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

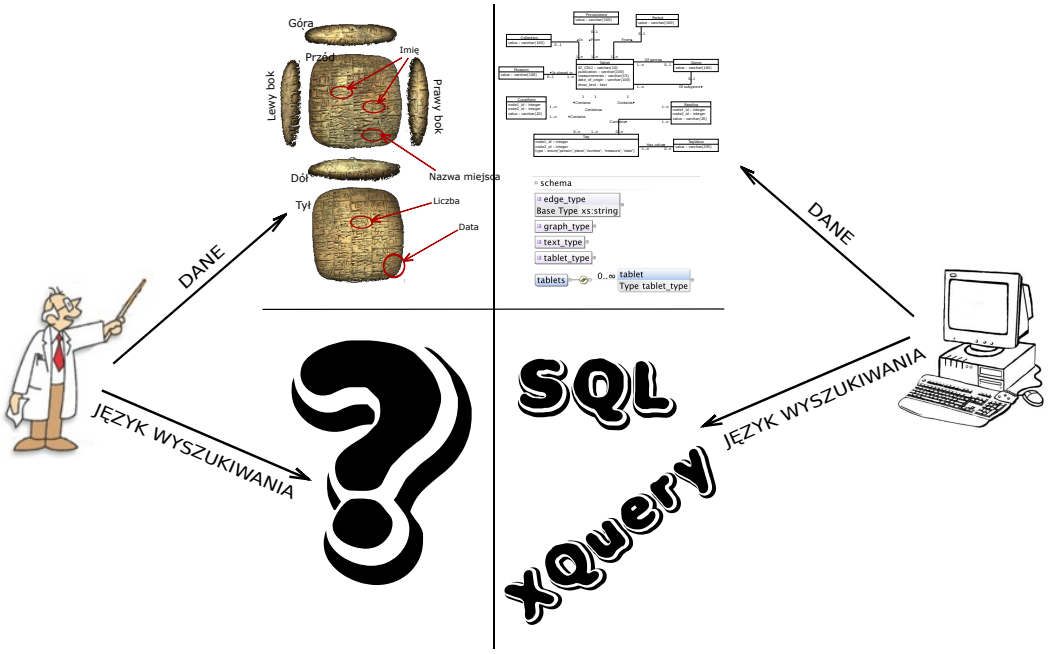
H. INFORMATION SYSTEMS
H.2. DATABASE MANAGEMENT
H.2.3 Languages

Tytuł pracy w języku angielskim

Intuitive query language TQL (Tablets Query Language)

Spis treści

Wprowadzenie



Rysunek 1: Przedstawienie problemu

Niniejsza praca dotyczy problemu przetwarzania baz danych tabliczek sumeryjskich przez osoby nieznające specyficznych dla baz danych języków zapytań.

Istnieje wiele baz danych zawierających teksty odczytane z tabliczek sumeryjskich (najbardziej znana - CDLI zawiera ich prawie 225 tys.). Sumerolodzy zajmują się badaniem i przetwarzaniem tych tekstów, jednak wyszukiwanie interesujących ich tabliczek jest dosyć niewygodne. Wynika to przede wszystkim z nieznajomości specyficznych dla baz danych języków zapytań. Większość serwisów internetowych udostępnia formularze ułatwiające wprowadzanie kryteriów wyszukiwania, jednakże mają one ograniczone możliwości (nie pozwalają na skomplikowane konstrukcje). Dlatego istnieje potrzeba stworzenia narzędzia, które będzie łączyło w sobie jak największą siłę wyrazu i łatwość użycia przez osoby znające jedynie dziedzinę problemu. Celem projektu przedstawionego w niniejszej pracy jest zaprojektowanie i implementacja języka Tablets Query Language (TQL) spełniającego powyższe wymagania.

TQL jest podstawą do tworzenia podobnych języków wyszukiwań dostosowanych do potrzeb innych grup ludzi, np. językoznawców. 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.

Zgodnie z paradygmatem języków dziedzinowych (Domain Specific Languages, DSL) TQL jest nakładką na inne języki zapytań (np. SQL). W związku z tym dla każdego sposobu reprezentacji danych należy skonstruować translator, którego zadaniem będzie przetłumaczenie zapytania. W ramach niniejszej pracy przedstawione zostaną dwa przykładowe translatory.

Część I

Problem przeszukiwania bazy tabliczek sumeryjskich

Rozdział 1

Podstawowe pojęcia

1.1. Pojęcia dziedzinowe

Sumerolog - naukowiec zajmujący się odczytywaniem pisma klinowego w języku sumeryjskim. Na potrzeby tej pracy to pojęcie jest rozszerzone do wszystkich ludzi zajmujących się odczytywaniem tabliczek klinowych (także w innych językach) i czerpiących z nich wiedzę historyczną.

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

Proweniencja (ang. provenience) - pojęcie używane przez sumerologów, oznacza miejsce pochodzenia/znalezienia glinianej tabliczki

Kliny (ang. cunes) - znaki występujące na glinianych tabliczkach.

Odczyty (ang. readings) - sposób transkrypcji klinów. Tabliczki elektroniczne są zapisane za pomocą odczytów. Na ogół jeden klin odpowiada jednemu odczytowi, ale może odpowiadać wielu różnym sekwencjom odczytów. Dodatkowo jeden odczyt może być zapisany za pomocą sekwencji klinów.

Pieczęć (ang. seal) - część tabliczki zawierająca znak rozpoznawczy autora

1.2. Pojęcia informatyczne

Alfabet (zbiór Σ, zbiór symboli terminalnych) - zbiór symboli (np. słów kluczowych, znaków specjalnych, literałów), z których zbudowane są słowa - konstrukcje języka.

Zbiór symboli nieterminalnych - zbiór symboli pomocnicznych, rozłączny z alfabetem.

Słowo nad alfabetem Σ - skończony ciąg symboli należących do zbioru Σ.

Język nad alfabetem Σ - zbiór słów nad alfabetem Σ.

Reguły gramatyki - reguły definiujące sposób tworzenia słów nad danym alfabetem. Każda reguła jest postaci $S1 \rightarrow S2$, gdzie $S1$ i $S2$ to ciągi symboli terminalnych i nieterminalnych, przy czym w ciągu $S1$ musi wystąpić przynajmniej jeden symbol nieterminalny.

Reguły określają możliwe podstawienia symboli w wyprowadzanym słowie - ciąg $S1$ można zastąpić przez $S2$.

Gramatyka - formalny sposób definiowania języka. Składa się z czterech elementów: zbioru symboli terminalnych, zbioru symboli nieterminalnych, symbolu startowego (należącego do zbioru symboli nieterminalnych) oraz zbioru reguł gramatyki. Wyprowadzanie słowa należącego do języka rozpoczynamy od symbolu startowego, przeprowadzamy podstawienia zgodnie z regułami gramatyki i kończymy, gdy wszystkie symbole w słowie należą do zbioru symboli terminalnych. Język określony przez gramatykę jest to zbiór słów, które są możliwe do wyprowadzenia z symbolu startowego za pomocą reguł gramatyki.

Struktura leksykalna języka - definicja symboli terminalnych (alfabetu).

Struktura składniowa języka - opis składni języka, zapisany np. za pomocą reguł gramatyki.

Semantyka - znaczenie i funkcja poszczególnych konstrukcji języka.

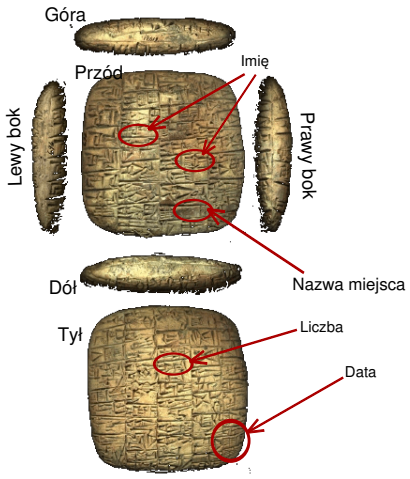
1.3. Pojęcia paradygmatyczne

Język dziedzinowy (ang. domain-specific language, DSL) - język programowania szczególnego zastosowania, rozwiązujący specyficzny problem lub zajmujący się wąską dziedziną, stworzony specjalnie na potrzeby danej dziedziny i do niej dostosowany.

Rozdział 2

Dziedzina problemu

Głównym pojęciem dziedziny jest tabliczka rozumiana dwojako - jako fizyczna tabliczka gliniana lub jako tabliczka w formie cyfrowej. Jej najważniejszym elementem jest treść zapisana klinami (w tabliczce glinanej) lub odczytami (w tabliczce elektronicznej), która może zawierać m. in. imiona osób lub bóstw, liczby, jednostki (np. przy opisywaniu wypłat), miejsca, daty. Część tych elementów można przetłumaczyć na współczesny język (np. jednostki przeliczyć na SI, datę opisową na datę liczbową BC). Ponadto gliniane tabliczki są zapisywane z różnych stron (od góry, z przodu, z tyłu itp) oraz mogą zawierać pieczęcie - co znajduje odzwierciedlenie w treści tabliczki elektronicznej.

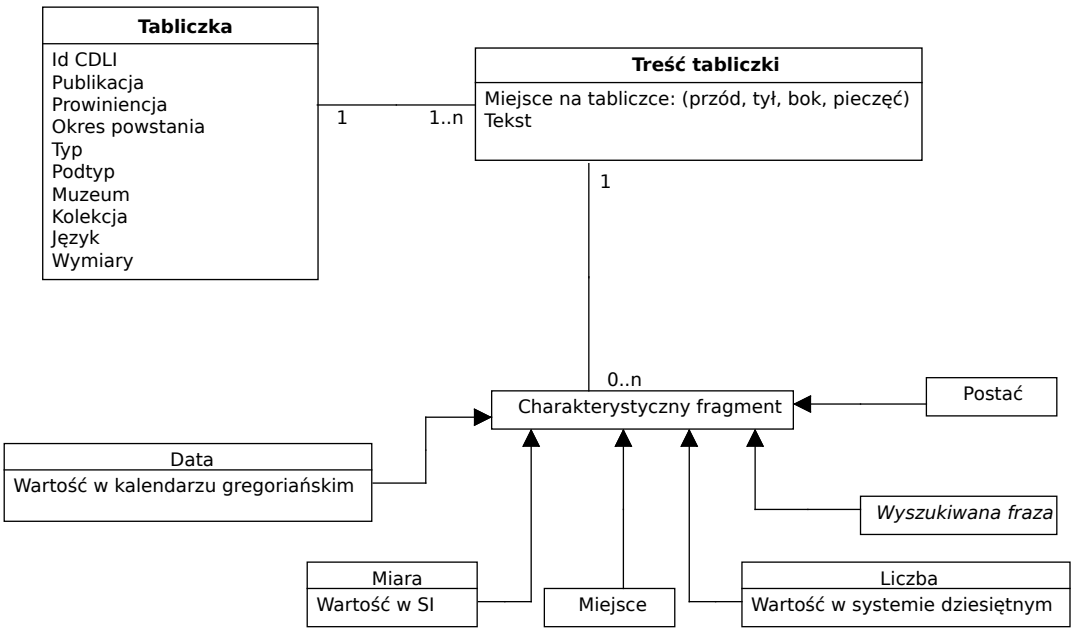


Rysunek 2.1: Gliniana tabliczka - struktura

Niestety odczyty zawarte w cyfrowym zapisie tabliczki są tylko jednym z wariantów tłumaczenia z klinów. Ponieważ w cyfrowej wersji nie ma klinów, możliwe są pomyłki w tłumaczeniach, które ciężko zweryfikować. Choć są one dosyć mało prawdopodobne, sumerolodzy chcieliby mieć możliwość wyszukiwania po alternatywnych tłumaczeniach.

Poza treścią tabliczki cyfrowa wersja zawiera także metadane informujące m. in. o jej miejscu znalezienia, czasie powstania, kolekcji do której obecnie należy, publikacji, w której się pojawiła. Są one istotne, gdyż często pozwalają na określenie o czym jest tabliczka bez dokładnej analizy jej treści. Jednym z atrybutów, który w znacznym stopniu pomaga zidentyfikować tabliczkę jest publikacja.

Sumerolodzy oczekują możliwości wyszukiwania po metadanych, po treści tabliczki (po odczytach) i po alternatywnych tłumaczeniach (po klinach). Dodatkową zaletą byłoby wyszukiwanie po tagach (imionach, jednostkach, datach itp) W pierwszej wersji języka implementujemy tylko wyszukiwanie po odczytach i metadanych.



Rysunek 2.2: Co powinna zawierać tabliczka w formie elektronicznej

Rozdział 3

Wcześniejsze rozwiązania

W chwili obecnej nie ma czegoś takiego jak język dostosowany do potrzeb sumerologów. Są strony internetowe oferujące wyszukiwanie za pomocą formularzy.

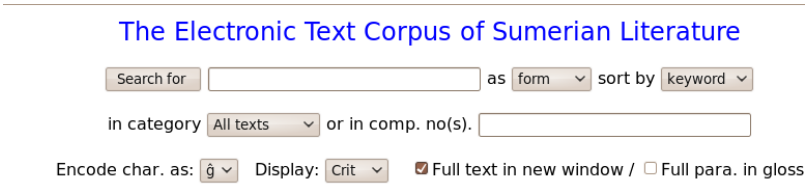
3.1. The Cuneiform Digital Library Initiative [3]

Rysunek 3.1: Formularz wyszukiwania na CDLI

Największa znana nam baza tekstów sumeryjskich (ok. 225 tys. tekstów), wyszukiwanie po praktycznie wszystkich możliwych parametrach, choć trochę mało wygodne. Dla każdej metadanej jest pole tekstowe z możliwymi opcjami wyszukiwania: “begins with“, “contains”, ”does not contain”. Dla treści jest pole tekstowe z opcjami “word“, “part of word“ oraz checkbox ”advanced search syntax”. Brakuje wyjaśnienia jak używać “Advanced search syntax”. Brakuje możliwości tworzenia złożonych zapytań.

3.2. The Electronic Text Corpus of Sumerian Literature [4]

Baza znacznie mniejsza, zawiera głównie teksty literackie. Wyszukiwanie mało rozbudowane.



Rysunek 3.2: Formularz wyszukiwania na etcsl

Część II

Definicja języka TQL

Rozdział 4

Gramatyka

4.1. Struktura leksykalna

4.1.1. Literały

String

Literał `String` jest ciągiem dowolnych znaków w cudzysłowiu (`"`). Nie może zawierać jedynie znaków "`"`" niepoprzedzonych "`\`".

SłowoOdLitery

Literał `SłowoOdLitery` to ciąg liter, cyfr oraz znaków "`-`", "`'`", "`_`", zaczynający się od litery, z wyjątkiem słów kluczowych.

SłowoOdLiczby

Literał `SłowoOdLiczby` 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)

4.2. Struktura składniowa 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*⟩ ::= ⟨*Nazwa pola*⟩ `:` ⟨*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*⟩ ::= `SłowoOdLitery`
| `SłowoOdLiczby`

⟨*Nazwa pola*⟩ ::= `SłowoOdLitery`

⟨*Nazwa*⟩ ::= `String`

Rozdział 5

Semantyka

Język TQL umożliwia wyszukiwanie na podstawie kryteriów dotyczących następujących danych:

Opis	**Nazwa pola w TQL**
numer tabliczki w bazie CDLI	cdli_id
miejsce pochodzenia (proweniencja)	provenience
okres powstania	period
typ i podtyp	genre
rok powstania	year
publikacja	publication
treść (odczyty)	text
treść (kliny)	cunetext
kolekcja	collection
muzeum	museum

Język można łatwo rozszerzać, aby umożliwić tworzenie kryteriów wyszukiwania w oparciu o inne dane (np. zawartość pieczęci).

Poniżej przedstawiamy semantykę wybranych przykładów.

5.1. Zapytanie 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”

5.2. Zapytanie 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

5.3. 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“

5.4. Wywołanie zapytania zdefiniowanego

5.4.1. Zwykłe

```
search "zwierzęta w Gar*a"
```

Wynikiem zapytania będą dokładnie te tabliczki, które spełniają wszystkie warunki zapytania ”zwierzęta w Gar*a“.

5.4.2. Z dodatkowym warunkiem wyszukiwania

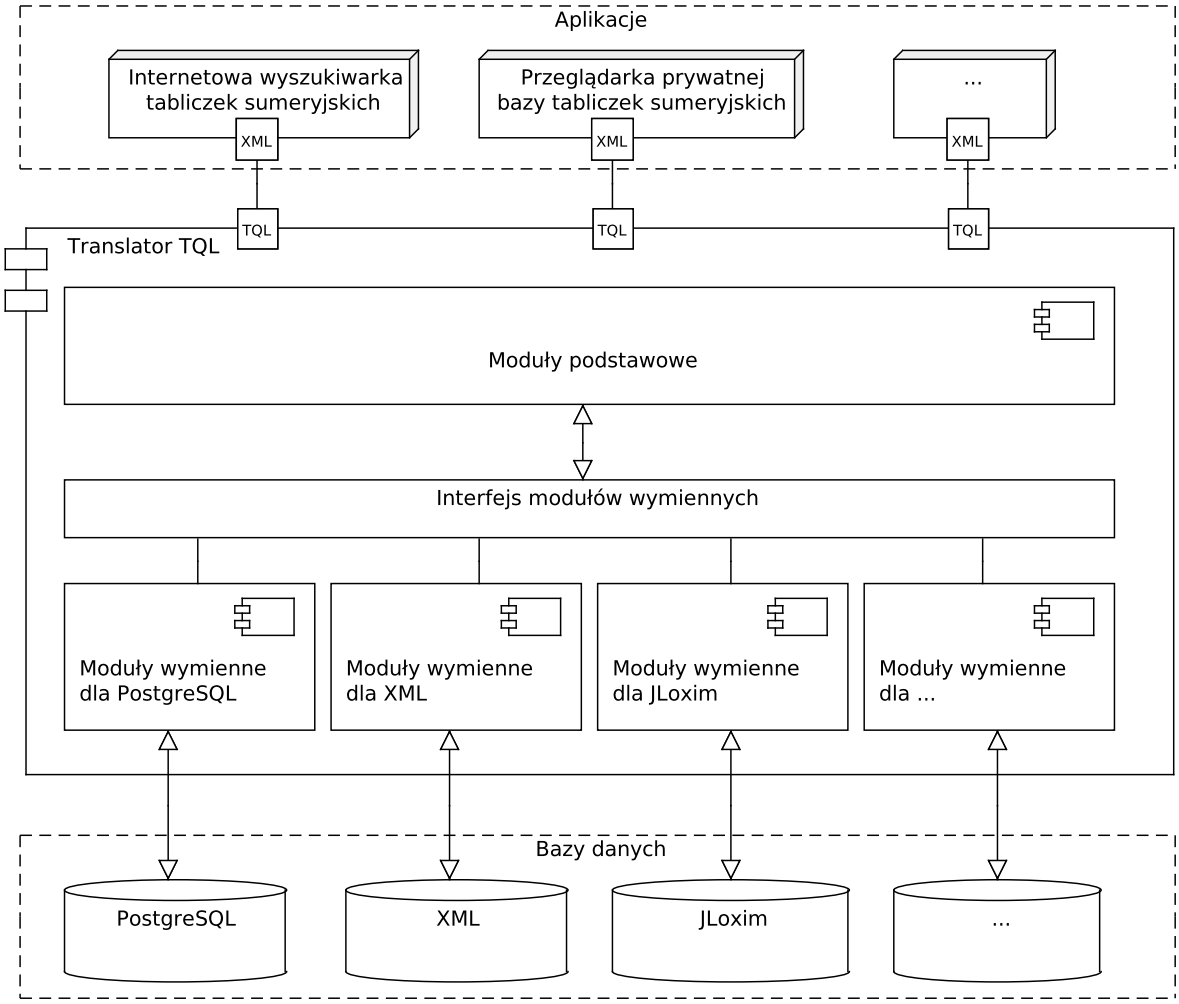
```
search
  text: adad-tilati
in "zwierzęta w Gar*a"
```

Wynikiem zapytania będą wszystkie tabliczki, które:

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

Część III

Implementacja



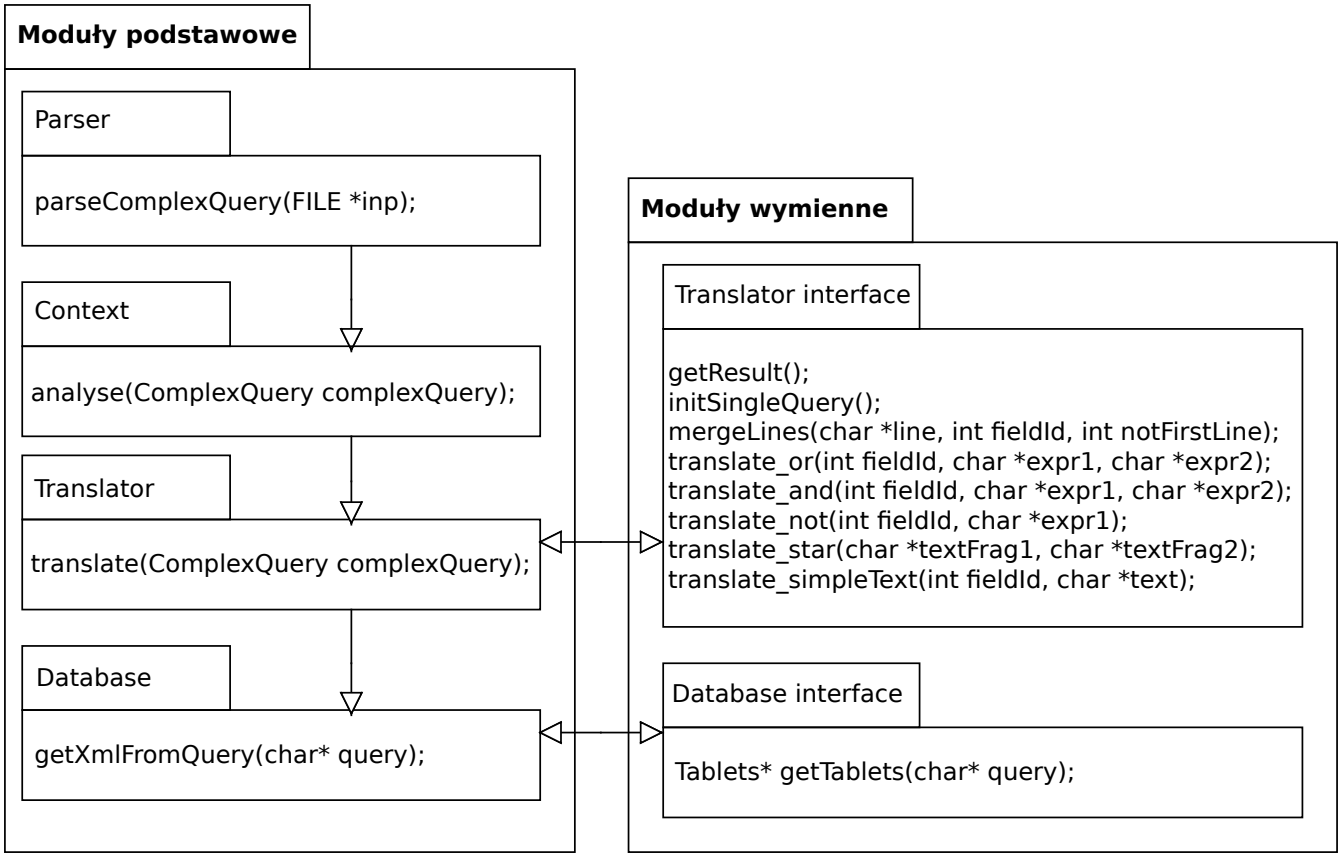
Rysunek 5.1: Struktura systemu korzystającego z translatora

Jednym z głównych założeń języka TQL jest niezależność od struktury danych. W związku 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. Makefile domyślnie buduje obie implementacje tql-a;



Rysunek 5.2: Podział programu na moduły

Rozdział 6

Moduły podstawowe

6.1. 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.cpp
- Parser.h
- TQL.y
- TQL.l

6.2. 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.cpp
- Context.h

6.3. Translator

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

- Translator.cpp
- Translator.h
- Translator_interface.h (interfejs modułu translatora zależnego od bazy danych)

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

6.4. Baza

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

- Database.cpp
- Database.h
- Database_interface.h (interfejs modułu bazy zależnego od bazy danych)

Wywołuje funkcję getTablets() z Database_interface.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;
```

6.5. Pliki pomocnicze

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

- Absyn.cpp
- Absyn.h

Tablica symboli:

- Symbols.cpp
- Symbols.h

Obsługa błędów:

- Err.cpp
- Err.h

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

- Cexplode.cpp
- Cexplode.h

Rozdział 7

Moduły wymienne

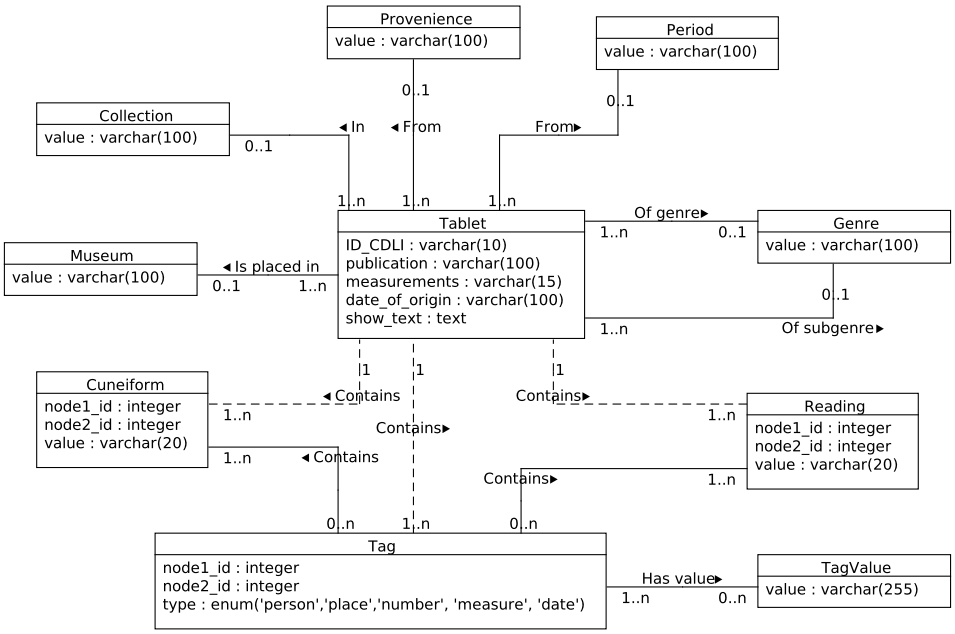
Pliki zależne od wyboru konkretnej bazy danych to:

- Translator_<nazwa>.cpp - dla modułu translatora
- Database_<nazwa>.cpp - dla modułu bazy

Ich interfejsy są wspólne dla wszystkich baz danych.

7.1. Baza PostgreSQL

7.1.1. Diagram encji



Rysunek 7.1: 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).

7.1.2. Translator_postgres

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.

Stałe fragmenty 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 zapytań o atrybuty tabliczki

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 zapytań o treść tabliczki

Przy tłumaczeniu zapytań o treść tabliczki 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

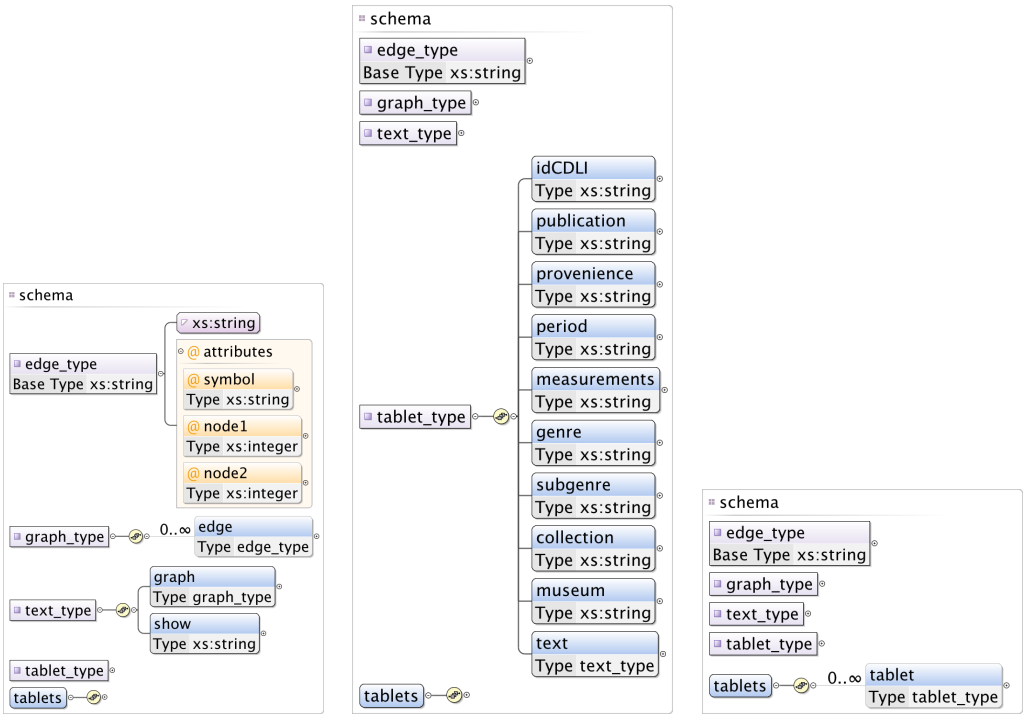
Operator	Tłumaczenie
+	SELECT * FROM (SELECT id_tab, CAST(array_accum(nodes) as TEXT) as nodes, COUNT(DISTINCT id_seq) as seq, <id_sekw> as id_seq FROM (<zapytanie1> UNION <zapytanie2>) as c GROUP BY id_tab) as b WHERE b.seq=2
−	SELECT id_tab, '' as nodes, 0 as seq, <id_sekw> as id_seq FROM ((SELECT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

7.1.3. Database_postgres

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.

7.2. Baza XML

7.2.1. Schemat dokumentu



(a) element text (b) element tablet (c) element tablets

Rysunek 7.2: Schematy poszczególnych elementów (kolor pomarańczowy oznacza atrybuty, a niebieski elementy)

Schemat dokumentu zamieszczony powyżej jest graficznym przedstawieniem dokumentu XML Schema (dodatek A) wygenerowanym przez program Oxygen.

Przy jego tworzeniu, podobnie jak w bazie relacyjnej, skorzystałyśmy z pomysłu dra Wojciecha Jaworskiego, aby przedstawić treść tabliczki w formie grafu. Każda krawędź tego grafu (odpowiadająca odczytowi) jest oddzielnym elementem, zawierającym atrybuty *node1*, *node2* i *symbol*. Atrybuty *node1* oraz *node2* oznaczają numery węzłów grafu, natomiast atrybut *symbol* to typ symbolu znajdującego się na danej krawędzi. Dodatkowo przechowujemy treść tabliczki w formie napisu (element *show*).

Metadane tabliczek przechowywane są w postaci podelementów elementu *tablet*.

7.2.2. Translator_xml

Do przeszukiwania dokumentu XML wykorzystujemy język XQuery, który jest częścią rekomendacji W3C dotyczącej XML.
Proste zapytanie TQL jest tłumaczone na pojedynczą konstrukcję FLWOR (For Let Where Order by Return).

Stałe fragmenty zapytania

Każde zapytanie w części For zawiera:

```
FOR $tablet IN .//tablet
```

a w części Return:

```
RETURN <tablet>
{$tablet/idCDLI}
{$tablet/publication}
{$tablet/provenience}
{$tablet/period}
{$tablet/measurements}
{$tablet/genre}
{$tablet/subgenre}
{$tablet/collection}
{$tablet/museum}
{$tablet/text/show}
<seq>...</seq>
</tablet>
```

Zawartość elementu seq zależy od ilości sekwencji, po których wyszukujemy.

Tłumaczenie zapytań o atrybuty tabliczki

Konstrukcja	**Tłumaczenie na XQuery**
provenience: wartosc	`fn:matches($tablet/provenience,'^wartosc$')`
publication: wartosc	`fn:matches($tablet/publication,'^wartosc$')`
period: wartosc	`fn:matches($tablet/period,'^wartosc$')`
genre: wartosc	`(fn:matches($tablet/genre,'^wartosc$')` `or fn:matches($tablet/subgenre,'^wartosc$'))`
cdli_id: wartosc	`fn:matches($tablet/idCDLI,'^wartosc$')`

Tłumaczenie zapytań o treść tabliczki

Każda sekwencja, po której wyszukujemy powoduje dodanie do zapytania następujących konstrukcji:

- do części Let:

```
let $seq <id_sekw> := (
for $edge_end in $tablet//edge
for $edge_start in $tablet//edge
where (
fn:matches($edge_start,'^<sekw[0]>$')
and (
some $edge1 in $tablet//edge[@node1=$edge_start/@node2]
satisfies (fn:matches($edge1,'^<sekw[1]>$')
and ...
and fn:matches($edge_end,'^<sekw[dl_sekw-1]>$')))))
return <seq<id_sekw>> {$edge_start/@node1} {$edge_end/@node2} </seq<id_sekw>>
```

- do części Where

```
$seq<id_sekw>
```

- do części Return w elemecie seq

```
$seq<id_sekw>
```

Tłumaczenie operatorów

Poniższe tłumaczenia dotyczą zarówno konstrukcji prostych jak i złożonych.

Operator	**Tłumaczenie**
/	(<zapytanie1> or <zapytanie2>)
–	not (<zapytanie_negowane>)
+	(<zapytanie1> and <zapytanie2>)
*	.*

Zapytania złożone

Zapytanie złożone, składające się z wielu zapytań prostych tłumaczymy na sekwencję zapytań XQuery połączonych znakiem ','.

7.2.3. Database_xml

Odpowiada za wywołanie zapytania i zapisanie wyniku do struktury Tablets. Jako bazę danych wykorzystujemy plik XML, określony w pliku konfiguracyjnym xml.conf. Do wyszukiwania wykorzystujemy procesor XQuery Zorba. Posiada on API m.in. do C++, które pozwala na przekazanie zapytania do bazy oraz przetworzenie wyniku.

Podsumowanie

Podsumowanie projektu

Udało nam się zaprojektować i zaimplementować język TQL, który spełnia założone przez nas wymagania. Jest intuicyjny i prosty w użyciu dla osób znających jedynie dziedzinę problemu, co potwierdza opinia dr hab. Marka Stępnia. Jednocześnie minimalnie ogranicza siłę wyrazu - pozwala tworzyć skomplikowane zapytania. Zatem cel naszego projektu został osiągnięty.

Język TQL ma kilka ograniczeń w stosunku do typowych języków zapytań. Przede wszystkim nie pozwala wpływać na postać wyniku, co utrudnia np. zbieranie danych statystycznych. Można jednak stworzyć narzędzia do samej prezentacji wyników zapytań, które pokonają to ograniczenie. Poza tym TQL jest jedynie językiem wyszukiwania - nie można z jego pomocą zmieniać danych znajdujących się w bazie, ani dodawać nowych. Akceptujemy to, gdyż zgodnie z definicją języków dziedzinowych, TQL jest językiem wąskiego zastosowania.

Jednak, dzięki swoim specyficznym cechom, można również dostosować go do wykorzystania w innych dziedzinach. Należy tu wspomnieć o braku ograniczenia ilości i nazw pól, po których można wyszukiwać oraz przystosowaniu do wyszukiwania wg kryteriów dotyczących danych tekstowych. Na podstawie TQL można łatwo stworzyć rodzinę języków dla różnych dziedzin zajmujących się przeszukiwaniem tekstów.

W ramach niniejszej pracy prezentujemy dwie implementacje języka TQL. Pierwsza wymagała dużo pracy, gdyż należało stworzyć zarówno moduły niezależne od bazy danych, jak i zależne. Jednak druga implementacja była łatwa. Pomimo tego, że docelowe bazy danych bardzo się różnią, wymagała jedynie przetłumaczenia poszczególnych konstrukcji TQL na XQuery. Pozwala to sądzić, że każda kolejna implementacja, dla różnych typów baz i różnych schematów danych, nie będzie wymagała dużego wysiłku.

Możliwości rozwoju

W pierwszej kolejności chcemy stworzyć własną stronę z wyszukiwarką tabliczek, która byłaby dostępna dla Wydziału Historii UW i rozwijana we współpracy z naukowcami z tego wydziału. Myślimy również o nawiązaniu współpracy z projektem CDLI, opisanym w rozdziale 3. Implementacja TQL dla bazy CDLI, wraz z interfejsem webowym, mogłaby być przydatnym narzędziem dla sumerologów na całym świecie.

Gdy TQL spotka się z zainteresowaniem naukowców, będziemy rozwijać język dodając przede wszystkim wyszukiwanie po klinach i po tagach. Dodatkowo chciałybyśmy stworzyć narzędzie do analizy uszkodzonych fragmentów i wykrywania pomyłek (np. literówek) w odczytanych tekstach. Bez takiego narzędzia można przy wyszukiwaniu pominąć cenne tabliczki zawierające pożądaną treść. Dalszy rozwój projektu będzie zależał przede wszystkim od potrzeb zgłaszanych przez sumerologów.

Appendices

Dodatek A

Schemat bazy danych xml

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:complexType name="edge_type">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="symbol" type="xs:string"/>
      <xs:attribute name="node1" type="xs:integer"/>
      <xs:attribute name="node2" type="xs:integer"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="graph_type">
  <xs:sequence>
    <xs:element name="edge" minOccurs="0" maxOccurs="unbounded" type="edge_type"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="text_type">
  <xs:sequence>
    <xs:element name="graph" type="graph_type"/>
    <xs:element name="show" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="tablet_type">
  <xs:sequence>
    <xs:element name="idCDLI" type="xs:string"/>
    <xs:element name="publication" type="xs:string"/>
    <xs:element name="provenience" type="xs:string"/>
    <xs:element name="period" type="xs:string"/>
    <xs:element name="measurements" type="xs:string"/>
    <xs:element name="genre" type="xs:string"/>
    <xs:element name="subgenre" type="xs:string"/>
    <xs:element name="collection" type="xs:string"/>
    <xs:element name="museum" type="xs:string"/>
    <xs:element name="text" type="text_type"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="tablets">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="tablet" type="tablet_type" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

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

[3] http://cdli.ucla.edu

[4] http://etcsl.orinst.ox.ac.uk/

Spis rysunków