Okienkowy język zapytań ciągłych

Marcin Gorawski[1], **Andrzej Skierski**[1]

Streszczenie: W artykule prezentowany jest okienkowy język zapytań CQL w agentowym systemie zarządzania strumieniami danych DSMS. W takim systemie można na bieżąco zadawać ciągłe zapytania w nowatorskim języku CQL, opartym o rozszerzoną i zmodyfikowaną algebrę, opisującą operację na strumieniach danych. Dane w strumieniach napływają na bieżąco, więc nie jest znana ich ilość ani zawartość, jaka zostanie odebrana. Odebrane dane ze strumienia po przetworzeniu są archiwizowane lub niszczone.

Słowa kluczowe: CQL, język ciągłych zapytań, okna.

1. Wprowadzenie

W strumieniowej bazie danych, dane przedstawione są za pomocą strumieni, przez które rozumie się nieograniczone zbiory elementów zawierające krotkę oraz stempel czasowy. System do zarządzania taką bazą, powszechnie nazywamy jest system zarządzania zarządzania strumieniami danych (DSMS-Data Stream Management System). Do obsługi DSMS w systemie Agentów (Gorawski, Bańkowski, 2006), używany jest okienkowy język ciągłych zapytań (CQL - Continues Query Language) (Gorawski, Gębczyk, 2006). Zbiór informacji, zawierający powiązane ze sobą tematycznie dane jest reprezentowany poprzez zbiór rekordów. Rekord to pojedyncza porcja danych składająca się z grupy atrybutów. W artykule rekord należący do schematu strumienia oznaczymy symbolem <s>.

Strumień jest nieograniczonym zbiorem elementów <s,t>, gdzie s to rekord należący do schematu strumienia, a t jest stemplem czasowym elementu.

Nieprzetworzonym strumieniem wejściowym (RAW Input Stream) nazywamy strumień z surowymi danymi z zewnętrznych źródeł np. czujników. Oznaczamy go symbolem Sr, jest on nieskończoną sekwencją rekordów o tym samym schemacie.

Strumienie fizyczne (ang. physical streams) odnoszą się do strumieni przetworzonych w algebrze fizycznych operatorów. Różnią się one od nieprzetworzonych strumieni wejściowych tym, że do każdego rekordu dołączany jest przedział czasowy określający jego czas życia. Elementy strumienia przetwarzane są przez operatory algebry fizycznej zgodnie z przypisanym im terminem ważności. Element wygasa, po upływie zadanego czasu. Elementy wygasłe mogą zostać usunięte z systemu lub zachowane jako dane historyczne (Kramer, Seeger, 2005).

[1] Wydział Automatyki, Elektroniki i Informatyki, Politechnika Śląska, ul. Akademicka 16, 44-100 Gliwice
e-mail: {m.gorawski,a.skierski}@polsl.pl

2. Język CQL

Język CQL zbliżony jest składnią do języka SQL. Składnię instrukcji SELECT można podzielić na następujące główne klauzule (w nawiasach kwadratowych wymieniona jest opcjonalna składnia).

```
SELECT lista_kolumn_selekcji
FROM tabela_1 [, .... , tabela_N]  [ SELECT podzapytanie ....]
[JOIN warunek_złączenia]
[WHERE warunek_wyszukania]
[UNION SELECT]
```

Najprostsze zapytanie składa się z klauzul SELECT i FROM. Wszystkie poprawnie rozpoznane elementy wymienione po słowie kluczowym SELECT są umieszczane w tabeli wynikowej. Każdy z wyliczanych atrybutów rekordów reprezentowany jest jako oddzielna kolumna tabeli. Jako wynik całościowy mogą pojawić się elementy w oddzielnych kolumnach. W języku zdefiniowano także funkcje agregujące, które wymagają wcześniejszego zdefiniowania okna czasowego rekordów. Wynik zapytania można wyobrazić sobie jako wirtualną tabelę, którą można wykorzystać jako źródło danych dla kolejnego zapytania. W klauzuli FROM dopuszczalne jest podanie fizycznej tabeli lub podzapytania, które będzie reprezentowało wirtualną tabelę. Klauzula FROM podobnie jak w SQL'u ma za zadanie wskazanie tabel źródłowych. Dodatkowo w języku CQL istnieje możliwość podania własności rekordów, które mają być wypisane jako wynik zapytania. Wszystkie rekordy, które są przeterminowane, czyli ich datownik jest spoza zakresu zapytania są odfiltrowywane przez operator FROM opisany w dalszej części artykułu. Klauzula WHERE używana jest do określenia warunków względem, których mają być odfiltrowane niepożądane rekordy. Porządek wykonywania działań wewnątrz klauzuli WHERE ustala priorytet operatora. W zdefiniowanym języku przyjęto następujący porządek (malejąco): nawiasy, AND, OR. Mechanizm porównywania wyników korzysta z porządku słownikowego z uwzględnieniem wielkości liter, czyli wartość „KATOWICE" różna jest od wartości „Katowice". Możliwe jest wykonywanie ciągłego zapytania jako sumy kilku podzapytań. Służy temu klauzula UNION, która łączy dane z kilku źródeł dając na wyjście sumę wszystkich dostępnych danych. Zapytanie zawierające klauzule UNION można również potraktować jako tabelę wirtualną i umieścić ją jako źródło danych dla kolejnego zapytania.

```
Select Liczniki.localization
From  (
Select *
From Licznik1
Union
Select *
From Licznik2
Union
Select *
From Licznik3
```

```
Union
Select *
From Licznik4
Union
Select *
From Licznik5
Union
Select *
From Licznik6
) AS "Liczniki"
Where Liczniki.value > 100 AND Liczniki.value <= 1000
```

Podczas definiowania podzapytań istotne jest, podanie wszystkich kolumn, które będą wykorzystane w kolejnych zapytaniach, gdyż nie wymienione kolumny zostaną odfiltrowane. W zamieszczonym powyżej przykładzie: podzapytania operują na licznikach 1-6 oraz wypisują wszystkie dostępne kolumny ze swoich tabel. W rzeczywistości wystarczyłoby wymienić value i localization, gdyż tylko one są wykorzystywane w głównym zapytaniu. Dane umieszczane są do strumieni wejściowych w kolejności w jakiej są ułożone w bazie danych (zgodnie ze swoim datownikiem). Wykonując zapytanie na strumieniowej bazie danych za pomocą standardowego zapytania, domyślnie ustawia się ważność wszystkich elementów jako nieograniczone w czasie. Cechą charakterystyczną w językach CQL jest możliwość ograniczenia zapytania w czasie:

```
Select ...
From licznik_20 [RANGE 20d]
Where ...
```

Zapis `from licznik_20` określa jako ważne wszystkie rekordy zawarte w tym źródle, natomiast zapis `from licznik_20 [range 20h]` określi ważność elementów na dwadzieścia godzin. Dostępne parametry określające jednostki czasu to: `ms` (mili sekund), `s` (sekund), `m` (minut), `h` (godzin), `d` (dni), `M` (miesięcy), `y` (lat). Jeżeli nie będzie podana żadna jednostka czasu to domyślną jednostką będą minuty.

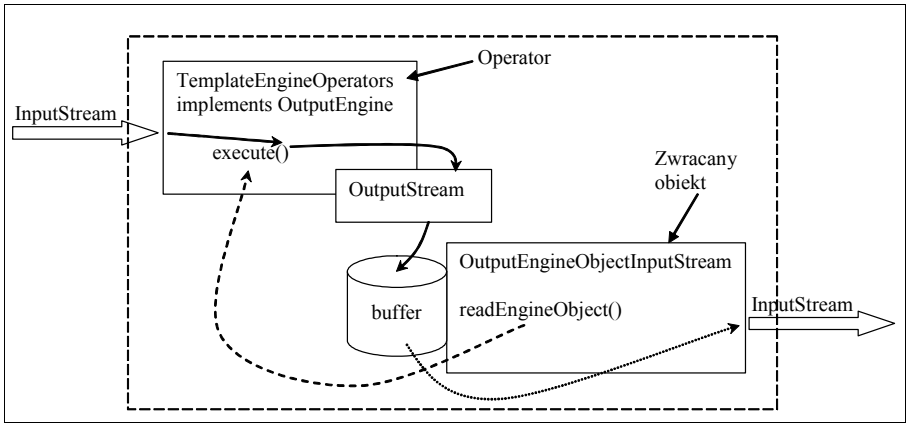
W celu zwiększenia komfortu pracy możliwe jest stosowanie komentarzy w zadawanym pytaniu. Rozróżniane są dwa typy komentarzy. Pierwszy powoduje pominięcie frazy pomiędzy znakami /* i */. Drugi natomiast pomija frazę zapytania za znakiem // do końca wiersza. Komentarzy nie wolno wstawiać do nawiasów kwadratowych.

Polecenia i wartości pisane pomiędzy nawiasami kwadratowymi muszą znajdować się w jednej linii. Natomiast polecenia wpisane pomiędzy nawiasami zwykły-mi mogą posiadać dowolną ilość białych znaków.

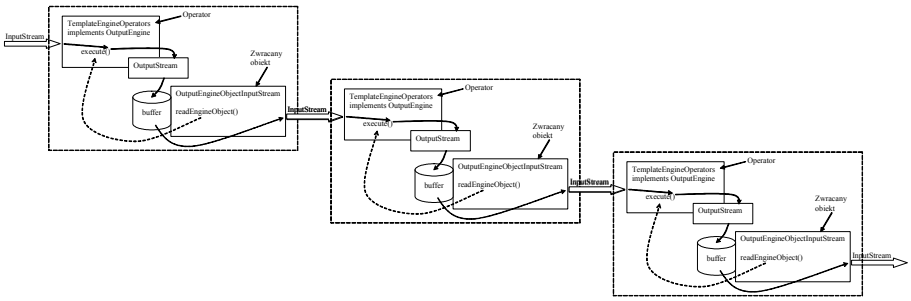
3. Operatory

Budowa operatora została przedstawiona na rysunku 1.

Na wejściu i wyjściu występuje strumień wejściowy, dzięki takiej budowie łatwo łączy się operatory pomiędzy sobą, co przedstawia rysunek 2.



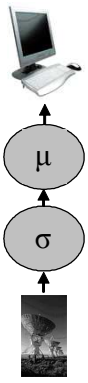
Rysunek 1. Budowa operatora



Rysunek 2. Łączenie operatorów

Operatory po lewej stronie odpowiedzialne są za czytanie danych ze źródeł takich jak plik, bufor czy nadajnik. Pozostałe operatory uruchamiane są kolejno po odebraniu paczki danych. Użytkownik wprowadzając zapytanie, powoduje uruchomienie kompilatora, który tworzy i łączy kolejne operatory. Uruchamianie operatorów rozpoczyna się od operatora konsumpcyjnego położonego najbliżej wyjścia (na rysunku najbardziej po prawej stronie), do tych które są położone najbliżej źródła danych (po lewej stronie) nazywany operatorem strumieniowych danych źródła. W operatorze `Consume` wywoływana jest jego metoda `execute()` mająca za zadanie odczyt obiektów. Obiekty są odczytywane przy pomocy metody `readEngineObject()` z klasy `OutputEngineObjectInputStream`, która w celu odczytu danych wywołuje metodę `execute()` (przerywana strzałka) dla podległego jej operatora. Sytuacja taka powtarza się do chwili, gdy wykonane zostaną wszystkie operatory. Strzałka kropkowana reprezentuje metodę execute(). Przykładowe użycie tych operatorów ma postać:

Drzewo to równoważne jest poniższemu zapytaniu napisanym w trybie tekstowym w języku CQL:



Rysunek 3. Łączenie operatorów

```
Select * from sensor1 where timeID<90000
```

Analizator składniowy z powyższego zapisu wygeneruje poniższe polecenie:

```
consuming(filter(readDataSource('192.168.0.1 port 7000.dat'),
'timeID<90000')) .readEngineObject();
```

Operatory źródła i konsumpcji są kluczowymi elementami to tworzenia zapytań, natomiast wszystkie pozostałe operatory są elementami pomocniczymi.

Operator wzorcowy
Wejście : *stream Sin, dodatkowy parametr*
Wyjście : *stream Sout*

```
1 while (true)
2   PhysicalOperator ph = readEngineObject();
3   if (ph==null)
4     write(ph);
5     close();
6     return;
7   else if (warunek())
8     write(ph);
9   return;
```

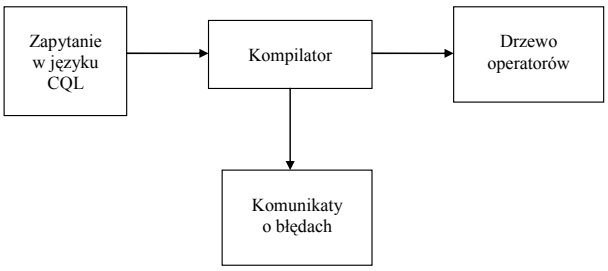
Każdy z operatorów zbudowany jest na podstawie powyższego schematu. Wszystkie są do siebie bardzo podobne. Każdy z nich pobiera obiekty w nieskończonej pętli (wiersz 1) z wejścia, aż do momentu spełnienia jednego z warunków. Warunek pierwszy ma miejsce w przypadku, gdy zapytanie dobiega końca. Ma to miejsce dla zapytania na danych skończonych: w danym momencie czasu wszystkie dane są zgromadzone w bazie, nie nadpływają żadne nowe dane lub użytkownik zakończy ciągłe zapytanie. W takich sytuacjach wysyłany jest obiekt null, który powoduje zakończenie każdego z uruchomionych operatorów.

W wierszu 7 wywoływana jest funkcja, wykonywająca zadania w zależności od

rodzaju operatora. W sytuacji, gdy w funkcji zwróci prawdę to obiekt zostanie wysłany na wyjście

4. Analizator zapytań

Kompilator to program tłumaczący zapis w języku źródłowym na zapis jemu równoważny w języku wynikowym (Aho, Sethi, Ullman, 2002). W systemie strumieniowej bazy danych, językiem źródłowym jest CQL. Za pomocą kompilatora napisane rozkazy tłumaczone są do postaci drzewa operatorów, które następnie jest uruchamiane. Drzewo operatorów to skierowany acykliczny graf, którego wierzchołkami są operatory, a krawędziami są strumienie fizyczne, po których przesyłane są rekordy. Wierzchołki liści służą do pobrania rekordów, wszystkie pośrednie wierzchołki wykonują na nich operacje, a korzeń reprezentuje wynik.



Rysunek 4. Schemat kompilacji zapytania CQL

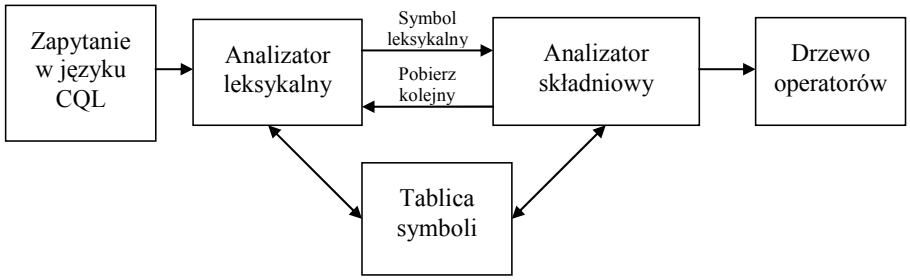
Kompilacja składa się z dwóch części: analizy i syntezy. W pierwszej części wykonuje się rozłożenia tekstu wejściowego na części składowe a następnie tworzy się reprezentację pośrednią. Druga część polega na przekształceniu reprezentacji pośredniej w program wynikowy (Aho, Sethi, Ullman, 2002). W zbudowanym kompilatorze wyróżniamy następujące etapy kompilacji:

- analizę liniową, jej zadaniem jest pobranie znaków wpisanych przez użytkownika i pogrupowanie ich w symbole leksykalne (tokeny)
- analizę hierarchiczną, tworzone jest drzewo rozbioru w oparciu o zdefiniowaną gramatykę
- analiza semantyczna, wyszukiwane są błędy semantyczne oraz wyznaczane informacje potrzebne w fazie generacji kodu wynikowego

4.1. Analiza leksykalna

Analizator leksykalny stanowi pierwszy etap kompilacji. Wczytuje on z wejścia ciąg znaków i konwertuje go na strumień symboli leksykalnych na potrzebę analizy składniowej. Na żądanie od analizatora składniowego, analizator leksykalny pobiera z wejścia znak po znaku aż do momentu zidentyfikowania kolejnego symbolu (Aho, Sethi, Ullman 2002). Lekser tworzy tokeny, których identyfikacja jest

procesem liniowym. Właściwe zdefiniowanie tokenów skutkuje prostszą gramatyką analizatora składniowego.



Rysunek 5. Schemat kompilacji zapytania CQL

Podczas analizy leksykalnej odfiltrowywane są białe znaki (znak odstępu, tabulacji i nowej linii), oraz komentarze. Każdy nierozpoznany ciąg znaków lub znak, będzie powodował wygenerowaniu błędu. Do analizy leksykalnej został użyty program *JFlex* w wersji 1.4.1. *(JFlex: The Fast Lexical Analyser Generator) (Gerwin, 2005)*. Program ten jest odpowiednikiem popularnego Flex, z tą różnicą, że stworzony kod jest wygenerowany w Javie.

4.2. Analiza składniowa

Analizator składniowy zwany również parserem to program, który w oparciu o gramatykę tworzy drzewo rozbioru. Do utworzenia analizy ciągłych zapytań, skorzystano z wsparcia kompilatora gramatyki BYACC/J w wersji 1.13.

Zgodnie z rysunkiem 5, po odebraniu symbolu leksykalnego, analizator składniowy sprawdza, czy zapis jest zgodny z gramatyką języka CQL. Jeżeli okaże się, że wpisany ciąg instrukcji jest różny od żądanej gramatyki, to zostanie wygenerowany błąd i zapytanie się nie wykona. Podczas inicjalizacji drzewa rozbioru, ustala się dodatkowe parametry operatorów, takie jak źródła danych skąd mają być pobierane rekordy, warunki jakie mają być spełnione przy pobieraniu danych itp. Po zakończeniu pracy analizatora składniowego otrzymujemy gotowe do pracy drzewo operatorów.

4.3. Drzewo operatorów

Operatory są tak zaprojektowane aby w prosty sposób skonfigurować je w drzewo operatorów reprezentujące zapytanie w języku CQL.

Rysunek 6. Symbol końcowego wyniku zapytania

Symbol na rysunku 6 oznacza końcowy wynik zapytania, reprezentowany jest po przez klasę `Consume`, magazynuje on w `SweepArea` wszystkie wyniki.

Rysunek 7. Symbol pliku przechowującego dane

Podczas pracy odbiornika, dane gromadzone są w pliku (rysunek 7) dzięki temu jest możliwy dostęp do danych historycznych podczas realizacji zapytania.

Rysunek 8. Symbol pojedynczego operatora

W zamieszczonym poniżej rysunku 9 koło symbolizuje pojedynczy operator, strzałki oznaczają strumienie łączące operatory.

5. Ciągłe zapytania

Pakiety zawierające zmienną `long timeID` dają pewność, że rekordy posiadające swoje własne stemple czasowe zachowają swoją pierwotną wartość. Dla przykładu takimi danymi mogą być najróżniejsze pomiary, gdzie istotny jest dokładny czas pomiaru. W sytuacji, kiedy dane rejestrowane przez system nie zawierają własnego datownika (nie posiadają zmiennej `long timeID`), to system nada mu własny stempel czasowy, zgodny z czasem zarejestrowania danego obiektu w systemie. Takie dane będą przekłamane o czas przesyłu ich ze źródła do systemu.

Zapytanie bez użycia Range:

1 Jeśli zapytanie dotyczy aktywnego połączenia, nie zwalniać buforów
2 Pobierz wszystkie dane z pliku dla operatorów
3 Pobierz dane dla operatorów z buforów jeszcze nie zarchiwizowanych
4 Pobieraj nieustannie kolejne nadchodzące dane ze strumienia wejściowego
5 W przeciwnym razie
6 Pobierz wszystkie dane z pliku dla operatorów
7 Zakończ zapytanie

Algorytm ciągłego zapytania

Do ciągłych zapytań został stworzony pewien system wykonywania niepoprawnych zapytań. W przypadku, gdy użytkownik zada błędne zapytanie, np. w postaci:

```
Select zone, COUNT(zone) AS 'z1', AVG(zone) AS 'z2',
timeID, MAX(timeID)
from table3 WHERE zone < 0
```

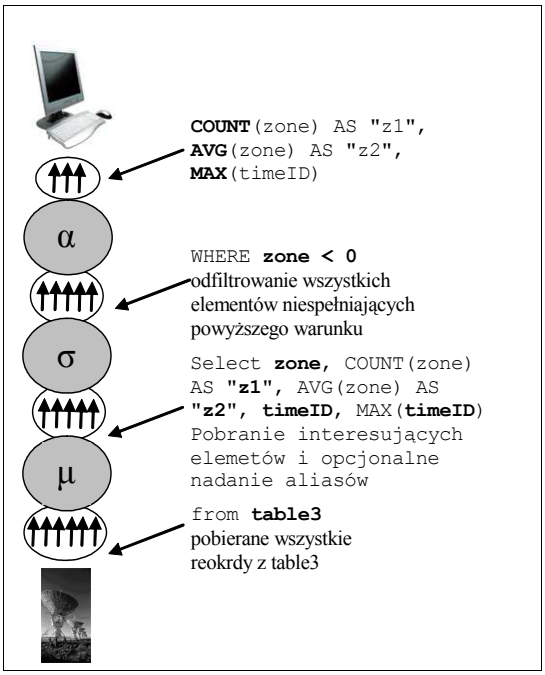
to operatory dzięki swej budowie odfiltrują niemożliwe do wykonania wyniki.

W naszym przypadku zostanie wykonane zapytanie w postaci:

```
Select COUNT(zone) AS 'z1', AVG(zone) AS 'z2', MAX(timeID)
from table3 WHERE zone < 0
```

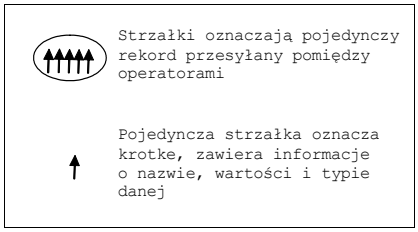
NR	z1	z2	timeID
1	1	-4	1 157 223 600 218
2	2	-4	1 157 223 605 218
3	3	-4	1 157 223 606 218
4	4	-5	1 157 223 607 218
5	6	-3	1 157 223 610 218

Tabela 1. Wydruk zapytania



Rysunek 9. Wykonywanie operatorów

Takie działanie zawdzięczamy szeregowemu wykonywaniu się operatorów. Po wyjściu elementu z operatora filtrowania, mającego na celu pomniejszyć ilość wyników, wszystkie dane wytypowane w selekcji i spełniające warunek filtrowania, trafiają do operatora agregacji. Krotki `zone` i `timeID`w tym miejscu zostają odfiltrowane.



Rysunek 10. Opis strumieni

6. Zakończenie

Celem pracy było stworzenie systemu umożliwiającego wykonywanie ciągłych zapytań na strumieniowej bazie danych. Struktura agentów programowych została rozszerzona o możliwość wykonywania ciągłych zapytań na danych docierających do systemu w bieżącej chwili.

Stworzona prototypowa strumieniowa baza danych, może przechowywać zarejestrowane dane przez urządzenia pomiarowe i przesyłać je strumieniami do innych agentów. Zaprojektowany język CQL ciągłych zapytań został wzorowany na języku SQL.

Do wyliczenia zapytań kierowanych do strumieniowej bazy danych służą operatory fizyczne, połączone ze sobą w strukturze skierowanego grafu acyklicznego. Liśćmi grafu są tabele (źródła danych wysłane przez nadajniki albo inne grafy operatorów), a korzeniem jest tabela wynikowa. Każdy z operatorów otrzymuje na wejście dane strumieniowe, realizuje na nich zdefiniowane operacje, po czym wysyła wyniki na wejście kolejnego z operatorów. Kolejność operatorów ustalana jest przez analizatory zapytań CQL. Aby móc przeprowadzać obliczenia w czasie rzeczywistym, zaproponowane rozwiązania umożliwiają jednoczesne wykonywanie procesów ładowania, uaktualniania i pobierania informacji z hurtowni danych.

Dzięki przetwarzaniu danych na bieżąco, użytkownicy mogą analizować zachodzące zjawiska w sposób, który do tej pory był niemożliwy do wykonania przy użyciu zwykłych baz danych.

Obecna implementacja ciągłych zapytań i strumieniowej bazy danych jest rozwiązaniem prototypowym. W dalszej pracy należy zoptymalizować metody wyliczania wartości, zwiększyć liczbę typów danych, co pozwoliłoby opisać większą grupę zjawisk rejestrowanych przez odbiorniki. W celu zwiększenia oszczędności zasobów pamięciowych można zmienić sposób zapisu danych, gdyż obecne rozwiązanie przechowuje nadmiarową ilość danych.

Literatura

Aho, A.V., Sethi, R., Ullman, J.D. (2002) Compilers: Principles, Techniques, and Tools. *Addison-Wesley, ISBN: 0201100886*

Gerwin, K. (2005) JFlex User's Manual. *pp.12-42*

Gorawski, M., Bańkowski, S. (2006) Software Agents System. *WKL, BDAS'06, ISBN 978-83-206-1611-5,pp. 139-147*

Gorawski, M., Gebczyk, W. (2006) Realization of Continuous Queries with kNN Join Processing in Spatial Telemetric Data Warehouse. *DEXA Workshops 2006: 632-636*

Kramer, J., Seeger, B. (2005) A Temporal Foundation for Continuous Queries over Data Streams. *COMAD 2005: pp.70-82*

Windowed Continuous Query Language

The paper presents the windowed CQL query language in the agent system for DSMS data streams management. In such system user can ask ad-hoc continuous queries in newly created CQL language. CQL language bases on the expanded and modified algebra, which describes operations on data streams. Data in streams is loaded into system on the fly, so its quantity or content is unknown. Received data from a stream are processed is archived or destroyed.