**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

marzec 2011

## 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.

Celem tej pracy jest stworzenie intuicyjnego dla sumerologów języka zapytań o nazwie Tablets Query Language (TQL) przeznaczonego do przeszukiwania zbiorów tabliczek. Ma on stanowić znaczące uproszczenie w stosunku do istniejących już języków zapytań, między innymi dzięki wprowadzeniu pojęć naturalnych dla rozważanej dziedziny. Język ten pozwala na tworzenie skomplikowanych zapytań wyszukujących, natomiast nie udostępnia funkcjonalności tworzenia i modyfikowania bazy. Można go rozszerzać i zmieniać tak, by mógł służyć do zastosowań z innych dziedzin.

W pracy przedstawiono definicję języka TQL wraz z jego przykładowymi implementacjami. Jedna z nich jest oparta na relacyjnej bazie danych, a druga na bazie w postaci pliku XML.

Praca jest napisana jako projekt dwuosobowy wspólnie z (...).



## Słowa kluczowe

tabliczka, sumerologia, kliny, odczyty, język dziedzinowy, DSL, Domain-Specific Languages

## 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

## Opis problemu

Ok. 3500 lat p.n.e. starożytni Sumerowie, jako prawdopodobnie pierwsi na świecie, zaczęli używać pisma. Ze względu na kształt liter odciśniętych za pomocą trzciny w miękkiej glinie zostało ono później nazwane pismem klinowym. Sumerowie używali go głównie w celach administracyjnych. Zapisywali między innymi rachunki za dostawy zwierząt oraz rozliczenia z wykonanej na polu lub w warsztacie pracy i należnej zapłaty. Jednymi z ciekawszych tekstów są tzw. listy królów, na których znajdują się daty panowania kolejnych władców i ich osiągnięcia w poszczególnych latach. Ważniejsze tabliczki wypalano, jednak większość po pewnym czasie niszczono, aby zużytą glinę można było ponownie wykorzystać. Do dzisiaj przetrwały głównie te tabliczki, które zostały wypalone podczas przypadkowego pożaru archiwum. Znaczna część z nich jest zniszczonych, jednak wciąż stanowią ogromną wartość historyczną.

Na podstawie zachowanych tabliczek można się wiele dowiedzieć o historii Sumeru, a także o życiu zwykłych ludzi w tym okresie, o ich zarobkach, zasiłkach i dniach wolnych. Odczytywaniem tych informacji zajmują się sumerolodzy z całego świata. Aby łatwiej dzielić się zdobytą wiedzą, zaczęli tworzyć cyfrowe bazy tabliczek dostępne w internecie. Największa z nich to Cuneiform Digital Library Initiative (CDLI), zawierająca prawie 225 tys. tekstów.

Jednak wyszukiwanie interesujących tabliczek może być uciążliwe.

Sumerolodzy mogliby skorzystać z języków zapytań specyficznych dla baz danych (np. SQL), ale po pierwsze większość z nich nie zna tych języków, a po drugie budowanie w ten sposób złożonych zapytań dotyczących danych tekstowych jest czasochłonne. Takie języki były tworzone z myślą o bardziej generycznych zastosowaniach i nie odpowiadają potrzebom sumerologów. Z tych powodów większość istniejących obecnie serwisów internetowych udostępnia formularze ułatwiające wprowadzanie kryteriów wyszukiwania. Jednak mają one ograniczone możliwości i nie pozwalają na konstruowanie bardziej skomplikowanych zapytań. Dlatego istnieje potrzeba stworzenia narzędzia wspomagającego wyszukiwanie, które będzie łączyło w sobie jak największą siłę wyrazu oraz łatwość użycia dla osób znających jedynie dziedzinę problemu.

Przy tego typu problemach z pomocą przychodzi informatyka.

## Propozycja rozwiązania

Do tej pory powstało wiele różnych systemów informatycznych, a doświadczenie stopniowo zdobywane przy ich budowie prowadzi do ciągłego rozwijania nauki zwanej inżynierią oprogramowania, zajmującej się praktyczną stroną wytwarzania systemów. Obecnie jest znanych bardzo wiele podejść do tworzenia oprogramowania, a każdemu odpowiada gałąź problemów, w których sprawdza się najlepiej.

Najbardziej popularne jest programowanie zorientowane obiektowo (ang. *Object–Oriented Programming*, OOP), które polega na modelowaniu świata rzeczywistego w postaci *obiektów*. Obiekty posiadają dane oraz metody, które mogą te dane zmieniać. Program jest zbiorem obiektów komunikujących się między sobą.

Innym podejściem jest architektura zorientowana na usługi (ang. *Service–Oriented Architecture*, SOA), w której definiuje się niezależne od siebie usługi (services) i udostępnia tylko ich interfejsy, ukrywając implementację.

Rozważając problem sumerologów, zdecydowałyśmy się na programowanie zorientowane na język (ang. *Language–Oriented Programming*). Polega ono na stworzeniu języka odpowiadającego dziedzinie problemu, czyli tzw. języka dziedzinowego (ang. *Domain–Specific Language*, DSL). Dopiero w tym nowym języku rozwiązuje się konkretny problem (np. znalezienie

tekstów sumeryjskich z epoki Ur III dotyczących owiec). Opisywanie problemów jest wtedy znacznie prostsze i bardziej naturalne, a dzięki temu wygodne dla ludzi związanych tylko z konkretną dziedziną.

Wyróżnia się dwa rodzaje języków dziedzinowych – "wewnętrzne" (ang. *internal*), które zachowują składnię istniejących już języków i ograniczają tylko ich możliwości, oraz "zewnętrzne" (ang. *external*), które są zupełnie nowymi językami, tłumaczonymi na istniejące wcześniej [1]. W naszym przypadku narzucającym się podejściem było stworzenie nowego, zewnętrznego DSL, zaprojektowanego specjalnie do wyszukiwania tabliczek sumeryjskich, który miałby składnię przejrzystą dla sumerologa.

Do każdej bazy zawierającej tabliczki można skonstruować moduł pozwalający na wyszukiwanie z użyciem takiego DSL. Zatem jest to rozwiązanie, które może zostać wykorzystane przez wiele różnych serwisów, zarówno nowo powstających, jak i tych już istniejących. Dzięki temu wyszukiwarki, internetowe oraz stacjonarne, mogłyby mieć taki sam lub bardzo podobny interferjs, co znacznie ułatwiłoby pracę sumerologom. DSL pozwoliłby usystematyzować prace nad wszelkimi wyszukiwarkami tabliczek sumeryjskich, a przez to również pracę sumerologów. Jest to niewątpliwa zaleta takiego rozwiązania. Wyraźnie widać jego przewagę nad podejściem polegającym na niezależnym rozwijaniu interfejsów poszczególnych wyszukiwarek, czego efektem są coraz bardziej skomplikowane formularze do wpisywania zapytań.

### Tablets Query Language

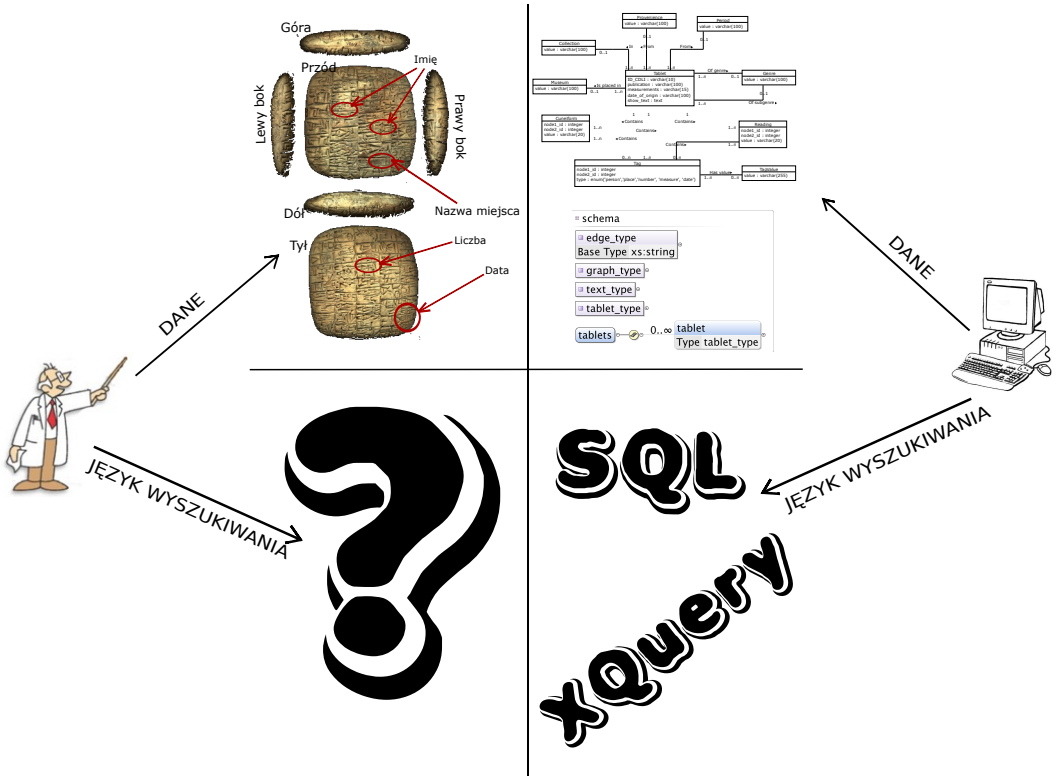
Wychodząc naprzeciw potrzebom sumerologów prezentujemy stworzony przez nas język dziedzinowy do wyszukiwania tabliczek sumeryjskich – *Tablets Query Language* (TQL). Został on zaprojektowany przede wszystkim jako intuicyjny i zrozumiały dla osób z dziedziny problemu. Ma także minimalnie ograniczać siłę wyrazu tzn. pozwalać na konstruowanie jak najbardziej skomplikowanych zapytań. Chcemy również, żeby był niezależny od sposobu reprezentacji tabliczek. W niniejszej pracy udowodnimy, że TQL spełnia wszystkie te wymagania.

Projektowanie języka dziedzinowego może polegać na wykorzystaniu istniejącego języka (wzorzec "Languge exploitation" [2]) lub na stworzeniu od podstaw innego niż istniejące do tej pory (wzorzec "Language invention" [2]). Ze względu na to, że TQL ma być używany głównie przez osoby nieznające języków zapytań do baz danych, zdecydowałyśmy się na to drugie podejście. Pozwala ono na lepsze dopasowanie TQL do potrzeb przyszłych użytkowników.

Przy wyborze sposobu implementacji TQL kierowałyśmy się potrzebą umożliwienia pracy z różnymi bazami tabliczek. Aby to zapewnić, zastosowałyśmy wzorzec "Preprocessor" [2], a konkretnie "Source–to–source transformation". Oznacza to, że TQL jest automatycznie tłumaczony na inne, istniejące wcześniej języki zapytań bardziej ogólnego zastosowania, np. SQL, XQuery [3], odpowiednio do sposobu reprezentacji danych.

Sposobem reprezentacji danych nazywamy rodzaj bazy danych (np. relacyjna, obiektowa, XML), konkretną implementację serwera (np. PostgreSQL [4], Zorba [5]) oraz schemat danych. Dla każdego takiego sposobu reprezentacji zapytanie w docelowym języku może się różnić, a zatem różnić się będzie również program tłumaczący TQL na ten język. Wynika z tego, że dla każdego sposobu reprezentacji danych należy stworzyć oddzielny program tłumaczący. W ramach niniejszej pracy, oprócz języka TQL, zaprezentujemy dwie prototypowe implementacje komponentu do wyszukiwarki, zawierającego m.in. program tłumaczący. Ten komponent, nazywany dalej translatorem, tłumaczy zapytanie TQL na zapytanie w języku specyficznym dla danej bazy, przekazuje przetłumaczone zapytanie do bazy, odbiera wynik i przedstawia go w odpowiedniej postaci. Pokażemy implementacje translatora dla bazy PostgreSQL (z językiem wyszukiwania SQL) oraz dla bazy XML (z językiem wyszukiwania XQuery).

TQL może być także podstawą do tworzenia podobnych języków wyszukiwania 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.




Rysunek 1: Przedstawienie problemu

# Rozdział 1

# Problem przeszukiwania bazy tabliczek sumeryjskich

## 1.1. Podstawowe pojęcia

### 1.1.1. 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.

### 1.1.2. Pojęcia informatyczne

**Alfabet (zbiór $\Sigma$, zbiór symboli terminalnych)** – zbiór symboli (np. słów kluczowych, znaków specjalnych, literałów), z których zbudowane są słowa – konstrukcje języka.

**Zbiór symboli nieterminalnych** – zbiór symboli pomocniczych, rozłączny z alfabetem.

**Słowo nad alfabetem $\Sigma$** – skończony ciąg symboli należących do zbioru $\Sigma$.

**Język nad alfabetem $\Sigma$** – zbiór słów nad alfabetem $\Sigma$.

**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.

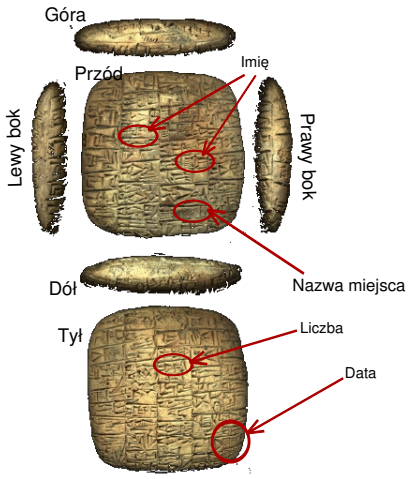
### 1.1.3. Pojęcia paradygmatyczne

**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.

## 1.2. Dziedzina problemu

Podstawowym elementem rozpatrywanej przez nas dziedziny jest *tabliczka sumeryjska*. Pojęcie to odnosi się zarówno do fizycznej *tabliczki glinianej* jak i do jej cyfrowej reprezentacji, odpowiedniej do przechowywania na komputerze.

Tabliczki gliniane mają różne kształty i rozmiary. Mogą być zapisane z wielu stron (z przodu, z tyłu, od góry, z boku, itp.), a także mogą zawierać pieczęcie, na których również znajduje się tekst. Przez kilka tysięcy lat istnienia ulegały też uszkodzeniom uniemożliwiającym obecnie ich pełne odczytanie. Przykładowa gliniana tabliczka zaprezentowana jest na rysunku 1.1.




Rysunek 1.1: Gliniana tabliczka – struktura

Tekst na tabliczkach jest zapisany za pomocą *klinów*. Pismo klinowe jest prawdopodobnie najstarszym pismem na świecie (konkurują z nim jedynie hieroglify egipskie) [6]. Początkowo składało się z piktogramów z czasem coraz bardziej upraszczanych. Służyło pomocą w rozliczeniach i umowach, zapisywano np. ilość i rodzaj obiecanych ludziom zwierząt.

Treść tabliczek to najczęściej dokumenty urzędowe dotyczące czynności administracyjnych i gospodarczych, produkcji i dystrybucji rozmaitych artykułów, uroczystości religijno–kulturowych, umów cywilnoprawnych itp. [7] Często można spotkać w niej między innymi imiona osób i bóstw, liczby, jednostki (np. mina – jednostka wagi), miejsca, daty. Niektóre z tych elementów można przetłumaczyć na współczesny język, na przykład jednostki można przeliczyć na SI, datę opisową na datę wg obecnego kalendarza gregoriańskiego.

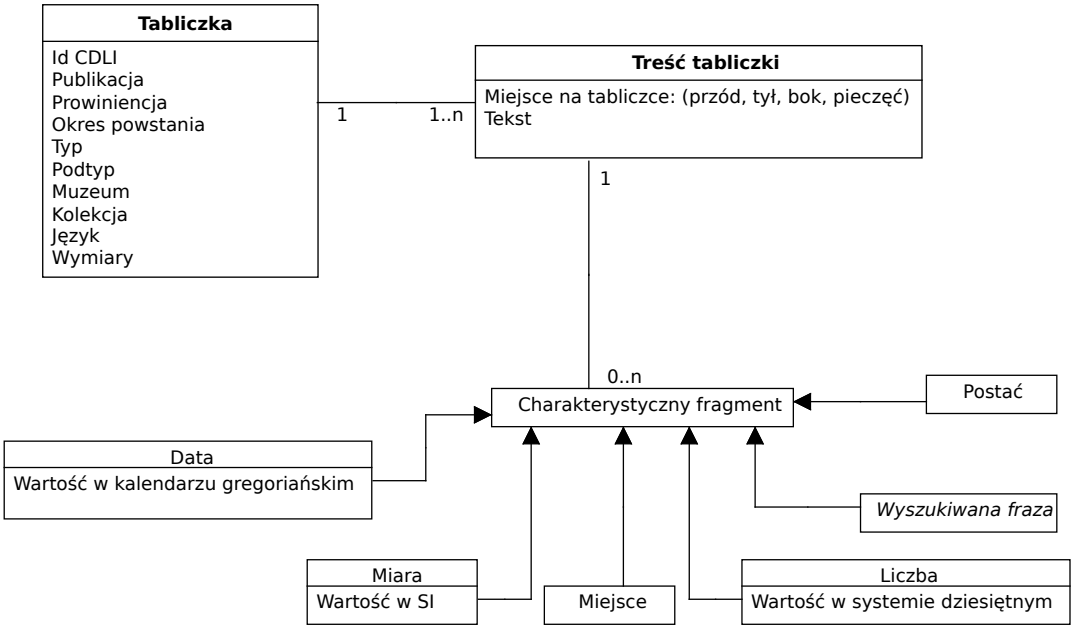
Dla ułatwienia pracy sumerolodzy tłumaczą zapis klinowy na tzw. *odczyty*. Są one sposobem zapisania tekstu sumeryjskiego za pomocą współczesnych znaków – liter alfabetu łacińskiego i cyfr arabskich, na podstawie prawdopodobnego brzmienia słów w języku sumeryjskim. W tłumaczeniu uwzględniony jest także kształt tabliczki i rozmieszczenie poszczególnych fragmentów tekstu (zgodnie z rysunkiem 1.1).

Odczyty zawarte w cyfrowym zapisie tabliczki są tylko jednym z wariantów tłumaczenia z klinów. Przede wszystkim dlatego, że jeden klin może zostać odczytany na wiele różnych sposobów. Ponadto odczytowi może odpowiadać nie tylko jeden klin, ale także ich sekwencja. Ponieważ w cyfrowej wersji nie ma klinów, trudno jest zweryfikować ewentualne pomyłki w tłumaczeniach.

W takiej właśnie formie tabliczki elektroniczne są przechowywane w pamięci komputerów. Niestety ten zapis może być mylący ze względu na niejednoznaczności występujące przy tłumaczeniu.

Źródłem dodatkowych problemów są uszkodzenia tabliczek. W wielu przypadkach naukowcy mogą domyślić się jak wyglądał brakujący fragment. Zawsze natomiast informację o uszkodzeniu oraz domniemaną brakującą treść zapisują w tabliczce cyfrowej. Ponieważ pewne konwencje opisywania tych informacji zostały przyjęte dosyć późno i nie przez wszystkich sumerologów, są one zapisywane na różne sposoby. Również tutaj możliwe są pomyłki i niejednoznaczności.

Tabliczka cyfrowa, oprócz treści, zawiera także informacje, które nie były zawarte na tabliczce glinianej (rysunek 1.2). Są to metadane, takie jak miejsce znalezienia tabliczki,




Rysunek 1.2: Co powinna zawierać cyfrowa reprezantacja tabliczki

okres, w którym powstała, czy nazwa kolekcji, do której obecnie należy. Te informacje są istotne przy przeszukiwaniu, gdyż często pozwalają na określenie o czym jest tabliczka bez dokładnej analizy jej treści. W praktyce, wśród sumerologów, atrybutem, który w znacznym stopniu pomaga zidentyfikować tabliczkę, jest informacja o jej publikacji.

Nie ma wątpliwości, że cyfrowa postać tabliczek sumeryjskich wraz z podstawowymi możliwościami wyszukiwania znacząco ułatwia sumerologom ich badania. Brakuje im jednak możliwości znajdowania tabliczek na podstawie bardziej skomplikowanych i rozbudowanych kryteriów. Wychodząc naprzeciw tej potrzebie, chcemy stworzyć język, w którym będą oni mogli w łatwy sposób wyrażać, jakich tabliczek potrzebują, i który jednocześnie będzie można wykorzystać do przeszukiwania baz danych. Język ten powinien przede wszystkim umożliwiać wyszukiwanie na podstawie treści tabliczki (odczytów), alternatywnych tłumaczeń (wymaga to przetłumaczenia tabliczki z powrotem na kliny) oraz metadanych. Dodatkową zaletą byłoby wyszukiwanie po specyficznych fragmentach (tagach), takich jak imiona, jednostki, daty. W pierwszej wersji języka implementujemy tylko wyszukiwanie po odczytach i metadanych.

## 1.3. Wcześniejsze rozwiązania

Dotychczasowe rozwiązania problemu przeszukiwania bazy tabliczek opierają się na pomysle stworzenia formularza, w którym, wypełniając odpowiednie pola, można określić jakich tabliczek szukamy. Formularze różnią się od siebie przede wszystkim poziomem skomplikowania. Im bardziej rozbudowany, tym więcej można w nim zdefiniować warunków dotyczących parametrów wyszukiwania, ale jednocześnie tym trudniej z niego korzystać. Z punktu widzenia dziedziny naszego problemu podstawową wadą takich rozwiązań jest brak elastyczności. Nie ma możliwości tworzenia złożonych konstrukcji, zarówno na poziomie pojedynczego pola formularza, jak i grupy pól.

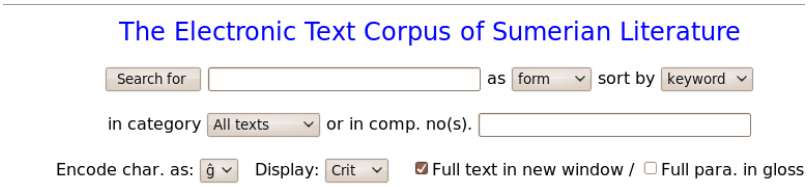
Poniżej przedstawimy dwie przykładowe strony internetowe oferujące wyszukiwanie za pomocą formularzy.

### 1.3.1. The Cuneiform Digital Library Initiative [8]


Rysunek 1.3: Formularz wyszukiwania na stronie CDLI

Największą znaną nam bazą tekstów sumeryjskich jest The Cuneiform Digital Library Initiative (CDLI), która zawiera ok. 225 tys. tabliczek. Posiada stosunkowo rozbudowany formularz, który pozwala na wyszukiwanie po wielu parametrach tabliczki (m.in. po danych dotyczących czasu i miejsca powstania, komentarzach CDLI). Dla każdego parametru jest pole tekstowe z możliwymi opcjami wyszukiwania: “begins with“, “contains”, ”does not contain”. Do wyszukiwania na podstawie treści tabliczki jest pole tekstowe z opcjami “word“, “part of word“. Jest również checkbox ”advanced search syntax”, jednak brakuje wyjaśnienia, jak go używać. Nie można tworzyć bardziej skomplikowanych warunków dla poszczególnych parametrów (np. nie można wyszukać tabliczek, które powstały w Uruk lub w Garshanie) ani łączyć kilku zapytań w jedno zapytanie złożone.

### 1.3.2. The Electronic Text Corpus of Sumerian Literature [9]




Rysunek 1.4: Formularz wyszukiwania na stronie ETCSL

The Electronic Text Corpus of Sumerian Literature (ETCSL) posiada znacznie mniejszą bazę niż CDLI, składającą się z około 400 tekstów, głównie literackich. Ma ograniczone możliwości wyszukiwania po metadanych – tylko po kategorii tekstu, jednak udostępnia tworzenie bardziej skomplikowanych zapytań dotyczących treści tabliczki. Pozwala określić typ wyszukiwanego słowa – do wyboru: form, lemma, label, pos, emesal, sign, a także jego znaczenie w tekście (np. czy jest imieniem bóstwa) oraz część mowy, do której należy. Można również wyszukiwać na podstawie fragmentu słowa.

# Rozdział 2

# Definicja języka TQL

## 2.1. Tablets Query Language

Tablets Query Language (TQL) jest naszą propozycją rozwiązania problemu wyszukiwania tabliczek sumeryjskich. Jest to zewnętrzny język dziedzinowy stworzony do tego, aby służyć sumerologom jako język zapytań. Formalna definicja składni TQL znajduje się w rozdziale 2.2. Jest ona zaprojektowana od podstaw, dzięki czemu jest bardzo prosta i intuicyjna, co będzie widać na przykładach opisujących semantykę w rozdziale 2.3. Wiele języków dziedzinowych, w przeciwieństwie do TQL, bazuje na istniejących już językach. Są to tzw. wewnętrzne języki dziedzinowe. Jednak w przypadku naszego problemu było to niewskazane rozwiązanie, gdyż składnia przejęta z istniejącego języka zapytań bardziej ogólnego zastosowania byłaby nieintuicyjna, a tworzone zapytania długie i skomplikowane. Udało nam się połączyć prostą składnię TQL z dużą siłą wyrazu. Język daje możliwość formułowania złożonych wyrażeń do wyszukiwania na podstawie pojedynczej metadanej lub treści tabliczki oraz możliwość łączenia wielu zapytań w jedno.

Dodatkowo jednym z głównych założeń przyjętych przy tworzeniu języka TQL jest niezależność od rzeczywistej reprezentacji danych. Przy konstruowaniu go nie brałyśmy pod uwagę sposobu fizycznej reprezentacji tabliczek, czyli rodzaju bazy danych oraz schematu danych. Skupiłyśmy się jedynie na dziedzinie problemu, czyli na tym, co zawiera tabliczka oraz na podstawie jakich informacji chcemy wyszukiwać. Zakładamy, że niezależnie od reprezentacji danych takie wyszukiwanie będzie możliwe, chociaż oczywiście skonstruowanie odpowiedniego zapytania może być skomplikowane. Przetłumaczenie zapytania TQL na zapytanie w języku odpowiednim do reprezentacji danych, np. SQL, XQuery, jest zadaniem programu tłumaczącego. Dzięki temu praca polegająca na przetłumaczeniu zapytania z "języka dziedziny" na "język komputerów" jest wykonana tylko raz dla każdego sposobu reprezentacji danych, i to przez programistów, a nie sumerologów.

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 |

| Opis | Nazwa pola w TQL |
|---|---|
| treść (odczyty) | text |
| kolekcja | collection |
| muzeum | museum |

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

W kolejnych dwóch rozdziałach przedstawimy gramatykę i semantykę zaprojektowanego przez nas języka TQL.

## 2.2. Gramatyka

W tym rozdziale przedstawimy gramatykę zaprojektowanego przez nas języka. W pierwszej części pokażemy strukturę leksykalną TQL, czyli elementy, z których buduje się zapytania. W drugiej części zaprezentujemy reguły tworzenia zapytań zapisane w formie reguł gramatyki w notacji BNF.

### 2.2.1. Struktura leksykalna

#### Literały

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

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

Literał `SłowoOdLiczby` to ciąg liter, cyfr oraz znaków "`-`", "`'`", "`_`", zaczynający się od cyfry.

#### Słowa kluczowe

```
as      define  in
search
```

#### Znaki specjalne

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

### 2.2.2. Struktura składniowa języka

Poniżej przedstawimy reguły gramatyki TQL w notacji BNF. Nieterminale są zapisane pomiędzy "⟨" a "⟩". Symbole "::=" (produkcja), "|" (lub) i "$\epsilon$" (pusta reguła) należą do notacji BNF. Wszystkie pozostałe symbole to terminale.

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

⟨*Lista Zapytań*⟩ ::= ⟨*Zapytanie*⟩
| ⟨*Zapytanie*⟩ ⟨*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*⟩
| `search` ⟨*Nazwa*⟩ ⟨*Lista Pustych Linii*⟩
| ⟨*Lista Pustych Linii*⟩

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

⟨*Linia Zapytania*⟩ ::= ⟨*Nazwa pola*⟩ `:` ⟨*Wyrażenie*⟩

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

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

⟨*Wyrażenie2*⟩ ::= ⟨*Wyrażenie3*⟩ `*` ⟨*Wyrażenie3*⟩
| ⟨*Tekst*⟩
| `(` ⟨*Wyrażenie*⟩ `)`

⟨*Wyrażenie3*⟩ ::= ⟨*Wyrażenie2*⟩
| $\epsilon$

⟨*Lista Pustych Linii*⟩ ::= $\epsilon$
| ⟨*Pusta Linia*⟩ ⟨*Lista Pustych Linii*⟩

⟨*Pusta Linia*⟩ ::= `\n`

⟨*Tekst*⟩ ::= `String`
| ⟨*Słowo*⟩

⟨*Słowo*⟩ ::= `SłowoOdLitery`
| `SłowoOdLiczby`

⟨*Nazwa pola*⟩ ::= `SłowoOdLitery`

⟨*Nazwa*⟩ ::= `String`

Powyższa gramatyka jest gramatyką bezkontekstową.

## 2.3. Semantyka

Poniżej przedstawiamy semantykę wybranych przykładów.

### 2.3.1. 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”.

### 2.3.2. 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.

### 2.3.3. 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“.

### 2.3.4. Wywołanie zapytania zdefiniowanego

#### Zwykłe

```
search "zwierzęta w Gar*a"
```

Wynikiem zapytania będą dokładnie te tabliczki, które spełniają wszystkie warunki zapytania "zwierzęta w Gar*a".
Takie zapytanie jest równoważne następującemu zapytaniu prostemu:

```
  provenience: Gar*a
  period: Ur III
  text: "udu ban"/mash2
```

Zakładając, że *"zwierzęta w Gar*a"* są jak w sekcji 2.3.3.

#### 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".

Takie zapytanie jest równoważne następującemu zapytaniu prostemu:

```
  provenience: Gar*a
  period: Ur III
  text: ("udu ban"/mash2)+adad-tilati
```

Zakładając, że *"zwierzęta w Gar*a"* są jak w sekcji 2.3.3.

## 2.4. Przykład użycia TQL

Zaprezentujemy przykład problemu, który został rozwiązany za pomocą języka TQL.

Chcemy, w zbiorze tabliczek z okresu Ur III, dotyczących dostaw owiec (udu), odnaleźć te, które nie były sporządzone pod nadzorem adad-tilati. Przyjmujemy, że tabliczki, które były sporządzone pod nadzorem danego urzędnika zawierają jego imię, być może z jakąś końcówką.

W znanych nam wyszukiwarkach, np. na stronie CDLI nie da się tego zrobić - nie można wyszukać tabliczek, które nie zawierają podanego słowa. Natomiast w języku TQL takie zapytanie da się łatwo wyrazić, np.

```
text: udu --adad-tilati*
period: Ur III
```

Co więcej, można zapytanie

```
text: udu
period: Ur III
```

zapisać jako zapytanie zdefiniowane, a następnie wyszukiwać za jego pomocą tabliczki z odpowiednimi imionami lub bez, np.

```
define
  text: udu
  period: Ur III
as "owce w Ur III"

search
  text: --adad-tilati*
in "owce w Ur III"

search
  text: adad-tilati* + {d}inanna
in "owce w Ur III"
```

Takie rozwiązanie jest znaczącym ułatwieniem, ponieważ pozwala uniknąć wpisywania wiele razy podobnych zapytań.

# Rozdział 3

# Implementacja

Przy implementacji języka TQL zastosowałyśmy podejście polegające na tzw. preprocessingu – wzorzec Preprocessor [2]. W naszym przypadku polega to na przetłumaczeniu zapytania w języku TQL na zapytanie w innym, istniejącym już języku zapytań wyższego poziomu, a następnie skorzystaniu z przetłumaczonego w ten sposób zapytania. W tej części pracy przedstawimy sposób zaimplementowania programu tłumaczącego.

Jednym z głównych założeń języka TQL, szczególnie ważnym z punktu widzenia implementacji, jest niezależność od struktury danych. W związku z tym istotną cechą programu tłumaczącego zapytania w TQL (translatora) jest możliwość dostosowania go do współpracy z różnymi bazami danych. Wynikiem tego jest podział translatora na dwa rodzaje modułów (rysunek 3.1):

1. **podstawowe** – niezależne od struktury danych, zajmujące się głównie parsowaniem i analizą składniową zapytania;

2. **wymienne** – zależne od struktury danych, tłumaczące zapytanie z TQL na język odpowiedni dla używanej bazy danych i wywołujące je.

Na rysunku 3.2 przedstawiamy ogólnie strukturę systemu, który korzysta z translatora TQL. Taki system, oprócz modułów podstawowych translatora, zawiera bazę tabliczek w dowolnej formie, implemetację modułów wymiennych umożliwiającą korzystanie z tej bazy, oraz interfejs użytkownika – do wprowadzania zapytań i wyświetlania wyników.

W niniejszej pracy prezentujemy dwie prototypowe implementacje translatora zawierające własne zestawy modułów wymiennych: dla bazy PostgreSQL oraz XML. Wybór modułów wymiennych odbywa się na poziomie kompilacji. Jest to rozwiązanie najprostsze do zaimplementowania, jednak wymaga osobnego programu dla każdego rodzaju bazy danych.

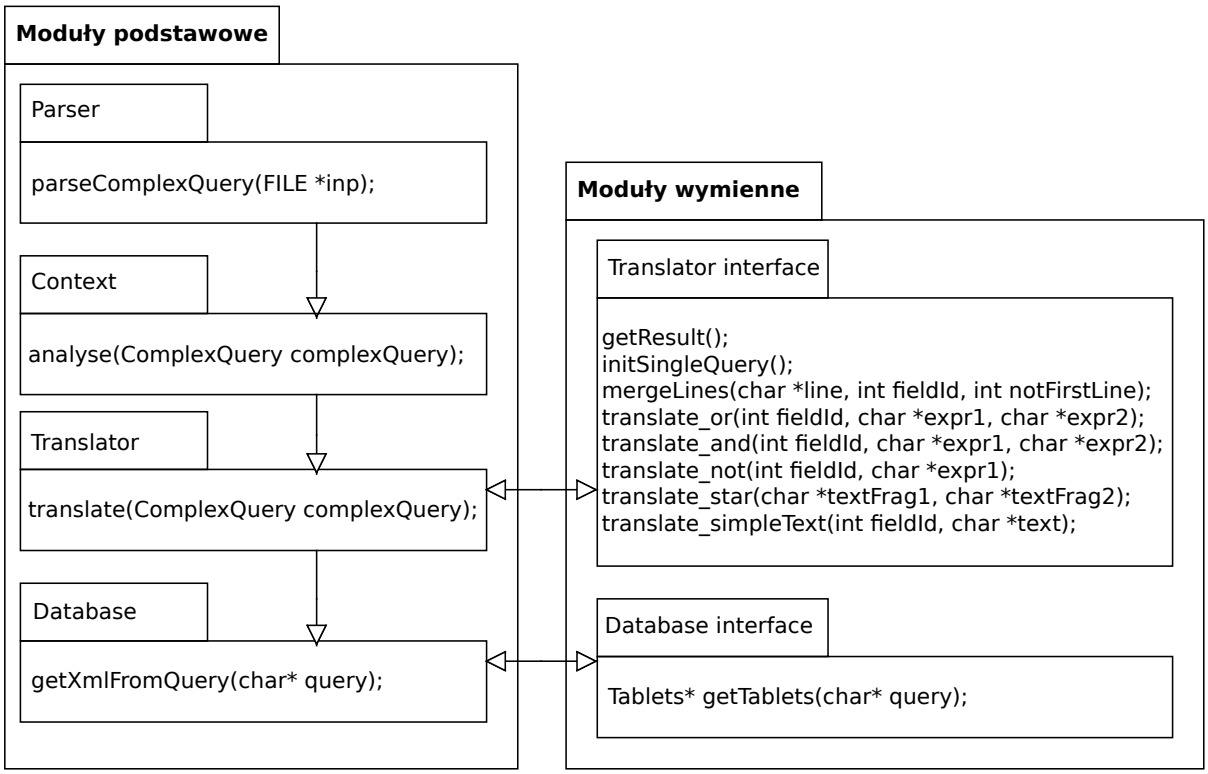
Stworzony przez nas Makefile domyślnie buduje obie implementacje translatora TQL.

## 3.1. Moduły podstawowe

W tym rozdziale przedstawimy dokładniej poszczególne moduły podstawowe translatora.

### 3.1.1. Parser

Parser został utworzony za pomocą narzędzia BNFC [10]. Na podstawie etykietowanej gramatyki BNF narzędzie to tworzy parser oraz szkielet analizatora składni w wybranym języku:




Rysunek 3.1: Podział programu na moduły

C, C++, C#, F#, Haskell, Java lub OCaml. BNFC tworzy również pliki wejściowe dla generatora leksera (np. Flex) oraz dla generatora parsera (np. Bison). Dodatkowym produktem jest dokument w formacie Latex, który zawiera specyfikację zaprojektowanego języka.

Po automatycznym utworzeniu parsera, zmieniłyśmy nazwy stałych oznaczających symbole na bardziej intuicyjne. Następnie dodałyśmy tablicę symboli, usunęłyśmy niepotrzebne funkcje z interfejsu i ogólnie uporządkowałyśmy kod.

Moduł parsuje zapytanie w języku TQL, tworząc drzewo struktury składniowej, które jest zdefiniowane w pliku pomocniczym Absyn.h.

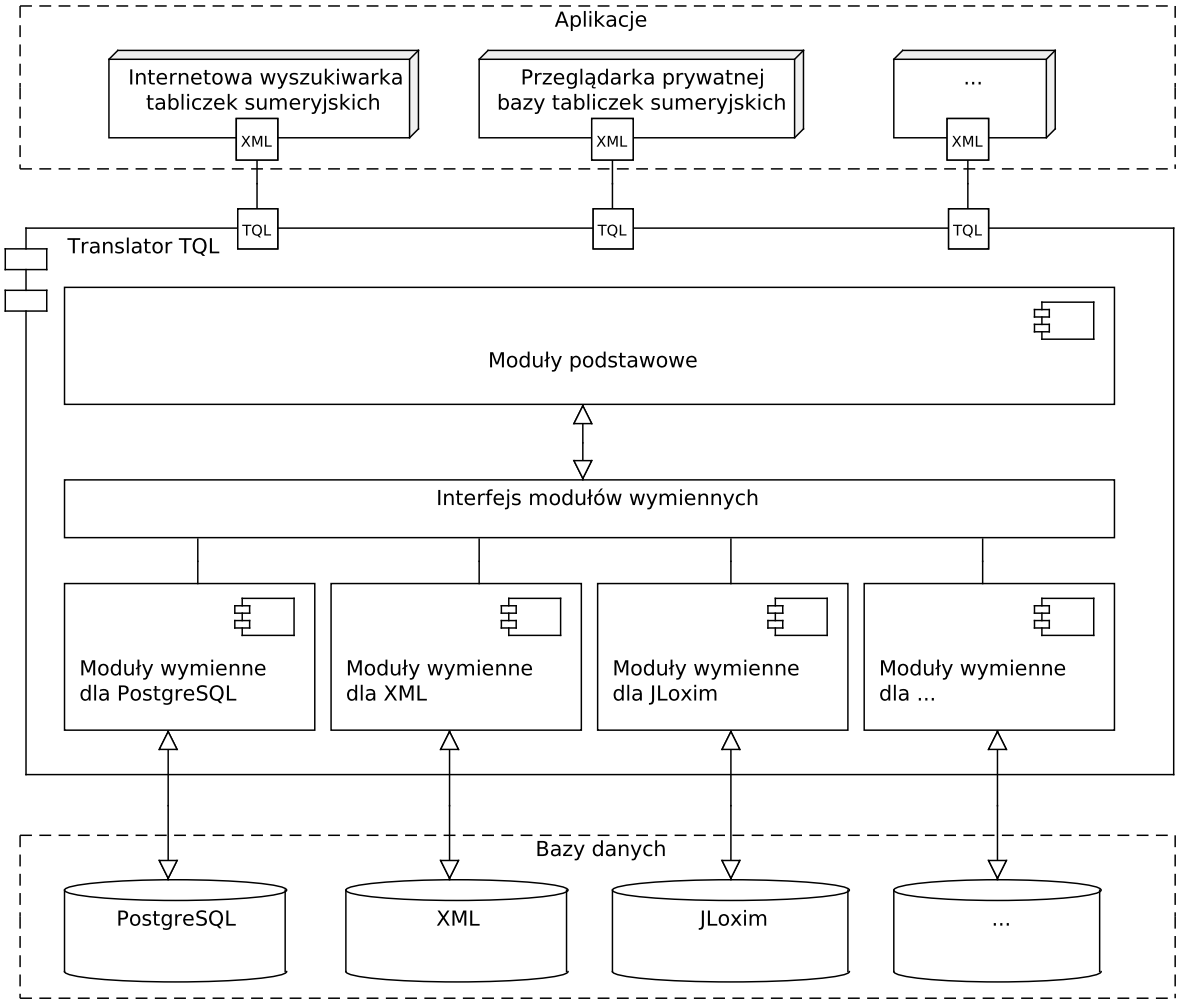
Główna funkcja tego modułu to `parseComplexQuery(FILE *inp)`. Argumentem jest wskaźnik do pliku zawierającego zapytanie TQL, a wynikiem odpowiednie drzewo struktury składniowej.

Na parser składają się następujące pliki:

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

### 3.1.2. Analizator kontekstowy

Podstawową funkcją w tym module jest `analyse(ComplexQuery complexQuery)`, której argumentem oraz wynikiem jest drzewo struktury składniowej zapytania TQL. Funkcja ta




Rysunek 3.2: Struktura systemu korzystającego z translatora

sprawdza, czy podano prawidłowe nazwy pól, na podstawie których następuje wyszkukiwanie, oraz upraszcza drzewo – z wywołania zapytania (wywołanie *search in*) tworzy zapytanie proste.

Analizator kontekstowy składa się z następujących plików:

- Context.cpp
- Context.h.

### 3.1.3. 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(ComplexQuery complexQuery)` przechodzi całą strukturę drzewa, wywołując w razie potrzeby odpowiednie funkcje z Translator_interface. Następnie pobiera przetłumaczone zapytanie za pomocą funkcji getResult() i przekazuje je jako wynik.

### 3.1.4. Baza

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

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

Główną funkcją w tym module jest `getXmlFromQuery(char *query)`, która wywołuje funkcję `getTablets(char *query)` z Database_interface.h, jako parametr podając przetłumaczoną treść zapytania. Dostaje w wyniku strukturę danych Tablets, wypełnioną informacjami o wyszukanych tabliczkach. Następnie na podstawie otrzymanej struktury tworzy dokument XML i przekazuje go jako wynik wywołania zapytania.
Struktura Tablets zawiera metadane tabliczki, jej treść oraz informację o frazach, na podstawie których tabliczka została znaleziona. Poniżej przedstawiamy definicję 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 - frazy wyszukiwania
} Tablet;

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

Zakładamy, że w bazie danych znajdują się informacje potrzebne do wypełnienia powyższej struktury (nie licząc fraz wyszukiwania).

### 3.1.5. Pliki pomocnicze

Definicje struktur danych i funkcji służących do budowy drzewa struktury składniowej (wygenerowane za pomocą BNFC [10], 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 [11]:

- Cexplode.cpp
- Cexplode.h.

## 3.2. Moduły wymienne

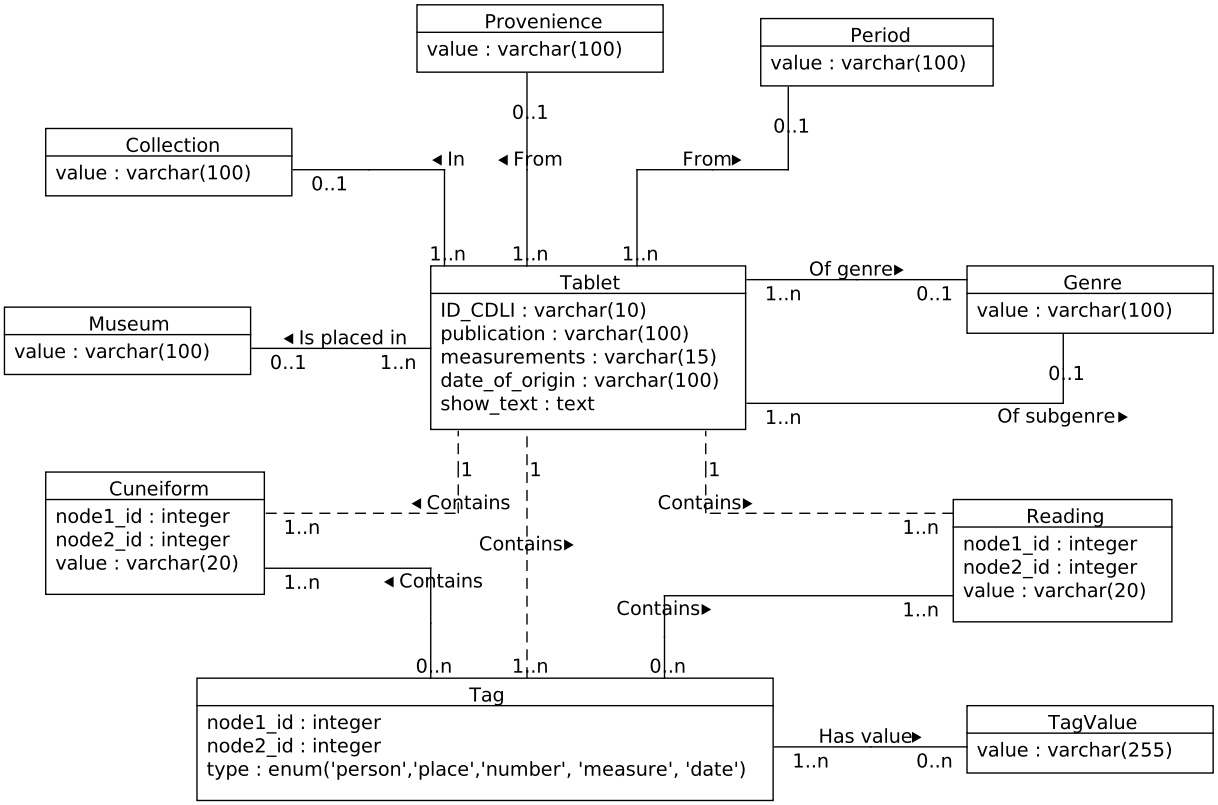
Pliki zależne od wyboru konkretnej bazy danych to:

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

Ich interfejsy są wspólne dla wszystkich baz danych.

### 3.2.1. Baza PostgreSQL

#### Diagram encji




Rysunek 3.3: Diagram encji

Jednym z problemów przy projektowaniu bazy danych był wybór takiej reprezentacji treści tabliczki, żeby efektywnie wyszukiwać zarówno treść konkretnej tabliczki, jak i tabliczki, których treść spełnia podane kryteria (wg odczytów lub zapisu klinowego).

W przypadku pierwszego problemu narzucające się rozwiązanie to przechowywanie treści jako otwarty tekst. Natomiast najlepszym rozwiązaniem drugiego jest reprezentacja treści tabliczki w formie grafu, którego krawędziami są odczyty i kliny (zgodnie z pomysłem dr Wojciecha Jaworskiego [12, s.374]). Zdecydowałyśmy się na połączenie obu sposobów. Odczyty oraz kliny przechowujemy w tabelach Reading i Cuneiform, natomiast otwarty tekst w kolumnie show_text tabeli Tablet.

Dzięki temu rozwiązaniu, szukając tabliczki zawierającej podane odczyty (lub kliny), korzystamy z reprezentacji grafowej w tabeli Reading (lub Cuneiform). Natomiast zawsze korzystamy z otwartego tekstu w tabeli Tablet w celu wyświetlenia treści tabliczki. Kiedy zatem wyszukujemy w tabeli Reading (lub Cuneiform), musimy być w stanie powiązać znalezione odczyty (kliny) z całą treścią tabliczki w tabeli Tablet. Osiągamy to, zapisując węzły w grafie w 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_postgres

Moduł *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'` |

| Konstrukcja | Tłumaczenie na SQL |
|---|---|
| genre: wartosc | `g1.value LIKE 'wartosc' OR g2.value LIKE 'wartosc'` |
| 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 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_sekw> 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 pseudokodzie:

**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<br>id_tab,<br>CAST(array_accum(nodes) as TEXT) as nodes,<br>COUNT(DISTINCT id_seq) as seq,<br><id_sekw> as id_seq<br>FROM<br>(<br><zapytanie1><br>UNION<br><zapytanie2><br>)<br>as c<br>GROUP BY id_tab |

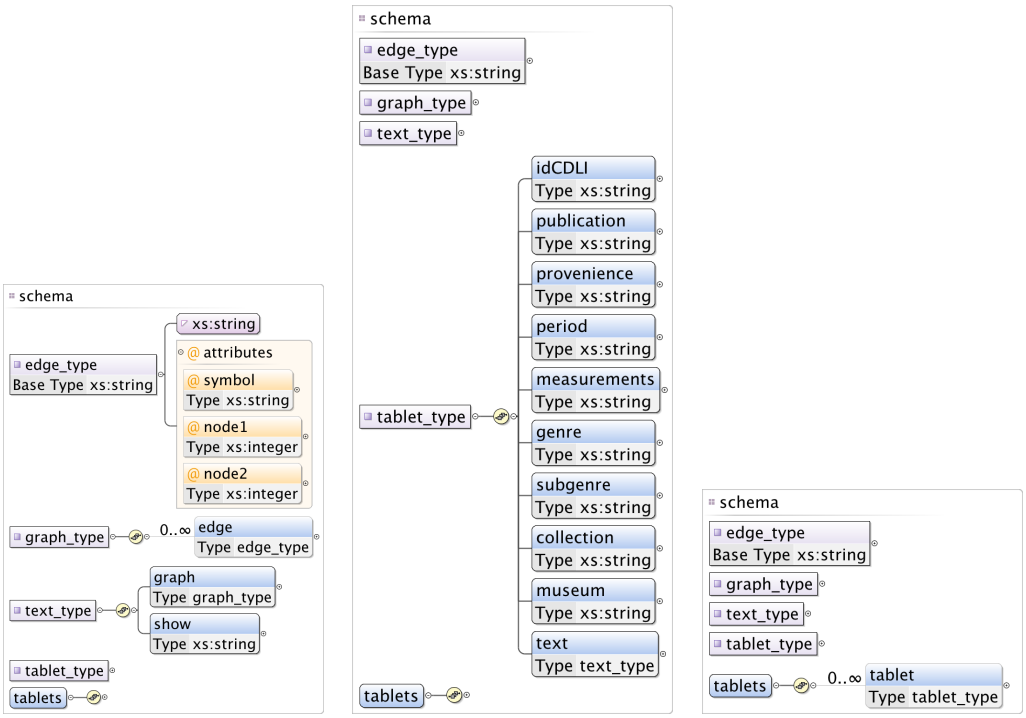
| Operator | Tłumaczenie |
|---|---|
| + | SELECT * FROM<br>(SELECT id_tab,<br>CAST(array_accum(nodes) as TEXT) as nodes,<br>COUNT(DISTINCT id_seq) as seq,<br><id_sekw> as id_seq<br>FROM<br>(<zapytanie1><br>UNION<br><zapytanie2>)<br>as c<br>GROUP BY id_tab<br>) as b<br>WHERE b.seq=2 |
| − | SELECT<br>id_tab,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

### Database_postgres

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

### 3.2.2. Baza XML

#### Schemat dokumentu




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

Rysunek 3.4: Schematy poszczególnych elementów XML
kolor pomarańczowy oznacza atrybuty, a niebieski elementy

Schemat dokumentu zamieszczony powyżej jest graficznym przedstawieniem dokumentu XML Schema (dodatek 3.2.2) wygenerowanym przez program Oxygen.

Jest on oparty, podobnie jak w bazie PostgreSQL, na pomyśle 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*.

#### Translator_xml

Do przeszukiwania dokumentu XML wykorzystywany jest język XQuery, będący 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($tablet/genre,'^wartosc$')`<br>`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 ','.

### Database_xml

Moduł *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 [5]. Posiada on API m.in. do C++, które pozwala na przekazanie zapytania do bazy oraz przetworzenie wyniku.

# Podsumowanie

## Podsumowanie projektu

Przedstawiona w niniejszej pracy realizacja języka TQL spełnia najważniejsze postulaty. Po pierwsze jest on intuicyjny i prosty w użyciu dla osób znających jedynie dziedzinę problemu. Po drugie minimalnie ogranicza siłę wyrazu, pozwalając na tworzenie skomplikowanych zapytań.

Język TQL ma jednak kilka ograniczeń w stosunku do typowych języków zapytań. Największym z nich jest brak wpływu na postać wyniku, co utrudnia np. zbieranie danych statystycznych (m.in. nie da się spytać o ilość tabliczek spełniających dane kryteria). Można jednak stworzyć narzędzia do samej prezentacji wyników zapytań, które pokonają to ograniczenie.

Warto również zastanowić się nad możliwością sortowania wyników zapytań. W obecnej wersji TQL moża to zrobić na poziomie interfejsu, jednak dodanie takiej funkcjonalności do języka pozwoliłoby na sortowanie bezpośrednio w bazie danych.

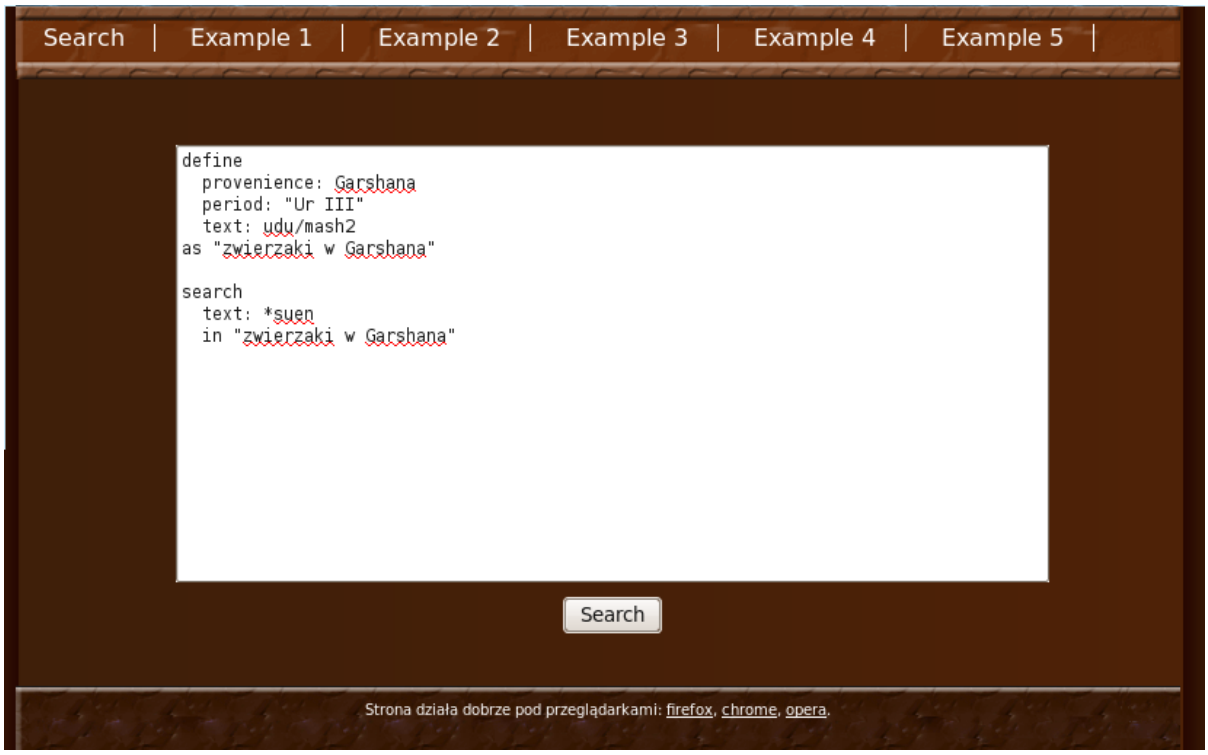
Należy pamiętać, że TQL jest jedynie językiem wyszukiwania – w przeciwieństwie do wielu innych języków zapytań nie udostępnia możliwości zmieniania danych znajdujących się w bazie, ani dodawania nowych.

Siła wyrazu języka TQL jest ograniczona w minimalnym stopniu. Przykładem zapytania, którego nie sa się w nim wyrazić, jest "znajdź tabliczki, które zawierają frazę zaczynającą się od słowa udu, której drugim słowem nie jest słowo ban". Ogólnie TQL pozwala tworzyć zapytania w oparciu o frazy, określone w całości lub częściowo, ale nie pozwala konstruować warunków dotyczących fragmentów fraz. Godzimy się na to ograniczenie, ponieważ najczęściej nie jest to potrzebne.

Dzięki specyficznym cechom języka TQL, można 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 zaprezentowane zostały dwie implementacje języka TQL. Po stworzeniu pierwszej, czas potrzebny na drugą był już niewielki. Największym zadaniem było przetłumaczenie poszczególnych konstrukcji TQL na XQuery. Pozwala to sądzić, że każda kolejna implementacja, dla różnych typów baz i różnych schematów danych, nie będzie wymagała dużego nakładu pracy.

## Możliwości rozwoju




Rysunek 3.5: Okno do wprowadzania zapytania

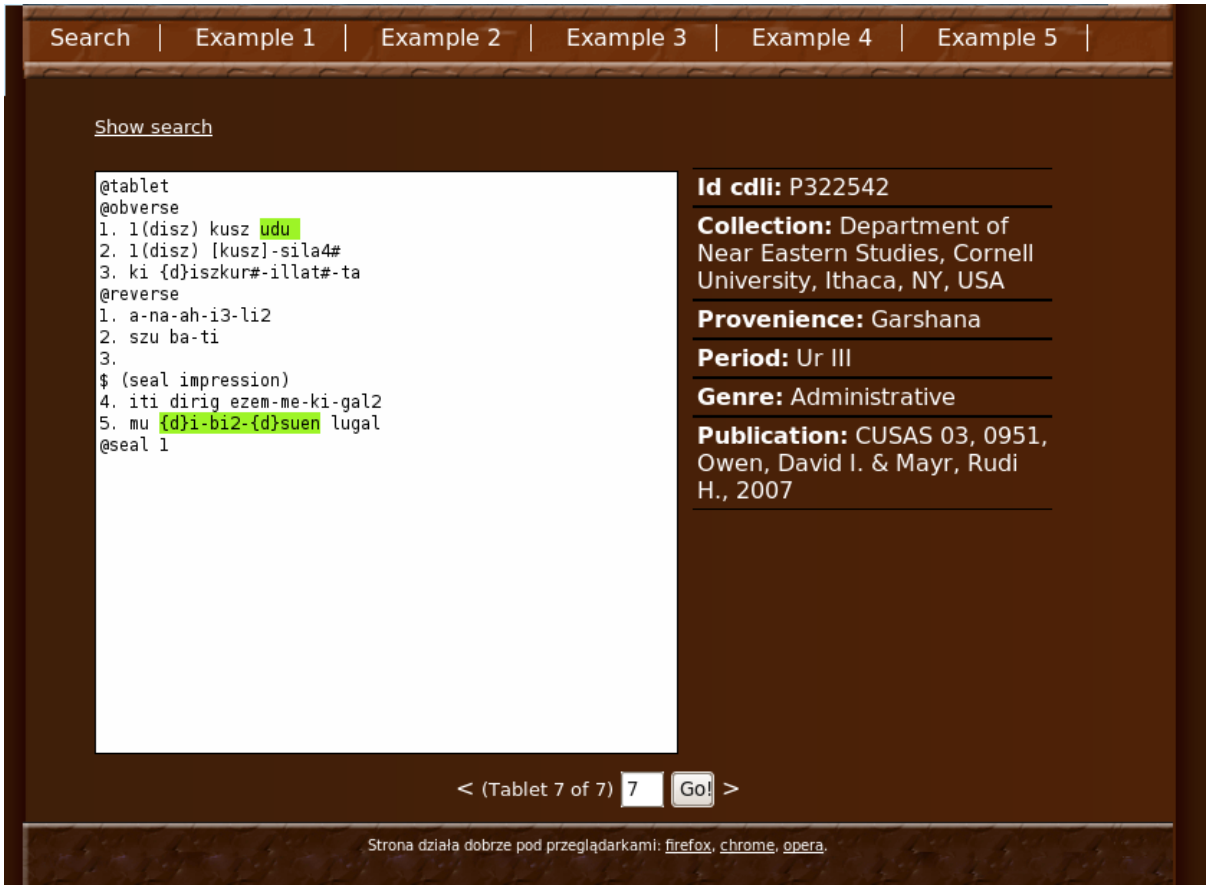
Jako dodatek do pracy stworzyłyśmy stronę internetową umożliwiającą zadanie zapytania w języku TQL i prezentującą wyniki. Jej wygląd jest przedstawiony na rysunkach 3.5 oraz 3.6. Stanowi ona zalążek przyjaznego interfejsu użytkownika dla naukowców, którzy będą współpracować z TQL. Plany jej rozwoju uwzględniają między innymi funkcjonalność graficznego budowanie zapytania. Na chwilę obecną serwis internetowy jest w trakcie udostępniania Wydziałowi Historii UW. Jego pracownicy będą pierwszymi testerami całego rozwiązania.

Poza tym warto nawiązać współpracę z projektem CDLI, (zob. rozdział 1.3.1). Implementacja TQL dla bazy CDLI, wraz z interfejsem WWW, z pewnością byłaby przydatnym narzędziem dla sumerologów na całym świecie.

W dalszej perspektywie można rozwinąć język TQL między innymi o wyszukiwanie wg klinów i wg tagów. Pierwsze z nich wymaga przetłumaczenia odczytów na kliny oraz umieszczenia tych informacji w bazie danych. Drugie natomiast potrzebuje narzędzia do wykrywania poszczególnych tagów (postaci, miar, dat, liczb itp) lub do ich zaznaczania przez sumerologów.

Dodatkowo warto stworzyć narzędzie do analizy uszkodzonych fragmentów i wykrywania błędów w odczytach tekstów. Dzięki niemu można by zmniejszyć ryzyko pominięcia istotnych tabliczek podczas wyszukiwania.

Rozwój projektu będzie zależał również od potrzeb zgłaszanych przez samych sumerologów.




Rysunek 3.6: Sposób prezentacji wyników zapytania

# Dodatki

## 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] M. Fowler, http://martinfowler.com/bliki/DomainSpecificLanguage.html

[2] M. Mernik, J. Heering, A. M. Sloane, *When and How to Develop Domain-Specific Languages*

[3] XQuery, http://www.w3.org/XML/Query/

[4] PostgreSQL, http://www.postgresql.org/

[5] Zorba - procesor XQuery, http://www.zorba-xquery.com/

[6] M. Kuckenburg, *Pierwsze słowo*, Państwowy Instytut Wydawniczy, Warszawa 2006, rozdział 8, 9

[7] A. Powałka, M. Stępień, J. Tyszkiewicz, *Automatyczna analiza dokumentów sumeryjskich*, Czas i przestrzeń, http://www.czasiprzestrzen.wuw.pl/?id=str,sumeryjskich,1,0

[8] Cuneiform Digital Library Initiative, http://cdli.ucla.edu

[9] The Electronic Text Corpus of Sumerian Literature, http://etcsl.orinst.ox.ac.uk/

[10] BNF Converter, http://www.ohloh.net/p/12800

[11] C-explode, http://maz-programmersdiary.blogspot.com/2008/09/c-explode.html

[12] W. Jaworski, *Contents Modelling of Neo-Sumerian Ur III Economic Text Corpus*, Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008), strony 369-376 Manchester, August 2008

# Spis rysunków