UNIVERSITY OF OSLO
Department of informatics

Database of Genes Involved in DNA repair

Master thesis
60 credits

Lise Henriksen

15th May 2007

Table of contents

1 Abstract

Genomes are constantly exposed to both internal and external agents causing DNA to get damaged. Lack of sufficient DNA repair might cause defects in development of the cell or even cancer. There exists several DNA repair pathways, and many genes are involved in these processes. These genes may be grouped into families using the same pathway while repairing DNA. In the human genome, there are approximately 150 known DNA repair genes.

A database containing genes involved in DNA repair in different organisms and which of these are homologs to each other would be a useful and helpful tool for scientists. While making mutants in the laboratory in order to characterize a gene, it is important to know what function a certain gene has in other organisms.

The database developed was built on a relational model were Natural language Information Analysis Method (NIAM) and Unified Modeling Language (UML) were used to make the design. The MySQL database system and PHP programming were used extensively to manipulate and retrieve information from the database.

An object role model was made on background of what information was needed about the genes and a class diagram was derived from this model. Wood et al. (2005) has provided an overview (http://www.cgal.icnet.uk/DNA_Repair_Genes.html) of human DNA repair genes and in which mechanism repair genes in the human genome take part; these genes were the first included in the database. Information from The National Centre for Biotechnology Information (NCBI), including the HomoloGene system, was used to find homologs to the genes provided in this overview and also other essential information about all these genes. Some of the information from NCBI was collected manually, but most of it was derived from files on their ftp-sites. The tables in the database were filled with all this information and some temporary tables were also made to retrieve just the information needed from these large files downloaded from the NCBI. A dynamic web-interface was made on top of the database. For this PHP was used embedded with HTML and MySQL.

The result of this work is a relational database with a dynamic interface. The user can pick which of the 18 organisms to include and either select a DNA repair system or do a free text search on a text field. If the search is on DNA repair system, all genes in the system picked

will be grouped into orthologs. If the text field is used, the system will search for this string in gene name, synonym and description of the gene. The resulting output will be grouped into homologs here as well. For both choices the output provides links to NCBI's HomoloGene, Taxonomy Browser and Entrez Gene. The Database of Genes Involved in DNA repair is available at http://dna.uio.no/lise/database/rep.php.

2 Introduction

There are approximately 150 known genes which are involved in DNA repair in the human genome. DNA is exposed to both internal and external factors which may cause damage to it. Reduced DNA repair activity causes certain types of cancer and may also be involved in aging. It is essential for an organism to have a well functioning repair system and several repair pathways are conserved in many organisms.

DNA repair is therefore an important field for research and several Norwegian research groups are involved on a high international level. On The National Hospital in Norway, scientists are involved in a European research project focusing on DNA repair, and this project is a part of that.

A database which contains information about which DNA repair genes exists in different organisms and which of these are orthologous genes will be a helpful tool for the scientists. The implementation should be done in a database system with a web-interface. It should be possible to choose which organisms to include in a search as well as search for a repair system. The search should provide links which gives easy access to other sites on the Internet with further information about each gene.

3 Background

This chapter describes the background theory needed for developing the database. Chapter 3.1 is mostly cell biology and some basic knowledge about biology is assumed.

3.1 DNA repair

Genomes suffer a huge amount of damages every day, either spontaneously or from exposure to genotoxic environmental agents (Friedberg E C et al. 2006). The primary structure of DNA gets extensively altered at physiological temperatures and pH. Examples of such are hydrolysis where purine residues are lost and transformation of cytosine to uracil. DNA also reacts with frequent byproducts from metabolism in such a way that the coding part gets damaged. These are examples of spontaneously changes to DNA. Tobacco smoke is an example of a genotoxic environmental agent that causes damage to the genome. These are just a few examples of damage a genome suffers. Genomes have several mechanisms for DNA repair and without repair systems genes would quickly be inactivated. In the human genome, deficiency in DNA repair might cause developmental effects or even cancer, depending on which type of deficiency it is (Krokan et al. 2004). There are a few different categories of repair systems which most cells possess.

3.1.1 Direct reversal of DNA damage

Mutations caused by ionization and radiation often results in nicks which are repaired by direct repair. Direct reversal of DNA damage is the most efficient and accurate mechanism by which damage to DNA can be repaired (Friedberg E C et al. 2006). This kind of repair is believed to be essentially error free due to the limited biochemical events that occurs during the repair mechanisms and they are all catalyzed by single polypeptide enzymes. When DNA is exposed to UV radiation at wavelengths around 260 nm, adjacent pyrimidines may become covalently linked by the formation of pyrimidine dimers (Weber S 2005). This kind of

damage may inhibit DNA polymerase and may result in mutations that interfere with DNA replication and transcription (Friedberg E C et al. 2006) or even cell death (Kao et al. 2005). Photolyase is a photoenzyme which cleaves the cyclobutane ring of the pyrimidine dimers. This photoreactivation brings pyrimidine dimers back to their monomeric form, thus reverses the dimerization, and is known in many bacteria and a few eukaryotes (Brown T A 2002). It is however absent in humans. Enzymes involved in photoreactivation converts cyclobutyl dimers back to monomeric form when stimulated with light from a certain wavelength.

3.1.2 Reversal of alkylating base damage

Alkylating agents exists everywhere; they are generated during metabolism and found in air, water and foods, though at low concentrations (Drabløs F et al. 2004). DNA has numerous potential nucleophilic reaction sites for the electrophilic alkylating compounds (Friedberg et al. 2006). MeCl (Methyl Chloride) is an alkylating agent which is emitted globally from biomass burning and biological synthesis. Other alkylating agents worth mentioning are: MeCl, MeBr and MeI. Alkylating agents can either have one or to reactive groups that potentially may react with DNA, thus they are monofunctional or bifunctional. The agents are both mutagenic and genotoxic thus most of the research is done in the area of DNA damage (Drabløs F et al. 2004). It is however known that alkylating agents may also damage RNA. This may contribute to cytotoxicity. Adducts from alkylating agents may be formed on O-atoms in phosphodiesters and at all O- and N-atoms in nucleobases (Krokan H E et al. 2004). In humans, alkylation at the O^6 position of guanine may lead to mutation or cell death (Begley and Samson 2004). When repaired, the methyl group of O^6-methylguanine is transferred to DNA alkyltransferase (AGT) which again inactivates AGT, thus it is called a suicide reaction. The Ada enzyme of *E. coli* is an example of such.

Oxidative demethylation

The *E. coli* protein AlkB which is a part of the Ada regulon, has shown to be an oxidative DNA demethylase that repair cytotoxic lesions in DNA (Drabløs F et al. 2004). AlkB repairs 1-meA and 3-meC in single- and double-stranded DNA (Figure 1). Eukaryotes seem to have

several AlkB homologues (hABH) while bacteria usually have only one. Eight ABHs are identified in the human genome.

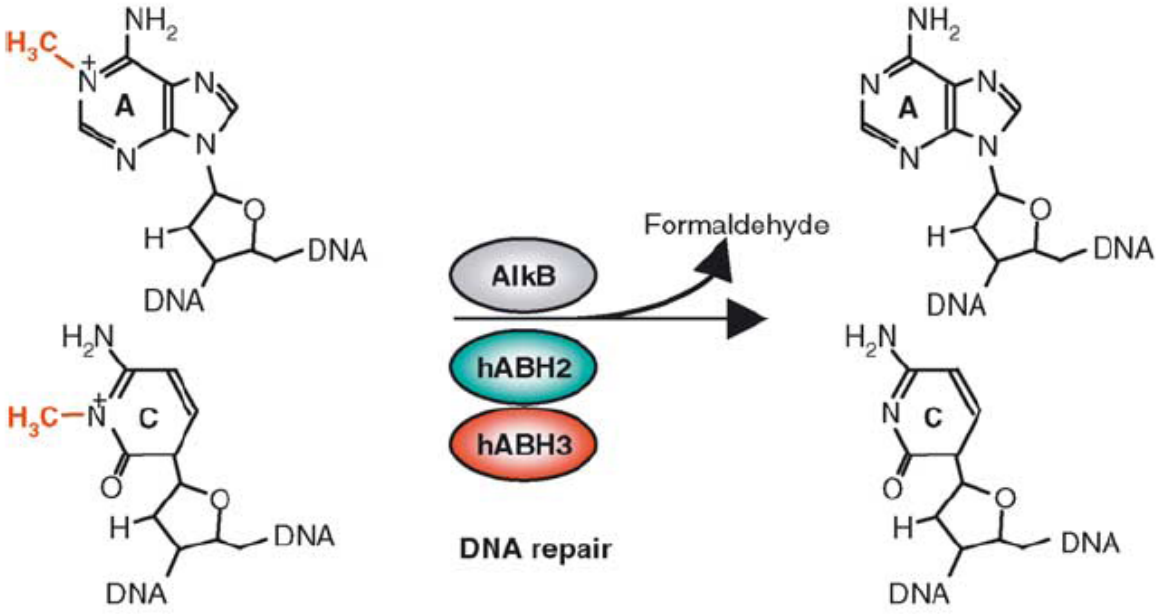

Figure1. AlkB and the two human homologs hABH2 and hABH3 in oxidative demethylation (Drabløs F et al. 2004).

3.1.3 Base excision repair

Although alklylation damage is repaired by direct repair, base excision repair (BER) is the main mechanism to repair both DNA alkylation and oxidative damage (Fortini P et al. 2003). BER is used to repair bases with minor damages. One example of such a damage is deamination of 5-methylcytosine in DNA (Friedberg et al. 2006). The result of this deamination is a T-G mispair. BER is initiated by a DNA glycosylase which specificity determines at which range the nucleotides can be repaired (Brown T A 2002). The

glycosylase cleaves the N-glycosidic bond between the damaged base and the sugar component of the nucleotide. After the cleavage, a single-strand break is generated and the BER mechanism may go through two different pathways which again are divided into several steps (Fortini P et al. 2003). The two different pathways are named the short-patch BER and the long-patch BER (Figure 2). The difference between these pathways is the gap-size and the enzymes involved. In summary the damaged bases are removed and new DNA is synthesised.

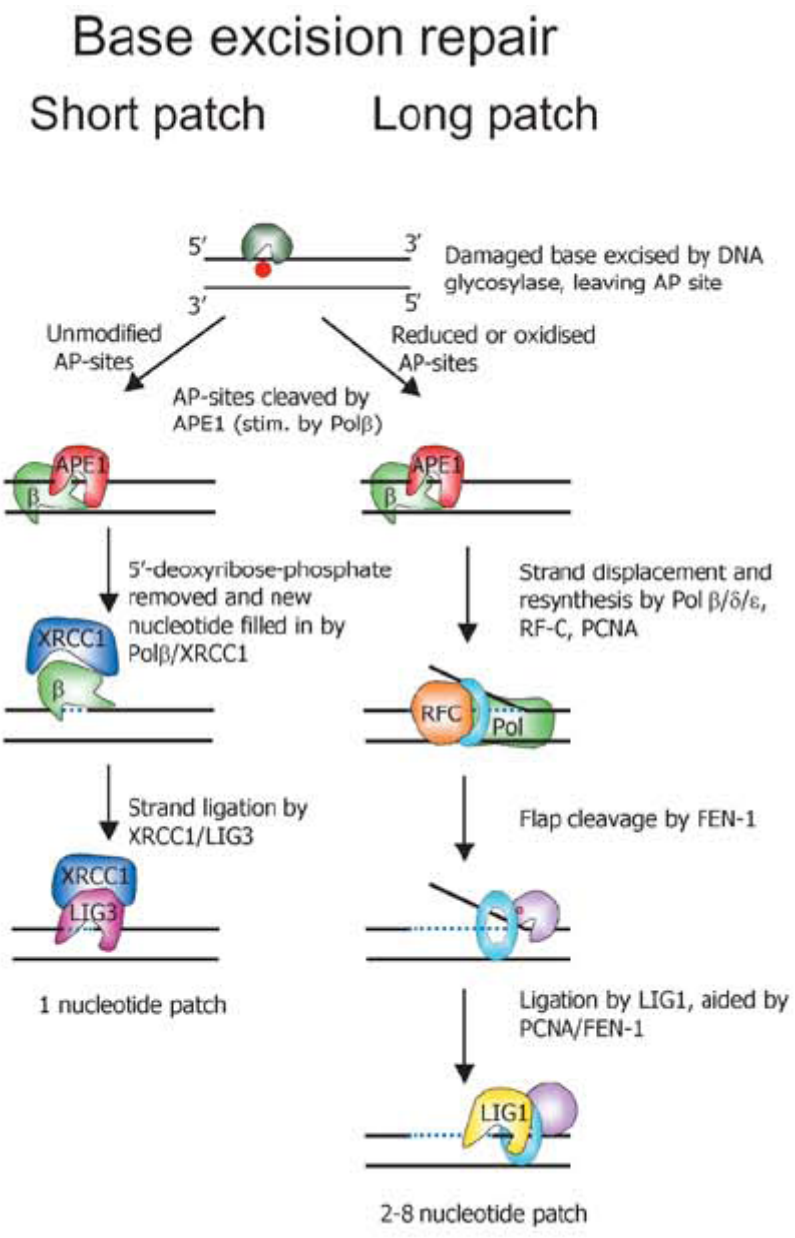

Figure 2. Short-patch BER and long-patch BER (Krokan H E et al. 2004).

3.1.4 Nucleotide excision repair

Nucleotide excision repair (NER) deals with more extended forms of damage. Examples of such are intrastrand crosslinks and bases modified by large chemical groups, typical bulky adducts (Brown T A 2002). The biochemichal features of NER in prokaryotes and eukaryotes are distinct; the number of proteins involved differs (Friedberg et al. 2006). Despite this difference, NER has common steps in all known organisms: damage recognition, cutting and releasing of the oligomer and resynthesis and ligation (Krokan H E et al. 2004). A segment of singlestranded DNA is excited and replaced by new DNA. The process includes a dark repair process which also corrects cyclobutyl dimers (Brown T A 2002). In prokaryotes the length of the removed stretch is about 12-13 nucleotides long while in eukaryotes it is about 24-32 nucleotides long (Reardon J T et al. 2005). Two subpathways have been identified for NER: global genome repair (GGR) and transcription-coupled repair (TCR) (Krokan H E et al. 2004). Figure 3 illustrates these pathways. Even if NER in prokaryotes and eukaryotes have the same basic steps, the factors involved are not evolutionary related and they have no sequence homology (Reardon J T et al. 2005).

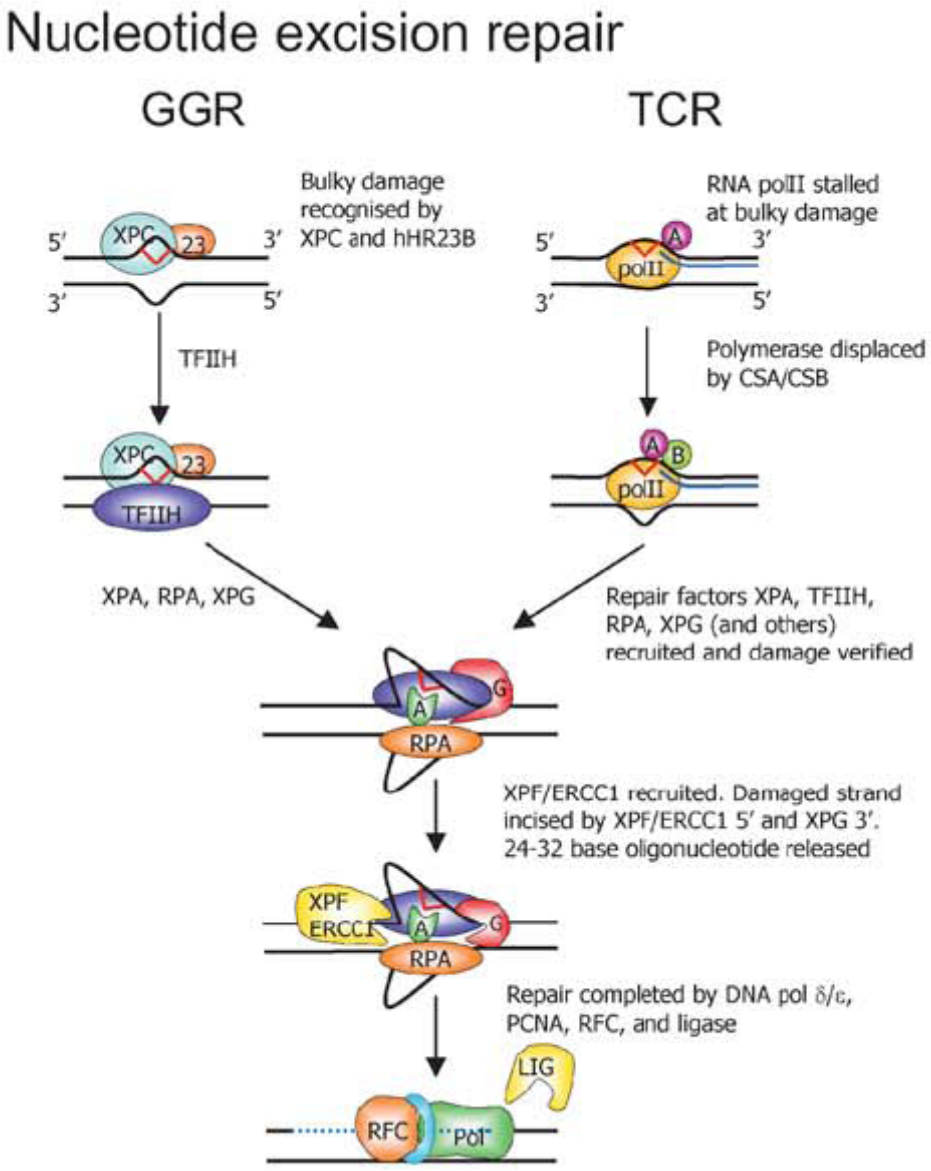

Figure 3. Illustration of the two subpathways of NER (Krokan H E et al. 2004).

3.1.5 Mismatch repair

Mismatched basepairs in DNA can arise by DNA replication, formation of heteroduplex between two homologous DNA molecules or by deamination of 5-methylcytosine (Friedberg et al. 2006).This means that in MMR, absence of base pairing must be detected instead of defective bases. Event though the mismatched base is intact, the machinery must be able to detect the mismatch from the correct base. Cells that lack the MMR process are mutators; they have a much higher mutation rate than normal. In humans this may result in cancer. The

proteins from the MutS and MutL families are used for all organisms to carry out the repair mechansim. In *E.coli* MutS initiates the process by binding to mismatched DNA (Kunkel T A et al. 2005). MutS interacts with MutL and activates MutH which cleaves the newly synthesized strand and DNA helicase detaches the resulting nick (Brown T A 2002) . The gap is filled in by DNA polymerase and sealed by DNA ligase and the replication error is corrected (Kunkel T A 2005). Eukaryotic MMR is much like MMR in *E. coli* but some proteins may differ depending on witch kind of mismatch is present. Figure 4 illustrates the process.

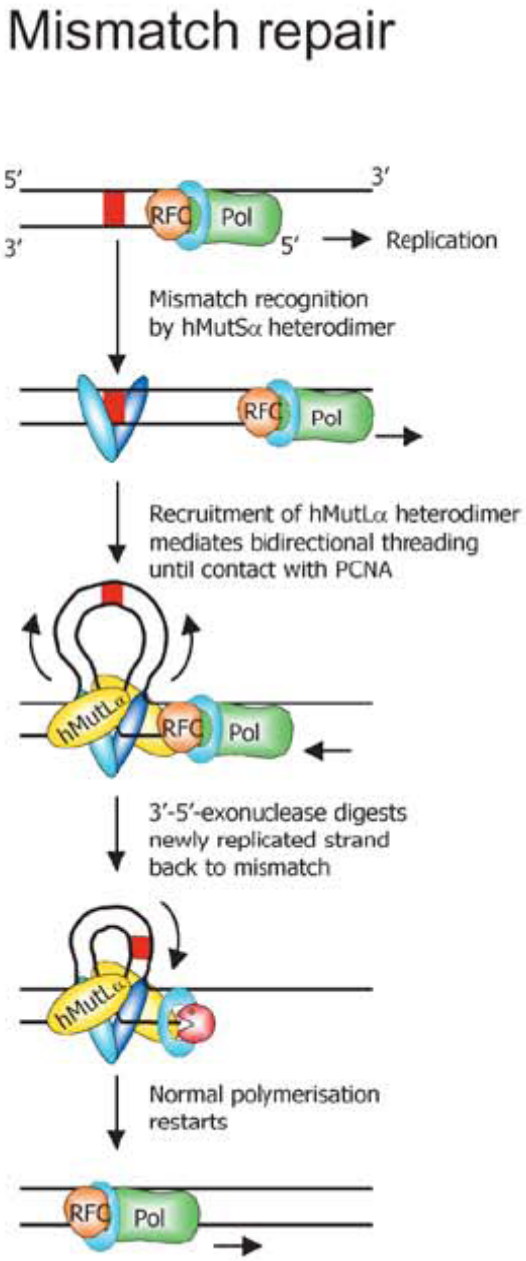

Figure 4. Illustration of the mismatch repair process (Krokan H E et al. 2004).

3.1.6 Recombination repair

Double-stranded breaks (DBS) are generated by ionizing radiation and some chemical mutagens and they are also made by the cell (Brown T A 2002). Mammalian cells repairs DBS through damage signalling, cell cycle arrest, homologous recombination (HL) and non-homologous recombination (NHEJ) (Krokan H E et al. 2004). The pathway seems to be dependent of cell cycle. Non-homologous recombination is not dependent on sequence homology, and in the G_0/G_1phase of the cell cycle, it is believed to be the dominant mechanism (Krokan H E et al. 2004). NHEJ is initiated by binding of the protein Ku, which is a part of a multi-component protein complex, to either side of the break and a protein kinase (DNA-PK_{cs}) (Brown T A 2002). DNA-PK_{cs} changes conformation upon binding and this activates several other repair proteins (Krokan H E et al. 2004). The DNA ends are trimmed and gaps are filled. Finally Ku recruits an Xrcc4-DNA-ligase complex which finishes up. Homologous recombination is not fully understood, but it is believed that ATM kinase senses the DBS (krokan H E et al. 2004). This initiates activation of repair proteins involved in the G_1, S and the G_2/M checkpoints and histone proteins. This activation is followed by involvement from HR factors. The mechanism is illustrated in figure 5.

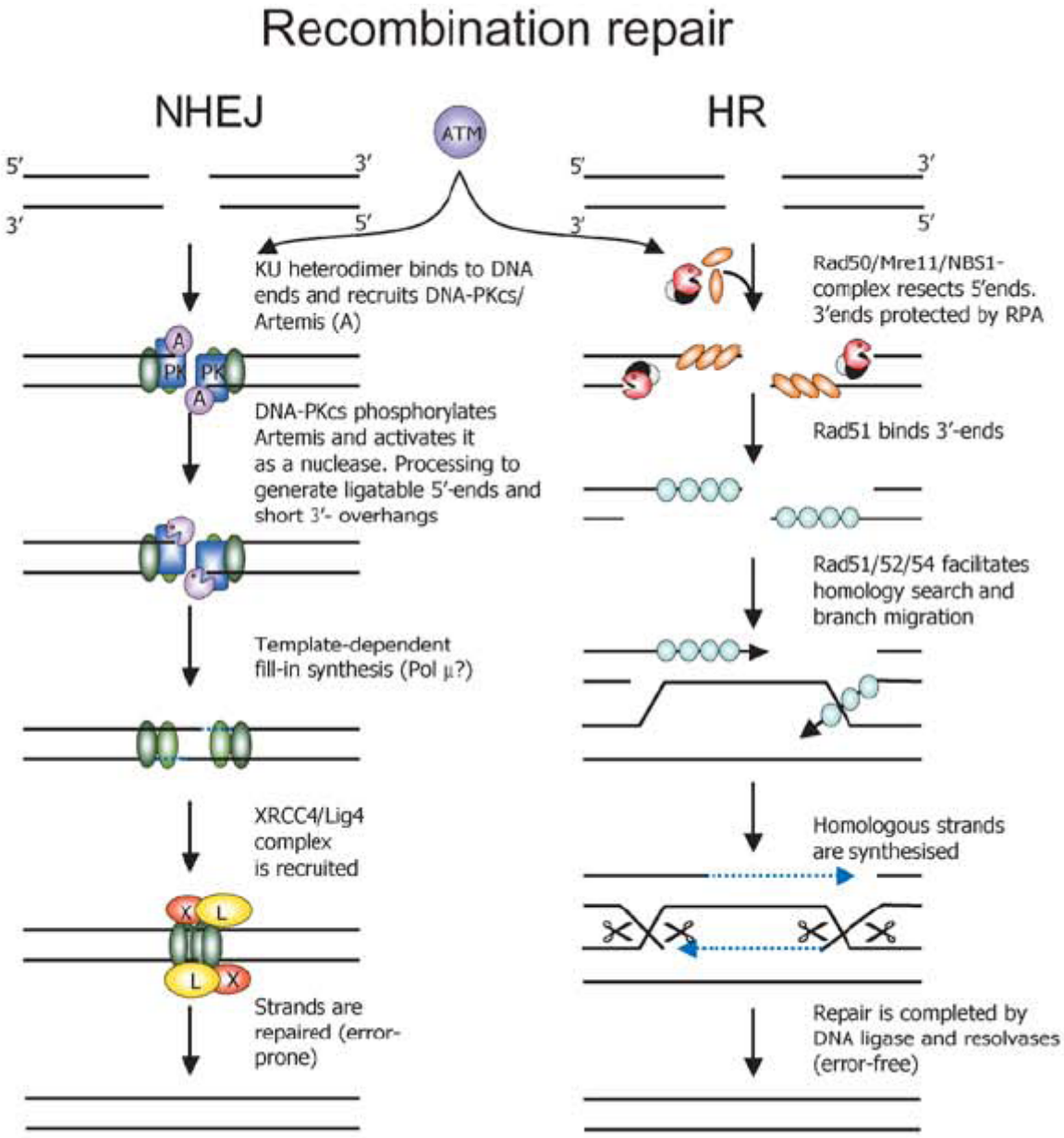

Figure 5. Illustration of Recombination repair; NHEJ and HR (Krokan H E et al. 2004)

3.2 Human DNA repair genes

The list of Human DNA Repair Genes include genes with gene products which functions are to recognize and repair damaged DNA in addition to genes having strong sequence homology to repair genes in other organisms (Wood et al. 2005).

The genes are grouped into the following groups;

- Base excision repair (BER)
- Direct reversal of damage
- Repair of DNA-protein crosslinks
- Mismatch excision repair (MMR)
- Nucleotide excision repair (NER)
- Homologous recombination
- Non-homologous end-joining
- Modulation of nucleotide pools
- DNA polymerases (catalytic subunits)
- Editing and processing nucleases
- Rad6 pathway
- Chromatin structure
- Genes defective in diseases associated with sensitivity to DNA damaging agents
- Other identified genes with suspect to DNA repair function

Several of these functions are described in the previous sections.

3.3 Orthologs

Genes with a common ancestor are homologs (Eidhammer et al. 2004). Homologs might be orthologs or paralogs. If they are orthologs, they are different due to evolution of species. Paralogs on the other hand, are different due to gene duplication within species. Orthologous genes normally have the same function in different species.

In the database, we are primarily interested in identifying orthologs of the known DNA repair genes, but paralogs are also of interest.

3.4 Resources from The National Centre for Biotechnology Information (NCBI)

NCBI was created in 1988 to develop information system for molecular biology (Wheeler D L et al. 2006). NCBI provides both data retrieval –and computational systems for biological data. Some of the resources from NCBI are used throughout this work.

3.4.1 Taxonomy database

The Taxonomy database indexes ~205000 named organisms and include several thousands new organisms a month. The taxonomy browser gives the taxonomic position as well as displaying several links to other services given by NCBI. It also provides an identification number corresponding to each organism.

The Taxonomy Database and Taxonomy ftp is available at http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/ and ftp://ftp.ncbi.nih.gov/pub/taxonomy/ respectively.

3.4.2 HomoloGene

HomoloGene atomatically detects homologs among annotated genes from 18 eukaryotes including *Homo sapiens, Pan troglodytes, Mus musculus, Rattus norvegicus, Drosophila melanogaster, Anopheles gambiae, Caenorhabditis elegans, Saccharomyces pombe, Sacchromyces cerevisiae, Eretmothecium gossypii, Neurospora crassa, Magnaporthe grisea, Arabidopsis thaliana* and *Oryza sativa* (Wheeler D L et al. 2006).

HomoloGene uses DNA similarity between close related species and measures protein similarity between more distant organisms to find orthologs. The program blastp is used to do sequence search and a matching procedure guided by the taxonomic tree makes sure organisms closely related to each other are compared first.

Homology and phenotype information from HomoloGene is based on Online Mendelian Inheritance in Man (OMIM), Mouse Genome Informatics (MGI), Zebrafish Information

Network (ZFIN), Saccharomyces Genome Database (SGD), Clusters of Orthologous Groups (COG) and FlyBase. The entries include both paralogs and orthologs.

HomoloGene is located at http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=homologene and the corresponding ftp-site is available at ftp://ftp.ncbi.nih.gov/pub/HomoloGene/.

3.4.3 Entrez Gene

Entrez Gene provides information about genes and links to other gene related resources at NCBI (Wheeler D L et al. 2006). Examples of information provided is: chromosomal location, sequences, names and links to other resources at NCBI (Maglott D et al. 2005). Genes included in Entrez Gene are typically genes from completely sequenced genomes or genes that are under investigation in scientific research. A unique identifier, GeneID, of type integer which is specific for species is provided for all genes in the database. GeneID will not change for a gene even if some of the other records are changed.

Entrez Gene is available at http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Gene and the ftp-site is located at ftp://ftp.ncbi.nih.gov/gene/.

3.5 Other resources

3.5.1 Inparanoid

Inparanoid is a eukaryotic ortholog database with parwise ortholog groups (O'Brien et al. 2005). 17 eukaryotes are included in the database; *Anopheles gambiae, Caenorhabditis briggsae, Caenorhabditis elegans, Drosophila melanogaster, Danio rerio, Takifugu rubripes, Gallus gallus, Homo sapiens, Mus musculus, Pan troglodytes, Rattus norvegicus, Oryza sativa, Plasmodium falciparum, Arabidopsis thaliana, Escherichia coli, Saccharomyces cerevisiae and Schizosaccharomyces pombe*. Datasets from these genomes are from Ensembl (see next section) and at the time being, they were the only organisms completely sequenced.

The reason for choosing Ensembl as a source is due to their quick release of whole genomes as well as providing reliable data.

When searching for orthologs, first a pairwise comparison with the program Blast is done; all species are compared with all. The result from Blast is then run trough Inparanoid where the orthologous clustering is done. Inparanoid takes two datasets of sequences as input, from two different species (Remm et al. 2001). The datasets must be in separate files and contain complete sets of protein sequences from the two species. The orthologs are then detected based on pairwise similarity scores. Best scoring pairs serves as a central point to which additional orthologs might be clustered. A cut-off value decides were to stop adding new genes to each cluster.

To identify genes and proteins, Inparanoid uses Ensembl and UniProt identifiers. A gene may have more than one transcript, thus different identifiers. Inparanoid uses the longest transcript only. This is explained by the competitive nature of Inparanoid which could cause a short and a long transcript of the same gene to end up in different clusters because they exist in more than one organism.
The Inparanoid browser is located at: http://inparanoid.sbc.su.se/.

3.5.2 Ensembl

Ensembl is a bioinformatic project and it is available at http://www.ensembl.org (Birney et al. 2004) as a browser and it is also possible to download software for handling large genomes from this site. Some of the features Ensembl provides are: organizing information about large genomes, automatic annotating of genomes and finding orthologous relationships between genes. Ensembl provides much of the same features as NCBI and the project also collaborate with NCBI.

3.6 Data Model

3.6.1 Relational Data Model

Data are collected and structured in a database. In a relational database, data are structured in tables which are also called relations (Skagestein 2002). Figure 6 illustrates a relation.

Each column in the table has a header which serves as an attribute for the column (Garcia et al. 2002). A column has a domain and all values in a column must belong to this domain (See figure 6 for illustration).

Values are of an elementary type which can not be broken down to smaller components; they are atomic and typically of type integer or string. Lists and sets are examples of values that might be broken down to smaller components and thus are not legal values for an attribute.

Each line in the table is a tuple (except from the header-line) and all tuples in a table must have an equal number of values. Thus a tuple is a list of values.

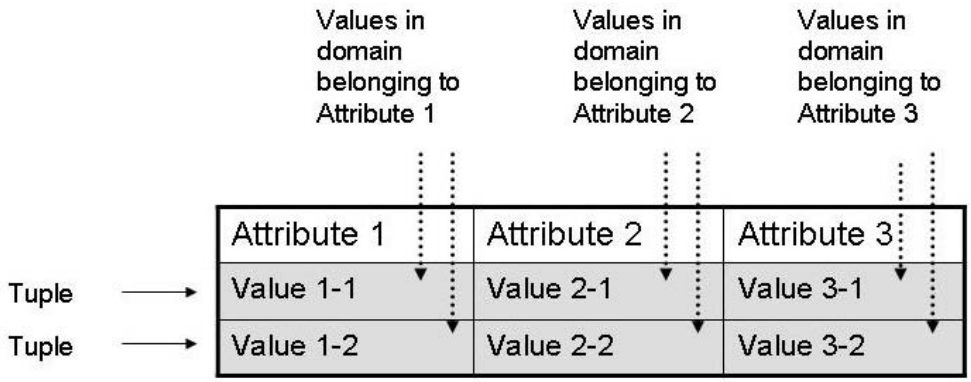

Figure 6. An illustration of a table in a relational database. Read text for details.

All relations in the database have a primary key which consist of one or more attributes. When a primary key is chosen, all values in the column of the primary key must be different for all lines. If, in figure 6, Attribute 1 and Attribute 2 together where to serve as a primary key, Value 1-1 and Value 2-1 must be different than Value 1-2 and Value 2-2 when

concatenating them. A primary key is normally marked with a horizontal line above the attribute in the relation.

3.6.2 Object Role Model

A verbal illustration of relations is through an Object Role Model (ORM). The technique used to describe this modelling is called Natural language Information Analysis Method (NIAM). NIAM describes relationships between entities, and the use of language in the model makes it easy to understand. The relationships may be converted into tables. In summary the model describes what kind of data the system should store (Skagestein 2002).

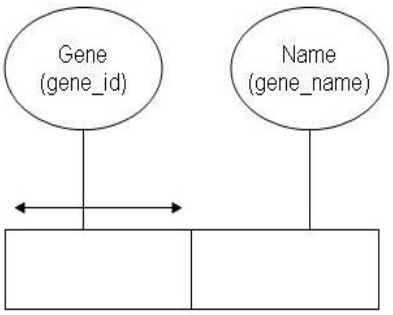

Figure 7. An ORM with Gene and Name as object types. Their reference type is in parenthesis. The double arrow is a uniqueness constraint.

An object may be an entity or an attribute and is illustrated by an ellipse with name and reference type in parenthesis (Halpin 1996). The object should be connected to at least one role (Skagestein 2002). The role is the part played by the object and it has shape as a rectangle (Halpin 1996). If a role is connected to more than one entity, the roles must have different names. A double-arrow in an ORM shows a uniqueness-constraint; from figure 7 one can read that the attribute gene_id must be unique. An ORM can be read in any direction and the model can be read out as plain words. This way it is easy to understand for everyone involved in the developing phase even if they are not computer scientists.

3.6.3 UML

Unified Modeling Language (UML) presents the roles grouped in ORM, in classes. This method gives a better visualisation of the groups (Skagesten 2002). Relations between the classes are written on the edges between the classes. All classes with one-to-one relations are usually grouped in the same tables in the database.

3.7 PHP/WEB

PHP stands for Hypertext Preprocessor and is a scripting language which is suited for Web-development (Williams H E et al. 2005). PHP can easily be combined with HTML in a web application as seen in example code in figure 8.

```
<html>
<head>
<title>PHP combined with HTML</title>
</head>
<body>
<?php
echo “PHP and HTML”;
?>
</body>
<html>
```

Figure 8. Example of how to combine PHP and HTML

HTML generates static HTML-pages, but while combining it with PHP it is possible to dynamically generate web-pages and even access a database. While HTML is used for formatting, PHP is used for making pages that can change due to user input or accessing a database.

When a web page with PHP code is executed by the web server, the server sends the result back to the web page as a HTML document after execution.

PHP is written in blocks and combined with HTML. The blocks start with “<?php” and ends with “?>”. Within the blocks PHP statements comes as series. The statements all end with a semicolon. When the PHP code is executed on the web server, the code is replaced by the output in the same order as the blocks.

PHP works excellent together with MySQL while accessing a database to retrieve information for output in a web page.

3.8 MySQL

MySQL is an open source relational database management system (RDBMS) which is developed by the private company MySQL AB (http://www.mysql.com). MySQL is part of an open source application software stack called LAMP. LAMP is an abbreviation of Linux, Apache, MySQL and PHP/Perl/Python and they are all obviously open source products.

MySQL uses SQL to manipulate/retrieve information from the database and can interact with PHP, Perl and Python (Giacomo 2005). Thus for a web-application MySQL is an alternative.

Tables are made with a CREATE TABLE statement; names of all attributes and their types (e.g. integer, varchar etc.) and lengths are stated as well as a primary key. There is one such statement for each table in the database.

After making tables they must be filled with data. This can be done by using INSERT or LOAD DATA INFILE on a file which is already in the right format for the table.

The database can then be manipulated/queried using SQL. If integrated with a script language, this may all be done from the script running the web interface and the result of a query is shown in this interface. Alternatively all operations must be done from the command line.

4 Design and implementation

This chapter describes how the design of the database was made and highlights some of the implementation done.

4.1 Description of the database

There already exists a database of DNA repair genes named repairGenes which is available at http://www.repairgenes.org. The repairGenes system was developed by the Bioinformatics group at the Centre of Molecular Biology and Neuroscience (http://www.cmbn.no) at the University of Oslo in 2002. This system processes selected sequences from the protein database SWISSPROT, followed by static generation of HTML-pages. The classification of genes is based on the Gene Ontology Annotation (GOA). GOA is a bioinformatics project run by the European Bioinformatics Institute (EMBL-EBI, http://www.ebi.ac.uk/) and they aim to classify genes according to the systems specified in Gene Ontology (GO).

The new system was planned to be a relational database with dynamically generated HTML-pages. In this system, the user should be able to make searches on either gene name or description of DNA repair genes and choose one or more organisms. The system should contain some information about each gene, but mainly hold links to more information about the specific gene on the Internet.

4.2 Structure of the database

4.2.1 Object Role Model

Before making any decisions about which tables the database should have, the real world objects of interest were identified together with their relations to each other.

The very first thing to establish was how to identify a gene in a unique way. Since NCBI is the main source for collecting data, it was decided to use their *GeneID* as the unique identifier for all genes. Next was the question of what information was needed about each gene:

Obviously it was important to have the name of a gene, but some genes have more than one name; one official name and several synonym names while some have an official name only. Since the official name and the synonym names are not the same, it was decided that they are two separate objects; name and synonym.

The name of the organism from which the gene belongs to naturally became an object as well as the description of a gene and its chromosomal location.

Since the purpose of the database was to group othologous genes, homolog was added as an object.

DNA repair genes are, as mentioned earlier, grouped into repair systems and subclasses of repair systems. Both system and the subclass were added to the list of objects and they were named system name and family name respectively.

Ensembl id was considered as an object since it was thought that the database could be somehow integrated with a database developed by Jarle Breivik et al. at The Medical Faculty, University of Oslo, also focusing on DNA repair genes. Genes in this project were identified by an identifier from Ensembl. It was decided not to integrate the two databases at this point, both because it would be very time-consuming but also because there was not found a way to make sure that the transcript for a gene from Ensembl was the correct gene referred to in NCBI, due to the different identifiers.

Links to other resources on the Internet was a possible object. It was however not included because it was thought that inks would be integrated in the interface through the output of queries.

Since an organism has a taxonomic id, one more object came on the list.

For simplicity reasons it was decided to enumerate both family name and system name such that they both got a unique identifier, so system id and family id were added to the list.

The objects finally became:

- Gene
- Name
- Synonym
- Organism
- Taxonomy Id
- Family name
- Family id
- Description
- Chromosome location
- System name
- System id
- Homolog.

The drawing of an ORM was started with objects and their representation in parenthesis. The representations came naturally after the objects were decided and they will all be explained in the section “Attributes and keys”. Finally relationships between objects had to be established and were added to the model.

The resulting model is illustrated in figure 9. There are two “is-a”-relations in the model, drawn as coloured arrows; a synonym “is-a” name and a homolog “is-a” gene. Thus synonym is a subtype of name and homolog is a subtype of gene. Roles are drawn as rectangles with their verbal meaning on top of the rectangle. An example from figure 9 of a role is every gene “has” a chromosome location. The coloured lines in figure 9 shows two possible routes through the model. The coloured text below the model corresponds to the route.

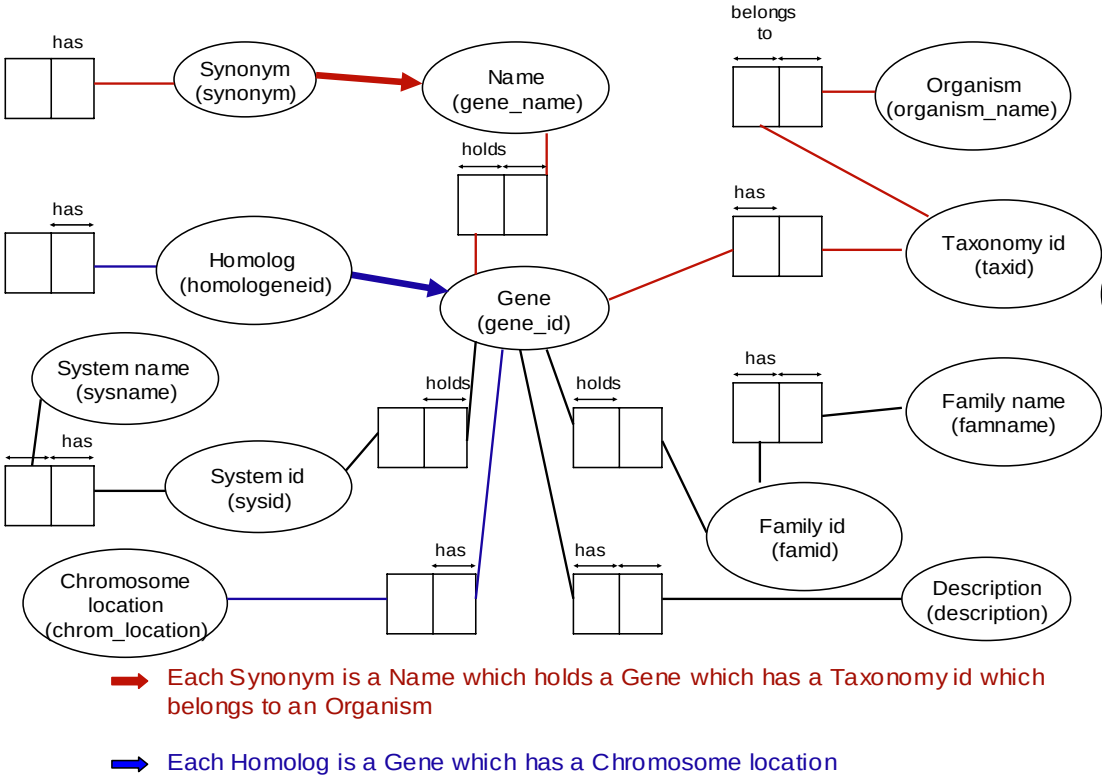

Figure 9. An Object Role Model of the database. Coloured lines illustrates two possible routes through the model. Text below the figure corresponds to the route with the same colour.

4.2.2 Tables and UML diagram

The Object Role Model in figure 9 can also be described as an UML-diagram. Such a diagram was made to help finding the tables in the database. Figure 10 gives an illustration of the resulting diagram. The only attributes in the classes are the representation of the objects as seen in ORM. The numbering on the edges gives the one-to-one and many-to-many relations. In figure 10 this can be read as e.g. “a gene has only one name and a name has only one gene” and “a gene has only one Taxonomy id while a Taxonomy id has many genes”.

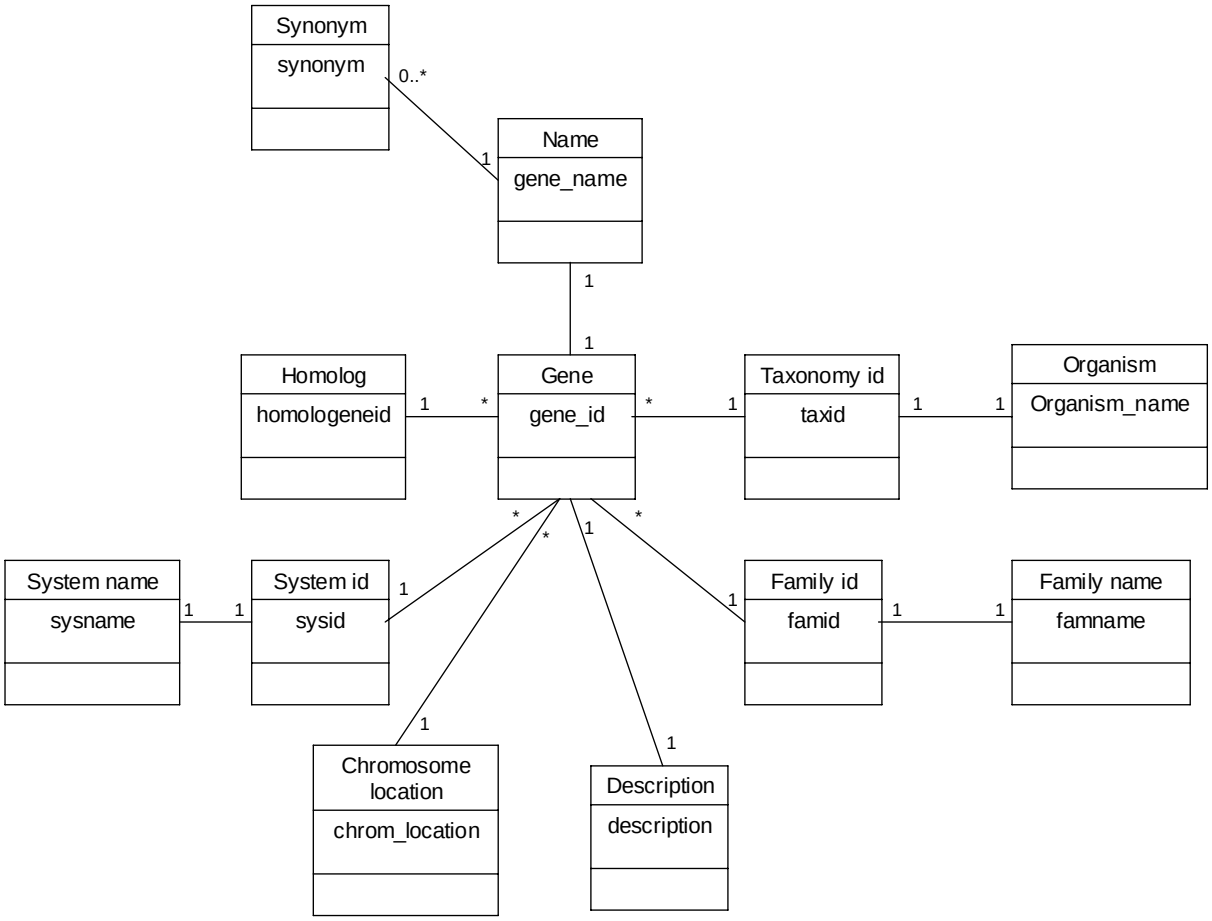

Figure 10. UML diagram directly modelled from ORM in figure 9. Read text for details.

All classes with one-to-one-relations where grouped to one relation, this to make sure no elements are repeated in a relation. At first this seemed like a good choice of relations. It was however decided to group Homolog and Chromosome location and the relation Gene (see figure 11), due to the fact that it did not seem to be useful to have these relations standing alone.

After the grouping process above, the database ended up with five relations (tables) which together give all the information about each gene. The tables were named gene, synonym, taxonomy, famname, and sysname. The table gene holds most of the information about a gene and is linked to all the other tables. The taxonomy table holds the names of all the organisms represented in the database and also the taxonomy id corresponding to each organism. The last two tables, famname and sysname, holds the names of the DNA repair systems which the genes are grouped into. Figure 11 shows a class diagram of the database and the relations

between the tables. The numbering above the four bottom tables gives “one-to-many” and “none-to-many” relations as in figure 10. All attributes will be described in the next section.

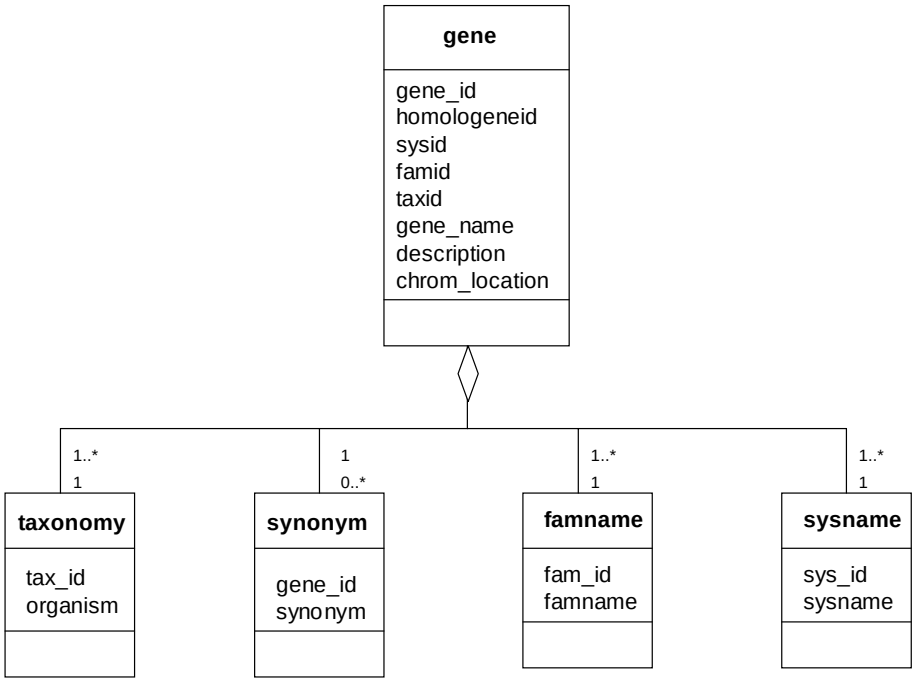

Figure 11. Class diagram of the tables in the database. Table-names are in bold. Attributes in the middle of the table.

4.2.3 Attributes and keys

All attributes and keys will be described in this section.

gene_id:

gene_id was extracted from NCBIs ftp-site where it is named GeneID. The value in GeneID is unique for all genes. The attribute was therefore a natural choice of primary key for gene as well as a foreign key. gene_id was also decided to be an attribute in synonym, but in this table

got function as a primary key only together with the attribute synonym. gene_id was chosen to be of type integer, like in NCBI, and NOT NULL.

homologeneid:

The attribute homologeneid was decided to be the unique identification for the orthologous group a gene belongs to. The value was extracted from NCBIs ftp-site and is there named HID. There are normally several genes in one group, and homologeneid will be the same for several genes. homologeneid is an integer, like in NCBI. It was first decided that homologeneid should be NOT NULL, but it was discovered that a few genes are not grouped into orthologous groups, so the value was changed to NULL.

gene_name:

A gene has an official name. The attribute gene_name was decided to hold the official name of a gene. It is of type varchar and NOT NULL.

descripton:

Genes have a text description which gives some more information than the name only. The attribute description is a varchar and NOT NULL.

chrom_location:

The attribute chrom_location gives the location of the gene on a chromosome; which chromosome, which arm and the location on the arm. The values were extracted from NCBIs ftp-site. chrom_location is a varchar and NOT NULL.

synonym:

Some genes may have more names than the official name, named synonyms. Synonyms were decided to be kept in the attribute synonym. Since some genes have more than one alias, synonym does not have a unique corresponding gene_id. It is therefore sufficient that synonym and gene_id together have the role of primary key. synonym is varchar and NOT NULL.

tax_id/taxid:

All organisms have a taxonomic id, tax_id. This is a unique identifier for all organisms and it was extracted from NCBIs ftp-site. In this database, the value corresponds to the attributes tax_id and taxid. In the taxonomy table it has function as both a primary key and a foreign key. In the table gene it is a foreign key only. tax_id/taxid is integer and NOT NULL.

organism:

The name of an organism is in the attribute organism. A name corresponds to only one tax_id. organism is a varchar and NOT NULL.

sysname/famname:

The classification of the DNA repair systems is based on Human DNA repair genes (Wood et al. 2005). The attribute sysname is the name of the repair systems and famname is an subclass of the repair systems. Both of the attributes are varchar and NOT NULL.

sys_id/fam_id

sys_id and fam_id are unique identifiers and primary keys in their tables. The attributes correspond to the sysid and famid in gene. They are manually made integers which are NOT NULL.

4.3 Data

4.3.1 Orthologous genes from Inparanoid

The program Inparanoid, which is previously described, classifies orthologous groups of genes. The plan was to download pairwise Inparanoid analysis for all the organisms existing in HomoloGene (see chapter 3). After retrieving all the results, the idea was to use Entrez Gene (see chapter 3) to gather essential information about each gene. Since a gene might have several transcripts, it was crucial to have the right identification for the gene. As mentioned earlier, Inparanoid uses Ensembl and UniProt identifiers while Entrez Gene has a different unique identifier, *GeneID*.

Ensembl provides the Biomart system which is able to “transfer” the Ensembl gene identificators into the right Entrez Gene identificators for several organisms. This was thought to be an excellent way for overcoming the problem with different identifiers, thus making sure information about the right gene was collected from Entrez Gene. While investigating Ensembls’ Biomart, it was however discovered that not all organisms were represented here. Several of the eukaryotes in HomoloGene where lacking; *Saccharomyces pombe, Eretmothecium gossypii, Neurospora crassa, Magnaporthe grisea, Arabidopsis thaliana* and *Oryza sativa* id’s. It did not seem to be possible to find the right Entrez identificators for the genes from these organisms in Inparanoid, so it was decided that Inparanoid was not the way to go while using NCBI as a source for information about the genes.

4.3.2 Recording data from Human DNA Repair Genes

Repair genes from the human genome are listed and grouped in functions as described in background chapter (Wood et al. 2005). This list was used as a basis for finding the corresponding orthologous genes in other organisms. Genes from the list were manually recorded, as well as the fourteen functions which the genes were grouped into.

These genes are later referred to as initial genes. The repair systems listed in background chapter corresponds to the repair systems in Database of DNA repair genes and they were numbered from 1 to 15. When all the necessary information from Human DNA Repair Genes was collected, more information about each gene was recorded from Entrez Gene as well as the homolog gene id from Entrez HomoloGene as described in next section, “Step one”.

4.3.3 Recording data from Entrez HomoloGene and Entrez Gene

The information from Entrez HomoloGene and Entrez Gene was recorded in two steps.

Step one:

This step describes how information about the initial genes was recorded from Entrez gene and Entrez HomoloGene.

Entrez gene was used to retrieve *Gene Id*, *official name*, *other aliases*, *description* and *chromosome location* for all initial genes. The information corresponds to the attributes *gene_id*, *gene_name*, *synonym*, *description* and *chrom_location*, respectively, in the tables *gene* and *synonym*. At this point this was done manually.

The homolog identification (homolog gene id) was retrieved from Entrez HomoloGene for all initial genes. This step was also recorded manually. A few genes did not have a homolog gene id, thus the value was NULL for these genes. The homolog gene id corresponds to the attribute *homologeneid* in the table *gene*.

```
284131:15562:10:7:9606:FLJ35220:hypothetical protein FLJ35220:17q25.3
7319:68308:11:7:9606:UBE2A:ubiquitin-conjugating enzyme E2A:Xq24-q25
7320:37761:11:7:9606:UBE2B:ubiquitin-conjugating enzyme E2B:5q23-q31
56852:48572:11:7:9606:RAD18:RAD18 homolog:3p25-p24
7336:55739:11:7:9606:UBE2V2:ubiquitin-conjugating enzyme E2 variant 2:8q11.21
7334:2512:11:7:9606:UBE2N:ubiquitin-conjugating enzyme E2N:12q22
3014:68227:12:7:9606:H2AFX:H2A histone family, member X:11q23.2-q23.3
10036:4003:12:7:9606:CHAF1A:chromatin assembly factor 1, subunit A:19p13.3
641:47902:13:7:9606:BLM:Bloom syndrome:15q26.1
7486:6659:13:7:9606:WRN:Werner syndrome:8p12-p11.2
9401:3144:13:7:9606:RECQL4:RecQ protein-like 4:8q24.3
472:30952:13:7:9606:ATM:ataxia telangiectasia mutated:11q22-q23
2175:108:13:7:9606:FANCA:Fanconi anemia, complementation group A:16q24.3
2187:51880:13:7:9606:FANCB:Fanconi anemia, complementation group B:Xp22.2
2176:109:13:7:9606:FANCC:Fanconi anemia, complementation group C:9q22.3
2177:13212:13:7:9606:FANCD2:Fanconi anemia, complementation group D2:3p26
2178:11066:13:7:9606:FANCE:Fanconi anemia, complementation group E:6p22-p21
2188:75185:13:7:9606:FANCF:Fanconi anemia, complementation group F:11p15
2189:3402:13:7:9606:FANCG:Fanconi anemia, complementation group G:9p13
```

Figure 12. The format of the file “gene.txt”

All the information needed was written to the file “gene.txt”. A small fraction of this file is shown in figure 12. The file “gene.txt” is colon-delimited and has one gene for each line. The columns in the file are: *gene_id*, *homologeneid*, *sysid*, *famid*, *taxid*, *gene_name*, *description* and *chrom_location*.

The table *gene* was made as shown in figure 15 (Appendix 9.2) and a script which would take care of loading the file “gene.txt” into the table *gen*e was coded as shown in figure 16 (Appendix 9.3).

After retrieving information about all initial genes, it was possible to record all orthologs to these genes. This process is described as step two below.

Step two:

In the first step, *homologeneid* was recorded for all initial genes, except those genes with no known homologs, according to NCBI. Step two describes how taxonomy ids' to all organisms were retrieved as well as how orthologs to intital genes were found and recorded.
On September 22 2006, the file "taxdmp.zip" was downloaded from ftp://ftp.ncbi.nih.gov/pub/taxonomy/ (Entrez Taxonomy). When unpacked the files "citiations.dmp", "delnodes.dmp", "division.dmp", "gc.prt", "gencode.dmp", "merged.dmp", "names.dmp", "nodes.dmp" and "readme.txt" were revealed. The file "names.dmp" contained *tax_id*, *name_tx*t, *unique name*, and *name class* and contained more than 400000 organisms (see ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump_readme.txt for further explanation about the files). To retrieve taxonomy id and organism name from this file only, the table *tax_tmp* was made (see Appendix 9.2) and the file "names.dmp" was loaded into this table with a script much like the one shown in Appendix 9.3. A SELECT statement was run against the file "tax_tmp".

The file "gene_info.gz" was downloaded from ftp://ftp.ncbi.nih.gov/gene/DATA (Entrez Gene) and unzipped on October 3. 2006. The unzipped file was named "gene_info". The file "gene_info" was a tab-delimited file with one line for each gene. Each line contained the following information: *tax_id*, *GeneID*, *Symbol*, *locusTag*, *Synonyms*, *dbXrefs*, *chromosome*, *map location*, *description*, *type of gene*, *Symbol from nomenclature authority*, *Full name from nomenclature authority*, *Nomenclature status*, *Other designations* and *Modification date* (see ftp://ftp.ncbi.nih.gov/gene/ README for further explanation about these columns). The file contained more than 2 million genes. Only a small fraction of the information from this file was needed and it was decided to load the file as a whole into a temporarily table in the database, *geneInfo_tmp*, and retrieve the information needed with MySQL queries. The format of the temporarily table and the syntax for making it is shown Appendix 9.1. The procedure of how to make tables and loading them with data is described in section 4.4.

The file "homologene.data" was downloaded from Entrez HomoloGene on October 4. 2006. The file was located at: ftp://ftp.ncbi.nih.gov/pub/HomoloGene/current.This file was also tab-delimited with the columns *HID*, *Taxonomy ID*, *Gene ID*, *Gene Symbol*, *Protein gi* and

Protein accession (see ftp://ftp.ncbi.nih.gov/pub/HomoloGene/README for further details about the columns). It contained nearly 180000 genes with one gene for each line. The column *HID* corresponds to the attribute *homologeneid*. A temporary table, *homoloGene_tmp*, was made and the file "homologene.data" was loaded into this table.

A SELECT statement to retrieve all attributes in the table gene was run against the table *homoloGene_tmp* and *geneInfo_tmp* for all genes in *homoloGene_tmp* (Appendix 9.4). This procedure is described in section 4.4.3. The result of this query was written to the text-file "orthOutfile.txt". The file "orthOutfile.txt" was loaded directly into the table gene by using a similar script as used for all loading procedures (see section 4.4.2).

All orthologs were now found and loaded into the table *gene*. The final number of genes became 1385.

As shown in figure 11, the attribute *synonym* should be in the table *synonym*. Another query was run against *orth_tmp* (Appendix 9.7) to retrieve all synonyms and the result was written to the file "synonymOutfile.txt". In this text-file the first column was *gene_id* and the second column was *synonym*. If a gene had more than one synonym, they were separated by a "|", see figure 13. This file could therefore not be loaded directly into the table synonym which has only one gene_id and one synonym on each line.

```
11606   AI265500|Aogen|Serpina8
24179   ANRT|Ang|AngII|MGC105326|PAT
```

Figure 13. Two lines from synonymOutfile.txt showing that synonyms are separated by "|".

The file "synonymOutfile.txt" was therefore run through a python script which wrote the content to a new file, "fixed_file.tab", this one with one gene and one synonym on each line (this script is described in section 4.4.4). The two lines in figure 13 were then modified to the eight lines as shown in figure 14 which is a small fraction of the file "fixed_file.tab".

```
11606   AI265500
11606   Aogen
11606   Serpina
24179   ANRT
24179   Ang
24179   AngII
24179   MGC105326
24179   PA
```

Figure 14. A small fraction of the file fixed_file.tab, this one on the right format to be loaded into the table synonym

The file “fixed_file.tab” was loaded into the table *synonym* with a script pretty much like the one shown in figure 16, except for the specification of colon-delimited columns.

4.3.4 The tables sysname and famname

All initial genes were grouped into DNA repair pathways (Wood et al. 2005). Some of these pathways had families within.

The table *sysname* was made as shown in Appendix 9.2 and the DNA repair pathways were numbered from 1 to 15, this functioning as a unique id. Id and pathway were manually loaded into the table *sysname* as the attributes *sys_id* and *sysname*. The same procedure was followed when making *famname* and loading it with data.

Even if the table *famname* was made and all genes in the database were connected to this table through a unique id it was decided not to retrieve the family names through the queries because this information was not considered important when genes were already grouped into repair systems.

4.4 MySQL-queries and PHP

4.4.1 Creating tables

A script called “make_table.php” was made on the purpose of making all tables (see Appendix 9.2). This script was run every time a new table was being made as a new CREATE TABLE statement with attributes was added. Figure 15 gives an outline of how this was done. The table made in this specific example is *gene*, but all other tables were made in a similar manner.

```
CREATE TABLE IF NOT EXISTS gene (
gene_id int(15) NOT NULL default 0,
homologeneid int(11) NULL,
sysid int(10) NOT NULL default 0,
famid int(10) NOT NULL default 0,
taxid int(11) NOT NULL default 0,
gene_name varchar(100) NOT NULL default ‘’,
description varchar(100) NOT NULL default ‘’,
chrom_location varchar(20) NOT NULL default ‘’,
PRIMARY KEY (gene_id)
);
```

Figure 15. The table gene created with MySQL-code. All tables in MySQL are made in a similar manner.

Since the script “make_table.php” was run several times, the statement IF NOT EXIST was added to all CREATE TABLE-statements. This to make sure a table was made only once.

4.4.2 Loading tables with data

All tables in the database were loaded automatically with data through scripts. Data were first in a text-file which was formatted in such a way that it was possible to load it directly in to the table. Figure 16 gives an example of such a script. That specific example shows how the table gene was loaded with data.

```
<?php
include("opendb.php");

//selecting the correct database
mysql_select_db("lise");

//reading the file gene.txt and loading the table gene
$load = "LOAD DATA INFILE '/users/lihe/database/gene.txt' IGNORE INTO TABLE gene
FIELDS TERMINATED BY ':' ";

$results = mysql_query($load)
or die( mysql_error());
?>
```

Figure 16. PHP-script embedded with MySQL for loading the table gene.

In figure 16 the statement LOAD DATA INFILE is reading the rows of the file "gene.txt" into the table *gene*. The whole path of the file is stated. The IGNORE statement was added to the script to make sure no data was added twice. Last, the statement FIELDS TERMINATED BY was added in this script because the file was colon-delimited. Most of the files loaded into the tables were however tab-delimited and this last statement was then not necessary. All tables in the database were loaded in this manner.

4.4.3 Querying temporary tables to find all gene info for homologs

The temporarily tables *tax_tmp*, *geneInfo_tmp* and *homoloGene_tmp* was used as described in section 4.3.3 to retrieve all attributes in *taxonomy* and *gene* for all genes in *geneInfo_tmp*. In figure 17 the query run against *geneInfo_tmp* and *homoloGene_tmp* shown. The query run against *tax_tmp* is shown in Appendix 9.9.

After the SELECT statement all attributes are selected in the same order as the attributes in gene. This to make sure the output file would be on such a format that it was possible to load it directly into the table gene. The only information retrieved from *homoloGene_tmp* was *homolog_id* which corresponds to the attribute homologeneid. This can be seen from the query in figure 17. The result from this query was written to the text-file "orthOutfile.txt".

```
<?php

  include("opendb.php");

  //selecting the correct database
  mysql_select_db("lise");

  //retrieving all attributes in gene for all genes in
  //homoloGene_tmp and writing them to the txt-file
  //orthOutfile.txt
  $outfile = "SELECT g.taxid, g.gene_id, g.name, g.synonyms, ".
  "g.map_location, g.description, h.homolog_id ".
  "FROM geneInfo_tmp g, homoloGene_tmp h ".
  "WHERE g.gene_id = h.geneid ".
  "INTO OUTFILE '/users/lihe/database/orthOutfile.txt' ";
  $result = mysql_query($outfile)
  or die(mysql_error());

  echo "records exported to txt-file\n";

?>
```

Figure 17. Query for finding all attributes in gene for all genes in the temorarily table homoloGene_tmp. The result is written to the text-file orthOutfile.txt.

4.4.4 Handling gene synonyms

As described in section 4.3.3, a python script was used to modify the file "synonymOutfile.txt" in such a way that it could be loaded into the table *synonym*. Figure 18 shows this script which was made by Gard Thomassen at CMBN (Appendix 9.4). It reads the infile and removes all gene_id's with no synonyms, makes a new line and repeat the previous gene_id as many times as there are synonyms.

```
import sys

infile = open(sys.argv[1],'r')
outfile = open(sys.argv[2], 'w')

for line in infile:
    line = line.split('\t')
    if not (line[1][0] == "-"):
        num = line[0]
        names = line[1].split('|')
        names[-1]=names[-1][:-2]
        for name in names:
            outfile.write(num+"\t"+name+"\n")
```

Figure 18. fixe_fil.py.

4.4.5 Search form with checkboxes

A form was needed so search criteria could be taken in from the browser. The script “rep.php” was made for this purpose (see Appendix 9.8). In the file “rep.php” checkboxes was made so that the user may choose which organisms to include in the search. In addition to having one checkbox for each organism there was also made a checkbox to select all organisms at once. Due to the fact that this procedure is very complicated in PHP, it was decided to use a small Javascript found at http://www.phpfreaks.com/quickcode/Checking-all-checkboxes-in-a-form/311.php to solve the problem. This script is shown in figure 19 and is also embedded in the PHP-script in Appendix 9.8.

```
<script language="JavaScript">
function Checkall(form){
   for (var i = 1; i < form.elements.length; i++){
    eval("form.elements[" + i + "].checked = form.elements[0].checked");
  }
}
</script>
```

Figure 19. Checking all elements in a form with Javascript

The user should be able to choose either to search for DNA repair system or search the database with at text string, so both choices were made possible in the file “rep.php”. As seen in the file “rep.php” in Appendix 9.8, all input is sent to the script “search.php” (Appendix 9.10).

4.4.6 Searching the database

As mentioned in previous section, all input from the browser through the form “rep.php” was sent to the script “search.php” (Appendix 9.10). First a test was made, to make sure at least one of the boxes is checked from the browser. A message to the user to do so was also made. Next three different conditions were tested for; if a search string is sent through the browser, if a DNA repair system is picked or if the search button is pressed without making a choice (these tests will from now on be named 1, 2 and 3 respectively). The three different tests and the following action will now be described:

Test 1:

First an if statement checks both that at least one box is checked and that a DNA repair system is not picked. If this condition is true another if statement checks that the text string sent from the browser is not empty. If the string is empty, a message will be sent to the browser. If, on the other hand, the condition is true, a query is made to the database. The query looks for the search string in the attributes *gene_name*, *synonym* and *description*. If the query returns a result, this result is sent to the function *make_table*. If the query does not return any result, a message will be sent to the browser.

Test 2:

The condition in this test is that a DNA repair system must have been picked and at east one box checked. If true, another test checks if only one such system is picked. If this condition is true also, a query is made to retrieve the header for this DNA repair system. Next follows a query to retrieve all genes in this DNA repair system. The result is sent to the function *make_table*. If the previous condition is not true, another one is checked for; if all systems are picked. When this condition is fulfilled a query is made to retrieve all genes in all DNA repair systems for the organisms picked. The result is sent to the function *make_table*.

Test 3:

The condition in this test is that no choices are made, but the search button is pressed. If so, a message will be sent to the browser.

The functions *make_table*, *table_border* and *table_space* print the result. In the ouput format, orthologs are grouped and there is some space between the groups. The function *make_table* also adds links to the output for the first three attributes written.

5 Results

The result of this work is a relational database with a dynamic interface. 18 organisms and 1385 genes are included. The user can pick which organisms to include, but at least one most be selected. Then it is possible to search either on DNA repair system or for gene name in a text field. If the search is on DNA repair system, all genes in the system picked will be grouped into orthologs. If the text field is used, the system will search for this string in gene name, synonym and description of the gene. The resulting output will be grouped into orthologs here as well. For both choices the output provides links to NCBI's HomoloGene, Taxonomy Browser and Entrez Gene.

The Database of Genes Involved in DNA repair has a homepage as shown in figure 20. From this site it is possible to choose which organisms to include in the search, at least one is required. Next it is possible to make a search on either DNA repair system or enter a gene name. If DNA repair system is chosen, it is possible to choose one specific system or all systems. If a gene name is searched for, the search will go through synonyms and description as well as official name. Two examples will illustrate the features provided:

Example one:

On figure 21, two organisms are checked, *H. sapiens* and *M. musculus*. Next, a DNA repair system is chosen: Direct reversal of damage. The search button is pressed. The resulting output of this search is shown in figure 22. Six genes match the search criteria and they are grouped in three groups of orthologs; two genes in each group. The heading shows that these six genes is grouped within Direct reversal of damage. The output in the three leftmost columns provides links to Entrez HomoloGene, Entrez Taxonomy Browser and Entrez Gene respectively. Underneath the result there is a link back to the homepage called 'Back to search'.

Example two:

In figure 23, the boxes *H.sapiens*, *M. musculus*, *A. thaliana* and *R. norvegicus* are checked and the search string 'alk' is entered in the text field. The search button is pressed and the result of the search is shown in figure 24. The output of the search is six genes grouped in two orthologous groups with three genes in each group. Note that *A. thaliana* is not represented because the search string 'alk' did not return any result in this organism.

Next the link ‘16393’ in the column Homolog Gene Id was clicked. The Id links to Entrez HomoloGene and the result of clicking the link is shown in figure 24. The page shows all orthologous genes having the Homolog gene identification ‘16393’. Note that *A. thaliana* is represented on this page due to the fact that ‘alk’ was no longer a term, only the link to Entrez HomologGene with Id ‘16393’ was followed.

Back in figure 24 the link ‘Homo sapiens’ is clicked and the result of doing so is shown in figure 26. Links in the column Organism are connected to Entrez Taxonomy Browser.

Again back in figure 24, this time clicking on ‘ALKBH3’ in the column Official Name. The result of clicking this link is shown in figure 27. Links in the column Official name are connected to Entrez Gene.

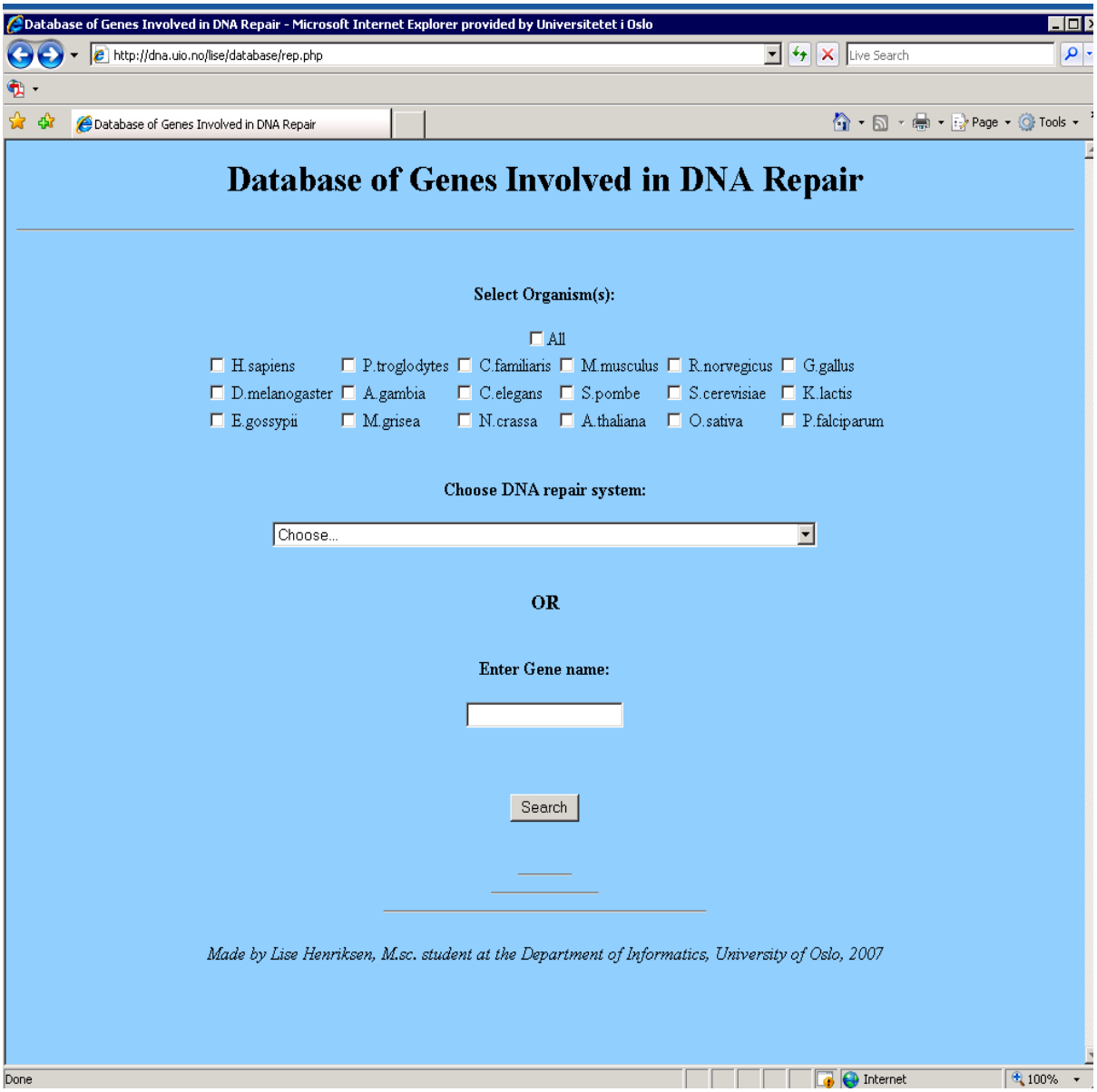

Figure 20. Homepage.

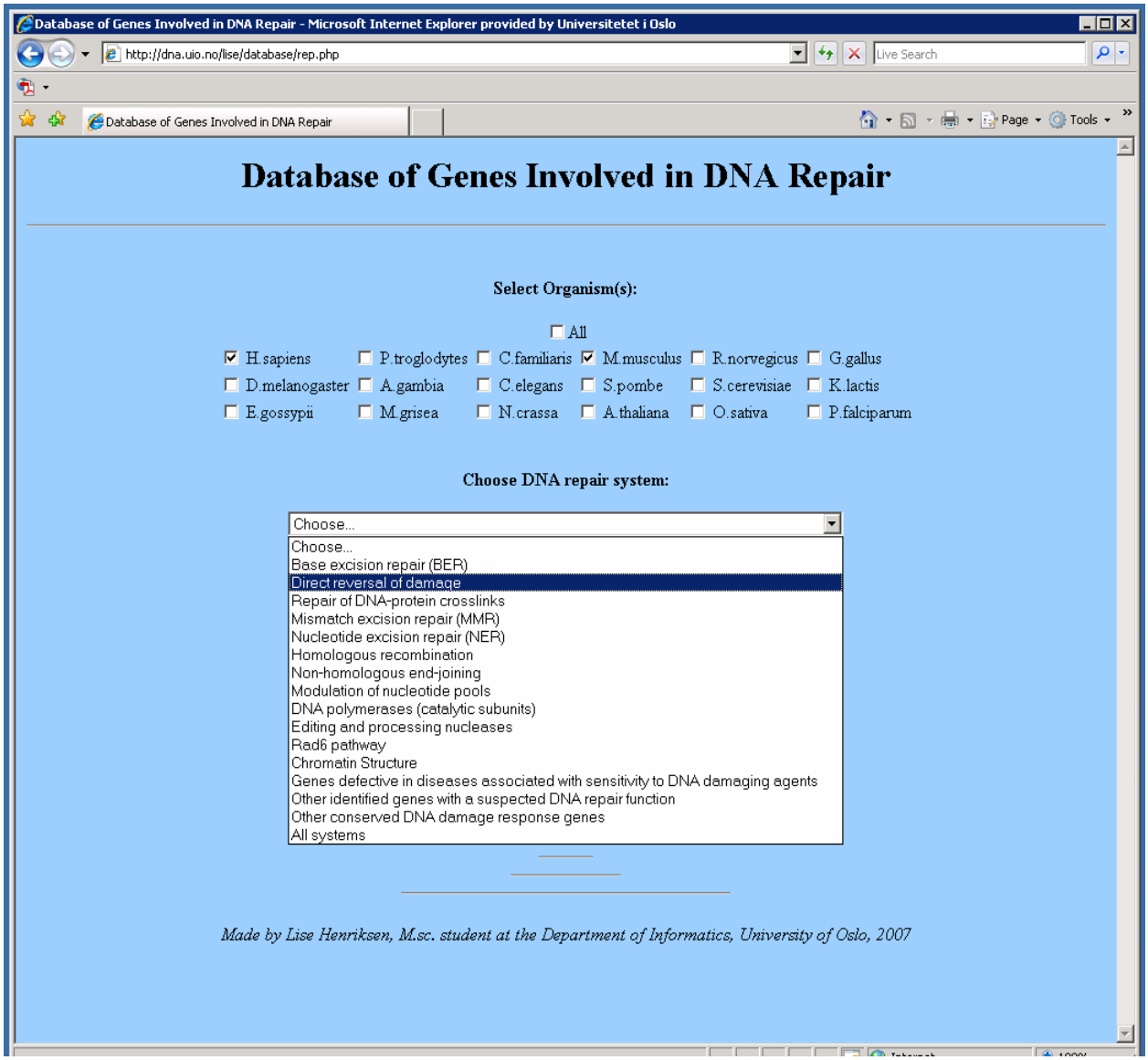

Figure 21. Search based on DNA repair system.

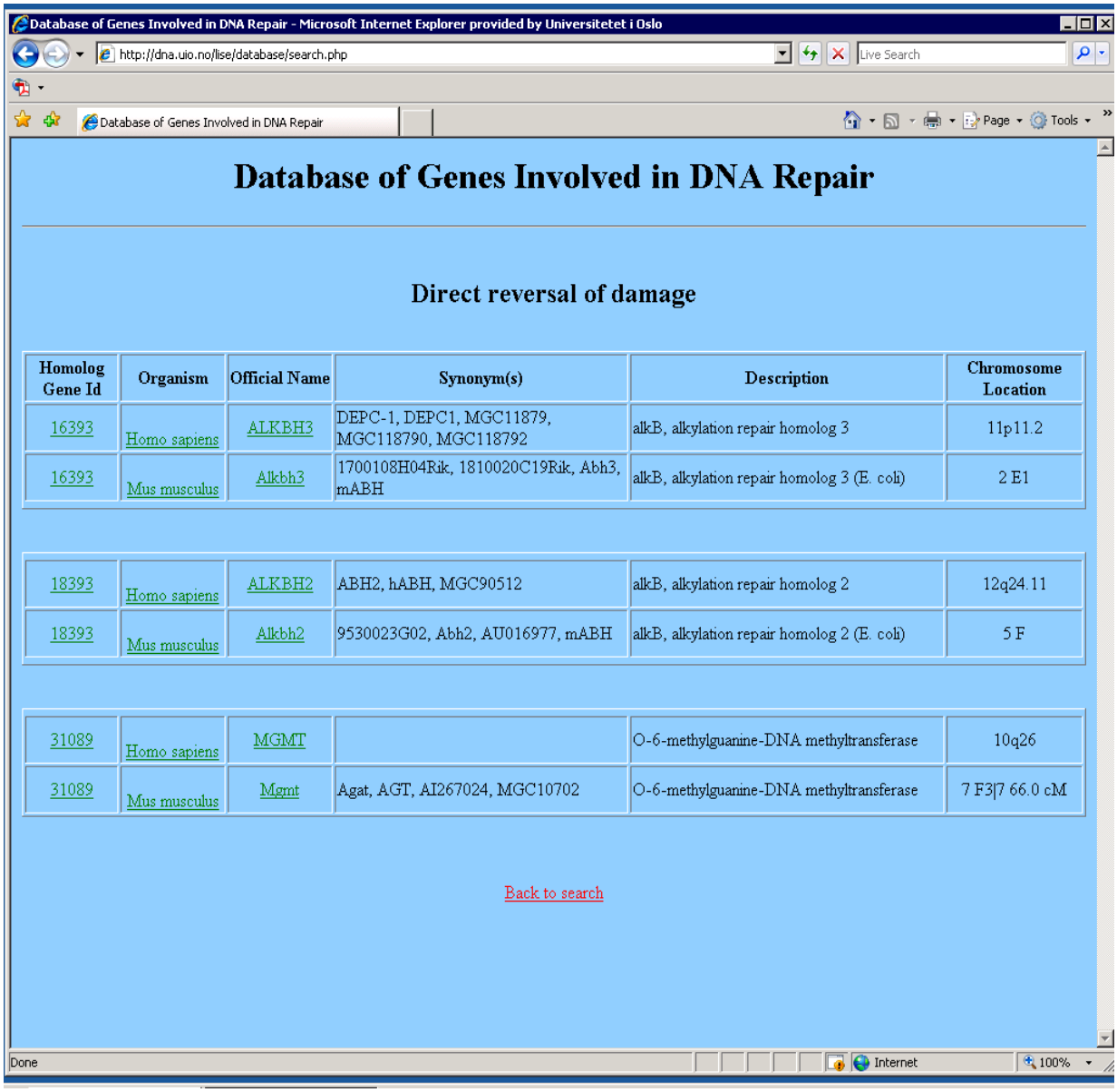

Figure 22. Result of search specified in figure 21.

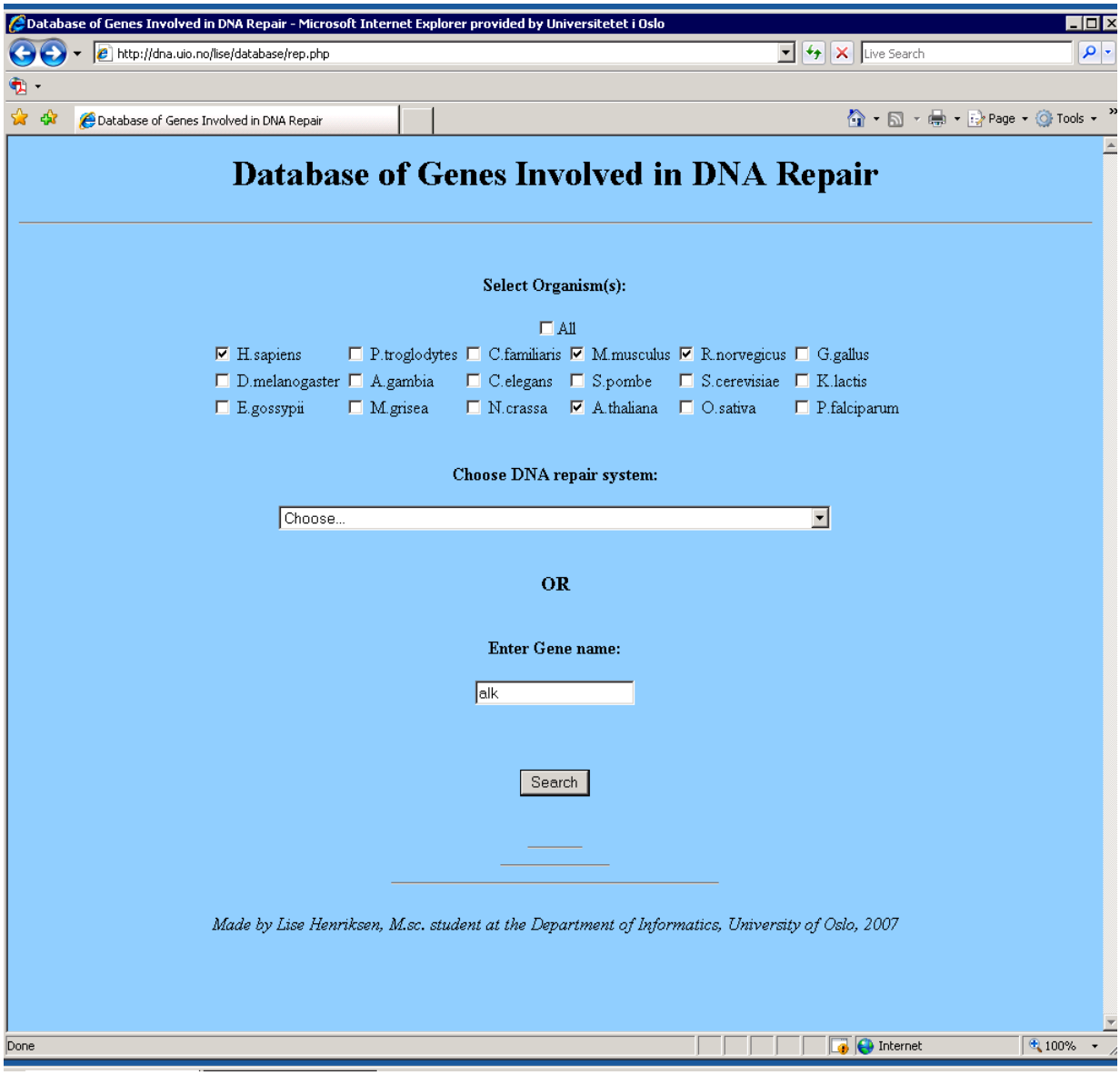

Figure 23. Search based on Gene name.

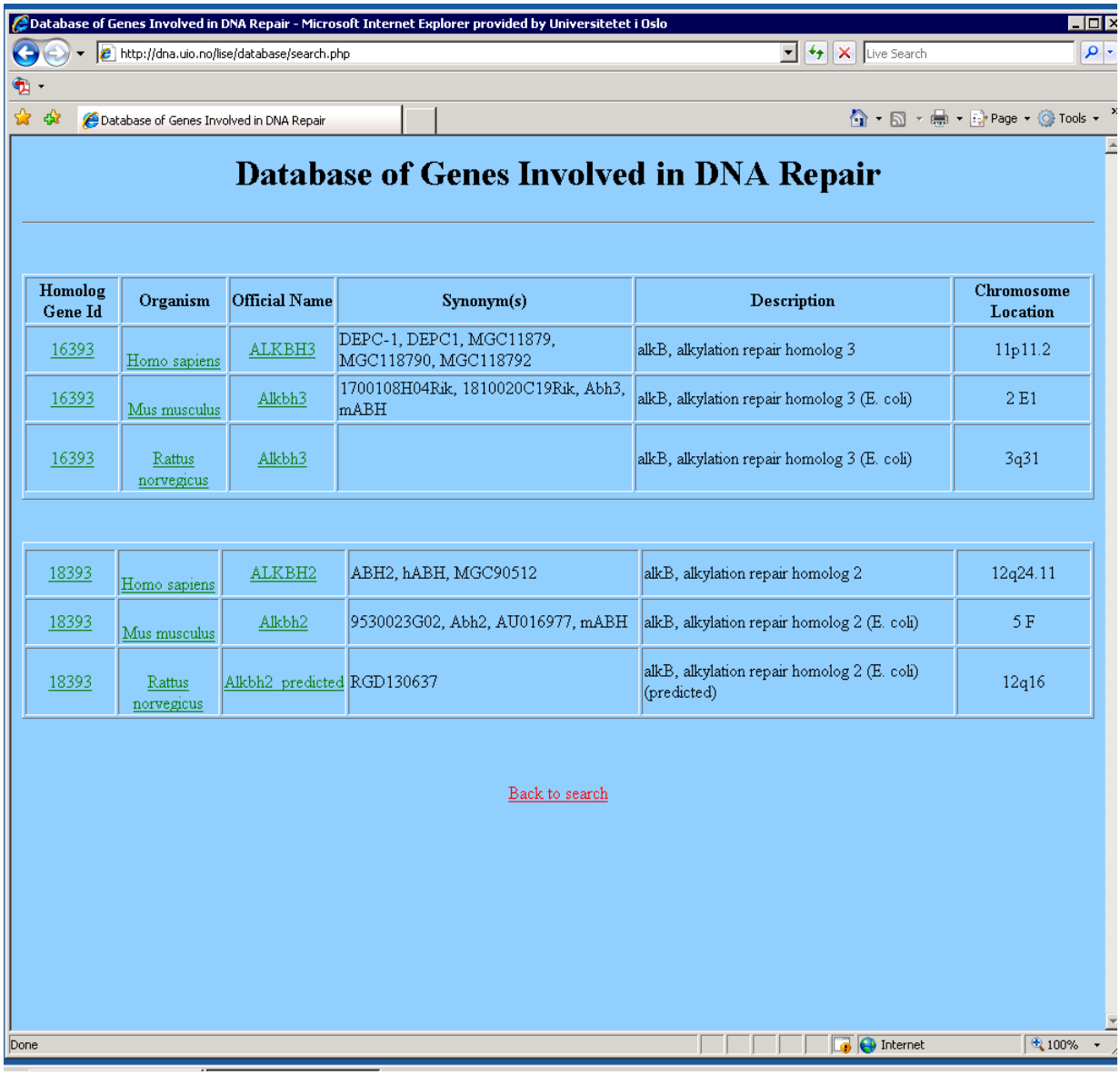

Figure 24. Result of search specified in figure 23.

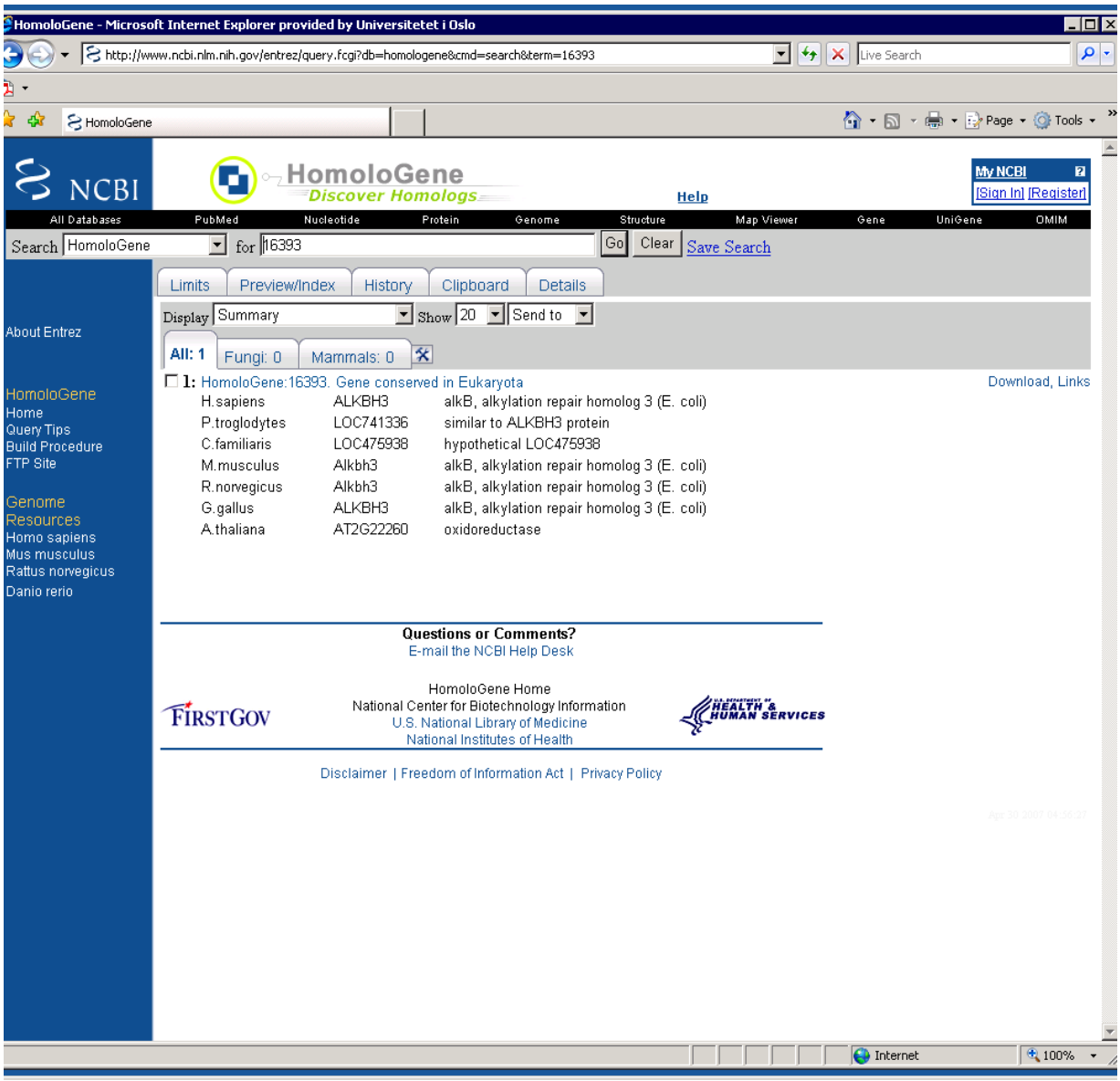

Figure 25. In figure 24, the Homolog Gene Id 16393 was clicked. The result of that action is shown here.

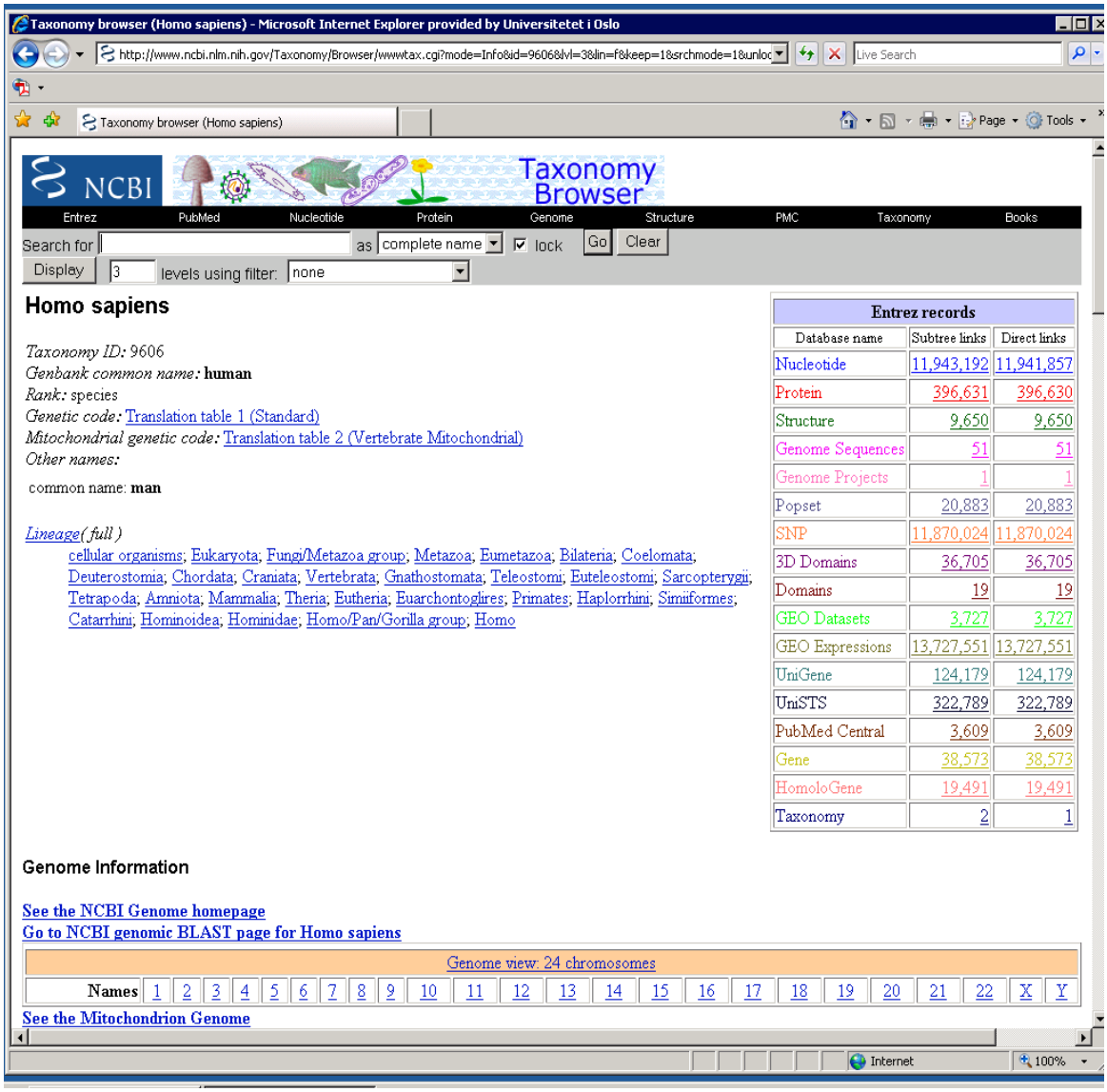

Figure 26. The link ‘Homo sapiens’ is clicked in figure 24. The result of doing so is shown here.

Figure 27. The link ‘ALKBH3’ in column Offical Name was clicked in figure 24 and the result is shown here.

6 Future work

There are many interesting extensions possible for this database. Some of them will be mentioned here.

- This database was originally planned to contain DNA repair genes from all genomes with completely sequenced genomes. But as discussed in section 4.3.1, Inparanoid could not be used for finding the orthologous groups and Entrez HomoloGene was then the only resource used for finding orthologs. HomoloGene has so far only included 18 organisms, and these are included in the Database of Genes involved in DNA repair. In the future, an extension such as including DNA repair genes from all organisms which genomes are completely sequenced would come naturally. It would however probably be necessary to use a different information system than NCBI to identify the genes.

- Another possible extension is to include not only DNA repair genes in the database, but other genes as well. The genes could still be grouped in orthologs, but it would perhaps then be natural to exclude the pathways to get a broader perspective. The amount of data available in the database would increase a lot and it would be a helpful tool not only for researchers focusing on DNA repair genes, but all scientists doing research on DNA.

- Through the links to Entrez Gene in the column official name, it is possible to get information about sequences for genes. An interesting extension could be to have the opportunity to see sequence alignments for genes in an orthologous group in this database. The structure of the database would then have to be changed as well as most of the PHP code.

- The database has links to Entrez HomoloGene, Entrez Gene and Enterez Taxonomy Browser. A possible extension is to include links to some other databases on the Internet as well. This could be useful especially if the different databases do not provide similar information.

- When searching for all organisms and all DNA repair systems, the database system needs a few seconds to retrieve the result. The reason for this is the function make_table in the script search.php. A double for-loop causes the time consumed to be N^2. This could be improved especially if more amounts of data are being loaded into the database.

7 Acknowledgements

Several people have supported, helped and advised me throughout my studies. First I would like to thank my supervisor Torbjørn Rognes for being patient and available when most needed. Thank you for your supervising and for pushing me a little when I lost tracks. I sure needed that.

I would also like to thank all the people at the CMBN bioinformatics group and at the bioinformatics group at the Department of informatics for their advice and support and also for their joyful company during lunches. Thank you, Gard Thomassen, for fixe_fil.py. Thank you, Jon Myrseth, for helping me with my database when needed.

Thank you, Ragnar Normann, for discussing databases with me.

Thank you, Jon Grov, for discussing SQL during coffee-breaks and commenting on my work.

Thank you, Hans Krokan, for lending me your illustrations.

I would also like to thank all my family and friends. I couldn't have done this without you.

Last, but not least, I would like to thank my daughter Ylva for being the most patient seven year old ever. Who said that kids these days can't entertain themselves?

I love you

8 References

Begley T J, Samson L D (2004), Network responses to DNA damaging agents, DNA repair 3 (2004), 1123-1132

Birney E, Andrews D T, Bevan P, Caccamo M , Chen Y, Clarke L, Coates G, Cuff J, Curwen V, Cutts T, Down T, Eyras E, Fernandez-Suarez X M, Gane P, Gibbins B, Gilbert J, Hammond M, Hotz H R, Iyer V, Jekosch K, Kahari A, Kasprzyk A, Keefe D, Keenan S, Lehvaslaiho H, McVicker G, Melsopp C, Meidl P, Mongin E, Pettett R, Potter S, Proctor G, Rae M, Searle S, Slater G, Smedley D, Smith J, Spooner W, Stabenau A, Stalker J, Storey R, Ureta-Vidal A, Woodwark C K, Cameron G, Durbin R, Cox A, Hubbard T, Clamp M (2004), An overview of Ensembl, Genome Res., 14:925-928, 2004

Brown T A (2002), Genomes, 2nd ed, BIOS Scientific Publishers Ltd, Oxford

Doherty A, Daffron T (2000), Nick Recognition by DNA Ligases, J. Mol. Biol. (2000) 296, 43-56

Drabløs F, Emadoldin F, Aas P A, Vaagbø C B, Kavli B, Bratlie M S, Peña-Diaz J, Otterlei M, Slupphaug G, Krokan H E (2004), Alkylation damage in DNA and RNA – repair mechanisms and medical significance, DNA Repair 3 (2004), 1389-1407

Eidhammer I, Jonassen I, Taylor W R (2004), Protein Bioinformatics An Algoritmic approach to sequence and structure analysis, John Wiley & Sons Ltd, England

Friedberg C, Walker G C, Siede W, Wood R D, Schultz R A, Ellenberger T (2006), DNA repair and Mutagenesis, 2nd ed, ASM Press, Washington DC

Fortini P, Pascucci B, Parlanti E, D'Errico M, Simonelli V, Dogliotti E (2003), The base excision repair: mechanisms and its relevance for cancer susceptibility, Biochimie (2003), 1053-1071

Garcia-Molina H, Ullman J D, Widom J (2002), Database Systems The Complete Book, Prentice-Hall, Inc., New Jersey

Giacomo M D (2005), MySQL: Lessons learnd on a digital library, Published by the IEE Computer Society

Halpin T (1996), Business rules and object role modelling, Database Programming & Design, October 1996

Kao Y, Saxena C, Wang L, Sancar A, Zhong Dongping (2005), Direct observation of thymine dimer repair in DNA by photolyase, PNAS, No. 45, Vol. 102, 16128-16132

Krokan H E, Kavli B, Slupphaug G (2004), Novel aspects of macromolecular repair and relationship to human disease, J Mol Med , 82:280-297

Kunkel T A, Erie D A (2005), DNA Mismatch Repair, Annu. Rev. Biochem 2005, 74:681-710

Maglott D, Ostell J, Pruitt K D, Tatusova T (2005), Entrez Gene: Gene-centered information at NCBI, Nucleid Acids Research, 2005 January 1; 33(Database Issue): D54-D58

O'Brien K P, Remm M, Sonnhammer E L L (2005), Inparanoid, a comprehensive database of of eukaryotic orthologs, Nucleic Acids Research, Vol 33, Database issue, D476-D480

Remm M, Storm C E V, Sonnhammer E L L (2001), Automatic clustering of orthologs and in-paralogs from pairwise species comparsions, J. Mol. Biol. , 314, 1041-1052

Reardon J T, Sancar A (2005), Nucleotide Excision repair, Progress in Nucleic Acid Research and Molecular Biology, Vol. 79

Skagestein G (2002), Systemutvikling - fra kjernen og ut, fra skallet og inn, Høyskoleforlaget 2002

Weber S (2005), Biochimica et Biophysica Acta, 1707 1-23

Wheeler D L, Barrett T, Benson D A, Bryant S H, Canese K, Chetvernin V, Church D M, DiCuccio M, Edgar R, Federhen S, Geer L Y, Helmberg W, Kapustin Y, Kenton D L, Khovayko O, Lipman D J, Madden T L, Maglott D R, Ostell J, Pruitt K D, Schuler G D, Schriml L M, Sequeira E, Sherry S T, Sirotkin K, Souvorov A, Starchenko G, Suzek T O, Tatusov R, Tatusova T A, Wagner L, Yaschenko E (2005), Database resources of the National Center for Biotechnology Information, Nucleic acid research, 2006, Vol. 34, Database Issue, D173-D180

Williams H E, Lane D (2005), The PHP Scripting Language (excerpted from chapter two of the book Web Database Applications with PHP and MySQL, 2004), O'Reilly Media, 2005-09-29

Wood R D, Mitchell M, Lindahl T (2005), Human DNA repair genes, Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis, Vol. 577, 275-283

9 Appendix

Code made during development of the database is shown in this Appendix. A short explanation of the code is on top of each section.

9.1 opendb.php

Opens the connection to the database. This script is included in top of all other scripts made.

```
<?php
$dbc =
mysql_connect("127.0.0.1:/tmp/mysql.sock","lise","flyplass");
if (!$dbc) {
  echo "<p>ERROR: Unable to establish connection with database.</p>";
  exit;
}
if (!mysql_select_db("lise")) {
  echo "<p>ERROR: Unable to find selected database!" . " *** ERROR: " . mysql_error();
  exit;
}

?>
```

9.2 make_table.php

This code makes all tables in the database. The script was run every time a new table was added.

```
<?php

include("opendb.php");

$gene = "CREATE TABLE IF NOT EXISTS gene (
gene_id int(15) NOT NULL default 0,
homologeneid int(11) NULL,
sysid int(10) NOT NULL default 0,
famid int(10) NOT NULL default 0,
taxid int(11) NOT NULL default 0,
gene_name varchar(100) NOT NULL default '',
description varchar(100) NOT NULL default '',
chrom_location varchar(20) NOT NULL default '',
PRIMARY KEY (gene_id)
)";


$synonym = "CREATE TABLE IF NOT EXISTS synonym (
gene_id int(15) NOT NULL default 0,
synonym varchar(100) NOT NULL default '',
PRIMARY KEY (gene_id, synonym)
)";


$taxonomy = "CREATE TABLE IF NOT EXISTS taxonomy (
tax_id int(11) NOT NULL default 0,
organism_name varchar(50) NOT NULL default '',
```

```
PRIMARY KEY (tax_id)
)";


$famname = "CREATE TABLE IF NOT EXISTS famname (
fam_id int(10) NOT NULL default 0,
famname varchar(75) NOT NULL default '',
PRIMARY KEY (fam_id)
)";


$sysname = "CREATE TABLE IF NOT EXISTS sysname (
sys_id int(10) NOT NULL default 0,
sysname varchar(75) NOT NULL default '',
PRIMARY KEY (sys_id)
)";


$tax_tmp = "CREATE TABLE IF NOT EXISTS tax_tmp (
tax_id int(7) NOT NULL default 0,
name varchar(150) NOT NULL default '',
unique_name varchar(150) NOT NULL default '',
name_class varchar(50) NOT NULL default '',
PRIMARY KEY (tax_id, name, name_class)
)";

$homoloGene_tmp = "CREATE TABLE IF NOT EXISTS homoloGene_tmp (
homolog_id int(6) NOT NULL default 0,
taxid int(7) NOT NULL default 0,
geneid int(15) NOT NULL default 0,
gene_symbol varchar(100) NOT NULL default '',
protein_gi int(15) NOT NULL default 0,
protein_acc varchar(20) NOT NULL default 0,
PRIMARY KEY (homolog_id, taxid, geneid)
)";

$geneInfo_tmp = "CREATE TABLE IF NOT EXISTS geneInfo_tmp (
taxid int(7) NOT NULL default 0,
gene_id int(15) NOT NULL default 0,
name varchar(100) NOT NULL default '',
locus_tag varchar(50) NOT NULL default '',
synonyms varchar(150) NOT NULL default '',
dbXrefs varchar(150) NOT NULL default '',
chromosome varchar(10) NOT NULL default '',
map_location varchar(1000) NOT NULL default '',
description varchar (1000) NOT NULL default '',
type varchar(100) NOT NULL default '',
symb_from_nom varchar(150) NOT NULL default '',
full_name varchar(150) NOT NULL default '',
status varchar(5) NOT NULL default '',
PRIMARY KEY (gene_id)
)";

$orth_tmp = "CREATE TABLE IF NOT EXISTS orth_tmp (
taxid int(7) NOT NULL default 0,
geneid int(15) NOT NULL default 0,
name varchar(100) NOT NULL default '',
synonyms varchar(150) NULL ,
chrom_location varchar(20) NOT NULL default '',
description varchar(200) NOT NULL default '',
homologid int(11) NOT NULL default 0,
PRIMARY KEY (geneid, homologid)
)";

$taxid_tmp = "CREATE TABLE IF NOT EXISTS taxid_tmp (
taxid int(7) NOT NULL default 0,
PRIMARY KEY (taxid)
)";
```

```
mysql_query($gene) or die(mysql_error());
mysql_query($synonym) or die(mysql_error());
mysql_query($taxonomy) or die(mysql_error());
mysql_query($famname) or die(mysql_error());
mysql_query($sysname) or die(mysql_error());
mysql_query($tax_tmp) or die(mysql_error());
mysql_query($homoloGene_tmp) or die(mysql_error());
mysql_query($geneInfo_tmp) or die(mysql_error());
mysql_query($orth_tmp) or die(mysql_error());
mysql_query($taxid_tmp) or die(mysql_error());

?>
```

9.3 load_tables

All tables in the database were loaded by scripts similar to load_gene.php. The names of all these scripts: load_gene.php, load_synonym.php, load_taxonomy.php, load_famname.php, load_sysname.php, loadTax_tmp.php, loadHomoloGene_tmp.php, loadOrth_tmp.php and loadTaxid_tmp.php. The only difference being the name of the file and that most text-files being loaded were tab-delimited. load_gene.php will therefore be the reference for all scripts used for loading.

```
<?php
include("opendb.php");

//selecting the correct database
mysql_select_db("lise");

//reading the file gene.txt and loading the table gene
$load = "LOAD DATA INFILE '/users/lihe/database/gene.txt' IGNORE INTO TABLE gene
FIELDS TERMINATED BY ':' ";

$results = mysql_query($load)
or die( mysql_error());

?>
```

9.4 fixe_fil.py

Formates the file containing all the synonyms so it can be loaded directly into the table.

```
import sys

infile = open(sys.argv[1],'r')
outfile = open(sys.argv[2], 'w')

for line in infile:
    line = line.split('\t')
    if not (line[1][0] == "-"):
        num = line[0]
        names = line[1].split('|')
        names[-1]=names[-1][:-2]
        for name in names:
            outfile.write(num+"\t"+name+"\n")
```

9.5 HomologOut.php

Querying gene and orth_tmp to find orthologs to all initial genes.

```
<?php

include("opendb.php");

//selecting the correct database
mysql_select_db("lise");

//finding all homologs to the initial genes and writing
//them to the txt-file homologOutfile.txt
$outfile = "SELECT o.geneid, o.homologid, g.sysid, g.famid, o.taxid, ".
"o.name, o.description, o.chrom_location ".
"FROM orth_tmp o, gene g ".
"WHERE o.homologid = g.homologeneid ".
"INTO OUTFILE '/users/lihe/database/homologOutfile.txt' ";
$result = mysql_query($outfile)
or die(mysql_error());

echo "records exported to txt-file\n";

?>
```

9.6 orthologOut.php

Retrieving all attributes in gene for all genes in homoloGene_tmp.

```
<?php

include("opendb.php");

//selecting the correct database
mysql_select_db("lise");

//retrieving all attributes in gene for all genes in
//homoloGene_tmp and writing them to the txt-file
//orthOutfile.txt
$outfile = "SELECT g.taxid, g.gene_id, g.name, g.synonyms, ".
"g.map_location, g.description, h.homolog_id ".
"FROM geneInfo_tmp g, homoloGene_tmp h ".
"WHERE g.gene_id = h.geneid ".
"INTO OUTFILE '/users/lihe/database/orthOutfile.txt' ";
$result = mysql_query($outfile)
or die(mysql_error());

echo "records exported to txt-file\n";

?>
```

9.7 synonymsOut.php

Retrieving geneid and synonym from orth_tmp.

```
<?php

include("opendb.php");

//selecting correct database
mysql_select_db("lise");

//retrieving geneid and synonym from orth_tmp
$outfile = "SELECT geneid, synonyms ".
"FROM orth_tmp INTO OUTFILE '/users/lihe/database/synonymOutfile.txt' ";
$result = mysql_query($outfile)
or die(mysql_error());

echo "records exported to txt-file\n";

?>
```

9.8 rep.php

Making the form on the homepage

```
<html><body>
<head>
<title>Database of Genes Involved in DNA Repair</title>
</head>
<body
BGCOLOR="#99ccff"TEXT="black"LINK="#008800"VLINK="red"ALINK="#ffff00"><center><h1>D
atabase of Genes Involved in DNA Repair</h1></center>
<HR>

<p>
<br />
</p>
<script language="JavaScript">
function Checkall(form){
   for (var i = 1; i < form.elements.length; i++){
    eval("form.elements[" + i + "].checked = form.elements[0].checked");
  }
}
</script>

<form method = POST action = "search.php">
<center>
<b>Select Organism(s):</b><BR><BR>
<input type="checkbox" onClick="Checkall(this.form);" />All<br />
<table>
<tr>
<td><input type="checkbox" NAME="box[]" VALUE="9606" >
H.sapiens<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE="9598" >
P.troglodytes<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE="9615" >
C.familiaris<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "10090">
M.musculus<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "10116" >
R.norvegicus<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "9031" >
G.gallus<BR></td>
</tr>
<tr>
<td><input type="checkbox" NAME="box[]" VALUE= "7227" >
D.melanogaster<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "180454" >
A.gambia<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "6239" >
C.elegans<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "284812" >
S.pombe<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "4932" >
S.cerevisiae<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "284590" >
K.lactis<BR></td>
</tr>
<tr>
<td><input type="checkbox" NAME="box[]" VALUE= "33169" >
E.gossypii<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "242507" >
M.grisea<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "5141" >
N.crassa<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "3702" >
A.thaliana<BR></td>
```

```
<td><input type="checkbox" NAME="box[]" VALUE= "39947" >
O.sativa<BR></td>
<td><input type="checkbox" NAME="box[]" VALUE= "36329" >
P.falciparum<BR></td>
</tr>
</table>
</center>
<center>
<p>
<br />
</p>
<p>

<p><b>Choose DNA repair system:</b></p> <SELECT NAME="System">
<OPTION VALUE=0>Choose...
<OPTION VALUE=1>Base excision repair (BER)
<OPTION VALUE=2>Direct reversal of damage
<OPTION VALUE=3>Repair of DNA-protein crosslinks
<OPTION VALUE=4>Mismatch excision repair (MMR)
<OPTION VALUE=5>Nucleotide excision repair (NER)
<OPTION VALUE=6>Homologous recombination
<OPTION VALUE=7>Non-homologous end-joining
<OPTION VALUE=8>Modulation of nucleotide pools
<OPTION VALUE=9>DNA polymerases (catalytic subunits)
<OPTION VALUE=10>Editing and processing nucleases
<OPTION VALUE=11>Rad6 pathway
<OPTION VALUE=12>Chromatin Structure
<OPTION VALUE=13>Genes defective in diseases associated with sensitivity to DNA
damaging agents
<OPTION VALUE=14>Other identified genes with a suspected DNA repair function
<OPTION VALUE=15>Other conserved DNA damage response genes
<OPTION VALUE=16>All systems
</select>

<p>
<br />
</p>
<p>
<big><b>OR</b></big>

<p>
<br />
</p>
<p>


<p><b>Enter Gene name:</b></p> <input name="search" type="text" />

<p><br /></p>
<p><br /></p>
<big><input type="submit" value="Search"/></big>
</p>
</center>
</form>
<p><br /></p>

<HR WIDTH="50">
<HR WIDTH="100">
<HR WIDTH="300">
</html>

<P>
<center><I>
<TV>Made by Lise Henriksen, M.sc. student at the Department of Informatics,
University of Oslo, 2007
</TV></I>
</center>
</P>
```

9.9 taxPros.php

Retrieving taxonomy id and organism name.

```
<?php
include("opendb.php");

//correct database
mysql_select_db("lise");

$outfile = "SELECT tax_id, name FROM tax_tmp WHERE name_class = 'scientific name'
INTO OUTFILE '/users/lihe/database/taxonomy.txt' ";
$result = mysql_query($outfile)
or die(mysql_error());
echo "records exported to txt-file";

?>
```

9.10 search.php

Queries the database as specified from the homepage and sends the result to the browser.

```
<html><body>
<head>
<title>Database of Genes Involved in DNA Repair</title>
</head>
<body
BGCOLOR="#99ccff"TEXT="black"LINK="#008800"VLINK="red"ALINK="#ffff00"><center><h1>D
atabase of Genes Involved in DNA Repair</h1></center>
<HR>
</body>
<p>
<br />
</p>

<?php

include "opendb.php";

$search = $_POST['search'];
$System = $_POST['System'];


if (count($_POST['box']) == 0){
  echo "<CENTER>";
  echo "Please check one or more box(es)";
  echo "<p> <br /> </p>";
  echo "</CENTER>";
}
else if (count($_POST['box']  > 0) && $System == 0) {
  if($search != "") {
  $query = "(SELECT g.gene_id, g.homologeneid, t.organism_name, g.gene_name, ".
    "GROUP_CONCAT(s.synonym SEPARATOR ', ') AS synonyms, ".
    "g.description, g.chrom_location ".
    "FROM taxonomy t, sysname sys, gene g ".
    //LEFT JOIN since some genes do not have synonyms
    "LEFT JOIN synonym s ON (g.gene_id = s.gene_id) ".
    "WHERE g.taxid = t.tax_id ".
    "AND t.tax_id IN (".implode(",",$_POST['box']).") ".
    "AND g.sysid = sys.sys_id ".
    "AND g.gene_id IN (SELECT g.gene_id ".
```

```
    "FROM gene g ".
    "WHERE gene_name LIKE '%$search%') ".
    "GROUP BY g.gene_id) ".

    "UNION ".

    "(SELECT g.gene_id, g.homologeneid, t.organism_name, g.gene_name, ".
    "GROUP_CONCAT(s.synonym SEPARATOR ', ') AS synonyms, ".
    "g.description, g.chrom_location ".
    "FROM taxonomy t, sysname sys, gene g ".
    "LEFT JOIN synonym s ON (g.gene_id = s.gene_id) ".
    "WHERE g.taxid = t.tax_id ".
    "AND t.tax_id IN (".implode(",",$_POST['box']).") ".
    "AND g.sysid = sys.sys_id ".
    "AND g.gene_id IN (SELECT s.gene_id ".
    "FROM synonym s ".
    "WHERE synonym LIKE '%$search%') ".
    "GROUP BY g.gene_id) ".

    "UNION ".

    "(SELECT g.gene_id, g.homologeneid, t.organism_name, g.gene_name, ".
    "GROUP_CONCAT(s.synonym SEPARATOR ', ') AS synonyms, ".
    "g.description, g.chrom_location ".
    "FROM taxonomy t, sysname sys, gene g ".
    "LEFT JOIN synonym s ON (g.gene_id = s.gene_id) ".
    "WHERE g.taxid = t.tax_id ".
    "AND t.tax_id IN (".implode(",",$_POST['box']).") ".
    "AND g.sysid = sys.sys_id ".
    "AND g.gene_id IN (SELECT g.gene_id ".
    "FROM gene g ".
    "WHERE description LIKE '%$search%') ".
    "GROUP BY g.gene_id) ".
    "ORDER BY homologeneid, organism_name";


  $result = mysql_query($query) or die(mysql_error());
  $nrows = mysql_num_rows($result);

if ($nrows != 0) {
    $i = 0;
    make_table($result, $i);

  } else {
    echo "<CENTER>";
    echo "Your search did not return any results, please try again";
    echo "<p> <br /> </p>";
    echo "</CENTER>";
  }
  }
  else {
  echo "<center>";
  echo "Please choose a DNA repair system or search string";
  echo "<UL>";
  echo "<center>";
  echo "</center>";
  echo "</UL>";
  echo "</center>";
  }
}

else if ($System > 0 && !empty($_POST['box'])) {

  if($System != "" AND $System != 16 AND $System != 0) {

    //Making header for each system
    $sys = "SELECT s.sysname FROM sysname s WHERE s.sys_id = ".$System;
    $res = mysql_query($sys) or die(mysql_error());
```

```
    while ($row = mysql_fetch_assoc($res)) {
      extract($row);
      foreach($row as $value) {
        echo "<center><h2>$value</h2></center>";
      }
    }
      ?>
      <p><br /></p>
      <?php

         $query = "SELECT g.gene_id, g.homologeneid, t.organism_name, g.gene_name,
".
         "GROUP_CONCAT(s.synonym SEPARATOR ', ') AS synonyms, ".
         "g.description, g.chrom_location ".
         "FROM gene g LEFT JOIN synonym s ON g.gene_id = s.gene_id, ".
         "sysname sys, taxonomy t ".
         "WHERE g.sysid = sys.sys_id ".
         "AND g.taxid = t.tax_id ".
         "AND t.tax_id IN (".implode(",",$_POST['box']).") ".
         "AND g.sysid = $System ".
         "GROUP BY g.gene_id ".
         "ORDER BY homologeneid, organism_name";

$result = mysql_query($query) or die(mysql_error());
    $i = 0;
    make_table($result, $i);

  }

else if($System == 16) {

    $num = 1;

    while ($num <= 14) {

      $query = "SELECT g.gene_id, g.homologeneid, t.organism_name, g.gene_name, ".
        "GROUP_CONCAT(s.synonym SEPARATOR ', ') AS synonyms, ".
        "g.description, g.chrom_location ".
        "FROM gene g LEFT JOIN synonym s ON g.gene_id = s.gene_id, ".
        "taxonomy t, sysname sys ".
        "WHERE g.sysid = sys.sys_id AND g.taxid = t.tax_id ".
        "AND t.tax_id IN (".implode(",",$_POST['box']).") ".
        "AND sys.sys_id = $num ".
        "GROUP BY g.gene_id ".
        "ORDER BY homologeneid, organism_name";

      $result = mysql_query($query) or die(mysql_error());
      make_header($num);

      make_table($result, $i);
      $num = $num + 1;

    }
      ?>
      <p><br /></p>
      <?php
//If the user is not taking a choice..
}else if($search == "") {
  echo "<center>";
  echo "<UL>";
  echo "<center>";
  echo "<A HREF = 'http://dna.uio.no/lise/rep.php'>Back to search</A>";
  echo "</center>";
  echo "</UL>";
  echo "</center>";
}

}
```

```
function make_table($result, $i) {

  if ($result) {
    $nrows = mysql_num_rows($result);
    $nfields = mysql_num_fields($result);
  }

  $count = 0;
  /* Making table */
  echo "<TABLE BORDER='1' WIDTH='100%'>";
  if ($result) {
/* $count will set the value on $temp later */
    $count = NULL;
    echo"<TR>\n";
    echo "<th WIDTH='9%'>Homolog Gene Id</th>\n";
    echo "<th WIDTH='10%'>Organism</th>\n";
    echo "<th WIDTH='10%'>Official Name</th>\n";
    echo "<th WIDTH='28%'>Synonym(s)</th>\n";
    echo "<th WIDTH='30%'>Description</th>\n";
    echo "<th WIDTH='13%'>Chromosome Location</th>\n";
    echo "</TR>";

for (;$i<$nrows;$i++) {
      $rowarr = mysql_fetch_row($result);
      for ($j=1;$j<$nfields;$j++) {
        $val = $rowarr[$j];
        $count++;
        if ($count == 1) {
          /* $temp keeps track of the homologs */
          $temp = $val;
        }
        if ($val == "") {
          $val = " ";
        }
        echo "<TD>";
        /* homologeneid is written when $j=1 */
        if ($j == 1) {
          /* When $temp == $val homologs are written in groups */
          if ($temp == $val) {
            echo "<CENTER><a
href=\"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=homologene&cmd=search&term=
" . $val . "\">$val</a></CENTER>";
          }else if ($temp != $val) {
            table_space();
            table_border($val);
            $temp = $val;
          }
        }
else if ($j == 2) {
          echo "<BR>";
          $query = "SELECT g.taxid FROM gene g, taxonomy t WHERE t.organism_name =
'$val' AND g.taxid = t.tax_id";
          $res = mysql_query($query) or die(mysql_error());
          $row = mysql_fetch_array($res);
          extract($row);
          echo "<center><a
href=\"http://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?mode=Info&id=" .
$row[taxid] . "&lvl=3&lin=f&keep=1&srchmode=1&unl\
ock\">$val</a></center>";
        }
        else if ($j == 3) {
          echo "<center><a
href=\"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene&cmd=Retrieve&dopt=full
_report&list_uids=" . $rowarr[0] . "\">$val</a\
></center>";
        }
        else if ($j == 6){
```

```
          echo "<center>$val</center>";
        }
        else {
          echo "$val";
        }
        echo "</TD>";
      }
      echo "</TR>";
    }
  }//if($result)
  if ($result) mysql_free_result($result);
  echo "</TABLE>";
  echo "<p>";
  echo "<br />";
  echo "</p>";

}

function table_border($val) {
  echo "<TABLE BORDER='1' width='100%'><TR>";
  echo "<th width='9%'></th>";
  echo "<th width='10%'></th>";
  echo "<th width='10%'></th>";
  echo "<th width='28%'></th>";
  echo "<th width='30%'></th>";
  echo "<th width='13%'></th>";
  echo "</TR><TD>";
  echo "<CENTER><width='9%'><a
href=\"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=homologene&cmd=search&term=
" . $val . "\">$val</a></CENTER>";
}

function table_space() {
  echo "</TD></TR></TABLE>";
  echo "<p><br /></p>";
}

echo"<p> <br /> </p>";
echo"<center>";
echo "<A HREF = 'http://dna.uio.no/lise/database/rep.php'>Back to search</A>";
echo "</center>";



//if all systems are picked, this function makes a header for each of them
function make_header($num) {

  $sys = "SELECT s.sysname FROM sysname s WHERE s.sys_id = ".$num;
  $res = mysql_query($sys) or die(mysql_error());
  while ($row = mysql_fetch_assoc($res)) {
    extract($row);
    foreach($row as $value) {
      echo "<center><h2>$value</h2></center>";
    }

  }
}


?>
</html>
```