

Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Imię i nazwisko

Nr albumu: nralbumu

Intuicyjny język wyszukiwania TQL (Tablets Query Language)

**Praca magisterska
na kierunku INFORMATYKA**

Praca wykonana pod kierunkiem
dra Roberta Dąbrowskiego
Instytut Informatyki

czerwiec 2010

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora (autorów) pracy

Streszczenie

Sumerologia jest dziedziną badań nad antycznym językiem Sumerów, w której kluczowym zagadnieniem jest przeszukiwanie dużych zbiorów informacji zapisanych na odnalezionych tabliczkach sumeryjskich.

W pracy przedstawiono definicję przeznaczonego dla sumerologów intuicyjnego języka przeszukiwania zbiorów tabliczek (Tablets Query Language) wraz z jego przykładową implementacją opartą na relacyjnej bazie danych.

Celem tej pracy jest stworzenie języka zapytań intuicyjnego dla sumerologów, stanowiącego znaczące uproszczenie w stosunku do SQL dzięki wprowadzeniu pojęć naturalnych dla rozważanej dziedziny. Jednocześnie TQL nadal pozwala na tworzenie skomplikowanych zapytań wyszukiwujących, natomiast nie udostępnia funkcji tworzących i modyfikujących bazę. Można go rozszerzać i zmieniać tak, by mógł służyć też do innych zastosowań.

Słowa kluczowe

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

H. INFORMATION SYSTEMS
H.2. DATABASE MANAGEMENT
H.2.3 Languages

Tytuł pracy w języku angielskim

Intuitive query language TQL (Tablets Query Language)

Spis treści

Wprowadzenie	5
1. Podstawowe pojęcia	7
1.1. Definicje	7
2. Wcześniejsze rozwiązania	9
3. Dziedzina problemu	11
4. Definicja języka TQL	15
4.1. Gramatyka	15
4.1.1. Struktura leksykalna	15
4.1.2. Słowa kluczowe	15
4.1.3. Znaki specjalne	15
4.1.4. Komentarze	15
4.1.5. Struktura syntaktyczna języka	16
4.2. Semantyka	16
4.2.1. Zapytania proste	17
4.2.2. Zapytania złożone	17
4.2.3. Zapytanie zdefiniowane	18
4.2.4. Wywołanie zapytania zdefiniowanego	18
5. Implementacja	19
5.1. Moduły podstawowe	20
5.1.1. Parser	20
5.1.2. Analizator kontekstowy	21
5.1.3. Translator	21
5.1.4. Baza	21
5.1.5. Pliki pomocnicze	22
5.2. Moduły wymienne	22
5.2.1. Baza PostgreSQL	23
5.2.2. Baza XML	27
6. Podsumowanie	29
Bibliografia	31

Wprowadzenie

Sumerolodzy posiadają bazę danych składającą się z prawie 50 tys. tabliczek sumeryjskich w wersji elektronicznej. Potrzebują prostego i intuicyjnego języka służącego do ich wyszukiwania, który jak najmniej będzie ograniczał siłę wyrazu, a jego wykorzystanie będzie powodowało jak najmniejszy narzut czasowy.

Istnieją też inne grupy ludzi potrzebujące podobnego języka (np. językoznawcy). Większość programów ułatwiających tworzenie zapytań jest skomplikowana, daje ograniczone możliwości lub jest przystosowana głównie do przetwarzania danych liczbowych. Tablets Query Language rozwiązuje te problemy: jest prosty i intuicyjny, przystosowany głównie do tekstów, minimalnie zmniejsza siłę wyrazu oraz łatwo go rozbudowywać.

Język TQL jest nakładką na inne języki (m.in. SQL). Dla każdego z nich, w zależności od reprezentacji danych, należy skonstruować translator, którego zadaniem będzie przetłumaczenie zapytania. W ramach niniejszej pracy przedstawione zostaną dwa przykładowe translatory.

Rozdział 1

Podstawowe pojęcia

1.1. Definicje

Sumerolodzy - ludzie, którzy zajmują się odczytywaniem pisma klinowego w języku sumeryjskim. Na potrzeby tej pracy to pojęcie jest rozszerzone do wszystkich ludzi zajmujących się odczytywaniem tabliczek sumeryjskich i wyciąganiem z nich wiedzy historycznej.

Tabliczka - w tej pracy tabliczka będzie oznaczała tabliczkę sumeryjską w wersji elektronicznej (chyba, że zostanie zaznaczone inaczej). Dla rozróżnienia, kiedy będziemy mówić o “prawdziwej”, glinianej tabliczce, będziemy używać pojęcia **gliniana tabliczka**

Prowiniencja - pojęcie używane przez sumerologów, oznacza miejsce pochodzenia/znalezienia glinianej tabliczki

Kliny - znaki występujące na glinianych tabliczkach.

Odczyty - sposób transkrypcji klinów, występuje na tabliczkach elektronicznych.

Pieczęć - część tabliczki zawierająca znak rozpoznawczy autora

Rozdział 2

Wcześniejsze rozwiązania

W chwili obecnej nie ma czegoś takiego jak język dostosowany do potrzeb sumerologów. Są strony internetowe oferujące wyszukiwanie, jak np.

- **The Cuneiform Digital Library Initiative** (<http://cdli.ucla.edu>) - największa znana nam baza tekstów sumeryjskich, wyszukiwanie po praktycznie wszystkich możliwych parametrach, choć trochę mało wygodne. Brakuje wyjaśnienia jak używać “Advanced search syntax”
- **The Electronic Text Corpus of Sumerian Literature** (<http://etcsl.orinst.ox.ac.uk/>) - baza znacznie mniejsza, zawiera głównie teksty literackie. Wyszukiwanie mało rozbudowane.

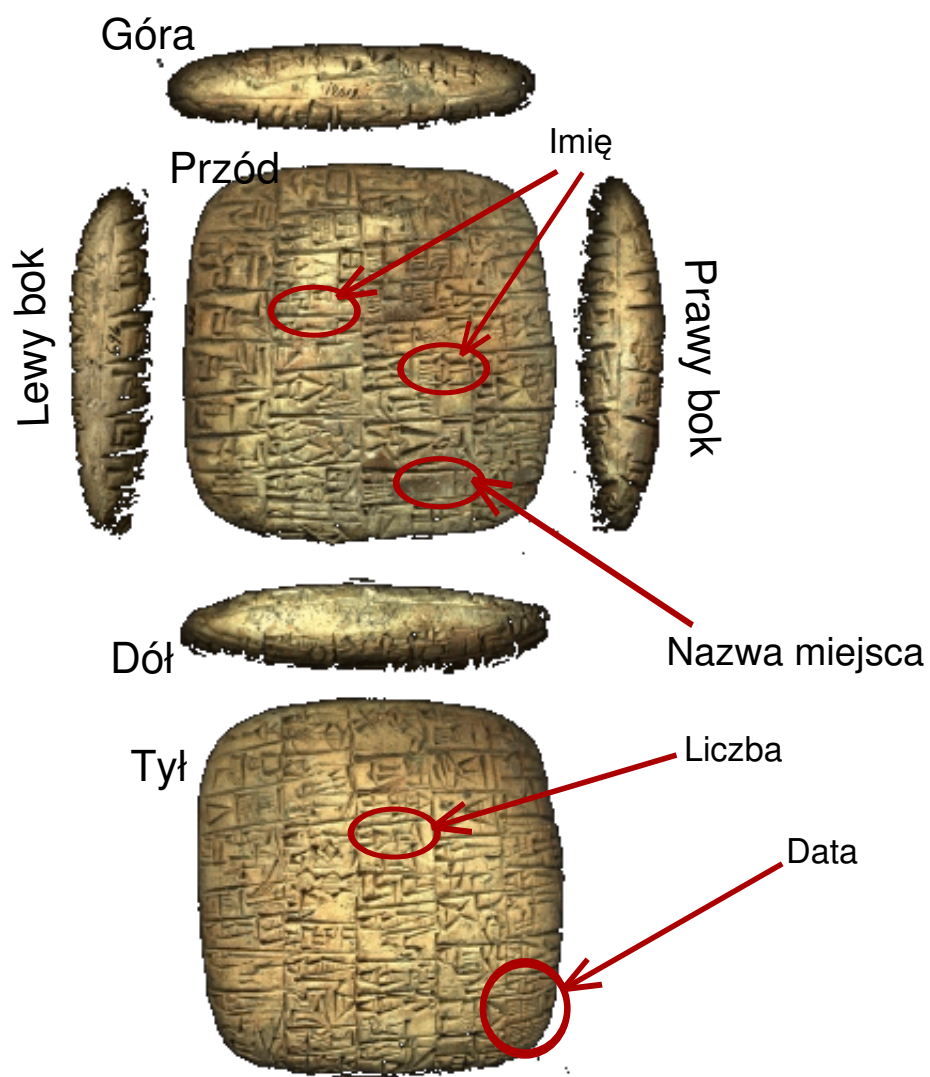
Rozdział 3

Dziedzina problemu

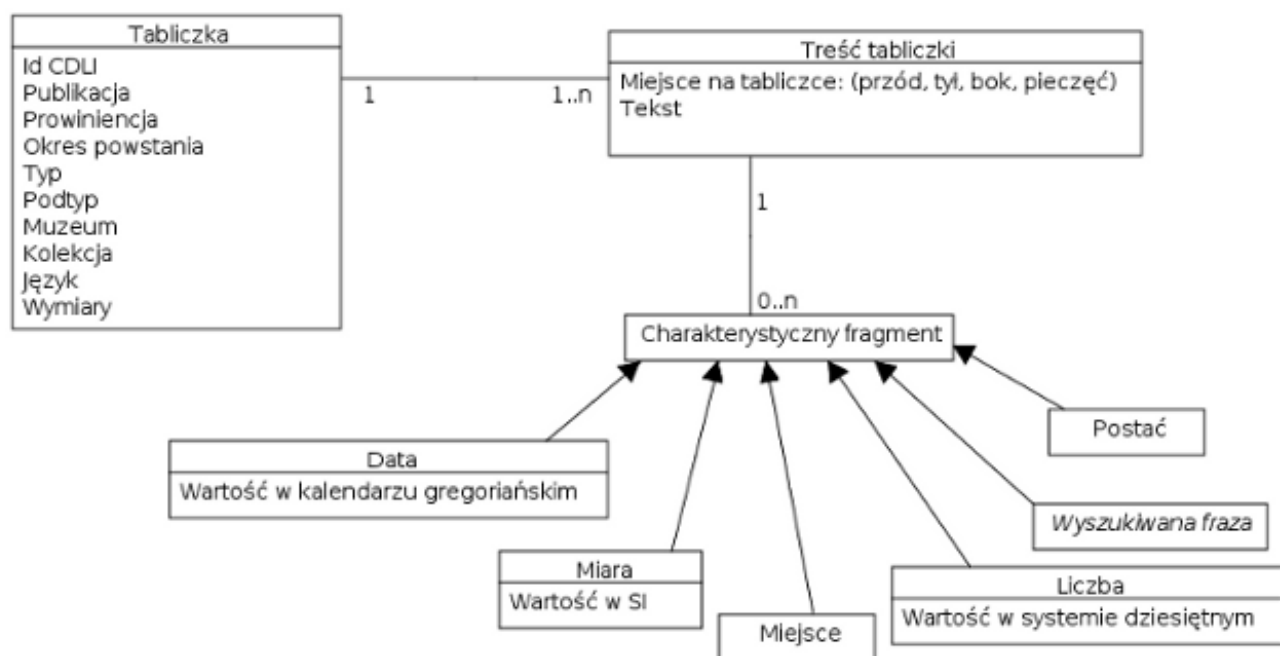
Głównym pojęciem jest tabliczka. Ma ona swoje metadane i treść. Tabliczka jest rozumiana dwojako - jako fizyczna tabliczka gliniana zapisana klinami lub jako tabliczka w formie cyfrowej zapisana odczytami. Może ona zawierać elementy znaczące takie jak imię jakiejś osoby, liczba, jednostka (np. przy opisywaniu wypłat), miejsce, data, imię bóstwa. Część tych elementów da się przetłumaczyć na współczesny język (np. jednostki przeliczyć na SI, datę na datę liczbową BC). Gliniane tabliczki są zapisywane z różnych stron (od góry, z przodu, z tyłu itp). Poza tym zawierają pieczęcie.

Sumerolodzy rozpoznają tabliczki po publikacjach - wiedzą mniej więcej o co chodzi jak widzą publikację.

Odczyty zawarte w cyfrowym zapisie tabliczki są wariantem tłumaczenia z klinów. W cyfrowej wersji nie ma klinów, stąd też możliwe są pomyłki w tłumaczeniach, które ciężko zweryfikować. Są też uszkodzone fragmenty, które zostały cyfrowo zapisane w najróżniejszej formie.



Rysunek 3.1: Gliniana tabliczka - struktura



Rysunek 3.2: Co powinna zawierać tabliczka w formie elektronicznej

Rozdział 4

Definicja języka TQL

4.1. Gramatyka

4.1.1. Struktura leksykalna

String

Literał $\langle String \rangle$ jest ciągiem dowolnych znaków w cudzysłowie ("). Nie może zawierać jedynie znaków " " niepoprzedzonych " \ ".

Słowo Od Litery

Literał $\langle Słowo Od Litery \rangle$ to ciąg liter, cyfr oraz znaków "- ", "' ", "_ ", zaczynający się od litery, z wyjątkiem słów kluczowych.

Słowo Od Liczby

Literał $\langle Słowo Od Liczby \rangle$ to ciąg liter, cyfr oraz znaków "- ", "' ", "_ ", zaczynający się od cyfry.

4.1.2. Słowa kluczowe

```
as      define  in
search
```

4.1.3. Znaki specjalne

```
( )      +
/  --    *
:        \n (koniec linii)
```

4.1.4. Komentarze

W chwili obecnej język nie zawiera komentarzy.

4.1.5. Struktura syntaktyczna języka

Nieterminale są pomiędzy "⟨ a ⟩". Symbole "::=" (produkcja), "|" (lub) i "ε" (pusta reguła) należą do notacji BNF. Wszystkie pozostałe symbole to terminale.

$$\begin{aligned}
\langle \text{Zapytanie Złożone} \rangle &::= \langle \text{Lista Zapytań} \rangle \\
\langle \text{Zapytanie} \rangle &::= \langle \text{Lista Linii Zapytania} \rangle \langle \text{Lista Pustych Linii} \rangle \\
&| \text{define } \backslash n \langle \text{Zapytanie} \rangle \text{ as } \langle \text{Nazwa} \rangle \langle \text{Lista Pustych Linii} \rangle \\
&| \text{search } \backslash n \langle \text{Zapytanie} \rangle \text{ in } \langle \text{Nazwa} \rangle \langle \text{Lista Pustych Linii} \rangle \\
&| \langle \text{Lista Pustych Linii} \rangle \\
\langle \text{Linia Zapytania} \rangle &::= \langle \text{Identyfikator} \rangle : \langle \text{Wyrażenie} \rangle \\
\langle \text{Wyrażenie} \rangle &::= \langle \text{Wyrażenie} \rangle + \langle \text{Wyrażenie1} \rangle \\
&| \langle \text{Wyrażenie} \rangle / \langle \text{Wyrażenie1} \rangle \\
&| \langle \text{Wyrażenie1} \rangle \\
\langle \text{Wyrażenie1} \rangle &::= -- \langle \text{Wyrażenie1} \rangle \\
&| \langle \text{Wyrażenie2} \rangle \\
\langle \text{Wyrażenie2} \rangle &::= \langle \text{Tekst} \rangle * \langle \text{Tekst} \rangle \\
&| \langle \text{Tekst} \rangle * \\
&| * \langle \text{Tekst} \rangle \\
&| \langle \text{Tekst} \rangle \\
&| (\langle \text{Wyrażenie} \rangle) \\
\langle \text{Lista Zapytań} \rangle &::= \langle \text{Zapytanie} \rangle \\
&| \langle \text{Zapytanie} \rangle \langle \text{Lista Zapytań} \rangle \\
\langle \text{Lista Linii Zapytania} \rangle &::= \langle \text{Linia Zapytania} \rangle \backslash n \\
&| \langle \text{Linia Zapytania} \rangle \backslash n \langle \text{Lista Linii Zapytania} \rangle \\
\langle \text{Pusta Linia} \rangle &::= \backslash n \\
\langle \text{Lista Pustych Linii} \rangle &::= \epsilon \\
&| \langle \text{Pusta Linia} \rangle \langle \text{Lista Pustych Linii} \rangle \\
\langle \text{Tekst} \rangle &::= \langle \text{String} \rangle \\
&| \langle \text{Słowo} \rangle \\
\langle \text{Słowo} \rangle &::= \langle \text{Słowo Od Litery} \rangle \\
&| \langle \text{Słowo Od Liczby} \rangle \\
\langle \text{Identyfikator} \rangle &::= \langle \text{Słowo Od Litery} \rangle \\
\langle \text{Nazwa} \rangle &::= \langle \text{String} \rangle
\end{aligned}$$

4.2. Semantyka

Język TQL umożliwia wyszukiwanie na podstawie następujących danych:

Opis	Nazwa pola w TQL
numer tabliczki w bazie CDLI	cdli_id
miejsce pochodzenia (proweniencja)	provenience

Opis	Nazwa pola w TQL
okres powstania	period
typ i podtyp	genre
rok powstania	year
publikacja	publication
treść	text
kolekcja	collection
muzeum	museum

Język można łatwo rozszerzać, aby umożliwić pytanie o inne dane (np. wyszukiwać po klinach czy po zawartości pieczęci).

Poniżej przedstawiamy semantykę wybranych przykładów.

4.2.1. Zapytania proste

```
provenience: Gar*
period: "Ur III"
genre: Administrative
text: udu + (masz2/ugula) --szabra
```

Wynikiem zapytania będą wszystkie tabliczki, które:

- pochodzą z miejscowości o nazwie zaczynającej się na “Gar”
- pochodzą z okresu Ur III
- są dokumentami administracyjnymi
- zawierają słowo “udu” oraz conajmniej jedno ze słów “masz2” lub “ugula”
- nie zawierają słowa “szabra”

4.2.2. Zapytania złożone

```
provenience: Ur
period: "Ur III"/"Ur IV"
text: udu --szabra
```

```
text: masz2/ugula
publication: *tan
provenience: Ur
```

Wynikiem zapytania będą wszystkie tabliczki, które:

- pochodzą z miejscowości Ur
- pochodzą z okresu Ur III lub Ur IV
- zawierają słowo “udu”
- nie zawierają słowa “szabra”

oraz wszystkie tabliczki, które:

- zawierają słowo "masz2" lub "ugula"
- zostały opublikowane w pracy, której nazwa kończy się na "tan"
- pochodzą z miejscowości Ur

4.2.3. Zapytanie zdefiniowane

```
define
  provenience: Gar*a
  period: Ur III
  text: "udu ban"/mash2
as "zwierzęta w Gar*a"
```

Wynikiem zapytania (po jego wywołaniu) będą wszystkie tabliczki, które:

- pochodzą z miejscowości, których nazwy zaczynają się na "Gar" i kończą na "a"
- pochodzą z okresu Ur III
- zawierają co najmniej jedną z fraz "udu ban" lub "mash2"

4.2.4. Wywołanie zapytania zdefiniowanego

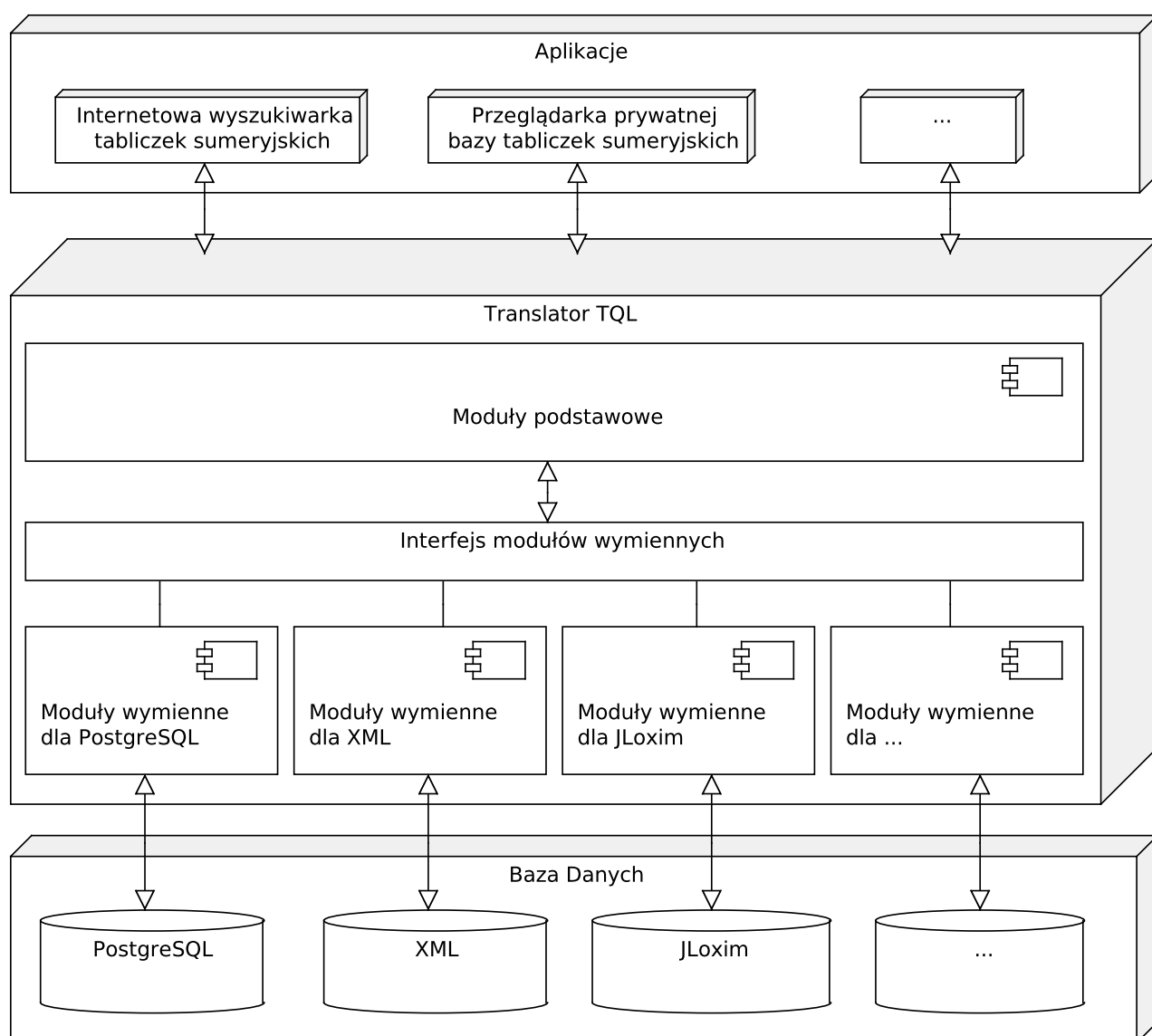
```
search
  text: adad-tilati
in "zwierzęta w Gar*a"
```

Wynikiem zapytania będą wszystkie tabliczki, które:

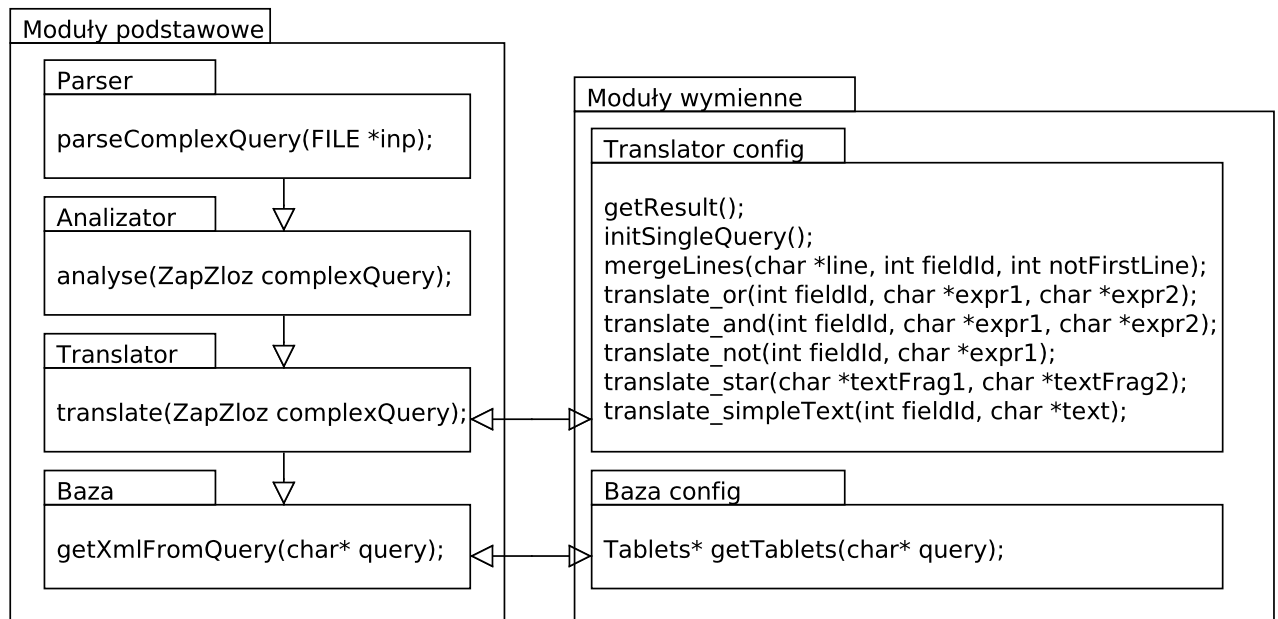
- spełniają wszystkie warunki zapytania "zwierzęta w Gar*a"
- zawierają słowo "adad-tilati"

Rozdział 5

Implementacja



Rysunek 5.1: Struktura systemu korzystającego z translatora



Rysunek 5.2: Podział programu na moduły

Jednym z głównych założeń jakie miał spełniać język TQL jest niezależność od struktury danych. Wynikiem tego jest podział translatora na 2 rodzaje modułów:

1. **Podstawowe**, które są niezależne od struktury danych i zajmują się głównie parsowaniem i analizą składniową zapytania.
2. **Wymienne**, które zależą od struktury danych, tłumaczą zapytanie TQL na język odpowiedni dla używanej bazy danych i wywołują je.

Wybór modułu wymiennego odbywa się na poziomie kompilacji.

5.1. Moduły podstawowe

5.1.1. Parser

Parser został utworzony za pomocą narzędzia BNFC. Następnie zostały w nim wprowadzone modyfikacje:

- poprawienie nazw stałych oznaczających symbole na bardziej intuicyjne,
- dodanie tablicy symboli,
- usunięcie niepotrzebnych funkcji z interfejsu,
- uporządkowanie kodu.

Moduł ten parsuje zapytanie w języku TQL, tworząc drzewo struktury składniowej, które jest zdefiniowane w pliku pomocniczym Absyn.h. Na parser składają się następujące pliki:

- Parser.c

- Parser.h
- TQL.y
- TQL.l

5.1.2. Analizator kontekstowy

Moduł analizuje drzewo struktury składniowej w następujący sposób:

- sprawdza, czy podano prawidłowe nazwy pól,
- upraszcza drzewo - z wywołania zapytania (wywołanie search in) tworzy zapytanie proste.

Składa się z następujących plików:

- Context.c
- Context.h

5.1.3. Translator

Zadaniem translatora jest przetłumaczenie drzewa składni abstrakcyjnej na zapytanie w docelowym języku. Składa się z następujących plików:

- Translator.c
- Translator.h
- Translator_config.h (interfejs modułu translatora zależnego od bazy danych)

Tłumaczenie poszczególnych elementów drzewa zależy od implementacji interfejsu zawartego w pliku Translator_config.h. Funkcja translate() przechodzi całą strukturę drzewa, wywołując w razie potrzeby odpowiednie funkcje z Translator_config. Następnie pobiera przetłumaczone zapytanie za pomocą funkcji getResult(), aby przekazać je do modułu bazy.

5.1.4. Baza

Moduł bazy jest odpowiedzialny za wywołanie przetłumaczonego zapytania i przekazanie wyniku w określonej formie - jako XML. Składa się z następujących plików:

- Database.c
- Database.h
- Database_config.h (interfejs modułu bazy zależnego od bazy danych)

Wywołuje funkcję getTablets() z Database_config.h, jako parametr podając przetłumaczoną treść zapytania. Funkcja ta zwraca strukturę danych Tablets, wypełnioną informacjami o wyszukanych tabliczkach. Następnie na podstawie otrzymanej struktury tworzony jest dokument XML.

Definicja struktury Tablets:

```

typedef struct{
    char* id;
    char* id_cdli;
    char* publication;
    char* measurements;
    char* year;
    char* provenience;
    char* period;
    char* genre;
    char* subgenre;
    char* collection;
    char* text;
    Tags *tags; // specjalnie oznaczone miejsca w tekście (w pierwszej wersji frazy wyszukiw
} Tablet;

typedef struct{
    int size;
    Tablet *tabs;
} Tablets;

```

Wszystkie niezbędne informacje powinny się znajdować w bazie danych.

5.1.5. Pliki pomocnicze

Definicje struktur danych (wygenerowane za pomocą BNFC, następnie uproszczone):

- Absyn.c
- Absyn.h

Tablica symboli:

- Symbols.c
- Symbols.h

Obsługa błędów:

- Err.c
- Err.h

Moduł do dzielenia tekstu względem separatora, pobrany z internetu [2]:

- Cexplode.c
- Cexplode.h

5.2. Moduły wymienne

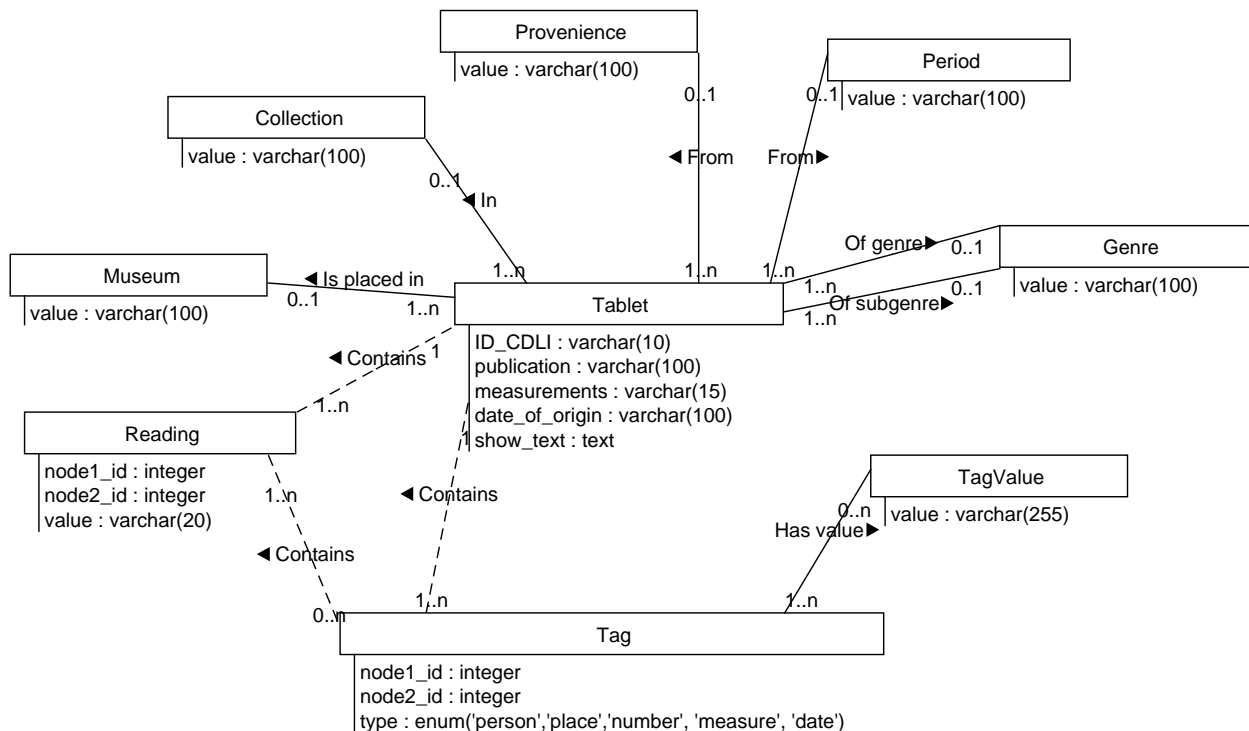
Pliki zależne od wyboru konkretnej bazy danych to:

- Translator_config.c - dla modułu translatora
- Database_config.c - dla modułu bazy

Ich interfejsy są wspólne dla wszystkich baz danych.

5.2.1. Baza PostgreSQL

Diagram encji



Rysunek 5.3: Diagram encji

Sposób przechowywania treści tabliczki był problematyczny: trzeba było zapewnić szybkość wyszukiwania po treści tabliczki biorąc pod uwagę wieloznaczność odczytów, jednocześnie szybkość wyciągania konkretnej tabliczki - całej treści. Przez to treść tabliczki jest przechowywana podwójnie: z powodu 1 formie grafu, którego krawędziami są odczyty i tagi (zgodnie z pomysłem dr Wojciecha Jaworskiego[1, s.13-24]) w tabelach Reading i Tag, z powodu 2: jako otwarty tekst (kolumna w tabeli Tablet). Aby zapewnić połączenie pomiędzy tymi sposobami, węzły są liczbami postaci:

`<numer węzła w tabliczce> * 1 000 000 + <id tabliczki>`

gdzie numer węzła w tabliczce to numer kolejnego słowa (słowa są oddzielone spacjami i końcem linii) pomnożone przez 10 (żeby umożliwić wstawienie kilku węzłów w jednym słowie np. pozwolić na przetłumaczenie jednego słowa na sekwencję trzech klinów).

Przerywane linie na diagramie encji oznaczają, że klucz obcy do tabeli Tablet nie występuje wprost w tabeli Reading, jednak nie jest trudno go odczytać ;p

Translator_config

Dostaje poszczególne fragmenty drzewa struktury zapytania i tłumaczy je na SQL. Przetłumaczone fragmenty zbiera do buforów (select, from, where), które następnie odpowiednio skleja. Każde proste zapytanie jest tłumaczone na jednego selecta; jak jest kilka prostych zapytań to są sklepane UNION.

Inicjalizacja zapytania

Tłumaczenie prostego zapytania zaczyna się od inicjalizacji buforów przechowujących poszczególne części wynikowego SQL-a. Select jest inicjowany na

```
SELECT t.id, t.id_cdli, t.publication, t.measurements, t.origin_date,  
       p.value as provenience, pd.value as period,  
       g1.value as genre, g2.value as subgenre,  
       c.value as collection, t.text
```

From jest inicjowany

```
FROM tablet t  
     LEFT JOIN provenience p ON p.id=t.provenience_id  
     LEFT JOIN collection c ON c.id=t.collection_id  
     LEFT JOIN genre g1 ON g1.id=t.genre_id  
     LEFT JOIN genre g2 ON g2.id = t.subgenre_id  
     LEFT JOIN period pd ON pd.id = t.period_id
```

Where jest inicjowany na

WHERE

Tłumaczenie konstrukcji prostych

Poniższe tłumaczenia są dodawane do klauzuli "where" i łączone "AND".

Konstrukcja	Tłumaczenie na SQL
provenience: wartosc	p.value LIKE 'wartosc'
publication: wartosc	t.publication LIKE 'wartosc'
period: wartosc	pd.value LIKE 'wartosc'
year: wartosc	t.origin_date LIKE 'wartosc'
genre: wartosc	g1.value LIKE 'wartosc' OR g2.value LIKE 'wartosc'
cdli.id: wartosc	t.cdli_id LIKE 'wartosc'

Konstrukcja	Tłumaczenie na SQL
museum: wartosc	t.museum LIKE 'wartosc'
collection: wartosc	c.value LIKE 'wartosc'

Tłumaczenie operatorów:

Operator	Tłumaczenie
/	OR
-	NOT
+	AND
*	%

Tłumaczenie konstrukcji złożonych

Została zaimplementowana tylko jedna konstrukcja złożona - przy zapytaniu o treść tabliczki (pole "text"). Korzystamy przy tym zapytaniu z przedstawienia treści tabliczki w formie grafu. Krawędziami grafu są słowa, jedyną funkcją węzłów jest zachowanie kolejności. Graf jest w tabeli readings. Id węzłów składają się z numeru tabliczki i kolejnego numeru węzła tabliczki (nr_wezla * 1 000 000 + id_tabliczki). Uznajemy, że słowa są oddzielone spacjami.

Pojawienie się wyszukiwania po treści tabliczki niesie za sobą konieczność dodania do klauzuli "from"

```
INNER JOIN (
  <wynikowe zapytanie o treść tabliczki>
) AS sequence ON sequence.id_tab = t.id
```

Natomiast do select dodajemy:

```
, sequence.nodes as nodes
```

Gdzie <wynikowe zapytanie o treść tabliczki> to kombinacja zapytań typu:

```
SELECT
  id_tab,
  CAST(array_accum(nodes) as TEXT) as nodes,
  COUNT(DISTINCT id_seq) AS seq,
  <id_seq> AS id_seq
FROM (
  SELECT
    r1.node1_id % 1000000 AS id_tab,
    '{' || r1.node1_id || ',' || r1.node2_id || '}' AS nodes,
    1 AS id_seq
  FROM
    readings r1
    LEFT JOIN readings r2 ON (r2.node1 = r1.node2)
```

```

LEFT JOIN readings r3 ON (r3.node1 = r2.node2)
...
LEFT JOIN readings r<dl_sekw> ON (r<dl_sekw>.node1 = r<dl_sekw-1>.node2)
WHERE
  r1.value LIKE '<sekw[1]>'
AND
  r2.value LIKE '<sekw[2]>'
AND
  r3.value LIKE '<sekw[3]>'
AND
  ...
AND
  r<dl_sekw>.value LIKE '<sekw[<dl_sekw>]>'
) AS a
GROUP BY id_tab

```

Zmienne użyte w powyższym pseudo-kodzie:

id_sekw - kolejny numer sekwencji (przydatny przy bardziej skomplikowanym zapytaniu - do ich rozróżniania)

dl_sekw - ilość słów składających się na wyszukiwaną sekwencję

sekw - tablica zawierająca słowa składające się na wyszukiwaną sekwencję

Operator	Tłumaczenie
/	<pre> SELECT id_tab, CAST(array_accum(nodes) as TEXT) as nodes, COUNT(DISTINCT id_seq) as seq, <id_sekw> as id_seq FROM (<zapytanie1> UNION <zapytanie2>) as c GROUP BY id_tab </pre>

Operator	Tłumaczenie
+	<pre> SELECT * FROM (SELECT id_tab, CAST(array_accum(nodes) as TEXT) as nodes, COUNT(DISTINCT id_seq) as seq, <id_sekw> as id_seq FROM (<zapytanie1> UNION <zapytanie2>) as c GROUP BY id_tab) as b WHERE b.seq=2 </pre>
—	<pre> SELECT id_tab, '' as wezly, 0 as sekw, <id_sekw> as id_sekw FROM ((SELECT id as id_tab from tabliczka) EXCEPT (SELECT id_tab from <zapytanie_negowane> as a)) as b </pre>
*	%

Database_config

Odpowiada za wywołanie zapytania na konkretnej bazie. Korzysta z pliku database.conf, który zawiera dane dostępu do bazy. Korzysta z biblioteki libpq-fe.h do postgresa. Zwrócony wynik zapisuje do struktury Tablets.

5.2.2. Baza XML

Rozdział 6

Podsumowanie

Bibliografia

- [1] Wojciech Jaworski, *Modelowanie treści sumeryjskich tekstów gospodarczych z epoki Ur III*, <http://nlp.ipipan.waw.pl/NLP-SEMINAR/071119.pdf>, 19 listopada 2007
- [2] <http://maz-programmersdiary.blogspot.com/2008/09/c-explode.html>

Spis rysunków

3.1. Gliniana tabliczka - struktura	12
3.2. Co powinna zawierać tabliczka w formie elektronicznej	13
5.1. Struktura systemu korzystającego z translatora	19
5.2. Podział programu na moduły	20
5.3. Diagram encji	23