Datavetenskap

Karin Johansson

Margaretha Kindberg

Design av en kursdatabas samt utveckling av ett författargränssnitt

---

Examensarbete, C-nivå

Pubnum 23

Design av en kursdatabas samt utveckling av ett författargränssnitt

Karin Johansson

Margaretha Kindberg

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 mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Karin Johansson Margaretha Kindberg

Godkänd, 2000-05-30

Handledare: Hua Shu

Examinator: Stefan Lindskog

Sammanfattning

I denna C-uppsats redovisas ett examensarbete som utförts inom det datavetenskapliga programmet vid Karlstads universitet. Arbetet har utförts under vårterminen 2000 på Tech School Svenska AB i Karlstad. Det är ett företag som producerar interaktiv utbildning på Internet. Företaget gav uppdraget att designa en relationsdatabas som ska innehålla det kursmaterial som ska publiceras på elevernas webbgränssnitt. Som en del i arbetet ingick även att utveckla ett webbaserat författargränssnitt till databasen.

Arbetet inleds med en undersökning där de krav och önskemål som finns på den nya databasen och det nya författargränssnittet fastställs. Stor vikt har lagts vid databasdesignen som har utförts i de tre metodstegen konceptuell, logisk och fysisk design. Databasen visade sig bli relativt stor och komplex vilket har medfört att implementering av författargränssnittet har avgränsats till vissa delar.

Arbetet med författargränssnittet har inneburit att designa ett webbgränssnitt som databasen har publicerats på. Webbpubliceringen har skett med ASP-teknik, vilket är en intressant och modern teknik.

I uppsatsen redovisas arbetets alla faser samt de resultat som nåtts. Arbetet har lett till en genomtänkt databasdesign som är väl dokumenterad. Databasen är implementerad i MS SQLserver. Med hjälp av ett webbaserat författargränssnitt kan kursmaterial läggas in i databasen.

A Design of a Courseware Database and an Author Interface

Abstract

This report describes a Bachelor's Project by the Department of Computer Science at Karlstad University. The project took place during the spring term 2000 and was located at Tech School AB in Karlstad. Tech School AB produces interactive education on the Internet.

The goal of the project is to design a relational database for the course material, which should be published on the web interface for students and to develop an interface to the database for authors. The work starts with an analysis of requirements for the database. The database design has been the most important part of the work and was done in three steps: conceptual, logical and physical design. The project has among other things led to a carefully designed database. The database turned out to be relatively complex, which resulted in a limited implementation of the author interface. The work with the author interface consists of publishing the database on the web interface for authors. The database is implemented using MS SQLserver. The web publishing has been carried out by using ASP, which is an interesting and modern technology. The phases and results of the Bachelor's Project, as well as a detailed documentation of the database, are described in this report.

Tack!

Först av allt vill vi tacka Tommy Grönbeck, vice VD vid Tech School AB, som gjort det möjligt för oss att utföra detta givande och intressanta examensarbete.

Varmt tack riktas till vår tekniske handledare Thomas Berglund, vid Tech School AB, för teknisk support och hjälp under arbetets gång.

Vidare vill vi tacka vår handledare Hua Shu vid Universitetet i Karlstad för värdefull vägledning och stöd.

Slutligen vill vi tacka Anna Hederstedt, Magnus Bohman, Kjell-Owe Pettersson och Fredrik Hellström för ett gott och konstruktivt samarbete.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

I denna C-uppsats redovisar undertecknade sitt examensarbete på dataingenjörsutbildningen 120p vid Karlstads universitet. Examensarbetet har utförts på företaget Tech School AB som är lokaliserat på Karolinen i Karlstad. Företaget gav uppdraget att designa en relationsdatabas. Arbetet inleddes med att undersöka vilka krav och förväntningar användarna hade på databasen. Detta resulterade i ett underlag till databasdesignen. Den har utförts i tre faser som innebär konceptuell, logisk samt fysisk design. Det ingick även i uppgiften att utveckla ett författargränssnitt som tillåter dynamik med databasen. Eftersom databasdesign och webbteknik är mycket intressanta områden har detta arbete varit både engagerande och givande.

1.1 Bakgrund

Tech School AB är ett nystartat företag som tillhandahåller interaktiv utbildning via Internet. Det studiematerial som visas på elevernas webbgränssnitt hämtas ur företagets databas. Materialet läggs in i databasen via ett webbaserat författargränssnitt. Systemet anses vara otillräckligt när det gäller flexibilitet och utvecklingsmöjligheter.

1.2 Uppdraget

Att det befintliga systemet är otillräckligt beror till stor del på en begränsad databas. Företaget gav därför uppdraget att designa en ny relationsdatabas som uppfyller vissa krav. Den bör stödja kursförfattarnas önskemål och krav på författargränssnittet. Det betonas att designen ska vara väl genomtänkt innan implementering och att databasen ska kunna växa med företaget. Till databasen ska det byggas ett webbgränssnitt som med underliggande teknik kan hantera kommunikation med databasen.

1.3 Arbetets faser

Det arbete som beskrivs i denna rapport består huvudsakligen av fyra arbetssteg:

- Framtagande av underlag
- Databasdesign
- Design av webbgränssnitt
- Webbpublicering, dvs sammankoppling mellan databas och webbgränssnitt

Stor vikt har lagts vid att designa en väl fungerande databas. Vid framtagande av webbgränssnitt och vid webbpublicering har det varit nödvändigt att göra avgränsningar pga arbetets storlek. Under arbetets gång har information inhämtats i relevant litteratur, på Internet samt från berörda handledare.

2 Framtagande av underlag för databasdesign och gränssnittsdesign

Insamling av information från teknisk handledare och kursförfattare på Tech School AB har gett en god överblick över den information som systemet ska behandla. Kursförfattarna är de personer som lägger in studiematerial i databasen. När första kontakten togs hade de redan arbetat en tid med att genom ett författargränssnitt lägga in material i den befintliga databasen. Denna erfarenhet har varit en stor resurs vid framtagande av underlag.

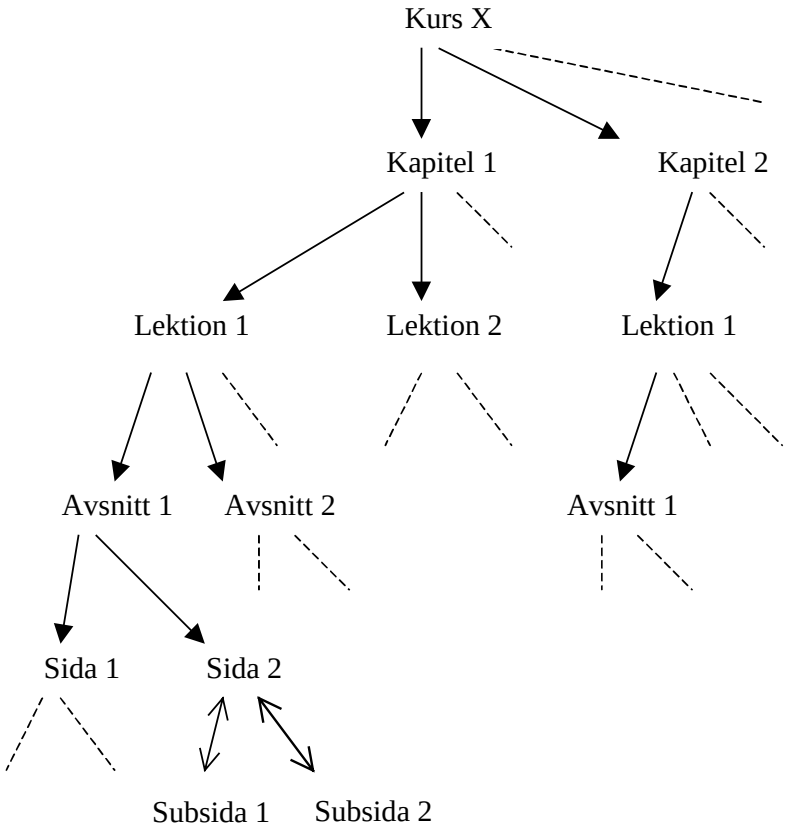
Under arbetets gång har god kommunikation med teknisk handledare och kursförfattare resulterat i ett konstruktivt samarbete. Förutom diskussioner har även kursförfattarnas arbete med att lägga in material i den befintliga databasen studerats. Deltagande i företagets återkommande projektmöten med alla projektdeltagare har gett inblick i hela arbetet kring kurserna.

2.1 Underlag till databas

Databasen ska innehålla utbildningsmaterial samt information om hur detta ska visas på elevernas webbgränssnitt.

2.1.1 Studiematerialets uppbyggnad och struktur

Ett första steg i undersökningen kom att bli att analysera och beskriva studiematerialets innehåll och uppbyggnad. Det finns en generell struktur som är lika för alla kurser som företaget publicerar på webben. Den påminner om strukturen i en vanlig bok. Kurserna indelas i alltmer detaljerade delar. En kurs består av flera kapitel, ett kapitel består av flera lektioner osv med indelningar i avsnitt och sida. Till dessa olika nivåer finns det information som tex introduktion eller sammanfattning. Det är på sidorna som det huvudsakliga kursinnehållet visas. Till sidorna kan det finnas subsidor som ger en fördjupning i ämnet som visas på respektive sida (se Figur 2.1).



Figur 2.1: Kursmaterialets struktur

2.1.2 Resultat av undersökning om databas

Följande förväntningar på den nya databasen konstateras som ett resultat av undersökningen:

- Databasen ska upprätthålla den struktur som kursmaterialet är uppbyggt efter.
- Det ska finnas möjlighet att lagra övergripande information om kurser, kapitel, lektioner, avsnitt och sidor.
- Kursförfattaren ska ges goda möjligheter att välja och kombinera sätt att presentera materialet på och det är viktigt att databasen stödjer denna funktionalitet.
- Databasen ska ge förutsättningar för att kurserna kan presenteras på ett sammanhängande och lättnavigerat sätt på elevgränssnittet.
- Kurserna ser olika ut och har olika innehåll. Det bör därför finnas en möjlighet att tillföra nya presentationssätt till databasen.

- På lite längre sikt är det viktigt att databasen kan ”växa” på ett tillfredsställande sätt. Företaget kommer i snabb takt att utöka antalet kurser och allt kursmaterial ska kunna läggas in i databasen utan att dess grundstruktur påverkas.
- Databasen ska designas med hänsyn till prestanda.
- Det ska finnas möjlighet att lägga in, ta bort, uppdatera, ta fram information ur kursdatabasen.

2.2 Underlag till författargränssnitt

Eleverna kan i nuvarande elevgränssnitt navigera i kursmaterialet genom att klicka på ett index som visas på skärmen. På vänstra sidan av skärmbilden, som här benämns *vänstersida*, visas kursmaterialet. Varje enskild vänstersida innehåller ett antal komponenter, där vissa ska ha ett förutbestämt utseende.

Det är kursförfattaren som i författargränssnittet konstruerar utseendet på elevernas vänstersida. Med utgångspunkt från hur befintligt elevgränssnitt är utformat utreds vilka funktioner som är viktiga att skapa i det nya författargränssnittet.

2.2.1 Resultat av undersökning om författargränssnitt

Undersökningen resulterar i följande punkter:

- Vid skapandet av en vänstersida ska kursförfattaren i författargränssnittet kunna ange hur materialet ska presenteras på elevgränssnittet. Det är viktigt att eleven får kursinnehållet tydligt och pedagogiskt presenterat och därför ska kursförfattaren ges goda möjligheter att välja och kombinera sätt att presentera materialet på.
- Återkommande moment i författarens arbete med att lägga in kursmaterial i databasen ska kunna utföras på ett enkelt och tidsbesparande sätt.
- Det bör åskådliggöras i författargränssnittet var i kursstrukturen författaren befinner sig.

2.3 Sammanfattning av undersökningen

Eftersom det vid tiden för undersökningen fanns ett fungerande författargränssnitt var det naturligt att undersöka hur det var konstruerat och vad som kunde fungera bättre. Enligt författarna var det befintliga författargränssnittet för begränsat. Det tillät inte den flexibilitet i möjligheter att presentera kursmaterial på som författarna önskade.

En annan nackdel var att författaren i vissa fall själv var tvungen att hålla reda på var i kursstrukturen han/hon befann sig eftersom det inte visades på skärmen. Begränsningarna ansågs till stor del bero på konstruktionen på den befintliga databasen.

Den befintliga databasen visade sig ha även andra begränsningar. Den är svår att bygga ut om man vill utöka antalet presentationsmöjligheter. Den är även konstruerad så att det skapas många nullvärden, dvs tomma minnesplatser, när kursmaterial läggs in.

En fördel med den befintliga databasen är att den är enkel med få tabeller. Den lösning på databasdesign som presenteras i denna rapport ger en mycket mer komplex lösning med många tabeller och relationer dem emellan. Det är naturligt att ett flexibelt system med många valmöjligheter kräver större komplexitet i databasdesignen.

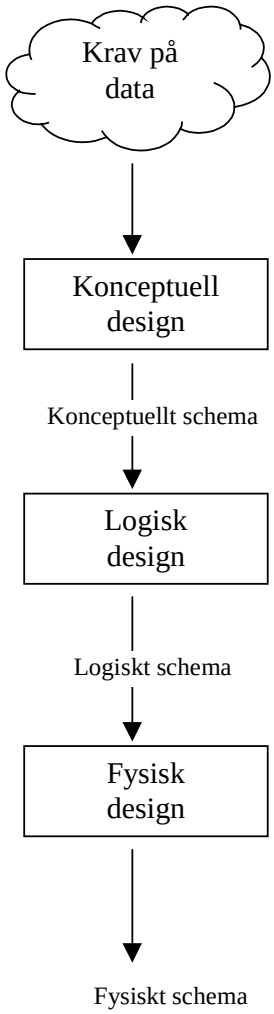
3 Databasdesign

Databasdesign är den process där organisationen av databasen bestäms, inklusive dess struktur och innehåll. Databasdesign spelar en central roll i informationssökningshanteringen i de flesta organisationer och det är ofta en komplex uppgift som involverar beslut på olika nivåer. Komplexiteten hanteras mer överskådligt om man delar in uppgiften i mindre delar och försöker att lösa varje enskild del med specifika metoder och tekniker.

Det finns två typer av angreppssätt vid utveckling av ett informationssystem, datastyrt angreppssätt och funktionsstyrt angreppssätt. Ett datastyrt angreppssätt innebär att fokus sätts på data och dess egenskaper. Först designas databasen och sedan tillämpningarna (applikationerna) som använder databasen. I ett funktionsdrivet angreppssätt fokuserar man på applikationerna snarare än data [1].

Ett datastyrt angreppssätt har använts för att designa kursdatabasen. Databasdesign vid ett datastyrt angreppssätt delas normalt in i tre faser (se Figur 3.1).

- Den första fasen, som kallas konceptuell design, resulterar i en abstraktion på hög nivå som ska representera verkligheten.
- Den andra fasen som kallas logisk design översätter den konceptuella representationen till specifikationer som kan implementeras och hanteras av ett datorsystem.
- Den tredje fasen kallas fysisk design och bestämmer den fysiska lagringsstrukturen och åtkomstmetoder som behövs för effektiv åtkomst av innehållet i databasen från sekundärminnen.



Figur 3.1: Databasdesignens tre faser

Den konceptuella och logiska designen kan utföras oberoende av valet av specifikt databashanteringssystem, tex MS SQLserver.

3.1 Konceptuell design

Syftet med den konceptuella designen är att beskriva informationsinnehållet i den aktuella verksamheten man ska bygga ett informationssystem för. Konceptuell design utgår från det framtagna underlaget och resulterar i ett konceptuellt schema. Ett konceptuellt schema är en beskrivning av databasens struktur gjord på en hög abstraktionsnivå. Vid framtagandet av det konceptuella schemat har ett objektorienterat angreppssätt använts.

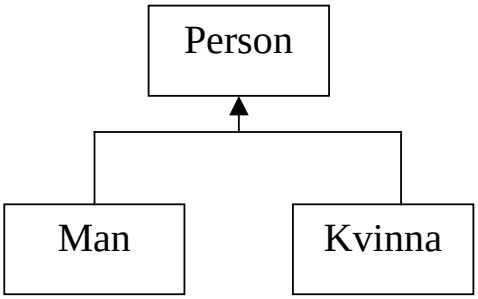
Det generella objektorienterade synsättet tillhandahåller ett sätt att organisera/strukturera verkligheten konceptuellt. Organisationen baseras på olika typer av företeelser. De kallas här objekttyper och förekommer i en avgränsad del av verkligheten. Till dessa objekttyper kan sedan attribut kopplas, operationer göras mm. Viktiga begrepp inom objektorientering är klassificering, generalisering samt aggregering:

Klassificering:

En klassificering innebär att en viss objekttyp tillskrivs ett specifikt objekt. När ett objekt är klassificerat är det en instans av en speciell objekttyp.

Generalisering:

En entitet E är en generalisering av en grupp entiteter E_1, E_2,…E_n om varje objekt av dessa entiteter också är ett objekt av entiteten E. Exempel på denna struktur är entiteten ”person” som är en generalisering av entiteterna ”kvinna” och ”man” (se Figur 3.2).



Figur 3.2: Exempel på generalisering

Denna struktur är mycket användbar i objektorienterade sammanhang på grund av fundamentala arvsegenskaper. I en generalisering ärvs alla egenskaper och metoder som existerar för den generella klassen av alla underklasser.

Aggregering:

En aggregering definierar en ny klass från en mängd andra klasser vilka representerar den nya klassens ingående delar.

3.1.1 ER-modellen

En konceptuell modell är ett språk som används för att beskriva den del av verkligheten som är intresserant. Den modell som främst har använts för att beskriva kursdatabasens konceptuella schema heter Entity-Relationship (ER) modellen [1]. Den grafiska representationen av ER-modellen visas i bilaga A.1.

De komponenter som ingår i ER modellen är:

- *entiteter*, vilka representerar klasser av föremål och företeelser i verkligheten som man vill lagra information om. Exempel på entiteter i en databas över personer är person, man, kvinna samt anställd. Entiteter representeras grafiskt av en rektangel.
- *relationer*, som kan ses som en association mellan entiteter. Exempel på en relation är ”bor i” mellan entiteterna stad och person. Relationer representeras grafiskt av en romb.
- *attribut*, vilka beskriver eller karaktäriserar gemensamma egenskaper för en entitet eller en relation. Exempel på attribut till entiteten person är namn, personnummer samt yrke.
- *sammansatta attribut*, (composite attribute), vilka är grupper av attribut som har samhörighet vad gäller mening eller användning. Det sammansatta attributet adress betecknar attributgruppen gata, postnummer samt postadress.
- identifierare. Med hjälp av en entitets identifierare kan man unikt bestämma alla förekomster av entiteten. Identifierare kallas också kandidatnycklar.
- *generaliseringshierarkier* mellan entiteter. En entitet E är en generalisation av en grupp entiteter E_1, E_2, E_n. om varje objekt av dessa entiteter också är ett objekt av entiteten E.

ER-modellen introducerades 1976 av Peter Chen och har blivit alltmer populär. Originalversionen innehöll endast begrepp som entiteter, relationer och attribut. Senare utvecklades modellen och komponenter som sammansatta attribut och generalisationshierarkier lades till. Vidareutveckling av modellen kan ses som naturligt och nödvändigt då det objektorienterade synsättet alltmer används i systemutveckling. Idag behövs det konceptuella modeller som är så uttrycksfulla att de kan användas till att konceptuellt beskriva viktiga begrepp inom objektorientering.

3.1.2 Framtagande av databasens konceptuella schema

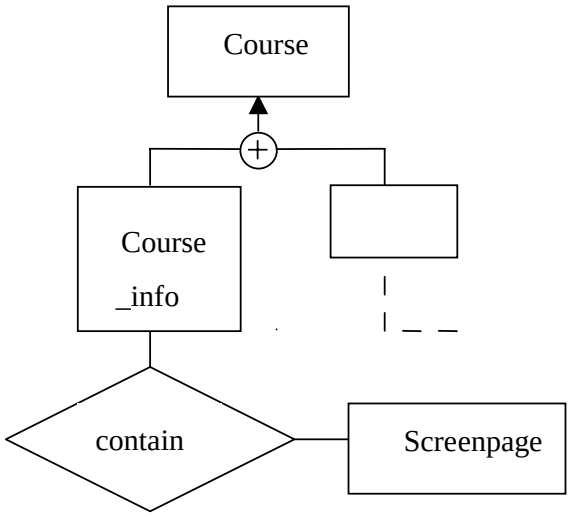
Med det framtagna underlaget som utgångspunkt har föremål och företeelser som det ska lagras information om specificerats. Dessa företeelser kallas objekt. Objekt med samma egenskaper (attribut) bildar en objekttyp. Ett annat ord för objekttyp är benämningen klass.

Det var lätt att hitta objekttyperna kurs, kapitel, lektion, avsnitt, sida samt subsida. Ytterligare företeelser kunde observeras. Vissa av dessa har gemensamma egenskaper och samband. De delades in i två tydliga grupper som kallas komponent och subkomponent.

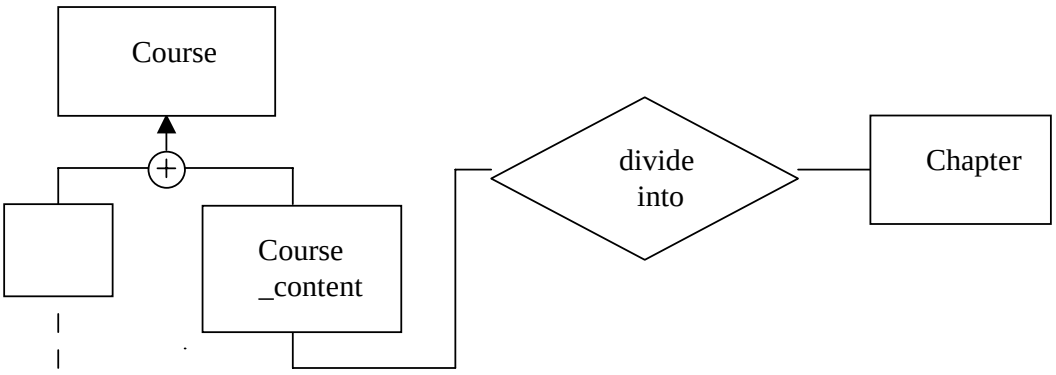
De entiteter som formar kursmaterialets struktur i det konceptuella schemat är kurs (*Course*), kapitel (*Chapter*), lektion (*Lesson*), avsnitt (*Section*), sida (*Page*) samt subsida (*Subpage*).

Entiteterna *Course, Chapter, Lesson, Section* samt *Page* består av två delar:

- En informationsdel som kan innehålla introduktion, sammanfattning samt annan viktig information som bör kopplas till respektive entitet. Entiter som beskriver detta förfarande har *_info* som prefix i det konceptuella schemat. Exempel på detta visas i Figur 3.3. I detta fall finns en direkt relation till entiteten *Screenpage*.
- En innehållsdel som åskådliggör kursmaterialets struktur, en kurs delas in i kapitel, varje kapitel delas in i lektioner osv. Entiteter som beskriver detta förfarande har *_content* som prefix i det konceptuella schemat. Exempel på detta visas i Figur 3.4.

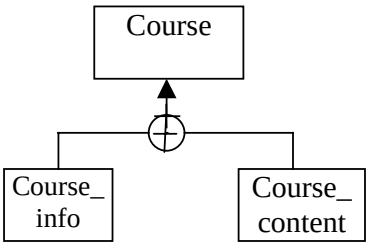


Figur 3.3: Informationsdelen i Course



Figur 3.4: Innehållsdelen i Course

I konceptuell databasdesign benämns ovanstående struktur som en aggregering. En aggregering definierar en ny klass från en mängd andra klasser som representerar den nya klassens ingående delar. I det fallet blir *Course* summan (aggregeringen) av klasserna *Course_info* och *Course_content* (se Figur 3.5). Omvänt blir det att *Course_info* är en del av (is a part of) *Course*.



Figur 3.5: Aggregering

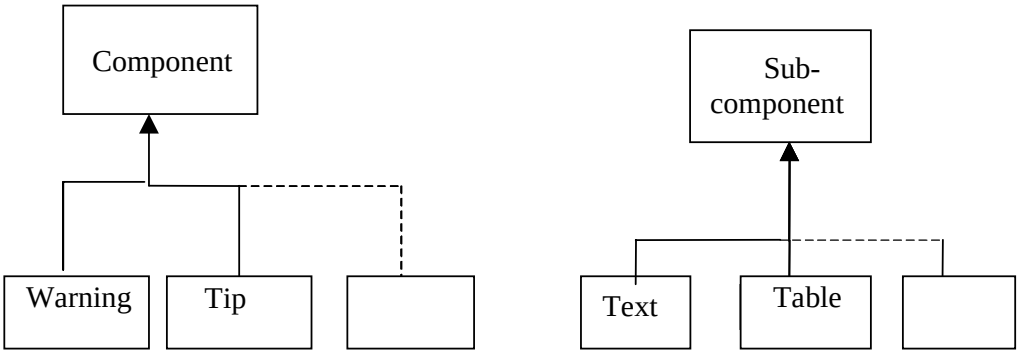
Kursförfattaren arbetar med att lägga in kursmaterial som ska visas på enskilda vänstersidor i elevgränssnittet (vänstra delen i webbgränssnittet). Företeelsen vänstersida ses som en entitet och denna benämns *Screenpage* i det konceptuella schemat. En vänstersida byggs upp av ett antal fördefinierade komponenter som var och en blir en egen entitet i det konceptuella schemat.

De fördefinierade komponenterna har den gemensamma egenskapen att de har en specifik layout. De benämns *NotaBene, Tip, Exercise, Warning, Hidden_answer, Question, No_frame, Download, Read_more, Example, Code* och *Extras* i det konceptuella schemat. Komponenten *No_frame* används när subkomponenter ska visas på en webbsida utan någon synlig komponent-layout. Komponenten *Extras* ger utrymme för att skapa speciella layouter som inte redan finns representerade i databasen.

I konceptuell databasdesign finns en struktur som kallas arv. Alla komponenterna har en gemensam egenskap, nämligen att de utgör en specifik layout. De kan därför sägas ärva egenskapen av en ”superklass”. Denna klass kallas *Component* i det konceptuella schemat. Strukturen kan ses som en arvshierarki. Detta innebär att t ex ett objekt av klassen *Warning* är ett (is a) objekt av klassen *Component.*

Till en viss komponent kan subkomponenter kopplas. På elevgränssnittet visas subkomponenterna som innehåll i komponenten. Kursförfattaren använder subkomponenter för att skapa kursmaterialets innehåll. Befintliga subkomponenter är *Mouse_over, List, Text, Table, Picture, Heading, File* samt *Execute*.

Alla subkomponenterna har den gemensamma egenskapen är att de utgör innehållet i ett kursmaterial. De ärver egenskaper av klassen subkomponent. Denna klass kallas *Subcomponent* i det konceptuella schemat. Strukturen kan även i detta fall ses som en arvshierarki. Detta innebär att tex ett objekt av klassen *Text* är ett (is a) objekt av klassen *Subcomponent.*



Figur 3.6: Arvshierarki

Hittills har de mest tydliga entiteterna beskrivits men det finns flera viktiga företeelser. Det krävs en förfining av vissa entiteter för att lösningen ska bli flexibel och generell. De sätt som komponenter kan presenteras på, alltså hur layouten ska se ut, definieras som egna entiteter. De benämns med prefixet *pres_* till komponentnamnet. Denna lösning ger en möjlighet till att i framtiden enkelt utöka antalet presentationssätt för varje enskild komponent. Entiteten *Dictionary* står för den ordlista systemet i framtiden ska innehålla.

Det konceptuella schemat visar också vilka typer av relationer som finns mellan de beskrivna entiteterna. Relationer beskrivs grafiskt som namngivna romber. Namnet beskriver kort vilken relation som råder mellan entiteterna. Dessutom redovisas relationstypen.

Det finns tre grundtyper av relationer som kan betecknas 1:1, 1:M och M:M. Relationen 1:1 innebär att ett objekt i en klass endast kan ha en koppling till ett objekt i en annan klass. 1:M innebär att ett objekt kan ha kopplingar till många objekt i en annan klass. M:M betyder att ett objekt i en klass kan ha en relation till många objekt i en annan klass samtidigt som ett objekt i den andra klassen kan ha kopplingar till många objekt i den första klassen.

Det måste finnas möjlighet att identifiera alla objekt unikt i databasen. Denna identifierare kallas primärnyckel och det måste alltid finnas en primärnyckel till varje entitet. Alla attribut och entiteter redovisas i en separat entitet-attribut-katalog. Attributen beskrivs i en attributkatalog. Där visas även vilka attribut som fungerar som primärnycklar.

3.1.3 Resultat av konceptuell design

Den konceptuella designen har resulterat i:

- ett konceptuellt schema, se bilaga A.2.
- en entitet/attribut-katalog som beskriver entiteterna närmare, se bilaga A.3.
- en attribut-katalog som sammanfattar attributen, se bilaga A.4.
- en gemensam begreppsapparat, med samordning av egenskaper och namnsättning, samt verksamhetsorienterad beskrivning av vad som ska lagras i databehandlingssystemet.
- att berörda på ett tidigt stadium fått en uppfattning om databasens struktur.

3.2 Logisk design

Den logiska designen utgår från det konceptuella schemat och resulterar i ett logiskt schema. Ett logiskt schema är en beskrivning av strukturen i databasen och kan behandlas av databashanteringssystemets mjukvara. En logisk modell är ett språk som används för att specificera logiska scheman. De mest använda logiska modellerna tillhör de tre klasserna relation, nätverk och hierarkisk. Den logiska designen beror av den klass den valda datamodellen tillhör. Kursdatabasen designas enligt relationsmodellen.

3.2.1 Relationsmodellen

Relationsmodellen framfördes 1970 av Edgar F. Codd. Modellen är en enkel, kraftfull och formell modell för att representera verkligheten. De grundläggande elementen i modellen är relationen. Relation är den matematiska termen för tabell. Ett relationsdatabasschema är en uppsättning av tabeller.

Relationsmodellen beskriver ett sätt att se på data från en logisk synvinkel. Tre aspekter av data tas upp i modellen:

- Datastrukturer (eller objekt), huvudsakligen tabeller.
- Datamanipulation (eller operationer). Operationer är bl a SELECT, PROJECT och JOIN. Kortfattat beskrivet är SELECT en operation som extraherar ett antal specificierade rader från en tabell medan PROJECT extraherar ett antal specificerade kolumner. JOIN slår ihop två specificerade tabeller som har en gemensam kolumn till en tabell.
- Dataintegritet. Integritet handlar om att se till att data i databasen är korrekt. Åtminstone två integritetsregler måste finnas i en relationsdatabas:
- Varje tabell måste ha en kolumn där värdet för varje rad är unikt.
- Databasen får inte innehålla några främmandenycklar som inte motsvaras av en primärnyckel med samma värde. Det är främmandenycklarna som genom att koppla till en primärnyckel i någon annan tabell skapar relationer mellan tabeller. Det krävs därför att motsvarande primärnyckel finns för att data i databasen ska vara korrekt.

Ett språk (t ex SQL) som implementerar relationsoperationer sägs vara icke proceduriellt, dvs användaren beskriver vad som skall göras, inte hur det skall göras. Hur en operation skall utföras bestämmer optimeraren som är en komponent i DBHS (se kap 3.3.1). En relationsdatabas är en databas som implementerar hela eller delar av relationsmodellen.

3.2.2 Relationsdatabaser

Relationsdatabaser baserar sig på relationsmodellen och är ett system som uppfyller åtminstone följande krav:

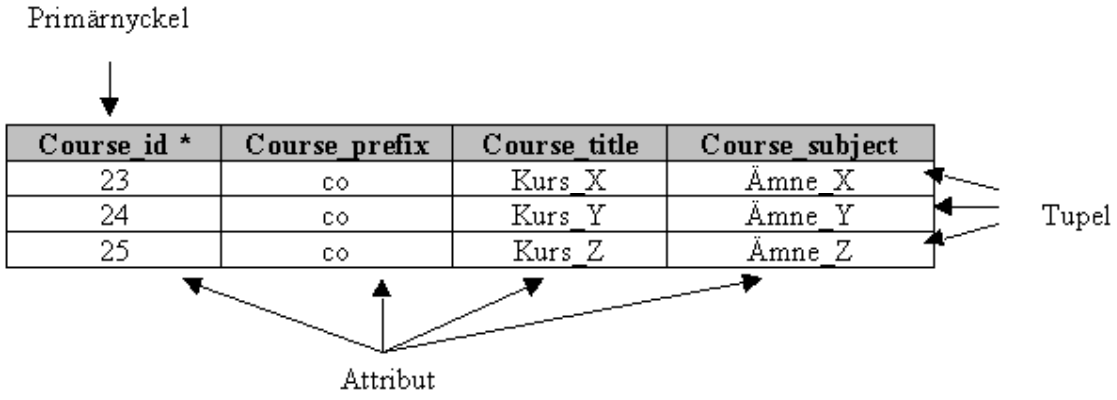
- användaren ser data i form av tabeller och inget annat än tabeller
- de operationer användaren kan göra på databasen är sådana som genererar nya tabeller från de ursprungliga, så kallade vyer. Åtminstone måste operationerna SELECT (kallas också RESRICT), PROJECT och JOIN stödjas (se kap 3.3.3).

För att komma åt informationen i en databas behövs ett speciellt frågespråk. För relationsdatabaser heter standardspråket SQL (Structured Query Language).

3.2.3 Tabeller i relationsdatabaser

Varje kolumn i en tabell lagrar ett visst attribut, dvs en viss egenskap hos entiteten. Det innebär att varje enskild rad i kolumnen beskriver ett visst objekt av entiteten genom de värden som ligger på den specifika raden som kallas tupel. Vissa kolumner fungerar som nycklar. Nycklarna används för att skapa relationer mellan tabeller [2].

För att ett attribut ska vara en kandidatnyckel krävs det att alla instanser i den kolumnen är unika, dvs samma värde får inte återkomma i kolumnen. En primärnyckel väljs i mängden kandidatnycklar och används som identifierare av tabellens instanser. Med främmandenyckel menas en nyckel som kopplar till en primärnyckel i en annan tabell för att skapa en relation.



Figur 3.7: Beskrivning av en tabells uppbyggnad

I den logiska designen är det viktigt att man har fastställt vilka relationer som föreligger mellan olika entiteter.

Förenklat kan tre typer av relationer som finns beskrivas som följer. Anta att A och B är två entiteter som står i någon relation till varandra samt att A och B representeras av var sin tabell:

- 1:1 - relation mellan A och B betyder att en viss instans av tabellen A endast kan relatera till en instans av tabellen B och vice versa. I den logiska designen innebär det att tabell A har en främmandenyckel till B och att B har en främmandenyckel till A.
- 1:M – relation mellan A och B innebär att en viss instans av tabellen A kan kopplas till flera instanser av tabellen B, men att en instans av tabellen B endast kan kopplas till en instans av tabellen A. I den logiska designen betyder det att tabell B har en främmandenyckel till tabell A, dvs främmandenyckeln ska finnas på ”M-sidan”.
- M:M-relation mellan A och B innebär att en instans av tabellen A kan relatera till flera instanser av tabellen B och vice versa. I den logiska designen krävs det att det skapas en ny tabell som fungerar som kopplingstabell mellan A och B.

För att konvertera det konceptuella schemat till ett logiskt schema görs en *mappning*.

3.2.4 Mappning från konceptuellt schema till logiskt schema

Utvecklingen av kursdatabasens konceptuella schema har skett med ett objektorienterat angreppssätt. Eftersom informationen ska lagras i en relationsdatabas måste det konceptuella schemat på något sätt överföras (mappas) till ett logiskt schema för en relationsdatabas. Mappningen är en viktig del i databasdesignen eftersom den i allra högsta grad påverkar de fyra generella egenskaper eller styrkor (gereneral forces) som är utmärkande för databaser. Viss vägledning i hur mappningen påverkar styrkorna ges i Tabell 3.1.

Pattern	Performance			Space Consump-tion	Flexibility, Maintaina-bility	Ad-hoc Queries
	Write/ Update	Single Read	Poly-morphic Queries			
Single Table Aggregation	+	+	*	+	-	-
Foreign Key Aggregation	-	-	*	+	+	+
One Inheritance Tree, One Table	+ o	+ o	+	-	+	+
One Class, One Table	-	-	- o	+	+	-
One Inheritance Path, One Table	+	+	-	+	-	-
Objects in BLOBs	+ o	+ o	o	+	-	-
Foreign Key Association	-	o	*	+	+	+
Association Table	-	o	*	+	+	+
+ good	- poor	* irrelevant	o depends, see detailed discussion			

Tabell 3.1 : Översikt av mappningsmönster och dess egenskaper

Prestanda

Prestanda är en av de viktiga egenskaperna som det måste tas hänsyn till vid val av mappningsmönster. Med prestanda menas antalet diskaccesser som inträffar i systemet. Det tar förhållandevis lång tid att accessa externa medier som hårddiskar och därför är det önskvärt med så få accesser som möjligt för att hantera informationen. Vissa mappningsmönster är effektiva vid läsning i databasen men dåliga vid uppdateringar. God prestanda innebär ofta redundans, dvs dubbellagring.

Utnyttjande av lagringsutrymme

Det finns mappningsmönster som innebär att databasens lagringsutrymme utnyttjas väl medan andra typer av mappningar skapar många null-värden. Ofta är det så att hög prestanda innebär många null-värden och låg prestanda innebär få nullvärden.

Underhållsmöjligheter och flexibilitet

Ibland är det viktigt att databasen är flexibel, dvs att det går att ta bort/ lägga till attribut och klasser i den befintliga databasstrukturen. Redundans innebär att systemet blir mindre flexibelt och besvärligare att underhålla.

Stöd för transaktioner

I affärssystem som utför stora mängder transaktioner är det ofta viktigt att data är redo för online-transaktioner. Det innebär att snabb rekonstruktion av data krävs för optimal prestanda. Vid vissa transaktioner krävs det att data presenteras i former som passar ad hoc- frågor, dvs snabbt påkomna frågor, och det gynnas bla av liten redundans.

Eftersom ovanstående egenskaper ofta motverkar varandra på olika sätt, måste det vid valet av mappningsmönster ske en utvärdering av vilka egenskaper som är viktigast hos den aktuella databasen. En översikt av mappningsmönster och vilka av ovanstående kriterier som uppnås visas i Tabell 3:1. För närmare beskrivning hänvisas till referens [8].

3.2.5 Framtagande av kursdatabasens logiska schema

I en relationsdatabas innebär logisk design att det konceptuella schemat mappas till lämpliga tabeller. I dessa tabeller lagras databasens innehåll. Det är mycket viktigt att tabellerna utformas korrekt och att de kopplas samman på rätt sätt eftersom det är grundläggande för att databasen ska kunna fungera tillfredsställande.

Ett objektorienterat angreppssätt har använts vid design av kursdatabasens konceptuella schema. De strukturer som kan utläsas ur det konceptuella schemat är följande:

- entiteten *Course* är en aggregering av gruppen entiteter *Course_info* och *Course_normal.*
- entiteten Component är en generalisering av gruppen entiteter *NB, Tip, Exercise Warning, Hidden_answer, Question, No_frame, Download ,Read_more, Example, Code* samt *Extras*. Ett objekt ur exempelvis *Warning* är ett objekt av klassen *Component.* Detta bildar strukturen arvshierarki.
- entiteten *Sub_component* är en generalisering av gruppen entiteter *Mouse_over, List, Text, Table, Picture, Heading, File* samt *Execute.*

  Ett objekt ur exempelvis *Text* är ett objekt av klassen *Subcomponent.* Även detta är en arvshierarki.

I följande exempel visas inte alla attribut som tillhör respektive entitet utan endast de som används för att beskriva tillvägagångssättet vid mappning från det konceptuella schemat till tabeller i det logiska schemat.

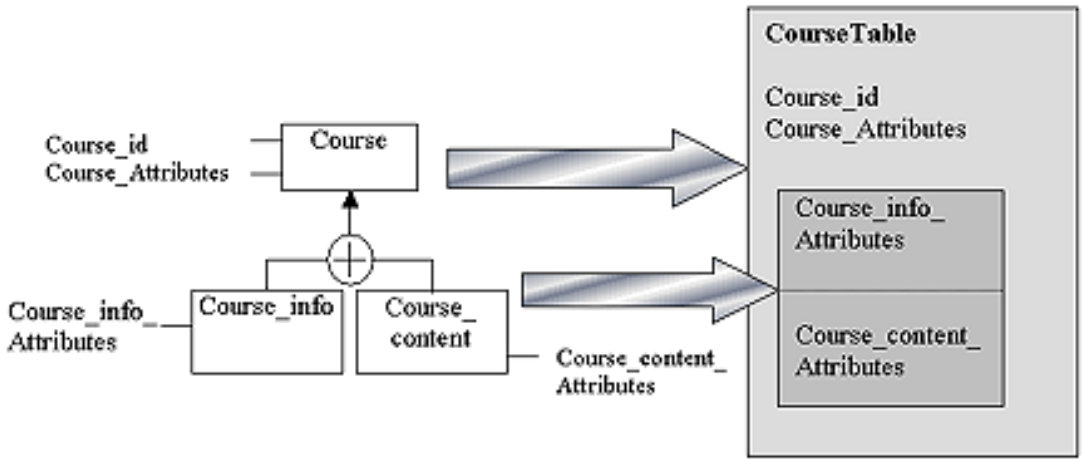
I det första fallet (se punktlistan ovan) skall en aggregering mappas till tabeller i en relationsdatabas. Ur det konceptuella schemat utläses att en kurs kan innehålla delarna kursinformation och övrigt kursinnehåll (se Figur 3.5). Dessa entiteter kallas *Course_info* och *Course_content.*

De aggregerade objekten tillhör entiteterna *Course_info* och *Course_content* medan de aggregerande objekten tillhör entiteten *Course*. Med aggregerande objekt menas här det objekt som ”slår samman” objekt av entiteterna *Course_info* och *Course_normal*. Med aggregerade objekt menas de objekt som ”slås samman”.

Det finns flera sätt mappa aggegeringar till tabeller. Den valda lösningen i detta fall innebär att de aggregerade objektens attribut läggs i samma tabell som det aggregerande objektets attribut.

Vid mappning av aggregeringen har följande arbetsgång använts (se Figur 3.8):

- Det aggregerande objektets attribut mappas till en tabell, dvs objekt av *Course* mappas till en tabell.
- De aggregerade objektens attribut integreras i denna tabell. Attributen från objekten av *Course_info* samt *Course_normal* integreras i den skapade kurstabellen, *CourseTable.*



Figur 3.8: Mappning av aggregering

Detta tillvägagångssätt påverkar de ovan nämnda generella egenskaperna (styrkorna) enligt följande:

Prestanda:

Den valda lösningen är optimal vad avser prestanda eftersom endast en tabell behöver accessas för att hämta ett aggregerande objekt med alla sina aggregerade objekt.

Utnyttjande av lagringsutrymme:

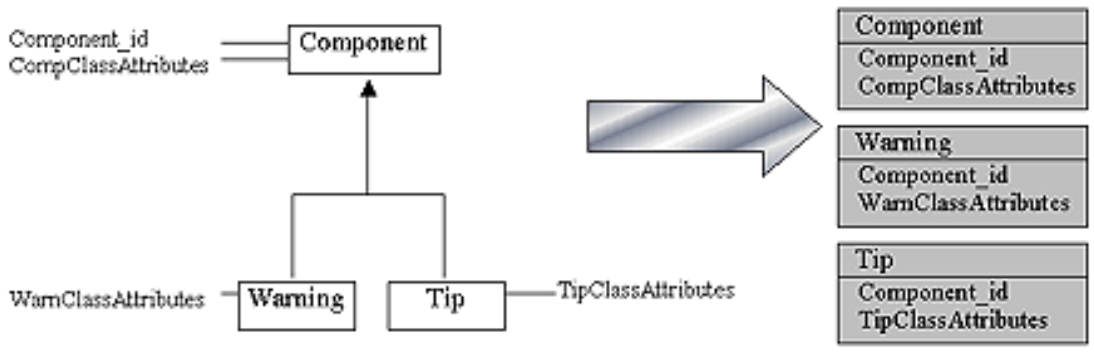
Lösningen är bra vad gäller utnyttjande av lagringsutrymme. Detta mappningsmönster skapar inte många nullvärden.

Underhållsmöjligheter och flexibilitet:

Lösningen är ej flexibel. Varje ändring i de aggregerade objekttyperna medför en ändring i den aggregerande objekttypen.

Det finns även ett flertal varierande sätt att mappa arvshierarkier till tabeller i en relationsdatabas. En *Component* är ett representationssätt kursförfattaren kan använda för att beskriva någon del av kursinnehållet. *Warning* är en *Component*, liksom *Tip* och detta resulterar i en arvshierarki (se Figur 3.6).

Den valda lösningen i detta fall innebär att varje ingående klass mappas till en egen separat databastabell (se Figur 3.9). Attributet ”Component_id” som unikt identifierar tabellen ”Component” läggs in som attribut i de övriga tabellerna. Denna lösning har också valts vid mappning av generaliseringen ”Sub_component”.



Figur 3.9: Mappning av arvshierarki

Detta tillvägagångssätt där varje klass i en arvshierarki mappas till en separat tabell påverkar de ovan nämnda generella styrkorna på följande sätt:

Prestanda:
Den valda lösningen ger inte optimal prestanda. Denna mappning är kostsam vad avser operationer för skrivning och uppdatering av databasen.

Utnyttjande av lagringsutrymme:
Lösningen är nära optimal vad gäller lagringsutrymme. De enda redundanta attribut som behövs är identifikationsattribut som är nödvändiga för att länka ihop nivåerna i hierarkin.

Underhållsmöjligheter och flexibilitet:
Lösningen möjliggör god flexibilitet och utveckling av schemat kan lätt göras. Det går lätt att ta bort och lägga till attribut och klasser i den befintliga databasstrukturen.
Lösningen ger goda underhållsmöjligheter på grund av liten redundans.

3.2.6 Beskrivning av databasens logiska schema

Vid mappning av kursdatabasens konceptuella schema till logiskt schema har en avgränsning skett. En detaljerad beskrivning av detta logiska schema redovisas i bilaga B.3.
I den logiska databasdesignen säkras kursmaterialets struktur med tabellerna *Course*, *Chapter*, *Lesson*, *Section*, *Page*, *Subpage* och relationerna dem emellan. Det finns en relation till *Screenpage* från många positioner i strukturen. *Screenpage* är den tabell som innehåller information om hur en vänstersida ska se ut. Här beskrivs vissa av det logiska schemats tabeller.

Course

En rad i tabellen *Course* är en specifik kurs. Tabellen innehåller information om respektive kurs titel, ämne (om den ingår i en serie kurser tex ”Office”) och författare. Den kan även hålla information om vem som är ansvarig för kursen samt när den är skapad. Vid ändringar i kursen kan det sparas ”när” och ”av vem” detta utfördes. Till en kurs kan kursinformation byggas direkt av *Screenpage*. Om man vill lagra ytterligare kursinnehåll måste detta kopplas till ett kapitel.

Chapter

Tabellen *Chapter* innehåller alla kapitel i databasen. En rad i tabellen *Chapter* är ett specifikt kapitel. Varje kapitel måste tillhöra en kurs och flera kapitel kan kopplas till samma kurs.

Förutom de nycklar som krävs innehåller tabellen information om varje kapitels titel och nummer (i en kurs finns kapitel nummer 1, 2, 3, jfr en vanlig bok). Ett kapitel kan innehålla ett valfritt antal lektioner. Till ett kapitel kan det kopplas en kapitelinformation som kan byggas direkt av *Screenpage.*

Page

Tabellen innehåller alla sidor i databasen. En rad i tabellen *Page* är en specifik kurssida. Varje sida måste tillhöra ett avsnitt och flera sidor kan kopplas till samma avsnitt. Förutom de nycklar som krävs innehåller tabellen information om varje sidas nummer och ev titel. En sida kan kopplas direkt till *Screenpage* som i sin tur kopplas till de komponenter som innehåller det aktuella kursmaterialet. Det krävs ej att det finns subsidor knutna till sidan. Till en sida kan det kopplas ett valfritt antal subsidor som innehåller kursmaterial, tex mer ingående information om det som dess sida visar.

Screenpage

En instans av entiteten *Screenpage* är en specifik vänstersida. Varje *Screenpage* är kopplad till en bestämd plats i kursmaterialets struktur. En *Screenpage* kan innehålla:

- kursmaterial på en sida
- kursmaterial på en subsida
- information om kurs, kapitel, lektion, avsnitt.

Component_Screenpage

Tabellen fungerar som en kopplingstabell mellan *Component* och *Screenpage* eftersom dessa har en M:M-relation. M:M-relationen innebär att en Screenpage kan innehålla flera komponenter *och* att en *Component* kan kopplas till flera *Screenpage,* dvs det ger en möjlighet att återanvända samma komponent flera gånger.

Component

Det finns ett antal fördefinierade komponent-typer. Dessa har en eller flera förutbestämda möjliga layouter. En instans av tabellen *Component* innehåller information om vilken typ av komponent det är samt identifierar sig med en rad i den aktuella komponenttyp-tabellen. Varje komponenttyp har en egen tabell.

Warning

En instans av tabellen *Warning* är en varningsruta med en eller flera fördefinierade layouter. Till varje specifik *Warning*-rad kan det kopplas en eller flera subkomponenter.

Warning_presentation

En instans av tabellen innehåller ett layout-alternativ för en varningsruta. I *Warning_presentation* lagras alla möjliga layouter till komponenten *Warning.*

Subcomponent.

En *Subkomponent* är alltid av någon fördefinierad typ. Varje *Subkomponent*-typ har en egen tabell och dessa kopplas till *Subcomponent.* En rad i tabellen *Subcomponent* lagrar den HTML-formaterade koden som ger den aktuella subkomponenten.

3.2.7 Resultat av logisk design

Den logiska designen har resulterat i:

- ett logiskt schema (se Bilaga B.2).
- en detaljerad beskrivning av logiskt schema (se Bilaga B.3).
- en grund till den fysiska designen
- en flexibel databas i vissa viktiga avseenden.

Mappning av det konceptuella schemat till ett logiskt schema har skett efter vissa valda mönster. Detta har lett till flexibilitet vad gäller att lägga till nya komponenter i databasen. Dessa komponenter bygger upp hur en *Screenpage* ska se ut. Det har även lett till flexibilitet vad gäller att lägga till nya *subkomponenter*. Dessa subkomponenter bygger upp en komponents innehåll. I dessa fall har lösningen lett till mycket lite redundans och goda underhållsmöjligheter. Det finns även möjlighet till att utöka antalet layouter för befintliga komponenter.

3.3 Fysisk design

Fysisk design utgår från det logiska schemat och resulterar i ett fysiskt schema. Ett fysiskt schema är en beskrivning av implementeringen på sekundärminnet. Det beskriver lagringsstrukturen och åtkomstmetoden som används för att effektivt komma åt data. Den fysiska designen är beroende av ett specifikt databashanteringssystem (DBHS).

3.3.1 Databashanteringssystem

Ett databashanteringssystem (DBHS) är ett system som sörjer för accesser och organisation samt har kontroll över all lagrad information i databasen. Det finns många olika typer av DBHS, allt från små system som körs på persondatorer till mycket stora system som körs på stordatorer. DBHS har traditionellt delats in i tre olika strukturer.

De kallas hierarkiska-, nätverks- och relationsstrukturer där namnet refererar till sättet systemet organiserar informationen internt. På senare år har också andra databashanteringssystem utvecklats, t ex objektorienterade DBHS och logiska DBHS.
Funktioner i ett DBHS

- DBHS måste kunna förstå olika datadefinitionsspråk (DDL) och kunna översätta dem till objektkod.
- DBHS måste kunna förstå ett datamanipuleringsspråk (DML) som låter användaren hämta, lägga till eller ta bort data i databasen. Frågor kan var planerade eller ad hoc.
- DBHS måste kunna garantera säkerhet och integritet.
- DBHS måste kunna sköta återhämtning och access från flera användare samtidigt.
- DBHS måste ha en datakatalog där metadata kan lagras. Där lagras detaljerad information om systemets olika delar. Exempel på vad som kan lagras är information om tabeller, index, användare, integritets- och säkerhetsregler.

Ett relationsdatabashanteringssystem (RDBHS) bygger på relationsstrukturen där data lagras i form av relaterade tabeller. Det är mycket vanligt att RDBHS används i organisationer och företag.

3.3.2 Implementering

Den fysiska designen är beroende av ett specifikt databashanteringssystem. Kursdatabasens logiska design är baserad på relationsmodellen. Tech School AB använder sig av ett RDBHS som heter Microsoft SQL Server, vilket användes vid den fysiska designen av kursdatabasen. En databas i MS SQL Server innehåller tabeller, vyer och procedurer. Flera administrativa funktioner såsom skapa och återställa backup, hantera användare, och allokering av filutrymme sker på databasnivå. I MS SQL Server lagras kursdatabasen på åtminstone två filer. En fil lagrar själva kursdatabasen och en fil används till databaslogg.

DBHS kan administreras bla genom lagrade procedurer eller genom det grafiska användargränssnittet SQL Server Enterprize Manager [4]. Med detta verktyg skapas lätt tabeller med tillhörande attribut och datatypen anges. Verktyget Query Analyzer används för att ta fram Transact-SQL-kommandon. Kommandon exekveras med hjälp av detta verktyg och resultatet sparas.

3.3.3 SQL-frågor och testning

SQL är ett standardiserat frågespråk för att komma åt informationen från en databas. Orginalversionen kallas för SEQUEL (Structured English Query Language) och skapades i mitten på 1970-talet. SQL blev introducerat som kommersiellt databassystem 1979 av Oracle Corporation. Historiskt sett, har SQL varit favoriten bland förekommande frågespråk för databashanteringssystem som kör på minidatorer och stordatorer. SQL är ett frågespråk för relationsdatabaser. SQL stöder distribuerade databaser, dvs databaser som är fördelade över flera datasystem. Detta möjliggör att flera användare på lokala nätverk har tillgång till samma databas samtidigt.

I språket SQL ingår två typer av kommandon. Data Definition Language-kommandon (DDL-kommandon) som låter användaren definiera och modifiera schemat i databasen. Det betyder att dessa kommandon används för att definiera de olika objekt som databasen består av. Data Manipulation Language-kommandon (DML-kommandon) låter användaren skapa, hantera och modifiera data i databasen. Dessa kommandon används för att manipulera eller utföra operationer på objekt i databasen.

Följande operationer kan utföras på en databas med hjälp av frågespråket SQL :

- att lägga till nya, tomma filer i en databas
- att ta bort filer från databasen
- att lägga till ny data (poster) till en fil i databasen
- att hämta data från existerande filer i databasen
- att uppdatera existerande filer i databasen
- att ta bort data från existerande filer i databasen.

Transact-SQL är en delmängd (subset) av standard SQL som MS SQL Server använder för att implementera, accessa och underhålla databaser. Här ges ett exempel på hur en ny kurs skapas i kursdatabasen med Transact-SQL.

Följande Transact SQL-fråga lägger in den nya kursens egenskaper som titel, ämnestillhörighet, kursansvarig och kursförfattare i tabellen *Course* i kursdatabasen. I första parantesen anges namnen på de tabellkolumner som ska innehålla respektive egenskap. I den andra parantesen står de värden som ska lagras för den aktuella kursen.

```
insert into Course (Course_prefix, Course_title,
Course_subject, Course_responsible, Course_author)
values ( ”Kursprefix X”,”Kurs X”,” Kursämne X” ,
”Kursansvarig”, ”Kursförfattare”)
```

Det är möjligt att formulera SQL-frågan i Vbscript i ASP-kod (se kapitel 5.3) och sedan kompilera och köra den i MS SQLserver.

I Transact-SQL finns möjligheten att skapa procedurer som kan anropas med parametervärden. Eftersom det är effektivare att anropa färdiga procedurer än att köra frågan direkt varje gång i MS SQLserver, gjordes valet att formulera alla manipulerande SQL-frågor som procedurer. Ovanstående SQL-fråga får som procedur följande utseende:

```
CREATE procedure insert_Course
      @co_tit    varchar(50),@co_sub    varchar(50),@co_res
      varchar(50),   @co_aut varchar(50)
as
insert    into    Course    (Course_prefix,    Course_title,
Course_subject, Course_responsible,
     Course_author)
values (@co_pre, @co_tit, @co_sub, @co_res, @co_aut)
select @@identity
```

Proceduren har namnet insert_Course och tar emot de värden som ska läggas in i tabellen *Course* med parametrar. Till parametrarna ska även anges hur stora värden (antal tecken) som kan lagras på respektive plats i databasen. I sista raden returnerar proceduren det recordset som skapats. Ett recordset är en tabellrad och i det här fallet innehåller den alla värden som sparats om den aktuella kursen i tabellen ”Course”. Under rubriken 5.4.1 beskrivs närmare varför proceduren ska returnera värden.

3.3.4 Resultat av fysisk design

Den fysiska designen har resulterat i

- en implementerad databas som bygger på det logiska schemat.
- ett fysiskt schema, (se bilaga C.1).
- en databas som
  - kan lagra information om kurser, kapitel, lektioner, avsnitt samt sidor.
  - ger goda möjligheter att välja och kombinera sätt att presentera kursmaterialet på.
  - är flexibel vad gäller möjligheten att utöka presentationssätten.
  - lagrar kursmaterial
  - innehåller få nullvärden
  - stöder SQL-kommandon. Ett exempel är att lägga in kursförfattare till viss kurs.

3.4 Underhåll

Efter avslutad databasdesign sammanställdes en dokumentation av det som framkommit under de tre faserna. Denna dokumentation innehåller de scheman och specifikationer som tagits fram. Dokumentationen är viktig för framtida användning och underhåll av databasen. Det är viktigt att understryka att det framtagna konceptuella schemat skall ingå som en del i dokumentationen. I framtiden kan detta schema användas som utgångspunkt vid ev byte av DBHS.

4 Författargränssnitt

Det ställs flera krav på databasens författargränssnitt. Det ska anpassas till författarnas behov samt fungera effektivt mot databasen. Författaren ska kunna interagera med databasen på ett enkelt och logiskt sätt utan att egentligen behöva förstå hur databasen är uppbyggd och fungerar.

4.1 Fördelar med webbaserat gränssnitt till kursdatabasen

Författargränssnittet är ett av flera gränssnitt till ett fungerande webbaserat system. Eftersom Tech School AB producerar webbpublicerade utbildningar, finns det inom företaget goda kunskaper i webbpublicering. Webbaserade system ger möjligheter att lägga ut delar av systemet på Internet, vilket sker på Tech School AB när eleverna kopplar upp sig mot systemets elevgränssnitt via Internet. Allmänt kan det konstateras flera fördelar med att bygga webbaserade gränssnitt. Det är en relativt lätt teknik som är billig och tar kortare tid att utveckla än traditionella gränssnitt. Ett webbaserat system ligger centralt på servern och applikationen behöver inte installeras manuellt hos varje användare. Systemet blir därför lätt att installera och uppgradera [6].

4.2 Konstruktion av webbaserat gränssnitt

Vid konstruktion av det webbaserade författargränssnittet har implementering skett i HTML och Javascript.

4.2.1 HTML

HTML (Hyper Text Markup Language) var från början enbart ett verktyg för att strukturera text. Termen ”Markup” grundar sig på den tiden då manusgranskaren manuellt markerade i dokumenten hur boktryckaren skulle framställa texten. Språket består av struktureringskommandon, sk taggar, som bestämmer hur materialet ska visas på skärmen. Taggarna tolkas sedan av ett program som kallas webbläsare. De ger samma effekt i olika webbläsare, men det kan se olika ut på skärmen beroende på hur webbläsaren programmerats.

HTML är i grunden ett statiskt språk, dvs det har endast som funktion att presentera ett färdigt material på ett visst sätt. Det har lagts ner mycket arbete på att utveckla metoder och tekniker för att göra webbsidor dynamiska. Med dynamisk webbsida menas att användaren ska ha en möjlighet att interagera med sidan. När HTML fick tillägget ramar, dvs möjligheten att placera flera webbdokument i olika fält på bildskärmen, kan man säga att en enkel form av dynamik skapades. Användaren gavs då möjligheten att länka sig mellan olika dokument. Innehållet i dokumenten var dock lika statiskt som förut.

Med dynamiska webbsidor menas dock i regel sidor där användaren kan påverka innehållet i det enskilda webbdokumentet. Genom att bädda in små program i HTML-koden har det blivit möjligt och programmen kan exekveras på antingen klienten eller servern. Båda sätten har fördelar och nackdelar och ofta använder man en kombination av dem för att få de bästa förutsättningarna för webbdynamik.

4.2.2 Formulär

Vid exekvering på server, tex vid ASP-teknik (se kap 5.3), skickas information till serverprogrammet från klienten. Det är vanligt att informationen ligger i sk formulär som laddas till programmet på servern.

Ett webbformulär kan liknas vid en vanlig blankett som har olika fält med plats för text. Det måste finnas en ”submit”-knapp som användaren klickar på när formuläret har fyllts i och är färdigt att skickas till serverprogrammet. Designen av formulär är viktig. De ska vara ändamålsenliga och ha snygg layout, men kanske viktigast av allt är att de är lätta att fylla i Det ska tydligt synas vilken typ av information som ska stå i varje fält.

Det finns några tumregler som säger att [6]:

- Ett formulär inte får vara för stort. Användaren bör kunna se hela formuläret utan att använda rullisten och är viktigt att tänka på att användaren kan sitta vid en mindre skärm än utvecklaren av formuläret.
- Det är logiskt att fylla i ett formulär uppifrån och ner. Låt användaren fylla i information i naturlig ordning och lägg alltid ”submit”-knappen längst ner.
- Storleken på textfält bör i möjligaste mån anpassas till det innehåll som ska fyllas i.

Det är lämpligt att ge formuläret en enkel struktur där de olika fälten och knapparna ligger ordnade efter räta linjer [7].

Det går i viss mån att kontrollera om formulär har fyllts i korrekt innan det laddas till serverprogrammet.

Med små script-program går det att kontrollera att fält till exempel är ifyllda med minsta antal tecken. Eftersom det minskar belastningen på server samt ofta är effektivare att köra program lokalt kan det vara klokt att kontrollera formuläret innan informationen skickas till serverprogrammet. Det är mycket vanligt att använda Javascript, som har stöd i både Netscapes och Microsofts webbläsare, för detta ändamål.

4.2.3 Javascript

Javascript är ett scriptspråk som i grunden utvecklats av Netscape under namnet Livescript och sedan förändrades i samarbete med Sun Microsystems till ett mer javaliknande scriptspråk. Det har samma syntax som Java och påminner om objektorienterad programmering. Javascript har stöd i både Netscapes och Microsofts webbläsare och är därför mycket populärt att använda för exekvering på klienten. Javascript är lämpligt för att hantera användarstyrda händelser som tex ett klick på en knapp på en webbsida. När en viss händelse sker exekveras en viss förutbestämd kod. Javascript-koden kan ligga inbäddad direkt i HTML-koden eller som en funktion deklarerad tidigare på HTML-sidan.

4.3 Design av kursdatabasens gränssnitt

I underlaget till författargränssnittet (se kap 2.2), redovisas tre viktiga önskemål på webbgränsnittets funktionalitet. Kursförfattaren ska med lätthet kunna bygga pedagogiskt upplagda kurser. Det ska visas var i kursstrukturen kursförfattaren för tillfället befinner sig och vissa återkommande moment ska underlättas. Vid utveckling av det webbaserade författargränssnittet visade det sig vara nödvändigt med två alternativa tillvägagångssätt vid inläggandet av studiematerial.

Alternativ 1

Vid skapandet av en ny kurs ska egenskaper som titel etc registreras. Sedan ska systemet automatiskt visa formulär för att skapa nytt kapitel till samma kurs. Automatiken upprepas nedåt i kursstrukturen tills ny sida skapats. Systemet ska hålla reda på var i kursstrukturen författaren befinner sig och detta ska åskådliggöras på skärmen. När sidan skapats, eventuellt subsidan, ska kursförfattaren kunna välja komponenter och subkomponenter samt skriva in den text som konstruerar den eller de *Screenpage* som bygger sidan eller subsidan.

Alternativ 2

Kursförfattaren ska kunna välja att skapa en viss information, sida eller subsida till någon kurs, lektion, kapitel eller avsnitt. Tillhörighet i kurstrukturen ska anges och systemet ska lagra den informationen samt visa och bearbeta det formulär som krävs.

För att minska på antalet återkommande moment ska det vara möjligt att skapa ytterligare ett objekt av samma typ utan att behöva ange information om kursstruktursplacering igen. Exempelvis om kursförfattaren har skapat sida 1 till Kurs X, kapitel 3, avsnitt 2 och vill skapa sida 2 ska systemet klara detta utan att författaren måste ange placering i kursstrukturen återigen. Det ska synas på skärmen vilken kurs, lektion etc som kursförfattaren arbetar med för tillfället.

Bilaga D.1 visar hur författarens webbgränssnitt ser ut om författaren i index till vänster valt att arbeta med en kurs. Det är tänkt att författaren ska kunna arbeta enligt alternativ 2 genom att klicka i index till vänster på ett visst alternativ, tex sida. Vid implementering har arbetet avgränsats till alternativ 1. Vid klick på ”Skapa ny kurs” visas formulär med inmatningsfält där vissa egenskaper för kursen kan anges, se bilaga D.2. Vid klick på knappen ”Lägg in ovanstående kurs i databasen” läggs kursen in i databasen. Hur detta sker beskrivs närmare under rubriken 5.3 ”Webbpublicering av kursdatabasen”. Bilaga D.3 visar den information som returneras från webbservern till webbläsaren. Nu kan kursförfattaren välja på att avsluta eller att skapa ett kapitel till samma kurs. Om kursförfattaren väljer det sistnämnda visas ett formulär, se bilaga D.4, för inmatning av vissa kapitelegenskaper. När detta fyllts i och förmedlats till servern fås viss information tillbaka, se bilaga D.5.
Att formulärens fält har fyllts i kontrolleras av ett javascript. Detta ger fördelar som mindre belastning på servern eftersom exekveringen sker hos klienten. En enkel kontroll av inmatade data kan göras innan något sänts till servern. Kontrollen innebär att om något fält är tomt när det ska skickas till servern visas en meddelanderuta med feltext, se exempel på kursformuläret D.6. HTML-kod och Javascript till kursformulären för att lägga in ny kurs samt nytt kapitel redovisas i bilaga D.7 resp bilaga D.8.

I koden till kursformulären anger HTML-kommandot <form> var formuläret börjar. Med ”method” anges vilken metod som skall användas för att överföra informationen till servern. Detta kan göras med antingen ”get” eller ”post”. Här används ”post” vilket innebär att en obegränsad mängd information kan skickas till servern. Med attributet ”action” anges webbadressen till den ASP-sida som skall ta emot den inmatade informationen.

```
<form method=post action="save_Course.asp">
```

Textfält där kursförfattaren kan skriva in information om en kurs fås av koden:

```
<input type=text name="Course_title"size=50 value="" >
<input type=text name="Course_subject" size=50 value="" >
```

Med kommandot <input> och dess attribut kan formulärets utseende varieras på en mängd olika sätt. Det attribut som reglerar vilken typ av inmatningssätt som skall användas är ”type”.

För att kontrollera huruvida formulärens fält är ifyllda används Javascript. Detta innebär fördelar som mindre belastning på servern eftersom exekveringen sker hos klienten. En enkel kontroll av inmatade data kan göras innan något sänts till servern.

För att lägga in scriptkod på en HTML-sida används <script>-märkordet. Ett exempel på detta är:

```
<HTML>
  <SCRIPT LANGUAGE = "JavaScript">
      kod skriven i Javascript
  </SCRIPT>
      resten av HTML-sidan
</HTML>
```

Funktioner är vanliga i Javascript och kan skrivas in varsomhelst i HTML-koden. Funktionerna ”checkForm” och ”isNonEmpty” läggs tidigt i koden eftersom koden läses uppifrån och ner och anrop bara kan ske till funktioner som är deklarerade tidigare på sidan.

När formuläret är ifyllt väljer kursförfattaren att skicka iväg informationen genom att klicka på en knapp. Om något fält har glömts att fyllas i skickas ingen information till servern utan kursförfattaren får ett felmeddelande. Detta sker genom att knappen ”Tryck innebär att formulärets värden läggs in i databasen” är kopplad till funktionen ”checkForm”.

```
<input type=submit  value="Lägg in ovanstående kurs i
       databasen"   onclick="checkForm(this.form)">
```

Funktionen `checkForm` kontrollerar huruvida alla fält är ifyllda. Denna funktion anropar i sin tur funktionen `isNonEmpty` som kontrollerar om fältet är tomt eller inte. Om ett fält är tomt returneras ”false ” till funktionen `checkForm` och informationstext läggs i variabeln `retval` (returvärde).

Om variabeln `retval` inte är tom skrivs felmeddelande med variabelns innehåll ut på skärmen. Tom variabel innebär att alla fält är ifyllda och informationen skickas till servern. Kodexemplet nedan visar kontroll av formulärfält.

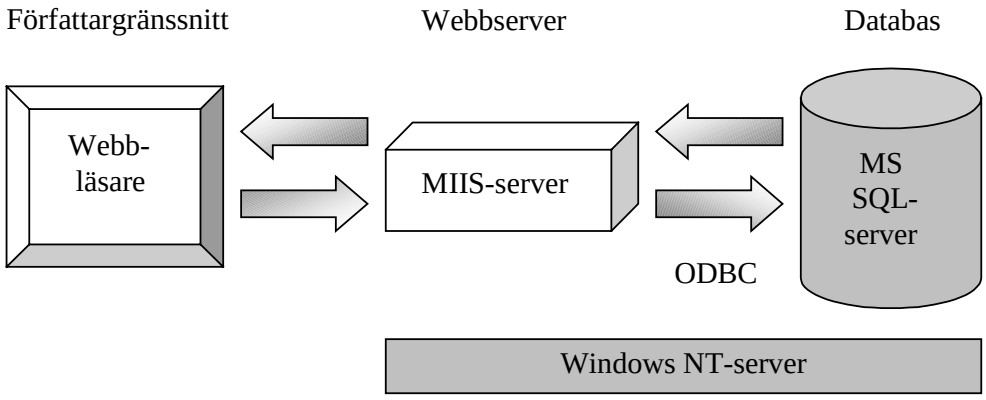
```
function checkForm(form)
      {
        var retval ="";
         if(!isNonEmpty(form.Course_title.value))
        {
        retval += "- Kurstitel\n";
      }
```

5 Webbpublicering

I detta avsnitt redovisas den teknik som ligger mellan webbgränssnitt och databas. Utvecklingen inom området går mycket snabbt och att skapa webbdynamik genom exekvering på webbserver har blivit vanligt både på det globala Internet och på företags lokala Intranet/Extranet [3]. På Internet går det exempelvis bra att boka resor och handla. På företag kan system baseras på en webbserver som skickar information mellan databaser och lokala klienters webbläsare. Det finns många tekniker som utför det som användaren begär. I det här fallet används ASP (Active Server Pages).

5.1 Arkitektur

Webbpublicering av kursdatabasen görs i Microsoft-miljö med Windows NT som plattform. På den plattformen körs webbservern MIIS (Microsoft Internet Informaton Server). MIIS är utvecklat för Windows NT-server och fungerar endast i Windowsmiljö.



Figur 5.1: Arkitektur

5.2 ODBC

Det har arbetats fram en standard för att koppla program till databaser. Microsoft har gjort en implementation av denna standard som kallas ODBC (Open DataBase Connectivity). Denna implementation har nu blivit industristandard.

Tanken bakom ODBC är att en komponent ligger mellan programmet och databasen. Det finns en sådan komponent eller drivrutin för Oracle, en annan för Microsoft SQL Server osv. Fördelen är att alla dessa drivrutiner ser likadana ut från programmet sett. Om programmet kommunicerar med en databas från Oracle via ODBC, och databasen byts ut mot MS SQLserver, behöver programmet inte skrivas om. Det enda som behövs är att ODBC-drivrutinen för den nya databasen installeras.

5.3 ASP

ASP (Active Server Pages) är en modern teknik utvecklad av Microsoft och den är anpassad till Microsofts egna produkter. ASP-sidor kan bara exekveras på Microsofts servrar som tex IIS (Internet Information Server) eller MPW (Microsoft Personal Webserver). Databaskopplingar via ASP fungerar mot de flesta databashanterare som tex Oracle, Microsoft Access och SQL-server. Eftersom ASP returnerar svar i form av vanlig HTML-kod, är tekniken på klientens sida plattformsoberoende. Det är en relativt lättillgänglig teknik som, om man tex jämför med traditionella CGI (Common Gateway Interface) och Perl, är lätt att använda. ASP-tekniken ger goda förutsättningar för databaskopplingar. Databasen kan ligga på webbservern eller finnas som en applikation till den.

Till skillnad mot CGI där programmen ligger separat från HTML-koden, bakas ASP-koden in i själva HTML-dokumentet. ASP är inte ett eget programmeringsspråk utan består av en samling objekt som kan kommunicera med servern och användarens webbläsare. Genom att använda vissa teckenkombinationer som kodavgränsare kan servern känna av var i HTML-koden som ASP-koden ligger[5].

5.3.1 ASP-objekten

ASP bygger på sex olika objekt som har olika användningsområden, Application, Request, Response, Server, Session samt ObjectContext. Objekten innehåller ett antal metoder och egenskaper som programmeraren kan använda [6].

- Application används för att den lagrade informationen ska kunna delas av alla användare på webbplatsen.
- Request används för all kommunikation från användaren. Den information som användaren matat in i ett formulär slussas till ASP-sidan på servern genom det här objektet.
- Response används för all kommunikation till användaren. När en ASP-sida ska skicka en HTML-sida till användaren görs det med hjälp av Response-objektet.
- Session används för att temporärt spara information som är knuten till en viss användare. Det kan till exempel behövas när användaren ska kunna gå vidare mellan olika sammanhängande formulär utan att uppkopplingen mot servern bryts.
- Server används för att importera externa objekt när de inbyggda objekten inte räcker till.
- ObjectContext används för interaction med Microsoft Transaction Server.

Det går bra att använda olika scriptspråk som tex Javascript eller VB-script i samband med ASP-programmering.

5.3.2 VBscript

I det här arbetet har VBscript använts till ASP-sidorna. VBscript är en förenklad scriptversion av programmeringsspråket Visual Basic. Det är väl anpasssat till Microsofts produkter och anses vara ett lätt språk att programmera i.

VBscript är utvecklat av Microsoft och används flitigt i deras produkter. Det är tex ofta VBscript som används i bakgrunden vid skapandet av makron i MS Word och MS Excel. En nackdel med VBscript är att det endast kan garanteras fungera i Microsoft-miljö. Det har tex inte stöd i Netscapes miljö och är därför inte så flitigt använt till program som ska exekveras i Netscapes webbläsare. Eftersom ASP alltid körs på Microsofts servrar är det inte något problem att använda VBscript vid ASP-programmering.

5.4 Webbpublicering av kursdatabasen

Till Tech School AB:s databaskoppling har ASP-teknik använts. Här redovisas som ett exempel hur ett ASP-program tar emot informationen i ett ”Skapa ny kurs” -formulär, lägger in värdena i databasen och returnerar en HTML-sida till webbläsaren varifrån användaren arbetar.

5.4.1 Beskrivning av ASP-kod

ASP-koden ”save_Course.asp” lägger in kursformulärets värden i databasen och returnerar en ny HTML-sida till webbläsaren. Hela koden visas i bilaga E.1 och E.2 och här följer en närmare beskrivning av vissa kodrader.

```
Set DatabasKoppling = Server.CreateObject("ADODB.Connection")
```

Det finns inga inbyggda funktioner i ASP för databashantering. Därför skapas eller importeras en sk ActiveX-komponent som heter ADO(ActiveX Data Objects) med metoden `CreateObject` i objektet `Server.` Objektet `Connection` används för kommunikation med databaser. En lokal instans av objektet `Connection` som döps till `DatabasKoppling` skapas med hjälp av kommandot `Set`. Instansen `DatabasKoppling` kan sedan användas för att komma åt alla metoder i `Connection`. En sådan metod är Open som används med argumenten `Xjobb` (datakällan) och `SA` (användarnamn) för att öppna databasen.

```
DatabasKoppling.Open "Xjobb" , "SA" , ""
```

Med hjälp av Vbscript och textkonkatenering skapas proceduranropet i variabeln SQL. Textkonkatenering innebär att flera textsträngar läggs samman med ”&” till en textsträng. När kursfomuläret fyllts i som i bilaga D.2 blir exempelvis

```
SQL = insert_Course  @co_tit=’Kurs X’,@co_sub=’Kursämne X’,
@co_res=’Kursansvarig X’,@co_aut=’Kursförfattare X’
```

Returvärdet frå SQL-anropet sparas i variabeln `kursid.` `Connection`-metoden `Execute används` för att köra proceduren som `SQL` representerar.

```
Set kursid=DatabasKoppling.Execute(SQL)
```

SQL-proceduren returnerar hela tabellraden, ett recordset, som skapats i databasen. Sessionvariabeln ”`kursid_se"` får värdet på första fältet i recordset som är det kursid som automatiskt genererats i databasen. Kurstiteln sparas i sessionsvariabeln `kurstit_se`.

```
Session("kursid_se")=kursid.fields(0)
Session("kurstit_se")=Request.Form("Course_title")
```

Databasen stängs och instansen `Databaskoppling` nollställs.

```
DatabasKoppling.close
set DatabasKoppling=nothing
```

Efterföljande kod skapar den HTML-sida som returneras till webbläsaren (se bilaga D.3).

5.4.2 Resultat webbpublicering av databasen

Ett övergripande resultat av webbpubliceringen är att kursförfattarna har möjlighet att sitta vid en vanlig webbläsare och utföra sitt arbete. Det krävs inte att ett speciellt verktyg installeras på just den datorn där kursförfattaren arbetar. Det kommer att krävas någon typ av inloggning för behöriga kursförfattare.

Ett resultat som webbpubliceringen av kursdatabasen gett är att det åskådliggörs i författargränssnittet var i kursstrukturen kursförfattaren arbetar för tillfället. Det uppfyller ett av önskemålen som kom fram i undersökningsfasen.

Exempelvis utför koden som redovisats tidigare att det när en kurs har skapats och lagts in i databasen returneras ett svar som bla innehåller kurstitel. Om kursförfattaren väljer att fortsätta och skapar ett kapitel till kursen, lagras kapitlet i databasen med informationen om kurstillhörighet. Webbservern returnerar sedan en HTML-sida till kursförfattarens webbläsare som visar vilket kapitel som lagts in samt till vilken kurs det hör. I ASP-koden ”save_Chapter.asp” (se bilaga E.2), läggs värdet i sesssionsvariabeln `kurstit_se,` som skapats i ”save_Course.asp”, i variabeln `titel_kurs` och motsvarande sker med kursens id. Kurstiteln kan sedan returneras tillsammans med kapitelvärden för att visas i webbläsaren, (se bilaga D.5). Kursens id används som identifierare för den kurs som kapitlet tillhör, dvs den behövs för att upprätthålla kursdatabasens struktur. Ovanstående visar att sessionsvariabler är en möjlighet som finns i ASP-tekniken för att komma förbi problemet med HTTP-protokollets tillståndslöshet.

6 Resultat av hela arbetet

Att designa ett informationssystem är en komplex uppgift. Ett informationssystems livscykel kan delas upp i rimlighetsstudier, uppsamlande och analys av krav, design, prototyputveckling, implementation, utvärdering och testning samt uppstartande och avveckling av systemet. Designdelen innefattar databasdesign samt applikationsdesign.
De delar arbetet främst har fokuserats på är framtagande av underlag, databasdesign och gränssnittsdesign, samt viss implementering.

En god dialog med författare och handledare på företaget resulterade i att undersökningen gav klar bild av de viktigaste önskemål och krav som ställdes på databasen och författargränssnittet.

Stor vikt har lagts vid genomförandet av databasdesignen. Den har utarbetats metodiskt och med eftertanke. Viktiga resultat är ett konceptuellt schema med tillhörande entitet/attribut-katalog. Alla attribut beskrivs mer ingående i en attribut-katalog.

Mappning av det konceptuella schemat till ett logiskt schema har skett med hänsyn till de krav som ställts samt de viktiga styrkorna prestanda, flexibilitet, underhållsmöjligheter och utnyttjande av lagringsutrymme. Detta har lett till en flexibel databas vad gäller utbyggbarhet av komponenter för att presentera kursmaterialet. Resultat från den logiska designen är ett logiskt schema samt en detaljerad beskrivning av detta.

Implementeringen av kursdatabasen gick snabbt med det utförliga logiska schemat som grund. Resultatet visas i ett fysiskt schema.

Ett viktigt resultat som webbpubliceringen av kursdatabasen har gett är att det åskådliggörs i författargränssnittet var i kursstrukturen kursförfattaren befinner sig. Inläggning av kursmaterial i databasen sker med hjälp av SQL-kommandon genom författargränssnittet.

Det webbaserade författargränssnittet behöver utvecklas, implementeras samt testas innan det kan anses färdigt. Det verkar inte finnas begränsningar i ASP-tekniken som hindrar att det går att genomföra. Det arbete som redovisas i den här rapporten behandlar ett område som ligger i tiden. Det har varit lärorikt och givande med flera olika metoder och tekniker inblandade.

Referenser

[1] C. Batini, S. Ceri and S. B. Navathe. *Conceptual Database Design - an entity-relation approach.* The Benjamin/Cummings Publishing Company, Inc, 1992.

[2] C. J. Date. *AN INTRODUCTION TO DATABASE SYSTEM.* Addison-Wesley Publishing Company, Sixth edition, 1995.

[3] Charles Stross, *The Web Architect's handbook. Addison-Wesley, 1996*

[4] D. K. Rensin, A. M. Fedorchek and W. C. Amo. *Microsoft SQL Server 7 Secrets.* IDG Books Worldwide, Inc, 1999.

[5] Erik Ronne. *Avancerad Pocket ASP, ACTIVE SERVER PAGES.* DOCENDO, 2000.

[6] Martin Omander. *[DIN GUIDE TILL] Internet programmering.* BONNIER ICON, 1998.

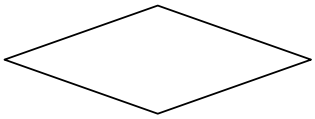
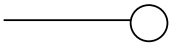
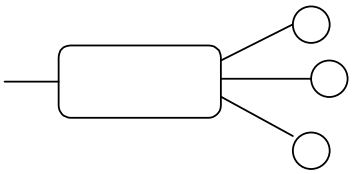
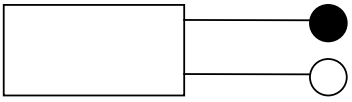
[7] Sanjaya Hettihewa, *Lär dig Active Server pages 2.0 på 21 dagar*. Pagina AB, 1999

[8] Wolfgang Keller. *Mapping Objects to Tables, A Pattern Language.* http://www.sdm.de/g/arcus/, 1997.

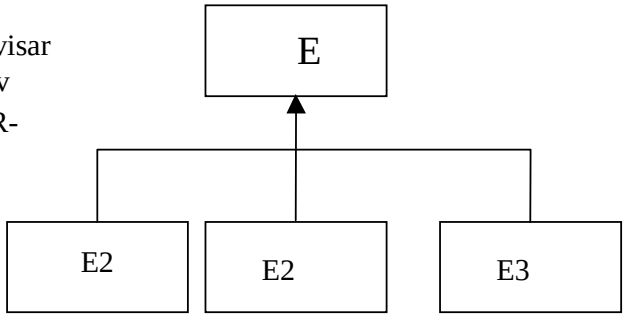
[9] http://msdn.microsoft.com/scripting/default.htm?/scripting/vbscript/default.htm

A Bilagor till konceptuell design

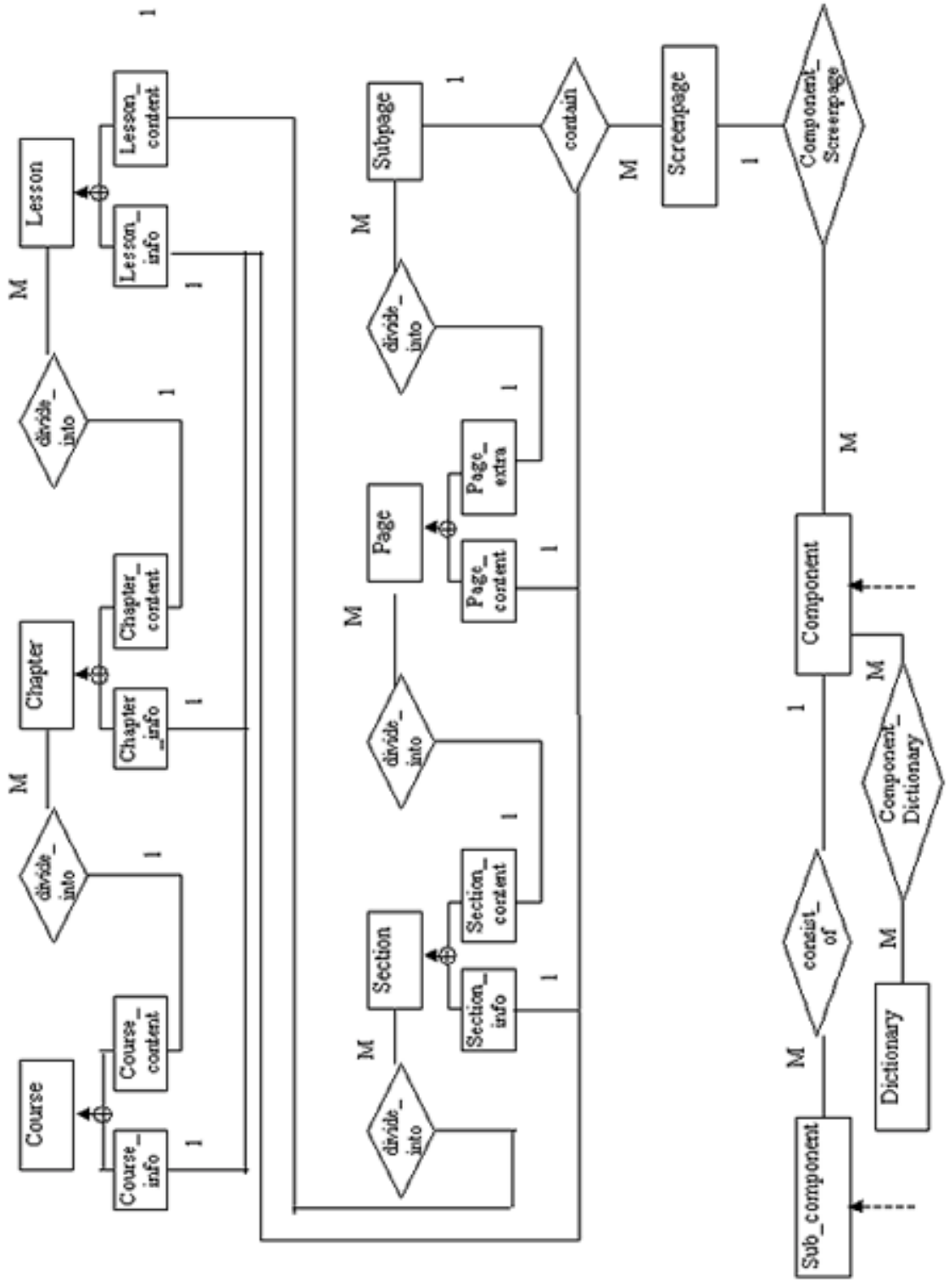
A.1 Grafisk representation av ER-modellen

Begrepp	Grafisk symbol
Entitet	
Relation	
Attribut	
Sammansatt attribut	
Identifierare	

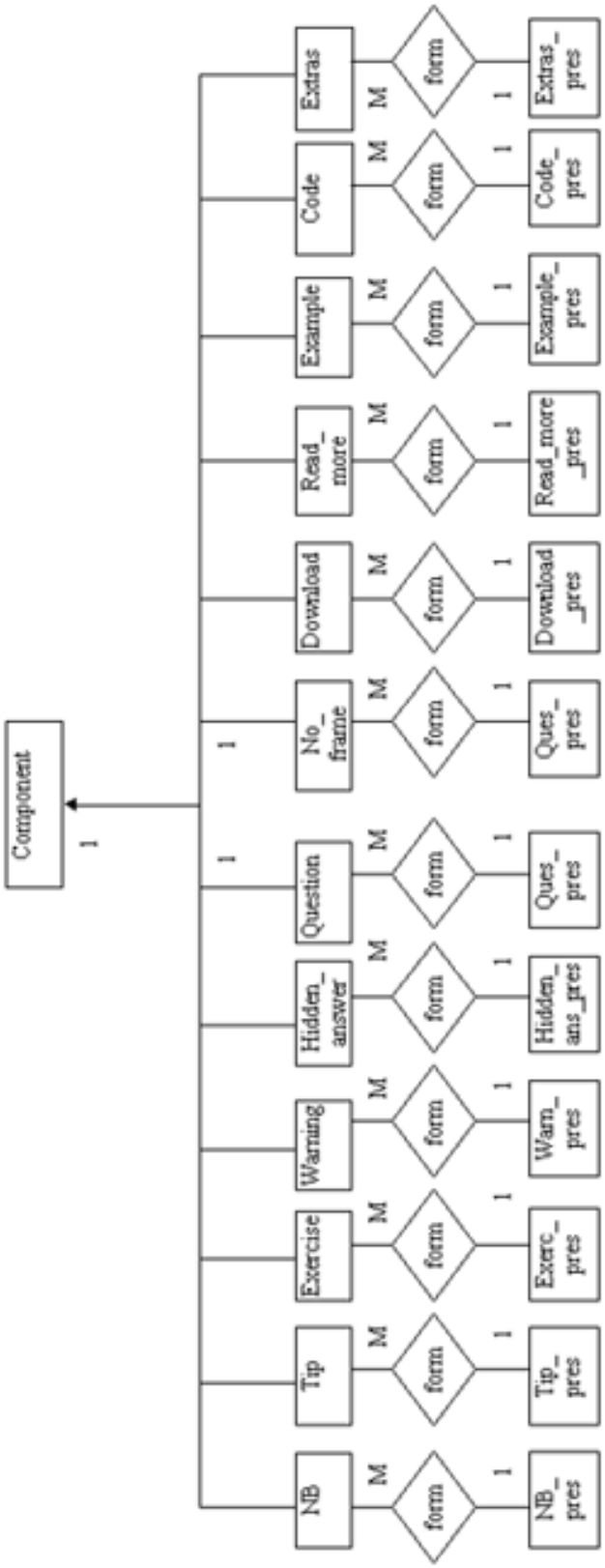
Figuren till höger visar representationen av generalisering i ER-modellen.



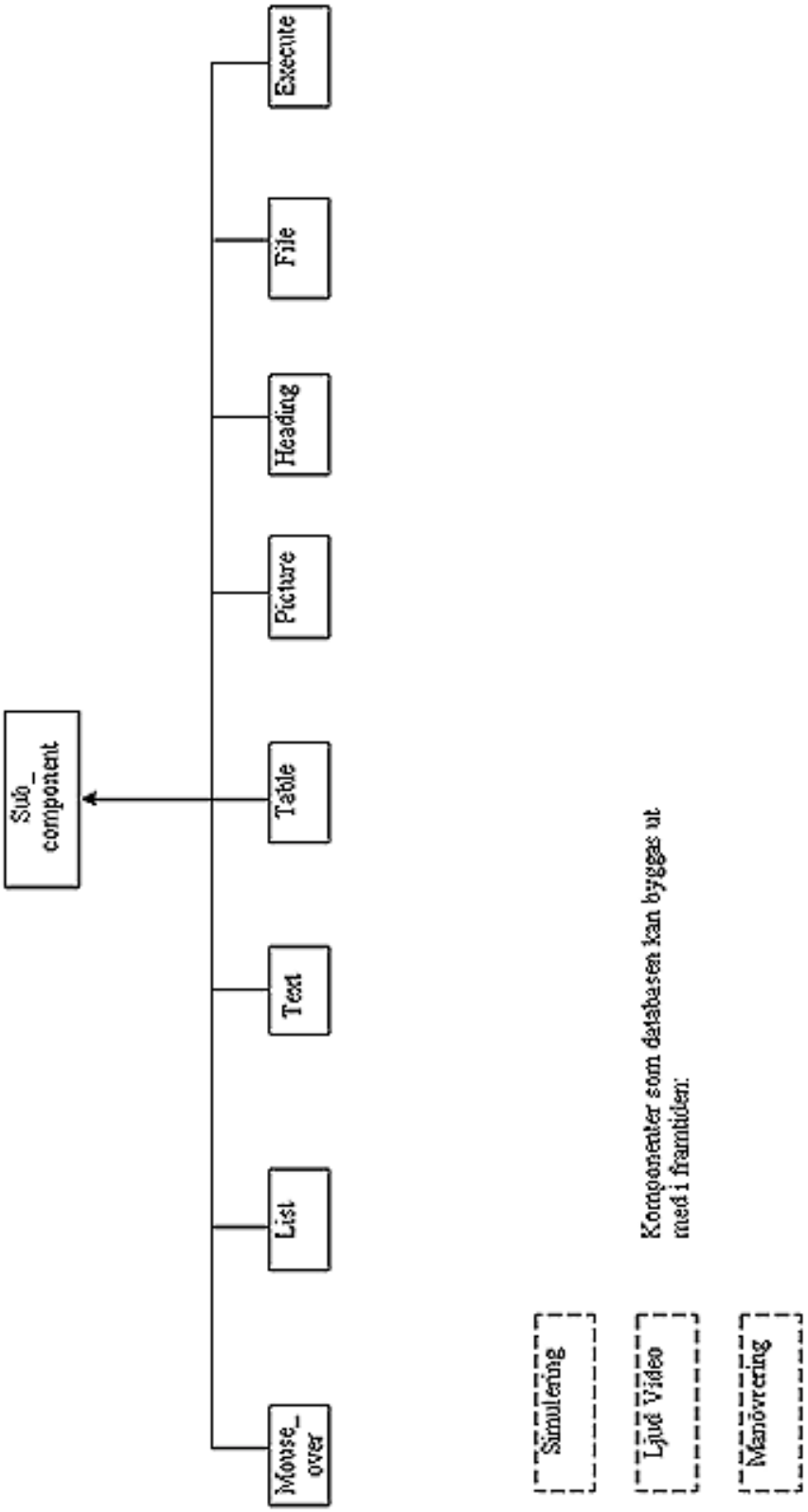
A.2 Konceptuellt schema: Kursstruktur



A.3 Konceptuellt schema: Komponenter



A.4 Konceptuellt schema: Subkomponenter



A.5 Entitet / Attribut – katalog

(Primärnyckel markeras med *)

Objekttyp: *Course*

Beskrivning: Med *Course* avses alla kurser som Tech School AB erbjuder på webben.

Attribut	Datatyp
Course_id *	INT
Course_pre *	VARCHAR
Course_title	VARCHAR
Course_subject	VARCHAR
Course_author	VARCHAR
Course_created	DATETIME
Course_modif_by	VARCHAR
Course_modif_date	DATETIME

Objekttyp: *Chapter*

Beskrivning: Med *Chapter* avses alla kapitel som ingår i någon kurs.

Attribut	Datatyp
Chapter_id *	INT
Chapter_pre *	VARCHAR
Course_id	INT
Course_pre	VARCHAR
Chapter_num	TINYINT
Chapter_title	VARCHAR

Objekttyp: *Lesson*

Beskrivning: Med *Lesson* avses alla lektioner som ingår i något kapitel.

Attribut	Datatyp
Lesson_id *	INT
Lesson_pre *	VARCHAR
Chapter_id	INT
Chapter_pre	VARCHAR
Lesson_num	TINYINT
Lesson_title	VARCHAR

Objekttyp: *Section*

Beskrivning: Med *Section* avses alla avsnitt som ingår i någon lektion.

Attribut	Datatyp
Section_id *	INT
Section_pre *	VARCHAR
Lesson_id	INT
Lesson_pre	VARCHAR
Section_num	TINYINT
Section_title	VARCHAR

Objekttyp: *Page*

Beskrivning: Med *Page* avses alla sidor som ingår i något avsnitt.

Attribut	Datatyp
Page_id *	INT
Page_pre *	VARCHAR
Section_id	INT
Section_pre	VARCHAR
Page_num	TINYINT
Page_title	VARCHAR

Objekttyp: *Subsida*

Beskrivning: Med *Subsida* avses alla subsidor som finns till någon sida.

Attribut	Datatyp
Subpage_id *	INT
Subpage_pre *	VARCHAR
Page_id	INT
Page_pre	VARCHAR
Subpage_num	TINYINT
Subpage_title	VARCHAR

Objekttyp: *Screenpage*

Beskrivning: Med *Screenpage* avses alla enskilda vänstersidor som presenterar kursmaterialet på elevgränssnittet.

Attribut	Datatyp
Screen_page_id *	INT
Belong_to_id	INT
Belong_to_pre	VARCHAR
Screen_order	TINYINT
Screen_type	VARCHAR

Objekttyp: *Component_Screenpage*

Beskrivning: Med *Component_Screenpage* avses den entitet som behövs för att skapa en M:M-relation mellan *Component* och *Screenpage*.

Attribut	Datatyp
Comp_Screen_id *	INT
Component_id	INT
Screenpage_id	INT

Objekttyp: *Component*

Beskrivning: Med *Component* avses en generalisering av ett antal fördefinierade komponenttyper. En komponent beskriver hur layouten ska se ut på elevernas gränssnitt.

Attribut	Datatyp
Component_id *	INT
Specification_id	INT
Specification_name	VARCHAR

Objekttyp: *Warning*

Beskrivning: Med *Warning* avses komponent-typen varningsruta.

Attribut	Datatyp
Warning_id *	INT
Warning_name *	VARCHAR
Component_id	INT
Warning_pres_name	VARCHAR

Objekttyp: *Warning_presentation*

Beskrivning: Med *Warning_presentation* avses varje möjligt utseende på komponententypen *Warning*.

Attribut	Datatyp
*Warning_pres_name**	VARCHAR
Warning_code	VARCHAR

Objekttyp: *No_frame*

Beskrivning: Med *No_frame* avses en typ av komponent. Denna komponenttyp används när exempelvis en text ska ligga direkt på en S*creenpage*. Den har ingen synlig ram på elevgränssnittet.

Attribut	Datatyp
No_frame_id *	INT
No_frame_name *	VARCHAR
Component_id	INT
No_frame_pres_name	VARCHAR

Objekttyp: *No_frame_presentation*

Beskrivning: Med *No_frame_presentation* avses varje möjlig layout till komponenten *No_frame*.

Attribut	Datatyp
*No_frame_pres_name**	VARCHAR
No_frame_code	VARCHAR

Objekttyp: *Subcomponent*

Beskrivning: Med *Subcomponent* avses en generalisering av möjligt kursmaterial som kan kopplas till någon komponent. Kursmaterial kan exempelvis visas som lista eller text.

Attribut	Datatyp
Sub_component_id *	INT
Component_id	INT
Sub_specification_id	INT
Sub_specification_name	VARCHAR

Objekttyp: *List*

Beskrivning: Med *List* avses subkomponenttypen lista.

Attribut	Datatyp
List_id *	INT
List_name *	VARCHAR
Sub_component_id	INT
List_code	VARCHAR

Objekttyp: *Text*

Beskrivning: Med *Text* avses subkomponenttypen text.

Attribut	Datatyp
Text_id *	INT
Text_name *	VARCHAR
Sub_component_id	INT
Text_code	VARCHAR

A.6 Attribut – katalog

Termer	Benämning	Typ
Course_id	Identifierande nummer på kurs.	INT
Course_pre	Prefix som identifierar att det är en kurs.	VARCHAR
Course_title	Titel på kurs.	VARCHAR
Course_subject	Ämnestillhörighet för kurs.	VARCHAR
Course_author	Namn på kursförfattare.	VARCHAR
Course_created	Datum när kurs skapas.	DATETIME
Course_modif_by	Namn på person som ändrat i kurs.	VARCHAR
Course_modif_date	Datum när senaste ändring i kurs skett.	DATETIME
Chapter_id	Identifierande nummer på kapitel.	INT
Chapter_pre	Prefix som identifierar att det är ett kapitel.	VARCHAR
Chapter_num	Nummer på kapitel i viss kurs.	INT
Chapter_title	Titel på kapitel.	VARCHAR
Lesson_id	Identifierande nummer på lektion.	INT
Lesson_pre	Prefix som identifierar att det är en lektion.	VARCHAR
Lesson_num	Nummer på lektion i visst kapitel.	INT
Lesson_title	Titel på lektion.	VARCHAR
Section_id	Identifierande nummer på avsnitt.	INT
Section_pre	Prefix som identifierar att det är ett avsnitt.	VARCHAR
Section_num	Nummer på avsnitt i viss lektion.	INT
Section_title	Titel på avsnitt.	VARCHAR
Page_id	Identifierande nummer på sida.	INT
Page_pre	Prefix som identifierar att det är en sida.	VARCHAR
Page_num	Nummer på sida i visst avsnitt.	INT
Page_title	Titel på sida.	VARCHAR
Subpage_id	Identifierande nummer på subsida.	INT
Subpage_pre	Prefix som identifierar att det är en subsida.	VARCHAR
Subpage_num	Nummer på en subsida.	INT
Subpage_title	Titel på en subsida.	VARCHAR
Screen_page_id	Identifierande nummer på en vänstersida.	INT

Belong_to_id	Identifierande nummer på den kurs, kapitel, lektion, avsnitt, sida eller subsida som vänstersidan tillhör.	INT
Belong_to_pre	Identifierar om vänstersidan tillhör en kurs, kapitel, lektion, avsnitt, sida eller subsida.	VARCHAR
Screen_order	Vänstersidas ordningsnummer.	TINYINT
Screen_type	Typ av vänstersida.	VARCHAR
Comp_Screen_id	Identifierande nummer för entiteten *Component_Screenpage*.	INT
Component_id	Identifierande nummer på någon typ av komponent.	INT
Specification_id	Identifierande nummer på en viss typ av komponent.	INT
Specification_name	Identifierande namn på komponenttypen.	VARCHAR
Warning_id	Identifierande nummer på en komponent av typen *Warning*.	INT
Warning_name	Identifierar att komponenten är en *Warning*.	VARCHAR
Warning_pres_name	Identifierande namn på viss *Warning*-layout.	VARCHAR
Warning_code	Kod som genererar viss *Warning*-layout.	VARCHAR
No_frame_id	Identifierande nummer på en komponent av typen *No_frame*.	INT
No_frame_name	Identifierar att komponenten är en *No_frame*.	VARCHAR
No_frame_pres_name	Identifierande namn på viss *No_frame*-layout.	VARCHAR
No_frame_code	Kod som genererar viss *No_frame*-layout.	VARCHAR
Sub_component_id	Identifierande nummer på någon subkomponent.	INT
Sub_specification_id	Identifierande nummer på en viss typ av subkomponent.	INT
Sub_specification_name	Identifierande namn på en viss typ av subkomponent.	VARCHAR
List_id	Identifierande nummer på en subkomponent *List*.	INT
List_name	Identifierande namn på att det är en lista.	VARCHAR
List_code	Kod som genererar en viss lista.	VARCHAR
Text_id	Identifierande nummer på en subkomponent *Text*.	INT
Text_name	Identifierande namn på att det är en text.	VARCHAR
Text_code	Kod som genererar en viss text.	VARCHAR

B Bilagor till logisk design

B.1 Logiskt schema

Logiskt schema enligt relationsmodellen (relationsschema):

Teckenförklaring: * Identifierande primärnyckel

COURSE

Course_id *	Course_prefix	Course_title	Course_subject

Course_responsible	Course_author	Course_created	Course_modif_by

Course_modif_date

CHAPTER

Chapter_id *	Chapter_prefix*	Course_id	Course_prefix

Chapter_num	Chapter_title

LESSON

Lesson_id *	Lesson_prefix*	Chapter_id	Chapter_prefix

Lesson_num	Lesson_title

SECTION

Section_id *	Section_prefix*	Lesson_id	Lesson_prefix

Section_num	Section_title

PAGE

Page_id *	Page_prefix*	Section_id	Section_prefix

Page_num	Page_title

SUBPAGE

Subpage_id *	Subpage_prefix*	Page_id	Page_prefix

Subpage_num	Subpage_title

SCREENPAGE

Screen_page_id *	Belong_to_id	Belong_to_prefix	Screen_order

Screen_type

COMPONENT_SCREENPAGE

Comp_Screen_id *	Component_id	Screenpage_id

COMPONENT

Component_id *	Specification_id	Specification_name

WARNING

Warning_id *	Warning_name*	Component_id	Warning_pres_name

WARNING_PRESENTATION

Warning_pres_name *	Warning_code

NO_FRAME

No_frame_id *	No_frame_name*	Component_id	No_frame_pres_name

NO_FRAME_PRESENTATION

No_frame_pres_name *	No_frame_code

SUBCOMPONENT

Sub_component_id *	Component_id	Sub_specification_id	Sub_specification_name

LIST

List_id *	List_name*	Sub_component_id	List_code

TEXT

Text_id *	Text_name*	Sub_component_id	Text_code

B.2 Detaljerad beskrivning av logiskt schema

Den logiska designen av kursdatabasen har förutom *Course, Lesson, Section, Page, Subpage* och *Screenpage* begränsats till komponenterna *No_frame* och *Warning* med tillhörande presentation samt subkomponenterna *List* och *Text*. Övriga komponenter och subkomponenter implementeras i den fysiska designen på motsvarande sätt.

Databasen har två viktiga funktioner att fylla. Dels ska den upprätthålla den struktur som kursmaterialet är uppbyggt efter. Dels ska den, med utgångspunkt från hur författaren vill att materialet ska presenteras på elevgränssnittet, lagra kursmaterialet på ett lättillgängligt sätt.

Med kursmaterialets struktur menas att varje kurs delas in i kapitel, varje kapitel delas in i lektioner, varje lektion delas in i avsnitt och varje avsnitt delas in i sidor. Sidorna kan slutligen innehålla flera subsidor. Strukturen påminner mycket om det sätt som en vanlig bok normalt är uppbyggd på. I den logiska databasdesignen säkras strukturen med tabellerna *Course, Chapter, Lesson, Section, Page, Subpage* och relationerna dem emellan.

Course

En instans av tabellen *Course* är en specifik kurs. Tabellen innehåller information om respektive kurstitel, ämne (om den ingår i en serie kurser tex ”Office”) och författare. Den kan även hålla information om vem som är ansvarig för kursen samt när den är skapad. Vid ändringar i kursen kan det sparas ”när” och ”av vem” detta utfördes.

Till en kurs kan det kopplas en kursintroduktion, en kursinformation (tex info om kursens författare, presentation av handledare) samt en kurssammanfattning. Dessa kan byggas direkt med *Screenpage*. Om man vill lagra ytterligare kursinnehåll måste detta kopplas till ett kapitel.

Chapter

Tabellen innehåller alla kapitel i databasen. En instans av tabellen *Chapter* är ett specifikt kapitel. Varje kapitel måste tillhöra en kurs och flera kapitel kan kopplas till samma kurs. Förutom de nycklar som krävs innehåller tabellen information om varje kapitels titel och nummer (i en kurs finns kapitel nummer 1, 2, 3, jfr en vanlig bok).

Ett kapitel kan innehålla ett valfritt antal lektioner.

Till ett kapitel kan det kopplas en kapitelintroduktion och en kapitelsammanfattning. Dessa kan byggas direkt av *Screenpage*. Om man vill lagra ytterligare kapitelinnehåll måste detta kopplas till en lektion.

Lesson

Tabellen innehåller alla lektioner i databasen. En instans av tabellen *Lesson* är en specifik lektion. Varje lektion måste tillhöra ett kapitel och flera lektioner kan kopplas till samma kapitel. Förutom de nycklar som krävs innehåller tabellen information om varje lektions nummer och ev titel.

Till en lektion kan det kopplas en lektionsintroduktion, ett antal avsnitt samt en lektionssammanfattning. Det lektionsinnehåll som ej tillhör introduktionen eller sammanfattningen måste kopplas till ett avsnitt.

Section

Tabellen innehåller alla avsnitt i databasen. En instans av tabellen *Section* är ett specifikt avsnitt. Varje avsnitt måste tillhöra en lektion och flera avsnitt kan kopplas till samma lektion. Förutom de nycklar som krävs innehåller tabellen information om varje avsnitts nummer och ev titel.

Till ett avsnitt kan det kopplas en avsnittsintroduktion, ett antal sidor samt en avsnittssammanfattning. Det avsnittsinnehåll som ej tillhör introduktionen eller sammanfattningen måste kopplas till en sida.

Sida

Tabellen innehåller alla sidor i databasen. En instans av tabellen *Page* är en specifik kurssida. Varje sida måste tillhöra ett avsnitt och flera sidor kan kopplas till samma avsnitt. Förutom de nycklar som krävs kan tabellen innehålla information om varje sidas nummer och ev titel.

En sida knyts till den/de *Screenpage*-instanser som i sin tur har kopplats till de komponenter som innehåller det aktuella kursmaterialet. Det krävs ej att det finns subsidor knutna till sidan. Till en sida kan det kopplas ett valfritt antal subsidor som innehåller kursmaterial, tex mer ingående information om det som dess sida visar.

Subsida

Tabellen innehåller alla subsidor i databasen. En instans av tabellen *Subpage* är en specifik subsida. Varje subsida måste tillhöra en sida och flera subsidor kan kopplas till samma sida. Förutom de nycklar som krävs innehåller tabellen information om varje subsidas nummer och ev titel.

Component_Screenpage

Tabellen fungerar som en kopplingstabell mellan *Component* och *Screenpage* eftersom dessa har en M:M-relation. M:M-relationen innebär att en S*creenpage* kan innehålla flera komponenter *och* att en *Component* kan kopplas till flera skärmsidor, dvs det ger en möjlighet att återanvända samma komponent flera gånger.

Följande del av databasen innehåller information om hur varje enskild vänstersida kan byggas. En vänstersida byggs upp av ett valfritt antal komponenter. Komponenterna kan vara av olika fördefinierade typer, se beskrivning av *Component* i denna bilaga. Komponenterna byggs i sin tur upp av en eller flera olika typer av subkomponenter, se beskrivning av *Subcomponent* i denna bilaga. Det är i subkomponent-tabellerna som det HTML-formaterade kursmaterialet lagras. Databasen är konstruerad så att användaren ska ha stor valfrihet att välja komponenter när denne konstruerar varje enskild vänstersida. De fördefinierade layouterna på komponenterna begränsar användarens frihet något, men tanken bakom detta är att kurserna på ett enkel sätt ska få en enhetlig layout.

Screenpage

En instans av entiteten *Screenpage* är en specifik vänstersida. Varje *Screenpage* är kopplad till en bestämd plats i kursmaterialets struktur. En vänstersida kan

- visa information om *Course, Chapter, Lesson, Section* eller *Page.*
- visa kursmaterial tillhörande en *P*age.
- visa kursmaterial tillhörande en *Subpage.*

Flera *Screenpage*-instanser kan kopplas till samma instans av entiteterna *Course, Chapter, Lesson, Section, Page* eller *Subpage*. Det betyder att varje information, kurs-sida eller kurs-subsida kan bestå av en eller flera *Screenpage.*

Component

Det finns ett antal fördefinierade komponent-typer. Dessa har en eller flera förutbestämda möjliga layouter. En instans av tabellen ”Component” innehåller information om vilken typ av komponent det är samt identifierar sig med en instans i den aktuella komponenttyp-tabellen (1:1-koppling). En komponent är alltid av någon av följande typer:

- *No_frame*
- *Warning*
- *NotaBene*
- *Tip*
- *Exercise*
- *Hidden_answer*
- *Question*
- *Download*
- *Read_more*
- *Example*
- *Code*
- Extras (Ger en möjlighet att utöka med speciella komponent-layouter)

Varje komponenttyp har en egen tabell. Det logiska schemat har som tidigare nämnts avgränsats till komponenterna *No_frame* och *Warning.* De övriga komponenterna skulle dock ha implementarats i det fysiska schemat på motsvarande sätt som *No_fram*e och *Warning*.

No_frame

En instans av tabellen *No_frame* är en komponenttyp med ”osynlig” ram. Det kan tex vara aktuellt när en vanlig text ska ligga på en skärmsida.

Warning

En instans av tabellen *Warning* är en komponenttyp med ett eller flera fördefinierade layouter. Till varje specifik *Warning*-instans kan det kopplas en eller flera subkomponenter.

Warning_presentation

En instans av tabellen *Warning_presentation* innehåller ett layout-alternativ för alla instanser av *Warning*. I *Warning_presentation* lagras alla möjliga layouter till komponenten *Warning.*

Subcomponent

En subkomponent är alltid av en fördefinierad typ. Befintliga subkomponenttyper är:

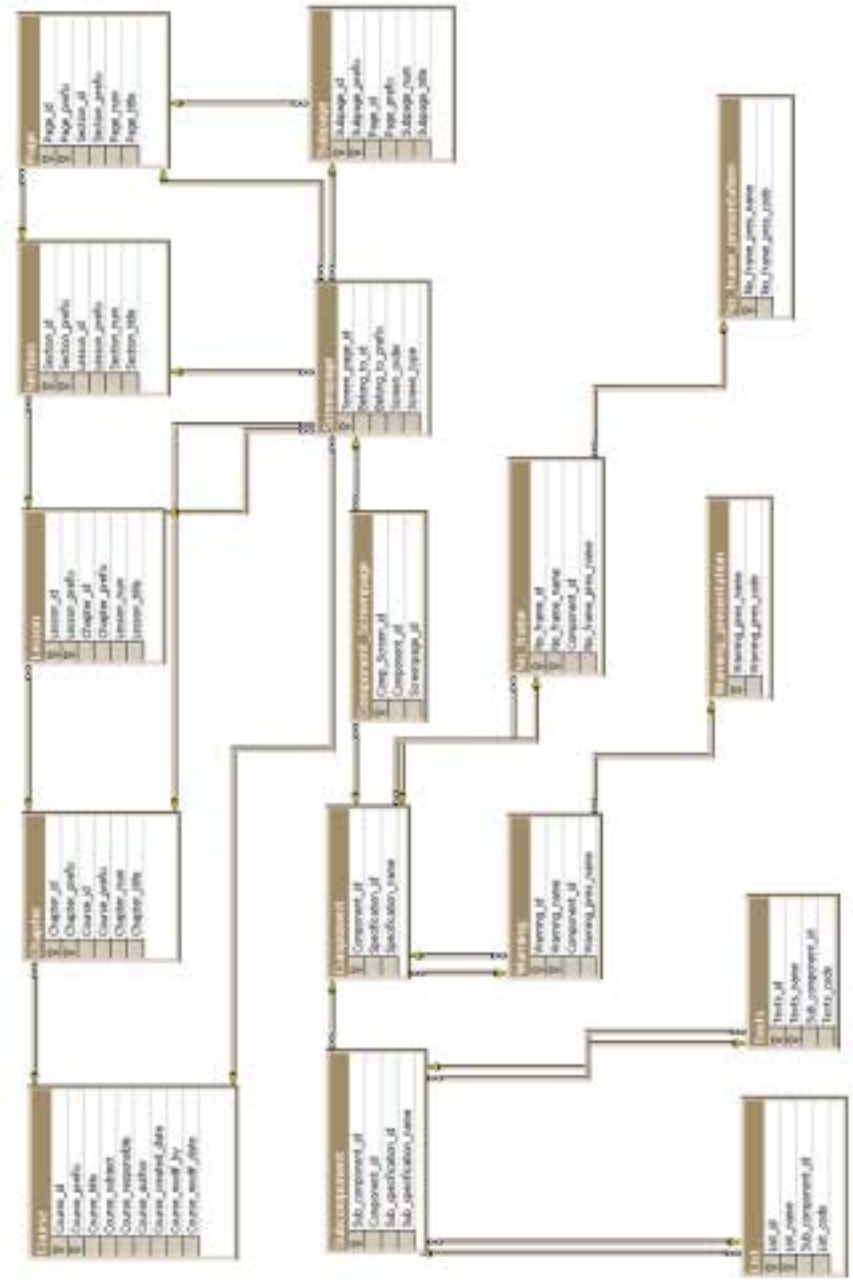
- *List*
- *Text*
- *Mouse_over*
- *Table*
- *Picture*
- *Heading*
- *File*
- *Execute*

Varje subkomponenttyp har en egen tabell och dessa kopplas till *Subcomponent*-tabellen med 1:1-koppling. En instans i en tabell för en viss *Subcomponent-typ* lagrar den HTML-formaterade koden som ger den aktuella subkomponenten.

Det logiska schemat har begränsats till att hantera subkomponenterna *Text* och *List* .Övriga subkomponenter kan implementeras i det fysiska schemat på samma sätt som dessa.

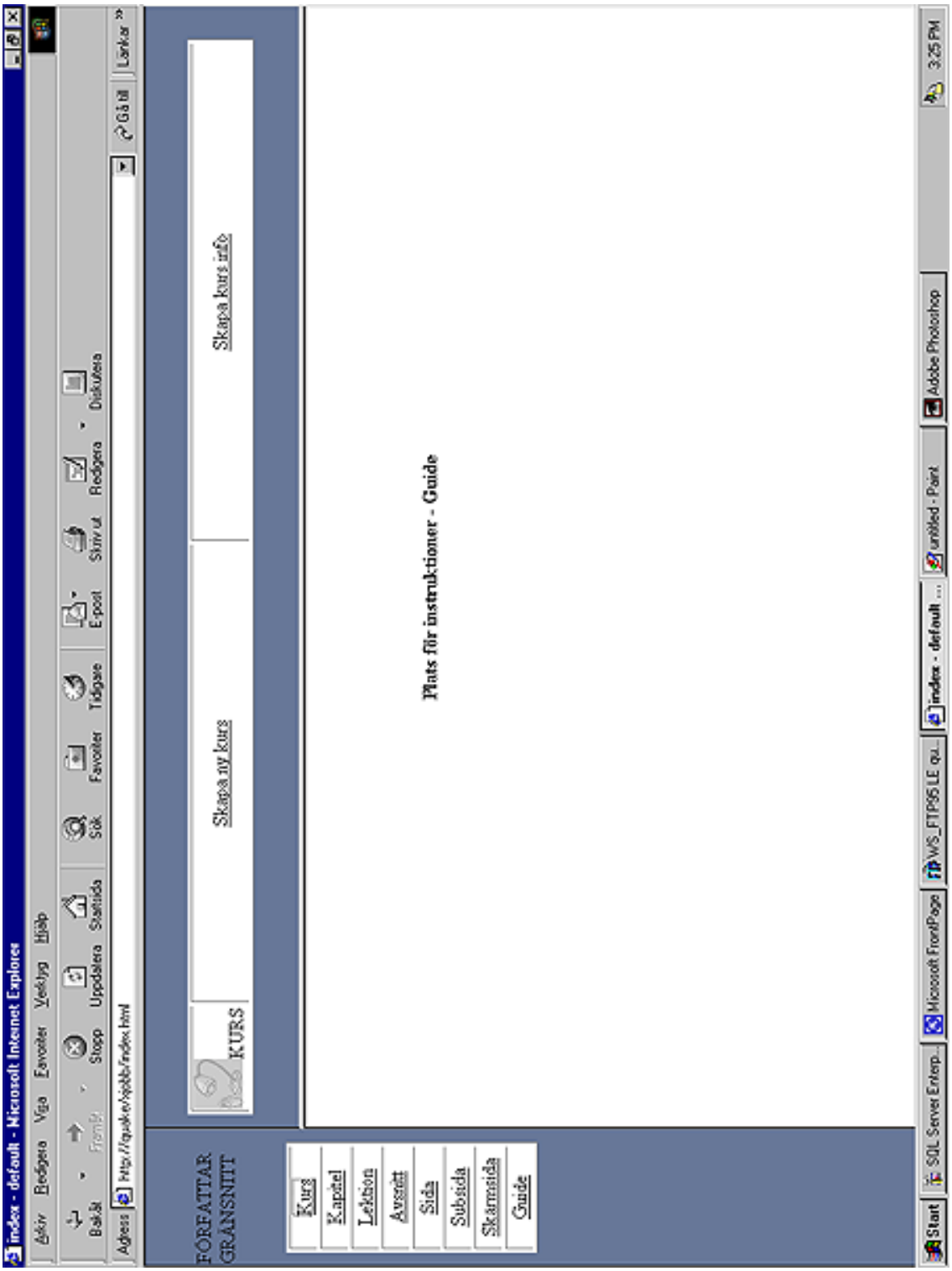
C Bilagor till fysisk design

C.1 Fysiskt schema

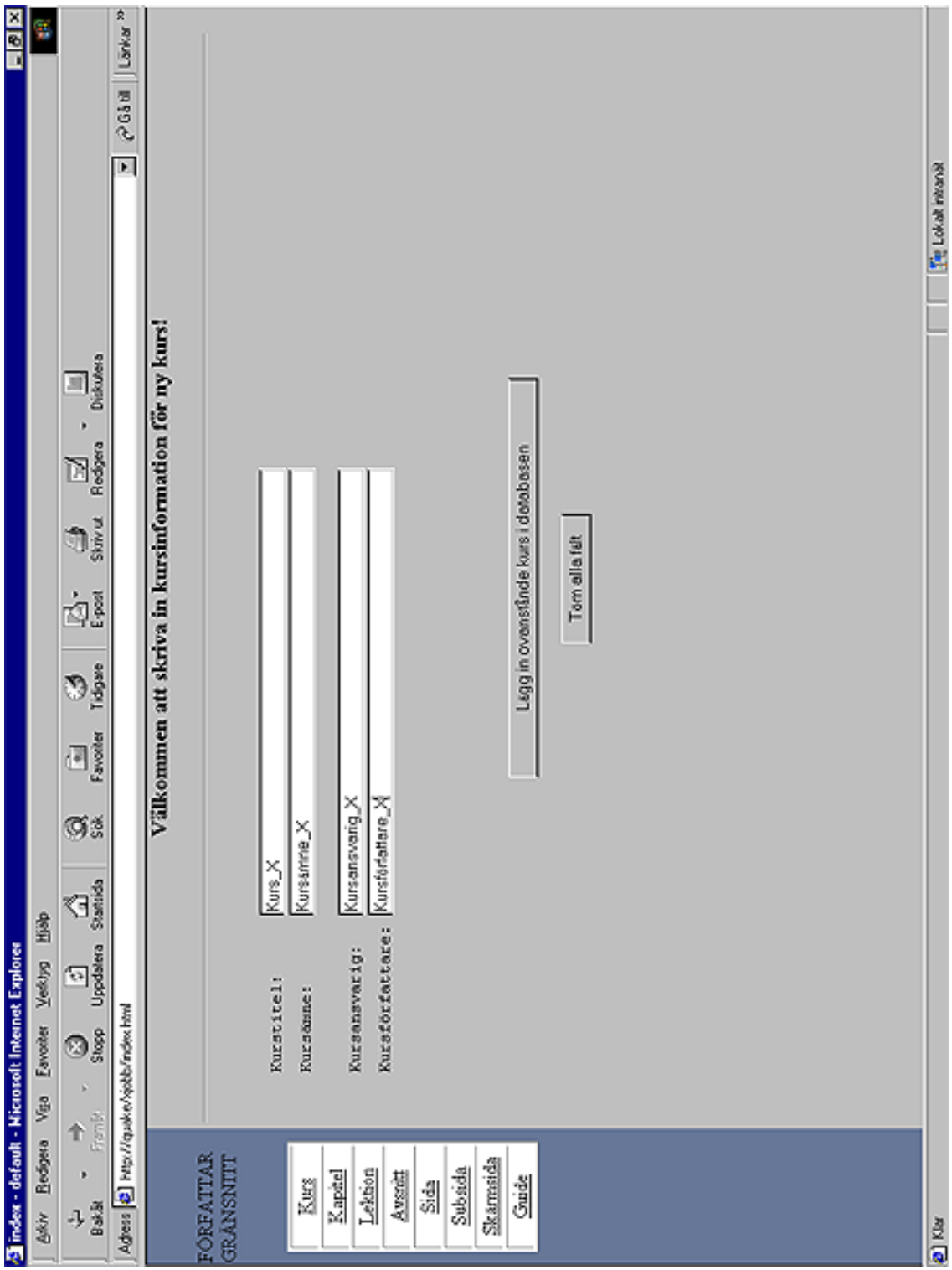


D Bilagor till gränssnittsdesign

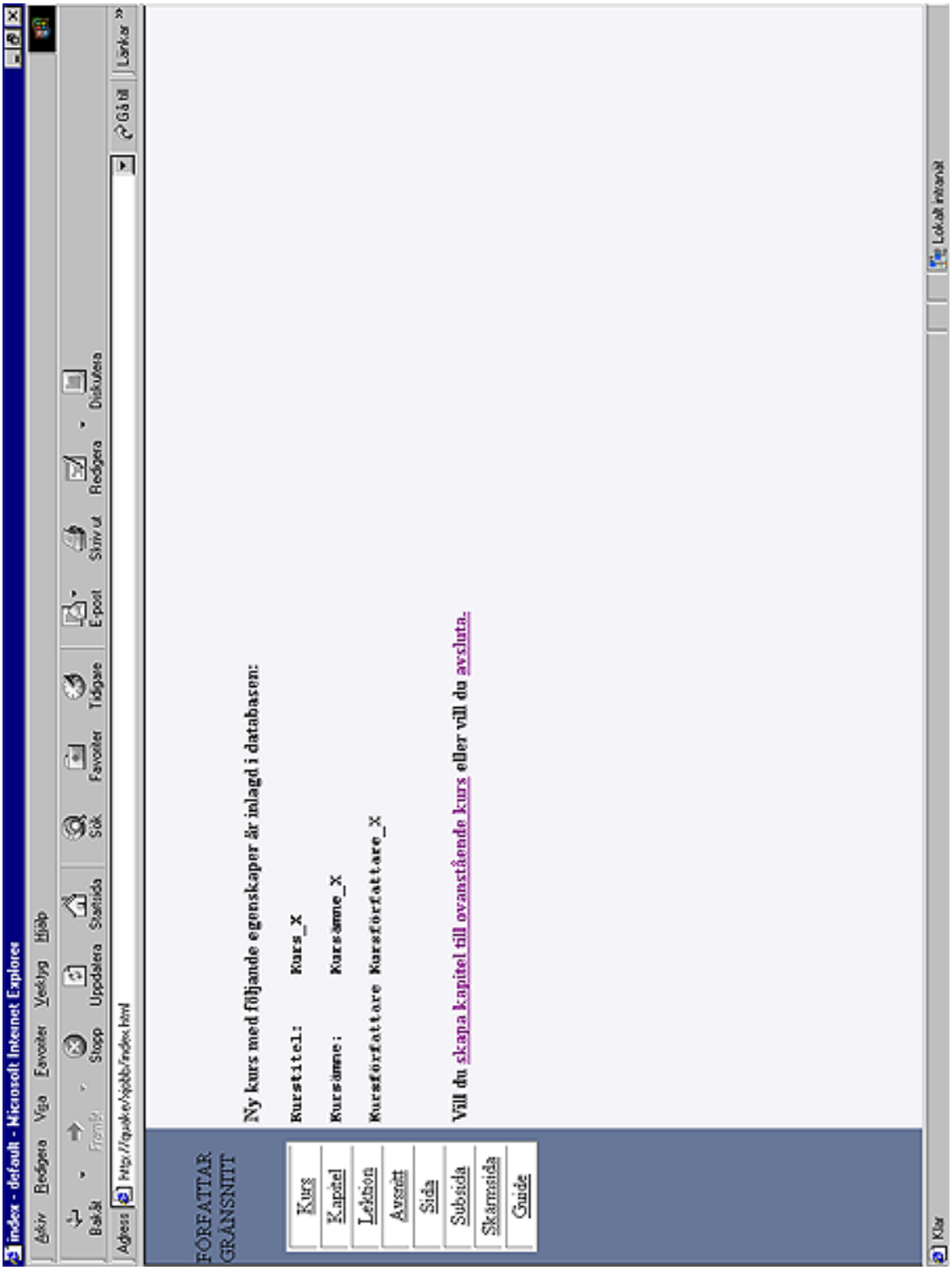
D.1 Gränssnitt: Meny



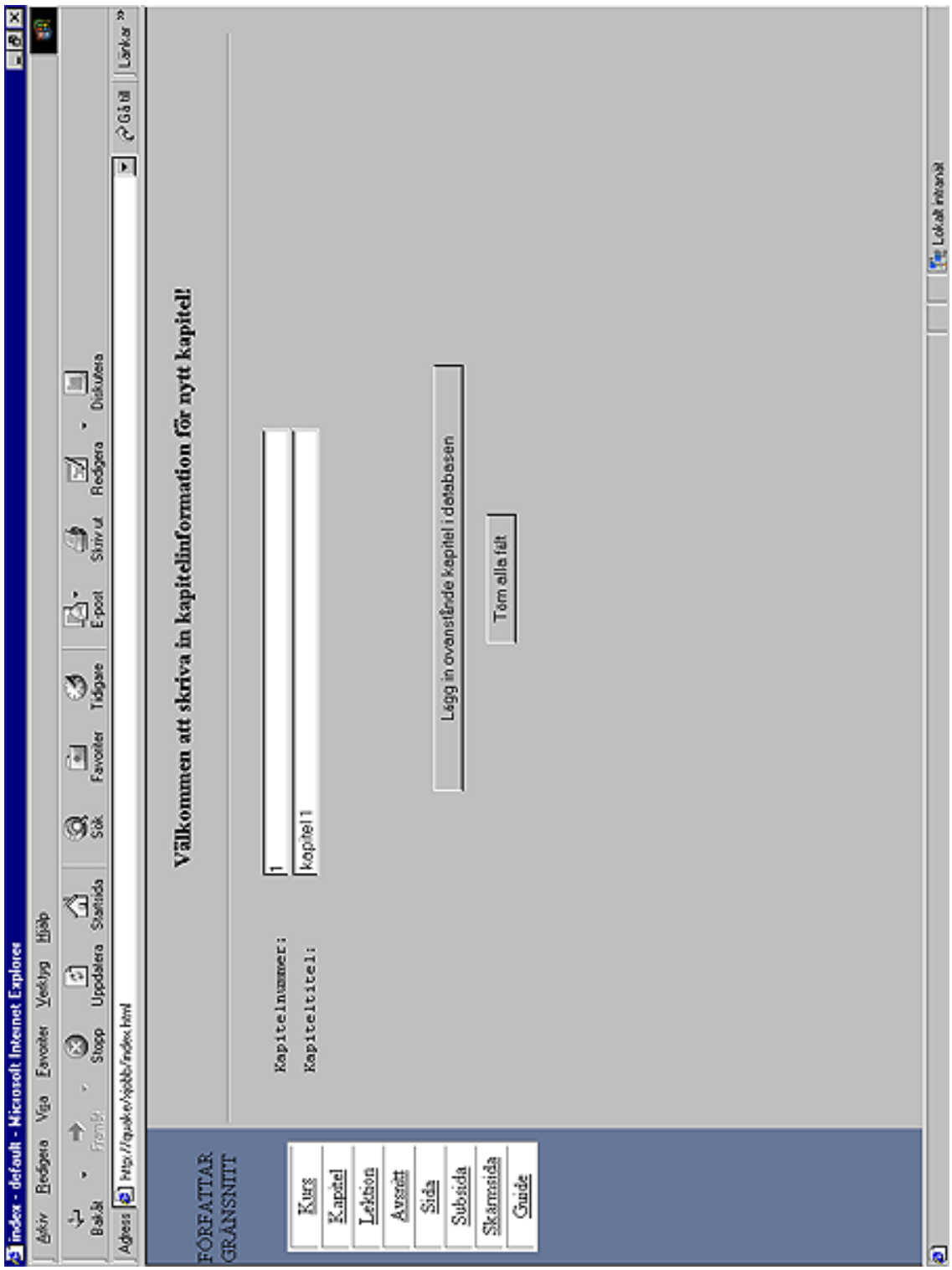
D.2 Gränssnitt: Inmatning av kurs



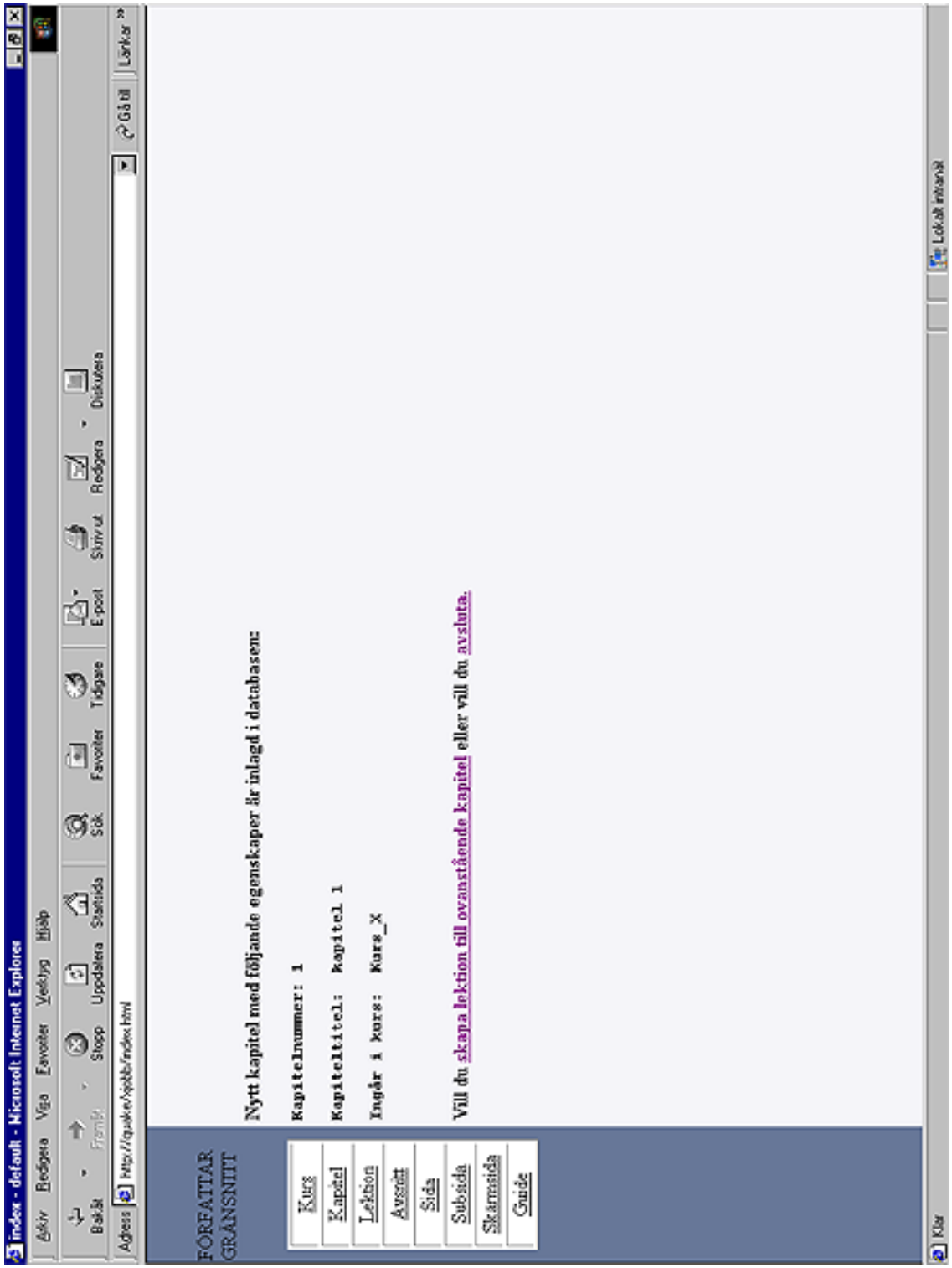
D.3 Gränssnitt: Svar på inmatning av kurs



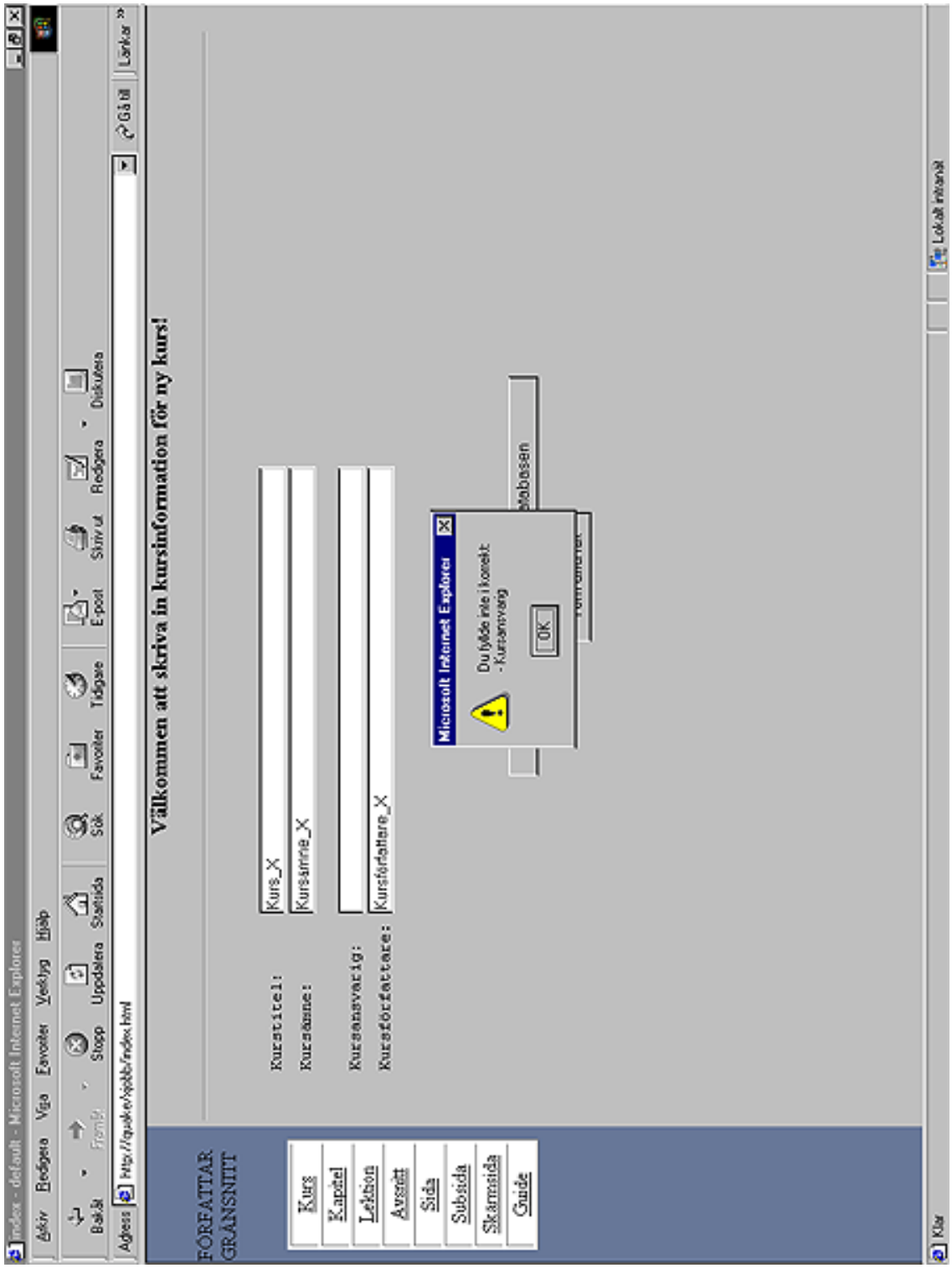
D.4 Gränssnitt: Inmatning av nytt kapitel



D.5 Gränssnitt: Svar på inmatning av nytt kapitel



D.6 Gränssnitt: Felaktig inmatning av kurs



D.7 Kod till formulär: form_save_Course.html

```
<!-- Nykursformulär-->
<html>
    <head>
      <title>Skriva in en ny kurs</title>
<!-- Validering av inmatade värden med JavaScript -->
      <SCRIPT LANGUAGE = "JavaScript">
      function checkForm(form)
      {
       var retval ="";
       if(!isNonEmpty(form.Course_title.value))
       {
         retval += "- Kurstitel\n";
       }
       if(!isNonEmpty(form.Course_subject.value))
       {
         retval += "- Kursämne\n";
       }
       if(!isNonEmpty(form.Course_responsible.value))
       {
         retval += "- Kursansvarig\n";
       }
       if(!isNonEmpty(form.Course_author.value))
       {
         retval += "- Kursförfattare\n";
       }
       if(retval != "")
       {
         alert("Du fyllde inte i korrekt:\n" + retval);
       }
       else
       {
         form.submit();
       }
      }
      function isNonEmpty(text)
      {
```

```
       if(text == "")
       {
         return false;
       }
       else
       {
         return true;
       }
      }
      </SCRIPT>
    </head>
    <body bgcolor="#C0C0C0" <br>
     <h3 align="center">Välkommen att skriva in kursinformation      för
  ny kurs!</h3>
      <hr>
<!-- Skapa formulär med fält för inmatning av attributvärden -->
     <form method=post action="save_Course.asp">
         <code><pre>
     Kurstitel:<input type=text name="Course_title" size=50 value="" >
     Kursämne:       <input type=text name="Course_subject" size=50
value="" >
     Kursansvarig:    <input type=text name="Course_responsible" size=50
value="">
     Kursförfattare:   <input   type=text   name="Course_author"   size=50
value="">
         </pre></code>
         <center><br> <br> <br>
<!-- Tryck innebär att formulärets värden läggs in i databasen-->
     <input type="button"  value="Lägg in ovanstående kurs i databasen"
onclick="checkForm(this.form)"><br><br>
        </center>
        <center>
     <input type="reset" value= "Töm alla fält" >
        </center>
       </form>
  </body>
</html>
```

D.8 Kod till formulär: form_save_Chapter.html

```
<!—Nytt kapitelformulär-->

<html>
   <head>
      <title>Skriva in ett nytt kapitel</title>
<!-- Validering av inmatade värden i JavaScript -->
      <SCRIPT LANGUAGE = "JavaScript">
      function checkForm(form)
      {
       var retval ="";

       if(!isNonEmpty(form.Chapter_number.value))
       {
         retval += "- Kapitelnummer\n";
       }

       if(!isNonEmpty(form.Chapter_title.value))
       {
         retval += "- Kapiteltitel\n";
       }

       if(retval != "")
       {
         alert("Du fyllde inte i korrekt:\n" + retval);
              }
       else
       {
         form.submit();
       }
      }

      function isNonEmpty(text)
      {
       if(text == "")
       {
         return false;
       }
       else
       {
```

```
          return true;
        }
       }

       </SCRIPT>

 </head>
   <body bgcolor="#C0C0C0"><br>
      <h3 align="center">Välkommen att skriva in kapitelinformation för
nytt kapitel!</h3>
       <hr>
<!-- Skapa formulär med fält för attributvärden -->
       <form method=post action="save_Chapter.asp" >
          <code><pre>
     Kapitelnummer:       <input type=text name="Chapter_number" size=50
value="">
     Kapiteltitel:         <input type=text name="Chapter_title" size=50
value="">
          </pre></code>
          <center> <br> <br> <br>

<!-- Tryck innebär att formulärets värden läggs in i databasen-->
     <input type="button"  value="Lägg in ovanstående kapitel i databasen"
onClick = checkForm(this.form)>
     <br><br>

          </center>
          <center>
     <input type=reset  value="Töm alla fält">
     </center>
      </form>
  </body>
</html>
```

E Bilagor till Webbpublicering

E.1 ASP-kod : save_Course.asp

```
<%@ Language=VBScript%>
<HTML>
<!-- Spara kurs -->
<%
dim kursid
’ Skapar databaskoppling:
Set DatabasKoppling = Server.CreateObject("ADODB.Connection")

' Öppnar databasen med metoden "Open":
DatabasKoppling.Open "Xjobb" , "SA" , ""

' Anropar SQL-proceduren insert_Course. Skapar anrop med
’textkontatenering, tex blir SQL= insert_Course @co_pre=
’'co',@co_tit='kursA'
SQL = "insert_Course "
SQL = SQL & "@co_pre='" & "co" & "',"
SQL = SQL & "@co_tit='" & Request.Form("Course_title") & "',"
SQL = SQL & "@co_sub='" & Request.Form("Course_subject") & "',"
SQL = SQL & "@co_res='" & Request.Form("Course_responsible") & "',"
SQL = SQL & "@co_aut='" & Request.Form("Course_author") & "'"

'Returvärdet från proceduren "insert_Course" läggs i variabeln ’kursid
  Set kursid=DatabasKoppling.Execute(SQL)

  'Använder Session-objektet för att spara variabelvärden under hela
  ’sessionen.
  Session("kursid_se")=kursid.fields(0)
  Session("kurstit_se")=Request.Form("Course_title")

  DatabasKoppling.close
  set DatabasKoppling=nothing
  %>
```

```
  <head>
    <title> Respons på att ny kurs är inlagd </title>
  </head>
  <body bgcolor="#F5F5F9">
  <br> <br> <br> <br>
  <B>Ny kurs med följande egenskaper är inlagd i databasen:
  <br>
  <pre>
  Kurstitel:     <%= Request.Form("Course_title")%>

  Kursämne:      <%= Request.Form("Course_subject")%>

  Kursförfattare <%= Request.Form("Course_author")%>

  </pre>
  Vill du <a  href="form_newchapter.html"  target="content">skapa kapitel
till ovanstående kurs</a>
  eller vill du <a  href="index.html"  target="_top">avsluta.  </B>
  </body>
  </html>
```

E.2 ASP-kod : save_Chapter.asp

```
  <%@ Language=VBScript%>
  <HTML>
  <!—Spara kapitel -->
  <%
  dim titel_kurs
  dim id_kurs
  dim kapitelid
  ' Skapa databaskoppling:
    Set DatabasKoppling = Server.CreateObject("ADODB.Connection")

  ' Öppna databasen med metoden "Open":
```

```
  DatabasKoppling.Open "Xjobb" , "SA" , ""

' Hämtar kurstitel och kursid: Finns som Sessions-variabler
titel_kurs = Session("kurstit_se")
id_kurs = Session("kursid_se")

' Anropa SQL-proceduren insert_Chapter.
SQL = "insert_Chapter "
SQL = SQL & "@cha_pre='" & "ch" & "',"
SQL = SQL & "@co_id=" & id_kurs & ","
SQL = SQL & "@co_pre='" & "co" & "',"
SQL = SQL & "@cha_num=" & Request.Form("Chapter_number") & ","
SQL = SQL & "@cha_tit='" & Request.Form("Chapter_title") & "'"

'Returvärdet från proceduren "inser_Chapter" läggs i variabeln kapitelid
Set kapitelid=DatabasKoppling.Execute(SQL)
'Använder Session-objektet för att spara variabelvärden under hela
’sessionen.
Session("kapitelid_se")=kapitelid.fields(0)
Session("kapiteltit_se")=Request.Form("Chapter_title")
DatabasKoppling.close
set DatabasKoppling=nothing
%>
<head>
  <title> Respons på att nytt kapitel är inlagd </title>
</head>
<body bgcolor="#F5F5F9">
<br> <br><br><br>
<B>
Nytt kapitel med följande egenskaper är inlagd i databasen:
<br>
<pre>
Kapitelnummer: <%= Request.Form("Chapter_number")%>
Kapiteltitel:  <%= Request.Form("Chapter_title")%>
Ingår i kurs:  <%= titel_kurs%>
</pre>
Vill du <a   href="form_newlesson.html">skapa lektion till ovanstående
kapitel</a>
eller vill du <a  href="index.html"  target="_top">avsluta.  </B>
</body>
</html>
```