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 wyszukujących, 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

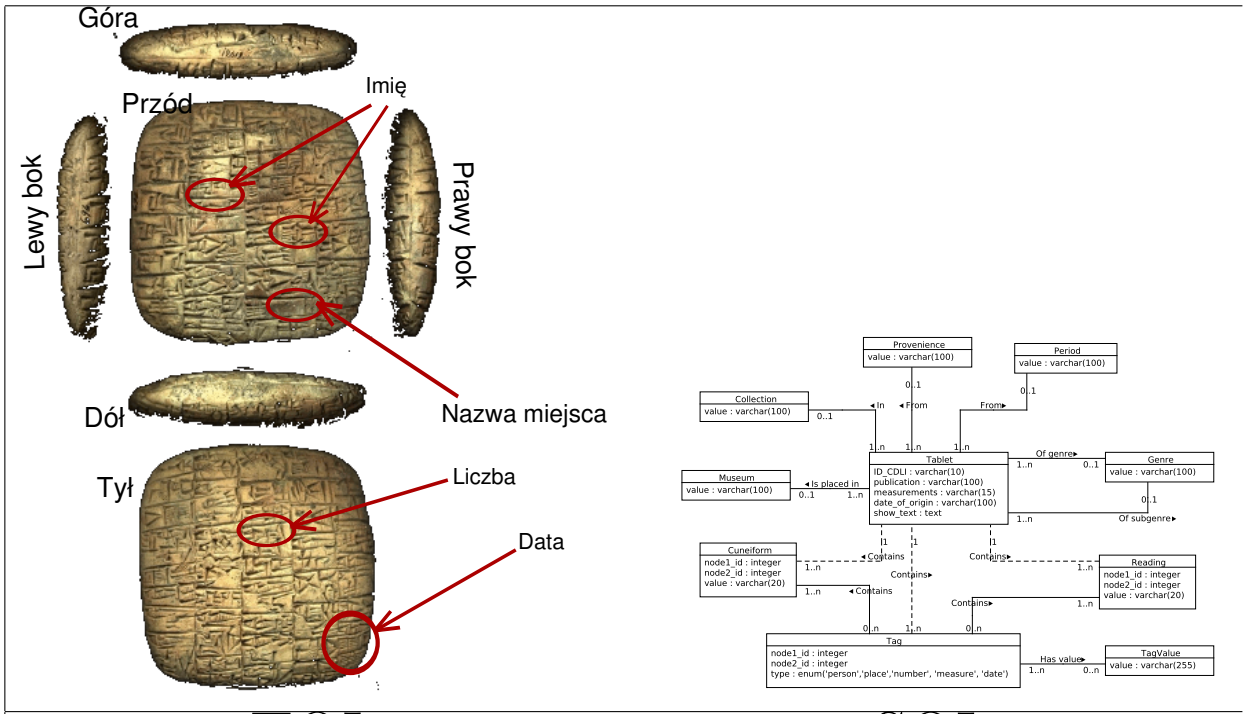
H. INFORMATION SYSTEMS
H.2. DATABASE MANAGEMENT
H.2.3 Languages

Tytuł pracy w języku angielskim

Intuitive query language TQL (Tablets Query Language)

Spis treści

Wprowadzenie



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

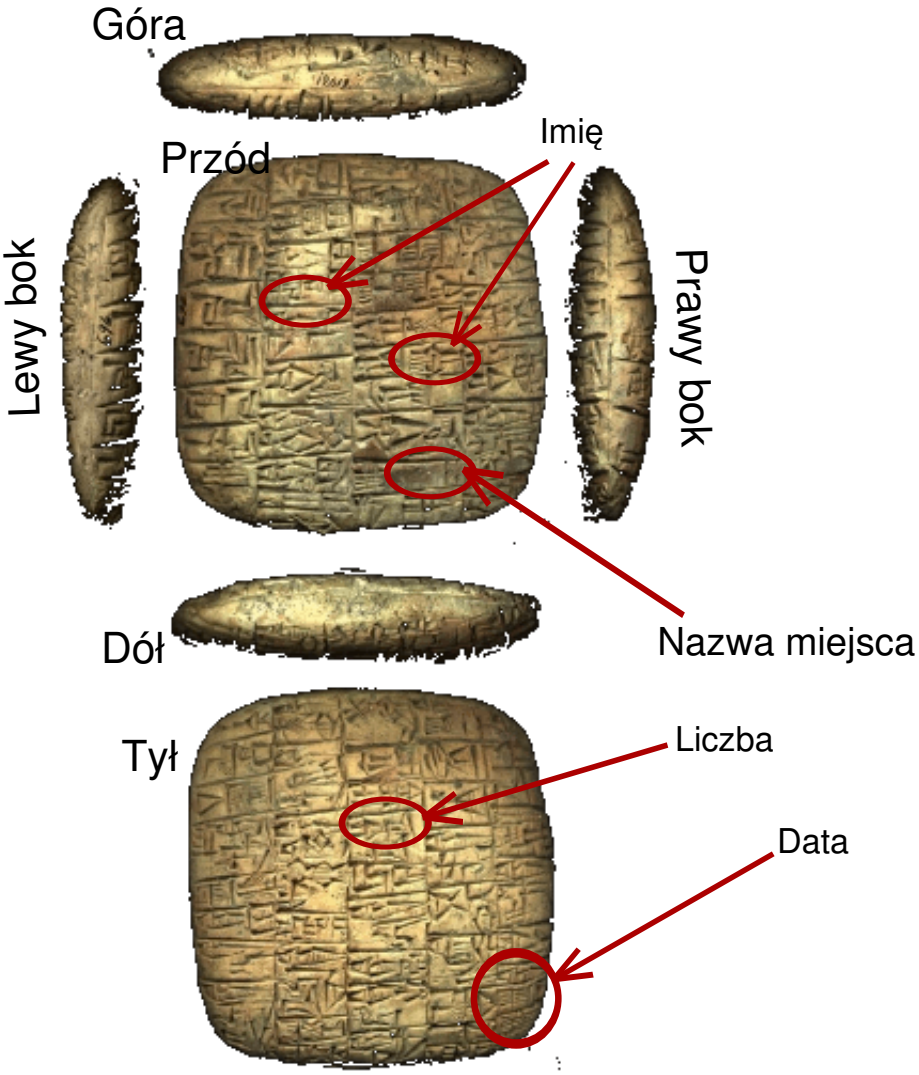
Dziedzina problemu

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.

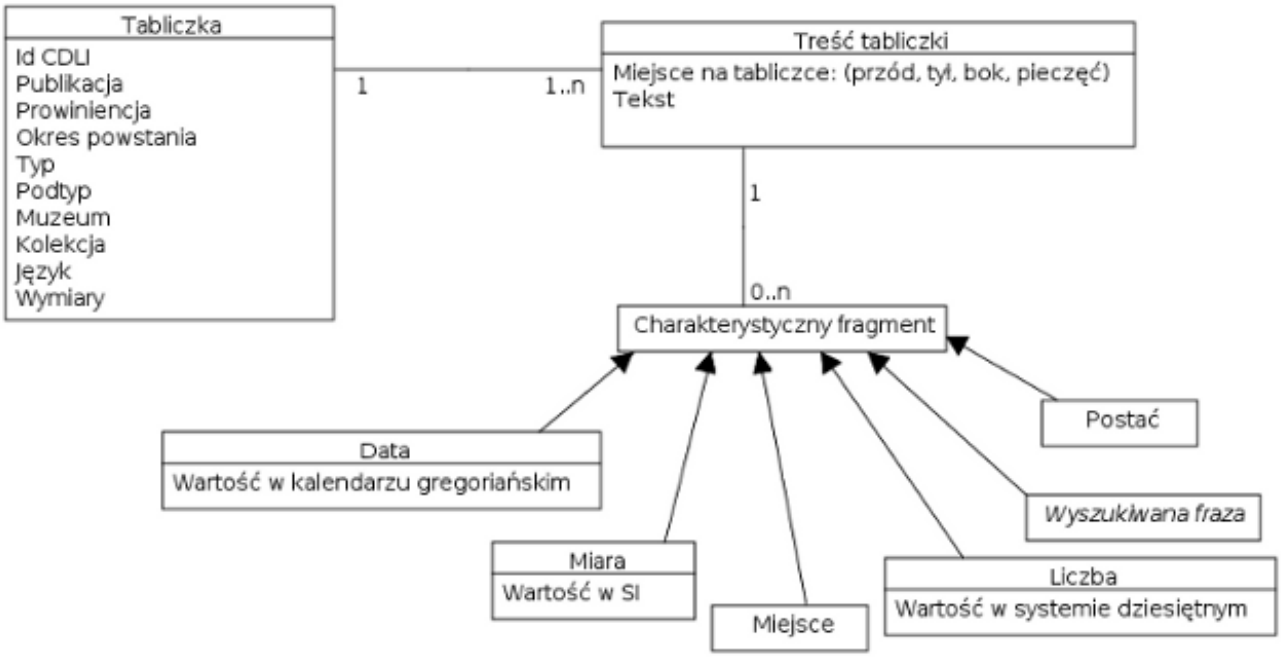
Sumerolodzy rozpoznają tabliczki po publikacjach - wiedzą mniej więcej o co chodzi jak widzą publikację.

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.

Sumerolodzy oczekują możliwości wyszukiwania po metadanych, po treści tabliczki (odczyty) i po możliwych innych tłumaczeniach (po klinach). W pierwszej wersji języka implementujemy tylko wyszukiwanie po odczytach.



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 ⟨*String*⟩ jest ciągiem dowolnych znaków w cudzysłowiu (`"`). Nie może zawierać jedynie znaków “`"`“ niepoprzedzonych ”`\`“.

Słowo Od Litery

Literał ⟨*Słowo Od Litery*⟩ to ciąg liter, cyfr oraz znaków ”`-`“, ”`'`“, ”`_`“, zaczynający się od litery, z wyjątkiem słów kluczowych.

Słowo Od Liczby

Literał ⟨*Słowo Od Liczby*⟩ 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.

⟨*Zapytanie Złożone*⟩ ::= ⟨*Lista Zapytań*⟩

⟨*Zapytanie*⟩ ::= ⟨*Lista Linii Zapytania*⟩ ⟨*Lista Pustych Linii*⟩
| `define \n` ⟨*Zapytanie*⟩ `as` ⟨*Nazwa*⟩ ⟨*Lista Pustych Linii*⟩
| `search \n` ⟨*Zapytanie*⟩ `in` ⟨*Nazwa*⟩ ⟨*Lista Pustych Linii*⟩
| ⟨*Lista Pustych Linii*⟩

⟨*Linia Zapytania*⟩ ::= ⟨*Identyfikator*⟩ `:` ⟨*Wyrażenie*⟩

⟨*Wyrażenie*⟩ ::= ⟨*Wyrażenie*⟩ + ⟨*Wyrażenie1*⟩
| ⟨*Wyrażenie*⟩ `/` ⟨*Wyrażenie1*⟩
| ⟨*Wyrażenie1*⟩

⟨*Wyrażenie1*⟩ ::= −− ⟨*Wyrażenie1*⟩
| ⟨*Wyrażenie2*⟩

⟨*Wyrażenie2*⟩ ::= ⟨*Tekst*⟩ `*` ⟨*Tekst*⟩
| ⟨*Tekst*⟩ `*`
| `*` ⟨*Tekst*⟩
| ⟨*Tekst*⟩
| `(` ⟨*Wyrażenie*⟩ `)`

⟨*Lista Zapytań*⟩ ::= ⟨*Zapytanie*⟩
| ⟨*Zapytanie*⟩ ⟨*Lista Zapytań*⟩

⟨*Lista Linii Zapytania*⟩ ::= ⟨*Linia Zapytania*⟩ `\n`
| ⟨*Linia Zapytania*⟩ `\n` ⟨*Lista Linii Zapytania*⟩

⟨*Pusta Linia*⟩ ::= `\n`

⟨*Lista Pustych Linii*⟩ ::= ϵ
| ⟨*Pusta Linia*⟩ ⟨*Lista Pustych Linii*⟩

⟨*Tekst*⟩ ::= ⟨*String*⟩
| ⟨*Słowo*⟩

⟨*Słowo*⟩ ::= ⟨*Słowo Od Litery*⟩
| ⟨*Słowo Od Liczby*⟩

⟨*Identyfikator*⟩ ::= ⟨*Słowo Od Litery*⟩

⟨*Nazwa*⟩ ::= ⟨*String*⟩

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 (po odczytach)	text
treść (po klinach)	cunetext
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 conajmniej jedną z fraz "udu ban" lub "mash2"

4.2.4. Wywołanie zapytania zdefiniowanego

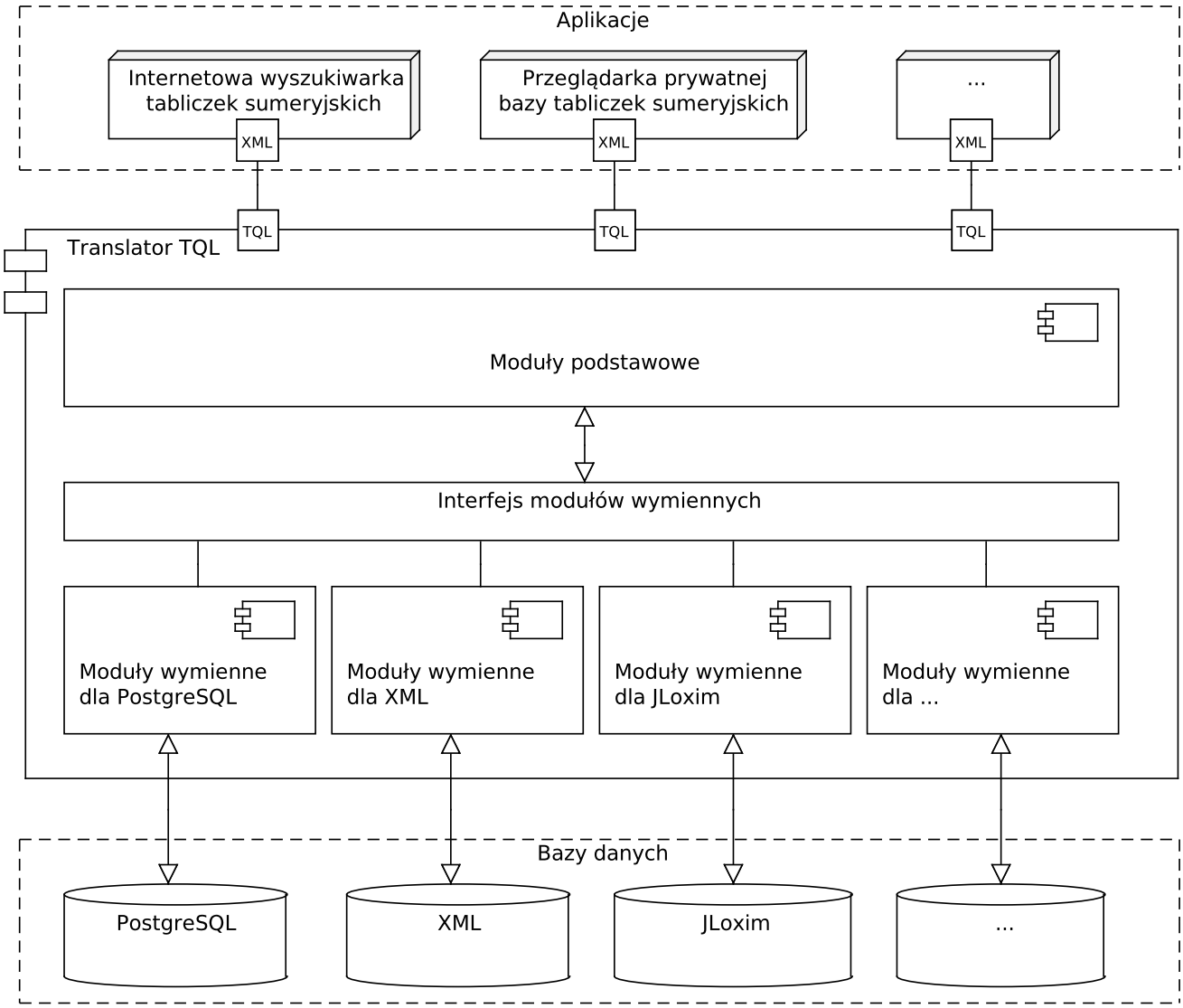
```
search
  text: adad-tilati
in "zwierzęta w Gar*a"
```

Wynikiem zapytania będą wszystkie tabliczki, które:

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

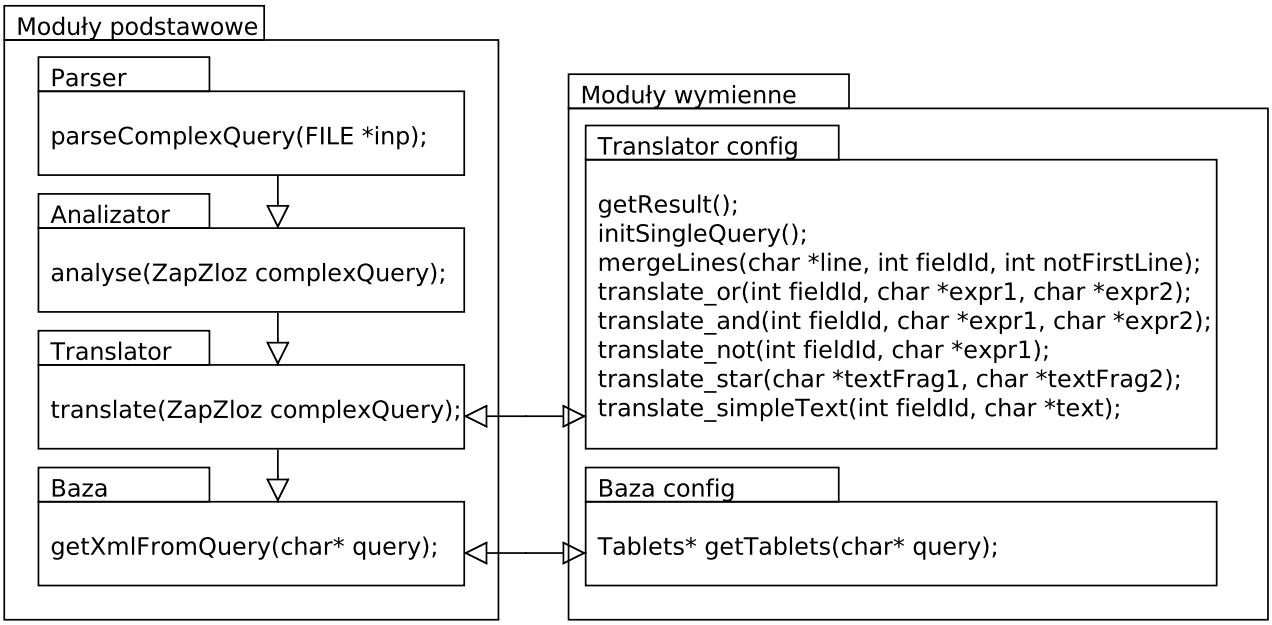
Rozdział 5

Implementacja



Rysunek 5.1: Struktura systemu korzystającego z translatora

Jednym z głównych założeń języka TQL jest niezależność od struktury danych. W związku



Rysunek 5.2: Podział programu na moduły

z tym istotną cechą translatora jest możliwość dostosowania do współpracy z różnymi bazami danych. Wynikiem tego jest podział translatora na 2 rodzaje modułów:

1. **Podstawowe** – niezależne od struktury danych i zajmujące się głównie parsowaniem i analizą składniową zapytania.
2. **Wymienne** – zależne od struktury danych, tłumaczące zapytanie TQL na język odpowiedni dla używanej bazy danych i wywołujące 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 tekscie
                // (w pierwszej wersji frazy wyszukiwania)
} Tablet;

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

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

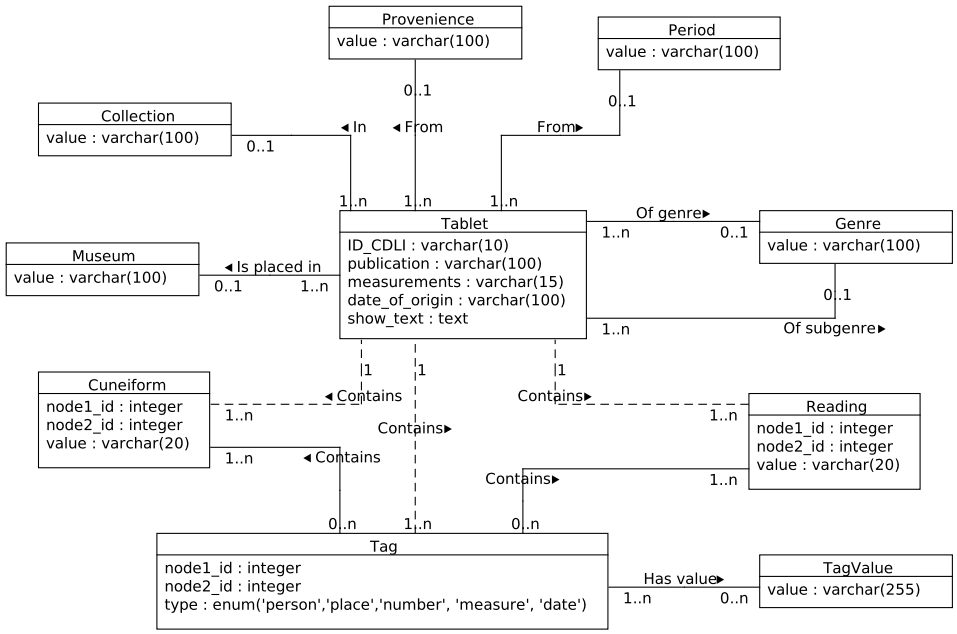
Pliki zależne od wyboru konkretnej bazy danych to:

- 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

Jednym z problemów przy projektowaniu bazy danych był wybór takiej reprezentacji treści tabliczki, żeby efektywnie wykonywać następujące operacje:

- wyszukiwanie po treści tabliczki (po odczytach, klinach i tagach),
- wyszukiwanie treści konkretnej tabliczki.

Najlepszym rozwiązaniem pierwszego problemu jest reprezentacja treści tabliczki w formie grafu, którego krawędziami są odczyty, kliny i tagi (zgodnie z pomysłem dr Wojciecha Jaworskiego[1, s.13-24]). Natomiast w przypadku drugiego problemu narzuca się przechowywanie treści jako otwarty tekst. Zdecydowałyśmy się na połączenie obu sposobów. Odczyty, kliny i tagi przechowujemy w tabelach Reading, Cuneiform i Tag, natomiast otwarty tekst w kolumnie show_text tabeli Tablet. Aby zapewnić możliwość odwzorowania treści tabliczki między reprezentacjami, 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 opisany powyżej związek pomiędzy id węzła (node1_id i node2_id) a id tabliczki (Tablet.id).

Translator_config

Tłumaczy otrzymane fragmenty drzewa struktury zapytania na język SQL. Przetłumaczone fragmenty zbiera do buforów (*select*, *from*, *where*), które następnie odpowiednio łączy. Każde proste zapytanie TQL jest tłumaczone na pojedyncze zapytanie SQL. Tłumaczenie kilku prostych zapytań łączone jest za pomocą 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 początkowo zawiera pusty ciąg znaków.

Tłumaczenie konstrukcji prostych

Poniższe tłumaczenia są dodawane do bufora *where* i łączone za pomocą 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'`

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

Tłumaczenie operatorów:

Operator	Tłumaczenie
/	OR
–	NOT
+	AND
*	%

Tłumaczenie konstrukcji złożonych

Przy tłumaczeniu konstrukcji złożonych korzystamy z przedstawienia treści tabliczki w formie grafu.

Pojawienie się wyszukiwania po treści tabliczki niesie za sobą konieczność dodania do bufora *from*:

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

natomiast do *select* dodajemy:

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

gdzie <wynikowe zapytanie o treść tablczki> 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
    t1.node1_id % 1000000 AS id_tab,
    '{' || t1.node1_id || ',' || t<dl_sekw>.node2_id || '}' AS nodes,
    1 AS id_seq
  FROM
    <nazwa_tabeli> t1
    LEFT JOIN <nazwa_tabeli> t2 ON (t2.node1 = t1.node2)
```

```
    LEFT JOIN <nazwa_tabeli> t3 ON (t3.node1 = t2.node2)
    ...
    LEFT JOIN <nazwa_tabeli> t<dl_sekw> ON (t<dl_sekw>.node1 = t<dl_sekw-1>.node2)
 WHERE
    t1.value LIKE '<sekw[1]>'
   AND
    t2.value LIKE '<sekw[2]>'
  AND
    t3.value LIKE '<sekw[3]>'
  AND
    ...
  AND
    t<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 rozróżniania podzapytań)

dl_sekw - ilość słów składających się na wyszukiwaną sekwencję

sekw - tablica zawierająca słowa składające się na wyszukiwaną sekwencję

nazwa_tabeli - nazwa tabeli, w której wyszukujemy (Reading lub Cuneiform)

Operator	**Tłumaczenie**

/	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

Operator	Tłumaczenie
+	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
−	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
*	%

Database_config

Odpowiada za wywołanie zapytania na konkretnej bazie i zapisanie wyniku do struktury Tablets. Korzysta z pliku database.conf, który zawiera dane dostępu do bazy (host, port, użytkownik, hasło, nazwa bazy) oraz biblioteki libpq-fe.h do PostgreSQL.

5.2.2. Baza XML

W budowie :)

Rozdział 6

Podsumowanie

Bibliografia

[1] Wojciech Jaworski, *Modelowanie tresci 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