

České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

**Návrh a prototypová implementace databáze pro snadnější
práci se strukturami nukleových kyselin**

Bc. Ondřej Čečák

Vedoucí práce: Ing. Michal Valenta, Ph.D.

Odborný konzultant: Mgr. Daniel Svozil, Ph.D.

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný

Obor: Výpočetní technika

květen 2011

“Each of the cells in your body carries about 1.5 gigabytes of genetic information, an amount of information that would fill two CD ROMs or a small hard disk drive. Surprisingly, when placed in an appropriate egg cell, this amount of information is enough to build an entire living, breathing, thinking human being. – David S. Goodsell”

Acknowledgments

I would like to thank everyone who helped me improving this text – for support, many corrections, valuable pieces of advice and patience. I thank namely my parents Šárka and Pavel Čechákovi, my sister Barbora Čecháková, my pampering fiancée Martina Matějková, my supervisors Michal Valenta, Dan Svozil and Petr Čech. I also thank my friends Aleš Hakl and Kateřina Srnská. Finally, I thank staff of the Study Office at my alma mater.

Prohlášení

Prohlašuji, že jsem svou magisterskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám žádný důvod proti užití tohoto školního díla ve smyslu § 60 zákona § 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 13.5.2011

.....

Abstract

PostgreSQL is one of the most powerful open-source Database Management Systems. It supports a wide variety of native data types. This thesis describes PostgreSQL DBMS, design and implementation of the user database supporting complex queries on nucleic acids structure including the related data population tool and the prototype web application for demonstration and easy usage. This database based on standard PostgreSQL allows to work with sophisticated bioinformatics data simply.

Abstrakt

PostgreSQL je jedním z nejsilnějších open source systémů řízení databáze. Podporuje širokou škálu vlastních datových typů. Tato práce popisuje SŘDB PostgreSQL, návrh a implementaci uživatelské databáze s podporou komplexních dotazů na strukturu nukleových kyselin včetně souvisejícího nástroje pro import dat a prototypové webové aplikace pro snadnou použitelnost a demonstranci databáze. Tato databáze postavená na standardním PostgreSQL umožňuje jednoduše a rychle pracovat se sofistikovanými bioinformatickými daty.

Contents

List of Figures	xiii
1 Introduction	1
2 Problem Description and Target Specification	2
2.1 Nucleic Acids and Its Commonly Used Parameters	2
2.1.1 First Discovery of Nucleic Acids and DNA	2
2.1.2 Basic Terminology	2
2.2 PostgreSQL: The World's Most Advanced Open-Source Database	3
2.2.1 Brief History	3
2.2.2 Features	4
2.3 Aim of the Thesis	4
3 Basics of Nucleic Acid Structure Analysis	5
3.1 Basic Blocks	5
3.2 Structure	5
3.2.1 Base Pairing	5
3.2.2 Topology	6
3.2.3 Base Pair Geometry	7
3.3 Data Sources and It's Usage	9
3.3.1 Protein Data Bank	9
3.3.2 3DNA, Analysis of Three-dimensional Nucleic Acid Structures	10
3.3.3 Jmol, an Open-source Java Viewer for Chemical Structures in 3D	11
3.4 Use Cases	12
4 Analysis of Nucleic Acids Using RDBMS	13
4.1 Basics of Database Management System Usage	13
4.1.1 Data Redundancy Elimination – Normalization	13
4.1.2 Creating and Execution Queries on the Data	13
4.1.3 Search Speed	14
4.2 Relation Database Systems for the PDB	14
5 Specification of PostgreSQL Database	15
5.1 Support Tools Needed	15
5.2 List and Description of Requested Features	15
6 Application Analysis and Design	17
6.1 Application Components	17
6.2 Database Population Tool	17
6.2.1 Data Download	17
6.2.2 Analysis of Downloaded Data	18
6.2.3 Needed Categories from Dictionary <code>mmcif_pdbx.dic</code>	19
6.3 Database	25
6.3.1 Database Entities	25
6.4 Web Application	30
6.4.1 Requirements	30
6.4.2 Use Cases	30
6.4.3 Performing Queries	30
6.4.4 Viewing Results	31

6.4.5	Structure Details	31
6.4.6	Deleting Entries	31
6.4.7	Importing List of Structures	31
6.4.8	Checking Progress and Results of Import	32
6.4.9	Exporting Structure Data to CSV	32
6.4.10	Design of User Interface	32
6.5	Another Possibilities	33
7	Implementation	39
7.1	Database Definition	39
7.1.1	Data Types	39
7.1.2	Sequences	40
7.1.3	Indexes	41
7.1.4	User Data Types	42
7.1.5	Special Data Types	42
7.2	Data Import	42
7.2.1	Software Requirements	43
7.2.1.1	mmLib	43
7.2.1.2	Psycopg2	43
7.2.1.3	3DNA	43
7.2.2	Implementation	43
7.2.2.1	Connection to Database	44
7.2.2.2	Data Download	44
7.2.2.3	mmLib Data Analysis	44
7.2.2.4	3DNA Data Analysis	45
7.3	Prototype Web Application	51
7.3.1	Software Requirements	51
7.3.2	Implementation	52
7.3.2.1	Overall Application	52
7.3.2.2	JavaScript Form	52
7.3.2.3	Database Queries	53
7.3.2.4	Jmol Usage	55
7.3.2.5	Background Imports Queue	57
7.3.2.6	Background Imports Worker	57
7.3.2.7	CSV Export	58
7.4	Another Possibilities	59
8	Verification	60
8.1	Overview Before Test	60
8.2	Import of Nucleic Acids	60
8.2.1	Verification Script	61
8.2.2	Errors	62
8.2.3	Import Results	63
8.3	Testing	64
8.3.1	Search Results Verification	64
8.3.2	Technology Requirements	64
8.3.3	Speed of Search	64
8.3.4	Usability Testing	66
9	Conclusion	67
9.1	Future Development	67

A Bibliography	69
B Selected Content of Covered DVD	74
C Database Scheme and Structure	75
C.1 Entity-Relation Diagram	75
C.2 Structure in Data Definition Language	80
D Installation Manual	87
D.1 Quick Start	87
D.2 Detail Installation Instructions	87

List of Figures

2.1	“Central dogma” of molecular biology/genetics in typical scheme.	2
3.1	Chemical structure of DNA/RNA bases.	5
3.2	An <i>AT</i> base pair demonstrating three intermolecular hydrogen bonds.	6
3.3	A <i>GC</i> base pair demonstrating two intermolecular hydrogen bonds.	6
3.4	Numbered ribose carbons on a nucleotide, a base is attached to a ribose ring. .	6
3.5	Chemical structure of RNA with drawn 5'-to-3' direction.	7
3.6	Illustration describing base pair geometry parameters.	8
3.7	Reference frame for idealized helical DNA.	9
3.8	Jmol Visualization of Transposase TC3A1-65 from <i>Caenorhabditis Elegans</i> , PDB ID 1TC3.	12
3.9	Basic Use Case for nucleic acids researcher.	12
6.1	Activity diagram of the database population tool.	24
6.2	Entity: Biomolecule.	25
6.3	Entity: Chain.	25
6.4	Entity: Residue.	26
6.5	Entity: StructureParameter.	26
6.6	Entity: Atom.	26
6.7	Entity: BiomoleculeData.	27
6.8	Entity: MethodNMRData.	27
6.9	Entity: MethodXRayData.	28
6.10	Entity: CustomField.	28
6.11	Entity: Import.	29
6.12	Entity: ImportQueue.	29
6.13	Web Application Use Case Diagram.	34
6.14	Wireframe: Basic screen layout, welcome screen.	35
6.15	Wireframe: Query form.	35
6.16	Wireframe: Table with query results.	36
6.17	Wireframe: Table with query results.	36
6.18	Wireframe: Structure detail with visualization.	37
6.19	Wireframe: Delete of the structure and all relevant records.	37
6.20	Wireframe: Import list of the structures.	38
6.21	Wireframe: Progress detail of pending import.	38
7.1	A small example of a 3-5 B-tree.	42
7.2	Screenshot of web application – the query form.	53
7.3	Screenshot of web application – detail with Jmol demonstration.	56
7.4	Screenshot of web application – highlighted results and option for CSV export.	58
C.1	Entity-Relation diagram: Base part	75
C.2	Entity-Relation diagram: Biomolecule details	76
C.3	Entity-Relation diagram: Biomolecule data methods	77
C.4	Entity-Relation diagram: Structure parameters (same as derived tables). . . .	78
C.5	Entity-Relation diagram: Import table.	79
C.6	Entity-Relation diagram: ImportQueue table.	79

1 Introduction

This master's thesis was written with focus on making the nucleic acids structure analysis easier with the PostgreSQL, enterprise level Database Management System. Its main target is to build the database using PostgreSQL which allows to perform queries on the structure of nucleic acids in a simply, fast and user friendly way.

As the secondary target, the relevant data will be parsed and imported into PostgreSQL database which will be used by the newly developed web application.

2 Problem Description and Target Specification

This chapter presents the basic facts about nucleic acids, PostgreSQL and the main targets of this work.

2.1 Nucleic Acids and Its Commonly Used Parameters

Nucleic acids are biological molecules essential for life. Together with proteins, nucleic acids make up the most important macromolecules; each is found in abundance in all living things. [1]

2.1.1 First Discovery of Nucleic Acids and DNA

Even that nucleic acids (including the DNA molecule) were first discovered by Swiss doctor Friedrich Miescher in 1869 during analysis pus of discarded surgical bandages and sperm of salmon, the correct structure of DNA was introduced by two young scientists from Cambridge, American James D. Watson and British Francis Crick. The original paper was published by Nature magazine in 1953, the structure from X-ray diffraction was described as double helix.

Francis Crick later in 1958 articulated the “central dogma of molecular biology” which deals with the detailed residue-by-residue transfer of sequential information. It states that the information cannot be transferred back from protein to either protein or nucleic acid. In other words, once the information gets into protein, it can’t flow back to nucleic acid. [2] [3]



Figure 2.1: “Central dogma” of molecular biology/genetics in typical scheme.

James D. Watson and Francis Crick with New Zealand-born English physicist and molecular biologist Maurice Wilkins were awarded jointly by The Nobel Prize in Physiology or Medicine 1962 “for their interpretation of the genetic code and its function in protein synthesis”. [4] Further research showed up that the genetic code was based on non-overlapping triplets of bases, called codons, allowing Har Gobind Khorana, Robert W. Holley and Marshall Warren Nirenberg to decipher the genetic code. This work was awarded by The Nobel Prize in Physiology or Medicine 1968 “for their interpretation of the genetic code and its function in protein synthesis”. [5]

2.1.2 Basic Terminology

Nucleic acid is a chemical compound composed of polynucleotide chains which keeps in its structure genetic information. In this way, nucleic acids direct the course of protein synthesis, thereby regulating all cell and also indirectly whole organism activities.

The most common nucleic acids are DeoxyriboNucleic Acid (DNA) and RiboNucleic Acid (RNA). Polynucleotide chain is from chemical point of view biopolymer, composed of nucleotide monomers. In DNA and in RNA there are always four specific nucleotides. By their order in chain, we can get a large number of combinations. The sequence of specific nucleotides, the primary structure, keeps the genetic information.

Nucleotides are molecules composed of nucleobase, five-carbon sugar (ribose or 2'-deoxyribose) and one or more phosphate (ester of phosphoric acid). From the structure point of view, the main logic unit is nucleotide which itself consists of specific atoms, chemical elements with known position.

The location of elements in three-dimensional space give us more nucleotides parameters – distance of elements, angles and torsions, connected with parameters of element groups.

Nucleotides are organized in chain which creates complementary helix. By putting helices over, we can get the double helix, base genetic structure of DNA and RNA.

The parameters of nucleotide groups in chain give us many geometrical parameters for our analysis in database.

2.2 PostgreSQL: The World's Most Advanced Open-Source Database

PostgreSQL is an object-relational database management system (ORDBMS) [6] – it provides a management system which allows developers to integrate a database with their own custom data types and methods.

PostgreSQL Database Management System source code is released under the BSD license and thus is a free open-source software. As many other open-source projects, PostgreSQL is not controlled by any single company, it is developed by a worldwide community.

BSD license, which defines details of PostgreSQL usage, is probably the most liberal open-source license. It allows to use, modify and distribute source code and other parts in any form, open or closed source. Any further modifications, enhancements, or changes belong to their authors.

2.2.1 Brief History

The history of software called PostgreSQL is quite long. The name itself was changed several times – starting with Ingres, or more precisely POSTGRES (derived from post-Ingres) to PostgreSQL which originated as a result of discussion between the community and PostgreSQL Core Team and lasts until now. The current name still contains letters SQL despite the ubiquitous support of the SQL Standard.

Everything started with the Ingres project at UC Berkeley [8] and project leader Michael Stonebraker, who after his return to the Academia in 1985, started another project called post-Ingres. From this time on, the source code is fully separated out from Ingres.

The original papers describing the basics of the system were released in 1986. In 1988, the project team had a running prototype version. There are four major POSTGRES versions, the first version 1 was released in June 1989 to quite a small number of users while the last version 4 in 1993 (primarily meant as a cleanup) closed the project with a huge number of users.

Although the original project was officially ended, the BSD license allowed the community to continue the development. In 1994, two Berkeley students Andrew Yu and Jolly Chen added an SQL language interpreter to the original code and created Postgres95 which was consequently released to the web.

The first non-university development server was provided by Marc Fournier at Hub.Org Networking Services in July 1996. Roughly at the same time, Bruce Momjian and Vadim B. Mikheev started stabilizing the Berkeley code. The first open-source version was released in August 1996.

In the same year, the project was renamed again to the current name PostgreSQL.

The PostgreSQL project still honors the open-source model and makes yearly major releases (the current stable version at the beginning of February 2011 was 9.0.3).

2.2.2 Features

PostgreSQL is presented as an enterprise class database, it is fully ACID-compliant, has full support for foreign keys, joins, views, triggers and stored procedures. It includes the most of SQL92 and SQL99 data types. It also includes support of binary large objects storage. It has a native programming interfaces for C/C++, Java, .NET and many others.

As an enterprise class database, PostgreSQL has many advanced features such as Multi-Version Concurrency Control (MVCC), point in time recovery, tablespaces, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer and write ahead logging for fault tolerance. It is highly scalable.

One of the advanced features is a support of compound, unique, partial, and functional indexes which can use any of B-tree, R-tree, hash, or GiST storage methods. Other features include table inheritance, rules systems and database events.

Aside from many features, the documentation of PostgreSQL is truly exceptional.

2.3 Aim of the Thesis

This work is focused on nucleic acids structure queries and its support in PostgreSQL. Its main goals are:

- To familiarize with specifics of nucleic acids data and its sources.
- To design and implement a set of needed data structures of storing and working with nucleic acids structure in PostgreSQL.
- To import nucleic acids data from the Protein Data Bank with added parameters by analysis using external program 3DNA.
- To design and implement a proof-of-concept web application for performing user friendly queries on nucleic acids structure and its visual view in graphical applet Jmol.

3 Basics of Nucleic Acid Structure Analysis

In this chapter, the nucleic acid structure analysis is being described in more details. It will be used in the next chapters for description of application requirements.

3.1 Basic Blocks

Nucleic acids polymer comprises repeated units, consists of nucleotides, sugar and one or more phosphate.

Four distinct types of nucleotide bases were isolated – adenine, cytosine, guanine and thymine. Thymine is replaced by uracil in ribonucleid acids. Nucleotides are in sequences of nucleic acids described using single-letter codes derived from first letter – *A*, *C*, *G*, *T* and *U*. Each base could be assigned to one of two nucleobase groups, purine with letter code *Y* (adenine, guanine) and pyrimidine with letter code *R* (cytosine, thymine, uracil).

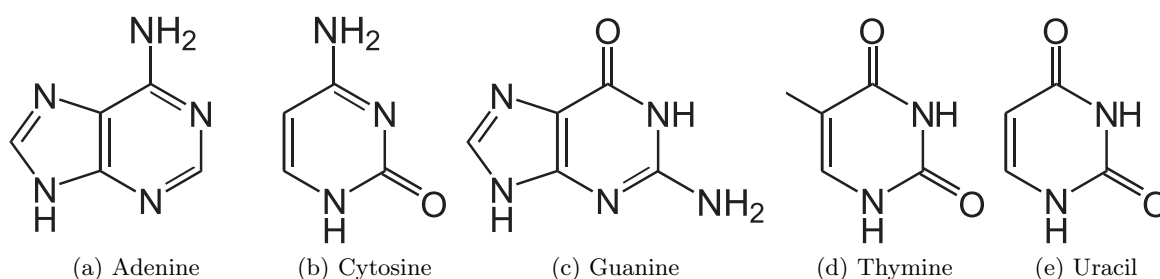


Figure 3.1: Chemical structure of DNA/RNA bases.

Nucleic acid types differ in the structure of the sugar – DNA contains 2'-deoxyribose, RNA contains ribose. The difference between these two monosaccharides is only the presence of a hydroxyl group.

Phosphate groups are usually coded as single letter *p*. Phosphates and sugars in nucleic acids creating sugar-phosphate backbone through phosphodiester linkages.

3.2 Structure

3.2.1 Base Pairing

Experiments have shown that DNA samples taken from different cells of the same species have always the same amounts of adenine and thymine, as the amounts of cytosine and guanine are, later on articulated as “Chargaff’s rules” by Austrian chemist Erwin Chargaff: DNA from any cell of all organisms should have a 1:1 ratio of pyrimidine and purine bases and, more specifically, the amount of guanine is equal to cytosine and the amount of adenine is equal to thymine. This pattern is found in both strands of the DNA. [9]

As James D. Watson and Francis Crick then proposed, each of these pairs of purine and pyrimidine bases are held together by specific hydrogen bonds. In the canonical “Watson-Crick DNA base pairing”, adenine forms a base pair with thymine, and guanine forms a base pair with cytosine. In RNA, thymine is replaced by uracil in base pair with adenine.

The adenine-thymine (*A-T*) base pair has two hydrogen bonds compared to the three in a guanine-cytosine (*G-C*) one. The initial Watson-Crick model assigned just two hydrogen bonds to the *G-C* pair, and the third hydrogen bond emerged some three years later. [10]

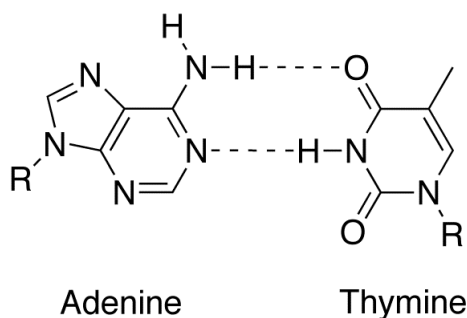


Figure 3.2: An *AT* base pair demonstrating three intermolecular hydrogen bonds.

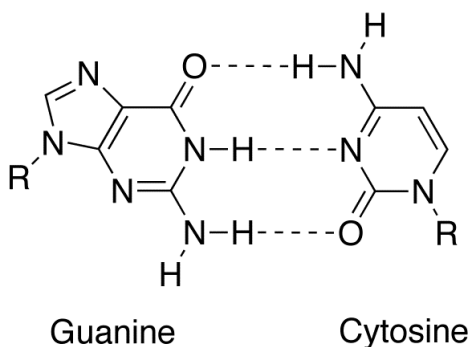


Figure 3.3: A *GC* base pair demonstrating two intermolecular hydrogen bonds.

3.2.2 Topology

In conventional nucleic acid nomenclature, the carbons which the phosphate groups attach to are the 3'-end and the 5'-end carbons of the sugar. This gives nucleic acids directionality, and the ends of nucleic acid molecules are referred to as 5'-end and 3'-end. The chemical convention of naming carbon atoms in the nucleotide sugar-ring numerically gives rise to a 5'-end and a 3'-end (usually pronounced as the “five prime end” and the “three prime end”). By convention, single strands of DNA and RNA sequences are written in 5'-to-3' direction.

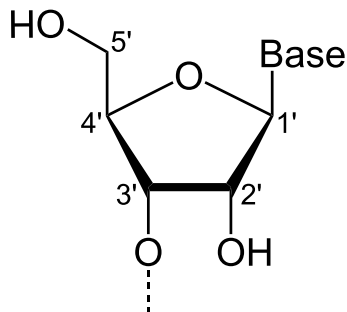


Figure 3.4: Numbered ribose carbons on a nucleotide, a base is attached to a ribose ring.

DNA molecule consists of two strands coiled around each other. The sugar-phosphate backbone is on the outside of the double helix, since the bases are on the inside – connected in pairs directly toward a base on the second strand. Since the strands of the DNA double helix run in

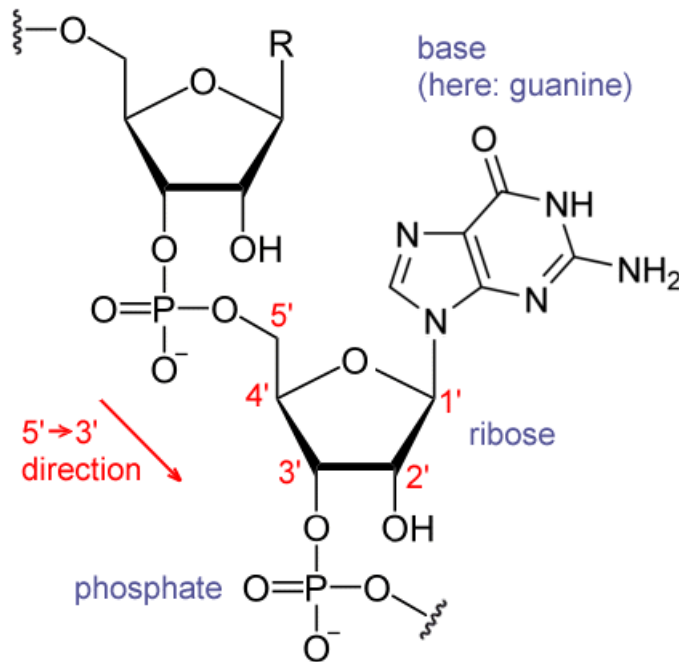


Figure 3.5: Chemical structure of RNA with drawn 5'-to-3' direction.

opposite directions (5'-to-3' direction and vice versa), it's creating antiparallel relation.

For example, the following sequence illustrates pair double-stranded patterns in DNA and equivalently in RNA. The top strand is written by convention in 5'-to-3' direction, so the bottom anti-parallel strand is written from from 3' end to 5' end:

ATCGATTGAGCTCTAGCG
TAGCTAACTCGAGATCGC

AUCGAUUGAGCUCUAGCG
UAGCUAACUCGAGAUCGC

Nucleic acids are in general very large molecules. Known biological nucleic acid molecules start at tens (10) in RNA and end in billions (1,000,000,000) of nucleotides. Human chromosome 1, which is a single molecule, has in total 247,199,719 bases. The whole human cell DNA was read in 2003 during 13-year action "Human Genome Project" coordinated by the U.S. Department of Energy and the National Institutes of Health, it had in total 3,079,843,747 nucleotides in 23 pairs of large linear sequences. [11]

3.2.3 Base Pair Geometry

The base pair geometry, respectively geometric deviations from ideal structure, is important parameter of DNA structure as it affects process of genetic information transfer. For example, DNA-binding protein regulates the expression of one or more genes by blocking key DNA operator. This operator contains a pyrimidine-purine step that becomes positively super-coiled forming a kink in the phosphodiester backbone. This is how the protein checks for the recognition site as it allows the DNA duplex to follow the shape of the protein. In other words, recognition happens through indirect readout of the structural parameters of the DNA, rather

than via specific base sequence recognition. [12] The “indirect readout” is the catchall opposite of “direct readout” which could be defined as preference for a particular pattern of hydrogen bond donors, hydrogen bond acceptors, and non-polar groups of the base pairs presented in the major and/or minor grooves of DNA. [13]

To enable the complete description of DNA geometry, a number of rotation and translation parameters have been introduced. [14] These parameters could be divided into two groups for the base pairs, pair of two bases on both strands, and base steps, two nucleotides in chain sequence connected in two pairs with anti-parallel strand – two consecutive base pairs.

- There are six main base-pair parameters – Shear, Buckle, Stretch, Propeller, Stagger, Opening,
- and six main base-step parameters – Shift, Tilt, Slide, Roll, Rise, Twist.

These values precisely define the location and orientation in space of every base or base pair in a nucleic acid molecule relative to its predecessor along the axis of the helix. Together, they characterize the helical structure of the molecule.

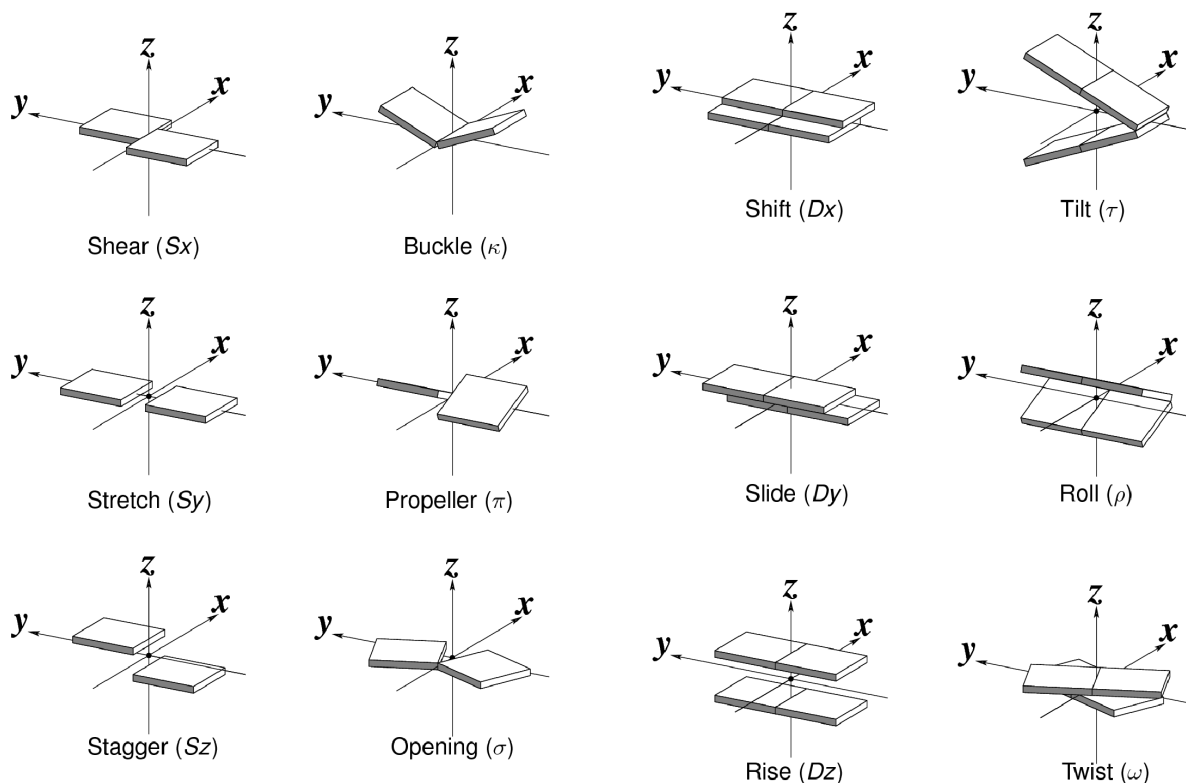


Figure 3.6: Illustration describing base pair geometry parameters.

The base parameters could be counted from position of atoms in three-dimensional space. Standardized coordinate reference frame for calculation of base parameters (at figure 3.7) is available in “A Standard Reference Frame for the Description of Nucleic Acid Base-pair Geometry” [15]. It has the x-axis directed towards the major groove along the pseudo two-fold axis of an idealized Watson-Crick base pair (shown as \bullet). The y-axis is along the long axis of the base pair, parallel to the C1'-C1' vector. The position of the origin is clearly dependent on the geometry of the bases and the base pair.

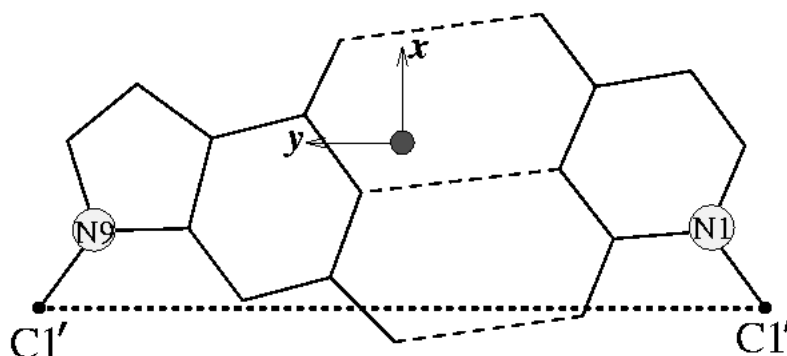


Figure 3.7: Reference frame for idealized helical DNA.

3.3 Data Sources and It's Usage

Data about nucleic acids could be obtained from public databases. Probably the most common database is Protein Data Bank (PDB), 3D structural data repository of large biological molecules derived from X-ray diffraction and NMR studies. This representation was created in the 1970's and a large amount of software using its file format has been written.

3.3.1 Protein Data Bank

The base of Protein Data Bank was created in 1969, when Edgar Meyer started with software for storing growing amount of atomic coordinate files in a common format suitable for geometric and graphical evaluation. Currently, PDB is maintained by Worldwide Protein Data Bank (wwPDB). The mission of the wwPDB is to maintain a single Protein Data Bank Archive of macromolecular structural data that is freely and publicly available to the global community. [16]

The database is updated weekly, each Tuesday night. In 1971 it originally contained 7 structures, as of 22nd February 2011 it contains in total 71,415 structures. The database is free to use, all data could be downloaded from website <http://www.pdb.org/>.

Most structures are determined by X-ray diffraction, technique based on observing the scattered intensity of an X-ray beam hitting a sample as a function of incident and scattered angle, polarization, and wavelength or energy. About 15 % of structures are currently determined by protein nuclear magnetic resonance spectroscopy (protein NMR), technique that exploits the magnetic properties of certain nuclei.

The file format initially used by the PDB was called the PDB file format (currently at revision 3.2), text file limited to 80 characters per line with hundreds to thousands of lines with atomic coordinates. This format was replaced by enhanced Crystallographic Information File – mmCIF file format in 1996. In 2005, XML version of PDB file called PDBML was introduced.

Each structure file could be obtained by PDB code – four-character alphanumeric unique identifier. For example, 2 polymers compound *Solution Structure of a control DNA Duplex*, released 23rd February 2011, have assigned PDB ID 2L8Q. Part of the PDB file looks like:

```
COMPND  5 MOL_ID: 2;
COMPND  6 MOLECULE: DNA (5'-D(*GP*CP*GP*TP*AP*GP*CP*AP*TP*GP*CP*G)-3');
COMPND  7 CHAIN: B;
```

```

COMPND      8 ENGINEERED: YES
SOURCE      MOL_ID: 1;
SOURCE      2 SYNTHETIC: YES;
SOURCE      3 MOL_ID: 2;
SOURCE      4 SYNTHETIC: YES
KEYWDS      DNA, POXVIRUS, HPMP
...
REMARK 210 EXPERIMENTAL DETAILS
REMARK 210 EXPERIMENT TYPE                : NMR
REMARK 210 TEMPERATURE                    (KELVIN) : 298
REMARK 210 PH                            : 7.0-7.2
REMARK 210 IONIC STRENGTH                 : 70
REMARK 210 PRESSURE                      : AMBIENT
REMARK 210 SAMPLE CONTENTS               : 2 MM CONTROL DNA DUPLEX, 95%
REMARK 210                               H2O/5% D2O
...
SEQRES      1 A   12   DC  DG  DC  DA  DT  DG  DC  DT  DA  DC  DG  DC
SEQRES      1 B   12   DG  DC  DG  DT  DA  DG  DC  DA  DT  DG  DC  DG
...
ATOM         1  O5'  DC  A   1          6.550  17.386  -2.304  1.00  0.00      O
ATOM         2  C5'  DC  A   1          7.059  18.467  -1.521  1.00  0.00      C
ATOM         3  C4'  DC  A   1          6.628  18.415  -0.040  1.00  0.00      C
ATOM         4  O4'  DC  A   1          5.226  18.655   0.068  1.00  0.00      O
ATOM         5  C3'  DC  A   1          6.946  17.075   0.644  1.00  0.00      C
...

```

3.3.2 3DNA, Analysis of Three-dimensional Nucleic Acid Structures

Data from the Protein Data Bank could be analyzed by many free or commercial programs. On the list of the widely used software there is also 3DNA, versatile package for the analysis, rebuilding, and visualization of three-dimensional nucleic acid structures, based on a standard reference frame. [17] 3DNA was created by Dr. Xiang-Jun Lu in the laboratory of Professor Wilma K. Olson at the Department of Chemistry and Chemical Biology, Rutgers, the State University of New Jersey.

3DNA is at the beginning of 2011 distributed in two versions – out-of-date version 3DNA v1.5 free for download, and new v2.0 which can't be distributed directly, the tarball with programs and its password must be requested individually. Software is distributed as the closed-source binary for various platforms. From our example point of view, the usage of program is the same in both versions, so for the demonstration we will use v1.5.

Installation in Linux environment is pretty straightforward, at first we extract the archive from 3DNA homepage:

```
http://3dna.rutgers.edu:8080/3DNA_v1.5/MacOSX_Intel_X3DNA_v1.5.tar.gz
```

Then we announce its root location via environment variable X3DNA. The whole 3DNA software package have many other options how to do analysis or visualization of files with nucleic acids, we will just start the first analysis using command:

```
findpair %pdb-file.pdb stdout | analyze
```

The tools will generate few text files with variable output, local base pair and base step parameters are in file `bp_step.par`.

For the file from tutorial documentation, PDB ID 1TC3, the output is presented below. Even that the file is in plain, easy-to-read text, it is possible to analyze it and use it in database by simple regular expressions.

19 base-pairs

0 ***local base-pair & step parameters***

	Shear	Stretch	Stagger	Buckle	Prop-Tw	Opening	Shift	Slide	Rise	Tilt	Roll	Twist
G-C	-0.18	-0.27	0.58	8.57	-9.77	-8.26	0.00	0.00	0.00	0.00	0.00	0.00
G-C	0.16	-0.26	0.13	-5.72	-10.04	-3.52	0.19	-1.13	3.65	5.58	3.88	39.73
G-C	-0.10	-0.25	0.42	-1.54	-7.63	-0.67	0.43	-1.67	2.94	-3.14	10.32	29.99
G-C	0.02	-0.33	0.08	-3.56	-6.03	1.74	0.17	-2.24	3.13	1.31	6.72	31.31
G-C	-0.21	-0.24	-0.13	-3.78	-0.69	3.13	0.08	-1.92	3.33	3.67	8.74	27.62
G-C	0.04	-0.22	-0.35	-8.83	-11.08	-3.03	0.50	-1.91	3.41	4.23	5.60	31.56
G-C	-0.32	-0.40	-0.38	-16.06	-17.27	-3.53	0.18	-2.14	3.43	4.68	10.29	27.30
T-A	0.43	-0.40	-0.05	-11.97	-14.80	12.03	-0.07	-0.42	3.15	-0.69	5.91	35.56
C-G	0.52	-0.43	-0.34	-10.52	4.23	3.87	0.25	0.76	3.46	3.31	0.88	37.83
C-G	0.27	-0.33	0.52	-15.53	-4.19	-2.52	-1.73	0.69	3.77	-7.18	-1.10	31.39
T-A	-0.18	-0.23	-0.13	-9.98	-5.91	-1.56	0.35	-0.32	3.19	5.36	6.16	26.98
A-T	-0.57	-0.49	0.47	0.67	-7.27	-0.77	0.41	1.89	3.06	-6.05	-2.74	45.60
T-A	-0.34	-0.35	0.33	-3.99	-14.34	-5.83	-0.34	-0.18	3.44	1.73	-3.42	31.69
A-T	-0.69	-0.02	-0.04	-2.98	-0.45	12.91	0.29	2.18	3.37	-2.75	-6.21	49.54
G-C	-0.51	-0.07	-0.61	3.47	-9.27	12.05	0.33	1.29	3.25	-0.15	15.17	23.41
A-T	-0.66	-0.15	-0.12	12.32	-13.66	3.93	-0.68	0.30	2.96	-4.96	0.68	35.87
A-T	-0.41	-0.38	0.73	21.10	-23.76	-1.56	-0.06	-0.03	2.97	-6.65	-0.95	34.68
C-G	0.40	-0.55	0.33	-5.21	-10.24	-3.28	0.37	-1.03	3.93	4.89	1.28	38.27
T-A	0.56	-0.49	0.46	-21.17	-8.34	-4.72	-0.19	-1.07	3.61	1.91	4.78	37.16

3.3.3 Jmol, an Open-source Java Viewer for Chemical Structures in 3D

Even that 3DNA could be used to static visualizations, we can view helices of DNA, one of the most beautiful molecules in living cells, in real-time 3D using open-source application Jmol.¹

Jmol is able to open many file formats including PDB, is cross-platform and supports all major web browsers. At the beginning of 2011, the latest stable version was 12.0. [18]

The tool consists of three main components:

- JmolApplet, the web browser applet.
- Jmol application, the stand-alone Java application distributed in jar.
- JmolViewer, the development toolkit for integration with custom application.

For basic demonstration, we use just Jmol application started by Java Run-time Environment. For further demonstration of new database, we will use JmolApplet integrated in simple page accessible by web browser. With Jmol it's possible to customize the program's behavior including colors, structure details and main menu so it will be a good way how to add direct visual view of the examined molecule.

Installation and usage is also pretty simple – we extract the ZIP archive from homepage and run the software via prepared shell script `jmol.sh`. Then we will be able to open the PDB file directly with Jmol and immediately see and adjust view in 3D visualization.

File analyzed by 3DNA above, PDB ID 1TC3, is rendered in figure 3.8.

¹Jmol, written by capital J, lower case mol. Jmol was started as an OpenScience project and it's currently being developed by many contributors. The exact spelling is required to avoid confusing, since there is another project project called JMol. Will York wrote the JMol viewer at the Complex Carbohydrate Research Center, University of Georgia, USA. The home page was at <http://www.ccruc.uga.edu/~will/jmol/jmol.html> (it is no longer there).

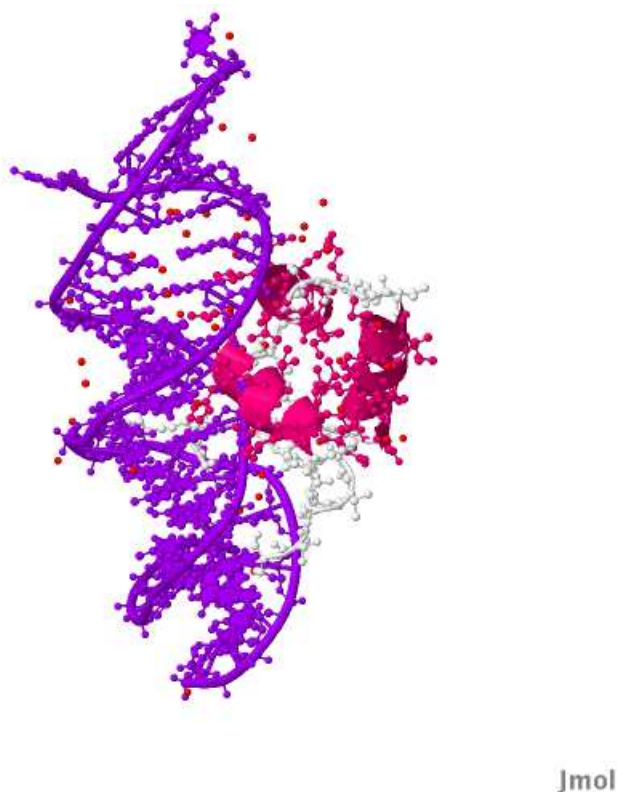


Figure 3.8: Jmol Visualization of Transposase TC3A1-65 from *Caenorhabditis Elegans*, PDB ID 1TC3.

3.4 Use Cases

We can define the simple UML Use Case diagram for the nucleic acids database – we have a researcher, scientist who is placing queries on nucleic acids structure, either directly in SQL or via custom application:

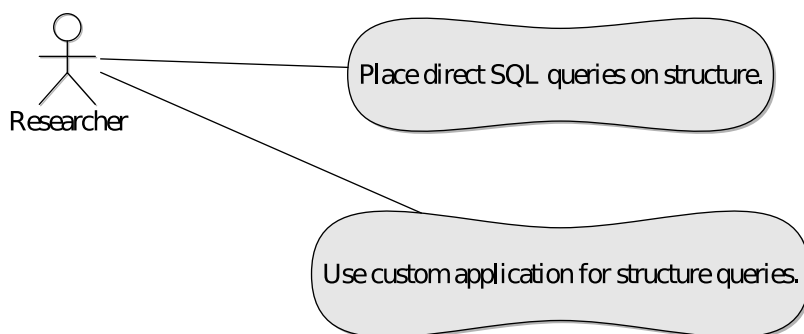


Figure 3.9: Basic Use Case for nucleic acids researcher.

4 Analysis of Nucleic Acids Using RDBMS

Previous chapter showed what are the specifics of data used for structure analysis of nucleic acids. In the following chapter there are described ways and tools, how to work with this specific data in relational database management system (RDBMS).

4.1 Basics of Database Management System Usage

There are two main purposes of database management systems:

- to eliminate data redundancy,
- to create and execute queries on the data.

4.1.1 Data Redundancy Elimination – Normalization

The first purpose is simple, with no data redundancy are operations for additions, deletions and modifications effective as it could be done by just one table write and naturally propagated in whole database.

The inventor of relation model, Edgar F. Codd, introduced scheme of database with low redundancy level by usage of normalization concept. In 1970, First Normal Form (1NF, also called Minimal form) was published, Second Normal Form (2NF) and Third Normal Form (3NF) followed in year 1971. In 1974, Mr. Codd with Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF, also called 3.5NF). The original concept was then enhanced by higher normal forms which are not concerned with functional dependencies only. At beginning of 2011, the latest form is 6NF, written by Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002. [19]

The purpose of 1NF is basically to allow data to be queried and manipulated using a “universal data sub-language” grounded in first-order logic, table faithfully represents a relation and has no repeating groups.

2NF could be briefly described as no non-prime attribute in the table and is functionally dependent on a proper subset of a candidate key.

3NF says that every non-prime attribute is non-transitively dependent on every candidate key in the table.

In the end, BCNF require that every non-trivial functional dependency in the table is a dependency on a superkey.

Since the Protein Database File format is basically focused to be easily readable by humans, we can eliminate some level of data redundancy in flat text of PDB files using principles of normalization, typical relation database approach to storing information.

4.1.2 Creating and Execution Queries on the Data

Working with data by queries is a simple way, how to operate with the database. If we have all data from some nucleic acids group or source parsed and stored in database management system, we can then simply execute queries like “How many residues there are in a specific relation?” or “What are the PDB IDs of all nucleic acids with specific base-step parameters?”.

Typical way how to create the queries on object-relation databases are in database computer language Structured Query Language (SQL), developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in 1970. [20] Despite not adhering to the relational model as described by author of normalization rules, it became the most widely used database language, mainly due easy use – typical query could look like: `SELECT title, year FROM books WHERE price > 100 ORDER BY year;`

4.1.3 Search Speed

Typical important criteria of database usage is speed of searching, either for getting data from database or storing them. We can simply compare PDB flat file with database model consisting of tables in Boyce-Codd Normal Form.

Complexity of searching in each single PDB file from desired set is easy, we need to check one item at time, without jumping in the file. This will give us complexity of sequential searching (4.1), where n is number of lines in all files.

$$O(n) \tag{4.1}$$

On the other hand, complexity of searching in indexed database (when we are looking for data using some data organization) is smarter, we are able to jump in data, so the number of search operations grows more slowly than the list does. That gives us complexity of indexed searching (4.2), where n is number of rows in tables.

$$O(\log n) \tag{4.2}$$

In addition, usage efficiency is also influenced by I/O requirements, the need for reading data from storage is less by using indexed database.

But there is one main disadvantage of relational database design, the indirect storage requirements. With complex indexed structures, the need for free space could be several times bigger than clean data itself.

4.2 Relation Database Systems for the PDB

There are two interesting papers regarding usage of PDB data in relation database system. First, relational database based on the mmCIF schema [21] exists for Advanced Searching on PDB web site. Second, PDB-SQL project [22] introduced scheme for importing all PDB data into MySQL database originally designed for the storage of alpha carbon coordinates and other types of information. The result was huge reduction of time required for searching, e.g. from more than 1 week spent in extracting alpha carbon information from every structure in the PDB, they were able to get it via SQL in approximately 20 minutes.

5 Specification of PostgreSQL Database

In this chapter, the full specification of requested database features is described.

5.1 Support Tools Needed

Few tools will be needed for complete working with the database – scripts for populating the database with data from PDB parsed by 3DNA and its demonstration using web application with graphical output using JmollApplet.

The complete list of tools needed is:

- Script for getting specific or all data from PDB, it necessary analysis using 3DNA tool and preparing output in SQL INSERTs (possibly performed in database connection).
- Simple web application with JmollApplet which will allow to perform the queries on nucleic acid structure.

5.2 List and Description of Requested Features

First, the data from PDB and 3DNA needs to be stored in appropriate data structure. The attributes required are:

- 4 letter PDB Name
- structure name/title
- data about NMR method – date, pressure, temperature, pH, ionic strength
- data about XRay method – resolution, method, date, temperature, pH, number of crystals
- one or more custom fields which could be defined or used by time of importing data and later as simple extension of otherwise “static” database structure

Then, the basic queries must be supported:

- Query by the acid attribute – unique identifier, 4 letter PDB Name, name.
- Query by chain – unique identifier, order/sequence information.

Finally, advanced structure queries must be supported as well:

- Query on dinucleotids.
- Query by 6 base-pair parameters: Shear, Buckle, Stretch, Propeller, Stagger, Opening.
- Query by 6 base-step parameters: Shift, Tilt, Slide, Roll, Rise, Twist.
- Query on overlap area between polygons: i1-i2, i1-j2, j1-i2, j1-j2 (plus same for base ring atoms only).
- Query on origin and mean normal vector, 6 coordinates: Ox, Oy, Oz, Nx, Ny, Nz.
- Query on local base-pair helical parameters: X-disp, Y-disp, h-Rise, Incl., Tip, h-Twist.
- Query on classification of each dinucleotide step: Xp, Yp, Zp, XpH, YpH, ZpH.

- Query on global parameters based on C1'-C1' vectors: disp., angle, twist, rise.
- Query on main chain and chi torsion angles: alpha, beta, gamma, delta, epsilon, zeta, chi.
- Query on sugar conformational parameters: tm and P.
- Query on strand P-P and C1'-C1' virtual bond distances: P-P, C1'-C1'.
- Query on helix radius: O4 and C1.
- Query on position and local helical axis vector: Px, Py, Pz, Hx, Hy, Hz.

6 Application Analysis and Design

The main goal of this thesis is to design and implement a set of needed data structures of storing and working with nucleic acids structure and to prepare a proof-of-concept web application for placing user friendly queries on nucleic acids structure and its visual view. In this chapter, the main application components are designed and described, so it will be possible to build the whole software project in the next chapter.

6.1 Application Components

The application will consist of there main components.

- Database Population Tool, the software for importing data from PDB to database.
- Database, the database itself, for the structure queries.
- Web Application, the demonstration of database usage.

6.2 Database Population Tool

The purpose of database population tool is simple – to parse one of more PDB files into database scheme defined later in this chapter.

In more detail, tool will handle three main tasks:

- Download data from Protein Data Bank.
- Analyze data – save details about experiment and structure in database.
- Calculate base-pair and base-step parameters.

6.2.1 Data Download

Although the data of nucleic acids could be downloaded from PDB homepage one by one, we will need some better way how to get the data in more comfortable way. The first approach is to download all data, either by public archive at <ftp://ftp.wwpdb.org/pub/pdb> or by one of the regular snapshots published yearly at <ftp://snapshots.rcsb.org/>. The snapshots are easy to get and update by smart data files synchronization via rsync. [23] At the beginning of April 2011, the whole FTP site had approx. 130 million of files.

For our needs, we will download the files one by one, but automatically and with cache (which could be pointed on FTP archive mirror). The URI for downloading [24] data could be static as

```
http://www.rcsb.org/pdb/files/4hhb.pdb.gz
http://www.rcsb.org/pdb/files/4hhb.cif.gz
```

or exported more dynamically via export options as

```
http://www.rcsb.org/pdb/download/downloadFile.do?fileFormat=xml&\
compression=NO&structureId=4HHB
```

6.2.2 Analysis of Downloaded Data

There are few sub-tasks of data analysis. The first one is about choosing the proper file format of data, as parsing of PDB entries can be very simple or quite tricky, depending on type of the data that you want to extract and on data source. PDB is available in three main file formats – original PDB file format, mmCIF and PDBML.

Original “legacy” PDB file is typically the worst option, although it’s very easy to use by humans, there are limitations like maximum 80 non-control ASCII characters per line and some missing details which are available only at extended file formats. The PDB file format is well documented, specification version 3.20 [25] from September 2008 is at approximately 188 A4 pages. The automatic way how to parse the data could be class PdbSAXParser based on Java SAX2 Parser in open-source project BioJava. [26]

Next possibility, mmCIF, macromolecular Crystallographic Information File, is based on CIF, Crystallographic Information File. As it’s written on mmCIF homepage [27], CIF is a subset of STAR (Self-defining Text Archive and Retrieval format), general and flexible way how to store text and numerical data. The dictionary specifying all needed fields was released in 1997, almost 30 years after initial PDB file specification. Automated analysis of the (mm)CIF files can be easily done by the STAR parser, for example there is a free implementation in Perl available at <http://pdb.sdsc.edu/STAR/index.html> or mmCIF parser in Python from the BioPython package, C-based code which uses flex (fast lexical analyzer generator) [28] or the Python Macromolecular Library [29].

PDBML, Protein Data Bank Markup Language, is a source of data in XML which could be easily read on different software platforms. In fact it’s the direct translation of mmCIF format, so the XML schema is using same PDB Exchange Data Dictionary as mmCIF format. As before, there are specialized parsers for this type of file, for example the BioPython can import such files.

Even though the analysis of data storage in the mmCIF is automatic, we will still need to know where to look at the analyzed attributes. And this is the time when we need to check more deeply the specification of mmCIF dictionaries. [31]

The dictionary is defined by Dictionary Definition Language (DDL), by rules that are providing a framework from which it is possible to define dictionary of the terms needed by the discipline. The DDL provides a convention for naming and defining data items within the dictionary, declaring specific attributes of those data items, for example, a range of values and the data type, and for declaring relationships between data items. The resulting dictionary is then defined as a list of specifically allowed groups of letters which allow to save final data. [32] The data is organized in categories, where each has typically several items, some of them are mandatory.

For example, the data of *Crambe hispanica subsp. abyssinica*, PDB ID 1CBN, first line in PDB file – the header

```
HEADER      PLANT SEED PROTEIN                        11-OCT-91    1CBN
```

will be structured according mmCIF dictionary (with some extra data)

```
data_1CBN
#
_entry.id    1CBN
#
_audit_conform.dict_name      mmcif_pdbx.dic
_audit_conform.dict_version   1.0670
_audit_conform.dict_location  http://mmcif.pdb.org/dictionaries/ascii/mmcif_pdbx.dic
```



```
#
_database_2.database_id      PDB
_database_2.database_code    1CBN
#
loop_
_database_PDB_rev.num
_database_PDB_rev.date
_database_PDB_rev.date_original
_database_PDB_rev.status
_database_PDB_rev.replaces
_database_PDB_rev.mod_type
1 1994-01-31 1991-10-11 ? 1CBN 0
2 2003-04-01 ?          ? 1CBN 1
3 2009-02-24 ?          ? 1CBN 1
#
_struct.entry_id '1CBN'
_struct.keywords.text          'PLANT SEED PROTEIN'
```

The main difference from PDB file format is name-value pairing with explicit reference to each item of data, the *name* construct is in the form *_category.extension*. The PDB file is typically having the interpretation left to the software reading the file whereas the characteristics of specific data in mmCIF are exactly explained in dictionary mentioned above and described below.

We should also note the usage of STAR *loop_* construct, simple way how to write multiple values for the same data item.

6.2.3 Needed Categories from Dictionary `mmcif_pdbx.dic`

The dictionary describing the data organization in the mmCIF is in `mmcif_pdbx.dic`, easily readable in HTML version.

We will need to get the data from following topics:

- Data about structure discovery, name, authors, date of release and deposition, citation, experiment method and its details.
- Data about experiment (mainly X-Ray diffraction and NMR) and its parameters including refinement of structure, conditions and methods used to grow crystals, etc.
- Data about structure of the polymer.
- Structural data, chains, atoms with position.

1. Category `citation`

Data items in the CITATION category record details about the literature cited as being relevant to the contents of the data block.

- `title` (not mandatory)

The title of the citation; relevant for journal articles, books and book chapters.

2. Category `citation_author`

Data items in the CITATION_AUTHOR category record details about the authors associated with the citations in the CITATION list.

- `citation_id` (mandatory)

This data item is a pointer to `_citation.id` in the CITATION category. We will use it for saving data, if the `citation.id` will be `primary`.

- `name` (mandatory)

Name of an author of the citation; relevant for journal articles, books and book chapters. The family name(s), followed by a comma and including any dynastic components, precedes the first name(s) or initial(s).

3. Category `exptl`

Data items in the EXPTL category record details about the experimental work prior to the intensity measurements and details about the absorption-correction technique employed.

- `method` (mandatory)

The method used in the experiment. We will consider `X-RAY DIFFRACTION` as X-Ray diffraction method, `SOLUTION NMR` as NMR.

4. Category `database_PDB_rev`

Data items in the DATABASE_PDB_REV category record details about the history of the data block as archived by the Protein Data Bank (PDB). These data items are assigned by the PDB database managers and should only appear in a data block if they originate from that source.

- `date` (not mandatory)

Date the PDB revision took place. Taken from the REVDAT record.

- `date_original` (not mandatory)

Date the entry first entered the PDB database in the form `yyyy-mm-dd`. Taken from the PDB HEADER record.

5. Category `entity_poly`

Data items in the ENTITY_POLY category record details about the polymer, such as the type of the polymer, the number of monomers and whether it has nonstandard features.

- `pdbx_seq_one_letter_code_can` (not mandatory)

Canonical chemical sequence expressed as string of one-letter amino acid codes. Modifications are coded as the parent amino acid, where possible.

```

A  for alanine or adenine
B  for ambiguous asparagine/aspartic-acid
R  for arginine
N  for asparagine
D  for aspartic-acid
C  for cysteine or cystine or cytosine
Q  for glutamine
E  for glutamic-acid
Z  for ambiguous glutamine/glutamic acid
G  for glycine or guanine
H  for histidine
I  for isoleucine
L  for leucine
```

K for lysine
M for methionine
F for phenylalanine
P for proline
S for serine
T for threonine or thymine
W for tryptophan
Y for tyrosine
V for valine
U for uracil

6. Category `refine_hist`

Data items in the `REFINE_HIST` category record details about the steps during the refinement of the structure. These data items are not meant to be as thorough a description of the refinement as is provided for the final model in other categories; rather, these data items provide a mechanism for sketching out the progress of the refinement, supported by a small set of representative statistics.

- `d_res_low` (mandatory)

The highest value for the interplanar spacings for the reflection data for this cycle of refinement. This is called the lowest resolution.

- `d_res_high` (mandatory)

The lowest value for the interplanar spacings for the reflection data for this cycle of refinement. This is called the highest resolution.

7. Category `exptl_crystal_grow`

Data items in the `EXPTL_CRYSTAL_GROW` category record details about the conditions and methods used to grow the crystal.

- `method` (not mandatory)

The method used to grow the crystals.

- `temp` (not mandatory)

The temperature in Kelvins at which the crystal was grown. If more than one temperature was employed during the crystallization process, the final temperature should be noted here and the protocol involving multiple temperatures should be described in `_exptl_crystal_grow.details`.

- `pH` (not mandatory)

The pH at which the crystal was grown. If more than one pH was employed during the crystallization process, the final pH should be noted here and the protocol involving multiple pH values should be described in `_exptl_crystal_grow.details`.

8. Category `refine`

Data items in the `REFINE` category record details about the structure-refinement parameters.

- `ls_number_reflns_all` (not mandatory)

The number of reflections that satisfy the resolution limits established by `_refine.ls_d_res_high` and `_refine.ls_d_res_low`.

- `ls_number_reflms_obs` (not mandatory)

The number of reflections that satisfy the resolution limits established by `_refine.ls_d_res_high` and `_refine.ls_d_res_low` and the observation limit established by `_reflms.observed_criterion`.

9. Category `pdbx_nmr_spectrometer`

The details about each spectrometer used to collect data for this deposition.

- `field_strength` (not mandatory)
Select the field strength for protons in MHz.

10. Category `pdbx_nmr_ensemble`

This category contains the information that describes the ensemble of deposited structures. If only an average structure has been deposited skip this section.

- `conformers_submitted_total_number` (not mandatory)
The number of conformer (models) that are submitted for the ensemble.

11. Category `pdbx_nmr_exptl_sample_conditions`

The experimental conditions used to for each sample. Each set of conditions is identified by a numerical code.

- `temperature` (not mandatory)
The temperature (in Kelvin) at which NMR data were collected.
- `pressure` (not mandatory)
The pressure at which NMR data were collected.
- `pH` (not mandatory)
The pH at which the NMR data were collected.
- `ionic_strength` (not mandatory)
The ionic strength at which the NMR data were collected.

12. Category `atom_site`

Data items in the `ATOM_SITE` category record details about the atom sites in a macromolecular crystal structure, such as the positional coordinates, atomic displacement parameters, magnetic moments and directions.

The data items for describing anisotropic atomic displacement factors are only used if the corresponding items are not given in the `ATOM_SITE_ANISOTROP` category.

- `group_PDB` (not mandatory)
The group of atoms to which the atom site belongs. This data item is provided for compatibility with the original Protein Data Bank format, and only for that purpose.
- `id` (mandatory)
The value of `_atom_site.id` must uniquely identify a record in the `ATOM_SITE` list. Note that this item need not be a number; it can be any unique identifier. This data item was introduced to provide compatibility between small-molecule and macromolecular CIFs.
- `type_symbol` (mandatory)
This data item is a pointer to `_atom_type.symbol` in the `ATOM_TYPE` category.

- **label_entity_id** (mandatory)
This data item is a pointer to `_entity.id` in the ENTITY category.
- **Cartn_x** (not mandatory)
The x atom-site coordinate in angstroms specified according to a set of orthogonal Cartesian axes related to the cell axes as specified by the description given in `_atom_sites.Cartn.transform_axes`.
- **Cartn_y** (not mandatory)
The y atom-site coordinate in angstroms specified according to a set of orthogonal Cartesian axes related to the cell axes as specified by the description given in `_atom_sites.Cartn.transform_axes`.
- **Cartn_z** (not mandatory)
The z atom-site coordinate in angstroms specified according to a set of orthogonal Cartesian axes related to the cell axes as specified by the description given in `_atom_sites.Cartn.transform_axes`.
- **occupancy** (not mandatory)
The fraction of the atom type present at this site. The sum of the occupancies of all the atom types at this site may not significantly exceed 1.0 unless it is a dummy site.
- **B_iso_or_equiv** (not mandatory)
Isotropic atomic displacement parameter, or equivalent isotropic atomic displacement parameter, *B eq*, calculated from the anisotropic displacement parameters.
- **Cartn_x_esd** (not mandatory)
The standard uncertainty (estimated standard deviation) of `_atom_site.Cartn.x`.
- **Cartn_y_esd** (not mandatory)
The standard uncertainty (estimated standard deviation) of `_atom_site.Cartn.y`.
- **Cartn_z_esd** (not mandatory)
The standard uncertainty (estimated standard deviation) of `_atom_site.Cartn.z`.
- **occupancy_esd** (not mandatory)
The standard uncertainty (estimated standard deviation) of `_atom_site.occupancy`.
- **B_iso_or_equiv_esd** (not mandatory)
The standard uncertainty (estimated standard deviation) of `_atom_site.B_iso_or_equiv`.
- **auth_asym_id** (mandatory)
Author's strand id.
- **auth_seq_id** (not mandatory)
Author's sequence identifier.
- **auth_comp_id** (not mandatory)
An alternative identifier for `_atom_site.label_comp_id` that may be provided by an author in order to match the identification used in the publication that describes the structure.
- **auth_atom_id** (not mandatory)
An alternative identifier for `_atom_site.label_atom_id` that may be provided by an author in order to match the identification used in the publication that describes the structure.

13. Category `atom_site_anisotrop`

Data items in the `ATOM_SITE_ANISOTROP` category record details about anisotropic displacement parameters. If the `ATOM_SITE_ANISOTROP` category is used for storing these data, the corresponding `ATOM_SITE` data items are not used.

- `id` (mandatory)
This data item is a pointer to `_atom_site.id` in the `ATOM_SITE` category.
- `type_symbol` (mandatory)
This data item is a pointer to `_atom_type.symbol` in the `ATOM_TYPE` category.
- `pdtx_auth_seq_id` (not mandatory)
Pointer to `_atom_site.auth_seq_id`.
- `pdtx_auth_comp_id` (not mandatory)
Pointer to `_atom_site.auth_comp_id`.
- `pdtx_auth_asym_id` (not mandatory)
Pointer to `_atom_site.auth_asym_id`.
- `pdtx_auth_atom_id` (not mandatory)
Pointer to `_atom_site.auth_atom_id`.

The operation scheme of database population tool is demonstrated at the simple activity diagram, figure 6.1.

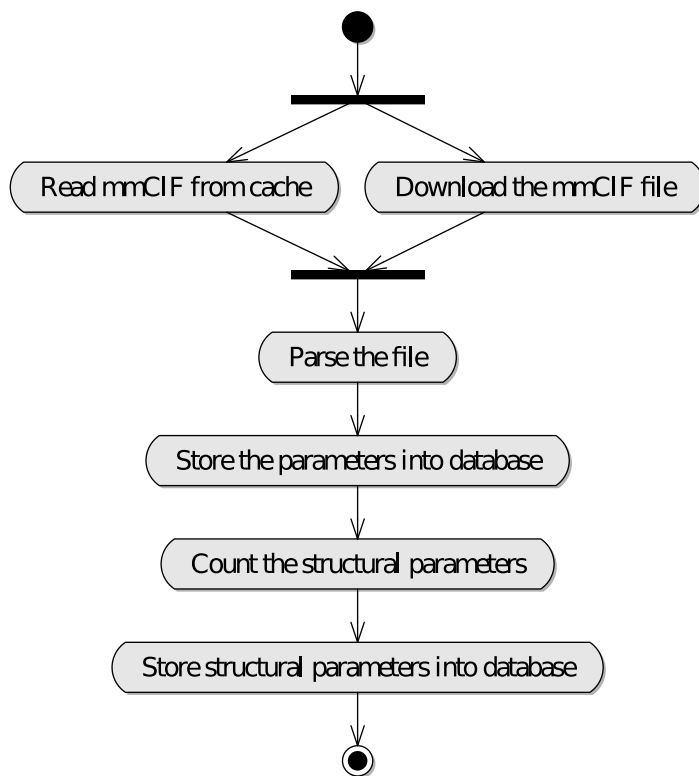


Figure 6.1: Activity diagram of the database population tool.

6.3 Database

The database consists of 9 main data tables – Biomolecule, BiomoleculeData, Chain, Residue, Atom, MethodNMRData, MethodXRayDataData, MethodXRAYData and StructureParameter – and 2 support tables Import and ImportQueue. There are also 10 tables derived from StructureParameter with same structure and relation – OverlapArea, OriginMeanNormalVector, HelicalParameter, StepClassification, C1GlobalParameter, TorsionAngle, SugarConformationalParameter, PCVirtualBondDistance, HelixRadiusRadialDisplacement, PositionAndLocalHelicalAxisVector.

There is possibility to simply “extend” the database scheme via user-defined custom data types/values by table CustomField.

6.3.1 Database Entities

The database entities are designed as:

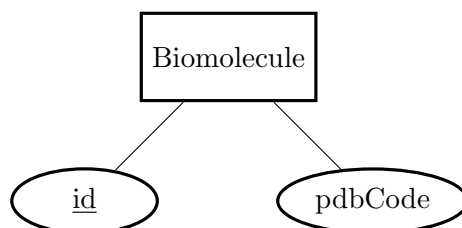


Figure 6.2: Entity: Biomolecule. Biomolecule is the main key of the scheme. It's addressed by users via pdbCode.

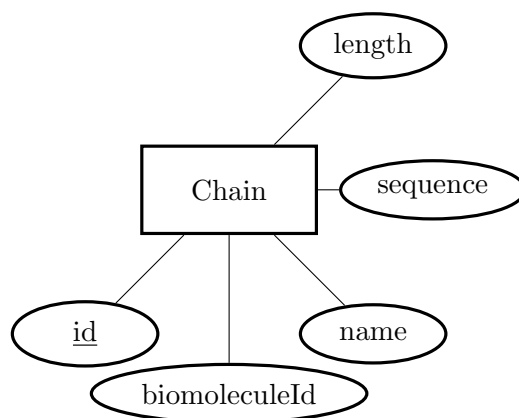


Figure 6.3: Entity: Chain. Chain is related to Biomolecule by biomoleculeId attribute.

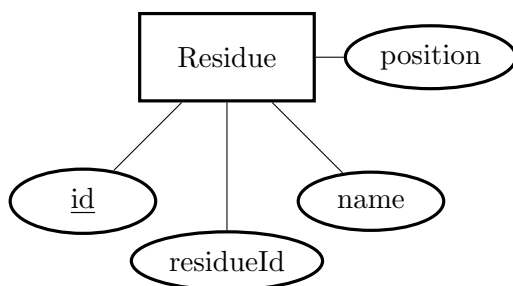


Figure 6.4: Entity: Residue. Residue is related to Chain by chainId attribute.

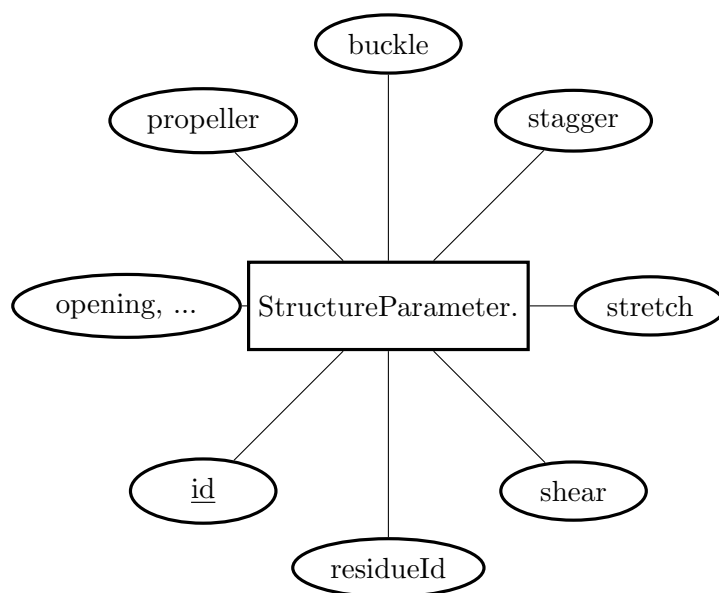


Figure 6.5: Entity: StructureParameter. StructureParameter is related to Residue by residueId attribute. The derived tables are very similar, only with specific different columns.

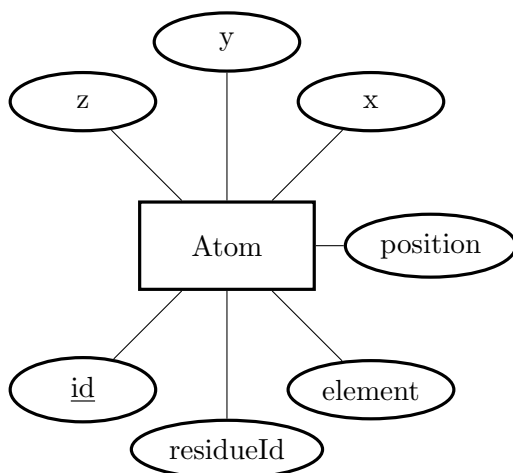


Figure 6.6: Entity: Atom. Atom is related to Residue by residueId attribute.

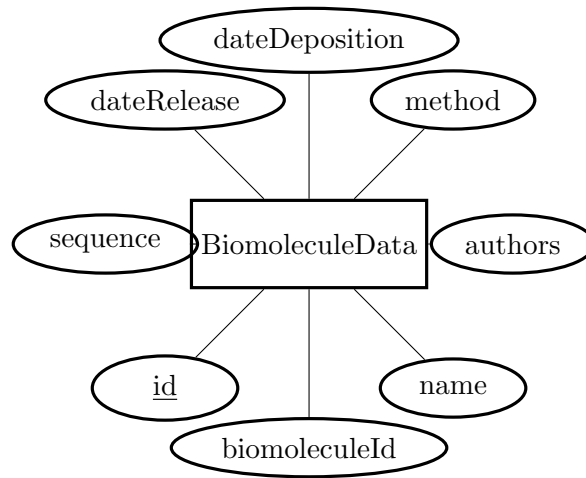


Figure 6.7: Entity: BiomoleculeData. BiomoleculeData is related to Biomolecule by biomoleculeId attribute.

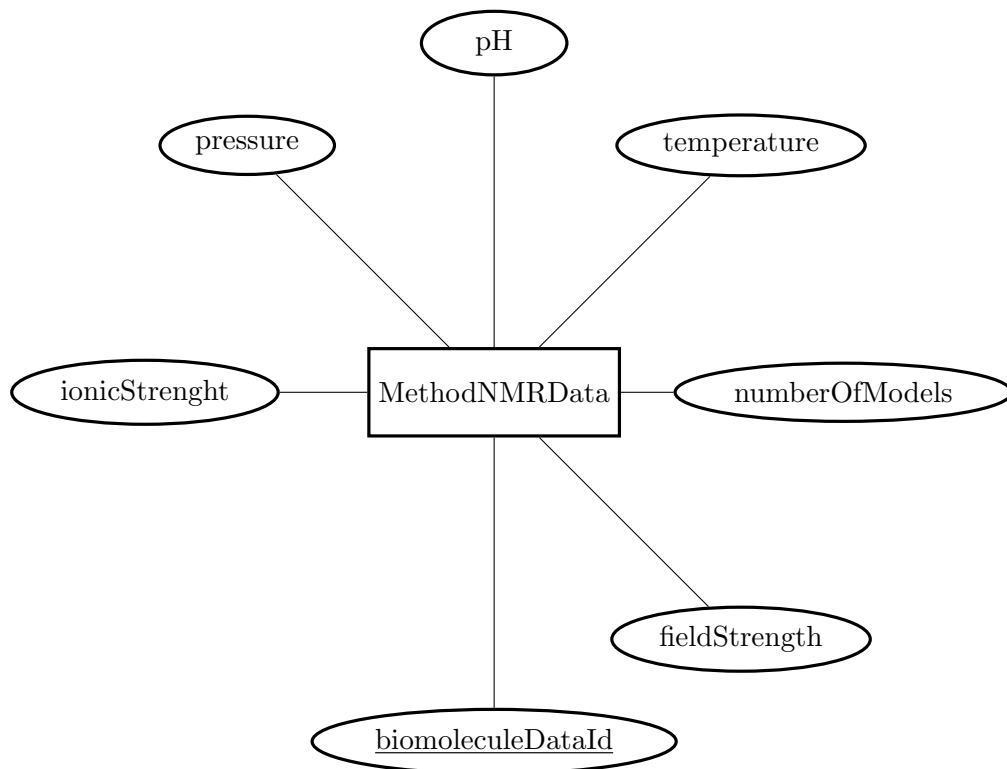


Figure 6.8: Entity: MethodNMRData. methodNMRData is related to BiomoleculeData by biomoleculeDataId attribute.

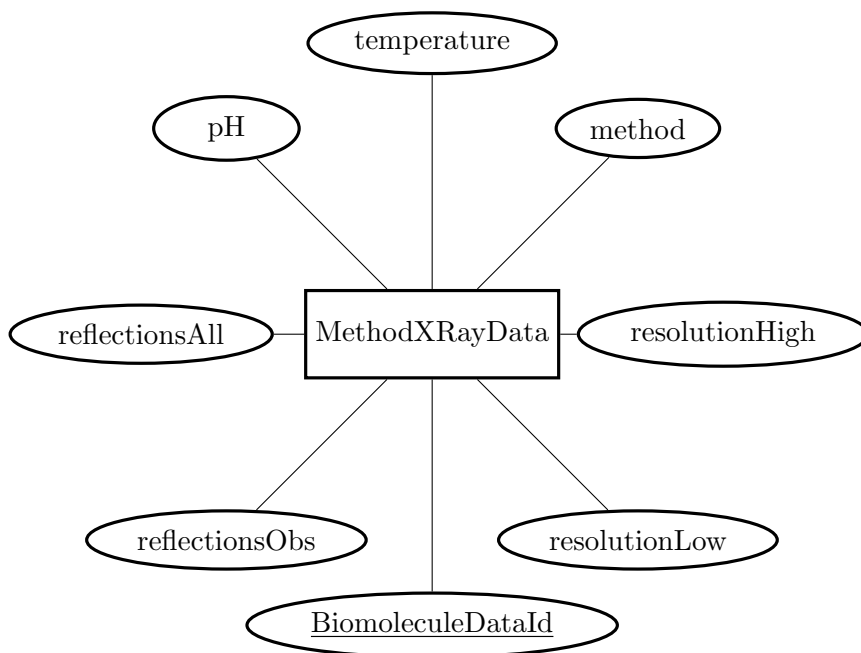


Figure 6.9: Entity: MethodXRayData. MethodXRayData is related to BiomoleculeData by biomoleculeDataId attribute.

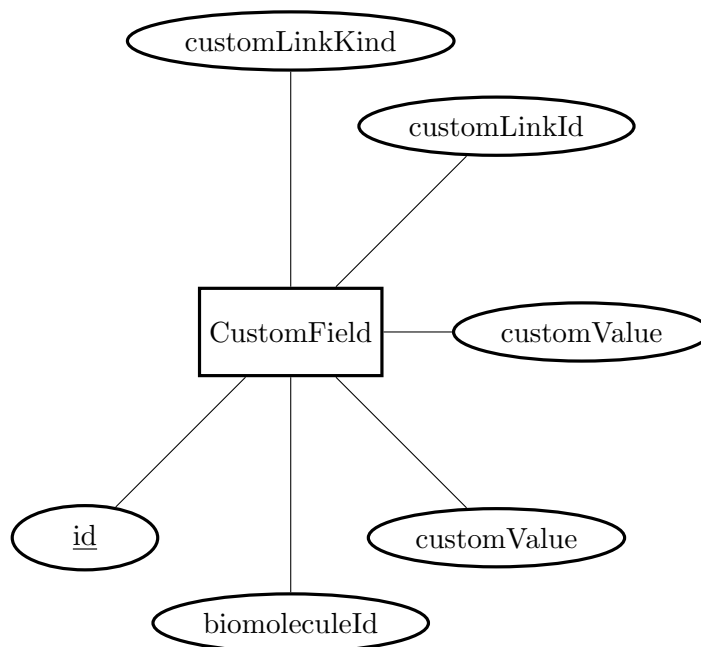


Figure 6.10: Entity: CustomField. CustomField is related to Biomolecule by biomoleculeId attribute.

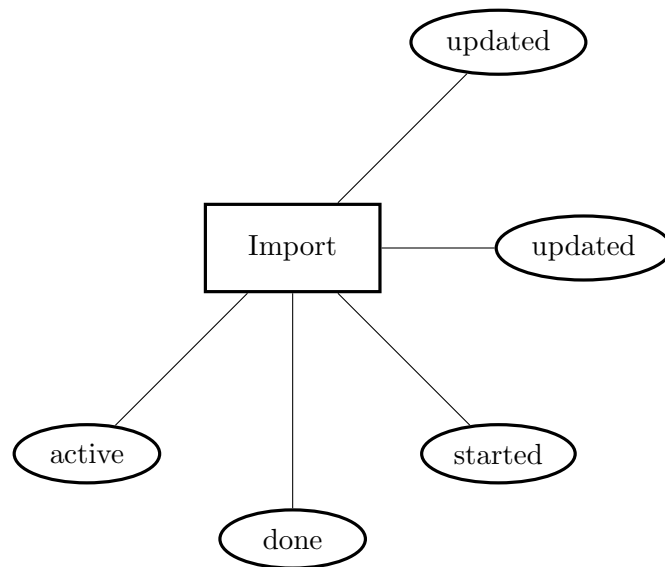


Figure 6.11: Entity: Import. Import is independent table without direct relation.

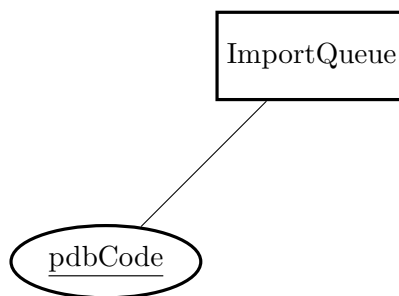


Figure 6.12: Entity: ImportQueue. ImportQueue is independent table without direct relation.

6.4 Web Application

The Web Application will serve as prototype use case scenario of the Nucleic Acids database. It will allow users to perform queries in user-friendly way with presentation of structure details including Jmol 3D chemical structure viewer.

6.4.1 Requirements

The requirements are derived from the database features. According to the general rules of software engineering, we have two groups of requirements – functional and non-functional. The functional requirements are directly describing needs for specific actions and sequences done by users or the application. The non-functional requirements are general, affecting typically the speed of the system, software requirements for user or application provider.

I have identified the following basic functional requirements:

- Possibility to query the basic structure parameters.
- Possibility to query extended parameters counted from 3DNA.
- Possibility to view details of the structure including link to PDB and 3D real-time view.
- Possibility to delete the not-needed structure including all relevant parts in the database.
- Possibility to browse the database and see the summary statistics.
- Possibility to import a list of structures.
- Possibility to check progress/results of import.
- Possibility to export data of structures to CSV file.

The non-functional requirements were identified as follows:

- From user point of view, possibility to run the application via web browser with/without Java and JavaScript support.
- From user point of view, possibility to get the results quickly withing few minutes (based on database size).
- From the system maintainer/provider point of view, possibility to run the application on server with GNU/Linux OS with option to use the other systems (like MS Windows server).

6.4.2 Use Cases

Based on the functional requirements, the use case scenarios for the whole application were created. Using UML Use Case diagram, the usage scenario is presented at figure 6.13 and more described in the next paragraph.

6.4.3 Performing Queries

The performing of queries is possible on the database of nucleic acids, the possible choices are simply list of all database attributes divided into four main groups – structure details (like structure name or date of deposition), XRay method details (like resolution or pH), NMR method details (like temperature or pressure) and query on structure (shear or twist).

The queries for attributes could be constructed using following operators:

- The attribute equals (=) given value.
- The attribute doesn't equal (!=) given value.
- The attribute is greater (>) than given value.
- The attribute is less (<) than give value.
- The attribute is greater than or equal to (>=) given value.
- The attribute is less than or equal to (<=) given value.
- The attribute has sub-string (LIKE) of given value.
- The value of attribute is not presented in database (IS NULL).
- The value of attribute is presented in database (NOT NULL).

Each two queries in the same group could be connected by logical operators AND and OR. The operator precedence is same as in PostgreSQL. That means the execution order from the top down is:

1. LIKE (checks for matching pattern comparison in a string).
2. < > <= >= (quantity comparisons).
3. = != (equality comparison).
4. AND (logical AND conjunction).
5. OR (logical OR conjunction).

The logic between query groups is based on logical AND conjunction which means the result array is the intersection of all partial results from all active groups.

6.4.4 Viewing Results

The results are displayed as a simple table with structures matching given criteria. The residues, which are part of the results, are highlighted by yellow background color.

6.4.5 Structure Details

From the query results or from direct browsing, user is able to get a structure details. It's the list of all attributes in the database including link to the PDB original biomolecule record with 3D real-time view on the structure itself.

6.4.6 Deleting Entries

The structures could be deleted from the database simply via action Delete in each table with structures. All relevant data like residues or atoms are deleted with parent entries as well.

6.4.7 Importing List of Structures

Comfortable import of structures is possible via text form of comma-separated PDB codes. This format is currently used at search results of Protein Data Bank, so user can simply copy&paste the desired results into form for import.

User can choose, if all structures in the database should be deleted prior inserting and also if newly imported structures should be overwritten by new.

The import will be then performed in the background as it can take long time to process.

6.4.8 Checking Progress and Results of Import

Progress of import is counted and displayed at progress screen, accessible from form to importing. User will be able to refresh the page simply to get latest data with simple statistics – count of imported structures and count of structures in the import queue.

6.4.9 Exporting Structure Data to CSV

Export of result data like structure parameters or also complete atoms specification is possible, the format for data exchange is Command-Separated Values (CSV), simple format with one-line header and values separated by commands [33], for example:

```
Year,Make,Model,Length  
1997,Ford,E350,2.34  
2000,Mercury,Cougar,2.38
```

6.4.10 Design of User Interface

The user interface must be designed always with the focus on simple and clear usage, in ideal scene according the user-must-not-know technique. User must not be interrupted by any buttons or fields which aren't part of the application. The first step in user interface design is creation of wireframes. Website wireframes, also known as a page schematics or screen blueprints, are basic visual representations of skeletal framework of the website. [34] The wireframe design is always prepared before real programming which makes the programmers' work easier, he can get the full overview of program's features and the application interface itself.

The wireframes are created by Balsamiq Mockups tool [35]. The concept of this tool is to give the customer interface overview in "comic style", in similar way as scanned notes from paper. The purpose is to present the basic application/interface structure, so this kind of the presentation is advantage – the user or the customer is then as even non-professional able to think and discuss the design features and not style specifics like button shapes or colors.

Although the simple style allows the user or customer simply understand the concept of the website interface, it could be also considered as disadvantage, for example when the wireframes are compared to full proposal of user interface from some user interface designer tool by user who doesn't see the difference between wireframe and graphical design. Such situations could be avoided also by Balsamiq Mockups, either by slight style change (Arial or Verdana font instead of default Comics Sans) or by extensions like Napkee [36] which allows to export the Mockups into functional pages based on HTML and JavaScript.

The user interface of the web application will be divided into few separated screens:

- Welcome screen,
- query form,
- query result table,
- database browse table,
- entry detail overview,
- delete of an entry,
- about screen,
- structure list import form,

- import progress and result form.

On the welcome screen at figure 6.14, we can see the basic page layout of the web application. The screen is split into two parts – the navigation (in real with some dark-color background) and the data space. The links from navigation will lead to tasks on database and basic information, the data part will be used for performing queries and showing the results.

The query form, at figure 6.15, is the main option for the users to get the specific data. The concept of the page is simple, users are able to pick from selects desired parameters, its operators and insert comparing values. The form is divided into four sections according database logic – query on basic details, queries on one of two main experimental methods and query on structural parameters.

If necessary, the form could be very easily extended by buttons on right side – each click will add a fresh new line for performing the extended query. The queries could be joined together again by selects, the logic between results could be set in AND/OR fashion.

The brief explanation about values and it's types will be written on the bottom of the page, just in case the users are starting with the tool and haven't seen the manual first.

User can perform the query using button in the bottom. After the data are processed, user is informed by the results via query results table from the figure 6.16, he gets the count of returned structures as well as the detail overviews in the table, PDB IDs, structure name and option to get more details.

From this screen, user is also able to perform a new query.

Browsing through the structures imported in database is also possible, user will see table very similar to results table, figure 6.17, to query results with PDB ID, structure name and same options. In the top of the pages, user is informed about status of database – statistics with structures quantity in the database.

The detail of the structure, accessible from query results or direct browsing gives the users two main groups of information – the full data from the database about structure itself and the Jmol representation of the chosen structure.

If necessary, user can delete the structure and all relevant entries via delete option in table which brings him into page from figure 6.19. The page has same layout as the last page – About, simple overview of prototype application and contact of maintainer.

Import of structures is possible via form at figure 6.20, including check boxes for import parameters.

From this importing form, user can access the overview with import progress or import results in case, when the importing is finished as demonstrated at figure 6.21.

6.5 Another Possibilities

All structures, application components and import tools were designed using author's best experience.

Although the proposed application design unites components using its function, we can easily imagine many other ways, as one example monolithic application with internal database, parsing and presentation features. Argument for the current design is compliance with Unix philosophy: "Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface." [37] [38].

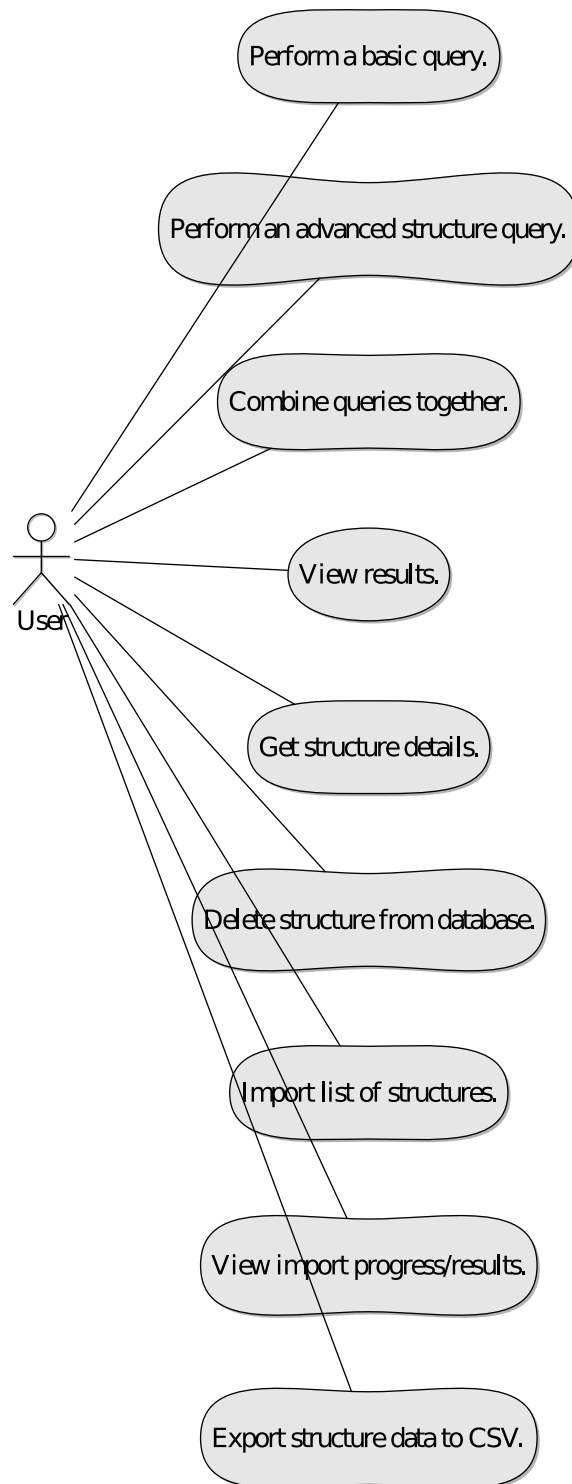


Figure 6.13: Web Application Use Case Diagram.

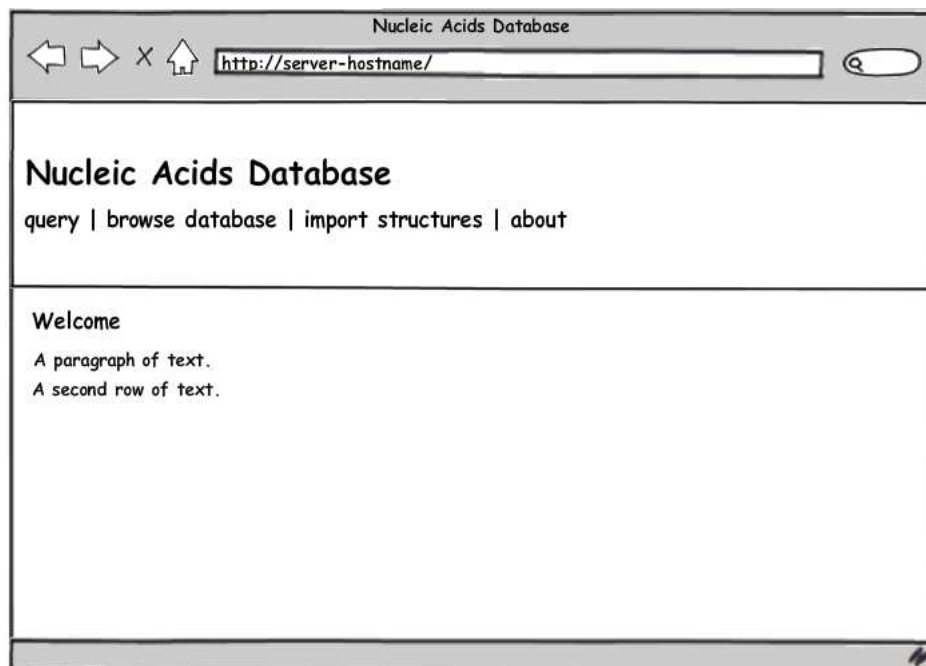


Figure 6.14: Wireframe: Basic screen layout, welcome screen.

The wireframe shows a web browser window titled "Nucleic Acids Database". The address bar contains "http://server-hostname/query". The main content area has a header with the title "Nucleic Acids Database" and a navigation menu with links: "query | browse database | import structures | about". Below the header is a section titled "Simply place a query using form." with four query forms:

- Query on detail**
 - Form 1: Name [dropdown] LIKE [dropdown] human [text] AND [dropdown] [button: Add detail query]
 - Form 2: Date of deposition [dropdown] >= [dropdown] 2001-01-01 [text] AND [dropdown] [button: Add query]
- Query on XRay method**
 - Form 3: Parameter [dropdown] = [dropdown] ... [text] AND [dropdown] [button: Add query]
- Query on NMR method**
 - Form 4: Parameter [dropdown] = [dropdown] ... [text] AND [dropdown] [button: Add query]
- Query on structure**
 - Form 5: Parameter [dropdown] = [dropdown] ... [text] AND [dropdown] [button: Add query]

At the bottom of the form section is a "Submit query" button.

Figure 6.15: Wireframe: Query form.

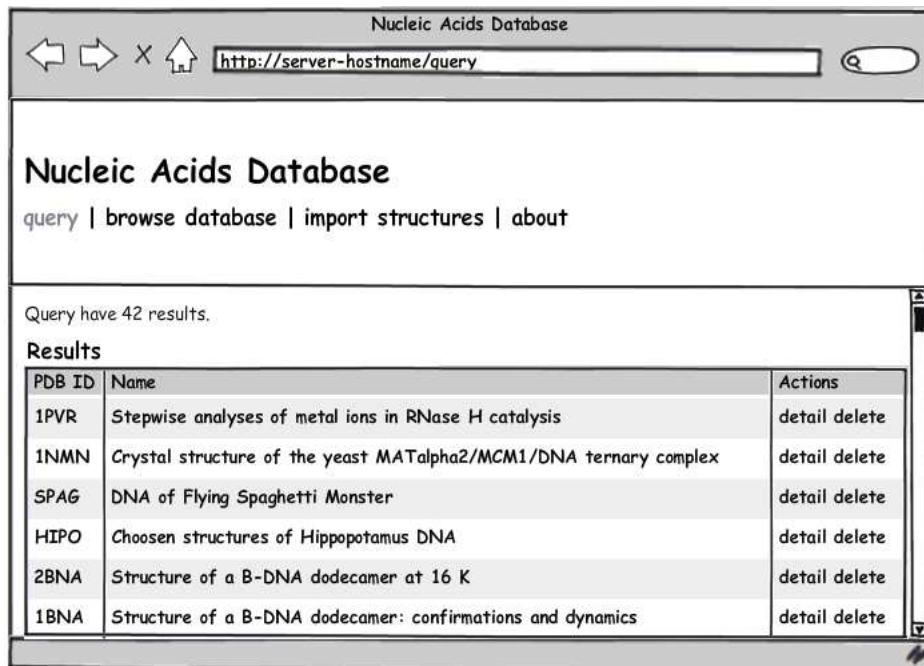


Figure 6.16: Wireframe: Table with query results.

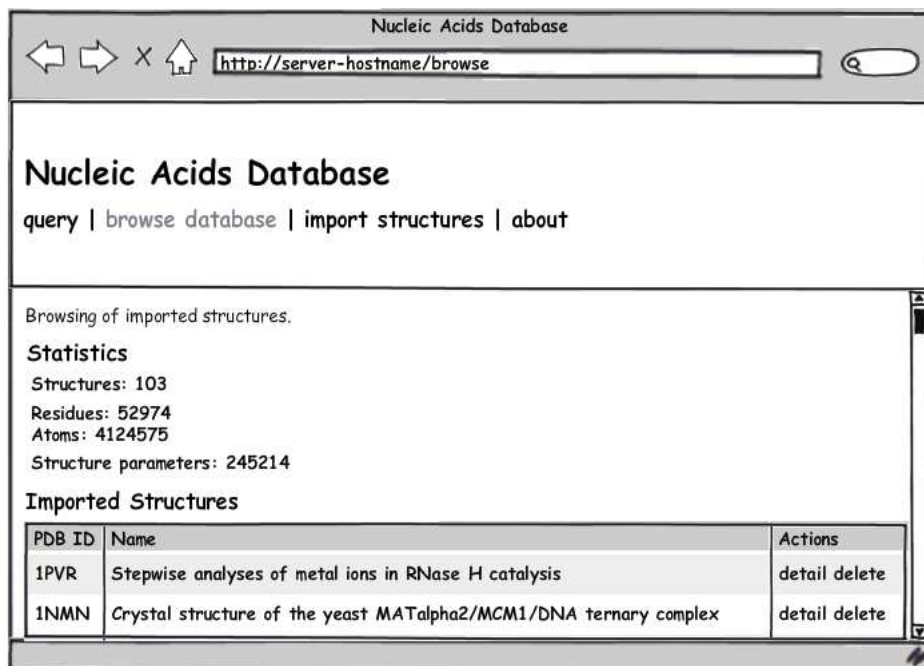


Figure 6.17: Wireframe: Table with query results.

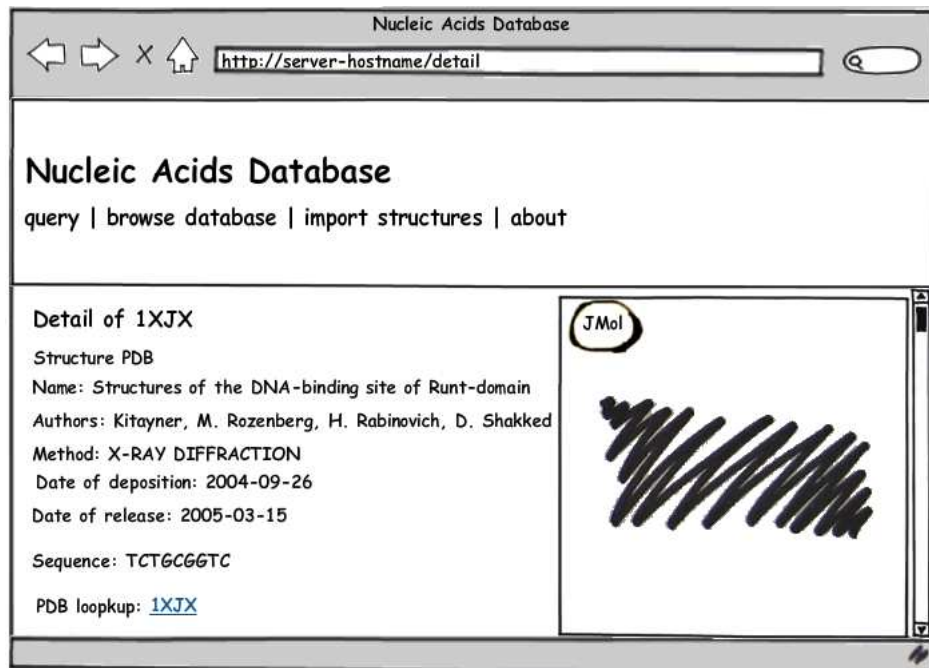


Figure 6.18: Wireframe: Structure detail with visualization.

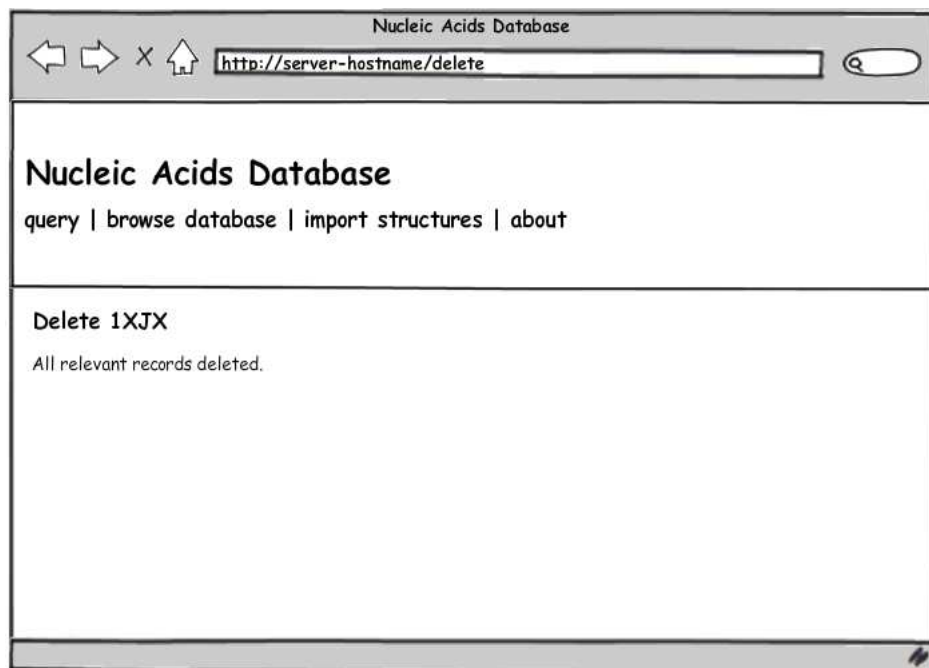


Figure 6.19: Wireframe: Delete of the structure and all relevant records.

The wireframe shows a web browser window titled "Nucleic Acids Database". The address bar contains "http://server-hostname/import". The page header includes the title "Nucleic Acids Database" and a navigation menu with links: "query", "browse database", "import structures", and "about". The main content area contains the instruction "Insert the PDB IDs of structures into form below." followed by an example: "Example: 1O1D, 17RA, 1FV8, 3BBV, 8ICP". Below this is a large, empty rectangular text input field. At the bottom of the form, there are two checkboxes: "Delete all structures before importing:" and "Overwrite existing structures:", both currently unchecked. A "Submit list" button is positioned to the right of these checkboxes.

Figure 6.20: Wireframe: Import list of the structures.

The wireframe shows a web browser window titled "Nucleic Acids Database". The address bar contains "http://server-hostname/progress". The page header includes the title "Nucleic Acids Database" and a navigation menu with links: "query", "browse database", "import structures", and "about". The main content area is titled "Importing progress" and displays the following information: "Imported started at 2011-04-30 12:14:01", "Import is currently being processed: yes", "Imported structures: 106", and "Pending structuers: 295".

Figure 6.21: Wireframe: Progress detail of pending import.

7 Implementation

In this chapter, the implementation of all needed software components is described. The first is the database definition, since the populating tool and web application are using it. Then the populating, data importing tool is introduced with tool for presentation as well. Software written for self-testing and other ways of verification are analyzed in the next independent chapter.

7.1 Database Definition

Database structure with all tables and entities was described above, in the previous chapter. We will use PostgreSQL, although there could be some specialties related only to PostgreSQL, the principles in database definition could be with small updates used also with other object-relational database management systems.

7.1.1 Data Types

The most important task is to choose appropriate data types of columns in database tables. The database requirements are based on the mmCIF dictionary, we will need 5 main types for data storage:

- Variable string with static length.
- Long string with variable length.
- Date.
- Integral number.
- Decimal, floating-point number.

We can assign the data types simply using PostgreSQL documentation [39]. The variable string with static length is defined as `CHARACTER VARYING(n)` or `VARCHAR(n)` – variable-length with limit *n*. Limited strings will be useful for storing PDB codes with limitation for exactly 4 characters, any over-length value will be truncated to *n* characters without raising an error as SQL standards require, so it must be used carefully.

Long string with variable length is defined as `TEXT`, variable unlimited length, which stores strings of any size. Although the type text is not in the SQL standard, several other SQL database management systems have it as well. We will use it for descriptions and other similar purposes.

The date definition is simple, we are interested in structure date of deposition and date of release, so we need simple type for storing date. The shortest is 4-byte `DATE`, date (no time of day). The range from 4713 BC to 5874897 AD will definitely cover our needs.

For integral numbers, we can choose mainly between 4-byte `INTEGER` and 8-byte `BIGINT`. The difference is in range of values, typical choice for integer is defined between -2.147.483.648 and +2.147.483.647, while large-range integer takes range from -9.223.372.036.854.775.808 to 9.223.372.036.854.775.807. The integral number will be used only for data about NMR method, number of models, chain length and residue position in the chain. In all these cases, the 4-byte type will be definitely fine, we could even consider 2-bytes, small-range integer `SMALLINT` with range from -32.768 to +32.767, but that could possibly lead into error in extreme situation e.g. with chain position in the future.

The floating-point number will be stored in inexact, variable-precision numeric types. In practice, these types are usually implementations of IEEE Standard 754 for Binary Floating-Point Arithmetic (single and double precision, respectively), to the extent that the underlying processor, operating system, and compiler support it. Inexact means that some values cannot be converted exactly to the internal format and are stored as approximations, so that storing and retrieving value might show slight discrepancies. The decimal data in mmCIF has usually 3 or 4 decimal places, so REAL type with a range of at least 1×10^{-37} to 1×10^{37} with a precision of at least 6 decimal digits on most platforms will be suitable enough.

The definition itself is then only question of SQL, as we said above all types except TEXT have SQL-standard spelling. For example the entity `MethodNMRData` will look like:

```
-- Entity MethodNMRData
CREATE TABLE MethodNMRData (
    biomoleculeDataId INTEGER PRIMARY KEY,
    fieldStrength REAL,
    numberOfModels INTEGER,
    temperature REAL,
    pressure TEXT,
    pH REAL,
    ionicStrenght TEXT
);
```

For the support tables, we will use two more datatypes:

- Timestamp.
- Small integer.

The only difference is, that `TIMESTAMP` (date and time) takes 8 bytes, twice as size of `DATE`.

7.1.2 Sequences

The links between tables are realized as integer relations – integer IDs pointing to IDs, usually generated primary keys. For this generating, we will use PostgreSQL sequences, similar, but not identical, to the `AUTO_INCREMENT` concept in MySQL.

PostgreSQL is using data types `SERIAL` and `BIGSERIAL`, aliases to automatically created `SEQUENCE`. For example, table defined as:

```
CREATE TABLE tablename (
    colname SERIAL
);
```

is equivalent to:

```
CREATE SEQUENCE tablename_colname_seq;
CREATE TABLE tablename (
    colname integer NOT NULL DEFAULT nextval('tablename_colname_seq')
);
ALTER SEQUENCE tablename_colname_seq OWNED BY tablename.colname;
```

(BIGSERIAL is same only with `bigint` instead of `integer`.)

To insert the next value of the sequence into the serial column, we will specify that the serial column should be assigned its default value. This can be done either by excluding the column from the list of columns in the `INSERT` statement, or through the use of the `DEFAULT` key word.

The `SERIAL` data type with possible 2 billions IDs should be enough, the biggest table for Atoms will require only approximately 30 millions unique identifiers.

The relation is then created using function `currval()` performed after insertion, for example:

```
INSERT INTO tablename (col1, col2) VALUES ('data1', 'data2');
SELECT currval('tablename_id_seq');
```

As we can see, we need to ensure that race conditions in this case are handled properly. If we had for example two database clients (in our case real scenario, more connections will be used for load balancing of database population tool) working with the same table (inserting data), the one of the clients could be interrupted by second client in the middle of this two commands – it could insert data into table with increasing the sequence number, but before reading of that value would be another insert performed by second client. PostgreSQL is avoiding such situation by special design of function `currval()`, it returns the last value generated by the sequence for the current transaction, so the value can't be affected by another client.

We will use sequences for all relations, that means effectively almost all tables.

7.1.3 Indexes

Indexes are primarily used to enhance database performance, it will save us time and CPU load during execution of queries with `WHERE` conditions. On our simple database structure with relations, index using B-tree will be good enough. B-tree¹ is a tree data structure that keeps data sorted and allows searches, sequential accesses, insertions, and deletions in logarithmic amortized time. [40] Another index possibilities are hash, GiST, and GIN. Users can also define their own index methods, but that is fairly complicated. [41]

In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-defined range. When data is inserted or removed from a node, its number of child nodes changes. In order to maintain the pre-defined range, internal nodes may be joined or split. Because a range of child nodes is permitted, B-trees do not need re-balancing as frequently as other self-balancing search trees, but may waste some space, since nodes are not entirely full. Searching is similar to searching a binary search tree. Starting at the root, the tree is recursively traversed from top to bottom. At each level, the search chooses the child pointer (sub-tree) whose separation values are on either side of the search value.

The indexes could be created simply by `CREATE INDEX` statements, for example at table `MethodXRayData`:

```
-- Entity MethodXRayData
CREATE TABLE MethodXRayData (
  biomoleculeDataId INTEGER PRIMARY KEY,
  resolutionLow REAL,
  resolutionHigh REAL,
  method TEXT,
  temperature REAL,
```

¹Not to be confused with binary tree which is a tree data structure in which each node has at most two child nodes, usually distinguished as “left” and “right”.

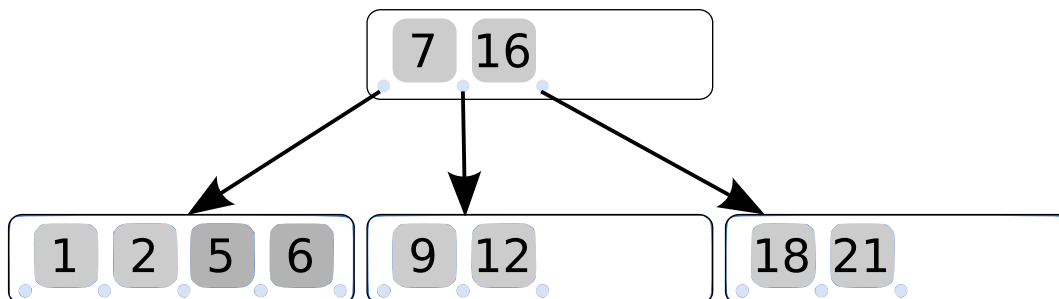


Figure 7.1: A small example of a 3-5 B-tree.

```

pH REAL,
reflectionsAll REAL,
reflectionsObs REAL
);

```

```

CREATE INDEX MethodXRayData_resolutionLow_idx ON MethodXRayData(resolutionLow);
CREATE INDEX MethodXRayData_resolutionHigh_idx ON MethodXRayData(resolutionHigh);
CREATE INDEX MethodXRayData_method_idx ON MethodXRayData(method);
CREATE INDEX MethodXRayData_temperature_idx ON MethodXRayData(temperature);
CREATE INDEX MethodXRayData_pH_idx ON MethodXRayData(pH);
CREATE INDEX MethodXRayData_reflectionsAll_idx ON MethodXRayData(reflectionsAll);
CREATE INDEX MethodXRayData_reflectionsObs_idx ON MethodXRayData(reflectionsObs);

```

7.1.4 User Data Types

PostgreSQL can be extended to support new data types which could be interesting for improving performance and related requirements to storage. But creating a new base type requires implementing functions to operate on the type in a low-level language, usually C. Such functions are compiled into dynamically loadable objects (also called shared libraries) and are loaded by the server on demand. Main attribute of compiled objects is the requirement for proper preparation – the object must be compiled (to all related platforms and all versions) which is quite difficult, definitely not worth increasing the possible speed in our case. [42]

7.1.5 Special Data Types

More interesting could be usage of one of the “special data types“, e.g. with spatial data support. Then we could also insert into database data with lower storage requirements, but mainly perform specialized queries on data (in a way “return all atoms *near* specified position”). Since the PostgreSQL has native support only for 2-dimensional data, we would need again some kind of extension like PostGIS [43] which is again quite complicated from maintenance point of view. In general, this way would be interesting for possible future extensions.

7.2 Data Import

Importing tool could be basically written in any type of programming language, this task is mainly about data analysis and related operation on the database connection. From the possible options to automatic analysis of mmCIF files mentioned in research above, I’ve chosen

mmLIB, Python Macromolecular Library, in spring of 2011 at version 1.0. [29] [30]

Since the main library is written in Python, I've chosen for the data importing and database population tool Python as well.

7.2.1 Software Requirements

The importing tool will require few packages to be installed – custom library mmLib, custom binaries from 3DNA and Python library Psycopg2. The tool is also using `wget` to get the data from URIs.

7.2.1.1 mmLib

In every environment, the importing tool expects installed mmLib. The installation could be done simply by following this steps:

1. We need to download the pymmlib package, available at <http://sourceforge.net/projects/pymmlib/files/>.
2. We check the mmLib dependencies and install any necessary modules.

```
python setup.py checkdeps
```

In my case with Debian GNU/Linux, packages `python-numpy`, `python-opengl`, `python-gtk` and `python-gtkglext1` had to be installed via package management system.

3. The module compilation and installation is then simple, we need to do as `root`:

```
python setup.py build
python setup.py install
```

The library is then installed at Python's `site-packages/` directory.

7.2.1.2 Psycopg2

We will need library for accessing the PostgreSQL database in easy way, I've chosen Psycopg – PostgreSQL database adapter for Python [44]. With package system, the installation is again pretty straight-forward, in Debian GNU/Linux we just need to install package `python-psycopg2`.

7.2.1.3 3DNA

Installation of 3DNA [17] tool was already described in the research, we should only mention there that since it's 32bit application, we would probably need 32bit user space libraries on 64bit operating systems. In Debian GNU/Linux it's provided by `ia32libs` package which needs to be installed.

7.2.2 Implementation

The implementation of importing tool has few basics features – working with database, data downloading, mmCIF analysis via mmLib and 3DNA data analysis.

7.2.2.1 Connection to Database

Connection to database is defined by `conn_string` from configuration at beginning of the script. After each SQL statement, `commit()` method must be called in order to perform the `COMMIT` in the transaction.

7.2.2.2 Data Download

Data are downloaded directly from PDB using `wget` command and URIs:

```
"http://www.rcsb.org/pdb/files/" + pdbId + ".cif.gz"
"http://www.rcsb.org/pdb/files/" + pdbId + ".pdb.gz"
```

Even with usage of small parts of PDB, the structures with DNA/RNA, we still need to download few gigabytes of data. To make the repeated downloading easier, the downloaded and decompressed files are stored in the `cache/` directory, so the file doesn't have to be downloaded twice or more.

7.2.2.3 mmLib Data Analysis

Analysis of mmCIF file with mmLib is quite easy, basically all we need is to analyze the data via method `LoadStructure`. In some cases, the extraction of data fails inside the mmLib, so we need to protect the calling by try-catch section with marks for verification script.

```
try:
    structure = LoadStructure(file="cache/" + pdbId + ".cif")
except:
    print "WARNING: LoadStructure failed"
```

To extract data from mmCIF is very easy with mmCIF dictionary. For example, if we want to extract list of authors from table `citation_author`, we simply call `Structure` method `cifdb.get_table`:

```
authors = structure.cifdb.get_table("citation_author")
authorList = []
for author in authors:
    if author["citation_id"] == "primary":
        authorList.append(author["name"])
```

In some cases, the data in expected tables are not filled in or even present (the non-mandatory fields). As a protection, I've defined function which is replacing the question mark symbol (?) for not present data as `None` values which will be then equivalent to `NULL` values in the database layer:

```
# function fixNA: handle not available values correctly
def fixNA(value):
    if value == "?":
        value = None
    return value

title = fixNA(structure.cifdb.get_table("citation")["title"])
```

Not even presented tables are handled directly by try-catch section:

```
try:
    reflectionsAll = fixNA(structure.cifdb.get_table("refine")["ls_number_reflns_all"])
    reflectionsObs = fixNA(structure.cifdb.get_table("refine")["ls_number_reflns_obs"])
except:
    reflectionsAll = None
    reflectionsObs = None
```

There are also few cases where is expected single floating-point number replaced by its range. For example if the pH of NMR method is 7.0-7.2 instead of some single value, we will adjust it and insert in relevant custom field:

```
if type(pH).__name__=='str':
    if (pH.find("-") != -1):
        phReal=pH[0:pH.find("-")]
        cursor.execute("INSERT INTO CustomField (biomoleculeId, customName, customValue) \
                        VALUES (%s, %s, %s)", (biomoleculeId, "MethodNMRData.pH.range", pH,))
        link.commit()
        pH = phReal
```

7.2.2.4 3DNA Data Analysis

Base pair and base step parameters are analyzed by programs from 3DNA package. During the implementation, two ways of accessing data were developed:

1. If possible, convert mmCIF file using mmLib into PDB file and analyze it using 3DNA with direct order of residues.
2. If analysis from above fails, download relevant PDB file and run 3DNA analysis on that with mapped order of residues.

The analysis in first case is done simply, the program creates temporary folder, export there PDB file via mmLib and run `find_pair` and `analyze` tools. Environment variable `X3DNA` must exists, it contains path to 3DNA installation:

```
SaveStructure(file=pdbId + ".pdb", structure=structure, format="PDB")

os.environ["X3DNA"]=basedir_3dna
returnCode = subprocess.call([basedir_3dna + "bin/find_pair", pdbId + ".pdb", \
pdbId + ".inp"])
if returnCode != 0:
    print "WARNING: find_pair return non-zero status when parsing structure ID "\
        + pdbId
returnCode = subprocess.call([basedir_3dna + "bin/analyze", pdbId + ".inp"])
if returnCode != 0:
    print "WARNING: find_pair return non-zero status when parsing structure ID "\
        + pdbId
```

In some cases, the structure data from 3DNA was directly parsable without any need for reordering the output. We only adjust the residue names from mmLib (so we will have it represented by single upper letter) and from 3DNA and check, if the sequence is matching.

```
for line in open('bp_step.par','r').readlines():
    data = re.split('\W+', line);

...

if (data[0] == "A") or (data[0] == "C") or (data[0] == "G") or (data[0] == "T") \
or (data[0] == "U"):
    # find all float numbers in the string
    parameter = re.findall(r"[+-]? *(?:\d+(?:\.\d*)?|\.\d+)(?:[eE][+-]?\d+)?", line)

...

# if the conditions match, perform the insert, but online in transaction, we will \
decide about commit/rollback later
if (save == "yes"):
    cursor.execute("INSERT INTO StructureParameter(residueId, shear, stretch, stagger, \
buckle, propeller, opening, shift, slide, rise, tilt, roll, twist) VALUES (%s, %s, \
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)", (id, parameter[0], parameter[1], \
parameter[2], parameter[3], parameter[4], parameter[5], parameter[6], parameter[7], \
parameter[8], parameter[9], parameter[10], parameter[11],))
else:
    print "WARNING: expectation of method 1 (" + "D" + data[0] + " == " + name + "/" + \
nameNew + ") about position failed, please check manually -- possible issue at 3DNA\
output!"
    done = False
```

As it appeared later, this method isn't working well for some special cases when the structure matches the presumption only by chance. Also the exported PDB file from mmCIF wasn't correctly usable, some data important for 3DNA wasn't same – the output was different with original and converted PDB file. That's why there is tool included only second method based on original PDB file and analysis in the final version of importing tool.

The idea of this new method is based on analysis of .inp from 3DNA to get correct chain and residue mapping.

For some structures, this mapping is equivalent to order in PDB file, for example PDB ID 1BNA, *Structure of a B-DNA dodecamer: conformation and dynamics*:

```
1BNA.pdb
1BNA.out
2          # duplex
12         # number of base-pairs
1  1      # explicit bp numbering/hetero atoms
1  24  0 #   1 | ....>A:...1_[.DC]C-----G[.DG]:..24_:B<.... 0.50 0.06 ...
2  23  0 #   2 | ....>A:...2_[.DG]G-----C[.DC]:..23_:B<.... 0.37 0.25 ...
3  22  0 #   3 | ....>A:...3_[.DC]C-----G[.DG]:..22_:B<.... 0.33 0.21 ...
4  21  0 #   4 | ....>A:...4_[.DG]G-----C[.DC]:..21_:B<.... 0.60 0.18 ...
5  20  0 #   5 | ....>A:...5_[.DA]A-----T[.DT]:..20_:B<.... 0.35 0.03 ...
```

```

6   19  0 #    6 | ....>A:...6_: [.DA]A-----T[.DT]:...19_:B<.... 0.19 0.17 ...
7   18  0 #    7 | ....>A:...7_: [.DT]T-----A[.DA]:...18_:B<.... 0.36 0.13 ...
8   17  0 #    8 | ....>A:...8_: [.DT]T-----A[.DA]:...17_:B<.... 0.34 0.10 ...
9   16  0 #    9 | ....>A:...9_: [.DC]C-----G[.DG]:...16_:B<.... 0.26 0.06 ...
10  15  0 #   10 | ....>A:...10_: [.DG]G-----C[.DC]:...15_:B<.... 0.40 0.27 ...
11  14  0 #   11 | ....>A:...11_: [.DC]C-----G[.DG]:...14_:B<.... 0.65 0.59 ...
12  13  0 #   12 | ....>A:...12_: [.DG]G-----C[.DC]:...13_:B<.... 0.60 0.26 ...
##### Base-pair criteria used:  4.00  0.00 15.00  2.50 65.00  4.50 ...
##### 0 non-Watson-Crick base-pairs, and 1 helix (0 isolated bps)
##### Helix #1 (12): 1 - 12

```

Some structures have completely different mapping, for example PDB ID 106D, *Solution structures of the i-motif tetramers of d(TCC), d(5methylCCT) and d(T5methylCC): novel NOE connections between amino protons and sugar protons*:

```

106D.pdb
106D.out
2          # duplex
6          # number of base-pairs
1   1      # explicit bp numbering/hetero atoms
6   12  0 #    1 | ...1>B:...6_: [.DT]T-***-T[.DT]:...12_:D<...1 8.42 0.05 ...
7    1  0 #    2 | ...1>C:...7_: [MCY]c-***-c[MCY]:...1_:A<...1 2.11 0.17 ...
5   11  0 #    3 | ...1>B:...5_: [.DC]C-***-C[.DC]:...11_:D<...1 3.59 0.74 ...
8    2  0 #    4 | ...1>C:...8_: [.DC]C-***-C[.DC]:...2_:A<...1 3.19 0.07 ...
4   10  0 #    5 | ...1>B:...4_: [MCY]c-***-c[MCY]:...10_:D<...1 3.24 0.07 ...
9    3  0 #    6 | ...1>C:...9_: [.DT]T-***-T[.DT]:...3_:A<...1 9.08 0.56 ...
##### Base-pair criteria used:  4.00  0.00 15.00  2.50 65.00  4.50 ...
##### 6 non-Watson-Crick base-pairs, and 1 helix (0 isolated bps)
##### Helix #1 (6): 1 - 6

```

And some structures have some irregularity after 1:1 mapping, for example PDB ID 10MH, *Mechanism of inhibition of DNA (cytosine C5)-methyltransferases by oligodeoxyribonucleotides containing 5,6-dihydro-5-azacytosine*:

```

10MH.pdb
10MH.out
2          # duplex
11         # number of base-pairs
1   1      # explicit bp numbering/hetero atoms
1   24  0 #    1 | ....>B:.402_: [.DC]C-----G[.DG]:.433_:C<.... 0.41 0.18 ...
2   23  0 #    2 | ....>B:.403_: [.DC]C-----G[.DG]:.432_:C<.... 0.57 0.16 ...
3   22  0 #    3 | ....>B:.404_: [.DA]A-----T[.DT]:.431_:C<.... 0.29 0.13 ...
4   21  0 #    4 | ....>B:.405_: [.DT]T-----A[.DA]:.430_:C<.... 0.24 0.15 ...
5   20  0 #    5 | ....>B:.406_: [.DG]G-----C[.DC]:.429_:C<.... 0.49 0.04 ...
6   19  9 #    6 x ....>B:.407_: [5CM]c-----G[.DG]:.428_:C<.... 0.72 0.47 ...
8   17  0 #    7 | ....>B:.409_: [.DC]C-----G[.DG]:.426_:C<.... 0.47 0.27 ...
9   16  0 #    8 | ....>B:.410_: [.DT]T-----A[.DA]:.425_:C<.... 0.25 0.01 ...
10  15  0 #    9 | ....>B:.411_: [.DG]G-----C[.DC]:.424_:C<.... 0.38 0.09 ...
11  14  0 #   10 | ....>B:.412_: [.DA]A-----T[.DT]:.423_:C<.... 0.59 0.23 ...
12  13  0 #   11 | ....>B:.413_: [.DC]C-----G[.DG]:.422_:C<.... 0.79 0.29 ...
##### Base-pair criteria used:  4.00  0.00 15.00  2.50 65.00  4.50 ...

```

```
##### 0 non-Watson-Crick base-pairs, and 2 helices (0 isolated bps)
##### Helix #1 (6): 1 - 6
##### Helix #2 (5): 7 - 11
```

The code for solving such issues is similar to the the first method introduced above, only extended with conversion table for residue position and chain to cover other than 1:1 mapping:

```
# get from .inp file new order of data
conversionArray = {}
conversionArrayChain = {}
try:
    for line in open(pdbId + '.inp', 'r').readlines():
        data = re.split('\W+', line);

        # find all float numbers in the string
        parameter = re.findall(r"[+-]? *(?:\d+(?:\.\d*)?|\.\d+)(?:[eE] [+-]? \d+)?", \
                                line)

        if (len(parameter) > 10):
            currentChain = line[line.find(">") + 1]
            conversionArrayChain[int(parameter[3])] = currentChain

            positionStart = line.find(">") + 3
            positionCurrent = positionStart
            positionEnd = line.find("_")
            numberBuilder = ""

            if (positionEnd == -1):
                print "WARNING: unidentified position of residue, skipping"
                continue

            while(positionCurrent != positionEnd):
                numberBuilder = numberBuilder + line[positionCurrent]
                positionCurrent += 1

            if(conversionArray.has_key(currentChain) == False):
                conversionArray[currentChain] = {}

            conversionArray[currentChain][int(parameter[3])] = \
                int(re.sub(r'\D', "", numberBuilder.strip(".")))
```

The code from original method is then extended with correct chain detection and residue position replacement in main loop over file `bp_step.par`:

```
currentChain=conversionArrayChain[offset]
if (conversionArray[currentChain].has_key(offset) != True):
    print conversionArray[currentChain]
    continue

cursor.execute("SELECT id, name, position FROM residue WHERE chainId=" + \
```

```
chains[currentChain] + " AND position=" +      \
str(conversionArray[currentChain][offset]) +";")
records = cursor.fetchall()
```

3DNA analysis ends on some structures with non-zero error code, or the required files aren't created – such situation are covered by try-catch block.

The rest of parameters are generated by analysis of file `.out` from 3DNA. This file have completely different structure that `bp_step.par` and `.inp` file, it's divided into sections by lines with stars:

```
*****
3DNA v2.0 [June 8, 2008] (by Dr. Xiang-Jun Lu; 3dna.lu@gmail.com)
*****
1. The list of the parameters given below correspond to the 5' to 3' direction
   of strand I and 3' to 5' direction of strand II.

2. All angular parameters, except for the phase angle of sugar pseudo-
   rotation, are measured in degrees in the range of [-180, +180], and all
   displacements are measured in Angstrom units.
*****
File name: 1BNA.pdb
Date and time: Tue May 10 01:58:06 2011

Number of base-pairs: 12
Number of atoms: 566
*****
```

This sections are easily parsable – the importing tool is separating it by simple state machine, which is changing states on line with stars. Please note the python construction `re.compile()`, the regular expression is compiled once and then used many times without need for new compilation or interpretation.

```
for line in open(pdbId + '.out', 'r').readlines():
    # find section delimiters
    pattern = re.compile("^\\*[76]{$");
    if (pattern.match(line)):
        state += 1
        inSameState = False
        regexpCond=False
```

The context of sections is then analyzed by second state machine, which is changing states on detection of known section label.

```
if (inSameState == False):
    if re.match("^Overlap area in Angstrom\\^2.*", line):
        stage += 1
        inSameState = True
    elif re.match("^Origin \\(Ox, Oy, Oz\\) and mean normal vector \\(Nx, Ny, Nz\\) \\
        of each base-pair.*", line):
```

```

    stage += 1
    inSameState = True
elif re.match("^Local base-pair helical parameters.*", line):
    stage += 1
    inSameState = True
elif re.match("^Classification of each dinucleotide step in a right-handed \
               nucleic acid.*", line):
    stage += 1
    inSameState = True

```

The data of relevant structure parameters are then saved during analysis of same section with correct context.

Lines from the section are divided by context variable which enables number parsing.

```

elif stage==6:
    # Main chain and chi torsion angles
    if (regexCond == True):
        ...
    elif re.match("^.base.alpha.beta.gamma.delta.epsilon.zeta.chi.*", line):
        regexCond = True

```

Activation of condition variable `regexCond` will initialize line analysis in similar way as parsing and saving base structure parameters. The typical short section looks like:

```

*****
Classification of each dinucleotide step in a right-handed nucleic acid
structure: A-like; B-like; TA-like; intermediate of A and B, or other cases

    step      Xp      Yp      Zp      XpH      YpH      ZpH      Form
1 Tc/cT      ---      ---      ---      ---      ---      ---      ---
2 cC/Cc      ---      ---      ---      ---      ---      ---      ---
3 CC/CC      0.18     0.32    -1.63    -0.48     0.19    -1.90
4 Cc/cC      ---      ---      ---      ---      ---      ---      ---
5 cT/Tc      ---      ---      ---      ---      ---      ---      ---
*****

```

The condition variable in this case will detect the header of the table (`Xp`, `Yp`, ...) and start with parsing next lines. The first column on each line is residue mapped with same scheme from `.inp` file via `conversionArray`.

```

parameter = re.split("?:? *", line)
if (len(parameter) < 10):
    # don't parse rest of section after hit on first strand
    inSameState = False
if (len(parameter) == 10):
    # get current residue.id
    offset = int(parameter[1])
    currentChain=conversionArrayChain[offset]
    if (conversionArray[currentChain].has_key(offset) != True):
        print conversionArray[currentChain]

```



```

        continue

    try:
        cursor.execute("SELECT id, name, position FROM residue WHERE chainId=" + \
                        + chains[currentChain] + " AND position=" + \
                        + str(convecurrentChain[offset]) + ";")
    except KeyError:
        # the residue is not in conversion table, very likely unknown
        continue
    records = cursor.fetchall()

    for record in records:
        id = str(record[0])
        position = str(record[2])

    # insert
    cursor.execute("INSERT INTO TorsionAngle(residueId, alpha, beta, gamma, \
                delta, epsilon, zeta, chi) VALUES (%s, %s, %s, %s, %s, \
                %s (id, fixNAreal(parameter[3]), fixNAreal(parameter[4]), \
                fixNAreal(parameter[5]), fixNAreal(parameter[6]), \
                fixNAreal(parameter[7])(parameter[8]), \
                fixNAreal(parameter[9]),))

    link.commit()

```

In cases, where the data are not presented (typically marked by ---), are the values inserted into REAL datatypes checked by function `fixNAreal`, similar function to `fixNA`.

```

# function fixNAreal: handle not available values correctly
def fixNAreal(value):
    if value == "---":
        value = None
    elif value == "----":
        value = None
    elif re.match("^---.*", value):
        value = None
    return value

```

7.3 Prototype Web Application

The web application should serve as a prototype example of Nucleic acids database usage. As designed in the previous chapter, it should give user the option to perform queries on data of imported structures in detailed display.

7.3.1 Software Requirements

I've chosen PHP, Hypertext Preprocessor, [45] as programming language for the dynamically generated web application for quick speed of development and very simple usage. The software requirements include PHP with module for connection to PostgreSQL database, without any special needs for specific PHP version or related web server – the application was tested and

developed on PHP 5.2.6 with Apache 2.2.9, but it will work also on e.g. any PHP 5.x with web server or PHP with FastCGI. As an addition, the application can store data for background import, the importing itself is done by shell script, periodically started from the system. That will mean additional requirement – some way, how to schedule and run the commands from the system. At Unix-based systems, this is usually task to “cron” system service.

7.3.2 Implementation

The implementation is done in four main parts – the application itself in PHP, providing the overall navigation and operation, then related supporting functions for dynamically extended form in JavaScript, related database queries in SQL and Jmol usage.

7.3.2.1 Overall Application

Configuration of web application is in file `config/config.php`, where connection to database and URI rewrite is defined, which is enabled by default. The address rewrite is very simple, all requests for non-existent pages should be redirected to file `index.php` with parameter for current page written in format `?page=`.

In Apache with `mod_rewrite`, this redirection could be done via this simple regular expression: (base directory is in `/srv/http/`)

```
RewriteCond /srv/http/%{SCRIPT_FILENAME} !-f
RewriteCond /srv/http/%{SCRIPT_FILENAME} !-d
RewriteRule ^(.*/)([^\.]*)$ /index.php?page=$2 [L,NS]
```

Operation of `index.php` is very simple – it generates the main layout of user interface and include pages from `pages/` directory according given parameter.

The main pages are **browse**, page with overview of structures imported in database, **delete**, page for deletion of one or more structures (all structures can be deleted via parameter **all**), **detail**, page for details about the structure including usage of Jmol applet with similar page **result** with result highlight, form for importing list of structures in page **import** linked with page for checking progress and results at page **progress** and **query**, the main part – interface for constructing and performing queries. Sample screenshot with query form is displayed at Figure 7.2.

7.3.2.2 JavaScript Form

JavaScript is used at page **query**, it generates additional fields of query groups via buttons “Add query detail”. The page is designed to be usable even with not working JavaScript support in browser, but then the user will be able to use the form in a way as it is from start.

Base code of the used JavaScript is function `addField()` for adding fields to form, the logic is simple – add a new element ending with serial suffix `_number`.

For example:

```
if ($kind == 'detail') {
    var tbody = document.getElementById("detail");
    var ctr = tbody.getElementsByTagName("input").length;

    var input;

    if (document.all) { //input.name doesn't work in IE
```

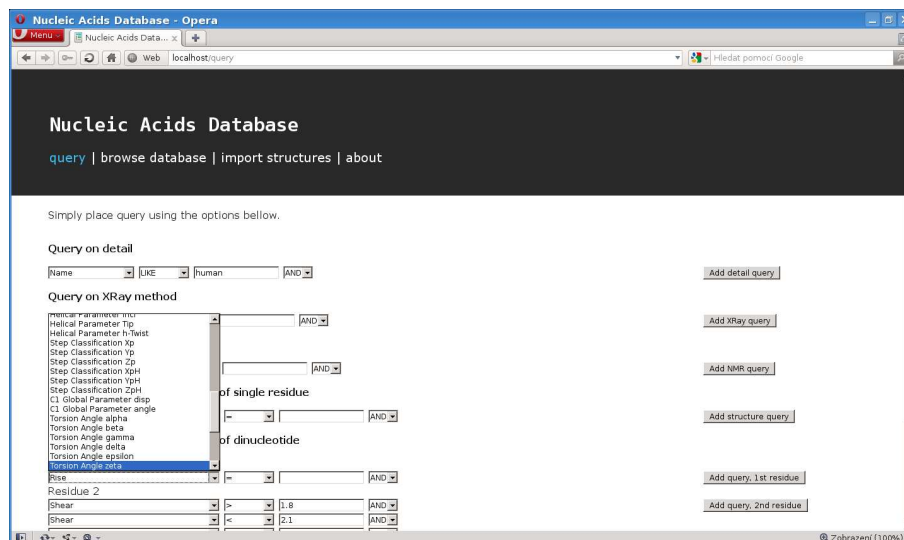


Figure 7.2: Screenshot of web application – the query form.

```

input = document.createElement('<input name="detail_'+ctr+'">');
} else {
input = document.createElement('input');
input.name = "detail_"+ctr;

input.id = input.name;
input.type = "text";
input.value = "";
input.className = "textfield";

cell.appendChild(input);
// insert white space
cell.appendChild(document.createTextNode(String.fromCharCode(160)))

```

7.3.2.3 Database Queries

The database queries are reconstructed from POST values sent by query form.

The logic of query is built for each group, the script first checks, if the input is non-empty or if the operation is NULL or NOT NULL, and if yes, it parses data for this group and also checks in cycle additional fields. The query is always reconstructed from four blocks, data source (projected as name of table), operation (SQL operation, e.g. equals), the input (data inserted by user used as condition) and logic operator for following group field (logical AND, logical OR).

Result of groups are saved into array `$results` and the final result is calculated by intersection of all saved group sub-results into final result.

The time spent on performing query is measured by function `microtime()`, application takes via this function timestamp before performing the database queries and after, the time spent is than simply calculated by deduction one time from the another.

Typical SQL query for structure containing in the name string “human”, deposited after beginning of year 2000 and having shear between 1.5 and 2 will look like:

```
SELECT biomoleculeId FROM biomoleculeData \
  WHERE name LIKE '%human%' AND datedeposition > '2000-01-01';
SELECT DISTINCT residueId FROM structureParameter \
  WHERE shear < '2' AND shear > '1.5';
```

The intersection is then done via additional SQL query constructed as:

```
SELECT DISTINCT biomoleculeId \
  FROM chain JOIN residue ON chain.id=residue.chainId \
  WHERE 1=0 OR residue.id=106657 OR residue.id=109315 OR residue.id=109397 \
    OR residue.id=109399 ...
```

If there are more structure parameters and therefor more tables to use, the join based on common attribute `residueId` is constructed. For example Shear structure parameter with main chain torsion angles and sugar conformational parameters:

```
SELECT StructureParameter.residueId \
  FROM StructureParameter
  JOIN TorsionAngle ON StructureParameter.residueId=TorsionAngle.residueId \
  JOIN SugarConformationalParameter \
    ON TorsionAngle.residueId=SugarConformationalParameter.residueId WHERE \
  shear > '0.27' AND shear < '3' AND alpha < '-60' AND tm < '49.8';
```

Query on dinucleotide parameters are processed simply, first parameters for both residues are prepared into associative array with residue chain, position and ID:

```
$query="SELECT $select.residueId, residue.chainId, residue.position FROM $from \
  WHERE $where;";

$result = pg_exec($link, $query);

$linkResultsResidue1 = array();
while($row = pg_fetch_row($result)) {
  $linkResultsResidue1[$row[1]][$row[2]] = $row[0];
}
```

Then, smaller link result array is used as searched needle in haystack

```
// evaluation of aswer -- selection to dinucleotides
$linkResults = array();

// choose smaller array to joining
if (count($linkResultsResidue1, COUNT_RECURSIVE) < \
count($linkResultsResidue2, COUNT_RECURSIVE)) {
  $linkResultsNeedle = $linkResultsResidue1;
  $linkResultsHaystack = $linkResultsResidue2;
  $linkResultOperationIsAdd = true;
} else {
```

```

$linkResultsNeedle = $linkResultsResidue2;
$linkResultsHaystack = $linkResultsResidue1;
$linkResultOperationIsAdd = false;
}

```

Searching criterion is simple, the residues must be next each other in correct order:

```

foreach ($linkResultsNeedle as $key => $value) {
    $position = key($value);

    if($linkResultOperationIsAdd){
        $position++;
    } else {
        $position--;
    }

    if ($linkResultsHaystack[$key][$position]){
        array_push($linkResults, $linkResultsHaystack[$key][$position]);

        array_push($highlightResidue, $linkResultsHaystack[$key][key($value)]);
        array_push($highlightResidue, $linkResultsHaystack[$key][$position]);
    }
}

```

7.3.2.4 Jmol Usage

The data of structure displayed at detail page is also demonstrated by Jmol applet. It's invoked using JavaScript library called jslibrary [46], it's starting and controlling the Java applet.

First, the JavaScript library itself is loaded, it's done in head of HTML page.

```
<script src="/js/Jmol.js" type="text/javascript"></script>
```

Then, the applet is initialized, configured and pointed at mmCIF file stored at `files/` directory, where the importing script keeps copies of the imported files, or where the data is copied by the maintainer of application. If the importing tool and web application are on same system, it's good practice to have data on one place – then the directory `files/` could be changed to symbolic link to `cache` folder.

```

<div id="jmol">
<script type="text/javascript">
jmolInitialize("/jar");
jmolApplet(500, "load files/<?php echo $parameter; ?>.cif");
jmolSetCallback("language", "en");
</script>
</div>

```

The result residues are in Jmol applet highlighted by bigger atoms, first the exact position of needed residues are selected:

```

$molHighlight = "";

foreach ($highlightResidues as $highlightResidue) {
    $query = "SELECT DISTINCT residue.position \
        FROM atom JOIN residue ON atom.residueId=residue.Id \
            JOIN chain ON residue.chainId=chain.Id \
                JOIN biomolecule ON chain.biomoleculeId=biomolecule.id \
        WHERE pdbCode='$parameter' AND residue.id=" . $highlightResidue . " ";
    $result = pg_exec($link, $query);
    $row = pg_fetch_row($result);
    if ($row[0]) {
        $molHighlight .= "select " . $row[0] . " "; spacefill 0.75; "
    }
}

```

Then the result is given to Jmol applet via jslibrary. [47]

```

jmolScript('select all; spacefill 0.4;');
jmolScript('<?php echo $molHighlight; ?>');

```

The user is able to change the highlighted view simply using proper buttons under the Jmol applet, the JavaScript call is similar to `jmolScript()` call, this action is invoked by `onClick()` event.

```

jmolRadio('select all; spacefill 0.4;', "Default view", false );
jmolRadio('<?php echo $molHighlight; ?>', "Highlight matching residues", false );

```

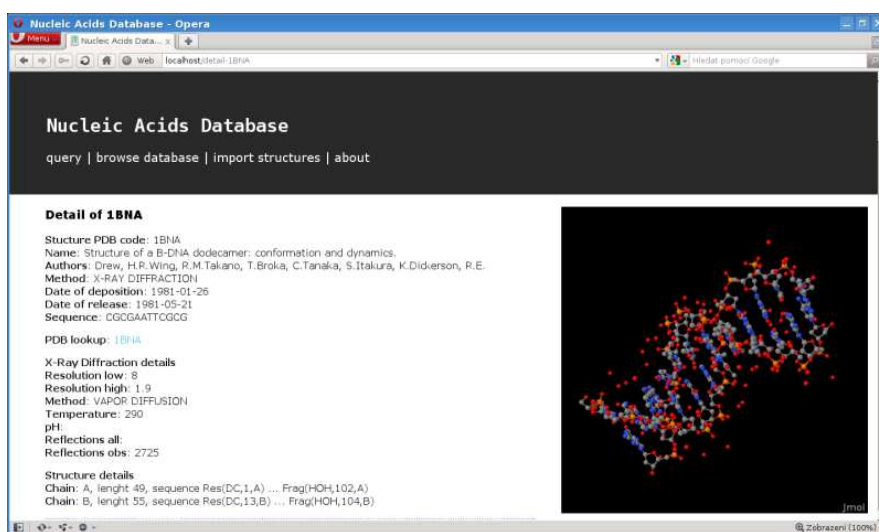


Figure 7.3: Screenshot of web application – detail with Jmol demonstration.

7.3.2.5 Background Imports Queue

User is able to setup an import of defined list of structures. The structures are separated by commas, as this format is typically used at Protein Data Bank at search results. The list is parsed and stored into database queue in table `ImportQueue` which is periodically checked by scheduled shell script – worker.

The details about import queue and its progress is stored in table `Import` which is used by import worker as well as progress page.

7.3.2.6 Background Imports Worker

The background worker is simple shell script periodically checking size of queue in table `ImportQueue`. If it's greater than zero, it updates status at table `Import` and invokes import script with proper parameter – the PDB code of structure.

The scheduled action could be set very simply at Unix-like systems via cron, for example by this 5-minute check defined as part of `crontab`:

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=ondrej@cecak.cz
HOME=/
```

```
# NADB Background Worked
*/5 * * * *      oc      /home/oc/nadb/cron/worker.sh
```

```
# vim:set syntax=crontab nowrap:
```

The configuration of shell script is written in the beginning of script. The database access is defined in PostgreSQL `.pgpass` file at home directory (or different directory given to shell script by `$HOME` environmental variable).

The importing worker script is implemented as simple loop over the queue.

```
if [ $count -gt 0 ]; then
  while [ $count -gt 0 ]; do
    echo "UPDATE import SET active=1;" | psql -q -Unadb

    echo "SELECT pdbCode FROM importQueue LIMIT 1;" | psql -q -Unadb \
    | head -n3 | tail -n1 | sed 's/ *//' | while read pdbCode; do
      echo "DELETE FROM importQueue WHERE pdbCode='$pdbCode';" | psql \
      -q -Unadb
      cd $pathToImport || exit 1
      ./import.py $pdbCode | tee -a $pdbCode.log
    done
    echo "UPDATE import SET done=done+1;" | psql -q -Unadb

    count="'echo \"SELECT COUNT(*) FROM importQueue;\" | psql -q -Unadb \
    | grep -v row | grep '[0-9]' | sed 's/ *//''"
  done

  echo "UPDATE import SET active=0,finished=NOW();" | psql -q -Unadb
fi
```

The multiple invoking from cron is protected by simple locking via files.

```
# lock test
if [ -e ~/cron-bin/stamps/nadb-worker ]; then
    echo "ERROR: Stamp exists."
    exit 1
fi

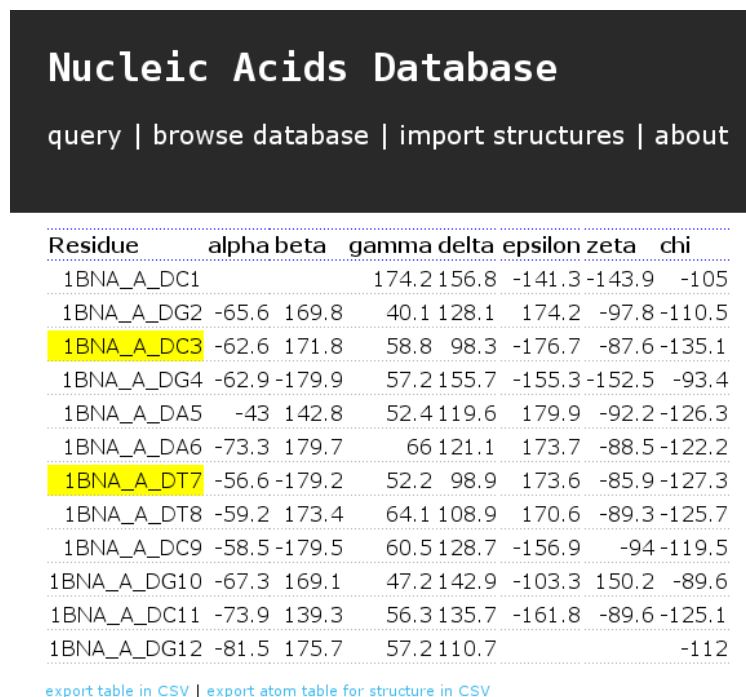
# lock acquire
touch ~/cron-bin/stamps/nadb-worker
```

7.3.2.7 CSV Export

Option to export data into CSV is available under each result table. The export itself is done by script `export.php` with two parameters – PDB structure ID `id` and table `table` for export.

The result is directly written in raw output as `text/csv` content type, so typical web browser ask user to download file by download manager or directly allows to open the file in correct software (for example Excel from Microsoft Office or Calc from OpenOffice.org).

```
header('Content-Type: text/csv');
header('Content-Disposition: attachment; \
    filename="nadb-' . $pdbid . '-' . $table . '.csv"');
```



Residue	alpha	beta	gamma	delta	epsilon	zeta	chi
1BNA_A_DC1			174.2	156.8	-141.3	-143.9	-105
1BNA_A_DG2	-65.6	169.8	40.1	128.1	174.2	-97.8	-110.5
1BNA_A_DC3	-62.6	171.8	58.8	98.3	-176.7	-87.6	-135.1
1BNA_A_DG4	-62.9	-179.9	57.2	155.7	-155.3	-152.5	-93.4
1BNA_A_DA5	-43	142.8	52.4	119.6	179.9	-92.2	-126.3
1BNA_A_DA6	-73.3	179.7	66	121.1	173.7	-88.5	-122.2
1BNA_A_DT7	-56.6	-179.2	52.2	98.9	173.6	-85.9	-127.3
1BNA_A_DT8	-59.2	173.4	64.1	108.9	170.6	-89.3	-125.7
1BNA_A_DC9	-58.5	-179.5	60.5	128.7	-156.9	-94	-119.5
1BNA_A_DG10	-67.3	169.1	47.2	142.9	-103.3	150.2	-89.6
1BNA_A_DC11	-73.9	139.3	56.3	135.7	-161.8	-89.6	-125.1
1BNA_A_DG12	-81.5	175.7	57.2	110.7			-112

export table in CSV | export atom table for structure in CSV

Figure 7.4: Screenshot of web application – highlighted results and option for CSV export.

7.4 Another Possibilities

We can imagine a lot of another possible implementations of the given task, but all of them will be based on similar tasks scheme – data download, analyzing and indexed storage. From this point of view, the given implementation is one functional example, demonstrating more algorithms than specifics of programming language, the same result could be easily archived using Java or .NET instead of Python etc.

Arguments for specific technologies are mentioned above, mainly Python’s Macromolecular Library support and PostgreSQL features.

8 Verification

As Douglas Adams said, we should first see, later think and then test. But always see first. Otherwise we will only see what we are expecting.

8.1 Overview Before Test

Let's see first. We have a database structure for storing data about nucleic acids. We have a web application for performing queries and displaying results. We have a tool for importing data in mmCIF and PDB format. We have a source of data in mmCIF and PDB – the RCSB PDB, Protein Data Bank.

Let's think now. We can take all the relevant data from PDB (that will in fact cause massive testing), import it into database and then perform ten completely different queries over the database using web application.

So the test will be carried out using this sequence:

1. Download data of all structures with DNA and RNA from Protein Data Bank.
2. Import all data into database via importing tool.
3. Perform ten different queries over database using web application.
4. Check the results and verify details.

And the presumption of the tests:

- All structures from PDB will be imported into database without errors in import tool or database scheme.
- All ten different queries over database will be performed by web application without error.
- All results will be displayed correctly (including Jmol applet) and result data will fit the query conditions.

8.2 Import of Nucleic Acids

Our first step will be relevant data download from the Protein Data Bank. This could be done simply by accessing PDB homepage at <http://www.rcsb.org/pdb/> and using following tool (according navigation on page from end of April 2011):

- Locate left menu “Search”,
- choose “Advanced search”,
- change “Choose Query Type” as “Structure Features” – “Macromolecule Type”,
- change search condition “Contains DNA” from “ignore” to “Yes”.
- Submit query.
- Choose “Download selected” and download all data in compressed mmCIF.
- Repeat the process for all structures which contain RNA.

8.2.1 Verification Script

At the end of April, the data set of all structures which contains DNA or RNA has 5316 items with total size of uncompressed mmCIF approximately 7 gigabytes.

The verification import was done by simple script written in BASH. It calls importing tool `import.py` for every structure from the `import/cache/` directory, saving it's output and searching specific strings in it.

The verification part of script is constructed via following statements:

```
grep 'Time used' out.err.$$ > /dev/null
if [ $? -eq 0 ]; then
    echo -n "OK"
else
    echo -n "ERROR"
fi

grep 'VERIFY: OK' out.$$ > /dev/null
if [ $? -eq 0 ]; then
    echo " (DOUBLE CHECKED)"
else
    grep 'WARNING: expectation of method 2' out.$$ > /dev/null
    if [ $? -eq 0 ]; then
        echo " (3DNA)"
    else
        grep "find_pair return non-zero status" out.$$ > /dev/null
        if [ $? -eq 0 ]; then
            echo " (3DNA)"
        else
            grep 'WARNING: LoadStructure failed' out.$$ > /dev/null
            if [ $? -eq 0 ]; then
                echo " (mmLIB)"
            else
                echo " (SUSPECTED)"
                cp out.$$ out.$file
                cp out.err.$$ out.err.$file
            fi
        fi
    fi
fi
fi
```

The output of the verification script looks like

```
2QEK OK (DOUBLE CHECKED)
2QEX OK (3DNA)
2QFJ OK (3DNA)
2QH2 OK (DOUBLE CHECKED)
2QH3 OK (DOUBLE CHECKED)
2QH4 OK (DOUBLE CHECKED)
2QHB OK (DOUBLE CHECKED)
2QK9 OK (DOUBLE CHECKED)
2QKB OK (DOUBLE CHECKED)
2QKK OK (DOUBLE CHECKED)
```

```

2QL2 OK (DOUBLE CHECKED)
2QNC OK (DOUBLE CHECKED)
2QNF OK (DOUBLE CHECKED)
2QNH ERROR (mmLIB)
2QOJ OK (DOUBLE CHECKED)
2QOU OK (DOUBLE CHECKED)

```

First is the name of structure, then result (OK or ERROR) and detail (DOUBLE CHECKED as OK, mmLIB as problem with usage of mmLib and 3DNA as problem with usage of 3DNA).

The verification script could be run in parallel – the importing has perfect load balancing options, the structures to parse could be distributed to as many nodes as you have. Approximate time of run on one typical CPU core will be counted in days.

8.2.2 Errors

As it appeared during testing, there were two types of errors on tested structures – errors in mmLib and errors in 3DNA.

mmLibs errors are usually caused by error during parsing the mmCIF and creating the Structure object. For example, structure PDB ID 1AMD, *NMR STUDY OF DNA (5'-D(*TP*GP*TP*AP*CP*A)-3') SELF-COMPLEMENTARY DUPLEX COMPLEXED WITH A BIS-DAUNORUBICIN WP-652, MINIMIZED AVERAGE STRUCTURE* fails on

```
structure = LoadStructure(file="cache/" + pdbId + ".cif")
```

with stack trace

Traceback (most recent call last):

```

File "./test.py", line 65, in <module>
    structure = LoadStructure(file="cache/" + pdbId + ".cif")
File "/usr/lib/python2.5/site-packages/mmLib/FileIO.py", line 121, \
in LoadStructure
    return mmCIFStructureBuilder(**args).struct
...
File "/usr/lib/python2.5/site-packages/mmLib/Library.py", line 416,\
in library_construct_monomer_desc
    atom1 = cif_row.getitem_lower("atom_id_1")
File "/usr/lib/python2.5/site-packages/mmLib/mmCIF.py", line 82, \
in getitem_lower
    return dict.__getitem__(self, clower)

```

KeyError: 'atom_id_1'

There are also structures, where 3DNA tool is failing, for example PDB ID 1F6U, *NMR structure of the HIV-1 nucleocapsid protein bound to stem-loop sl2 of the psi-RNA packaging signal. Implications for genome recognition*

```

..... reading file: baselist.dat .....
unknown residue CG1 201 on chain B [#57]
Check base [CG1] and add it to file <baselist.dat>
See the FAQ section at http://3dna.rutgers.edu/ for details

```

There are also structures, where the 3DNA hasn't produced any output, but the error wasn't found (marked as OK (3DNA), this can occur when the 3DNA hasn't found any base pairs to count on, for example PDB ID 3HZI, *Molecular mechanisms of HipA-mediated multidrug tolerance and its neutralization by HipB*

handling file <3HZI.pdb>

```

..... reading file: misc_3dna.par .....

..... reading file: baselist.dat .....
uncommon residue ATP 500 on chain A [#516] assigned to: a

..... reading file: atomlist.dat .....

..... reading file: atomlist.dat .....
no base-pairs found for this structure

```

8.2.3 Import Results

The overall import results are following:

- 4776 structures imported with 3DNA data
- 325 structures imported without 3DNA data (e.g. no base pairs found)
- 215 structures not imported due to mmLib parsing error

That means we have imported approx. 95.96 % structures into database successfully. The time spent on importing was measured using BASH built-in command `time`, there were 3 threads on 64bit Linux server with Intel Core2 Quad CPU 2.66 GHz:

```

%real    1008m41.753s
%user    152m8.390s
%sys     11m54.713s

%real    1433m6.321s
%user    283m0.305s
%sys     18m7.180s

%real    1823m32.422s
%user    335m24.558s
%sys     22m11.027s

```

Since the server done also tasks for other users, the total time spent on importing is summary of `user` (time in user mode) and `sys` (time in system time) lines. For our 5316 parsed structures we have approx. 12.5 hours of user mode time and approx. 52 minutes in system mode which give us average import time of one structure 0.153 seconds. Please note that this time is very variable, on one hand quickly parsing could be done in much less time, on the other hand the complex structures take long time for mmLib and 3DNA analysis.

In total, the statistics from import:

- 5,101 structures
- 4,303,860 residues
- 36,071,202 atoms
- 12,600,746 structure parameters

- 1,905 megabytes in SQL data dump (with COPY statements), 604 megabytes in SQL dump compressed via gzip

All imported data could be correctly displayed by the web application including 3D view in Jmol.

8.3 Testing

The result database contains data which could be easily tested using defined functional and non-functional requirements at section with design of web application.

8.3.1 Search Results Verification

Verification of search result could be done easily, user can see in the structure overview or detail, if it matches given criteria.

For example, we will formulate query for all structures with “human” in structure title, date of deposition between year 2000 and 2010 and shear of two residues between 0.10 and 0.20.

The query will be written in this form:

```
Name LIKE "human" AND
Date of deposition >= "2000-01-01" AND
Date of deposition "2011-01-01"
Shear >= 0.10
Shear <= 0.20
```

Result: query on detail returns 158 results, query on structure returns 3048 results; intersection will give us 81 results in 0.98 seconds. All matches given query criteria.

Similar story was for previous testing of 9 similar tests, all were correct.

8.3.2 Technology Requirements

The technology requirements are simple, PostgreSQL (with PHP) on server, JavaScript and Java support in client’s browser. One of the requirements on application was also possibility to work with web application without Java and JavaScript – the query form is working, but only with one possible query for each section.

8.3.3 Speed of Search

Speed of searching in the database depends mainly on database size. But even with the database with all nucleic acids the queries are quick enough. For example, let’s analyze database query plan for previous sample query.

At first, we will have to find all `biomoleculeId` for structures with “human” string in `name` and with proper `dateDeposition` limitation. Table `biomoleculeData` is not big, so the database will do simple sequential scan:

```
EXPLAIN SELECT biomolecule
FROM biomoleculeData
WHERE name LIKE '%human%' AND dateDeposition >= '2000-01-01' AND \
      dateDeposition < '2011-01-01';
      QUERY PLAN
```

```
Seq Scan on biomoleculadata (cost=0.00..332.28 rows=26 width=4)
```

```
Filter: ((name ~~ '%human% '::text) AND (datedeposition >= \
'2000-01-01'::date) AND (datedeposition < '2011-01-01'::date))
```

This will be first temporary result. Then we will have to find all `residueId` with given structure `shear` parameter using index on `shear` and bitmap heap scan:

```
EXPLAIN SELECT residueId
FROM structureParameter
WHERE shear >= 0.1 AND shear <= 0.2;
QUERY PLAN
-----
Bitmap Heap Scan on structureparameter (cost=650.36..4055.58 rows=26348 width=4)
  Recheck Cond: ((shear >= 0.1::double precision) AND (shear <= 0.2::double \
precision))
-> Bitmap Index Scan on structureparameter_shear_idx (cost=0.00..643.77 \
rows=26348 width=0)
    Index Cond: ((shear >= 0.1::double precision) AND (shear <= 0.2::double \
precision))
```

Then we have to get `biomoleculeId` via link between `Residue` and `Chain`. This query will be performed using default indexes on primary keys of `Residue` and `Chain`, nested loop and sorting:

```
EXPLAIN SELECT DISTINCT *
FROM chain JOIN residue ON chain.id=residue.chainId
WHERE 1=0 OR residue.id=...;
QUERY PLAN
-----
Unique (cost=17.03..17.06 rows=1 width=61)
-> Sort (cost=17.03..17.04 rows=1 width=61)
    Sort Key: chain.id, chain.biomoleculeid, chain.name, chain.sequence, \
chain.lenght, residue.name, residue."position"
-> Nested Loop (cost=0.00..17.02 rows=1 width=61)
    -> Index Scan using residue_pkey on residue \
(cost=0.00..8.74 rows=1 width=15)
        Index Cond: (id = 1)
    -> Index Scan using chain_pkey on chain \
(cost=0.00..8.27 rows=1 width=46)
        Index Cond: (chain.id = residue.chainid)
```

This gives us second result. Query answer will be intersection of results, even on database with almost 2 GB data will be answer computed in few seconds.

There are also more complex queries, for example when there are used parameters from joined tables, but that's also simple and quick query:

```
EXPLAIN ANALYZE SELECT StructureParameter.residueId \
FROM StructureParameter \
JOIN TorsionAngle ON StructureParameter.residueId=TorsionAngle.residueId \
JOIN SugarConformationalParameter \
ON TorsionAngle.residueId=SugarConformationalParameter.residueId \
WHERE shear >'0.27' AND shear <'3' AND alpha <' -60' AND tm <'49.8';
```

QUERY PLAN

```

-----
Hash Join (cost=10915.00..13861.59 rows=6407 width=4) \
(actual time=109.546..164.538 rows=16096 loops=1)
  Hash Cond: \
    (sugarconformationalparameter.residueid = structureparameter.residueid)
  -> Seq Scan on sugarconformationalparameter \
    (cost=0.00..2406.54 rows=126931 width=4)
    (actual time=0.009..23.530 rows=132378 loops=1)
    Filter: (tm < 49.8::real)
  -> Hash (cost=10731.55..10731.55 rows=14676 width=8) \
    (actual time=109.498..109.498 \
    rows=16346 loops=1)
    -> Hash Join (cost=5441.99..10731.55 rows=14676 width=8) \
    (actual time=48.309..105.549 rows=16346 loops=1)
      Hash Cond: (structureparameter.residueid = torsionangle.residueid)
      -> Bitmap Heap Scan on structureparameter (cost=1496.41..5427.61 \
      rows=61280 width=4) (actual time=8.040..21.094 rows=57851 loops=1)
        Recheck Cond: ((shear > 0.27::real) AND (shear < 3::real))
        -> Bitmap Index Scan on structureparameter_shear_idx \
        (cost=0.00..1481.09 rows=61280 width=0) \
        (actual time=7.594..7.594 rows=57851 loops=1)
          Index Cond: ((shear > 0.27::real) AND (shear < 3::real))
      -> Hash (cost=2803.20..2803.20 rows=69630 width=4) \
      (actual time=40.171..40.171 rows=70781 loops=1)
        -> Seq Scan on torsionangle \
        (cost=0.00..2803.20 rows=69630 width=4) \
        (actual time=0.010..23.226 rows=70781 loops=1)
        Filter: (alpha < (-60)::real)
Total runtime: 165.594 ms

```

8.3.4 Usability Testing

The web application is only in prototype version, the user interface could be more comfortable, but still the basic operation is not a problem for typical users. Hallway testing (rather than using an in-house, trained group of testers, just five to six random people, indicative of a cross-section of end users, are brought in to test the product, or service; the name of the technique refers to the fact that the testers should be random people who pass by in the hallway [48]) of query form shows that all testing subjects (Martina, Karel, Pavel, Otakar and Nikola) were able to perform it easily.

9 Conclusion

Target of this master thesis was to design, develop and build the database of nucleic acids data using PostgreSQL, database allowing to perform queries on structure of nucleic acids simply, fast and in a user friendly way.

The database scheme, importing tool and web application – all licensed under GNU/GPL – were implemented. Web application is available for all users at URI <http://nadb.cecak.cz/> and using query form it's possible to get all needed data.

The examination of initial design requirements showed that even complex information like nucleic acids parameters could be stored in flat relation table structure with ability for quick searching. More than 95 % of all known nucleic acids from Protein Databank were imported with full details, the rest was not correctly recognized with underlying external application used for importing.

As described and proved in Verification chapter, the target of the thesis was achieved, the database queries are simple to formulate, extend and perform. The database and application will help people at The Institute of Chemical Technology, Prague (ICT) in demonstration and research.

9.1 Future Development

One of the suggested extensions of this work could be database population tool – communication with 3rd party application 3DNA could be standardized via an exactly defined application interface or file format.

The prototype web application for performing queries could be extended from user interface point of view, it could be upgraded from prototype to fully tested and optimized tool.

Also the underlying 3rd party mmLib could be improved, in order to analyze all data without any unexpected states.

A Bibliography

- [1] *Nucleic Acid*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Nucleic_acid
(last fetched on February, 19th 2011)
- [2] *Central Dogma*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Central_Dogma
(last fetched on February, 19th 2011)
- [3] *Kapitoly z lékařské biologie 1*
Prof. MUDr. Petr Goetz, CSc., Univerzita Karlova v Praze, Prague, CZ
Jinočany H&H1994
ISBN: 80-85787-56-3
- [4] *The Nobel Prize in Physiology or Medicine 1962*
http://nobelprize.org/nobel_prizes/medicine/laureates/1962/
(last fetched on February, 19th 2011)
- [5] *The Nobel Prize in Physiology or Medicine 1968*
http://nobelprize.org/nobel_prizes/medicine/laureates/1968/
(last fetched on February, 19th 2011)
- [6] *PostgreSQL Homepage*
<http://www.postgresql.org/>
(last fetched on February, 19th 2011)
- [7] *PostgreSQL*
Wikipedia, the free encyclopedia
<http://en.wikipedia.org/wiki/Postgres>
(last fetched on February, 19th 2011)
- [8] *Ingres Frequently Asked Questions*
<http://www.bizyx.com/ingres/faq.htm>
(last fetched on February, 19th 2011)
- [9] *Chargaff's Rules*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Chargaff's_rules
(last fetched on February, 26th 2011)
- [10] *Principles of Nucleic Acid Structure*
Stephen Neidle, The School of Pharmacy, University of London, London, UK
Academic Press, Elsevier, 2008
ISBN: 978-0-12-369507-9
- [11] *Human Genome Project Information*
U.S. Department of Energy and the National Institutes of Health
http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml
(last fetched on February, 26th 2011)

- [12] *Repressor*
Wikipedia, the free encyclopedia
<http://en.wikipedia.org/wiki/Repressor>
(last fetched on February, 26th 2011)
- [13] *Protein-Nucleic Acid Interactions*
Phoebe A. Rice, Carl C. Correll
Royal Society of Chemistry, 2008
ISBN: 978-0-85404-272-2
- [14] *Definitions and nomenclature of nucleic acid structure components*
R.E. Dickerson
Molecular Biology Institute, University of California, Los Angeles, CA 90024, USA, 1989
<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC317523/>
- [15] *A Standard Reference Frame for the Description of Nucleic Acid Base-pair Geometry*
Wilma K. Olson and Xiang-Jun Lu
Rutgers University, Piscataway, USA
Journal of Molecular Biology (2001) 313: 229 - 237
<http://ndbserver.rutgers.edu/archives/report/tsukuba/tsukuba.pdf>
- [16] *Worldwide Protein Data Bank Official Website*
Worldwide Protein Data Bank
<http://www.wwpdb.org/>
(last fetched on February, 26th 2011)
- [17] *3DNA Home Page (Nucleic Acids Structures; Lu & Olson)*
Xiang-Jun Lu & Wilma K. Olson (2008). 3DNA: a versatile, integrated software system for the analysis, rebuilding and visualization of three-dimensional nucleic-acid structures
Nat Protoc. 3(7), 1213-27.
<http://rutchem.rutgers.edu/~xiangjun/3DNA/index.html/>
(last fetched on February, 26th 2011)
- [18] *Jmol: an open-source Java viewer for chemical structures in 3D*
<http://jmol.sourceforge.net/>
(last fetched on March, 12th 2011)
- [19] *Database normalization*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Database_normalization
(last fetched on March, 12th 2011)
- [20] *SQL*
Wikipedia, the free encyclopedia
<http://en.wikipedia.org/wiki/SQL>
(last fetched on March, 12th 2011)
- [21] *The RCSB Protein Data Bank: a redesigned query system and relational database based on the mmCIF schema*
Nucleic Acids Res. 2005 January 1; 33(Database Issue): D233–D237.
Published online 2004 December 17. doi: 10.1093/nar/gki057.
2005 Oxford University Press

- [22] *PDB-SQL: a storage engine for macromolecular data*
Edward E. Pryor, Jr., Wake Forest University, Winston-Salem, NC
Jacquelyn S. Fetrow, Wake Forest University, Winston-Salem, NC
ACM-SE 45 Proceedings of the 45th annual southeast regional conference
New York, NY, USA 2007
ISBN: 978-1-59593-629-5
- [23] *Template script for mirroring PDB SNAPSHOTS FTP archive using rsync*
Bojan Beran, Wolfgang Bluhm
`ftp://snapshots.rcsb.org/rsyncSnapshots.sh` (last fetched on March, 19th 2011)
- [24] *RCSB PDB – Latest Released Structures*
`http://www.rcsb.org/pdb/static.do?p=home/faq.html#download`
(last fetched on March, 19th 2011)
- [25] *PDB File Format v. 3.2*
`ftp://ftp.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v32_A4.pdf` (last
fetched on March, 19th 2011)
- [26] *BioJava: an Open-Source Framework for Bioinformatics*
R.C.G. Holland; T. Down; M. Pocock; A. Prlić; D. Huen; K. James; S. Foisy; A. Dräger;
A. Yates; M. Heuer; M.J. Schreiber
Bioinformatics 2008; doi: 10.1093/bioinformatics/btn397
`http://biojava.org/wiki/Main_Page`
(last fetched on March, 19th 2011)
- [27] *mmCIF Resources*
`http://mmcif.rcsb.org/mmcif-early/background/index.html`
`http://mmcif.rcsb.org/`
(last fetched on March, 19th 2011)
- [28] *BioPython*
`http://biopython.org/wiki/Main_Page`
(last fetched on March, 19th 2011)
- [29] *mmLib: Python Macromolecular Library*
Jay Painter
`http://pymmlib.sourceforge.net/`
(last fetched on March, 19th 2011)
- [30] *mmLib Python toolkit for manipulating annotated structural models of biological macro-
molecules*
Jay Painter and Ethan A Merritt
Journal of Applied Crystallography
ISSN 0021-8898
- [31] *mmcif_pdbx.dic*
`http://mmcif.rcsb.org/dictionaries/mmcif_pdbx.dic/Index/index.html`
(last fetched on March, 27th 2011)
- [32] *Methods in Enzymology: The Macromolecular Crystallographic Information File (mmCIF)*
Philip E. Bourne, Helen M. Berman, Brian McMahon, Keith D. Watenpaugh, John West-
brook and Paula M.D. Fitzgerald
`http://mmcif.rcsb.org/pubs/methenz.html`
(last fetched on April, 2nd 2011)

- [33] *Comma-separated values*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Comma-separated_values
(last fetched on April, 9th 2011)
- [34] *Website wireframe*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Website_wireframe
(last fetched on April, 9th 2011)
- [35] *Balsamiq Mockups*
<http://balsamiq.com/products/mockups>
(last fetched on April, 10th 2011)
- [36] *Napkee – make your mockups come alive*
<http://www.napkee.com>
(last fetched on April, 10th 2011)
- [37] *Unix philosophy*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Unix_philosophy
(last fetched on April, 10th 2011)
- [38] *Basics of the Unix Philosophy*
The Art of Unix Programming
Eric Steven Raymond
<http://www.faqs.org/docs/artu/ch01s06.html>
(last fetched on April, 10th 2011)
- [39] *PostgreSQL 8.4: Data Types*
PostgreSQL: Documentation
<http://www.postgresql.org/docs/8.4/static/datatype.html>
(last fetched on April, 16th 2011)
- [40] *B-tree*
Wikipedia, the free encyclopedia
<http://en.wikipedia.org/wiki/B-tree>
(last fetched on April, 16th 2011)
- [41] *PostgreSQL 8.4: CREATE INDEX*
PostgreSQL: Documentation
<http://www.postgresql.org/docs/8.4/static/sql-createindex.html>
(last fetched on April, 16th 2011)
- [42] *PostgreSQL 8.4: C-Language Functions*
PostgreSQL: Documentation
<http://www.postgresql.org/docs/8.3/static/xfunc-c.html>
(last fetched on April, 16th 2011)
- [43] *PostGIS: Homepage*
<http://postgis.refractory.net/>
(last fetched on April, 16th 2011)

- [44] *Psycopg – PostgreSQL database adapter for Python*
Federico Di Gregorio
<http://initd.org/psycopg/docs/>
(last fetched on April, 17th 2011)
- [45] *PHP: Hypertext Preprocessor*
The PHP Group
<http://www.php.net>
(last fetched on April, 17th 2011)
- [46] *Jmol.js JavaScript Library*
Jmol: an open-source Java viewer for chemical structures in 3D.
<http://www.jmol.org/>
<http://jmol.sourceforge.net/jslibrary/>
(last fetched on April, 17th 2011)
- [47] *An Introduction to Jmol Scripting*
Nathan Silva and David Marcey
©2007
<http://www.callutheran.edu/BioDev/omm/scripting/molmast.htm>
(last fetched on April, 17th 2011)
- [48] *Usability testing*
Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Usability_testing#Hallway_testing
(last fetched on April, 23th 2011)

B Selected Content of Covered DVD

Most important content/directories of attached DVD:

- `import/`, the data importing tool,
- `db/`, database definition in SQL and the full database dump from PostgreSQL,
- `webapp/`, web application sources including Jmol Java applet,
- `verification/`, software for verification of data import, including verification logs and the result database itself,
- `software/`, 3rd party software needed for the importing tool, 3DNA and mmLib,
- `thesis/`, \LaTeX sources of this thesis with complete PDF version,
- `thesis/images/`, source of bitmap images used in this thesis,
- `thesis/metapost/`, source of MetaPost schemes used in this thesis.

C Database Scheme and Structure

C.1 Entity-Relation Diagram

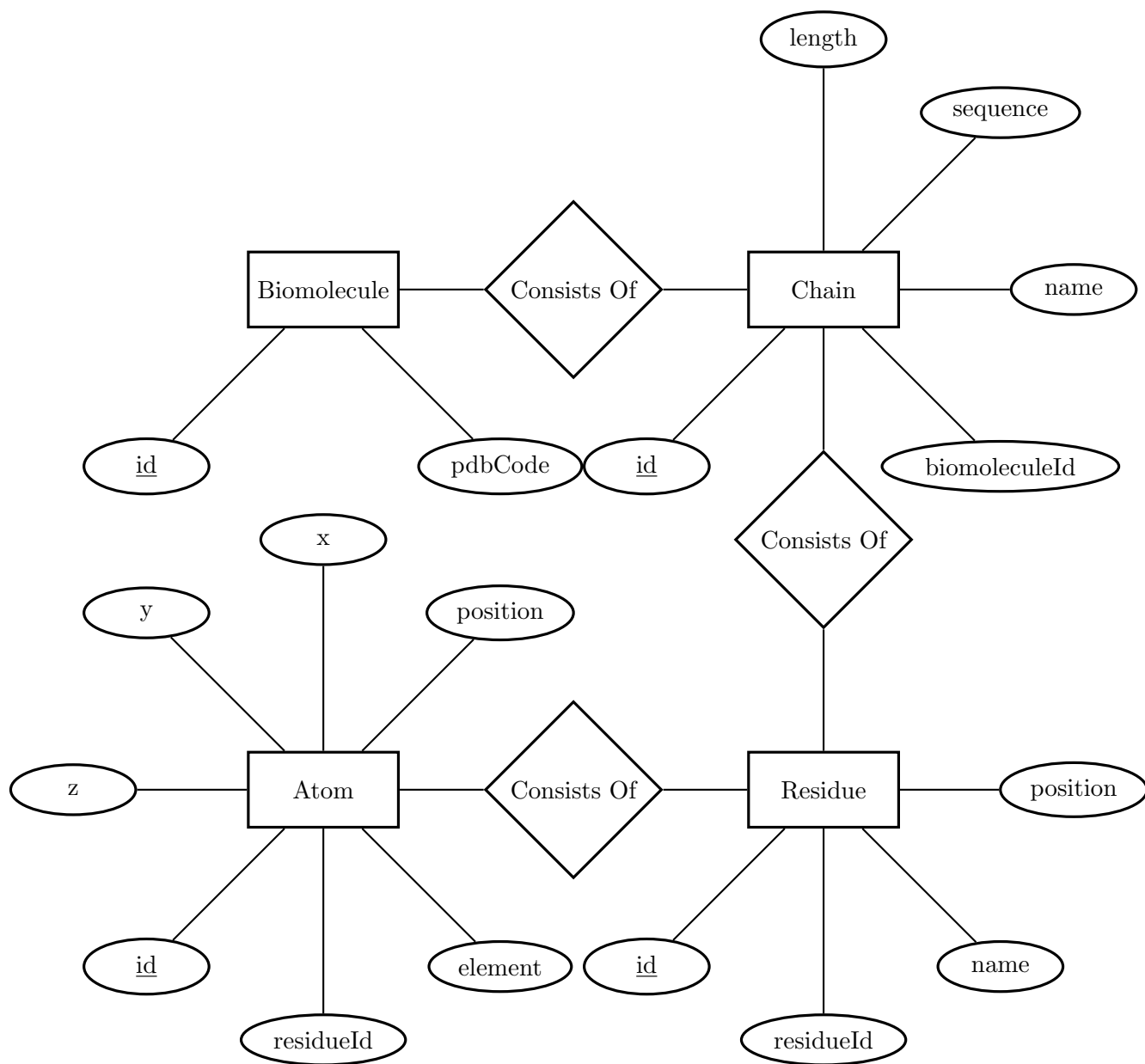


Figure C.1: Entity-Relation diagram: Base part

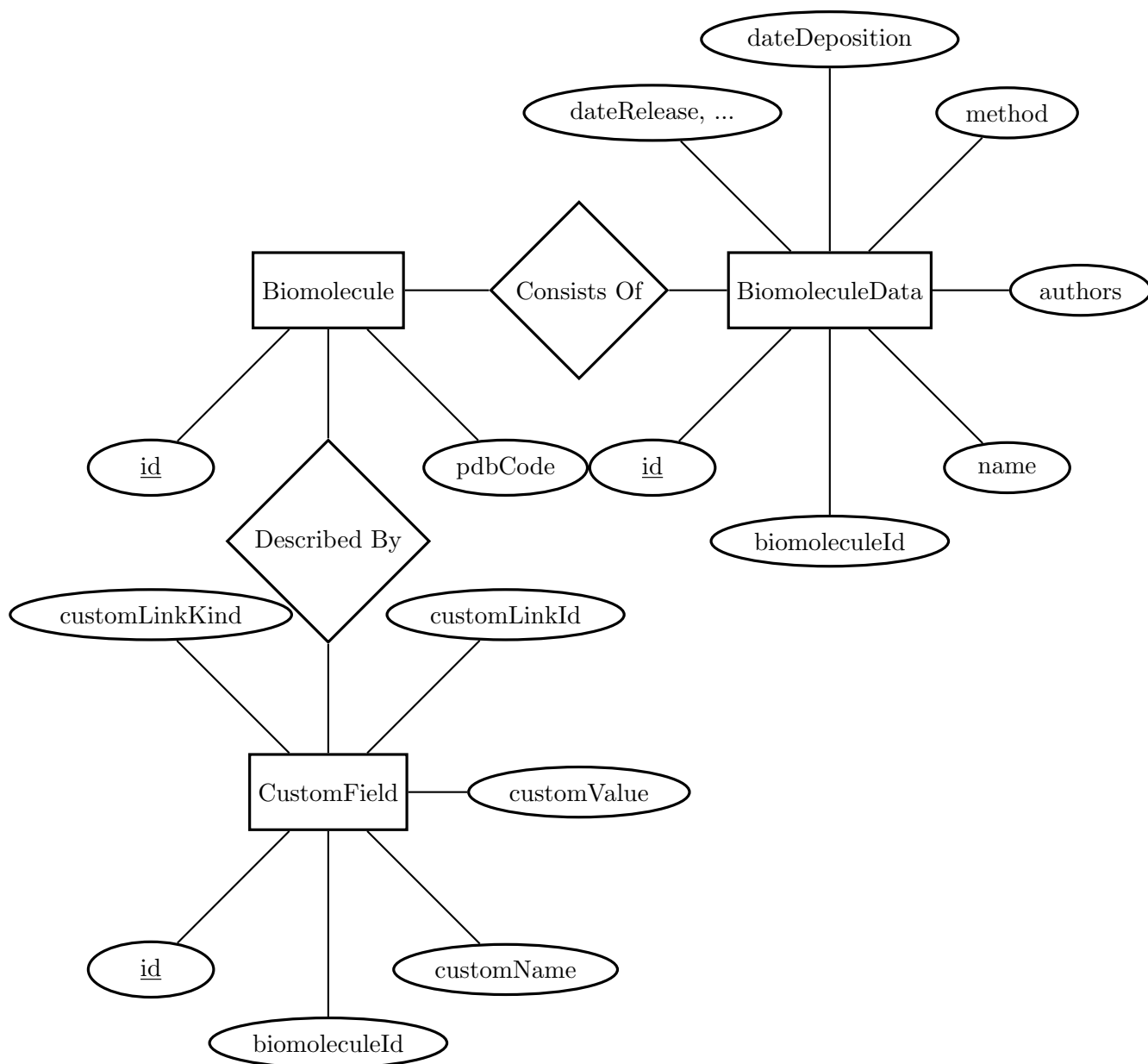


Figure C.2: Entity-Relation diagram: Biomolecule details

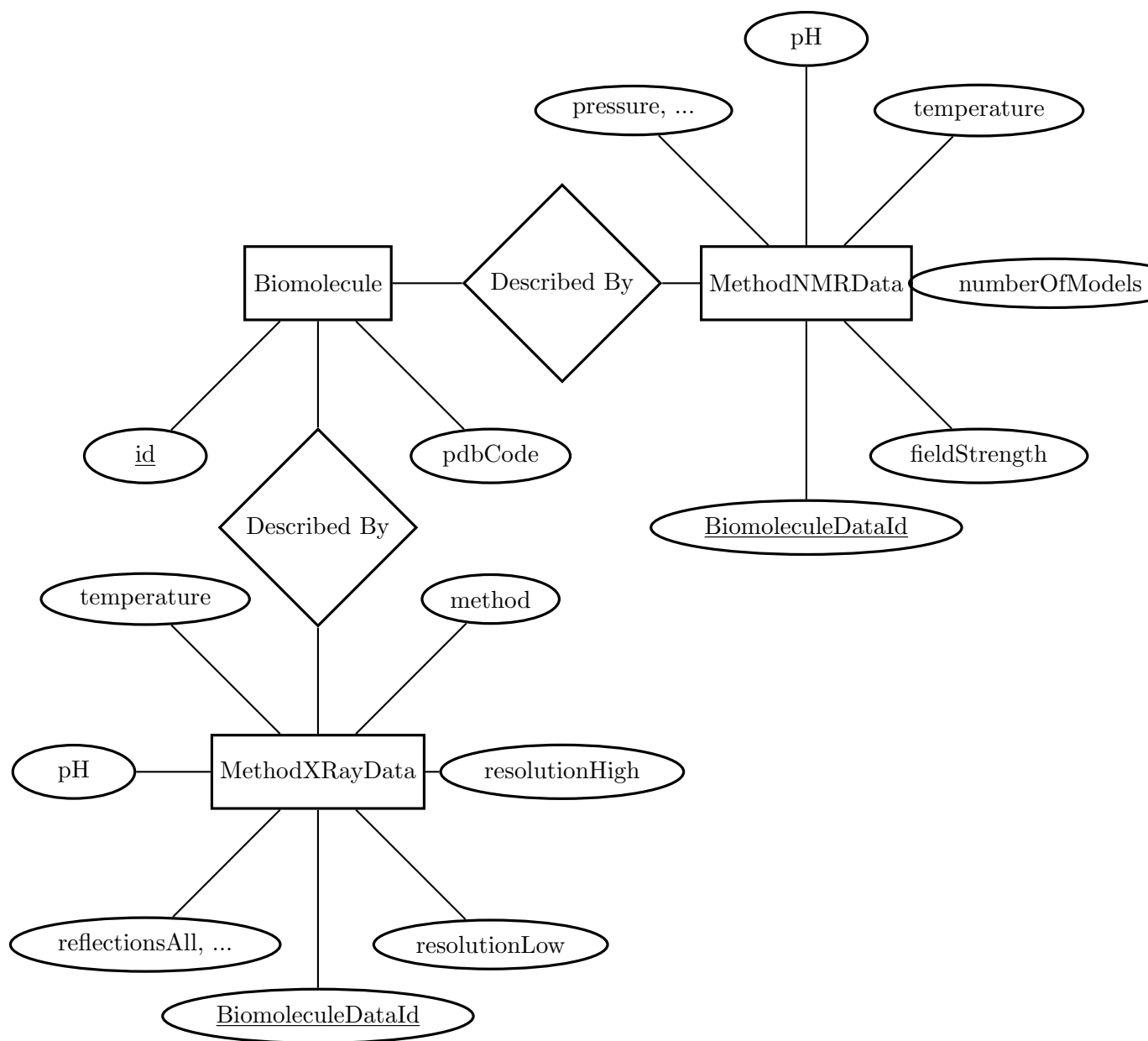


Figure C.3: Entity-Relation diagram: Biomolecule data methods

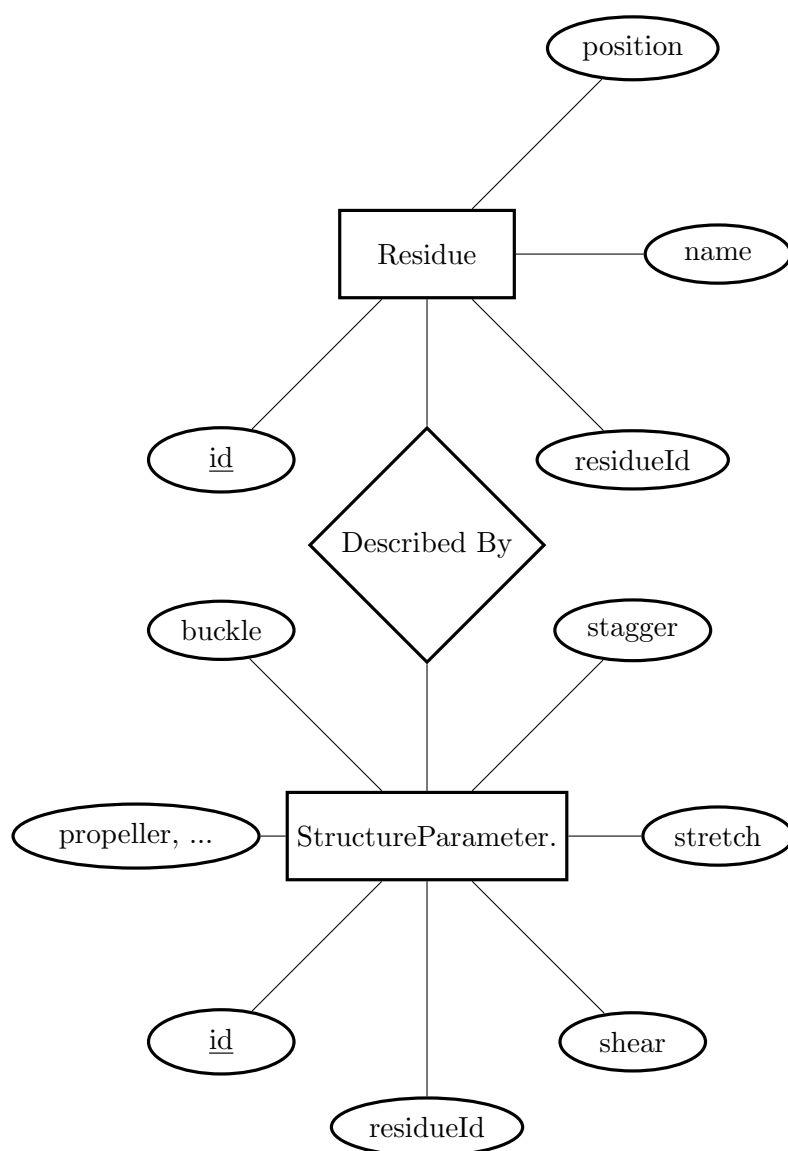


Figure C.4: Entity-Relation diagram: Structure parameters (same as derived tables).

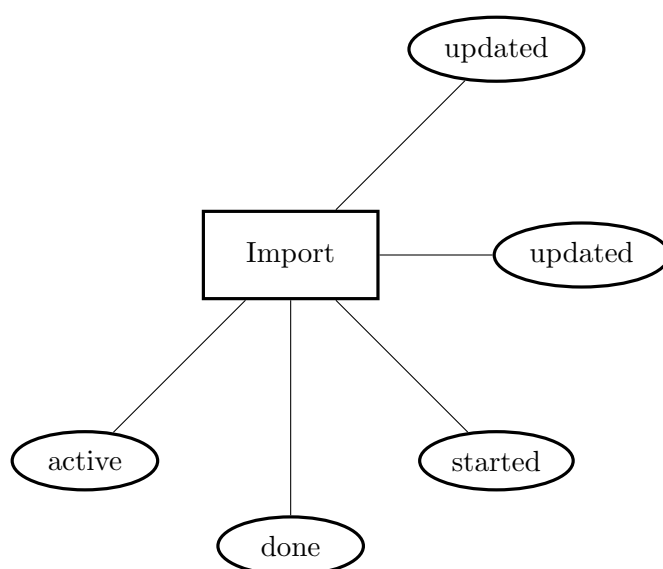


Figure C.5: Entity-Relation diagram: Import table.

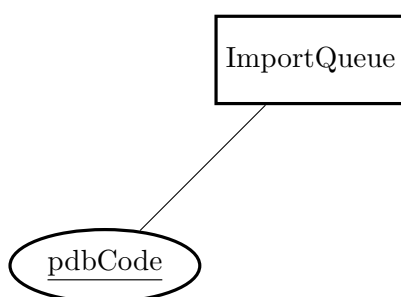


Figure C.6: Entity-Relation diagram: ImportQueue table.

C.2 Structure in Data Definition Language

```
-- Entity Biomolecule
CREATE TABLE Biomolecule (
  id SERIAL PRIMARY KEY,
  pdbCode VARCHAR(4) UNIQUE
);

-- Entity BiomoleculeData
CREATE TABLE BiomoleculeData (
  id SERIAL PRIMARY KEY,
  biomoleculeId INTEGER UNIQUE,
  name TEXT,
  authors TEXT,
  method TEXT,
  dateDeposition DATE,
  dateRelease DATE,
  sequence TEXT
);

CREATE INDEX biomoleculeData_dateDeposition_idx ON BiomoleculeData(dateDeposition);
CREATE INDEX biomoleculeData_dateRelease_idx ON BiomoleculeData(dateRelease);

-- Entity MethodXRayData
CREATE TABLE MethodXRayData (
  biomoleculeDataId INTEGER PRIMARY KEY,
  resolutionLow REAL,
  resolutionHigh REAL,
  method TEXT,
  temperature REAL,
  pH REAL,
  reflectionsAll REAL,
  reflectionsObs REAL
);

CREATE INDEX MethodXRayData_resolutionLow_idx ON MethodXRayData(resolutionLow);
CREATE INDEX MethodXRayData_resolutionHigh_idx ON MethodXRayData(resolutionHigh);
CREATE INDEX MethodXRayData_method_idx ON MethodXRayData(method);
CREATE INDEX MethodXRayData_temperature_idx ON MethodXRayData(temperature);
CREATE INDEX MethodXRayData_pH_idx ON MethodXRayData(pH);
CREATE INDEX MethodXRayData_reflectionsAll_idx ON MethodXRayData(reflectionsAll);
CREATE INDEX MethodXRayData_reflectionsObs_idx ON MethodXRayData(reflectionsObs);

-- Entity MethodNMRData
CREATE TABLE MethodNMRData (
  biomoleculeDataId INTEGER PRIMARY KEY,
  fieldStrength REAL,
  numberOfModels INTEGER,
  temperature REAL,
  pressure TEXT,
```

```
pH REAL,
ionicStrenght TEXT
);

CREATE INDEX MethodNMRData_fieldStrength_idx ON MethodNMRData(fieldStrength);
CREATE INDEX MethodNMRData_numberOfModels_idx ON MethodNMRData(numberOfModels);
CREATE INDEX MethodNMRData_temperature_idx ON MethodNMRData(temperature);
CREATE INDEX MethodNMRData_pressure_idx ON MethodNMRData(pressure);
CREATE INDEX MethodNMRData_pH_idx ON MethodNMRData(pH);
CREATE INDEX MethodNMRData_ionicStrenght_idx ON MethodNMRData(ionicStrenght);

-- Entity Chain
CREATE TABLE Chain (
    id SERIAL PRIMARY KEY,
    biomoleculeId INTEGER,
    name TEXT,
    sequence TEXT,
    length INTEGER
);

CREATE INDEX Chain_biomoleculeId_idx ON Chain(biomoleculeId);

-- Entity Residue
CREATE TABLE Residue (
    id SERIAL PRIMARY KEY,
    chainId INTEGER,
    name TEXT,
    position INTEGER
);

CREATE INDEX Residue_chainId_idx ON Residue(chainId);
CREATE INDEX Residue_position_idx ON Residue(position);

-- Entity Atom
CREATE TABLE Atom (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    element TEXT,
    position INTEGER,
    x REAL,
    y REAL,
    z REAL
);

CREATE INDEX Atom_residueId_idx ON Atom(residueId);

-- Entity CustomField
CREATE TABLE CustomField (
    id SERIAL PRIMARY KEY,
    biomoleculeId INTEGER,
```

```

    customName TEXT,
    customValue TEXT,
    customLinkId TEXT,
    customLinkKind TEXT
);

-- Entity StructureParameter
CREATE TABLE StructureParameter (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    shear REAL,
    stretch REAL,
    stagger REAL,
    buckle REAL,
    propeller REAL,
    opening REAL,
    shift REAL,
    slide REAL,
    rise REAL,
    tilt REAL,
    roll REAL,
    twist REAL
);

CREATE INDEX StructureParameter_residueId_idx ON StructureParameter(residueId);
CREATE INDEX StructureParameter_shear_idx ON StructureParameter(shear);
CREATE INDEX StructureParameter_stretch_idx ON StructureParameter(stretch);
CREATE INDEX StructureParameter_stagger_idx ON StructureParameter(stagger);
CREATE INDEX StructureParameter_buckle_idx ON StructureParameter(buckle);
CREATE INDEX StructureParameter_propeller_idx ON StructureParameter(propeller);
CREATE INDEX StructureParameter_opening_idx ON StructureParameter(opening);
CREATE INDEX StructureParameter_shift_idx ON StructureParameter(shift);
CREATE INDEX StructureParameter_slide_idx ON StructureParameter(slide);
CREATE INDEX StructureParameter_rise_idx ON StructureParameter(rise);
CREATE INDEX StructureParameter_tilt_idx ON StructureParameter(tilt);
CREATE INDEX StructureParameter_roll_idx ON StructureParameter(roll);
CREATE INDEX StructureParameter_twist_idx ON StructureParameter(twist);

-- Entity OverlapArea
CREATE TABLE OverlapArea (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    i1_i2 REAL,
    i1_i2Ring REAL,
    i1_j2 REAL,
    i1_j2Ring REAL,
    j1_i2 REAL,
    j1_i2Ring REAL,
    j1_j2 REAL,
    j1_j2Ring REAL);

```



```

CREATE INDEX OverlapArea_residueId_idx ON OverlapArea(residueId);
CREATE INDEX OverlapArea_i1_i2_idx ON OverlapArea(i1_i2);
CREATE INDEX OverlapArea_i1_i2Ring_idx ON OverlapArea(i1_i2Ring);
CREATE INDEX OverlapArea_i1_j2_idx ON OverlapArea(i1_j2);
CREATE INDEX OverlapArea_i1_j2Ring_idx ON OverlapArea(i1_j2Ring);
CREATE INDEX OverlapArea_j1_i2_idx ON OverlapArea(j1_i2);
CREATE INDEX OverlapArea_j1_i2Ring_idx ON OverlapArea(j1_i2Ring);
CREATE INDEX OverlapArea_j1_j2_idx ON OverlapArea(j1_j2);
CREATE INDEX OverlapArea_j1_j2Ring_idx ON OverlapArea(j1_j2Ring);

-- Entity OriginMeanNormalVector
CREATE TABLE OriginMeanNormalVector (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    Ox REAL,
    Oy REAL,
    Oz REAL,
    Nx REAL,
    Ny REAL,
    Nz REAL
);

CREATE INDEX OriginMeanNormalVector_residueId_idx ON OriginMeanNormalVector(residueId);
CREATE INDEX OriginMeanNormalVector_Ox_idx ON OriginMeanNormalVector(Ox);
CREATE INDEX OriginMeanNormalVector_Oy_idx ON OriginMeanNormalVector(Oy);
CREATE INDEX OriginMeanNormalVector_Oz_idx ON OriginMeanNormalVector(Oz);
CREATE INDEX OriginMeanNormalVector_Nx_idx ON OriginMeanNormalVector(Nx);
CREATE INDEX OriginMeanNormalVector_Ny_idx ON OriginMeanNormalVector(Ny);
CREATE INDEX OriginMeanNormalVector_Nz_idx ON OriginMeanNormalVector(Nz);

-- Entity HelicalParameter
CREATE TABLE HelicalParameter (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    X_disp REAL,
    Y_disp REAL,
    h_Rise REAL,
    Incl REAL,
    Tip REAL,
    h_Twist REAL
);

CREATE INDEX HelicalParameter_residueId_idx ON HelicalParameter(residueId);
CREATE INDEX HelicalParameter_X_disp_idx ON HelicalParameter(X_disp);
CREATE INDEX HelicalParameter_Y_disp_idx ON HelicalParameter(Y_disp);
CREATE INDEX HelicalParameter_h_Rise_idx ON HelicalParameter(h_Rise);
CREATE INDEX HelicalParameter_Incl_idx ON HelicalParameter(Incl);
CREATE INDEX HelicalParameter_Tip_idx ON HelicalParameter(Tip);
CREATE INDEX HelicalParameter_h_Twist_idx ON HelicalParameter(h_Twist);

```

```
-- Entity StepClassification
CREATE TABLE StepClassification (
  id SERIAL PRIMARY KEY,
  residueId INTEGER,
  Xp REAL,
  Yp REAL,
  Zp REAL,
  XpH REAL,
  YpH REAL,
  ZpH REAL
);

CREATE INDEX StepClassification_residueId_idx ON StepClassification(residueId);
CREATE INDEX StepClassification_Xp_idx ON StepClassification(Xp);
CREATE INDEX StepClassification_Yp_idx ON StepClassification(Yp);
CREATE INDEX StepClassification_Zp_idx ON StepClassification(Zp);
CREATE INDEX StepClassification_XpH_idx ON StepClassification(XpH);
CREATE INDEX StepClassification_YpH_idx ON StepClassification(YpH);
CREATE INDEX StepClassification_ZpH_idx ON StepClassification(ZpH);

-- Entity C1GlobalParameter
CREATE TABLE C1GlobalParameter (
  id SERIAL PRIMARY KEY,
  residueId INTEGER,
  disp REAL,
  angle REAL,
  twist REAL,
  rise REAL
);

CREATE INDEX C1GlobalParameter_residueId_idx ON C1GlobalParameter(residueId);
CREATE INDEX C1GlobalParameter_disp_idx ON C1GlobalParameter(disp);
CREATE INDEX C1GlobalParameter_angle_idx ON C1GlobalParameter(angle);
CREATE INDEX C1GlobalParameter_twist_idx ON C1GlobalParameter(twist);
CREATE INDEX C1GlobalParameter_rise_idx ON C1GlobalParameter(rise);

-- Entity TorsionAngle
CREATE TABLE TorsionAngle (
  id SERIAL PRIMARY KEY,
  residueId INTEGER,
  alpha REAL,
  beta REAL,
  gamma REAL,
  delta REAL,
  epsilon REAL,
  zeta REAL,
  chi REAL
);
```

```

CREATE INDEX TorsionAngle_residueId_idx ON TorsionAngle(residueId);
CREATE INDEX TorsionAngle_alpha_idx ON TorsionAngle(alpha);
CREATE INDEX TorsionAngle_beta_idx ON TorsionAngle(beta);
CREATE INDEX TorsionAngle_gamma_idx ON TorsionAngle(gamma);
CREATE INDEX TorsionAngle_delta_idx ON TorsionAngle(delta);
CREATE INDEX TorsionAngle_epsilon_idx ON TorsionAngle(epsilon);
CREATE INDEX TorsionAngle_zeta_idx ON TorsionAngle(zeta);
CREATE INDEX TorsionAngle_chi_idx ON TorsionAngle(chi);

-- Entity SugarConformationalParameter
CREATE TABLE SugarConformationalParameter (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    tm REAL,
    P REAL
);

CREATE INDEX SugarConformationalParameter_residueId_idx \
    ON SugarConformationalParameter(residueId);
CREATE INDEX SugarConformationalParameter_tm_idx \
    ON SugarConformationalParameter(tm);
CREATE INDEX SugarConformationalParameter_P_idx \
    ON SugarConformationalParameter(P);

-- Entity PCVirtualBondDistance
CREATE TABLE PCVirtualBondDistance (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    PP REAL,
    C1C1 REAL
);

CREATE INDEX PCVirtualBondDistance_residueId_idx ON PCVirtualBondDistance(residueId);
CREATE INDEX PCVirtualBondDistance_PP_idx ON PCVirtualBondDistance(PP);
CREATE INDEX PCVirtualBondDistance_C1C1_idx ON PCVirtualBondDistance(C1C1);

-- Entity HelixRadiusRadialDisplacement
CREATE TABLE HelixRadiusRadialDisplacement (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    P REAL,
    O4 REAL,
    C1 REAL
);

CREATE INDEX HelixRadiusRadialDisplacement_residueId_idx \
    ON HelixRadiusRadialDisplacement(residueId);
CREATE INDEX HelixRadiusRadialDisplacement_P_idx \
    ON HelixRadiusRadialDisplacement(P);
CREATE INDEX HelixRadiusRadialDisplacement_O4_idx \

```

```

        ON HelixRadiusRadialDisplacement(04);
CREATE INDEX HelixRadiusRadialDisplacement_C1_idx \
        ON HelixRadiusRadialDisplacement(C1);

-- Entity PositionAndLocalHelicalAxisVector
CREATE TABLE PositionAndLocalHelicalAxisVector (
    id SERIAL PRIMARY KEY,
    residueId INTEGER,
    Px REAL,
    Py REAL,
    Pz REAL,
    Hx REAL,
    Hy REAL,
    Hz REAL
);

CREATE INDEX PositionAndLocalHelicalAxisVector_residueId_idx \
    ON PositionAndLocalHelicalAxisVector(residueId);
CREATE INDEX PositionAndLocalHelicalAxisVector_Px_idx \
    ON PositionAndLocalHelicalAxisVector(Px);
CREATE INDEX PositionAndLocalHelicalAxisVector_Py_idx \
    ON PositionAndLocalHelicalAxisVector(Py);
CREATE INDEX PositionAndLocalHelicalAxisVector_Pz_idx \
    ON PositionAndLocalHelicalAxisVector(Pz);
CREATE INDEX PositionAndLocalHelicalAxisVector_Hx_idx \
    ON PositionAndLocalHelicalAxisVector(Hx);
CREATE INDEX PositionAndLocalHelicalAxisVector_Hy_idx \
    ON PositionAndLocalHelicalAxisVector(Hy);
CREATE INDEX PositionAndLocalHelicalAxisVector_Hz_idx \
    ON PositionAndLocalHelicalAxisVector(Hz);

-- Entity ImportQueue
CREATE TABLE ImportQueue (
    pdbCode VARCHAR(4) PRIMARY KEY
);

-- Entity Import
CREATE TABLE Import (
    active SMALLINT,
    done INTEGER,
    started TIMESTAMP,
    updated TIMESTAMP,
    finished TIMESTAMP
);

```

D Installation Manual

This installation manual will guide you with configuration of the whole system. It's simple, so it should go smooth. The manual is written for nadb package version 1.0, but it will be usable very likely for new versions as well.

D.1 Quick Start

1. Create PostgreSQL user, DB and define structure from `db/init-db.sql`.
2. Install web application from `webapp/` directory at web server with PHP support.
3. Install mmLib at system with Python and copy the `import/` directory.
4. Install 3DNA.
5. Install cron worker from `cron/` directory.
6. Verify installation by importing structures by cron and check the display.

D.2 Detail Installation Instructions

Create PostgreSQL user, DB and define structure from `db/init-db.sql`.

Typically login as PostgreSQL administrative user and run `createuser` and `createdb` commands.

```
su - postgres
createuser -P nadb
createdb -O nadb nadb
cat db/init-db.sql | psql -d nadb
```

If necessary, update `pg_hba.conf` to allow proper user authentication (e.g. by password or ident).

Install web application from `webapp/` directory at web server with PHP support.

PHP and module for PostgreSQL is required. Configuration is in file `config/config.php`. Update there mainly connection string to PostgreSQL nadb database.

Install mmLib at system with Python and copy the `import/` directory.

```
python setup.py checkdeps
python setup.py build
python setup.py install
```

Update the configuration in the beginning of `import.py` script – connection string and the target directory for 3DNA.

Install 3DNA.

Extract the content of gzipped tar, e.g. under `/opt/` directory.

Install cron.

Copy the files and configure for them automatic start.

Verify installation by importing structures by local `import.py` cron and check the data in web application.