Datavetenskap

Patrik Grenfeldt och Anders Forssten

Skapandet av ett grafiskt användargränssnitt i Visual Basic till en befintlig databas

---

Examensarbete

2003:20

Skapandet av ett grafiskt användargränssnitt i Visual Basic till en befintlig databas

Patrik Grenfeldt och Anders Forssten

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

Patrik Grenfeldt

Anders Forssten

Godkänd, Date of defense

Handledare: Christer Andersson

Examinator: Stefan Lindskog

Sammanfattning

Vi har på uppdrag av Lernia i Karlstad konstruerat ett nytt användargränssnitt och affärslogik till en befintlig databas för hantering av information om kursdeltagare. Uppgiftens ursprung är en systemkonstruktion, som lider av en gradvis dekadens i funktionalitet. Detta på grund av icke bakåtkompatibla uppgraderingar av systemmiljön. Rapporten är en grundlig genomgång av det befintliga systemet samt utvecklandet av det nya systemet. Det ursprungliga systemet är utvecklat i databassystemet Access och det framtagna systemet har utvecklats i Visual Basic 6.0. Vi har även granskat systemets möjligheter att utvecklas vidare.

The Creation of a graphical user interface in Visual Basic for an existing database

On behalf of Lernia in Karlstad, we have constructed a new user interface and business logic to an existing database system for handling student information. The task's origin is a system design that suffers from a gradual decadence in functionality due to system upgrades, which are not backward compatible. This report investigates the existing system in detail, and also describes the developement of the new system The original system was developed in the database system Access and the new system has been developed in Visual Basic 6.0. We have also examined the possibilities to further develop the system.

Tack

Erik Lindgren, för din hjälp med datamaskering.

Lernia i Karlstad som stått för vårt uppdrag.

Stefan Forssten, för all assistans med VB.

Jenny Björfeldt och Henrik Sjölin, för felrättningar.

Per Strömgren, för dina råd om svenskabruk.

Christer Andersson, för all din tid med den här rapporten.

Innehåll

Figurer

Tabeller

1 Inledning

1.1 Bakgrund

Vi har utfört vårt examensarbete på Lernia, som är ett av Sveriges största utbildningsföretag. Lernia har utbildningar för personer som är arbetslösa, personer som behöver höja sin kompetens inom ett specifikt område samt för företag som har behov att kompetensutveckla sina anställda. Lernia håller kurser i vitt skilda områden som exempelvis datakunskap samt plåt och svets. För att hålla reda på vilka kurser och kursmoment en viss studerande har deltagit i och dennes resultat har systemutvecklarna på Lernia skapat ett system där denna typ av information lagras. Från detta system kan användare av systemet sedan skriva ut intyg som visar att en kursdeltagare gått en viss kurs respektive vilka moment denne klarat av.

Systemet används i skrivande stund av två personer på två olika avdelningar. Systemet är uppdelat i två delar, där den ena delen är själva relationsdatabasen innehållande all information och den andra delen är ett gränssnitt till databasen skapat i Access.

När användare installerat nyare versioner av Access har systemunderhållare fått konvertera den grafiska presentationen. För närvarande körs Access 2000 på en plattform baserad på operativsystemet Microsoft Windows 2000. Relationsdatabasen har inte förändrats nämnvärt sedan systemet började användas. Databasdelarna ligger på en nätverksenhet som alla som behöver använda systemet kommer åt.

1.2 Problem

Systemet fungerar i nuläget endast med Access på grund av att det grafiska gränssnittet är gjort i Access och därmed inte är en fristående applikation. Detta medför att Access måste finnas installerat för det ska gå att använda systemet. Dessutom uppstår det problem i gränssnittet då det från början var konstruerat i en tidig version av Access och behöver justeras varje gång byten till nyare versioner sker. Detta har lett till en rad nödlösningar som

blir allt svårare att ändra och hantera efter varje versionsbyte. Detta yttrar sig exempelvis genom att felmeddelanden dyker upp då användare öppnar inmatningsformulär samt vid val av vissa menyalternativ. Detta är ett stort problem eftersom vissa av dessa fel gör att körningen avbryts. Dessutom uppstår problem om olika versioner av Access används eftersom databasen inte är fullt bakåtkompatibel och inte fungerar i t.ex Access 97 om den konverterats för att användas i Access 2000.

Ännu ett problem är att gränssnittet är föråldrat och inte särskilt smidigt att använda p.g.a knappar med otydlig innebörd samt ett menysystem som ligger kant i kant med Access egna menyer som har liknande menyikoner (se figur 2.2 på sidan 7).

1.3 Syfte

På grund av dessa ovan beskrivna Access-relaterade problem vill uppdragsgivaren att databasens gränssnitt skall implementeras i Visual Basic 6.0 (VB6) eftersom det då går att vidareutveckla databasen med befintliga kunskaper.

Uppdragsgivaren vill även att systemet ska ha framtidsvärde och bör vara implementerad med moderna databaskomponenter (se krav 2 i avsnitt B.1 på sidan 38 för detaljer om kravet).

1.4 Mål

Databasen ska fungera fristående från Access med samma funktionalitet som innan.
Vi har delat in uppgiften i ett antal delmål för att kunna prioritera vissa delar och eventuellt utelämna andra. Sekundärmålen görs i mån av tid.

- Primärmål
  - Skapa ett nytt grafiskt användargränssnitt med hjälp av Visual Basic 6.0, vilket motsvar gränssnittet i Access-versionen. Detta innebär att gränssnittet ska innehålla *formulär* (se appendix A, sidan 35 för introduktion av begreppet for-

mulär i VB6) motsvarande de gamla formulären i det gamla systemet. Gränssnittet ska vara smidigare att använda jämfört med det gamla systemet med tydligare knappar och menysystem.

  - Skapa logiken mellan databasen och användargränssnittet, d.v.s funktionalitet för *datahygien* (se appendix A på sidan 35 för introduktion av begreppet datahygien) och databasåtkomst.

- Sekundärmål

  - Skapa utskriftsrapporter. Skapa logik för automatiskt utskriftsupplägg innehållande information om en kursdeltagare och de kurser inklusive kursmoment denne har slutfört. Ett antal olika upplägg ska vara tillgängliga för användaren att göra utskrifter med.

  - Arkivera databasinnehåll, d.v.s flytta poster som har skrivits ut, från databasen till en annan databas (ett arkiv). Detta för att mängden data i systemets databas ska hållas hanterbar.

1.5 Avgränsning

Vi avgränsar oss från att konstruera ett system med ett ”perfekt” användargränssnitt då de kosmetiska justeringarna tar lång tid och detta utan att utöka det funktionella värdet nämnvärt. Tyngden läggs istället på att försöka skapa ett funktionellt gränssnitt snarare än utseende och därigenom kommer användargränssnittet att förbättras på vissa punkter där användarvänligheten kan utökas, se sektion 2.

1.6 Förutsättningar

- *Utvecklingssystem:* Windows 2000, Service Pack 2

- *Målsystem:* Windows 95, Windows 2000, Windows XP

- *Utvecklingsmiljö/Programmeringsspråk:* Microsoft Visual Basic 6.0, Service Pack 4

- *Databas:* Ursprungligen skapad med Microsoft Access 2, senare konverterad med nyare versioner t.o.m. Access 2000.

1.7 Läsanvisningar

- I kapitel 2 kommer konstruktionslösningen att presenteras. Här analyseras det ursprungliga gränssnittet och databasen. Detta ligger till grund för kraven som går att hitta i appendix B.1

- I kapitel 3 kommer implementationen presenteras. Denna har tagits fram för att genom implementation av konstruktionslösningen presenterad i kapitel 2 uppfylla de krav som ställts på det färdiga systemet.

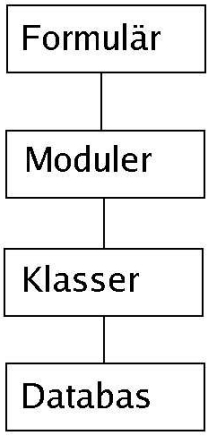
- I kapitel 4 återfinns en beskrivning av hur systemet ska testas för att se om det i sin implementation uppfyller de ställda kraven.

- I kapitel 5 återges erfarenheter och rekommendationer, erhållna från utvecklingsarbetet, samt en utvärdering av arbetet.

2 Beskrivning av konstruktionslösningen

I detta kapitel visar och beskriver vi det ursprungliga gränssnittet, som utvecklats i Access samt gör en genomgång av den programmeringsmodell som systemet kommer att implementeras efter. Vi beskriver det ursprungliga gränssnittets funktionalitet, dess komponenter och visar på punkter där gränssnittet fungerar ologiskt eller har utvecklingsbar användarvänlighet. Endast primärmålen kommer att ingå i konstruktions- och implementationslösningen.

2.1 Upplägg för vår applikation



Figur 2.1: Konstruktionslösningens lagerstruktur

Konstruktionen är uppdelad i tre delar: *formulär*, *moduler* och *klasser*. Dessa tre delar utgör programmet som kommunicerar med databasen, se figur 2.1. För mer information om de olika programspråkstermerna se appendix A på sidan 35.

`Formulären` i konstruktionslösningen innehåller alla visuella komponenter och utgör det grafiska gränssnittet mot användaren. De innehåller ingen logik, utan använder sig av *funktioner* och *procedurer* (se se appendix A på sidan 35 för information om funktioner och procedurer) i moduler för att utföra handlingar. När vi har skapat formulären har vi utgått från det ursprungliga gränsnittet och sedan försökt att förbättra det, men samtidigt sparat mycket av den ursprungliga designen. I flera av formulären finns både menyer och en verktygsrad som finns för att förenkla nyttjandet av systemets funktionalitet.

`Modulerna` i konstruktionslösningen innehåller all logik för gränssnittet och varje modul har instanser av de klasser som behöver användas i just den modulen. Modulerna använder sig av klasserna för att utföra operationer på databasen genom att anropa

metoder (se appendix A på sidan 35) och sätta *egenskaper* (se nedan) i motsvarande klass.

`Klasserna` i konstruktionslösningen är den del som kommunicerar med databasen och innehåller alla de metoder som läser och sparar i databasen. Man skulle kunna säga att klasserna utgör ett gränssnitt mot databasen och varje klass innehåller egenskaper (s.k properties, se appendix A för information) som implementeras motsvarande attributen i aktuell tabell. Vissa klasser har även extra egenskaper för att ta reda på information som kan hittas längs en relation.

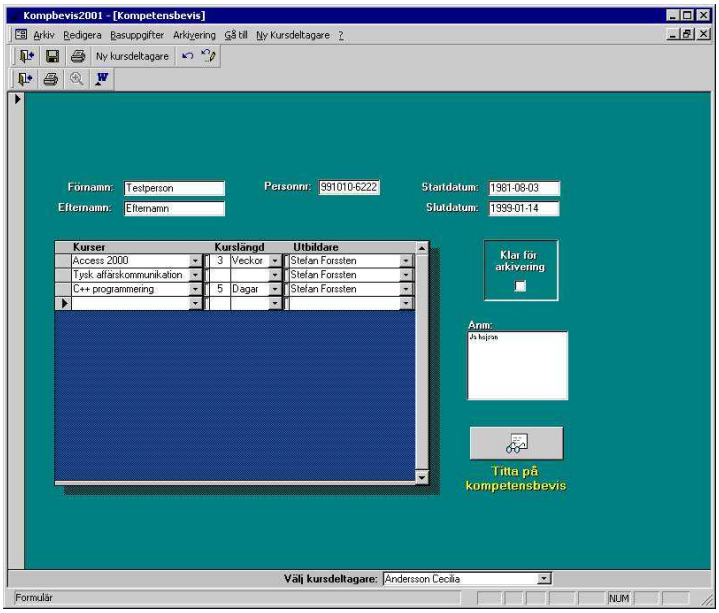
Det finns många brister i den ursprungliga databasen och vi har designat om delar av den för att den ska fungera bättre och bli lättare att förstå.

2.2 Det ursprungliga gränssnittet

I följande avsnitt går vi igenom alla formulär i det gamla systemet. Genomgången inkluderar funktionalitet samt menysystem.

Formuläret `KDKompbevis` är uppdelat i två delar, en övre och en undre. Med upplösningen 800x600 ser man endast en del i taget och vad som avgör vilken del som visas beskrivs nedan. Vid högre upplösningar ses hela eller delar av det som för tillfället inte ska synas. Denna begränsning kommer att undvikas i konstruktionslösningen genom att skapa två fristående formulär.

Formuläret `KDKompbevis` övre del (se figur 2.2) är det formulär som visas automatiskt när användaren startar systemet. Formuläret är uppbyggt av ett antal grafiska komponenter för visning och inmatning av data. Användaren får välja en kursdeltagare med hjälp av rullgardinslistan längst ner i formuläret. Information om kursdeltagaren visas i textrutorna och i den stora grafiska komponenten, som kan liknas vid en *DataRepeater* (se appendix A), i mitten av formuläret.



Figur 2.2: Accessformuläret `KDKompbevis` övre del

Inmatning av data sker direkt i dessa rutor och komponenter samt att datan kontrolleras för att få rätt format för lagring i databasen. Exempel på detta är inmatningen av startdatum, som måste vara av ett korrekt format samt anmärkningen som får vara max 100 tecken lång.

Inmatad data sparas automatiskt utan att användaren behöver göra ett explicit val. Detta gör spara-knappen i verktygsfältet överflödig. Knappen ovanför texten ”Titta på kompetensbevis” visar den undre delen av `KDKompbevis` (se figur 2.3).

Menysystemet innehåller huvudalternativen `Arkiv`, `Redigera`, `Basuppgifter`, `Arkivering`, `Gå till`, `Ny kursdeltagare` och `?`, vilka alla kommer att beskrivas översiktligt nedan.

Bild-knappen t.v. om `Arkiv` frambringar menyalternativ för fönsterhantering, exempelvis för att maximera eller minimera fönster. Detta är en knapp som kommer med på grund av att det är ett Access-gränssnitt och den existerar utan att höja använ-

darvänligheten, då dessa val naturligt görs med de tre små ikonerna längst upp till höger i stort sett alla Windowsapplikationer. Den exkluderas från konstruktionslösningen i denna form och funktionalitet för att minimera och återställa kommer att implementeras på sin naturliga position d.v.s längst upp till höger i fönstret. Alternativet för att maximera implementeras ej p.g.a att det uppstår förskjutningar i komponenternas placeringar om de placeras dynamiskt och användargränssnittet får då ett utseende som inte kan förutses och därmed blir svårt att planera för.

Under menyalternativet `Arkiv` finns `Spara` och `Avsluta`, vilka sparar aktuell information i databasen respektive avslutar systemet. Dessa alternativ implementeras med primärmålen. Vidare finns `Skrivarinställning` och `Skriv ut`, vilka hör till sekundärmålen.

Under alternativet `Redigera` finns `Ta bort`, som tar bort aktuell kursdeltagare. Detta alternativ implementeras med primärmålen. Vi anser dock att detta alternativ bör ligga under `Arkiv` eftersom det är en funktion som medför en ändring i databasen. Vidare finns alternativen `Ångra` och `Ångra fält` som har en och samma funktion d.v.s att ångra senaste inmatning i ett textfält. Dessa ersätts av oss genom att användaren kan välja att inte spara när han/hon byter post. Alternativet `Design` som tar användaren till Access designläge implementeras inte eftersom motsvarighet saknas i VB6. `Markera post` som markerar aktuell post ger ingen användarfunktionallitet och implementeras ej.

Under `Basuppgifter` finns valen `Kurs` och `Utbildare`, som visar formulären figur 2.4 respektive figur 2.5. Dessa alternativ implementeras med primärmålen.

Under `Arkivering` finns `Arkivera utskrivna kd`, som flyttar alla poster som är markerade i kryssrutan "Klara för arkivering" till en separat databas. Detta underlättar hanteringen av databasen eftersom antalet poster i huvuddatabasen minskar. Denna funktion tillhör sekundärmålen.

Under alternativet `Gå till` finns `Första`, `Sista`, `Nästa` och `Föregående` som inte kommer att implementeras som menyalternativ i vår applikation. Vi anser att de är överflödiga eftersom användaren lätt kan markera en kursdeltagare i listan. Därutöver markerar alternativen `Första` och `Sista` inte den kursdeltagare som har ett efternamn som placerar denne först i listan, utan istället den som har lägsta respektive högsta interna identifikationsnummer i databasen vilket inte är logiskt ur ett användarperspektiv.

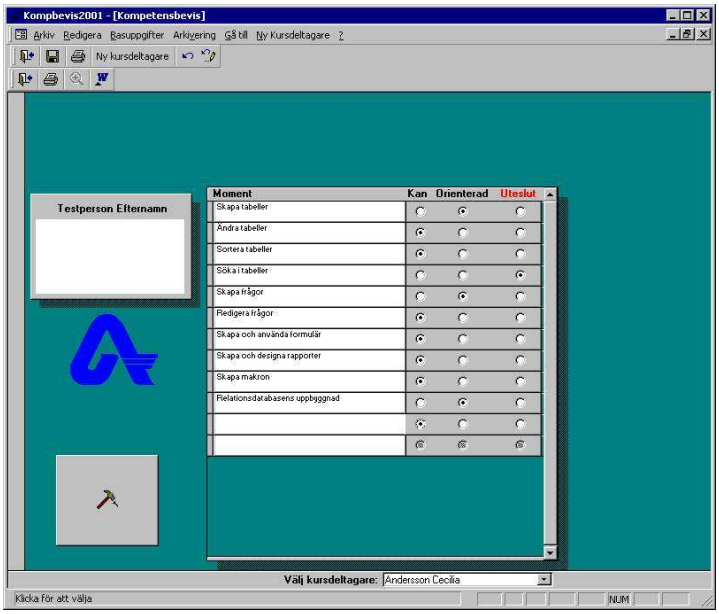
Under `Ny kursdeltagare` finns `Lägg till ny kd` vilket skapar en tom post. Denna funktion är grundläggande för databasens funktionalitet och alternativet bör ligga under `Arkiv` eftersom det medför en ändring i databasen. Menyalternativet och funktionaliteten implementeras med primärmålet.

Under alternativet `?` finns ett menyalternativ för hjälp om Access (inte systemet i sig). Vi kommer inte att implementera denna typ av hjälp i vår applikation. Inget krav på hjälpfunktionalitet finns, men vi kommer att implementera hjälptexter som visas när användaren håller muspekaren över knappar eller liknande komponenter.

Verktygsfältet innehåller ett antal knappar för hantering av databasen. De två knapparna med en dörr på stänger formuläret och tar användaren till Access designläge. Knapparna med skrivare på är till för utskrifter och avgränsas till sekundärmålen. Knappen med en Word-symbol kopierar data för att infogas i ett Microsoft Officedokument. Detta är inte en funktion som kommer att implementeras då den måste avgränsas p.g.a tidsvinst gentemot affärsvärde. En knapp med ett förstoringsglas finns i fältet, men är aldrig tillgänglig, d.v.s den är alltid grå och går aldrig att använda och implementeras därför inte. Vidare finns knappar för att spara, ångra inmatning, samt skapa en ny kursdeltagare och dessa implementeras enligt menyalternativen.

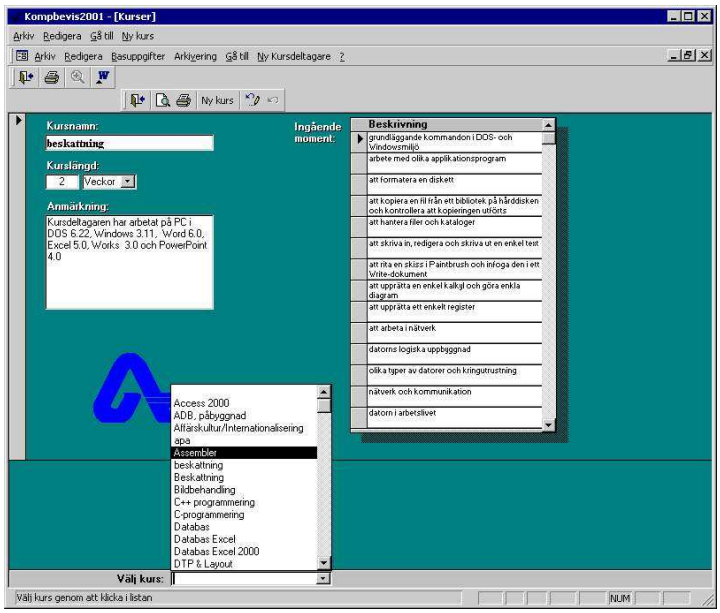
Vi anser att både menyerna och verktygsfälten är väldigt otydliga och röriga sett ur ett användarperspektiv och vi kommer i det nya systemet att strukturera dem

enhetligare med övriga Microsoft applikationer. Ångra-funktionen känns omotiverad då användare endast tillåts skriva i ett fält i taget utan att spara ändringar. Ångra-funktionen kan därigenom endast ångra ett steg och den implementeras ej.



Figur 2.3: Accessformuläret `KDKompbevis` undre del

Formuläret `KDKompbevis` undre del (se figur 2.3) visar information om en kursdeltagares bedömningar för de olika momenten som ingår i aktuell kurs. Detta formulär används också när användaren ska ange vilken bedömning kursdeltagaren fått på varje delmoment. Momenten och bedömningarna visas i den grafiska komponenten i mitten av formuläret. Den tomma textrutan (till vänster i figuren) innehåller ingen information alls, utan är en rest från en gammal version av databasen. Denna textruta kommer att uteslutas ur vår applikation eftersom den inte fyller någon funktion. Den stora knappen nere till vänster tar användaren tillbaka till `KDKompbevis` övre del och att denna knapp ska ha en hammare som symbol ser varken vi eller uppdragsgivaren någon logik i. Menyraden och verktygsfältet är detsamma som i formulärets övre del, se ovan för information.



Figur 2.4: Accessformuläret `Kurser`

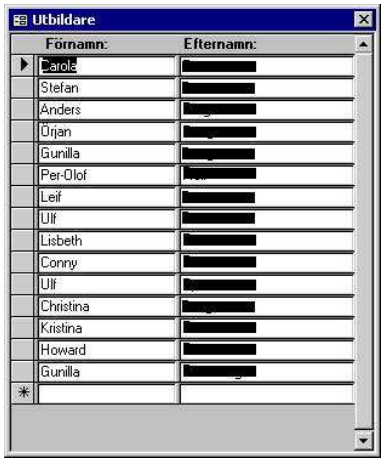
Formuläret `Kurser` (se figur 2.4) används för att hantera kurser med tillhörande kursmoment. I listan längst ner på formuläret väljer användaren kurs, och information om aktuell kurs visas i övriga delar av formuläret. Listan till höger i figuren innehåller alla delmoment för kursen och textfälten till vänster visar kursinformation.

Formulärets menysystem innehåller dubbla menyrader, varav den undre är exakt samma som i formuläret `KDKompbevis`. Uppdragsgivaren anser att den inte hör hemma i formulären och att den hamnat där p.g.a nödlösningar samt tidsbrist. Den övre menyraden är den som vi anser hör till kursformuläret och beskrivs nedan.

Under valet `Arkiv` finns menyalternativen `Stäng`, som stänger formuläret samt `Skrivarinställning`, `Förhandsgranska` och `Skriv ut`. Funktionen för att förhandsgranska fungerar inte och kommer inte att implementeras av oss i den nya applikationen eftersom det hör till sekundärmålen, liksom utskriftsfunktionen. `Redigera` och `Gå Till` innehåller samma alternativ och funktionalitet som i `KDKompbevis`, se ovan. Under `Ny Kurs` finns alternativet `Lägg till ny kurs` som skapar en tom kurspost

i databasen.

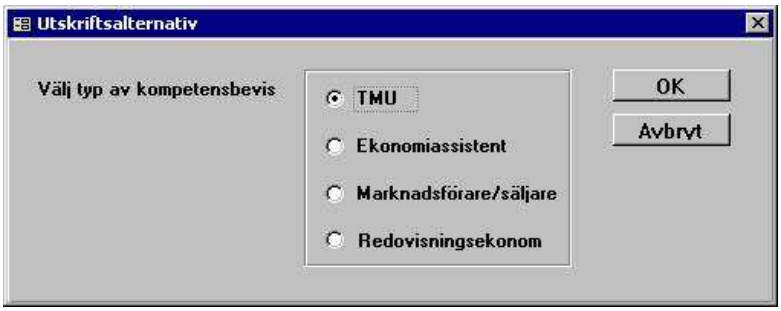
Verktygsfältet är på många sätt likt det i `KDKompbevis` och lider av samma problem, d.v.s. funktioner som känns onödiga (t.ex knappen som används för att avsluta och gå till Access designläge, se tidigare avsnitt) och alternativ som aldrig är tillgängligt (förhandsgranska).



Figur 2.5: Accessformuläret `Utbildare`

Formuläret `Utbildare` (se figur 2.5) visar de utbildare som finns registrerade i databasen och låter användaren ändra eller ta bort befintliga utbildare samt lägga till nya utbildare. I databasen finns ett fält kallat ”Ämne” där användaren kan ange vilket ämne utbildaren håller kurser i. Detta fält kan inte ändras från det ursprungliga användargränssnittet utan måste ändras genom att ändra direkt i databasen. Vi kommer att implementera en funktion som låter användaren skriva in ett ämne direkt i formuläret och samtidigt försöka göra hela formuläret mera användarvänligt och lättanvänt.

I formuläret `Utskriftsalternativ` (se figur 2.6) väljer användaren vilken typ av kompetensbevis som ska skrivas ut. Detta formulär hör till sekundärmålen.

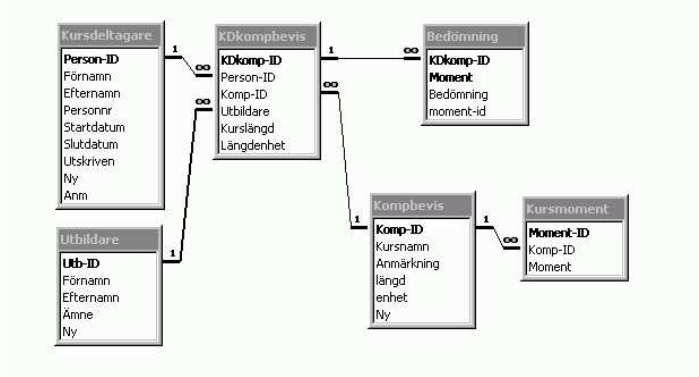


Figur 2.6: Accessformuläret `Utskriftsalternativ`

2.3 Databasscheman/relationer

De två relationsdiagrammen nedan visar det ursprungliga utseendet och hur det ser ut efter att vi arbetat om databasen. Namnen på flera tabeller och fält har ändrats för att öka läsbarheten och för att skapa konsekvens i namngivningen. Ett antal onödiga fält har tagits bort, vilket förutom att göra databasen lättare att hantera även har minskat databasens storlek till mindre än halva originalstorleken.

2.3.1 Ursprunglig design



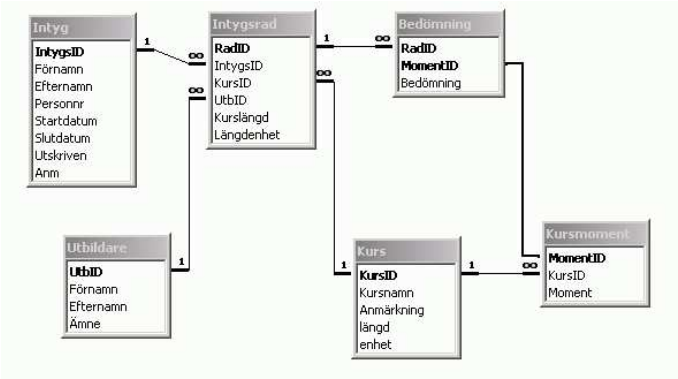
Figur 2.7: Ursprungligt databasschema/relationer

Det ursprungliga databasschemat (se figur 2.7) innehåller många tabeller och attribut

med oklar innebörd och det finns även fall där namngivningen är inkonsekvent. Dessutom finns fält som inte används samt dubbla uppsättningar av en del information. Därför har vi ändrat på namn för att få konsekvens och tydliggöra samt tagit bort vissa attribut och alla förändringar har skett av eller i samråd med uppdragsgivaren. I avsnitt 2.3.2 visar vi de ändringar vi gjort och hur det nya databasschemat ser ut.

2.3.2 Framtagen design

Figur 2.8 och tabell D.1 i appendix D visar de ändringar som gjorts i databasen. Många tabeller och attribut har fått nya namn för att förtydliga vad dess poster har för syfte att modellera. Alla tabell- och fältnamn innehållande "-" har fått nya namn på grund av svårigheter i Visual Basic att hantera namn innehållande detta tecken.



Figur 2.8: Framtaget databasschema/relationer

Tabellen `Intyg` innehåller poster med information om varje kursdeltagare. Här lagras personuppgifter om kursdeltagaren samt information om när personen påbörjade och eventuellt avslutade sin utbildning på Lernia. Om en kursdeltagare fått sitt utbildningsintyg utskrivet ska detta reflekteras i *Utskriven* för att systemet ska veta att denna person är aktuell för arkiveringsrutinen (se sekundära krav 2 på sidan 40).

Tabellen `Intygsrad` innehåller poster med information om varje kurs för en viss kursdeltagare. En kursdeltagare är en person som är registrerad på en eller flera kurser hos

Lernia. För varje kurs en person registreras på skapas en post i tabellen `Intygsrad`. I posten sparas information om vem som utbildades, vem som var utbildare på kursen samt vilken kurs som lästes.

Tabellen `Utbildare` innehåller poster med information om varje utbildare. Varje post innehåller information om vad utbildaren heter i för- och efternamn samt vilket ämne som personen håller utbildning i.

Tabellen `Kurs` innehåller poster med information om varje kurs. Varje post innehåller information om vad en kurs heter, kursens tidsomfattning samt ett fält där övrig information kan lagras. Även om fälten `Längd` och `Enhet` är snarlika fälten `Kurslängd` och `längdenhet` så skiljer de sig genom att det som lagras här är specifikt för dessa kurser medan den data som lagras i tabellen `Intygsrad` är specifikt för en kursdeltagares utbildningstillfälle. Därför har uppdragsgivaren önskat bibehålla denna fältstruktur.

Tabellen `Kursmoment` innehåller poster med information om varje kursmoment tillhörande en viss kurs. I varje post lagras information om till vilken kurs ett moment hör samt en beskrivning av momentet.

Tabellen `Bedömning` innehåller poster med information om bedömning av en kursdeltagares prestation i varje kursmoment tillhörande en viss kurs.

Samtliga tabellers attribut finns beskrivna mer ingående i appendix D.

2.4 Summering

Analysen av databasen och de förändringar som gjorts har gjort databasen betydligt mindre i storlek, vilket kommer att ge en prestandaförbättring då en mindre mängd data måste hämtas från disk. Databasen har även blivit mera lättanvänd då tabeller och attribut har fått namn som tydligare beskriver deras funktion och kopplingen mellan de olika tabellerna är lättare att se och förstå. Dessutom har vi minskat redundansen i databasen vilket minskar risken för inkonsekvens i datan.

Efter analysen av det gamla gränssnittet och diskussion med uppdragsgivaren finns en

bra grund att stå på när det nya gränssnittet ska skapas. Vi har en bra uppfattning om hur det nya systemets gränssnitt ska se ut och vi kommer att implementera det med den lagerstruktur som beskrivits ovan. I kapitel 3 beskriver vi i detalj hur vi implementerat det nya systemet och hur alla komponenter arbetar och kommunicerar med varandra.

3 Beskrivning av implementation

3.1 Introduktion

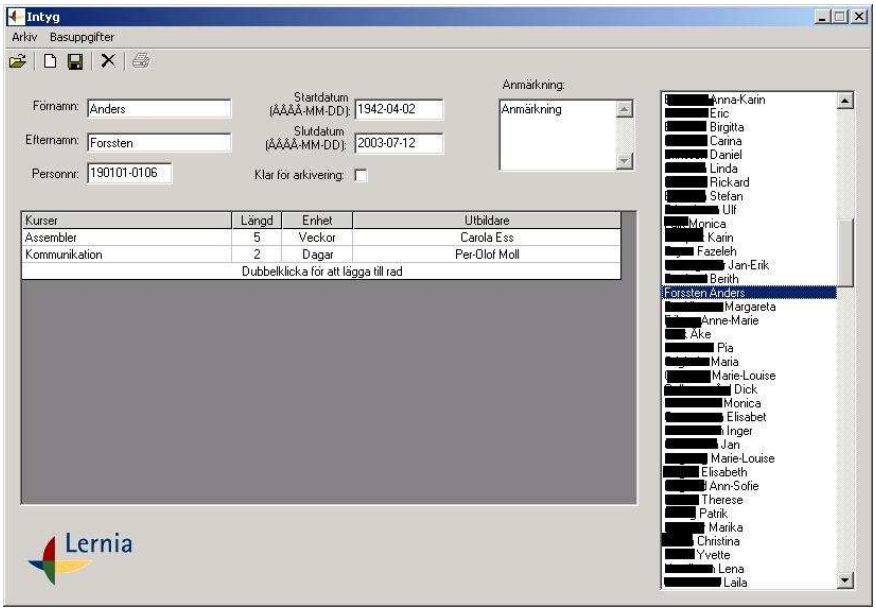
När vi implementerat systemet har vi byggt gränssnittet med funktionalitet för ett formulär i taget. Dessa har implementerats med klasserna först med funktionaliteten för databasåtkomst, sedan modulerna som står för formulärens funktionalitet och sist formuläret med dess grafiska komponenter. Anledningen är att en klass används av flera moduler. Samma förhållande gäller mellan formulären och modulerna.

3.2 Formulär, moduler och klasser

Detta avsnitt beskriver de formulär, moduler och klasser som skapats samt deras funktionalitet. För mer information om moduler, klasser, formulär och de grafiska komponenterna som ingår i formulären se appendix A på sidan 35.

3.2.1 Formulären

Formuläret `Intyg`, som motsvarar Accessformuläret KDKompbevis övre del, innehåller en *listruta* (till höger i figur 3.1) där namnen på alla kursdeltagare skall visas. När användaren klickar på en rad i listan uppdateras formuläret med information tillhörande vald kursdeltagare. Detta uppdaterar alla textfält i formuläret samt *FlexGrid*-komponenten (den grafiska komponent för visning av data som syns i nedre delen till vänster i figur 3.1, för mer information se appendix A på sidan 35), som visar alla kurser som kursdeltagaren är registrerad



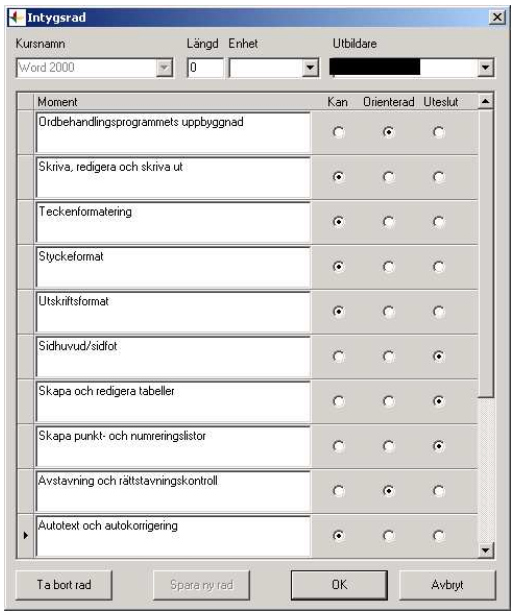
Figur 3.1: Formuläret `Intyg`

på.

När användaren ändrar i ett textfält så sätts en flagga som indikerar att någonting har ändrats. Denna flagga används för att se till att förändringar sparas när användaren byter kursdeltagare i listan eller när användaren stänger formuläret. En kontroll utförs och användaren tillfrågas om han vill spara ändringarna innan han går vidare. Menyalternativet `Spara` återställer flaggan till sitt ursprungsläge.

Eftersom textfält kan ha förutbestämda egenskaper, som en viss typ eller längd, eller kan vara obligatoriska, så utförs en kontroll innan användaren sparar för att undvika felaktiga poster i databasen. `FlexGrid`-komponenten innehåller information om vilka kurser aktuell person har deltagit i d.v.s kursnamn, längden på kursen samt vem som varit utbildare på kursen. Genom att dubbelklicka på en rad i `FlexGrid`-komponenten öppnas formuläret `Intygsrad` där användaren kan redigera eller ta bort raden. Dubbelklickar användaren på raden längst ner (märkt ”Dubbelklicka för att lägga till rad”), se figur 3.1, öppnas formuläret `Intygsrad` med endast tomma fält, så att användaren kan lägga till en rad.

Menyer finns för att lägga till, ta bort och spara kursdeltagare samt för att avsluta applikationen. Det finns även menyalternativ för att ta sig till formulären `Utbildare` och `Kurs`. Från menyn kan användaren även välja vilken databasfil gränssnittet ska jobba med. Om det finns osparade förändringar ställs en förfrågan till användaren om dessa ska sparas innan öppningen av nästa databasfil sker.



Figur 3.2: Formuläret `Intygsrad`

Formuläret `Intygsrad`, som motsvarar Accessformuläret KDKompbevis undre del, kan användas för att ändra information om ett kursintyg, eller för att skapa nya och ta bort gamla intyg. Det innehåller fält och komborutor där användaren får ange kursnamn, kurslängd och utbildare. Det finns även en `DataRepeater` (en grafisk komponent för visning och inmatning av data. `DataRepeater`-komponenten är den stora rutan i mitten i figur 3.2, för mer information se appendix A, sidan 35) som innehåller de moment som ingår i aktuell kurs samt bedömningar för varje moment som kursdeltagaren gått igenom. Bedömningen av varje moment både visas och ändras med samma visuella komponenter (radioknappar).

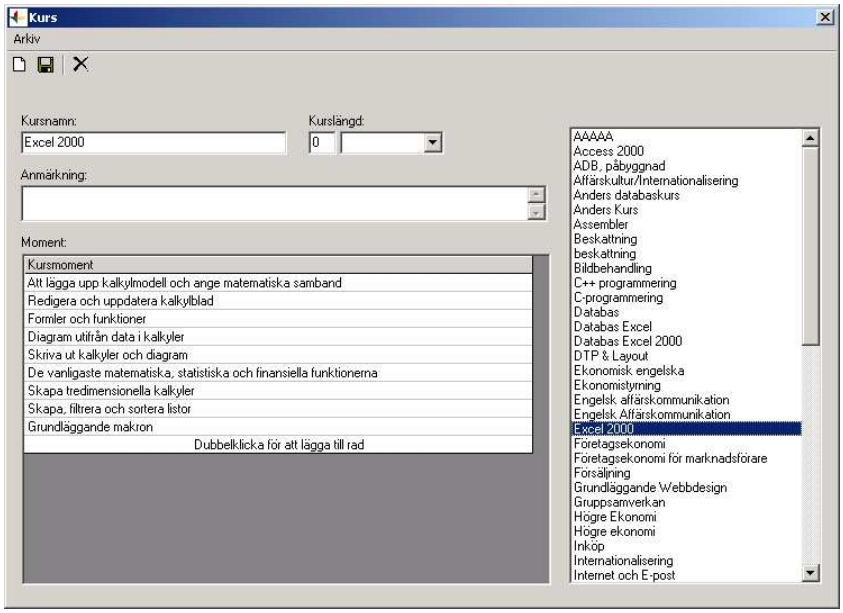
Längst ner på formuläret finns knappar för att radera och spara intygsraden samt ok-

och avbrytknappar. När användaren väljer att spara raden eller trycker ok utförs en kontroll på all inmatad data, och om något fält är felaktigt informeras användaren om detta och får möjlighet att korrigera inmatningen.

Figur 3.3: Formuläret `Utbildare`

Formuläret `Utbildare` innehåller information om de utbildare som finns och det används för att ändra information om en utbildare eller för att lägga till och ta bort utbildare. Det finns en lista med namn på alla utbildare och när användaren väljer en utbildare i listan visas information om denne i textfälten och användaren kan ändra data direkt i dessa textfält. Menyer finns för att lägga till och ta bort utbildare samt stänga formuläret.

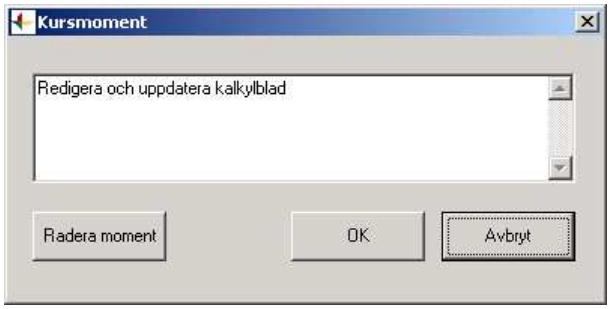
Formuläret `Kurs` används för att visa information om kurser och deras delmoment samt för att lägga till, ta bort eller ändra kurser. Det innehåller en lista med alla kursnamn. När användaren väljer en kurs i listan visas information om kursen (kursnamn, kurslängd, längdenhet, anmärkning) i textfälten. Dessutom visas en lista på kursens olika delmoment i en `FlexGrid`-komponent. Att ändra kursinformation görs direkt i textfälten och när användaren väljer att spara utförs precis som tidigare en kontroll av all inmatad data och ett



Figur 3.4: Formuläret `Kurs`

felmeddelande visas om något fält är felaktigt.

Hanteringen av kursmoment sker på ungefär samma sätt som i fallet med formulären `Intyg` och `Intygsrader`, d.v.s genom att användaren dubbelklickar i `FlexGrid`-komponenten på ett moment för att ändra eller ta bort det samt genom att användaren dubbelklickar på sista raden för att lägga till ett nytt moment. I båda fallen öppnas formuläret `Kursmoment`, som är det formulär där förändringarna utförs.



Figur 3.5: Formuläret `Kursmoment`

Formuläret `Kursmoment` används för att redigera ett kursmoment. I textfältet kan användaren mata in/redigera kursmomentets namn, och knappar finns för att spara inmatningen, ta bort kursmomentet eller avbryta inmatningen utan att spara.

3.2.2 Modulerna

Modulerna är som tidigare nämnts en behållare för kod. Varje formulär har en tillhörande modul med logik för gränssnittet och ett antal funktioner och procedurer för hantering av gränssnittet och kommunikation med databasen via klassobjekt.

I modulerna skapas instanser av de klasser som behövs och dessa initieras samtidigt som formuläret öppnas. Förutom att initiera objekt anropar initieringsproceduren även andra procedurer som är nödvändiga när ett formulär öppnas. Dessa procedurer är normalt sett procedurer för att uppdatera de grafiska komponenterna med information ur databasen samt procedurer för att ge komponenter vissa egenskaper. Ett specialfall är när formuläret `Intyg` öppnas och även en procedur för att välja databas körs. Denna procedur kontrollerar en textfil med information om sökväg till databasen och visar även en öppna-dialog om så skulle behövas.

När vi har skapat de olika modulerna har vi försökt att vara konsekventa vid namngivning och till viss del även funktionaliteten. Ett bra exempel på detta är funktionaliteten för att uppdatera formulärens grafiska komponenter som är uppbyggd av ett antal underprocedurer. Proceduren `uppdateraIntygsformulär`, som uppdaterar alla komponenter i formuläret `Intyg` använder i sin tur procedurerna `uppdateraKursdeltagarLista`, `uppdateraIntygsText` och `updateIntygsradGrid` för att uppdatera alla delar av formuläret. Denna lösning gör att vi kan välja att uppdatera hela formuläret eller välja ut vissa aktuella delar som skall uppdateras.

I varje modul finns en funktion som kontrollerar inmatad data innan denna ska sparas i databasen. När funktionen anropas kontrollerar den att alla inmatade värden uppfyller de villkor som ställs och returnerar i så fall `true`. Skulle något värde vara felaktigt visas ett

meddelande som informerar användaren om vad som blivit fel och `false` returneras till den anropande funktionen, som då kan avstå från att spara informationen. I kodexemplet nedan visas en del av kontrollfunktionen `intygsradIndataOK`, som kontrollerar att en textsträng inte är längre än 10 tecken.

```
If Len (frmIntygsrad.cmbEnhet.Text) > 10 Then
  MsgBox "Kursenhet får inte vara mer än 10 tecken", vbCritical
  intygsradIndataOK = false
End If
```

Förutom dessa generella exempel innehåller varje modul ett antal funktioner och procedurer specifika för just den modulen. Ett exempel är den procedur som hanterar markeringen av olika rader i `FlexGrid`-komponenten.

3.2.3 Klasserna

Klasserna är den del där all databaskommunikation sker. Instanser av klasserna skapas i modulerna och i vissa fall även i andra klasser. Kommunikationen med databasen sker genom att skapa en koppling mot databasen där man öppnar aktuell databas samt ett *Recordset*-objekt, instansierad av en ADO klass (`ADODB.Recordset`), som innehåller den data som läses in från databasen. All databaskommunikation i klasserna sker genom att manipulera detta `Recordset`-objekt.

När en instans av en klass initieras öppnas databasen och ett `Recordset`-objekt skapas. Om ett objekt av en klass behöver användas initieras detta också. När denna initiering skett kan klassens metoder användas tills dess att nedkopplingen sker, vilket stänger kopplingen till databasen och `Recordset`-objektet.

Många av metoderna i klasserna innehåller logik och anrop till andra metoder, medan andra metoder inte är annat än ett skal till en metod i `Recordset`-objektet. Det hade varit fullt möjligt att från en modul anropa dessa metoder i `Recordset`-objektet direkt, men

genom att kapsla in dem i metoder i klasserna skapas en mer strukturerad kod, som är lättare att ändra och bygga ut. Ett exempel på en sådan inkapsling är metoden `Update`, som används för att uppdatera och spara `Recordset`-objektet till databasen. Denna metod består endast av ett anrop till `Recordset`-objektets `update`-metod, d.v.s den kod som uppdaterar `Recordset`-objektet.

En stor del av klasserna utgörs av `Let`- och `Get`-metoder, som används för att hämta data från och spara data till databasen. `Get`-metoderna används för att läsa data från databasen och fungerar så att när de anropas läser de data från `Recordset`-objektet och sätter sedan sig själva till detta värde. Som exempel kan vi använda kodraden `frmIntyg.txtPersonnr = objIntyg.Personnr`. Här läses `objIntyg.Personnr` av och det inlästa värdet visas i textrutan `txtPersonnr` i formuläret `Intyg`.

Koden för `Get`-metoden personnr:

```
Public Property Get Personnr () As String
  If adoRecordset ("Personnr") <> "" Then
    Personnr = adoRecordset ("Personnr")
  Else
    Personnr = " "
  End If
End Property
```

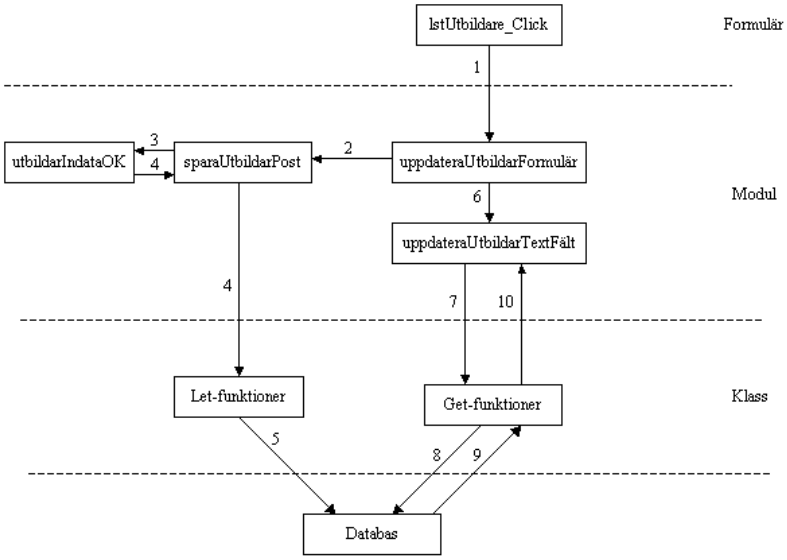
`Let`-metoderna fungerar bakvänt, d.v.s de sätts till något och använder sedan det värdet till att uppdatera `Recordset`-objektet. Detta gör att man kan använda dessa metoder som vanliga variabler. Koden `objIntyg.Personnr = "720803-6210"`, använder `Let`-metoden `Personnr` i en instans av klassen `Intyg`. `Personnr` sätts till strängen `"720803-6210"` precis som med en vanlig variabel.

Koden i `Let`-metoden `Personnr`:

```
Public Property Let Personnr (ByVal vdata As String)
adoRecordset ("Personnr") = vdata
End Property
```

3.2.4 Systemöversikt

När en användare klickar på en knapp eller matar in en text i det grafiska gränssnittet sker som visats ovan en rad händelser i de olika delarna av systemet. Nedan följer ett typexempel på hur olika funktioner, procedurer och metoder anropas, samt hur information skickas i programmet.



Figur 3.6: Programflöde

Exemplet består i att användaren klickat på en utbildare i listan med utbildare i formuläret `Utbildare`. Figur 3.6 visar programflödet och hur de olika funktionerna anropas i exemplet. Det första som händer är att en klick-händelse för listan i formuläret anropas. I denna händelse finns en rad kod som anropar proceduren `uppdateraUtbildarFormulär` i modulen `Utbildare`.

Det första som sker i `uppdateraUtbildarFormulär` är en kontroll av flaggan `blnUtbildareFormDataChanged`. Är denna flagga satt har något textfält ändrats och användaren tillfrågas om han/hon vill spara dessa ändringar. Väljer användaren att spara anropas proceduren `sparaUtbildarPost` i samma modul. Denna procedur anropar funktionen `utbildarIndataOK`, vars funktionalitet finns beskriven ovan, för att kontrollera att de inmatade värdena är korrekta.

Är allt korrekt uppdateras `Recordset`-objektet i objektet `objUtbildare` med de nya värdena. Uppdateringen sker genom att först sätta de egenskaper i objektet som motsvarar textfälten i formuläret. Som ett sista steg i anropas `update`-metoden i `Recordset`-objektet och när informationen sparats uppdateras listan för att återspegla förändringarna. Uppdateringen sker genom att först ta bort posten som ska uppdateras. Sedan läses egenskaperna i `objUtbildare` för den aktuella posten av och en ny post adderas till listan med de nya värdena.

När proceduren ovan slutförts ska `Recordset`-objektet sättas till att peka på den utbildare som valts i listan. Detta sker genom att sätta egenskapen `UtbID` till ett ID-nummer som sparats i listan för varje utbildare. Egenskapen `UtbID` sätts i två steg: först sätts `Recordset`-objektet till den första posten, och sedan används ID-numret för att hitta den önskade posten.

Det sista steget är att uppdatera textfälten med informationen om utbildaren. Detta sker genom att `uppdateraUtbildarFormulär` anropar proceduren `uppdateraUtbildarTextFält`, som använder ett antal egenskaper i utbildarobjektet för att sätta textfälten i formuläret `Utbildare`.

3.3 Summering

Vi har nu visat hur vårt system är uppbyggt och hur de olika delarna arbetar med varandra. För att säkerställa funktionaliteten i systemet och för att kunna upptäcka och åtgärda eventuella fel måste en systemtest utföras. I kapitel 4 beskriver vi testningens olika steg

och hur den kommer att utföras.

4 Testning

Testningen av systemet pågår under hela arbetets gång och utförs både av oss själva och av berörd personal på Lernia. Denna testning utförs för att säkerställa att programmet har full funktionalitet samt är användbart för personalen på Lernia. Vi har definierat ett antal testfall, vilka kommer att användas av oss för att testa programmets funktionalitet. Dessa testfall finns listade i appendix C med början på sidan 41. Eftersom det inte är vi som ska använda systemet låter vi personal på Lernia arbeta med systemet och utvärdera det för att sedan komma med förslag på förändringar.

4.1 Resultat av testning utförd av personal på Lernia

Efter att ha utvärderat den första utgåvan av programmet hade testpersonen på Lernia några synpunkter på programmet och förslag på förändringar och förbättringar. Dessa ändringar rörde både programmets utseende och funktionalitet. Vi har nedan listat de synpunkter vi fick:

1. Automatiskt stor bokstav i början på namn.
2. Automatisk kontroll av inmatade personnummer.
3. Datumkontroll, slutdatum måste ligga senare i tiden än startdatum.
4. Indatamask på inmatningsfält för personnummer och datum.
5. Rullningslist för kursdeltagarlistan.
6. Sökfunktion för kursdeltagarlistan.

Förutom dessa synpunkter fanns det även andra problem som framkom under testningen. Dessa problem berodde till största delen på att målmiljön skiljde sig avsevärt från utvecklingsmiljön. Då målmiljön hade en begränsad skärmupplösning fick inte hela programmet plats på skärmen, vilket resulterade i att programmet blev relativt otympligt att hantera. Detta är dock inget vi kommer att försöka åtgärda då måldatorn snart kommer att bytas ut mot en modernare dator, vilken har möjlighet att använda sig av högre skärmupplösning. Av de synpunkter i listan ovan är det två som har sin grund i problem i målmiljön: rullningslisten och sökfunktion i kursdeltagarlistan. Av någon anledning fungerade inte rullningslisten på målsystemet. Detta ledde till att användaren var tvungen att använda piltangenterna för att navigera i listan och användaren som testat programmet efterfrågade då en sökfunktion. Detta problem verkar uppstå då fönstret blir för stort och löstes genom att minska fönsterstorleken.

Efter att alla efterfrågade förändringar gjorts levererades en ny testversion till testpersonen på Lernia, som var nöjd med programmets funktionalitet.

4.2 Resultat av testfall

Systemtestet har skett på den version av programmet som togs fram efter de kommentarer vi fick från Lernias personal.

Alla testfall har testats på målmiljön Windows 2000. I målmiljön Windows 95 har alla testfall utförts utom testfall nummer 14, vilket ej kan utföras utan en installation av Access på den aktuella miljön. Att detta testfall inte går att utföra beror på att man inte manuellt kan kontrollera att kursmomenten tagits bort om inte Access finns installerat på datorn. Inga testfall har testats på målmiljön Windows XP eftersom detta endast är en tänkt målmiljö som ej existerar på Lernia ännu.

Under testningen upptäcktes två buggar. Den första gäller alla formulär som innehåller en `FlexGrid`-komponent, där den funktion som används för att avmarkera rader inte fungerar. Detta visar sig exempelvis när användaren försöker ta bort en rad. Den an-

dra buggen upptäcktes i samband med testningen av testfall `Ändra en kurs längdenhet`. Det som hände var att när testpersonen valde att spara förändringen byttes förändringen tillbaka till värdet som fanns lagrat i databasen. Övriga testfall gav positiva resultat.

Den första buggen kommer inte att åtgärdas p.g.a tidsbrist, samt att den har en ringa inverkan på systemets funktionalitet. Den andra buggen betraktades som viktig, och har rättats till, då dess inverkan kunde ge direkt felaktiga data i databasen vilket är oförenligt med det andra uppsatta primärmålet.

4.3 Summering

Testningen har fungerat bra och särskilt testningen utförd av Lernias personal har uppmärksammat oss på brister som annars inte hade upptäckts. Efter att testningen slutförts med gott resultat anser vi att systemet är färdigt att tas i bruk. Det finns delar som kan förbättras ytterligare, men detta har lämnats till Lernias personal att implementera. Testningen har också gett oss användbara erfarenheter och visat på de problem som kan uppstå när man utvecklar en applikation. Mer om erfarenheter och problem finns i kapitel 5.

5 Erfarenheter och rekommendationer

Efter att ha utvecklat systemet har vi fått många nya kunskaper och erfarenheter. Då detta var första gången vi utförde ett arbete av den här typen, d.v.s att arbeta med Visual Basic och dess databashantering, krävdes det att vi studerade mycket material rörande Visual Basic och dess sätt att hantera databaser. Att en färdig mall redan fanns i det gamla systemet för den underliggande databasen och det grafiska utseendet underlättade arbetet då vi inte behövde designa databasen och gränssnittet från grunden.

Arbetet med att implementera systemet visade sig ta längre tid än vi först räknade med och vi insåg att vi inte skulle hinna uppnå sekundärmålen. Anledningen till att arbetet tog så lång tid berodde till stor del på att vi hade ganska begränsade grundkunskaper om

Visual Basic och dess databaskomponenter. Vi valde då att inrikta oss på att färdigställa de delar av programmet som hör till primärmålet, d.v.s användargränssnittet och den bakomliggande logiken, och att helt utesluta de delar som hör till sekundärmålet. Se avsnitt 1.4 för utförligare information om primär- och sekundärmålen.

Från början reflekterade vi inte över att målmiljön skiljde sig från utvecklingsmiljön, men då problem uppstod (se nästa avsnitt för mer information) blev vi tvungna att ha detta i åtanke då vi fortsatte arbetet. Att kontrollera hur mål- resp. utvecklingsmiljö skiljer sig åt borde ha skett tidigare för att undvika de problem som uppstod.

5.1 Problem

Tömma en DataRepeater. När man växlade från en kursdeltagare med poster som visades i `DataRepeater`-komponenten till en utan, låg dom gamla posterna kvar. Detta löstes genom att explicit uppdatera datarepeteraren och den databaskomponent den beror på.

Spara information i en DataRepeater. Vid konstruktionen av formuläret `IntygsRad` (se figur 3.2) var det svårt att åstadkomma att bedömningsvalen användaren gjort sparades till databasen. När testpersonen försökte klicka i ett antal bedömningar för en kurs olika moment så sparades dom inte när formuläret stängdes genom att klicka på `OK`-knappen. Detta för att den grafiska komponenten som valen görs i inte uppdaterar databasen om den inte styrs av den databaskomponent den är kopplad till. Lösningen går ut på att databaskomponenten styrs så att den stegar sig igenom alla poster en efter en för att fånga eventuella användarval när användaren klickar på `OK`-knappen i formuläret.

Indatamaskering. Vi blev ombedda att implementera indatamaskering för personnummer och datum (se appendix B.1) och stötte på problem då detta skulle implementeras. Vi provade först att modifiera de befintliga textrutorna genom att per-

sonnummerinmatningen delades till två textfält, vilka slogs samman när användaren sparade. Denna lösning fungerade, men kändes något klumpig och hade blivit mycket klumpig att använda för datuminmatningen. Genom att titta i Visual Basics hjälpfiler och komponenter hittade vi en komponent, `MaskedEditBox`, vars användningsområde var exakt det vi sökte. Denna komponent fungerar som en vanlig textruta, men har utökad funktionalitet för indatamaskering. Ett litet problem uppstod dock då vi använde denna komponent. Om en person i databasen saknar personnr, startdatum eller slutdatum tömde vi textrutan. P.g.a indatamaskeringen tillät inte komponenten att vi matade in en tom textsträng och detta resulterade i ett fel. Vi löste problemet genom att mata in ett antal "_"-tecken till textrutan där siffrorna skall stå. "_"-tecknet anger i den här typen av textruta att fältet är tomt och lämpar sig alltså utmärkt för att tömma rutan.

Personnummerskontroll. För att kontrollera att ett personnummer är korrekt används en algoritm för kontroll av att den sista siffran i numret som beror på de nio första är korrekt. Denna algoritm hittades som Java-kod på Internet[4] och implementerades med VB-kod i programmet.

Målmiljön. Eftersom målmiljöerna är olika operativsystem med uppemot åtta år på nacken har det uppstått många problem vars ursprung vi inte kunnat finna. När systemet testades på en Windows 95 maskin visades inte rullningslisten som ska finnas i intygsformuläret då fler kursdeltagare än vad som kan ses på en gång existerar. När testpersonen matade in nya uppgifter om en kurs och klickade på en annan kurs så han fick en förfrågan om han ville spara och om han sedan svarade ja kraschade applikationen.

När testpersonen öppnade formuläret för att fylla i bedömningar visades inga kursmoment. Detta beror på att den ActiveX-komponent som ska repeteras i `DataRepeater`-komponenten inte kopierats med av den installationsguide som VB6

skapat. Problemet löstes genom att manuellt infoga komponenten i installationsprogrammet.

5.2 Slutsatser och utvärdering

Vi har haft en dator tillgänglig på Lernia att utveckla systemet med i Visual Basic. Vi har även haft tillgång till en egen dator där vi skrivit rapporten. Fördelen med detta upplägg är att vi har kunnat arbeta med rapporten samtidigt som vi utvecklat systemet. Nackdelen är att vi inte har parprogrammerat hela tiden även om vi befunnit oss på samma plats, vilket har lett till att koden inte har granskats lika noga som den skulle ha gjorts om vi båda hade suttit hela tiden framför samma dator. Vi tycker dock att fördelen var större än nackdelen.

Vi är nöjda med de delar av systemet vi färdigställt och vi anser att det är betydligt bättre än det gamla. Systemet som helhet är vi inte helt nöjda med eftersom vi inte lyckades uppfylla alla de uppsatta målen, men vet att systemutvecklarna på Lernia kommer att bygga vidare på systemet med de sekundära målen så fort vi lämnar över utvecklingen till dem.

De bra sakerna med att komma till ett företag som studenter är att man får tillgång till mångåriga praktiska erfarenheter av de områden där vi tidigare i stort sett endast haft teoretiska kunskaper, vilka de förmedlat oss då vi inte vetat hur vi ska gå tillväga.

Om vi hade som uppgift att skapa detta system igen och haft lite större valmöjligheter skulle vi valt ett objektorienterat språk eftersom vi då skulle kunnat skapa en generell datatabellsklass i stället för att upprepa kod i alla de liknande klasserna med alla nackdelar som därpå följer. T.ex så är sannolikheten många gånger större att det smyger in fel i koden då den ska skapas i nästintill identiska uppsättningar på flera ställen. På 100 rader kod är risken för minst ett fel 39%, medan risken är 99% om man har 1000 rader kod [5]. Dessutom blir koden mycket svår att underhålla när man gör ändringar som måste göras på ett otal liknande ställen p.g.a detta.

5.3 Vidareutveckling

Då vi inte hade tid att implementera sekundärmålen bör detta vara det första att titta på vid en eventuell vidareutveckling av systemet. De kommer med största sannolikhet att implementeras inom en överskådlig framtid av Lernias personal då det ligger i deras intresse att systemet ska kunna användas med full funktionalitet samt att kunskapen för att göra det finns hos personalen.

I dessa dagar är IT-avdelningarnas intresse att sänka kostnaderna för det administrativa arbetet och behålla de befintliga maskinparkerna så länge som möjligt. Många företag väljer lösningen med tunna klienter som endast kör ett enkelt webb-gränssnitt och låter en server ta hand om logiken. I Lernias fall vore det möjligt att fortsätta köra sin befintliga datapark genom att installera mindre krävande system och endast ha en eller flera kraftfulla servrar.

Att integrera det utvecklade systemet med övrig information om kursdeltagarna anser vi vara ett naturligt steg om systemet ska vidareutvecklas. I nuläget finns information om kurdeltagarnas adresser, telefonnummer, kursavgifter m.m. lagrade på ett fristående system. Denna information bör samlagras med övrig information om studenternas kursdeltagande för att minska redundant information och underlätta hanteringen av studerande.

Om systemet återimplementeras i ett senare skede är sannolikt ett plattformsoberoende objektorienterat språk att föredra. Detta för att det på plattformsmarknaden råder oro och eftersom myndigheter och många företag väljer att köra marknadsledande plattformar och det ser osäkert ut vad som imorgon blir marknadsledande skulle detta innebära enklare övergångar till andra plattformar. Implementationen skulle säkerställa att systemet går att köra oberoende vilken väg företaget väljer i det rådande plattformskriget.

5.4 Summering

Systemet är inte färdigt. Mycket mer tid behövs, för vår del, för studier av Crystal Reports om vi skulle kunna implementera de sista bitarna. Dock är alla primärmål nådda vilket vi

är nöjda med. Systemets utvecklingsmöjligheter är många men intressantast för Lernia är antagligen en systemintegration. Intressantast för oss att vad det gäller vidareutveckling vore en implementation i ett objektorienterat språk samt stöd för tunna klienter i formen av ett webbgränssnitt.

Referenser

[1] John Connell. *Beginning Visual Basic 6 Database Programming.* Wrox press, 1998.

[2] Hank Marquis Eric A Smith, Valor Whisler. *Visual Basic 6 Bible.* IDG Books Worldide, Inc., 1998.

[3] Steven Holzner. *Visual Basic 6 black book.* Coriolis Technologi Press, 1998.

[4] KTH. Vetenskapligheter. http://www.nada.kth.se/kurser/kth/2D1343/vetenskap.html, Maj 2003.

[5] Martin Omander. *Avancerad Visual Basic 6.0.* Bonnier Icon, 1998.

[6] Evangelos Petroutsos. *Mastering Visual Basic 6.* SYBEX Inc., 1998.

[7] Computer Sweden. Ordlista. http://computersweden.idg.se/tjanster/sprakwebb/ord.asp, Maj 2003.

A Terminologi

ActiveX-kontroll En ActiveX-kontroll är en egendefinierad komponent som kan användas i Visual Basic 6. Man designar dem i ett valfritt programspråk som ett eget program och kompilerar dem för att sedan använda dem som komponenter i andra program. De används som vanliga komponenter i Visual Basic och kan placeras i formuläret och kan även användas av andra komponenter t.ex `DataRepeater`-komponenten.

ActiveX Data Object (ADO) ADO är precis som sin föregångare, Data Access Object (DAO), en komponent för åtkomst och manipulering av databaser och erbjuder bättre prestanda och bättre möjligheter till kommunikation med databaser än DAO. ADO kommunicerar via Microsofts gränssnitt OLE DB med olika typer av databaser. OLE DB är ett komplicerat gränssnitt men tack vare ADO kan utvecklare enklare använda OLEDB för att jobba mot en databas, för mer information se [1].

Datahygien Att se till att lagrade data är korrekta och aktuella. [7] Det vill säga att kontrollera att indata är av ett visst format eller uppfyller vissa krav som kan kontrolleras.

DataRepeater `DataRepeater`-komponenten är en grafisk komponent som används för att visa data genom att använda en ActiveX-kontroll. Man designar först en ActiveX-kontroll med fält för visningen av data i datamängden och denna ActiveX-kontroll används sedan i `DataRepeater`-komponenten. Varje rad i `DataRepeater`-komponenten är en instans av ActiveX-kontrollen och kan användas både för visning och inmatning av data. En rad skapas för varje post i mängden med de aktuella dataposterna.

FlexGrid `FlexGrid`-komponenten är en grafisk komponent som används för att visa data i ett tabulärt format med rader och kolumner, vilket gör att den lämpar sig mycket väl för att visa databasposter. Nackdelen är att den inte är gjord för datainmatning.

Detta innebär att applikationen antingen får fånga knapptryckshändelser i `FlexGrid`-komponenten för att komma runt problemet eller att användaren får mata in data i ett fält separerat från `FlexGrid`-komponenten och uppdatera den denna därefter. För mer information se [6].

Formulär i Visual Basic 6.0 Formulären är den grafiska delen av programmet och fungerar som mallar [3] som utvecklare placerar grafiska komponenter på t.ex knappar och listor. Formulären är det grafiska gränssnittet mot användaren i en VB6 applikation.

Funktioner, procedurer och metoder Funktioner, procedurer och metoder är på många sätt lika. Skillnaden mellan funktioner och procedurer i VB är att en funktion returnerar ett värde medan en procedur utför en uppgift utan att returnera något. En metod är inte en VB-specifik term, utan används av oss när vi talar om funktioner och procedurer i klasserna eftersom man normalt sett använder ordet metod då man pratar om funktioner i klasser.

Klasser i Visual Basic 6.0 Klasser i VB6 stödjer i stort objektorienteringskonceptet, det som saknas är arv, för mer information se[5].

Modal Det finns två typer av modalitet: applikationsmodal och systemmodal. Att ett fönster är applikationsmodalt innebär att inget annat fönster tillhörande applikationen kan få fokus medan det modala fönstret är öppet. Man kan dock byta fokus till andra applikationer. Om fönstret är systemmodalt kommer det alltid att synas även då användaren växlar till en annan applikation. Anledningen till att skapa ett modalt fönster är att applikationen kräver att användaren ska svara på en viktig fråga för att komma vidare eller för att vara säker på att användaren har läst en viss information.

Moduler i Visual Basic 6.0 En modul i VB6 fungerar som en samling programkod. Moduler har trots vad [3] påstår inte mycket med objektorienterad programmering

att göra, då de saknar arv (liksom VB6 i stort), polymorfism och instansiering av objekt. Det de kan vara bra till är att separera koden för formulärets händelser för att formulärkoden ska bli överskådlig, vilket snarare placerar moduler under konceptet strukturerad programmering.

Object Linking and Embedding Data Base (OLE DB) OLE DB är ett lågnivågränssnitt för databasåkomst som stöder många olika typer av datakällor, t.ex relationsdatabaser och filsystem. För mer information se [2]. I vår systemimplementation kommunicerar ADO med OLE DB, som i sin tur kommunicerar med själva databasen.

Properties i Visual Basic 6.0 För att ändra en egenskap, kallad *property* i VB6, måste egenskapen ha en *Let*-funktion, och för att få reda på vad en egenskap har för värde måste den ha en *Get*-funktion. En egenskap skiljer sig från en variabel på så sätt att egenskaper har dessa funktioner för att ställa och hämta värdet. Funktionerna kan innehålla flera satser som ska exekveras vid samma anrop. T.ex kanske en kontroll av det inmatade värdet skall ske i samband med sättandet och vid felaktigheter kan ett felmeddelande visas som talar om vad som var fel. Detta är ett sätt få kod som hör till en egenskap att exekveras i samband att den tilldelas ett värde, även om egenskapen ser ut som en konventionell variabel. För mer information se [5].

B Krav

B.1 Krav för primärmålen

1. Krav på utvecklingsmiljön
   (a) Microsoft Visual Basic 6.0
2. Databasåtkomst med hjälp av Microsofts ActiveX Database Object (ADO)
3. Strukturerad programmering
   (a) Lagerstruktur
   (b) Moduler
4. Liknar det tidigare systemet till utseende
   (a) Ett formulär för skapande, inmatning och ändring av kursdeltagares personuppgifter och kursdeltagande
   (b) Ett formulär för skapande, inmatning och ändring av bedömningar på olika kursmoment för ett visst kursdeltagande
   (c) Ett formulär för skapande, inmatning och ändring av kurser och uppgifter om densamma
   (d) Ett formulär för skapande, inmatning och ändring av utbildare och uppgifter om densamma
5. Samma funktionalitet som det tidigare systemet ur ett databasperspektiv
   (a) Det ska vara möjligt att lägga till en ny kursdeltagare i systemet med:
      i. Förnamn som ska lagras med första bokstaven versal
      ii. Efternamn som ska lagras med första bokstaven versal

iii. Personnummer som ska vara ett giltigt svenskt personnummer, d.v.s kontrollsiffran är korrekt

iv. Datum för deltagarens utbildningsstart

v. Datum för deltagarens utbildningsslut som ska vara ett datum samma som eller senare än startdatumet

vi. Anmärkning

(b) Det ska vara möjligt att ta bort en kursdeltagare ur systemet

(c) Det ska vara möjligt att ändra en kursdeltagares personuppgifter

(d) Det ska vara möjligt att ange vilka kurser en person deltagit i, längden på kursen och vem som är utbildaren för den aktuella kursen

(e) Det ska vara möjligt att ändra i en post om en kurs en person deltagit i, längden på kursen och vem som är utbildaren för den aktuella kursen

(f) Det ska vara möjligt att lägga till kurser i systemet

i. Kursnamn

ii. Kurslängd

iii. Längdenhet

iv. Anmärkning

(g) Det ska vara möjligt att ta bort kurser ur systemet

(h) Det ska vara möjligt att redigera kursuppgifter

(i) Det ska vara möjligt att lägga till utbildare i systemet

i. Förnamn som ska lagras med första bokstaven versal

ii. Efternamn som ska lagras med första bokstaven versal

iii. Ämne

(j) Det ska vara möjligt att ta bort utbildare ur systemet

(k) Det ska vara möjligt att redigera uppgifter om utbildare

(l) Det ska vara möjligt att lägga till kursmoment i systemet

(m) Det ska vara möjligt att ta bort kursmoment ur systemet

(n) Det ska vara möjligt att redigera uppgifter om kursmoment

(o) När användaren anger vilka kurser en person deltagit i ska bedömningar för alla kursmoment skapas

(p) När användaren tar bort post om vilken kurs en person deltagit i ska bedömningar för alla tillhörande kursmoment tas bort

(q) Det ska vara möjligt att redigera uppgifter om kursmomentsbedömningar

6. Inmatningen skall kontrolleras mot databasens tillåtna värden för varje värde som skall sparas

7. Användaren ska kunna välja databasfil att arbeta mot

B.2 Sekundära krav

1. Krav på utvecklingsmiljön

   (a) Crystal Decisions Crystal Reports 8.5

2. Alla poster som har fått utskriven-flaggan satt i tabellen intyg ska kunna flyttas till en annan databas (d.v.s arkiveras)

3. Det ska vara möjligt att skriva ut intyg

C Testfall

1. **Lägg till en ny kursdeltagare i systemet**

   - **Beskrivning:** försöker lägga till en ny kursdeltagare i systemet med förnamn, efternamn, personnummer, datum för deltagarens utbildningsstart, datum för deltagarens utbildningsslut och en anmärkning
   - **Indata:** förnamn = ”Ada”, efternamn = ”Andersson”, personnummer = 19010101-0106, startdatum = 19900101, slutdatum = 19900102, anmärkning = valfri text
   - **Utdata:** den nya personen dyker upp listan och den inmatade personinformationen visas då systemtestaren klickar fram personen i listan
   - **Utförande:** formuläret `Intyg` framme. Välj `Ny post` i menyn före datainmatningen, och välj `Spara post` i menyn efter datainmatningen

2. **Ta bort en kursdeltagare ur systemet**

   - **Beskrivning:** försöker ta bort en kursdeltagare ur systemet
   - **Indata:** ingen
   - **Utdata:** den valda personen finns ej i systemet
   - **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan. Välj `Radera post` i menyn samt klicka `OK` i den *modala* (för förklaring av modalitet se appendix A, sidan 35) rutan

3. **Ändra en kursdeltagares förnamn**

   - **Beskrivning:** försöker ändra en kursdeltagares förnamn
   - **Indata:** förnamn != nuvarande värde
   - **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan

- **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan. Välj `Spara post` i menyn efter datainmatningen samt välj `OK` i den modala rutan.

4. **Ändra en kursdeltagares efternamn**

   - **Beskrivning:** försöker ändra en kursdeltagares efternamn
   - **Indata:** efternamn != nuvarande värde
   - **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan
   - **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan och välj `Spara` i menyn efter datainmatningen

5. **Ändra en kursdeltagares personnummer**

   - **Beskrivning:** försöker ändra en kursdeltagares personnummer
   - **Indata:** personnummer = ett korrekt personnummer, != nuvarande värde
   - **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan
   - **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan och välj `Spara` i menyn efter datainmatningen

6. **Ändra en kursdeltagares startdatum**

   - **Beskrivning:** försöker ändra en kursdeltagares startdatum
   - **Indata:** startdatum = valfritt datum på formen *åååå*mm*dd*, != nuvarande värde samt tidigare än slutdatum
   - **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan

- **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan och välj `Spara` i menyn efter datainmatningen

7. **Ändra en kursdeltagares slutdatum**

   - **Beskrivning:** försöker ändra en kursdeltagares slutdatum
   - **Indata:** slutdatum = valfritt datum på formen *ååååmmdd*, != nuvarande värde samt senare än startdatum
   - **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan
   - **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan och välj `Spara` i menyn efter datainmatningen

8. **Ändra en kursdeltagares anmärkning**

   - **Beskrivning:** försöker ändra en kursdeltagares anmärkning
   - **Indata:** anmärkning != nuvarande värde
   - **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan
   - **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan och välj `Spara` i menyn efter datainmatningen

9. **Ny intygsrad**

   - **Beskrivning:** försöker skapa en post med information om ett kursdeltagande
   - **Indata:** kursnamn = Valfri i listan, längd = 3, enhet = valfri i listan, utbildare = valfri i listan
   - **Utdata:** den inmatade informationen dyker upp på en rad i `FlexGrid`-komponenten

- **Utförande:** formuläret `Intyg` framme. Markera en valfri kurs. Dubbelklicka på raden märkt ”Dubbelklicka för att lägga till ny rad”. Mata in indata och välj sedan `OK`

10. **Ändra intygsrad**

    - **Beskrivning:** försöker ändra i en post med information om ett kursdeltagande
    - **Indata:** längd != nuvarande värde, enhet = valfri i listan, != nuvarande värde, utbildare = valfri i listan, != nuvarande värde
    - **Utdata:** den inmatade informationen dyker upp på en rad i `FlexGrid`-komponenten
    - **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan med minst en kurs. Dubbelklicka på en kursrad, mata in indata och välj sedan `OK`

11. **Ta bort intygsrad**

    - **Beskrivning:** försöker ta bort en post med information om ett kursdeltagande
    - **Indata:** ingen
    - **Utdata:** den valda intygsraden finns ej kvar i `FlexGrid`-komponenten
    - **Utförande:** formuläret `Intyg` framme. Markera valfri person i listan med minst en kurs. Dubbelklicka på en kursrad. Klicka på `Ta bort rad` och välj sedan `OK`

12. **Nya kursmomentsbedömningar**

    - **Beskrivning:** försöker skapa bedömningar för en kurs en person gått
    - **Indata:** kursnamn = valfri i listan, längd = 4, enhet = valfri i listan, utbildare = valfri i listan
    - **Utdata:** bedömningar för den valda kursen i `DataRepeater`-komponenten i formuläret `Intygsrad`

- **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan. Dubbelklicka på raden märkt ”Dubbelklicka för att lägga till ny rad” och välj `Spara` efter datainmatningen

13. **Redigera kursmomentsbedömningar**

    - **Beskrivning:** försöker redigera kursmomentsbedömningar för en kurs en person gått
    - **Indata:** Ändra bedömningar genom att klicka på radioknapparna
    - **Utdata:** bedömningarna som ändrats är fortfarande ändrade nästa gång systemtestaren dubbelklickar på samma kursrad för denna person
    - **Utförande:** formuläret `Intyg` framme. Markera en valfri person med minst en kurs. Dubbelklicka på en kursrad och välj `OK` efter datainmatningen.

14. **När användaren tar bort en post om vilken kurs en person deltagit i ska bedömningar för alla tillhörande kursmoment tas bort**

    - **Beskrivning:** försöker ta bort bedömningar med information om ett kursdeltagande
    - **Indata:** ingen
    - **Utdata:** De bedömningar som hörde till intygsraden finns ej kvar i tabellen bedömningar i Access
    - **Utförande:** formuläret `Intyg` framme. Markera en valfri person i listan med minst en kurs. Dubbelklicka på en kursrad. Välj `Ta bort rad` och sedan `OK`

15. **Lägg till en ny kurs i systemet**

    - **Beskrivning:** försöker lägga till en ny kurs i systemet med kursnamn, kurslängd, längdenhet och en anmärkning

- **Indata** kursnamn = valfri text, kurslängd = 2, längdenhet = valfri längdenhet, anmärkning = valfri text
- **Utdata:** den nya kursen dyker upp listan och den inmatade kursinformationen visas då systemtestaren klickar fram kursen i listan
- **Utförande:** formuläret `Kurs` framme. Välj `Ny kurs` i menyn före indatainmatning och välj `Spara kurs` i menyn efter datainmatningen

16. **Ta bort en kurs ur systemet**

- **Beskrivning:** försöker ta bort en kurs ur systemet
- **Indata:** kursnamn = valfri kurs i listan
- **Utdata:** den valda kursen försvann ur listan
- **Utförande:** formuläret `Kurs` framme. Välj en kurs, som inte har några kursdeltagare, välj sedan `Radera kurs` i menyn och välj `OK` efter datainmatningen

17. **Ändra en kurs namn**

- **Beskrivning:** försöker ändra en kurs namn
- **Indata:** kursnamn != nuvarande värde
- **Utdata:** inmatade kursinformationen visas då systemtestaren klickar fram kursen i listan
- **Utförande:** formuläret `Kurs` framme. Välj `Spara kurs` i menyn efter datainmatningen

18. **Ändra en kurs längd**

- **Beskrivning:** försöker ändra en kurs längd
- **Indata:** kurslängd != nuvarande värde

- **Utdata:** inmatade kursinformationen visas då systemtestaren klickar fram kursen i listan
- **Utförande:** formuläret `Kurs` framme. Välj `Spara kurs` i menyn efter datainmatningen

19. **Ändra en kurs längdenhet**

    - **Beskrivning:** försöker ändra en kurs längdenhet
    - **Indata:** längdenhet = valfri i listan, != nuvarande värde
    - **Utdata:** inmatade kursinformationen visas då systemtestaren klickar fram kursen i listan
    - **Utförande:** formuläret `Kurs` framme. Välj `Spara kurs` i menyn efter datainmatningen

20. **Ändra en kurs anmärkning**

    - **Beskrivning:** försöker ändra en kurs anmärkning
    - **Indata:** anmärkning != nuvarande värde
    - **Utdata:** inmatade kursinformationen visas då systemtestaren klickar fram kursen i listan
    - **Utförande:** formuläret `Kurs` framme. Välj `Spara kurs` i menyn efter datainmatningen

21. **Lägg till en ny utbildare i systemet**

    - **Beskrivning:** försöker lägga till en ny utbildare i systemet med förnamn, efternamn, och ämne
    - **Indata:** förnamn = ”Caesar”, efternamn = ”Carlsson”, ämne = ”Computer Science”

- **Utdata:** den nya kursen dyker upp listan och den inmatade kursinformationen visas då systemtestaren klickar fram kursen i listan
- **Utförande:** formuläret `Utbildare` framme. Välj `Ny utbildare` i menyn före datainmatning och `Spara utbildare` väljs i menyn efter datainmatningen

22. **Ta bort en utbildare ur systemet**

- **Beskrivning:** försöker ta bort en utbildare ur systemet
- **Indata:** ingen
- **Utdata:** den valda utbildaren försvann ur listan
- **Utförande:** formuläret `Utbildare` framme. Markera en valfri person i listan. Välj `Ta bort utbildare` i menyn och sedan `OK` efter datainmatningen

23. **Ändra en utbildares förnamn**

- **Beskrivning:** försöker ändra en utbildares förnamn
- **Indata:** förnamn != nuvarande värde
- **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan
- **Utförande:** formuläret `Utbildare` framme. Markera en valfri person i listan. Välj `Spara utbildare` i menyn efter datainmatningen

24. **Ändra en utbildares efternamn**

- **Beskrivning:** försöker ändra en utbildares efternamn
- **Indata:** efternamn != nuvarande värde
- **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan

- **Utförande:** formuläret `Utbildare` framme. Markera en valfri person i listan. Välj `Spara utbildare` i menyn efter datainmatningen

25. **Ändra en utbildares ämne**

- **Beskrivning:** försöker ändra en utbildares ämne
- **Indata:** ämne != nuvarande värde
- **Utdata:** inmatade personinformationen visas då systemtestaren klickar fram personen i listan
- **Utförande:** formuläret `Utbildare` framme. Markera en valfri person i listan. Valj `Spara utbildare` i menyn efter datainmatningen

26. **Lägg till kursmoment i systemet**

- **Beskrivning:** försöker lägga till kursmoment i systemet
- **Indata:** moment = ”Fylla i deklarationen”
- **Utdata:** den inmatade informationen dyker upp på en rad i `FlexGrid`-komponenten
- **Utförande:** formuläret `Kurs` framme. Välj en kurs utan kursmoment i listan. Dubbelklicka på raden märkt ”Dubbelklicka för att lägga till ny rad”. Välj `OK` efter datainmatningen

27. **Ta bort kursmoment ur systemet**

- **Beskrivning:** försöker ta bort kursmoment ur systemet
- **Indata:** ingen
- **Utdata:** det valda kursmomentet finns ej kvar i `FlexGrid`-komponenten
- **Utförande:** formuläret `Kurs` framme. Välj en kurs med moment i listan. Dubbelklicka på en rad med ett moment . Välj `Radera moment` och sedan `OK`

28. **Ändra ett kursmoment i systemet**

    - **Beskrivning:** försöker ändra ett kursmoment i systemet
    - **Indata:** moment != nuvarande värde
    - **Utdata:** den inmatade informationen syns på en rad i `FlexGrid`-komponenten
    - **Utförande:** formuläret `Kurs` framme. Välj en person i listan med minst en kurs. Dubbelklicka på en rad med ett moment. Välj `OK` efter datainmatningen

29. **Databasval**

    - **Beskrivning:** försöker öppna en giltig databasfil från valfri enhet
    - **Indata:** giltig databasfil
    - **Utdata:** inget felmedelande
    - **Utförande:** Formuläret `Intyg` framme. Välj menyalternativet `Arkiv` och sedan `Öppna databas`. Välj sedan en giltig databas och sedan `Öppna`

D Tabeller

Attribut-typ:	*Tidigare namn:*	*Nuvarande namn:*	*Information:*
Tabell	Kursdeltagare	Intyg	Förtydligande
Fältnamn	Person-ID	IntygsID	Förtydligande samt ”-”
Fältnamn	Ny		Ingen nuvarande funktion - borttagen
Tabell	KDKompbevis	Intygsrad	Förtydligande
Fältnamn	KDKomp-ID	radID	Förtydligande samt ”-”
Fältnamn	Person-ID	IntygsID	Förtydligande samt ”-”
Fältnamn	Utbildare	UtbID	Konsistens med tabellen Utbildare
Fältnamn	Person-ID	IntygsID	Förtydligande samt ”-”
Tabell	Utbildare	Utbildare	
Fältnamn	Utb-ID	UtbID	”-”
Fältnamn	Ny		Ingen nuvarande funktion - borttagen
Tabell	Kursmoment	Kursmoment	
Fältnamn	Moment-ID	MomentID	”-”
Fältnamn	Komp-ID	kursID	Förtydligande samt ”-”
Tabell	Kompbevis	Kurs	Förtydligande
Fältnamn	Komp-ID	kursID	Förtydligande samt ”-”
Fältnamn	Ny		Ingen nuvarande funktion - borttagen
Tabell	Bedömning	Bedömning	
Fältnamn	KDKomp-ID	radID	Förtydligande samt ”-”
Fältnamn	Moment		Ingen nuvarande funktion - borttagen
Fältnamn	moment-id	MomentID	”-”,

Tabell D.1: Ändrade/borttagna databasattribut

Attribut:	*Typ:*	*Beskrivning:*
IntygsID	långt heltal	Pimärnyckel, räknare
Förnamn	text[50]	Kursdeltagarens förnamn
Efternamn	text[50]	Kursdeltagarens efternamn
Personnr	text[12]	Kursdeltagarens personnummer
Startdatum	datum/tid	Kursdeltagarens utbildningstart på Lernia
Slutdatum	datum/tid	Kursdeltagarens utbildningslut på Lernia
Utskriven	Ja/Nej	Om kursdeltagaren är klar för arkivering
Anm	text[100]	Övrigt om kursdeltagaren

Tabell D.2: Attribut i databastabellen `Intyg`

Attribut:	*Typ:*	*Beskrivning:*
RadID	långt heltal	Pimärnyckel, räknare
IntygsID	långt heltal	Främmandenyckel från tabell Intyg
KursID	långt heltal	Främmandenyckel från tabell Kurs
UtbID	långt heltal	Främmandenyckel från tabell Utbildare
Kurslängd	byte	Kursensdeltagandets längd
Längdenhet	text[10]	Kursensdeltagandets längdenhet, t.ex dagar, veckor

Tabell D.3: Attribut i databastabellen `Intygsrad`

Attribut:	*Typ:*	*Beskrivning:*
UtbID	långt heltal	Pimärnyckel, räknare
Förnamn	text[50]	Utbildarens förnamn
Efternamn	text[50]	Utbildarens efternamn
Ämne	text[50]	Ämnet som utbildaren undervisar i

Tabell D.4: Attribut i databastabellen `Utbildare`

Attribut:	*Typ:*	*Beskrivning:*
KursID	långt heltal	Pimärnyckel, räknare
Kursnamn	text[50]	Kursens namn
Anmärkning	text[255]	Övrigt om kursen
längd	byte	Kursens längd
enhet	text[10]	Kurslängdens enhet, t.ex dagar, veckor

Tabell D.5: Attribut i databastabellen `Kurs`

Attribut:	*Typ:*	*Beskrivning:*
MomentID	långt heltal	Pimärnyckel, räknare
KursID	långt heltal	Främmandenyckel från tabell Kurs
Moment	text[255]	Beskrivning av momentet

Tabell D.6: Attribut i databastabellen `Kursmoment`

Attribut:	*Typ:*	*Beskrivning:*
MomentID	långt heltal	Sammansatt nyckel, främmandenyckel från tabell Kursmoment
RadID	långt heltal	Sammansatt nyckel, främmandenyckel från tabell Intygsrad
Bedömning	byte	Bedömning, 1=Kan, 2=Orienterad, 3=Uteslut

Tabell D.7: Attribut i databastabellen `Bedömning`