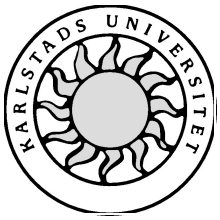


Datavetenskap

Liselotte Thorén

Ulf Nilsson

Design och implementation av en webbaserad databas

Examensarbete, C-nivå

2000:28

Design och implementation av en webbaserad databas

Liselotte Thorén

Ulf Nilsson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Liselotte Thorén

Ulf Nilsson

Godkänd: 2000-05-30

Handledare: Johan Garcia

Examinator: Stefan Lindskog

Sammanfattning

Vi har arbetat på uppdrag av forskarna i PRTP-gruppen vid institutionen för datavetenskap vid Karlstads Universitet. Målet med vårt arbete var att skapa ett system för förenklad hantering av länkar till forskningsrelaterat material och information om vetenskapliga konferenser. Systemet skall finnas tillgängligt för allmänheten på forskargruppens hemsida. För forskarna skall även ett administrativt gränssnitt finnas tillgängligt. Vi är i skrivande stund inte klara med arbetet, resultatet av vårt arbete så här långt är ett fungerande gränssnitt för administration av systemet. Vi har dock för avsikt att slutföra arbetet efter att denna rapport är klar.

Design and Implementation of a Web based Database

Abstract

We have worked on assignment of the scientists in the PRTP reasearch group at the department of computer science at Karlstad University. The goal of our work was to create a system for managing URLs to related research material and information regarding scientific conferences. The system will be available for public use from the research group homepage. For the scientists there will also be an administrative interface available. When this is written, the work is still not finished. The result so far is a working interface for administration of the system. Our intention is to finish the work after this report has been written.

Innehållsförteckning

1 Introduktion 1

2 Inledning 1

2.1 Bakgrund 1

2.2 Syfte 1

3 Förutsättningar och krav 2

3.1 Förutsättningar 2

3.2 Krav 2

3.2.1 Information som systemet skall innehålla
3.2.2 Funktionella krav
3.2.3 Gränssnittskrav
3.2.4 Utbyggnad och flexibilitet

3.3 Ändringar och tillägg av krav 3

4 Bakgrunds fakta 4

4.1 Resonemang 4

4.2 Javaservlets 5

4.2.1 Alternativ till Java servlets
4.2.2 Fördelar med Java servlets

4.3 MySQL 8

4.3.1 Altemativ till mySQL

4.4 JDBC 9

4.4.1 Jämförelse av JDBC och ODBC

5 Beskrivning av konstruktionslösningen 11

5.1 Prototyp 11

5.2 Databasen 11

5.3 Servlets 12

6 Implementation och testning 13

6.1 Implementation och testning av databasen 13

6.1.1 Exempel SQL-frågor

6.2 Implementation och testning av servlets 16

6.2.1 Exempelservlet

6.3 Administrationsgränssnittet 21
6.3.1 AddUrl
6.3.2 DeleteUrl
6.3.3 EditUrl
6.3.4 AddConference
6.3.5 DeleteConference
6.3.6 EditConference
6.3.7 AddSubject
6.4 Använda rgränssnittet 25
6.4.1 Conferences
6.4.2 SearchUrl
6.4.3 BrowseUrl
6.4.4 SearchConference

7 Projektberättelse 26

8 Slutsatser 28

Referenser 29

A Relationsdatamodell 30
A.1 Mappning av E/R-modellen till relationsdatamodellen. 30
A.2 Beskrivning av de olika relationerna 30

B Källkod för servleten addUrl.java 37

Figurförteckning

Figur 4.1: Servletens livscykel. 6
Figur 4.2: JDBC two-tier model (hämtat ur [2]) 10
Figur 5.1: E/R-modell 12
Figur 6.1: Grafiskt gränssnitt. 13
Figur 6.2: Gränssnitt för att skapa tabell. 14
Figur 6.3: Exempel formulär. 18
Figur 6.4: Exempel förhandsvisning. 19
Figur 6.5: Prototyp för listning av konferenser. 26

1 Introduktion

Uppgiften för vårt examensarbete var att konstruera ett webbaserat databassystem och det här dokumentet beskriver hur utvecklingsarbetet har fortlöpt. Vi börjar med att presentera bakgrunden till arbetet och de krav som vår uppdragsgivare ställt, samt hur dessa ändrats under arbetets gång. Ett resonemang som klargör hur vi kommit fram till de val av komponenter som har använt tas sedan upp, tillsammans med en kort beskrivning av hur de olika komponenterna fungerar och några alternativ till desamma. Delstegen i utvecklingsarbetet beskrivs i form av konstruktionslösning, implementation och testning, samt en projektberättelse. Slutligen tar vi upp de erfarenheter och slutsatser som vi tagit till oss under arbetet.

2 Inledning

Eftersom vi båda är intresserade av webbaserade databaser, var valet inte svårt när vi skulle söka examensarbete, då en av forskargrupperna på universitet erbjöd en uppgift som gick ut på att skapa en sådan.

2.1 Bakgrund

Institutionen för datavetenskap vid Karlstads Universitet har en forskargrupp som arbetar med att ta fram ett delvis tillförlitligt transportprotokoll (Partly Reliable Transport Protocol, PRTP). För att underlätta deras arbete behövs verktyg för hantering av länkar till relaterat forskarmaterial och information om vetenskapliga konferenser. I dagsläget lagras alla länkar i en vanlig textfil. Eftersom antalet länkar har ökat och nu är stort, har det blivit opraktiskt att hantera dessa som en fil och ett behov för en databas har uppstått. Man har även funnit ett behov av motsvarande hantering för information om vetenskapliga konferenser.

2.2 Syfte

Syftet med vårt arbete är att designa och implementera ett system för förenklad hantering av länkar och konferensinformation som skall finnas tillgängligt på webben för alla inom gruppen, samt även för allmänheten.

3 Förutsättningar och krav

Det här kapitlet beskriver de krav som uppdragsgivaren (PRTP-gruppen) har ställt, samt de förutsättningar under vilka systemet skall fungera.

3.1 Förutsättningar

Gränssnittet ska vara webbaserat.
Systemet skall bestå av en databas som skall ligga på en Linux webserver av typ Apache, eftersom det är den typ av webserver som PRTP-gruppen använder.

3.2 Krav

Systemet skall bestå av två delar, en för administration av databasen och en för sökning. Den administrativa delen skall endast finnas tillgänglig för forskargruppens personal, medan den andra skall vara tillgänglig för alla. Hur tillgängligheten till den administrativa delen skall hanteras kommer uppdragsgivaren själv sköta om.

3.2.1 Information som systemet skall innehålla

Informationen som skall finnas om länkar skall bestå av: sökväg/URL, namn/titel på länken, en kort beskrivning av innehållet, nyckelord för sökning så som typ av sida, ämne och eventuellt subämne (t.ex. ämne: transportprotokoll och subämne: TCP), namn på den som lagt till länken samt datum då den lades in i databasen.

Informationen som skall finnas om konferenser skall bestå av: sökväg/URL, konferensens namn, en kort beskrivning av vad konferensen behandlar, ämne, senaste datum för att skicka in artiklar, datum för konferensen, plats, URLer till artiklar/skrifter som läggs ut i samband med konferensen s.k. proceedings.

3.2.2 Funktionella krav

Det administrativa gränssnittet skall ge möjlighet till:

- inmatning av nya URLer/konferenser
- bortagning av URLer/konferenser som inte längre är aktuella
- editering av befintliga URLer/konferenser
- inmatning av namn, ämne och typ av sida

- generering av statiska sidor för URLer/konferenser, t.ex. lista ämnen för URLer/konferenser där varje ämne generar en statisk HTML-sida, vilket minskar antal anrop till databasen
- selectboxar skall finnas där man väljer namn, ämne respektive typ av sida i de gränssnitt där dessa förekommer

I gränssnittet för sökning skall man kunna göra sökningar på valda delar av den information som finns lagrad om URLer och konferenser i databasen. Man skall även kunna lista ämnen och subämnen för URLerna, varifrån man sedan kan välja att visa ”klickbara” URLer för respektive ämne/subämne, s.k. browsing. För konferenser skall man kunna välja att visa vilka konferenser som äger rum under en viss månad ett visst år. På samma sätt skall man även kunna visa senaste datum för att skicka in artiklar till konferenser. Listade ämnen, subämnen och månader skall vara lagrade som statiska HTML-sidor. När nya URLer och konferenser skall läggas in i databasen skall det kontrolleras så att inte dubletter kan läggas in, t.ex. om olika personer ger samma URL olika titlar.

3.2.3 Gränssnittskrav

Gränssnittet skall vara tydligt, lätt att läsa och använda, så att inga manualer krävs. Det bör även gå i stil med institutionens övriga webbsidor.

3.2.4 Utbyggnad och flexibilitet

Eventuellt skall flera forskargrupper kunna använda systemet, man kan då tänka sig att lägga till en söknyckel för forskargrupper. Systemet kan i framtiden komma att användas på olika typer av servrar. Man kan även tänka sig funktionalitet för att cacha sidor som man vill ha kvar och eventuellt automatisk utrensning av sidor som ej längre finns tillgängliga. I övrigt har inga krav om utbyggnad uttalats men flexibilitet och möjligheter till uppdateringar bör eftersträvas.

3.3 Ändringar och tillägg av krav

De flesta ändringar som varit aktuella har främst gällt utseendet hos gränssnittet och vilka attribut som skall användas vid sökning. Från början var det bestämt att man bara skulle kunna söka konferenser genom att ange dess namn, senare kom vår uppdragsgivare fram till att man även behöver ha ämne som söknyckel. Efter det möte där prototypen presenterades för forskargruppen tillkom följande ändringar och tillägg:

- Vid sökning efter URLer så skall man även kunna söka på den person som har lagt till länken och på det datum då den lades in, samt tidigare än eller efter detta datum.
- När man lägger till URLer så skall det finnas en pop-up-ruta för subsubjects, där man kan välja ett subämne som redan finns eller lägga till ett nytt.
- Man ville även ha en kontroll på delsträngar av sökvägen till URLer (t.o.m. första biblioteket t.ex. www.cs.kau.se/johan/…..) som varnar, så att man inte lägger till flera URLer som länkar till samma artikel/konferens.
- Man ville även ha s.k. wildcard-sökning d.v.s. sökning på delsträngar t.ex. video*.

Även under arbetet med implementationen kom vi fram till en del förbättringar och ändringar tillsammans med uppdragsgivaren. När man editerar konferenser så är det inte nödvändigt att kunna byta ämne, en möjlighet som inte heller finns när man editerar URLer. När det gäller statiska HTML-sidor för browsing av URLer, skall dessa sidor uppdateras varje gång en uppdatering av de tabeller som har samband med URLer sker. Eftersom det kommer att bli många uppdateringar har vi istället valt att skapa sidorna dynamiskt genom anrop till databasen vid browsing av URLer.

4 Bakgrundsfakta

Det här kapitlet beskriver hur vi har resonerat när vi valt databas och programmeringsspråk, samt kort om hur dessa fungerar och några alternativ.

4.1 Resonemang

Tillsammans med forskargruppen bestämdes att informationen ska ligga i en databas för att underlätta underhåll, eventuell utbyggnad och för att skapa större flexibilitet. Det skulle inte vara praktiskt att implementera databasen som en vanlig fil, eftersom den data som skall lagras är mer komplex och omfattande i storlek samt har inbördes relationer. I samråd med handledaren valdes mySQL eftersom denna databas är väldokumenterad samt lätt att installera och administrera. Vi har valt att använda javaservlets för att hämta information och göra operationer på databasen, bland annat för att uppnå plattformsoberoende. För att servlets skall kunna kommunicera med databasen så behövs en koppling och eftersom vi arbetar med Java så är JDBC ett naturligt alternativ, kapitel 4.4 motiverar detta.

4.2 Javaservlets

En servlet är ett stycke Javakod som läggs in och körs inuti en ”servlet engine”, t.ex. en webserver. Den tar emot och svarar på requests från klienter. När en klient t.ex. behöver information från en databas, kan en servlet köras för att ta emot en request, hämta och behandla data som klienten behöver och returnera det till klienten.

Servlets kan användas för olika webrelaterade applikationer för kommunikation med applets, databaser eller annan mjukvara via sockets och RMI (Remote Method Invocation) mekanismer, eftersom de har tillgång till hela Javas klassbibliotek.

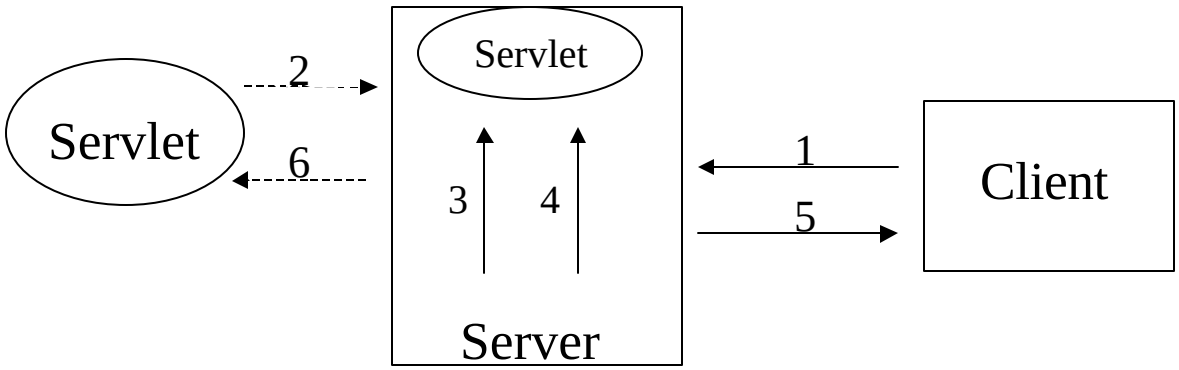
Servletarkitekturen består av två paket: javax.servlet och javax.servlet.http. Paketet javax.servlet innehåller allmänna gränssnitt som implementeras, samt klasser som ärvs av alla servlets. Paketet javax.servlet.http innehåller klasser som ärvs när man skapar http-specifika servlets. Centralt för arkitekturen är gränssnittet javax.servlet.Servlet som tillhandahåller en stomme för alla servlets.

Servletgränssnittet definierar metoder för initiering av en servlet, för att ta emot och svara på requests från en klient och för att ta bort en servlet och dess resurser. Metoderna beskriver servletens livscykel. Ett typiskt scenario för hur metoderna i en servlets livscykel fungerar illustreras i figur 4.1:

1. En användare skriver in en URL (Uniform Resource Locator) till en webbrowser. Browsern genererar en HTTP-request (HyperText Transfer Protocol) för URLen och skickar den till servern.
2. HTTP-requesten tas emot av servern som mappar den till en specifik servlet. Servleten laddas dynamiskt till serverns adressutrymme (minne), där den ligger kvar och är tillgänglig för att behandla fler HTTP-requests som tas emot från klienter.
3. Servern anropar servletens initieringsmetod *init()*, när servleten först läggs in i minnet.
4. Servern anropar servletens servicemetod *service()*, som avgör vilken typ av request GET, POST o.s.v. som klienten begärt. HTTP-requesten behandlas sedan utifrån detta i metoderna *doGet()*, *doPost()* eller motsvarande. Servicemetoden anropas för varje HTTP-request.
5. Servleten kan läsa data från HTTP-requesten och formulera ett HTTP-response för klienten.
6. Slutligen kan servern besluta att ta bort servleten ur minnet. Algoritmen för att ta bort servleten ur minnet är specifik för varje server. Servern anropar metoden *destroy()* för

att släppa resurser som allokerats för servleten. Minne som allokerats för servleten och dess objekt kan sedan återhämtas genom ”garbage-collection”.

Ovanstående fakta är hämtade ur [1], [2], [3].



Figur 4.1: Servletens livscykel.

4.2.1 Alternativ till Java servlets

Det finns ett antal olika alternativ till javaservlets, bl.a. CGI (Common Gateway Interface), PHP (Personal Home Page), serverside javascript och Microsofts ASP (Active Server Pages).

- CGI är en av de vanligaste lösningarna för webapplikationer. CGI-applikationen är en oberoende modul som tar emot begäran från en webserver. Applikationen behandlar den mottagna datan och skickar tillbaks ett svar till servern, vanligtvis som HTML(HyperText Markup Language). Servern skickar svaret vidare till browsern. CGI har blivit en standard som används av de flesta webservrar.

  CGI medför dock vissa problem, av vilka de vanligaste är:

  - En webserver skapar en ny process varje gång den tar emot en CGI-request, vilket är kostsamt i form av processor och minnesresurser.
  - Trots att CGI kan implementeras i de flesta språk, så är det vanligaste plattforms-oberoende språket Perl. Perl kräver dock att servern startar en ny tolk för varje begäran, vilket tar längre tid än att köra kompilerad kod och tar mycket resurser.
  - Eftersom CGI körs i helt separata processer från webservern, så vet inte browsern vad som händer om ett CGI-program terminerar innan den har gett ett svar till servern. Browsern fortsätter att vänta på svar tills dess timer går ut.

- PHP är ett serverside HTML-inbäddat språk [4]. I PHP skriver man HTML-script med inbäddad kod som utför något, till skillnad från CGI-script som är skrivna i andra språk med kommandon för att skriva ut HTML. På liknande sätt som javaservlets, exekverar koden för PHP på servern.
- Serverside javascript är en annan lösning för implementation av dynamiska websidor, där javascript bäddas in i förkompilerade HTML-sidor. Genom att förkompilera websidorna förbättras prestandan, men bara servrarna Netscapes Enterprise och FastTrack har stöd för detta.
- ASP är Microsofts lösning på problemet med dynamiska websidor. Även ASP är inbäddat i HTML-sidor men inte förkompilerat. På samma sätt som serverside javascript är även ASP bundet till en specifik server, nämligen Microsofts Internet Information server.

4.2.2 Fördelar med Java servlets

- Effektiv, eftersom servletens initieringskod endast exekveras första gången den laddas in i webservern. När servleten finns i servern behöver bara anrop göras till servicemetoderna, vid nya förfrågningar. Den här tekniken är mer effektiv än att ladda in ny exekverbar kod vid varje förfrågan [2].
- Persistent, eftersom servlets kan bevara sitt tillstånd mellan requests. När en servlet laddas, ligger den kvar i minnet medan den servar inkommande requests.
- Portabel, eftersom servlets utvecklas i Java som är plattformsoberoende. Detta tillåter att servlets flyttas till andra operativsystem utan att koden behöver ändras.
- Robust, eftersom servlets är utvecklade med tillgång till hela JDK (Java Development Kit). Java tillhandahåller en väldefinierad exception-hierarki för felhantering och det finns en garbage-collector för att undvika problem med minnesläckage.
- Utbyggbar, eftersom servlets utvecklats i ett objektorienterat språk som Java, kan de byggas ut till nya objekt som bättre tillgodoser behoven.
- Säkert, eftersom servleten körs på serversidan och ärver säkerheten som webservern tillhandahåller. Servlets kan också dra fördel av javas securitymanager.

4.3 MySQL

MySQL är en SQL-databasserver utvecklad av TcX. Det är en liten, multithreaded, multiuser och snabb databas som erbjuder APIs (Application Programming Interface) för många olika språk.

Följande är några av de karakteristika i mySQL som är relevanta för vårt projekt:

- Har stöd för automatisk ökning i kolumner. När man skapar en tabell väljer man om man vill ha en kolumn med index. Varje gång man lägger till en rad i tabellen ökas ett ”index” i kolumnen.
- Portabel, d.v.s. den kan användas på olika plattformar.
- Mycket väldokumenterad bl.a. eftersom den har många användare.
- Saknar stöd för främmandenycklar, så sett att främmandenycklar inte uppdateras automatiskt vid ändringar i databasen.
- Kan inte hantera nästlade select-satser, istället kan man använda s.k. kartesiska produkter för att ställa frågor som berör flera tabeller.
- Varje mySQLclient tilldelas en tråd i mySQLserver, vilket tillåter flera olika användare access till en tabell samtidigt. Alla operationer är atomiska, d.v.s. ingen annan användare kan ändra resultatet av en fråga under tiden som frågan exekveras.

MySQL är en databas som fortfarande är under utveckling. Ovanstående fakta är hämtade ur [5], [6], [7].

4.3.1 Alternativ till mySQL

Sybase är ett av flera alternativ till mySQL, den är dock större och har funktionalitet som gör den mer komplex. Nedan följer några exempel på vad som karaktäriserar sybase, enligt [8]:

- Har stöd för främmandenycklar, relationer mellan primär och främmandenycklar definieras när tabellerna skapas.
- Portabel.
- Stored procedures, vilket är en utökning av SQL. SQL är ett icke-proceduriellt språk, där manipulation av data sker i steg. Sybase har utökat SQL med proceduriella kontrolluttryck, vilka kallas transact-SQL, utökningen inkluderar bl.a. villkor (if) och loopar (while). Några av fördelarna med Sybase stored procedures är:
  - SQL-frågorna i procedurerna kompileras bara en gång.

- Om många användare behöver anropa samma fråga med olika parametrar så kan det vara praktiskt att lägga frågan i en procedur som lagras centralt och delas av flera.

4.4 JDBC

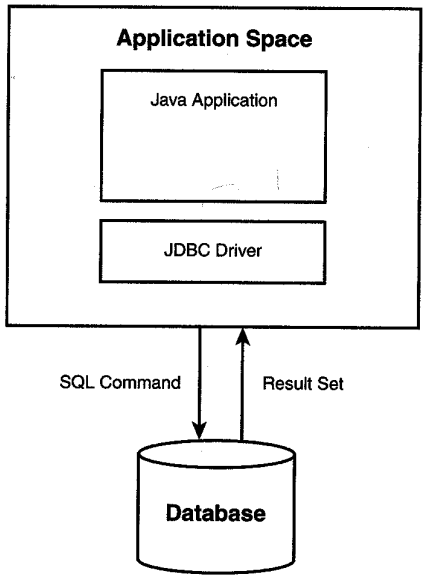
JDBC (Java Database Connectivity) är en ren java-API som används för att exekvera SQL-uttryck. Den erbjuder en mängd klasser och interface som kan användas när man skriver databasapplikationer. Genom att använda JDBC, slipper man skriva olika program för att accessa olika typer av databaser, som t.ex. Sybase och mySQL. En applikation skriven i Java kan köras på olika plattformar[9]. Kombinationen av java och JDBC gör att ett program kan köras på olika plattformar utan ändringar. JDBC utökar möjligheterna för vad som kan göras i java, t.ex. med Java och JDBC API så är det möjligt att skapa en websida som innehåller en applet som använder information som hämtas från en remote databas.

Grundläggande JDBC-samverkan kan delas in i följande fyra steg (sidan 146 [2]):

1. Öppna en förbindelse till databasen.
2. Exekvera ett SQL-uttryck.
3. Behandla resultatet.
4. Stänga förbindelsen till databasen.

JDBC erbjuder stöd för two- och three-tier databasaccessmodell. När man använder two-tier databasaccessmodell, vilket vi har gjort, så kommunicerar Java applikationen direkt med databasen. Detta genomförs genom att man använder en JDBC-driver som sänder kommandon direkt till databasen. Databasen sänder resultatet direkt tillbaka till applikationen, se figur 4.2.

Three-tier-modellen är lite mer komplicerad. JDBC-drivern skickar kommandon till ett mellanskikt, som i sin tur skickar kommandon till databasen. Resultatet skickas tillbaka till mellanskiktet, som skickar det vidare till applikationen.



Figur 4.2: JDBC two-tier modell (hämtat ur [2])

JDBC är ett ”lågnivåinterface”, enligt [9], vilket innebär att det anropar SQL-kommandon direkt. Denna egenskap gör att JDBC fungerar bra och är enklare att använda än andra databaskopplings-APIs.

4.4.1 Jämförelse av JDBC och ODBC

ODBC (Open Database Connectivity) API är ett av de mest använda interfacen för access av relationsdatabaser. ODBC erbjuder möjlighet att koppla samman nästan alla typer av databaser på nästan alla typer av plattformar, enligt [9].

Enligt ODBC-standarden skall det finnas en ODBC-driver för varje typ av databas. Applikationer kommunicerar med en specifik databas genom ODBC-drivern för den specifika databasen och genom en s.k. drivermanager. Med andra ord översätter drivermanagern mellan applikationen och ODBC-drivern och ODBC-drivern översätter mellan drivermanagern och den specifika databasen [10].

Man kan använda ODBC från java men det är enklare att använda JDBC, eftersom ODBC använder ett interface byggt i C och anrop från java till C kod har ett antal nackdelar i form av t.ex. implementation och portabilitet för applikationer. Direkta översättningar av ODBC C API till java API är inte önskvärt då java t.ex. inte har stöd för pekare vilket ofta används i C. ODBC är mer komplext än JDBC som är byggt för att vara enkelt samtidigt som det tillåter att användaren gör mer avancerade operationer. JDBC API är ett naturligt javainterface för

grundläggande SQL-abstraktioner och koncept. Istället för att bygga JDBC från grunden har man valt att utgå ifrån ODBC.

5 Beskrivning av konstruktionslösningen

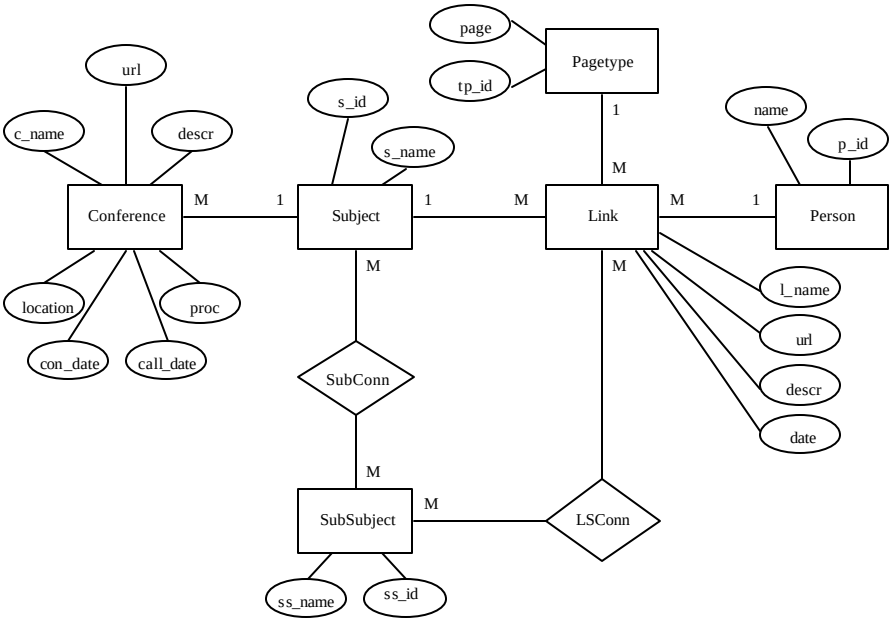
Kapitlet beskriver hur vi gått till väga under konstruktionsarbetet.

5.1 Prototyp

Det första steget i arbetet var att göra en prototyp av användargränssnittet. Detta för att underlätta det fortsatta arbetet och för att vara säkra på att vi var överens med vår uppdragsgivare om vad systemet skulle innehålla och klara. Ett antal HTML-sidor som kopplats samman visar hur det är tänkt att systemet skall fungera. Inga operationer kan utföras då det inte finns någon databas bakom, men man kan klicka sig genom sidorna och se hur det är tänkt att resultaten skall presenteras.

5.2 Databasen

Utifrån den information som databasen skall hålla och hur denna information skall användas gjorde vi en E/R(Entitet/Relations)-modell, se figur 5.1. Med prototypen som utgångspunkt identifierades sedan de frågor som kommer att behövas för att hantera databasen och för att hämta information från densamma. Frågorna listades i form av SQL-satser. En mappning av E/R-modellen mot relationsdatamodellen gjordes där entiteter och deras attribut listades och typer bestämdes, även deras nycklar identifierades och listades se bilaga A.



Figur 5.1: E/R-modell

5.3 Servlets

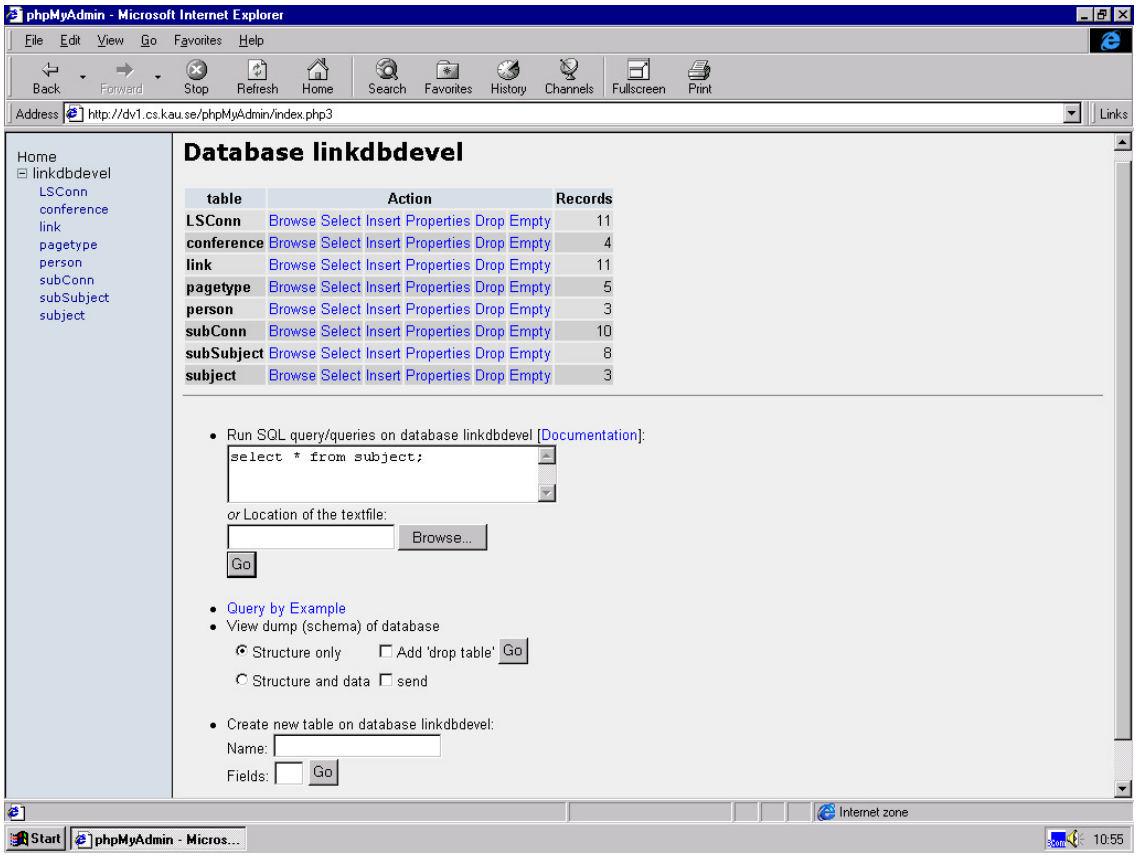
Det här steget bestod till största delen av implementation, eftersom vi redan tidigare identifierat de komponenter som behövs och hur de skall fungera tillsammans. Vi gjorde inget direkt förarbete innan vi började koda, istället bestämde vi oss för att bygga servletarna i delsteg och att varje operation som t.ex. ”lägg till” och ”ta bort” URL skall göras i separata servlets.

6 Implementation och testning

Kapitlet beskriver hur arbetet med implementation och testning fortlöpte.

6.1 Implementation och testning av databasen

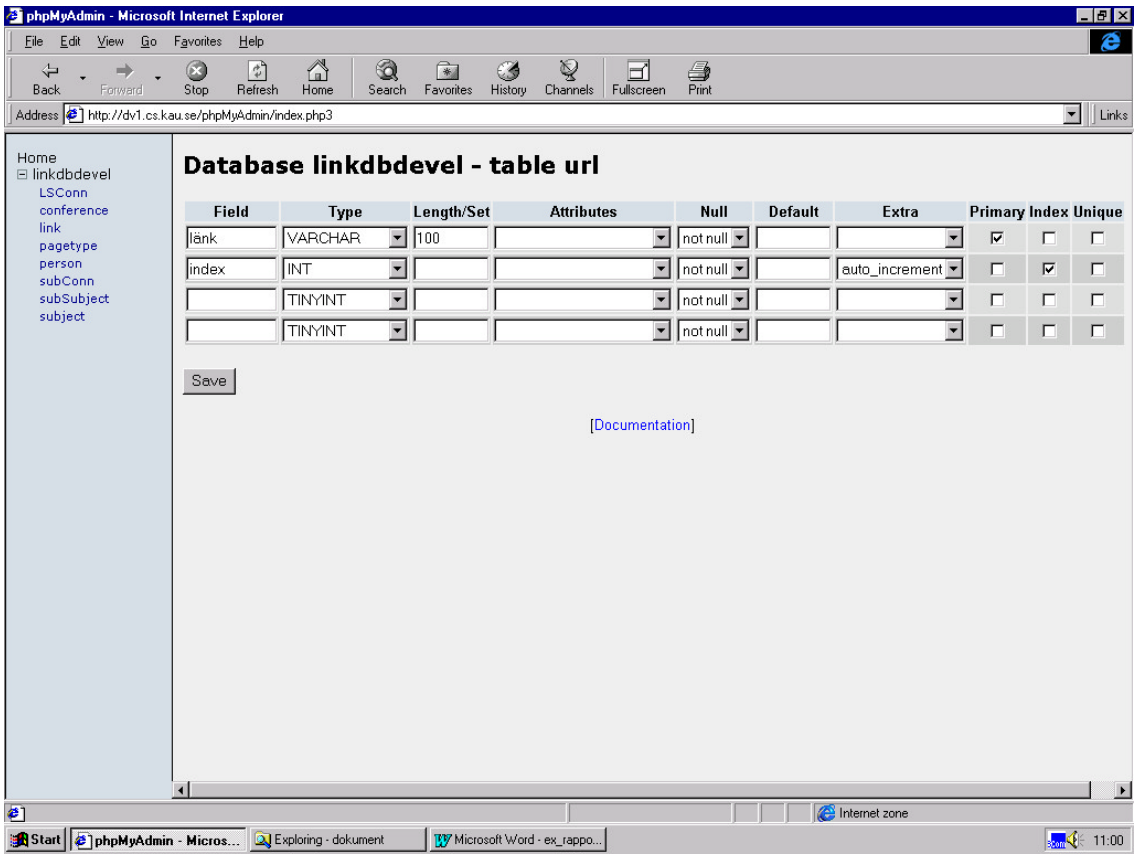
Med relationsdatamodellen som underlag skapades de tabeller med respektive attribut som databasen skall hålla. Arbetet med databasen förenklades med hjälp av ett grafiskt gränssnitt som mySQL tillhandahåller, där man kan skapa och editera tabeller och deras innehåll, se figur 6.1.



Figur 6.1: Grafiskt gränssnitt.

För att skapa en tabell namnges denna och det antal fält som tabellen skall innehålla anges, se figur 6.1. Fälten namnges sedan med attributnamn och ett antal parametrar så som typ (int,

varchar o.s.v.) och längd för respektive attribut sätts, enligt figur 6.2. Man väljer även vilka attribut som skall vara obligatoriska (not null) och om en kolumn skall ha index och om den då skall ökas automatiskt.



Figur 6.2: Gränssnitt för att skapa tabell.

De frågor som listades under konstruktionsarbetet låg till grund för de SQL-satser som behövs för att utföra operationer på databasen. SQL-satserna testades sedan mot databasen i det grafiska gränssnittet, figur 6.1, för att se att resultatet blev det önskade.

Några direkta problem har inte förekommit under arbetet med SQL-frågorna. Vi fick dock byta strategi i ett fall. Vi hade från början tänkt använda uttrycket ”exists”, för att se om det sökta värdet finns i databasen, men det visade sig att det inte fanns stöd för uttrycket i mySQL. Istället har vi genom select-satser sökt i databasen och låtit servletarna utvärdera resultatet för att se om det sökta värdet finns i databasen.

6.1.1 Exempel SQL-frågor

Vi har här valt att ta upp de olika typer av frågor som vi implementerat, eftersom de flesta frågorna ofta förekommer i olika varianter.

Den vanligaste typen av fråga hämtar alla rader från en viss kolumn i en viss tabell. Ett exempel är den SQL-sats som används för att sätta värden i select-boxen för ämne i formuläret för att lägga till URLer i figur 6.3:

```
select s_name from subject;
```

En annan typ av fråga som förekommer ofta hämtar ett värde för en angiven parameter. Ett exempel är den fråga som hämtar id för angivet ämne:

```
select s_id from subject where s_name='subject';
```

Ett par exempel på hur man lägger in värden i databasen illustreras av följande frågor:

```
1. insert into link values('url', 'u_name', 'descr', 'datum', 'sub_id',
   'tp_id', 'p_id');

2. insert into subject values('', 'subject');
```

I det första exemplet läggs information om en länk in i tabellen link, url är själva URLen, u_name är titeln på URLen, descr är en beskrivning av innehållet i artikeln som URLen länkar till, datum - då man lägger in länken, sub_id, tp_id och p_id är främmandenycklar.

Det andra exemplet visar hur ett nytt ämne läggs in i databasen. Den tomma strängen motsvarar indexkolumnen vilken sätts automatiskt av databasen.

Att ta bort värden ur databasen kan t.ex. göras med följande fråga, som tar bort den rad ur tabellen link som matchar den angivna URLen:

```
delete from link where url='url';
```

När man vill göra en sökning utan att fylla i hela söksträngen, kan man göra som i följande exempel, där man anger en del av sökvägen till en URL följd av %. Alla URLer som matchar delsträngen returneras.

```
select l_name, url from link where url like 'urlInput%';
```

Ett exempel på en av de mer komplexa frågorna är när man skall lista alla URLer i sök-gränssnittet. Här kan många olika kombinationer av parametrar förekomma. Användaren väljer själv vilka fält han vill fylla i. För de fält som lämnats tomma sätts parametrarna (title, descr o.s.v.) till ”%” i servleten, vilket innebär att de matchar vad som helst. Resultatet ordnas efter datum (m.h.a. order by) med det senaste datumet först (desc). För att slippa dubletter vid presentationen av resultatet används distinct.

```
select distinct link.url, l_name, descr, date, p_name, page, s_name
from link, person, pagetype, subject, subSubject, LSConn
where l_name like 'title'
and descr like 'descr'
and ss_name like 's_subject'
and page like 'page'
and s_name like 'subject'
and p_name like 'submitter'
and date like 'datum'
and link.tp_id=pagetype.tp_id
and link.p_id=person.p_id
and link.s_id=subject.s_id
and link.url=LSConn.url
and LSConn.ss_id=subSubject.ss_id order by date desc;
```

6.2 Implementation och testning av servlets

Arbetet med implementationen och testningen har skett parallellt eftersom vi har byggt servletarna steg för steg och testat efter varje steg. Testningen har skett på så sätt att vi först har kontrollerat de parametrar som vi tar emot från formulären och att alla obligatoriska fält är ifyllda. Därefter har nödvändig data hämtats från databasen och kontrollerats. Slutligen har vi kontrollerat att de värden som angivits har lagts in i databasen. Här finns dock några brister.

Om man matar in strängar som är längre än vad som definierats i databasen så lagras bara så mycket av strängen som får plats, t.ex. s_name i tabellen subject (se bilaga B) får vara maximalt tjugo tecken. Om man skriver in en längre sträng, lagras bara de tjugo första tecknen i databasen.

Inga kontroller finns heller för att man matar in rätt typ av tecken, t.ex. om man skriver in bokstäver där det skall vara siffror och omvänt. Vi förutsätter att användarna följer instruktionerna i formulären. Det finns dock ett fall då det kan skapa vissa problem om man

skriver in fel typ av tecken. Ett sådant exempel är om man skriver in andra tecken än siffror i fält som i databasen är satta till typen date så läggs datumet 0000-00-00 in i databasen.

Ett av de problem som vi stött på uppstod när vi skulle hämta en textsträng från en select-box. Strängen klipptes då vid blankradstecken (mellanslag). Problemet löstes genom att ersätta blankradstecknet med ett annat tecken redan när man lade in värdet i select-boxen och sedan byta tillbaka innan det lagras i databasen.

Slutligen fick vi problem när vi ville koppla en viss typ av länk till en viss metod i servleten för browsing av URLer i användargränssnittet. Beroende på om länken var ett ämne eller ett subämne så skulle olika metoder anropas. Detta var dock inte möjligt eftersom länkar bara kan referera till andra filer eller till en punkt inom ett HTML-dokument. Det gick inte att referera till en punkt i samma servlet. Eftersom vi var tvungna att avbryta arbetet så har vi ännu inte löst problemet, men har en idé om att referera till en annan servlet, som utför arbetet, till vilken man skickar med en parameter för ämne/subämne.

6.2.1 Exempelservlet

Systemet består av två delar: ett gränssnitt för sökning och ett för administration av databasen. Med prototypen som grund byggdes ett antal servlets. En för varje operation som användarna kan utföra mot databasen i de olika gränssnitten. Vi har valt att gå igenom en servlet för att visa den generella uppbyggnaden och har begränsat oss till att endast ta upp exempel som är specifika för servlets och JDBC. Den servlet som vi har valt lägger till en ny URL i databasen. Källkod se bilaga B.

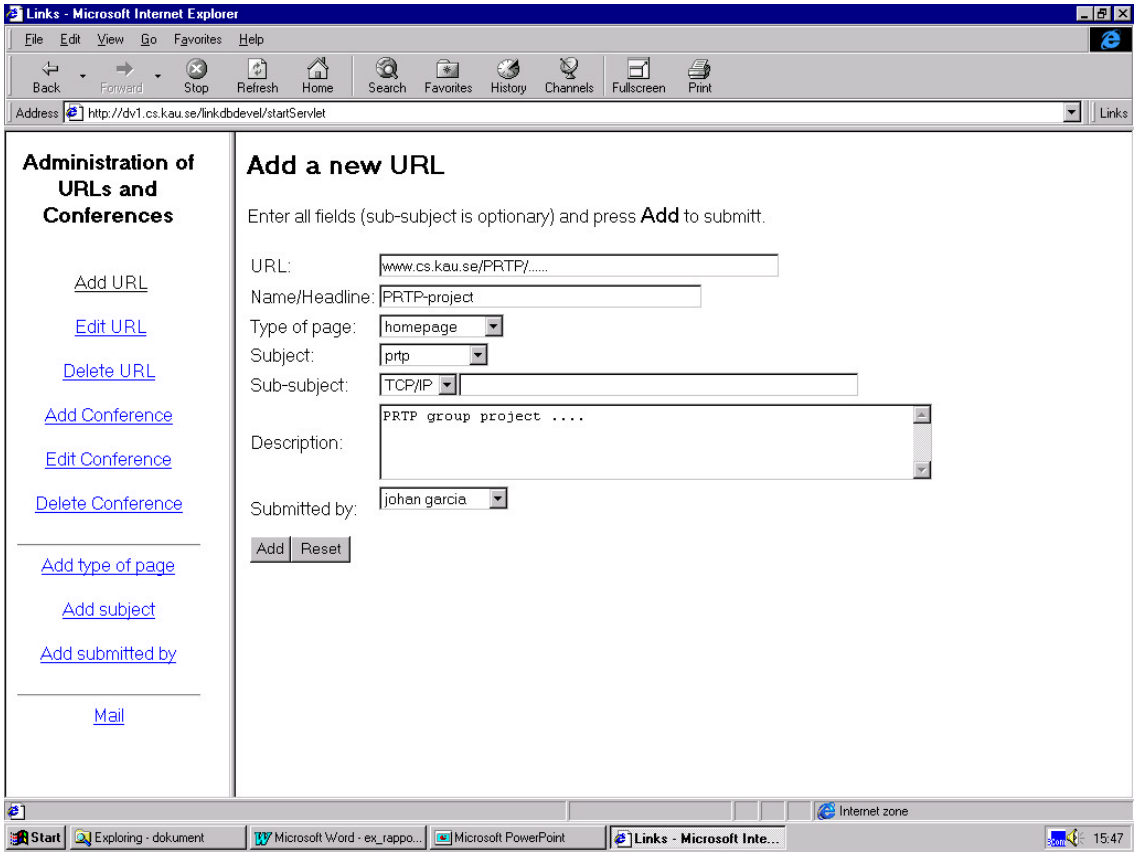
Det första som sker när servleten laddats in i servern är att metoden *init()* anropas. Metoden skickar servletConfig objektet till sin förälder som sparar objektet, samt skapar och initierar de resurser som behövs för att behandla requests. Eftersom inga resurser skapas i den här servleten så behöver ingen *destroy()*-metod implementeras.

```
public void init(ServletConfig config) throws ServletException
    {
         super.init(config);
    }
```

I nästa steg startas servleten genom en GET-request till metoden *doGet()*. Metoden tar emot objekten HttpServletRequest som innehåller information från klienten och HttpServletResponse som kommer att innehålla den information som skall skickas tillbaks till

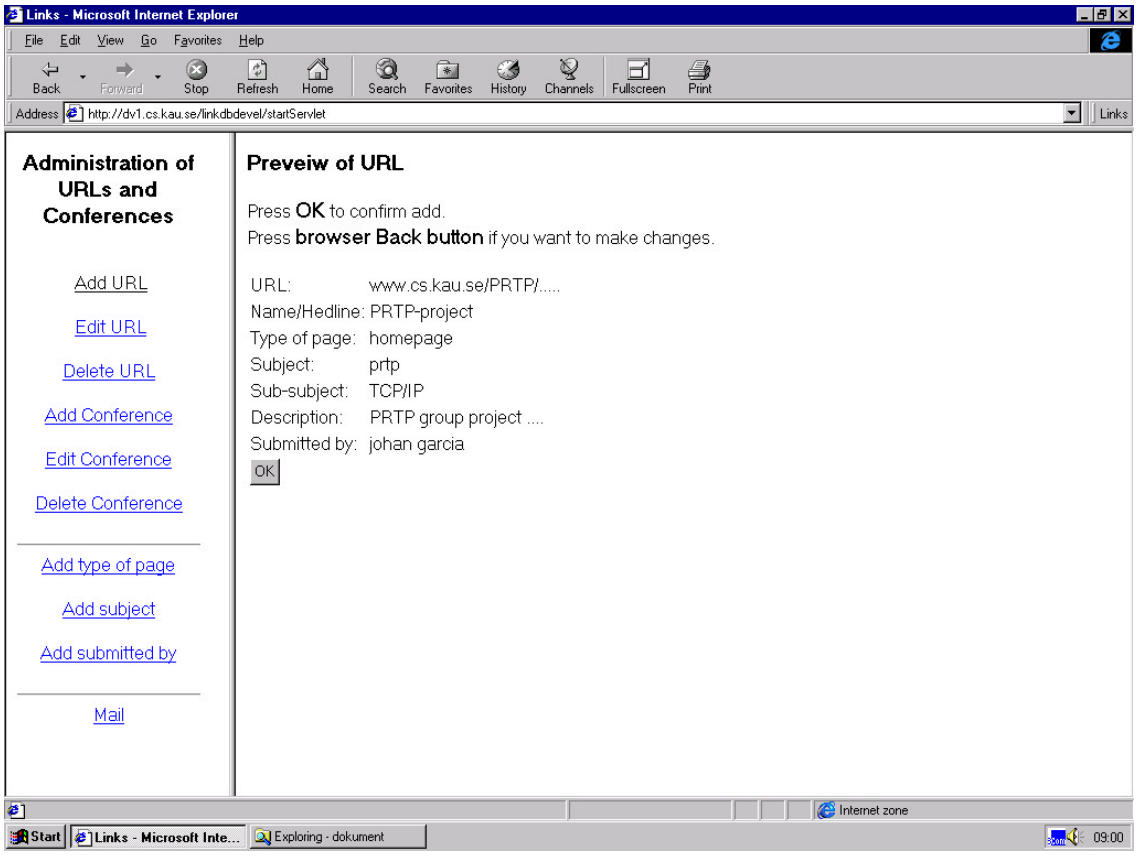
klienten. I metoden *setContentType()* sätts den typ som svaret skall innehålla. Metoden *getWriter()* skapar ett objekt för att HTML-text skall kunna skickas tillbaka till klienten i HttpServletResponse-objektet. Metoden *mkForm()* skriver ut formuläret där man skriver in information om den URL som man vill lägga in i databasen, se figur 6.3.

```
public void doGet (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
    {
         res.setContentType("text/html");
         out = res.getWriter ();
        mkForm();
    }
```



Figur 6.3: Exempel formulär.

När man har fyllt i alla data i formuläret och tryckt på ”Add-knappen” så visas den information som angivits i formuläret, se figur 6.4. Om man då är nöjd trycker man ”OK” och datan läggs in i databasen.



Figur 6.4: Exempel förhandsvisning.

Metoden *doPost()* fungerar på samma sätt som *doGet()*, men den behandlar POST-requests istället. Metoderna *doGet()* och *doPost()* är s.k. servicemetoder. När användaren gjort en knapptryckning i något av servletens formulär, sker en POST-request till *doPost()*-metoden. Alla parameternamn läses in från det aktuella formuläret till en Enumeration varifrån man sedan hämtar önskad parameter.

```
public void doPost  (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
         res.setContentType("text/html");
         out = res.getWriter ();
         Enumeration e = req.getParameterNames();
         String name = null;

         while(e.hasMoreElements()){
            name = (String)e.nextElement();

            if(name.equals("add")) {
              url = req.getParameter("url");
              ....
            }
            if(name.equals("ok")){
              add();
              break;
            }
         }
    }
```

När ett anrop till databasen skall ske, måste man först skapa en koppling. Detta görs i två steg. Först laddas JDBC-drivern och sedan skapas en databaskoppling. Metoden *Class.forName(...).newInstance()* skapar en instans av drivern och registrerar den hos DriverManagern, enligt kodexemplet nedan. Det är allt som krävs för att ladda JDBC-drivern.

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

Efter att drivern har laddats, görs ett anrop till metoden *DriverManager.getConnection()*, enligt kodexemplet nedan, vilken returnerar en koppling till databasen.

```
Connection dbc = DriverManager.getConnection("jdbc:mysql://localhost/
linkdbdevel?user=root");
```

När man sedan vill utföra en operation mot databasen måste ett statement-objekt skapas, vilket används för att skicka SQL-uttrycket till databasen. Detta görs enligt följande:

```
Statement stmt = dbc.createStatement();
stmt.execute("select url from link");
```

Resultatet av frågan hämtas med *stmt.getResultSet()* och läggs i ett ResultSet-objekt, enligt följande:

```
ResultSet rs = stmt.getResultSet();
```

Resultatet som hämtas från databasen bearbetas enligt följande:

```
while(rs.next()){
   url = rs.getString("url");
}
```

När alla operationer mot databasen är slutförda stängs förbindelsen på följande sätt:

```
rs.close();
dbc.close();
```

6.3 Administrationsgränssnittet

Här följer en beskrivning av de servlets som byggts för de operationer som kan göras i det administrativa gränssnittet.

6.3.1 AddUrl

Den här servleten används för att lägga till en ny URL med tillhörande information i databasen.

- Det första som sker när servleten körs är att de värden som skall visas i select-boxarna hämtas från databasen. Därefter skrivs formuläret ut se figur 6.3.
- När användaren trycker på knappen ”Add”, kontrolleras om flera befintliga eller ett /flera nya subsubjects angivits i formuläret samt att alla fält som är obligatoriska är ifyllda.
- Om alla fält är ifyllda, görs en förhandsvisning av angivna data, se figur 6.4.
- När användaren bekräftar förhandsvisningen, genom att trycka på knappen ”OK”, kontrolleras att den angivna URLen inte redan finns i databasen.
- Om URLen inte finns, hämtas id för subject, submitter och pagetype från respektive tabell i databasen samt datum, vilket hämtas från systemet. Därefter läggs den angivna information om URLen in i tabellen link i databasen. Har flera subsubject angivits, delas de isär och det kontrolleras om respektive subsubject finns i databasen eller ej. De subsubjects som inte redan finns läggs in i tabellen subsubject. Därefter läggs url

och id för subsubject in i kopplingstabellen LSConn, samt id för subject resp. subsubject läggs in i kopplingstabellen subConn.

- Slutligen presenteras en sida som talar om att operationerna mot databasen är klara.

6.3.2 DeleteUrl

Den här servleten används för att ta bort en URL med tillhörande information ur databasen.

- När servleten körs, skrivs först en sida ut där man får mata in den URL eller del av den URL som man vill ta bort.
- De URLer som eventuellt matchar sökningen och deras namn hämtas från databasen när användaren trycker på knappen ”Find”.
- URLen och dess titel/namn skrivs därefter ut på en sida tillsammans med en knapp för varje URL, så att man kan välja vilken man vill ta bort.
- När användaren trycker på en knapp för att ta bort en URL, hämtas id för de subsubjects som URLen eventuellt har, från tabellen LSConn. Därefter tas alla tupler som matchar URLen bort ur tabellen LSConn. En sökning görs sedan i tabellen LSConn för att se om fler URLer har samma subsubjects som den URL som nu skall tas bort. Om subsubjecten inte finns i fler tupler i tabellen LSConn så tas de bort ur tabellen subSubject, även de tupler i tabellen subConn som matchar subsubjecten tas bort. Till sist tas tupeln med den valda URLen bort ur tabellen link.
- Slutligen presenteras en sida som talar om att operationerna mot databasen är klara.

6.3.3 EditUrl

Den här servleten används för att ändra information om en URL som finns i databasen.

- När servleten körs, skrivs först en sida ut där man får mata in den URL som man vill editera.
- När användaren trycker på knappen ”Find”, hämtas alla data om den angivna URLen från databasen. Ett formulär med dessa uppgifter ifyllda visas, i vilket användaren kan göra sina ändringar.
- När man trycker på knappen ”Edit”, kontrolleras att alla obligatoriska fält är ifyllda.
- Om alla fält är ifyllda, görs en förhandsvisning av angivna data på samma sätt som i figur 6.4.
- När användaren bekräftar förhandsvisningen, genom att trycka på knappen ”OK”, tas URLen bort ur databasen på samma sätt som i delete URL. Därefter läggs informationen om den editerade URLen in i databasen på samma sätt som i ’AddUrl’.

- Slutligen presenteras en sida som talar om att operationerna mot databasen är klara.

6.3.4 AddConference

Den här servleten används för att lägga till en ny konferens med tillhörande information i databasen.

- Det första som sker när servleten körs är att de värden som skall visas i select-boxen hämtas från databasen. Därefter skrivs ett formulär ut på motsvarande sätt som i servleten addUrl.
- När användaren trycker på knappen ”Add”, kontrolleras att alla fält som är obligatoriska är ifyllda.
- Om alla fält är ifyllda, görs en förhandsvisning av angivna data.
- När användaren bekräftar förhandsvisningen, genom att trycka på knappen ”OK”, kontrolleras att den angivna konferensen inte redan finns i databasen.
- Om konferensen inte finns, hämtas id för subject från tabellen subject i databasen. Därefter läggs den angivna information om konferensen in i tabellen conference i databasen.
- Slutligen presenteras en sida som talar om att operationerna mot databasen är kla ra.

6.3.5 DeleteConference

Den här servleten används för att ta bort en konferens med tillhörande information ur databasen.

- När servleten körs, skrivs först en sida ut där man får mata in namnet/titeln eller del av namnet/titeln på den konferens som man vill ta bort.
- De konferenser som eventuellt matchar sökningen hämtas från databasen när användaren trycker på knappen ”Find”.
- Konferensens titel/namn skrivs därefter ut på en sida tillsammans med en knapp för varje konferens, där man kan välja vilken man vill ta bort.
- När användaren trycker på en knapp för att ta bort en konferens, tas tupeln med den valda konferensen bort ur tabellen conference.
- Slutligen presenteras en sida som talar om att operationerna mot databasen är klara.

6.3.6 EditConference

Den här servleten används för att ändra information om en konferens som finns i databasen.

- När servleten körs, skrivs först en sida ut där man får mata in namnet/titeln på den konferens som man vill editera.
- När användaren trycker på knappen ”Find”, hämtas alla data om den angivna konferensen från databasen. Ett formulär med dessa uppgifter ifyllda visas, i vilket användaren kan göra sina ändringar.
- När man trycker på knappen ”Edit”, kontrolleras att alla obligatoriska fält är ifyllda.
- Om alla fält är ifyllda, görs en förhandsvisning av angivna data.
- När användaren bekräftar förhandsvisningen, genom att trycka på knappen ”OK”, tas konferensen bort ur databasen på samma sätt som i DeleteConference. Därefter läggs informationen om den editerade konferensen in i databasen på samma sätt som i AddConference.
- Slutligen presenteras en sida som talar om att operationerna mot databasen är klara.

Add-, delete- och editConference skall även generera statiska HTML-sidor, denna funktion är dock inte implementerad ännu.

6.3.7 AddSubject

Den här servleten används för att lägga till ett nytt ämne i databasen.

- När servleten körs, skrivs först ett formulär ut där man får mata in det ämne som man vill lägga in i databasen.
- När användaren trycker på knappen ”Add”, kontrolleras att data har matats in i formuläret, samt att det angivna ämnet inte redan finns i databasen.
- Om ämnet inte finns, läggs det in i tabellen subject i databasen.
- Slutligen presenteras en sida som talar om att operationerna mot databasen är klara.

AddTypeOfPage och AddSubmitter ser ut exakt som AddSubject, men handhar typ av sida respektive namn på den som lagt till URL.

6.4 Användargränssnittet

Här följer en beskrivning av de servlets som skall användas för de operationer som kan göras i användargränssnittet, dessa servlets är dock ej färdiga ännu.

6.4.1 Conferences

Den här servleten skall lista konferenser efter datum då de äger rum respektive senaste datum för att skicka in artiklar. Listningen skall ske månadsvis. Servleten är inte implementerad ännu, men kommer att länka in ett antal statiska sidor som genereras av servletarna add-, delete- och editConference när uppdateringar i tabellen conference sker. Det kommer att finnas två sådana sidor för varje månad och det aktuella året: en för konferenser som äger rum under månaden och en för konferenser vars senaste datum för att skicka in artiklar förfaller under månaden. De statiska sidorna skall sedan kopplas till länkar som visas när servleten körs. När man klickar på en länk skall den statiska sidan för länken visas som i figur 6.5.

6.4.2 SearchUrl

Den här servleten listar alla URLer beroende på angivna parametrar. Servleten är i stort sett klar.

- Det första som sker när servleten körs är att de värden som skall visas i select-boxarna hämtas från databasen. Därefter skrivs ett formulär ut på liknande sätt som i servleten addUrl och addConference.
- När användaren trycker på knappen ”Search”, görs en sökning i databasen med de i formuläret angivna parametrarna som indata.
- Resultatet av sökningen presenteras i tabellform på liknande sätt som figur 6.5, där URLerna skall vara klickbara länkar.

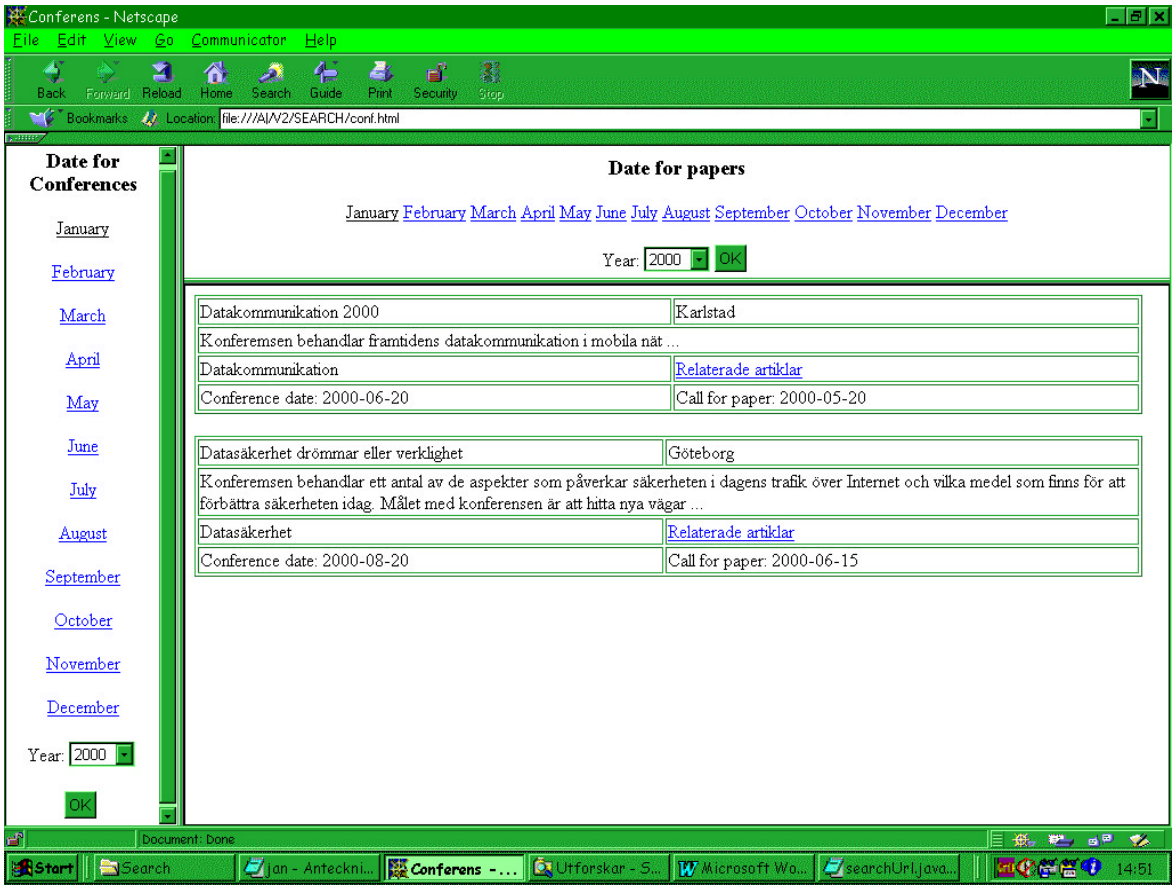
6.4.3 BrowseUrl

Den här servleten listar alla URLer efter ämne respektive subämne. Servleten är påbörjad och en del arbete återstår innan den är klar.

- Det första som sker när servleten körs är att alla ämnen och deras respektive subämnen hämtas från databasen och listas som länkar.
- När användaren klickar på någon av länkarna, skall alla URLer som har matchande ämne/subämne presenteras på samma sätt som för searchUrl.

6.4.4 SearchConference

Den här servleten listar alla konferenser beroende på angivna parametrar. Servleten är inte implementerad ännu, men den kommer i stort sett att vara uppbyggd på samma sätt som SearchUrl. Presentationen av resultatet kommer se ut på liknande sätt som i figur 6.5.



Figur 6.5: Prototyp för listning av konferenser.

7 Projektberättelse

Vid vårt första möte träffade vi alla som jobbar i PRTP-gruppen (vår uppdragsgivare) och de berättade hur de hade tänkt att systemet skall fungera och vad de ville ha ut av det. Vid nästa möte med handledaren diskuterade vi kring materialet från mötet med forskargruppen. Det mesta verkade klart men måste specificeras i en kravspecifikation och senare i form av en

prototyp för att försäkra att vi förstått varandra. Vi diskuterade även ansvarsfrågor och avgränsningar för uppgiften.

Materialet från dessa möten låg sedan till grund för exjobbsspecifikationen, tidsplanen och kravspecifikationen. Under den följande veckan arbetade vi med kravspecifikationen, som fick skrivas om ett antal gånger, samt inläsning.

När vi enats om kravspecifikationen påbörjades arbetet med prototypen som även den har ändrats och utökats ett flertal gånger, allt eftersom uppdragsgivaren funnit brister och möjligheter. Arbetet med prototypen pågick parallellt med design (konstruktion) och implementation av databasen under sju veckor, varav de tre första veckorna upptogs av enbart prototypen.

Först efter ungefär fem veckor tog vi itu med dokumentationen på allvar. Tidigare hade vi bara skrivit de dokument som beskrivits ovan. Vid den här tidpunkten var alla viktiga beslut fattade om vilket programspråk och vilken databas vi skulle använda, och vi kände att vi hade något konkret att skriva om.

När ungefär halva tiden för examensarbetet gått fick PRTP-gruppen se en version av prototypen och möjlighet att lämna synpunkter på densamma. I samband med detta uppstod nya krav på ändringar, främst vad det gällde utseendet. Dessa ändringar påverkade även vissa SQL-frågor och behov av ytterligare frågor uppstod. När prototypen var godkänd implementerades databasen och SQL-frågorna.

Under de följande tre veckorna arbetade vi uteslutande med rapporten. Vårt mål var att ge en generell beskrivning av de komponenter som vi använt under arbetets gång samt att ta upp något möjligt alternativ på varje område.

När arbete pågått under totalt tolv veckor lämnade vi den första versionen av rapporten till vår handledare för granskning. Rapporten diskuterades vid ett av våra kontinuerliga möten där vi kom fram till att materialet fortfarande var ganska tunt men att vi var inne på rätt spår och skulle jobba vidare som planerat.

De närmaste två veckorna arbetade vi intensivt med servletarna i förhoppning om att hinna klart med arbetet innan slutrapporten skulle färdigställas och lämnas in. Vi stötte dock på små problem gång på gång och var till slut tvungna att sätta ett stoppdatum för konstruktionsarbetet, för att få tid till rapporten. Vi är nära målet och har för avsikt att färdigställa systemet efter att rapporten lämnats för opposition.

8 Slutsatser

Ett av de svåraste momenten var att avgränsa arbetet, då både vi själva och vår uppdragsgivare kommit på tillägg och ändringar i systemet under arbetets gång. De flesta ändringarna har gällt prototypen och därmed systemets underliggande funktionalitet. Trots att arbetet med prototypen var en lång process som tagit ganska mycket tid, har det varit bra att kunna göra förändringar innan implementationen påbörjats.

Även arbetet med implementationen tog längre tid än beräknat. Detta berodde bland annat på att vi inte hade någon större erfarenhet av java sedan tidigare. En hel del tid gick åt till att lära sig att hitta i javas klassbibliotek, som för övrigt är ett mycket bra hjälpmedel. Vi hade nog inte heller insett hur mycket arbete som krävdes vid implementationen.

Eftersom vi inte nått ända fram till slutmålet, kan vi i nuläget inte avgöra hur väl systemet fungerar. Administrationsgränssnittet är i stort sett klart men ytterligare tester återstår. I gränssnittet för sökning återstår fortfarande en del implementering som skall slutföras efter att detta dokument lämnats in.

Vidare kan vi konstatera att böcker är ett trevligt informationsmedium. Vi har lagt ner mycket möda på att söka information om de olika komponenterna som vi har använt oss av på Internet. Det finns en uppsjö av sidor med produktinformation och FAQs men väldigt få där man kan hitta en mer generell beskrivning av funktionaliteten hos komponenterna.

Referenser

[1] http://www.java.sun.com/docs/books/tutorial/servlets/lifecycle/index.html

[2] James Goodwill. *Developing Java Servlets.* Sams Publishing. 1999.

[3] Patrick Naughton, Herbert Schildt. *Java 2: The Complete Reference, Third Edition.* McGraw-Hill Companies (U.S.A.). 1999.

[4] http://www.noyau.com/reference/php3/introduction.html

[5] http://www.mysql.com/Manual_chapter/manual_toc.html

[6] http://SAL.KachinaTech.com/H/1/MYSQL.html

[7] David Axmark, Michael Widenius. MySQL Introduction, Linux Journal issue 67, november 1999, pg 82-89.

[8] http://www2.dgsys.com/~dcasug/sybintro/intro.html

[9] http://www.java.sun.com/docs/books/jdbc/intro.html

[10] http://www.wnet.net/~gturner/newJDBC.html

A Relationsdatamodell

A.1 Mappning av E/R-modellen till relationsdatamodellen.

Entiteter	*Attribut*
Conference	url, c_name, descr, location, con_date, call_date, proc, s_id
Subject	s_id, s_name
Link	url, l_name, descr, date, s_id, tp_id, p_id
Pagetype	tp_id, page
SubSubject	ss_id, ss_name
SubConn	s_id, ss_id
LSConn	url, ss_id

A.2 Beskrivning av de olika relationerna

Conference

Beskrivning: innehåller information om konferenser.

Attribut		
url	Beskrivning:	sökväg/adress till sidor/filer som finns på nätet
	Datatyp:	varchar(100)
	Verifieringsuttryck:	-
	Obligatorisk:	ja
c_name	Beskrivning:	konferensens namn/titel
	Datatyp:	varchar(50)
	Verifieringsuttryck:	-
	Obligatorisk:	ja

descr	Beskrivning:	kort beskrivning av innehåll
	Datatyp:	text
	Verifieringsuttryck:	-
	Obligatorisk:	ja
location	Beskrivning:	ort där konferensen äger rum
	Datatyp:	varchar(50)
	Verifieringsuttryck:	-
	Obligatorisk:	ja
conn_date	Beskrivning:	datum då konferensen äger rum
	Datatyp:	varchar(21)
	Verifieringsuttryck:	-
	Obligatorisk:	ja
call_date	Beskrivning:	sista datum för inlämning av papper
	Datatyp:	date
	Verifieringsuttryck:	-
	Obligatorisk:	ja
proc	Beskrivning:	länk till relaterat material
	Datatyp:	varchar(100)
	Verifieringsuttryck:	-
	Obligatorisk:	nej
s_id	Beskrivning:	id-nummer för ämne
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	url
Primärnyckel:	url
Främmandenycklar:	s_id

Subject

Beskrivning: innehåller alla ämnen

Attribut

s_id	Beskrivning:	id-nummer för ämne
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja
s_name	Beskrivning:	ämne
	Datatyp:	varchar(30)
	Verifieringsuttryck:	-
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	s_id, s_name
Primärnyckel:	s_id
Främmandenycklar:	-

Link

Beskrivning: innehåller information om URLer

Attribut

url	Beskrivning:	sökväg/adress till sidor/filer som finns på nätet
	Datatyp:	varchar(100)
	Verifieringsuttryck:	-
	Obligatorisk:	ja
l_name	Beskrivning:	länkens namn/titel
	Datatyp:	varchar(50)
	Verifieringsuttryck:	-
	Obligatorisk:	ja

descr	Beskrivning:	kort beskrivning av innehåll
	Datatyp:	text
	Verifieringsuttryck:	-
	Obligatorisk:	ja
date	Beskrivning:	datum då URLen läggs in i databasen
	Datatyp:	date
	Verifieringsuttryck:	>0
	Obligatorisk:	ja
s_id	Beskrivning:	id-nummer för ämne
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja
tp_id	Beskrivning:	id-nummer för sidtyp
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja
p_id	Beskrivning:	id-nummer för person
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	url
Primärnyckel:	url
Främmandenycklar:	s_id, tp_id, p_id

Pagetype

Beskrivning: innehåller alla typer av sidor

Attribut

tp_id	Beskrivning:	id-nummer för sidtyp
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja
page	Beskrivning:	typ av sida
	Datatyp:	varchar(20)
	Verifieringsuttryck:	-
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	tp_id, page
Primärnyckel:	tp_id
Främmandenycklar:	-

Person

Beskrivning: innehåller namn på alla som kan uppdatera databasen

Attribut

p_id	Beskrivning:	id-nummer för person
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja
name	Beskrivning:	namn
	Datatyp:	varchar(30)
	Verifieringsuttryck:	-
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	p_id, name
Primärnyckel:	p_id
Främmandenycklar:	-

SubSubject

Beskrivning: innehåller alla subämnen

Attribut

ss_id	Beskrivning:	id-nummer för subämne
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja
ss_name	Beskrivning:	subämne
	Datatyp:	varchar(30)
	Verifieringsuttryck:	-
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	ss_id, ss_name
Primärnyckel:	ss_id
Främmandenycklar:	-

SubConn

Beskrivning: kopplar samman tabellerna Subject och SubSubject

Attribut

s_id	Beskrivning:	id-nummer för ämne
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja

ss_id	Beskrivning:	id-nummer för subämne
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	(s_id, ss_id)
Primärnyckel:	(s_id, ss_id)
Främmandenycklar:	s_id, ss_id

LSConn

Beskrivning: kopplar samman tabellerna Link och SubSubject

Attribut

url	Beskrivning:	sökväg/adress till sidor/filer som finns på nätet
	Datatyp:	varchar(100)
	Verifieringsuttryck:	-
	Obligatorisk:	ja

ss_id	Beskrivning:	id-nummer för subämne
	Datatyp:	int(10)
	Verifieringsuttryck:	>0
	Obligatorisk:	ja

Nycklar

Kandidatnycklar:	(url, ss_id)
Primärnyckel:	(url, ss_id)
Främmandenycklar:	url, ss_id

B Källkod för servleten addUrl.java

```
import java.io.*;
import java.util.*;
import java.lang.*;

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

import java.sql.*;

public
class  addUrl extends HttpServlet {


    PrintWriter out;
    String url;
    String title;
    String page;
    String subject;
    String s_subject;
    String submitter;
    String descr;
    String s_id;
    String p_id;
    String tp_id;
    String datum;
    boolean newSubSubject = false;

/*****************************************************************************
*  Metod som skapar och initierar de resurser som behövs för att behandla requests.
*****************************************************************************/
    public void init(ServletConfig config) throws ServletException
    {
            super.init(config);
    }

/*****************************************************************************
*  Metod som tar hand om get-request.
*****************************************************************************/
    public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
            res.setContentType("text/html");
            out = res.getWriter ();
        mkForm();
    }

/*****************************************************************************
*  Metod som tar hand om post-request.
*****************************************************************************/
    public void doPost  (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
            res.setContentType("text/html");
            out = res.getWriter ();
            Enumeration e = req.getParameterNames();
            String name = null;

            while(e.hasMoreElements()){
               name = (String)e.nextElement();

               if(name.equals("add")) {
```

```
                url = req.getParameter("url");
                title = req.getParameter("title");
                page = req.getParameter("pagetype");
                subject = req.getParameter("subject");
               s_subject = req.getParameter("subSubjectFromSelect"); //select-box

                if(s_subject.equals("Nothing"))
                  s_subject = "";

                else if(s_subject.equals("New")){
                  s_subject = req.getParameter("subSubjectFromTextfield");
                  //textfield

                  if(!(s_subject.equals("")))
                        newSubSubject = true;
                }

                descr = req.getParameter("description");
                submitter = req.getParameter("submitterName");
                submitter = submitter.replace('_', ' ');

                if(url.equals("")|| title.equals("") || descr.equals(""))
                       out.println("<h2>Errormessage: Please go<a
                       href=http://dv1.cs.kau.se/linkdbdevel/addUrl>"+
                       " back </a> and enter empty fields."+e+"</h2>");
                else
                       addPrev();
              }

              if(name.equals("ok")){
                add();
                break;
              }
            }
    }

/*****************************************************************************
*  Metod som lägger till ny post i databasen.
*****************************************************************************/
    public void add()
    {
       boolean exists = false;
       try {
          Class.forName("org.gjt.mm.mysql.Driver").newInstance();
       }
       catch (Exception E) {
           out.println("Unable to load driver.");
           E.printStackTrace();
       }
       try {
          Connection dbc =
          DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
          root");
          Statement stmt = dbc.createStatement();

          stmt.execute("select url from link");
          ResultSet rs = stmt.getResultSet();

              while (rs.next()) {
                  if(url.equals(rs.getString("url")))
                  {
                         out.println("<html><head>"+
                         "<H2>The url allredy exists!</H2>"+
                         "</head></html>");
                         exists = true;
                         break;
              }
          }
```

```
if(exists == false)
{
  get_s_id();
  get_p_id();
  get_tp_id();

  Calendar calendar = new GregorianCalendar();
  int year = calendar.get(calendar.YEAR);
  int month = calendar.get(calendar.MONTH)+1;
  int day = calendar.get(calendar.DATE);
  datum = year+"-"+month+"-"+day;

  stmt.execute("insert into link values('"+url+"', '"+title+"',
  '"+descr+"', '"+datum+"', '"+s_id+"', '"+tp_id+"', '"+p_id+"')");

  if(newSubSubject == true){
          StringTokenizer st = new StringTokenizer(s_subject, ";");

          while(st.hasMoreElements()){
              String token = (String)st.nextElement();

              stmt.execute("select ss_name from subSubject where
             ss_name='"+token+"'");
             //kollar om subsubjectet finns i db:n

              rs = stmt.getResultSet();
              if(rs.getString("ss_name") == null)
                stmt.execute("insert into subSubject values('',
               '"+token+"')");

              stmt.execute("select ss_id from subSubject where
             ss_name='"+token+"'");
              rs = stmt.getResultSet();
              String ss_id = rs.getString("ss_id");

              if(!(ss_id.equals(""))){
                stmt.execute("select ss_id from LSConn where
               ss_id='"+ss_id+"' and url='"+url+"'");
                rs = stmt.getResultSet();

                if(rs.getString("ss_id") == null)
                  stmt.execute("insert into LSConn values('"+url+"',
                 '"+ss_id+"')");

                stmt.execute("select ss_id from subConn where
               ss_id='"+ss_id+"' and s_id='"+s_id+"'");
                rs = stmt.getResultSet();

                if(rs.getString("ss_id") == null)

                  stmt.execute("insert into subConn
                 values('"+s_id+"', '"+ss_id+"')");
             }
  }
  }
  else{
    if(!(s_subject.equals(""))){
      stmt.execute("select ss_id from subSubject where
     ss_name='"+s_subject+"'");
      rs = stmt.getResultSet();
      String ss_id = rs.getString("ss_id");
      stmt.execute("insert into LSConn values('"+url+"',
     '"+ss_id+"')");
      stmt.execute("select ss_id from subConn where ss_id='"+ss_id+"'
     and s_id='"+s_id+"'");
      rs = stmt.getResultSet();
      if(rs.getString("ss_id") == null)
```

```
                    stmt.execute("insert into subConn values('"+s_id+"',
                    '"+ss_id+"')");
                  }
                }
                newSubSubject = false;
                out.println("Operation succeded!");
              }
              rs.close();
           stmt.close();
               dbc.close();
       }
       catch (SQLException E) {
          out.println("SQLException: " + E.getMessage());
          out.println("SQLState:     " + E.getSQLState());
          out.println("VendorError:  " + E.getErrorCode());
       }
    }

/*************************************************************************
* Metod som hämtar s_id för angivet ämne .
*************************************************************************/
    public void get_s_id()
    {
       try {
          Class.forName("org.gjt.mm.mysql.Driver").newInstance();
       }
       catch (Exception E) {
           out.println("Unable to load driver.");
           E.printStackTrace();
       }
       try {
          Connection dbc =
          DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
          root");
               Statement stmt = dbc.createStatement();

          stmt.execute("select s_id from subject where s_name='"+subject+"'");
          ResultSet rs = stmt.getResultSet();
               s_id = rs.getString("s_id");

          rs.close();
          stmt.close();
               dbc.close();
       }
       catch (SQLException E) {
          out.println("SQLException: " + E.getMessage());
          out.println("SQLState:     " + E.getSQLState());
          out.println("VendorError:  " + E.getErrorCode());
       }
    }



/*************************************************************************
*  Metod som hämtar p_id för angiven person.
*************************************************************************/
    public void get_p_id()
    {
       try {
          Class.forName("org.gjt.mm.mysql.Driver").newInstance();
       }
       catch (Exception E) {
           out.println("Unable to load driver.");
           E.printStackTrace();
       }
       try {
          Connection dbc =
          DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
```

```
            root");
            Statement stmt = dbc.createStatement();

            stmt.execute("select p_id from person where p_name='"+submitter+"'");
            ResultSet rs = stmt.getResultSet();
                p_id = rs.getString("p_id");

            rs.close();
            stmt.close();
                dbc.close();
        }
        catch (SQLException E) {
            out.println("SQLException: " + E.getMessage());
            out.println("SQLState:     " + E.getSQLState());
            out.println("VendorError:  " + E.getErrorCode());
        }
    }

/*************************************************************************
*  Metod som hämtar tp_id för angiven sidtyp.
*************************************************************************/
    public void get_tp_id()
    {
        try {
            Class.forName("org.gjt.mm.mysql.Driver").newInstance();
        }
        catch (Exception E) {
             out.println("Unable to load driver.");
             E.printStackTrace();
        }
        try {
            Connection dbc =
            DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
            root");
            Statement stmt = dbc.createStatement();

            stmt.execute("select tp_id from pagetype where page='"+page+"'");
            ResultSet rs = stmt.getResultSet();
            tp_id = rs.getString("tp_id");

            rs.close();
            stmt.close();
            dbc.close();
        }
        catch (SQLException E) {
            out.println("SQLException: " + E.getMessage());
            out.println("SQLState:     " + E.getSQLState());
            out.println("VendorError:  " + E.getErrorCode());
        }
    }

/*************************************************************************
* Metod som hämtar alla pagetypes från databasen och skriver in dem i select-
* boxen.
*************************************************************************/
    public void getTypeOfPage()
    {
        try {
            Class.forName("org.gjt.mm.mysql.Driver").newInstance();
        }
        catch (Exception E) {
             out.println("Unable to load driver.");
             E.printStackTrace();
        }
        try {
            Connection dbc =
            DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
            root");
```

```
         Statement stmt = dbc.createStatement();

         stmt.execute("select page from pagetype order by page");
         ResultSet rs = stmt.getResultSet();

         while(rs.next()){
            String pg = rs.getString("page");
            out.println("<option value="+pg+" selected>"+pg+"</option>");
         }
         rs.close();
         stmt.close();
         dbc.close();
      }
      catch (SQLException E) {
         out.println("SQLException: " + E.getMessage());
         out.println("SQLState:     " + E.getSQLState());
         out.println("VendorError:  " + E.getErrorCode());
      }
   }

/*************************************************************************
*  Metod som hämtar alla subjects från databasen och skriver in dem i select-boxen.
*************************************************************************/
   public void getSubject()
   {
      try {
         Class.forName("org.gjt.mm.mysql.Driver").newInstance();
      }
      catch (Exception E) {
          out.println("Unable to load driver.");
          E.printStackTrace();
      }
      try {
         Connection dbc =
         DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
         root");
         Statement stmt = dbc.createStatement();

         stmt.execute("select s_name from subject order by s_name");
         ResultSet rs = stmt.getResultSet();

         while(rs.next()){
            String sub = rs.getString("s_name");
            out.println("<option value="+sub+" selected>"+sub+"</option>");
         }
         rs.close();
         stmt.close();
         dbc.close();
      }
      catch (SQLException E) {
         out.println("SQLException: " + E.getMessage());
         out.println("SQLState:     " + E.getSQLState());
         out.println("VendorError:  " + E.getErrorCode());
      }
   }

/*************************************************************************
* Metod som hämtar alla subsubjects från databasen och skriver in dem i select-
* boxen.
*************************************************************************/
   public void getSubSubject()
   {
      try {
         Class.forName("org.gjt.mm.mysql.Driver").newInstance();
            }
            catch (Exception E) {
             out.println("Unable to load driver.");
             E.printStackTrace();
```

```
        }
        try {
           Connection dbc =
           DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
           root");
           Statement stmt = dbc.createStatement();

           stmt.execute("select ss_name from subSubject order by ss_name");
           ResultSet rs = stmt.getResultSet();

           String sub = "New";
           out.println("<option value="+sub+" selected>"+sub+"</option>");
           sub = "Nothing";
           out.println("<option value="+sub+" selected>"+sub+"</option>");

           while(rs.next()){
             sub = rs.getString("ss_name");
             out.println("<option value="+sub+" selected>"+sub+"</option>");
           }
           rs.close();
           stmt.close();
           dbc.close();
        }
        catch (SQLException E) {
           out.println("SQLException: " + E.getMessage());
           out.println("SQLState:     " + E.getSQLState());
           out.println("VendorError:  " + E.getErrorCode());
        }
    }

/*****************************************************************************
* Metod som hämtar alla submitters från databasen och skriver in dem i select-
* boxen.
*****************************************************************************/
    public void getSubmitter()
    {
        try {
           Class.forName("org.gjt.mm.mysql.Driver").newInstance();
        }
        catch (Exception E) {
            out.println("Unable to load driver.");
            E.printStackTrace();
        }
        try {
           Connection dbc =
           DriverManager.getConnection("jdbc:mysql://localhost/linkdbdevel?user=
           root");
           Statement stmt = dbc.createStatement();

           stmt.execute("select p_name from person order by p_name");
           ResultSet rs = stmt.getResultSet();

           while(rs.next()){
              String subm = rs.getString("p_name");
              String val = rs.getString("p_name");
             val = val.replace(' ', '_');
              out.println("<option value="+val+" selected>"+subm+"</option>");
           }
           rs.close();
           stmt.close();
           dbc.close();
        }
        catch (SQLException E) {
           out.println("SQLException: " + E.getMessage());
           out.println("SQLState:     " + E.getSQLState());
           out.println("VendorError:  " + E.getErrorCode());
        }
    }
```

```
/************************************************************************
* Metod som skriver ut formuläret.
************************************************************************/
    public void mkForm()
    {
            out.println("<HTML>"+
            "<HEAD>"+
            "<TITLE>Add new URL</TITLE>"+
            "</HEAD>"+
            "<BODY TEXT=#000000 BGCOLOR=#FFFFFF>"+
            "<H2>Add a new URL</H2>"+
            "Enter all fields (sub-subject is optionary) and press <font
            size=+1>Add</font> to submitt."+
            "<BR><FORM ACTION=http://dv1.cs.kau.se/linkdbdevel/addUrl
            method=post><table width=60%>"+
            "<tr><td>URL: <td><INPUT type=text name=url size=50>"+

            "<tr><td>Name/Headline:<td><INPUT type=text name=title size=40>"+
            "<tr><td>Type of page: <td><select name=pagetype>");
            getTypeOfPage();

            out.println("</select>");

            out.println("<tr><td>Subject: <td><select name=subject>");
            getSubject();
            out.println("</select>");

            out.println("<tr><td>Sub-subject: <td><select
            name=subSubjectFromSelect>");
            getSubSubject();

            out.println("</select>"+
            "<INPUT type=text name=subSubjectFromTextfield size=50>"+

            "<tr><td>Description: <td><TEXTAREA name=description COLS=60
            ROWS=4></TEXTAREA>"+

            "<tr><td>Submitted by:<td><select name=submitterName>");
            getSubmitter();
            out.println("</select>"+
            "<P><tr><td><INPUT type=submit name=add value=Add><INPUT type=reset>"+

            "</table></FORM>"+
            "</BODY>"+
            "</HTML>");
    }

/************************************************************************
*  Metod som skriver ut en förhandsvisning av resultatet.
************************************************************************/
    public void addPrev()
    {
            out.println("<HTML><HEAD>"+
            "<TITLE>Result of add URL</TITLE></HEAD>"+
            "<BODY TEXT=#000000 BGCOLOR=#FFFFFF>"+
            "<FORM action=http://dv1.cs.kau.se/linkdbdevel/addUrl method=post>"+

            "<H3>Preveiw of URL</H3>"+
            "Press <FONT SIZE=+1>OK</FONT> to confirm add."+
            "<BR>Press <FONT SIZE=+1> browser Back button</FONT> if you want to
            make changes."+
            "<P><table width=30%><!--begin-->"+

            "<tr><td>URL: <td>"+url+
            "<tr><td>Name/Hedline: <td>"+title+

            "<tr><td>Type of page: <td>"+page+
```

```
                "<tr><td>Subject: <td>"+subject+
                "<tr><td>Sub-subject: <td>"+s_subject+
                "<tr><td>Description: <td>"+descr+

                "<tr><td>Submitted by: <td>"+submitter+

                "<tr><td><INPUT type=submit name=ok value=OK>"+

                "</table></BODY>"+
                "</HTML>");
    }
}
```