CODEN:LUTEDX/(TEIE-3024)/1-40/(2013)

Industrial Electrical Engineering and Automation

Centralisering av Helsingborgs Bryggeris informationshantering

Joachim Jonasson
Daniel Gustafsson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Centralisering av Helsingborgs Bryggeris informationshantering

LUNDS UNIVERSITET
Lunds Tekniska Högskola
LTH Ingenjörshögskolan vid Campus Helsingborg

Institutionen för Mätteknik och Industriell Elektroteknik

Avdelningen för Industriell Elektroteknik och Automation

Examensarbete av:
Joachim Jonasson
Daniel Gustafsson

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2013

Sammanfattning

Projektet har pågått på heltid under vårterminen 2013 och har utförts tillsammans med Helsingborgs Bryggeri. Bristfällig informationshantering har under en tid varit ett problem för Helsingborgs Bryggeris interna och externa kommunikation. Av den anledningen har projektet skapats, där målet är att utveckla en databas som centraliserar informationshanteringen för Helsingborgs Bryggeri. Till databasen har det skapats ett grafiskt användargränssnitt, i form av en hemsida, så att Helsingborgs Bryggeri kan utnyttja databasen. I användargränssnittet har det även skapats funktioner som behandlar informationshanteringen både internt och externt som exempelvis kommunikation med företaget Fortnoxs redovisningssystem. Databasen är utvecklad i SQL-script och gränssnittet är skapat med hjälp av CSS, HTML, XML och PHP-script. Databasen skyddas med autentisering och gränssnittet är programmerat så att det skyddar databasen mot olika typer av attacker.

Nyckelord: Databas, Gränssnitt, PHP, SQL, HTML

Abstract

The project has been in full-time during the spring semester 2013, and has been performed together with Helsingborgs Bryggeri. Insufficient information management has for some time been a problem for Helsingborgs Bryggeri´s internal and external communications. For this reason, the project has been created, where the goal is to develop a database that centralizes information management for Helsingborgs Bryggeri. For the database there have been created a graphical user interface in the form of a website, so that Helsingborgs Bryggeri can utilize the database. In the user interface there has been created functions that deal with the management of information both internally and externally, such as communication with the company Fortnoxs accounting system. The database is developed in SQL-scripting and the interface is created using CSS, HTML, XML and PHP-script. The database is protected by authentication and the interface is programmed so that it protects the database against different types of attacks.

Keywords: Database, Interface, PHP, SQL, HTML

Förord

Den här rapporten är ett resultat på ett examensarbete som utförts tillsammans med Helsingborgs Bryggeri. Examensarbetet har utförts på helfart under vårterminen 2013 på programmet Elektroteknik med automation på Lunds Tekniska högskola.
Examensarbetet grundar sig i databashantering och programmering av ett grafiskt användargränssnitt för att öppna kommunikationsvägar för informationshantering.

Vi vill tacka Helsingborgs Bryggeri för deras engagemang i vårt examensarbete och även tacka för ett gott samarbete.

Vi tackar även vår handledare Mats Lilja och examinator Per Karlsson för den goda vägledningen genom examensarbetet.

Joachim Jonasson och Daniel Gustafsson 2013

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Helsingborgs Bryggeri är ett mikrobryggeri som är i en mycket hög tillväxtfas. Bryggeriet startade 2011 och har under utvecklingsfasen använt sig av olika tjänster för informationshantering. Idag är hanteringen av information alldeles för spridd och mycket dubbelarbete görs på grund av detta.

Det finns en önskan hos Helsingborgs Bryggeri att förenkla bryggningsprocessen med hjälp av att delvis automatisera den samtidigt som man behåller det hantverksmässiga arbetet för att brygga. För att möjliggöra förenklingar i arbetet hos Helsingborgs Bryggeri behövs en centraliserande databas vilken kan ersätta större delen av föregående datorprogram som används för att lagra information.

1.2 Syfte och Mål

För att påbörja en utveckling av processen behövs en strukturering och centralisering av informationshanteringen på bryggeriet och även med informationskanalerna kring bryggeriet. Syftet med examensarbetet blir att lägga en grund för att bryggeriet ska kunna utveckla processen vidare. För att strukturera informationshanteringen behöver en databas skapas som kan hantera och centralisera informationsflödet. Databasen ska kommunicera med delar i processen och även utbyta information med andra informationstjänster, t. ex Fortnox, vilket innebär att kommunikationsvägar måste undersökas och utvecklas. För att enkelt tillgå databasen och tjänsterna den erbjuder behövs ett grafiskt användargränssnitt som enkelt kan användas av databasens alla användare.
Målet med projektet är att Helsingborgs Bryggeri skall erhålla ett system som de kan använda för att underlätta sitt dagliga arbete genom att få en smartare och förenklad informationshantering. Det ska även finnas möjligheter för vidare utveckling av projektet.

1.3 Problemformulering

Strukturera en centralisering för Helsingborgs Bryggeris informationshantering till grund för en vidareutveckling av bryggeriprocessen.

- Skapa en databas som kan samla all information.
- Skapa ett gränssnitt som kan behandla informationen
- Utföra kommunikation med redovisningssystemet Fortnox
- Undersöka kommunikationsmöjligheter med PLC

1.4 Avgränsningar

Den högst prioriterade delen i examensarbetet är databasen. Databasen innebär mycket förarbete i form av undersökning om vad den ska innehålla och hur den ska användas. Databasen ska även struktureras och programmeras. Den andra prioriterade delen av examensarbetet blir att skapa ett användarvänligt grafisk användargränssnitt till databasen. Koppling med order och faktureringssystemet Fortnox för kommunikation med databasen kommer att avsluta projektet men är den minst prioriterade delen i examensarbetet. Koppling till PLC implementeras inte i systemet men det utförs förberedelser och det anges endast förslags på hur kopplingen mellan PLC och databas kan utföras.

1.5 Disposition

Kapitel 1: Inledning
Här presenteras en bakgrund till hur projektet uppkommit och det beskrivs även de avgränsningar, mål och syfte som finns i examensarbetet.
Kapitel 2: Nulägesbeskrivning
Här anges en nulägesbeskrivning som förklarar problematiken som finns hos Helsingborgs Bryggerier innan projektets start.
Kapitel 3: Teknisk bakgrund
Här anges den tekniska bakgrunden som förklarar vilka tekniska verktyg som använts för att skapa interface och databas.
Kapitel 4: Metod
Här beskrivs tankesättet om hur databasen och gränssnittet är skapat.
Kapitel 5: Databas
Här beskrivs alla entiteter och relationer som databasen innehåller.
Kapitel 6: Användarvänlig anpassning
Beskriver hur det grafiska gränssnittet anpassats efter Helsingborgs Bryggeris användare.
Kapitel 7: SQL-injection och Cross site scripting
Här beskrivs hur databasen skyddas mot eventuella framtida attacker.
Kapitel 8: Gränssnitt
Här beskrivs hur gränssnittet har utformats och vilka funktioner som finns för Helsingborgs Bryggeris användare.
Kapitel 9: Utvecklingsmöjligheter
Här beskrivs det möjligheter för utveckling av projektet som har framkommit under projektets gång.
Kapitel 10: Slutsatser
Här avslutas rapporten med olika kommentarer om projektet.

2 Nulägesbeskrivning

Helsingborgs Bryggeri har inte en egen IT-avdelning och heller ingen resurs för att ha det i nuläget. Bryggeriet är en liten "processindustri" som idag körs mestadels manuellt.

De datamedier som Helsingborgs Bryggeri använder sig av i nuläget är order och fakturering via det molnbaserade systemet Fortnox. Ifrån Fortnox skrivs följesedlar, plockordrar ut och systemet används bland annat även som bokföringsprogram.

I nuläget finns inget system för att veta vad och hur mycket som finns i deras befintliga materiallager eller råvarulager. Fortnox uppfyller heller inte Helsingborgs Bryggeris krav på att hantera information om olika lager, då Fortnox system verkar vara mer anpassat för artiklar deras kunder säljer.

Helsingborgs Bryggeri arbetar i huvudsak med att manuellt styra bryggeriprocessen. Alla produktionsrelaterade siffror redovisas på olika papper och blad under tillverkningsprocessen. Olika mallar i programmet Excel finns utformade för verksamhetsnivåer och produktion men inga kopplingar finns som skickar över information mellan dem.

Det finns idag ingen samlad plats för att kommunicera viktig information igenom och man får istället använda sig av telefon och e-post. Exempelvis om Helsingborgs Bryggeris lagerpersonal ska ta emot en beställning av ett material så finns det inte lagrat i något internt system vad som är beställt eller hur mycket.

Största problemet Helsingborgs Bryggeri har i nuläget är att större delen av informationen dubbellagras. Detta innebär praktiskt för personalen att de manuellt måste skriva samma information på flertalet ställen.

Helsingborgs Bryggeri har införskaffat en PLC med visionen av att den skall kunna lagra mätvärden från processen och tillhandahålla dem till ett framtida databassystem.

3 Teknisk bakgrund

En av projektets tekniska huvuduppgifter är att skapa ett grafiskt användargränssnitt i form av en hemsida som kan arbeta tillsammans med den databas som skapas för att hantera och lagra information. Det grafiska användargränssnittet är baserat på PHP-script, HTML-script och databasen på SQL-script.

3.1 Databashanterare

För att lägga en bra grund till den här typen av projekt så måste en välfungerande och stabil databashanterare väljas. Eftersom det grafiska användargränssnittet är skrivet i PHP-script så krävs en databashanterare i PHP-utförande. I projektet används phpMyAdmin [1] som är ett gratis verktyg för att hantera SQL-databaser.

PhpMyAdmin använder sig av ett webb-browser baserat användargränssnitt för att kommunicera med databas och är speciellt utvecklat för att arbeta mot PHP.

3.2 Gränssnitt till databas

Kommunikationen mellan det grafiska användargränssnittet och databas utförs med hjälp av script-språket PHP [2]. PHP är ett språk för både generell programmering och webbutveckling och är det språk i projektets gränssnitt som arbetar mot databasen och möjliggör dynamiska funktioner på hemsidan.

3.3 Grafiskt användargränssnitt

Tillsammans med PHP används HTML [3] för att bygga upp hemsidans grafiska användargränssnitt. HTML är ett script-språk som hanterar utseende på hemsidor och en mer webb-browser baserad kommunikation än PHP, t. ex används den för att skapa länkar. Det grafiska användargränssnittet är programmerat i editorn Notepad++ [4] som är en avancerad version av Notepad. Notepad++ ger programmeraren en del hjälpmedel som förstärker tydligheten i den skrivna kodens struktur.

3.4 Fortnox API

Helsingborgs Bryggeri använder sig av företaget Fortnox [5] för att administrera en del av sin informationshantering t. ex bokföring och fakturering. För att hämta information från Fortnox så används Fortnox egna API (Application Programming Interface) baserat på script-språket XML. XML [6] är ett språk som märker information i dokument med strukturerad information, vilket möjliggör för enkel inhämtning av den med hjälp av till exempel PHP. I princip kan du använda alla funktioner i Fortnox genom olika anrop från ditt gränssnitt, både för att skicka data till deras databas och hämta data från deras databas. Exempelvis kan användaren med hjälp av Fortnox fakturera sina kunder.

```
12
13  host = "https://api.fortnox.se/ext/get_contact.php?token=2594f2b9ef16f38e61c60b6b572e281d&db=203050&id=all"
14
```

Figur 3.1 Anrop på Fortnox API.

Bilden ovan visar ett anrop från gränssnittet till Fortnox API med adressen https://api.fortnox.se/ext. Anropet gör så att Fortnox API hämtar data ifrån Fortnoxs databas. I detta fall hämtas kunduppgifter med hjälp av kommandot get_contact. För att urskilja vilka kunduppgifter som hämtas ur databasen så måste ett id anges och i detta fall ska gränssnittet hämta alla kunduppgifter som finns i Fortnoxs databas. Detta visas i bilden ovan där id=all och för att hämta specifik kund anges den kundens id.

För att kommunicera med Helsingborgs bryggeris databas hos Fortnox så måste gränssnittet ange vilken databas, i anropet, som API ska hämta information ifrån. Verifiering av vem som kallar på Fortnox API görs med hjälp av ett token. Som bilden ovan visar är ett token i Fortnox API ett väldigt långt lösenord. Token fungerar som en biljett för att ha möjlighet att skicka data säkert mellan två olika terminaler.

3.4.1 Client URL

När anrop till Fortnox API görs returneras data i form av XML-script. För att hämta data ur XML-scriptet använder gränssnittet Client URL (cURL), vilket är en metod för att kommunicera med servrar med olika typer av protokoll som finns, exempelvis i detta fall https. Client URL tar emot och skickar data i form av olika format via PHP-script.

```
$ch=curl_init();
$host = "https://api.fortnox.se/ext/get_contact.php?token=2594f2b9ef16f38e61c60b6b572e281d&db=203050&id=all";
 curl_setopt ( $ch, CURLOPT_URL,         $host );
 curl_setopt ( $ch, CURLOPT_TIMEOUT,       120);
 curl_setopt ( $ch, CURLOPT_RETURNTRANSFER,   1 );
 curl_setopt ( $ch, CURLOPT_SSL_VERIFYPEER,   0 );
 $data=curl_exec($ch);
 $result = simplexml_load_string($data);
```

Figur 3.2 cURL konfiguration i PHP-script.

Bilden ovan visar ett exempel ur PHP-script där information som hämtas från Fortnox API behandlas. Först görs en initiering av en session för dataöverföring med cURL. Curl_setopt används sedan för att ställa in olika inställningar för den här sessionen av dataöverföring. Först anges rätt URL vilken ska användas i kommunikationen och en typ av "time to live" anges i form av Curlopt_Timeout. Till sist anges att hämtad data ska samlas i en sträng med hjälp av kommandot curlopt_returntransfer. Curl_exec exekverar sessionen och gränssnittet kan, som visas på sista raden, lagras som ett XML-dokument i en PHP-variabel.

3.5 Wampserver

Utvecklingen av projektets grafiska användargränssnitt har gjorts med hjälp av en applikation som heter Wampserver. Wampserver [7] är ett program för webbutvecklare som samlar applikationer som apacheserver, MySql och PHP i ett enda skal för enkel användning.

Wampserver valdes framför den liknande programmet Xampp [8]. Programmen testades och bedömdes som lika välfungerande och stabila. Valet föll slutligen på Wampserver som visade sig ha ett mycket mer lättanvänt användargränssnitt än Xampp. Wampserver kommer inte användas för att lagra databasen och lägga upp hemsidan på nätet eftersom Helsingborgs Bryggeri har en önskan om att inte ha en egen server som måste underhållas

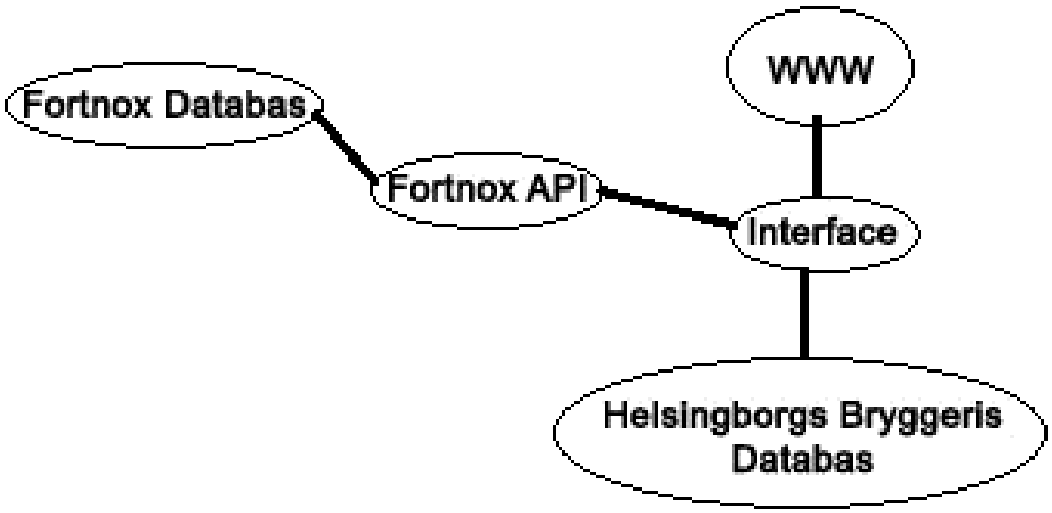
på bryggeriet. Hemsida och databas läggs istället hos företaget FS-data som får lagra hemsidan och databasen.

4 Metod

För att arbeta fram en välfungerande relationsdatabas så har Helsingborgs Bryggeris situation och informationshantering studerats grundligt. Efter det har en utförlig text skrivits ihop om Helsingborgs Bryggeri för att identifiera tabellnamn, variabler och primära nycklar. Detta gjordes för att skapa en stark grundstruktur i databasen som sedan kunde byggas vidare när projektet löpte. När tabellnamn, variabler och primärnycklar identifierats så ritades ett E/R-diagram för att skapa korrekta relationer mellan tabellerna. Målet har hela tiden varit att skapa databasen efter normalformen Boyce-Codd för att det inte ska uppstå några andra funktionella beroenden än de som önskas.

I projektet har det upptäckts att det inte alltid går att följa den optimala databasstrukturen. Vid en del tillfällen, som när databasen exempelvis lagrar historik, behövs dubbellagringar göras i databasen, med anledning av att man i det grafiska användargränssnittet kan ta bort en produkt ur databasen, men man vill inte ta bort historiken om produkten. Detta problem har lösts genom att dela upp databasen i egna små delar som alla är i skapade efter Boyce-Codd.

Gränssnittet och hemsidan har designats för att göra databasen användbar för Helsingborgs Bryggeri och för att möjliggöra kommunikationer av informationshantering.



Figur 4.1 Överskådligbild på strukturen.

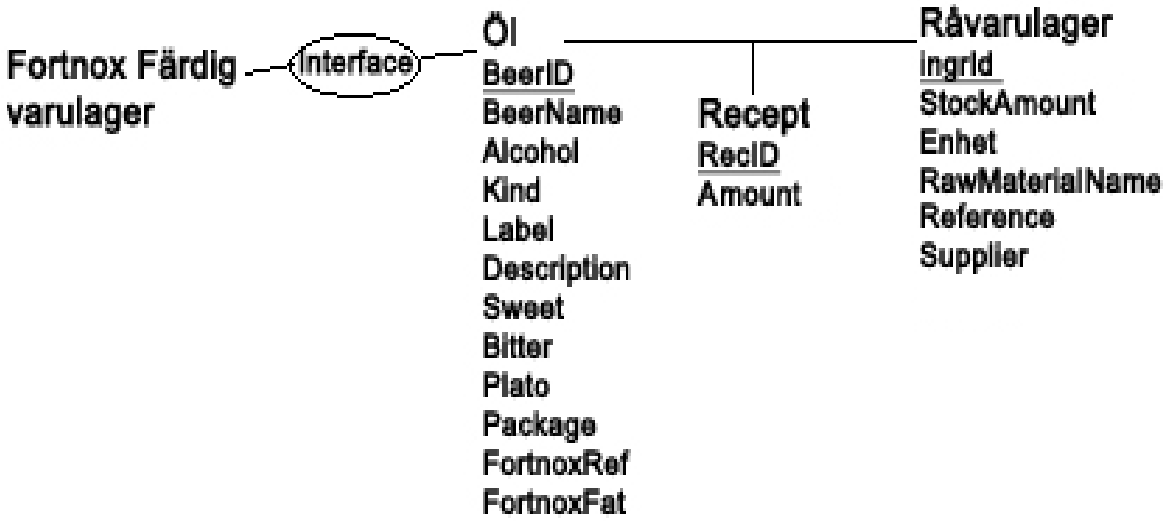
Det har varit viktigt att hålla en god struktur i den kod som har skrivits för gränssnittet. Detta medför att det är enklare för programmeraren att känna igen sig i den kod som läses.

5 Databas

Helsingborgs Bryggeri är ett producerande företag och har därför ett materiallager, produktion och ett färdigt produktlager. Projektets databas skapas efter hur bryggeriets produktion är uppbyggd. I nedanstående rubriker förklaras de olika entiteternas funktion och uppbyggnad. Databasen är skapad med hjälp av ett E/R-diagram för att relationerna mellan de olika entiteterna inte ska bli felaktiga.

5.1 Entiteter och relationer

Den första delen av databasen är anpassad efter produktion av olika typer av recept för att brygga öl. De är ihopkopplade med relationer som bygger huvudsakligen på att när öl bryggs så uppdateras de olika råvarorna som finns i lager. Varje gång ett öl bryggs så minskas mängden av respektive råvara i råvarulagren efter respektive recept, vilket är en förenkling för ett producerande företag gällande lagerhantering. Databasen innehåller även andra delar som lagrar entiteter för exempelvis autentisering, behörighet och kundregister.



Figur 5.1 Överskådlig bild på relationen mellan de olika entiteterna.

5.1.1 Råvarulager

Råvarulagerentiteten är uppbyggd för att ha en direktkoppling till entiteten Öl. Den ska innehålla alla råvaror som behövs för att tillverka de olika ölen efter respektive recept. Entiteten innehåller lagermängden av de olika råvarorna och dessa uppdateras löpande efterhand som olika öl tillverkas. Entiteten innehåller även variabler för informationslagring för de olika typerna av råvaror.

5.1.2 Recept

Eftersom entiteterna Öl och Råvaror har en många till många relation så krävs en relationstabell i form av entiteten Recept, där Recept har som funktion att vara relationen mellan de olika råvarorna från råvarulagret till de olika ölen som finns lagrade i Öl-entiteten. I Recept anges hur mycket av varje ingrediens som används i varje bryggning av de olika ölen.

5.1.3 Öl

Öl lagrar de olika ölen som Helsingborgs Bryggeri tillverkar. Varje öl får en egen identitet (Öl-Id), vilket är en genomgående metod i hela databasen som till exempel att en råvara får även den ett eget ingrediens-Id. Detta utförs så på grund av att det ska vara enkelt att skilja på variabler i databasen som kan råka få samma namn, exempelvis så kan det finnas flera sorters humle. Det är därför viktigt att varje entitet får en unik nyckel som inte kan blandas ihop med någon annans. Entiteten Öl samlar information om de olika ölen i form av olika typer av produktbeskrivningar, som exempelvis alkoholhalt, sort och grad av beska. Produktbeskrivningen finns med för att enklare kommunicera information om ölen från bryggare till annan personal. När en öl skapas i databasen kan kopplingar göras till liknande artiklar i Fortnox genom att lagra Fortnox referenser i olika variabler i entiteten.

5.1.4 Order

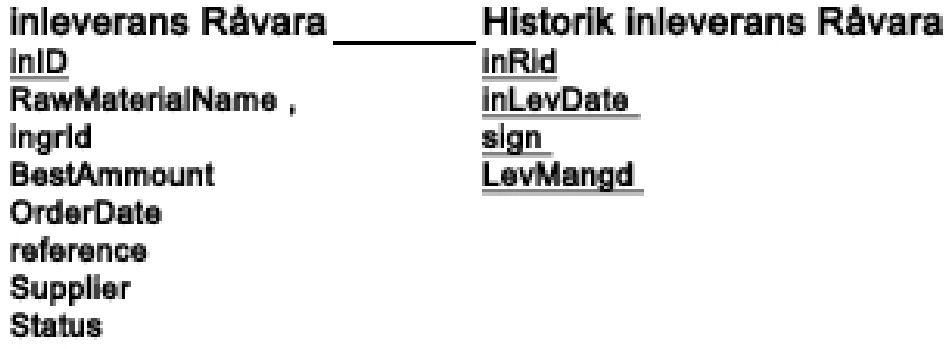
Till en början skapades entiteter för att lagra aktuella ordrar och makulerade ordrar hämtade via Fortnox API. Detta för att löpande uppdatera Helsingborgs Bryggeris färdigvarulager till rätt antal. Det som kopplade samman de olika ölen och ordrarna var entiteten Kolli eftersom de uppstod en många till många relation mellan dem.

När den här metoden implementerades upptäcktes att den inte var särskilt effektiv och tog lång tid för gränssnittet och databasen att exekvera. Istället valdes en lösning med Fortnox som fyller samma funktion men som arbetar mycket mer effektivt.

5.1.5 Materiallager

Helsingborgs Bryggeri använder sig av olika typer av material för tillverkning och försäljning av sin öl och dessa måste lagras i databasen. Materialentiteten har ingen koppling till övriga databasen eftersom det inte finns någon bestämd relation mellan Öl och vilket material som används till varje öl. Den här kopplingen kan i framtiden göras liknande som kopplingen mellan Öl, Recept och Råvara när Helsingborgs Bryggeri exempelvis har bestämt vilka flaskor som används för respektive öl.

5.1.6 Inleveranser - avisering och bekräftelse



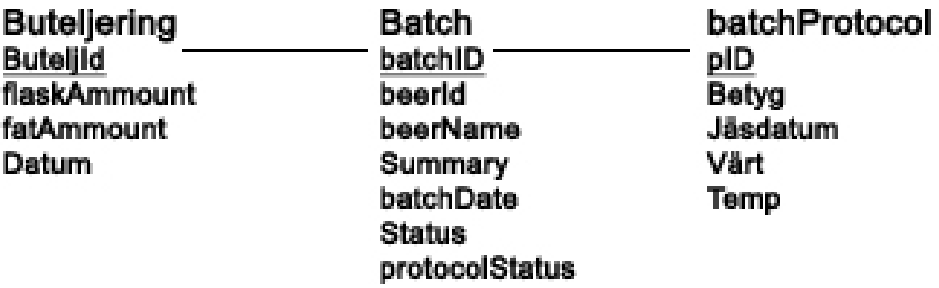
Figur 5.2 Överskådlig bild på relationen mellan de olika entiteterna.

För att personalen enkelt skall ha tillgång till vad för råvarumaterial och övrigt material som beställs så finns entiteter för detta ändamål. Detta möjliggör även för personalen att vid beställning se vad som är beställt och även korrigera den inkommande leveransen så de olika lagren blir korrekt uppdaterade.

Efter en inleverans är bekräftad så samlas respektive inleverans i en entitet som hanterar historiken för inleveranser, så att Helsingborgs Bryggeri enkelt ska kunna följa leveranser och olika leverantörers statistik.

5.1.7 Batch och färdigvarulager

Entiteterna Batch, Batchprotocol och Buteljering lagrar ölen som tillverkas för respektive batch. Vid buteljering kan användaren ange hur många flaskor eller fat av respektive batch som buteljerats och detta lagras i entiteten Buteljering vilken är kopplat till entiteten Batch.
Entiteten Buteljering håller isär en batch av en viss öl om det är fat eller flaska eftersom en öl lagras som antingen fat eller flaska i Fortnox.
Batchprotocol ersätter ett manuellt protokoll som Helsingborgs Bryggeri använt sig av då de utför kontroller av de olika batcherna som tillverkas.
De här två entiteterna ger möjligheten att ta fram historik om tidigare gjorda batcher. Exempelvis kan användaren ta fram vilka ingredienser som användes till en batch samt vilken leverantör som levererade de olika ingredienserna.



Figur 5.3 Överskådlig bild på relationen mellan de olika entiteterna.

När användaren angett antal öl som buteljerats av en batch, så sänds informationen vidare till Fortnox API så informationen kan lagras i Helsingborgs Bryggeris databas hos Fortnox.

Ett färdigvarulager lagras inte i egna databasen och gränssnittet använder sig istället av Fortnox artikelregister. Med den här metoden undviks det att information blir dubbellagrad eftersom Helsingborgs Bryggeri behandlar ordrar direkt i Fortnox och då måste artikeln och antal vara registrerade i artikelregistret. Olika typer av ordrar och makulerade ordrar uppdaterar även artikelregistret direkt i Fortnox. Det hade inte blivit effektivt att göra det med det egna gränssnittet och databasen eftersom Fortnox databas gör det snabbare internt än att kommunicera information till och från Helsingborgs Bryggeris databas. Metoden som valts undviker även att gränssnittet behöver hantera och lagra ordrar och makulerade ordrar i databasen, vilket hade blivit väldigt tidskrävande när antalet lagrade ordrar och makulerade ordrar stigit i antal.

5.1.8 Kundregister

För att lagra Helsingborgs Bryggeris kundregister skapades entiteten Kund. Information som lagras i Kund hämtas direkt från Fortnox via Fortnox API. Här lagras kundnamn, adress och e-post.

5.1.9 Users

Entiteten Users lagrar all information om respektive användare som gränssnittet behöver veta för att kunna autentisera användaren vid en inloggning på hemsidan. Användarnamn och lösenord är grundläggande för entiteten men den innehåller även variabler i form av booleska variabler som används för att kontrollera användarens behörigheter.

Alternativet till entiteten Users hade varit att skapa två entiteter där ena enbart lagrade användarnamn och lösenord men som hämtade information från en andra entitet som specificerade olika grader av behörighet. Det här alternativet ansågs dock vara mer passande för en större organisation med fler användare. Eftersom Helsingborgs Bryggeri är ett mindre företag där personal får ta ett större ansvar så är det bra att administratören kan ge olika typer av behörighet till olika användare.

6 Användarvänlig anpassning

För att databasen ska hållas fri från dubbellagringar och andra felaktigheter så är det viktigt att anpassa gränssnittet efter användare. På de ställen i det grafiska användargränssnittet där användaren har möjligheten att skriva till databasen för att skapa fler tabeller är det viktigt att göra så att användaren inte kan använda databasen felaktigt. Ett exempel är att om en användare ska lägga en inleveransavisering av en råvara, så måste ingrediensen redan finnas registrerad i råvarulagret. De råvaror som är registrerade i råvarulagret visas i en rullista på inleveranser och med den metoden undviker gränssnittet att användaren dubbellagrar information.

Figur 6.1 Bild från gränssnittet. Här visas hur en avisering genomförs.

Det är viktigt att de funktioner som finns på hemsidan gör så mycket av arbetet som ska utföras som möjligt för att underlätta arbetet för Helsingborgs Bryggeris personal. Exempelvis är det viktigt att det ska krävas ett fåtal klick på hemsidan och kan inmatningar vara färdigskrivna så ska de helst vara det.

7 SQL-injection och Cross site scripting

Av säkerhetsskäl så måste databasen vara skyddad mot felaktiga och obehöriga indata. Dessa skydd måste finnas överallt där indata skrivs från det grafiska användargränssnittet till databasen.

Attackerna kan ske både igenom SQL-injection och XSS (cross site scripting).

Det finns en funktion för att skydda databasen mot SQL-injection [9] som heter *mysql_real_escape_string()* den behandlar indata som skrivs till databasen och gör det omöjligt att skicka SQL-script in till databasen. En SQL-injection attack kan vara script skrivna för att radera en hel databas.

För att skydda mot XSS-attacker används funktionen *htmlentities*, vilket är ett tillvägagångssätt för att byta ut farliga tecken, exempelvis de tecken som används för att skriva de olika script-språken som kan användas för att manipulera gränssnittet och databasen.

```php
<?php

if(isset($_POST['namn'])){
    if($_POST['namn']==""){
    }else{

$namn=mysql_real_escape_string(htmlentities($_POST['namn']));
$mangd=mysql_real_escape_string(htmlentities($_POST['mangd']));
$ref=mysql_real_escape_string(htmlentities($_POST['ref']));
$lev=mysql_real_escape_string(htmlentities($_POST['lev']));
$enhet=mysql_real_escape_string(htmlentities($_POST['enhet']));

$sql="insert into rawmaterial(RawMaterialName,StockAmmount,reference,Supplier,enhet)
values('$namn','$mangd','$ref','$lev','$enhet')";
mysql_query($sql);
```

Figur 7.1 Bilden visar hur de olika funktionerna används i PHP-scriptet.

Funktionen tar emot indata i variablerna och behandlar dem från obehöriga tecken. Ett script kan vara mycket farligt för databasen och förstöra databasens uppbyggnad helt.

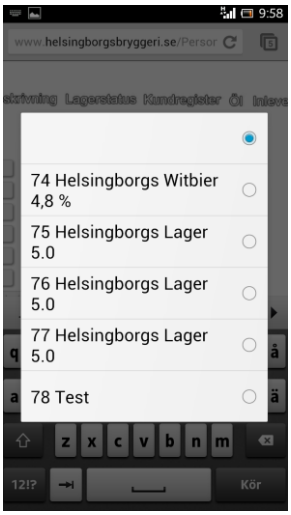
Ett exempel på ett script som kan skrivas in är följande
<script type="text/javascript">alert("Hello");</script>
Om scriptet skulle lagras utan behandling i databasen så skulle det exekverats vid varje tillfälle då gränssnittet anropat det.
Men eftersom gränssnittet behandlar scriptet så kommer det se ut som följande och vara helt ofarligt för gränssnitt och databas.
<script type="text/javascript">ale

8 Gränssnitt

Det grafiska användargränssnittet är uppbyggt i en hemsida med olika flikar som innehåller olika funktioner, främst till för att kommunicera med databasen. Det grafiska användargränssnittet är skapat för att vara lätt att använda och för att uppfylla de krav som Helsingborgs Bryggeris olika avdelningar har på informationshantering.

Figur 8.1 Bild på gränssnittet. Här visas startsidan.

Hemsidan är programmerad så att den arbetar med dynamiska funktioner för att vara snabbare och kraftfullare. Med hjälp av dynamiska funktioner blir man inte begränsad av sitt grafiska användargränssnitt eftersom den då skapar utrymme efter behov. Tillsammans med en databas som alltid är dynamisk får användaren en dynamisk helhetslösning. Det enda som begränsar lösningen är det utrymme där det grafiska användargränssnittet och databasen lagras på.



8.1 Kompatibelt användargränssnitt

Eftersom det finns ett flertal enheter som kan kommunicera data mellan varandra så behövs ett grafiskt användargränssnitt som blir kompatibelt med så många enheter som möjligt. På Helsingborgs Bryggeri används bland annat Ipad, PC och Samsungs surfplatta med flera. Något som dessa enheter har gemensamt är att de klarar av att läsa en webbsida med hjälp av sina olika webbläsare. Istället för att tillverka applikationer för de olika enheterna så anses det lättare att använda en

hemsida som grafiskt användargränssnitt eftersom den blir kompatibel med alla de olika enheterna.
En hemsida som grafiskt användargränssnitt möjliggör även att användaren kan använda valfri enhet, som till exempel en dator på ett internetcafé, för att kommunicera med databasen.

8.2 Autentisering

För att skydda databasen från obehöriga användare så måste gränssnittet innehålla någon typ av autentisering till hemsidan. Anledningen till att hemsidan skyddas är att det är den enda kommunikationsvägen rakt till databasen och den enda som skaparen kontrollerar och kan skydda.
Gränssnittet skyddas med autentisering via användarnamn och lösenord.

Första autentiseringssättet som användes var att lösenordsskydda de filer som bygger upp hemsidan. Detta gjordes genom att skydda den biblioteksmapp som samlade alla filer. Varje gång användaren startade en helt ny webbläsarsession och försökte få tillgång till hemsidan var den tvungen att autentisera sig med användarnamn och lösenord, vilket gjordes med hjälp av en Ht-access fil.

Det här sättet av autentisering fungerade på ett säkert sätt men vid en vidarekoppling till olika grader av behörighet för användare så visade sig det att vara bristfälligt då Ht-access skyddar filbiblioteket.
Istället valdes det att utföra autentisering genom databasen eftersom det då enkelt kan implementeras vilka behörigheter en användare har.

8.2.1 Behörighet

Tillsammans med en autentisering av användare anger även databasen till gränssnittet vilken typ av behörighet som användaren skall ha. De olika behörigheterna beror av vilken behörighet användaren blir tilldelad. Exempelvis kan inte användare ett ändra lagerstatus men kan tillverka öl. För att hålla en god administration av databasen så finns det en administratör som kan kontrollera att all information hanteras som den ska.

Bilden visar funktionen, under fliken Användare, på hemsidan där administratören kan skapa nya användare och ange olika behörigheter.
När en användare autentiserats och databasen har lämnat data om användarens behörighet till gränssnittet så skapas en ny session för användaren. Den nya sessionen lagrar att användaren är inloggad och vilka funktioner denne har tillgång till.

De variabler i PHP-scriptet som lagrar behörighetsdata är globala, i det samlade filbiblioteket, så att alla de olika kodade filerna som bygger upp gränssnittet kan använda sig av dem.
Detta möjliggör att flytta vidare behörighetsdata för användaren enkelt när hemsidan hänvisar till gränssnittets olika filer. För att säkerhetssystemet inte skall kunna manipuleras används här skydd mot SQL-injection och XSS.

Anledningen att använda olika behörigheter till hemsidan är i nuläget främst ett sätt att förhindra att felaktiga indata skickas till databasen, som exempelvis kan komma ur enkla misstag av en okunnig användare. Men på längre sikt om Helsingborgs Bryggeri väljer att utveckla vidare så kan det finnas större anledning att ha olika typer av behörigheter för användare.

Exempelvis om kunder till Helsingborgs Bryggeri skulle behöva tillgång till hemsidan finns det då möjligheten att begränsa vilken information som respektive kund har tillgång till.

De olika behörigheterna är idag specifika för olika funktioner på hemsidan med anledning av att det inte kommer vara ett stort antal användare av hemsidan. Alternativet hade varit att skapa olika grader av behörigheter där de olika graderna specificeras och en användare enkelt tilldelas en grad. Detta anses onödigt då det i princip hade skapats nästan en behörighetsgrad per användare.

```php
<?php
include_once('resources/init.php');
session_start();
if(isset($_POST['username'])){
    if((!$_POST['username']=="") && (!$_POST['password']=="")){
$user=mysql_real_escape_string(htmlentities($_POST['username']));
$pass=mysql_real_escape_string(htmlentities($_POST['password']));
$sql="select * from users where Username='$user'";
$answer=mysql_query($sql);
$ans=mysql_fetch_assoc($answer);
if($ans['Password']==$pass){
$_SESSION['user']=$user;
$_SESSION['pass']=$pass;
$_SESSION['tabort']=$ans['tabort'];
$_SESSION['laggtill']=$ans['laggtill'];
$_SESSION['tillverka']=$ans['tillverka'];
$_SESSION['bestall']=$ans['bestall'];
$_SESSION['uppdatera']=$ans['uppdatera'];
$_SESSION['admin']=$ans['admin'];

$_SESSION['start']=time();
$_SESSION['slut']=$_SESSION['start']+ (30*60); // en halvtimme
header('Location:Startsida.php');
}else{
header('Location:index.php');
}
}else{
header('Location:index.php');
}
```

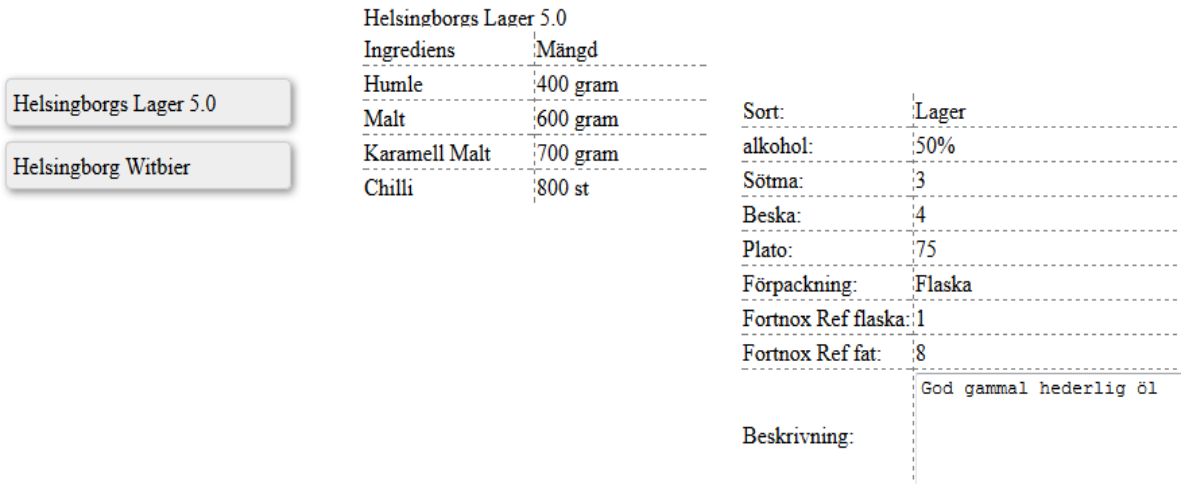
Figur 8.4 Utdrag ur gränssnittkoden. Här visas hur session används.

Bilden ovan visar ett utdrag ur gränssnittkoden dit gränssnittet hänvisas efter ett anrop med hjälp av en form_action. Koden är skapad för att autentisera användaren som vill få tillgång till hemsidan. Först startas en ny session som är den funktion som ger möjlighet att skapa och lagra de globala variablerna som i detta fall blir behörighetsdata. Gränssnittet kontrollerar sedan om någon har skrivit in användarnamn och lösenord annars blir användaren vid ett anrop tillbakaskickad till inloggningssidan. När användaren har skrivit in sina uppgifter så kontrollerar gränssnittet att det inte är någon form av SQL-injection eller XSS-attack som användaren försöker att utföra. Därefter kontrolleras det om användaren finns lagrad i databasen och en jämförelse utförs mellan det inskrivna lösenordet och lösenordet lagrat för den angivna användaren i databasen. Till sist, med en godkänd autentisering, hämtas användarens olika behörigheter och en sessionstid anges och när den är slut så loggas användaren ut automatiskt. På hemsidan finns det en flik, som heter Användare, som endast visas för den som har administratörsbehörighet där

användaren har möjlighet att skapa nya användare och ange behörighet för dem.

8.3 Produktbeskrivning

För att enkelt kommunicera ut information till anställda om varje öl som bryggeriet säljer innehåller hemsidan en flik som hanterar produktbeskrivningar.



Figur 8.5 Bild på gränssnittet. Här visas en produktbeskrivning.

På produktbeskrivning kan användaren se vilka ingredienser och hur mycket av dem varje öl innehåller. Det anges även information som lätt behöver vara tillgänglig för de olika avdelningarna som exempelvis beska och sötma. För varje öl som läggs in i databasen skapas det en knapp som i den vänstra delen av bilden, som hämtar recept och produktbeskrivningarna för respektive öl när dem aktiveras. Om användaren väljer att ta bort ölen ur databasen så försvinner knappen och produktbeskrivningen.

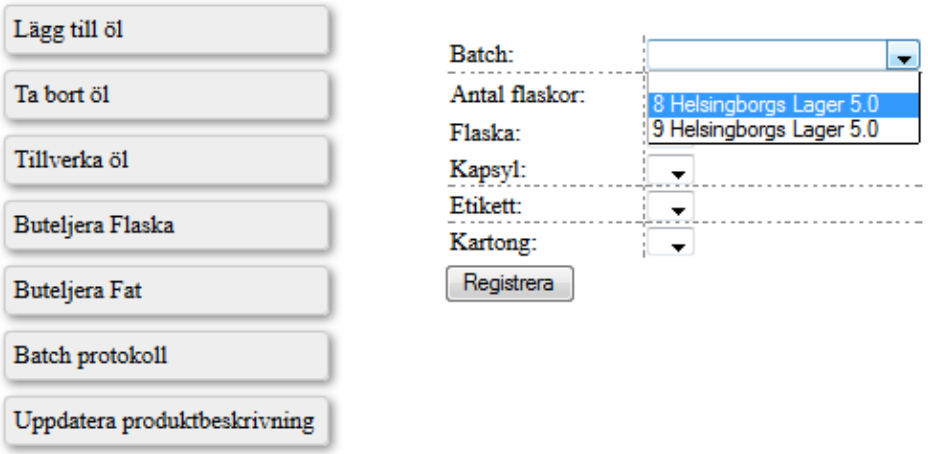
8.4 Lagerstatus

En viktig del för att förbättra informationshanteringen på bryggeriet är en form av systematisk lagerhållning. Tre olika lager har av den anledningen skapats i databasen och dessa kan alla visas och hanteras i fliken lagerstatus.
I lagerstatus kan användaren lägga till och ta bort råvaror och material vilket är en viktig grundläggande funktion eftersom exempelvis ett recept kan bara skapas på de råvaror som finns i råvarulager.

8.5 Öl

I fliken öl byggs kopplingar i databasen mellan de olika öl som skapas och råvarorna som finns i råvarulagret, eftersom i den här fliken anger användaren recepten för varje öl. Användaren får välja de olika ingredienserna i en lista från råvarulagret och sedan ange mängd av varje ingrediens som ska

användas. Den här typen av inmatning har valts för att användaren inte ska ha möjlighet att skapa recept med ingredienser som inte finns med i råvarulagret. Fördelen blir att användaren inte behöver skriva ingrediensens namn fler gånger än när den läggs till i råvarulagret.
Under fliken öl kan bryggaren gå in och ange när han tillverkat en öl och om det var några förändringar i receptet för ölen. När bryggaren trycker på tillverkning av en vald öl uppdateras råvarulagret automatiskt, en ny batch skapas och historik om tillverkningen lagras i databasen. En säkerhetsrutin har lagts till ifall användaren oavsiktligt tryckt på tillverkning. Detta har gjorts i form av en dialogruta som måste bekräftas innan tillverkningen utförs.



Figur 8.6 Bild på gränssnittet. Här visas hur man buteljerar en flaska.

I funktionerna Buteljera Flaska och Buteljera Fat kan annan personal ange hur många flaskor och fat som det blev av batchen efter buteljeringen.

Antalet buteljerade flaskor och fat av ett viss öl skickas sedan vidare automatiskt med hjälp av gränssnittet och Fortnox API till databasen hos Fortnox.

En enkel funktion som lagts till i fliken öl är "Uppdatera produktbeskrivning" där bryggaren efter behov kan uppdatera beskrivningen av de olika ölen.

8.6 Inleverans

Hemsidan hanterar även inleveranser av befintligt material. Funktionen möjliggör för användaren att enkelt avisera till övrig personal att en beställning är gjord av exempelvis en råvara. När sedan leveransen kommer kan lagerpersonalen via hemsidan bekräfta beställningen, ange levererat antal och signera leveransen. Levererat antal uppdaterar därefter tillhörande lagersaldo per automatik och historik om leveransen och beställningen lagras därefter i databasen. Inleverans är uppbyggd liknande fliken öl där användaren måste välja ur en lista, som hämtas från råvarulager och materiallager, vad den

ska beställa. På detta sätt undviker hemsidan att göra dubbellagringar av råvara eller material i databasen.

Figur 8.7 Bild på gränssnittet.

Av den anledningen måste användaren vid beställning, av för databasen okänt material, registrera materialet i lagerstatusfliken först. Med hjälp av den här metoden undviker man även här att skriva namnen på råvaror och material mer än en gång.

I fliken inleverans finns det en funktion för att uppdatera all information som finns om respektive råvara och material i databasen.

8.7 Historik

Under fliken Historik kan användaren söka ut information för alla batcher och inleveranser som har gjorts. Här kan man även ta fram varje batchprotokoll som någonsin fyllts i. Skulle någon information anges fel under tillverkningen så finns det funktioner under fliken Historik där användaren kan gå in och korrigera den lagrade informationen, om olika batcher, i efterhand.

8.8 Kommunikation med databas

För att utföra kommunikation mellan databasen och gränssnittet initieras databasen i PHP-scriptet. Detta utförs genom att skapa en extra fil som heter init.php som utför initieringen till databasen.

```
1  <?php
2          $db_host="localhost";
3          $db_user="root";
4          $db_pass="";
5          $db_name="testhb";
6          @mysql_connect("$db_host","$db_user","$db_pass")
7          or die ("Couldnt connect to MySql");
8          @mysql_select_db("$db_name") or die ("No name");
9          ?>
```

Figur 8.8 PHP-script från init.php.

Init.php inkluderas senare i varje PHP-script som behöver kommunicera med databasen istället för att skriva ny kod för initiering varje gång. Efter

initieringen har gjorts är det enkelt att förmedla SQL-frågor och påståenden till databasen från gränssnittet.

9 Utvecklingsmöjligheter

Eftersom databasen har fått en bra grundstruktur så finns det stora möjligheter att bygga på fler entiteter vid behov, om Helsingborgs Bryggeri vill utöka användandet av databasen för sin informationshantering.

Det finns en plan om att i framtiden när Helsingborgs Bryggeri har en fungerande PLC, som hämtar korrekta mätvärden ur processen, att skapa utrymme för detta i databasen och visa upp processen grafiskt i det grafiska användargränssnittet. Först och främst behövs det göras en kalibrering av de givare som ska användas för att få in korrekta värden.
För att skicka information från PLC till databasen behövs ytterligare utrustning för att hantera den kommunikation som uppkommer. Eftersom Helsingborgs Bryggeri har PLCn Mitsubishi FX3U så finns det olika alternativ för att utveckla kommunikationsvägen. Ett av alternativen hade varit att använda sig av OPC datahub [10] som översätter PLC-protokoll så att exempelvis program i olika operativsystem kan ta del av den information som PLC hämtar från en industriprocess.
Det går även att skriva en applikation som utför kommunikationen mellan PLC och databas, men detta anses vara det svåraste och mest tidskrävande alternativet att utföra kommunikationen på.
Helsingborgs Bryggeri använder sig idag av en E1000 terminal som är en form av ett HMI (Human-Machine Interface) som idag till största del endast visar upp information på en skärm. Till detta HMI finns det en produkt, BE DataCollector, som kan samla information från E1000 terminaler och sända vidare direkt till en databas.

I framtiden kommer BE DataCollector [11] att användas av Helsingborgs Bryggeri då den anses vara den tryggaste och smidigaste lösningen för att bygga upp kommunikationen mellan PLC och databas. Att programmera en egen applikation för kommunikationen kan möjligtvis vara en fortsättning av det här examensarbetet
Under projektets gång har det arbetats fram en färdig lösning för databasen, med hjälp av olika entiteter, för att ta emot och lagra information på ett korrekt sätt i databasen.

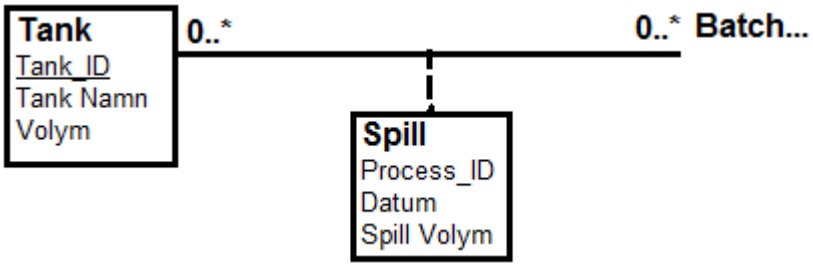


Bild 9.1 Bilden visar en ett exempel hur spill kan lagras.

Bilden ovan visar ett exempel hur olika spill kan lagras när vätskor pumpas över från en tank till en annan. Eftersom entiteterna Tank och Batch har en många till många relation krävs det en mellanentitet i form av Spill. De olika volymdifferenserna mellan de olika tankarna för en batch lagras i variabeln *Spill Volym* i entiteten Spill. Den här lösningen har inte implementerats i projektet men anses vara en utvecklingsmöjlighet för Helsingborgs Bryggeri vid sammankoppling med PLC. Det finns även möjlighet att skapa utrymme i databasen för andra värden som exempelvis tryck och temperatur. Ytterligare en möjlighet för framtiden hade varit en annan typ av kommunikation mellan PLC och gränssnitt då det kan skapas vidare gränssnitt för att styra bryggprocessen från hemsidan.

Integrationen med Fortnox kan växa sig större då det finns stora fördelar att använda sig av Fortnox API, som exempelvis att helt och hållet ha möjlighet att styra Fortnox med hjälp av gränssnittet som har skapats. Mer integration med Fortnox innebär även en mindre spridd informationshantering då all hantering kan gå via projektets gränssnitt.

10 Slutsats

Examensarbetet har under projekttiden fortlöpt på ett genomgående bra sätt. Samarbetet med Helsingborgs Bryggeri och deras samarbetspartners har fungerat väl och användargränssnitt och databas har utvecklats så att det tillgodoser Helsingborgs Bryggeris krav. Under projektet har det varit en del utmaningar med att anpassa sig till de nya kraven som uppstått då gränssnitt och databas har utvecklats. De nya kraven har uppkommit då det är enklare att se nya lösningar på problem efterhand som lösningar utvecklas. Exempelvis var databasen till en början en enhet men fick delas upp i mindre delar för att möta Helsingborgs Bryggeris krav om att kunna lagra historik, men fortfarande ha möjligheten att ta bort ur databasen via det grafiska gränssnittet. Eftersom databasen har skapats med en bra grundstruktur så har det varit enkelt att förändra databasens struktur under projektets gång. Den goda strukturen i den kod som har skrivits för att skapa gränssnittet har medfört att det varit enkelt att utveckla gränssnittet efter de krav som uppkommit under projektets gång.

Referenser

Här ska vi byta referensmetod och istället använda oss av klammer metoden.

[1] http://www.phpmyadmin.net/home_page/index.php 2013-04-15 kl 09:00
[2] http://php.net/ 2013-04-15 kl 09:30
[3] http://www.w3schools.com/html/html_intro.asp
[4] http://notepad-plus-plus.org/ 2013-04-16
[5] http://www.fortnox.se/ 2013-04-16
[6] http://www.xml.com/ 2013-04-16
[7] http://www.wampserver.com/en/ 2013-04-17
[8] http://www.apachefriends.org/en/index.html 2013-04-17
[9] http://php.net/manual/en/security.database.sql-injection.php
[10] http://www.opcdatahub.com/WhatIsOPC.html 2013-05-22
[11] http://www.beijer.se/web/web_se_be_se.nsf/AllDocuments/557D2A612EECAC5AC125796600470E8F 2013-05-23

Appendix A

Koden för att skapa databasen

```
CREATE TABLE IF NOT EXISTS batch (
  batchID int(11) NOT NULL AUTO_INCREMENT,
  beerId int(11) DEFAULT NULL,
  beerName varchar(30) DEFAULT NULL,
  Ammount int(11) DEFAULT NULL,
  Summary varchar(500) DEFAULT NULL,
  batchDate datetime NOT NULL,
  Status int(11) NOT NULL DEFAULT '0',
  protocolStatus int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (batchID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci
AUTO_INCREMENT=115;

CREATE TABLE IF NOT EXISTS batchprotocol (
  pID int(11) NOT NULL AUTO_INCREMENT,
  batchID int(11) DEFAULT NULL,
  Betyg varchar(30) DEFAULT NULL,
  PRIMARY KEY (pID),
  FOREIGN KEY batchID (batchID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS beer (
  BeerId int(11) NOT NULL AUTO_INCREMENT,
  BeerName varchar(50) DEFAULT NULL,
  fortnoxRef varchar(50) NOT NULL,
  fortnoxFat varchar(50) NOT NULL,
  alcohol double DEFAULT NULL,
  kind varchar(30) DEFAULT NULL,
  label varchar(30) DEFAULT NULL,
  description varchar(400) DEFAULT NULL,
  sweet int(11) DEFAULT NULL,
  bitter int(11) DEFAULT NULL,
  plato varchar(10) DEFAULT NULL,
  package varchar(30) DEFAULT NULL,
  PRIMARY KEY (BeerId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS buteljering (
  buteljId int(11) NOT NULL AUTO_INCREMENT,
  flaskAmmount int(11) DEFAULT NULL,
  fatAmmount int(11) DEFAULT NULL,
  datum datetime DEFAULT NULL,
  batchId int(11) DEFAULT NULL,
  PRIMARY KEY (buteljId),
  FOREIGN KEY batchId (batchId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;
```

```
CREATE TABLE IF NOT EXISTS customer (
  cusId int(11) NOT NULL AUTO_INCREMENT,
  companyName varchar(50) DEFAULT NULL,
  address varchar(50) DEFAULT NULL,
  email varchar(50) DEFAULT NULL,
  PRIMARY KEY (cusId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS fvl (
  FVLID int(11) NOT NULL AUTO_INCREMENT,
  beerId int(11) DEFAULT NULL,
  Ammount int(11) DEFAULT NULL,
  PRIMARY KEY (FVLID),
  FOREIGN KEY beerId (beerId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS inlevmaterial (
  mID int(11) NOT NULL AUTO_INCREMENT,
  MaterialName varchar(30) NOT NULL,
  MaterialID int(11) NOT NULL,
  BestAmmount int(11) DEFAULT NULL,
  OrderDate datetime DEFAULT NULL,
  reference varchar(50) DEFAULT NULL,
  Supplier varchar(50) DEFAULT NULL,
  status int(11) DEFAULT NULL,
  PRIMARY KEY (mID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS inlevmaterialid (
  inMid int(11) NOT NULL AUTO_INCREMENT,
  mID int(11) DEFAULT NULL,
  inLevDate datetime DEFAULT NOT NULL,
  sign varchar(30) DEFAULT NULL,
  LevMangd int(11) DEFAULT NULL,
  PRIMARY KEY (inMid),
  FOREIGN KEY mID (mID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci
AUTO_INCREMENT=2 ;

CREATE TABLE IF NOT EXISTS inlevravara (
  inID int(11) NOT NULL AUTO_INCREMENT,
  RawMaterialName varchar(50) DEFAULT NULL,
  ingrId int(11) NOT NULL,
  BestAmmount int(11) DEFAULT NULL,
  OrderDate datetime DEFAULT NOT NULL,
  reference varchar(30) DEFAULT NULL,
  Supplier varchar(50) DEFAULT NULL,
  Status int(11) DEFAULT NULL,
  PRIMARY KEY (inID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_cI;
```

```
CREATE TABLE IF NOT EXISTS inlevravaraid (
  inRid int(11) NOT NULL AUTO_INCREMENT,
  inID int(11) DEFAULT NULL,
  inLevDate datetime DEFAULT NOT NULL,
  sign varchar(30) DEFAULT NULL,
  LevMangd int(11) DEFAULT NULL,
  PRIMARY KEY (inRid),
  FOREIGN KEY inID (inID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS materials (
  MaterialID int(11) NOT NULL AUTO_INCREMENT,
  Type varchar(30) NOT NULL,
  MaterialName varchar(30) NOT NULL,
  ammount int(11) DEFAULT NULL,
  enhet varchar(30) NOT NULL,
  info varchar(30) DEFAULT NULL,
  Refnr varchar(50) DEFAULT NULL,
  supplier varchar(30) NOT NULL,
  PRIMARY KEY (MaterialID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS rawmaterial (
  ingrId int(11) NOT NULL AUTO_INCREMENT,
  StockAmmount int(11) DEFAULT NULL,
  enhet varchar(30) NOT NULL,
  RawMaterialName varchar(50) DEFAULT NULL,
  reference varchar(30) DEFAULT NULL,
  Supplier varchar(50) DEFAULT NULL,
  PRIMARY KEY (ingrId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;

CREATE TABLE IF NOT EXISTS recipe (
  RecID int(11) NOT NULL AUTO_INCREMENT,
  ammount varchar(30) DEFAULT NULL,
  ingrId int(11) DEFAULT NULL,
  BeerId int(11) DEFAULT NULL,
  PRIMARY KEY (RecID),
  FOREIGN KEY BeerId (BeerId),
  FOREIGN KEY ingrId (ingrId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;
```

```
CREATE TABLE IF NOT EXISTS users(
  Username varchar(30) NOT NULL DEFAULT " ",
  Password varchar(30) DEFAULT NULL,
  tabort tinyint(1) DEFAULT NULL,
  laggtill tinyint(1) DEFAULT NULL,
  tillverka tinyint(1) DEFAULT NULL,
  bestall tinyint(1) DEFAULT NULL,
  uppdatera tinyint(1) DEFAULT NULL,
  admin tinyint(1) DEFAULT NULL,
  PRIMARY KEY (Username)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_swedish_ci;
```

Appendix B

Ordlista

Batch- Är ett parti ur en produktion,

Boyce-codd- Normal form i en relationsdatabas.

CSS- Cascading Style Sheets. Är en filtyp som används tillsammans med HTML som beskriver utseendet.

Entitet- Beskriver vilka variabler som lagrad i varje tabell.

HMI- Human-Machine Interface

HTML-HyperText Markup Language

PHP- Hypertext preprocessor

SQL- Structured Query Language

XML- Extensible Markup Language