Abstract

This document describes the background, design, and what results that were obtained when developing the software called PHP Order Manager. PHP Order Manager is a software for helping a small company manage their orders. This includes entering the orders when they arrive and handle changes during the production. The system uses MySQL for the database back end and PHP for interfacing the database. PHP Order Manager has great support for modules and has a nice user administration tool.

1 Introduction

The project started with a request from DesginaDoors AB (DD). DD is a small company situated in Örnsköldsvik with fifteen to twenty employees. They manufacture custom made wooden doors to both private persons and companies. Today DD use a paper-based order management system which is very inconvenient for the company because when for example a change is made, new papers has to be printed out and be delivered out into the factory. All the papers are made in Excel but they also have a small Access database that they don’t use so much because they have to have Access on every computer. DD wanted a better solution for their order management system, a database with a web based interface that could contain all the information that today is in Excel and Access. The main feature of the wanted system is that you should be able to register new orders, change existing ones and delete orders that have been cancelled. To be useful, the system should also support some kind of user and group management so that the personnel only can access information that is relevant to their line of work. The system will also give the company benfits because it will allow for statistical analysis of the order intake and the production time. With such a system, you could for example check the time every order spend in each section of the factory and thereby find slow phases in the production.
2 Approach

With this information as a base an Extended Entity-Relationship (EER) diagram were created and later mapped into a database schema. The database schema comprises all orders and the status of an order. The schema includes temporal features to record changes of the database. Further there are schemas that keeps the users and groups and is used for authentication and authorization.

The software that were chosen for this project were MySQL and PHP. The choice of MySQL was not really a choice, it was a requirement from DesignaDoors. Both MySQL and PHP are however very nice development platforms which are easily integrated to a web interface.

The PHP part of the project (what the user actually can see) is designed with a modular support in mind. This means that extensions made to the script will popup right into the menu of the web interface if it is developed using the right module architecture. An extension to the script does not even have to be related to the original software. This can for example be a "e-mail plug-in" that uses the companies server to login the personells to their personal e-mail accounts. A module can also be a wrapper for another script with the only purpose to integrate it seamlessly with the web interface. The layout of the interface have been chosen so it fits the actual look of DD homepage, partly because the users should not feel that everything is totally new.

A database involving changes among several users within a corporate requires that all changes made by the employees are recorded in the database. For this purpose temporal features are included in the database. This involves so called logical deletes and storing of history versions for each updated tuple. The technique used is so called tuple versioning and in this specific case the interest is transaction time, which records when the changes to the database were made according to the system clock.

Every user is forced to login before any function can be used. The authentication is done using usernames and MD5 encrypted passwords stored in the database. Authorization of which functions a user may use are based on which group the user belongs to. This means that users are placed in groups according to a permission level. For example the group Administrators have significantly higher permission level than users in group workers. Every module defines a set of functions that the module itself contains. Before any function is shown in the interface menu the database is queried to check that the user really is authorized to do this. Totally four tables are involved for user and permission management.

Three of the tables in the database are used to manage all orders (see figure 1). To every order there exists a table called "Order_info" and a table called "Station" that both refers to a specific order number. An order can consist of several articles and it will be represented as several tuples in the table "Order_info". During the production of a door the piece of wood passes several stations in the factory. To make it easy to follow a certain order during production these stations are represented in the table "Station". When an order arrives at a station one of the employees just ticks off the order and it will be visible at the web page.

The point when orders are to be changed is critical. This because several users could be doing changes to the same order without knowing about each other. To solve this problem and thereby prevent corrupt orders, the transaction start time
(TST) is used. When a user chooses an order for changing the TST for the active order tuple is stored and just before commit of a change this TST timestamp will be compared to the TST of the active tuple at this moment. If this two values differ in some way the change will not take place and the user will be informed about it.

One further benefit with having orders in a temporal database instead of on papers is the possibility to easy generate statistics. Every change of an order can be tracked and displayed which could be very useful if misunderstandings occur. Every station does not spend the same amount of time working on an order so therefore an overview over the time distribution can be presented for requested orders.

If there are some special information about an order or some information that a specific user or set of users should be aware of, one can use the simple message system that are built in.

### 3 Results

The system have this far only been tested by our selves but the results have been positive. During the beginning of the summer the application will be introduced for DD and tested by them. In figure 2 a screenshoot is shown from one view.
The current system supports some types of queries that are interesting for the company, a few of these are

- Find history versions of an order
- Show the time every order have spent in each production phase

4 Discussion

The foundation of this work is the database. This made it naturally that we started to design the database (with the requirements in mind) in a early stage in the project. In the beginning an EER-schema with ISA operations was in mind but after some research about MySQL this idea was rejected. MySQL is not like PostgreSQL. In the version of MySQL that the database is implemented in (which is the latest stable version) inheritance cannot be handled for ISA operations. Assertions of different kinds were also not supported which made the work a little bit harder, these things had to be moved out to PHP instead.

During the project the contact with DD has been almost non-existent, so for that reason it is hard to know if the development of the system is on the right track. The reason to why the contact was so scarce was that DD didn’t really know what they wanted and needed and didn’t have the time to discuss it. This has made the work harder and after the system is shown for them it probably needs some modification.
There are quite a few improvements that could be done, even without getting feedback from DD. The statistics for example could be improved by supporting more types of queries and by showing the results in graphs instead of in text as it is now. Another thing that could be improved is the message system. It would be nice to have a more dynamic message system where you get some kind of notification when a new message has been read by the receiver.

None of the improvements mentioned above are difficult to implement. The limited amount of time and the focus on the PHP application design (including module support and user administration) and the design of the database led to the decision to only implement a subset of the queries. The modular support and the currently implemented queries shows how easily this can be extended.