po wstawieniu algorytm zapisuje całość do bufora i czeka na kolejny komunikat. Na powyższym diagramie jest to przypadek kolejnych wyborów: TAK, TAK, NIE TAK lub NIE, TAK.

9.5 Wybrane problemy realizacji systemu

System FuzzyQ można podzielić na dwa moduły: część napisaną w języku C# oraz część napisaną w języku Flash. Podczas projektowania systemu, w momencie, gdy zapadła decyzja o wyborze dwóch środowisk programistycznych, wielką niewiadomą było, w jaki sposób połączyć oba moduły tak, aby komunikowały się ze sobą i współpracowały w ramach jednej aplikacji. Autorzy oprogramowania Visual Studio .Net 2003, czyli firma Microsoft, posiadająca bogate wsparcie dla developerów tworzących oprogramowanie w tym środowisku, nie poruszają tematu współpracy ze środowiskiem firmy Macromedia. Jest to raczej zdumiewające, ponieważ środowiska te nie są konkurencyjne wobec siebie, a oprogramowanie tworzone z ich pomocą dotyczy zupełnie różnych dziedzin informatyki. Nie znalazłszy pomocy wśród materiałów udostępnianych przez Microsoft, autor przeniósł sferę poszukiwań na obszar

dokumentacji i wsparcia dla developerów, firmy Macromedia. Wywołanie filmu Flash z poziomu aplikacji w C# nie stanowiło problemu, natomiast komunikacja między tymi modułami, wzajemne przekazywanie komunikatów nie było rzeczą oczywistą i było sporym utrudnieniem w realizacji projektu.

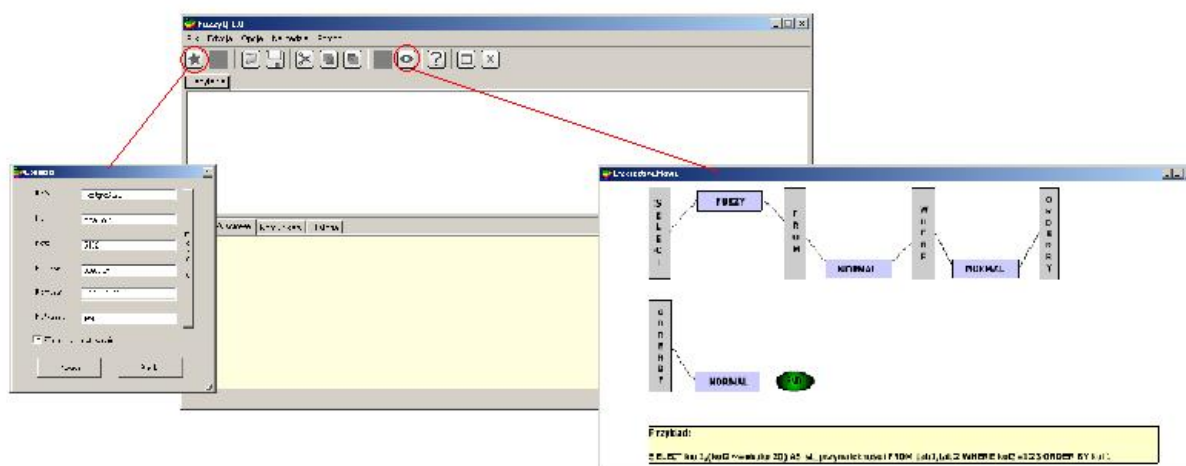
10. Interfejs systemu FuzzyQ

System FuzzyQ jest narzędziem do analizowania i konstruowania zapytań FuzzySQL^[2]. Instalacja systemu zostanie omówiona w załączniku nr 1 tej pracy, a deinstalacja w załączniku nr 2.

W skład systemu wchodzi trzy obiekty:

- Okno główne aplikacji FuzzyQ,
- Okno parametrów połączenia,
- Kreator zapytań.

Po uruchomieniu systemu na pierwszym planie pojawi się okno główne programu. Korzystając z paska narzędzi lub menu okna, możemy uzyskać dostęp do pozostałych obiektów. Wszystkie trzy okna wraz z zaznaczeniem ikon, które je uruchamiają, przedstawia rysunek (rys. 36).

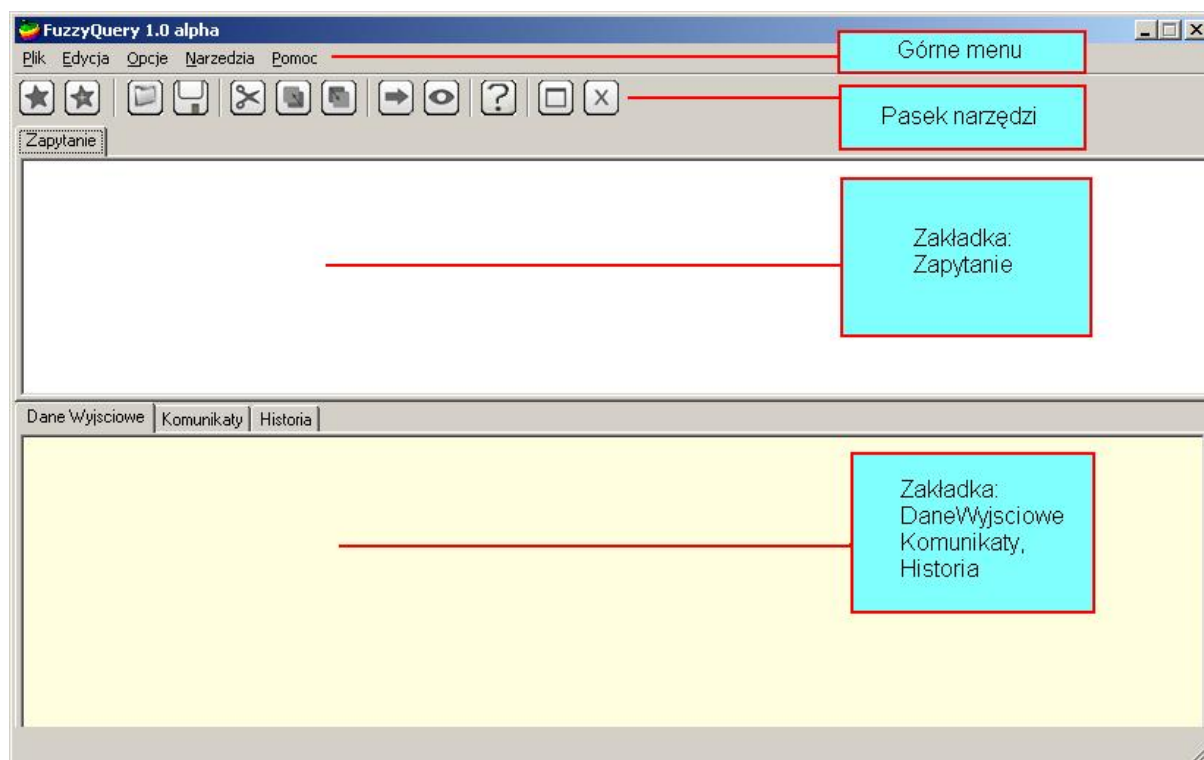


Rys. 36 Widok wszystkich formatek z zaznaczeniem miejsc wywołania z okna głównego

Zamknięcie lub zakończenie pracy w oknach kreatora zapytań oraz parametrów połączenia powoduje powrót do okna głównego aplikacji. Każde z okien zostanie omówione w oddzielnym podrozdziale.

10.1 Okno główne systemu

Okno główne systemu przedstawia się następująco:



Rys. 37 Okno główne systemu FuzzyQ

Można w nim wyróżnić kilka ważnych części, które zostały przedstawione i opisane na rysunku (rys. 37). Są to:

10.1.1 Górne menu - zawiera pozycje takie jak :

- **Plik - Połącz/Rozłącz** odnosi się do połączenia z BD. Naciśnięcie przycisku „Połącz” spowoduje otwarcie okna „parametry połączenia”. Przy założeniu, że poprawnie podano wszystkie parametry, potwierdzenie przyciskiem „OK” spowoduje nawiązanie połączenia z bazą danych. Od tego momentu aż do zamknięcia aplikacji lub do momentu wciśnięcia przycisku „Rozłącz” , utrzymywane jest połączenie z bazą danych.

Nowy odnosi się do zakładki „Zapytanie”, wybranie tej pozycji z menu spowoduje wyczyszczenie pola tekstowego w zakładce „Zapytanie”.

Zapisz/Zapisz Jako otwierają okna dialogowe zapisu oraz zachowują zawartość pola tekstowego zakładki „Zapytanie” do pliku o nazwie podanej przez użytkownika. W przypadku kolejnego użycia funkcji „Zapisz” nazwa pliku pamiętana jest z poprzedniego zapisu. „Zapisz Jako” zawsze otwiera

okno dialogowe zapisu, bez względu na to, czy użytkownik wykonywał już zapis do pliku, czy nie.

Exit - zamyka całą aplikację

- **Edycja** - Cut/Copy/Paste/Select All odnosi się do tekstu w polu tekstowym zakładki „Zapytanie”.

Wytnij wycina zaznaczony fragment tekstu do bufora.

Kopiuuj kopiuje zaznaczony fragment tekstu do bufora.

Wklej wkleja zawartość bufora.

Zaznacz wszystko zaznacza cały tekst w polu.

- **Opcje** - „Zapisuj dane wyjściowe do pliku xml”- jeśli ta opcja jest zaznaczona, opcja MENU->Plik->Zapisz/Zapisz jako zostaje poszerzona o dodatkowe funkcje: zapisuje treść zakładki „Zapytanie” oraz wynik zapytania do pliku o tej samej nazwie, ale innym rozszerzeniu (*.xml).
- **Czcionka** - odnosi się do zakładki „Zapytanie” i powoduje włączenie okna z właściwościami czcionek (kolor, wielkość, kształt).
- **Narzędzia - Zapytanie** uruchamia proces przesyłu treści zakładki „Zapytanie” do bazy danych i prezentuje wynik zapytania w zakładce „Dane wyjściowe”. **FuzzyKreator** powoduje uruchomienie okna obiektu wspomagającego budowanie zapytań.
- **Pomoc - Index** powoduje wyświetlenie pliku pomocy.

W programie dostępne są skróty klawiszowe do niektórych opcji menu. Przedstawione zostały one w tabeli (tab.3).

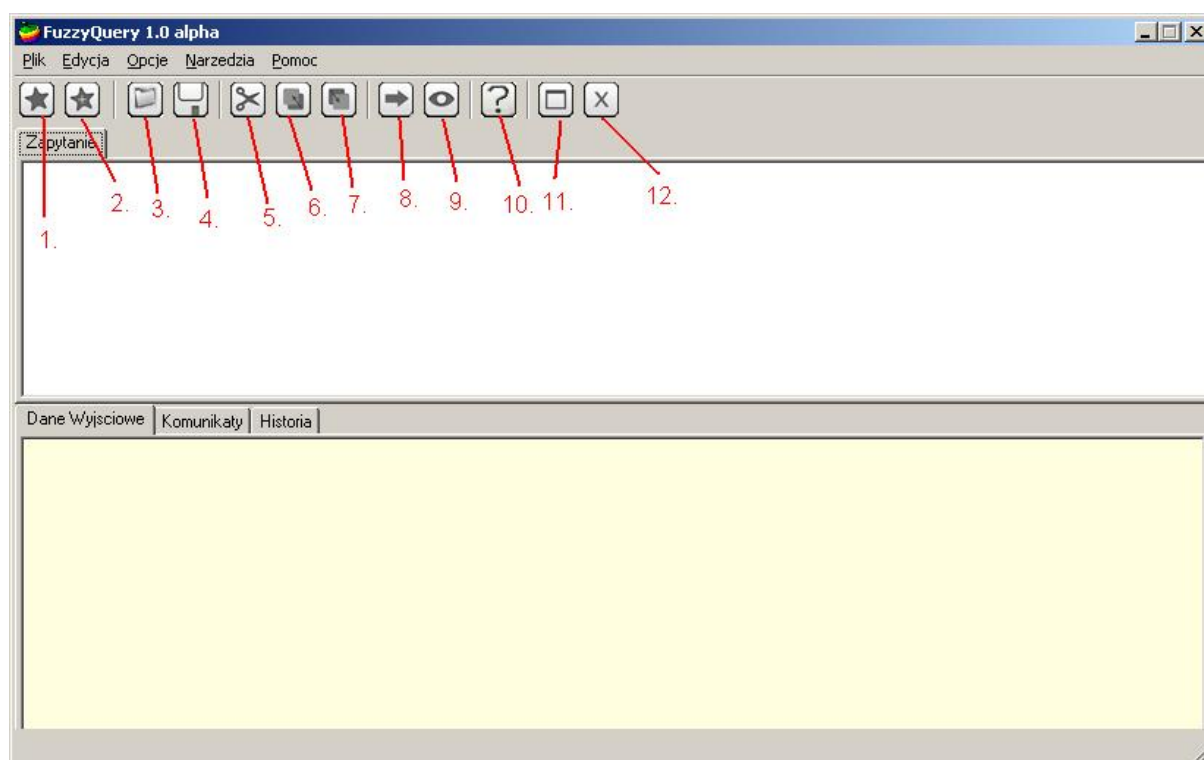
<u>Klawisz</u>	<u>Funkcja</u>
F1	Index
F2	Zapisz
F5	Zapytanie
F6	FuzzyKreator
Ctrl+Q	Nowy
Ctrl+X	Wytnij
Ctrl+C	Kopiuuj

Ctrl+V	Wklej
Ctrl+A	Zaznacz wszystko





Tab. 3 Tabela skrótów klawiszowych systemu FuzzyQ









10.1.2 Pasek narzędzi

Na pasku narzędzi znajduje się dwanaście ikon (rys. 38).

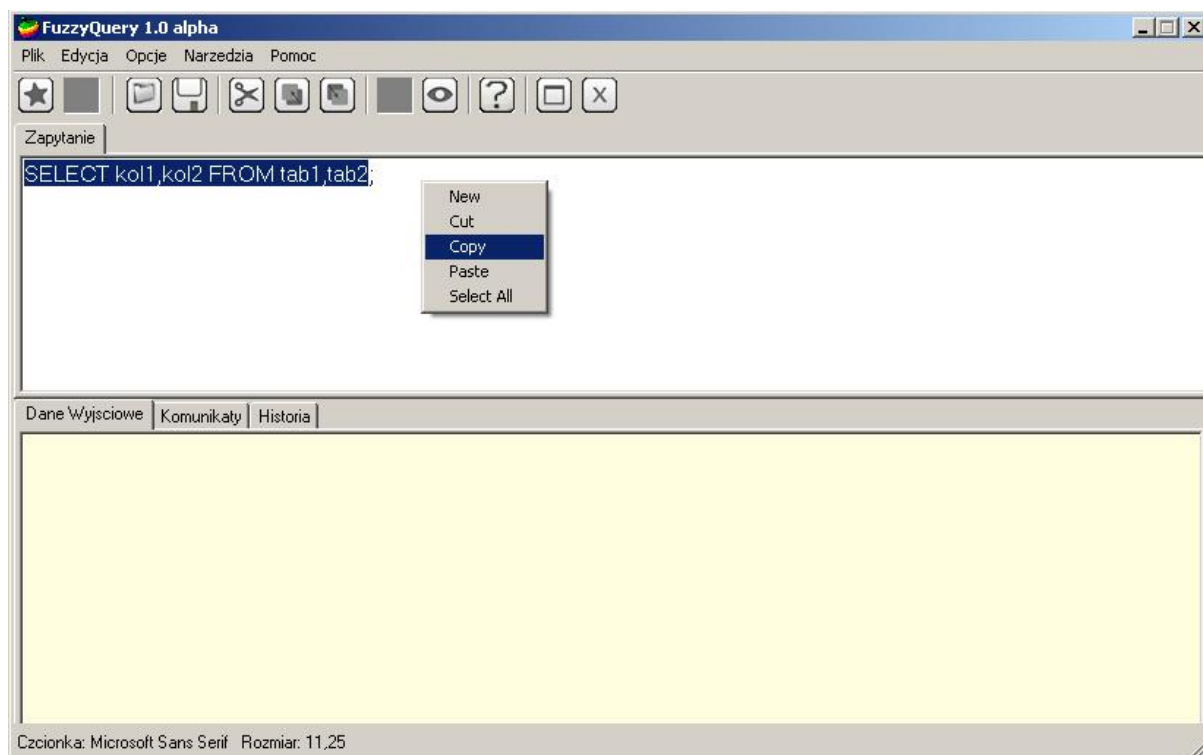


Rys. 38 Pasek narzędzi okna głównego aplikacji FuzzyQ

1.  - Połącz – ustanawia połączenie z BD, uruchamia zakładkę „parametry połączenia”.
2.  - Rozłącz – zrywa połączenie z BD.
3.  - Otwórz – otwiera okno dialogowe open, użytkownik wybiera plik testowy(*.txt), z którego załadowane zostanie zapytanie.
4.  - Zapisz – otwiera okno dialogowe save, użytkownik podaje nazwę pliku, do którego zapisane zostanie zapytanie. Jeżeli zaznaczona jest opcja „menu>Opcje>Zapisz dane wyjściowe do pliku *.xml”, to zapisany zostanie również wynik zapytania do pliku o tej samej nazwie, ale rozszerzeniu *.xml.

5.  - Wytnij – Wycina zaznaczony fragment tekstu z pola tekstowego zakładki „Zapytanie”.
6.  - Kopiuj – Kopiuje do bufora zaznaczony fragment tekstu z pola tekstowego zakładki „Zapytanie”.
7.  - Wklej – Wkleja z bufora zaznaczony fragment tekstu do pola tekstowego zakładki „Zapytanie”.
8.  - Wykonaj Zapytanie – uruchamia procedurę przesłania zapytania do bazy danych i odbiór oraz prezentację wyniku.
9.  - Załaduj kreatora - uruchamia okno „Kreatora Zapytań” .
10.  - Pomoc – uruchamia plik pomocy.
11.  - Zapisz historię – otwiera okno dialogowe, gdzie użytkownik podaje nazwę pliku, do którego zapisana zostanie treść zakładki „Historia”.
12.  - Wyjdź – Kończy działanie i zamyka aplikację.

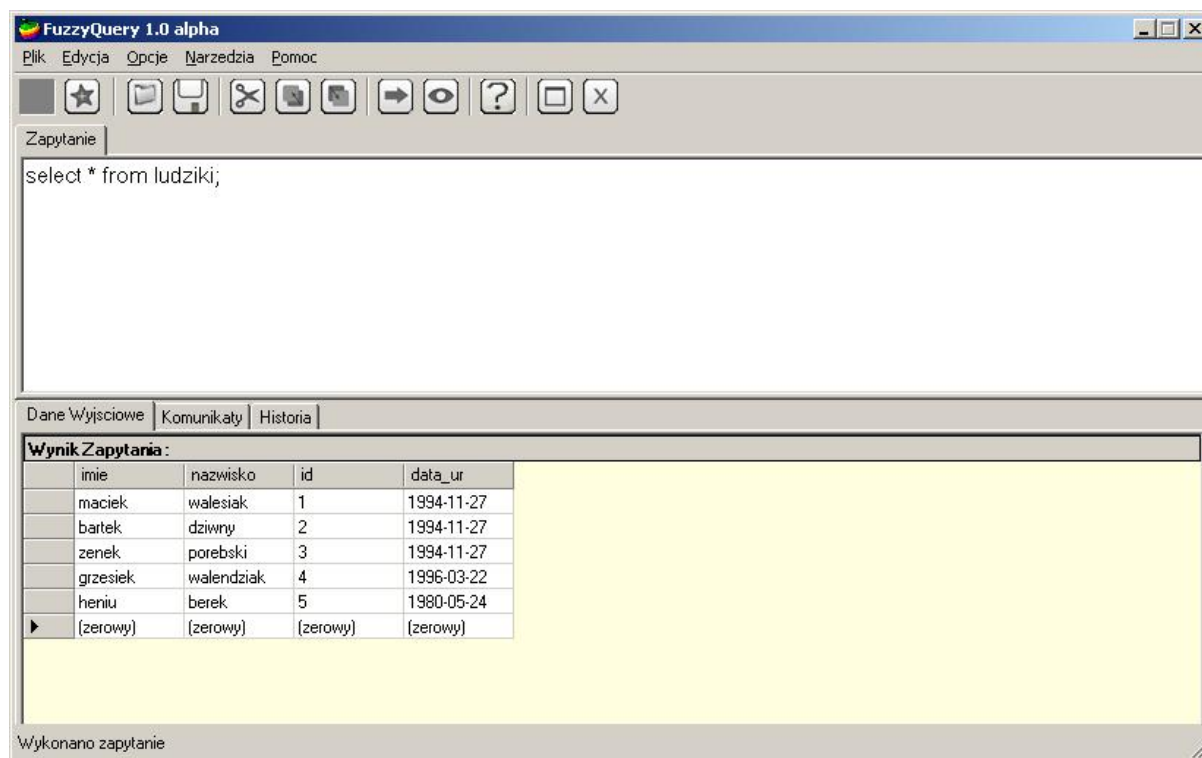
10.1.3 Zakładka „Zapytanie” - Do zakładki tej użytkownik wprowadza zapytanie, które, po naciśnięciu przycisku „Uruchom zapytanie”, zostaje przesłane do bazy danych. Jeżeli jest skonstruowane poprawnie, zostaje zwrócony wynik i wyświetlony w zakładce „Dane wyjściowe”. Rysunek (rys. 39) przedstawia wygląd zakładki wraz z jej menu kontekstowym.



Rys. 39 Zakładka „zapytanie” okna głównego systemu FuzzyQ

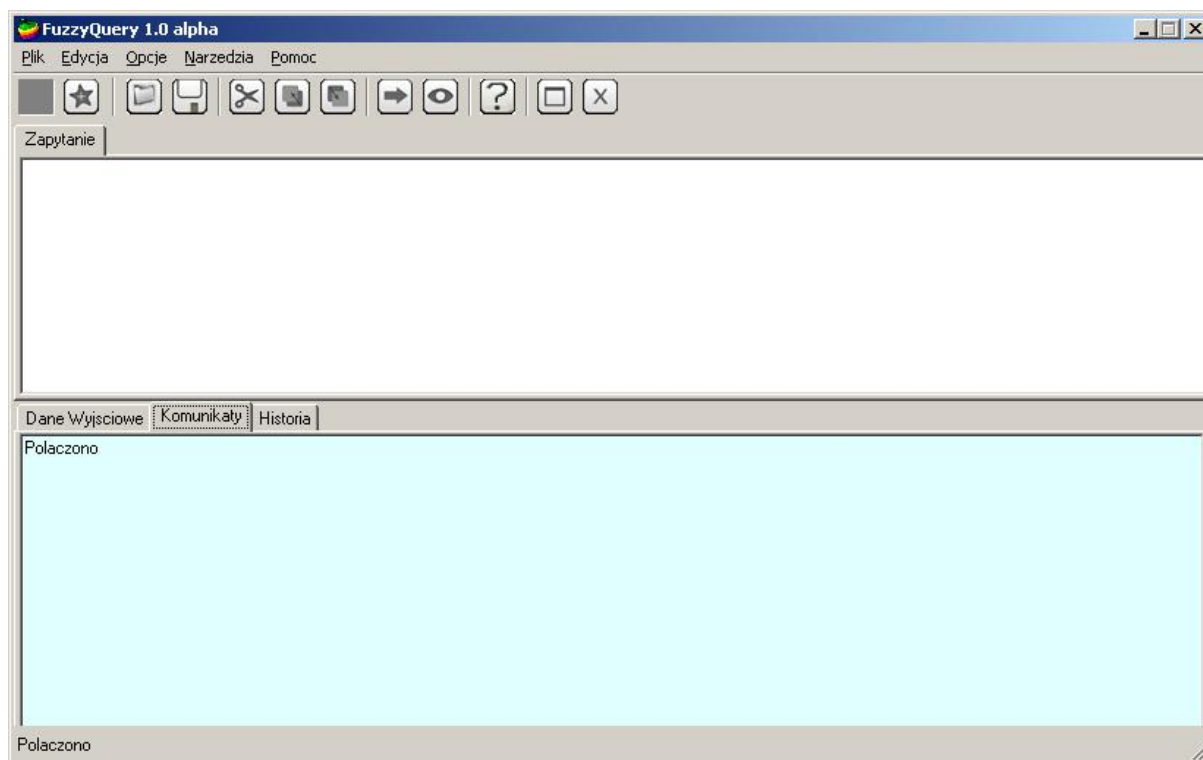
10.1.4 Zakładki: „Dane wyjściowe”, „Komunikaty”, „Historia”

- Dane wyjściowe – jeżeli zapytanie wysłane do bazy danych było poprawne, to baza danych zwróci zestaw danych, które zostaną przekazane do systemu FuzzyQ. Aplikacja natychmiast po otrzymaniu wyniku zaprezentuje go w zakładce „Dane wyjściowe”. Dane zwrócone i wyświetlone w tej zakładce można sortować według wybranej kolumny w kierunku malejącym lub rosnącym naciśkając myszką w nazwę kolumny. Umożliwia to zaimplementowana właściwość kontrolki Data Grid. Przykład wyświetlenia danych w zakładce pokazuje rysunek (rys. 40).



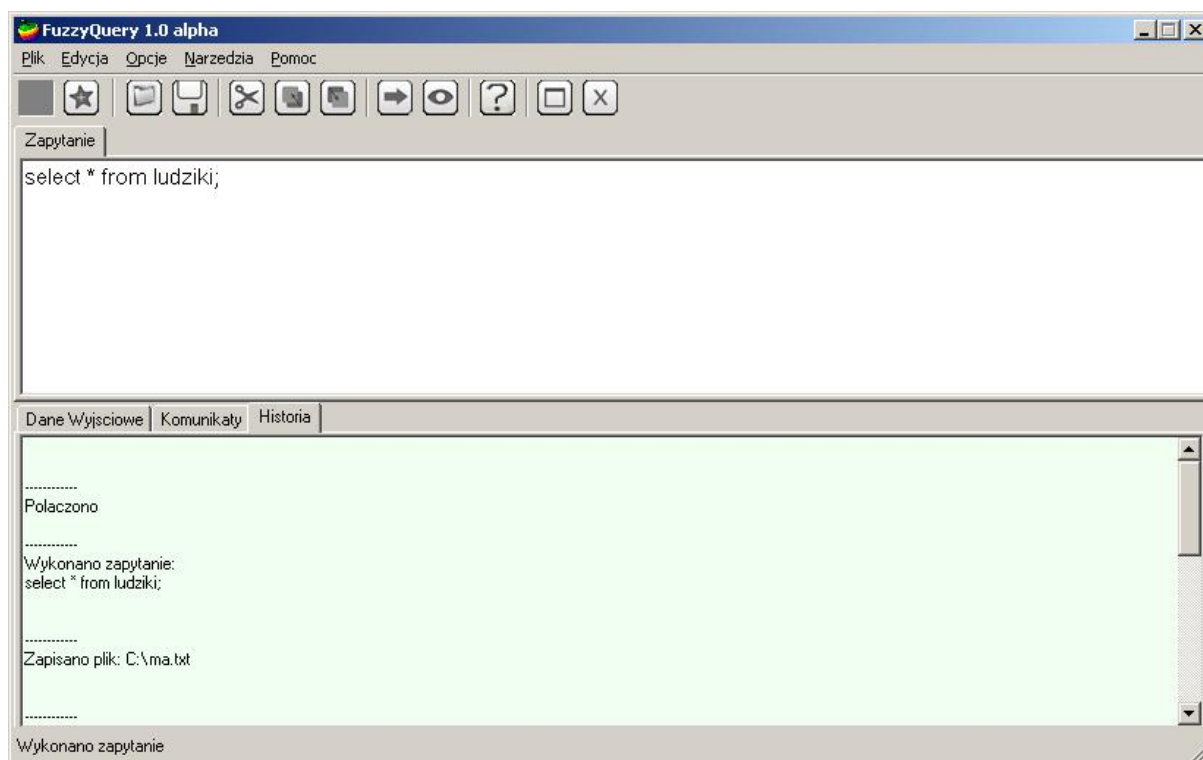
Rys. 40 Zakładka „dane wyjściowe” okna głównego systemu FuzzyQ

- Komunikaty – każdy komunikat związany z wykonaniem lub nie pewnej operacji zostaje odnotowany w zakładce „Komunikaty”. Wszystkie komunikaty z obsługiwanych błędów są wpisywane w pole tekstowe tej zakładki. Każdy nowy komunikat programu nadpisuje stary, który jest automatycznie przenoszony do pola tekstowego zakładki „Historia”. W przypadku gdy np. składnia zapytania wysłanego do bazy danych jest niepoprawna, w zakładce „Komunikaty” pojawi się błąd, zwrócony przez bazę danych, określający miejsce wystąpienia błędu, co umożliwia jego sprawne wykrycie i eliminację. Rysunek (rys.41) prezentuje zakładkę zaraz po ustanowieniu połączenia z bazą danych.



Rys. 41 Zakładka „komunikaty” okna głównego systemu FuzzyQ

- Historia – każdorazowe pojawienie się nowej treści komunikatu w zakładce „Komunikaty” powoduje przeniesienie treści starego komunikatu do pola tekstowego zakładki „Historia”.



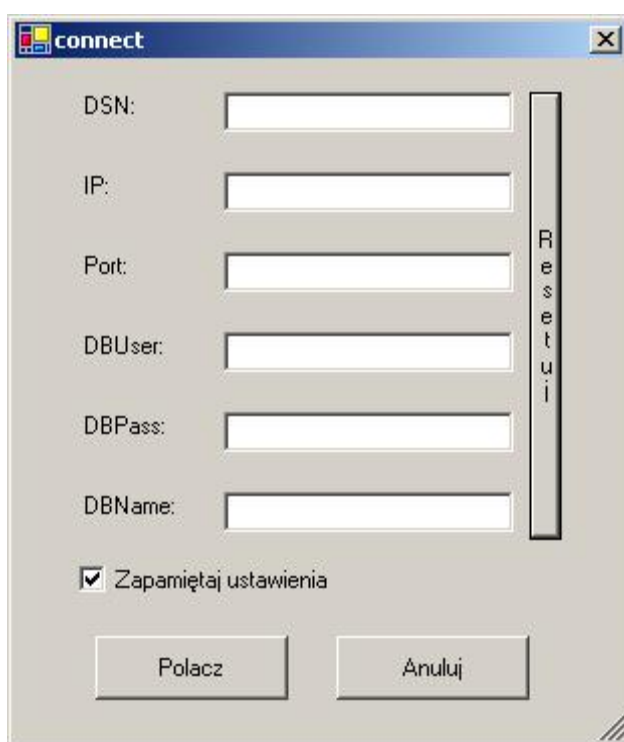
Rys. 42 Zakładka „historia” okna głównego systemu FuzzyQ

Komunikaty w tej zakładce są kolejgowane i oddzielone od siebie separatorem: „----”, jak widać na rysunku (rys. 42).

Każda z trzech zakładek posiada inny odcień tła, aby przy częstym przełączaniu zakładek użytkownik nie „zagubił się” i zawsze wiedział, która z zakładek jest aktualnie zakładką aktywną.

10.2 Okno parametrów połączenia z BD

Aby połączyć się z bazą danych, wybieramy z menu Plik->Połącz lub bezpośrednio klikając przycisk „Połącz” na pasku narzędzi. Pojawi się okno „Connect” z parametrami połączenia (rys. 43).

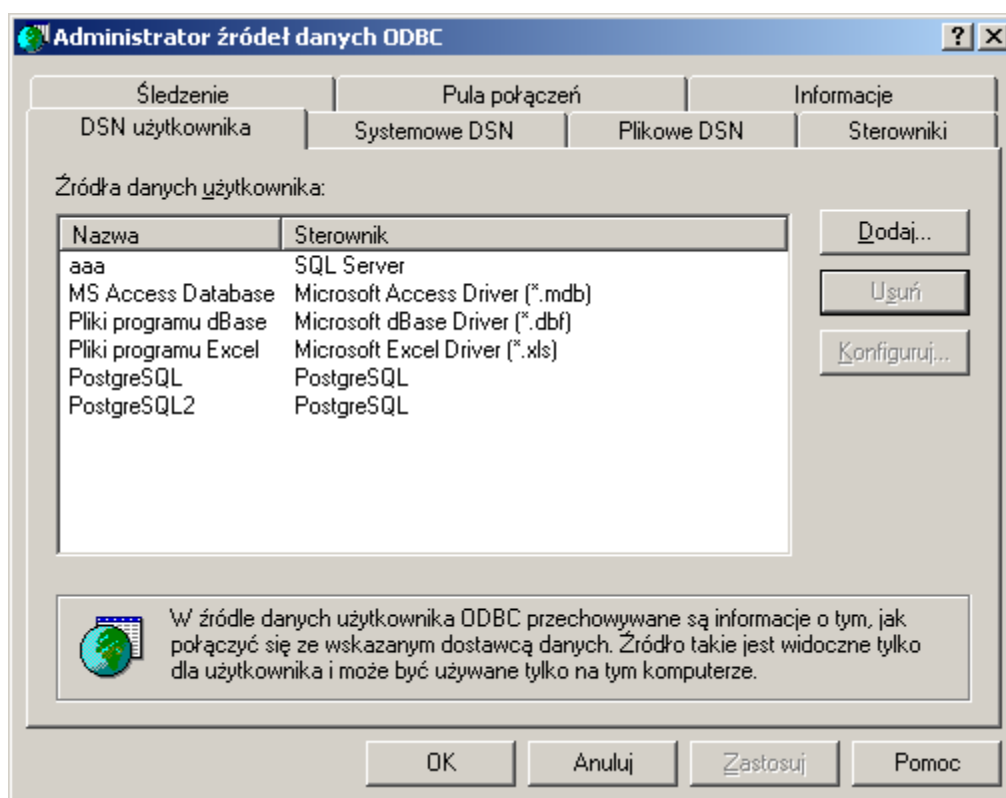


Rys. 43 Widok okna parametrów połączenia

Aby połączenie z bazą danych się powiodło, należy podać następujące parametry:

- DSN (ang. Data Source Name), to źródło danych ODBC zarejestrowane w systemie. Do programu dołączone zostały sterowniki ODBC dla bazy danych PostgreSQL. Po instalacji, w podkatalogu „PSQLOdbc” znajdują się trzy pliki: psqlodbc.dll, psqlodbc.reg i psqlodbc.txt. Pierwszy (*.dll) jest dynamiczną

biblioteką, zawierającą funkcje tworzące interfejs ODBC. Drugi (*.reg) jest plikiem wykonywalnym, zawierającym polecenia dla rejestru systemu Windows, wprowadzającym odpowiednie klucze i wartości do drzewa rejestru, tak aby system mógł korzystać z funkcji pliku psqlodbc.dll. Dodatkowo, bibliotekę dll należy ręcznie umieścić w katalogu systemowym np. „C:\Windows\System32”, ponieważ właśnie w tym katalogu system będzie jej szukać. Ostatni z plików *.txt zawiera skrótowe informacje o instalacji interfejsu ODBC dla bazy danych PostgreSQL. Przed instalacją warto sprawdzić, czy aby na pewno jest to konieczne, czy już nie istnieją w systemie jakieś zarejestrowane sterowniki, z których możnaby skorzystać. Aby tego dokonać, należy w systemie Windows wykonać następujące czynności: START-> Ustawienia-> PanelSterowania-> NarzędziaAdministracyjne-> ŹródłaDanychODBC - Otworzy się okno (rys.44)



Rys. 44 Okno systemu Windows umożliwiające dodanie sterownika ODBC

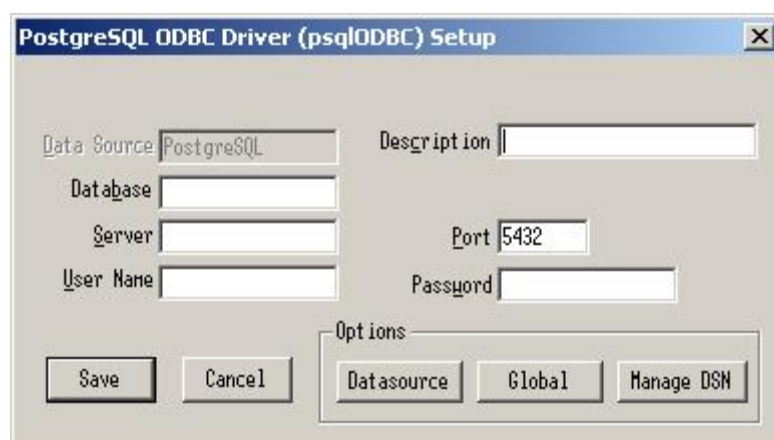
Jeżeli w systemie istnieje już sterownik ODBC dla bazy danych, jego nazwa widoczna będzie w powyższym oknie w kolumnie „Sterownik”. W przeciwnym

razie, musimy dodać go ręcznie za pomocą przycisku „Dodaj”. Na ekranie pojawi się okno kreatora tworzenia nowego źródła danych:



Rys. 45 Widok okna kreatora nowego źródła danych.

Jeżeli wcześniej uruchomiono plik psqldb.reg i rejestracja biblioteki psqldb.dll przebiegła pomyślnie to na liście sterowników powinna znajdować się pozycja PostgreSQL, co widać na rysunku (rys.45). Należy zaznaczyć wybrany sterownik, na bazie którego utworzone zostanie źródło danych i wybrać przycisk „Zakończ”. System poprosi nas o wprowadzenie parametrów w oknie „PostgreSQL ODBC Driver Setup” (rys.46).



Rys. 46 Widok okna parametrów połączenia ze źródłem danych PostgreSQL

Nie trzeba podawać na tym etapie żadnych parametrów, należy nacisnąć przycisk „Save” i nazwę tak tworzonego źródła można wpisać w polu „DSN” aplikacji FuzzyQ. Ręczne wprowadzanie źródła danych ODBC posiada wiele różnych opcji, które są szczegółowiej opisane w publikacjach na temat administracji systemu Windows.

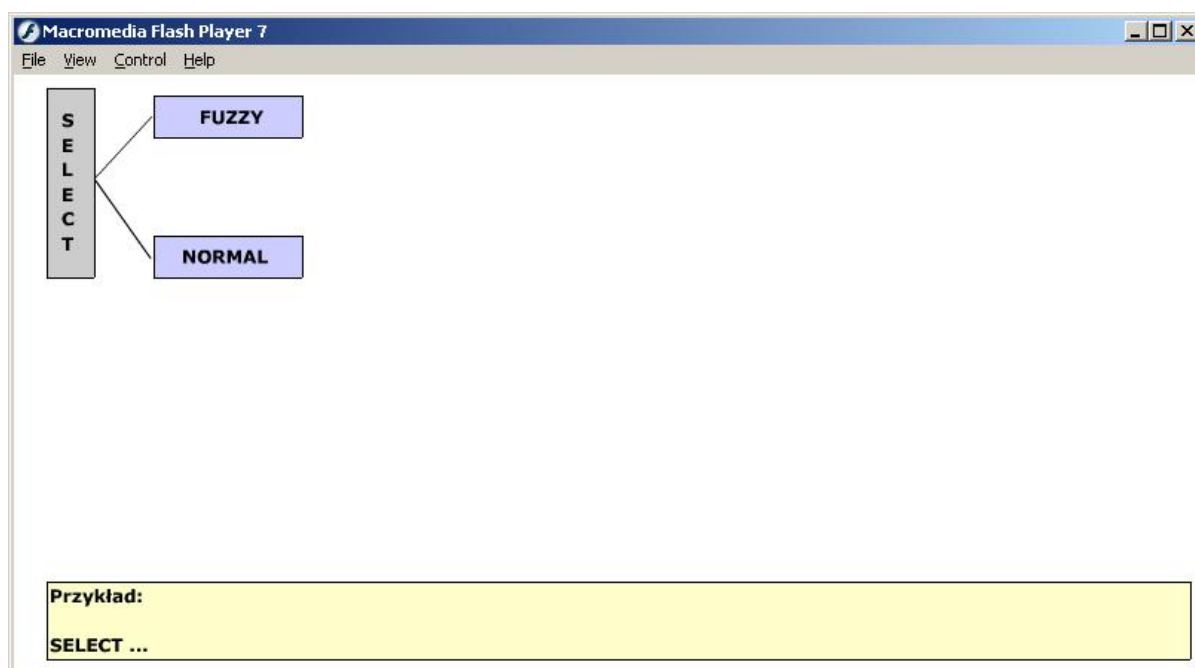
- IP – w polu tym należy podać adres IP bazy danych, z którą zamierzamy się połączyć,
- Port – to port, przez który będzie się łączyć z wybraną bazą danych, aby uzyskać informację na jakim porcie możliwe jest połączenie należy skontaktować się z administratorem bazy danych
- DBUser – jest to nazwa konta użytkownika bazy danych, w szczególnych przypadkach może być to nazwa użytkownika systemu Windows. Zależy to od tego, czy baza danych pozwala na łączenie się z kont systemowych (Windows Authentication),
- DBPass – to hasło dostępowe użytkownika,
- DBName – to nazwa startowej bazy danych,

Gdy wszystkie parametry zostaną podane, można przystąpić do próby połączenia naciskając przycisk „Połącz”. Jeżeli połączenie przebiegnie poprawnie (program nie zwróci żadnych błędów) oraz, gdy w momencie połączenia pole „Zapamiętaj ustawienia” było zaznaczone, system utworzy w katalogu programu plik o nazwie „po-lacz.ini”. W pliku tym zapisze wszystkie parametry połączenia z wyjątkiem hasła. Przy każdym następnym uruchomieniu program wczyta zawartość pliku i wypełni automatycznie odpowiednie pola. Przycisk „Resetuj” wyczyści wszystkie pola tekstowe służące do wprowadzania parametrów połączenia, a przycisk „Anuluj” spowoduje zamknięcie okna i powrót do okna głównego aplikacji FuzzyQ.

10.3 Kreator zapytań

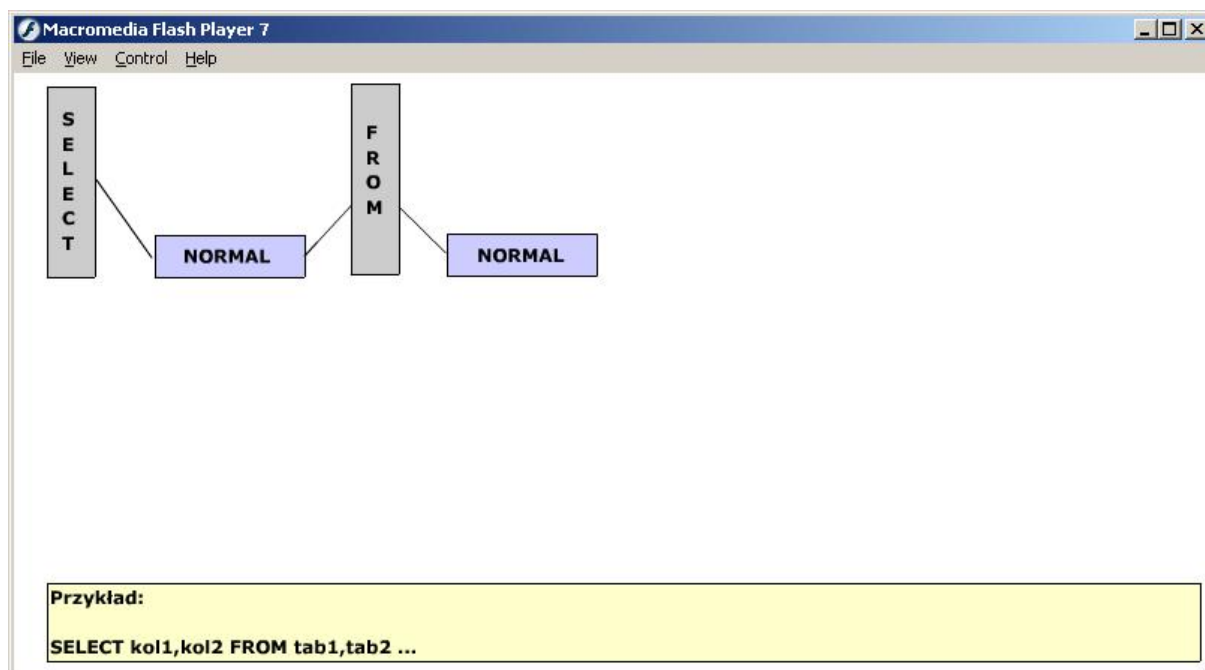
Kreator zapytań jest dydaktyczną częścią całego systemu. Za jego pomocą użytkownik dowie się, jak budować zapytania w języku SQL z zastosowaniem warunków rozmytych. Jego głównym zadaniem jest wskazanie miejsc występowania wartości rozmytych. Kreator zapytań jest uruchamiany z poziomu okna głównego.

Moduł ten został napisany w języku programowania flash z wykorzystaniem języka Action Script 2.0. Jest to interaktywny film, czyli taki, w którym główną rolę odgrywa użytkownik. To od jego decyzji zależy, jak będzie się rozwijać tworzenie zapytania. Budowany szkielet zapytania jest widoczny na każdym etapie w polu „Przykład”. Program startuje od słowa kluczowego SELECT. Następnie użytkownik podejmuje decyzję, czy element rozmyty ma wystąpić po danym słowie kluczowym, czy nie. Podjęcie decyzji jest równoznaczne z naciśnięciem odpowiedniego przycisku. Wybór składa się najczęściej z dwóch przycisków FUZZY oraz NORMAL:



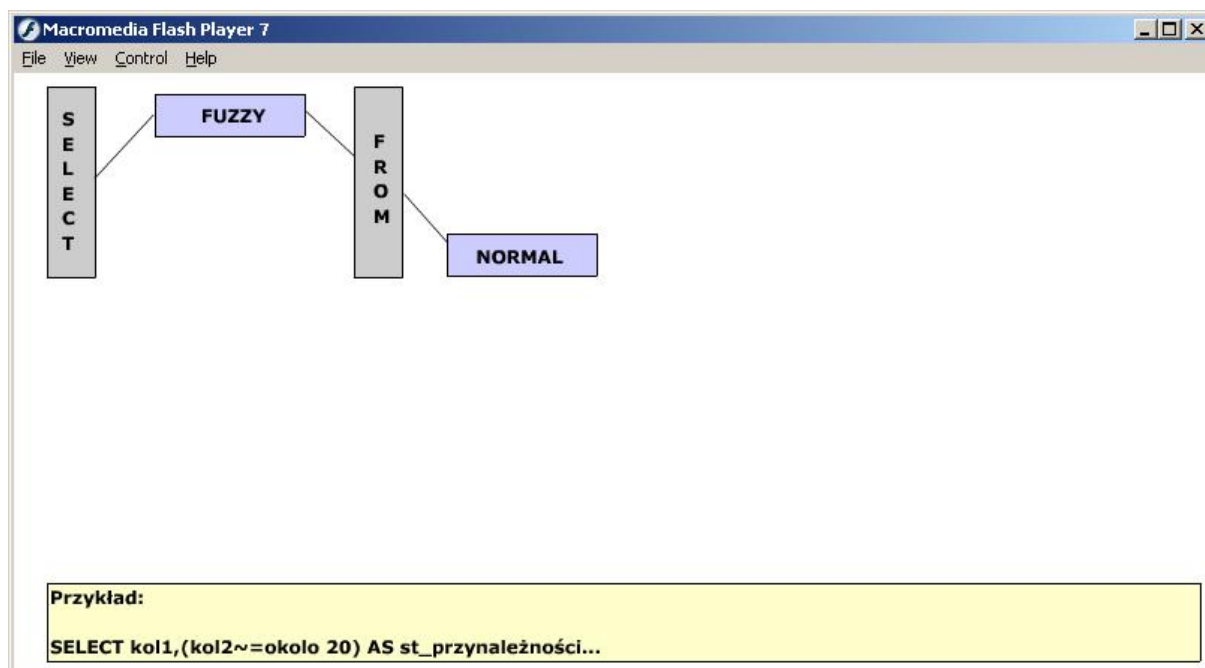
Rys. 47 Okno „Kreatora zapytań” – przykładowy wybór

Wybranie jednej z opcji powoduje przejście do kolejnego słowa kluczowego. Jednocześnie druga z możliwości, która nie została wybrana, zostaje usunięta. Rysunki (rys. 48 i rys. 49) przedstawiają dwie sytuacje. W pierwszej z nich po słowie kluczowym SELECT wybrano opcję NORMAL (rys. 48).



Rys. 48 Okno „Kreatora zapytań” – wybór opcji normal

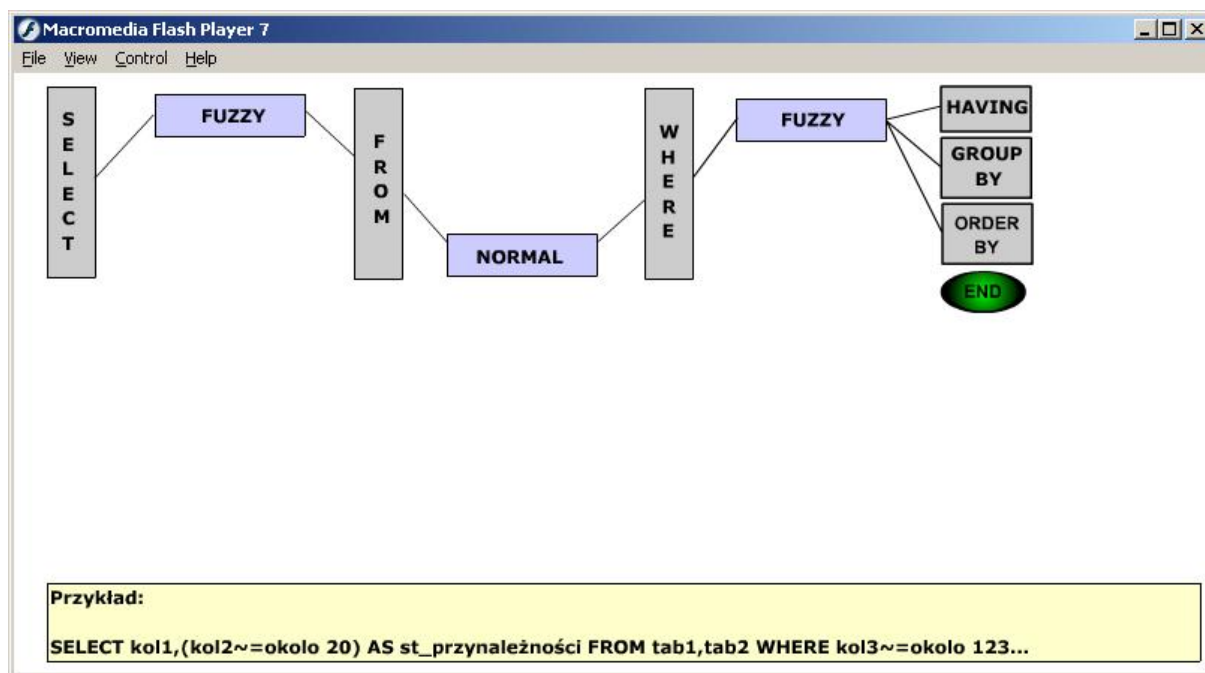
Natomiast w drugim wybrano opcję FUZZY (rys. 49).



Rys. 49 Okno „Kreatora zapytań” – wybór opcji fuzzy

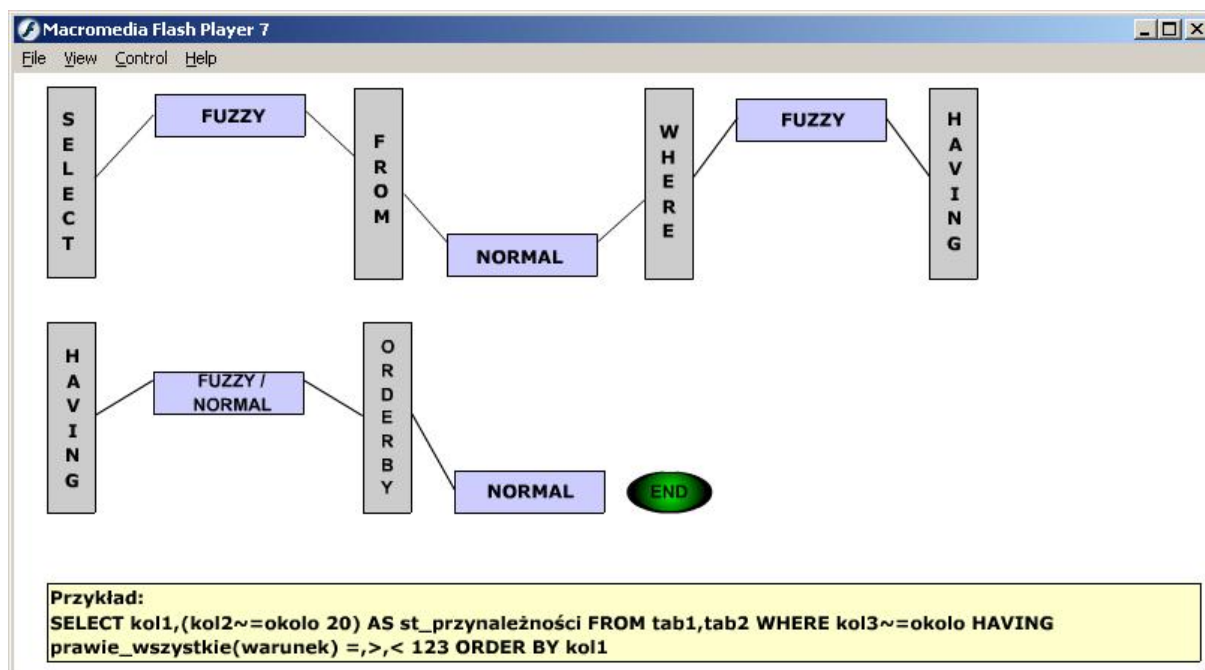
Jeżeli po słowie kluczowym nie może wystąpić element rozmyty, użytkownik będzie miał do wyboru tylko jedną decyzję. Taki przypadek występuje po słowie kluczowym FROM, co widać na rysunkach (rys. 48 i rys. 49). Na tym przykładzie naciśnięcie

przycisku NORMAL powoduje przejście do wyboru słowa kluczowego, ponieważ po słowie kluczowym FROM mogą wystąpić słowa WHERE, HAVING, GROUP BY i ORDER BY. Pojawia się również dodatkowa opcja wyboru. Jest nią zielony przycisk END. Wybranie go powoduje zakończenie budowy zapytania na tym etapie. Przycisk występuje wszędzie tam, gdzie zakończenie zapytania jest możliwe.



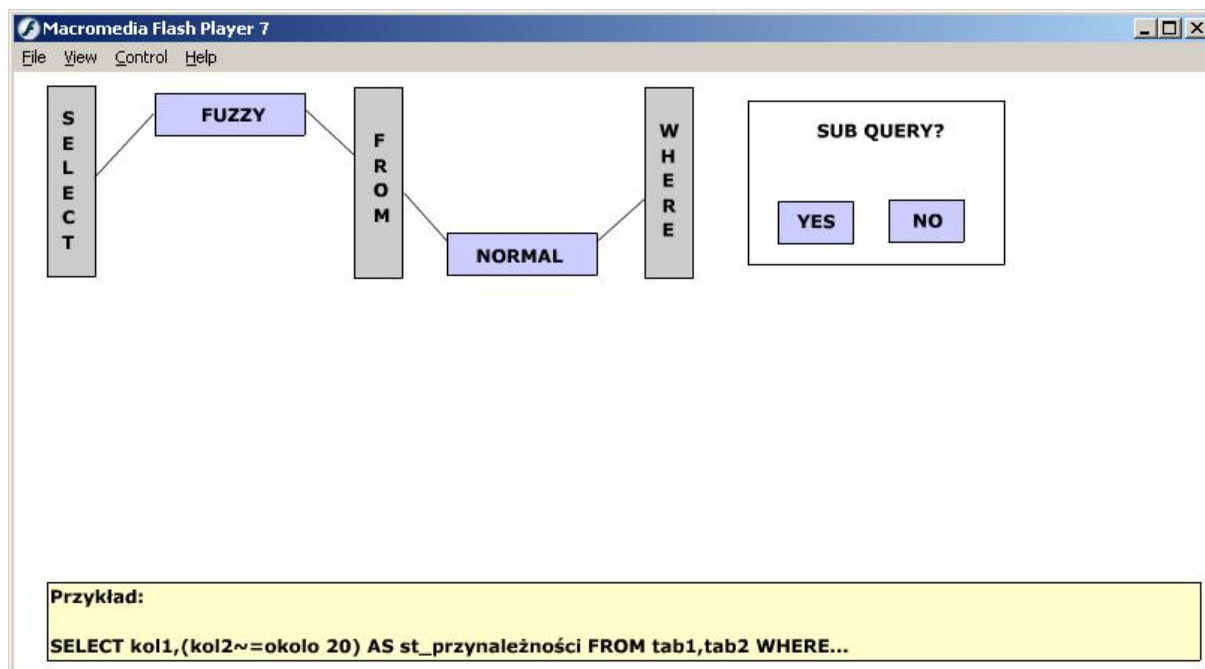
Rys. 50 Okno „Kreatora zapytań” – przykładowe miejsce występowania przycisku end

Naciśnięcie przycisku „END” powoduje wysłanie do aplikacji głównej zapytania utworzonego w polu „Przykład”. Po wybraniu słowa kluczowego „ORDER BY” nie może wystąpić już żadne inne słowo kluczowe i jedyną dostępną dla użytkownika opcją będzie „END” (rys. 51).



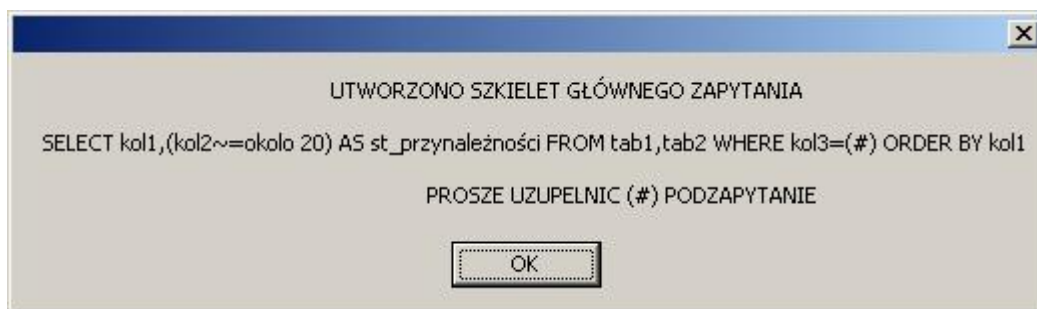
Rys. 51 Okno „Kreatora zapytań” – przykładowe zapytanie rozmyte zakończone słowem kluczowym order by

Po słowie kluczowym „WHERE” oraz „HAVING” może wystąpić podzapytanie jako jedna ze stron równania/nierówności. Użytkownik decyduje czy chce utworzyć zapytanie złożone wybierając jedną z opcji: „YES” lub „NO” (rys. 52).



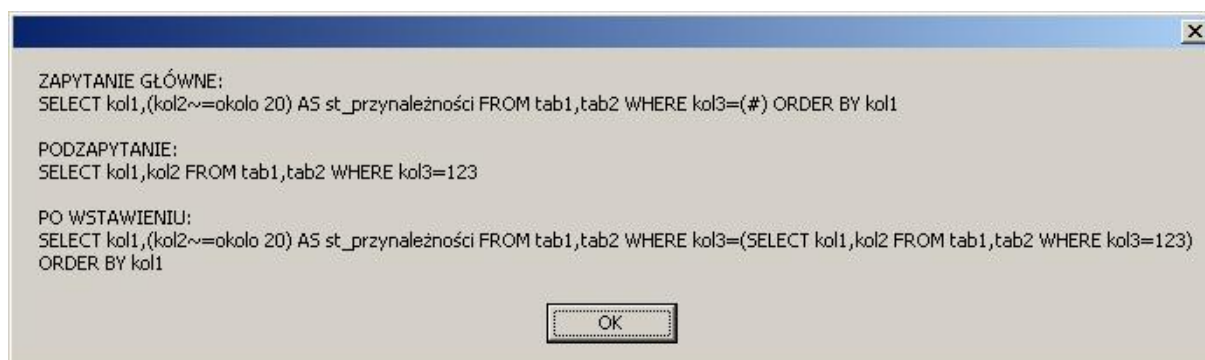
Rys. 52 Okno „Kreatora zapytań” – tworzenie zapytania złożonego

Jeżeli użytkownik zdecydował się na zapytanie złożone, zostanie on poproszony o kontynuowanie zapytania głównego, następnie, gdy zakończy tworzenie zapytania głównego, kreator powróci na początek, aby użytkownik wprowadził szkielety podzapytań. Po każdym zakończeniu zapytania program informuje, na jakim etapie tworzenia znajduje się użytkownik. Jeżeli jest to zakończenie zapytania głównego pojawi się komunikat (rys. 53).



Rys. 53 Okno „Kreatora zapytań” – komunikat zakończenia budowy zapytania głównego

Znaczek „#” oznacza, że w tym miejscu użytkownik podjął decyzję o wprowadzeniu podzapytania i zostanie on zastąpiony w przyszłości przez treść zapytania. Po zakończeniu budowy podzapytania pojawi się okienko informujące o stanie budowy (rys. 54).



Rys. 54 Okno „Kreatora zapytań” – komunikat zakończenia budowy podzapytania

Jeżeli po wstawieniu podzapytania do zapytania głównego całość jest kompletna, to zostanie ona wysłana do okna głównego aplikacji FuzzyQ. W przeciwnym wypadku kreator znów powróci do słowa wyjściowego „SELECT” i sytuacja będzie powtarzana, aż do momentu skonstruowania wszystkich wymaganych podzapytań. Zbudowane w ten sposób zapytanie zostanie wstawione do pola tekstowego zakładki „Zapytanie” w oknie głównym aplikacji FuzzyQ. Zapytanie to nie jest gotowe, aby przesłać

je do bazy danych. Jest to jedynie szkielet zapytania, użytkownik może w łatwy sposób wyedytować go, aby dostosować jego postać do bazy danych, na której pracuje (np. zmienić nazwy tabel i kolumn) oraz do warunków podanych w zadaniu.

11. Uruchamianie i testowanie systemu FuzzyQ

Każdy programista tworząc aplikację już na etapie projektowania musi zaplanować jej testowanie. Testowanie polega na wyszukiwaniu błędów w działaniu danej aplikacji, a uruchamianie to proces służący eliminacji tych błędów.

Jest wiele różnych metod testowania. W teorii najlepsze efekty daje przejście wszystkich gałęzi grafu przepływu sterowania, uwzględniając wszystkie możliwe zestawy danych wejściowych. Jest to jednak często niemożliwe lub prawie niemożliwe w praktyce ze względu na nieskończenie wielkie zbiory danych wejściowych oraz ograniczenia czasowe. Z tego względu dane testowe ogranicza się do wartości skrajnych, niepopularnych oraz tych najczęściej występujących. Dobrze dobrany zestaw danych daje identyczne wyniki jak przetestowanie wszystkimi możliwymi zestawami danych, dlatego ich dobór jest niezwykle ważny.

System, który jest tematem tej pracy, można podzielić na dwie części ze względu na środowiska programistyczne, w jakich powstawały. Pierwsza z nich to część napisana w Visual Studio .NET, a druga to moduł napisany w Macromedia Flash MX 2004. Obie części były testowane osobno, a następnie w całości po zakończeniu pracy nad systemem.

11.1 Przebieg testowania i uruchamiania na etapie tworzenia systemu FuzzyQ

W trakcie tworzenia aplikacji testowanie i uruchamianie następowało zaraz po zakończeniu implementacji każdej z funkcji systemu. W przypadku, gdy testy takie nie były możliwe, powstawały aplikacje pomocnicze mające na celu sprawdzenie poprawności działania fragmentów aplikacji. Na etapie projektowania założono, że część dydaktyczna aplikacji powstanie w języku Flash. Jednak założenie to było uwarunkowane pomyślnym przebiegiem testów współpracy i komunikacji z główną

częścią aplikacji napisaną w C#. W tym celu utworzono prostą aplikację w języku Flash, która składała się z jednego przycisku i jednego pola tekstowego. Wstępnym celem było wzajemne przysyłanie i odbieranie komunikatów tekstowych. Gdy komunikacja przebiegała bezbłędnie, jej implementacja została przeniesiona do głównego projektu, który powstawał w środowisku Macromedia. Główny projekt ma strukturę drzewa decyzyjnego, którego korzeniem jest wyjściowe słowo kluczowe SELECT. Na początku realizacji tej części projektu testowanie i uruchamianie odbywało się po dopisaniu każdej nowej „gałęzi” drzewa. Kompilacja przebiegała szybko i sprawnie, a przejście każdej z możliwych ścieżek przyczyniło się do wykrycia wielu błędów. Problem pojawił się w miarę rozrostu kodu programu. Każda nawet najmniejsza poprawka wymagała ponownej rekompilacji całego projektu. W końcowej jego fazie kompilacja oraz testy zajmowały bardzo dużo czasu. Projekt osiągnął około 11 tysięcy klatek, z czego klatek kluczowych powstało około 700.

Utworzenie aplikacji pomocniczej było konieczne również na etapie tworzenia kodu odpowiedzialnego za połączenie z bazą danych. Nie istniał wtedy jeszcze graficzny interfejs systemu (GUI), dlatego niezbędne było utworzenie aplikacji, która przetestuje wszystkie funkcje klasy odpowiedzialnej za to połączenie. Po stwierdzeniu poprawności w działaniu modułu, poddano go testom na połączenie z różnymi popularnymi bazami danych, uruchomionymi na różnych środowiskach systemowych. Testy wypadły bardzo dobrze - program łączył się i komunikował z bazami danych:

- MS SQL Server(Windows)
- PostgreSQL Windows
- PostgreSQL Linux
- MySQL Linux

Takie testowanie każdej, dopiero co napisanej części spowodowało wykrycie i usunięcie wielu błędów, które w późniejszym etapie realizacji mogłyby być trudne do wyeliminowania.

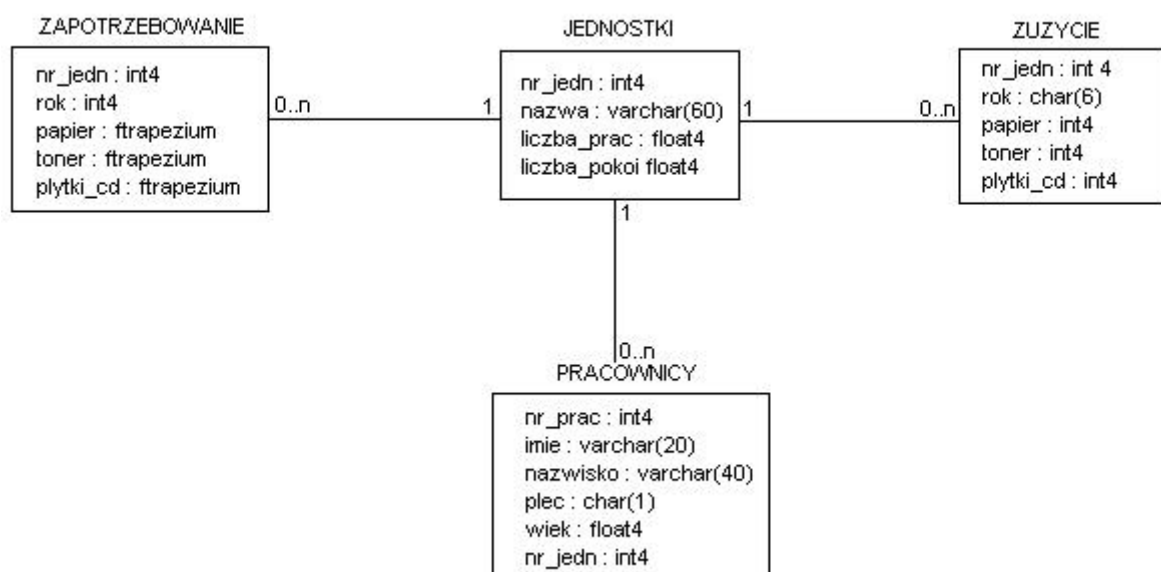
11.2 Przebieg testowania i uruchamiania systemu FuzzyQ na etapie wersji „alpha”

Wersją „alpha” systemu przyjęło się nazywać skończony projekt, ale będący w trakcie finalnych testów. Testy te wraz z przykładowymi scenariuszami działania zostaną opisane w tym rozdziale.

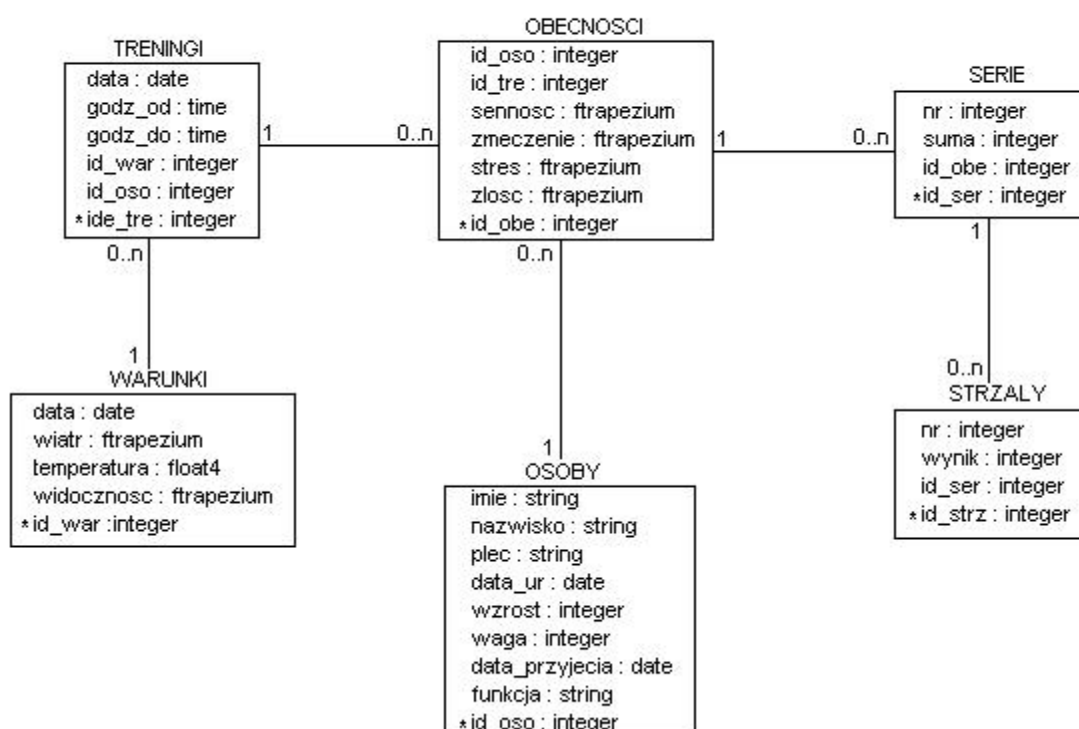
Gotowy system, złożony z przetestowanych modułów, został poddany dogłębnym testom polegającym na stosowaniu różnych scenariuszy działania. Szczególny nacisk położono na tzw. test „złośliwego użytkownika”, czyli starano się wykonać wszystkie nietypowe sytuacje, które przy normalnym użytkowaniu nie miałyby miejsca. Do testowania użyto bazy danych PostgreSQL na platformie Linux. Baza ta jest dostępna pod adresem IP :157.158.191.13 i porcie: 5432. Aby w sposób automatyczny zostały wypełnione parametry połączenia, należy w katalogu programu utworzyć lub wyedytować plik o nazwie „polacz.ini”. Jego zawartość powinna wyglądać następująco:

PostgreSQL2#157.158.191.13#5432#wlochaty#wlochaty

W polu DBPass należy podać hasło dostępu do bazy danych, którym jest następujący ciąg znaków podany w cudzysłowach :”wlochaty123”. Połączenie z bazą przebiegło bez problemów. W powyższej bazie danych zostały zaimplementowane dodatkowe typy i operatory rozmyte oraz skrypty tworzące bazy danych: „Strzelcy” i „Zakłady” będące częścią pracy doktorskiej autorstwa dr inż. Bożeny Małysiak. Aby przybliżyć strukturę obu baz w pracy zamieszczono ich schematy (rys. 55 i rys. 56).



Rys. 55 Schemat bazy danych ZAKŁADY



Rys. 56 Schemat bazy danych STRZELCY

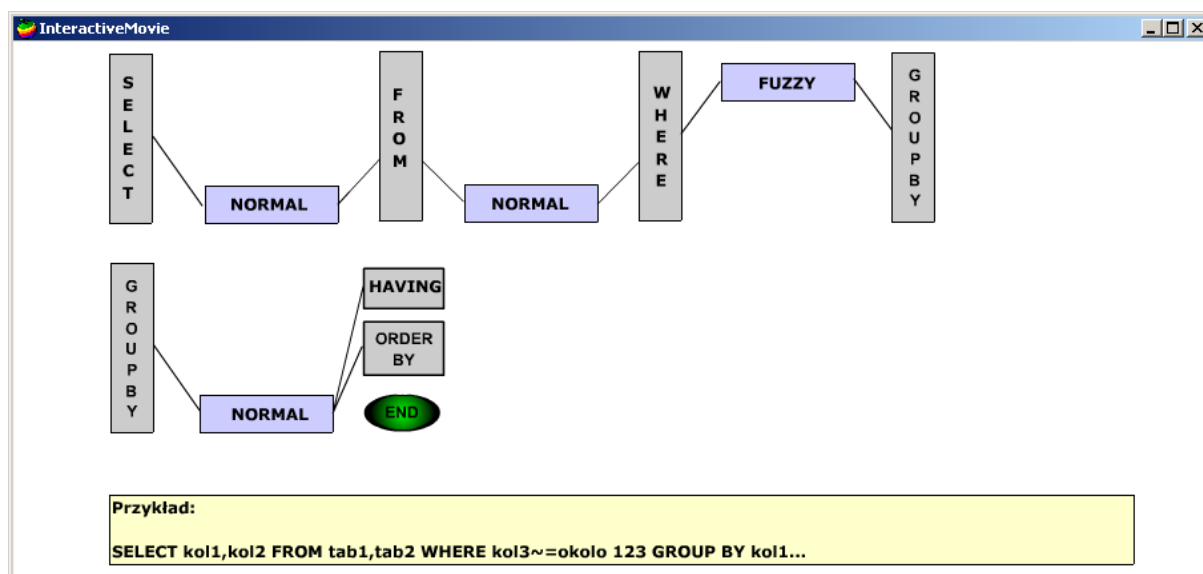
Treści przykładowych zadań, do realizacji których zastosowano zapytania rozmyte języka SQL, zostały utworzone na podstawie przykładów zamieszczonych w pracy

doktorskiej dr inż. Bożeny Małysiak lub w całości zapożyczone w celu przetestowania działania systemu FuzzyQ.

Przykład 1:

„Wyszukaj id oraz daty treningów wraz z liczbą zawodników, którzy przystępując do treningu byli wyspani. W odpowiedzi powinny znaleźć się wszystkie wiersze spełniające ze stopniem zgodności większym niż 0.5 kryteria pytania.”

Widok „kreatora zapytań” dla tego przykładu (rys. 57).



Rys. 57 Kreator zapytań dla przykładu nr.1

Przycisk END zakończył pracę z kreatorem i wygenerował następujący szkielet zapytania:

SELECT kol1,kol2 FROM tab1,tab2 WHERE kol3~=okolo 123 GROUP BY kol1

Aby zapytanie to zwróciło poprawne wyniki musi zostać dostosowane do odpowiedniej bazy danych, czyli w tym przypadku do bazy danych „STRZELCY”. Kolumny jakie muszą zostać zwrócone to: „id_tre”, „data” z tabeli „treningi” oraz „count(id_oso)” z tabeli „obecności”. Dodatkowo należy dokonać złączenia obu tabel za pomocą operatora JOIN:

SELECT t.id_tre,t.data,count(ob.id_oso) FROM obecności ob JOIN treningi t ON
ob.id_tre = t.id_tre *WHERE kol3~=okolo 123 GROUP BY kol1*

Na dalszym etapie musimy dostosować warunek filtrujący we frazie WHERE oraz grupujący we frazie GROUP BY. Zadaniem jest wyszukanie osób, które były wyspane, a więc musimy odwołać się do pola „senność” w tabeli „obecności”. Korzystamy z gotowej funkcji i warunek we frazie WHERE prezentuje się następująco: *sennosc ~= wyspany()*. W zadaniu podane mamy, że stopień przynależności powinien być większy od 0.5 a więc ostatecznie zapytanie z dostosowanym warunkiem filtrującym wygląda tak:

```
SELECT t.id_tre,t.data,count(ob.id_oso) FROM obecności ob JOIN treningi t ON  
ob.id_tre = t.id_tre WHERE (sennosc ~= wyspany())> 0.5 GROUP BY kol1
```

Ostatnim krokiem będzie dostosowanie warunku grupującego. Ostateczna postać zapytania wygląda następująco:

```
SELECT t.id_tre,t.data,count(ob.id_oso) FROM obecności ob JOIN treningi t ON  
ob.id_tre = t.id_tre WHERE (sennosc ~= wyspany())> 0.5 GROUP BY t.id_tre,t.data
```

W wyniku wykonania zapytania otrzymano zbiór wynikowy, zaprezentowany na rysunku 58.

Wynik Zapytania :			
	id_tre	data	count
►	4	2002-01-16	2
	3	2002-01-11	4
	5	2002-01-18	7
	2	2002-01-09	5
	8	2002-01-30	10
	9	2002-02-01	3
	10	2002-02-06	2
	7	2002-01-25	2
*			

Rys. 58 Wynik zapytania z przykładu nr. 1

Przykład 2:

„Wyszukaj treningi przeprowadzone w najslabszej widoczności, na których choć jeden zawodnik trafił dziesiątkę.”

Postępując podobnie jak w przykładzie pierwszym tworzymy szkielet zapytania za pomocą „kreatora zapytań”. Wygenerowano szkielet następującej postaci:

SELECT kol1,(kol2~=około 20) AS st_przynależności FROM tab1,tab2 WHERE kol3=(SELECT kol1,kol2 FROM tab1,tab2)

W przykładzie tym będziemy dostosowywać zapytanie do bazy danych „STRZELCY”. Kolumny, które muszą zostać zwrócone to „ id_tre” , „data” z tabeli treningi oraz widoczność jako zmienną lingwistyczną z tabeli warunki. Złączeń za pomocą operatora JOIN dokonuje się zgodnie ze schematem zamieszczonym na rysunku 55. Zapytanie po uwzględnieniu powyższych zmian wygląda następująco:

```
SELECT DISTINCT t.id_tre, t.data, widocznosc_to_str(widocznosc)
FROM treningi t JOIN warunki w ON t.data = w.data
      JOIN obecności ob ON ob.id_tre = t.id_tre
      JOIN serie se ON ob.id_obe = se.id_obe
      JOIN strzały st ON st.id_ser = se.id_ser
WHERE kol3=
      ( SELECT kol1,kol2
        FROM tab1,tab2 )
```

W warunku filtrującym muszą być uwzględnione dwa rozmyte czynniki: „najgorsza widoczność” oraz „najlepszy wynik strzału”. Do ustalenia najgorszej widoczności posłużono się podzapytaniem zwracającym wszystkie możliwe widoczności i wybrano z nich te najgorsze. Kompletne zapytanie, gotowe do wysłania do bazy danych, wygląda następująco:

```
SELECT DISTINCT t.id_tre, t.data, widocznosc_to_str(widocznosc)
FROM treningi t JOIN warunki w ON t.data = w.data
      JOIN obecności ob ON ob.id_tre = t.id_tre
      JOIN serie se ON ob.id_obe = se.id_obe
      JOIN strzały 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.data = w.data)
```

Przykład 3:

„Wyświetl te jednostki, których największe zamówienie na papier wynosiło kilkanaście sztuk. W odpowiedzi powinny znaleźć się wiersze o stopniu przynależności co najmniej 70 procent”

Za pomocą kreatora zapytań tworzy się następujący szkielet zapytania:

```
SELECT kol1,kol2 FROM tab1,tab2 GROUP BY kol1 HAVING sum(kol2) =,>,<  
około 123;
```

W przykładzie tym należy dostosować szkielet zapytania do bazy danych „ZAKŁADY”. Kolumny, które chcemy wyświetlić to „nr_jedn” z tabeli zapotrzebowanie. Po uwzględnieniu tych dostosowań, zapytanie prezentuje się następująco:

```
SELECT nr_jedn FROM zapotrzebowanie GROUP BY nr_jedn  
HAVING sum(kol2) =,>,< około 123;
```

Grupowanie w powyższym przypadku jest oczywiście wg kolumny „nr_jedn”. Po słowie kluczowym „HAVING” korzystamy z funkcji „max” wyznaczającej w tym przypadku maksymalne zamówienie na papier i szukamy takiej wartości, która wynosiłaby około kilkunastu sztuk. W tym przypadku korzystamy z funkcji „około_kilkanaście”. Ostatnim warunkiem na który musimy zwrócić uwagę jest stopień przynależności nie mniejszy niż 70 procent, czyli większy bądź równy 0.7. Końcowe zapytanie prezentuje się następująco:

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

Powyższe trzy zapytania to tylko wybrane przykłady scenariuszy zrealizowanych przez autora, mające na celu sprawdzenie działania systemu w praktyce.

11.3 Wnioski końcowe z testowania i uruchamiania

Powyższe testy systemu wypadły bardzo pozytywnie. Wszystkie znalezione błędy zostały wyeliminowane. Należy jednak pamiętać, że żadne testy nie gwarantu-

ją nam wyeliminowania wszystkich błędów w działaniu systemu. Im większy projekt, tym trudniej zauważyć i wyeliminować wszystkie usterki. Testowanie systemu należy kontynuować szczególnie w początkowej fazie wdrożeń, gdzie ryzyko wystąpienia komplikacji jest zwiększone. Nawet, gdy system po wdrożeniu działa bezawaryjnie należy pamiętać, iż mogą się pojawić inne zewnętrzne czynniki niemożliwe do przewidzenia na etapie projektowania oraz wykonania, które spowodują błąd aplikacji.

12. Podsumowanie

Wszystkie cele, postawione na początku realizacji pracy dyplomowej, zostały zrealizowane. Program FuzzyQ, który jest wynikiem tej pracy, czeka jeszcze jeden ostatni i najważniejszy test. Głównym założeniem projektu jest to, że ma on być aplikacją dydaktyczną. Mimo wszelkich starań, autor programu nie jest w stanie obiektywnie ocenić programu pod względem efektywności nauczania języka SQL rozszerzonego o elementy rozmyte, bez przetestowania go na grupie studentów. Aplikacja składa się z dwóch głównych modułów: części dydaktycznej napisanej w języku Flash, tzw. „Kreatora zapytań” oraz części analizującej zapytania SQL, napisanej w języku C#. Budowa modularna programu umożliwia wprowadzenie poprawek lub dalszej rozbudowy projektu, bez potrzeby tworzenia wszystkiego od podstaw. Być może dopiero praktyczne zajęcia laboratoryjne wskażą niedoskonałości lub zainspirują do wprowadzenia nowych rozwiązań do systemu. Możliwe jest utworzenie np. innej wersji „kreatora zapytań” i uruchamianie go bez jakiegokolwiek ingerencji w kod źródłowy C#.

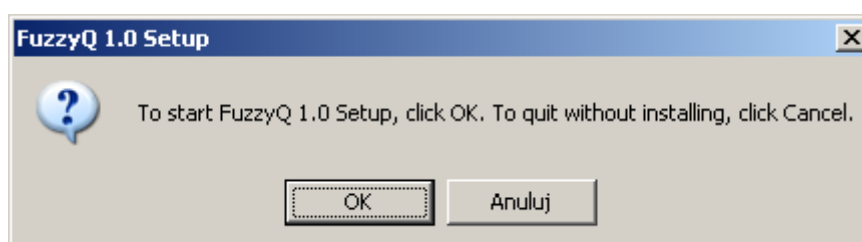
Autor przedstawionego tu projektu, podczas pisania pracy dyplomowej, pozyskał cenne doświadczenia z zakresu programowania komputerów. Zupełnie nieznanym wcześniej autorowi było środowisko Macromedia Flash MX 2004 oraz związany z nim język programowania Flash. Pogłębiona została również wiedza dotycząca programowania w środowisku Visual Studio .NET.

Autor projektu ma nadzieję, że praca włożona w realizację systemu FuzzyQ przyczyni się do wzrostu popularności elementów rozmytych występujących w języku SQL oraz poszerzy wiedzę na temat tego zagadnienia u licznej grupy studentów Politechniki Śląskiej w Gliwicach.

Załączniki:

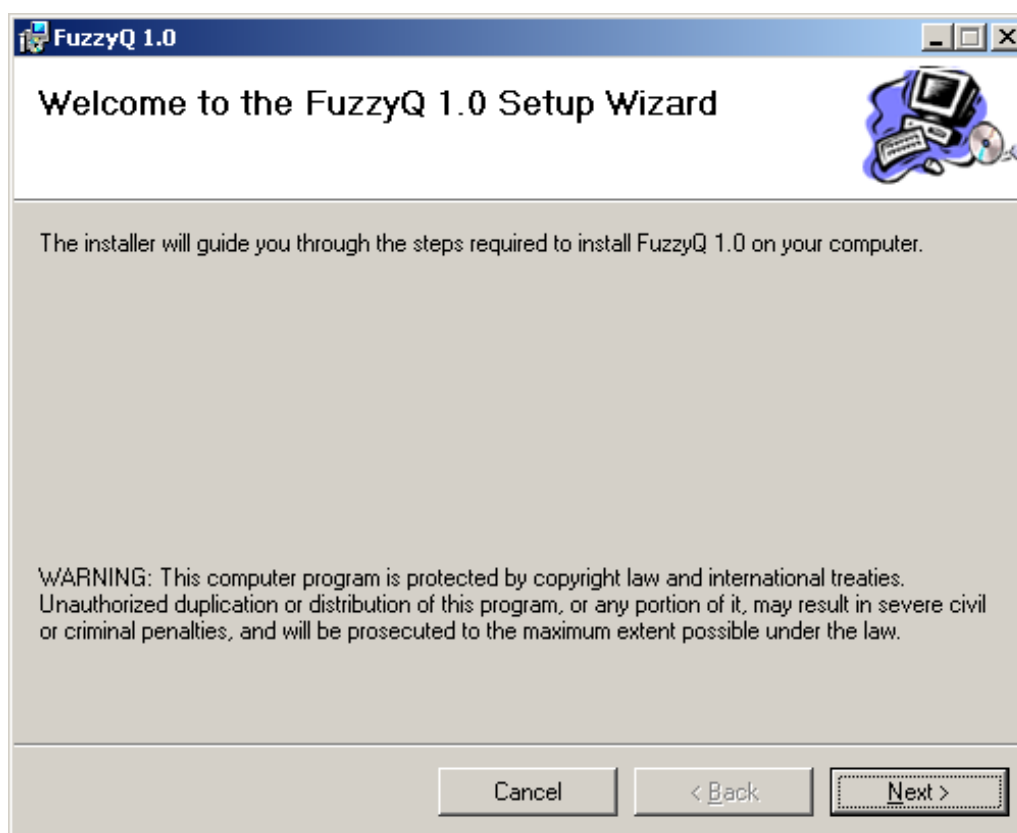
Nr 1: Instalacja systemu FuzzyQ

Instalację systemu rozpoczynamy od uruchomienia programu setup.exe. Pojawi się okno:



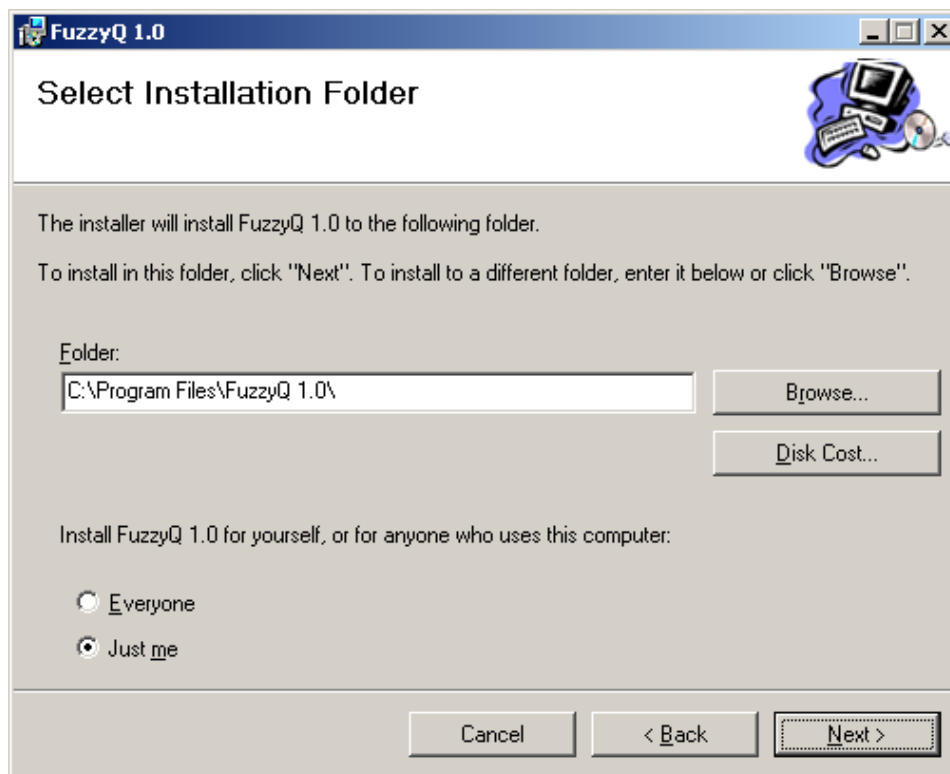
Rys. 1 Instalacja systemu FuzzyQ – komunikat rozpoczęcia instalacji

Aby kontynuować, należy nacisnąć przycisk „OK”, aby przerwać instalację na tym etapie należy wybrać przycisk „Anuluj”. Po naciśnięciu przycisku „OK” program przejdzie do kolejnej formatki:



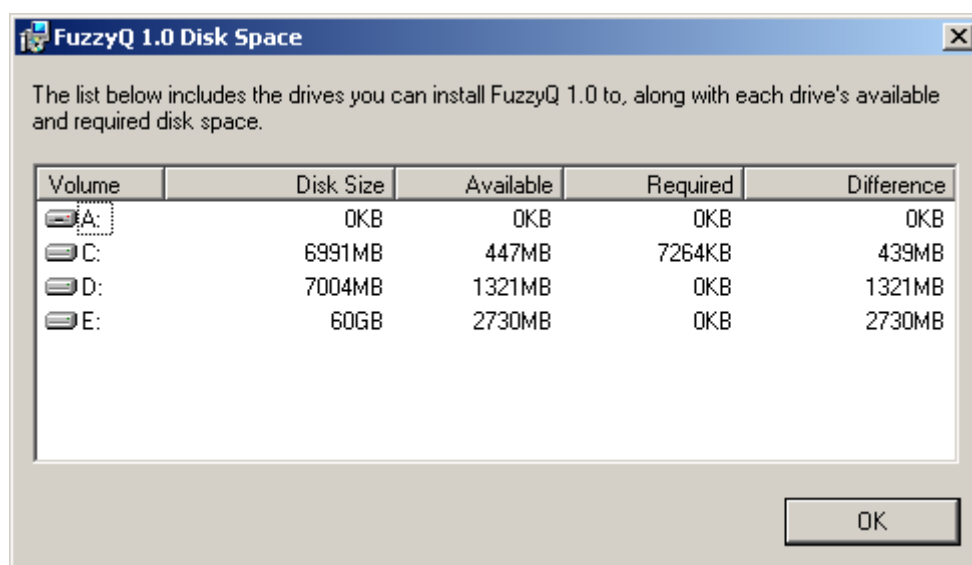
Rys. 2 Instalacja systemu FuzzyQ – komunikat o prawach autorskich

Możliwa jest do wyboru opcja „Next”, która przeniesie do kolejnej formatki albo opcja „Cancel”, która zakończy instalację na tym etapie. Po przyciśnięciu przycisku „Next” pojawi się kolejna formatka (rys. 3)



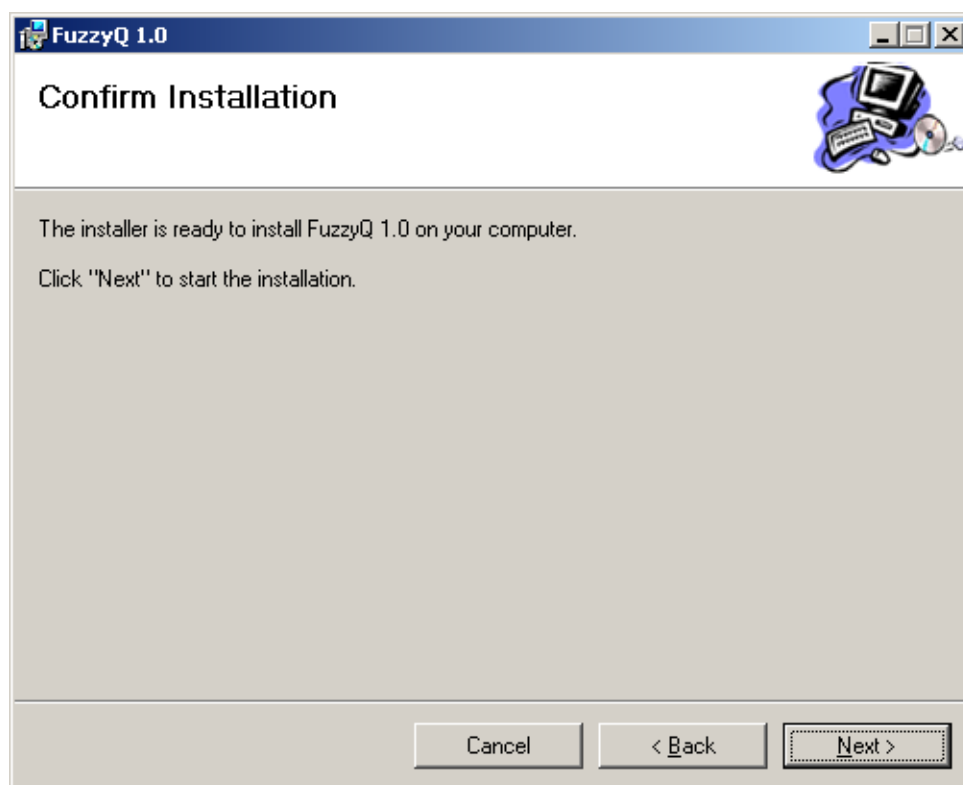
Rys. 3 Instalacja systemu FuzzyQ – wybór folderu instalacji

W formatce tej należy ustawić ścieżkę instalacji aplikacji FuzzyQ oraz dostępność programu na innych kontach użytkowników Windows. Przycisk „DiskCost” pozwala sprawdzić wszystkie partycje pod względem dostępności wolnego miejsca (rys. 4).



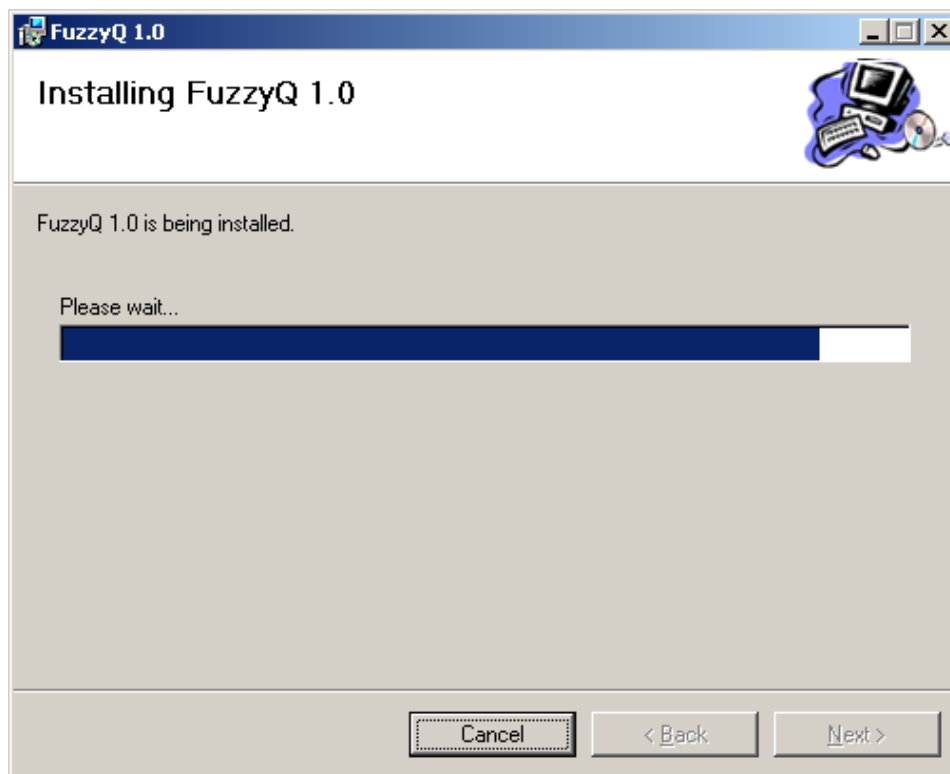
Rys. 4 Instalacja systemu FuzzyQ – informacje o wolnym miejscu na wszystkich dyskach logicznych

Podobnie jak we wcześniejszych formatkach przycisk „Next” pozwala nam na kontynuowanie instalacji, przycisk „Cancel” anuluje ją na tym etapie, a przycisk „Back” pozwoli na powrót do poprzedniej formatki. Po wciśnięciu przycisku „Next” pojawi się formatka (rys. 5).



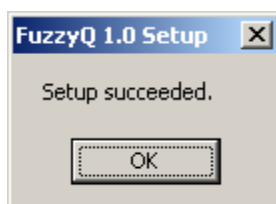
Rys. 5 Instalacja systemu FuzzyQ – potwierdzenie instalacji programu FuzzyQ

Naciśnięcie przycisku „Next” spowoduje rozpoczęcie instalacji programu FuzzyQ. Analogicznie jak w poprzedniej formatce, przycisk „Cancel” kończy instalację, a przycisk „Back” powraca do poprzedniej formatki.



Rys. 6 Instalacja systemu FuzzyQ – postęp instalacji programu FuzzyQ

Powyższa formatka ukazuje postęp instalacji programu FuzzyQ. Po zakończeniu czynności związanych z instalacją pojawi się komunikat o pomyślnym zainstalowaniu aplikacji (rys. 7).



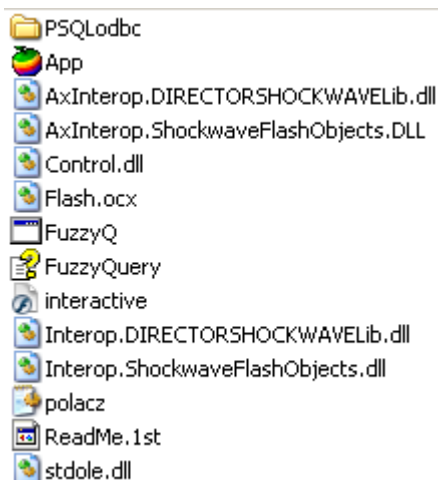
Rys. 7 Instalacja programu FuzzyQ – komunikat zakończenia instalacji programu FuzzyQ

Jeżeli komputer nie posiada zainstalowanego środowiska Microsoft .NET Framework w wersji co najmniej 1.1, program dodatkowo uruchomi instalację Frameworka. Po zakończeniu instalacji na pulpicie znajdzie się ikona skrótu do programu (rys. 8).



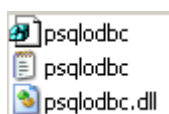
Rys. 8 Ikona programu FuzzyQ

W katalogu podanym podczas instalacji (rys. 9) pojawiają się następujące pliki:



Rys. 9 Widok plików programu FuzzyQ po udanej instalacji w systemie

Katalog „...\PSQLOdbc” zawiera następujące pliki (rys. 10):

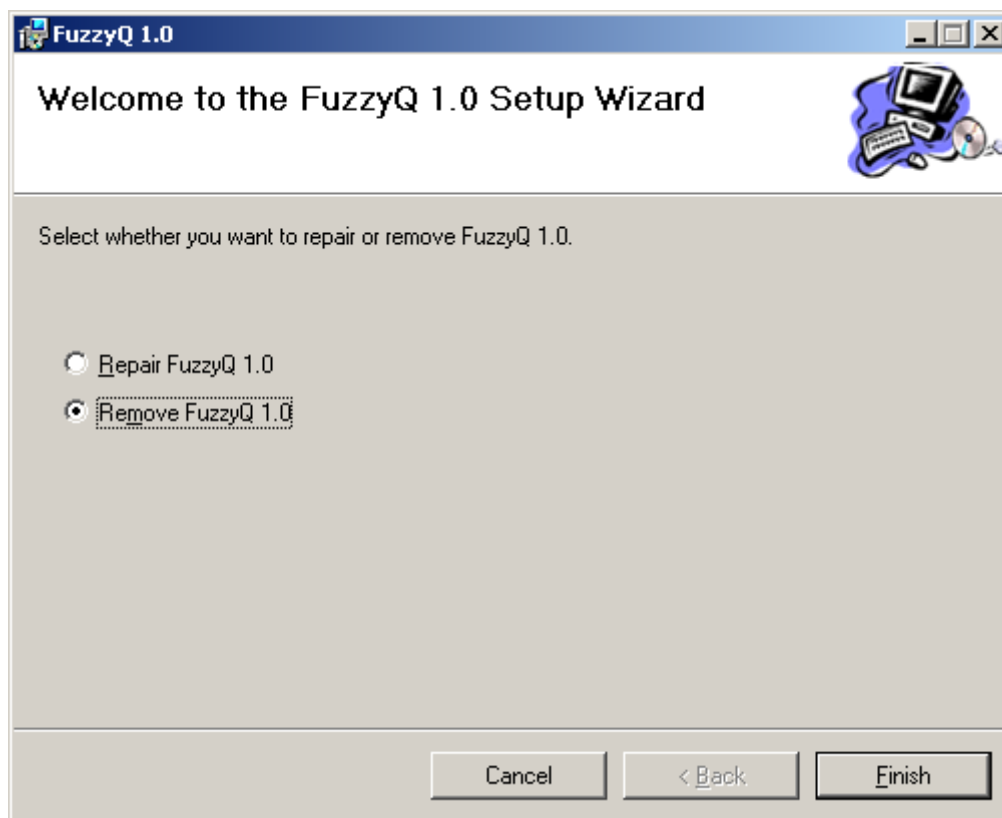


Rys. 10 Widok plików sterownika ODBC dla bazy PostgreSQL

Są to pliki odpowiedzialne za rejestrację sterownika ODBC dla bazy danych PostgreSQL. Pliki te są rozpowszechniane na licencji GNU i zostały dodane do programu, ponieważ w przyszłości ma on współpracować z bazą danych PostgreSQL. Pliki te nie są niezbędne do działania programu FuzzyQ, natomiast, aby program współpracował z bazą danych PostgreSQL, wymagany jest w systemie Windows zarejestrowany sterownik ODBC. Program posiada wsparcie w postaci sterownika ODBC tylko dla bazy danych PostgreSQL, ponieważ do tej bazy danych przewidziane są skrypty z pracy doktorskiej pani dr Bożeny Małysiak, a program docelowo ma współpracować z bazą danych, na której zaimplementowano dodatkowe funkcje i operatory rozmyte, czyli język FuzzySQL^[2]. Aby zainstalować sterownik ODBC bazy danych PostgreSQL należy zapoznać się z opisem znajdującym się w pliku „psqlodbc.txt” i uruchomić plik „psqlodbc.reg”.

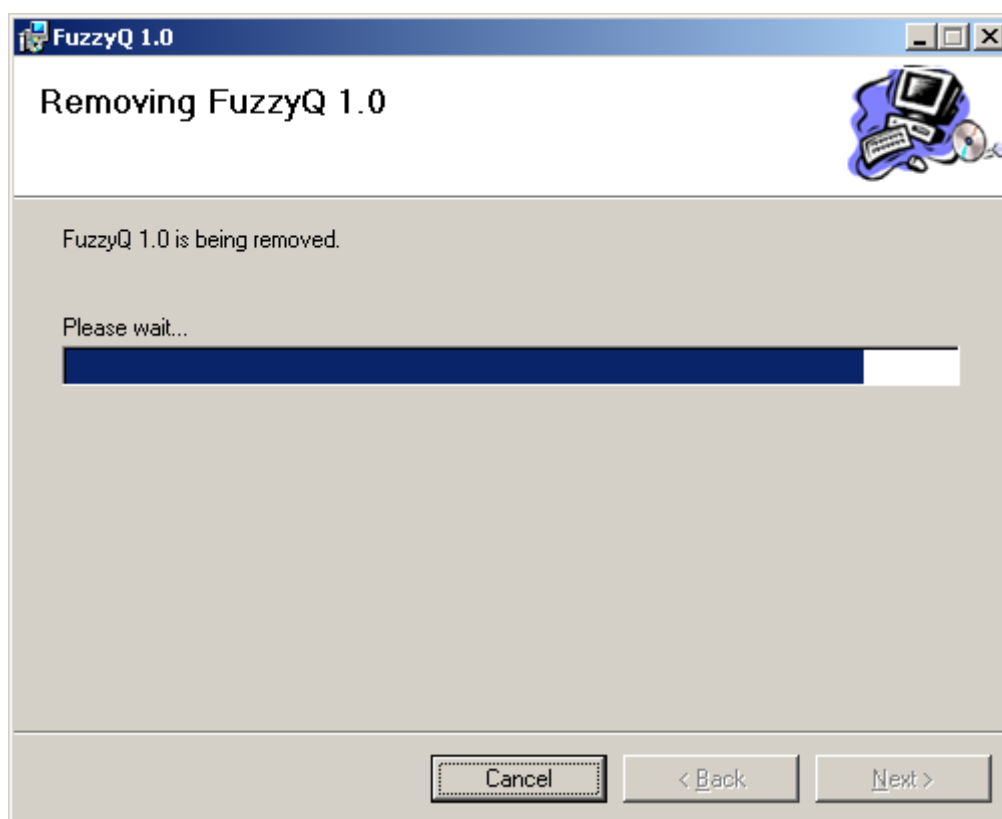
Nr 2: Deinstalacja systemu FuzzyQ

Gdy w systemie istnieje zainstalowana kopia programu FuzzyQ ponowne uruchomienie pliku setup.exe spowoduje wyświetlenie formatki rozpoczynającej proces deinstalacji programu (rys. 1).



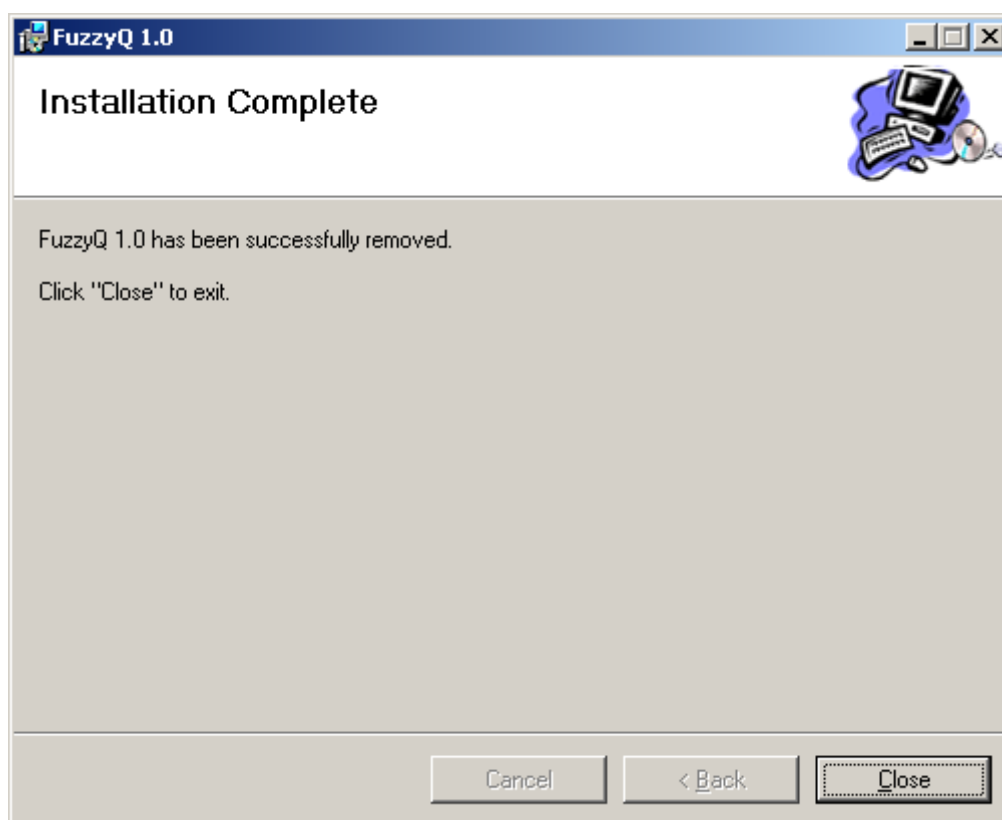
Rys. 1 Deinstalacja systemu FuzzyQ – okno wyboru czynności

Zaznaczenie pola „Repair FuzzyQ 1.0” i naciśnięcie przycisku „Finish” spowoduje reinstalację systemu, natomiast zaznaczenie pola „Remove FuzzyQ 1.0” i zatwierdzenie w analogiczny sposób przyciskiem „Finish” spowoduje odinstalowanie systemu. Podobnie jak przy instalacji, również pojawia się formatka z paskiem postępu (rys. 2).



Rys. 2 Deinstalacja systemu FuzzyQ – Postęp deinstalacji systemu FuzzyQ

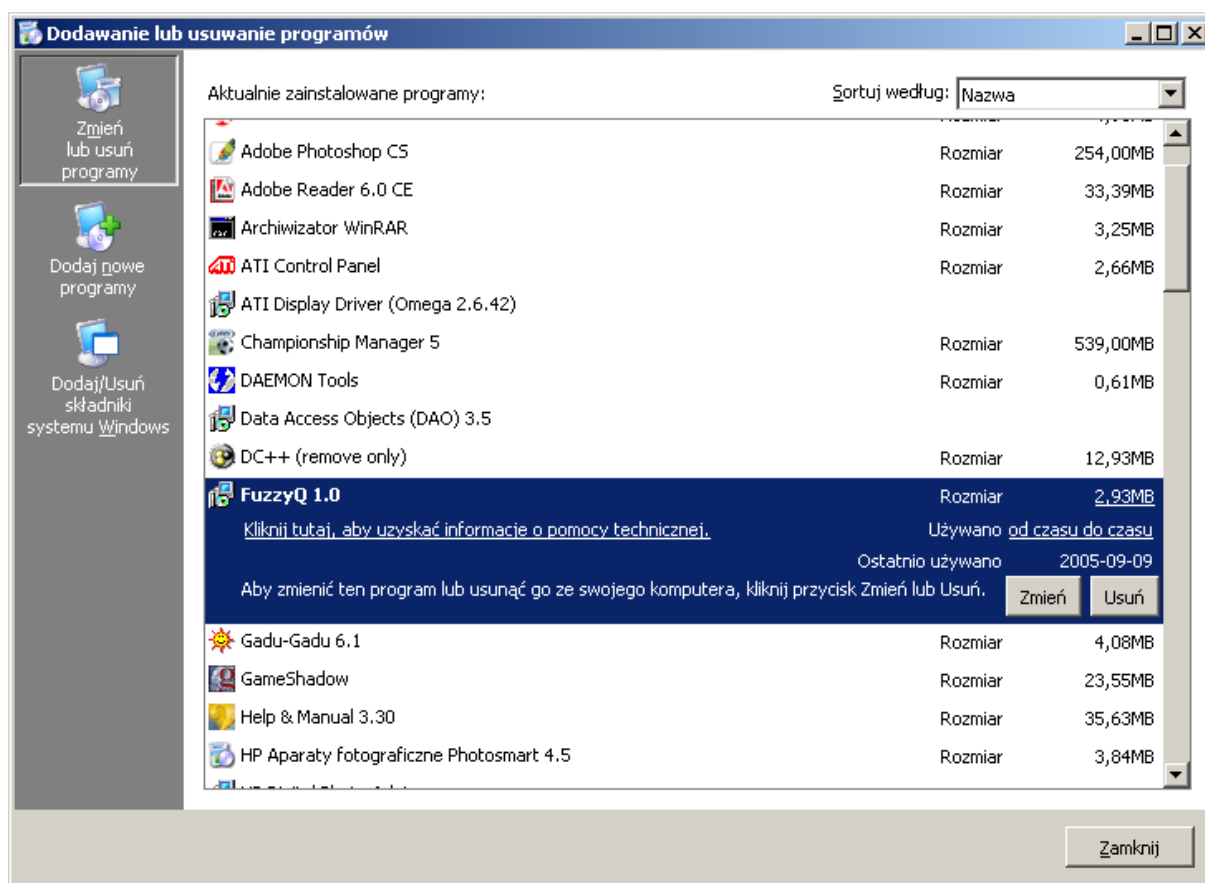
A po odinstalowaniu końcowa formatka przedstawiona jest na rysunku 3.



Rys. 3 Deinstalacja systemu FuzzyQ – komunikat końcowy

Przycisk „Close” kończy działanie deinstalatora.

System można odinstalować również za pomocą narzędzi Windows. W „Panelu sterowania” należy wybrać „Dodaj i Usuń Programy”, następnie zaznaczyć nazwę programu FuzzyQ i nacisnąć przycisk „Usuń” (rys. 4).



Rys. 4 Deinstalacja systemu FuzzyQ – deinstalacja za pomocą narzędzi systemowych Windows

BIBLIOGRAFIA

- [1] Kalen Delaney - „MS SQL SERVER 2000” – Wydawnictwo MICROSOFT PRESS,
- [2] dr inż. Bożena Małysiak - "Metody aproksymacyjnego wyszukiwania obiektów w bazach danych" – rozprawa doktorska,
- [3] <http://pl.wikipedia.org> Wikipedia, wolna encyklopedia internetowa,
- [4] http://www.isep.pw.edu.pl/ZakladNapedu/dyplomy/fuzzy/podstawy_FL.htm
- [5] dr inż. Bożena Małysiak - STUDIA INFORMATICA Volume 23, Number 4 (51), Gliwice 2002 - rozdziały 9 i 10
- [6] dr inż. Bożena Małysiak - STUDIA INFORMATICA Volume 24, Number 2A (53), Gliwice 2003 - rozdział 17
- [7] dr inż. Bożena Małysiak STUDIA INFORMATICA Volume 25, Number 2 (58), Gliwice 2004 - rozdział 9
- [8] http://www.pcai.com/web/ai_info/fuzzy_logic.html
- [9] www.msdn.com The Microsoft Developer Network.
- [10] <http://www.flashfanatiker.de/blog/archives/000032.html>
- [11] http://www.macromedia.com/devnet/flash/articles/stock_history.html
- [12] <http://www.lcc.uma.es/~ppgg/FSQL.html>