OBJECT ORIENTED REPRESENTATION AND PROCESSING OF GENOMIC DATA IN RELATIONAL DATABASE

Summary. The article shows a method of storing genomic data as the object in the relational database server. It presents a method of source data migration which is stored in the weakly determined text files on ftp servers. What is more, it describes the formal structure of the Common Language Runtime (CLR) class used to define user data type. Implementations of compulsory and optional methods are also presented. Furthermore, the paper shows a set of implemented matching algorithms and methods of using them to build adherence matrix. Finally, the paper presents some efficiency tests which prove the advantages of the proposed algorithms.

Keywords: genome, object oriented representation, relational database, searching methods, string matching

OBIEKTOWA REPREzentacja i przetwarzanie Danych Genetycznych w relacyjnej bazie Danych


Słowa kluczowe: genom, reprezentacja obiektowa, relacyjna baza danych, metody wyszukiwania, dopasowanie łańcuchów
1. Introduction

There are many methods of analysis of genome data for the different species described in the literature. However, most of them neglect methods of storing data. Authors use data written in text files with weakly determined structure, stored directly in the operating system. In most cases, they focus on improving performance due to the use of different hardware solutions: GPU [18, 19] or programmable arrays FPGA [20]. It is true that in the content of the publication there appears a phrase "non-redundant amino acid database" and reference to the BLAST tool, but in real it refers to a FTP server which stores compressed files [21]. The indices introduced in publications refer to the division of the input data into logical partitions, which are then processed on different cores. It gives the effect of parallelization of calculations. Another interesting approach may apply to partial trees and covariance [22], which is similar to Aho-Corasick algorithm. Although in this case, the author refers to the database, it is difficult to identify its structure. Whereas, in the paper [23] it is proposed to use indexing for k-mers (substrings of length k). They are similar to those, which are used for searching files in the operating system.

This approach can of course, give opportunity to develop a number of search methods, but does not allow the use of multiple properties offered by strong commercial database servers, such as: optimization of disk access or parallel processing by using physically partitioned tables. Additionally, the results of matching are not stored anywhere. Storage of these results allows us to use the classical mechanisms of databases for quick search, such as different types of indexes.

This approach can be mentioned in a few papers, [24] where the authors postulate to use MySQL database and classical SELECT queries. To obtain higher efficiency, they used the B-tree indexing. Additionally they pointed out the possibility of using other relational databases such as Oracle and PostgreSQL. The same database type was used in [25]. Authors indicated a significant decrease in processing efficiency above the certain level of the number of analyzed data, which is an immanent feature of the proposed environment. A comprehensive search system for codons proteins was built on the example of a single species (Trypanosoma cruzi) and it is described in [26]. The authors proposed an XML file containing already pre-processed data as a storage system. Queries are processed by using a dedicated process (framework RDF) which in the basic version is used to describe networks. In [27] there was proposed a comprehensive foundation of building a database in the relational model. However, the focus was mainly on the implementation of data mining task to analyze the risk of cancer. A mixed object-relational approach is postulated by the authors in [28]. It assumes the mapping of a single object to the one table of the relational schema.
It is worth emphasizing that it is widely thought databases are only intended for the storage of generic data. In fact, most database servers have implemented object-oriented representations of complex types for example vector geometry data (Spatial). Additionally, they offer the ability to create user defined object-oriented datatypes. Their implementation allows you to connect the properties of relational servers with features of object-oriented paradigm. This significantly increases the processing capabilities, especially in the context of performing complex analysis. These features led to the development of mechanism of genetic data integration and their storage in the object-oriented form, and methods for processing on the database server-side.

2. Genetic data and their storage

One of the most credible and the most powerful sources of the original genome data are the ones carried out within the project Ensembl (http://www.ensembl.org/). It is a joint venture of The European Bioinformatics Institute and the Sanger Institute. The project’s website provides i.a. genome data at ftp://ftp.ensembl.org/pub/. It has a multi-level folder structure, where data is stored in the form of compressed text files.

The SSIS (SQL Server Integration Services) package was created by the author for the automation of the process of retrieving data to the local hard disk with a flattening of the structure into a single directory. The main part of this was a C# script which contains the following steps:

- Connection type definition;
- Establishing a new connection instance;
- Recursively for each level of the directory structure:
  - Detection of directory names of the given level;
  - Establishing of the new connection instance;
  - Loading files from a given level of the operating system folder.

The local structure consists of a set of compressed files, which were automatically decompressed using subsequent C# script. The result is a set of text files with a complex weakly determined structure. These files contain a header whose content depends on the described species. This part of each file has to be omitted before insertion to the database data which contains DNA. To obtain entries in dictionary tables, the scheme of source filenames is used – species.typ.chromosom.txt.

Using such a naming rule, the package which fills tables of relational structure was built. It consists of the following steps:

- Establishing the connection with the directory storing text data;
- For each file, division of its name, to isolate the components of the description;
- Establishing a connection with the database server;
  - Checking the existence of entry for species, if the insert row does not exist, read species ID;
  - Checking the existence of entry for type, if the insert row with read before spices ID does not exist, read type ID;
  - Checking the existence of entry for chromosome, if insert row does not exist, read chromosome ID;
  - Checking the existence of entry for genome, if insert row with read before type and chromosome IDs does not exist, using data streaming;
    - Creating of an asynchronous instance for streaming data from file to the server;
    - Opening connections in await mode;
    - Implementing of the data transmission in await mode;
    - Closing connection and release resources of the streaming instance.

In the flowchart presented below (Fig. 1), the need for using streaming transmission is worth noting down. Because of large volume of files, in the simple transmission we obtain interruption of recording due to the timeout. Streaming with forced await option is the only effective solution to migrate very large volumes of data in a single INSERT operation.

![Relational scheme a of database for storing genome data](image)

**Fig. 1.** Relational scheme a of database for storing genome data

**Rys. 1.** Relacyjny schemat bazy danych do zapisu danych genetycznych

The goal of all steps is to fill tables of simple relational structure shown in Fig. 1. This does not concern only GenomeBin column that contains genome data in object-oriented form. This type will be described in the next section. It should be noted down, that the column Genome containing character data is added only to verify the system. In the final implementation it may be deleted without any detriment.

To improve performance, physical partitioning of tables has been applied. For the used database platform, it is possible to establish a division to maximum of 8 physical files. If they are placed on separate physical devices (disks) it allows significant parallelization of read and write operations.
3. Implementation of object oriented paradigm in relational databases

Most commercial database servers allow the use of object oriented languages to create items that meet the object oriented paradigm. These classes can be mapped to the procedural elements of these servers. This applies to: Oracle [16], PostgreSQL and MS SQL Server [17]. In each of these environments it is possible to define:

- store procedures;
- functions;
- aggregate functions;
- triggered procedures (triggers);
- object oriented, user defined datatypes.

I concentrate on the syntax which is used in the work and was implemented on server from Microsoft.

The most important, from the point of view of the issues presented in the paper, is user defined datatype. Its creation requires to define structure (abstract class), which must include:

- compulsory properties;
  - set of properties that describe the object, using generic or enumerated types;
  - `Is_Null` property returns `true`, if the object has no value, otherwise `false`;
- obligatory methods;
  - accessors allowing access to the values of the structure (i.e. getters, setters)
  - `Parse()` – method that allows for transcription of input data, whose type must not match the internal representation, to the destination type;
  - `ToString()` – method that displays the internal representation of data as the text, it is relevant especially when the object represents collection;
  - `IsNull` – method that returns current value of `Is_Null` property;
  - method which validates input data, method’s name is a parameter in the declaration for compiler .NET;
  - `Write` – mandatory when at least one of object properties is not generic, provides a record of properties to the binary representation (i.e. serialization);
  - `Read` – mandatory when at least one of object properties is not generic, provides a read of properties from the binary representation (i.e. deserialization);
- optional methods, not mandatory (additional methods);
  - any number of methods which operate on the object oriented data, defined by this structure.
Created structure must be compiled into a library *.dll. The next step is to create on the database side the so called assembly. It constitutes the representation of the compiled class object on the side of MS SQL Server.

CREATE ASSEMBLY AssemblyName
FROM 'C:\Path to file\file.dll'
WITH PERMISSION_SET = SAFE;

The last element is creation of user defined type

CREATE TYPE TypeName
EXTERNAL NAME AssemblyName.StructureName;

The type defined this way can be used on the same rules as all other predefined types (variable declarations, define the columns in the table). An additional advantage is the possibility of referring on the database side to their properties and methods, as in any object oriented programming languages.

4. Representation of the genome in a relational database

4.1. Object representation of the genome

Object structure of the genome contains one property that represents the alphanumeric notation of the type SQLString, which is equal to the nvarchar(max) on the server side. Validation function called ValidateGen() checks, using regular expressions, whether description of nucleotides takes one of the permitted values: A – adenine, C – cytosine, G – guanine, T – thymine (for DNA), U – uracil (for RNA) or N (nucleotide has been not sequenced, the value of unknown).

In optional methods, function Size() that determines the number of nucleotides in the genome was developed. Its primary task is to verify the correctness of data migration to the object oriented representation.

4.2. Pattern matching

Alphabet \( \Sigma \) is called arbitrary, non-empty set of unique elements. Elements of the alphabet \( a_i \) are called characters or symbols.

$$\Sigma = \{a_i\} \quad \forall i, j, i \neq j, a_i \neq a_j$$  \hspace{1cm} (1)

A word \( \nu \) over the alphabet \( \Sigma \) we call any sequence of elements of the alphabet. The word that contains an empty string elements was denoted as \( \varepsilon \). The pattern matching task consists of checking whether the word \( \nu \) of length \( n \) is the subword of word \( w \) of a length
Object oriented representation and processing of genomic data in relational database

$l \geq n$, that is, whether there is such $k \leq k+n-1 \leq l$, that $w_k = v_0, w_{k+1} = v_1, \ldots, w_{k+n-1} = v_{n-1}$.

The way to search, using the above definition, is inefficient because in order to obtain all possible matchings it needs at least $l-n$ comparisons. In extreme cases, this number may reach $(l-n)n$. There exist many algorithms that improve efficiency of pattern matching task.

- Rabin Karp algorithm [1] involves comparing the value derived from the application of any chosen hash function $h$ instead of characters, for example Rabin's function defined as

$$h = x^2 \mod N, \quad x < N$$  \hspace{1cm} (2)

applied to the subpatterns of length $n$. Directly compared are only these substrings, for which the values of the hash function are equal. The efficiency of this algorithm is strongly dependent on the computing complexity of hash function and it is estimated to be $O(n+l)$.

- Knuth Morris Pratt algorithm [2] uses information about the first pattern position incompatible with the search string in such a way that next matching attempt ignores all compatible values. This means that each time $m$ compatible characters are compared the sample next matching begins with $m+l$.

- Boyer Moore algorithm [3] is analogous to the Knuth Morris Pratt algorithm, but the comparison is performed from the last character of the searching string. The result is that in the absence of conformity of the $i$-th character counted from the end, comparison position can be shifted by the length of the matching string $n$.

- Horspool Algorithm [4] is a simplification of Boyer Moore algorithm. In the first step there is preprocessing performed, wherein the number of letters of the alphabet used in the string is counted. Then the characters and the reference string are compared using the data stored in the dictionary that was obtained in the first phase. In analogy to the basic Boyer Moore algorithm, comparison begins by the last character in the analyzed string.

- Apostolico Giancarlo Algorithm [5] is the next variant of the Boyer Moore algorithm. It preserves the principle of comparing characters from the end of the matched text and the method of its movement relative to the pattern. Improvement of search efficiency is obtained by storing this data in the auxiliary table.

All the above described algorithms have been implemented as the optional methods in the object-oriented datatype that represents the genome. In addition, $Find()$ method, which uses the internal function $IndexOf()$ was implemented as a reference method. Because it finds only the first occurrence of the string in the text, it was extended to the function $FindMulti()$. It periodically calls basic function to find all occurrences of the search phrase.

Of course, there are much more pattern-matching algorithms known from the literature e.g.: Sanday [15], Galil-Giancarlo, Apostolico-Crochemore, Colussi, Zhu-Takaoka, Galil-
Seiferas and many others. Probably some of them will also be implemented in object-oriented datatype, not just because of researches on efficiency, but as additional reference methods, which validate the results obtained so far. The greatest hopes are connected with Sunday algorithm, which is described in literature as one of the highest performance.

The object-oriented type created, may be tested on database server-side by assigning values to variables and execution of a simple selecting query. The length of the variable @a, in which we are searching has been deliberately shortened.

DECLARE @a GenomT, @b GenomT
SET @a =CAST('TGGTACGGGCCCTTACGGGCAGGCAAGGTA' AS GenomT)
SET @b =CAST('CGGGC' AS GenomT)
SELECT @a.FindMulti(@b), @a.Rabin_Karp(@b), @a.Knuth_Morris_Pratt(@b),
@a.Boyer_Moore(@b), @a.Horspool(@b), @a.Apostolico_Giancarlo(@b)

Adding to the query in FROM clause the Genomes table as the source and replacing variable @a column name Genomes we can solve pattern matching task in all records.

5. Searching of sequences and construction of the mapping matrix

Geneticists in numerous researches point out some nucleotide sequences fulfilling a particular role in genome. They are often called cassettes or promoters, and play a major role in the initiation of the transcription process. They indicate the places where protein is encoded. The list of the most important of them, with references to the literature describing in more detail their functions is presented in Table 1.

<table>
<thead>
<tr>
<th>Sequence_Id</th>
<th>Name</th>
<th>Genome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chi sequence [8]</td>
<td>GCTGGTGG</td>
</tr>
<tr>
<td>2</td>
<td>Pribnow-Schaller [9]</td>
<td>TATAAT</td>
</tr>
<tr>
<td>3</td>
<td>Kozak strong [10]</td>
<td>ACCAUUGG</td>
</tr>
<tr>
<td>4</td>
<td>Kozak [10]</td>
<td>GCCAUUGG</td>
</tr>
<tr>
<td>6</td>
<td>TATA sequence [9]</td>
<td>TATAAA</td>
</tr>
<tr>
<td>7</td>
<td>Palindrome 1[12]</td>
<td>AGGCGCT</td>
</tr>
<tr>
<td>8</td>
<td>VNTR 1</td>
<td>ACTACT</td>
</tr>
<tr>
<td>10</td>
<td>B recognition element 2</td>
<td>GGACGC</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Most of them are simple, relatively short sequences of nucleotides appearing repeatedly in the genetic code. Finding their occurrence in a single chromosome does not pose major problems. The references to their position in the analysis of genetic data, because of their
characteristics will be relatively frequent. So it is worth storing results of searching in additional database table, whose structure is shown in Table 2.

Table 2

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type/ Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check_ID</td>
<td>Int PRIMARY KEY</td>
</tr>
<tr>
<td>Sequence_ID</td>
<td>Int FOREIGN KEY</td>
</tr>
<tr>
<td>Genome_ID</td>
<td>Int FOREIGN KEY</td>
</tr>
<tr>
<td>Position</td>
<td>Int</td>
</tr>
</tbody>
</table>

Data from the mapping table can be used for logical division of genomic data and comparisons of fragments encoding proteins. The natural solution to insert data into this table would be to use triggers for tables Genomes and Sequences, however, such a solution seems to be inefficient. The trigger constructed in the first table slows down long time of load of genome data. In the case of feeding the second table would force search all saved genomes. It will lock most of the resources and slow down access to the database. Therefore, the most reasonable solution seems to be the use of a separate process launched via Scheduler.

6. Efficiency tests

To check correctness of the implemented methods, the results of string matching for all of them using some of the promoters and representative set of genome data were compared. But more interesting was the verification of their efficiency. The test was based on measuring of query execution times for matching each from cassettes (Table 1) in sets of records, whose number was changing from 100 up to 800.

As the result minimum, maximum and average execution time was taken into account, for each records number of genome data. In the first test, conducted on machine with 16GB RAM fast increase of maximum time starting from number of records equal 700 was observed. So it was repeated on computer with 32GB RAM. In this case the phenomenon disappeared (Fig. 2).

It is necessary to point out, that the minimum and maximum for different matching sequences during increase number of analyzed records were obtained, too. Despite the fact that, this test was performed using environment distant from the target one, satisfactory results were obtained.

To prove high efficiency of usage, the user defined datatypes and methods defined inside them were copied to separated classes and mapped to Transact SQL functions. In analogous conditions tests for functions efficiency were conducted as before for CLR types. The times of their executions were about 3 times longer (Fig. 3).
The next checked feature was the dependence of execution time on matching string length. This test has to be realized separately because the length of promoters is relatively short, but in other cases the user may wish to work with longer sequences. The basic long sequence was obtained by copying a group of nucleotides which appeared after Chi sequence (include it) from one of real chromosomes. To perform this test, the string was divided into
shorter parts, which start from its beginning and have the length from 10 up to 100 elements. This numerical experiment proves that time of execution is practically independent on the matched text length (Fig. 4).

![Fig. 4](image.png)

**Fig. 4.** Influence of the length of the searching sequence on the time of query execution

**Rys. 4.** Wpływ długości wyszukiwanej sekwencji na czas wykonywania zapytania

### 7. Conclusion and future works

The main goal that prevailed when creating the solution was to verify the possibility of using object types for storage and processing of very large data. It involved the development of the means of storage, as well as building the mechanisms of gathering, transmission, and data load. These tasks were successfully realized, despite the use of a computer with relatively weak parameters. The same conclusion applies to the verification of the use of many algorithms for pattern-matching. The performed efficiency tests have confirmed the effectiveness of the proposed approach.

The use of mapping table for storing results of matching for nucleotides sequences (promoters), proposed in the paper, leads to effective, logical partitioning of genetic data. It gives significant improvement of analysis efficiency by using the indexes.

The next stage expected to be implemented is the migration of database to a larger, more efficient hardware environment, which will allow for the storage and processing of more numerous set of data. Finally, the development of simple, lightweight client application, can give access to the environment for researchers dealing with genetic data.
BIBLIOGRAPHY

Omówienie

Artykuł prezentuje implementację systemu przeznaczonego do składowania i przetwarzania danych genetycznych za pomocą obiektowych typów użytkownika z zastosowaniem silnego, komercyjnego serwera baz danych. Przedstawiona została krytyczna analiza rozwiązań opisanych w literaturze. Wskazano na bardzo małą liczbę opracowań, w których przetwarzanie jest wykonywane po stronie serwera. Większość z rozwiązań literaturowych wykorzystuje pliki przechowywane bezpośrednio w folderach systemu operacyjnego.

Praca przedstawia metodę migracji i integracji danych pochodzących z serwera ftp, który oferuje wiarygodne dane genetyczne wielu organizmów. W celu pobrania oraz zapisania do lokalnej struktury danych wykorzystane zostały pakiety SQL Server Integration Services zawierające skrypty napisane w języku C#. Kolejnym krokiem było automatyczne rozpakowanie skompresowanych plików oraz wyizolowanie tej ich części, która zawiera czyste dane genetyczne. Tak jak poprzednio wykorzystano pakiety MS SQL SSIS. Korzystając z logicznej struktury nazw plików danych, przygotowano kolejny pakiet, który sprawdza istnienie właściwych wpisów w relacyjnej strukturze tabel (rysunek 1), a w przypadku ich niewystępowania dokonuje właściwych operacji wstawiania danych. Aby zapobiec przekroczeniu czasu operacji podczas zasilania tabeli bardzo dużym wolumenem danych, zastosowano transmisję strumieniową z ustawioną opcją oczekiwania na potwierdzenie zakończenia transmisji.

Praca zawiera również opis zasad tworzenia klasy obiektowej, która jest przekształczana do postaci typu użytkownika, a który może być wykorzystywany po stronie relacyjnego serwera baz danych. Wskazano na obligatoryjne właściwości i metody, omawiając ich rolę w procesie przetwarzania danych. Przedstawiono ponadto miejsce i sposób dodawania metod fakultatywnych do tak skonstruowanego obiektu. Wskazano również na dwukrokowy sposób odwzorowania struktury na typ użytkownika.

Zdefiniowano pojęcie alfabetu (1) oraz przedstawiono zadanie dopasowania do wzorca. Wskazany został bardzo szeroki wybór algorytmów, mających na celu znaczną poprawę efektywności wyszukiwania. W sposób syntetyczny przedyskutowane zostały algorytmy: Rabina-Karpa, Knutha-Morrissa-Pratta, Boyera-Moore’a, Horspoola, Apostolico-Giancarlo. Wszystkie opisane metody zostały zaimplementowane w klasie obiektowej oraz poddane walidacji z wykorzystaniem wbudowanej funkcji IndexOff(). Przedstawiony został prosty skrypt pozwalający na przeprowadzenie sprawdzenia poprawności omówionych rozwiązań. Wykonana została walidacja otrzymanych wszystkimi metodami wyników zarówno dla prostego, statycznego łańcucha wejściowego, jak również dla reprezentatywnej grupy rekordów zawierających rzeczywisty genom.

Schemat relacyjny został wzbogacony o tabelę słownikową, zawierającą najważniejsze opisane w literaturze sekwencje (tzw. kasety, startery) – tabela 1, wskazujące na miejsca,
w których dokonywane jest rozplecenie helisy w procesie rekombinacji. Dodana została również tabela zawierającą informacje o wykrytych poprzednio dopasowaniach – macierz przyлегania – tabela 2. Zaproponowane rozwiązanie oferuje wiele pozytywnych cech, głównie związanych z poprawą efektywności wyszukiwania. Wiąże się to z możliwością używania dobrze znanych w bazach danych metod wynikających ze stosowania indeksów.

Przedstawione zostały wyniki testów wydajności. Pierwszy z nich polegał na pomiarze czasu wyszukiwania każdego z łańcuchów (starterów) pokazanych w tabeli 1 w zestawach rekordów o liczebności od 100 do 800 opisujących genom. Test przeprowadzono dla maszyny o 16 GB RAM. Wyznaczono najdłuższy, najkrótszy oraz średni czas dla każdej z długości zestawu rekordów i zaobserwowano znaczny wzrost najdłuższego z czasów po przekroczeniu 600 rekordów (rys. 2). Test powtórzono dla maszyny o większej pamięci 32 GB. W tym przypadku zjawisko nie pojawiło się, co świadczy o znacznym wpływie zasobów na wydajność procesu. Należy zaznaczyć, że czasy maksymalne i minimalne były uzyskiwane dla różnych wyszukiwanych sekwencji, co świadczy o słabym wpływie parametru wyszukiwania na czas uzyskiwania wyników. Kolejny test, mający na celu potwierdzenie wydajności stosowania typów obiektowych, polegał na wykorzystaniu funkcji Transact SQL. Podstawą definicji funkcji były metody wykorzystane w typie obiektowym, tym razem mapowane na element proceduralny. Otrzymane czasy wyszukiwania są około trzy razy dłuższe niż w przypadku metod typu obiektowego (rys. 3), co potwierdza założenie dotyczące wydajności. Trzeci test polegał na dopasowaniu łańcuchów nukleotydów o zmieniającej się od 10 do 100 długości w zmieniających się jak poprzednio zestawach rekordów opisujących genom. Zaobserwowano brak wyraźnej zmiany czasu wyszukiwania wraz ze zmianą długości dopasowywanej frazy (rys. 4).

W końcowej części pracy syntetycznie podsumowano otrzymane rezultaty oraz wskazano na potencjalne kierunki rozwoju tworzonego systemu.

Adres

Adam PELIKANT: Politechnika Łódzka, Instytut Mechatroniki i Systemów Informatycznych, ul. Stefanowskiego18/22, 90-924 Łódź, Polska, adam.pelikant@p.lodz.pl.