Development of a Laboratory Information Management System for Medical Genetic Investigations (LIMS)

Katharina Albers

THESIS

In partial fulfilment of the requirements for the degree Bachelor of Science in Bioinformatics

Free University Berlin

Department of Mathematics and Computer Sciences

Advisors:

Dr. Annika Hinze

Dr. Andreas Kuß

Max Planck Institute

for Molecular Genetics

Ihnestraße 63-73

14195 Berlin

June 26th - August 17th 2006

Abstract

Studying the genetic factor underlying a set of diseases with modern high-throughput techniques generates huge amounts of data, posing a challenge for data management. In this thesis a database management system called FIDB based on MySQL was developed to handle clinical and experimental genetic data. For the convenience of the users, a web interface was developed to insert, update, delete and display the data. In addition security aspects were taken care of.
Currently FIDB is able to organise and store data gathered by analysing 150 consanguineous families with autosomal recessive mental retardation.

Abriss

Das Erforschen von vererbbaren Krankheiten mit modernen Techniken und hoher Durchsatzleistung produziert sehr große Mengen Daten und stellt eine Herausforderung an die Datenverwaltung dar. In dieser Arbeit wurde das Datenbankmanagementsystem FIDB, basierend auf MySQL, entwickelt, um klinische und experimentelle genetische Daten handzuhaben. Um FIDB benutzerfreundlich zu machen, wurde ein web interface entwickelt, um die Daten einzutragen, zu aktualisieren und zu löschen und die Datenbasis anzuzeigen. Zusätzlich wurden Sicherheitsaspekte bedacht.
Gegenwärtig organisiert und speichert FIDB Daten, die bei der Analyse von ca. 150 consanguinen Familien mit autosomal rezessiver geistiger Behinderung gesammelt wurden.

Contents

1 Motivation

Since the gene was first proposed as the unit determining inheritance in all living things, the curiosity about the concepts of life drove many scientists to research in this field. It was found out about 40 years later that DNA is the material holding the inherited information. And only 2001, another 50 years later, the whole human genome was read, now giving the sequence of the about 3,000,000,000 base pairs composing the DNA, but still leaving open lots of questions.
In the whole genome there are about 25,000 genes, some of them being investigated quite well, but most largely unknown in their mechanisms and effects. A good way to find the line of action of genes is the investigation of diseases, caused by mutations in certain genes.
Nowadays, with the emergence of modern techniques, it is possible to systematically study the genetic factor underlying a set of diseases. This in turn will generate huge amounts of data, which are impossible to store in the traditional way, where various information is separately stored in different files. This posed challenge can be solved by a database management system, which allows not only storing, but also updating and querying the data in a convenient and consistent way.

The aim of this work was to develop a database management system to store, organise and search through clinical and experimental data, called FIDB (Family Information Database). At present FIDB is used to store information about autosomal recessive mental retardation in consanguineous families[1], gathered by the Research Group Familial Cognitive Disorders of the Department Ropers within the Max Planck Institute for Molecular Genetics. Later other groups within the department or collaborating groups might contribute their data to produce a broader view on the aetiology of mental disorders.

[1] *Mental retardation* is often defined as the Intetelligence quotient (IQ) being below 70 (WHO, 1980). *Autosomes* are the non-sex chromosomes, of which there are 22 in humans. *Recessive* means that an inherited mutation only causes a disease when both autosomes - in humans all the autosomes exist twice - are concerned.

2 Data Collection

The data used in this project are medical data concerning families with inherited mental retardation. Most of the families investigated by the group Familial Cognitive Disorders are nonsyndromic mentally retarded. That means, apart from being mentally retarded the affected family members do not have other clinical features and cannot otherwise be distinguished from healthy individuals. However, as clinicians may become more aware of subtle features, it is often very useful to have pictures and a thorough description of the patients and their development.

That the disorder is inherited strongly suggests that a gene defect is responsible for that observed phenotype[2]. In order to elucidate such underlying gene defects, a strategy called positional cloning[3] is applied as it is still impossible to sequence the whole genome of affected individuals in a convenient time and money frame.

The strategy of positional cloning is to map the location of a human disease gene first and then use the mapped location on the chromosome to clone the gene.

In order to map the chromosomal location the genotype of several affected and unaffected family members has to be known. A genotype is the specific genetic composition of the genome of an individual and is often inferred from sequence analysis of small polymorphic DNA segments, called markers, which are evenly spread over the entire genome. In this project 10,000 single nucleotide polymorphisms (SNPs) were applied as markers and have been analysed for each individual.

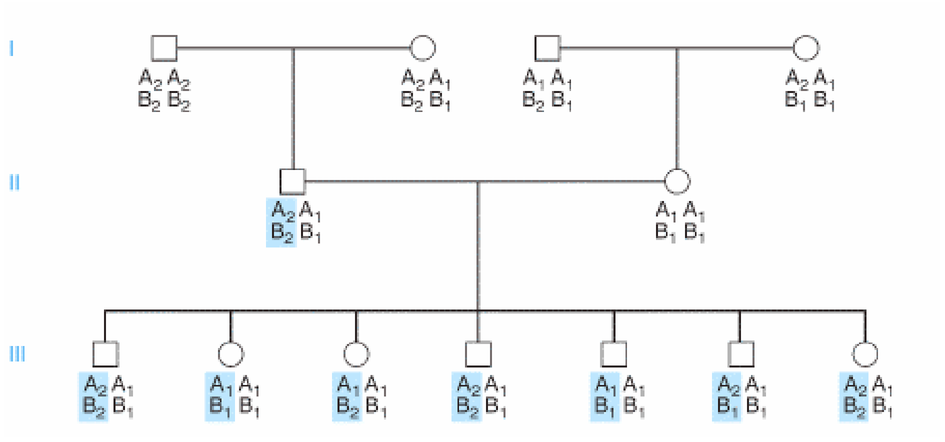


Figure 2.1: Pedigree with two markers.

[2] The *phenotype* is the physical appearance of an individual.
[3] *Positional cloning* means cloning a gene knowing only its chromosomal location (Strachan, 2004).

During meiosis (production of egg and sperm cells) in a process called recombination, fragments of genomic DNA are exchanged between parental chromosomes. When two markers are located close, they are very often inherited together because the recombination or crossing-over is a random process and very unlikely to happen between these markers located very close to each other. On the other hand, markers located further away or on different chromosomes are not necessarily inherited together. This phenomenon is taken advantage of to search for the markers inherited together with diseases, i.e. the markers located close to the gene carrying the disease causing mutation. Usually this task is fulfilled by a statistical method called linkage analysis, where the likelihood of any specific genomic region to carry a disease causing mutation can be calculated.

The intervals with a high probability to carry a mutation and also not being too big are then investigated further: a genomic database is searched for genes in the interval, which are then prioritised with different criteria to estimate. One criterion is for example, where and when the gene is expressed. Expression in the brain or during development indicates a gene likely to cause mental retardation.

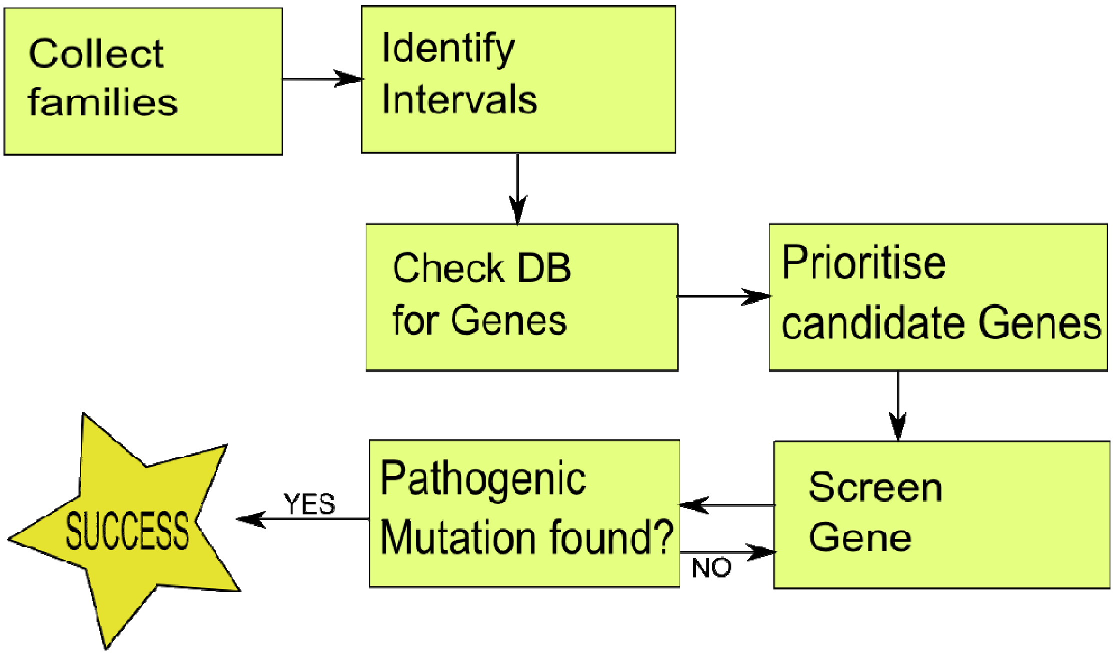


Figure 2.2: Data Collection. Every family is checked for intervals with a high probability to carry a mutation. The genes in the intervals are then prioritised and screened for a pathogenic mutation.

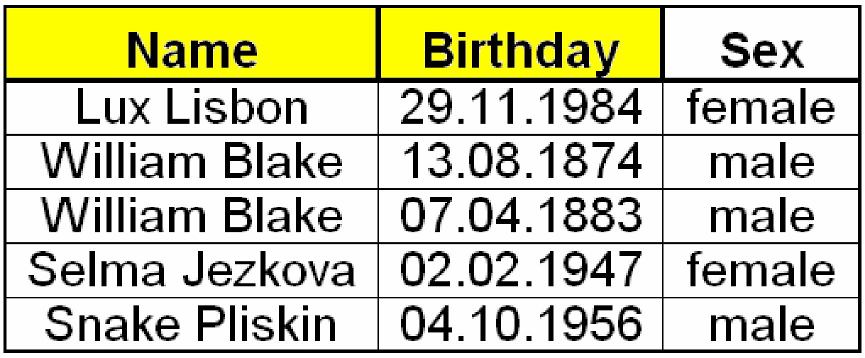
The genes rated promising are then sequenced and compared among the family members for cosegregation, which denotes in this case whether a variation is inherited together with the illness. Any variation found is then tested in unrelated healthy individuals. If it is also found there, it is most likely a polymorphism. Otherwise a variety of assays can be designed, to test the functional relevance of this mutation. If the screening of the first gene was unsuccessful and did not reveal a pathogenic mutation, the next gene has to be screened. And so forth until hopefully something interesting is found.

In the analysis of about 150 consanguineous families, being genotyped by 10,000 Affymetrix chips, about 400 intervals were found after two years of linkage analysis. By now about 150 genes have been screened, all this generating a huge amount of data. These data used to be stored in lots of different files, impossible to search through and keep updated. In addition, results of different people working on this matter were not comparable in a convenient way. A database management system storing data in a standardised and consistent way, which makes them comparable, searchable and easily updateable, is in need.

3 Database

The development of a database management system (DBMS) makes sense, as soon as one has to deal with big and cross linked amounts of data. The use of a DBMS asserts that data from different users in different places can be inserted in a standardised format. The organisation of the stored data, the database schema, allows for a sensible combination of the inserted information.
The data stored in a database are called the data basis, while the database management system is the whole of the programs, manipulating and using the inserted data. Sometimes the DBMS is called database system.

There are different database models, forming the logical level of a DBMS. The most widely used is the relational data model, which pools the data in two-dimensional tables, called relations (Korth, 1991). Every column of a table contains data of the same type, while the rows contain data belonging to one object. Such a table could for example consist of three columns, holding the name, birthday and sex of a person, whereas a row contains the information about one person. The columns are addressed by their names, called attribute names. The rows are addressed by the value of key attributes, which characterise the object unambiguously. In the example the key could be the combination of name and birthday (with the name one could not be sure that only one person is called like that, while one can be pretty sure that no two persons with the same name are also born on the same day). One could find the information about the sex of a person by entering the name and birthday of this person.



Name	Birthday	Sex
Lux Lisbon	29.11.1984	female
William Blake	13.08.1874	male
William Blake	07.04.1883	male
Selma Jezkova	02.02.1947	female
Snake Pliskin	04.10.1956	male

Figure 3.1: Table of a relational database. Name and birthday build the primary key; the rows could not be distinguished by the name alone.

The attributes of a table one chose to be the defining ones are called the primary key. By referencing the primary key in another table, one can establish a connection between these different tables. In the referencing table the primary key of the referenced table is called the foreign key.
Some database systems also support the definition of constraints[4], which have to be fulfilled, so that the database is in a consistent[5] state. If alterations violate these constraints, the system would reject these changes (Codd, 1985).

To avoid redundancies and anomalies, the database schema can be checked for the compliance with normal forms, which make sure that anomalies cannot occur through updating, inserting or deleting data. The relations that do not conform to the normal forms can in most cases be decomposed into more tables, on condition that this can be done preserving the information and dependencies (Kemper, 2001).

A query language builds the interface between the database system and the user, whereby the existing tables can be linked and displayed as newly combined tables (Codd, 1970). The standardised query language (SQL) is supported by almost every database system. SQL has a comparatively simple syntax and in spite of the name "query language" provides not only commands to query the data basis, but also commands to manipulate (insert, update and delete) the data and to build the relations (Matthiessen, 2000). Despite the standardisation of SQL, different database systems differ in complex queries.

The deployed database system is MySQL 4.1.21 (http://dev.mysql.com). MySQL is a very popular open source database system, is easy to set up and administer, with high performance (DuBois, 2000). It was first released in 1995 and has been continuously enhanced.
MySQL supports SQL and most of the integrity constraints, like primary key and foreign key. The databases in this system can be accessed from more than one person simultaneously, so that the data can be shared. At the same time MySQL provides data security through access control.
Seeing all this, MySQL is a suitable system for FIDB.

---

[4] A *constraint* is a condition that can be defined; in the earlier example a constraint would be that no two persons with the same name and the same birthday can be inserted.
[5] *Consistency* means that the database contains no contradictions, or more precisely that the DB is accurate, correct and valid.

4 Web Interface

Most database systems offer a simple user interface allowing entering of SQL-statements. An easier to use interface needs to be able to translate actions to SQL statements specific to the database schema, thereby allowing users who are not familiar with SQL to manipulate and display the data.

There are two options to build the interface: an autonomous software packet or a web interface. The software packet has to be installed on the user's computer and is called two-tier architecture, because there are only two layers: the DBMS and the program packet on the client computer (Figure 4.1). In this case a connection between the database server and the client computer has to be established.

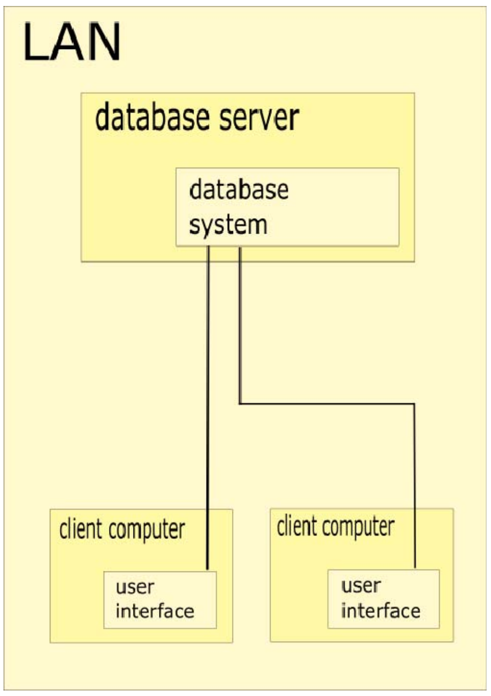


Figure 4.1: Two-tier Architecture. The database system is directly addressed by the client computer.

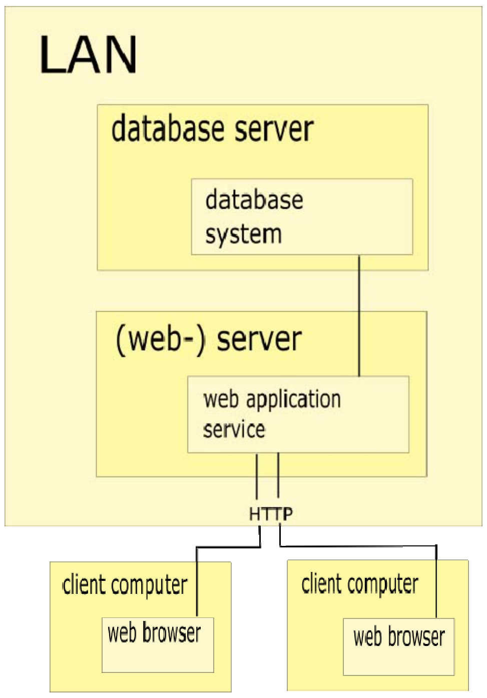


Figure 4.2: Three-tier Architecture. The database system is addressed via a web server, so that the user can utilise the interface by a browser.

If a web interface is developed, the users can access via a network and only need to have a browser installed on their computer. This is called three-tier architecture, because of the three layers of the application, namely the database system, the (web-) server and the client computers (Figure 4.2). For FIDB a web interface was chosen because it is more convenient for the users, than the first option. The complicated implementation of an interface for a web browser is countervailed by the independence from an installed user interface.

To develop the web interface, Java (http://www.sun.com/java) was used. Java is an object-oriented and platform independent programming language. To run the programs, the written code is compiled to Java-byte code and then interpreted and executed by a Java virtual machine. This virtual machine is specific to the actual platform and provides a virtual computer for the Java application, so that every program written in Java can be run on every platform. Java was chosen for FIDB development, because it has very convenient implementations for our purpose, including Java-Servlets, JSPs (Java Server Pages), JavaBeans and JDBC (Java Database Connectivity).

Java-Servlets are programs that run on a web server and build web pages. That means that instead of having a static web page, the web pages are built on the fly. This is very useful for pages with changing content and consequently for database systems, which naturally change a lot.
JSPs allow including Java code in the static contents of web pages. Thus the design and the logic of implementation can be kept apart. To execute the JSPs, a JSP compiler compiles the JSPs into Java Servlets (Hall, 2003).
The difference between JSPs and servlets is substantially that servlets are Java programs with a HTML page as output, while JSPs are HTML pages with embedded Java code. JSPs facilitate the development of the design, while servlets facilitate the implementation of extensive program logic.
I used the JSPs mainly to write forms to display, insert and change the data, while the servlets interact with the database.

JavaBeans are software components used as containers for data transfer. In web applications they allow to encapsulate the logic behind it and remove the bulk of the scriptlet code that would otherwise clutter up the JSPs (Avedal, 2000).

With JDBC Java provides an interface to the relational database systems of different producers. This API (application programming interface) provides methods for establishing and managing connections to a database, as well as querying and updating data in a database. Querying and updating are achieved by forwarding SQL statements to the database and changing the outcome to a form usable in Java (Avedal, 2000). In FIDB a MySQL interface of JDBC was used.

In the web pages I also used Java Script to generate adequate alerts, for example when the fields in a JSP form are not filled in correctly. Java Script – despite the name – does not have a lot in common with Java (the only common ground being their debt to the C programming language syntax). It is a prototype based scripting language and can be used in different host environment applications: the best known being web technologies. Here it is used to write functions that are embedded in HTML pages – or in this case JSPs - to perform tasks not being possible in HTML alone. One can, for example, check if the user submitted a name and a password on a login page and alert him if he did not.

The web application server used in FIDB is Tomcat 5.5.17 (http://tomcat.apache.org) of the Apache Jakarta Project, because it supports the Java-Servlets and JSP technology and is convenient in its functions and features. Besides it is an open source program, which might be important for further development.

5 Database Development of FIDB

5.1 Conceptual Design

As pictured in Figure 5.1, the first step in designing a database model is the conceptual design, where the essential information of the "real world" is moulded. This part of the database development does not depend on the specific database system and not even on the database type. One could transfer this semantic specification not only to a relational data model, but also to a network, hierarchical or object-oriented data model (Kemper, 2001).

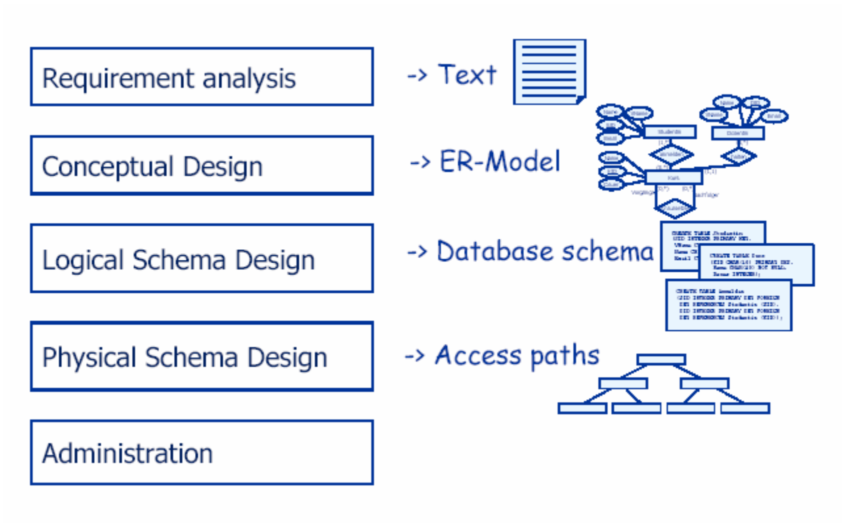


Figure 5.1: Steps in designing a database. After the requirement analysis, the conceptual design is developed. Based on this, a logical schema is generated. The last step in setting up the database is the physical design. After it is set up, it needs administration. (Figure taken from slides of the lecture "Einführung in die Datenbanksysteme", SS06, FU Berlin)

A widely used concept to model the conceptual design is the entity-relationship-model, where entities and relationships form the basis of the scheme (Chen, 1976). The entity-relationship model allows the description of a conceptual scheme to be written down without attention to efficiency or physical design (Ullman, 1988). It is not suitable for straight implementation, but is a good method to describe the regularities of the "real world".

In the entity-relationship model, as shown in Figure 5.2, entities are objects and are represented by rectangles. Relationships are depicted by rhombuses which relate two entities by connecting them. Additionally there are attributes and recursive relationships. The attributes are represented by ovals, characterise entities or relationships and are connected to entities or relations (Kemper, 2001). Recursive relationships are diagrammed by a relationship of an entity with itself where role names are added to the connecting lines.

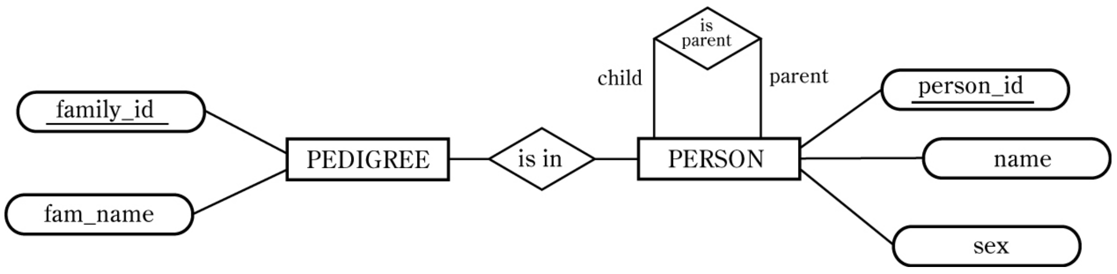


Figure 5.2: Entity-relationship model. The entities *pedigree* and *person* are connected by the relationship *is_in*. Pedigree has the attributes *family_id* and *fam_name* with *family_id* being the primary key. *Person* has not only attributes, but also a recursive relationship *is_parent* with the role names *child* and *parent*.

Supplementary to the entities and relationships, one can give the degree of a relationship, which is the number of entities associated in a relationship (Teorey, 1999) and is called cardinality. In the min-max notation, this is given by two characters for every entity, the first one denoting the minimum and the second one the maximum of possible relations with the related entity (see Figure 5.3).

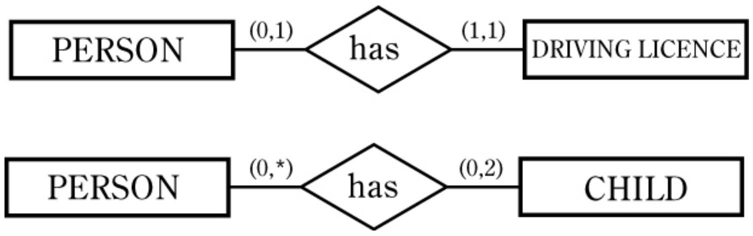


Figure 5.3: Cardinalities. Every *person* has a *driving license* or has no *driving licence*; a *driving licence* belongs to exactly one *person*. Every *child* has zero to two parents and a *person* can have every number of *children*.

With these definitions and concepts, an entity-relationship model with the cardinalities for FIDB is shown in Figure 5.4 and explained in the following sections.

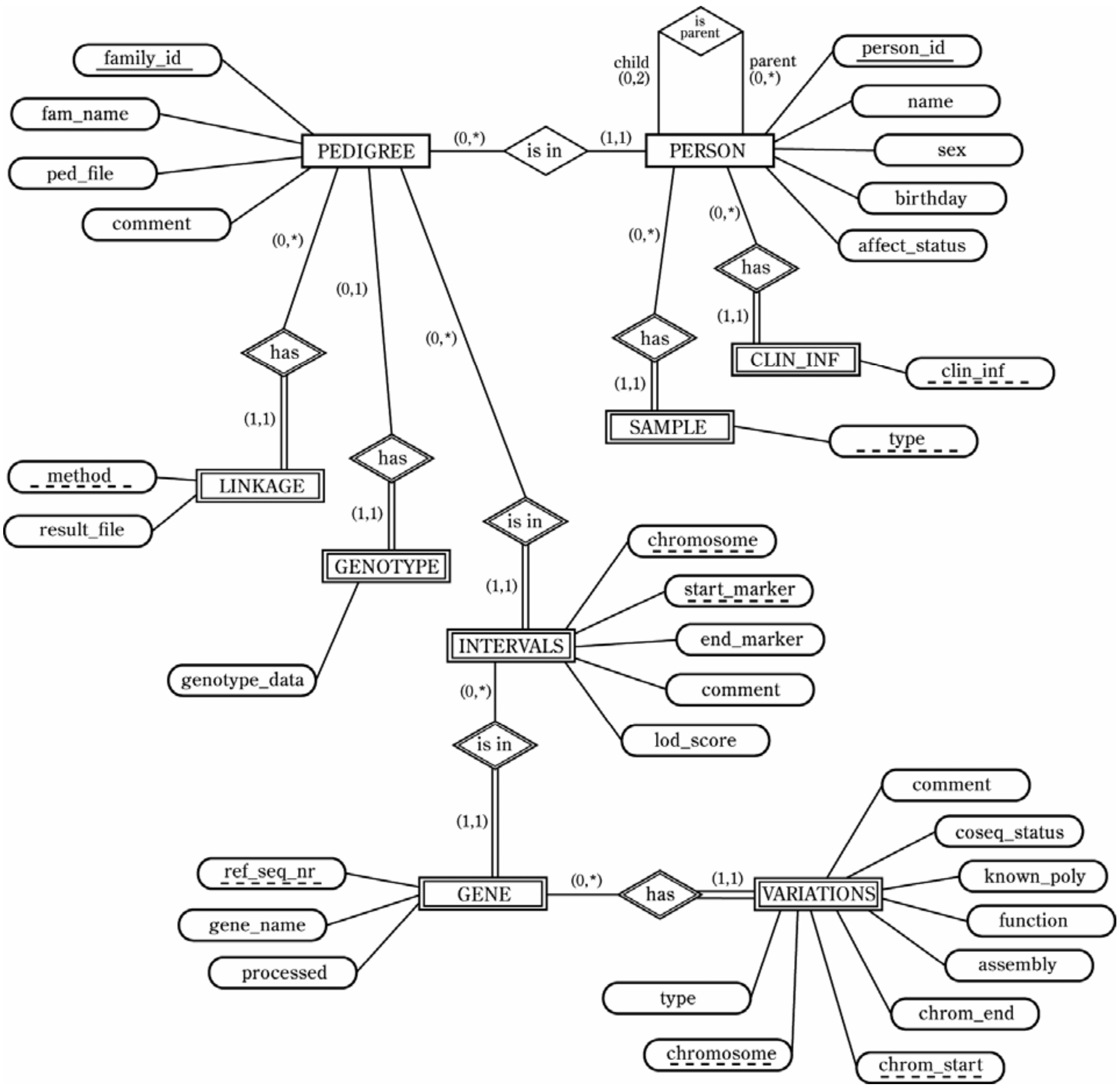


Figure 5.4: The entity-relationship model of FIDB.

Pedigree: The central entity in the database schema is the *pedigree*. All the other entities are linked to this one, even though some are not directly, but through another entity. The attributes *family_id, fam_name, ped_file* and *comment* belong to this entity. The *family_id* was given to every family when it was filed and is used in the laboratory to identify the families. So this attribute is a perfect primary key. The *ped_file* is a pedigree file, which holds information about the pedigree and is the input file for the linkage analysis. *Fam_name* and *comment* are self-explanatory.

Person: In every family there are *persons*, who also have an ID (*person_id*) and a *name*. Additional attributes include *sex, birthday* and *affect_status*; the *affect status* specifies if someone is mentally retarded.
The role *is_parent* can model a parent-child relationship.

Clin_inf: The affected *persons* have different symptoms, which are described in the entity *clin_inf* with only one attribute also called *clin_inf*. The clinical information was set as a separate entity, because otherwise one could not put in every quantity, but only a fixed number of symptoms for every patient.
This entity is weak, which is denoted by the double lines and means that it cannot be identified by its own key alone, but also needs the primary key of the entity it is linked to. In this case many persons can have the same symptoms, but every person has the same symptom only once. A weak entity must have the cardinality (1,1), because otherwise it would not be assignable.

Sample: For the entity *sample* it is the same as in *clin_inf*: to insert more than one, a new entity is introduced. The *type* could for example be “blood”, “DNA” or “cell line”.
Sample is also a weak entity.

Linkage: Three weak entities, namely *linkage, genotype* and *intervals* depend on the entity *pedigree*. The *linkage* has got a *method* and a *result_file*, where the *method* is the used program and setting (parametric or nonparametric) and the *result_file* is the output of the used linkage analysis program. Because every program is used with the same setting only once, the *method* is the weak key.

Genotype: The *genotype* does not need its own key, because genotyping is done only once for every family.

Intervals: The *intervals* are the ones that were found to be disease associated in the linkage analysis. They are defined by the *chromosome, a start* and an *end marker*. Additonally, there are the attributes *lod score*[6] and *comment*. The *chromosome* and the *start marker* serve as a weak key, because no two different intervals in the same family can start at the same position.

[6] The *lod score* is used to calculate the probability of a pedigree arising randomly or by genetic linkage. ***LOD* = log ((probability of birth sequence with a given linkage value) / (probability of birth sequence with no linkage))**

Gene: In the intervals, there are genes that were screened for disease causing mutations. These genes are identified by the RefSeq ID provided by the NCBI Reference Sequence project. Additionally the HUGO (Human Genome Organisation) gene name and the number of exons already screened are stored.

Variations: If something unusual was found in a screened *gene*, it is stored as the entity *variations*. A *variation* is identified by a *chromosome*, a *start* and an *end*. *Start* and *end* in this case are not denoted by markers, but instead by the base pair position on the chromosome. Since this information can differ in different versions of the human genome assembly, the *assembly* also has to be specified. More information about the variation is *type, function, known polymorphisms* and the *cosegregate status*. The *type* is the type of the variation, which might for example be an insertion or a deletion. The attribute *function* holds information about the function of the intact DNA, such as exon or intron. A *known polymorphism* means a variation already known to occur in the normal population. *Cosegregate status* means in this case, whether the variation and the illness are inherited together. For flexibility one can also add a *comment*.

5.2 Relational Design

To make use of a relational database system like MySQL, one has to generate a relational model, which models the entities and relationships in relations. The relations are then easily converted to tables, forming the relational database.

To get a relational model, one has to transfer the entity-relationship-model to relations. This is done by first converting each entity with all attributes to a relation. The key attributes of the entity also become the primary key of the relation. To transform a weak entity, the attributes of the weak entity and the key attributes of the identifying entity are included. Here the key attributes of both entities form the primary key.
Secondly the relationships are transformed: all the key attributes of the related entities are included. Which of these are picked to form the primary key depends on

the cardinality. The relationships of the weak entities were already transformed by including all the keys of the involved entities.
Recursive relationships are converted by renaming the primary keys, which have to be included twice. For example the recursive relationship is_parent could be converted to a relation with the attributes parent_id and child_id, each being a person_id.
In the last step, the obtained relational schema is simplified, whereby relations with the same key might be collected into one relation.

Below I will list the relations, but only describe the changes that had to be carried out during the transformation, as the entities and the attributes were described sufficiently in the previous part.

```
pedigree: {[    family_id: string,
                fam_name: string,
                ped_file: string,
                genotype_data: string,
                comment: text ]}
```

The curly brackets show, that a relation is a set of tuples. In the squared brackets the attributes the tuples are composed of are given. The type of the attribute is specified after the colons: string for a short text and text for a long text.
In the relation pedigree, genotype_data, being modelled in an own entity before, was added to the attributes, because for every family there is exactly one such file.
As in the some other tables, there is an attribute comment in pedigree. Since often nothing is inserted in this field of the table, one could put this attribute in an additional relation, to save disk space. In FIDB, I decided not to do this for all of the concerned tables, as the current data already contains many comments and also without splitting the tables, it is avoided to save the key a second time.

The relation person has got the foreign key family_id, referencing the relation pedigree. This replaces an extra relation for the relationship between these two. within_fam_id is the ID that one person has inside his pedigree. This makes it easier to associate the person in a picture of the corresponding pedigree and might later be used for some additional programs. The role is_parent is not modelled in this schema; instead the IDs of father and mother are inserted. This is the better choice, because in some cases one might only want to insert the affected person but not its parents. Modelled in a recursive relationship, the parents needed to be entered in an own row in the table person.

person: {[**person_id**: string,
within_fam_id: string,
family_id: string,
name: string,
sex: string,
birthday: date,
affect_status: boolean,
father: string,
mother: string]}

The domain of sex is given by the set
dom(sex): {'f', 'm'},
meaning that only these values are allowed to put into the table.
The type date describes a date; boolean allows only the values 'yes' or 'no' to be inserted.

clin_inf: {[**person_id**: string,
clin_inf: string,
comment: text]}

sample: {[**person_id**: string,
type: string]}

The weak entities clin_inf and sample are transformed in the usual way, where person_id references person.
So is the weak entity linkage, which is dependent on the relation pedigree and references pedigree with the foreign key family_id.

linkage: {[**family_id**: string,
method: string,
result_files: string]}

The entities intervals, gene and variations were changed quite a lot.
In intervals, an artificial key was introduced, because otherwise the primary key would consist of family_id, chromosome and start_marker. This would be cumbersome when the interval was referenced in gene and variations. With the key interval_id, the referencing is greatly simplified.

Because of this change, the primary key of the relation gene does not consist of four attributes, but only of ref_seq_nr and interval_id.

Another artificial key is introduced for variations. In this case it also replaces four attributes (ref_seq_nr, interval_id, chromosome and chrom_start) and makes the relation much easier to handle.

These artificial keys change both the table intervals and the table variations to entities that are not weak.

The assigning of the keys is later handled in the layer of the web interface, because MySQL does not cover the option "auto_increment" in a convenient way (http://sql-info.de/mysql/gotchas.html).

intervals: {[**interval_id**: int,
family_id: string,
chromosome: string,
start_marker: string,
end_marker: string,
lod_score: float,
comment: text]}

gene: {[**ref_seq_nr**: string,
interval_id: int,
gene_name: string,
processed: string]}

variations: {[**var_id**: int,
ref_seq_nr: string,
interval_id: int,
type: string,
chromosome: string,
chrom_start: int,
chrom_end: int,
assembly: string,
function: string,
known_poly: string,
coseg_status: string,
misc: text]}

dom(type): {'single', 'microsatellite', 'in-del', 'insertion', 'deletion'}

dom(function): {'unknown', 'locus', 'coding', 'coding-synon', 'coding-nonsynon', 'untranslated', 'intron', 'splice-site', 'cds-reference'}

dom(known_poly): {'unknown', 'yes', 'no'}

dom(coseg_status): {'unknown', 'yes', 'no'}

5.3 Tables and Constraints

The relational schema discussed in the previous part is converted to a relational database schema. The tables are created by the `CREATE TABLE` statement, such as

```
CREATE TABLE pedigree (
family_id        VARCHAR(20)      NOT NULL,
fam_name         VARCHAR(255)     NOT NULL default '',
ped_file         VARCHAR(255)     NOT NULL default '',
genotype_data    VARCHAR(255)     NOT NULL default '',
comment          TEXT,
PRIMARY KEY      (family_id)
);
```

Here, after specifying the table name, the attributes with their types are given in brackets. The text type in MySQL is `VARCHAR`, the maximum length of the stored string is defined in the brackets behind this type. `VARCHAR` can at most be 255 characters long. `TEXT` is also a text type, but can be much longer: at most 65,535 characters.

`NOT NULL` is an integrity constraint and ensures that no tuples without a value are inserted. The `default` value will be inserted in case no value is specified, so in these columns a default value is assigned to the attributes, being an implicit NULL-value. By defining these default values it was later easier to program the JSPs and servlets controlling the insertion of the data.

The primary key is specified at the end of the table definition, as are the foreign keys. Both key constraints can be named, but in MySQL this does not affect the error message. So I did this only for the foreign keys.

```
CREATE TABLE person (
person_id        VARCHAR(20)            NOT NULL,
within_fam_id    VARCHAR(10)            NOT NULL default '',
family_id        VARCHAR(20)            NOT NULL,
name             VARCHAR(255)           NOT NULL default '',
sex              ENUM('f', 'm'),
birthday         DATE,
affect_status    BOOL,
father           VARCHAR(20)            NOT NULL default '',
mother           VARCHAR(20)            NOT NULL default '',
PRIMARY KEY      (person_id),
CONSTRAINT       family_exists
FOREIGN KEY      (family_id)
REFERENCES       pedigree(family_id)    ON DELETE CASCADE
);
```

`ENUM` is a type that only allows inserting values specified in the brackets and is able to define the domains given earlier. `DATE` is a date type, in My SQL written yyyy-mm-dd, for example 2006-08-14. `BOOL` denotes a Boolean, meaning that this type can either be yes or no, or in the case of MySQL '0' or '1'.

`ON DELETE CASCADE` causes the row with the reference to be deleted, when the referenced entry is deleted. Another option would be `ON DELETE SET NULL`, but we do not allow `NULL`-values for foreign keys. If the `ON DELETE` instruction is omitted, one is not allowed to delete any referenced value as long as a foreign key referring to this row exists (http://dev.mysql.com/doc/refman/4.1/en/innodb-foreign-key-constraints.html).

I will not display more `create table` statements here, because they do not differ much from the relations in the relational schema. The whole schema in form of the SQL statements is provided in the appendix.

6 Web Interface Development of FIDB

To develop the web interface of FIDB I used Java servlets, Java server pages (JSPs), JavaBeans and a Java class without interaction with the hypertext transfer protocol (HTTP). Both servlets and JSPs are used to generate current HTML pages, which conform to the actual demands. The web server forwards these pages to the web browser of the client computer, where they are displayed.
The JavaBeans are classes with the same attributes as the relations and are used to hold the data to be displayed. The Java class without interaction with the HTTP is used to set up a connection pool.

6.1 Database Connection

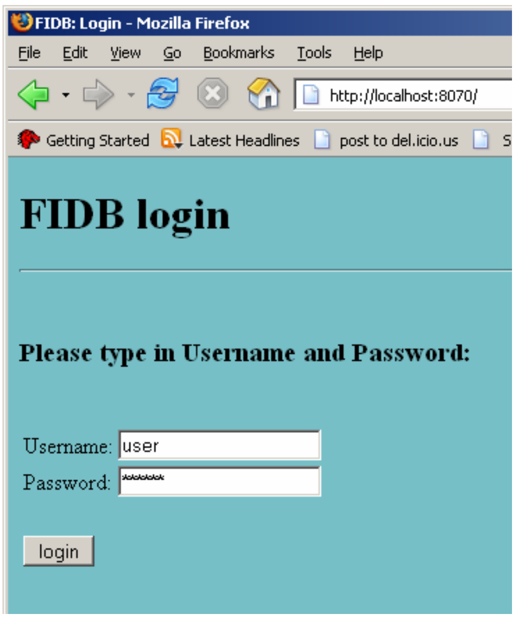
A Java class without interaction with the HTTP establishes the connections to the database. Together with settings in Tomcat's context.xml and server.xml, a connection pool is configured. Connection pooling is a technique used for sharing server resources among requesting clients; instead of establishing a new connection for every user, established connections are kept and allocated according to the demand.

6.2 Data Security

To keep the data basis secure, there are two groups of users: one is only allowed to read and search the data and the other one is also allowed to insert, delete and update data. To protect data privacy, the data basis is not accessible from outside of the institute, and users within the institute are assigned by the FIDB administrator to the appropriate group.

FORM based authentication, provided by Tomcat was chosen to handle the authentication of users. It allows for the administrator to write a login page (see Figure 6.1 and 6.2) and to define roles, complying with the groups specified in FIDB. The usernames, passwords and roles are stored in a separate MySQL database of FIDB.

The roles and their authorisation are specified in the web.xml. To get usernames, passwords and roles, a connection to the database is established in the server.xml. With this technique, both the username and the role are retrievable as long as the user is logged in.

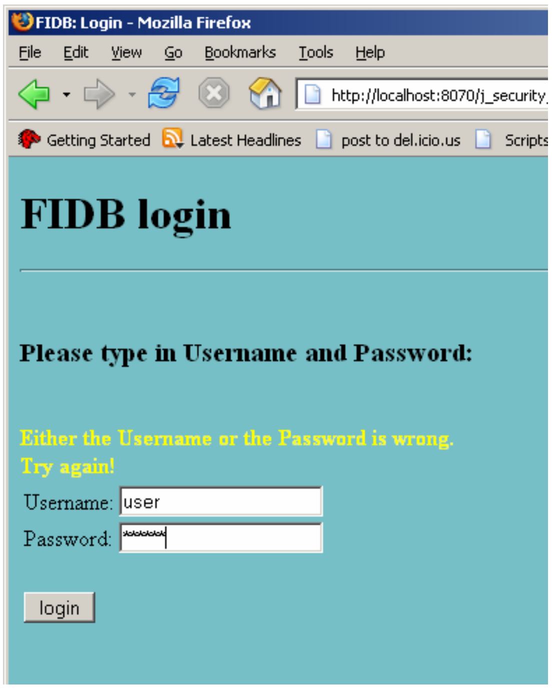

FIDB: Login - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/
Getting Started Latest Headlines post to del.icio.us
login

FIDB login

Please type in Username and Password:

Username: user
Password: ******

Figure 6.1: Login page.


FIDB: Login - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/j_security
Getting Started Latest Headlines post to del.icio.us Scripts
login

FIDB login

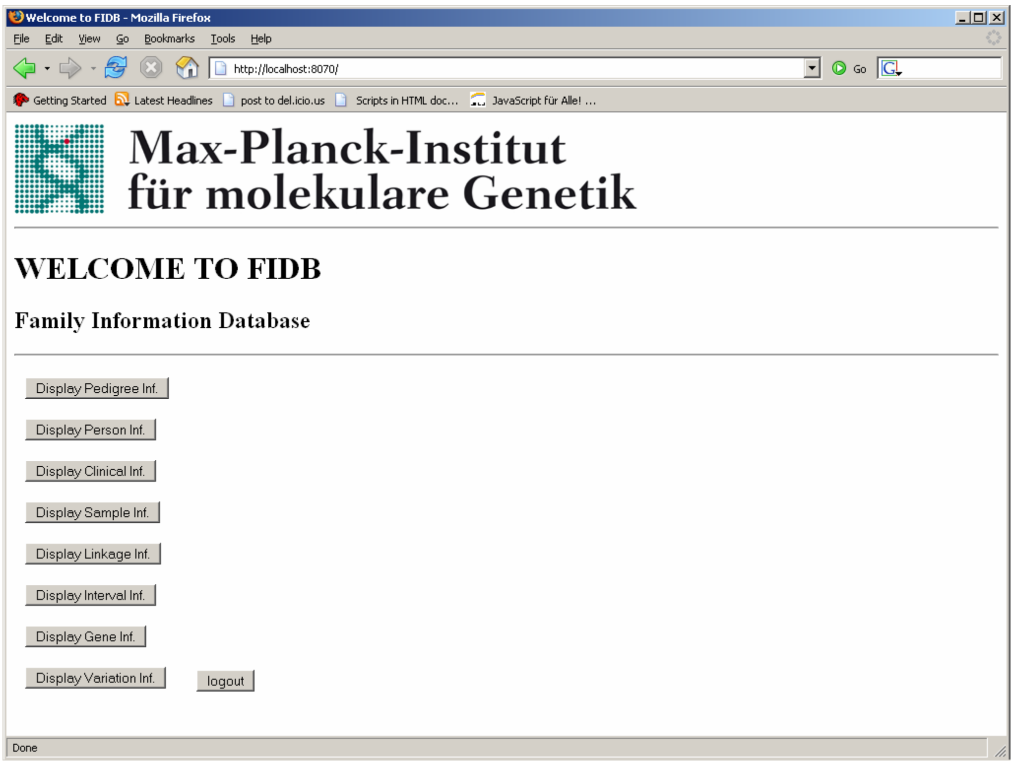
Please type in Username and Password:

Either the Username or the Password is wrong.
Try again!

Username: user
Password: ******

Figure 6.2: Login page after a failed try.

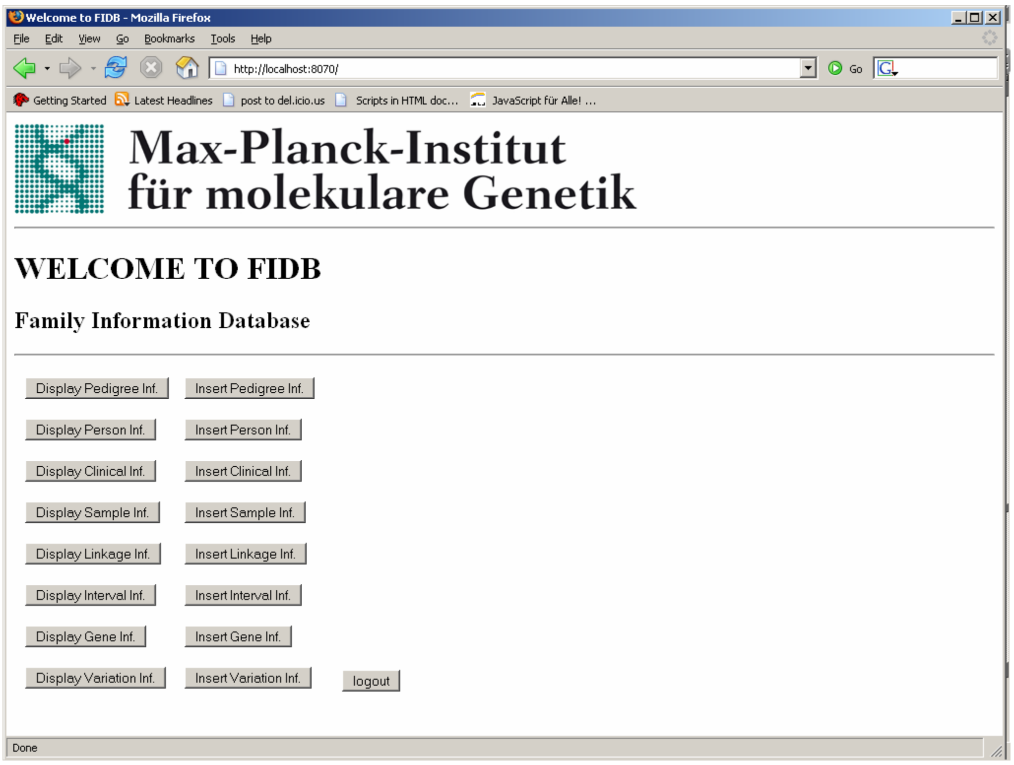
After login the user's role determines the pages the user is allowed to view. Additionally for the users with different roles, the same pages can be displayed with a different content. I used this to make sure, that the users only allowed to display the data basis, will not see any links to pages they are not allowed to use. For example the index page is different for the two groups (see Figure 6.3 and 6.4).



WELCOME TO FIDB

Family Information Database

Figure 6.3: Index page for users not allowed updating FIDB.



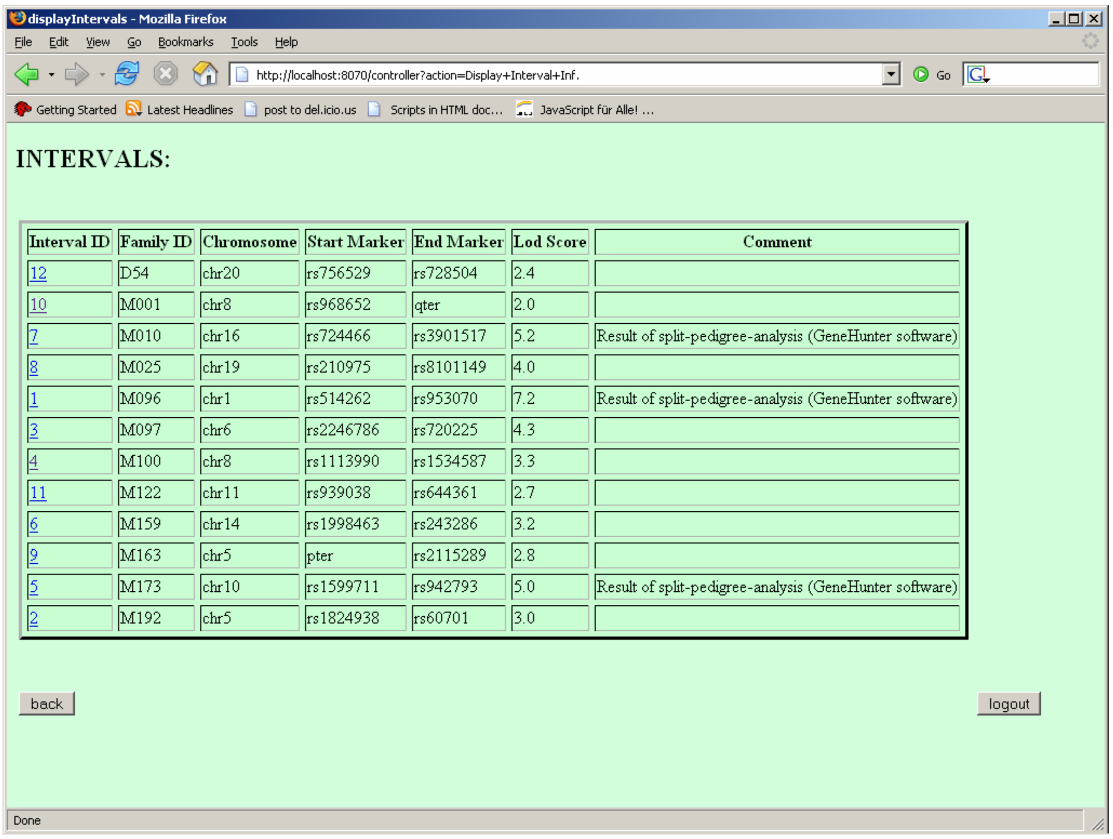
WELCOME TO FIDB

Family Information Database

Figure 6.4: Index page for users allowed updating FIDB.

6.3 Data Display

There are eight display pages, each displaying the contents of one relation of the database (for an example see Figure 6.5).


displayIntervals - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?action=Display+Interval+Inf.
Go
Getting Started
Latest Headlines
post to del.icio.us
Scripts in HTML doc...
JavaScript für Alle! ...
back
logout
Done

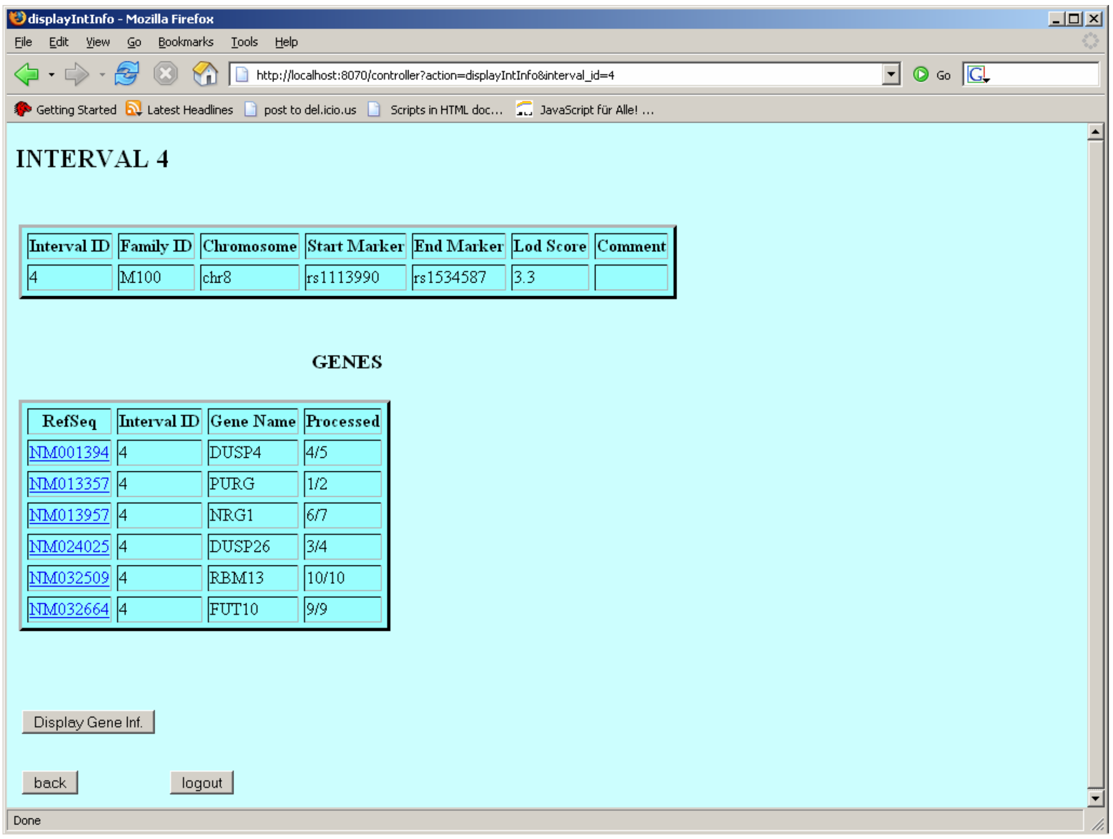
INTERVALS:

Interval ID	Family ID	Chromosome	Start Marker	End Marker	Lod Score	Comment
12	D54	chr20	rs756529	rs728504	2.4	
10	M001	chr8	rs968652	qter	2.0	
7	M010	chr16	rs724466	rs3901517	5.2	Result of split-pedigree-analysis (GeneHunter software)
8	M025	chr19	rs210975	rs8101149	4.0	
1	M096	chr1	rs514262	rs953070	7.2	Result of split-pedigree-analysis (GeneHunter software)
3	M097	chr6	rs2246786	rs720225	4.3	
4	M100	chr8	rs1113990	rs1534587	3.3	
11	M122	chr11	rs939038	rs644361	2.7	
6	M159	chr14	rs1998463	rs243286	3.2	
9	M163	chr5	pter	rs2115289	2.8	
5	M173	chr10	rs1599711	rs942793	5.0	Result of split-pedigree-analysis (GeneHunter software)
2	M192	chr5	rs1824938	rs60701	3.0	

Figure 6.5: Page displaying interval information (for users allowed to update FIDB). By clicking the interval_id, all available information about the interval is displayed.

If applicable hyperlinks are available to display related information (see Figure 6.5). When a pedigree is chosen, the persons in the pedigree, the path to the linkage files and the intervals are displayed.

Clicking a person will lead to a page where the clinical information and the samples are displayed. Similarly selecting a specific interval will display the genes belonging to this interval and clicking a gene will display all the variations within the gene that have been deposited in FIDB.


displayIntInfo - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?action=displayIntInfo&interval_id=4
Getting Started Latest Headlines post to del.icio.us Scripts in HTML doc... JavaScript für Alle! ...

INTERVAL 4

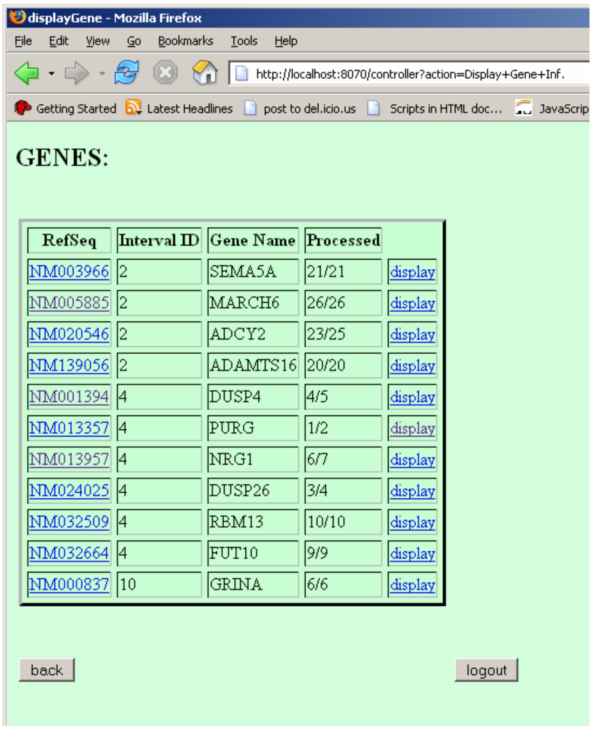
Interval ID	Family ID	Chromosome	Start Marker	End Marker	Lod Score	Comment
4	M100	chr8	rs1113990	rs1534587	3.3	

GENES

RefSeq	Interval ID	Gene Name	Processed
NM001394	4	DUSP4	4/5
NM013357	4	PURG	1/2
NM013957	4	NRG1	6/7
NM024025	4	DUSP26	3/4
NM032509	4	RBM13	10/10
NM032664	4	FUT10	9/9

Display Gene Inf.
back
logout
Done

Figure 6.6: Page displaying information belonging to one interval (for users not allowed to update FIDB).


displayGene - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?action=Display+Gene+Inf.
Getting Started Latest Headlines post to del.icio.us Scripts in HTML doc... JavaScrip

GENES:

RefSeq	Interval ID	Gene Name	Processed	
NM003966	2	SEMA5A	21/21	display
NM005885	2	MARCH6	26/26	display
NM020546	2	ADCY2	23/25	display
NM139056	2	ADAMTS16	20/20	display
NM001394	4	DUSP4	4/5	display
NM013357	4	PURG	1/2	display
NM013957	4	NRG1	6/7	display
NM024025	4	DUSP26	3/4	display
NM032509	4	RBM13	10/10	display
NM032664	4	FUT10	9/9	display
NM000837	10	GRINA	6/6	display

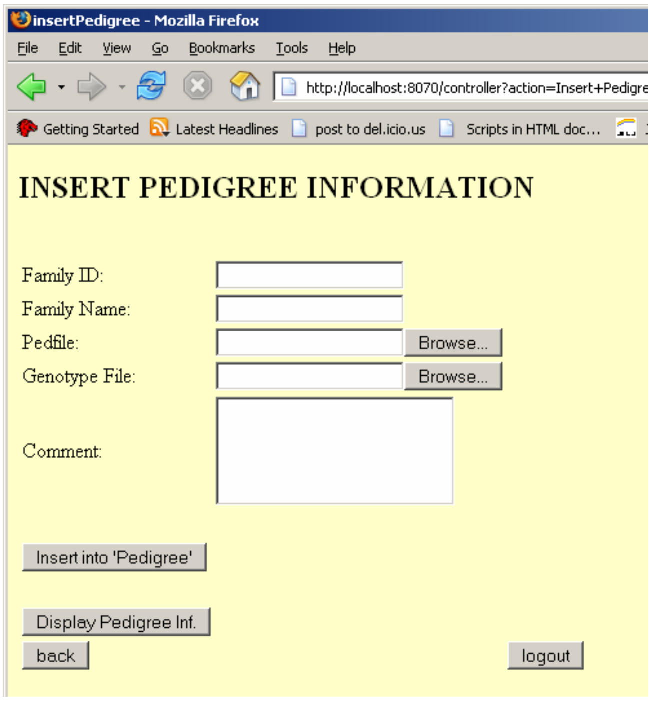
Figure 6.7: Gene information. (For users not allowed updating FIDB.) The display link leads to the UCSC Genome Browser website.

To help users to make use of the available genome annotation, there are links in the display of gene and variations (see Figure 6.7). Following such link leads to the to the UCSC Genome Browser website with the respective region displayed.

6.4 Data Manipulation

6.4.1 Data Insertion

The insert pages are only allowed to be accessed by the users allowed to update. Every relation has got an insert page, where all the attributes have text input fields, drop down lists or file select fields (for an example see Figure 6.8).



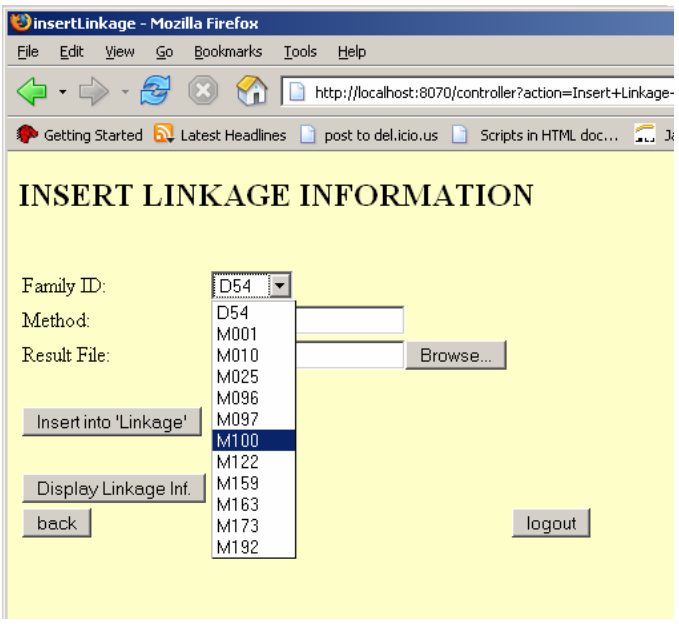
INSERT PEDIGREE INFORMATION

Family ID:
Family Name:
Pedfile:
Genotype File:
Comment:

Figure 6.8: Insert pedigree information.

With the file select, one can browse through the computer, but the path to the file is lost as soon as the submit button is clicked and only the name of the file is passed. As only the path to the file is needed, I wrote a function in Java script to save the path, by putting it into another invisible field.

When there are foreign keys in the relation, there is usually a drop down list with the already inserted keys (for an example see Figure 6.9).


insertLinkage - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?action=Insert+Linkage-
Getting Started
Latest Headlines
post to del.icio.us
Scripts in HTML doc...

INSERT LINKAGE INFORMATION

Family ID: D54

Method:

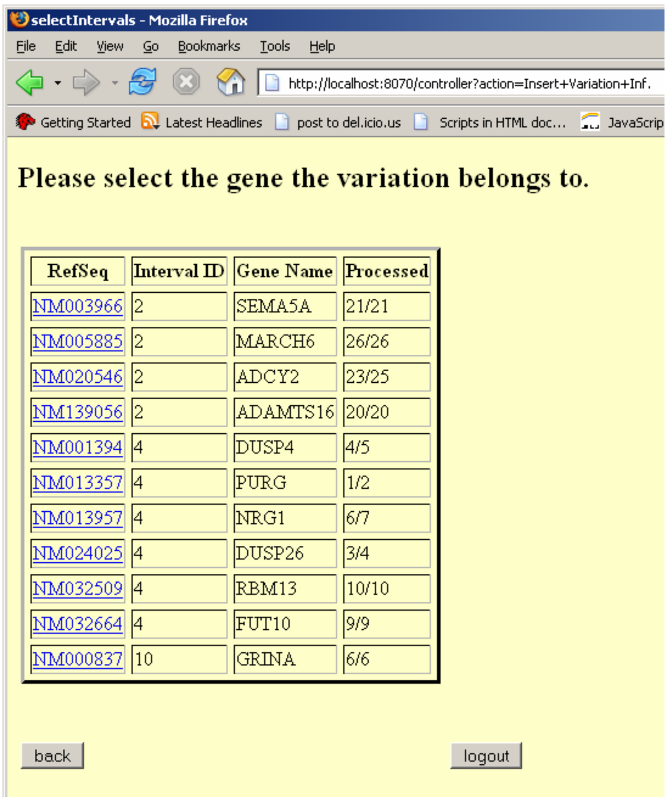
Result File: Browse...

D54, M001, M010, M025, M096, M097, M100, M122, M159, M163, M173, M192

Insert into 'Linkage'
Display Linkage Inf.
back
logout

Figure 6.9: Insert linkage information. Including a drop down list for family_id.

If one wants to insert a gene, the interval has to be picked first. The same is true for variations, where a gene has to be picked first (see Figure 6.10).

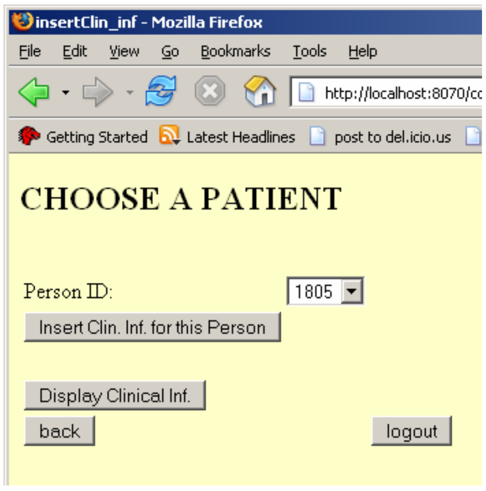

selectIntervals - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?action=Insert+Variation+Inf.
Getting Started
Latest Headlines
post to del.icio.us
Scripts in HTML doc...
JavaScrip
back
logout

Please select the gene the variation belongs to.

RefSeq	Interval ID	Gene Name	Processed
NM003966	2	SEMA5A	21/21
NM005885	2	MARCH6	26/26
NM020546	2	ADCY2	23/25
NM139056	2	ADAMTS16	20/20
NM001394	4	DUSP4	4/5
NM013357	4	PURG	1/2
NM013957	4	NRG1	6/7
NM024025	4	DUSP26	3/4
NM032509	4	RBM13	10/10
NM032664	4	FUT10	9/9
NM000837	10	GRINA	6/6

Figure 6.10: Select a gene. Before inserting variation information, the gene the variation was found in has to be picked.

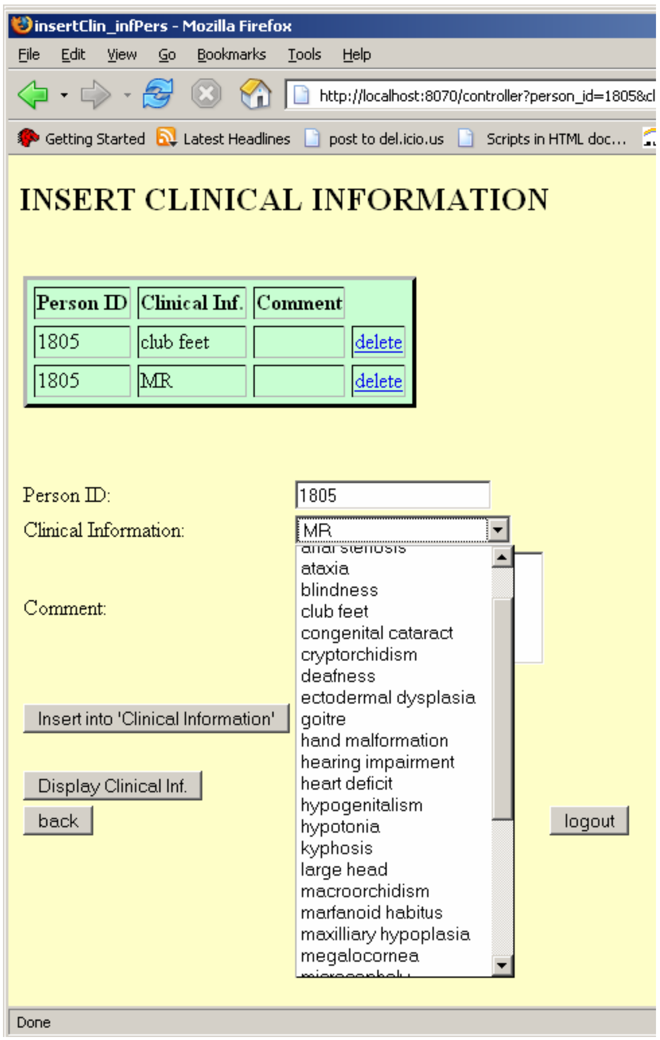
Another special case within the insert pages is the page to insert clinical information. A person is picked first, and then all the clinical features for this person can be inserted repeatedly (see Figures 6.11 and 6.12).



CHOOSE A PATIENT

Person ID:	1805

Figure 6.11: Choose a patient. Before inserting clinical information, a patient has to be chosen.


insertClin_infPers - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?person_id=1805&cl
Getting Started
Latest Headlines
post to del.icio.us
Scripts in HTML doc…

INSERT CLINICAL INFORMATION

Person ID	Clinical Inf.	Comment	
1805	club feet		delete
1805	MR		delete

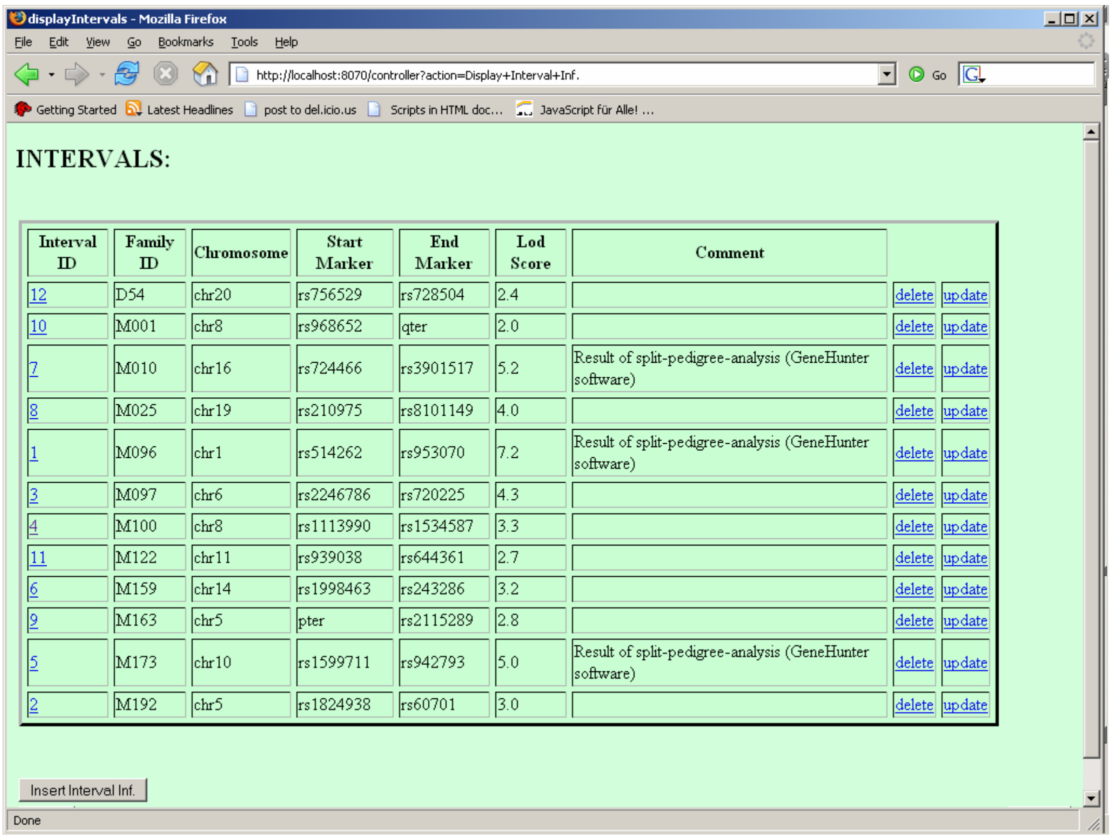
Person ID:	1805
Clinical Information:	MR
Comment:	

ataxia
blindness
club feet
congenital cataract
cryptorchidism
deafness
ectodermal dysplasia
goitre
hand malformation
hearing impairment
heart deficit
hypogenitalism
hypotonia
kyphosis
large head
macroorchidism
marfanoid habitus
maxilliary hypoplasia
megalocornea

Figure 6.12: Insert clinical information. All the clinical information for a patient can be filled in on the same page.

6.4.2 Data Updating and Deletion

The users with the right to change the data in the database do also have a link to delete every entry displayed on the display pages. For the tables consisting of more than two attributes, a link to update is also available for this group of users (see Figure 6.13).

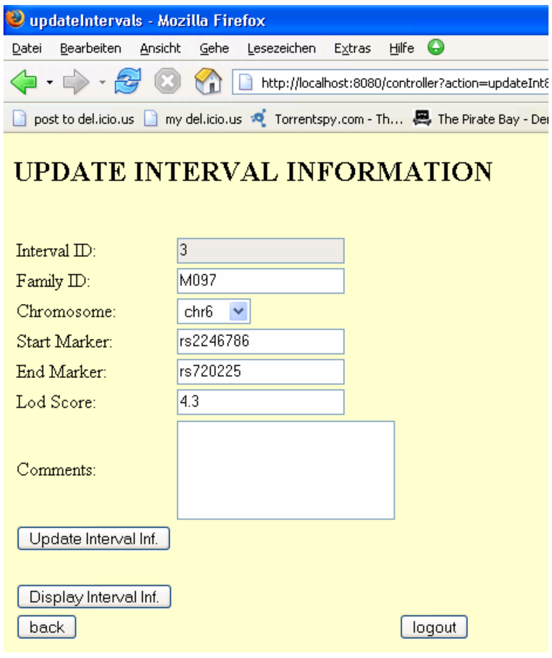

displayIntervals - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?action=Display+Interval+Inf.
Go
Getting Started Latest Headlines post to del.icio.us Scripts in HTML doc... JavaScript für Alle! ...
Insert Interval Inf.
Done

INTERVALS:

Interval ID	Family ID	Chromosome	Start Marker	End Marker	Lod Score	Comment		
12	D54	chr20	rs756529	rs728504	2.4		delete	update
10	M001	chr8	rs968652	qter	2.0		delete	update
7	M010	chr16	rs724466	rs3901517	5.2	Result of split-pedigree-analysis (GeneHunter software)	delete	update
8	M025	chr19	rs210975	rs8101149	4.0		delete	update
1	M096	chr1	rs514262	rs953070	7.2	Result of split-pedigree-analysis (GeneHunter software)	delete	update
3	M097	chr6	rs2246786	rs720225	4.3		delete	update
4	M100	chr8	rs1113990	rs1534587	3.3		delete	update
11	M122	chr11	rs939038	rs644361	2.7		delete	update
6	M159	chr14	rs1998463	rs243286	3.2		delete	update
9	M163	chr5	pter	rs2115289	2.8		delete	update
5	M173	chr10	rs1599711	rs942793	5.0	Result of split-pedigree-analysis (GeneHunter software)	delete	update
2	M192	chr5	rs1824938	rs60701	3.0		delete	update

Figure 6.13: Page displaying interval information (for users allowed to update FIDB).

All of the update pages have all the attributes displayed in input fields, with the key attributes not changeable (for an example see Figure 6.14). The non key attributes can be changed easily in this way.

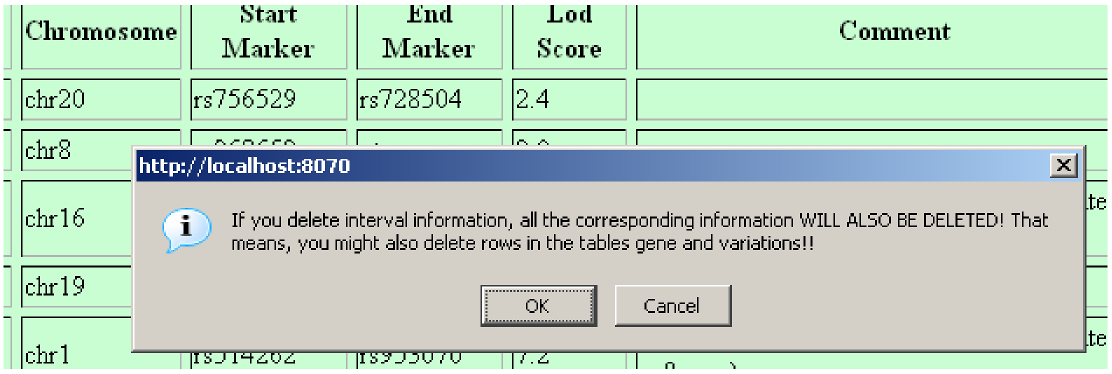

updateIntervals - Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://localhost:8080/controller?action=updateInt
post to del.icio.us
my del.icio.us
Torrentspy.com - Th...
The Pirate Bay - De
Update Interval Inf.
Display Interval Inf.
back
logout

UPDATE INTERVAL INFORMATION

Interval ID:	3
Family ID:	M097
Chromosome:	chr6
Start Marker:	rs2246786
End Marker:	rs720225
Lod Score:	4.3
Comments:	

Figure 6.14: Update interval information.

When a user tries to delete an entry that might be referenced in another relation like entries in pedigree, person, intervals and genes, Java Script was used to give an alert that more data might be deleted in other tables (see Figure 6.15).



Chromosome	Start Marker	End Marker	Lod Score	Comment
chr20	rs756529	rs728504	2.4	
chr8				
chr16				
chr19				
chr1				

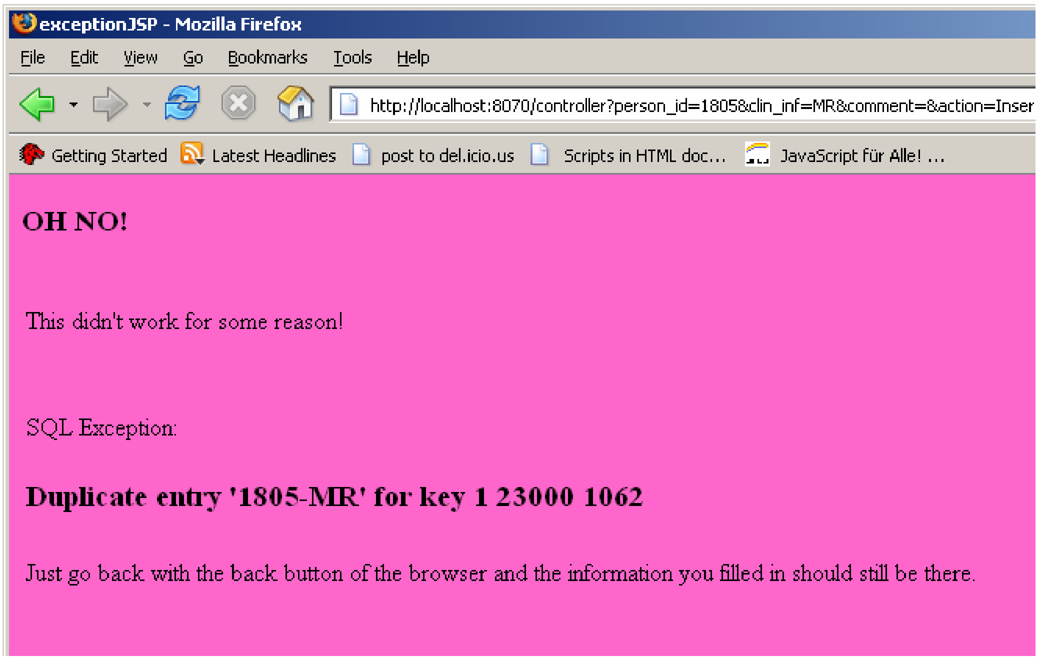
http://localhost:8070

If you delete interval information, all the corresponding information WILL ALSO BE DELETED! That means, you might also delete rows in the tables gene and variations!!

Figure 6.15: Alert when deleting an entry in intervals is tried.

6.5 Error Message

The error page displays the SQL error message, which is self explaining (see Figure 6.16).


exceptionJSP - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8070/controller?person_id=1805&clin_inf=MR&comment=&action=Inser
Getting Started
Latest Headlines
post to del.icio.us
Scripts in HTML doc...
JavaScript für Alle! ...

OH NO!

This didn't work for some reason!

SQL Exception:

Duplicate entry '1805-MR' for key 1 23000 1062

Just go back with the back button of the browser and the information you filled in should still be there.

Figure 6.16: Error page. Displaying the SQL exception.

7 Summary and Future Work

In this work a database management system to handle clinical and experimental data was developed. One is able to insert, update and delete the data using the web interface. Furthermore the data can be displayed in different manners.
Data security was taken into account and the different authority for two user groups are established. Up to now, the data gathered through the analysis of about approximately 150 families with autosomal recessive mental retardation have been inserted.

On this huge data basis it also makes sense to develop more sophisticated search options. For example, a function to be implemented is to search for all the families with a common clinical feature and then compare their linkage intervals.

8 Acknowledgments

First of all I want to thank Wei Chen for his support and helpful discussion. Further I want to acknowledge my supervisors Dr. Annika Hinze and Dr. Andreas Kuß for not only making this work possible, but also for the many vital comments on my work. In addition I thank Lars Jensen and Mahdi Motazacker for the explanation of biological issues.
Also, I thank Garreth Hollands and Jannis Uhlendorf for carefully proofreading the manuscript and Maximilian Sauermoser for his help on creating the figures. This work was funded by the Max Planck Society for the Advancement of Science.

9 References

Avedal, Karl / Ayers, Danny / Briggs, Timothy / Burnham, Carl / Halberstedt, Ari / Haynes, Ray / Henderson, Peter / Holden, Mac / Li, Sing / Malks, Dan / Myers, Tom / Nakhimovsky, Alexander / Osmont, Stéphane / Palmer, Grant / Timney, John / Tyagi, Sameer / Van Damme, Geert / Wilcox, Mark / Wilkinson, Steve / Zeiger, Stefan / Zukowski, John: "Professional JSP"; Birmingham 2000.

Bry, Francois / Kröger, Peter: "A Computational Biology Database Digest: Data, Data Analysis, and Data Management"; Munich 2003.

Chen, Peter P.: "The Entity-Relationship Model - Toward a Unified View of Data"; Transactions on Database Systems, Vol. 1, pp. 9-36, New York 1976.

Codd, E. F.: "A Relational Model of Data for Large Shared Data Banks"; Communications of the ACM, Vol. 13, No. 6, pp. 377-387, 1970.

Codd, E. F.: "Does Your DBMS Run By the Rules?"; Computer World, 21st October 1985.

DuBois, Paul: "MySQL"; Indianapolis 2000.

Hall, Marty / Brown, Larry: "Core Servlets and Java Server Pages"; 2nd edition, 2003.

Kemper, Alfons / Eickler, A.: "Datenbanksysteme, Eine Einführung"; 4., überarbeitete und erweiterte Auflage, München 2001.

Korth, Henry F. / Siberschatz, Abraham: "Database System Concepts"; 2nd edition, New York, 1991.

Louis, Dirk / Müller, Peter: “Java 2. Praxis der objektorientierten Programmierung“; München, 2004.

Strachan, Tom / Read, Andrew P.: "Human Molecular Genetics"; 3rd edition, New York 2004.

Teorey, Toby J.: "Database Modeling and Design"; 3rd edition, San Francisco 1999.

Ullman, Jeffrey D.: "Principles of Database and Knowledgebase Systems", vol. 1, Rockville 1988.

World Health Organization: "International classification of impalments, disabilities and handicaps"; World Health Organization, Geneva 1980.

10 Appendix

10.1 List of Figures

10.2 List of Abbreviations

API	Application programming interface
DB	Database
DBMS	Database management system
DNA	Desoxyribonucleic acid
FIDB	Family Information Database
HTML	Hypertext markup language
HTTP	Hypertext transfer protocol
HUGO	Human Genome Organisation
ID	Identification
IQ	Intelligence quotient
JDBC	Java database connectivity
JSP	Java server pages
LAN	Local area network
SNP	Single nucleotide polymorphism
SQL	Structured query language
WHO	World Health Organization

10.3 Source Code

Due to the extensive source code, I will only give a few examples and the SQL create statements.

The complete code can be requested at "albers@inf.fu-berlin.de".

The database schema in form of the SQL create statements:

```
CREATE TABLE pedigree (
family_id       VARCHAR(20)      NOT NULL,
fam_name        VARCHAR(255)     NOT NULL default '',
ped_file        VARCHAR(255)     NOT NULL default '',
genotype_data   VARCHAR(255)     NOT NULL default '',
comment         TEXT,
PRIMARY KEY     (family_id)
);

CREATE TABLE person (
person_id       VARCHAR(20)           NOT NULL,
within_fam_id   VARCHAR(10)           NOT NULL default '',
family_id       VARCHAR(20)           NOT NULL,
name            VARCHAR(255)          NOT NULL default '',
sex             ENUM('f', 'm'),
birthday        DATE,
affect_status   BOOL,
father          VARCHAR(20)           NOT NULL default '',
mother          VARCHAR(20)           NOT NULL default '',
PRIMARY KEY     (person_id),
CONSTRAINT      family_exists
FOREIGN KEY     (family_id)
REFERENCES      pedigree(family_id)   ON DELETE CASCADE
);

CREATE TABLE clin_inf (
person_id       VARCHAR(20)           NOT NULL,
clin_inf        VARCHAR(255)          NOT NULL,
comment         TEXT,
PRIMARY KEY     (person_id, clin_inf),
CONSTRAINT      person_exists
FOREIGN KEY     (person_id)
REFERENCES      person(person_id)     ON DELETE CASCADE
);

CREATE TABLE sample (
person_id       VARCHAR(20)           NOT NULL,
type            VARCHAR(50)           NOT NULL,
PRIMARY KEY     (person_id, type),
CONSTRAINT      person_exist
FOREIGN KEY     (person_id)
REFERENCES      person(person_id)     ON DELETE CASCADE
);

CREATE TABLE linkage (
family_id       VARCHAR(20)           NOT NULL,
method          VARCHAR(50)           NOT NULL,
result_files    VARCHAR(255)          NOT NULL default '',
PRIMARY KEY      (family_id, method),
CONSTRAINT       family_exist
FOREIGN KEY      (family_id)
REFERENCES       pedigree(family_id)   ON DELETE CASCADE
);

CREATE TABLE intervals (
interval_id      BIGINT                NOT NULL,
family_id        VARCHAR(20)           NOT NULL,
chromosome       VARCHAR(5)            NOT NULL default '',
start_marker     VARCHAR(30)           NOT NULL default '',
end_marker       VARCHAR(30)           NOT NULL default '',
lod_score        FLOAT(5,3),
comment          TEXT,
PRIMARY KEY      (interval_id),
CONSTRAINT       family_exis
FOREIGN KEY      (family_id)
REFERENCES       pedigree(family_id)   ON DELETE CASCADE
);

CREATE TABLE gene (
ref_seq_nr       VARCHAR(30)           NOT NULL,
interval_id      BIGINT                NOT NULL,
gene_name        VARCHAR(20)           NOT NULL default '',
processed        VARCHAR(20)           NOT NULL default '',
PRIMARY KEY      (ref_seq_nr, interval_id),
CONSTRAINT       interval_exists
FOREIGN KEY      (interval_id)
REFERENCES       intervals(interval_id) ON DELETE CASCADE
);

CREATE TABLE variations (
var_id           BIGINT           NOT NULL,
ref_seq_nr       VARCHAR(30)      NOT NULL,
interval_id      BIGINT           NOT NULL,
type             ENUM('single', 'microsatellite', 'in-del',
                 'insertion', 'deletion')
                                  NOT NULL,
chromosome       VARCHAR(5)       NOT NULL default '',
chrom_start      BIGINT,
chrom_end        BIGINT,
assembly         VARCHAR(20),
function         ENUM('unknown', 'locus', 'coding', 'coding-
                 synon', 'coding-nonsynon', 'untranslated',
                 'intron', 'splice-site', 'cds-reference')
                                  NOT NULL default 'unknown',
known_poly       ENUM('unknown', 'yes', 'no')
                                  NOT NULL default 'unknown',
coseg_status     ENUM('unknown', 'yes', 'no')
                                  NOT NULL default 'unknown',
misc             TEXT,
PRIMARY KEY      (var_id),
CONSTRAINT       gene_exists
FOREIGN KEY      (ref_seq_nr, interval_id)
REFERENCES       gene(ref_seq_nr, interval_id)
                                       ON DELETE CASCADE,
);
```

Examples of a JSP, a servlet and a JavaBean.

JSP to insert person information:

```
<%@page
    contentType="text/html"
    language="java"
    import="java.util.*"
    import="java.lang.Integer.*"
  %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>insertPerson</title>
        <script type="text/javascript">
        function chkForm () {
        if (document.form.person_id.value == "") {
            alert("Please fill in the Person ID!");
            document.form.person_id.focus();
            return false;
            }
        }
        </script>
        </head>
    <body bgcolor="#FFFFC8">

    <h2>INSERT PROBAND INFORMATION</h2>
    <br>

    <table>
            <form name="form" action="controller" method = "get"
            onsubmit="return chkForm()">
            <tr>
            <td>Person ID:</td>
            <td><input type="text" name="person_id"</td>
            </tr>
            <tr>
            <td>ID within the Pedigree:</td>
            <td><input type = "text" name="within_fam_id"></td>
            </tr>
            <tr>
            <td>Family ID:</td>
            <td>
            <select name="family_id">

                <jsp:useBean
                 id="pedi"
                 scope="request"
                 class="beans.pedigreeBean"/>

<% Vector pedigree = (Vector)request.getAttribute("selectPedigree");

for (Enumeration e = pedigree.elements() ; e.hasMoreElements() ;) {
                    pedi = (beans.pedigreeBean)e.nextElement();
                 %>
                <option> <%=pedi.getFamily_id()%> </option>

                <% } %>

            </select>
            </td>
            </tr>
            <tr>
            <td>Name:</td>
            <td><input type="text" name="name"</td>
            </tr>
            <tr>
            <td>Sex:</td>
            <td><select name="sex" size="1">
                <option>m</option><option>f</option>
            </select>
            </td>
            </tr>
            <tr>
            <td>Birthday (yyyy-mm-dd):</td>
            <td><select name="year">
                <%for ( int i = 1900; i <= 2010; i++ ) { %>
                <option><%=i%></option>
                <% } %>
            </select>
            <select name="month">
                <%for ( int i = 1; i <= 12; i++ ) { %>
                <option><%=i%></option>
                <% } %>
            </select>
            <select name="day">
                <%for ( int i = 1; i <= 31; i++ ) { %>
                <option><%=i%></option>
                <% } %>
            </select>
            </td>
            </tr>
            <tr>
            <td>Affect Status:</td>
            <td><select name="affect_status" size="1">
                  <option>affected</option>
                  <option>not affected</option>
            </select>
            </td>
            </tr>
            <tr>
            <td>Father (ID):</td>
            <td><input type = "text" name="father"></td>
            </tr>
            <tr>
            <td>Mother (ID):</td>
            <td><input type="text" name="mother"></td>
            </tr>
            <tr>
            <td><br></td>
            </tr>
            <tr>
```

```
            <td><input type="submit" name="action" value="Insert into
            'Person'"></td>
            </tr>
            </form>
            <form>
            <tr>
            <td><br></td>
            </tr>
            <tr>
            <td><input type="submit" name="action" value="Display
            Person Inf."></td>
            </tr>
            <tr>
            <td><input type="submit" name="action" value="back"></td>
            <td></td>
            <td><input type="submit" name="action"
            value="logout"></td>
            </tr>
        </form>
        </table>

    </body>
</html>
```

Servlet to insert person information:

```
/*
 * insertPerson.java
 *
 * Created on 21. Juni 2006, 10:10
 */

package db;
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.* ;

/**
 *
 * @author albers
 */
public class insertPerson extends HttpServlet {

    private String insertSQL = " INSERT INTO person VALUES ( ? , ? ,
? , ? , ? , ? , ? , ? , ? )  ;";

    public void destroy() {  }

    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, java.io.IOException {
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, java.io.IOException {
        processRequest(request, response);
    }

    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, java.io.IOException {
        Connection con = null ;

        try{
            pool p = new pool();
            con = p.get();

            int affect_status = 0;

if((request.getParameter("affect_status")).equals("affected"))
                affect_status = 1;
            else
if((request.getParameter("affect_status")).equals("not affected"))
                affect_status = 0;

            String birthday = "";
            String month, day = "";

            if(Integer.parseInt(request.getParameter("month")) < 10)
                month = "0" + (request.getParameter("month"));
            else
                month = request.getParameter("month");

            if(Integer.parseInt(request.getParameter("day")) < 10)
                day = "0" + (request.getParameter("day"));
            else
                day = request.getParameter("day");

            birthday = request.getParameter("year") + "-" + month +
"-" + day;



          PreparedStatement pstmt = con.prepareStatement(insertSQL);

          pstmt.setString(1, request.getParameter("person_id"));
          pstmt.setString(2, request.getParameter("within_fam_id"));
          pstmt.setString(3, request.getParameter("family_id"));
          pstmt.setString(4, request.getParameter("name"));
          pstmt.setString(5, request.getParameter("sex"));
          pstmt.setString(6, birthday);
          pstmt.setInt(7, affect_status);
          pstmt.setString(8, request.getParameter("father"));
          pstmt.setString(9, request.getParameter("mother"));

          pstmt.executeUpdate();

     //close data connection
```

```
      pstmt.close();

      String dispatch ="/selectPedigree?action=person" ;

            RequestDispatcher dispatcher =
request.getRequestDispatcher(dispatch) ;
            dispatcher.forward(request, response) ;

        }catch(SQLException ex){
      System.out.println("\nERROR:----- SQLException -----\n");

      request.setAttribute("getException", ex);
      String dispatch = "exceptionJSP.jsp";

      RequestDispatcher dispatcher =
request.getRequestDispatcher(dispatch) ;
      dispatcher.forward(request, response) ;

    }catch(Exception e ) {
      e.printStackTrace();
    }finally {
      try {
        if(con != null)
          con.close() ;
      }catch (SQLException ex)  {
        System.out.println("\nERROR:----- SQLException -----\n");
        System.out.println("Message:   " + ex.getMessage());
        System.out.println("SQLState:  " + ex.getSQLState());
        System.out.println("ErrorCode: " + ex.getErrorCode());
      }
    }

    }
    public String getServletInfo() {
        return "Servlet inserts Persons";
    }
}
```

JavaBean for the relation person:

```
/*
 * personBean.java
 *
 * Created on 9. Juni 2006, 12:54
 */

package beans;

import java.beans.*;
import java.io.Serializable;

/**
 *
 * @author albers
 */
```

```
public class personBean extends Object implements Serializable {

    private String person_id;
    private String within_fam_id;
    private String family_id;
    private String name;
    private String sex;
    private java.sql.Date birthday; //oder java.sql.date??
    private int affect_status;
    private String father;
    private String mother;

    /** Creates a new instance of personBean */
    public personBean() {}

    public String getPerson_id() {
        return person_id;
    }
    public String getWithin_fam_id() {
        return within_fam_id;
    }
    public String getFamily_id() {
        return family_id;
    }
    public String getName() {
        return name;
    }
    public String getSex() {
        return sex;
    }
    public java.sql.Date getBirthday() {
        return birthday;
    }
    public int getAffect_status() {
        return affect_status;
    }
    public String getFather() {
        return father;
    }
    public String getMother() {
        return mother;
    }

    public void setPerson_id(String person_id) {
        this.person_id=person_id;
    }
    public void setWithin_fam_id(String within_fam_id) {
        this.within_fam_id=within_fam_id;
    }
    public void setFamily_id(String family_id) {
        this.family_id=family_id;
    }
    public void setName(String name) {
        this.name=name;
    }
    public void setSex(String sex) {
        this.sex=sex;
    }
    public void setBirthday(java.sql.Date birthday) {
        this.birthday=birthday;
    }
    public void setAffect_status(int affect_status) {
        this.affect_status=affect_status;
    }
    public void setFather(String father) {
        this.father=father;
    }
    public void setMother(String mother) {
        this.mother=mother;
    }

}
```