# BioSeD - **Bio**logical **Se**quences **D**atabase

Flávio Manuel Fernandes Cruz
Instituto de Biologia Molecular, R. do Campo Alegre, 823
4150-180 Porto.
**flaviocruz.net - ei05011@fe.up.pt**

November 23, 2010

# Contents

**Contents** 1

**1 Introduction** 6

**2 Related Work** 8
2.1 Genotator . . . 8
2.2 BioDAS . . . 8
2.3 EnsMart . . . 9
2.4 CBS Genome Atlas Database . . . 10
2.5 Extensible open source content management systems and frameworks . . . 10
2.6 GenDB . . . 10

**3 System objects** 12
3.1 Sequences . . . 12
3.2 Labels . . . 12
3.3 Taxonomies . . . 14

**4 Design** 15
4.1 Architecture . . . 15
4.2 Relational Model . . . 16
4.2.1 User tables . . . 16
4.2.2 Taxonomy hierarchy . . . 16
4.2.3 Sequence, labels and label instances . . . 17
4.2.4 Miscellaneous tables . . . 18
4.3 File formats . . . 19
4.3.1 FASTA . . . 19
4.3.2 XML . . . 20
4.4 Query language . . . 24

**5 Implementation** 26
5.1 Technology . . . 26
5.2 Authentication . . . 28
5.3 Recording modifications . . . 29
5.4 Tables and views . . . 29
5.5 Database optimizations . . . 30
5.6 Label generation . . . 33
5.7 Search . . . 34

5.7.1 Query trees . . . 34
5.7.2 The transform label . . . 36
5.7.3 Parsing written queries . . . 36
5.8 Linking DNA and proteins . . . 37
5.9 Exporting data . . . 37
5.10 Importing data . . . 38
5.11 Search operations . . . 38
5.11.1 Histogram generation . . . 39
5.11.2 Subsequence generation . . . 40
5.12 Storing system wide information . . . 41
5.13 Importing the NCBI database . . . 41
5.14 Resetting the database . . . 41

**6 Default information** **45**
6.1 Labels . . . 45
6.2 Ranks . . . 47
6.3 Taxonomy name types . . . 48

**7 User Interface** **49**
7.1 Grid component . . . 50
7.2 Query interface . . . 51
7.3 Written query interface . . . 53
7.4 Sequence . . . 54
7.5 Changing sequence labels . . . 55
7.6 Batch uploading . . . 56
7.7 Taxonomy tree browsing . . . 57
7.8 Histograms . . . 57
7.9 Loading screens . . . 57

**8 Results** **61**
8.1 Sequence import . . . 62
8.2 Multiple sequence annotation . . . 62
8.3 Search . . . 63

**9 Conclusion** **65**

**A User manual** **68**
A.1 Installation . . . 68
A.2 Labels . . . 69
A.2.1 Creating new labels . . . 71
A.2.2 Import / Export . . . 72
A.3 Sequences . . . 72
A.3.1 Labels page . . . 73
A.3.2 Batch . . . 75
A.4 Taxonomies . . . 76
A.4.1 Managing trees . . . 76
A.4.2 Managing ranks . . . 79

A.4.3 Managing taxonomies . . . . . . . . . . 80
A.5 Search . . . . . . . . . . 81
A.5.1 Query input . . . . . . . . . . 82
A.5.2 Operations . . . . . . . . . . 84
A.5.3 Sub-sequences . . . . . . . . . . 85
A.5.4 Histograms . . . . . . . . . . 85
A.5.5 Export . . . . . . . . . . 85
A.5.6 Batch labels . . . . . . . . . . 85
A.5.7 Delete . . . . . . . . . . 86
A.5.8 Preview . . . . . . . . . . 86
A.5.9 Written queries . . . . . . . . . . 88
A.6 File formats . . . . . . . . . . 89
A.7 Administration . . . . . . . . . . 89
A.7.1 User management . . . . . . . . . . 89
A.7.2 Import / Export database . . . . . . . . . . 90
A.7.3 Application customization . . . . . . . . . . 91

**B Tables** **93**

# List of Figures


2.1 Genotator map view. . . . . . . . . . 9
2.2 The CBS Genome Atlas. . . . . . . . . . 11

3.1 Representation of a sequence object. . . . . . . . . . 12
3.2 A simple taxonomy tree. . . . . . . . . . 14

4.1 The system top level architecture. . . . . . . . . . 15
4.2 User tables. . . . . . . . . . 16
4.3 Taxonomies in the relational model . . . . . . . . . . 17
4.4 Label, sequences and label instances. . . . . . . . . . 17
4.5 Query language written in BNF. . . . . . . . . . 24

7.1 System home page. . . . . . . . . . 49
7.2 Grid component aspect. . . . . . . . . . 51
7.3 Search query input. . . . . . . . . . 52
7.4 Search operations. . . . . . . . . . 52
7.5 Search preview results. . . . . . . . . . 53
7.6 Search box. . . . . . . . . . 53
7.7 Wide search page. . . . . . . . . . 54
7.8 Add new sequence page. . . . . . . . . . 55
7.9 Example labels page. . . . . . . . . . 56
7.10 Available labels grid. . . . . . . . . . 57
7.11 Annotating a new object label. . . . . . . . . . 58
7.12 Upload sequences page. . . . . . . . . . 58
7.13 Upload report after sending a pair of DNA and protein files. . . . . . . . . . 59
7.14 Navigating through the NCBI tree. . . . . . . . . . 59
7.15 Plotting histograms. . . . . . . . . . 60
7.16 Example loading screen. . . . . . . . . . 60

A.1 System home page. . . . . . . . . . 69
A.2 Listing all labels . . . . . . . . . . 70
A.3 Import labels report . . . . . . . . . . 72
A.4 Sequence page. . . . . . . . . . 73
A.5 Example labels page. . . . . . . . . . 74
A.6 Annotating a new object label. . . . . . . . . . 75
A.7 Upload sequences page. . . . . . . . . . 76
A.8 Batch sequence report. . . . . . . . . . 77
A.9 Tree listing. . . . . . . . . . 77

A.10 Viewing a tree. . . . . 78
A.11 Browsing the NCBI tree. . . . . 79
A.12 Import rank report . . . . . 80
A.13 Editing taxonomy information. . . . . 81
A.14 Other names section. . . . . 81
A.15 Search query input. . . . . 82
A.16 Search operations. . . . . 85
A.17 Plotting histograms. . . . . 86
A.18 Export search page. . . . . 87
A.19 Add label to multiple sequences. . . . . 87
A.20 Search preview results. . . . . 87
A.21 Using view label. . . . . 88
A.22 Page with user information. . . . . 90

# List of Tables


3.1 Label properties. . . . . 13
3.2 Label types. . . . . 13

4.1 Label types and related fields. . . . . 18

5.1 Label generation environment. . . . . 33
5.2 Operators and values in query objects. . . . . 43

8.1 Sequence importing results. . . . . 62
8.2 Multiple sequence annotation. . . . . 63
8.3 Searching 404 sequences, no result translation. . . . . 63
8.4 Searching 404 sequences with **translated** translation. . . . . 64
8.5 Searching 778 sequences with no translation. . . . . 64
8.6 Searching 778 sequences with **translated** translation. . . . . 64

A.1 Label types and operators. . . . . 84
A.2 Operators and values in query expressions. . . . . 92

B.1 User table. . . . . 93
B.2 Configuration table. . . . . 93
B.3 History table. . . . . 94
B.4 TaxonomyNameType table. . . . . 94
B.5 TaxonomyName table. . . . . 94
B.6 TaxonomyRank table. . . . . 95
B.7 TaxonomyTree table. . . . . 95
B.8 Taxonomy table. . . . . 95
B.9 Label table. . . . . 96
B.10 Sequence table. . . . . 97
B.11 LabelSequence table. . . . . 97
B.12 Event table. . . . . 98
B.13 File table. . . . . 98

# Listings

4.1 Plain FASTA format. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
4.2 Complex FASTA format example. . . . . . . . . . . . . . . . . . . . . . . . . 20
4.3 An example Label XML file. . . . . . . . . . . . . . . . . . . . . . . . . . . 21
4.4 An Sequence XML file. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
4.5 An example Rank XML file. . . . . . . . . . . . . . . . . . . . . . . . . . . 22
4.6 An example Taxonomy tree XML file. . . . . . . . . . . . . . . . . . . . . . 22
4.7 Database in XML skeleton. . . . . . . . . . . . . . . . . . . . . . . . . . . 23
5.1 Terminal expression as a JSON object . . . . . . . . . . . . . . . . . . . . . 35
5.2 Parametrized terminal expression . . . . . . . . . . . . . . . . . . . . . . . 35
5.3 Structured expression as a JSON object . . . . . . . . . . . . . . . . . . . . 36
5.4 Resulting SQL query. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
5.5 Transform label SQL code template. . . . . . . . . . . . . . . . . . . . . . . 37
5.6 Making distribution table for the name label. . . . . . . . . . . . . . . . . . 39
5.7 Making distribution table for numeric labels. . . . . . . . . . . . . . . . . . 39
5.8 Making distribution table for non-numeric labels. . . . . . . . . . . . . . . . 40
5.9 Garbage collecting sequences. . . . . . . . . . . . . . . . . . . . . . . . . . 41
5.10 NCBI import algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
7.1 Grid component usage. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
8.1 MySQL database size query. . . . . . . . . . . . . . . . . . . . . . . . . . . 62

# List of Algorithms

1 SQL search algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

# Chapter 1

# Introduction

This report describes the design and implementation of a computer system that was built to store sequences and related annotations.

The main goal of the system is to be as flexible as possible, giving the possibility to attach an arbitrary number of annotations (which we call labels) to a sequence, to have different kinds of labels, to enable automatic label generation when a sequence is added into the system or is modified, and mainly, to make arbitrarily complex search queries using those annotated labels. Efficient search of sequences with some specific characteristics is then, the main objective of the system and its crucial functionality.

When searching for a certain set of sequences the user will also be able to apply operations to the whole sequence list, such as: add new labels, delete current labels, change label values, export the sequence list and get the translated sequence list, that is, get the related protein sequences from a list of DNA sequences.

Each label stored in the system has a certain number of properties that define the label behavior. These properties will indicate if the the label should be auto generated when a sequence is added, if it is user editable or deletable. More importantly, the system must store, if that is the case, the code used to generate default label values.

The system will support basic label types such as: integer, float, boolean, simple text, date, url, a sequence position, reference to a sequence, a file (basically the same as attaching files to a sequence) and a reference to a taxonomy.

Another important facility that must be present in the system is the use of taxonomies, supporting the storing of various taxonomy trees and a specific label type that is a reference to a taxonomy. The basic use case of this functionality is the creation of a label named 'species' and then attaching this 'species' label to specific sequences.

The NCBI taxonomy tree is present by default, but the system allows the creation of new and custom taxonomy trees.

Every data stored should also be exportable and importable, to and from files. The system accepts the FASTA and XML formats to export sequences and related annotations. The XML format is used to export everything else.

The system, implemented as a web application, uses the client/server design architecture. Some features, like searching and data viewing, can be used without client authentication, but everything else requires authentication. Some clients can be application administrators, given them the permission to edit user information, manage labels and do database maintenance.

The rest of the report is organized as follows. First, we analyze some systems that try

to do the same as ours. We will describe the advantages, faults and the flexibility of those systems. Next, we describe more thoroughly each system object, the sequence, the label and the taxonomy. With these objects in mind, we describe the design we come up with that tries to maximize flexibility and efficiency. The main object of analysis at this stage is the relational database we implemented to support the features. Following the design we describe some implementation details worth mentioning: the authentication system, how the label code is stored, how we generate the default label values, how do we transform the queries into SQL code and how we parse complex text queries into a manageable format, etc. Next, we describe important user interface elements, like the search and taxonomy tree browsing interfaces. Then we show time results for search operations to analyze the efficiency of the solution. Finally we describe the major difficulties, the future work and the project conclusions.

# Chapter 2

# Related Work

In this section we give an overview of some software systems that support sequence annotation and searching, and thus, are similar to the system we have implemented.

## 2.1 Genotator

Developed by Nomi Harris, Genotator [17] (formerly known as Genotater) is an annotation workbench consisting of a program that runs various sequence analysis programs, and a standalone annotation browser.

The goal of Genotator is to run a series of sequence analysis tools and display the results in such a way that various predictions can be compared. The user will then be able to examine all the annotations and select the ones that look the best. Useful annotations include homologies to known genes, possible gene locations, gene signals such as promoters, etc [18].

The Genotator back end runs several gene finders, homology searches (using blast), and signal searches and saves the results in .ace format. Genotator thus automates the tedious process of running a dozen different sequence analysis programs with a dozen different input and output formats.

Two displays are supplied: the map display (Figure 2.1), where the annotations are shown, and the sequence display. The application can also search a sequence for specific patterns using regular expressions, when in the sequence display.

Like BioSeD, Genotator can generate sequence annotations, although it fails to provide an integrated search environment over a set of sequences, using the generated sequences.

## 2.2 BioDAS

The Bio Distributed Annotation System (BioDAS [19]) defines a communication protocol used to exchange annotations on genomic or protein sequences. It is motivated by the idea that such annotations should not be provided by single centralized databases, but should instead be spread over multiple sites. Data distribution, performed by DAS servers, is separated from visualization, which is done by DAS clients. The advantages of this system are that control over the data is retained by data providers, data is freed from the constraints of specific organizations and the normal issues of release cycles, API updates and data duplication are avoided. [20]

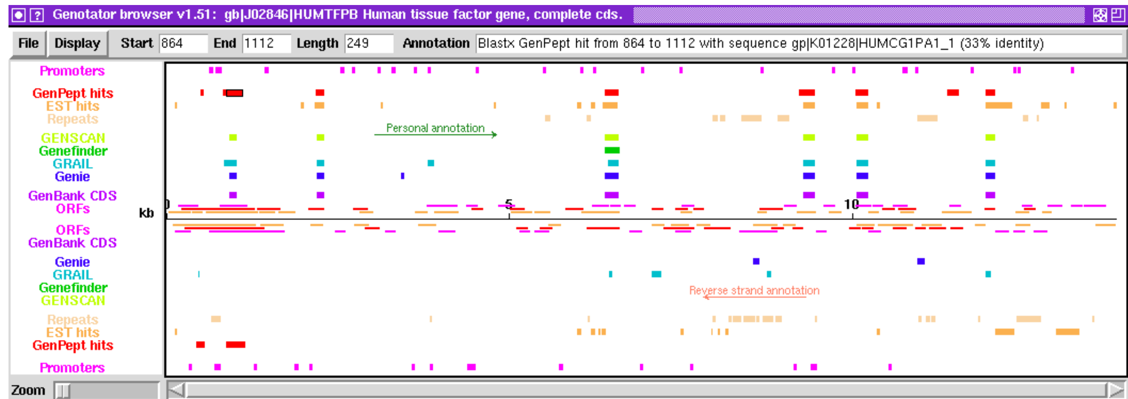



Figure 2.1: Genotator map view.

DAS is a client-server system in which a single client integrates information from multiple servers. It allows a single machine to gather up sequence annotation information from multiple distant web sites, collate the information, and display it to the user in a single view. Little coordination is needed among the various information providers. Some well known DAS clients are: Ensembl [21], Gbrowse [22], IGB [23], etc.

Contrarily to BioDAS, BioSeD works by centralizing the data on a single database, where the clients can then make use of it through a web interface. One distributed aspect of BioSeD is present in the data exchange facilities, as they make possible the integration of data from one application into another. In BioSeD every piece of data must adhere to the supported data formats, as there is no communication protocol.

## 2.3 EnsMart

The EnsMart system provides a generic data warehousing solution for fast and flexible querying of large biological data sets and integration with third-party data and tools.

The system consists of a query-optimized database and interactive, user-friendly interfaces. A wide variety of complex queries, on various types of annotations, for numerous species are supported. Users can group and refine biological data according to many criteria, including cross-species analyses, disease links, sequence variations, and expression patterns. Both tabulated list data and biological sequence output can be generated dynamically, in HTML, text, Microsoft Excel, and compressed formats.

The EnsMart database can be accessed via a public web site (like BioSeD), or through a Java application suite. Both implementations and the database are available for local installation, and can be extended to custom data sets. [25]

The EnsMart query process operates in three steps: select a genome focus, filter these by criteria, and output formatted results. First select the primary result of interest: species genome and its gene, EST gene or SNP contents. Next, filter the available set of these to satisfy specific questions, by choosing criteria among genome region, known genes or user-specified lists of gene ID, where these are expressed in anatomy and development stage, homology to other species, protein domains and SNP attributes. Finally decide on results output of features, structures, SNPs, and sequences, including IDs from Ensembl and many other databases, protein and microarray attributes, disease associations, species homologies,as

well as file formats suited to spreadsheets, database or other uses. [24]

Using various data sets for sequence annotation querying, ease of use and customizability are the main advantages of this system.

The search process in EnsSmart is very similar to the one present in BioSeD, but queries in our system can be potentially more complex and arbitrary. Also, the annotation information in BioSeD is more expansible and flexible, allowing a wide variety of annotation information.

More recently, the project was renamed to BioMart, but core functionalities are still the same. [26]

## 2.4 CBS Genome Atlas Database

The CBS [27] project provides a filesystem based database for genomic data. Everything is organized in the filesystem by directories: kingdom/genus/species/strain/segment. In each directory the system puts basic information like: the FASTA file, all protein sequences derived from the GenBank annotation, etc. They also have a set of Makefile's that are used to generate various sequence annotation values.

To visualize these data, they constructed different types of chromosomal maps (atlases) some of which include the Structural Atlas, Repeat Atlas and Genome Atlas (Figure 2.2). Each atlas is available in either vector graphic format (PS) or compressed bitmap (PNG). The intermediate files used to build these atlas are maintained as well. For each property calculated, there is a corresponding list of numerical values calculated for every base pair in the generate.

Simple sequence annotations are put into a MySQL database. Complex data is linked from the filesystem to the database.

Like BioSeD, the system can then use the annotated information present in the database to sort and search using those annotated values. Visualization wise, only generation of histograms based on a specific type of annotation is available in BioSeD, more complex visualizations of data are simply not available. The use of the filesystem to store annotations is not used in our system, as it only stores data in a relational database.

## 2.5 Extensible open source content management systems and frameworks

Sean Mooney and his group explored the idea of using open source content management systems and frameworks to handle large biomedical datasets and the deployment of bioanalytic tools.

In the paper *Extensible open source content management systems and frameworks: a solution for many needs of a bioinformatics group* [28] some approaches to the use of this kind of software in a bioinformatics setting are discussed.

## 2.6 GenDB

GenDB is a genome annotation system for prokaryotic genomes and supports manual as well as automatic annotation strategies. It is in use in more than a dozen microbial genome annotation projects.

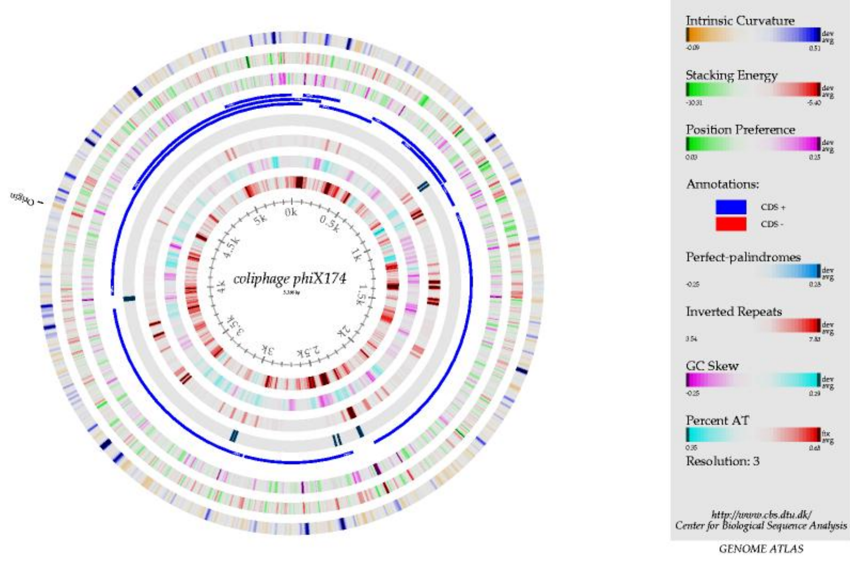



Figure 2.2: The CBS Genome Atlas.

In addition to its use as a production genome annotation system, it can be employed as a flexible framework for the large-scale evaluation of different annotation strategies.

The GenDB annotation engine will automatically identify, classify and annotate genes using a large collection of software tools, like for example, EMBOSS. BioSeD also supports automatic annotation based on certain events and can also use EMBOSS or any other tool.

GenDB offers user interfaces that allow expert annotation with large, geo-graphically dispersed teams of experts. Genes to be annotated can be categorized by functional class or gene location. A number of naming schemes (aka ontologies or functional classification schemes) are supported: GO, TIGR roles, COG, Monica Riley, MIPS. In addition to its use as a production genome annotation system, it can be employed as a flexible framework for the large-scale evaluation of different annotation strategies.

The modular system was developed using an object-oriented approach, and it relies on a relational database backend. Using a well defined application programmers interface (API), the system can be linked easily to other systems. [29] In BioSeD, systems can be linked using data interchange formats or web services using JSON [3].

# Chapter 3

# System objects

In this section we describe the main system objects: sequence, label, and taxonomy. Each object will be described by its properties, and basic organization.

## 3.1 Sequences

A sequence represents a succession of letters forming the primary structure of a DNA molecule or the structure of a protein. This succession of letters will form a special label instance of the label named 'content'. Another basic sequence label is the 'name' representing its name.

Given these two basic labels, a sequence can have an arbitrary number of labels, commonly called annotations. Each label will be an instance of a label object. An example sequence object is represented in Figure 3.1. For example we can have the label object 'length' (integer type) and have it instantiated in various sequences, and each instance representing the content length of each sequence.

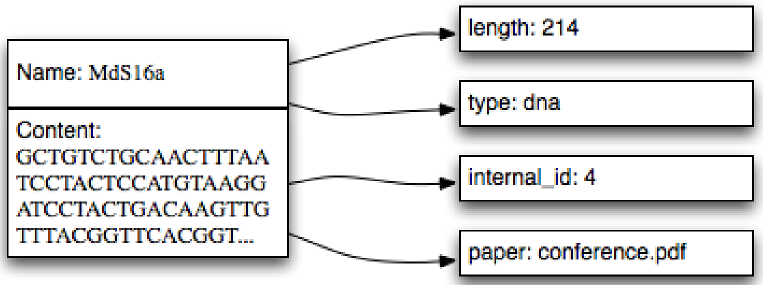



Figure 3.1: Representation of a sequence object.

## 3.2 Labels

A label object represents all the information required to instantiate, and manage label instances in the system sequences.

Each stored label has a set of properties. Each label property is described in table 3.1.

| Name | Type | Description |
|---|---|---|
| name | text | The label name. |
| type | enum | The label type. |
| default | boolean | If true this is a system label and cannot be changed. |
| must exist | boolean | If true each sequence must have one instance of this label. |
| auto on creation | boolean | If true one instance of this label will be generated using the label code for a newly created sequence. |
| auto on modification | boolean | If true and when a sequence content is modified, the corresponding label instance is automatically edited with a new value generated from the label code. |
| code | code block | The code used to generate a new label instance. This content is run in the sequence context and may access sequence data like the content. The code is in PHP. |
| valid code | code block | Code used to validate the label instance. It is also run in the sequence context. |
| action modification | code block | Block of code that is run after the sequence's name or content are modified. Useful to implement more specific label properties. |
| deletable | boolean | If true the label instances can be user deleted. |
| editable | boolean | If true the label instances can be user edited. |
| multiple | boolean | If true a sequence can have multiple label instances of the same label. |
| public | boolean | If true the label can be used in anonymous searches. |

Table 3.1: Label properties.

Label objects will generate label instances of some type (table 3.2). The type dictates the data format.

| Type | Description |
|---|---|
| integer | Simple integer type |
| float | Floating point numbers |
| text | String of text characters |
| object | Whole file: contains the file name and its content |
| position | Position in some other sequence: contains the starting position and length |
| reference | Reference to some other sequence stored in the system |
| taxonomy | Reference to a system's taxonomy |
| url | Uniform Resource Locator represented as simple text |
| bool | Boolean type |
| date | Represents a specific date / time pair |

Table 3.2: Label types.

## 3.3 Taxonomies

It is known that a taxonomy classifies another object. By relating sequences and taxonomies, we can classify some sequence by its originating species or subspecies.

In our system the taxonomy is an object that is linked to a specific rank and is aggregated in a taxonomy tree. A tree can support multiple roots.

A name given for each taxonomy is used, system wide, for reference. Unfortunately, it is well known that the same taxonomy can have multiple names for various reasons: commonly used misspellings, synonyms, names used in well known database banks (genbank), etc. To solve this problem, we associate pairs of secondary names and reasons for existence to each taxonomy.

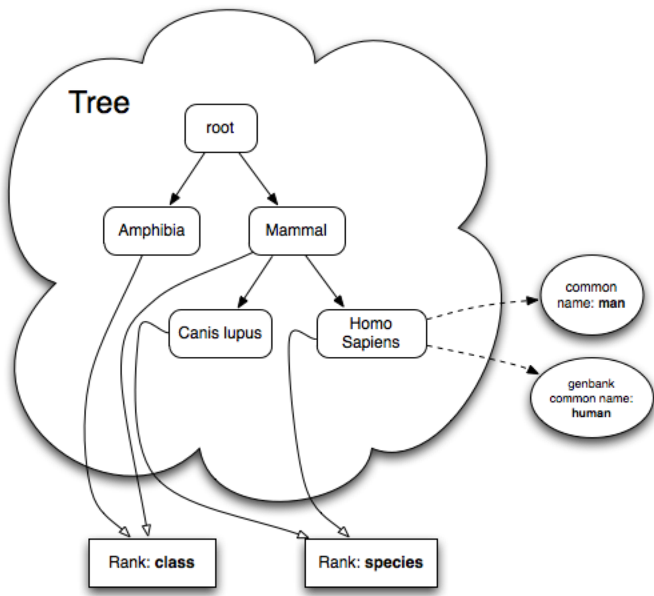



Figure 3.2: A simple taxonomy tree.

Each taxonomy can be used as label values of the type 'taxonomy' shown in table 3.2. Figure 3.2 shows an example taxonomy tree. The rounded squares represent taxonomies, each taxonomy point to a squared box representing the specific rank and some taxonomies (in the picture, "Homo Sapiens") can point to an arbitrary number of secondary names. An observation should be made: it is recurrent (and natural) that all taxonomies in the same tree level share the same rank.

# Chapter 4

# Design

This section presents the application design and its main components specified. We will start by explaining the top level architecture and how the main system components fit and work together then explaining the designed file formats used throughout the system and the query language specification.

As we have used a relational database, a relational model will be presented and analyzed. This model will show how the data has been organized to represent objects described in Section 3 and to make search functionalities more efficient.

## 4.1 Architecture

The system architecture follows the client/server model very closely (see Figure 4.1) because it is built as a standard web application. The server hosts the application and delivers html pages or JSON objects, through HTTP, to the clients on request. Each client, using a web browser, can access the system and generate new requests based on what the user is trying to accomplish.

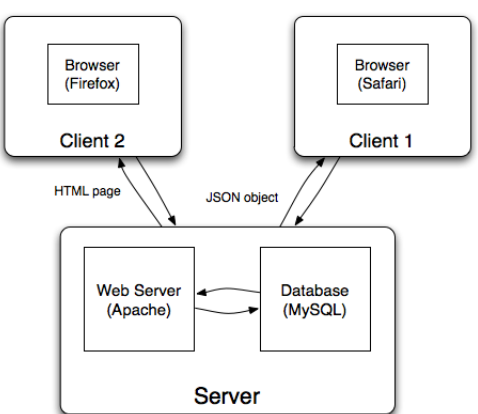



Figure 4.1: The system top level architecture.

## 4.2 Relational Model

Relational databases are a proven and reliable way of storing relational information. Given this, we designed a database model to store the system objects and everything that is needed to support basic features like user authentication.

### 4.2.1 User tables

To manage and store user information (Figure 4.2) we modeled two tables: the user table (Table B.1 on page 93) and the configuration table (Table B.2 on page 93). Each row in the user table represents an actual user and keeps information about the user name, password and email. The configuration table is used to store user configuration options, connecting an user with a configuration key and a serialized PHP [4] object, which represents the respective value.

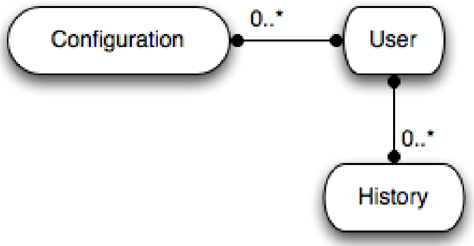



Figure 4.2: User tables.

In the history table (Table B.3 on page 94) each history row is referenced in some other database table in order to store which user and when that table object was created and who and when made the last modification/update.

### 4.2.2 Taxonomy hierarchy

The taxonomy objects presented in Section 3.3 are represented in the relational model as shown in Figure 4.3. Each tree is stored in the TaxonomyTree table (Table B.7 on page 95) and can be referenced by taxonomies themselves in the Taxonomy table (Table B.8 on page 95). Each taxonomy can point to a parent taxonomy and have multiple names represented in table TaxonomyName (Table B.5 on page 94).

In order to import taxonomies from external databases we created additional fields in the Taxonomy table, which, at cost of space, gives us faster import times. Each taxonomy name points to a TaxonomyNameType row (Table B.4 on page 94), which conveys the type of name we are trying to represent: synonyms, misspellings or something else. Each taxonomy rank is stored at the TaxonomyRank table (Table B.6 on page 95) and the rank itself can reference another parent rank.

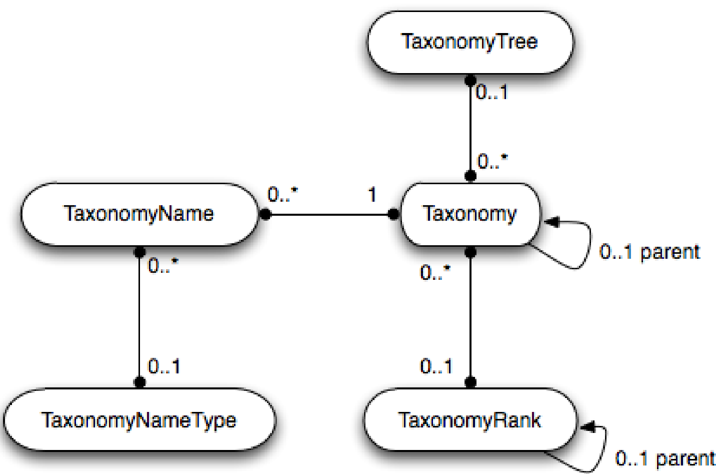



Figure 4.3: Taxonomies in the relational model

### 4.2.3 Sequence, labels and label instances

The relational design to store sequences and sequence's annotations is one of the most crucial and important aspect of the whole system, as it will greatly affect the search performance and how the sequence annotation will work at the storage level.

First, we designed a Sequence table (Table B.10 on page 97) which stores all system sequences. This table contains the value of the sequence's properties: name and content. From the user perspective these two sequence properties work as any other label, although they will skip the whole storage mechanism described below and be stored right at the Sequence table.

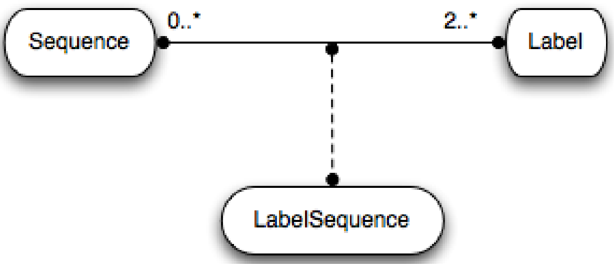



Figure 4.4: Label, sequences and label instances.

To store label information which can be used to instantiate label instances resulting in sequence annotations, we created the Label table (Table B.9 on page 96). For each Label row we store all the label properties (see Section 3.2), type, metadata and code used to be run at specific times in the system: automatic annotations, sequence modifications, label values validation, etc.

With these two tables in place, we created another table, LabelSequence (Table B.11 on page 97), which stores label instances, connecting a sequence with a label.

Each LabelSequence row contains fields for all kinds of data: integer, text, url, object, etc. But only one of these fields will be used to store the label value, if the label type is integer, only the integer data field will be used and everything else will be marked NULL.

One can argue that maintaining all those NULL fields are a waste of storage space. Fortunately in MySQL's InnoDB storage engine NULL fields don't waste unnecessary storage space, keeping the space costs at a minimum. Other relational systems also employ similar optimizations. PostgreSQL for example, uses bitmaps for NULL fields, only storing one bit for each NULL field and Oracle uses a technique called 'trailing nulls', that optimizes table space when the NULL fields appear at the row's end.

The Table 4.1 shows how each label type is stored in the LabelSequence table.

| **Type** | **Information** |
|---|---|
| Integer | Stored in the *int_data* field. |
| Text | *text_data*. |
| Float | *float_data*. |
| Bool | *bool_data*. |
| Date | *date_data*. |
| Url | *url_data*. |
| Ref | References to other sequences are stored in the field *ref_data*. This field is a foreign key to the Sequence table. |
| Obj | Reference to a file in the file table (field *obj_data*). |
| Position | The starting position is stored in the field *position_start* and the length in *position_length*. |
| Tax | References to taxonomies are stored in the column *taxonomy_data*. This field is a foreign key to the Taxonomy table. |

Table 4.1: Label types and related fields.

### 4.2.4 Miscellaneous tables

There are also other auxiliary tables that support non-core related tasks. For instance, the event table (Table B.12) stores the currently running events. Two kinds of events are supported: add/edit of label instances in multiple labels, sequence importation. Each event row contains information about some potentially long event in which the progress must be shown in the user information. Each client polls event information using the event code to refresh the interface.

Another auxiliary, yet important, table, is the file table (Table B.13). The file table stores label instance information for object type labels. Each row stores the file and an integer, representing the number of label instances linking to this file. When a new label instance refers to this file the reference count is increment and when an instance is deleted the count is decremented. When reaching zero the file is released from the database.

**Listing 4.1: Plain FASTA format.**

```
>AK315637
ELRLRYCAPAGFALLKCNDADYDGFKTNCSNVSVVHCTNLMNTTVTTGLLLNGSYSENRT
QIWQKHRTSNDSALILLNKHYNLTVTCKRPGNKTVLPVTIMAGLVFHSQKYNLRLRQAWC
HFPSNWKGAWKEVKEEIVNLPKERYRGTNDPKRIFFQRQWGDPETANLWFNCHGEFFYCK
MDWFLNYLNNLTVDADHNECKNTSGTKSGNKRAPGPCVQRTYVACHIRSVIIWLETISKK
TYAPPREGHLECTSTVTGMTVELNYIPKNRTNVTLSPQIESIWAAELDRYKLVEITPIGF
APTEVRRYTGGHERQKRVPFVXXXXXXXXXXXXXXXXXXXXXXVQSQHLLAGILQQQKNL
LAAVEAQQQMLKLTIWGVK
<...more sequences...>
```

## 4.3 File formats

We use files to import or export data in order to interchange information between different kinds of systems or among various instances of the application. These files can be in two different kinds of formats: XML or FASTA.

Among other things, those files are used throughout the system to: copy entire databases, import sequences, install new labels, import whole taxonomy trees or heterogenous integration.

### 4.3.1 FASTA

The FASTA format [6] is very well known in the bioinformatics field as it is used to store a specific set of DNA or protein sequences.

In our system, this format is used to export stored sequences or to import new ones.

We have designed two FASTA-like formats:

- Plain format

  In the plain format we just store the sequence name followed by its content.

- Complex format

  In this format we also store label instance information along the sequence data.

  The format starts by including one line comment, followed by a line telling which labels are included for each sequence. Those labels are separated by the character '|'.

  For each sequence line we put all the label instances separated by the character '|'. The order of the label instances must be equal to the label's order at the file's header.

  If the sequence does not have a specific label instance the string in that column should be empty "".

  If the label instance value is empty and that label is not editable and can be generated from code it will be automatically generated when imported.

Listing 4.2: Complex FASTA format example.

```
;flavio - Monday 19th October 2009 07:06:33 PM - sequence id 465
#name|length|internal_id|perm_public|type|translated|url
>AK315637|1554|465|0|dna|AK315637_p|[google -> http://google.pt Â§ ncbi -> http
    ://www.ncbi.nlm.nih.gov/]
ELRLRYCAPAGFALLKCNDADYDGFKTNCSNVSVVHCTNLMNTTVTTGLLLNGSYSENRT
QIWQKHRTSNDSALILLNKHYNLTVTCKRPGNKTVLPVTIMAGLVFHSQKYNLRLRQAWC
HFPSNWKGAWKEVKEEIVNLPKERYRGTNDPKRIFFQRQWGDPETANLWFNCHGEFFYCK
MDWFLNYLNNLTVDADHNECKNTSGTKSGNKRAPGPCVQRTYVACHIRSVIIWLETISKK
TYAPPREGHLECTSTVTGMTVELNYIPKNRTNVTLSPQIESIWAAELDRYKLVEITPIGF
APTEVRRYTGGHERQKRVPFVXXXXXXXXXXXXXXXXXXXXXXVQSQHLLAGILQQQKNL
LAAVEAQQQMLKLTIWGVK (...)
<...more sequences...>
```

For multiple labels, the label value is enclosed by square brackets '[]' and each instance, represented as *param -> value*, is separated by the character 'Â§'.

The special label 'name' is treated like any other label. If it is not included in the label's header, the first 10 sequence's content characters will be used by omission.

An example of this format can be seen in Listing 4.2.

### 4.3.2 XML

The XML format is widely used to export and import lots of different kinds of data throughout the system. This format can handle labels, sequences, taxonomy trees, ranks and the database itself.

- **Labels**

  Using the XML format we can export a set of labels. This file can then be imported in another system resulting in label installation or update.

  An example of this kind of file is shown in Listing 4.3 and as it can be seen, we store each label property as a XML tag.

  All the rules concerning empty label instances from the complex FASTA format are also present in this format.

- **Sequences**

  Besides the FASTA format, sequences can also be stored in XML files. The main difference between the FASTA format is that, given the structured and flexible nature of XML, it is easier to describe the sequence contents and its label instances.

  The same sequence represented in FASTA (Listing 4.2) can be seen formatted as XML in Listing 4.4.

Listing 4.3: An example Label XML file.

```
<labels>
  <label>
      <name>length</name>
      <type>integer</type>
      <comment></comment>
      <default>1</default>
      <must_exist>1</must_exist>
      <auto_on_creation>1</auto_on_creation>
      <auto_on_modification>1</auto_on_modification>
      <code>return strlen($content);</code>
      <valid_code>return $data &gt; 0;</valid_code>
      <editable>0</editable>
      <deletable>0</deletable>
      <multiple>0</multiple>
      <public>1</public>
   </label>
   <...more labels...>
</labels>
```

- **Ranks**

  To manage ranks across multiple application instances we designed a XML format to store taxonomy ranks.

  As it can be seen in Listing 4.5, for each rank we register its name and parent rank. This type of files is useful to copy rank sets around systems.

- **Taxonomy trees**

  We designed a XML format to store taxonomy trees, which is very useful to easily copy an entire taxonomy tree from one system to another.

  In this format, we store the tree name followed by a 'nodes' tag which will store, starting by the root taxonomies, the taxonomies from this tree. Each 'taxonomy' tag may contain an arbitrary number of 'taxonomy' tags which represent taxonomy's children.

- **Database**

  We designed another XML based format, this time to store the entire database. The skeleton for this format is presented in Listing 4.7 and it is organized as follows:

  - **labels**

    This section is exactly the same as the Label XML file.
  - **ranks**

    Idem, but for ranks.
  - **trees**

    A special tag containing all the taxonomy trees. Each tree is represented the way it is shown for the Taxonomy tree XML file.

Listing 4.4: An Sequence XML file.

```
<sequences>
<author>flavio</author>
<date>Tuesday 20th October 2009 12:59:53 AM</date>
<what>sequence id 465</what>
<labels>
   <label>length</label>
   <label>internal_id</label>
   <label>perm_public</label>
   <label>type</label>
   <label>translated</label>
   <label>url</label>
</labels>
<sequence>
   <name>AK315637</name>
   <content>ELRLRYCAPAGFALLKCNDADYDGFKTNCSNVSVVHCTNLMNTTVTTGLLLNGSYSENRT
  QIWQKHRTSNDSALILLNKHYNLTVTCKRPGNKTVLPVTIMAGLVFHSQKYNLRLRQAWC
  HFPSNWKGAWKEVKEEIVNLPKERYRGTNDPKRIFFQRQWGDPETANLWFNCHGEFFYCK
  MDWFLNYLNNLTVDADHNECKNTSGTKSGNKRAPGPCVQRTYVACHIRSVIIWLETISKK
  TYAPPREGHLECTSTVTGMTVELNYIPKNRTNVTLSPQIESIWAAELDRYKLVEITPIGF
  APTEVRRYTGGHERQKRVPFVXXXXXXXXXXXXXXXXXXXXXXVQSQHLLAGILQQQKNL
  LAAVEAQQQMLKLTIWGVK (...)</content>
   <label name="length">1554</label>
   <label name="internal_id">465</label>
   <label name="perm_public">0</label>
   <label name="type">dna</label>
   <label name="translated">AK315637_p</label>
   <label name="url" param="google">http://google.pt</label>
   <label name="url" param="ncbi">http://www.ncbi.nlm.nih.gov/</label>
</sequence>
</sequences>
```

Listing 4.5: An example Rank XML file.

```
<ranks>
   <rank>
      <name>class</name>
      <parent>phylum</parent>
   </rank>
   <rank>
      <name>family</name>
      <parent>order</parent>
   </rank>
   <...more ranks...>
</ranks>
```

**Listing 4.6: An example Taxonomy tree XML file.**

```
<tree>
   <name>example</name>
   <nodes>
      <taxonomy>
         <name>root_taxonomy</name>
         <rank>family</rank>
         <taxonomy>
            <name>child_taxonomy</name>
            <rank>genus</rank>
         </taxonomy>
         <taxonomy>
            <name>child_taxonomy2</name>
            <rank>genus</rank>
         </taxonomy>
      </taxonomy>
   </nodes>
</tree>
```

**Listing 4.7: Database in XML skeleton.**

```
  <biodata>
    <...labels...>
    <...ranks...>
    <trees>
      <...all taxonomy trees...>
    </trees>
    <...sequences...>
  </biodata>
```

- **sequences**
  This section follows the Sequence XML file structure.

## 4.4 Query language

A simple, yet arbitrarily complex, query language was designed to search stored sequences using annotated information present in label instances.

A simplified grammar in BNF format for this language is shown in Figure 23. Note that every label supports two basic unary operators: **exists** and **notexists**, when used they filter sequences that contain any value label or no value at all, respectively. Queries can be nested using the AND, OR and NOT operators. Parenthesis can also be used to group expressions.

⟨expression⟩→⟨expression⟩ AND ⟨expression⟩| ⟨expression⟩ OR ⟨expression⟩| NOT ⟨expression⟩|(⟨expression⟩) | ⟨terminal⟩
⟨terminal⟩→⟨label name⟩⟨unary operators⟩|⟨bool terminal⟩|⟨integer terminal⟩|⟨float terminal⟩|⟨position terminal⟩|⟨taxonomy terminal⟩|⟨text terminal⟩|⟨url terminal⟩| ⟨obj terminal⟩| ⟨date terminal⟩
⟨bool terminal⟩→⟨label name⟩|⟨label name⟩⟨bool operators⟩⟨bool value⟩
⟨bool operators⟩→⟨base operators⟩
⟨bool value⟩→ **true**| **false**
⟨unary operators⟩→**exists**|**notexists**
⟨base operators⟩→**is**| =| **eq**| **equal**
⟨integer terminal⟩→⟨label name⟩⟨numeric operators⟩⟨integer value⟩
⟨float terminal⟩→⟨label name⟩⟨numeric operators⟩⟨float value⟩
⟨position terminal⟩→⟨label name⟩⟨position type⟩⟨numeric operators⟩⟨integer value⟩
⟨numeric operators⟩→⟨base operators⟩|>| >=| <| <=
⟨position type⟩→**start**|**length**
⟨taxonomy terminal⟩→⟨label name⟩⟨taxonomy operators⟩⟨label value⟩
⟨taxonomy operators⟩→⟨base operators⟩| **like**
⟨url terminal⟩→⟨label name⟩⟨text operators⟩⟨url⟩
⟨text terminal⟩→⟨label name⟩⟨text operators⟩⟨label value⟩
⟨text operators⟩→⟨base operators⟩|**contains**| **starts**| **ends**| **regexp**
⟨obj terminal⟩→⟨text terminal⟩
⟨date terminal⟩→⟨label name⟩⟨date operators⟩⟨date value⟩
⟨date operators⟩→⟨base operators⟩| **after**| **before**
⟨date value⟩→⟨day⟩**-** ⟨month⟩ **-** ⟨year⟩
⟨label name⟩→ ⟨base label name⟩|⟨base label name⟩ [ ⟨string⟩ ]
⟨base label name⟩→"⟨string⟩"| ⟨string⟩
⟨label value⟩→"⟨string⟩"| ⟨string⟩

Figure 4.5: Query language written in BNF.

All labels support a basic set of operators: **is**, =, **eq** and **equal**. All those operators do the same thing and, depending on the label type, they filter sequences which contain the specified label value.

We can also specify a multiple label instance with the parameter selector, using *label_name[parameter]*. If an expression involves a multiple label that is not parameter specific, all label instances will be considered, instead of only one.

The following list specifies the differences for each label type:

- **Bool**

  Labels of this type can use the equal operation on values *true* or *false*. We can also skip the operator and value altogether and only keep the label name, as the example: *dna and length* $> 5$ instead of *dna is true and length* $> 5$.

- **Integer and float**

  Numeric labels use the basic comparison operators: $=$, $>$, $>=$, $<$, $<=$.

- **Position**

  For position labels we must first select between the start or the length component, and then an integer operator. Example: *label_name start* $> 5$.

- **Taxonomy and reference**

  For these kinds of labels we can also use the operator **like**, which has the same effect as the standard equal operator. Those operators work by searching all sequences or taxonomies where the name matches the provided regular expression and then filtering the result list of sequences who have at least one label instance point to the same sequence or taxonomy of the former search result.

- **Url and text**

  For these label types the operators provided are: **starts** (if the string starts with the provided value), **ends** and **regexp**, for regular expression matches.

- **Object**

  Object labels can use the same text operators to search the filename associated with the label instance.

- **Date**

  Date labels provide day based operators: **equal** (in the same day), **after** (after the day), **before** (before the day).

# Chapter 5

# Implementation

This section describes certain implementation aspects worthy to be mentioned and so, are particularly important to better understand the system.

We will cover technological aspects related to the application's architecture, database optimizations and how queries work and are translated into raw SQL code. Other important particularities like automatic label instantiation and code storage will also be analyzed.

Some aspects that we barely or not mentioned at all in the Design section will be presented in more depth here.

## 5.1 Technology

Given the program description made on chapter 1, its base architecture and restrictions, we had to chose a set of technologies to use in the implementation. Being a web application, we will differentiate between server and client technologies.

#### Server side

- **Apache**

  Apache [1] is used as the application's web server. It is a very widely used web server across the Internet.

- **MySQL**

  The relational database system used to store application data. MySQL [2] is open source software.

- **PHP and CodeIgniter**

  PHP [4] is a very widely used programming language to implement dynamic websites. *CodeIgniter* [7] is a full-featured PHP web-development framework, it adheres to the Model View Controller (MVC) design pattern to build web applications.

- **Smarty**

  Smarty [8] was used for the View component of the MVC pattern of *CodeIgniter*, functioning as an html template engine.

- **EMBOSS**

  EMBOSS [12] offers a very complete set of bioinformatics programs. It is used to compute the values for certain labels.

- **Python and MySQLdb**

  Python [10] is a dynamic scripting language and it was used to run small database scripts to install and update default system data. In the process, we used the MySQLdb [11] plugin for database interfacing.

### Client side

On the client side we used the standard HTML, CSS, JavaScript, and jQuery. Several jQuery plugins are used to easily implement certain types of user interface components.

- **HTML**

  HyperText Markup Language is the predominant markup language for web pages.

- **CSS**

  CSS is a style sheet language used to describe the look and formating of a document written in HTML.

- **JavaScript**

  Object oriented language available in the browser and used to dynamically change the web page content.

- **jQuery**

  jQuery is a client side javascript framework that concentrates on the "write less do more philosophy", it abstracts away browser incompatibilities and offers an easy way to manipulate DOM objects and write client logic.

  Techniques like asynchronous Javascript and XML are supported by jQuery. The library also supports plugins make it a flexible and open architecture.

- Impromptu - `http://trentrichardson.com/Impromptu/index.php`

  Impromptu is an extention to help provide a more pleasant way to spontaneously prompt a user for input. More or less this is a replacement for an alert, prompt, and confirm.

- Autocomplete - `http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/`

  Used to autocomplete an input field to enable users to quickly find and select some value, leveraging searching and filtering. The plugin can use remote sources, through AJAX.

- blockUI - `http://malsup.com/jquery/block/`

  BlockUI can block the user interface, simulating synchronous behavior when using AJAX. Widely used for *loading* screens.

- jQuery confirm - `http://nadiana.com/jquery-confirm-plugin`

  It displays a confirmation message before doing any image. Used to confirm data deletion.

- jeditable - `http://www.appelsiini.net/projects/jeditable`

  This plugin can transform a block of text into an input component, send the new data to the server and redisplay the original data. Used to in-place edit of fields.

- ThickBox - `http://jquery.com/demo/thickbox/`

  Its function is to show a single image, multiple images, inline content, iframed content, or content served through AJAX in a hybrid modal.

- jQuery validate - `http://bassistance.de/jquery-plugins/jquery-plugin-validation/`

  This plugin features an automatic form validation per field, displaying error messages when the user tries to send the form.

- TextGrow - `http://remysharp.com/2006/11/27/delicious-like-text-grow-jquery-plugin-2/`

  TextGrow when applied to a text field enables the field to grow as the user types text in it.

- jQuery UI - `http://jqueryui.com/`

  We used the date picker widget and some effects.

## 5.2 Authentication

In the system we have two kinds of users:

- **Normal user**

  Can do pretty much anything, except disable users, install new labels and do database management.

- **Administrator**

  Can do everything the normal user can do and: list users, disable users, install labels, cleanup the database, import database data, and change the database description.

The user type is stored along the user data.

Each user has a password to access the system. For each password we create a salted MD5 hash and then store it in the specific user row. To check if the password provided by the user is valid we generate a salted MD5 hash of the input and compare it to the stored hash.

This mechanism is widely used to prevent user passwords from being stolen when the database is compromised.

When the user successfully logs in we use the Session abstraction provided by *CodeIgniter* to indicate that the user is logged in for the next HTTP transactions.

## 5.3 Recording modifications

When a specific system object is modified or created, we need to store the user and time of that modification or creation, respectively. This information is used in the following tables: *user*, *label*, *label_sequence*, *sequence*, *taxonomy*, *taxonomy_rank* and *taxonomy_tree*. Each table has an history_id field which points to an history's table record. Each record stores pointers to creation and update users and respective time stamps.

We created triggers that automatically delete history information when any database record using it is removed. Update and insert triggers were also put in place to automatically update time information.

Some abstractions were created to automatically insert or update new database information along with history data. These abstractions are implemented in the *BioModel* class.

The BioModel class also implements some frequently used database operations and is used to subclass nearly all database models.

To know if the user is and who is logged in, we use the Session's data mentioned in the previous sub-section.

## 5.4 Tables and views

Apart from using the tables mentioned in the Relational Model Design's section, we also created tables views. These table views are used across the application to shorten SQL queries.

The views created, for each table, are the following:

- **history**
  - history_info - Joins the history table with the user table, to retrieve user information of the last update or creation.
- **label**
  - label_norm - Altered label table with the id column changed to label_id. Used by other views.
  - label_info_history - View joining label_norm and history_info views.
- **label_sequence**
  - label_sequence_extra - The table label_sequence joined with the taxonomy, sequence and file tables. The purpose is to display the taxonomy name and sequence name for label instances of the label types taxonomy and reference, respectively.
  - label_sequence_info - label_sequence_extra joined with label_norm.
- **sequence**
  - sequence_info_history - Table sequence joined with history_info view.
- **taxonomy**

- taxonomy_info - Taxonomy table joined with taxonomy_rank and taxonomy_tree, making access to taxonomy rank and tree name easy.
- taxonomy_info_history - taxonomy_info joined with history_info.

- **taxonomy_name**
  - taxonomy_name_info - taxonomy_name table joined with taxonomy_name_type.

- **taxonomy_name_type**
  - taxonomy_name_type_norm - taxonomy_name_type table normalized to use in taxonomy_name_info view.

- **taxonomy_rank**
  - taxonomy_rank_norm - taxonomy_rank table normalized to use in other views.
  - taxonomy_rank_parent_norm - taxonomy_rank table normalized to use in taxonomy_rank_info.
  - taxonomy_rank_info - Uses taxonomy_rank_norm and taxonomy_rank_parent to display the rank and parent rank name.
  - taxonomy_rank_info_history - taxonomy_rank_info joined with history_info.

- **taxonomy_tree**
  - taxonomy_tree_norm - taxonomy_tree normalized.
  - taxonomy_tree_info_history - taxonomy_tree_norm joined with history_info.

- **user**
  - user_norm - User table normalized to use in other views.
  - user_norm_creation - View used to create history_info.

The transactional storage engine used for the tables is InnoDB [5]. The main advantages of this storage engine are: *row-level locking*, *ACID compliant transactions*, *foreign key constraints support* and crash recovery. The support of foreign key constraints and ACID transactions were the biggest advantages against other engines like MyISAM.

## 5.5 Database optimizations

To shorten query execution time and disk space used, various database optimizations were made.

#### Space optimizations

For space related optimizations, we tried to carefully choose each column data type paying attention for how much space we needed for each piece of data. Appendix B describes data types for each table.

#### Query time optimizations

Some table fields are more crucial to search performance than others. With this in mind, we enumerated the most important fields and then created appropriate indexes for them.

In the following list we present the indexes created for each table:

- **user**
  - id - to locate user's using the id.
  - name - name is 32 bytes string and the index has the same size. Used to locate users by name.
  - history_id - foreign key index.
- **configuration**
  - user_id, key - used to search configuration values using the user id and configuration key.
  - user_id - foreign key index.
- **event**
  - id - primary key index.
  - event_code - to locate events using the event code.
- **file**
  - id - primary key index.
  - label_id, name - unique index.
  - label_id - foreign key index.
- **history**
  - id - to locate history records using the id / primary key index.
  - creation_user_id - foreign key index to creation user.
  - update_user_id - foreign key index to update user.
- **label**
  - id - primary index.
  - name - unique index and to locate labels by name.
  - history_id - foreign key index.
  - label_type - to locate labels by type.
- **label_sequence**
  - id - primary index.

- history_id - foreign key index.
- label_id - foreign key index. To search label instances by label.
- seq_id - foreign key index. To search label instances by sequence.
- ref_data - foreign key index. To locate the sequence this reference label points to.
- tax_data - foreign key index. To locate the taxonomy this taxonomy label points to.

- **sequence**
  - id - primary index.
  - history_id - foreign key index.
  - content - index with size 15 to do faster sequence searches using content.
  - name - index with size 10 to locate sequences by name.

- **taxonomy**
  - id - primary index.
  - import_id - to locate taxonomies using the import id.
  - rank_id - foreign key index. To locate taxonomies using rank.
  - history_id - foreign key index.
  - tree_id - foreign key index. To locate taxonomies by tree.
  - parent_id - foreign key index. To locate taxonomies by parent.
  - import_parent_id - to locate taxonomies using the parent import id.
  - name - index with size 16 to locate taxonomies by name.

- **taxonomy_name**
  - id - primary index.
  - type_id - foreign key index. To locate names by type.
  - tax_id - foreign key index. To locate specific taxonomy's names.

- **taxonomy_name_type**
  - id - primary index.
  - name - index with size 5 to locate name types by name.

- **taxonomy_rank**
  - id - primary index.
  - name - unique index.
  - history_id - foreign key index.
  - parent_id - foreign key index. To locate ranks by parent.

- **taxonomy_tree**
  - id - primary index.
  - name - unique index. To locate trees by name.
  - history_id - foreign key index.

## 5.6 Label generation

Our system supports automatic label generation using stored code to generate label instances in the database.

All the code stored is written in PHP and is evaluated with sequence and label information loaded in the execution environment (Table 5.1). All those variables can be used to generate a new label instance.

| Name | Description |
|---|---|
| sequence | array containing sequence information: id, name, content, creation and update status. |
| id | sequence id. |
| name | sequence name. |
| content | sequence content. |
| label_id | generating label id. |
| label_name | generating label name. |
| this | pointer to the model instance and *CodeIgniter* libraries. |

Table 5.1: Label generation environment.

When generating a multiple label value the code should return an array of **LabelData** instances. **LabelData** groups a multiple label parameter and a label value.

### Auto-creation labels

The field *auto_creation* on the **label** table is used to generate a label instance for a newly created instance. If this field is TRUE the code stored in the **code** field is run against the environment presented before to generate a new label instance. If the value returned by the code is **null** or an exception is thrown, the new label instance is not added.

When inserting a new sequence into the system we fetch all labels with *auto_creation* set to **TRUE** and then we run the process for each label found.

### Auto-modification labels

When changing the sequence content the system can, based on the sequence's label instances, select those labels in which property *auto_modification* is set to **TRUE** and then regenerate the instance values.

For each label instance we run the code stored in the **code** field to generate the new values. If the label has multiple values and is not editable then we remove all the previous label instances.

#### Action-modification labels

Another helpful feature in generating label instances is action-modification labels. Those labels have the field **action_modification** filled with PHP code, and can be executed to perform some maintenance function when the sequence content is altered.

For example, we could remove label instances of a label type when the content is changed in some arbitrary way.

#### Validating label instances

When inserting or updating label values the system supports custom validation through the **valid_code label** field. This field can contain arbitrary code that should return **TRUE** or **FALSE** when the value is valid or not, respectively. It is useful to validate user input and limit label values.

When executing the validation code, apart from the environment values shown before, we also make the label value available as the **data** variable.

If the data value is a multiple label instance, the user should use the functions **label_get_data** and **label_get_param** to get the real label value and the label parameter, respectively.

## 5.7 Search

The search functionality is handled by the *Search controller* and the *Search model*. The *Search model* handles SQL query generation and the controller provides an JSON/AJAX based interface to communicate with clients.

The http method **search/get_search** can receive as POST arguments the following list:

- **start**: start of result list.
- **size**: size of returned result list.
- **search**: the serialized JSON object representing the query.
- **labels**: list of labels attached to the result list.
- **transform**: the label in which we will transform the results. We will talk about this later on.

For convenience the http method **search/get_search_total** can return the number of sequences for a specific query. When we know the total number of rows in the result, we can then fetch partial results using the **get_search** method.

### 5.7.1 Query trees

To represent query expressions our system uses a tree like structure. The search term is represented as a JSON[3] object. The query is generated by the client and sent, in a serialized form, through the network to the server, the object is then de-serialized and transformed into an object that can be read using PHP. The server uses the new object to generate the equivalent SQL query expression and then fetch all results from the database. These results are then sent back to the client.

**Listing 5.1: Terminal expression as a JSON object**

```
{label: "length", oper:"gt", value: "500"}
```

**Listing 5.2: Parametrized terminal expression**

```
{label: "url", param: "info", oper:"eq", value: "http://mysite.com"}
```

A search/query expression can recursively be defined as:

- **Terminal expression**

  Forms the basic expression, contains the target label, an operator and, optionally, the value to compare.

  A terminal expression is shown in Listing 5.1 (meaning *length* $>$ *500*). The **label** property accepts the search label, the **oper** property is the operator and the value used in the expression is put in the **value** property.

  If the label is multiple, the expression can be extended with the **param** property, indicating the multiple parameter to use. Listing 5.2 displays the JSON object for the expression *url[info] = "`http://mysite.com`"*.

  See Table 5.2 for a complete list of operators and values for each label type.

- **Structured expression**

  A structured expression groups other expressions (terminal or structured) using the special operators: AND, OR and NOT. AND and OR can have as arguments multiple expressions, the NOT operator can have only one expression: the expression to negate.

  For the example shown in Listing 5.3 <*(length* $>$ *500 and name exists) or content regexp AGTG*> we can see that each structured expression contains the **oper** property naming the special operator and the **operands** property which contains a list of expressions.

For all label types the operators **exists** and **notexists** are also available. These operators do not need the **value** property, but they can also use the multiple label parameter to check the existence of a specific multiple label instance.

Listing 5.3: Structured expression as a JSON object

```
{oper: "or",
 operands: [
  { oper: "and",
    operands: [
      {label: "length", oper: "gt", value: "500"},
      {label: "name", oper: "exists"}
    ]
  },
  {label: "content", oper: "regexp", value:"AGTG"}
 ]
}
```

Listing 5.4: Resulting SQL query.

```sql
SELECT DISTINCT id, name
FROM sequence_info_history
WHERE ((EXISTS(SELECT label_sequence.id FROM label_sequence
               WHERE label_sequence.seq_id = sequence_info_history.id
          AND label_sequence.label_id = 1 AND int_data > 500 ))
      AND (TRUE)) OR (content REGEXP 'AGTG')
ORDER BY `name` ASC
```

### Generating SQL from query trees

Before retrieving the results from the database, the system needs to transform a query object into an SQL expression, which MySQL can understand and execute.

Algorithm 1 builds an SQL WHERE expression to use in a number of SQL queries: to fetch search total, to fetch sequences, to transform a result list, etc.

For example when searching for sequences where *<(length > 500 and name exists) or content regexp AGTG>* the SQL to fetch the result is shown in Listing 5.4.

## 5.7.2 The transform label

The query result can also be transformed by a specified label, that is, the user can input a label of type reference to transform all the sequences into the corresponding reference label instance.

If the original results only contain DNA sequences and if all those sequences contain a label named **protein** that links to the corresponding protein sequence, we can transform the results to get all the related protein sequences.

The transform operation happens at SQL code generation time. The template generated for this is shown in Listing 5.5, where **sql_where** is the resulting SQL WHERE condition mentioned before and **transform_label** is the transform label ID.

Listing 5.5: Transform label SQL code template.

```
SELECT id, name
FROM  (SELECT id AS orig_id FROM sequence_info_history WHERE $sql_where)
    all_seqs
      NATURAL JOIN
      (SELECT seq_id AS orig_id, ref_data AS id FROM label_sequence WHERE
         label_id = $transform_label
                                                AND ref_data IS NOT NULL)
                                                    label_seqs
      NATURAL JOIN sequence_info_history
ORDER BY 'name' ASC
```

### 5.7.3 Parsing written queries

The system features a search box where the user can write textual queries and get results. For example, one can input <*(length > 500 and name exists) or content regexp AGTG*> and the application will build the query JSON object to fetch the results.

Query parsing is performed in the server side and the client only sends the query text. First, a *tokenizer* class is used to separate the query text into tokens. These tokens are then processed by the parser, which builds the query object.

The *tokenizer* supports the **look ahead** operation to fetch the next token. This works by keeping an arbitrary number of tokens inside a queue.

The parser follows the grammar described in Section 9 and is implemented in a top-down fashion [13], building the query object as it consumes the tokens.

## 5.8 Linking DNA and proteins

The label **translated** is available by default in the application. This is a reference label that links DNA sequences to protein sequences and vice-versa. Some parts of the system code use this label in an hardcoded fashion, still the user can use it like any standard label.

When importing FASTA or XML files with sequences the user can input two files: one with DNA sequences, the other with proteins. The system will automatically add translated instances for each sequence pair, in the same order that they appear in the files.

Furthermore, when generating protein sequences from DNA, the application will also insert translated instances.

The system will also show a link for the translated sequence when viewing sequences.

Once the sequences are annotated with the translated label, the user can search using the label or transform a result set. For example, it is possible to fetch all protein sequences from a result set of DNA sequences.

The label can also be automatically generated for DNA sequences, creating a new protein sequence and then linking them.

## 5.9 Exporting data

As explained in Section 4.3, a set of file formats was defined to interchange data between other systems.

The following system objects can be exported:

- **sequences**: The user can export one sequence, all sequences or a search result. The user has the possibility to select which labels will be used in the exported file.

  To export a search result we use the query object and then export the resulting sequences.

- **labels**
- **ranks**
- **taxonomy trees**
- **whole database**: All sequences, labels, taxonomy trees (except the NCBI database) and ranks are exported.

The sequences can be exported in XML, FASTA, phylip, phylip 3, nexus, nexusnon, mega, meganon, paup and paupon formats. Remaining system objects can only be exported in the XML format.

All formats, except XML and FASTA, are generated by the seqret [14] EMBOSS utility using a temporary FASTA file and no label information is kept, only the sequence name and content.

## 5.10 Importing data

For each system objects a few considerations must be made concerning data importation:

- **sequences**: Only data in FASTA and XML formats are supported. When a sequence with the same name and same content is found in the system, the system updates it, otherwise the sequence is added to the database.

  The user can choose to do a simple import or an import followed by a DNA sequence conversion to protein and automatic sequence linking. The user can also import two files, one with DNA sequences and another with proteins. In this case, the sequences are linked pairwise.

- **labels**: If the label is already in the system, it is updated, if not, we add a new one.
- **ranks**: Works just like labels.
- **taxonomy trees**: Idem.
- **whole database**: All the cases above.

Listing 5.6: Making distribution table for the name label.

```
SELECT name AS distr, count(seq_id) AS total
FROM ($base_sql) seqs
       NATURAL JOIN
      (SELECT id AS seq_id, name FROM sequence) allseqs
GROUP BY name
ORDER BY name ASC
```

## 5.11 Search operations

In this section, we describe implementation details for operations that are run against a set of sequences generated from a query expression.

Please note that the export operation was already discussed in Section 5.9, in the context of other types of data.

### 5.11.1 Histogram generation

To analyze a search result the system features a histogram generation facility. This feature can plot an histogram for label instance values across a list of sequences. Every type of label can be analyzed, but numeric analysis like the average is only done for numeric types: integer and float.

This functionality is implemented in the Plotter *CodeIgniter* library. To get the distribution table we used the search data model and SQL generator algorithm to filter by query expression.

Using the histogram feature, the user can visualize the distribution of sequence length's for a specific search result, which includes: the histogram, number of sequences with that label, number of classes (lengths), the smallest length, the largest class, the average, median and mode length.

For special purpose labels, like **name**, **content**, **creation_user**, **creation_data**, **update_date** and **update_user** we generate specific SQL code, like, for example, in the Listing 5.6 for the **name** label. In this example, the variable **base_sql** contains SQL to fetch the sequence search list.

For non-special purpose labels, we need to differentiate between numeric and non-numeric labels.

For numeric labels we used the SQL code shown in Listing 5.7. Please note that if the label is numeric and multiple, the user can select, for each sequence, the minimum, maximum or average value representative for that sequence. On the code listing, the variable **field** represents the field for that kind of label type and **sql_distr** is a MIN(field), MAX(field) or AVG(field), defaulting to AVG(field) when the label is not multiple.

For non-numeric values, sequences with multiple values can be represented more than once, but if a multiple parameter value is given, we use it to filter the distribution. Listing 5.8 show us the code to get the distribution table.

Listing 5.7: Making distribution table for numeric labels.

```
SELECT distr, COUNT(distr) AS total
FROM (SELECT seq_id, $sql_distr AS distr
       FROM ($base_sql) seqs
            NATURAL JOIN
            (SELECT seq_id, $field FROM label_sequence
             WHERE label_id = $label_id $param_sql) labels
        GROUP BY seq_id) distr_table
GROUP BY distr
ORDER BY distr ASC
```

Listing 5.8: Making distribution table for non-numeric labels.

```
SELECT distr, COUNT(distr) AS total
 FROM ($base_sql) seqs
    NATURAL JOIN
  (SELECT seq_id, $field AS distr FROM label_sequence_extra
   WHERE label_id = $label_id $param_sql) labels
GROUP BY distr
ORDER BY distr ASC
```

### 5.11.2 Subsequence generation

When it makes sense, the user can use position label instances to create new sequences using the position information and the original sequence content, creating sub-sequences, that can be searched and analyzed. Of course, if the position is outside the sequence content an error is shown and no sequence is created.

The generation algorithm is implemented by the SubSequence *CodeIgniter* library and uses a query object as input. Once the sequence list is fetched, we iterate over it and get label instances of the target label.

For each label instance we check if a sub-sequence can be created, if so, the sequence is created. Once created, we add special label instances to it:

- **super**: links to the original sequence.
- **super_position**: original position from where it has been generated.
- **immutable**: true boolean instance meaning that the user can't change the new sequence content.
- **lifetime**: life time of the new sub-sequence, as a date instance.

For each sequence generation the user can indicate if the new sequences should be kept in the system, if it is not the case, we add the **lifetime** label instance as a date three days in the future.

The purpose of the **lifetime** label is to prevent the system from cluttering with sub-sequences and wasting disk space along the way.

Listing 5.9: Garbage collecting sequences.

```
SELECT seq_id
FROM label_sequence_info
WHERE name = 'lifetime' AND
      date_data IS NOT NULL AND
      NOW() > date_data
```

These new sequences are removed daily by a cron job that removes all sequences with an expired **lifetime** label.

This cron job is implemented as a Python script and uses the SQL query shown in Listing 5.9.

For the original sequence we add the following label instances:

- **subsequence**: multiple label linking to sub-sequences and parametrized with position information.

The **subsequence** label makes use of the **action_modification** code presented in Section 5.6, eliminating all subsequences when the original sequence content is altered.

## 5.12 Storing system wide information

The system can be customized with: a specific comment that will appear in the page header; and a background image. The comment is stored in the **configuration** table, with key = 'comment' and user_id = 0.

The background image is stored in the **file** table, with label_id = 0 and name = 'background'. The **type** column is set according to background's file type, either jpg or png.

## 5.13 Importing the NCBI database

The NCBI (National Center for Biotechnology Information) [15] makes available a dataset with updated taxonomy data. It encompasses a scientifically accepted and large taxonomy tree of biological data.

We implemented a Python script that is able to import that tree, fetching the required files [16] and then updating the database with new information. The script opens the nodes.dmp and names.dmp files, and starts reading the nodes file. For each entry in nodes file it advances in the names file that contains taxonomy names and the scientific name is set as the main taxonomy name, everything else just goes to the **taxonomy_name** table.

We also cache the taxonomy name types we find and insert or update along the way, thus avoiding database accesses.

As the available dataset uses an id / parent_id schema, we created new columns in the **taxonomy** table named **import_id** and **import_parent_id**. These fields store the imported IDs from the NCBI database and are also used as a means of tree navigation from parent to children when the **parent_id** field is not set.

Listing 5.10: NCBI import algorithm.

```
1. Open names.dmp.
2. Open nodes.dmp.
3. While there are entries in nodes.dmp:
  3.1 Get next node entry.
  3.2 Ensure rank is installed.
  3.3 Fetch all taxonomy names for this node and advance names.dmp file position
      .
  3.4 Search for a scientific name in the vector returned before.
  3.5 If taxonomy is to be updated:
    3.5.1 Remove old names.
    3.5.2 Update information.
  3.6 If taxonomy does not exist, creat it.
  3.7 Insert other names.
```

If an imported taxonomy is not present, it will be created, if not the case, it will be updated.

The algorithm used to import is shown in Listing 5.10.

## 5.14 Resetting the database

Database reset is a useful feature when one needs to remove everything that was inserted into the database in the course of operation.

As this feature is very destructive it can only be performed by users with administration rights.

This operation will remove the following data:

- All taxonomy trees, except the NCBI tree.
- All ranks except the system defaults.
- All non-default labels.
- All sequences.
- All normal users.
- All files in the **file** table.

| Label type | Operators | Values |
|---|---|---|
| URL, text and object | • *eq*: Equal comparison.<br>• *contains*: If the label contains a substring.<br>• *starts*: If the instance starts with.<br>• *ends*: Starts counterpart.<br>• *regexp*: Regular expression matching. | The value should be a string for all operators. |
| Bool | • *eq*: Equal comparison. | The value should contain **true** or **false**. |
| Integer and float | • *eq*: =<br>• *gt*: ><br>• *lt*: <<br>• *ge*: >=<br>• *le*: <= | Values should be serialized numbers as strings. |
| Position | • *eq*: =<br>• *gt*: ><br>• *lt*: <<br>• *ge*: >=<br>• *le*: <= | Values should be an object with the property **num** containing the value and the **type** property with either *start* or *length*, to indicate the position component to compare. |
| Date | • *eq*: Equal comparison.<br>• *before*: Date is before some date.<br>• *after*: Date is after some date. | Values should be in the form *dd-mm-yyyy* like *03-11-2009* |
| Taxonomy | • *eq*: Equal comparison.<br>• *like*: A taxonomy name to search for. | For the **eq** operator the value should be an object with the property **id** indicating the taxonomy ID. The **like** operator gets a taxonomy name and then searches all taxonomies with that name in the system and if the sequence points to any of them the query succeeds. |
| Reference | • *eq*: Equal comparison.<br>• *like*: A sequence name to search for. | The values and operators work just like the taxonomy labels, but applied for sequences. |

Table 5.2: Operators and values in query objects.

```
1  forall Query Sub-Expressions do
2      Check for errors;
3      Get label information from the database if a new label reference appears;
4  end
5  Expand query expression:  begin
6      forall Query Sub-Expressions do
7          Expand the query object to simplify it;
8          Transform reference and taxonomy 'like' operator into an OR, searching before
           hand for sequences or taxonomies that match;
9      end
10     expanded_query_object ← generate new expanded query object;
11 end
12 Transform expanded_query_object into SQL:  begin
13     if query is structured then
14         if special operator is NOT then
15             Grab the first argument;
16             sql_transform ← recursively transform argument into SQL;
17             return "NOT sql_transform";
18         end
19         else if special operator is AND or OR then
20             sql_arguments ← "";
21             forall Arguments do
22                 sql_argument ← recursively transform argument into SQL;
23                 sql_arguments ← sql_arguments + special operator + sql_argument;
24             end
25         end
26     end
27     else if query is terminal then
28         if label is special then
29             return Special SQL code for each field;
30         end
31         else if label is standard then
32             oper ← transform query operator into an SQL operator;
33             value ← transform query value into an SQL value;
34             field ← get table field for this label type;
35             if expression is multiple parameterized then
36                 param_sql ← build SQL code for parameter;
37             end
38             return "EXISTS(SELECT label_sequence_info.id FROM
               label_sequence_info WHERE label_sequence_info.seq_id =
               sequence_info_history.id AND label_sequence_info.label_id = " + label_id +
               " AND " + field + " " + oper + " " + value + param_sql + ")";
39         end
40     end
41 end
```

**Algorithm 1**: SQL search algorithm.

# Chapter 6

# Default information

When installing the application, base information is also installed. This section will walk along each system object in which we install default data.

## 6.1 Labels

Our application provides a set of basic default labels. These labels will be described in this section. Some labels were already presented before but they will also be enumerated.

- **length**: integer

  Automatically computes a sequence's length. It is installed once a sequence is created.

- **refseq**: reference

  Default reference label to link sequences with.

- **refpos**: position

  Default position label.

- **url**: url

  Multiple label to annotate sequences with URL's.

- **internal_id**: integer

  Automatically computed and installed once a sequence is created. Indicates the sequence database ID. Used by the application to generate a query expression for imported sequences, using greater, equal and lesser than operators.

- **perm_public**: bool

  Automatically computed and installed boolean label. Indicates if the sequence can be publicly available, when no user is logged in. By default its value is FALSE.

- **type**: text

  Automatically computed and installed type value. It is either 'dna' or 'protein'. It runs an algorithm that can detect if the sequence is a DNA or protein sequence. Also contains validation code to limit label instance values to 'dna' or 'protein'.

- **name**: text

  The sequence name (special purpose).

- **content**: text

  The sequence content (special purpose).

- **creation_user**: text

  User that created the sequence (special purpose).

- **update_user**: text

  User that made the last sequence update (special purpose).

- **creation_date**: date

  Time of sequence creation (special purpose).

- **update_date**: date

  Time of last sequence update (special purpose).

- **translated**: reference

  Used to link DNA sequences to protein sequences and vice-versa.

- **super**: reference

  Links a sub-sequence to its original sequence.

- **super_position**: position

  Indicates the position from which this sub-sequence was generated.

- **immutable**: bool

  Special label to indicate if the sequence content cannot be altered.

- **subsequence**: reference

  Multiple label to link the original sequence to its sub-sequences.

- **lifetime**: date

  Label that indicates when the sequence should be removed.

- **letters**: integer

  Multiple label that can count the distribution for each letter in the sequence. It is automatically generated but not inserted at creation time.

- **file**: object

  Default object label.

## 6.2 Ranks

A set of default ranks are also installed. For each rank we also link to the respective parent rank, as shown in the following list in the format *rank* - *parent rank*:

- superkingdom - no rank
- tribe - supertribe
- subgenus - genus
- family - order
- species subgroup - species group
- subforma - forma
- species group - no rank
- phylum - kingdom
- superclass - subphylum
- subphylum - phylum
- subspecies - species
- forma - subvarietas
- superorder - class
- infraorder - suborder
- subclass - class
- species - genus
- superphylum - class
- subvarietas - varietas
- kingdom - superkingdom
- subtribe - tribe
- subkingdom - kingdom
- no rank - no rank
- infraclass - subclass
- varietas - subspecies
- subfamily - family
- class - phylum

- supertribe - subfamily
- superfamily - order
- parvorder - order
- suborder - order
- genus - family
- order - class

## 6.3 Taxonomy name types

We also install a default set of taxonomy name types.

- synonym
- in-part
- blast name
- genbank common name
- equivalent name
- includes
- authority
- misspelling
- common name
- misnomer
- genbank synonym
- unpublished name
- anamorph
- genbank anamorph
- teleomorph
- acronym
- genbank acronym

# Chapter 7

# User Interface

In this section we will talk about the user interface. Being a web application, all interface elements are implemented using HTML, CSS and Javascript.

The system home page can be seen in Figure 7.1. It shows the user **admin** logged in. At the left side is shown the application menu, the header shows the custom database comment and a search box ready to receive query expressions. The bigger box is where page specific content is shown.

The main page features four linked images for application most used functionalities. From left to right, top to bottom: search page, sequence upload, taxonomy browsing and sequence list.

The green background can be customized for each installation.

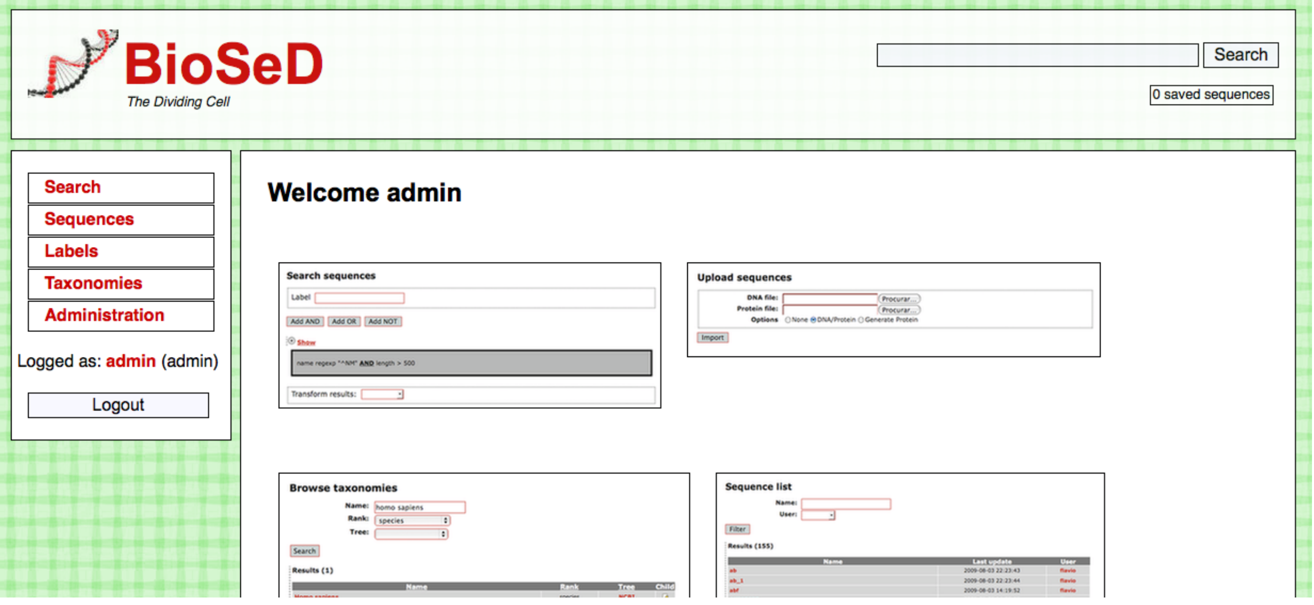



Figure 7.1: System home page.

For the rest of this section, we will describe the most interesting user interface components and interactions.

Listing 7.1: Grid component usage.

```
$('#user_list')
  .gridEnable({paginate: false})
  .grid({
    url: get_app_url() + '/profile',
    retrieve: 'get_all',
    fieldNames: ['Name', 'Complete name', 'Email', 'Last access'],
    fields: ['name', 'complete_name', 'email', 'last_access'],
    width: {
      name: w_user,
      email: w_email,
      last_access: w_update
    },
    links: {
      name: function (row) {
        return get_app_url() + '/profile/view/' + row.id;
      }
    }
  });

(...)

<div id="user_list"></div> <!-- the grid is placed in div #user_list -->
```

## 7.1 Grid component

The grid component is one of the most used interface components in the application. It works as component to display lists of things. It was implemented as a full jQuery plugin (jquery.mygrid.js).

It supports: custom columns, cell clicking, cell links, column hiding, pagination, cell edition, row remotion, whole grid reload and row highlighting. More importantly, the data it uses to display the grid can be fetched from the server (local data is also supported) and used dynamically with pagination.

An usage example is shown in Listing 7.1.

To see how a grid can look like please see Figure 7.2. The example lists all system labels.

**Results (20)** reload

| Name | Type | User | Total | Seqs | Others |
|---|---|---|---|---|---|
| content | text | --- | 404 | | |
| creation_date | date | --- | 404 | | |
| creation_user | text | --- | 404 | | |
| file | obj | admin | 404 | | |
| immutable | bool | --- | 0 | | |
| internal_id | integer | --- | 404 | | |
| length | integer | --- | 404 | | |
| lifetime | date | --- | 0 | | |
| name | text | --- | 404 | | |
| perm_public | bool | --- | 404 | | |
| refpos | position | --- | 0 | | |
| refseq | ref | --- | 0 | | |
| subsequence | ref | --- | 0 | | |
| super | ref | --- | 0 | | |
| super_position | position | --- | 0 | | |
| translated | ref | --- | 30 | | |
| type | text | --- | 404 | | |
| update_date | date | --- | 404 | | |
| update_user | text | --- | 404 | | |
| url | url | --- | 31 | | |

1

Figure 7.2: Grid component aspect.

## 7.2 Query interface

The search screen is probably the most important page in the whole application. It is structured into three parts: query input, operations and results.

A search screen showing the query input component is shown in Figure 7.3. This search page helps the user build the query expression and can display the results in real time.

To build a query the user must input the label name, operator and value and then select in which part of the search tree this new term will be placed. AND, NOT and OR operators can also be inserted. Optionally, one can also reset the whole query expression.

The query expression is shown in two formats: tree like and simple text. When selecting a term in the tree format the user can delete that sub-query. The simple text format is read-only. It is also possible to select parts of query in the tree to know how many results there are for that query sub-expression.

Every time the query expression changes, we store it in a cookie. When the search page is opened again the old query is restored. This proves useful when the user goes back and forth between the search page and each operations page, without losing any work.

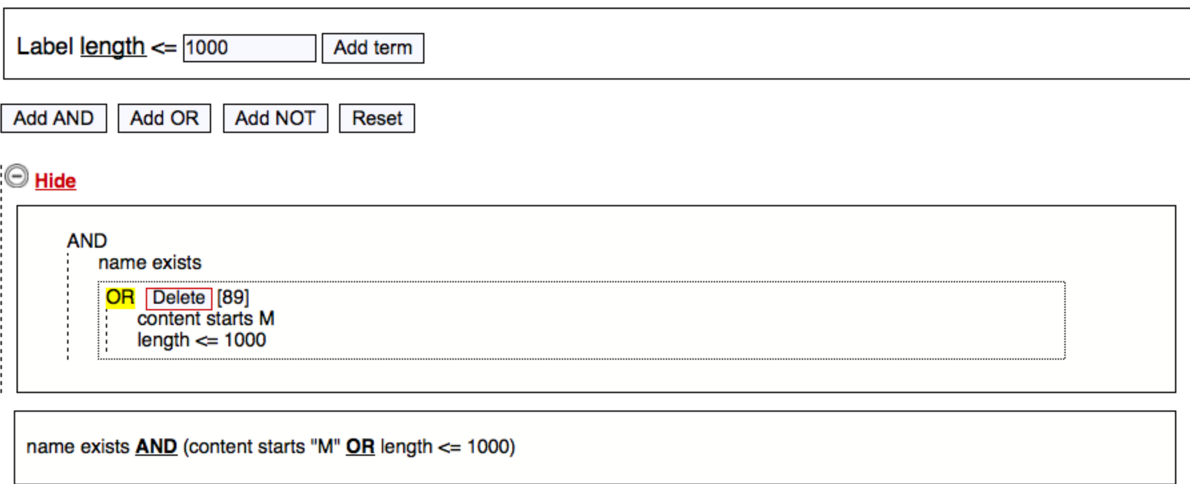



Figure 7.3: Search query input.

Operations wise, the search page features the following list:

- **Generate subsequences**: can generate sub-sequences using a **position** label and the current result list. The user has the option to keep the new subsequences around or not.
- **Generate histogram**: generate an histogram analysis based on the input label. When the label is numeric and multiple the user can choose a representative value for each sequence: minimum, maximum or average.
- **Export**: export the result sequence list to a file.
- **Add label**: adds label instances to all sequences in the result list.
- **Edit label**: same as before, but editing.
- **Delete label**: removes a specific label from the result list sequences.
- **Delete results**: simply delete the result list sequences.

Appearing right below the query input component, the operator component is shown in Figure 7.4.

Figure 7.4: Search operations.

The preview component shows all the sequences that match the query in a grid component. The grid component can be augmented with label instance information in **View label**. For example we could input the **length** label and then the grid will show the length for each sequence in the list.

Optionally we can also transform the results using the **Transform results** option and selecting a reference label. Everything is shown in Figure 7.5.

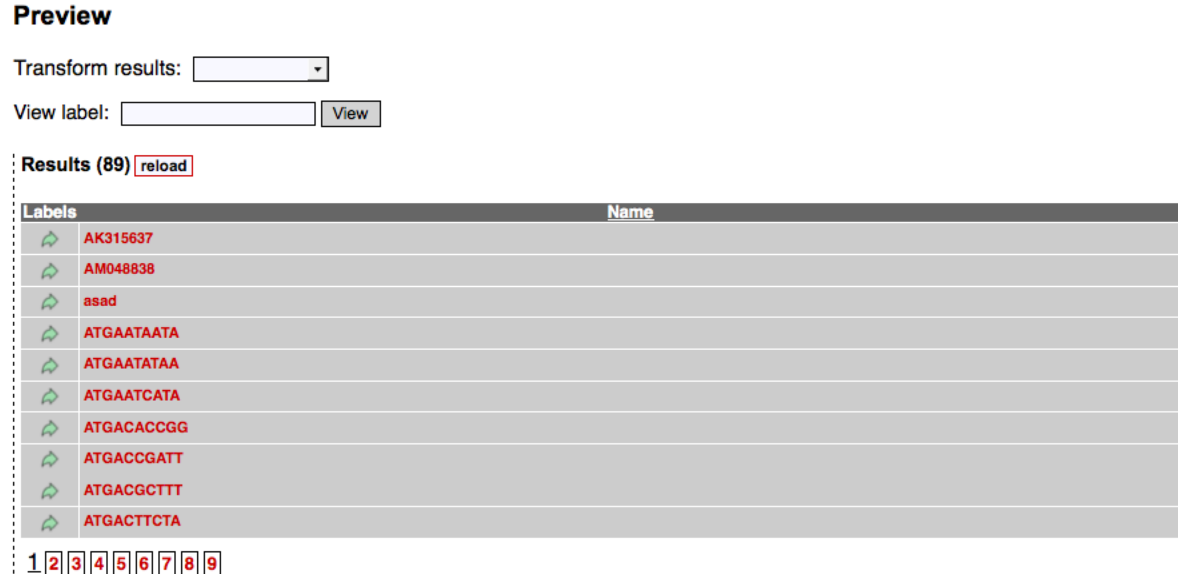



Figure 7.5: Search preview results.

Finally, three search screens are available: one can use all sequences, other only searches in DNA sequences and the third in protein sequences. Hopefully, the interface looks the same in all of them.

## 7.3 Written query interface

On the top of each page there is a search box (Figure 7.6) that can receive various kinds of input. The user can input query expressions like <*(length > 500 and name exists) or content regexp AGTG*> or some other text for system wide searches.

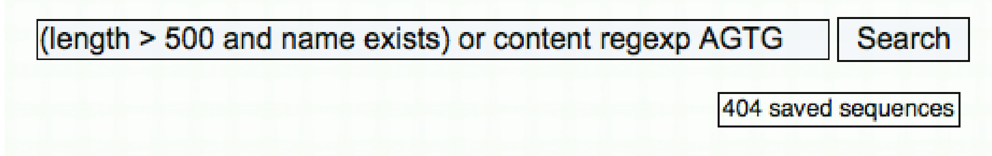



Figure 7.6: Search box.

When the server receives the POST query it tries to parse the expression as a query expression, if it fails the server will search across all system objects and display, in a different page, all the objects in which the text matched.

The objects used in this query are:

- labels
- ranks
- taxonomies
- sequences

This feature is useful to search the system in a fuzzy like way. For example, if the user types 'homo sapiens' in the box, the system will potentially display all taxonomies that contain 'homo sapiens', as shown in Figure 7.7.

**Wide search by "homo sapiens"**

*The search string provided was not valid (unknown label homo)*

**Taxonomies**

**Results (2)** reload

| Name | Rank | Tree |
| --- | --- | --- |
| Homo sapiens | species | NCBI |
| Homo sapiens neanderthalensis | subspecies | NCBI |

1

Figure 7.7: Wide search page.

## 7.4 Sequence

A new sequence can be inserted into the system by providing its name and content, as shown in Figure 7.8.

When the new sequence is DNA, an option named **Generate protein** can be checked to generate the respective protein sequence and link the DNA sequence with the translated protein.

Once the sequence is added the user is redirected to the sequence's page, where sequence's name, content and history information is shown. A button named **View Labels** is available to access the sequence labels page.

The labels page displays the sequence's annotated labels, labels available to add, missing labels and non-multiple labels that have more than one instance for this sequence, which we name the **bad multiple** labels.

The annotated labels list is always shown. The **Available** labels list is only shown if the sequence is not annotated with every system label, which is the most frequent case. The missing labels list is only shown if the sequence has not been annotated with mandatory labels, like **type** or **length**. The bad multiple labels list is only shown when, for some reason, the sequence is annotated with various instances of the same label that is not multiple, this can happen when a label, once multiple, no longer is.

Figure 7.8: Add new sequence page.

An example labels page in shown in Figure 7.9. Some sets of labels are not shown by default and must be displayed by clicking **Show**. Filtering of labels is also available as shown in the example page.

Some useful interactions were implemented: clicking in one missing label opens the available pages separator and highlights the label there, easing the process of annotating the sequence with missing labels; clicking in one bad-multiple label highlights the specific label instance in the annotated labels list.

## 7.5 Changing sequence labels

To annotate a sequence with new label instances, the user must go to the available labels list 7.10 and click the **add** icon. A new window appears to input the label instance value.

If the label supports automatic generation a checkbox named **Generate default value** is displayed. If checked in, the result will be a new label instance generated from the label code.

For each label type, this screen is slightly different. Here's a summary:

- **integer**, **float**: text field with numeric validation.
- **text**: simple text field.
- **url**: text field with URL validation.
- **bool**: a checkbox.
- **position**: two text fields with numeric validation.
- **taxonomy**: a searchable grid with taxonomies that can be selected.

**Sequence Labels**

**Name:** SEQ01
**Content:** AGTGTG...

**Associated labels**

Hide

**Name:**
**Type:**
**User:**

Filter

**Results (4)** reload

| Name | Data | Type | Edit | Delete |
| --- | --- | --- | --- | --- |
| length | 6 | integer | | --- |
| internal_id | 411 | integer | | --- |
| perm_public | No | bool | | --- |
| type | dna | text | | --- |

Show details

**Available labels**

Show

Figure 7.9: Example labels page.

- **reference**: a searchable grid with sequences that can be selected.
- **date**: text field with calendar widget.
- **object** (Figure 7.11): if the label has files attached a select list is displayed containing them. An upload field is also available.

To edit label instances, the process is similar, but uses the annotated labels list. Once a label instance is added or edited the page is updated to reflect the changes.

## 7.6 Batch uploading

The application features a page where the user can upload sequence files and have them imported into the system.

In this page (Figure 7.12) three options are available: upload a single file; upload both DNA and protein file, linking the sequences along the process; upload a DNA file and also generate protein sequences.

Once the files are uploaded and processed, a new page appears reporting the process (Figure 7.13). In this page, the status for each sequence found is displayed, if it was inserted or not, the label metadata found, if any, and the status for each annotated sequences labels.

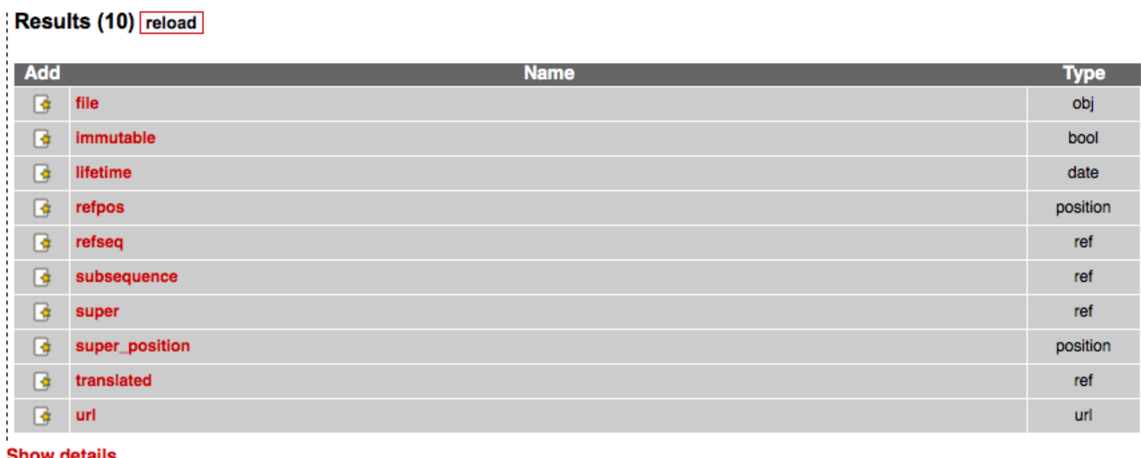



Figure 7.10: Available labels grid.

If the system found any empty sequences, then they are reported in this page. The system also creates query expressions to query the uploaded sequences, which can be useful to run various kinds of operations against the imported sequences.

## 7.7 Taxonomy tree browsing

Once the user selects a tree to browse from the system's taxonomy tree list, he can navigate through the tree, selecting nodes or going up in the hierarchy. This page also features a breadcrumb component, enabling the user to jump to a certain ancestor.

Every navigation step is done without page refreshes. An example page is shown in Figure 7.14.

## 7.8 Histograms

Histograms are useful to analyze a label's value distribution across a set of sequences. To display a histogram we implemented a jQuery plugin (file *jquery.plot.js*). This plugin accepts a javascript object representing the value distribution and creates HTML for the plot.

This plugin makes use of the width CSS property to display variable sized plot bars. An example plot bar is displayed in Figure 7.15.

## 7.9 Loading screens

For long server processing tasks and to give the user some feedback, we used the jquery blockUI plugin to implement loading screens.

These loading screens are used in three occasions: when uploading sequence files, when annotating sequences in batch or when editing label instances in batch.

For each loading screen and associate form we associate a random event code. This code is sent along the data to process and is used by the server to update the event status into the database. The client uses the event code to poll the server for information about the specific

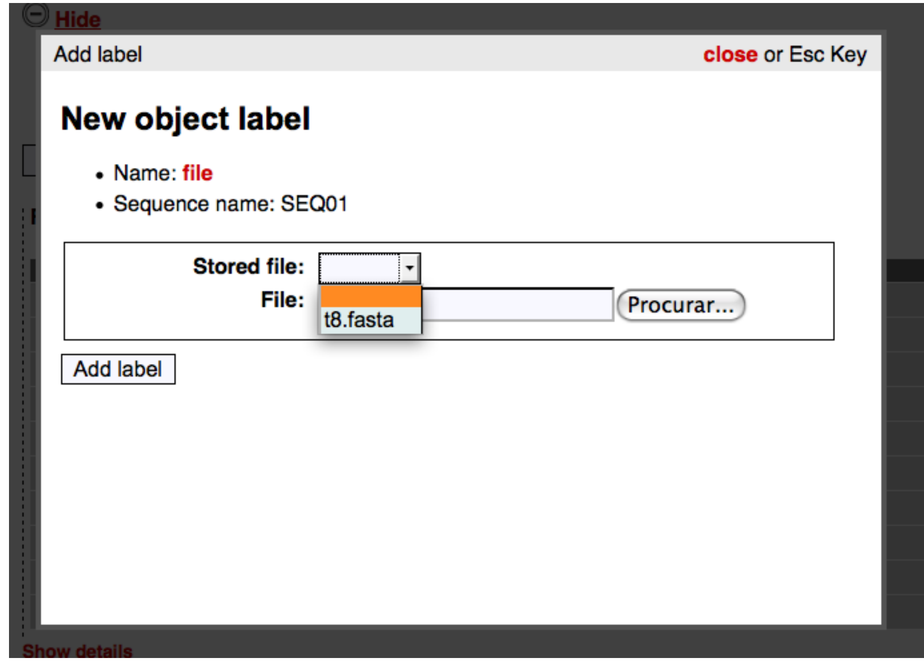



Figure 7.11: Annotating a new object label.

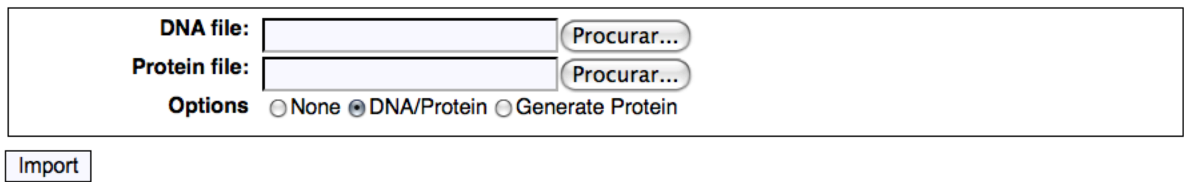



Figure 7.12: Upload sequences page.

event. When requested, the server returns HTML about the event, which is displayed in the loading screen.

An example loading screen (for sequence upload) is displayed in Figure 7.16.

Loading screens give an extra clue for when the server will complete the request and can prevent users from wondering if the server stopped responding or if the request was sent at all, among other problems.

## Batch results

The following sequences marked 'Yes' in 'New' were inserted into the database, the others were updated.

### DNA file

The imported sequences can be **batch manipulated**.

**Results (15)**

| New | Name | Content | Comment | Labels |
|---|---|---|---|---|
| No | NM_001127702 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127707 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127706 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127705 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127704 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127703 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127701 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127700 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001002236 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001002235 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |

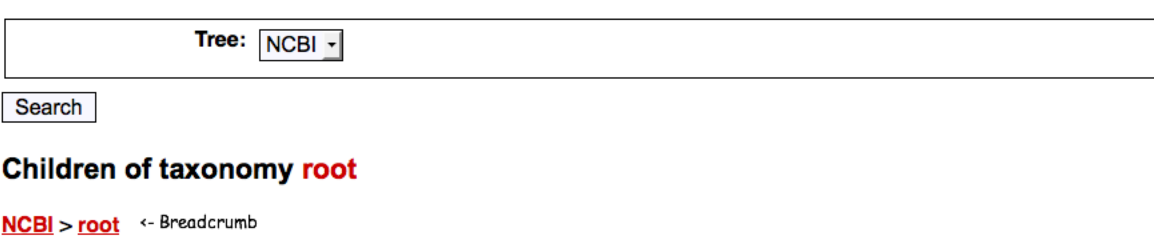
1 2

### Protein file

The imported sequences can be **batch manipulated**.

Figure 7.13: Upload report after sending a pair of DNA and protein files.

## Tree browsing




**Go up NCBI**

**Results (5)** reload

| Select | Name | Rank | Tree | Child |
|---|---|---|---|---|
| | cellular organisms | no rank | NCBI | |
| | other sequences | no rank | NCBI | |
| | unclassified sequences | no rank | NCBI | |
| | Viroids | no rank | NCBI | |
| | Viruses | no rank | NCBI | |

1

Figure 7.14: Navigating through the NCBI tree.

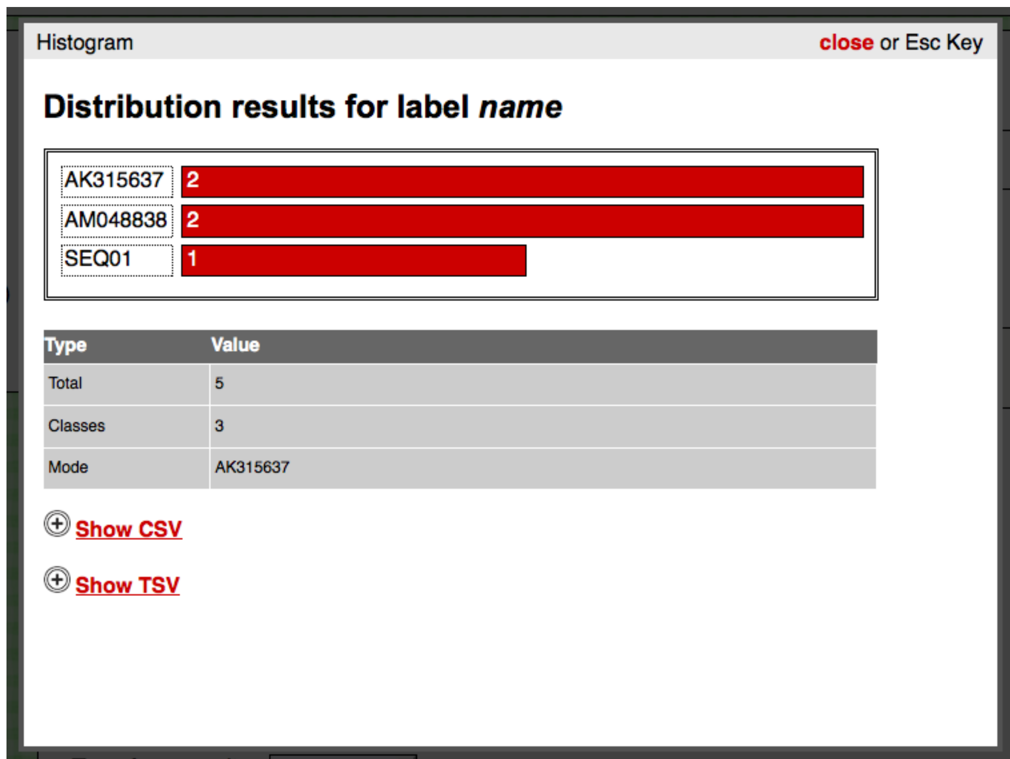



Figure 7.15: Plotting histograms.

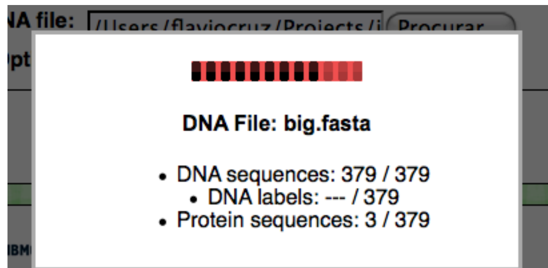



Figure 7.16: Example loading screen.

# Chapter 8

# Results

We next present some experimental results concerning time and space efficiency of various parts of the system. For our experiments we will feed scripted tasks into the server. No network latency will be involved as the scripts will be run directly in the server.

The environment for running our scripts is composed of the following components:

- Processor: 2GHz Intel Core 2 Duo
- Memory: 2GB 1067 MHz DDR 3
- Disk: 160GB 5400RPM
- Operating System: MacOSX 10.5
- MySQL 14.14 Distrib 5.1.32, for apple-darwin9.5.0 (i386)
- Apache 2.2.13
- PHP 5.3.0

To measure time and space results, we considered three main functionalities:

- Sequence import: Importing sequences into the system using the FASTA format is an important functionality and can be very time consuming. It is an important metric for how fast the system can insert new sequences, check existing annotation and update sequence annotations.
- Multiple sequence annotation: batch inserting or editing of label instances can give us some important insights on annotation performance.
- Search: Being the system core functionality it is important to measure its efficiency.

Time is measured by calculating the number of seconds and milliseconds spent executing the operation. For space measurements, we compute the database size using the SQL query presented in Listing.

Listing 8.1: MySQL database size query.

```
SELECT table_schema "Data␣Base␣Name", sum( data_length + index_length) / 1024 "
    Data␣Base␣Size␣in␣KB" FROM information_schema.TABLES WHERE table_schema = "
    BioSeD_Database";
```

## 8.1 Sequence import

Every FASTA file used in this section is available in the source code package at *fasta/*.

The scripts used to import sequence files were:

- tools/import_seq_file.sh: simple file import.
- tools/import_and_generate.sh: import a DNA file and generate proteins.
- tools/import_two.sh: import a pair of DNA and protein files.

The test files used were:

- fasta/big.fasta: 373 DNA sequences and 10 associated labels. 6 repeated sequences. All labels described in the file were installed in the system.
- fasta/serpina1.dna.fasta and fasta/serpina1.trans.fasta: pair of DNA and protein files, 15 sequences each. No labels.

The labels installed in the system during measurements are available in *fasta/test_labels.xml*. Initial database size was 247267KB.

| Test | Sequences in the system | Time | Database size | Size increment |
|---|---|---|---|---|
| (1) big.fasta with no protein translation | 0 | 1m2.931s | 249795KB | 2528KB |
| (2) big.fasta with protein translation | 0 | 2m1.767s | 250355KB | 3088KB |
| (3) serpina1.dna.fasta + serpina1.trans.fasta | 0 | 0m2.497s | 247347KB | 80KB |
| (4) big.fasta with protein translation | 748: test (2) results | 0m56.399s | 251379KB | 4112KB |
| (5) serpina1.dna.fasta + serpina1.trans.fasta | 748: test (2) results | 0m2.612s | 251939KB | 4672KB |
| (6) serpina1.dna.fasta + serpina1.trans.fasta | 778: test (5) results | 0m0.593s | 251939KB | 4672KB |

Table 8.1: Sequence importing results.

By analyzing Table 8.1 we can see that the number of sequences to import linearly affects the resulting time and space. For example, when importing translated sequences in the test (2), the test time nearly doubled from a simple import from the test (1). Another conclusion we can make is that the import time is greatly reduced when the sequences are already in the system.

## 8.2 Multiple sequence annotation

For multiple sequence annotation we loaded the system with hundreds of sequences and then batch annotated them.

The labels used in these tests are **url** (an URL multiple label) and **refpos** (a position label).

The scripts we used to run the tests were:

- tools/add_url_label.sh
- tools/add_refpos_label.sh
- tools/edit_refpos_label.sh

The database state from sequence importing test (6) was used and the initial database size was 251939KB.

| Test | Sequences | Time | Database size | Size increment |
|---|---|---|---|---|
| (1) annotate all sequences with **url** (google.pt, parameter google) | 778 | 0m7.913s | 253091KB | 1152KB |
| (2) annotate all sequences with **url** (sapo.pt, parameter sapo) | 778: test (1) | 0m8.135s | 253155KB | 1216KB |
| (3) annotate all sequences with **refpos** (start: 1, length: 10) | 778: test (2) | 0m8.029s | 253331KB | 1392KB |
| (4) edit all **refpos** instances (start: 2, length: 10) | 778: test (3) | 0m8.656s | 253331KB | 1392KB |

Table 8.2: Multiple sequence annotation.

The results can be seen in Table 8.2. Some conclusions can be made:

- Editing label values takes more time than adding them, as the edit operations requires more verifications.
- As we annotate the sequences with more labels, more time is needed to verify that we are not repeating label values. More annotations also means searching through larger database tables.

## 8.3 Search

For search we created the script tools/search.sh, which accepts a search expression and, optionally, a transform label.

First, we used a sequence set composed of serpina1.dna.fasta, serpina1.trans.fasta and big.fasta without DNA translation (404 sequences). Then we ran some test expressions against the set, then we augmented the set with big.fasta protein translations and ran the same test expressions and everything was timed.

| Expression | Results | Time |
|---|---|---|
| length exists | 404 | 0m0.371s |
| length $>$ 0 | 404 | 0m0.399s |
| length $>$ 0 and name regexp A | 47 | 0m0.157s |
| length $>$ 0 and name regexp A and content regexp A | 47 | 0m0.158s |
| length $>$ 0 and name regexp A and content regexp A and species like Drosophila | 43 | 0m8.668s |

Table 8.3: Searching 404 sequences, no result translation.

| Expression | Results | Time |
| --- | --- | --- |
| length exists | 30 | 0m0.135s |
| length $> 0$ | 30 | 0m0.209s |
| length $> 0$ and name regexp A | 4 | 0m0.136s |
| length $> 0$ and name regexp A and content regexp A | 4 | 0m0.146s |
| length $> 0$ and name regexp A and content regexp A and species like Drosophila | 0 | 0m8.951s |

Table 8.4: Searching 404 sequences with **translated** translation.

| Expression | Results | Time |
| --- | --- | --- |
| length exists | 778 | 0m0.526s |
| length $> 0$ | 778 | 0m0.323s |
| length $> 0$ and name regexp A | 90 | 0m0.183s |
| length $> 0$ and name regexp A and content regexp A | 90 | 0m0.179s |
| length $> 0$ and name regexp A and content regexp A and species like Drosophila | 43 | 0m9.628s |

Table 8.5: Searching 778 sequences with no translation.

| Expression | Results | Time |
| --- | --- | --- |
| length exists | 778 | 0m0.371s |
| length $> 0$ | 778 | 0m0.401s |
| length $> 0$ and name regexp A | 90 | 0m0.548s |
| length $> 0$ and name regexp A and content regexp A | 90 | 0m0.217s |
| length $> 0$ and name regexp A and content regexp A and species like Drosophila | 43 | 0m8.235s |

Table 8.6: Searching 778 sequences with **translated** translation.

From tables 8.3, 8.4, 8.5, and 8.6 we can see that the number of stored sequences affects the system performance. The performance also decreases when the search expression gets more complex.

# Chapter 9

# Conclusion

In this technical report we presented a possible design and implementation of a bioinformatics system that was built to store sequences and annotations, which we called labels.

We presented all the system objects that make up the bulk of the system: sequences, labels and taxonomies.

The design of a relational model with an emphasis on search facilities was put forward in the design section along with data formats and a query grammar. The query grammar, used for sequence searching, showed how search fits in the system. The main difficulty when designing the system was coming up with the relational model and making it efficient to store and retrieve thousands of rows of data, especially when using the NCBI database.

The implementation section explained how the queries are processed and run in an efficient manner using the MySQL database. Some database optimizations were listed and explained: indexes, views and keys. We build a query expression parser that is capable of parsing the grammar put forward in the design section. The parser helped turning query expressions into query objects and finally transforming them in efficient SQL code. In terms of difficulties, *CodeIgniter* had some problems when running multiple SQL queries; the Database plugin was caching all the queries, wasting a lot of memory. Another problem was related to cookie handling and encoding, mainly when storing regular expressions in cookies and some characters were lost during the encoding/decoding process.

The most interesting aspects of our user interface were displayed and described. Two types of search interfaces were designed: one more exploratory oriented, where you build the search expressions step by step and view the results in realtime and where each sub-expression can be analyzed; and another form of search, for more experienced users, where the expressions are written directly. The designed search page provides an interesting analysis and operation facility, making it easy to generate histograms for label distribution and run various tasks in a batch fashion over the result list.

Some project difficulties involved the recurrent problem in software engineering, which is the change of requirements during the project lifetime. More frequent and intensive testing were not always done and it caused some problems. The creation of a set of unit and interface tests should have been done to fight those issues.

We also made some benchmarking for the most interesting features: searching, sequence importing and multiple sequence annotation.

We conclude this report believing that the resulting system will be useful for sequence management and analysis, making the life easier for those who work with the system.

# References


[1] The Apache Software Foundation, http://www.apache.org/

[2] MySQL, Sun Microsystems, http://www.mysql.com/

[3] JavaScript Object Notation, http://json.org/

[4] PHP: HyperText Preprocessor, http://www.php.net/

[5] The InnoDB Storage Engine, http://dev.mysql.com/doc/refman/5.0/en/innodb.html

[6] FASTA format description, http://www.ncbi.nlm.nih.gov/blast/fasta.shtml

[7] CodeIgniter - Open Source PHP web application framework, http://codeigniter.com/

[8] Smarty : Template Engine, http://www.smarty.net/

[9] jQuery: The Write Less, Do More, Javascript Library, http://jquery.com/

[10] Python Programming Language, http://www.python.org/

[11] MySQLdb's User Guide, http://mysql-python.sourceforge.net/MySQLdb.html

[12] Rice, P. Longden, I. and Bleasby, A. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics* 16, (6) pp276–277, 2000.

[13] Aho, Sethi, Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986. ISBN 0-201-10088-6

[14] seqret, EMBOSS, http://emboss.sourceforge.net/apps/release/5.0/emboss/apps/seqret.html

[15] NCBI website, http://www.ncbi.nlm.nih.gov/

[16] NCBI taxonomy database, ftp://ftp.ncbi.nih.gov/pub/taxonomy/

[17] Harris, N.L. Genotator: A workbench for sequence annotation. *Genome Research* 7(7):754-762, 1997.

[18] Harris, N.L. About Genotator, http://www.fruitfly.org/~nomi/genotator/about.html

[19] Dowell RD, Jokerst RM, Day A, Eddy SR, and Stein L. The distributed annotation system. *BMC Bioinformatics 2001*; 2 7. pmid:11667947, 2001.

[20] BioDAS project. About DAS, http://www.biodas.org/wiki/Main_Page

[21] Ensembl project, How Ensembl uses DAS, http://www.ensembl.org/info/data/ensembl_das.html

[22] Stein LD et al. The generic genome browser: a building block for a model organism system database. *Genome Res 12: 1599-610*, 2002.

[23] Integrated Genome Browser, http://genoviz.sourceforge.net/

[24] D. Gilbert, Shopping in the genome market with EnsMart, *Briefings in Bioinformatics*, 2003.

[25] Kasprzyk A, Keefe D, Smedley D, London D, Spooner W, Melsopp C, Hammond M, Rocca-Serra P, and Cox T, Birney E. EnsMart: a generic system for fast and flexible access to biological data. *European Bioinformatics Institute*, 2004.

[26] Smedley D, Haider S, Ballester B, Holland R, London D, Thorisson G, Kasprzyk A. BioMart–biological queries made easy. *BMC Genomics*, 2009.

[27] Peter F. Hallin and David W. Ussery. CBS Genome Atlas Database: a dynamic storage for bioinformatic results and sequence data. *Bioinformatics* Volume 20 Issue 18, 2004.

[28] Sean D. Mooney and Peter H. Baenziger. Extensible open source content management systems and frameworks: a solution for many needs of a bioinformatics group. *Center for Computational Biology and Bioinformatics, Department of Medical and Molecular Genetics, Indiana University School of Medicine*, 2007.

[29] Folker Meyer, Alexander Goesmann, Alice C. McHardy, Daniela Bartels, Thomas Bekel, Jorn Clausen, Jorn Kalinowski, Burkhard Linke, Oliver Rupp, Robert Giegerich and Alfred Puhler. GenDB: an open source genome annotation system for prokaryote genomes. *Nucleic Acids Research*, 2003.

# Appendix A

# User manual

In this appendix we will cover the main use cases for the application, from a user point of view. After reading this section, the user should be able to install and use the application.

## A.1 Installation

Before installing the application we will need to download the install package from the project webpage. Two methods are available to download the application: downloading a release or through Subversion, getting the most recent source code updates and fixes.

First, point your browser to `http://code.google.com/p/ibmc-bio-db/`.

If you want to use Subversion, select **Source** and once the page has loaded you should see the Subversion checkout command. Run:

```
$ svn checkout http://ibmc-bio-db.googlecode.com/svn/trunk/ biosed
```

If you want to download a stable release, select **Downloads** and click on the most up-to-date version. Once the file as downloaded, run:

```
$ tar zxvf biosed-version.tar.gz
```

For both methods, enter into the **biosed** directory. Take a look at the file **README** and follow the instructions.

Once done, the application should be installed. Congratulations!

The install scripts creates one user for using the application, the *admin* user, and, as name says has more rights than *normal* users. Right now, you should point your browser to the application's location and login with this user.

Once the page has loaded, you should now see the main application's page. On the left is the main menu, in the header there is a search box and at the main box is located the main application's work area.

Right below the main menu there is a login box. Put *admin* in the user field and use the password you provided during the installation process.

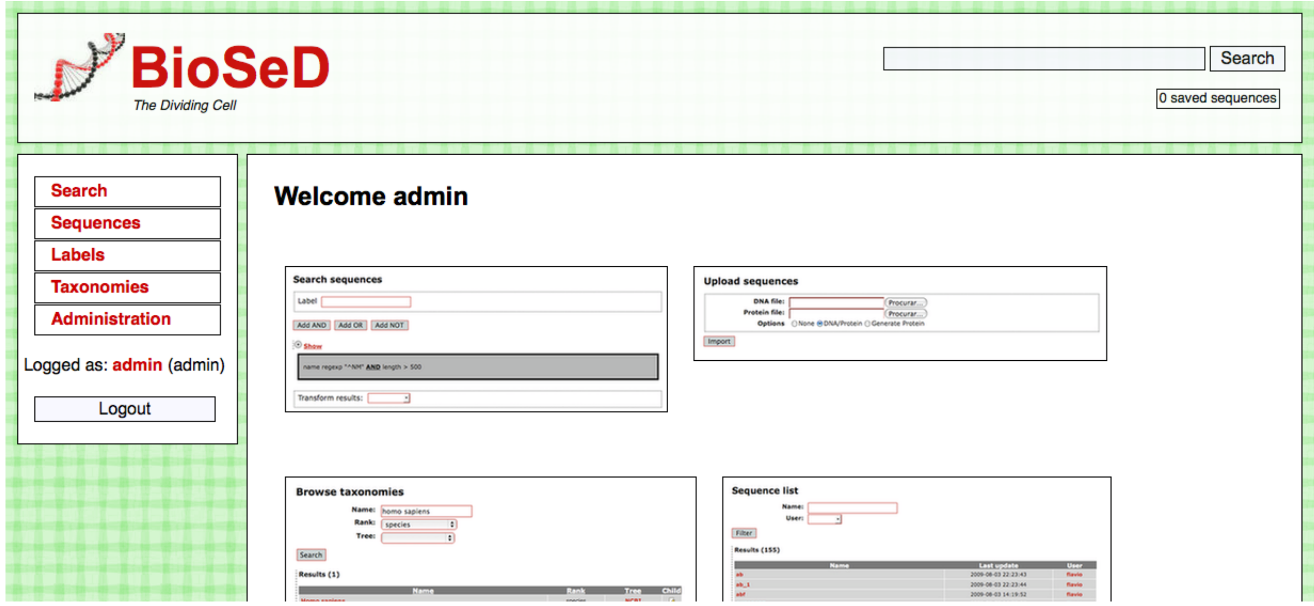



Figure A.1: System home page.

If you entered the correct password, the new page should look like Figure A.1.

## A.2 Labels

Label related options are found in the **Labels** submenu. For normal users only the options **Labels - List** and **Labels - Export** are available. The **admin** user has access to a few more options: **Labels - Add/New** and **Labels - Import**.

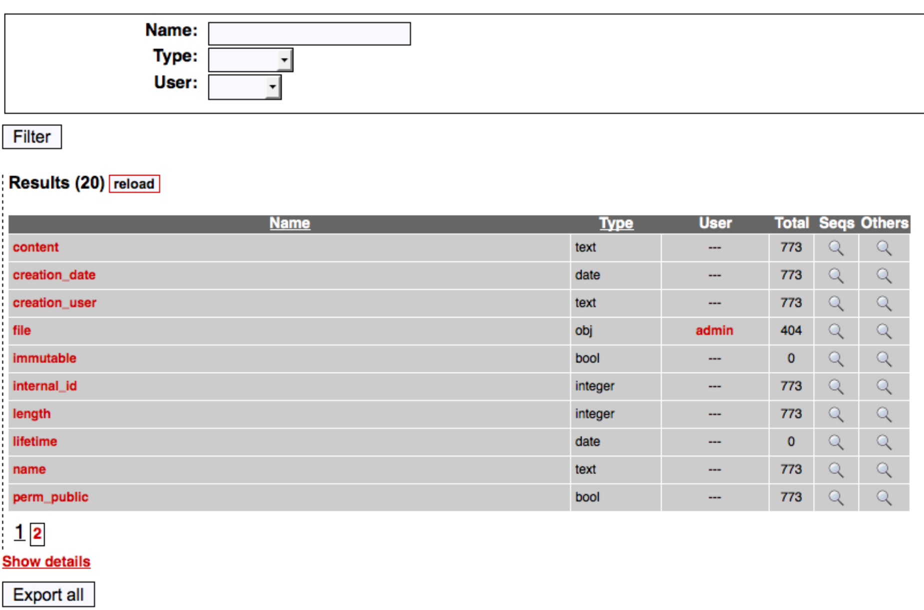



Figure A.2: Listing all labels

When listing labels, a page like Figure A.2 should appear. The traditional filter form is available and it is also possible to filter by label type.

The label types available are:

- Integer: integer value.
- Float: float value.
- Text: text value.
- Position: pair of start and length values, representing a sequence's segment.
- Reference: pointer to another sequence.
- Taxonomy: pointer to a taxonomy.
- URL
- Bool: true/false value.
- Date: day, month and year triplet.
- Object: an uploaded file.

Still in this page, one can push the button **Export all** to export the current set of labels being listed to XML. The button **Add new** redirects to the new label page, where one can create a new label.

In the label list grid, special notes should be said about the columns:

- The column **Total** tells how many sequences contain that label.

- Clicking on the label name redirects the application to the label's page.
- The column **Seqs** creates a new search page with sequences that contain that label.
- The column **Others** creates a new search page with sequences that do not contain that label.

### A.2.1 Creating new labels

If you are an administrator, pushing the **Add new** button a new form will be presented. The form contains all the fields needed to create a new label. They are:

- **Name**: the label's name.
- **Type**: the label's type.
- **Code**: the code to generate a new label value using the sequence information as input.
- **Validation code**: code to validate a new label instance. Should return true if the value is valid, false otherwise.
- **Modification code**: block of code run after the sequence's content is updated. Should not return anything.
- **Comment**: comment about the label.
- **Must exist**: if true, every sequence must be annotated with this label, when that doesn't happen the label goes to the sequence's missing list.
- **Generate on creation**: if true when a sequence is created, the sequence will be automatically annotated with this label using the **Code** field.
- **Generate on modification**: if true and if the sequence is already annotated with this label, the label value is automatically changed using the **Code** field when the sequence's content is altered.
- **Deletable**: if true a specific label instance can be user deleted.
- **Editable**: if true a specific label instance can be user edited.
- **Default**: if true the label will me made system default and cannot be edited thereafter.
- **Public**: if true the label can be made part of public (no login) searches.

All the code fields must be written in PHP [4].

Once created, you will be redirect to the label's page, where you can view or edit information about the label. Each field can be changed by clicking on it.

Other options are present: **Delete** prompts you to delete the label, **Export** exports the label to XML and **List labels** redirects you to the label list.

### A.2.2 Import / Export

To export all labels you can use the option **Labels - Export** from the main menu. This operation can be performed by any user.

Only the administrator is entitled to import files with labels. The option for this is **Labels - Import**. There you should upload a XML file containing labels. Once the file is processed a new report page is shown, as in Figure A.3.

**Import labels report**

The next table shows the import results:

**Results (1)**

| Success | Mode | Name | Type | Must exist | Creation | Modification | Deletable | Editable | Multiple | Default | Public |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Yes | Add | int_l | integer | No | No | No | Yes | Yes | No | No | No |

1

Figure A.3: Import labels report

The **Success** column tells if the label was successfully installed into the system and the **Mode** column if the label is new to the system (mode *add*) or the label was already present and it was only updated (mode *edit*).

## A.3 Sequences

All sequence related options can be found in the **Sequences** submenu. Three options can be found there: **List**, **Add/New** and **Batch**.

The **Sequences - List** option is used to list all system sequences. This page can also be accessed without logging in, where only the labels annotated with the label **perm_public** as true will appear. This page also features a filter form.

More options are available in that page, namely:

- **Export all**: exports all sequences in the grid to one of the following formats: FASTA, XML, Simple FASTA, phylip, phylip3, nexus, nexusnon, mega, meganon, paup, and paupnon.
- **Search**: launches a new search page with the sequences present in the grid.
- **Add new**: redirects to a new page, where one can insert a new sequence.

To insert a new sequence one can use the **Add new** button or the **Sequences - Add/New** option from the main menu. In the new sequence page, three fields are available: **Name**, for the sequence name; **Content**, for the sequence content and **Generate protein**, that when the sequence being insert is a DNA sequence, a protein sequence will be generated from it and the two sequences will be automatically linked using the **translated** label.

Once the sequence is inserted, the user will be redirected to the sequence page (Figure A.4). In this page, basic information about the sequence is shown, namely its name, content, a link to the translated sequence, and, in the case of sub-sequences, a link to the original

sequence. In this page it is also possible to export the sequence, by pushing the **Export** button.

The **Delete** button prompts you to delete the sequence and the **View labels** button redirects you the sequence's label page.

Figure A.4: Sequence page.

To edit sequence name or content just click in the respective name and content.

### A.3.1 Labels page

Through the **View labels** button we can access the sequence's label page.

The labels page displays the sequence's annotated labels, labels available to add, missing labels and non-multiple labels that have more than one instance for this sequence, which we name the **bad multiple** labels.

The annotated labels list is always shown. The **Available** labels list is only shown if the sequence is not annotated with every system label, which is the most frequent case. The missing labels list is only shown if the sequence has not been annotated with mandatory labels, like **type** or **length**. The bad multiple labels list is only shown when, for some reason, the sequence is annotated with various instances of the same label that is not multiple, this can happen when a label, once multiple, no longer is.

An example labels page in shown in Figure A.5. Some sets of labels are not shown by default and must be displayed by clicking **Show**. Filtering of labels is also available as shown in the example page.

Some useful interactions were implemented: clicking in one missing label opens the available pages separator and highlights the label there, easing the process of annotating the sequence with missing labels; clicking in one bad-multiple label highlights the specific label instance in the annotated labels list.

Clicking on the **Show details** link forces the annotated labels list to display more details about the labels.

Multiple labels are shown as *label_name[parameter]*, where *parameter* is the multiple label parameter.

Clicking **Add** the icon from the first column of the **Available labels** list popups a new window that will be used to create a new label instance. To delete any annotated label, just use the icons from the **Delete** column from the same list. The **Data** column displays the label value and for label types like reference, object, taxonomy or URL, a link is rendered that redirects you to the resource.

## Sequence Labels

**Name:** SEQ01

**Content:** AGTGTG...

### Associated labels

Hide

**Name:**

**Type:**

**User:**

Filter

**Results (4)** reload

| Name | Data | Type | Edit | Delete |
|---|---|---|---|---|
| length | 6 | integer | | --- |
| internal_id | 411 | integer | | --- |
| perm_public | No | bool | | --- |
| type | dna | text | | --- |

Show details

### Available labels

Show

Figure A.5: Example labels page.

The **Edit** icon from the annotated labels list launches a new window to edit the current label value. This window is similar to the one used to insert new label instances.

The edit or add label popup windows display forms where the user can insert or edit label values.

If the label supports automatic generation a checkbox **Generate default value** is displayed. If checked in, the result will be a new label instance generated from the label code.

For each label type, the popup window is slightly different. Next's a summary is enumerated:

- **integer**, **float**: text field with numeric validation.
- **text**: simple text field.
- **url**: text field with URL validation.
- **bool**: a checkbox.
- **position**: two text fields with numeric validation.
- **taxonomy**: a searchable grid with taxonomies that can be selected.
- **reference**: a searchable grid with sequences that can be selected.

- **date**: text field with calendar widget.
- **object** (Figure A.6): if the label has files attached a select list is displayed containing them. An upload field is also available.

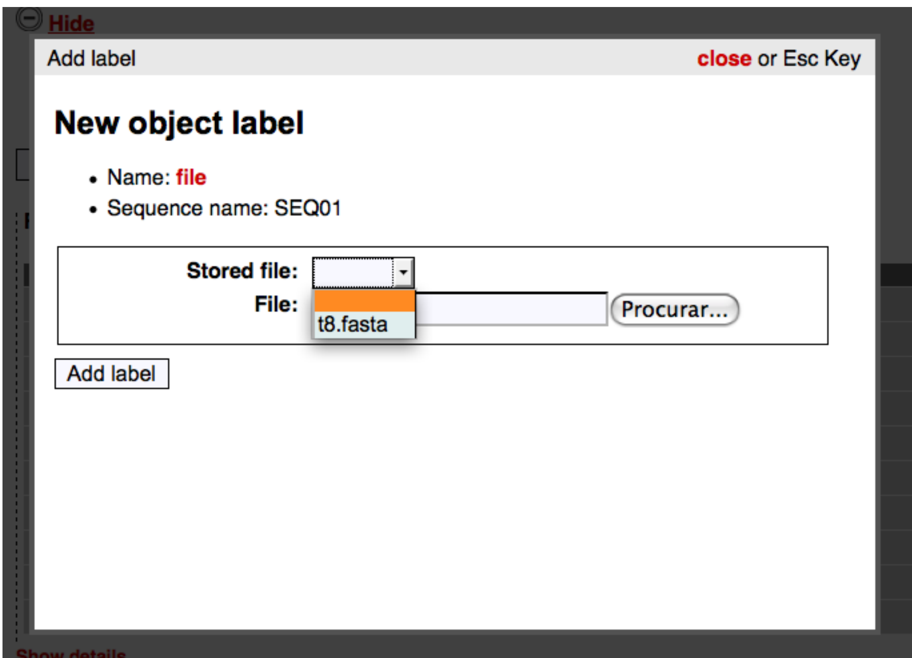



Figure A.6: Annotating a new object label.

Once a label instance is added or edited the labels page is updated to reflect the changes.

### A.3.2 Batch

Instead of inserting sequences one by one using the page described above, you can upload a file with several sequences in either XML or FASTA format. These files can be also annotated with labels as described in the section 4.3.

When uploading the file, the system will try to insert or update the stored sequences. A sequence is only updated if the application can found a previously stored sequence with the same name and content, everything else is treated as a new sequence.

To access this functionality use the **Sequences - Batch** main menu option.

In this page (Figure A.7) three options are available: upload a single file; upload both DNA and protein file, linking the sequences along the process; upload a DNA file and also generate protein sequences.

When the file is being processed a loading screen appears displaying the progress. When everything is done a report page like the Figure A.8 should appear.

If the **none** option was chosen, only a list of sequences is shown, for everything else, a list of DNA and protein sequences are shown.

If the files sent were annotated with labels, a label report is shown, indicating the state for each label. For example, if a label present in the file is not installed in the system, a

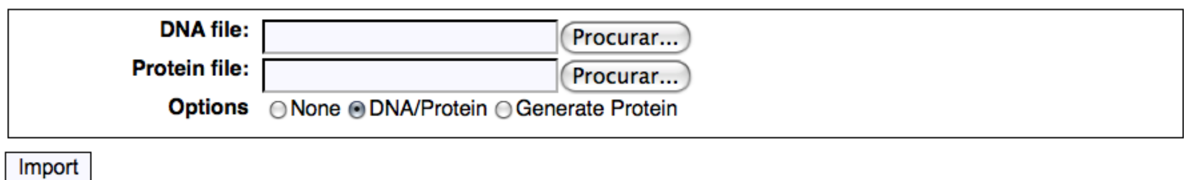



Figure A.7: Upload sequences page.

warning indicating the label is not installed will be shown. Only previously installed labels will be used when importing label values.

In each sequence list the **New** column indicates if the sequence is new and was inserted, or if it is old and it is being updated. The **Comment** label displays various kinds of informations and can tell when the sequence was only updated.

If the file was annotated with labels, a column named **Status** will appear in the sequence list. Clicking on the green arrow will popup a window with a grid, indicating for each label, the status for this sequence. For example, if a label is automatically generated when a sequence is created and its value was specified in the file, the label will not be updated and the text **Already inserted** will be shown.

Clicking on the **batch manipulated** link will redirect you to a new search page with only the imported sequences, which is useful to run various operations for all those sequences in batch mode.

## A.4 Taxonomies

To manage taxonomies we should pay attention to three things: trees, taxonomy ranks and taxonomies.

With taxonomy trees one can have multiple trees of taxonomies, which is useful to have custom taxonomies and more scientific trees like the NCBI taxonomy tree.

Ranks is one way to categorize taxonomies. The system installs a rich set of ranks, with parent/child relationships already defined.

To access taxonomy related features, use the **Taxonomies** submenu from the main menu.

### A.4.1 Managing trees

To list the currently defined taxonomy trees, select **Taxonomies - Trees - List**, there you should at least see the NCBI tree (Figure A.9).

## Batch results

The following sequences marked 'Yes' in 'New' were inserted into the database, the others were updated.

### DNA file

The imported sequences can be **batch manipulated**.

**Results (15)**

| New | Name | Content | Comment | Labels |
|---|---|---|---|---|
| No | NM_001127702 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127707 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127706 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127705 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127704 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127703 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127701 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001127700 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001002236 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |
| No | NM_001002235 | TGGGCAGGAACTGGGCACTGTGCCCAGGGCATGCACTGCC... | Sequence name and content are identical. | |

1 2

### Protein file

The imported sequences can be **batch manipulated**.

Figure A.8: Batch sequence report.

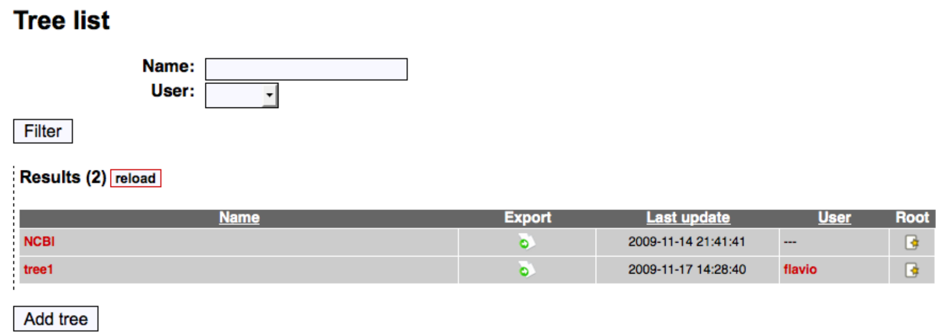



Figure A.9: Tree listing.

The user has the possibility to filter the tree list using a tree name or the user who made the last update.

From here you can select a tree to view or edit or select **Add tree** to create a new tree. To create a new tree you can also use **Taxonomies - Trees - Add/New**.

Each tree can also be exported to a XML file using the green **Export** button. The yellow **Root** button helps creating a new root taxonomy for this tree.

Selecting **Add tree**, you will be prompted for the tree name. Once created, you will be redirected to the tree's page, as in Figure A.10.

In it you can see history information and also edit the tree name by clicking on it. There is also four operation buttons:

- List trees: redirects you back to the tree list.
- Export: exports the tree to a XML file.
- Delete: prompts you to delete the tree (not available for the NCBI tree).
- Browse: enables you to browse the tree.

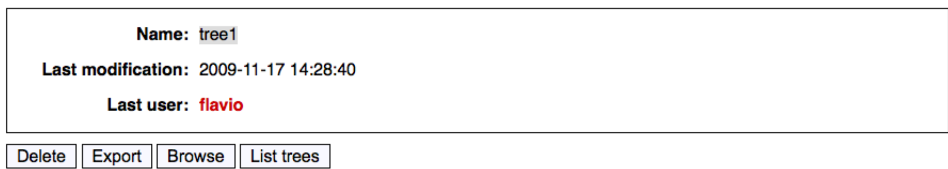



Figure A.10: Viewing a tree.

If you are an administrator you can make use of the option **Taxonomies - Trees - Import** to upload a taxonomy tree XML file. Once the file is uploaded, the system tries to insert all taxonomies from that tree into the system. If some taxonomies are found, updates are done.

### Tree browsing

The tree browser page provides an easy to use interface to navigate through the target tree.

In Figure A.11 we are navigating the NCBI tree, currently at node **root** and the grid is filled with **root**'s children.

We can go up in the hierarchy by clicking **Go up *node name***. To navigate into a taxonomy we click the green arrow in the select column for that taxonomy. We can also add a new taxonomy child, selecting the last icon from the **Child** column, it redirects us to a new taxonomy form, with the parent taxonomy already setup.

**Tree browsing**

Tree: NCBI

Search

**Children of taxonomy root**

NCBI > root <- Breadcrumb

Go up NCBI

Results (5) reload

| Select | Name | Rank | Tree | Child |
|---|---|---|---|---|
| | cellular organisms | no rank | NCBI | |
| | other sequences | no rank | NCBI | |
| | unclassified sequences | no rank | NCBI | |
| | Viroids | no rank | NCBI | |
| | Viruses | no rank | NCBI | |

1

Figure A.11: Browsing the NCBI tree.

Another important aspect is the breadcrumb component (in the example **NCBI** > **root**). Clicking in one breadcrumb name will make the current taxonomy change.

### A.4.2 Managing ranks

To list all system ranks use the option **Taxonomies - Ranks - List** from the main menu. This page gives you a filter enabled rank list, being possible to filter the rank list by rank name, parent rank name or the user who made the last rank change.

You can also order the rank names by alphabetical order, ascending or descending.

To export the current set of ranks just push the **Export all** button.

For each rank in the grid, there are two action icons: one named **Taxonomy**, that redirects us to the new taxonomy form where the rank is already selected, and the other, **Child** opens a new page with a form to create a new rank with the parent rank already set.

To create ranks, there's also the **Taxonomies - Ranks - Add/New** menu option. On this page, you should input the rank name and select the parent rank from the list of inserted ranks.

Once a rank is created, the system redirects you to the rank page where you can edit the rank name or parent rank by clicking in the respective name. Editing by clicking on the name, changes the rank name to a text field, where you can input the new name and then, when pushing the **OK** button, sends the changes back to the server.

Also, when on the rank page, there are two buttons at the end of the page: **Delete** prompts you to delete the rank and **List ranks** redirects you back to the rank list.

#### Import / Export

When logged in as **admin** you can export all system ranks through the option **Taxonomies - Ranks - Export**. The output file is XML.

The other way around, you can import a XML file with ranks, through the option **Taxonomies - Ranks - Import**. Once the file is processed by the server a page like Figure

A.12 is shown.

The column **Success** tells if the new rank was installed into the system or not, the column **Mode** indicates if the new rank was added or edited. The **Parent found** column tells if the parent was found in the system (if the system can't find it, it is created). The column **Original parent** was the rank parent before the import operation if the rank was already in the system and the column **Parent** just indicates the new parent rank.

**Import ranks report**

The next table shows the import results:

**Results (33)**

| Success | Mode | Name | Parent found | Original Parent | Parent |
|---|---|---|---|---|---|
| Yes | edit | ad | Yes | class | class |
| Yes | edit | class | Yes | phylum | phylum |
| Yes | edit | family | Yes | order | order |
| Yes | edit | forma | Yes | subvarietas | subvarietas |
| Yes | edit | genus | Yes | family | family |
| Yes | edit | infraclass | Yes | subclass | subclass |
| Yes | edit | infraorder | Yes | suborder | suborder |
| Yes | edit | kingdom | Yes | superkingdom | superkingdom |
| Yes | edit | no rank | Yes | no rank | no rank |
| Yes | edit | order | Yes | class | class |

1 2 3 4

Figure A.12: Import rank report

### A.4.3 Managing taxonomies

We have seen that there are lots of ways to get to the new taxonomy form. The standard way is to choose the option **Taxonomies - Add/New** from the main menu. There you should enter the taxonomy name, choose the rank and tree from a list of stored ranks and trees, respectively.

The rank and tree can be left empty, but as a recommendation, you should define them right from the beginning.

Once a taxonomy is created, the system redirects you to the taxonomy page. Each taxonomy page is composed of: a form where you can edit basic taxonomy information (Figure A.13), a list of optional taxonomy names (Figure A.14) and a list of children taxonomies.

In the first part (Figure A.13), you can edit the name and rank by clicking in the respective name. Changing the tree is not allowed. To change the parent you should click the red **Parent:** link and when a window popup appears you should search for your parent taxonomy, select the name from the grid and then push the **Select** button.

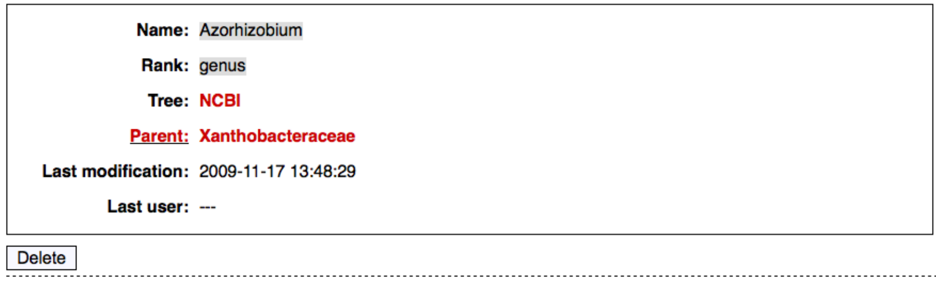



Figure A.13: Editing taxonomy information.

The **Other names** section provide a list of other names for the taxonomy. Each name can be deleted using the **Delete** column. To edit a name you should click on the name cell and then push the **OK** button. To edit the name type the process is identical. To add a name, just use the provided form.

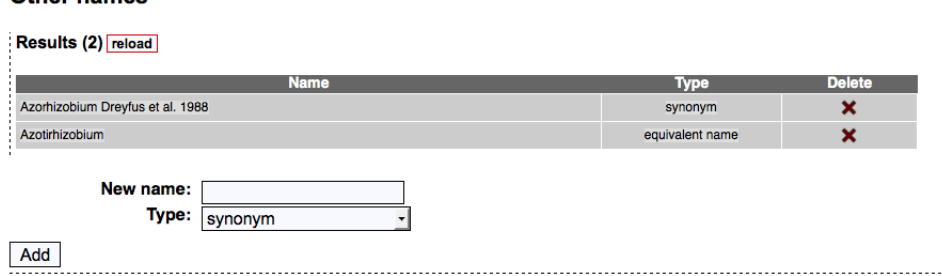



Figure A.14: Other names section.

The final section displays a grid with the children taxonomy. You can add new ones to the list by pushing the **Add child** button.

As the number of taxonomies can get very large, we provided a page where one can search by taxonomies by just using the name, tree or rank, or any combination of these.

To use this interface go to **Taxonomies - Browse**.

## A.5 Search

To search sequences using the annotated label information, one can use the search pages available from the main menu. Three pages are available:

- **Search - ALL**: to search all sequences.
- **Search - DNA**: search only DNA sequences.
- **Search - Protein**: search only protein sequences.

All the three search pages look the same, so everything we will describe for the rest of this section will apply for all of them.

Accessing any search page, we will rapidly discover three main sections in this page: the query input section, the operations section and the preview section.

### A.5.1 Query input

The query input section (as shown in Figure A.15) presents you controls to display and manipulate the query expression.

The query is presented in two formats: the tree, where you can select parts of the expression and preview in real time how many sequences are filtered using that sub-expression and the query text, where the query is presented in an human readable format. In the query tree view, apart from selecting where to insert sub-expressions, you can also delete parts of the expression by selecting an AND, OR, NOT or sub-expression and pressing **Delete**.

To insert a new query expression, you should choose where the expression will be put. That is possible selecting one of the previously inserted OR, AND or NOT expressions. If you simply want to create a simple AND query expression, press the **Reset** button and start inserting expressions. But if you want to create complex queries you should know how to insert expressions at different positions.

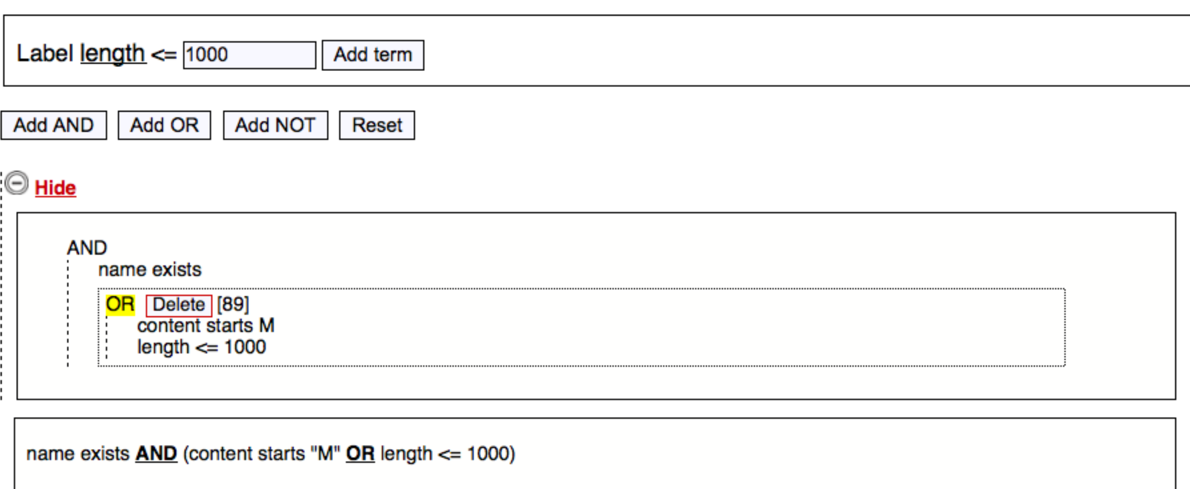



Figure A.15: Search query input.

When creating a new sub-expression, the process involves various steps:

- First, input the label name you want to use for this term in the **Label** field. The system will autocomplete the label names as you type.
- When a label is chosen, you must select the search operator. Please see the table A.1 for information about each operator.
- After the operator is chosen, you must, optionally, input the value for the operator.

  In resume, for each label type:

  - Text, URL and Object: Text field.

  - Integer, position and float: Numeric text field.
  - Bool: Checkbox.
  - Date: The date is input in a calendar widget. Please click on the text field to activate it.
  - Taxonomy: Text field for the **like** operator. For the **equal** operator you should click on the **Find taxonomy** link, search a taxonomy within the popup window and then click on the chosen taxonomy name.
  - Reference: Text field for the **like** operator. For the **equal** operator you should click on the **Find sequence** link, search a sequence within the popup window and then click on the chosen sequence.

- Press the **Add term** button. The new term should appear on the query views, and the preview section will be automatically updated.

Please remember that you can use the **exists** and **not exists** operators. These operators will filter sequences that are annotated with a label or not, respectively, and do not need values.

If the label is multiple, you can, optionally, input the multiple parameter. If the multiple parameter is not given, the search is done for all label instances. If given, the query will only use the specific label instance.

You can also insert AND, OR and NOT terms. Use the respective buttons.

| Label type | Operators |
|---|---|
| URL, text and object | • equal: Equal comparison.<br>• contains: If the label contains a substring.<br>• starts: If the instance starts with.<br>• ends: Starts counterpart.<br>• regexp: Regular expression matching. |
| Bool | • equal: Equal comparison. |
| Integer and float | • =<br>• ><br>• <<br>• >=<br>• <= |
| Position | • =<br>• ><br>• <<br>• >=<br>• <=<br>You should also select the position component, start or length for the term. |
| Date | • equal: Equal comparison.<br>• before: Date is before some date.<br>• after: Date is after some date. |
| Taxonomy | • equal: Equal comparison.<br>• like: A taxonomy name to search for. Using this operator will make system search for all taxonomies in the database with this name and then the query will match if a sequence points to any of them. |
| Reference | • equal: Equal comparison.<br>• like: A sequence name to search for. Works the same way as for taxonomies. |

Table A.1: Label types and operators.

### A.5.2 Operations

While you create your complex query you can see the preview list getting shorter and shorter, giving you immediate feedback. And now that you have your results, you can do things with them.

The operations section as it can be seen in Figure A.16, contains various operations you can do to the search result list.

Figure A.16: Search operations.

### A.5.3 Sub-sequences

To begin, we can generate sub-sequences using a position label from the result list. For example, if your sequences contain a position label for a specific sequence segment, you can generate sub-sequences for the complete set of sequences. To do that, you must select the position label from the **Generate subsequences** select list and then push the **Generate button**. If you want to keep your sub-sequences around longer than a few days, you should check the **Keep** checkbox. When not keeping the sequences around, the system will delete them after some hours.

Once the sub-sequences are generated, a report page will be shown.

### A.5.4 Histograms

Another operation is the **Generate histogram** (Figure A.17). This option can generate an histogram for that label distribution across the result list. For example, you could generate a length distribution for a given sequence set and then the system will generate the frequency histogram and display the distribution total and number of classes. For numeric labels the smallest class, largest class, average, median and mode values are also shown.

If your label is multiple and numeric, you can chose what value will be representative for each sequence. You can use the average value for all label instances from a sequence, the minimum or the maximum value. If the label is not numeric but multiple, all values will be considered.

In the popup window that appears when generating the histogram, you can also copy the distribution values to use with programs like Microsoft Excel.

### A.5.5 Export

Another important operation is the **Export** button. It can export your result sequences (Figure A.18) into the file formats mentioned before and supported by the system.

For formats like FASTA or XML, you can select the labels that will appear on the file, only exporting annotation that is important to the task at hand.

### A.5.6 Batch labels

Another common action to do is, for example, annotate a list of sequences with the same label instance. For this you can use the button **Add label**, to add, or the button **Edit label**, to edit.

Using the **Add label** option you will be redirected to a page that looks like Figure A.19. First you should enter the label name into the **Label** text field, optionally you can check the

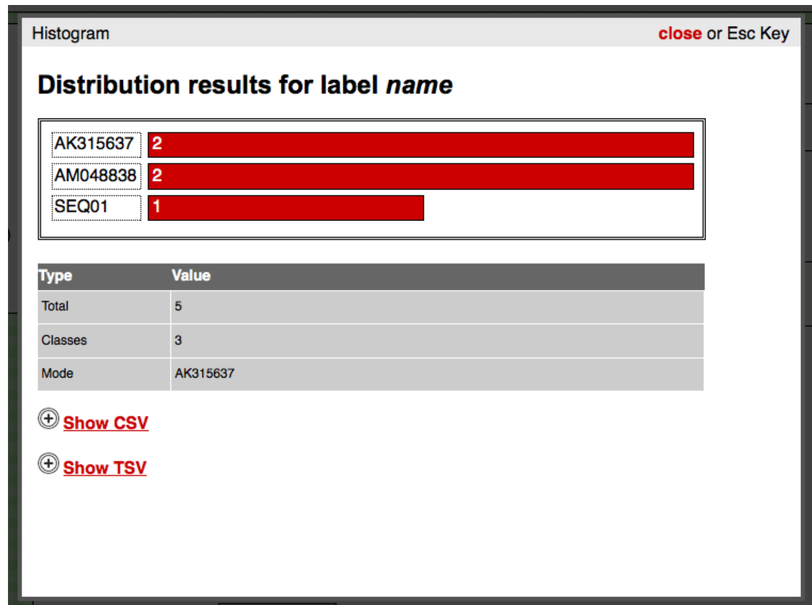



Figure A.17: Plotting histograms.

**Update** checkbox. When checked and if the sequence already contains that label instance, the current value will updated; if not checked, nothing will be done.

When ready press the **Next...** button. A popup window should appear. This window is very similar to the one we used to add a label to a sequence, so, it works the same way.

Once you input the label value, a report on the process should appear, right below the **Next...** button.

The **Edit label** operation works in a similar fashion, but instead of having the **Update** checkbox, it contains the **Add new** checkbox. This checkbox, when checked and when the sequence does not contain the label, forces the system to add a new label value. It is the counterpart of the **Update** option, but for the designed for the edit mode.

There is still another button, the **Delete label** button. This button gives you the possibility to delete a label from the set of sequences. After you put the label, press the **Next...** button and answer **Yes** and all annotations related to that label will disappear from the sequence list.

### A.5.7 Delete

If you want to delete your sequence list just select the button **Delete** and everything will be deleted. The action is irreversible, so use it with care!

### A.5.8 Preview

The last section (Figure A.20) present in the search page is the preview section. In it we can see the results (as a list of sequences) for the query being build.

## Export search

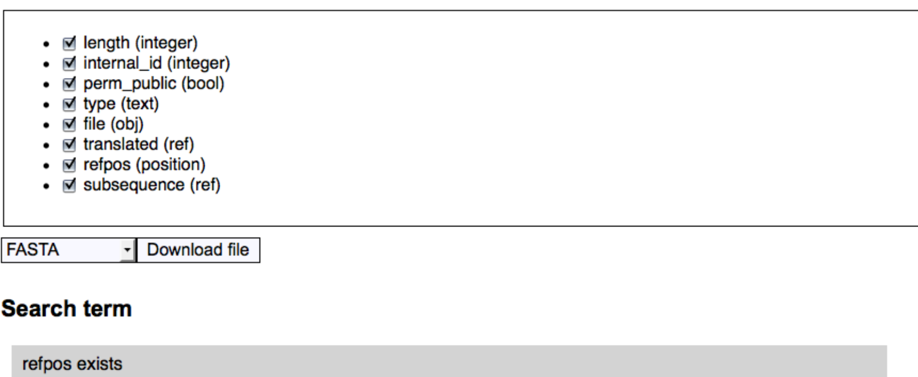



Figure A.18: Export search page.

## Add label to multiple sequences

**Label:** int_l

**Update:**

Next...

### Search term

refpos exists

### Sequences

**Results (1)** reload

| Change | Labels | Name | Last update | User |
|---|---|---|---|---|
| | | ACAAACCGTC | 2009-11-09 22:50:18 | admin |

1

Figure A.19: Add label to multiple sequences.

## Preview

Transform results:

View label: View

**Results (89)** reload

| Labels | Name |
|---|---|
| | AK315637 |
| | AM048838 |
| | asad |
| | ATGAATAATA |
| | ATGAATATAA |
| | ATGAATCATA |
| | ATGACACCGG |
| | ATGACCGATT |
| | ATGACGCTTT |
| | ATGACTTCTA |

1 2 3 4 5 6 7 8 9



Figure A.20: Search preview results.

One important option in this section is **Transform results**. Here you can select a reference label, and the current results will be transformed using the annotated reference label in each sequence. If not all sequences are annotated with that label, the new result set will be smaller than the original. If the reference label is multiple, the new result set can be potentially bigger.

Please note that the new, transformed, set will be used for the batch operations mentioned before.

The other option, **View label**, can add new label columns to the result grid, showing the label values for each sequence, as shown in Figure A.21.

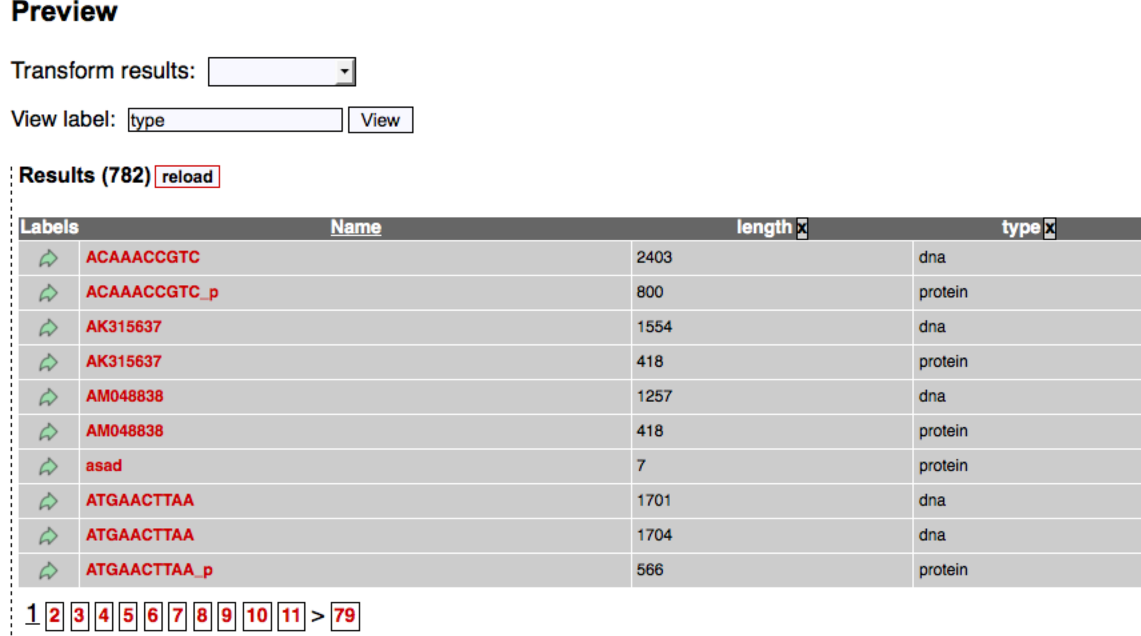



Figure A.21: Using view label.

Each new label column added can be removed by clicking the **X** on the column header.

### A.5.9 Written queries

Instead of using the query input section from the search page, you can use the search field on the top of each page to input arbitrarily complex queries.

The search field can also search for anything in the system: labels, taxonomies, ranks and sequences. When the specified search expression is not valid, the system will fallback to a wide search for the former objects, displaying a page with a grid for each entity where the query matched. So, for example, if you input 'homo sapiens', the system will display a grid with taxonomies with 'homo sapiens' in the name.

But the more interesting case is when using valid query expressions. A query expression is composed of terminal expressions and composed expressions.

A terminal expression contains a label name, an operator and, optionally, a value. The following expression is a terminal expression: *length* > *500*. The operators that do not need

a value, are the **exists** and **notexists** operators, like *species exists*. For everything else the form is *label_name operator value*. Operators and value information is shown in table A.2.

Another note: if you want to write values with spaces, wrap the value around ’ or ”.

Now, composed expressions can combine various other expressions, recursively, and use the special operators: AND, OR or NOT. AND and OR can be used like *expression1 AND expression2 AND expression3 ...*. The NOT operator can only have an expression as argument, like this: *NOT expression*.

You can also use parenthesis to group expressions. So for example, you can have things like: *<(length > 500 and name exists) or content regexp AGTG>*.

Once you input the expression, the system will analyze it and build a tree for it, redirecting you to the search page, where you can run batch operations against the resulting sequences.

## A.6 File formats

For file format information please read the section 4.3 from the main technical report.

## A.7 Administration

There are a few functionalities to do maintenance or administration tasks. These tasks can only be used when logged in as **admin**.

### A.7.1 User management

One of the key areas in administration is user management. Lets see how we can add new users, update user settings and so forth.

#### New user

To create a new user select **Administration - Users - Register** and input the user information. The username field is the name that should be used to login. You must also input the user’s password twice. Click **Do register**. If everything went well, a new user has been registered and a list of all system users is shown.

This list can be accessed through **Administration - Users - List**.

Create more users as needed. You can also logout and try the new registered users if you want.

#### User settings

Given that only administrators can register new users, normal users can modify their information, namely, the complete name, password, email and other settings.

To edit profile data, click on the user’s name, below the main menu. The new page presents user information and two buttons, as shown in Figure A.22.

User flavio

- Name: flavio
- Complete name: Flávio Cruz
- Email: flaviocruz@nowhere.com
- User type: Normal

Edit profile | Edit settings

Figure A.22: Page with user information.

Selecting **Edit profile** enables us to edit the complete name, email or password. When editing any information here, you should input your old password. If you want to change the current password fill the two text fields for that, if not, leave them empty.

The other button, **Edit settings** enables you to change other, aspect related settings, like the number of items per grid.

#### Managing other users

If you are an administrator using the **admin** account, you can edit other user profiles, through **Administration - Users - List** and then selecting the target user. When editing one user you should enter the **admin** password and not the user's current password.

For some reason, if you want to disable one user, go to the users list, select the user name and click the **Delete** option.

One very destructive feature is the **Database reset**. It can be accessed through the menu option **Administration - Reset Database** and removes all custom data from the database, which is:

- All taxonomy trees, except the NCBI tree.
- All ranks except the system defaults.
- All non-default labels.
- All sequences.
- All normal users.
- All files in the **file** table.

### A.7.2 Import / Export database

Another useful feature is the database export / import facilities. One can export the whole database as a XML file and then import it somewhere else, literally copying the source database.

What is exported?

- labels

- ranks
- taxonomy trees, except the NCBI tree
- sequences

To export the database, use the option **Administration - Export Database**.

To import a database XML file, go to **Administration - Import Database** from the main menu. There you should upload the file and, once processed, an import report is shown. The report is similar to the individual ones, but over various entities.

### A.7.3 Application customization

There are two ways of customizing the application:

- Changing the text that appears on the header: use the option **Administration - Database Description**.
- Changing the application's background: use the option **Administration - Database Background**. You should upload an JPG or PNG file.

| Label type | Operators | Values |
|---|---|---|
| URL, text and object | • equal: Equal comparison.<br>• contains: If the label contains a substring.<br>• starts: If the instance starts with.<br>• ends: Starts counterpart.<br>• regexp: Regular expression matching. | — |
| Bool | • equal: Equal comparison. | The value should be ”true” or ”false”. |
| Integer and float | • =<br>• ><br>• <<br>• >=<br>• <= | The value should be a number. |
| Position | • =<br>• ><br>• <<br>• >=<br>• <= | Before the operator you should indicate the position component to compare: ’start’ or ’length’. |
| Date | • equal: Equal comparison.<br>• before: Date is before some date.<br>• after: Date is after some date. | Values should be in the form *dd-mm-yyyy* like *03-11-2009*. |
| Taxonomy | • like: A taxonomy name to search for. | The **like** operator gets a taxonomy name and then searches all taxonomies with that name in the system and if the sequence points to any of them the query succeeds. |
| Reference | • like: A sequence name to search for. | The values and operators work just like the taxonomy labels, but applied for sequences. |

Table A.2: Operators and values in query expressions.

# Appendix B

# Tables

| Table | Fields |
| --- | --- |
| User | • id (SERIAL)<br>• name (CHAR)<br>• complete_name (VARCHAR)<br>• password (CHAR): MD5 hash of the user password.<br>• email (VARCHAR)<br>• user_type (ENUM): Can be 'admin' or 'user'.<br>• enabled (BOOL): TRUE if user is active.<br>• history_id (ID): Pointer to history table.<br>• last_access (TIMESTAMP): Timestamp of last access. |

Table B.1: User table.

| Table | Fields |
| --- | --- |
| Configuration | • user_id (ID): Pointer to user table.<br>• key (CHAR)<br>• value (VARCHAR) |

Table B.2: Configuration table.

| Table | Fields |
|---|---|
| History | • id (SERIAL)<br>• creation_user_id (ID): Pointer to user who created the object.<br>• creation (TIMESTAMP): Timestamp at creation's time.<br>• update_user_id (DI): Pointer to user who made the last object update.<br>• update (TIMESTAMP): Last update's timestamp. |

Table B.3: History table.

| Table | Fields |
|---|---|
| TaxonomyNameType | • id (SERIAL)<br>• name (VARCHAR) |

Table B.4: TaxonomyNameType table.

| Table | Fields |
|---|---|
| TaxonomyName | • id (SERIAL)<br>• name (VARCHAR)<br>• tax_id (ID): The taxonomy this name refers to.<br>• type_id (ID): Points to a TaxonomyNameType row. |

Table B.5: TaxonomyName table.

| Table | Fields |
| --- | --- |
| TaxonomyRank | • id (SERIAL)<br>• name (CHAR)<br>• history_id (ID): To store rank history information.<br>• parent_id (ID): Pointer to parent rank.<br>• is_default (BOOL): TRUE if system rank. |

Table B.6: TaxonomyRank table.

| Table | Fields |
| --- | --- |
| TaxonomyTree | • id (SERIAL)<br>• name (VARCHAR)<br>• history_id (ID): To store tree history information. |

Table B.7: TaxonomyTree table.

| Table | Fields |
| --- | --- |
| Taxonomy | • id (SERIAL)<br>• name (VARCHAR): Taxonomy's main name.<br>• parent_id (ID): Pointer to parent taxonomy in this table.<br>• rank_id (ID): Taxonomy rank.<br>• tree_id (ID): The tree where this taxonomy belongs.<br>• history_id (ID): To store taxonomy history information.<br>• import_id (ID): ID in the imported database.<br>• import_parent_id (ID): Parent ID in the imported database. |

Table B.8: Taxonomy table.

| Table | Fields |
|---|---|
| Label | • id (SERIAL)<br>• type (ENUM): Can be: integer, float, text, obj, position, ref, tax, url, bool or date.<br>• name (CHAR)<br>• comment (VARCHAR)<br>• history_id (ID): To store label history information.<br>• default (BOOL): If true, label is a system label.<br>• must_exist (BOOL): If true, each sequence must have an instance of this label.<br>• auto_on_creation (BOOL): If true and when a sequence is being added into the system, a label instance should be generated and connected to the new sequence.<br>• auto_on_modification (BOOL): If true and when a sequence content is being modified, the label instance should be edited and auto generated.<br>• code (TEXT): Code run to create a new label instance.<br>• valid_code (TEXT): Code run to valid a label instance value.<br>• deletable (BOOL): If true, the user can delete the label instances.<br>• editable (BOOL): If true, the user can manually modify the label instances.<br>• multiple (BOOL): If true, a sequence can have multiple instances of this label, only if they are distinguished by a parameter string.<br>• public (BOOL): If true, this label can be used in public searches.<br>• action_modification (TEXT): Code that is run when a sequence having label instances is modified. |

Table B.9: Label table.

| Table | Fields |
| --- | --- |
| Sequence | • id (SERIAL)<br>• content (TEXT): The sequence content.<br>• name (VARCHAR)<br>• history_id (ID): To store sequence history information. |

Table B.10: Sequence table.

| Table | Fields |
| --- | --- |
| LabelSequence | • id (SERIAL)<br>• seq_id (ID): Sequence pointer.<br>• label_id (ID): Label pointer.<br>• history_id (ID): To store instance history information.<br>• int_data (INT)<br>• text_data (VARCHAR)<br>• obj_data (ID)<br>• ref_data (ID)<br>• position_start (INT)<br>• position_length (INT)<br>• taxonomy_data (ID)<br>• url_data (VARCHAR)<br>• bool_data (BOOL)<br>• date_data (DATETIME)<br>• float_data (DOUBLE)<br>• param (TEXT): Used to distinguish between multiple label instances. |

Table B.11: LabelSequence table.

| Table | Fields |
| --- | --- |
| Event | • id (SERIAL)<br>• code (int): Event code.<br>• data (text): Event information. |

Table B.12: Event table.

| Table | Fields |
| --- | --- |
| File | • id (SERIAL)<br>• label_id (ID): Which label this file refers to.<br>• name (VARCHAR(512)): File name.<br>• count (INT): Reference count.<br>• data (LONGBLOB): File data, the file itself.<br>• type (TEXT): Optional field to indicate the file type. |

Table B.13: File table.