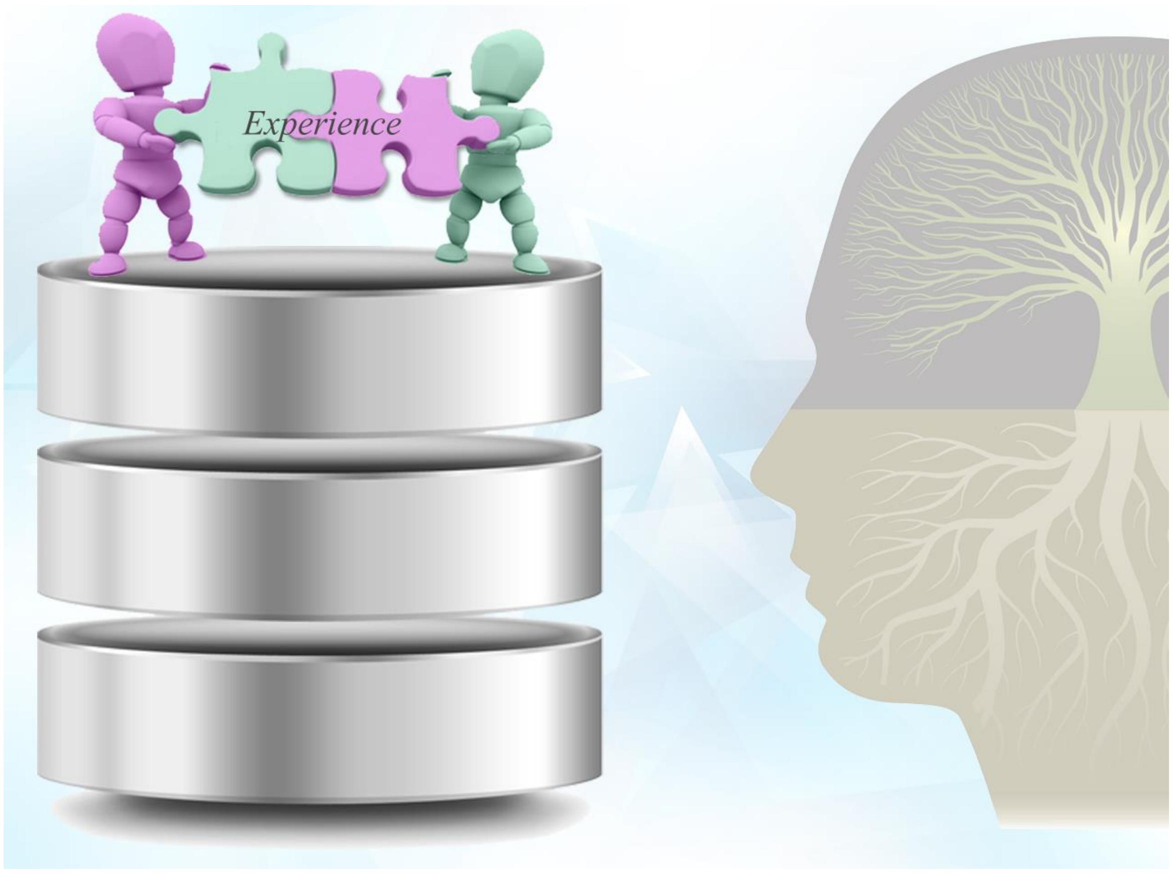
CHALMERS



Experience database

Pre-study and development at QRTECH AB

Master of Science Thesis

ANDREAS OHLDIN
MARCUS WELDEBORN NORLANDER

Chalmers University of Technology
Department of Applied Information Technology,
Göteborg, Sweden, 2012
Report nr: 2013:006
ISSN: 1651-4769

Experience database
Pre-study and development at QRTECH AB

By

ANDREAS H. OHLDIN
MARCUS G.E. WELDEBORN NORLANDER

Performed at: QRTECH AB
Mejerigatan 1, 412 76 Göteborg

Examiner:
Claes Strannegård
claes.strannegard@chalmers.se

Academic supervisor:
Claes Strannegård
claes.strannegard@chalmers.se

QRTECH supervisor:
Olof Bergquist
Mejerigatan 1, 41276 Göteborg
olof.bergquist@qrtech.se

Chalmers University of Technology
University of Gothenburg
Department of Applied Information Technology,
SE-412 96 Göteborg
Telephone + 46 (0)31-772 1000

Department of Applied Information Technology,
Göteborg, Sweden July 2012

Cover: Illustrates individuals exchanging experiences and how these experiences can be stored into a database.

Abstract

In today's enterprises, information is exchanged faster than ever before. Companies can gain a competitive advantage by implementing new IT tools, making their organization more efficient.

This thesis focus on developing a tool to facilitate the exchange of experiences inside the enterprise QRTECH AB situated in Göteborg, Sweden. The company found no solution on today's market and decided to develop a new instrument adjusted to their needs.

In this thesis an experience is defined as a knowledge a specific person has obtained during a certain period of time. An experience can be described using words, which in this thesis are called tags, in order to give experiences a context. This creates opportunities which are difficult to obtain when utilizing a traditional approach of categorizing.

The application consists of a MySQL database and a web site. PHP was used to enable the communication between the database and web site. The necessary languages used for the web site are XHTML and JavaScript. By utilizing the library JQuery and method AJAX the development process was facilitated.

The web site has the ability to search for experiences and projects in order for employees to find colleagues possessing knowledge requested. Employees can create a personal account to be able to add experiences and projects into the database. An administrator account has been implemented in order to enable the possibility to prepare the database with information since it is empty at the beginning.

By evaluating the performance of the web site, using two developed scripts, several improvements were implemented on the database queries.

Currently, the established functionalities create the foundation on which future features can be implemented. The objective for future works should be to continue improving the performance, functionality and layout and to review issues regarding security. However, the first issue to address is to solve the problems associated with Internet Explorer 8.0 since it is QRTECH's default browser.

Preface

This thesis was performed during the spring/summer in year 2012 by Andreas Ohldin and Marcus Weldeborn in collaboration with QRTECH AB and the Department of Applied Information Technology at Chalmers University of Technology. The objective was to develop an experience database for QRTECH AB in order to enable employees to internally locate knowledge.

We are two students, who have studied three years at the Automation and Mechatronics institution followed by two years at the Intelligent System Design Master program at Chalmers University of Technology. The years at Chalmers have given us wider knowledge in electronics, mechanics, mathematics and programming. It is mainly the knowledge in mathematics and programming which has proven to be useful in the project. In this thesis MySQL along with the software MySQL Workbench was utilized to build the database meanwhile the XHTML, JavaScript and PHP code was written in Adobe Dreamweaver which facilitated the web interface development.

This section is dedicated to the people who, during the project, have devoted their time to help and guide us:

QRTECH AB
Olof Bergqvist, our supervisor, for his creative ideas, guidance and commitment.
Fredrik Hansson, for his creative ideas and expertise.
Roger Hendelberg, Lars-Åke Johansson, Tomas Olsson, Peter Buch and Joakim Bergman for their feedback and discussions during this thesis.

Chalmers University of Technology
Claes Strannegård, for the help with thesis administration.
Jonas Almström Duregård, for giving us feedback on the database structure even though he was not obliged to.

Göteborg, Sweden,

Keywords: MySQL, PHP, HTML, JavaScript, Database, Web interface

Contents

Abbreviations

Abbreviation	**Name**
ASP	Active Server Pages
AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DBMS	Database Management System
DOM	Document Object Model
DTD	Document Type Definition
ER-diagram	Entity Relationship-diagram
FD	Functional Dependency
HTML	Hypertext Markup Language
MIT	Massachusetts Institute of Technology
PHP	PHP: Hypertext Preprocessor
SQL	Structured Query Language
SVN	Subversion
UI	User Interface
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Commonly used abbreviations

List of Figures

List of Tables

1 Introduction

Companies wants to improve their efficiency. A common approach is to provide a good customer service and utilize the resources, for instance the employees, in a more beneficial way. A good communication within the company tends to be a vital part in order to have a high efficiency. A software solution for registering and handling experiences has been requested by the employees at QRTECH. This thesis will describe a software solution for registering and searching for experiences.

1.1 Background

QRTECH is a consulting firm with approximately 80 employees and is constantly growing. It is contracted by companies to develop products and is mainly situated in Göteborg. QRTECH's business plan is to lease employees to other companies, which basically makes *knowledge* the commercial product. The idea to develop a software tool emerged at an internal conference since employees at QRTECH had problems finding colleagues with the right expertise. It would be beneficial if technicians could search among colleagues in order to receive help and expertise. Similarly, the sales organization could use the tool as customers calls to locate employees with the qualifications needed. This simplifies the process of assigning the right person to the right post and makes the enterprise QRTECH more efficient.
The experience database development is made at QRTECH where no previous work has been performed in this area.

1.2 Purpose

Employees at QRTECH occasionally find it hard to locate colleagues with a certain experience. The company has been evaluating several existing software solutions on the market but none has been satisfactory. The decision to develop an own customized tool was made. The aim with the software is to find knowledge within QRTECH and thereby utilize the already existing resources more efficiently.

1.3 Delimitations

The time frame for this Master thesis is 20 weeks, ending approximately at the 10th of July 2012. The short time frame is essentially the main reason why limitations must be made:

- Security - No encryption or protection against advanced SQL injections
- Design - A basic layout will be implemented

- Browser compatibility - Focus on establishing support for the following four browsers Internet Explorer 8+, Firefox 13+, Safari 5.1+ and Google Chrome 19+
- Tutorial section - Help texts will be available on the site, but a in-depth tutorial showing functionalities will not be created

1.4 Thesis description

This master thesis will initially incorporate an investigation to determine what functionalities the employees at QRTECH request. Interviews with individual employees and group discussions involving employees possessing useful expertise will be performed to determine the actual needs. Ideas of functionality, which goes beyond our work, will be taken into account as future development possibilities. It is therefore, of great importance that this master thesis work is not detrimental to future development.

An ER-diagram (Entity Relationship), describing the database structure, will be developed to receive an overview of the database and verify its functionalities. The choice of DBMS will be determined by studying literature before any SQL-code is written.

Sketches specifying the design and functionality of the user interface are to be established. Programming languages and access methods, of various types, will be evaluated. The web interface should be accessible via the internal network at QRTECH and be able to run in standard browsers.

The software development will begin by implementing the basic functions log-in, search and profile page. Additional functionality will be implemented as the basic functionality is completed. Improved layout, design and remaining software refinements are handled at the end of the time frame for the project.

Our master thesis work will continuously be documented throughout the project.

Thesis objectives

1. Collection / identification of relevant data to be stored
2. Creation of a data model for the database
3. Selection of development chain (programming languages, database access mechanism)
4. Selection of SQL relational database
5. Design of software
6. Implementation of the database and access functions
7. Development of an user interface to search for knowledge
8. Documentation and presentation of the work

2 Theory

This section explains the theory behind techniques and tools used in the Master thesis.

2.1 Databases

A database consists of tables, also called relations, making it possible to store information. Databases are frequently used within corporations and organizations to store valuable data e.g. registering orders placed by costumers.

A database is controlled by a DBMS (Database Management System), making the database efficient. The DBMS makes it possible to create and manage large amounts of information as well as monitoring the data, in order to keep information consistent over time. DBMS are by many considered to be one of the most complex software ever written. (Garcia-Molina, 2009)

2.1.1 Tables and Relations

In databases a two-dimensional table is called a *relation*. The relation Movies, see Table 1, consists of the attributes *Length*, *Year*, *Title* and *Genre*. The attributes describe the properties of the relation Movies. For example, the attribute *Year* contains the year when a certain film was produced. A relation can contain multiple rows where each row corresponds to a specific film, e.g. Braveheart or Pulp fiction.

Title	**Year**	**Length**	**Genre**
Braveheart	1995	177	Action
Pulp fiction	1994	154	Thriller

Table 1: Example of a relation containing *Movies*

Relations can have various constraints where the most fundamental one is the *key constraint*. A *key* is a set of attributes which uniquely identifies a row in a relation. For instance, the relation Movies can have a key attribute combination of *Title* and *Year*. The assumption is that two movies with the same title will not be produced in the same year. If the assumption holds, then the title and the year a movie was produced is enough to identify a specific movie.

The relation in Table 2 has the attributes *Name*, *Address*, *Gender* and *Date of birth* where the name of the movie star is the key. The two relations Movies and MovieStar can be connected by combining their individual keys, see Table 3. By introducing a combined relation, a movie can be said to have many movie stars and likewise a movie star can take part in several films. Garcia-Molina (2009)

Name	Address	Gender	Date of birth
Mel Gibson	Chelsea road	Male	1956-01-03
John Travolta	Hollywood road	Male	1954-02-18

Table 2: Example of a relation containing *Moviestars*

Title	Year	Name
Braveheart	1995	Mel Gibson
Braveheart	1995	Sophie Marceau
Pulp fiction	1994	John Travolta

Table 3: Example of a relation containing *StarsIn*

2.1.2 Views

Tables are stored physically in the database and are persistent. This means, the information stored in a database is said to be saved for an infinite time, as long as modifications aren't performed. Modifications are done by executing a query* to the database.

Views on the other hand do not physically exist in the database and contains no data on itself. Views are often called *Virtual Views* because these elements don't exist on their own. By using views, information from various tables can be combined into a common result. It is possible to query a view in order to receive a result and, in some sense, to modify views. (Garcia-Molina, 2009)

2.1.3 DBMS

A DBMS structures data in databases and makes it accessible. There exists various types of DBMS, for instance *Oracle*, *MySQL* and *Microsoft Access*. The main features of a DBMS, regardless of which brand, is the following:

- Provide facilities for creating the database structure
  - Define the logical structure of the data to be stored
  - Define relationships among data
- Provide the ability to insert, modify and delete data
  - Form-based or command-line interface
- Provide the ability to receive data

* A query is a question to the database to execute a command

  - Support for complex queries using Boolean algebra (AND, OR and NOT operators)

- Provide methods for restricting access to data
  - For instance creating usernames and passwords and assign access data to the user

DBMS are usually designed for multiple user access, though some systems are entirely intended to handle single users. Similarly, there exists DBMS for all sizes of organizations where larger installations are performed using mainframes. These are often categorized as enterprise edition DBMS and are expansive pieces of software. (Harrington, 2009)

2.1.4 ER-Diagram

In the initial process of developing a database various options are considered and changes are rapidly performed. A common method for describing databases on a higher level is the ER-diagram. An ER-diagram can describe schemas of databases graphically and visualize the design. However, the diagram do not contain any actual data, it is merely a graph. (Garcia-Molina, 2009)

An ER-diagram is a graph consisting of relationships, attributes and entity sets where each category is represented by various shapes:

- Relationships – Diamond
- Attributes – Oval
- Entity sets – Rectangular

Attributes are connected to entity sets using edges and the same applies for the connection between relationships and entity sets. Figure 1 illustrates the possible shapes and the connections enabled by edges in an ER-diagram. (Garcia-Molina, 2009)

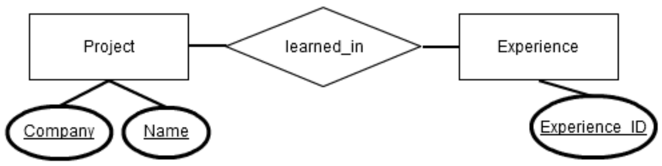


Figure 1: ER-diagram fundamentals

In Figure 1 the entity Project consists of two attributes *Company* and *Name*. Similarly, is the entity Experience constructed with its attribute *Experience ID* meanwhile the relationship *learned_in* establishes the connection between the entities Experience and Project.

2.1.5 Functional dependencies

A functional dependency (FD) on a specified relation R is describes as:

If two tuples of R have all their attributes equal each other A_1, A_2, ..., A_n then they have to agree on the same attributes of another list B_1, B_2, ..., B_m. The functional dependency can be written as A_1, A_2, ..., $A_n \rightarrow B_1$, B_2, ..., B_m, same as saying A_1, A_2, ..., A_n functionality determines B_1, B_2, ..., B_m. It can also be interpreted as, "Given the left side, the right side can be determined". (Garcia-Molina, 2009)

2.1.5.1 An applied example

A badly designed relation is displayed in Table 4 and it cannot be created in the database due to the FDs. To find out what's wrong with the design, the FDs are investigated further.

The entity Project has the attributes *Company*, *Name*, *Start date*, *End date* and *Employee*, see Table 4. The key of this relation is the tuple *Company* and *Name* and the functional dependency can be assumed to be:

- Company, Name → Start date, End date

If two rows in a relation have the same values on the attributes *Company* and *Name*, they will also have the same *Start date* and *End date* values. This is illustrated in Table 4 where the first and second row has the exact same *Start date* and *End date*. However, the attribute *Employee* is not a functional dependency since the following statement does not hold:

- Company, Name → Employee

Given the attributes *Company* and *Name*, a specific *Employee* can't be determined. It is visualized in Table 4, on the first and second row, where two different employees are connected to the same project. Therefore, the information about an employee should be placed into a separate table. An alternative solution is to include the *Employee* attribute as a key along with *Company* and *Name*, in order to allow projects to include several employees. (Garcia-Molina, 2009)

Company	**Name**	**Start date**	**End date**	**Employee**
Volvo	Engine V70	2012-01-01	2012-02-01	Per Ohldin
Volvo	Engine V70	2012-01-01	2012-02-01	Marcus Ek
Saab	Gearbox 9.3	2011-02-20	2011-08-20	Marcus Ek

Table 4: Bad design of a database table

2.2 Web development

The World Wide Web was created in year 1989 by Tim Berners-Lee. In late year 1990 Berners-Lee wrote the first web server and client program with a browser and an editor. He also wrote the first version of HTML which became the standardized formatting language for documents on the Web. In year 1994 Tim Berners-Lee founded W3C at MIT (Massachusetts Institute of Technology) in the United States and the organization develops standards for the World Wide Web. (*W3C*, 2012)

The demand of sophisticated and dynamic web pages has resulted in the implementation of JavaScript. HTML and JavaScript code can be written in the same document and interpreted by the web browser. The combination of HTML and JavaScript makes it possible to modify HTML code in real-time, enabling development of dynamic web pages. However, JavaScript is a *client side language* and cannot by itself receive data from a server. To implement real-time content on a web page a *server sided script* is needed and currently PHP is the most common one used. (Chapman, 2012)

A PHP script can receive requests from a JavaScript and perform tasks, for example fetch data from an external web server, and return the data to the JavaScript. The JavaScript inserts the received data into the client's HTML-document in order to display the content to the user.

2.2.1 Markup languages

The definition of *text*, in context of web development, often gets divided into two categories, *unordered structures* and *ordered structures.* An unordered structure only consists of plain text, while ordered structures includes more information in addition to the text. The ordered structure of documents is often split up into two parts, *layout* and *logic.* Layout describes visible parameters (colors, text sizes, margins etc.) while the logical part includes information regarding sections and references within the document. (Mounia, 2009)

The word *markup* means to highlight information and give it a context. In documents, opening and closing tags are used to mark information such as paragraphs, lists and headers. An opening tag has the formatting <book> meanwhile the closing tag has the following layout </book>, where the word book gives the tag a context. Markup languages can be used to convert unordered text into an ordered. The most commonly used markup languages are HTML, XHTML and XML and all of the languages follow the W3C standards. (Mounia, 2009)

2.2.1.1 HTML / XHTML

HTML stands for *Hypertext Markup Language* and is the basis, on which all information on the Internet uses to display content. A HTML document is divided into two parts, a set of instructions and content of information. The web browser needs the description in order to know how to display information to the user. (Brooks, 2007)

Extensible Hypertext Markup Language called XHTML is a further development of HTML, which combines XML and HTML. XHTML is supported by W3C and is a replacement for HTML 4.0. (Schwartz, 2000) XHTML demands a structure which is stricter compared to HTML. The main reason is to ease the workload and management on the web browsers, in order to avoiding misinterpretations. (Brooks, 2007)

Layout parameters need to be defined for all elements within a HTML document which can result in repetitive information. A *Cascading Style Sheet*, CSS, can be used to define layout parameters and apply them to multiple elements. (Schwartz, 2000)

2.2.2 Scripts

2.2.2.1 JavaScript

JavaScript is a script language running on the client side and can be used to create dynamic web sites. It is a powerful tool and applications like Google Docs and Google Calendar are built on the script (McPeak, 2010)

By using JavaScript, a static web site environment can be transformed into an interactive experience. A well written HTML code is not only important for the structure and presentation of a web page, but also for the communication with the JavaScript to be successful. (Goodman, 2010)

2.2.2.1.1 JQuery

JQuery is a JavaScript library facilitating the implementation of commonly used features for today's web applications. Actions written in JQuery requires less code writing for the developer compared to plain JavaScript. JQuery can perform basic actions such as event handling but also complex tasks such as dynamical modifications of web applications using AJAX, see section 2.2.3. Narayan (2011)

In Figure 2 and 3 two examples are presented, both performing the same task of reading an external file and inserting the information into a DOM object.

```
var xmlhttp;
xmlhttp=new XMLHttpRequest();
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("demo").innerHTML= \
                           xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","text.txt",true);
xmlhttp.send();
```

Figure 2: Code written in JavaScript
(*W3School*, 2012)

```
$("#demo").load(text.txt);
```

Figure 3: Code written in JavaScript with JQuery included
(*JQuery API:Load*, 2012)

2.2.2.1.2 JQuery UI

jQuery UI is an open source JavaScript library containing interface components and is based on the JQuery library. (Parker, 2011)

The components used in this thesis:

- Autocomplete - Provides suggestions as you type into a field
- Button - Theme support for buttons
- Datepicker - Interactive calendar overlay, simplifies input of a date into a field
- Dialog - A dialog window overlay, used to create dialogues
- Tabs - Put content into multiple sections and switch between them using tabs

2.2.2.1.3 DataTables

DataTables is an open-source plug-in for the jQuery Javascript library. It provides advanced interaction controls of HTML tables. (Jardine, 2011)

Example of key features:

- Variable length pagination
- On-the-fly filtering
- Multi-column sorting
- Support for themes

2.2.2.2 PHP

PHP, an abbreviation for *PHP: Hypertext Preprocessor*, is an open source project introduced in year 1995 by Rasmus Lerdorf. (MacInTyre, 2010)

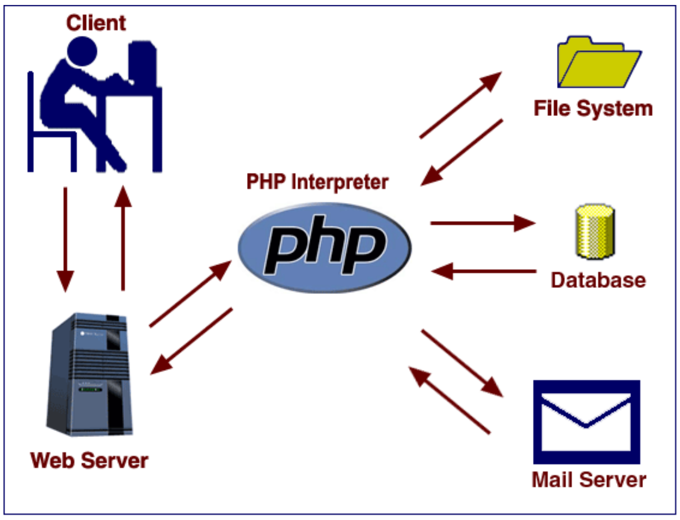


Figure 4: Basic overview showing PHP communication

PHP is a widely-used scripting language suited for web development. Contrary to JavaScript, PHP runs on the server-side, which requires the server to support PHP. As a PHP document is requested by a client, the server searches for embedded PHP sections to execute. As shown in Figure 4, the PHP interpreter can communicate with file systems, databases, and email servers before delivering a web page to the web server which return it to the client's browser.

In Figure 5 is a PHP-request to the server illustrated and the result returned to the client is visualized in Figure 6. (Welling, 2003)

```
<body>
<p> <?php echo date("Y-m-d"); ?> </p>
</body>
```

Figure 5: Code section read by server

In Figure 6, no trace can be seen that the PHP script has been modifying the HTML code. By using PHP, web sites can become dynamic compared to a static HTML document. (MacInTyre, 2010)

```
<body>
<p> 2012-06-21 </p>
</body>
```

Figure 6: Code section after PHP execution

PHP has support for various types of databases e.g. MySQL, Oracle, SQLite and MS SQL. The implementation of databases into PHP was released in year 1996 and currently companies like Facebook and Yahoo! use the technique. (MacInTyre, 2010)

2.2.3 Method: AJAX

At the beginning of the World Wide Web, updates of a HTML document were performed by sending a request and reload the entire document upon response. This caused screens to flicker and unnecessary large amount of data to be transferred each time a change was made. The AJAX technique has been able for use since year 1998 and solves this issue. AJAX is an acronym for *Asynchronous JavaScript and XML* and is a method used to send and receive data asynchronously on a web page, without disturbing the existing page. It has revolutionized the functionality of web applications. (Holzner, 2008)

AJAX consists of several technologies:

- Presentation using XHTML and CSS
- Dynamic visualization and interaction using DOM
- Data interchange and manipulation using XML and XSLT

- Asynchronous data retrieval using XMLHttpRequest
- JavaScript, binding all into one

The classic way for a user to interact with a web page is to perform an event, e.g. by clicking a link. A request will be sent to the server which responds by sending the corresponding HTML page back to the user. In some cases, it can be appropriate to reload the entire HTML page. However, it is often not needed and can result in unnecessary large amount of data being transmitted. By only updating specified parts of the HTML page the amount of data can be reduced.

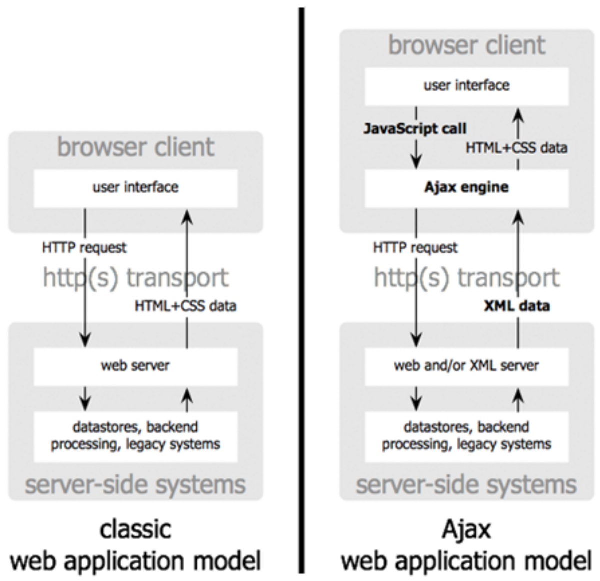


Figure 7: Traditional web applications model compared to the Ajax model (Garret, 2005)

AJAX works as a layer between the user browser and the web server. As a user interacts with a web page and generates a HTTP request, the JavaScript sends the appropriate request to the AJAX engine. The AJAX engine can handle several requests as background transactions making updates of a web page faster, see Figure 7 and 8. (Garret, 2005)

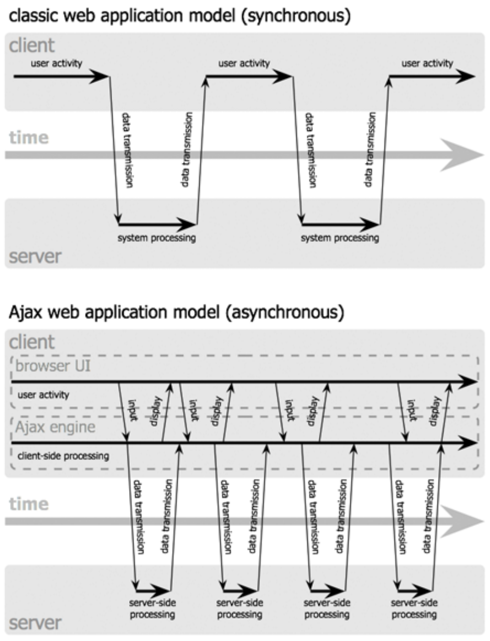


Figure 8: Synchronous and asynchronous communication (Garret, 2005)

3 Methodology

The methodology section aims to describe the work flow during the Master thesis.

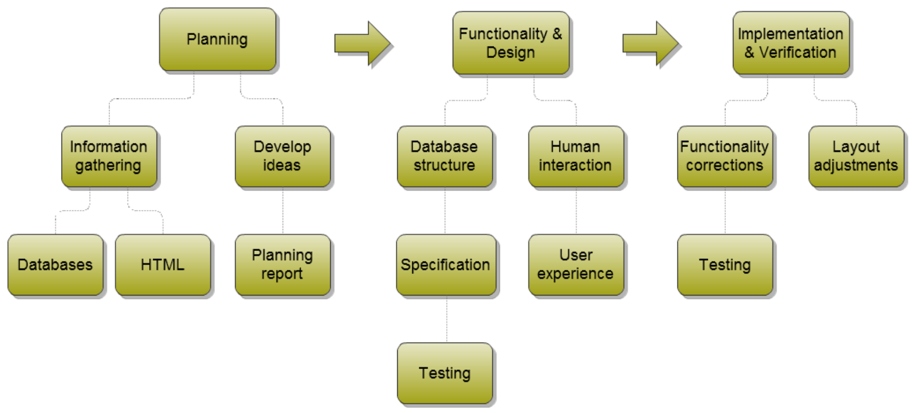


Figure 9: Box-diagram showing the working process

3.1 Planning

The planning process involved information gathering regarding the main components, databases and HTML. Performing tutorials proved to be important to obtain knowledge about HTML and associated tools e.g. JQuery, AJAX, XML, XHTML, PHP, CSS and XQuery in order to understand the possibilities.

To elucidate the functionalities being useful for QRTECH was an ongoing process and it was carried out in parallel with the information gathering. Interviews with employees at QRTECH provided useful information for the process of developing new ideas.

As the functionalities was established a planning report was created. The report included background, method, objective, delimitations and a Gantt chart visualizing the project time line, see Appendix D.

3.2 Functionality and Design

In the functionality and design phase, the development of the actual product began.

The design of the database included development of an ER-diagram and evaluation of the FDs (Functional Dependencies). The database requirements was determined and validated during the specification and testing steps. Testing implementations permeated the entire project and was continuously performed. The user experience and human interaction abilities are important and the software solution has to be intuitively easy to use.

During the entire project this was constantly kept in mind, in order for the final web application to be as qualitative as possible.

3.3 Implementation and Verification

The last phase is Implementation and verification and consists of the elements Functionality corrections, Testing and Layout adjustments. During this period, the work consisted of testing and debugging in order to correct errors and adjust settings and layout. By letting employees evaluate the web application, useful feedback can hopefully be provided in order to improve the final product.

4 Analysis of database structure

This section discuss database ideas, the structure chosen and also present detailed descriptions on how the most vital parts of the database are constructed.

4.1 Choice of DBMS

A minor investigation was performed at the beginning of this thesis to determine which DBMS was most suitable to utilize for developing the database. There exists many different DBMS for example MySQL, Oracle and MS SQL. A vital property in this thesis was to create a complete database solution while still keeping the expenses low. The best solution in this case was to utilize the open source software MySQL, since it does not require any license fees.

4.2 Ideas

The objective is to construct a database model which excludes redundancy of information and facilitates the user experience. In the database development process the two main ideas *Experience tree* and *Experiences and tags* evolved.

4.2.1 Experience tree

The initial approach was to divide experiences into categories, see Figure 10. The strategy of using a tree structure is a top-down approach, meaning searches will be performed from the top categories stepwise down in the experience tree in order to find the result.

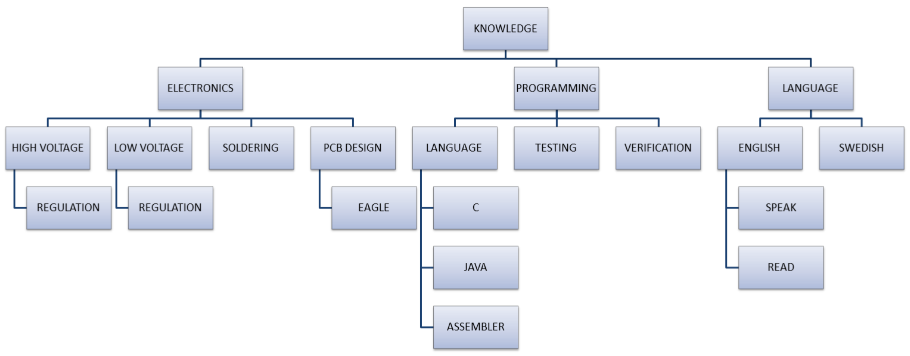


Figure 10: The structure of *Experience tree* idea

4.2.1.1 Inserting data

Employees at QRTECH could insert information into the tree by add new categories. A category could work as an experience by itself but also as a branch, containing one or more child-experiences, enabling the experience tree to grow over time.

4.2.1.2 Search

The search operations could be performed on the entire database or on specific categories. The user could restrict in which categories the search would be executed within by using scroll lists consisting of the categories available in the experience tree. A search field enables users to request information from the database.

4.2.1.3 Advantages

- Intuitively easy to understand
- Good database performance when searches are performed

4.2.1.4 Disadvantages

- Users have subjective opinions regarding where a certain experience should be placed in the tree. This could make it hard to categorize experiences uniformly and the same experience can exist at multiple places in the tree
- Inserting an experience requires the user to put additional time and effort to plan *where* the experience should be positioned in the tree, creating a reduced user experience
- High redundancy in the database

4.2.2 Experiences and tags

A concept to not categorize experiences emerged, confronting the problem from a different angle. By using keywords called *tags* to describe experiences, the idea *Experiences and tags* manages to avoid categorization.

Instead of using a top-down approach, the *Experiences and tags* idea utilizes the reversed approach bottom-up. This means, employees at QRTECH register an experience and uses tags to describe it, see Figure 11. Experiences are detailed information, since the approach is bottom-up, whereas tags can range from being general to detailed data.

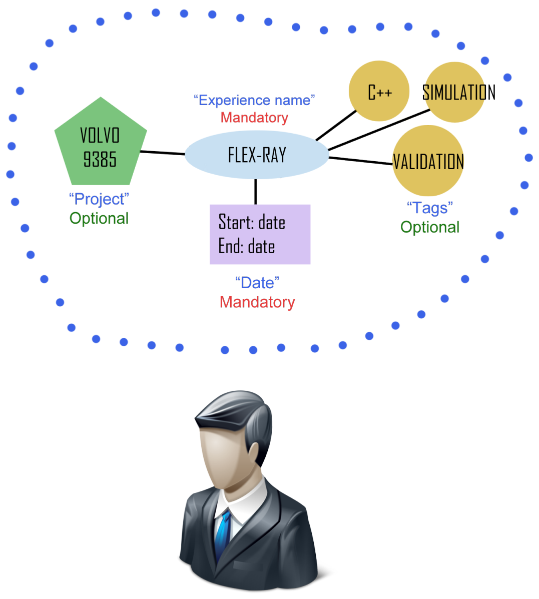


Figure 11: The structure of *Experience and tags* idea

4.2.2.1 Inserting data

An experience could be registered into the database e.g. FLEX-RAY in Figure 11. The mandatory parts of an experience are the *name* of the experience, *start-date* and *end-date*. The two dates are an important in order to create a time span which is used to define the weight of the experience as a search for experiences is performed.

Tags and projects are optional and can be applied to the experience in order to create a context for the experience. An experience can have zero or an infinitely number of tags depending on how well the user wants to describe it.

4.2.2.2 Search

By connecting several tags when inserting experiences, employees contribute to create an environment where colleagues with the requested knowledge easily can be found. Adding many tags increase the possibility to appear on the ranking board as searches are performed in the database.

4.2.2.3 Advantages

- Easy to describe an experience by connecting tags
- Improved user experience by avoiding categorization
- Low redundancy

4.2.2.4 Disadvantages

- The approach down-up can be complicated in the beginning for users to grasp
- Lower database performance, due to the data mining approach and the increased amount of data it needs to process

4.3 Choice of database structure

The *Experiences and tags* idea was chosen to be the database structure. The main reason was the dynamical approach to define an experience. By letting users connect tags to an experience, complex experiences could be inserted which otherwise would be hard, or even impossible, to categorize when using the *Experience tree* approach. An additional reason was the low redundancy of information in the database.

4.4 Database design

This section aims to describe the most vital parts of the database by using ER-diagrams, see Figure 12. An ER-diagram visualizing the entire database structure can be viewed in Appendix B.

4.4.1 ER-Diagram

In the following section, the database design in Figure 12 will be described.

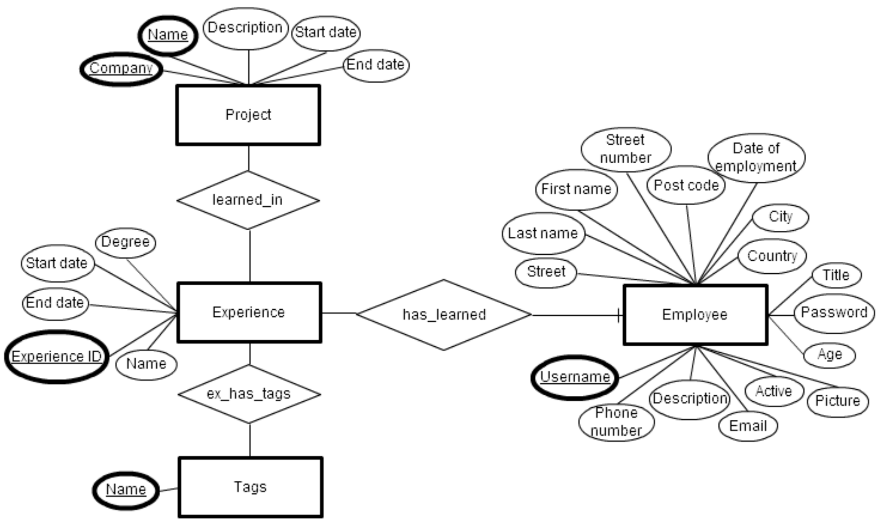


Figure 12: ER-diagram describing the most vital relations in the database

The core relationship of the database is the one between the tables Experience and Employee since it gives an experience a context meaning who obtained this experience. By using the Experience relation, data regarding projects and tags can be associated with an experience and thereby also an employee.

4.4.1.1 Employee

The relation Employee consists of the attributes *First name*, *Last name*, *Username*, *Password*, *Phone number*, *Street*, *Post code*, *Street number*, *City*, *Title*, *E-mail*, *Age*, *Country*, *Date of employment*, *Description*, *Active* and *Picture*, see Figure 12. All personnel related information will be stored in this relation. An employee is uniquely identified by its key attribute, *Username*, needed in order to log-in to the web site.

4.4.1.2 Has_learned

The table Employee is connected to the table Experience via the relation has_learned, see Figure 13. The two tables are connected by a many-to-one relation, meaning an employee can be attached to several experiences meanwhile an experience can only be associated with exactly one employee. If an experience ID is known, the information regarding the employee who obtained it is also known. The underlying concept of the database structure is to view each experience as unique, meaning it has to be obtained by exactly one employee.

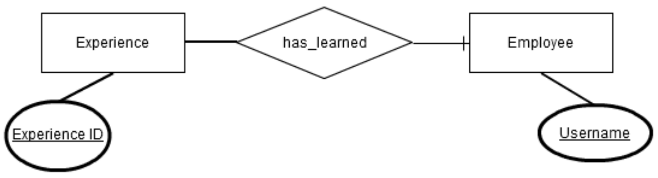


Figure 13: ER-diagram on the relation *has_learned*

The relation has_learned consists of the key attributes of the tables Employee and Experience, see Table 5. It illustrates users with specific experiences and enables the connection between the two tables Employee and Experience.

Experience ID	**Username**
1	Andohl
2	Marwel
3	Andohl

Table 5: Example on the relation *has_learned*

4.4.1.3 Experience

The relation Experience consists of the attributes *Experience ID*, *Experience name*, *Degree*, *Start date* and *End date* and *Employee username*, visualized in Figure 14. *Experience ID* represents the key and is an auto-incremental integer. The attribute continuously increase the integer as new experiences are inserted into the relation making each experience unique.

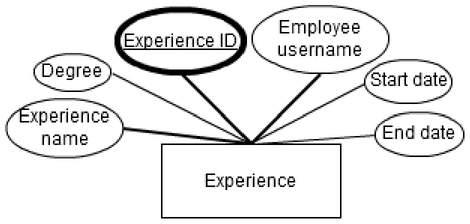


Figure 14: ER-diagram on the relation *Experience*

4.4.1.4 Ex_has_tags

The tables Experience and Tags are connected via the relation ex_has_tags using a many-to-many relationship, see Figure 15.

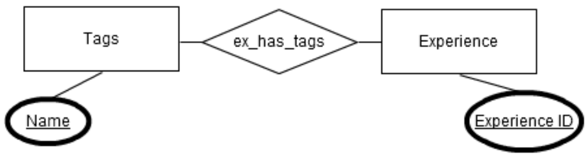


Figure 15: ER-diagram on the relation *ex_has_tags*

Table 6 illustrates the relation ex_has_tags with example data. The relation consists of tuples with the keys Experience ID and Tags, where none of the tuples are equal.

Tags	**Experience ID**
Math	1
Simulink	2
MySQL	3

Table 6: Example on the relation *ex_has_tags*

4.4.1.5 Tags

The table Tags only consist of one attribute which defines the tag name and it is also the key of this relation.

4.4.1.6 Learned_in

The relation learned_in establishes a connection between the tables Experience and Project by creating a table consisting of the keys *Experience ID*, *Company* and *Name*. The connection is a many-to-many relationship and Table 7 displays the structure of the relation learned_in.

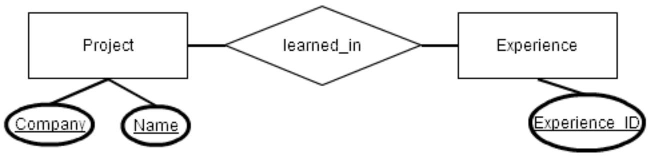


Figure 16: ER-diagram on the relation *learned_in*

Company	**Name**	**Experience ID**
Volvo	Engine V70	1
Saab	Gearbox 9.3	2
Volvo	Gearbox S60	3

Table 7: Example on the relation *learned_in*

4.4.1.7 Project

Project is a table consisting of the attributes *Company*, *Name*, *Start date* and *End date* where Company and Name together represent the key in the relation.

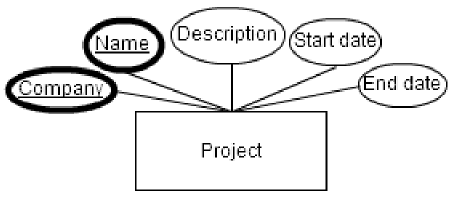


Figure 17: ER-diagram on the relation *Project*

5 Web site implementation

This section aims to describe the functionality, development and implementation of the web site.

5.1 Choice of server script language

A server-sided script was necessary in order to establish connections between the web site and the database. Today, several solutions exist on the market such as PHP, ASP and Python. The choice was determined to be PHP since it is open source and free to use. Additionally, PHP is used by enterprises like Yahoo! and Facebook which demonstrates that it is widely utilized.

5.2 Overview

The overview chapter demonstrates the web site structure, how communications are performed between the web site and database using PHP and the overall functionality.

5.2.1 Structure

The web site consists of fourteen html-files and two php-files. The html-files are stand alone solutions, each designed for one specific task. By importing these solutions into the file, *index.php*, the functionality is joined and combined into one unit, see Figure 18.

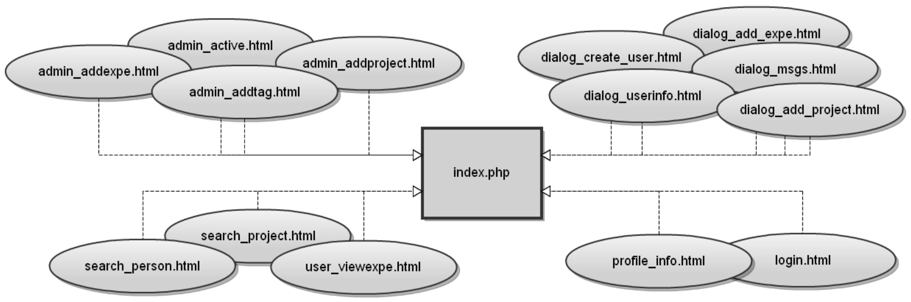


Figure 18: Overview showing all html-files included to build *index.php*

5.2.2 Database queries

To ease the maintenance, all queries from the web site to the MySQL database are handled by one php-file named *PHPQuery.php*, see Figure 19. Requests sent to PHPQuery.php consists of the necessary variables and a String, defining which function to execute. For example, as a user attempts to login, index.php includes the entered username and password together with the string ”check_login” as variables. ”check_login” is in this given case the function string. PHPQuery.php performs the query on the mySQL database and take action depending on the result it receives. In this particular case, the action is to return **true** if the username-password combination exists in the database, otherwise return **false** to index.php.

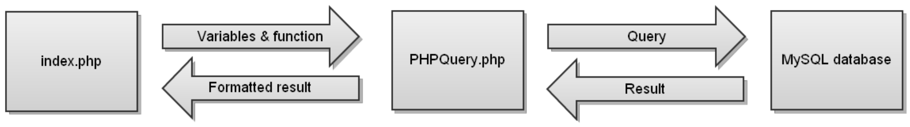


Figure 19: Overview showing how PHPQuery.php handles web site/database communication

5.3 In-depth description

This chapter describes the web site features more thoroughly, see Figure 20 to view the web site layout.

5.3.1 Navigation

Navigation buttons are displayed on the top of the web site, see Figure 20. The Search button, Figure 21a, is always available for users and is not affected by the status of being logged in or not. The Profile button, Figure 21b, is merely available as a user is logged in and enables the employees to view and edit personal experiences and information. The button is blurred in order to visualize that the feature is disabled to users not being logged in, demonstrated in Figure 20. The navigation bar also consists of the Admin button, see Figure 21c, merely visible to users logged in as administrator. Main reason for not displaying it until a user logs in as administrator is to avoid confusion.

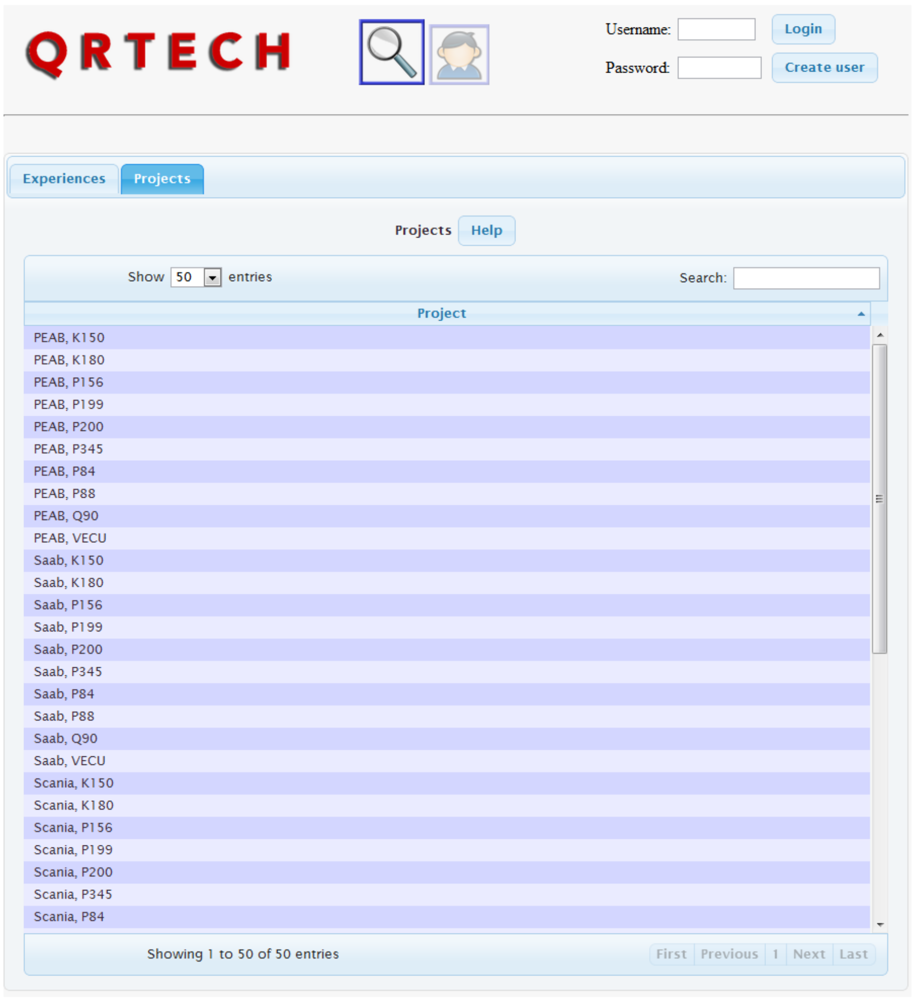
5.3.2 Search

As the navigation button Search is clicked, see Figure 21a, the user will be presented a page consisting of the two tabs Experiences and Projects.

5.3.2.1 Experiences

The purpose of the experience search feature is to enable employees to find colleagues possessing the knowledge requested. In Figure 22 the experience search view is visualized and it consists of the following features:

• Help button	- show/hide information regarding the search syntax
• Search field	- keywords are entered here to perform searches
• Data table	- contains all persons matching the entered keywords


QRTECH

Username: Login

Password: Create user

Experiences | Projects

Projects Help

Show 50 entries Search:

Project
PEAB, K150
PEAB, K180
PEAB, P156
PEAB, P199
PEAB, P200
PEAB, P345
PEAB, P84
PEAB, P88
PEAB, Q90
PEAB, VECU
Saab, K150
Saab, K180
Saab, P156
Saab, P199
Saab, P200
Saab, P345
Saab, P84
Saab, P88
Saab, Q90
Saab, VECU
Scania, K150
Scania, K180
Scania, P156
Scania, P199
Scania, P200
Scania, P345
Scania, P84

Showing 1 to 50 of 50 entries First Previous 1 Next Last

Figure 20: Screen-shot from the Experience database web interface. Currently showing the project search tab

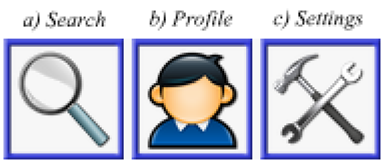
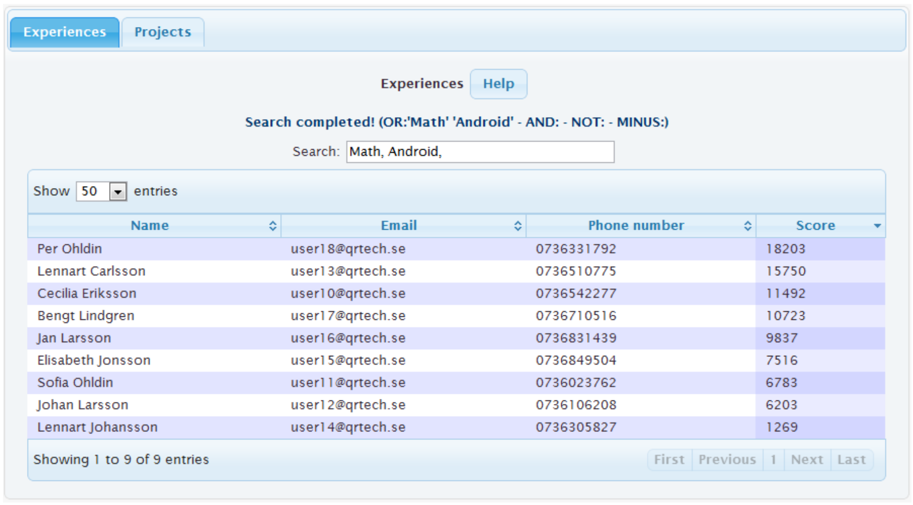


Figure 21: Navigation buttons

Searching for experience is achieved by typing keywords into the search field. To improve the user experience, an Autocomplete function is continuously giving the user suggestions from the database as he/she types into the search field. Multiple searches on experiences can be performed by separating words with a comma token. The following four operands can be used to



Search completed! (OR:'Math' 'Android' - AND: - NOT: - MINUS:)

Search: Math, Android,

Show 50 entries

Name	Email	Phone number	Score
Per Ohldin	user18@qrtech.se	0736331792	18203
Lennart Carlsson	user13@qrtech.se	0736510775	15750
Cecilia Eriksson	user10@qrtech.se	0736542277	11492
Bengt Lindgren	user17@qrtech.se	0736710516	10723
Jan Larsson	user16@qrtech.se	0736831439	9837
Elisabeth Jonsson	user15@qrtech.se	0736849504	7516
Sofia Ohldin	user11@qrtech.se	0736023762	6783
Johan Larsson	user12@qrtech.se	0736106208	6203
Lennart Johansson	user14@qrtech.se	0736305827	1269

Showing 1 to 9 of 9 entries

Figure 22: Search on experiences to find employees

constrain the search further:

- & = AND *syntax: Math, &Simulink*
- ! = NOT *syntax: Math, !Simulink*
- - = MINUS *syntax: Math, -Simulink*

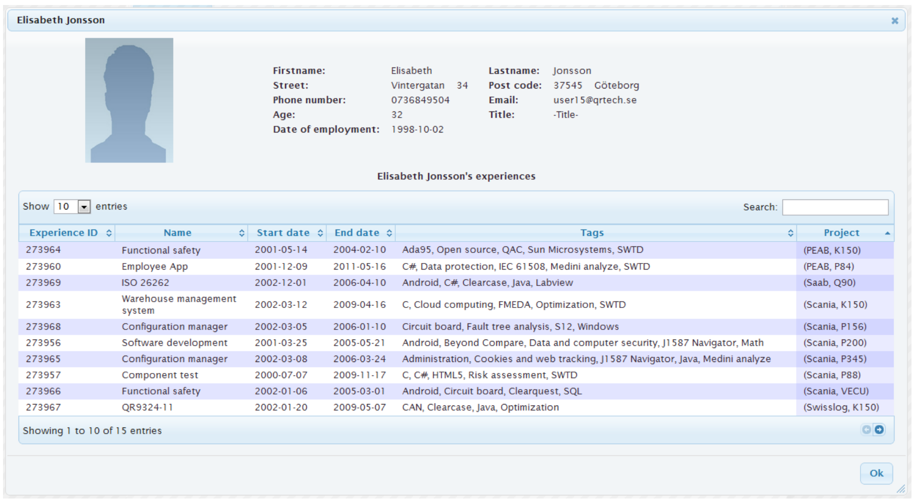
The result from a search consists of the employee name, email, phone number and score. The score is based on the total number of days the employee has been working with the experience requested.

The user has the ability to click on a row to trigger a dialogue displaying additional information about the specific user. The triggered dialogue contains contact information and experiences associated with the user, see Figure 23.

The flowchart in Figure 24 is used to illustrate the experience search algorithm used in the Experience database. The search process is initialized when the user enter a character in the search field. Instead of performing the actual search directly, a timer is started and waits 200ms before performing the search. If an additional character is entered in the search field the timer resets itself. This reduces the number of calls to database when the user is writing and the delay is barely noticeable to the user.

What *Perform search*, showed in Figure 24, is doing in detail is to:

1. Take the characters written in the search field and separate into words, based on comma (,) placement
2. Place each word into one of the four categories OR, AND, NOT or MINUS based on the prefix the word has. ("nothing", "&", "!" or "-")



Elisabeth Jonsson

Firstname:	Elisabeth	**Lastname:**	Jonsson
Street:	Vintergatan 34	**Post code:**	37545 Göteborg
Phone number:	0736849504	**Email:**	user15@qrtech.se
Age:	32	**Title:**	-Title-
Date of employment:	1998-10-02		

Elisabeth Jonsson's experiences

Show 10 entries Search:

Experience ID	Name	Start date	End date	Tags	Project
273964	Functional safety	2001-05-14	2004-02-10	Ada95, Open source, QAC, Sun Microsystems, SWTD	(PEAB, K150)
273960	Employee App	2001-12-09	2011-05-16	C#, Data protection, IEC 61508, Medini analyze, SWTD	(PEAB, P84)
273969	ISO 26262	2002-12-01	2006-04-10	Android, C#, Clearcase, Java, Labview	(Saab, Q90)
273963	Warehouse management system	2002-03-12	2009-04-16	C, Cloud computing, FMEDA, Optimization, SWTD	(Scania, K150)
273968	Configuration manager	2002-03-05	2006-01-10	Circuit board, Fault tree analysis, S12, Windows	(Scania, P156)
273956	Software development	2001-03-25	2005-05-21	Android, Beyond Compare, Data and computer security, J1587 Navigator, Math	(Scania, P200)
273965	Configuration manager	2002-03-08	2006-03-24	Administration, Cookies and web tracking, J1587 Navigator, Java, Medini analyze	(Scania, P345)
273957	Component test	2000-07-07	2009-11-17	C, C#, HTML5, Risk assessment, SWTD	(Scania, P88)
273966	Functional safety	2002-01-06	2005-03-01	Android, Circuit board, Clearquest, SQL	(Scania, VECU)
273967	QR9324-11	2002-01-20	2009-05-07	CAN, Clearcase, Java, Optimization	(Swisslog, K150)

Showing 1 to 10 of 15 entries

Figure 23: Specific employee information view

3. Send the categorized words to PHPQuery.php, which constructs a MySQL query and executes it on the database.
4. When the result is recieved from the server, PHPQuery formats the data so the JavaScript easily can insert it into a data table
5. PHPQuery sends the formatted data to the JavaScript

5.3.2.2 Projects

All projects performed at QRTECH are inserted into a data table on which searches can be executed. The data table presents the information in a list where each row consists of a composition of the company name and the actual project name, see Figure 20.

The data table shown in Figure 20 has an additional column, not visible to the user. The column consists of a paired list of all the *expereince names* and *tags* connected to the project together with the *firstname*, *lastname*, *email* and *username* of the users that have registered at least one experience to the specific project. An example of how the hidden column can look like is illustrated in Figure 25. The mechanism behind a project search is a string compare between the words entered in the search field and the keywords in the hidden column; If a project does not have the keywords entered, it is filtered out and immediately removed from the list.

Each row in the data table is clickable enabling the web site to display a dialogue with additional project information, see Figure 26.

At the top of the dialogue a header displays the unique combination of the company name and the project name. Additional information is added underneath such as start and end date, a description and two tables consisting of tags and employees associated with the specified project. The

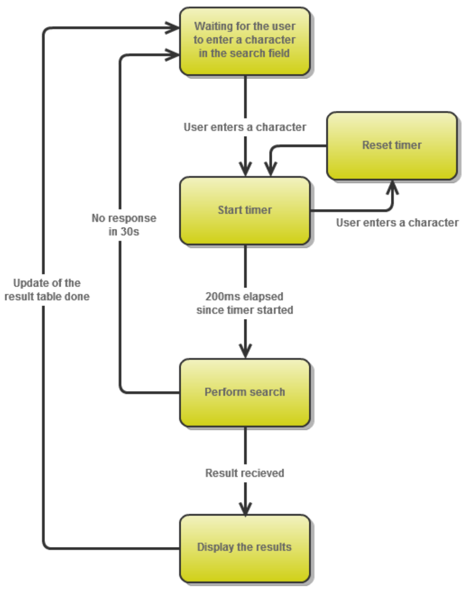
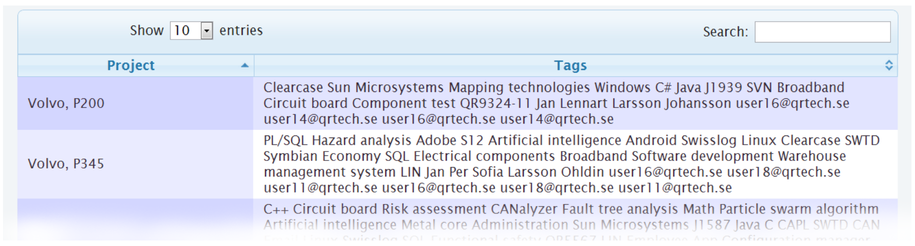


Figure 24: Flowchart illustrating the search algorithm used for experience search

tag table display tags associated with the project and how frequently these occur. The employee table presents essential data about the employees involved in the project. As a row in the employee table is clicked a dialogue is triggered, displaying profile information regarding the specific user. This function is equal the event in section 5.3.2.1, where contact information and experiences of a unique user can be viewed, see Figure 23.

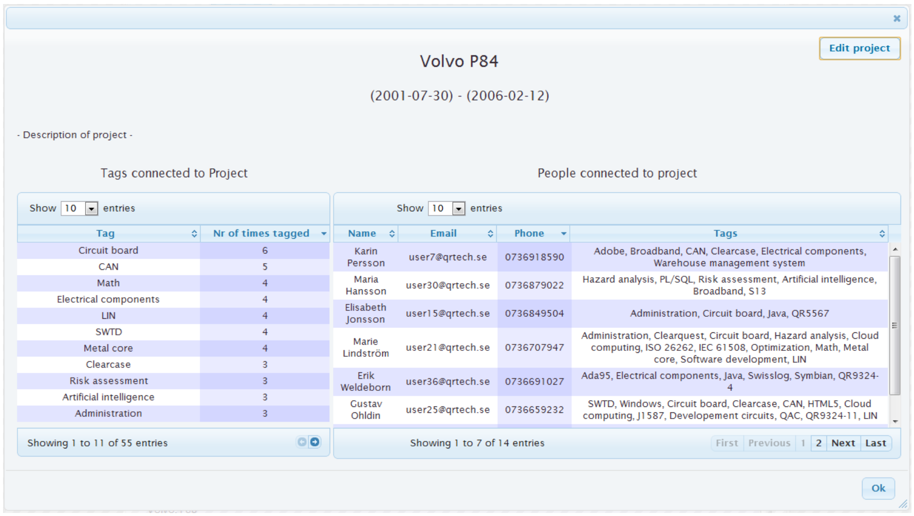
The Edit project button enables logged in users to edit project information, see Figure 27. The attributes *Project company* and *Project name* are keys in the database, meaning the combination of the two attributes has to be unique. Therefore, validating the new project combination is necessary since updates otherwise wouldn't be accepted by the database. In



Show 10 entries Search:

Project	Tags
Volvo, P200	Clearcase Sun Microsystems Mapping technologies Windows C# Java J1939 SVN Broadband Circuit board Component test QR9324-11 Jan Lennart Larsson Johansson user16@qrtech.se user14@qrtech.se user16@qrtech.se user14@qrtech.se
Volvo, P345	PL/SQL Hazard analysis Adobe S12 Artificial intelligence Android Swisslog Linux Clearcase SWTD Symbian Economy SQL Electrical components Broadband Software development Warehouse management system LIN Jan Per Sofia Larsson Ohldin user16@qrtech.se user18@qrtech.se user11@qrtech.se user16@qrtech.se user18@qrtech.se user11@qrtech.se
	C++ Circuit board Risk assessment CANalyzer Fault tree analysis Math Particle swarm algorithm Artificial intelligence Metal core Administration Sun Microsystems J1587 Java C CAPL SWTD CAN

Figure 25: Showing a part of the project search view. This Figure is specially made for the report and the Tags-column is normally not visible. (referred to as *hidden column* in the report)



Volvo P84

(2001-07-30) - (2006-02-12)

- Description of project -

Tags connected to Project

Show 10 entries

Tag	Nr of times tagged
Circuit board	6
CAN	5
Math	4
Electrical components	4
LIN	4
SWTD	4
Metal core	4
Clearcase	3
Risk assessment	3
Artificial intelligence	3
Administration	3

Showing 1 to 11 of 55 entries

People connected to project

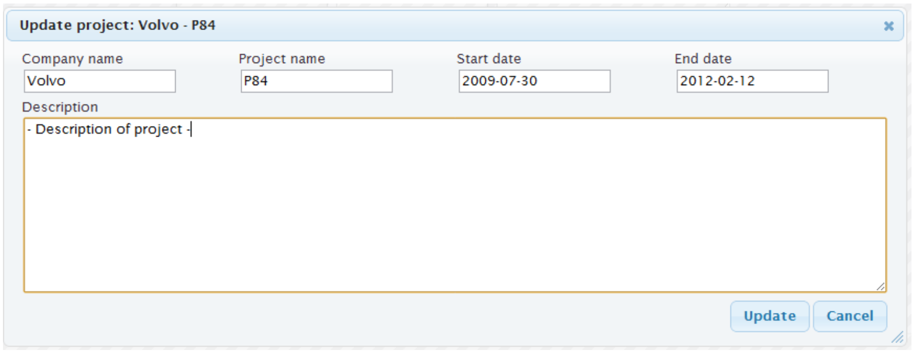
Show 10 entries

Name	Email	Phone	Tags
Karin Persson	user7@qrtech.se	0736918590	Adobe, Broadband, CAN, Clearcase, Electrical components, Warehouse management system
Maria Hansson	user30@qrtech.se	0736879022	Hazard analysis, PL/SQL, Risk assessment, Artificial intelligence, Broadband, S13
Elisabeth Jonsson	user15@qrtech.se	0736849504	Administration, Circuit board, Java, QR5567
Marie Lindström	user21@qrtech.se	0736707947	Administration, Clearquest, Circuit board, Hazard analysis, Cloud computing, ISO 26262, IEC 61508, Optimization, Math, Metal core, Software development, LIN
Erik Weldeborn	user36@qrtech.se	0736691027	Ada95, Electrical components, Java, Swisslog, Symbian, QR9324-4
Gustav Ohldin	user25@qrtech.se	0736659232	SWTD, Windows, Circuit board, Clearcase, CAN, HTML5, Cloud computing, J1587, Developement circuits, QAC, QR9324-11, LIN

Showing 1 to 7 of 14 entries

Figure 26: Project information dialogue

case, an issue occurs an error dialogue is displayed to the user explaining the situation.



Update project: Volvo - P84

Company name	Project name	Start date	End date
Volvo	P84	2009-07-30	2012-02-12

Description

- Description of project -

Figure 27: Update project information view

5.3.3 Profile

5.3.3.1 Create account

New visitors at the web site, employed by QRTECH, have the ability to register a personal account. As the button Create user is clicked, see the top right corner in Figure 20, a dialogue appears, see Figure 28. By filling out the form and pressing Create account a request is sent to the database to create the account. Validation of the form information is performed and the input data needs to be configured according to the following rules in order to be accepted:

- First and last name can only consist of letters, spaces and dashes
- The entered strings in the two password fields has to be equal
- The email need to have the syntax of an email and be unique in the database since will also be the username for the new user

The user will automatically be logged in as a new account have been registered and approved.

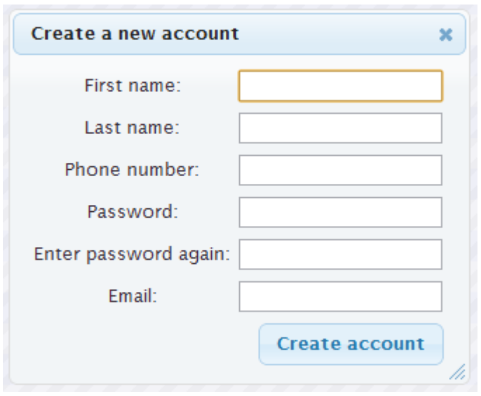


Figure 28: Create user dialogue

5.3.3.2 Login

As a user tries to login a request is sent to the database to validate the username and password entered and the database returns either 0 or 1.

- Returns 0 – Access denied
- Returns 1 – Access granted

Entering the incorrect information will trigger the web site to display an error dialogue. However, if access is granted, the username and password is stored as a cookie and will be automatically filled in next time the user visits the web site. Enable blurred or hidden buttons and change the current view will be performed as the user logs in. A user will be navigated to its profile page meanwhile the administrator will be presented the administrator page.

5.3.3.3 Profile page

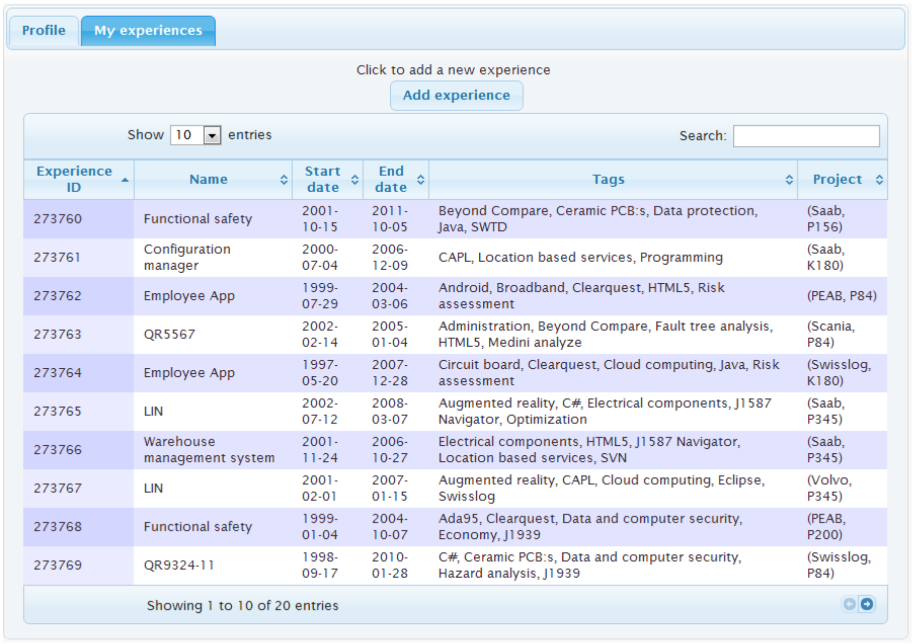
This view is presented to the user as he/she clicks the Profile navigation button, see Figure 21b, and enables users to view their profile page and experiences. Notice, the user has to be logged in to be able to utilize these functions.

An employee has the possibility to change the attributes *first name, last name, date of employment, street, post code, street number, city, country, password, title, age* and *email*. This is done by pressing the edit button. A dialogue will then appear with fields containing the current values of each attribute. The user has the possibility to change and save the attributes.

5.3.3.4 Personal Experiences

The Experience tab, presented in Figure 29, visualize all personal experiences associated with a given user. The list of experiences provide information regarding ID number, name, start- and end date, tags and projects associated with each given experience.

The search field can be used to filter the displayed experiences depending on keywords entered. The filter mechanism updates the data table continuously as the user types on the keyboard.



Click to add a new experience

Experience ID	Name	Start date	End date	Tags	Project
273760	Functional safety	2001-10-15	2011-10-05	Beyond Compare, Ceramic PCB:s, Data protection, Java, SWTD	(Saab, P156)
273761	Configuration manager	2000-07-04	2006-12-09	CAPL, Location based services, Programming	(Saab, K180)
273762	Employee App	1999-07-29	2004-03-06	Android, Broadband, Clearquest, HTML5, Risk assessment	(PEAB, P84)
273763	QR5567	2002-02-14	2005-01-04	Administration, Beyond Compare, Fault tree analysis, HTML5, Medini analyze	(Scania, P84)
273764	Employee App	1997-05-20	2007-12-28	Circuit board, Clearquest, Cloud computing, Java, Risk assessment	(Swisslog, K180)
273765	LIN	2002-07-12	2008-03-07	Augmented reality, C#, Electrical components, J1587 Navigator, Optimization	(Saab, P345)
273766	Warehouse management system	2001-11-24	2006-10-27	Electrical components, HTML5, J1587 Navigator, Location based services, SVN	(Saab, P345)
273767	LIN	2001-02-01	2007-01-15	Augmented reality, CAPL, Cloud computing, Eclipse, Swisslog	(Volvo, P345)
273768	Functional safety	1999-01-04	2004-10-07	Ada95, Clearquest, Data and computer security, Economy, J1939	(PEAB, P200)
273769	QR9324-11	1998-09-17	2010-01-28	C#, Ceramic PCB:s, Data and computer security, Hazard analysis, J1939	(Swisslog, P84)

Showing 1 to 10 of 20 entries

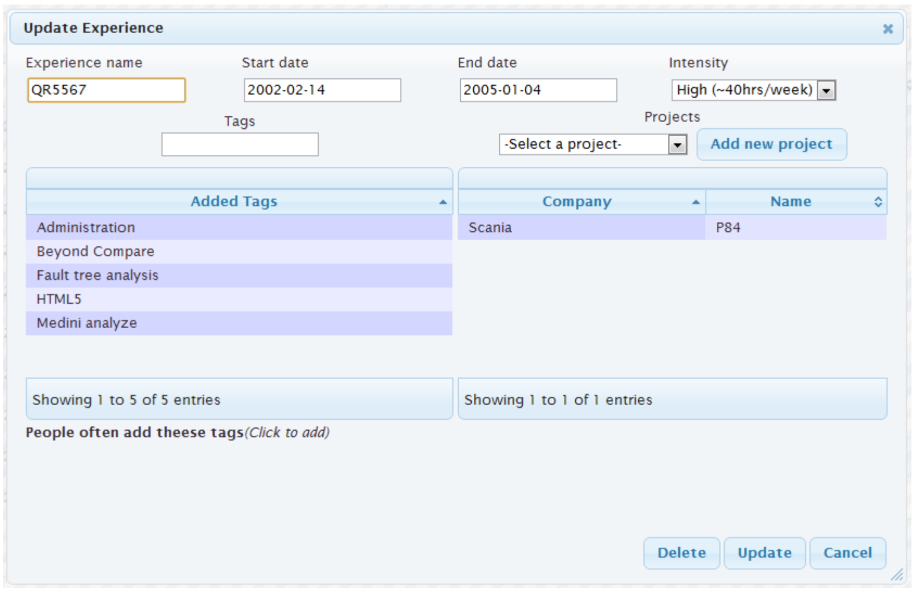
Figure 29: All personal experiences view

The button *Add experience*, in Figure 29, makes it possible for users to add new experiences by entering data into a dialogue. The dialogue has the similar appearance as the Update experience dialogue, see Figure 30. However, the difference is that the input fields are empty and the Update button is exchanged to an Add button.

Users can edit registered experiences by clicking the corresponding row in the data table. A dialogue is then displayed, see Figure 30, containing input fields with experience information gathered from the database. The following attributes are editable:

- Experience name - As the experience name is entered, suggestions of commonly used tags will be automatically updated and presented. The field has Auto-complete functionality.
- Start/End date - Define the time frame for the experience. The timespan between the two dates is used in the algorithm when searching for experiences. When the input field is clicked a calendar will appear making it simple for the user to insert a date.
- Intensity - This attribute defines the workload. It is selected from a scroll list, where *low* means 10 hours per week, *medium* 20 hours per week and *high* 40 hours per week.
- Projects - An experience can be connected to one or more projects by making selections from a scroll list containing all projects in the database. All users can create a new project by clicking the button *Add new project*.
- Tags - Multiple tags can be added to the experience by writing the tag name into the Tag field and press enter.

As the information in the dialogue is inserted the user clicks the update button to save it. The delete button removes the experience and all it's connections to projects and tags.



Experience name: QR5567
Start date: 2002-02-14
End date: 2005-01-04
Intensity: High (~40hrs/week)
Tags:
Projects: -Select a project-

Added Tags
Administration
Beyond Compare
Fault tree analysis
HTML5
Medini analyze

Showing 1 to 5 of 5 entries

Company	Name
Scania	P84

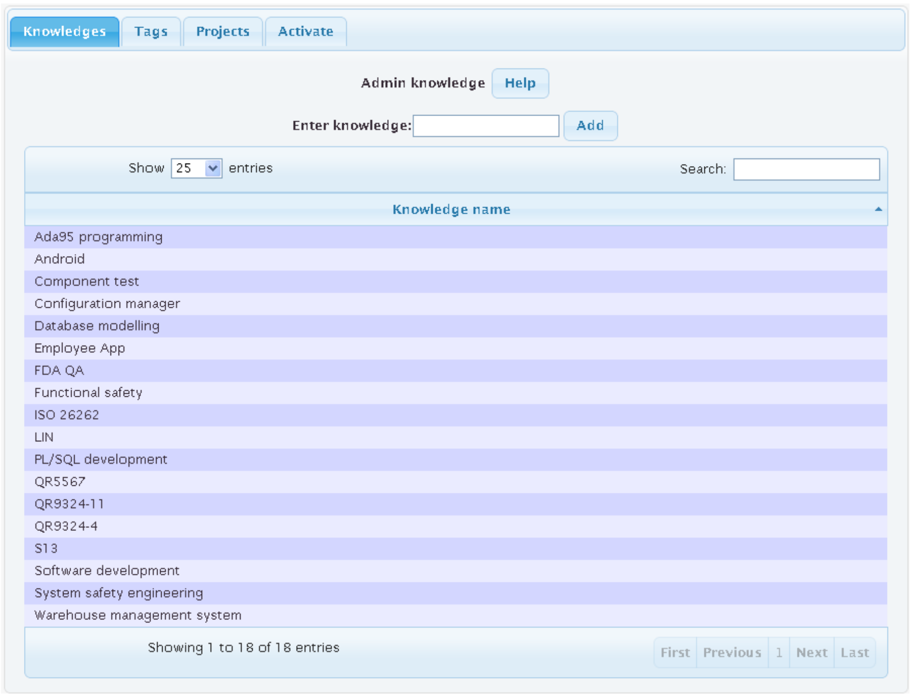
Showing 1 to 1 of 1 entries

People often add theese tags*(Click to add)*

Figure 30: Update experience information view

5.3.4 Administration

As an user is logged in as administrator the Admin button, see Figure 21c, will be visible. The administrator section consists of the four tabs Experience, Tag, Project and Activate, see Figure 31.



Knowledge name
Ada95 programming
Android
Component test
Configuration manager
Database modelling
Employee App
FDA QA
Functional safety
ISO 26262
LIN
PL/SQL development
QR5567
QR9324-11
QR9324-4
S13
Software development
System safety engineering
Warehouse management system

Showing 1 to 18 of 18 entries

Figure 31: Administrator view

5.3.4.1 Experience, Tag and Project

The main idea with the administrator experience, tag and project features is to provide the ability to edit and populate the experience database with suggestions. The database will initially be empty and the Autocomplete functionality will therefore not be able to give the users any guidance when entering information. This makes it harder for the users to use common language when performing various actions, for instance searching or adding experiences. To preprocess the database and solve the issues, three new tables were created in the database for administrator use only. This enables the administrator to create fictional experience, tags and projects, which will become real when the first user uses them. The administrator can then give the users suggestions, even though the database is empty, and thereby reduce redundancy.

Apart from the ability to add and remove fictional experiences, tags and projects the administrator account make it possible to perform changes on experience-, tag- and project names. These changes are updated on all its connections. For example, changing the tag name *Electrical* to *Electronics* will trigger the database to cascade updates on all elements having the tag *Electrical*. If the name is changed to a already existing name i the database, they get merged into one.

5.3.4.2 Activate/Deactivate employees

As a new user creates an account at the web site, the person is by default set to active in the database. Active is an attribute which each employee has and it is either 1 or 0 depending on if the employee is currently working at QRTECH or not.

The activate/deactivate functionality enables the administrator to view the employees currently working at QRTECH and former employed personnel. The administrator also has the authority to change the current status of employees between the two states *active* or *inactive*. Inactivated employees are not included as searches for experience are performed. However, the experiences gained by former employees are still stored in the database to data mine info about projects and in case persons return to QRTECH in the future. Their experiences and tags are also utilized by the system to give current employees suggestions and guidance regarding existing information in the database.

6 Performance evaluation

Section 6.1 and 6.2 describes how the tests were performed on the experience database in order to evaluate the performance.

6.1 Collecting data

This section explains how information was gathered by logging actions performed by users on the web site. The two types of logging implemented are *Query time logging* and *Usage logging.*

6.1.1 Query time logging

By registering information such as query time the performance of the database queries can be investigated. An example of how a part of the Query time log can look like is visualized in Table 8.

Each row in the Query time log consists of four attributes:

- Id - Uniquely identifies each row in the log
- Date - *When* was the call made (date and time)
- Query time - The time it took for the database to execute the query and receive the result
- Function - Which PHP-function were called

Id	**Date**	**Query time** [ms]	**Function**
⋮			
8492	2012-06-25 09:13	5.51915	insert_project
8493	2012-06-25 09:14	6.10995	Delete_admin_experience
8494	2012-06-25 09:14	7.66993	Insert_admin_new_expe
8495	2012-06-25 09:18	5.79405	delete_experience
8496	2012-06-25 09:25	7.49493	Insert_admin_new_tag
⋮			

Table 8: Example of rows from the Query time log.

6.1.2 Usage logging

The Usage logging gives a overview what actions are performed, which functionalities are used and who is using them. Table 9 is giving an example of how some rows in the Usage log can look like. The information gathered can be used later to analyse which users are most active and what parts of the web interface users tend to use most.

Each row in the usage log consists of four attributes:

- Id - Uniquely identifies each row in the log
- Date - *When* was the call made (date and time)
- Username - *Who*, which user, made the call
- Message - Describing in text what the call lead to

Id	**Date**	**Username**	**Message**
⋮			
1304	2012-06-15 15:35:44	anna-lena@qrtech.se	*created an account*
1305	2012-06-15 15:35:44	anna-lena@qrtech.se	*logged in*
1306	2012-06-15 15:38:11	anna-lena@qrtech.se	*updated the profile*
1307	2012-06-15 15:51:17	admin	*logged in*
1308	2012-06-15 15:51:58	admin	*created a tag*
⋮			

Table 9: Example of rows from the Usage log.

6.2 Average query time test

To be able to test and validate the database performance, two scripts were developed.

The first script enables the developer to create employees and all associated connections. A developer can determine how many employees to create and the number of experiences each will have. The number of tags and projects connected to each employee can also be specified.

The second script calls functions, presented in Appendix C, one-by-one and can be used to simulate traffic on the web server. This test utilizes both scripts.

The test is performed by evaluating how different numbers of employees and experiences affect the response time from the server depending on which function is used. Three different number of employees will be tested, 100, 200 and 400. The number of experiences associated with each employee is changed between 50, 100 and 200. All of the nine possible combinations will be tested and evaluated. The following parameters will be connected to each employee and have a constant value:

- 5 tags / experience

- 1 project / experience
- 40 projects and 66 tags available in the database

The Figures 32, 33 and 34 display the average time it took for the PHP-function to request and receive information from the database. Each function has been executed and logged at least 20 times to get a more precise average value. A time out was set to 30 seconds, meaning the request will be halted if the limit is exceeded. The time out only works as a upper-limit when testing; Calls to the database should be completed well below the time frame of 30 seconds to be acceptable. However, it indicates evidently which functions having performance issues as the experience database becomes larger.

For example, in the first test the number of experiences per employee was set to a constant value of 50. The first script added 100 employees, each with 50 experiences, and afterwards the second script was executed to generate function calls to the database. The query in Appendix C.3 was utilized to calculate the average values of each function.

This procedure was repeated nine times and the results are published in the Figures 32, 33 and 34.

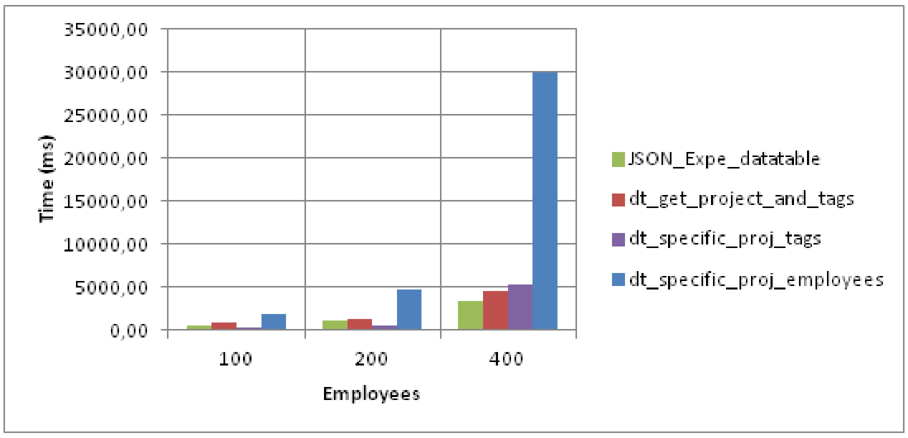


Figure 32: Performance test with 50 experiences/employee

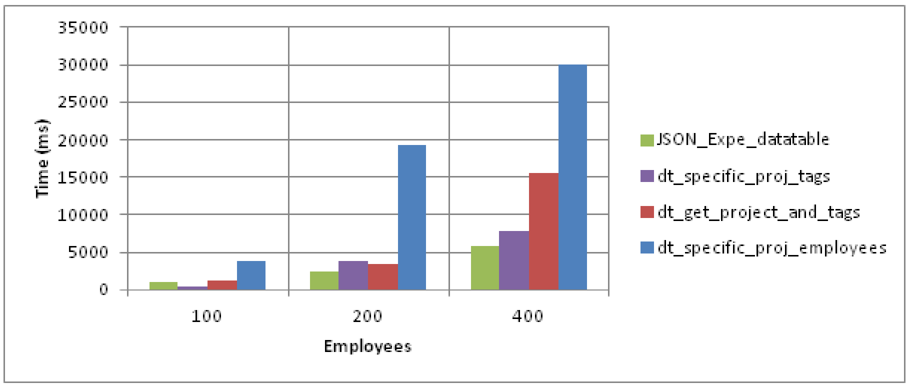


Figure 33: Performance test with 100 experiences/employee

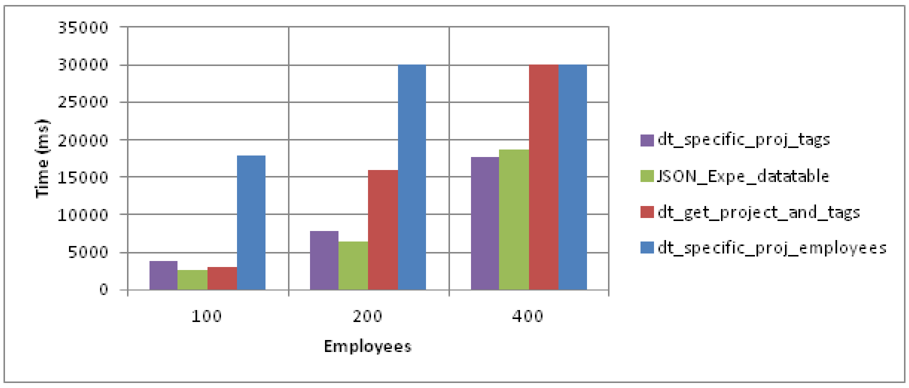


Figure 34: Performance test with 200 experiences/employee

6.2.1 Outcome

The four functions with the longest query times were:

- *dt_specific_proj_employees* - Requests information regarding all employees associated with a given project
- *dt_get_project_and_tags*– Lists all projects in a data table and all experiences, tags, usernames, emails, first and last names connected to each of them.
- *dt_specific_proj_tags* – Requests all tags connected to a given project including calculating how frequently each unique tag is connected to the given project
- *JSON_Expe_datatable* – Requests data regarding all experiences and its connected projects and tags for a certain user

To view the complete results for all functions from the tests, see Appendix C, where all query times are presented. By studying the outcome of the performance tests presented in Figures 32, 33 and 34 it could be determined that all four functions have performance issues when the number of employees and experiences were increased in the database.

6.2.2 Improvements

By analysing the functions optimizations was implemented and the results are visualized in Figure 35 and Table 10.

After the optimizations, the worst case scenario with 400 employees and 200 experiences/employee was executed again in order to evaluate the improvements. From the results it was found that all four of the former queries had been enormously improved. The results are as follows:

• *dt_get_project_and_tags*	30+ seconds → 16,40 seconds
• *dt_specific_proj_employees*	30+ seconds → 0,085 seconds
• *JSON_Expe_datatable*	18,66 seconds → 0,020 seconds
• *dt_specific_proj_tags*	17,72 seconds → 0,048 seconds

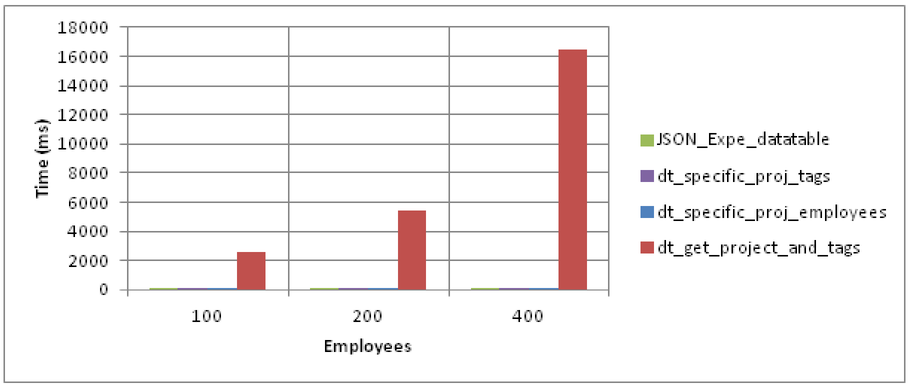


Figure 35: Performance test with 200 experiences/employee (After optimizations)

Function (200 exp/empl.)	100 empl.	200 empl.	400 empl.
dt_specific_proj_tags	14,53	25,35	47,8
JSON_Expe_datatable	19,83	20,16	19,85
dt_get_project_and_tags	2582,7	5409,14	16416,6
dt_specific_proj_employees	20,99	41,85	85,14

Table 10: The four queries with longest query time after optimization (Result in ms)

The improvements of the queries have made the web site faster. By using the test scripts the problem-filled areas could be detected and issues handled. The results from this test shows that it is worth considering developing even more sophisticated test environments to get improved analysis and make furthermore optimizations.

7 Results

The final Experience database consists of a web interface and a MySQL database.

The web interface provides three states in which a user can be in: *Not logged in*, *Logged in as user* and *Logged in as admin*. Users have access to different functionalities depending on the state, see Table 11. The table illustrates all the features that the website provides.

It exists functionalities in the database which aren't yet implemented in the web interface. An entire representation of the database structure can be seen in Appendix B.

Function	**Not logged in**	**Logged in as user**	**Logged in as admin**
Create user	x		
Search for employee/project	x	x	x
View employee information	x	x	x
View project information	x	x	x
Add/Edit projects		x	x
Edit own profile		x	x
Add/Edit/Remove own experiences		x	x
Add/Edit/Remove admin tag			x
Add/Edit/Remove admin experience			x
Add/Remove admin project			x
Activate/Deactivate employees			x

Table 11: Web site functionalities where "x" marks which feature is available

The web site supports the browsers Internet Explorer 9+, Firefox 13+, Safari 5.1+ and Google Chrome 19+. The absence of full support for Internet Explorer 8.0 is the main flaw but it can be avoided by using newer browser versions. The application is also adjusted to work on Smart phones.

At the end of this thesis test scripts were developed to evaluate the performance of the database queries made by the website. The results indicated that several of the queries had performance issues which had to be addressed. All problematic queries have been handled with only one exception. The performance of the problematic query was improved but is still not satisfactory. To improve it further structural changes needs to be performed and it wasn´t implemented due to the time limitation of this thesis.

8 Discussion

Developing a database structure which fulfilled all pre-defined requirements was proven to be harder than expected. One of the difficulties was to define what an experience is in order to enter and store it properly in the database. Our initial work with the *Experience tree* model, see section 4.2.1, was not satisfactory since the model made it difficult for users to categorize experiences. The transition to the *Experiences and tags* model, see section 4.2.2, was successful and it met all the requirements. By implementing tags the experiences could be defined dynamically, in comparison to the approach of categorizing experiences, and it also facilitated the data mining process. Additionally, tags decreased the redundancy of information in the database, making it more efficient.

Various strategies were applied during the development of the web site. Several iterations were performed until the final structure was established. Continuous iterations were necessary due to new ideas or approaches to implement functionalities. Minor mistakes were initially made regarding the development of HTML and JavaScript but since improvements were constantly implemented, most of the problems were solved.

Unfortunately, one problem that was not completely solved was full support for Internet Explorer 8.0. Installation of a newer version of Internet Explorer is currently not possible since QRTECH mainly utilize Windows XP as operating system which does not have support for a higher version than 8.0. Throughout this thesis, the main issues regarding web browsers were mainly perceived to be associated with Internet Explorer. However, no severe errors were detected in Internet Explorer 9.0, Chrome, Firefox or Safari as the web site was displayed.

The development of the web site was initially performed in smaller separate segments which all were standalone solutions presenting a part of the web interface. However, the need of a SVN (Subversion) service emerged later on as the different parts were to be combined into one solution.

The testing phase, which was performed late in this project, proved to be more interesting then we initially thought. The test utilized developed PHP-scripts to simulate users by making database calls through the web site. It supplied us with useful information indicating problematic areas in need of improvement which otherwise would not be located, see section 6. The web interfaces has not yet been put into operation at QRTECH and the *Usage logging*, see section 6.1.2, can't yet provide any useful results. However, over time the logging will indicate areas of the web page being used and which users are most active.

9 Conclusions

The database and web interface provides a satisfactory solution for performing the wanted functionality and is not detrimental to future implementations. Employees at QRTECH have been involved in the evaluation of the user interface to establish a qualitative product. The user experience can still be improved by making the layout more intuitive using pictures and additional help texts.

The query optimizations made at the end of the project, see section 6, proved to have a huge impact on the user experience, making the web site more responsive. Further optimizations can be performed but due to the project's time limitation these are to be considered as future improvements.

10 Future development

The objective of this section is to highlight features to improve.

- *Error codes* – As attempts are performed to insert inaccurate data into the database it returns an error code. All error codes aren't intuitively easy to understand, making it an area of improvement since the system should suit all employees at QRTECH regardless profession.
- *Security*
  - *MySQL injections* – Issues regarding MySQL injections can be investigated further in order to evaluate the safety of the experience database web site. A possible improvement is to implement the PHP function mysql_real_escape_string() before querying the database. The function replaces backslashes with other tokens for example to prevent deletions of an entire database. However, additional approaches need to be evaluated to find the most suitable solution.
  - *MySQL transactions* – The ability to perform an synchronized operation when executing several queries in a sequence can be improved by using the commands BEGIN, COMMIT and ROLLBACK.
  - *Password encryption* – User account passwords are stored as readable text in the database. It is preferable to add some sort of encryption.
- *Layout* – By investigating current research on layout of documents and web sites adjustments to fonts, text-sizes, colors etc. can be implemented to improve the user experience.
- *Add reference attribute* – During this thesis it was discussed to implement a new attribute to the Experience table called Reference. A reference can be the location of a document, a link to web site, simply plain text. The idea is to enable employees to share references to documentation regarding the specified experience.
- *Help texts* – The web site utilizes help texts to guide the user but improvements can always be performed on this area to increase the quality.
- *Tutorial* – Users should have the ability to view a presentation explaining the web site functionalities. The suggestions discussed during this thesis on how to present the information were to either do a video tutorial or a slide-show with pictures.
- *Unused database features* – The database is constructed for additional features beyond the implementations performed in this thesis.

  - *Product table* – The product table was at first thought of enabling users to add a connection between a project and a product. Several projects could for instance have contributed to finalize a specific product.
  - *Addtional employee email, phone number and address* – These three tables exist in the database with the objective to give users the ability to add additional emails, phone numbers and addresses. Currently users can merely have one email, phone number and address using the attributes on the table Employee.
  - *Wanted Experience* – The database include a table called Wanted Experience and it is similar to the table Experience. The idea is to allow employees to enter desired experiences to obtain in the future. A connection between projects and a wanted experience is not necessary since it's simply a request for future skills. However, the wanted experience can still have tags describing the desired experience. Enabling a wanted experience feature and making it visual to colleagues can be useful when assigning personnel to tasks or projects.
  - *CV tag* – Suggestions have been proposed by employees at QRTECH to prepare the database for future implementations of a CV functionality. A CV at QRTECH consists of seven areas of competence e.g. operating system, programming language etc. By enabling the database to connect a CV tag to an experience, the conditions to begin to implement the CV feature are met.

- *Diagrams* – The JavaScript library JSChart can be utilized to generate various types of diagrams and charts making it possible to display statistics in a graphical way.
- *Log data* – Improved data logging can be used for error checking and to locate problematic areas.
- *Testing environment* – An improved testing environment with a graphical user interface would make it easier to regularly perform tests on the web interface and database.
- *Search algorithm*
  - *Experience intensity* – Each experience has an attribute called Intensity. The objective is to use the attribute in the experience search algorithm to adjust the scores, e.g. *score = number of days* × *intensity.* This will make the scores from the rankings accurate since both the working pace and time is considered.
  - *Search logic* – Currently the operands OR, AND, NOT and MINUS can be used to configure search strings. However, NOT and MINUS can be considered as superfluous since employees use

  the web site to locate colleagues possessing experiences instead of the opposite.

- *Search optimization* – The search algorithm can be optimized to increase the search performance.

11 References

References

Brooks, D. R. (2007), *An introduction to HTML and JavaScript for Scientists and Engineers*, 1 edn, Springer Verlag, London.

Chapman, S. (2012), "What is javascript?", *Collected 2012-06-26* . `http://javascript.about.com/od/reference/p/javascript.htm`.

Garcia-Molina, Hector; Ullman, J. D. W. J. (2009), *Database Systems The Complete Book*, 2 edn, Pearson Education International.

Garret, J. J. (2005), "Ajax: A new approach to web applications", *Collected 2012-06-19* . `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`.

Goodman, Danny; Gustaff Rayl, C. (2010), *JavaScript Bible*, Wiley.

Harrington, J. L. (2009), *Relational Database Design*.

Holzner, S. (2008), *Ajax Bible*, John Wiley and Sons Inc.

Jardine, A. (2011), *Collected 2012-07-10* . `http://datatables.net`.

JQuery API:Load (2012), *Collected 2012-07-02* . `http://api.jquery.com/load/`.

MacInTyre, P. B. (2010), *PHP: The good parts*.

McPeak, Jeremy; Wilton, P. (2010), *Beginning JavaScript*.

Mounia, L. (2009), *XML Retrieval*.

Narayan, S. (2011), *Collected 2012-07-02* . `http://www.codeproject.com/Articles/157446/What-is-jQuery-and-How-to-Start-using-jQuery`.

Parker, R. D. W. T. (2011), *Collected 2012-07-10* . `http://jqueryui.com/demos`.

Schwartz, M. (2000), "Xhtml", *Computerworld* .

W3C (2012), *Collected 2012-05-06* . `http://www.w3.org/Consortium/facts#history`.

W3School (2012), *Collected 2012-07-02* . `http://www.w3schools.com/ajax/tryit.asp?filename=tryajax_first`.

Welling, Luke; Thompson, L. (2003), *PHP and MySQL Web Developement*, 2 edn, Sams Publishing.

Appendix A: Functional Dependencies

The functional dependencies which were derived during the development phase in the thesis are described in this section. View 2.1.5 for theory.

Tables written with bold text represent tables containing information about e.g. an employee. The other tables written without bold text represent tables containing relational information which connects tables e.g. employee_has_additional_email. These tables only consist of the keys from each table it connects. Underlined attributes are key attributes.

Office_phone(OFFI_ADD_phone_nr, OFFI_ADD_type)

office_has_additional_phone(OFFI_ADD_phone_nr, OFFI_name)
OFFI_name → Office.OFFI_name
OFFI_ADD_phone_nr → Office_phone.OFFI_ADD_phone_nr

Office_email(OFFI_ADD_email, OFFI_ADD_type)

office_has_additional_email(OFFI_name, OFFI_ADD_email)
OFFI_name → Office.OFFI_name
OFFI_ADD_email → Office_email.OFFI_ADD_email

Office(OFFI_name, OFFI_phone, OFFI_email, OFFI_street, OFFI_post_code, OFFI_country, OFFI_city, OFFI_street_nr)

works_in(OFFI_name, EMPL_username)
EMPL_username → Employee.EMPL_username
OFFI_name → Office.OFFI_name

Employee(EMPL_username, EMPL_first_name, EMPL_last_name, EMPL_date_of_employment, EMPL_phone_nr, EMPL_street, EMPL_post_code, EMPL_country, EMPL_city, EMPL_street_nr, EMPL_title, EMPL_age, EMPL_password, EMPL_picture, EMPL_description, EMPL_email, EMPL_active)

employee_has_additional_email(EMPL_ADD_email, EMPL_username)
EMPL_ADD_email → Employee_email.EMPL_ADD_email
EMPL_username → Employee.EMPL_username

Employee_email(EMPL_ADD_email, EMPL_ADD_type)

employee_has_additional_phone(EMPL_ADD_phone_nr, EMPL_username)
EMPL_ADD_phone_nr → Employee_phone.EMPL_ADD_phone_nr
EMPL_username → Employee.EMPL_username

Employee_phone(EMPL_ADD_phone_nr, EMPL_ADD_type)

employee_has_additional_address(EMPL_ADD_street,
EMPL_ADD_post_code, EMPL_ADD_street_nr, EMPL_username)
EMPL_ADD_street, EMPL_ADD_post_code, EMPL_ADD_street_nr →
Employee_address.(EMPL_ADD_street, EMPL_ADD_post_code,
EMPL_ADD_street_nr)
EMPL_username → Employee.EMPL_username

Employee_address(EMPL_ADD_street, EMPL_ADD_post_code,
EMPL_ADD_street_nr, EMPL_ADD_city, EMPL_ADD_type)

performed_at(PROJ_company, PROJ_name, OFFI_id)
PROJ_company, PROJ_name → Project.(PROJ_company, PROJ_name)
OFFI_id → Office.OFFI_id

Project(PROJ_company, PROJ_name, PROJ_start_date,
PROJ_end_date, PROJ_description)

learned_in(PROJ_company, PROJ_name, EXPE_id)
PROJ_company, PROJ_name → Project.(PROJ_company, PROJ_name)
EXPE_id → Experience.EXPE_id

Experience(EXPE_id, EXPE_name, EXPE_start_date, EXPE_end_date,
EXPE_degree, EMPL_username)
EMPL_username → Employee.EMPL_username

Wanted Experience(WEXPE_id, WEXPE_name, WEXPE_log_time,
EMPL_username)
EMPL_username → Employee.EMPL_username

Product(PROD_name)

developed_in(PROJ_company, PROJ_name, PROD_name)
PROJ_company, PROJ_name → Project.(PROJ_company, PROJ_name)
PROD_name → Product(PROD_name)

Tags(TAGS_name)

product_has_tags(PROD_name, TAGS_name)
TAGS_name → Tags.TAGS_name
PROD_name → Product.PROD_name

project_has_tags(PROJ_company, PROJ_name, TAGS_name)
TAGS_name → Tags.TAGS_name
PROJ_company, PROJ_name → Product.(PROJ_company, PROJ_name)

wex_has_tags(WEXPE_id, TAGS_name)
TAGS_name → Tags.TAGS_name
WEXPE_id → Wanted Experience.WEXPE_id

ex_has_tags(EXPE_id, TAGS_name)
TAGS_name → Tags.TAGS_name
EXPE_id → Experience.EXPE_id

Has_CV_tag(EXPE_id, CV_tag_name)
EXPE_id → Experience.EXPE_id
CV_tag_name → CV_tag.CV_tag_name

CV_tag(CV_tag_name)

Admin_experience(EXPE_name)

Admin_tag(TAGS_name)

Admin_project(PROJ_company, PROJ_name, PROJ_start_date, PROJ_end_date, PROJ_description)

Log(LOG_id, LOG_date, LOG_querytime, LOG_function)

Log_msg(LOGM_id, LOGM_user, LOGM_date, LOGM_function)

Appendix B: ER-diagram overview

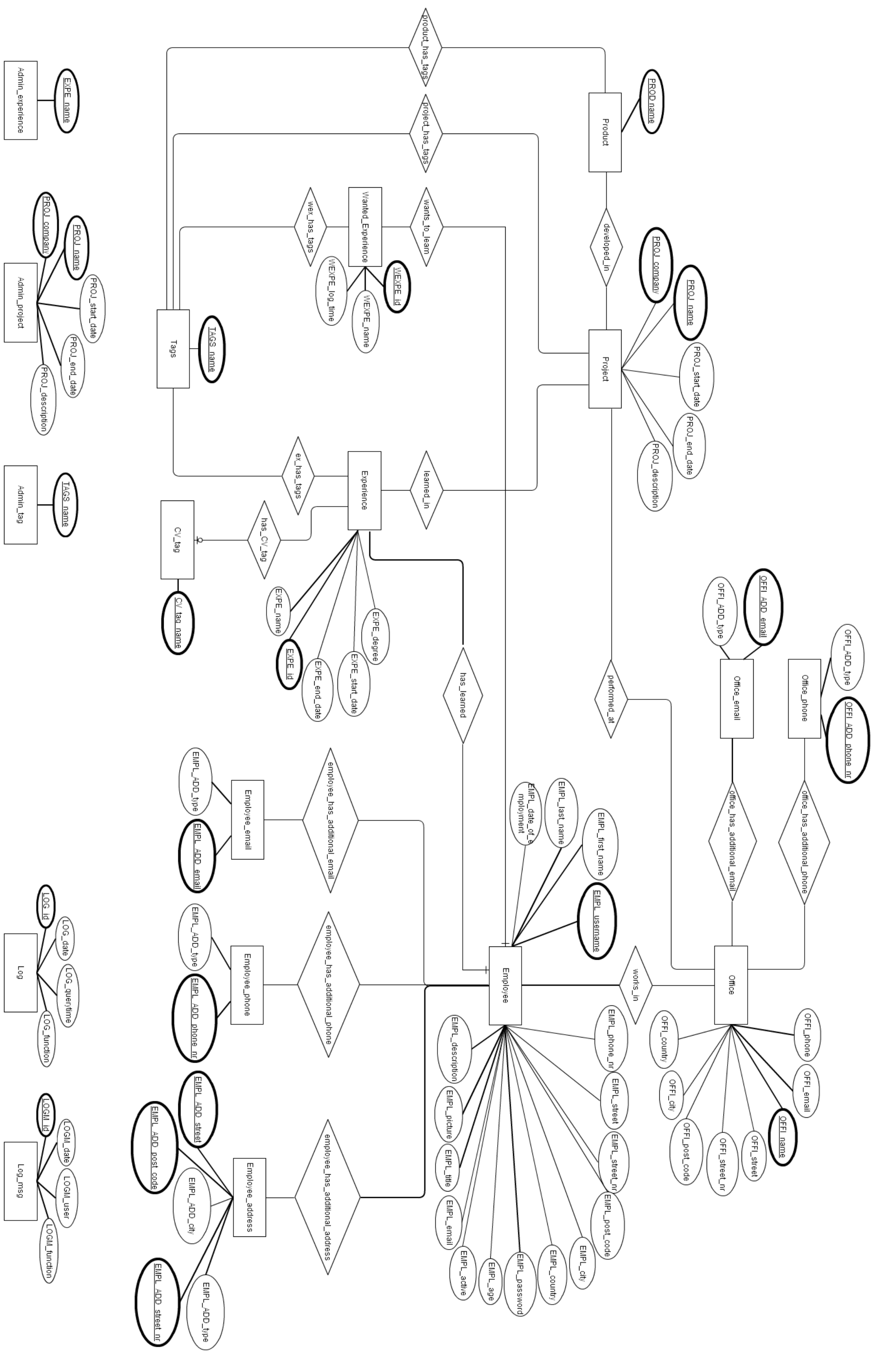


Figure 36: The entire ER-diagram

Appendix C: Performance evaluation results

This section demonstrates the results from the performance test performed regarding the queries to the database. All tests have been performed using a local web server.

C.1 Before improvements

The test in Table 12 has a constant value of 50 experiences and varies the number of employees between 100, 200 and 400.

PHP function	**100 empl.**	**200 empl.**	**400 empl.**
Admin_active_activateORdeactivate	80.46	60.76	73.15
DB_update_user_info	167.61	103.71	130.05
Delete_admin_experience	126.88	73.37	71.45
Delete_admin_tag	354.24	120.08	126.57
Insert_admin_new_expe	90.61	92.21	66.01
Insert_admin_new_tag	167.19	124.68	128.78
JSON_Admin_active_employees	2.86	3.90	7.98
JSON_Admin_add_expe	14.01	23.73	62.61
JSON_Admin_add_project	1.92	1.85	1.75
JSON_Admin_add_tag	2.24	2.01	1.73
JSON_Expe_datatable	559.01	1024.71	**3418.85**
JSON_LOAD_user_info_userinfo	0.87	0.74	0.68
check_login	100.83	53.35	71.14
create_user_check	77.79	71.62	66.15
delete_experience	78.31	69.36	65.64
dialog_updateexp_fill	203.10	278.85	639.75
dt_all_projects	2.50	2.22	2.13
dt_get_project_and_tags	842.34	1401.38	**4661.84**
dt_get_tag_suggestions	6.87	9.90	25.90
dt_specific_proj_employees	1947.65	4785.09	**30000+**
dt_specific_proj_tags	296.78	462.76	**5412.61**
get_exp_suggestions_json	14.99	23.82	63.31
get_specific_project_info	0.99	0.61	0.55
get_tag_and_exp_suggestions_json	16.18	26.21	67.07
get_tag_suggestions_json	2.10	2.24	2.01
insert_experience	91.13	62.75	81.89
insert_project	91.29	133.59	67.71
update_experience_name	94.05	77.89	239.72
update_project_info	89.61	98.25	250.26
update_tag_name	75.45	70.90	60.46

Table 12: Entire performance test results with 50 experiences

The test in Table 13 has a constant value of 100 experiences and varies the number of employees between 100, 200 and 400.

PHP function	**100 empl.**	**200 empl.**	**400 empl.**
Admin_active_activateORdeactivate	66.20	68.00	61.83
DB_update_user_info	124.96	135.64	123.51
Delete_admin_experience	61.04	71.71	67.47
Delete_admin_tag	146.81	131.98	126.67
Insert_admin_new_expe	62.15	73.71	65.00
Insert_admin_new_tag	112.88	147.23	123.11
JSON_Admin_active_employees	2.56	4.49	8.11
JSON_Admin_add_expe	26.21	49.17	98.22
JSON_Admin_add_project	1.73	1.77	1.75
JSON_Admin_add_tag	1.74	1.75	1.74
JSON_Expe_datatable	1042.90	2462.24	**5846.52**
JSON_LOAD_user_info_userinfo	0.68	0.66	0.67
check_login	82.16	75.93	73.45
create_user_check	55.74	66.77	66.86
delete_experience	54.59	65.86	61.75
dialog_updateexp_fill	251.04	461.54	961.58
dt_all_projects	2.07	2.08	2.08
dt_get_project_and_tags	1327.13	3451.45	**15566.73**
dt_get_tag_suggestions	9.99	18.82	35.98
dt_specific_proj_employees	3895.92	19307.77	**30000+**
dt_specific_proj_tags	392.72	3765.76	**7698.33**
get_exp_suggestions_json	26.69	50.34	101.61
get_specific_project_info	0.55	0.58	0.54
get_tag_and_exp_suggestions_json	25.56	48.66	91.97
get_tag_suggestions_json	2.08	2.00	2.04
insert_experience	66.30	81.92	72.38
insert_project	52.34	70.29	59.53
update_experience_name	94.00	115.99	158.05
update_project_info	93.99	251.59	269.88
update_tag_name	53.75	68.13	68.43

Table 13: Entire performance test results with 100 experiences

The test in Table 14 has a constant value of 200 experiences and varies the number of employees between 100, 200 and 400.

PHP function	**100 empl.**	**200 empl.**	**400 empl.**
Admin_active_activateORdeactivate	70.81	66.12	74.58
DB_update_user_info	139.60	170.10	145.37
Delete_admin_experience	68.02	69.70	160.56
Delete_admin_tag	136.43	148.22	163.15
Insert_admin_new_expe	153.92	66.29	64.62
Insert_admin_new_tag	148.78	163.15	135.79
JSON_Admin_active_employees	2.58	4.74	8.05
JSON_Admin_add_expe	51.15	96.96	203.35
JSON_Admin_add_project	1.80	1.77	1.81
JSON_Admin_add_tag	1.77	1.75	1.80
JSON_Expe_datatable	2720.09	6418.80	**18663.32**
JSON_LOAD_user_info_userinfo	0.67	0.69	0.69
check_login	71.88	77.69	96.79
create_user_check	74.19	66.55	63.54
delete_experience	77.04	67.04	71.62
dialog_updateexp_fill	473.16	1063.01	3823.87
dt_all_projects	2.13	2.10	2.13
dt_get_project_and_tags	3124.25	16016.97	**30000+**
dt_get_tag_suggestions	18.76	35.43	72.56
dt_specific_proj_employees	17889.71	30000+	**30000+**
dt_specific_proj_tags	3868.36	7877.18	**17726.46**
get_exp_suggestions_json	50.93	98.14	204.01
get_specific_project_info	0.56	0.56	0.56
get_tag_and_exp_suggestions_json	47.31	93.04	192.20
get_tag_suggestions_json	2.08	2.04	2.10
insert_experience	77.47	83.54	83.67
insert_project	70.14	67.11	67.37
update_experience_name	117.23	164.43	263.18
update_project_info	151.63	266.96	532.65
update_tag_name	72.27	65.42	71.47

Table 14: Entire performance test results with 200 experiences

C.2 After improvements

The test in Table 15 is performed after changes have been made to the queries. The results are based on the worst case scenario, 400 employees each with 200 experiences.

PHP function	**100 empl.**	**200 empl.**	**400 empl.**
Admin_active_activateORdeactivate	73.21	67.24	84.55
DB_update_user_info	157.45	221.06	142.36
Delete_admin_experience	75.77	77.90	73.44
Delete_admin_tag	165.50	155.11	129.72
Insert_admin_new_expe	85.09	72.51	70.25
Insert_admin_new_tag	144.36	128.40	141.72
JSON_Admin_active_employees	2.58	4.53	8.00
JSON_Admin_add_expe	46.26	89.27	178.22
JSON_Admin_add_project	1.95	1.79	1.72
JSON_Admin_add_tag	2.50	1.80	1.72
JSON_Expe_datatable	19.83	20.16	**19.85**
JSON_LOAD_user_info_userinfo	0.67	0.67	0.68
check_login	103.61	65.32	74.61
create_user_check	76.61	65.11	69.10
delete_experience	81.27	71.19	71.02
dialog_updateexp_fill	1.40	1.43	1.39
dt_all_projects	2.09	2.09	2.09
dt_get_project_and_tags	2582.69	5409.14	**16416.57**
dt_get_tag_suggestions	19.03	36.19	71.78
dt_specific_proj_employees	20.99	41.85	**85.14**
dt_specific_proj_tags	14.53	25.35	**47.80**
get_exp_suggestions_json	47.29	91.12	177.75
get_specific_project_info	0.56	0.55	0.55
get_tag_and_exp_suggestions_json	49.18	95.01	185.78
get_tag_suggestions_json	2.05	2.37	2.13
insert_experience	86.51	94.58	86.15
insert_project	80.17	72.87	70.93
update_experience_name	133.91	179.69	274.67
update_project_info	176.13	278.70	508.21
update_tag_name	77.82	65.16	68.55

Table 15: Entire results after optimizations with 200 experiences

C.3 Query - calculating the average query time

To compute the query times the following query was executed:

```
SELECT ROUND(AVG(LOG_querytime),2) AS Average_time,
                 LOG_func, COUNT(*) AS Nr_of_calls
FROM Log
GROUP BY LOG_func
ORDER BY Average_time DESC;
```

Figure 37: MySQL query used for average query time calculation

Appendix D: Gantt chart

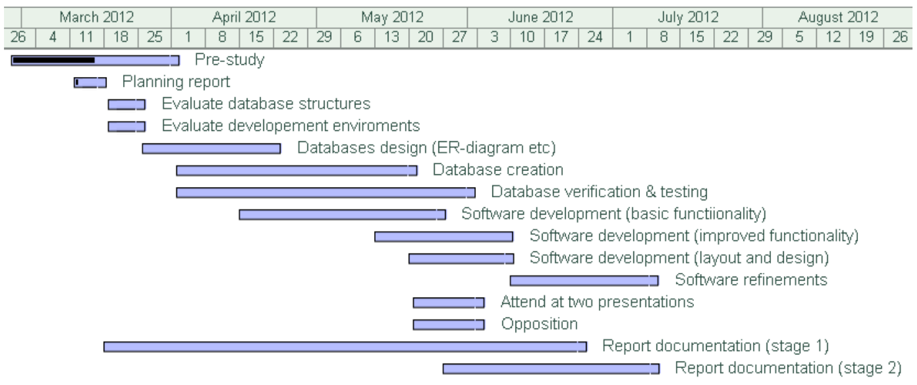


Figure 38: Gantt chart illustrating the project schedule

- Report documentation (stage 1)
  - Create ground structure
    - Headings
    - Layout
  - Continuously write under corresponding header
- Report documentation (stage 2)
  - Focus on refining the report
    - Uniform language and style
    - Look through references and figures