Umeå universitet
Institutionen för datavetenskap

14 juni 2004

Prototyp för webbaserad sammanställning av asfaltprover vid NCC Roads AB

Examensarbete Datavetenskap

Johan Bryngelsson (tm98jbn@cs.umu.se)
Jonas Domeij (dva99jdj@cs.umu.se)

Handledare:
Roger Lundberg (roger.lundberg@ncc.se)
Jerry Eriksson (jerry@cs.umu.se)

Abstract

This thesis describes the development of a prototype system for *NCC Roads AB* in Umeå. The goal of the prototype is to create a more simple and effective method for handling sample results and recipes. This prototype will be used to evaluate the possibility of using a similar system in production.

The resulting system is built around *Java* technology using an abstraction layer called *Struts* to separate the HTML content from the *Java* code. As storage of all information in the system we are using a database system called *PostgreSQL*.

Innehåll

Kapitel 1

Introduktion

Detta examensarbete har bedrivits vid *NCC Roads AB* i Umeå på uppdrag av arbetschef Roger Lundberg. Det uppstod genom personlig kontakt med Roger Lundberg där problemet presenterades och vi fick i uppgift att presentera en tänkbar lösning. Denna lösning togs sedan upp på ett styrelsemöte inom företaget. Styrelsemötet gav positivt gensvar och godkände den tänkta lösningen. Lösningsförslaget kommer att återges nedan.

1.1 Om NCC Roads AB

NCC Roads AB bedriver verksamhet i Sverige, Norge, Finland, Polen, Estland, Lettland, Litauen och delar av Ryssland. Verksamhetsområden inkluderar produktion av kross-, asfalt- och fabriksbetongprodukter samt beläggningsarbeten, vägservice, vägunderhåll och trafiksäkerhet i form av skyltning med mera. Årsproduktionen uppgår till cirka 6 miljoner ton asfalt, 25 miljoner ton kross- och stenmaterial. Detta gör *NCC Roads AB* till nordens största aktör i branschen. Huvudkontoret återfinns i Solna, Stockholm.

1.2 Syfte och mål med examensarbetet

NCC bedriver laboratorier som analyserar prover rörande asfaltbeläggning. Dessa laboratorier utför analys av produktion och beläggning åt *NCC* företags beläggning. När analys utförs åt andra än de själva gäller vissa regler kring tillgänglighet till analysvärden, vilket kallas *ackrediterat* arbete. För att hantera detta utfärdas ett kontrakt mellan parterna.

Prover för analys tas på kross- och stenmaterial som ska användas för specifika projekt, vilka kontrolleras enligt vissa krav av bland annat en beställare. Dessa krav kan liknas med en typ av recept på de olika beståndsdelarna i asfaltmassan, där deras storlek och mängd specificeras. Även lagd asfalt kontrolleras och detta görs genom att ur asfalten ta borrprover som kan liknas vid *hockeypuckar*. Dessa delar bryts ner och analyseras efter visst krav och recept. När analysen är klar avgörs om materialet möter krav- och receptspecifikationen. De analysvärden som inte är godkända medför att *NCC Roads AB* tvingas betala straffavgifter.

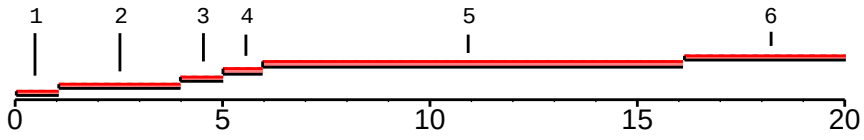
Hantering av analysvärden och övrig data gällande produktion och beläggning har hittills hanterats via ett system på laboratoriet. Informationen har sedan sparats i en databas endast tillgänglig för laborationspersonalen. Detta har inneburit att överföring av informationen till en beställare måste ske antingen via ett fax eller som ett *PDF*-dokument via elektronisk post (e-post). Detta har medfört att det oftast är endast en person som har resultatet av provet. När en sammanställning skall presenteras måste sammanställaren ha samtliga prover. Om ett provresultat saknas måste sammanställaren ta kontakt med laborationspersonalen. Dessa måste då söka efter provresultatet och sedan sända detta till sammanställaren. Detta förfarande tar mycket dyrbar tid ifrån personalen.

Syftet med examensarbetet är att effektivisera hanteringen av informationsflödet av analysvärden och sammanställningar samt göra dessa tillgängliga för användare direkt när laborationsresultatet färdigställts.

Målet med examensarbetet är att utveckla en prototyp av ett system som skall förenkla hantering och distribution av analysvärden och sammanställningar för *NCC Roads AB* i Umeå. Systemet skall i så stor utsträckning som möjligt inte påverka hanterandet av prover för laborationspersonalen.

1.3 Tidsdisposition och planering

Arbetet inleddes med att strukturera upplägget gällande tidsdisponering för de olika faserna i systemutvecklingen.



Figur 1.1: Tidsdiagram i veckor gällande arbetsplanering.

I figur 1.1 visas ett tidsdiagram över det planerade arbetet. Nedan förklaras de olika fasernas innehåll.

1 Besöka *NCC Roads AB* och dess laboratorie samt ha inledande möte med företagets handledare och övrig berörd personal. Få förståelse för deras problem och vilket syftet examensarbetet skulle uppfylla. Tidsåtgången för detta var planerat till en vecka.

2 Designfasen i systemutvecklingen där tänkt lösning skulle arbetas fram och designas. Tidsåtgång tre veckor.

3 Val av utvecklingsverktyg för att realisera systemet utifrån designen. Även självutbilding av verktygen om det var nödvändigt. Tid avsatt till detta var en vecka.

4 En vecka semester

5 Implementation av systemet skulle inledas. Under denna fas skulle även testkörningar och felsökning ingå. Avsatt tid för denna fas var tio veckor.

6 Sista fyra veckorna var avsatta till att avsluta rapport samt utbilda berörd personal på *NCC Roads AB* på de olika delarna av systemet.

Planeringen innefattade totalt tjugo veckor. Till ovanstående planering kan tilläggas att kontinuerligt arbete med rapport skulle ske under utvecklingens gång. Efter varje slutförd vecka skulle ett kortare möte med handledaren på företaget genomföras för att verifiera lösningens korrekthet. Vid inledningen av arbetet saknades handledare från universitet. Detta gjorde att möte med denne frångicks vid planering.

1.4 Förslag till lösning av problem

Efter att ha erhållit problembeskrivning från Roger Lundberg bestämdes möte med laborationschefen Lisbet Karlsson som presenterade nuvarande programvara. Programvaran de använde var relativt komplicerad vilket gjorde att vi kom fram till två olika lösningsförslag. Det första förslaget var ett fristående program och andra förslaget var ett webbaserat gränssnitt.

Vi analyserade dessa två lösningsförslag utifrån tre olika synvinklar. Dessa var lagring av information, gränssnitt och hur uppdatering av information skall fungera.

Lagring av information

För lagring hade vi två förslag, databas eller filsystem. En central databas ger snabb åtkomst för intressant information och stor flexibilitet. Genom att använda grupper med olika säkerhetsnivåer kan informationsflödet kontrolleras på ett lämpligare och säkrare sätt. Finns informationen i ett filsystem tvingas användaren spara den på lokal lagringsmedia, vilket innebär att säkerheten ligger hos användarens system. Detta är inte kontrollerbart för en administratör, vilket kan leda till att sekretessbelagd information kommer i fel händer. En uppdatering av databasen ger alla användare direkt samma information jämfört med filsystem där användaren först måste uppdatera informationen för att synkronisera med andra användare.

Gränssnitt

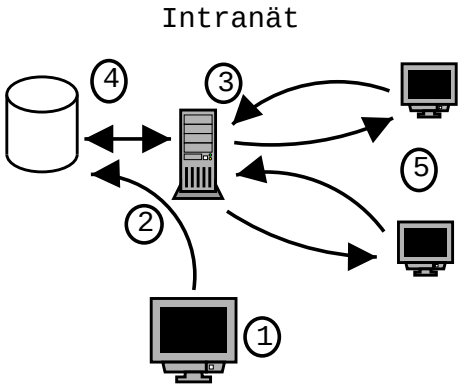
Ett fristående program och eventuella uppdateringar måste distribueras till alla användare. Detta fungerar om det är ett fåtal användare, dock inte om det rör sig om en större mängd. Underhållet blir varken hållbart eller kontrollerbart. Ett fristående program blir ofta komplicerat med en stor inlärningströskel för personalen. Ett webbaserat gränssnitt kan kontrolleras centralt när det gäller bland annat uppdateringar och är även intuitivt att använda. Vissa avvikelser kan dock förkomma i utseende beroende på webbläsare.

Uppdatering av information

Vi hade två förslag för uppdatering av information. Det första var ett gränssnitt där användaren själv matar in värden och det andra baserades på information från nuvarande program. Vid manuell inmatning av information finns alltid risken för fel. Manuell inmatning skulle även medföra dubbelt arbete för laboratoriepersonalen. Exportering av information från nuvarande program gör att inget dubbelarbete uppstår och att felrisken begränsas. Nuvarande program klarade inte exportera information vilket innebar att detta förslag krävde uppdatering av programmet.

1.5 Val av systemdesign

Efter att ha analyserat förslagen samt diskuterat med laboratoriepersonalen bestämde vi oss för följande övergripande design av systemet.



Figur 1.2: Design av system för laborationsresultat från asfaltprover

Uppdatering av laborationsresultat i databasen skulle göras genom att använda en utbyggd version av nuvarande laborationsprogram(1). Programmet skulle generera all nödvändig information som sedan skulle skickas upp till databasen via ett webbaserat gränssnitt(2). Användarsystemet skulle bestå av ett webbaserat system(3) kopplat till den central databasen(4). Användarna(5) ska kunna koppla upp sig mot systemet via sin webbläsare.

Effektiviseringen i denna design gällande användare är att denne inte behöver ta kontakt med laboratoriet för att få analyssvar eller sammanställningar. Användaren kan komma åt analysvärden relaterade till sin användare oavsett om denne befinner sig vid sin eller någon annans dator, enda kravet är att det finns en internetanslutning och en webbläsare. Effektivisering för laboratoriet blir kännbar då dessa inte behöver generera sammanställningar samt delvis inte behöva göra lika många utskick av dessa. Eftersom ackrediterat arbete kräver en signerad kopia måste fortfarande ett utskick i pappersform ske, de behöver dock endast användas vid verifiering av resultat.

Kapitel 2

Systemöversikt

2.1 Detaljerad beskrivning av systemdesign

Ett webbaserat system som bygger på en databas med analysvärden och sammanställningar där analysresultat laddas upp till databasen från laboratoriet via en webbserver innehållande en parser. Analysresultatet genereras av den befintliga programvaran genom en utökning. Denna utökning skulle utvecklaren av programvaran konstruera och tillhandahålla. Filtypen på analysresultat som laddas upp till databas skulle vara av ".xls" (*MS Excel*). När användare av systemet vill se prover eller göra en sammanställning loggar hon/han in med användarnamn och lösenord. Användarens behörighetsgrad kontrolleras i form av grupper med olika rättigheter beroende på vilken position i företaget användaren besitter. Det skulle finns tre typer av huvudkonton vilka var administratörs-, laboratorie- och användarekonto. Alla använder sig av ett webbaserat gränssnitt mot systemet. Administratören ska hantera alla användare och deras grupptillhörigheter med mera. Laboratoriekonto hanterar all uppdatering av databasen med avseende på analysvärden. Användarkonto hanterar all sammanställning av analysvärden för presentation.

2.2 Plattformar och utvecklingsmiljöer

För att realisera systemet har endast programvaror och utvecklingsmiljöer med öppen källkod utnyttjats. Detta gjordes av två skäl, inga licensavtal behövdes för företaget och alla de senaste produkterna kunde laddas ner från nätet. De olika programvarorna och miljöerna som använts under utvecklingen är *Tomcat*[16], *Struts*[14], *Log4j*[9], *JCharts*[7], *PostgreSQL*[13], *Java*[6], *Ant*[2] och *POI*[11]. En närmare förklaring av dessa kommer nedan. För systemet användes plattformarna *Windows* och *Linux*.

Systemet har utvecklat för *Windows* och *Internet Explorer* men fungerar även för *Mozilla*. Inriktningen mot *Windows* gjordes naturligt, då företaget använder detta. Enda undantaget är att databasen ligger på en *Linux* maskin då programvaran för databasen idag endast finns i en beta version för *Windows*. Tanken är dock att även databasen i framtiden ska ligga på en *Windows* maskin.

Tomcat är en *Java*-baserad webbserver utvecklad av *Jakarta-Apache* projektet och är utmärkt lämpad för webbapplikationer. Det som kan tilläggas är att *Struts*, *Log4J*, *Ant* och *POI* ingår i samma projekt.

Struts är ett ramverk för att konstruera webbapplikationer, speciellt där mycket information ska anges eller visas i formulärform. Denna typ av webbapplikationer behöver ofta verifieringar av olika indata vilket *Struts* tillhandahåller ett bra stöd för. *Struts* är som tidigare nämnts ett ramverk vilket bygger på *JavaBeans*, *JSP* (*Java Server Pages*) och servlets. *JavaBeans* är själva logiken i systemet. *JSP* är det som genererar vad användaren ser på skärmen. Servlets är kontrollerare av informationsflödet i systemets bakgrund, det vill säga mellan *JavaBeans* och *JSP*. *Struts* kommer nedan att behandlas under en egen avdelning med en mer detaljerad beskrivning.

Log4J är ett loggningsverktyg som används för att uppväga den undermåliga loggnings- och debuginformation som *Struts* erbjuder. *Log4J* är utvecklat under *Java* men på grund av sina kvaliteter gällande loggning görs portningar idag mot andra utvecklingspråk som exempel *C++*.

JCharts används för att göra grafdiagram. Även detta verktyg är utvecklat under *Java*. *JCharts* klara alla typer av grafdiagram som exempel stapeldiagram och linjediagram.

PostgreSQL är en objektrelationsdatabas med stöd för avancerade *SQL* funktioner. Denna databas är allmänt betraktad som den mest avancerade opensource databasen i världen.

Java Logiken i systemet har utvecklats med programspåket *Java*. Grunden till *Java*-miljön är uppfunnen och konstruerad av *Sun Microsystem*. Kompilering av stora system medför ofta problem gällande beroende o.s.v. För att förenkla till exempel kompileringsberoenden har *Ant* använts.

Ant kan jämföras med "*make*"-verktyget i *Unix*. *Ant* kan även användas för att kopiera, flytta och ta bort filer vilket förenklar och effektiviserar arbetet under utvecklingen.

POI har utvecklats för manipulering av *Microsoft Office* dokument- och kalkylfiler. Informationsfiler som kommer att genereras av laboratoriesystemet skall vara av *Excel* format. Sedan via *POI* hämtas den väsentliga informationen ur dokumentet läggs in i databasen.

Kapitel 3

Struts

Struts ett ramverk för webbaserade applikationer. Källkoden för mjukvaran är öppen för alla. Verktyget används av mjukvarukonstruktörer för att snabbt och enkelt utveckla webbaserade applikationer. *Struts* baseras på redan standardiserade teknologier som *JavaBeans*, *Java servlets* och *JavaServer Pages* (*JSP*). Produkten får användas av vem som helst för eget eller kommersiellt bruk utan avgifter eller restriktioner.[15]

3.1 Historik

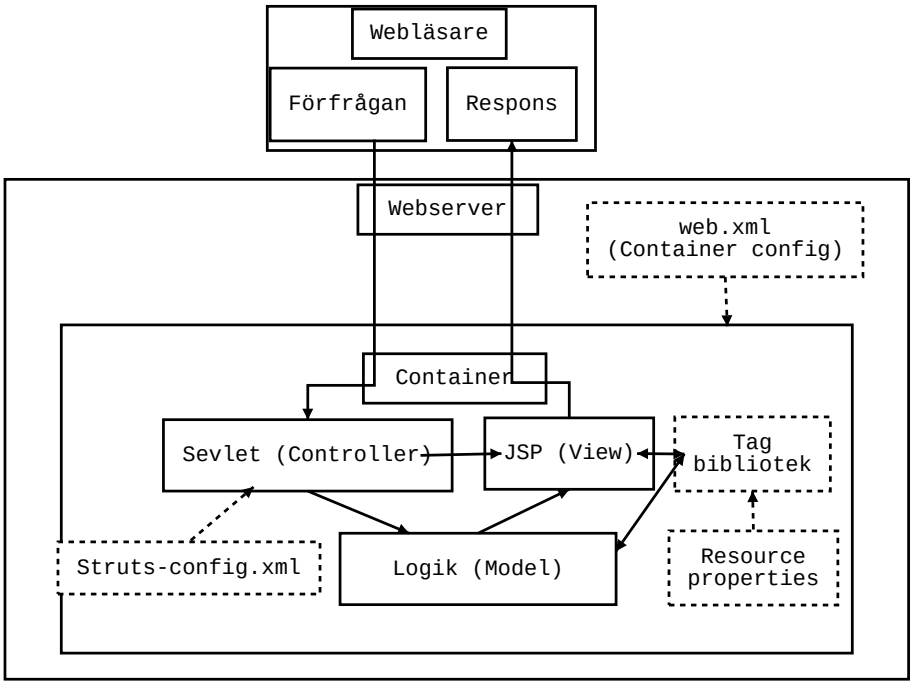
Struts tillhandahålls av *Apache Software Foundation* som en del av dess *Jakarta* projekt. *Jakarta* projektet består av flera lyckade produkter som t ex *Tomcat*, *Ant* och *Velocity*.

Den primära utvecklaren och designern för *Struts* var *Craig R. McClanahan*. Han är idag *Web Layer Architect* för plattformen *Java 2 Enterprise Edition*(J2EE). Det initiala utvecklandet av *Struts* skedde mellan maj 2000 och juni 2001 efter vilket version 1.0 offentliggjordes. Under denna tid bidrog närmare 30 stycken utvecklare med lösningar och tankar och därtill följe tusentals intresserade utvecklingen via *Struts* e-postlistor. Kodbasen för *Struts* hanteras och utvecklas aktivt för närvarande av tretton frivilliga personer som kallar sig för "*Committers*". Som kuriosa kan nämnas att numera laddas *Struts* ner mellan 10000 till 30000 gånger i veckan.[1]

3.2 Struts designmönster

Struts bygger på *Model-View-Controller* (*MVC*) designmönster. Applikationsflödet i MVC förmedlas via en central kontrollerare (*Controller*). Uppgiften för *Controller* är att delegera förfrågningar, t ex *HTTP* förfrågningar till valda hanterare. Hanteraren är bunden till en modell (*Model*). Varje hanterare agerar som en adapter mellan förfrågan och modell. Modellen representerar alternativt kapslar in en applikationslogik eller tillstånd. När hanteraren är klar med sin uppgift skickas ofta applikationsflöden tillbaka via *Controller* till lämplig presentationskomponent (*View*), t ex en *JSP* sida. För att hitta lämplig presentationskomponent används ofta mappningar från antingen en databas eller

en konfigurationsfil. Denna mappning gör att kopplingen mellan *View* och *Model* lätt kan förändras vilket innebär att en applikation enkelt kan skapas och underhållas.[10]
Jämfört med ovan har *Struts* tre stora delar. *Servlet* kontrolleraren (*Controller*) som finns i själva *Struts*, *JSP* sidor som presentations komponent (*View*) samt applikationslogiken (*Model*).



Figur 3.1: Strutsramverket och dess uppbyggnad

3.3 Funktionell översikt

Struts använder sig av *ActionServlet* för att ta emot *HTTP*-förfrågningar och sända *HTTP*-responser. *ActionServlet* kan jämföras med en minimal webbserver som är knuten till en webbserver via en *servletcontainer*. *Servleten* är ett vanligt *Java*objekt med tillgång till ett *API* av *HTTP*-specifika tjänster. När en förfrågan kommer till webbservern skickar servern förfrågan vidare till *servletcontainern*. *Containern* anropar berörd servlet som då erhåller förfrågan. För att hantera förfrågningar finns en annan *Struts* klass som heter *Action*. *ActionServleten* avgör vilken hanterare som ska användas för att behandla förfrågan genom att studera den *URI* som tillhör förfrågan. Om indata ska skickas till en *Action*, t ex uppdatera en databas samlar *ActionServleten* ihop data i en *Java*-böna (*JavaBean*). Till detta finns ytterligare en *Struts* klass, *ActionForm*. *Java*-bönan skapas genom att bönan ärver *ActionForm* klassen. *ActionServleten* avgör vilken *Java*-böna den ska använda på liknande sätt som ovan gällande förfrågan.

Alla *HTTP*-förfrågningar måste svaras med en *HTTP*-respons. För detta tillhandahålls klassen *ActionForward*. *ActionForward* klassen sparar sökvägen till en *JSP* sida under ett logiskt namn. När *Action* objekten har avslutat logiken väljer det en *ActionForward* och returnerar det till *servleten*. *Servleten* använder sig sedan av sökvägen i *ActionForward* objektet och anropar angiven sida och avslutar responsen.

web.xml och *struts-config.xml* är konfigurationsfiler för systemet. *web.xml* är en huvud konfigurationfil för alla *J2EE* applikationer. Den säger hur komponenterna i systemet samarbetar. Filen *web.xml* innehåller en global mappning till *Struts ActionServlet*. En vanlig benämning på en *Struts*-baserad *URL* är "<filnamn>.do". När mappningarna är klara görs alla "<filnamn>.do" *URL* förfrågningar via *ActionServleten*. Utdrag ur en *web.xml* fil följer nedan med förklaring.

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>
        org.apache.struts.action.ActionServlet
    </servlet-class>
<init-param>
    <param-name>application</param-name>
    <param-value>ApplicationResources</param-value>
</init-param>
<init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
.
.
.
</servlet>
```

Ovan är en deklarationen av *ActionServleten*.
Man ser att namnet på *servleten* sätts till *action* och att den är av typen *org.apache.struts.action.ActionServlet*. Även sökvägen "/WEB-INF/struts-config.xml" anges under parameternamnet *config*. Detta för att *ActionServleten* ska veta var denna konfigurationsfil återfinns. I deklarationen ovan kan även debugnivå och andra mappningar anges.

```
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>/do/*</url-pattern>
</servlet-mapping>
```

För att sätta den *Struts*-baserade *URL* beteckningen används mappningen ovan. Den säger att använd *URL* beteckningen ".do" för alla förfrågningar under *ActionServleten* med namnet *action*. Denna beteckning kan väljas utefter önskemål, alltså ".do" är inte något krav.

```
<taglib>
    <taglib-uri>struts/bean</taglib-uri
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

För att som utvecklare ha möjligheten att utnyttja befintliga *tag*-bibliotek i *JSP* koden måste sökvägen till *servleten* anges. Ovan är bara ett exempel då det redan finns flera olika bibliotek. Ett bibliotek som definitivt används är ”struts-html.tld”. Även ska nämnas att utvecklare har möjligheten att konstruera egna bibliotek och använda dessa på samma sätt som befintliga.

Förfrågningar som kommer till *servleten* hanteras sedan utefter mappningar specificerade i filen ”struts-config.xml”. Den konfigurerar servleten med avseende på flödet i systemet. Detta gör att flödet lätt förändras genom denna fil vilket ger ett flexibelt uppdelat system med lösa beroenden. Filen innehåller inte bara mappningar av *URL* utan även referenser till formulärbönor och globala *forwards*. *Forward* är en mappning mellan en parameter och ett anrop till en *action*-klass. När det kommer in en ”<filnamn>.do” *URL* förfrågan till *ActionServleten* hämtas sökvägen till önskad *JSP* sida via mappning i ”struts-config.xml”. Om till exempel förfrågan på ”/login.do” kommer till *ActionServleten* hämtas ”/pages/jsp/login.jsp” ur ”struts-config.xml”. Utdrag ur en ”struts-config.xml” fil med förklaringar följer nedan.

```
<form-beans>
    <form-bean
        name="\employeeForm"\
        type="\app.prog.EmployeeForm"/>
</form-beans>
```

Ovan definieras en form-böna för applikationen. Bönans namn är *employeeForm* och den är av typen *app.prog.EmployeeForm*. Efter en *action* har körts vilken använder denna form-böna kommer applikationen åt den via dess namn. För att uppdatera och hämta informationen knuten till form-bönan används *setter* respektive *getter* metoder. Dessa metoder har dock vissa begränsningar jämfört med traditionell *Java*-programmering. Metoderna kan inte överlagras. Överlagring av en metod i en form-böna resulterar i att *property* variabeln inte återfinns av *Java*. Namn på metoderna som refererar till samma *property* variabel måste ha matchande namn. Anrop till variabeln *name* görs genom metoderna *getName* och *setName*.

```
<action path="/insertEmployee"
    type="\app.prog.InsertEmployeeAction"
    name="\employeeForm"
    scope="\request"
    validate="\false">
<forward
    name="\success"
    path="/confirmation.jsp"/>
\indent</action>
```

Om en förfrågan på ”insertEmployee.do” kommer till *ActionServelet* görs en mappning mot ”struts-config.xml”. Om denna mappning återfinns körs dess *action*. I exemplet ovan mappas ”insertEmployee.do” mot ”path=”insertEmployee”.

Det som *ActionServelet* då får information om är typen på förfrågan *type=”app.prog.InsertEmployeeAction* och att använda form-bönan med namnet *employeeForm*. Det som också ges information om är form-bönans livstid. *scope*-parametern talar om under hur lång tid denna böna ska vara tillgänglig. Det finns tre olika typer av *scope* vilka är *page*, *request* och *application*. *Page* säger att bönan är endast tillgänglig under tiden en specifik sida visas. *Request* är tillgänglig under hela *HTTP*-förfrågan och application är tillgänglig under hela applikationens exekvering. Om vissa fält i form-bönan är av stor vikt och kräver information kan validering av detta ske genom att sätta parametern *validate* till sann. Då anropas automatiskt en metod som heter *validate* i form-bönan. I den metoden kan tester och verifieringar utformas för önskat resultat. Utvecklaren måste dock implementera denna metod själv. Felmeddelanden och varningar vid valideringen görs antingen statiskt kodade eller utnyttjas *application.resources*. Detta är ytterligare en fil hanterad med mappningar. Denna kommer att förklaras nedan. I exemplet ovan är validate satt till falskt vilket gör att *ActionServlet* vet att ingen valideringkontroll ska utföras. Vidare ges informationen om *forward*. Deklarationen säger att om den action som körs returnerar värdet *success* har allt utförts korrekt och då ska sidan ”confirmation.jsp” visas. ActionServlet skickar tillbaka en respons som säger vart webbläsaren kan hämta sidan ”confirmation.jsp”.

```
<message-resources
    parameter="\ApplicationResources"
    null="\false" />
```

ApplicationResources.properties består av mappningar vilka ger ett textmeddelanden för applikationen. Till exempel om ett formulärfält som kräver information inte fått denna information genereras ett felmeddelande. Detta felmeddelande kommer från mappningen i ”ApplicationResources.properties”. Denna hantering av mappningar gör det enkelt att ändra själva meddelandet men även vilket språk som ska användas. Fasta meddelanden blir både svåra att söka efter och uppdatera i ett stort system. Språkmässigt blir det även helt statiskt. Exempel på en mappning följer nedan. ”error.username.whitespace” mappas mot meddelandet som i detta fall formateras med *html*-kod för att uppnå önskat resultat.

```
error.username.whitespace=
    <LI><font color="\red">Användarnamnet får endast innehålla
    tecken av typen a-z och 0-9!</font></LI><br>
```

Eftersom meddelandet läggs in som *html*-kod kan vanlig textmanipulering som exempel sätt färg på texten utföras i denna fil. Exemplet ovan anger information till *servleten* angående ”ApplicationResources.properties”. Om namnet på resursfilen är ”ApplicationResources.properties” sätts *parameter* till *ApplicationResources*. ”null=false” säger att om en mappning som inte finns försöker användas returneras ett felmeddelande liknande ”???global.label.missing???”. Är den satt till true returneras en tom sträng.

Kapitel 4

PostgreSQL

PostgreSQL är en Objekt-relationell databas som har sina rötter i början av relationsdatabasernas grundande. Det är ett *OpenSource*-projekt där människor från hela världen hjälper till med utvecklingen av systemet och betraktas som många som den mest avancerade *OpenSource*-databasen.

4.1 Funktioner

PostgreSQL har många avancerade funktioner som endast återfinns dyra i kommersiella system. Några utmärkande funktioner som systemet stödjer är[4][12]:

- Uppfyller de grundläggande kraven i *ANSI* SQL 92 och 99, men även många delar av den utökade specifikationen samt egna funktioner.
- Stort antal fördefinierade datatyper samt möjlighet för användare att skapa egna.
- Stöd för lagring av *BLOB* (*Binary Large Objects*) som video, ljud, bilder samt text.
- *A.C.I.D* (*Atomicity*, *Consistency*, *Isolation*, *Durability*) stöd för transaktioner. Transaktioner är en följd av kommandon som hör samman, ett exempel är när en valutaöverföring skall ske mellan två konton och det utförs genom att först minskar beloppet för ena kontot följt utav en höjning i det andra.

  Atomicity är att antingen går hela transaktionen igenom eller utförs ingen del av transaktionen, värden som har ändrats återgår till det värde som gällde innan transaktionen.

  Consistency betyder att en transaktion antingen skapar ett nytt godkänt tillstånd med ny data eller om fel uppstår returnerar all data till tillståndet innan transaktionens start.

  Isolation är att alla transaktioner arbetar helt åtskilda, transaktionen jobbar på en egen kopia på databasen tills den är färdig då värdena kopierats till databasen.

  Durability säger att när en transaktion är slutförd är den garanterad att finnas kvar.

- Stöd för andra programmeringsspråk och gränssnitt som *Perl*, *Python*, *Zope*, *PHP*, *TCL/TK*, *ODBC*, *JDBC*, *C/C++*, *Embedded SQL*, *Delphi/Kylix/Pascal*, *ASP* och *Java*.
- Stödjer vyer enligt *ANSI* men även utökningar som möjlighet att uppdatera tabeller från dem.

Att *PostgreSQL* är ett *OpenSource* projekt hindrar inte företag att basera egna produkter på systemet. Det finns företag som erbjuder utbildning och support på *PostgreSQL*, men även de som erbjuder anpassade versioner.

4.2 Historik

PostgreSQL härstammar från en utveckling som startade i början av 1970 talet och bygger på relationsalgebra-modellen som är ett kraftfullt språk för att ställa relationsfrågor. Relationsalgebran började som ett projekt inom *IBM* av en anställd vid namn *Edgar F. Codd*[5]. Han var missnöjd med den då gällande modellen vilken lade bördan att söka information i databasen på användaren. *Codd* ville att bördan att hämta information skulle ligga på databassystemet. Systemet skulle endast känna igen enkla kommandon och det var upp till användaren att konstruera kommandon för att hämta den önskade informationen. Förslaget presenterades som ett antal *IBM* tekniska rapporter och en slutgiltig rapport 1970, "*A Relational Model of Data for Large Shared Data Banks*", där han presenterade ett nytt sätt för åtkomst och organisering av data[3]. Modellen som kallades "relationsmodellen" kan sammanfattas:

> It provides a means of describing data with its natural structure only – that is, without superimposing any additional structure for machine representation poses. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

Det kan översättas till två punkter:

1 Lagrad informationen skall vara fristående från hårdvara och oberoende av lagringssätt.

2 Automatisk navigation via ett högnivåspråk för åtkomst av lagrad information.

1973 startades projektet *Ingres* av två forskare på *UC-Berkeley*, *Michael Stonebraker* och *Eugene Wong*, som blev intresserade av relationsdatabaser. Namnet *Ingres* kommer från att databasen var tänkt att användas för geografisk data, *Interactive graphics and retrival system*. Projektet utvecklades med ekonomiskt stöd från Berkely och tre avdelningar av amerikanska försvaret.

Under 1974, efter att ha visat en tidigt fungerande prototyp och skrivit om koden till en mer hanterlig version spreds koden till en liten grupp människor, både inom och utanför den akademiska världen. *Ingres* blev sedan omskriven ett flertal gånger under de följande åren av dess medlemmar. Detta för att kunna utökas efter kommentarer från användare, medlemmarnas egna erfarenheter

samt nya idéer. Totalt mellan åren 1973 och 1979 var runt 30 personer, dock aldrig mer än 5-6 åt gången, med och utvecklade *Ingres*. Det språk som *Ingres* utvecklades för att ställa relationsfrågor kallades *QUEL*. Projektet *Ingres* avslutades 1982, *Relational Technologies/Ingres Corporation* släppte sedan en utökad version av *Ingres* som en kommersiell produkt.

Stonebraker påbörjas 1985 ett nytt projekt vid namn *Postgres* när *UC-Berkeley* ville utöka relationsmodellen till objekt. Det nya projektet använde en utökad version av språket *QUEL* som de döpte till *PostQUEL*. Den första demonstrationsversionen blev klar 1987 för att visas på 1988 års *ACM-SIGMOD* konferens. Den släpptes även till ett par externa källor 1989. Mellan 1989 och 1991 pågick en aktiv utveckling av systemet där ett regelsystem konstruerades samt utökades med fler finesser. Under åren 1991 till 1993 var den huvudsakliga utvecklingen kring *Postgres* inriktad på portabilitet och tillförlitlighet. Projektet *Postgres* avslutades 1993 efter att antalet användare nästan fördubblats och arbetsbördan att underhålla och hjälpa användare blivit för stor.

Postgres95 skapades 1994 av två före detta *UC-Berkeley* studenter som utökade *Postgres* med *SQL* stöd.

Vid 1996 var det klart att namnet *Postgres95* inte var lyckat varvid det byttes till *PostgreSQL* för att påvisa relationen till *Postgres* och det nya systemet med *SQL*. Utvecklingen av *PostgreSQL* har fortsatt sen dess och är under aktiv utveckling även idag.

4.3 Språk

Under utvecklingen av *PostgreSQL* var det tre språk för åtkomst av information som hade stor inverkan på hur det ser ut idag:

Hierarchical & Network Model

Innan relationsalgebran var de två stora modellerna hierarkiska och nätverks modellen.

Den hierarkiska modellen kom till i slutet av 1960 talet. Den gick ut på att informationen ordnades i en trädstruktur med en till många relationer. Ett exempel är om informationen ska lagras om en förälder till tre barn, då kommer de tre barnen att referera till samma förälder.
Nätverksmodellen skapades 1961 av *CODASYL*, *COnference on DAta SYstems Languages*, som var en frivillig organisation vilken skapades för att hjälpa utvecklingen av ett standardiserat programmeringsspråk. Nätverksmodellen byggde även denna på trädstrukturen men det var tillåtet för en nod att ha flera föräldrar. Därmed stödde modellen många till många relationer men den var både svår att implementera samt underhålla.
För att hämta information från databaserna beskrevs för programmet hur det skulle gå från tabell till tabell för att matcha mot de värden som sökes. Det gjorde att programmen blev väldigt specifika då utvecklaren fick ha i åtanke

hur informationen var lagrad i databasen. Programmet var tvunget att hantera alla felmeddelanden som uppstod, till exempel när inget resultat erhölls.

QUEL

QUEL är det språk som utvecklades av projektet Ingres för att ställa relationsfrågor. Jämfört med *SQL* var *QUEL* bättre genomtänkt eftersom den syntax som används såg lika ut för alla kommandon, i *SQL* varierar syntaxen mellan liknande kommandon. Under början av 1980 talet började fler och fler företag att gå över till *SQL* på grund av större spridning av *SQL* system. I och med att utbredningen av *SQL* ökade medförde det att användningen av *QUEL* minskade vilket har gjort att idag har *QUEL* ytterst liten utbredning. Numera används *QUEL* mest i den akademiska världen.

SQL

SQL har sina grunder från projektet *System/R* som startades av *Donald Chamberlin* i början av 1974, ungefär samtidigt *Ingres* uppkomst[8]. Det första språket som utvecklades var *SEQUEL*, *Structured English Query Language*. *System/R* blev sedan omskrivet från grunden mellan åren 1976 och 1977. Det nya språket blev först döpt till *SEQUEL/2* men blev på grund av rättighetsproblem bytt till *SQL*, *Structured Query Language*. Språket blev sedan standardiserat först av *ANSI*, *The American National Standars Institute*, 1986 följt av *ISO*, *International Standards Organization*, 1987.
1989 gav *ISO* och *ANSI* ut en specifikation över *SQL* kallad *SQL-89*, det har sedan kommit ut utökningar på standarden vid namn *SQL-92*, *SQL-99* och *SQL-2003*. Standarden är viktig för portabiliteten vilket medför att det finns möjlighet att ta ett *SQL* uttryck ifrån en databas och använda den direkt i en annan, dock har de flesta olika databaser egna utökningar som är specifik från produkt till produkt. En stor del av *SQL* språket har inte förändrats mellan uppdateringarna utan det går oftast använda uttryck som är skrivna mot *SQL-89* i nuvarande databaser.

4.4 Exempel

För att jämföra språken följer här ett exempel där vi ställer följande fråga i alla tre språken:
”Vem har konton där saldot är noll och hanterat av en avdelning beläget i Springfield?”

Vi har två tabeller:

- Konton som innehåller värden över innehavare, saldo och ett id över vilket kontor som handhåller konto.
- Kontor innehåller positionen för kontoret samt ett unikt id för varje kontor.

Hierarchical & Network Model

Nedan beskrivs en algoritm av tillvägagångsättet för att utföra frågan i hierarki- och nätverksmodellen:

1 Hämta alla konton med ett nollsaldo

2 För varje kontoresultat

  2.1 Hämta avdelningsidentifieraren för det här kontot

  2.2 Använd identifieraren för att få avdelningsnoteringen

  2.3 Om avdelningsnoteringen värde för position är ”Springfield”:

    2.3.1 Skriv ut ägarvärdet ur kontonoteringen.

QUEL

För att utföra samma sak i *QUEL* ställs följande fråga:

rang of a **is** Account
rang of b **is** Branch
retrieve a.owner
where
a.balance = 0 **and**
b.location = ”Springfield” **and**
a.branch_id = b.branch_id

SQL

Och slutligen motsvarande fråga i *SQL*:

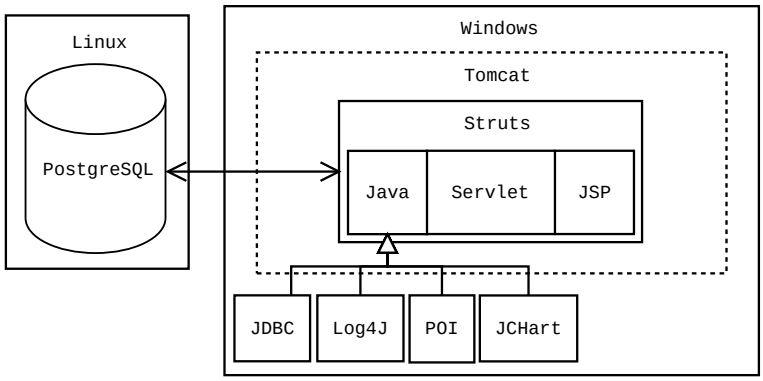
SELECT Account.owner
FROM Account, Branch
WHERE
Account.balance = 0 **AND**
Branch.location = ”Springfield” **AND**
Account.branch_id = Branch.branch_id;

Kapitel 5

Systembeskrivning

Det system som byggts upp är komplext och med många delar som skall samverka vilket medför att det är svårt att få en överblick över systemet. Här följer en beskrivning av hur delarna i systemet interagerar

5.1 Systemets uppbyggnad



Figur 5.1: Schematisk beskrivning av systemets uppbyggnad

Figur 5.1 beskriver hur systemets alla delar hänger ihop. För det första använder systemet två plattformar, *Linux* för databasen och *Windows* för *Struts* respektive *Tomcat*. Databasen är av typen *PostgreSQL* vilken är snabb, effektiv och avancerad. *Linux* valdes som plattform för databasen därför att i dagsläget finns inte *PostgreSQL* som stabil *Windows* version. *Tomcat* som är systemets webbserver innehåller alla nödvändiga konfigurationsfiler för *Struts* samt alla *Java*-klass filer. Detta visas i figuren genom den streckade rutan som innesluter dessa delar. Det som dock inte finns med i *Tomcat* är *Java*-kod filerna samt biblioteken *JDBC*, *Log4J*, *POI* och *JChart*. *JDBC* biblioteket innehåller ett *Java* gränssnitt med all nödvändig funktionalitet för att kommunicera och hantera databasen. *Log4J* är ett verktyg för att hantera felmeddelande och loggning av händelser under körning *POI* biblioteket används av systemet för att

manipulera filer av *Excel*-format. *JChart*-biblioteket tillhandahåller olika typer av diagram. För systemet har linjediagram utnyttjats. Mer gällande dessa finns under rubriken 2.2 Plattformar och utvecklingsmiljöer.

5.2 Algoritmbeskrivningar

För att förstå hur systemet fungerar beskriver vi här de steg som behöver utföras av systemet i de mer avancerade fallen.

Inloggning

För att kunna logga in en användare via webbgränssnittet måste systemet utföra dessa steg.

1 Användaren skriver in användarnamn och lösenord.

2 Lösenordet krypteras med en envägs krypteringsalgoritm (*SHA-1*).

3 Användarnamn och krypterade lösenordet skickas till databasen.

4 Databasen jämför användarnamnet och det krypterade lösenordet mot befintliga.

  4.1 Om det ej matchar skicka tillbaka ett meddelande om misslyckat inloggningsförsök.

5 Loginfunktionen skickar tillbaka ett nytt loginnamn, lösenord och en identifierare av vilken typ av användare det är.

6 Skapa en uppkoppling emot databasen via en databaspool genom att skicka login och lösenord till för att validera rättighet till datapoolen.

  6.1 Om inte användarnamnet finns i datapoolen koppla upp emot databasen med användarnamnet och lösenordet.

    6.1.1 Om inloggningen emot databasen misslyckas, avbryt och rapportera felet.

  6.2 hämta en godkänd referens till databaspoolen.

7 Skapa en ny session användare med rättigheten från användarinloggningen.

Provinskickning

När ett prov skall skickas via webbgränssnittet till databasen måste alla dessa steg utföras för att lyckas.

1 Presentera användaren ett webbgränssnitt för att välja ut dom två filerna som hör till provet som skall laddas upp.

2 Hämta ut filnamnen och validera att det är rätt typ på filerna.

  2.1 Om misslyckande, avbryt och presentera en felmeddelande.

3 Lägg dom två filerna på ett separat utrymme i databasen inför en felrapport om uppladdningen går fel.

4 Försök tolka informationen i dokumentet.

  4.1 Lägg till felmeddelandet i en tabell som länkar till filerna som sparas om nått skulle gå fel.

  4.2 Om misslyckande, avbryt och presentera en felmeddelande.

5 Presentera det tolkade informationen för användaren för godkännande, om dokumentet sedan tidigare finns i databasen presentera att det kommer bli en ny revision av det dokumentet.

  5.1 Om användaren avbryter, ta bort den sparande binära versionen och gå till steg *1*.

6 Börja databastransaktionen, vid fel gå till *7*

  6.1 Plocka ut informationen om den användare som försökte sätta in provet.

  6.2 Lägg till information som är relaterad till provet i angränsande tabeller.

  6.3 Lägg till huvudinformationen om provet.

  6.4 Om provet är ett beläggningsprov, beräkna vilken region det tillhör.

  6.5 Om kornkurva existerar, lägg in dessa värden och knyt dom till huvudprovet.

  6.6 Lägg till alla FAS-värden och knyt dom till huvudprovet.

  6.7 Lägg till *Excel* och *PDF* filerna.

7 Om fel uppstår i transaktionen.

  7.1 Backa tillbaka alla ändringar till innan början på databas transaktionen.

  7.2 Lägg till felmeddelandet i en tabell som länkar till filerna som sparades inför en felrapport.

  7.3 Avbryt och presentera en felmeddelande.

8 Markera att informationsuppladdningen lyckats.

9 Ta bort den binära versionen av proverna som sparades inför en felrapportering.

5.3 Databas

Databasen är en central del i systemet och innehåller information som är nödvändig för att systemet skall fungera. Informationen i databasen är lagrad med grundtanken att informationen inte skall vara duplicerad. För att underlätta hanteringen har information klassificerats efter vilken typ av information det är. Klassificeringen skiljer informationen genom att olika namnområden skapas. De namnområden som existerar är:

compiles: Här finns all data om sammanställningar

debug: Informationen här används endast vid felsökning av misslyckade uppladdning av prover och recept. *Excel* filen som skall parsas läggs först upp här, om utläsningen misslyckas lagras ett felmeddelande knytet till den binära *Excel* filen, om utläsningen lyckades tas informationen bort.

division: Här finns information som berör fabriker, laboratorier och regioner.

messages: Globala meddelanden som skickas till användaren lagras här.

public: Innehåller endast login funktionen.

recipes: Här har vi all information rörande recept.

samples: Provdatat lagras i det här området.

tolerances: Toleransvärden lagras i detta område.

users: Här har vi alla användar och grupp information.

5.4 Begränsningar

Systemet har ett antal begränsningar som beror på tidsbrist, planeringsfel och begränsa arbetsbördan. De tre olika områden där begränsningarna existerar är:

Felmeddelanden

De felmeddelanden som systemet presenterar vid fel är ytterst begränsade, det gör att det är svårt för en användare att ta reda på vad som är fel. De begränsningar som existerar är:

Inloggning: Här presenteras visserligen information om fel lösenord angivits, men inte om det uppstår andra problem som till exempel att databasuppkopplingen inte fungerar.

Timeout: Om en användare inte använder systemet inom en viss tidsintervall loggas användaren automatiskt ut, informationen för den är bristfällig.

Obehörig: Vid försök att komma åt en sida utan rätt rättighet presenteras en sida utan länkar för att gå tillbaka.

Uppladdning: Vid misslyckade insättning av prov och recept i databas presenteras ingen information om vilket värde som var fel. Det är i vår mening den största bristen i systemet.

Oföränderliga värden

För att underlätta vårt arbete har vi varit tvungna bygga in viss information i systemet utan möjlighet att enkelt kunna ändra. Det är värden som är skriven i *Java* koden eller i databasen utan möjlighet att enkelt kunna ändra.

- Namn och plats för fabriker och laboratorier.
- De gränsvärden som existerar.
- Vilka olika grupper av användare som existerar.
- Användarnamn och lösenord som används för inloggning av en användare i en grupp.
- Vilka olika belägningsregioner som existerar.
- Vilka typer av prover som används för att göra en sammanställning.

Prov och Recept

Utläsningen av information ifrån ett prov eller ett recept är starkt beroende av formateringen av *Excel*-filen. För att en lyckad inläsning skall ske måste ett stort antal krav uppfyllas. Det mest kritiska är de fält med värden som måste motsvara ett existerande i databasen. Ett exempel är fabriken ”Six-pack” som måste motsvara det värde som existerar i databasen, det medför att alla måste använda samma notation av namnet.

5.5 Användare

För användandet av systemet finns fyra olika användarkonton. Dessa fyra är administratör, laboratorieanvändare, administratör av beläggningskonton och vanlig användare. Vanliga användare delas upp i två grupper vilka är beläggning och konstruktion. Skillnaden är att beläggning har hårdare sekretesskrav på grund av ackreditering. Nedan följer en resultat demonstration på specifika delar för de fyra typerna av användare.

Administratör

Administratören har alla rättigheter när det gäller hanteringen av användarkonton. Denne kan lägga till, ta bort och ändra information gällande nya och befintliga användare. Det som dock inte kan göras utifrån denna användare är manipulera och hantera provresultat. Skulle administratören för systemet också vara i behov att utföra sådant måste ett speciellt användarkonto av önskad typ skapas. Administratören kan även skapa nyheter som når alla användare av systemet.

Vanliga användare

Vanliga användare består även de av ett antal grupper. Det som är gemensamt för alla är att de kan skapa sammanställningar inom den region de tillhör.

Global innebär att användaren har rättigheter att se alla provresultat. En global användare måste ange från vilken grupp av vanliga användare en sammanställning skall göras. Annars kan skapa sammanställningar från flera olika grupper skapas vilket inte är tillåtet.

Konstruktion är indelad i två kategorier, global konstruktion och konstruktion. Användare av typen global konstruktion är knyten till alla asfaltfabriker. Detta innebär att alla fabrikers provresultat kommer att vara av intresse för användaren. Konstruktionanvändaren är däremot knuten mot en eller flera fabriker som är av större intresse jämfört med övriga. Konstruktionanvändaren har skapats för att göra tillgängligheten till intressanta provresultat enklare. Det ska tilläggas att konstruktionsanvändaren har dock rättighet att se provresultat från samtliga fabriker. Båda kategorierna har rättighet att skapa sammanställningar som sträcker sig över samtliga fabriker.

Beläggning består också av två kategorier. Dessa är global beläggning och beläggning. Avdelningen beläggning är delad i tre regioner bestående av län. Global beläggningsanvändare har rättighet att se samtliga beläggningsprover oavsett region jämfört med beläggningsanvändaren som endast får se sin regions provresultat En beläggningsanvändare har ett antal kontonummer knutna till sig, dessa styr vilka arbetsobjekt den användaren är knuten till. Kontonumren används för att bestämma vilken region ett prov tillhör.

Laboratorie

Denna användare sköter hantering av provresultat. Det betyder uppladdning av provresultat och provrecept till databasen. Ett användarkonto av typen global är knutet till varje laboratorieanvändare. Även denne användare har rättighet att skapa nyheter som når alla användare.

Beläggningskonto-administratör

Den enda avsikt med detta användarkonto är att lägg till och ta bort kontonummer från användare av typen beläggningsanvändare.

Kapitel 6

Diskussion

För att sammanfatta examensarbetet kommer nedan en diskussion om de punkter som varit viktig.

6.1 Systemdesign

Ett webbaserat system kopplat till en databas ger oftast den optimala lösningen för ett informationssystem. En av fördelarna är att ingen speciell programvara krävs av systemets användare, systemet blir även ofta mer intuitivt samt att en central kontroll av systemet uppnås. Men det medför även nackdelar som att systemets utseendet kan variera kraftigt mellan olika webbläsare. Det är även en säkerhetsrisk eftersom all information finns lagrad på ett ställe.

6.2 Verktyg

Intrycket vi har fått av *Java* är att det är ett utmärkt verktyg använda för att bygga webbtjänster, den webbaserade tekniken har dock en hög inlärningströskel vilket avskräcker många. Det som är riktigt bra med *Java* är det stora utbudet av verktyg som underlättar arbetet, ett exempel är *POI* som används för att läsa ut information från *Excel* dokument. Det intryck vi fick av *Struts*var att det även här har en hög inlärningströskel samt att det existerar några begränsningar som är svåra att överkomma. Databasen *PostgreSQL* är riktigt smidig att arbeta mot, dock är det ett minus att det inte finns någon *Windows* version.

6.3 Resultat

För att förstå den förbättring som systemet är tänkte att ge så beskriver vi den skillnad i handhavande som drabbar personalen av systemet. För laboratoriepersonalen blir insättningen av prover ingen större förändring. Det extra steg de får utföra är att via sitt befintliga program generera en *Excel*-fil som tillsammans med ett *PDF*-dokument laddas upp via ett webbgränssnitt till *NCC*. Direkt när provet inlagt i systemet kan alla med rätt behörighet se den information som finns i provet.

För personalen på *NCC* blir information om prover och sammanställningar tillgänglig direkt utan att behöva begära *PDF*-dokumentet eller papperskopian av provresultatet. För en sammanställare av prover blir skillnaden stor. Eftersom informationen inte behöver kopieras från en papperskopia till ett sammanställningsdokument samt behöver inte sammanställaren räkna ut gränsvärdekurvor. Systemet medför att det enkelt går att välja ut de prover som skall sammanställas och automatiskt generera sammanställningar av proverna. Berörda användare kan sedan gå in och undersöka sammanställningarna som lagrats i systemet.

Systemet har dock ett antal begränsningar som försvårar arbetet. Största begränsningen är bristfälliga felmeddelanden ifall en insättningen av ett nytt prov eller recept misslyckas.

6.4 Testkörning

Systemet är tänkt att vara under testkörning inom *NCC*s intranät för utvärdering. Dock existerar ej den modul till det befintliga laboratoriesystemet som krävs för att överföra informationen mellan systemen.

6.5 Vidareutveckling

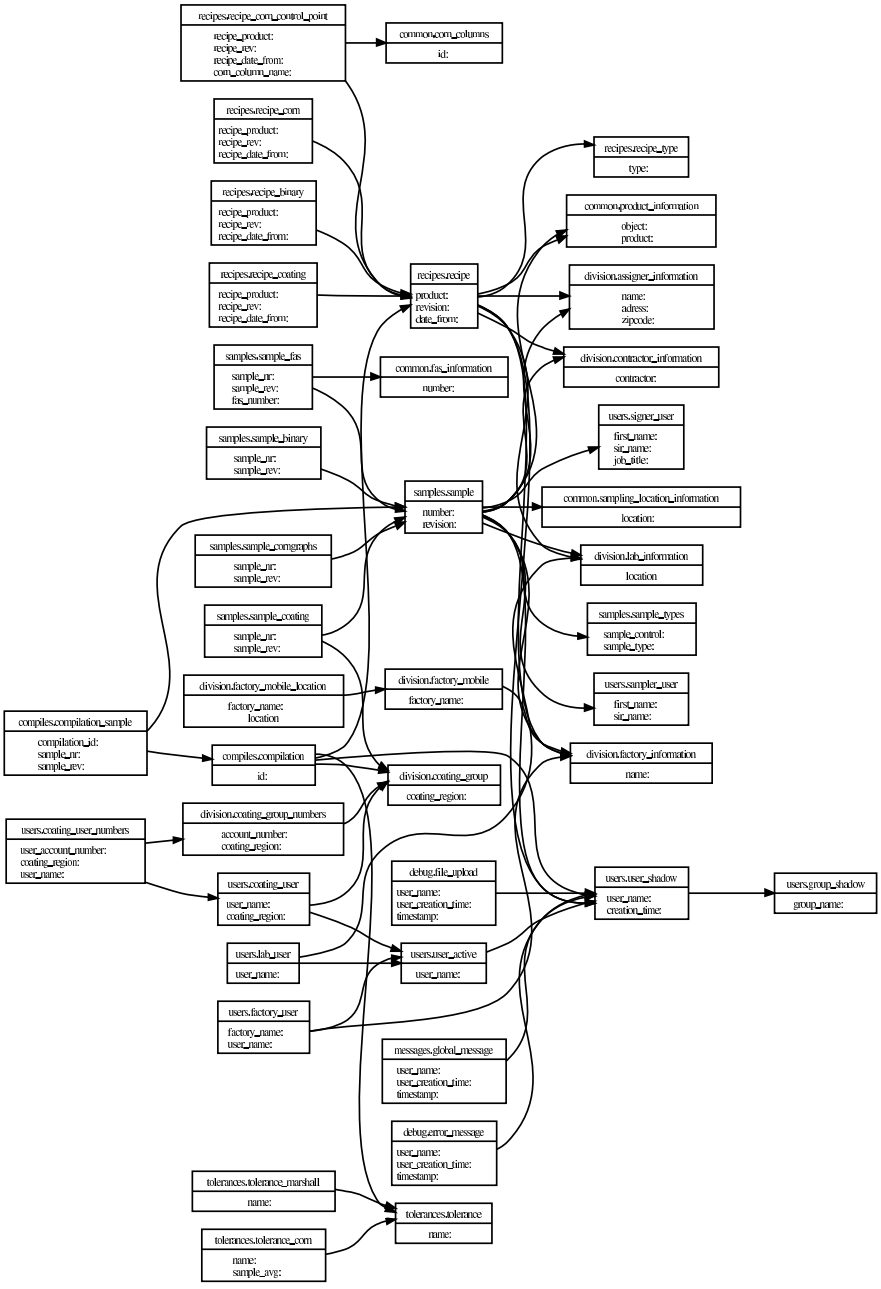
När systemet har testkörts inom *NCC* är det tänkt att ett nytt system baserat på prototypen skall utvecklas. Det nya systemet skall sedan användas inom hela *NCC* för att hantera provresultat och sammanställningar.

Bilaga A

Databastabeller

I figur A.1 visas de relationer tabeller har mellan varandra, antingen som en del i den primära nyckeln eller som ett värde som refererar till nyckeln i den andra tabellen. Eftersom det är svårt att urskilja vissa beroendet följer här en lista med relationerna.

- **samples.sample** beror på:
  - *common.product_information*
  - *common.sampling_location_information*
  - *division.assigner_information*
  - *division.contractor_information*
  - *division.factory_information*
  - *division.factory_mobile_location*
  - *division.lab_information*
  - *samples.sample_types*
  - *users.signer_user*
  - *users.sampler_user*
  - *users.user_shadow*
- **recipes.recipe** beror på:
  - *common.product_information*
  - *division.assigner_information*
  - *division.contractor_information*
  - *division.factory_information*
  - *division.factory_mobile_location*
  - *division.lab_information*
  - *recipes.recipe_type*
  - *users.user_shadow*
- **division.factory_mobile** beror endast på:
  - *division.factory_mobile_location*



Figur A.1: Relationsberoenden

- **compiles.compilation** beror på:
  - *tolerances.tolerance*
  - *recipes.recipe*
  - *users.user_shadow*

- **users.factory_user** beror på:
  - *division.factory_information*
  - *users.user_active*
- **users.lab_user** beror på:
  - *division.lab_information*
  - *users.user_active*

För att förstå hur databasen fungerar följer här en beskrivning av vilken funktion varje tabell uppfyller. Tabellerna är sorterade i olika kategorier beroende på vilken typ av data de innehåller.

A.1 Publika (Public)

Här finns några saker som inte hör hemma i några av de andra tabellerna.

Vyer

return_?_text är ett antal tomma vyer, där ? är ett av talen 2,3 eller 6 och säger hur många textfält bred tabellen är, den används endast för att kunna bestämma vilken typ av data som returnerats från vissa funktioner.

Funktioner

login(text, bytea) funktion är att logga in en användare genom att undersöka om det angivna lösenordet matchar mot en *SHA-1* checksumma av lösenordet. Login funktionen används av webservern när en användare försöker logga in, om användarnamn och lösenord stämmer returnerar funktionen all information som behövs för att skapa en ny koppling mot databasen.

A.2 Globala (Common)

De värden som finns här är konstanta värden som används både av ett recept och ett prov.

Tabeller

corn_columns säger vilka mätpunkter för kornkurvan som finns.

fas_information översätter ifrån fasnummer till ett namn.

product_information vilka produktnamn finns.

sampling_location_information innehåller alla provtagnings ytor.

A.3 Användare (Users)

Här finns användar relaterad information.

Tabeller

group_shadow innehåller lösenordet för de olika grupperna, den informationen returnerats utav inloggnings funktionen för att logga in en validerad användare.

user_shadow innehåller information om en användare, tabellen används för att kunna referera till vem som skapade en specifik sak, informationen här går inte ta bort utan bara skapa ny som är mer aktuell.

user_active är en tabell över vilka användare som är aktiva. Eftersom alla användarnamn som skapats måste sparas eftersom de är knutna till ett prov, det medför att systemet måste kunna bestämma vilka användare och användarinformation som är aktiv just nu.

lab_user för användare inom lab säger den här tabellen vilket lab som han jobbar på.

factory_user säger vilka fabriker som den här fabriksanvändaren är intresserad av.

coating_user knyter en beläggningsanvändare till en viss region.

coating_user_numbers knyter fakturanummer till beläggningsanvändare.

signer_user lagrar informationen om signerare av ett prov.

Vyer

all_users lista med information om på alla användare som existerat i systemet.

group_list listar alla grupper som existerar på systemet.

group_login används av loginfunktionen för att beskriva det resultat den returnerar.

user_information lista med information om användare som är aktiva.

user_list lista med information om användare som är aktiva, visar endast kort information om användaren.

Funktioner

change_user_information(name, text, text, text, text, text, text, text) används av admin för att ändra informationen för en användare.

get_user_name(name) konverterar ifrån ett användarnamn okänsligt för stora och små bokstäver till det motsvarande användarnamnet som databasen använder.

my_upper(name) konverterar ett namn till motsvarande med stora bokstäver.

add_user_coating_number(name, bigint) kopplar ett nytt beläggningskonto till en beläggningsanvändare. Om kontot redan existerar i en annan region men det inte är nån användare kopplat till det flyttas kontot över till den nya regionen. Flera användare kan vara kopplad till samma konto om de tillhör samma region, om regionen inte stämmer returneras namnet på en beläggningsanvändare som redan är kopplat till kontot.

change_user_group(name, text) byter vilken grupp en region tillhör, alla gruppspecifika kopplingar tas bort innan bytet görs.

delete_user(name) tar bort en användare ifrån listan över aktiva konton.

delete_user_coating_number(name, bigint) tar bort konton kopplat till en viss användare.

change_user_password(name, bytea) ändrar lösenordet för en viss användare, det nya lösenordet består av en *SHA-1* checksumma av användarlösenordet.

A.4 Meddelanden (Messages)

Innehåller nyheter som har lagts ut av behöriga användare.

Tabeller

global_message innehåller meddelanden som kommer att visas för alla användare när dom loggar in.

Vyer

all_global_messages globala meddelanden där användarnamnet kopplats mot det riktiga namnet.

Funktioner

message_count() returnerar antalet globala meddelanden som finns i systemet.

A.5 Arbetsenhet (Division)

Information om alla arbetsenheter och områden.

Tabeller

assigner_information lagrar information om beställare.

coating_group säger vilka olika regioner som finns.

coating_group_numbers här finns alla fakturanummer, ett fakturanummer kan endast existera i en region.

contractor_information lagrar informationen om vem det är som utför arbetet.

factory_information lagrar information om alla verk.

factory_mobile säger vilka verk som är mobila.

factory_mobile_location information om de platser mobila verk varit på.

lab_information information om alla laboratorier.

A.6 Prover (Sample)

Information gällande alla prover. Ordet giltig används här för att beskriva att en funktion först försöker matcha värdet mot ett existerande, om det ej finns skapas ett nytt värde i databasen och skickas tillbaka.

Tabeller

sample innehåller obligatorisk information om ett prov.

sample_binary innehåller binärversionen av ett prov (excel och pdf filen).

sample_coating säger vilken region som ett beläggningsprov tillhör.

sample_corngraphs lagrar kornkurvan.

sample_fas lagrar ett fasvärde.

sample_types berättar vilka olika typer av prover som finns (i vår prototyp endast *kvalitetskontroll* och *produktionskontroll*).

Vyer

production visar alla produktions prover.

quality visar alla regionsprover.

quality_noregion visar regionsprover som inte tillhör någon region.

Funktioner

calculate_group(integer, smallint) beräknar vilken region ett kvalitetsprov tillhör och lägger till det i den regionen. Funktionen måste anropas innan den binära provdatat är inlagt.

get_coating_region(integer, smallint) returnerar kodbeteckningen för vilken region ett prov tillhör.

get_coating_region_text(integer, smallint) returnerar text som beskriver vilken region ett visst prov tillhör.

get_sample_companies(text, text, text, text, text, text) plockar ut en giltig version av fabrik, leverantör och beställare. Ersättas av ”get_valid_assigner”, ”get_valid_contractor” och ”get_valid_factory”.

get_sample_object(text, text, text, text) returnerar giltig objekt data.

get_sample_type_specific_text(integer, smallint) returnerar text som beskriver ett prov. För beläggningprover returneras vilken region provet tillhör och för fabriksprover returneras fabriksnamnet.

get_sample_user_info(text, text, text, text, text, text) returnerar giltiga värden gällande provtagare, signerare och signeringsplats.

get_valid_assigner(text, text, text) returnerar en giltigt beställare.

get_valid_contractor(text) returnerar ett giltigt företag.

get_valid_factory(text, text) returnerar en giltig fabrik.

get_valid_product(text, text) returnerar en giltig produkttyp.

get_valid_sample_type(text, text, text, text) returnerar en giltig provtyp.

insert_fas(integer, smallint, text, numeric, numeric, text, text) lägger till ett nytt fasvärde till ett prov, skapar ny fasinformation om ett fasnummer om det inte existerar sen tidigare.

A.7 Recept (Recipes)

Recept informationen lagras inom detta område.

Tabeller

recipe fast information om ett recept.

recipe_binary innehåller binärversionen av ett recept (excel och pdf filen).

recipe_coating lagrar informationen om receptvärdena.

recipe_corn innehåller värdena för alla mätpunkter.

recipe_corn_control_point innehåller vilka mätpunkter som är kontrollpunkter.

recipe_type säger vilka typer av recept som finns.

tolerance innehåller dom fasta toleranserna.

tolerance_corn säger hur mycket som värdena i korkurvan får variera.

tolerance_marshall säger mål värden för marshall.

Vyer

recipe_active berättar vilka recept som är aktivt just nu, nyaste versionen.

Funktioner

get_valid_recipe_type(text) returnerar valid recept typ, sätter in i databasen om ingen redan existerar.

A.8 Felsökning (Debug)

Funktionen för det här området är att underlätta felsökningen av databasen, används främst vid uppläggning av prover och recept.

Tabeller

file_upload lagrar ett prov eller recept innan systemet försökt tolka informationen i det.

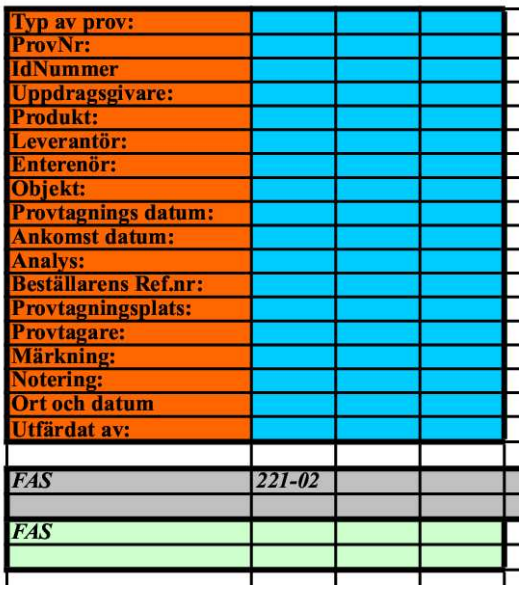
error_message lagrar allvarliga felmeddelanden som uppstår.

Bilaga B

Provresultat & Recept

För att överföra information rörande prov och recept från laboratoriesystemet till det nya systemet på *NCC* valde vi att använda ett *Excel* dokument. Utseendet på dokumentet följer en standardmall för att möjliggör utläsningen av information från det.

B.1 Enskilt prov



Figur B.1: Enskilt prov

Figur B.1 visar en mall för hur fälten är ordnade i ett prov, de fasta fälten utgör den övre delen av dokumentet och är ordnad i följande ordning:

Typ av prov: Är vilket typ av prov det är.

ProvNr: Provnummer för det här provet.

IdNummer: Kontonummer knutet till ett visst objekt.

Uppdragsgivare: Namn och adress till uppdragsgivaren, namn i fält ett och adress i fälten två och tre.

Produkt: Produktnamnet för det här provet.

Leverantör: Namn på fabriken, på mobila laboratorier används fält två för positionen på fabriken.

Entreprenör: Namn på entreprenören.

Objekt: Identifikationsnamn på objektet.

Provtagnings datum: Datum för provtagning.

Ankomst datum: Datum för ankomst till lab.

Analys: Fält 1 anger analysstart och fält 2 anger när analysen var klar.

Beställarens Ref.nr: Referensnummer för att identifiera beställaren.

Provtagningsplats: Identifierare för platsen där provtagningen gjordes.

Provtagare: Namn på provtagare.

Märkning: Om provet har nån speciell märkning.

Notering: Noteringar gällande provet.

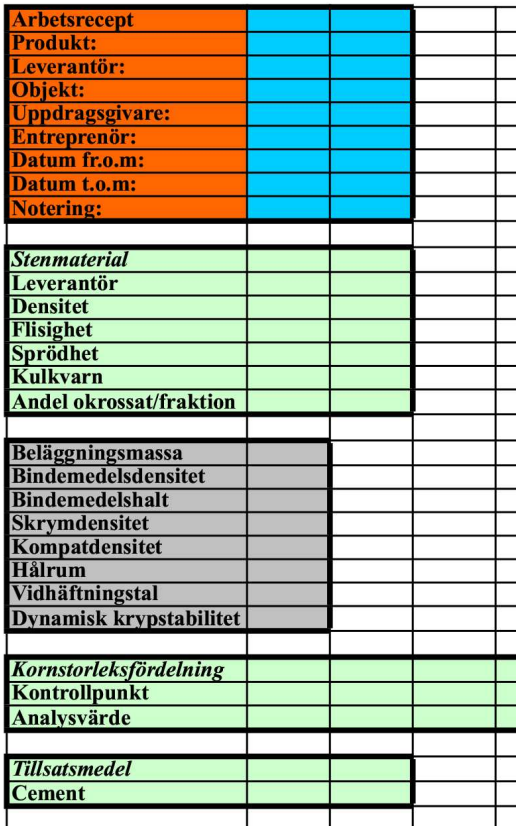
Ort och datum: Vilket datum som provet är utfärdat.

Utfärdat av: Förnamn, efternamn och titel på utfärdaren.

Efter det fasta fälten kommer den del som beskriver fasvärdena, de är uppbyggda av två rader som hör ihop. Den första raden har *FAS* i första fältet följt av en identifierare i nästa kolumn. I den tredje finns det namn som är associerat med identifieraren och slutligen i kolumn fyra är den storhet som värdena är angivna i. Den andra raden innehåller de värden som är uppmätta, kolumn 1 innehåller värdet och kolumn 2 innehåller variation om det existerar ett sådant värde.

Det enda värde som avviker från standardmallen är kornkurvan med fasvärdet *221-02*. Det som skiljer sig på första raden är att enheten för värdena (kolumn 3) inte är angiven. På andra raden är värdena uppstaplade efter siktvärdena. Den första kolumnen motsvarar procenten av sten som har en kornstorlek under $0.016mm$. Kolumnerna som följer beskriver procentsatsen av kornstorlekarna i stigande ordning, när mängden når 100 procent avbryts utskrifterna.

B.2 Recept



Figur B.2: Recept

Ett exempel på hur fälten i dokumentet ser ut kan ses i Figur B.2, där de fasta värdena är:

Arbetsrecept: Vilken typ av recept är det.

Produkt: Produktnamnet för det här receptet.

Leverantör: Namn på fabriken, fält 2 beskriver position för mobila fabriker.

Objekt: Identifikationsnamn på objektet.

Uppdragsgivare: Namn och adress till uppdragsgivaren, namn i fält 1 och adress i fälten 2 och 3.

Entreprenör: Namn på entreprenören.

Datum fr.o.m: Från vilket datum är provet giltigt

Datum t.o.m: Till vilket datum är provet giltigt, används dock ej av systemet.

Notering: Noteringar gällande receptet.

Det som följer sedan är olika block av värden. Första raden i blocket beskriver vilken typ av block det är. Efter det så följer en rad med en titel och ett antal värden kopplat till den titeln.

Eftersom utseendet på värdena i receptet varierade kraftigt var vi tvungen att lösa det på ett annorlunda sätt jämfört med det enskilda provet. Lösningen blev att systemet endast hämtar värden som används för att generera grafer och för att göra sammanställning av ett objekt.

Litteraturförteckning

[1] A Brief History of Struts. Webpage, 1 October 2003. http://jakarta.apache.org/struts/volunteers.html.

[2] The Apache Ant Project. Webpage, 11 June 2004. http://ant.apache.org.

[3] Edgar Frank Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[4] Command Prompt Inc, editor. *Practical PostgreSQL*, chapter 3. O'Reilly & Associates, January 2002.

[5] National Research Council. *Funding a Revolution: Government Support for Computing Research*, chapter 6. National Academy Press, January 1999.

[6] Java Technology. Webpage, 11 June 2004. http://java.sun.com.

[7] jCharts - Krysalis community project. Webpage, 11 June 2004. http://jcharts.sourceforge.net.

[8] John Kauffman. *Beginning SQL Programming*, chapter 2. Wrox Press Inc, March 2001.

[9] Welcome to Logging Services Project @ Apache. Webpage, 11 June 2004. http://logging.apache.org.

[10] Java BluePrints, Model-View-Controller. Webpage, 18 May 2004. http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[11] Jakarta POI - Java API To Access Microsoft Format Files. Webpage, 11 June 2004. http://jakarta.apache.org/poi/index.html.

[12] PostgreSQL 7.4 Documentation. Webpage, 22 January 2004. http://www.postgresql.org/docs/7.4/static/history.html.

[13] PostgreSQL. Webpage, 11 June 2004. http://www.postgresql.org/.

[14] Struts. Webpage, 11 June 2004. http://jakarta.apache.org/-struts/index.html.

[15] Ted N. Husted, Cedric Dumoulin, George Franciscus, David Winterfeldt. *Struts in Action*, chapter 1. Manning Publication Co., November 2002.

[16] Apache Tomcat. Webpage, 11 June 2004. http://jakarta.apache.org/-tomcat/index.html.