Systematisk udvikling af web-baserede systemer

Søren Madsen

LYNGBY 2005
EKSAMENSPROJEKT
NR. 04/05

IMM

Trykt af IMM, DTU

Forord

Dette projekt er udarbejdet ved Institut for Informatik og Matematisk Modellering på Danmarks Tekniske Universitet.

Projektet er udarbejdet perioden 1. September 2004 - 1. April 2005, under vejledning af Michael R. Hansen

---

Søren Madsen

Resumé

I forsøget på at forbedre udviklingen af web-anvendelser og undgå ad hoc løsninger, er mulighederne for automatisk at generere en web-anvendelse undersøgt. Til dette formål er en ikke-triviel web-anvendelse med en høj grad af datamodellering blevet brugt som case study. Kravet til de udviklede værktøjerne er, at de skal kunne generere web-anvendelsen automatisk på baggrund af en specification af systemet.

I dette speciale er der udviklet værktøjer til generering af web-baserede systemer på baggrund specificationen. Ud fra specificationen genereres der et relationsdatabaseskema med tilhørende integritetsbegrænsninger og basale brugerfunktioner til kommunikation med databasen. Der er udviklet generiske funktioner til anvendelse af generering af en grænsefladen til databasen på baggrund af typerne i specificationen.

Da den behandlede web-anvendelse er et typisk eksempel på et web-baseret system, er det vist at værktøjerne vil kunne modellere en bred vifte web-anvendelser.

Nøgleord: web-baserede systemer, databasedesign, automatisk generering, typesystem, prototype

Abstract

To improve development of web applications and to avoid ad hoc solutions, the possibilities for an automatic generation of web applications are examined. For this purpose a non trivial web application with a high degree of data modelling was used for a case study. The requirement for the tools developed is, that the process should be generated automatic from a specification of the system.

In this thesis, tools for generating web-based systems based on the specification were developed. From the specification of the system a relational database with integrity constraints and basic user functions were generated. Generic functions are developed for generating the interface with the database, based on the type declarations.

As the treated web application is a typical example of a web-based system, the tools are found to apply to a wide range of web applications.

Keywords: Web-based systems, database design, automatic generation, type system, prototype.

Indhold

Figurer

Tabeller

Kapitel 1

Introduktion

Dette speciale omhandler udvikling af web-baserede systemer. Et web-baseret system er et distribueret system. Systemet befinder sig på en web-server og kommunikation foregår via en web-browser. Mange web-baserede systemer introducerer forskellige teknologier (html, DBMS, java, flash, osv.) og er ofte udviklet på en ad hoc måde. Problemerne som dette indebærer er velkendte og er beskrevet i [GiM01]

En stor mængde web-baserede systemer er opbygget efter samme grundlæggende mønster. Der er en form for menusystem, til at styre sider og sektioner i systemet. Indholdet på siderne kan være meget forskelligartet og i mange tilfælde kræver det en dybere datamodellering at beskrive dette. Lagring af data foregår oftest i en relationsdatabase.

I specialet *Teoribaseret udvikling af pålidelige webbaserede systemer* [JeT03] [SAC03] er et web-baseret system med begrænset datamodellering behandlet. Deres tese var, at web-baserede systemer (i stor udstrækning) kan genereres ud fra beskrivelser af

1. Datamodellering og funktionelle krav
2. Brugerinteraktion

Tesen blev eftervist for et begrænset anvendelsesområde, hvor datamodelleringen kunne udtrykkes ved simple typeligninger.

Formålet i deres speciale var at medtage alle aspekterne i processen, men på bekostning af datamodelleringen. I processen er følgende aspekter be-

handlet: Specifikation af systemet, validering af input, flere brugere, adgangsrettigheder og navigationssikkerhed. Foruden begrænsningen i datamodellering var processen ikke fuldstændig automatisk.

I nærværende speciale lægges hovedvægten på en udvidelse af datamodelleringen således, at en stor mængde af systemer kan beskrives. Derudover skal det undersøges om også brugerinteraktionen kan genereres ud fra datamodelleringen.

Fordelene ved en sådan tilgangsvinkel er.

- Kort *time to market*.
- Høj kvalitet i form af konsistens fra web-grænsefladen til databasen.
- Ingen døde links i navigationen.
- Alle sider kan nås.

I nærværende projekt udvikles der et værktøj til generering af hele infrastrukturen (grænsefladen, brugerfunktioner, database) på basis af et langt mere udtryksrigt modelleringssprog end det fra [JeT03]. Det vises at værktøjet kan generere et kørende websystem for et ikke-trivielt case study. Til dette case study er der brugt det web-baserede system der findes på Danmarks Rocenter.

1.1 Idé

Idéen med specialet er, at der på baggrund af en enkel specifikation skal genereres et web-baseret system. Specifikationssproget er inspireret af typesystemer og modelleringsformer som kendes fra f.eks. funktionelle programmeringssprog SML [HaR99], Haskell og specifikationssprog som VDM og RAISE.

Ved at udvikle et passende typesystem, til at beskrive data i et web-baseret system, skabes der et større overblik over systemet. Typesystemet der udvikles skal være i stand til at beskrive data der optræder i web-baserede systemer generelt. Ved at udvikle et typesystem der kan beskrive sammensatte typer, kan semistruktureret data nemt beskrives. Typesystemet kan også udnyttes til hurtigt at udvikle en prototype af et web-baseret system.

Der gøres ikke noget forsøg på at udvikle et layout for det web-baserede system. Grunden til dette er, at to forskellige fremvisninger af de samme data kan være lige 'rigtige' og derfor kan der ikke gives nogen retningslinier for hvordan data bør repræsenteres.

1.2 Overordnede mål

Systemet består både af infrastrukturen, der håndterer persistent og konsistent lagring af data samt grænsefladen, der skal formidle kommunikation mellem brugeren og databasen.

Der skal udvikles et værktøj, der kan generere et skema til en database på baggrund af en specifikation.

Da de fleste web-baserede systemer i dag anvender relationsdatabaser til lagring af data, vil denne løsning også implementere en relationsdatabase til sikring af konsistent og persistent lagring.

Den mest generelle form for data er semistruktureret data, i dette speciale skal der udvikles en metode, der kan lagre dette i databasen uden tab af information.

Opbygningen af databasen skal være på en sådan måde at vedligeholdelse og udvidelse bliver så enkel som mulig.

Til at håndtere indholdet i databasen skal der genereres nogle brugerfunktioner, så der kan kommunikeres med databasen.

1.3 Relateret arbejde

Dette speciales tilgangsvinkel minder om den der anvendes inden for semistruktureret data, her modelleres data i XML og data lagres i en semistruktureret database.

Typesystemet som defineres i nærværende speciale kan beskrives vha. en kontekstfri grammatik. For semistruktureret data kan dette gøres med et XML skema [ABD00].

Fordelen ved at anvende en relationsdatabase til lagring af data er, at ydeevnen i en relationsdatabase er meget højere end i en semistruktureret database, især for web-anvendelser med store datamænger.

Af historiske årsager anvender mange Web-baserede systemer i høj grad relationsdatabaser til lagring af data. Dette skyldes at relationsdatabaser allerede var meget udbredte før Internettet kom frem. Den naturlige udvidelse til relationsdatabaserne var derfor at give adgang til dem fra en web-browser.

1.4 Specifikke mål

Til udvikling af værktøjer der automatisk kan generere web-baserede systemer, skal følgende krav opfyldes:

- Der skal udvikles et typesystem, der kan beskrive begreber i en række anvendelser af web-baserede systemer, hvor der typisk vil indgå mange skrukturerede værdier. Dette typesystem vil fungere som en specification for web-systemt.
- E/R diagram skal kunne genereres ud fra en specifikation, der er baseret på ovennævnte typesystem. Repræsentationen af E/R diagrammet skal som minimum indeholde information om entiteter, relationships, multipliciteten af relationships og markering af nøgler i entiteterne.
- Databaseskema skal genereres på baggrund af E/R diagrammet. Der skal udvikles et værktøj, der kan transformere et generelt E/R diagram til et databaseskema. Ved denne transformation skal normalformerne bevares.
- Brugergrænsefladen skal genereres på baggrund af specifikationen. Ud fra hvilke typer der er oprettet i databasen, skal der oprettes tilhørende funktioner på brugergrænsefladen.
- En software arkitektur skal udvikles som sikrer konsistens fra grænseflade til database.

Fokus i specialet har været at afprøve tesen, om at visse typiske webanvendelser kan genereres ud fra en simpel model og forstå begrænsningerne ved denne tilgangsvinkel. Der er ikke blevet lagt vægt på at lave et færdigt produkt og der er en række aspekter, der ikke er berørt. F.eks.

- Præsentation af data.
- Brugergrupper.
- Versionsstyring.

Kapitlerne er opbygget ved at begynde med et resumé, af hvad der bliver behandlet i kapitlet.

I specialet anvendes bl.a. emner inden for databaseteori, sprog og parsing.

Referencer er angivet i firkantede paranteser, og de findes i Litteraturlisten på side 83. I Appendiks A på side 85 findes der er oversigt over de væsentligste termer, der bliver anvendt.

Kapitel 2 Case study. Undersøgelse af et typisk web-baseret system der er repræsentativ for en stor mængde af Internet anvendelser. Systemet undersøges med henblik på at finde begreber der kan beskrive Internetsider generelt. Som eksempel undersøges hjemmesiden for det danske rocenter. Det er der to hovedårsager til, dels kendes opbygningen af hjemmesiden og begreberne omkring Danmarks Rocenter til mindste detalje og dels er der en høj grad af datamodellering på indholdssiderne.

Kapitel 3 Motivation for typesystem. Ud fra de begreber der blev fundet frem til i case studiet, udvikles der et typesystem der kan bruges til at beskrive en bred vifte af web-baserede systemer.

Kapitel 4 Databasedesign. Hvorledes typekonstruktorer i typesystemet kan transformeres til et E/R-diagram og videre over til et databaseskema.

Kapitel 5 Design af Grænsefladen. På baggrund af de typer der defineres i typesystemet forklares her, hvordan der kan sættes en web-baseret grænseflade til databasen op.

Kapitel 6 Implementation. Hovedfunktionerne der bruges til at generere databaseskemaet og sætte den web-baserede grænseflade op forklares i dette kapitel. Endvidere omtales de statiske funktioner der genererer grænsefladen på baggrund af typesystemet.

Kapitel 7 Konklusion. Status for specialet. Blev målene nået? Der gives også forslag til forbedringer af det implementerede system samt forslag til udvidelser.

Appendiks Her findes blandt andet en tabel med de termer der benyttes i rapporten, desuden er kildekoden til systemet listet her.

Kapitel 2

Case study

I dette kapitel beskrives et case study, der har en række egenskaber, der er typiske for en stor klasse af Internet anvendelser.

- Lagring af data i database.
- Browser tilgang.
- Forskellige slags brugere, med forskellige rettigheder.
- Høj grad af datamodellering.

Et begrebsapparat der kan modellere dette case study, vil være godt rustet til at dække en bred vifte af Internet anvendelser.

Eksemplet er valgt, fordi undertegnede har indgående kendskab til Danmarks Rocenter i forbindelse med daglig gang på stedet igennem en årrække.

Der vil igennem resten af specialet blive refereret tilbage til denne anvendelse, da den besidder mange af de egenskaber der bliver behandlet i specialet.

2.1 Danmarks Rocenter

I det følgende beskrives såvel anvendelsen, som den i dag findes på rocenteret, som ønskelige udbygninger af denne.

2.1.1 Eksisterende egenskaber

Først gennemgås de egenskaber, der allerede er implementeret på rocenterets hjemmeside.

Almene meddelelser

Danmarks Rocenters hjemmeside blev oprindeligt oprettet, fordi der var behov for et sted, hvorfra ledelsen kunne kommunikere med roerne. De meddelelser der bliver givet på siden, er alle knyttet til den daglige drift af rocentret. Et eksempel kunne være en besked om, at fysioterapeuten er forhindret i at komme en given dag.

Med tiden er der kommet udvidelser til systemet således, at der nu ligeledes findes information om de forskellige roere, der er tilknyttet landsholdet. Informationen dækker foruden personlige data også hvilke hold den enkelte roer er på samt hvilke resultater, der er opnået. Der er 24 forskellige hold/bådtyper at vælge imellem. Alle både har en unik forkortelse og beskrives bl.a. ved: navn, antal personer i båden og om båden er med på det olympiske program. Foruden aktuelle oplysninger om roeren er der tilknyttet en oversigt over hvilke resultater, den enkelte roer har opnået.

Adressefortegnelse

Der vedligeholdes en adressefortegnelse over de roere/ledere der er aktive i Danmarks Rocenter. Oplysningerne trækkes fra de informationer som brugerne af systemet selv har indtastet.

Kalender

En kalender indeholdende de aktiviteter som er relevante for brugerne af Danmarks Rocenters hjemmeside.

Indkøbskurv

På siden er der mulighed for at bestille landsholdstøj, dette forgår via en indkøbskurv, hvor brugeren ledes igennem bestillingen som afslutter med leveringsoplysninger.

2.1.2 Nye tiltag

Mulige forbedringer og udvidelser man kan forestille sig til Danmarks Rocenters hjemmeside er nævnt nedenunder.

Træningsprogrammer

Træningsprogrammer udarbejdes for 3-4 uger ad gangen og udleveres ca. en uge inden de skal tages i brug. Ved at lave træningsprogrammerne online undgås distributions problemer og dataopsamling i forbindelse med den daglige træning bliver også mulig.

Dagbog

Ved systematisk indsamling af træningsresultater kan der laves forskellige udtrækninger af resultater fra træningen og på den måde fås et bedre overblik over, hvilken træning der giver de største fremskridt. Selve indsamlingen bygger på træningsprogrammerne.

Registrering af test

Flere gange om året afholdes der test på roergometer[1], dette består af en 'ugetest', hvor den enkelte roer udfører fem tests i løbet af en uge. Testen følges op af en test i et af Team Danmarks testcentre hvor de forskellige fysiologiske parametre måles.

Ranglister

På baggrund af de af roerne indtastede data kan der laves forskellige ranglister til brug for både roere og trænere.

[1] Romaskine der bruges til indendørs rotræning om vinteren. Der afholdes også diverse mesterskaber i roergometer

Tilmelding til kaproninger

Hver gang landsholdet deltager i kaproninger skal der foretages tilmelding. En udvidelse til systemet kunne være at denne tilmelding skal ske via systemet.

Resultater fra kaproninger

Der findes kun en resultatdatabase, den vedligeholdes af FISA[2], det international roforbund, og den indeholder kun resultater fra regattaer afholdt af FISA, dvs. World Cup regattaer, Verdensmesterskaber samt Olympiske Lege. Det ville være interessant at have en database, der indeholder tider for alle de danske roere.

Tidsreservation hos fysioterapeuten

For at gøre brugen af fysioterapeuten mere strømlinet kunne der indføres et web-baseret reservationssystem.

Forum

Til diskussion af dagligdagens begivenheder.

Tilmelding til spisning

I løbet af sæsonen er der i de weekender, hvor der ikke er træningslejr eller kaproning fællesspisning på rocenteret for landsholdsroerne. For at optimere madlavningen er det oplagt med en koordineret tilmelding.

2.2 Præcisering af begreberne

I dette afsnit uddybes de væsentligste begreber, som optræder på rocenterets hjemmeside, med.

[2]Fédération Internationale des Sociétés d'Aviron

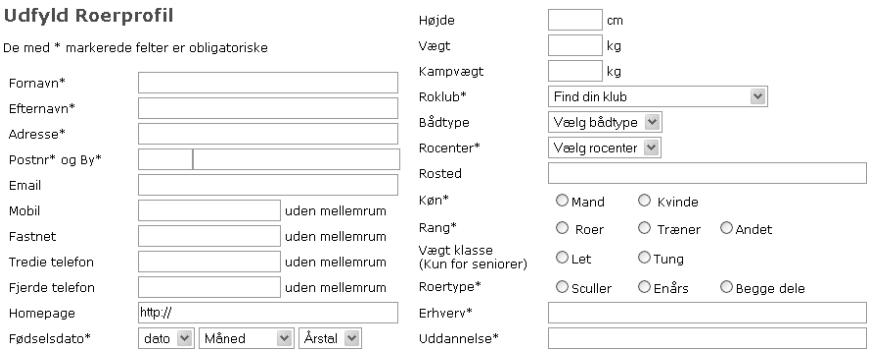
2.2.1 Almene Meddelelser

En *meddelelse* som optræder på hjemmesiden består af en *overskrift*, en *meddelelsestekst* og en *dato*. Meddelelserne vises på en *liste* med nyeste meddelelse øverst. Når der indtastes en ny *meddelelse* kommer den øverst på listen og skubber den nederste på listen ud.

Foruden at *meddelelserne* står på forsiden bliver de også vist i et arkiv.

2.2.2 Person Kartotek

Roerne er i systemet repræsenteret med deres personlige data. En opremsning af de vigtigste følge her: *navn*, *klub*, *bådtype*, *rocenter* (Der er både et hovedcenter, to regionalcentre og fem kraftcentre), *adresse*, *email*, *fødselsdato* (der indeholder typerne *dato*, *måned* og *år*) samt *telefonnummer*. Se Figur 2.1.



Figur 2.1: *Hovedparten af de data der indtastes når der oprettes en roerprofil i systemet på rocentret*

Desuden er der en del oplysninger, der retter sig til selve roningen. Typen *vægtklasse* indeholder information om roerens vægtklasse, som enten kan være *let* eller *tung*. En anden type, kaldet *roertype* indeholder information om hvilke type roeren er. Der er 3 muligheder:

- *Sculler* I sculler ros med en åre i hver hånd.
- *Enårs* Her er der som navnet antyder, kun en enkelt åre at holde fast på.

- *Begge* Både sculler og enårs roning beherskes.

For en komplet oversigt over typerne i personkartoteket se da Tabel B.1 på side 87. Til persondata hører der også en samling af resultater som personen har opnået. Et resultat er beskrevet vha. de fysiske faktorer: tidspunkt (dato), tid (den opnåede tid), begivenhed (VM, OL,...), placering og sted.

2.2.3 Kalender

Årets vigtigste begivenheder fremgår af en kalender. *Kalenderen* har følgende funktionalitet. På forsiden vises den aktuelle måned samt den næste måned. *Begivenhederne* i kalenderen optræder med forskellig farve alt efter hvilken *begivenhedstype*, der er tale om.

De øvrige attributter der er vedhæftet en begivenhed i kalenderen er *startdato*, *slutdato*, *navn* og *ekstra*. Under ekstra kan der angives en vilkårlig tekst der knytter sig til begivenheden (typisk adressen på den hjemmeside der knytter sig til begivenheden).

2.2.4 Indkøbskurv

Der er mulighed for at købe landsholdsrotøj, dette foregår ved, at der udvælges de *varer* der ønskes på en bestillingsside. Bestillingen afsluttes ved at udfylde en formular med leveringsoplysninger.

En *vare* er kendetegnet ved et *varenummer*. Varenummeret indeholder både information om *varetype* og *størrelse*. *Indkøbskurven* består af en liste af *varenumre*.

2.2.5 Træningsprogrammer

De træningsprogrammer der udføres kan generelt inddeles i fem forskellige kategorier alt efter hvilken *træning*, der er tale om. Disse fem kategorier er: *Intervaltræning*, *distancetræning*, *udholdenhedstrænig*, *tekniktræning* eller *test*. Foruden programmet er der også en *beskrivelse* knyttet til programmet, teksten er en kort sammenfatning af hvad programmet består af. Et eksempel kunne være "20'+7'+3' T24-26-28", hvor 20'+7'+3'er tre tidsangivelser og T24-26-28 er tre angivelser af tempi, yderligere forklaringen følger nedenfor.

Intervaltræning er kendetegnet ved, at *intervallerne* hvor man ror er angivet som en *tid*, desuden er der også en *pause* angivelse og en *tempo* angivelse. Pausen er en tidsangivelse mens tempoet er et heltal der angiver tagfrekvensen, dvs. antallet af rotag per minut. Intervaltræningen er en samling af intervaller.

Distancetræning er træning der hovedsagelig udføres når højsæsonen nærmer sig. Selve træningen består af flere korte distancer hvor *afstanden* angives i *meter* med en fastlagt *pause* imellem, denne bliver angivet enten som en *afstand* eller en *tid*. Distancetræningen er altså en samling af distancer.

Udholdenhedstræning har en *længde* angivet som en tid. I dette tidsrum skifter tempoet som tiden går. Som i eksemplet fra tidligere kunne det være 30 minutter, bestående af 20 min i tempo 24, 7 min i tempo 26 og 3 min i tempo 28.

Tekniktræning angives ved afstand samt muligvis en kommentar om hvilke teknikøvelser, der skal laves.

Test er den sidste form for træning. En test beskrives ved hvor lang denne er, hvilke formål den har, hvor den er udført samt eventuelle kommentarer til denne. Derfor skal der angives en *længde* der enten kan bestå af en *tid* eller en *distance* de sidste oplysninger er `sted`, `formaal`, `kommentar` og `beskrivelse`.

2.2.6 Dagbog

Til at registrere den daglige træning kan der bruges en dagbog. For hver person der er registreret i systemet, kan der kombineres med de træningsprogrammer, der skal udføres den pågældende dag.

For roning er det sådan at et program hvor *længden* er angivet med en *afstand*, så bliver *resultatet* af typen *tid* og vice versa.

2.2.7 Registrering af test

En speciel form for træning er, når der er tale om en test, til denne træning er der nemlig knyttet flere resultatdata end ved almindelig træning. De ekstra testresultater er: *Effekt*, *mælkesyre*, *ventilation*, *iltoptagelse* og andre fysiologiske målbare størrelser.

2.2.8 Ranglister

Ved at sammenligne testresultater for en test på tværs af roerne kan der gives et overbliksbillede af, hvem der for tiden ror hurtigst. Denne anvendelse kræver ikke yderligt input fra brugeren udover hvilken test det er, for at sammenligningen kan udføres.

2.2.9 Tilmeldinger og ansøgninger

Når der skal tilmeldes de internationale kaproninger: World Cups og verdensmesterskaber er der en mængde standard oplysninger, der skal udfyldes. Det samme gælder hvert efterår, når der skal skrives støtte ansøgninger til Team Danmark. Ansøgningerne skal som tilmeldingerne have et fast format.

Et værktøj der selv kan udfylde de rigtige ansøgninger og tilmeldinger på baggrund af det data, der er registreret, vil derfor være en stor hjælp, både med henblik på at formindske fejl, men også for at spare tid.

Kapitel 3

Motivation for typesystem

På baggrund af det case study der blev foretaget i Kapitel 2 på side 21, skal der findes en notation, der på en simpel og overskuelig måde kan bruges til at repræsentere begreberne i eksemplerne.

Til modellering af de begreber der er fundet i anvendelsen, har det vist sig, at det er nok med simple typer og nogle få sammensatte typer, hvor de sammensatte typer rummer: Kartesisk produkt, lister, disjunkt forening og tabeller.

Der redegøres for begreberne simple- og sammensatte typer i nærværende kapitel. Til forståelsen hører definitionen af en primitiv type, denne er givet i nedenstående.

3.1 Primitive typer

Eksemplerne indeholder en række begreber som kan modelleres med typer som heltal, decimaltal, tekststrenge, osv. Eksempler på disse fra personkartoteket kunne være: `højde` som er et heltal eller `fornavn` og `efternavn` der begge er en tekststreng.

Disse grundlæggende typer betegnes herefter som *primitive* typer. Ved en primitiv type forstås en af følgende `"int"`, `"string"`, `"real"` og `"bool"`

Andre nyttige typer der kunne tilføjes denne mængde kunne være `email` eller `dato`. I denne begrebsmodellering er det valgt kun at medtage de nævnte primitive typer, da det ikke tilføjer yderlig kompleksitet at medtage mange typer. En liste over hvilke typer der kunne være oplagte at understøtte, fås ved at undersøge hvilke typer der er understøttet i det database håndterings system (DBMS), som systemet skal bygge på. I MySQL findes der 29 indbyggende typer, en liste over disse kan findes i Appendiks A.1 på side 86.

3.2 Simple typer

For at kunne differentiere mellem de primitive typer, er det nødvendigt at indføre en ny type. En *simpel* type er defineret som primitiv type, men med sit eget typenavn.

En *simpel* type er en navngiven primitiv type. f.eks. `type navn = string`, `navn` er simpel og `string` er primitiv.

Bemærk at der i typesystemet skal være navneækvivalens og ikke kun strukturelækvivalens. `type navn = string` og `type beskrivelse = string`, er to forskellige typer. Typesystemet skal have denne egenskab, da der skal kunne skelnes mellem de navngivne typer.

3.3 Sammensatte typer

Mange af de sammensatte begreber fra anvendelsen kan beskrives ved hjælp af nogle få typekonstruktorer, som benyttes til at danne sammensatte typer ud fra allerede givne typer.

Disse typekonstruktorer gennemgås i det følgende.

3.3.1 Kartesisk produkt

Et *kartesisk produkt* er defineret som produktet af en række indgående typer (simple eller sammensatte).

Typeudtrykket $A_1 * A_2 * \ldots * A_n$ beskriver mængden af værdier $(a_1, a_2, \ldots, a_n)$, hvor a_i har typen A_i. Et eksempel fra anvendelsen kan ses i kalenderen, der består af en samling begivenheder, hvor en begivenhed erklæret ved:

```
begivenhed = startdato * slutdato * navn * ekstra
```

Her er en `begivenhed` i kalenderen erklæret ved brug af et kartesisk produkt. Begivenheden består af 4-tuplen (startdato, slutdato, navn, ekstra)

I eksemplet fra anvendelsen er `startdato` og `slutdato` begge erklæret ved brug af en sammensat type. Både `startdato` og `slutdato` er erklæret som en dato. (`startdato = dato` og `slutdato = dato`), altså et specialtilfælde af kartesisk produkt, hvor $n = 1$.

Eksemplet illustrerer også navneækvivalensen, grunden til at en begivenhed i kalenderen ikke kan erklæres som `begivenhed = dato * dato * navn * ekstra` er, at typenavnene senere skal bruges til at navngive attributterne i databasen.

3.3.2 Lister

Ved brug af typekonstuktoren `list` er det muligt at definere en sammensat type, der er en liste af elementer med samme type.

Typeudtrykket $A\ list$ beskriver den endelige (muligvis tomme) sekvens af værdier $a_1, a_2, \ldots, a_n$ hvor a_i er af typen A

Et eksempel fra anvendelsen kan findes under træningsprogrammer, hvor intervaltræning består af en liste af intervaller.

```
intervaltræning = interval list
```

Placeringen af interval elementerne er ikke ligegyldig, rækkefølgen har altså en betydning, denne rækkefølge opretholdes af listkonstruktionen.

3.3.3 Disjunkt Forening

Flere steder er det observeret at et begreb repræsenterer et valg blandt flere muligheder. Der er altså brug for en type der kan repræsentere dette valg.

Typeudtrykket $A_1|A_2|\ldots|A_n$ beskriver en disjunkt forening af værdier fra typerne $A_1, A_2, \ldots, A_n$ hvis $A_i = A_j \Rightarrow i = j$. En disjunkt forening er

kendetegnet ved, at fællesmængden mellem to vilkårlige indgående typer er tom, netop en type kan vælges ad gangen.

Fra eksemplet anvendes disjunkt forening i forbindelse med træningsprogrammer. Et træningsprogram kan være et blandt flere:

```
træning = intervalTræning | distanceTræning | ... | test
```

Dette udtrykker at træning er præcis en af de typer, der er specificeret på højresiden.

3.3.4 Tabel

I mange anvendelser findes *nøgler* der entydigt identificerer visse værdier. For at understøtte dette indføres begrebet en *tabel*.

I anvendelsen er der flere steder, hvor det er oplagt at gemme i et register. Eksempler er: `person`, `bådtype` og `træningsprogram`. Ved at introducere typekonstruktoren *tabel*, der skal fungere som lager, kan man ordne flere instanser af den samme type.

Typeudtrykket $(S_1 * S_2 * \ldots * S_n, A_1 * A_2 * \ldots * A_m)$ beskriver tabelindgange af formen $(s_1, s_2, \ldots, s_n, a_1, a_2, \ldots, a_m)$, hvor $(s_1, s_2, \ldots, s_n)$ tilsammen udgør nøglen for en indgang i tabellen. Typen på den i'te delnøgle er S_i. Tuplen $(a_1, a_2, \ldots, a_m)$ er de værdier, der er associeret med den givne nøgle. a_i har typen A_i.

For delnøglerne S_i gælder det, at de alle skal være en simple type. For attributterne er der ingen begrænsninger for typen.

Et eksempel fra anvendelsen kunne være personkartoteket, som unik reference kan bruges: `navn` og `fødselsdato` hvor resultatet så ville være `person`, der indeholder alt information om personen, der svarer til den reference der blev givet.

3.4 Type erklæringer

Typeudtryk for de begreber der er undersøgt i ovenstående er i høj grad inspireret af det typesystem der anvendes i ML [HaR99].

De forskellige typer erklæres som:

Simple typer : `type typeNavn = ptype`
Hvor `ptype` $\in$ `{int, string, real, bool}`
og `typeNavn` er navnet på den type der erklæres.

Kartesisk produkt : `type typeNavn =` t_1 `*` t_2 `* ...*` t_n
Hvor `typeNavn` er navnet på den kartesiske type der erklæres,
og t_i er en tidligere erklæret type.

Lister : `type` $typeNavn_B$ `=` $typeNavn_A$ `list`
Hvor $typeNavn_B$ er navnet på den liste type der erklæres.
og $typeNavn_A$ er typen på de variable som listen indeholder.

Disjunkt forening : `disjunkt typeNavn =` t_1 `|` t_2 `| ...|` t_n
Hvor `typeNavn` er navnet på den disjunkte type der erklæres,
og t_i er en tidligere erklæret type, desuden gælder der at $t_i \cap t_j = \emptyset$ for $i \neq j$.

Tabel : `table typeNavn = (`s_1`*`s_2`*...*`s_n`, `t_1`*`t_2`*...*`t_m`)`
Hvor `typeNavn` er navnet på den tabel der erklæres, og $s_1 * s_2 * \ldots * s_n$ er en nøgle, der identificerer en indgang i tabellen. Værdien af indgangen er $t_1 * t_2 * \ldots * t_m$.

3.5 Eksempel på type erklæringer

Et reduceret eksempel med baggrund i anvendelsen, vil med de beskrevne typeudtryk kunne udtrykkes ved:

```
type fornavn          = string
type efternavn        = string
type alder            = int
type tid              = int
type tidspunkt        = int
type letTræning       = tid
type navn             = fornavn * efternavn
type person           = navn * alder
type hårdTræning      = letTræning list
datatype træning      = letTræning | hårdTræning
table personkartotek  = (navn, person)
table træningsdagbog  = (tidspunkt * navn, træning)
```

Det lille eksempel illustrerer brugen af de forskellige typekonstruktorer. Først er der erklæret en række primitive typer til at indeholde værdierne, bl.a. typen `tid` der er tænkt til at skulle indeholde antallet af minutter,

`navn` er dannet som det kartesiske produkt mellem `fornavn` og `efternavn`. Typen `person` defineres som det kartesiske produkt mellem `navn` og `alder`. `letTræning` er blot en `tid` i dette eksempel. `hårdTræning` er erklæret som en liste af `letTræning`. Den disjunkte forening `træning` bestående af `letTræning` og `hårdTræning` har det gensidigt udelukkende valg mellem let og hård træning.

Til slut er der oprettet to tabeller til at indeholde de modellerede værdier, `personkartotek` og `træningsdagbog`. `personkartotek` er et register der indeholder værdien `person`, som nøgle til registeret bruges `navn`. Det andet register `træningsdagbog` har som nøgle `tidspunkt` og `navn` og indeholder som værdi `træning`.

3.6 Signatur til begrebsmodellering

Ud fra de typer og typekonstruktorer der blev introduceret i Afsnit 3 på side 29, bliver eksemplet fra Afsnit 2 på side 21 gennemgået med henblik på at knytte begreberne sammen med typer og typekonstruktorer. I Appendiks C.1 på side 90 er typerne for webanvendelsen listet.

Nedenstående er en fuld modellering af træningsprogrammer fra anvendelsen.

Træningsprogrammer

```
type beskrivelse = string
type tid         = int
type afstand     = int
type pause       = int
type tempo       = int
type sted        = string
type formaal     = string
type kommentar   = string

disjoint basal   = afstand | tid
type serie       = basal list
disjoint session = basal | serie

type distance    = afstand * pause * tempo
```

```
type interval    = tid * pause * tempo

type distanceT   = distance list
type intervalT   = interval list
type udholdT     = session list
type teknikT     = basal * tempo
type test        = basal * sted * formaal * kommentar * beskrivelse

disjoint program = intervalT | distanceT | udholdT | teknikT | test
table traening   = (beskrivelse, program)
```

Typerne som de ser du for træningsprogrammer i anvendelsen er

3.7 Grammatik for typesystemet

Til at beskrive en vilkårlig specification kan der opstilles følgende grammatik.

$$G_{type} \quad = \quad (V_{type}, T_{type}, P_{type}, S_{type}) \tag{3.1}$$

V_{type} variable :
`typeNavn`
T_{type} Terminal symboler :
`* ,| ,( ,) ,= ,',' ,list ,type ,disjoint ,table`
P_{type} Produktions regler :

```
        spec  →  decl spec | ϵ
        decl  →  typeDecl | disjDecl | tableDecl
    typeDecl  →  type typeName = typeExpr
    disjDecl  →  disjoint typeName = typeExpr
   tableDecl  →  table typeName = (typeExpr, typeExpr)
    typeExpr  →  pType | typeName | cType | lType
       pType  →  int | string | float | boolean
       cType  →  typeExpr | typeExpr * typeExpr
       lType  →  typeExpr list
```

(3.2)

S_{type} Start symbol :
`spec`

Grammatikken kan bruges til at validere specification der er skrevet ved hjælp af det udviklede typesystem, og den kan også bruge ved udvikling af en parser til generering af et typetræ.

Kapitel 4

Databasedesign

I forbindelse med opbygning af det web-baserede system er der brug for en persistent lagring af data. Til det formål anvendes en relationsdatabase.

Lagring af data i databasen skal ske uden tab af information. Dvs. både værdierne og den struktur som de sammensatte typer har, skal bevares.

Genereringen af databasen er delt op i tre trin. Første trin er en parsing af signaturen, der som resultat har en liste af typeerklæringer, som beskrevet i grammatikken i Afsnit 3.7 på side 35, Figur 4.1 viser en oversigt over transformationerne.

Figur 4.1: *Generering af databaseskema*

Ved oversættelsen fra listen af erklæringer skal hver enkelt erklæring fra signaturen behandles forskelligt, dels har typen betydning for behandlingen, men også den sammenhæng som den indgår i påvirker transformationen. De forskellige tilfælde bliver systematisk gennemgået i Afsnit 4.1 på næste side. Resultatet af denne oversættelse bliver en mængde entitetssæt og en mængde relationships, der beskriver forholdene imellem entitetssættene. Tilsammen udgør disse E/R diagrammet.

Sidste skidt mod databaseskemaet er transformationen af E/R diagrammet til et databaseskema. Transformationen følger de anvisninger der er givet

i [GUW02], i Afsnit 4.3 på side 45 bliver der argumenteret for de valg der skal tages ved transformationen, se Afsnit 4.3 på side 45.

Sidst i kapitlet undersøges databasen for redundans vha. normalformerne.

4.1 Generering af E/R diagrammet

Beskrivelsen af hvorledes simple typer og sammensatte typer bliver til attributter, entitetssæt og relationships, er beskrevet i dette afsnit.

De forskellige typer behandles forskelligt afhængig af typen og den sammenhæng som de indgår i.

4.1.1 Notation

Til at betegne de forskellige slags relationships benyttes notationen i Tabel 4.1.

Notation	Diagram notation	Betydning
1-1		En til En
1-n		En til Mange
p-1		Præcis en til En
p-n		Præcis en til Mange
n-n		Mange til Mange

Tabel 4.1: *Notation i forbindelse med relationship*

Et *p-1* relationship `R` mellem entitetssættene `A` og `B` repræsenterer integritetsbegrænsningen, at hvis der eksisterer et relationship mellem `A` og `B` så kan en entitet i `B` kun eksistere, hvis den relaterede entitet er oprettet i `A`. Dette angives i E/R diagrammet med et buet pilehoved.

Det samme gør sig gældende for et *p-n* relationship mellem entitetssættene `A` og `B`, her er der blot flere entiteter i `B`, der kan være afhængig af eksistensen af en entitet i `A`.

Entitetssæt markeres med rektangler og relationships med romber. Både svage entiteter og svage relationships markeres med dobbelt ramme.

Under transformationen bliver der kun genereret binære relationships, hvilket betyder, at der ikke eksisterer flervejs relationships.

4.1.2 Hovedidé bag transformationen

Under genereringen af entitetssættene skal der fastlægges en entydig nøgle. Der er to umiddelbare måder dette kan foregå på:

En metode er automatisk at generere en nøgle til hvert entitetssæt og tilhørende *1-1* relationships imellem disse. Der kan også benyttes en metode hvor nøglerne i så stor udstrækning som mulig, opnås ved at se på i hvilken sammenhæng entitetssættet benyttes i, sammenhængen mellem entitetssæt bliver så i stedet etableret vha. fremmednøgler.

Fordelen ved at benytte den første metode er, at de relationer der i sidste ende vil blive implementeret i databasen, vil have attributter, der svarer til de simple typer samt netop en automatisk genereret nøgle.

I den anden metode vil attributterne igen være at finde i de samme relationer, men ved denne metode, kan der ikke siges noget om hvor mange nøgler, der er i de enkelte relationer. Dette afgøres af i hvilke sammenhænge den aktuelle type optræder.

Den sidste metode vælges, fordi det i databasedesign gælder om at lave designet så simpelt som muligt. Ved metoden med fremmednøgler undgås *1-1* relationer mellem de entitetssæt, der indeholder de modellerede typer og der oprettes ikke ekstra nøgler.

Entiteter i E/R diagrammet

Ved generering af entiteter til E/R diagrammet opdeles typerne i simple og sammensatte typer. Entiteterne findes som de sammensatte typer i signaturen, og attributterne stammer fra de simple typer.

Der er små forskelle i hvordan entiteterne oprettes, alt afhængig af hvilken sammensat type der er tale om. Entiteterne markeres i E/R diagrammet med hvilken sammensat type de stammer fra, således at der ikke går information tabt, når signaturen bliver transformeret til et E/R diagram.

Hvis et entitetssæt ikke har en unik nøgle skal den markeres som svag. På denne måde signaleres det, at den resterende del af nøglen skal findes ved at undersøge de svage relationships, der støtter entitetssættet.

I databasedesignet som anvendes her antages det, at den nøgle der er givet ved erklæring af en tabel er unik. For de øvrige sammensatte typer vides

der ikke noget om nøglerne og det kan ikke antages, at der er en nøgle i disse. For entitetssæt der ikke er unikke, skal der hentes fremmednøgler i de entitetssæt som de er forbundet til.

Når der skal oprettes et entitetssæt hvor attributterne ikke kan antages at udgøre en nøgle, ken en af disse løsningsmetoder anvendes: Tillad kun at der oprettes præcis en entitet i entitetssættet, tillad at der oprettes entiteter så længe de er forskellige eller tillad, at der kan tilføjes en nøgle til entitetssættet.

For at afgøre hvilken metode der skal vælges ses på hvornår situationen opstår. Dette sker netop når entitetssættet ikke bliver refereret til fra nogen anden entitet, altså når entiteten er i toppen af type hierarkiet. Entiteter i toppen af hierarkiet vil typisk definerer udseendet (sektionerne) af det web-baserede system. I anvendelsen har typen i toppen af hierarkiet følgende udseende: `type drc = forsiden * information * meddelelser * ... * kalender * bådhold`

Et kartesisk produkt som topniveau vil være en typisk situation, denne kunne bruges til at definere sektionerne på web-systemet. Ønskes der to entiteter af det kartesiske produkt kan dette effektueres ved at oprette en liste, der indeholder det kartesiske produkt. En liste i toppen af hierarkiet vil føre til at web-systemet fremstår med en liste, ønskes to lister, kan dette igen implementeres ved brug af en liste, og man er tilbage ved udgangssituationen. En disjunkt forening som topniveau i hierarkiet vil bevirke, at kun en af typerne i den disjunkte forening vil være synlig, flere entiteter af den disjunkte forening kan igen klares ved brug af en liste.

Tilsammen giver denne analyse følgende muligheder for topniveauet, ønskes flere entiteter i et entitetssæt hvor der ikke er defineret nogen nøgle, kan dette opnås ved at lave en liste af disse. Imidlertid er en liste ikke den mest generelle måde at repræsentere flere entiteter af den samme type på. Til det vil typen tabel være mere velegnet, da nøglerne her angives af brugeren. I toppen af hierarkiet er der derfor kun tre forskellige muligheder. Enten er det en kartesisk entitet, en entitet indeholdende en disjunkt forening eller en tabel. Problemet med kartesisk produkt og disjunkt forening er, at de ikke har nogen nøgle defineret. Den disjunkte forening har ingen attributter og det kan ikke garanteres, at det kartesiske produkt indeholder en simpel type, der kan anvendes som nøgle. En løsning kunne være at lave en dummy attribut, der kunne fungere som nøgle, en anden mulighed ville være at bruge typenavnet på entiteten som nøgle. I begge tilfælde

minder situationen meget om at oprette en tabel og definere en nøgle til entitetssættet.

Konklusionen på analysen bliver, at der altid skal være en tabeltype som øverste niveau i det web-baserede system. Dette er ikke en beslutning, der begrænser web-systemet, tabellen bruges blot som udgangspunkt for systemet.

Relationships i E/R diagrammet

Under behandlingen af de sammensatte typer giver det anledning til et relationship, hver gang en sammensat type indeholder en anden sammensat type. De forskellige typer af relationships er forklaret ved gennemgangen af de forskellige sammensatte typer.

Oversættelsen af de fire forskellige sammensatte typer følger nedenstående fremgangsmåde.

4.1.3 Behandling af sammensatte typer

For at behandle alle tilfælde, gennemgås hver af de fire sammensatte typer. Under behandlingen af hver enkelt type redegøres der for hvordan de forskellige indgående typer skal behandles.

Der anvendes i gennemgangen følgende forkortelser for en simpel type og de forskellige sammensatte typer, se Tabel 4.2.

ST	En simpel type
KT	Et kartesisk produkt
LT	En liste med en vilkårlig type som listeelement
DT	En disjunkt forening
TT	En tabel

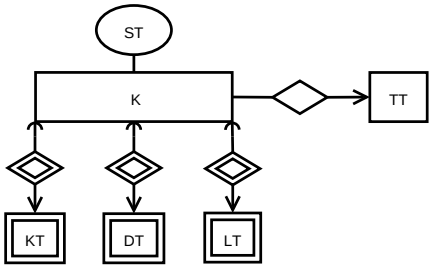
Tabel 4.2: *Forkortelser*

Kartesisk produkt

Det generelle udtryk for et kartesisk produkt er defineret ved:

```
K = ST * KT * DT * LT * TT
```

E/R diagrammet ses i Figur 4.2.



Figur 4.2: *E/R diagram af et generelt kartesisk produkt.*

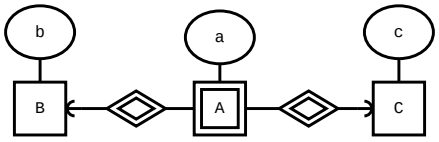
Simple typer der indgår i det kartesiske produkt er afhængige af eksistensen af det kartesiske produkt. Det samme gælder for de indgående sammensatte typer med undtagelse af `tabel` typen, derfor markeres disse med et *p-1* relationship, denne markering svarer til den måde attributterne behandles på. En tabel kan blive refereret til fra flere entiteter i entitetssættet derfor erklæres dette relationship som *n-1*. Entitetssættene, af typen `KT`, `DT` eller `LT`, der indgår i det kartesiske produkt erklæres alle for svage, fordi de skal samle deres nøgler fra det entitetssæt, som de er knyttet til.

Hvis en sammensat type på denne måde indgår i flere sammensatte typer vil der blive genereret et svagt relationship til hver af disse, og der vil blive oprettet nøgler tilsvarende. Det har den effekt, at hvis en sammensat type indgår i to forskellige typer, så vil der blive oprettet to fremmednøgler i denne, den ene af fremmednøglerne vil altid være udefineret, da en vilkårlig entitet kun giver anledning til et af de to relationships, i Figur 4.3 på modstående side er vist et eksempel på denne situation:

```
type A = a
type B = b * A
type C = c * A
```

hvor `a`, `b` og `c` er simple. `A`, `B` og `C` er alle kartesiske produkter og derfor sammensatte. En entitet i `B` vil medføre en entitet i `A`. I `A` vil der være en fremmednøgle der referere til `B`, men referencen til `C` vil være udefineret.

Denne egenskab er velegnet da nøglen fra blot det ene entitetssæt er unik. Da det er et *p-1* relationship, der er imellem entitetssættene bliver entitetssættet `A` også unik.

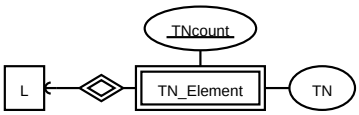


Figur 4.3: *Integritetsbegrænsninger når en sammensat type indgår i to forskellige sammensatte typer*

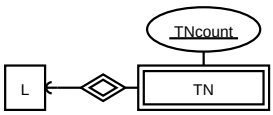
Lister

En liste defineres ud fra en vilkårlig ikke-primitiv type, den generelle repræsentation ser ud som på Figur 4.4 og Figur 4.5, typeligningen er givet ved:

`L = TN list` , hvor `L` og `TN` er typenavne.



Figur 4.4: *`TN` er en simpel type*



Figur 4.5: *`TN` er en sammensat type*

Der er to forskellige situationer, enten er `TN` defineret ud fra en simpel type, eller den er defineret via en sammensat type, det er ikke tilladt at anvende primitive typer til at definere listen ud fra.

I tilfældet hvor listeelementet er defineret ved en simpel type, skal der oprettes et entitetssæt til at indeholde entiteterne med den simple type. Dette entitetssæt navngives som på Figur 4.4 ved: `TN_Element`. Som attributter er der elementerne `TN` og `TNcount`. Rollen `TNcount` opfylder er dels som ordning af elementer i listen og dels som delnøgle til entitetssættet `TN_Element`. Ved denne konstruktion skabes der altså to entitetssæt.

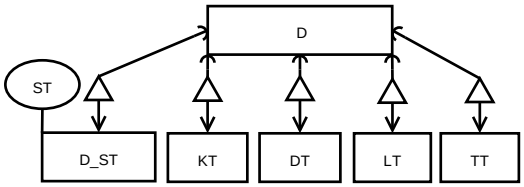
Er listeelementet i stedet defineret ved brug af en sammensat type, så eksistere entitetssættet der udgør listeelementet allerede og der skal blot tilføjes en delnøgle `TNcount` til dette entitetssæt.

I begge tilfælde skal entitetssættet der indeholder listeelementet markeres som svagt og der skal oprettes et svagt relationship til list entitetssættet `L`. Ved at indhente delnøglen fra `L` kan listeelementet danne en primær nøgle.

Disjunkt forening

Denne konstruktor giver anledning til et hierarki af typer, dette skyldes at de indgående typer alle er en undertype af den type, der er angivet på venstresiden. Eksempelvis `længde = tid | afstand`, tid er en længde og afstand er en længde. Den ene måles i en tidsenhed og den anden i et længdemål, men de er begge en længde. Det generelle tilfælde er givet i Figur 4.6, og har følgende generelle type ligning:

```
D = ST | KT | DT | LT | TT
```



Figur 4.6: *E/R diagram af en disjunkt forening*

Pr. definition er et *isa* relationship *1-1*, men her tilføjes den ekstra begrænsning at en under-entitet ikke kan eksistere uden rod-entiteten. Desuden gælder der om under-entiteterne at de kopierer nøglen fra rod entiteten. I [GUW02] anvendes der ikke et pilehoved til at angive *1-1* forholdet mellem rod-entiteten og under-entiteten, dette anføres her eksplicit sammen med integritetsbegrænsningen, at rod-entiteten skal eksistere, altså et *p-1* relationship.

Hvis en af de indgående typer er en simpel type, oprettes der et entitetssæt som er relateret via dette *isa* relationship, i denne entitet er eneste attribut den simple type. Det nye entitetssæt navngives `D_ST`. Grunden til at der oprettes et entitetssæt selvom det blot er en simpel type, er for at bevare opremsningsstrukturen. Hvis attributten blot var blevet tilføjet rod-entiteten, ville alle under-entitetssættene indeholde denne attribut.

For en opremsningstype gælder der at de udgør en disjunkt opdeling, i almindelighed kan det ikke antages, betragt f.eks. eksemplet

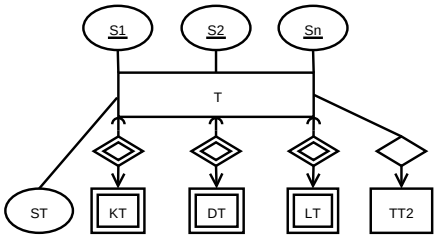
```
type danmarksRekord = bool
type verdensRekord = bool
disjoint rekord    = danmarksRekord | verdensRekord
```

Dette eksempel kan modellers som et isa relationship, men typerne der udgør dette isa hierarki er ikke disjunkte. I genereringen af E/R-diagrammet udnyttes kun implikationen fra disjunkt forening til isa relationship.

Tabel type

Den sidste sammensatte type er en tabel. Den generelle konstruktion af en tabel ser ud som på Figur 4.7. Typeligningen for det generelle tilfælde er:

```
T = (S1 * S2 * ... * Sn, ST * KT * DT * LT * TT)
```



Figur 4.7: *E/R diagram af Tabel konstruktionen*

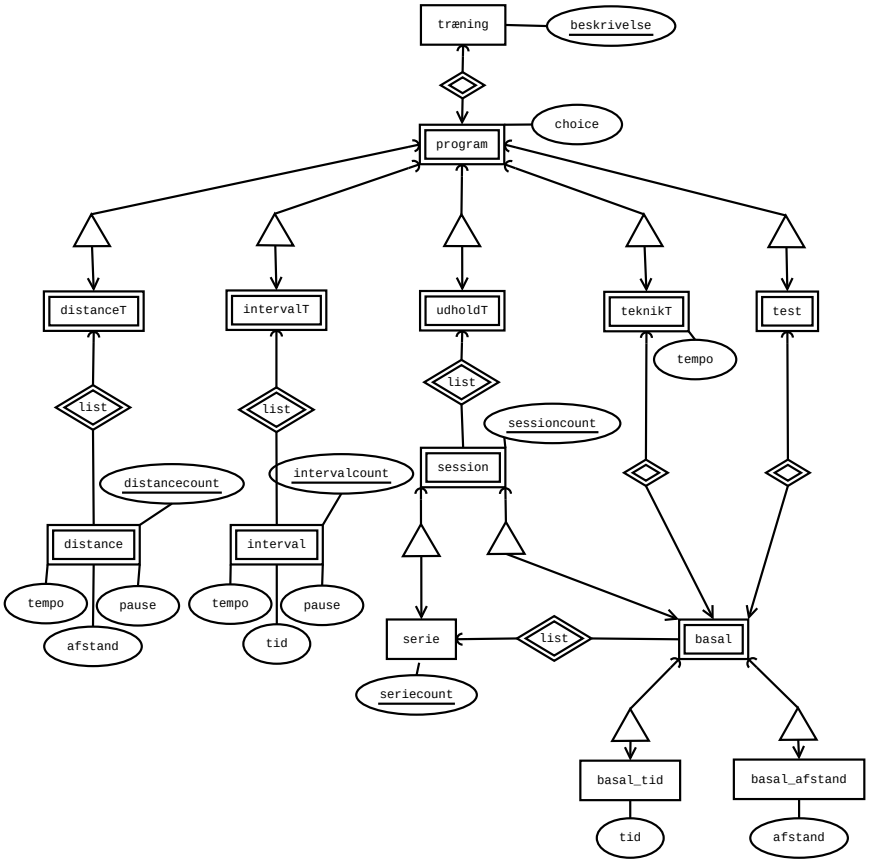
Værdierne {`s1`,`s2`,...,`sn`} er delnøgler i tabellen og tilsammen danner de en primær nøgle. Værdierne der er associeret til nøglen kan have sammen generelle struktur som typerne i et kartesisk produkt. Entiteten erklæres ikke svag da den indeholder en primær nøgle.

4.2 E/R diagram over Danmarks Rocenter

Anvendes den fremgangsmåde der er angivet ovenfor på den del af Danmarks Rocenter der omhandler træningsprogrammer, fås det E/R diagram der er givet i Figur 4.8 på næste side.

4.3 Generering af databaseskemaet

Oversættelsen fra E/R diagram til relationer og videre til et databaseskema vil blive beskrevet i dette afsnit. Oversættelsen til relationer følger de ret-



Figur 4.8: *E/R diagram over den del af Danmark Rocenter der vedrører træningsprogrammer.*

ningslinier, der er givet [GUW02] og omhandler hvorledes nøgler skal indsamles via svage relationships og kopieres ved isa relationships. Sidste del i processen, omhandler oversættelsen fra relationer til databaseskema.

4.3.1 Relationerne

En relation indeholder en eksakt beskrivelse af den information der skal implementeres i de tabeller der oprettes i databasen. Dette dækker: *tabelnavn*, *nøgler*, *attributter*, *fremmednøgler* samt *typen* på de indgående nøgler og attributter.

E/R diagrammet består af entitetssæt og relationships, der desuden begge optræde som svage. Ved oversættelsen giver følgende tre situationer anledning til relationer.

- Entitetssæt
- Relationships
- Svage entitetssæt med tilhørende svage relationships.

Et svagt relationship bliver ikke til en relation, informationen anvendes i stedet sammen med svage entitetssæt.

Entitetssæt

Et entitetssæt kan direkte omskrives til en relation med de attributter og nøgler, der er angivet ud for denne. Der skal ikke indhentes fremmednøgler i dette tilfælde.

Relationships

Et relationship omskrives til en relation ved at finde nøglerne i de to entitetssæt, som den forbinder. Den resulterende relation navngives ved at sammenkæde typenavnene fra de to entitetssæt(`TN1` og `TN2`) til `TN1-REL--TN2`. Der er ingen yderlige attributter i denne relation.

Svage entitetssæt

Et svagt entitetssæt omskrives til en relation ved at finde de nøgler der hører til gennem de støttende svage relationships, denne proces er beskrevet nedenfor. I denne sammenhæng behandles et isa relationship som et svagt relationship.

Nøgle indsamling til svage entiteter

Når der skal indhentes nøgler til svage entiteter, er det tilstrækkeligt kun at indsamle nøgler fra de entitetssæt, der støtter entitetssættet direkte. Nøgler skal altså ikke indsamles fra hele den svage kæde, hvis en sådan er til stede. Dette skyldes at støttende entitetssæt kan antages allerede at være behandlet og indeholdende en primær nøgle (og derfor ikke længere skal betragtes som svage).

Grunden til at alle støttende entitetssæt har en primær nøgle er, at de allerede er behandlet. Denne egenskab kan opnås ved at starte med at tildele fremmednøgler til de sammensatte typer, der er erklæret til sidst og derfor ikke indgår i nogen anden type. Da der ikke tillades cykliske erklæringer, vil den sidst definerede type ikke indgå i nogen anden type og har derfor ikke nogen fremmednøgler.

Den næstsidst erklærende type vil ikke kunne indgå i nogle typer der er defineret tidligere, derfor ingen referencer. Måske indgår den i den sidst definerede type, men denne er ikke svag og indeholder derfor en nøgle.

Denne proces kan fortsættes for alle de sammensatte typer i signaturen, resultatet af dette er at fremmednøgler kan indsamles, ved kun at indsamle fra 'nabo' entitetssæt sålænge nøgleindsamlingen foregår i den beskrevne rækkefølge.

Bruges denne fremgangsmåde på eksemplet i Figur 4.3 på side 43 fås følgende nøgle tildeling.

```
type C = c * A      (ingen fremmednøgler)
type B = b * A      (ingen fremmednøgler)
type A = a          (fremmednøgler fra B og C)
```

Da `A` indgår i to sammensatte typer indeholder den fremmednøgler til disse to.

Behandling af *isa* hierarki

Ved oversættelse af isa relationships til relationer er der tre muligheder: E/R-metoden, OO-metoden eller NULL-metoden. Her beskrives kort de tre metoder samt valget af E/R-metoden.

E/R Her bliver der dannet en relation for rod entitetssættet og en relation for hvert af underentitetssættene. Dette giver en total på n+1 relationer, hvis der er n underentitetssæt i *isa* hierarkiet.

OO Her dannes der en relation for hver delmængde der kan dannes af de indgående entitetssæt. Dvs. $2^{n+1} - 1$ relationer, i dette tilfælde er dog mange af disse relationer, der aldrig vil blive befolket eftersom under-entiteterne er disjunkte. Dette reducere OO-metoden til kun at indeholde 2n+1 entitetssæt.

NULL Attributterne fra under-entitetssættene flyttes op i rod-entitetssættet og under-entitetssættet slettes. Dette resulterer i blot en enkelt relation. Ulempen ved denne løsning er, at der bliver mange attributter der er tomme, når der oprettes en entitet.

Til illustration af de tre metoder udføres de for den disjunkte forening `roertype` der er erklæret i Afsnit C.1 på side 90. De typer der indgår i `roer` og `sculler` er alle simple.

```
type roer         = side * smig * hoejde * gearing
type sculler      = ssmig * bsmig * hoejde * hforskel
type begge        = roer * sculler
disjoint roertype = roer | sculler | begge
```

De fire tabeller der fremkommer ved E/R metoden er vist i Tabel 4.3.

roertype
id

roer
side
smig
hoejde
gearing

sculler
ssmig
bsmig
hoejde
hforskel

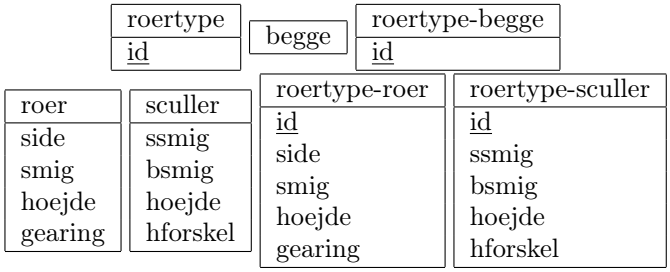
begge

Tabel 4.3: *Tabeller der fremkommer ved isa hierarki ved brug af E/R metoden*

Ved OO-metoden fremkommer de syv tabeller der ses i Tabel 4.4 på næste side.

De tre ekstra relationer der kommer ved OO-metoden, er præcis magen til dem der fås ved E/R-metoden, efter nøglerne i rod-entiteten er blevet kopieret til under-entiteterne. De to fremgangsmåder giver altså samme løsning.

Ved NULL-metoden bliver attributterne samlet i en tabel, se Tabel 4.5 på næste side.



Tabel 4.4: *Tabeller der fremkommer ved isa hierarki ved brug af OO-metoden*

roertype
id
roer_side
roer_smig
roer_hoejde
roer_gearing
sculler_ssmig
sculler_bsmig
sculler_hoejde
sculler_hforskel

Tabel 4.5: *Tabeller der fremkommer ved isa hierarki ved brug af NULL-metoden*

Det der adskiller de to fremgangsmåder mest er, at ved E/R-metoden skal der oprettes entiteter i to tabeller hver gang, og ved NULL-metoden bliver mindre end halvdelen af attributterne anvendt. Ud fra et effektivitetssynspunkt bør NULL metoden vælges da det i følge [GUW02] betragtes som mere resourcekrævende at lave et dobbelt opslag frem for at anvende den dobbelte lagerplads. Hvis en disjunkt forening bestod af mange typer ville NULL-metoden dog stå svagere.

I Afsnit 4.4.1 på side 52 argumenteres der for at uheldige omstændigheder ved at implementere isa relationship ved brug af NULL-metoden, kan føre til at databasen bryder 3. normalform, derfor vælges E/R-metoden.

E/R-metoden har den fordel, at den modellerer data, præcis som det er erklæret i signaturen. Ved at vælge E/R metoden kommer implementationen til at ligne måden som svage relationships bliver implementeret på.

4.4 Normalisering

I det følgende analyseres den opbyggede database, med henblik på at afgøre om den er fornuftigt designet mht. redundans.

Der kan ikke foretages en analyse af, om databasen overholder normalformerne på vanlig vis, da data ikke er kendte. I stedet analyseres om behandlingen af de sammensatte datatyper kan føre til, at nogle af normalformerne bliver brudt.

4.4.1 Normalformerne

1. normalform kræver atomisk data i attributterne i databasen. Værdier der skal gemmes i databasen er fremkommet ved, at værdier til de angivne primitive typer er indtastet i felterne. Der kan ikke testes for, om data der indtastes i et tekstfelt er atomisk eller ej. Et eksempel på en atomisk type kunne være "Christian d. 10. Allé"og et eksempel på en sammensat kunne være "Hovedgade 1. th.". Nå de simple typer bliver transformeret til at være attributter sker der ingen sammensætning af typerne og hvis de simple typer er atomiske ved indtastningen, er de det stadig, når de repræsenteres i databasen.

For at overholde 2. normalform skal der om tabellerne i databasen gælde, at en attribut ikke kan bestemmes ud fra en del af nøglen. For det kartesiske produkt er dette opfyldt, da et kartesisk produkt anvendes til typer, der hører naturligt sammen, men som ikke har nogen afhængighed. Problemet mht. 2. normalform er ved tabeltypen. Hvis der f.eks. er erklæret en tabel ved `tabel person = (navn * cprnr , køn * adresse)`, hvor de indgående typer alle er simple, så er det muligt at bestemme `køn` ud fra `cprnr`. En sådan afhængighed er det ikke mulig at fange automatisk, da det kræver yderligere kendskab til hvordan et cpr. nummer er konstrueret end blot at vide, at det er et heltal.

Når `tabel` implementeres i databasen sker det som en direkte afbildning af hvordan den var angivet i signaturen. En tabel der overholder en hvilken som helst normalform i signaturen, vil bevare denne egenskab i databasen.

3. normalform

Når en disjunkt type skal oversættes til relationer sker dette som efter E/R metoden, og data lægges altså ikke sammen i en relation. Havde NULL-metoden været valgt, kunne man være i den situation, at to underentiteter som får flyttet attributterne op i rod entiteten, introducerer en afhængighed mellem to af attributterne uden, at de i almindelighed har noget med hinanden at gøre. F.eks hvis en underentitet indeholdt `postnummer` og en anden underentitet indeholdt `by`. Her ville den resulterende relation både indeholde postnummer og by som attributter og der vil dermed være etableret en afhængighed mellem to attributter, og det tillades ikke i følge 3. normalform.

4.4.2 BC. normalform

De forskellige typer som tillades i denne modellering er gennemgået og det har vist sig, at de bliver implementeret i databasen, præcis som de blev erklæret i signaturen. Hvis typerne overholder Boyce-Codd normalform i signaturen, så vil de også gøre det i databasen.

Brugeren har ansvaret for at konstruere normaliserede typeerklæringer. Det eneste der kan garanteres ved genereringen af databasen er, at der ikke introduceres yderligere afhængigheder.

Kapitel 5

Design af grænsefladen

Der er brug for en grænseflade til databasen. I det følgende gives et forslag til hvordan den kan designes.

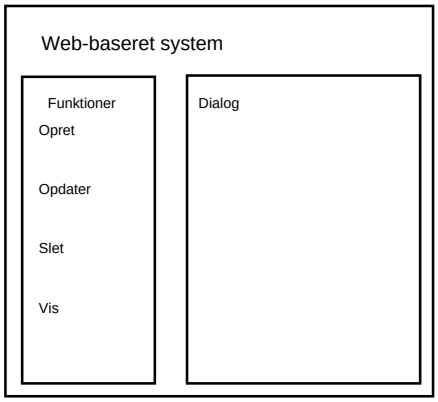
Ud fra de typekonstruktorer der er tilladte i signaturen, kan der opbygges en grænseflade der indeholder basale muligheder for at modificere indholdet af databasen. I Afsnit 5.2 på side 56 vil der blive introduceret en generisk opbygning af siden, som indeholder fire basale funktioner: *indsæt*, *vis*, *slet* og *opdater*. Funktionerne er baseret på typekonstruktorer og anhænger derfor ikke af signaturen.

Ud fra de typer der blev angivet i signaturen for det web-baserede system, kan det fastlægges, hvilket typer der skal kunne manipuleres med de basale funktioner, dette er grundlaget for, hvilke funktioner, der skal genereres.

5.1 Anvendelses specifik del

Den grafiske brugergrænseflade er designet til at give adgang til alle basale funktioner. Dette er opnået ved at have en menu i venstre side og en dialog i højre side. Figur 5.1 på næste side viser hvorledes siden grafisk er opbygget.

Under hver af de fire funktionstyper skal der genereres funktioner, der er specifikke for anvendelsen.



Figur 5.1: *Grænsefladens grafiske opbygning*

Hvis differentieret brugeradgang skulle implementeres, ville det på dette sted bl.a. medføre, at kun de funktioner som brugere har rettigheder til ville være synlige i menuen.

Funktionerne i menuen er genereret ud fra de typer der er erklæret ud fra `tabel` konstruktoren i signaturen. Dialogen er afhængig af de typer, der indgår i tabeltypen.

5.1.1 Tabeltypen

Ud fra betragtningen om at tabeller er den eneste af typerne, der indeholder sin egen nøgle, er det også kun tabellerne, det tillades at manipulere ved. På samme måde som det ikke var nogen begrænsning kun at tillade tabeltypen på øverste niveau i hierarkiet, er det heller ikke nogen begrænsning, at der kun kan ændres ved tabeltypen.

Det faktum at der kan ændres i en tabel medfører, at der kan ændres i de sammensattes typer der indgår i tabellen. I betragtning af at alle typer der indgår i det web-baserede system er repræsenteret i en tabel, muligvis gennem flere sammensatte typer, kan alle typer opdateres, når blot en tabel kan opdateres.

Med et konkret eksempel fra anvendelsen ses det, at det er en dårlig idé, hvis der kan ændres i andre typer end tabeltyperne. Til eksemplet anven-

des intervaltræning `intervalT`, der er en liste af det kartesiske produkt bestående af: `tid`, `pause` og `tempo`, der alle er simple typer

```
type interval    = tid * pause * tempo
type intervalT   = interval list
```

Et eksempel på to intervaltræningsprogrammer med hver tre intervaller kan ses i Tabel 5.1.

intervalCount	intervalT_traening_beskrivelse	tempo	pause	tid
1	3*10/3 min 26/28/30	26	3	10
2	3*10/3 min 26/28/30	28	3	10
3	3*10/3 min 26/28/30	30	3	10
1	15+10+5 min 26/28/30	26	3	15
2	15+10+5 min 26/28/30	28	3	10
3	15+10+5 min 26/28/30	30	3	5

Tabel 5.1: *tabellen interval, indeholdende data for to træningsprogrammer*

I tabellen bemærkes det, at der er relativt mange poster, hvilket bevirker at overblikket lettere mistes. For at lokalisere det konkrete interval skal fremmednøglen benyttes (den stammer fra tabellen `traening`). Som det ses i tabellen kan et bestemt interval ikke findes ved at kigge på attributterne i det kartesiske produkt alene (her ville linie 2 og 5 være ens). For at lokalisere et `interval` skal nøglen fra tabellen anvendes, fremgangsmåden for at finde en post direkte i `interval` er altså den samme, som hvis nøglen blev bestemt i `traening`.

Hvordan samspillet mellem to tabeller bliver ses i følgende eksempel der stammer fra anvendelsen.

```
table baadtype       = (baad, baadnavn * stoerelse * olympisk)
type resultat        = tidspunkt * baadtype * begivenhed *
                       sted * placering * tid
type resultater      = resultat list
table personKartotek =(navn* foedselsdato, personData * resultater)
```

Her optræder to tabeller, en liste og et kartesisk produkt. Funktioner til at opdatere i det to tabeller vil blive genereret automatisk, men det betyder ikke, at der ikke kan ændres resultater. Ønskes der en ændring i `resultat`, skal denne foretages igennem tabellen `personKartotek`. En ændring i `baadtype` kan derimod ikke etableres igennem personKartoteket,

selvom `baadtype` indgår i det kartesiske produkt. Ændringen der kan foretages i `personKartotek` er hvilken bådtype, der skal refereres til. Ønskes opdatering i beskrivelsen af båden er det i tabellen `baadtype`, der skal ændres. En opdatering i `baadtype` vil medføre, at alle der refererer til denne bliver opdateret.

Det er altså tilstrækkeligt at implementere funktioner, der virker direkte på tabeller. De fire basale funktioner vil nu blive gennemgået.

5.1.2 Menugenereringen

De typer der er erklæret med tabeltypen i signaturen, skal bruges til at danne funktioner i menuen. For typen tabel i signaturen skal de fire basale databasefunktioner genereres, der er omtalt i ovenstående. Den konkrete funktion der bliver oprettet på denne måde, er altså en kombination af typerne, der indgår i tabellen og så de generiske funktioner der er beskrevet i Afsnit 5.2.

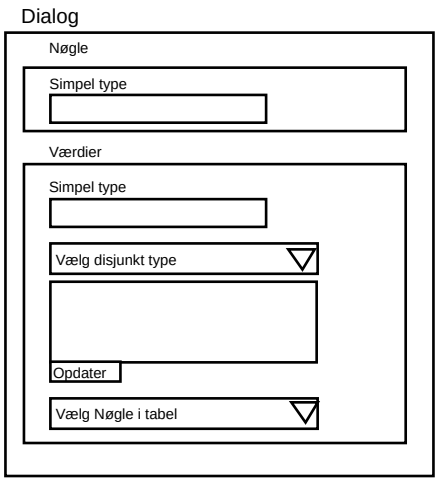
5.2 Generisk del

Den generiske opbygning af brugerdialogen samt database funktionerne udnytter den hierarkiske struktur i typesystemet.

5.2.1 Indsæt

Funktionen indsætter et element i den tabel, som den virker på. Argumenterne til funktionen er foruden tabelnavnet en nøgle og værdien af denne. Som nøgle kan anvendes et kartesisk produkt bestående af simple typer. Værdien fra tabellen er et generelt kartesisk produkt, som det blev defineret i Afsnit 4.1.3 på side 45.

Kravet til de simple typer er, at de skal danne en nøgle (unik), ellers kan posten ikke registreres i tabellen. Simple typer indtastes i systemet via et tekstfelt. En disjunkt forening implementeres med en dropdown-menu, hvor der kun kan vælges i mellem de forskellige typer der indgår i den disjunkte forening. Når der er valgt en type genereres indtastningsfelterne for denne type. Indsættelse af et kartesisk produkt sker ved, at der dannes



Figur 5.2: *Det generelle udseende af en indsættelses dialog*

indtastningsfelter for alle de indgående typer. For en liste laves der indtastningsfelter til de typer der indgår i liste elementet, desuden skal der være mulighed for at generere flere listeelementer i dialogen. Er en af de indgående typer selv en tabel, så skal der vises en oversigt over disse, så der kan vælges en reference i denne.

En skematisk oversigt over indsættelsesdialogen kan ses i Figur 5.2.

5.2.2 Vis

En oversigt over indholdet i en tabel genereres af denne funktion. Som argument til funktionen er kun tabelnavnet. Funktionen lister alle poster i den givne tabel. Visningen foregår som en tabel med to søjler, i den første er nøglen til tabellen og i den anden er værdien til nøglen, se Tabel 5.2 på næste side.

I tabellen symboliserer $\texttt{nøgle}_i$ et kartesisk produkt af simple typer som det blev defineret i Afsnit 4.7 på side 45. På værdisiden symbolisere $\texttt{værdi}_i$ et kartesiske produkt hvor de indgående typer er vilkårlige.

Visning af de øvrige sammensatte typer samt simple typer forgår efter følgende principper.

tabelnavn	
nøgle$_1$	værdi$_1$
nøgle$_2$	værdi$_2$
⋮	⋮

Tabel 5.2: *En tabel vises med nøglerne i venstre søjle og tabellens værdier for den givne nøgle i højre søjle*

Kartesisk produkt

Et kartesisk produkt vises ved at typenavnene på de indgående typer kommer til at stå som overskrift til søjlerne og i anden rækker kommer værdierne. En post i tabellen `test` fra anvendelsen, kan se ud på følgende måde: For de simple typer vises værdien og for de sammensatte typer sker

basal	sted	formaal	kommentar	beskrivelse
vis(basal)	drc	iltoptag.	optimering...	ingen

Tabel 5.3: *Et tabel vises med nøglerne i venstre søjle og værdierne i højre søjle*

visningen ved at kalde den generelle funktion `vis`.

Disjunkt forening

For den disjunkte type skal der ikke ske nogen visning. I dette skridt skal den valgte undertype findes, og den passende visningsfunktion for denne skal håndtere visningen.

Listkonstruktoren

I en liste er alle elementerne af samme type, derfor laves der kun en søjleoverskrift for visningen af elementerne i listen. I eksemplet med intervaltraening kommer det til at se ud som i Tabel 5.4 på modstående side.

tempo	pause	tid
26	3	10
28	3	10
30	3	10

Tabel 5.4: *Visning af listkonstruktoren*

traening			
3*10/3 min 26/28/30	tempo	pause	tid
	26	3	10
	28	3	10
	30	3	10
15+10+5 min 26/28/30	tempo	pause	tid
	26	3	15
	28	3	10
	30	3	5
⋮	⋮		

Tabel 5.5: *Oversigt over træning ved brug af den generelle visningsfunktion*

Generel visning

Anvendes den generelle visningsfunktion på `træning`, fås oversigten på , se Tabel 5.5, hvor det ses at visningen af de forskellige typer er nestede efter samme mønster som sammensætningen af tabeltypen.

5.2.3 Slet

Alle rækker i tabellen listes og for hver enkelt af dem skal der markeres en mulighed for at slette dem. Ved at undersøge de relationships som refererer til tabellen, kan man undersøge om den række der ønskes slettet indgår i nogen sammenhæng. Gør den det, skal det ikke være muligt at slette posten. Hvis en post derimod ikke refereres fra nogen anden type i databasen kan den slettes uden, at der opstår null referencer i databasen.

Integritetsbegrænsninger der blev sat op ved designet af databasen, sørger for at slette poster i alle de tabeller som tabeltypen anvender. Hvis typen selv indeholder en tabel skal indgangen ikke slettes i denne. Dette er opfyldt ved, at der ikke er nogen integritetsbegrænsningen imellem dem. I

anvendelsen haves følgende situation. Antag at en person `a` er tilknyttet bådtype `b`, så skal bådtypen `b` ikke slettes, hvis personen `a` bliver slettet i personkartoteket.

5.2.4 Opdater

Alle rækkerne i tabellen skal listes og ved hvert element skal der være et link der fører til en opdatering af den valgte række. Selve opdateringen skal foregå ved, at den valgte række i tabellen hentes ud fra databasen. Selve opdateringen kommer til at foregå ved at en indtastningsformular, svarende til den der fandtes for oprettelse, udfyldt med de værdier der er hentet ud af databasen.

Alle de ovenstående funktioner er generiske, og afhænger altså ikke specifikt af udseendet af typerne. I næste afsnit beskrives hvorledes de generiske funktioner er designet.

Kapitel 6

Implementation

De to værktøjer der er implementeret er uafhængige og de er derfor beskrevet i hvert deres afsnit.

Det første værktøj oversætter en signatur til et databaseskema, en repræsentation af typerne i PHP og definerer brugerfunktionerne med de funktioner der skal være tilgængelige på siden. Alle disse transformationer er foretaget i SML da det er velegnet til at fortolke en signatur, ved brug af værktøjerne lexx/yacc (SML skanner/parser).

Anden del bruges til at administrere de input der gives til systemet. Dvs. generere brugergrænsefladen, validere input, kommunikere med databasen. Funktionerne i denne del er generiske og dermed uafhængige af hvilke typer, der er defineret i systemet. De handler ud fra hvilke typekonstruktorer, der er brugt.

6.1 Database generering

I dette afsnit beskrives implementationen af de tre trin der udføres for at transformere et typesystem til et relationsdatabaseskema, som det er beskrevet i Kapitel 4 på side 37.

Sidst i afsnittet er det beskrevet, hvorledes typerne fra signaturen er repræsen<-teret i PHP.

```
signature implementationsEksempel = sig
  type a  = string
  type b  = string
  type c  = int
  type d  = int
  type K  = c * d
  type L  = K list
  table T = (b, a * K * L)
end
```

Tabel 6.1: *Signatur for lille eksempel system, systemet vil tjene som gennemgående eksempel i implementationskapitlet.*

Detaljer der har betydning for implementationen, er omtalt på de steder, de opstår. Både parseren og fortolkeren er implementeret i ML. Til at beskrive funktionerne anvendes MLs typesystem.

Til at støtte beskrivelsen af implementationen anvendes der et gennemgående eksempel i resten af kapitlet, signaturen til dette system er givet i Tabel 6.1.

6.1.1 Signatur → decl list

Første trin i transformationen består af en parsing af signaturen.

Til skanningen og parsingen bruges værktøjerne Lexx og Yacc. Keywords og grammatik til de to værktøjer er angivet i Afsnit 3.7 på side 35 og kildekoden findes i Appendiks D.1 på side 95.

Til at skanne og parse en fil indeholdende en signatur bruges følgende funktion:

```
parsef  : string -> decl list
```

Argumentet til `parsef` er en streng med navnet på filen, der indeholder signaturen, der skal parses som en tekststreng. Værdien af funktionen er en liste af typeerklæringer `decl list`. `decl`, der blev udviklet i Afsnit 3.7 på side 35, er i ML erklæret ved datatypen:

```
datatype decl = Typedecl of string * typeexpr
              | Tabledecl of string * typeexpr
              | Disjdecl of string * typeexpr
```

```
val declList =
   [Typedecl("a", Ptype "string"),
    Typedecl("b", Ptype "string"),
    Typedecl("c", Ptype "int"),
    Typedecl("d", Ptype "int"),
    Typedecl("K", Ctype(typeName "c", typeName "d")),
    Typedecl("L", Ltype(typeName "K")),
    Tabledecl("T",
              Ttype(typeName "b",
                    Ctype(Ctype(typeName "a",
                        typeName "K"), typeName "L")))]
    : decl list
```

Tabel 6.2: *Listen af typeerklæringer som de ser ud, efter signaturen for eksemplet er blevet skannet og parset.*

En erklæring er altså et navn kombineret med en af følgende: En typeerklæring (der kan være en erklæring af simple typer, lister og kartesisk produkt), en tabelerklæring eller en erklæring af en disjunkt forening.

I alle tilfælde indgår typeudtryk `typeExpr`, der er erklæret ved ML datatypen:

```
datatype typeexpr = typeName of string
                  | Ptype of string
                  | Ltype of typeexpr
                  | Ctype of typeexpr * typeexpr
                  | Ttype of typeexpr * typeexpr
                  | Dtype of typeexpr * typeexpr
```

Skanning og parsing af signaturen for eksemplet fører til den liste af typeerklæringer, der er givet i Tabel 6.2.

6.1.2 ParseTree list → E/R diagram

Andet trin i transformationen genererer et E/R diagram ud fra den erklæringslisten, kildekoden findes i Appendiks D.1.3 på side 106.

Ved forklaringen af funktioner benyttes forkortelserne i Tabel 6.3 på næste side til at betegne de forskellige typer der indgår. Bemærk at forkortelserne ikke er erklæret i ML, de anvendes alene som forklarende notation.

Forkortelse	SML repræsentation
`typeNavn`	`string`
`type`	En af følgende `"int"`, `"string"`, `"real"`, `"bool"` `"list"`, `"cart"`, `"disj"`, `"table"`
`mark`	En af følgende `"weak"`, `"table"`, `"auto"`,
`info`	En af følgende `"1-p"`, `"n-p"`, `"isa"`, `"1-p"`,
`att`	`typeNavn * type`
`key`	`att`
`ent`	`typeNavn * key list * att list * mark`
`relship`	`ent * ent * info * mark`
`dl`	`decl list`
`el`	`ent list`
`al`	`att list`
`rl`	`relship list`

Tabel 6.3: *Forkortelser af typerne der bruges ved transformeringen*

Selve transformationen er implementeret som fire funktioner og disse er beskrevet ved følgende:

```
getTypeList : dl * al -> al
```

Værdien af `getTypeList(dl,[])` er en liste bestående af `att`. Funktionen er rekursiv og den akkumulerende parameter initialiseres med den tomme liste. Under genereringsprocessen kontrolleres det, at de typer der indgår i sammensatte typer er erklæret ved at sammenligne med den akkumulerende liste.

```
getEntities : dl * el * al -> el
```

Værdien af `getEntities(dl,[],al)` er en `ent list`. Funktionen er akkumulerende, derfor initialiseres den med en tom `ent` liste.

Funktionen genererer en entitet for hver typeerklæring, der ikke er simpel. Attributterne til en entitet er de simple typer der indgår i typeerklæringen. De sammensatte typer medtages ikke som attributter, da de selv er entiteter. Ved erklæringer med brug af list og disjunkt forening kontrolleres der desuden for om de indgående typer er simple. Hvis de er simple bliver der oprettet entiteter til at indeholde disse efter den fremgangsmåde, som er beskrevet i Afsnit 4.1.3 på side 44.

```
getRelationships :  dl * rl * el * al -> rl
```

På baggrund af typeerklæringslisten identificeres relationships mellem de forskellige sammensatte typer, dvs. sammensatte typer der selv indeholder sammensatte typer. `getRelationships(dl,[],el,al)` er en rekursiv funktion der initialiseres med en tom liste af relationships. En deltype i et relationship er `mark`. Denne bruges til at markere hvilken slags relationship der oprettes, `info` kan indikere, at et relationship er svagt.

```
updateEntities : rl * el -> el
```

Entiteterne gennemløbes endnu en gang, men nu udnyttes markeringerne fra listen af relationships til at markere hvilke entiteter der er svage, hvilke der skal generere egen nøgle og hvilke der selv indeholder en nøgle. Ud fra definition af hvornår der skal anvendes en `tabel` antages det, at disse indeholder egne nøgler.

E/R diagrammet

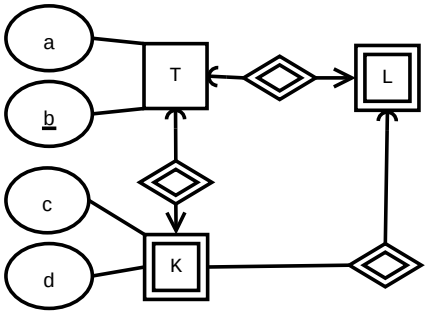
Resultatet af transformationen er to lister indeholdende entiteter hhv. relationships. Disse to lister udgør tilsammen E/R-diagrammet over databasen.

Resultatet af denne oversættelse ses for eksempelsystemet i Tabel 6.4. Figur 6.1 på næste side viser det samme E/R-diagram grafisk.

```
> rl =
   [("L", "K", "p-n", "weak"),
    ("T", "L", "p-1", "weak"),
    ("T", "K", "p-1", "weak")] :
    (string * string * string * string) list

> el =
   [("K", [("Kcount", "int")],
          [("d", "int"), ("c", "int")], "weak"),
    ("L", [], [], "weak"),
    ("T", [("b", "string")], [("a", "string")], "table")] :
    (string * (string * string) list *
              (string * string) list * string) list
```

Tabel 6.4: *E/R diagrammet som det ser ud efter signaturen for eksemplet er transformeret*



Figur 6.1: *E/R diagrammet som det ser ud efter signaturen for eksemplet er transformeret*

6.1.3 E/R → Database skema

Det tredje og sidste trin i transformationen over i databaseskemaet, kildekoden findes i Appendiks D.1.5 på side 121.

I dette trin indsamles fremmednøgler og der dannes relationer. Notationen der bruges til at repræsentere dette ses i Tabel 6.5

Forkortelse	SML repræsentation
fkeys	typeNavn * key list
relation	typeNavn * att list * key list * fkeys list

Tabel 6.5: *Forkortelser af typerne der bruges ved transformeringen*

Det første der sker i oversættelsen er, at entiteter og relationships bliver opdelt efter hvilken markering de har. Til dette bruges to funktioner

```
isISA : 'a * 'b * 'c * string -> bool
isWeak : 'a * 'b * 'c * string -> bool
```

Begge funktioner virker ved, at de sammenligner med, hvad der står som markering, det fjerde argument. `isISA` bruges kun på relationships, mens `isWeak` både kan bruges til at opdele relationships og entiteter.

I kombination med den indbyggede funktion `List.partition` fås en opdeling bestående af 'isa', 'svage' og 'normale' relationships samt 'svage' og 'normale' entiteter. Opdelingen bruges i `getKeyNames`.

```
getKeyNames : el * rl  * el -> el
```

Funktionen er rekursivt defineret `getKeyNames(el,[],el)` og værdien er en liste af entiteter, hvor alle entiteter er opdateret med alle fremmednøgler.

Funktionens første argument er den reverserede liste af de entiteter, der skal opdateres, dvs. de svage entiteter. Andet argument til funktionen er svage relationships og isa relationships. Disse har information om nøgleindsamling til de svage entiteter. Sidste argument er en liste der indeholder de opdaterede entiteter, denne liste indeholder alle entiteterne.

For hver svag entitet der skal opdateres skal samtlige svage relationships og isa relationships løbes igennem. Hvis entiteten indgår i et svagt relationships, indsamles der nøgler fra den entitet, som der refereres fra. Dette gøres ved at sammenkæde navnet på nøglen med tabelnavnet fra den entitet der indhentes fra. Er det et isa relationship kopieres nøglen i stedet.

Et eksempel. `A` er en svag entitet og der eksisterer et svagt relationship til `B`, der har nøglen `b`. Efter nøgleindsamlingen har `A` nøglen `B_b`.

```
wEnt2rel : el * rl * el -> relation list
```

Svage entiteter bliver transformeret til relationer ved at udnytte den information der ligger i svage relationships og isa relationships. Foruden oplysninger med nøgler, fremmednøgler og attributter bliver der for hver relation også oprettet en liste med de entiteter, som fremmednøglerne stammer fra.

```
ent2rel : el -> relation list
```

Denne funktion oversætter normale entiteter til relationer, entiteterne indeholder alt information og der ikke er nogen fremmednøgler.

```
rel2rel : rl * el -> relation list
```

Funktionen oversætter normale relationships til relationer. Nøglerne til relationerne findes i entitetslisten, der også er argument til funktionen. Nøglerne i relationen er de nøgler, der findes i entitetssættene som forbindes af relationshippet.

For det gennemgående eksempel er relationerne listet i Tabel 6.6 på næste side.

```
cSchema : relation list -> string
```

```
relations =
  [("K", [("Kcount", "int"), ("L_T_b", "string"),
          ("T_b", "string")],
         [("d", "int"), ("c", "int")],
         [("L", [("T_b", "string")]),
          ("T", [("b", "string")])]),
   ("L", [("T_b", "string")],
         [],
         [("T", [("b", "string")])]),
   ("T", [("b", "string")],
         [("a", "string")],
         [])] :
 (string * (string * string) list *
     (string * string) list *
          (string * (string * string) list) list) list
```

Tabel 6.6: *Relationerne som de ser ud lige inden databaseskemaet genereres*

Funktionen oversætter listen af relationer til SQL syntaks. Denne indeholder opsætning af integritetsbegrænsninger. Til at implementere integritetsbegrænsningerne bruges oplysningerne fra listen af fremmednøgler.

I forbindelse med oprettelse af integritetsbegrænsninger opstår der et problem når der skal oprettes poster i de tabeller, der bliver etableret i databasen. Problemet består i at en post ikke kan oprettes uden at alle fremmednøgler er oprettet. Dette løses ved at initialisere alle tabeller med en NULL post. Gøres dette kan der fra alle tabeller refereres til disse NULL poster når en fremmednøgle ikke er defineret. I Figur 6.7 på modstående side ses databaseskemaet for det gennemgående eksempel.

På Figur 4.3 på side 43 ses netop denne situation. Når der skal oprettes en post i `B` medfører det en post i `A`, fordi `A` er en sammensat type. Når der skal oprettes en post i `A`, er den nødt til at referere til NULL posten i `C` for at kunne blive oprettet.

I anvendelsen opstår problemet når der skal oprettes en type der indeholder `basal`, i det `basal` indgår i fire forskellige sammensatte typer.

```
CREATE TABLE ‘T‘ (
  ‘b‘ VARCHAR(30) ,
  ‘a‘ VARCHAR(30) ,
  PRIMARY KEY (b)
) TYPE = InnoDB; INSERT INTO ‘T‘ (  b,  a) VALUES (’’,’’);

CREATE TABLE ‘L‘ (
  ‘T_b‘ VARCHAR(30) ,
  INDEX (T_b ),
  FOREIGN KEY (T_b )
    REFERENCES T (b )
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  PRIMARY KEY (T_b)
) TYPE = InnoDB; INSERT INTO ‘L‘ (  T_b) VALUES (’’);

CREATE TABLE ‘K‘ (
  ‘Kcount‘ int ,  ‘L_T_b‘ VARCHAR(30) ,
  ‘T_b‘ VARCHAR(30) ,  ‘d‘ int ,  ‘c‘ int ,
  INDEX (L_T_b ),
  FOREIGN KEY (L_T_b )
    REFERENCES L (T_b )
      ON DELETE CASCADE
      ON UPDATE CASCADE,

  INDEX (T_b ),
  FOREIGN KEY (T_b )
    REFERENCES T (b )
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  PRIMARY KEY (Kcount,
  L_T_b,
  T_b)
) TYPE = InnoDB; INSERT INTO ‘K‘
          (  Kcount,  L_T_b,  T_b,  d,  c)
VALUES (’’,’’,’’,’’,’’);
```

Tabel 6.7: *Relationerne som de ser ud lige inden databaseskemaet genereres*

6.2 Grænsefladen generering

I [JeT03] blev implementationen gennemført i MSP, der er en scriptudvidelse til ML [MML]. I betragtning af at det ofte ikke tillades at installere software på en server i forbindelse med hosting, vil denne implementation i stedet anvende PHP [PHP] som scriptsprog. Forskellen fra MSP til PHP er at PHP ikke har et stærkt type system. Ved implementationen, skal både typer og værdier derfor modelleres.

I Afsnit 6.2.3 på side 74, den generiske del, forklares hvordan typer der er fremkommet ved brug af det typesystem, der blev udviklet i Afsnit 3.7 på side 35 kan modelleres i PHP. Herefter forklares det hvordan de fire basale funktioner er implementeret.

I Afsnit 6.2.2 på modstående side beskrives de to funktioner der genererer grænserfladen ud fra signaturen.

6.2.1 Struktur på PHP

PHP tilbyder ikke et stærk typesystem som ML gør. I PHP eksisterer der fire primitive typer, der alle kan behandles som strenge. Denne implementation benytter strenge til at modellere typerne.

Typetræet

Simple typer og sammensatte typer modelleres i PHP ved hjælp af PHP typekonstruktoren `array`. En simpel type bliver repræsenteret vha. to strengelementer i arrayet, `typenavn` og `type`, navnet på PHP-variablen er også `typenavn`. Strukturen for en simpel type i PHP ses i Tabel 6.8 på modstående side, hvor `prim` er en af følgende strenge `"int", "string","real", "bool"`.

Til at repræsentere de forskellige sammensatte typer benyttes igen PHPs typekonstruktor `array`. Denne gang med tre elementer. Første element er typenavnet, andet element er typen og tredje element typeudtrykket. De fire sammensatte typer bliver oversat på følgende måde.

Bemærk de de to ekstra kartesiske produkter der oprettes til nøgler og attributer ved tabeltypen, kun er en intern repræsentation.

Simple typer	
SML rep.	PHP rep.
`type a = prim`	`$a = array("a","prim");`

Sammensatte typer	
SML rep.	PHP rep.
`type K = a1 * ... * an`	`$K = array("K","cart", array($a1,..., $an );`
`type L = a list`	`$L = array("L","list", array($a));`
`disjoint D = a1|...|an`	`$D = array("D","disj", array($a1,...,$an));`

`table T = (a1 * ... an, b1 * ... bm)`
`$TKey = array("TKey", "cart", array($a1, ...,$an));`
⇒ `$TAtt = array("TAtt", "cart", array($b1, ...,$bm));`
`$T= array("T", "table",array($TKey, $TAtt));`

Tabel 6.8: *Typerne som de oversættes fra typesystemet til PHP.*

Når der er nestede erklæringer af sammensatte typer, opnås en træstruktur, heraf navnet typetræ. Typetræet for det gennemgående eksempel ses i Figur 6.10 på side 73

Værditræet

Til at indeholde værdien af en variabel, benyttes en lignende konstruktion, til forskel fra typetræet skal træet kun indeholde værdier.

6.2.2 Anvendelses specifik del

Ud fra signaturen genereres der en repræsentation af typerne i PHP. Typerepræsentationen skal følge de retningslinier, der blev givet i Tabel 6.8.

```
phpTypes dl -> string
```

For hvert element i listen genereres der en variabel i PHP, der skal indeholde typeinformationerne. `tabel` er et specialtilfælde, idet der skal oprettes ekstra variable til at indeholde nøgle og attributter, for det gennemgående eksempel får resultater der ses i Tabel 6.11 på side 73

```
Array (
    [0] => T
  [1] => table
    [2] => Array
        (
            [0] => Array
                (
                    [0] => TKey
                    [1] => cart
                    [2] => Array
                        (
                            [0] => Array
                                (
                                    [0] => b
                                    [1] => string
                                )
                        )
                )
            [1] => Array
                (
                    [0] => TAtt
                    [1] => cart
                    [2] => Array
                        (
                            [0] => Array
                                (
                                    [0] => a
                                    [1] => string
                                )

                            [1] => Array
                                (
                                    [0] => K
                                    [1] => cart
                                    [2] => Array
                                        (
                                            [0] => Array
                                                (
                                                    [0] => c
                                                    [1] => int
                                                )

                                            [1] => Array
                                                (
                                                    [0] => d
                                                    [1] => int
                                                )
                                        )
                                )
```

Tabel 6.9: *Typetræet for det gennemgående eksempel*

```
                        [2] => Array
                            (
                                [0] => L
                                [1] => list
                                [2] => Array
                                    (
                                        [0] => Array
                                            (
                                                [0] => K
                                                [1] => cart
                                                [2] => Array
                                                    (
                                                        [0] => Array
                                                            (
                                                                [0] => c
                                                                [1] => int
                                                            )

                                                        [1] => Array
                                                            (
                                                                [0] => d
                                                                [1] => int
                                                            )
                                                    )
                                            )
                                    )
                            )
                    )
            )
        )
)
```

Tabel 6.10: *Typetræet for det gennemgående eksempel(fortsat)*

```
<?php
    $a= array("a","string");
    $b= array("b","string");
    $c= array("c","int");
    $d= array("d","int");
    $K= array("K","cart",array($c,$d));
    $L= array("L","list", array($K));
    $TKey = array("TKey","cart",array($b));
    $TAtt = array("TAtt","cart", array($a,$K,$L));
    $T= array("T","table", array($TKey, $TAtt));
?>
```

Tabel 6.11: *Typerne i det gennemgående eksempel repræsenteret i PHP*

```
phpFun : dl -> string
```

Ud fra listen af typeerklæringer bliver menuen genereret. For hver tabel der forekommer i listen, bliver der sat fire menupunkter op, hvor hvert af disse menupunkter linker til de basale funktioner. Værdien af denne funktion er en streng, der indeholder den html kode, der genererer menuen.

For det gennemgående eksempel bliver der genereret den menu, der ses i Tabel 6.12

```
<h4>Opret</h4>
  <ul><li> <a href="index.php?action=create&amp;table=T">
    <strong>T</strong></a></li> </ul>
<h4>Opdater</h4>
  <ul><li> <a href="index.php?action=update&amp;table=T">
    <strong>T</strong></a></li> </ul>
<h4>Slet</h4>
  <ul><li> <a href="index.php?action=delete&amp;table=T">
    <strong>T</strong></a></li> </ul>
<h4>Vis</h4>
  <ul><li> <a href="index.php?action=view&amp;table=T">
    <strong>T</strong></a></li> </ul>
```

Tabel 6.12: *Brugermenuen for det gennemgående eksempel*

```
phpSwitch : dl -> string
```

Værdien af funktionen er en tekststreng indeholdende den PHP kode, der fortolker de funktionskald, der kommer frem menuen. For det gennemgående eksempel bliver der genereret den menu der ses i Tabel 6.13 på side 78.

Grænsefladen til det web-baserede system går gennem `index.php`, der sørger for at sende de postede parametre videre til de relevante funktioner. `index.php` er listet i Appendiks D.2.1 på side 123.

6.2.3 Generisk del

Alle funktionerne i den generiske del er implementeret i PHP, kildekoden er listet i Appendiks D.2.1 på side 123 of frem.

I det følgende gennemgås funktionaliteten for de brugerfunktioner der findes i menuen.

Indsæt

De tre vigtigste funktioner i indsættelsesdialogen er. `insertForm`, `updatetree` og `create`, i det følgende beskrives deres funktionalitet og indbyrdes forhold.

`insertform` er hovedfunktioen, denne funktion sørger for enten at opbygge en indtastningsformular vha. `create`, eller indsætte værditræet i databasen, beslutningen afgøres af parameteren `action`. Hvis værdien af `action` er *"opret"* skal typetræet indsætte i databasen og ellers laves indtastningsformularet vha. `create`

```
updatetree(typetree,valuetree,niveau) -> valuetree
```

Funktionen er rekursivt defineret, den virker ved at løbe typetræet igennem. For hvert blad i typetræet opdaters værditræet med de værdier, der er postet fra indtastningsformularen. Navnene på indtastningsfelterne er konstrueret ud fra den placering, som værdien skal have i værditræet.

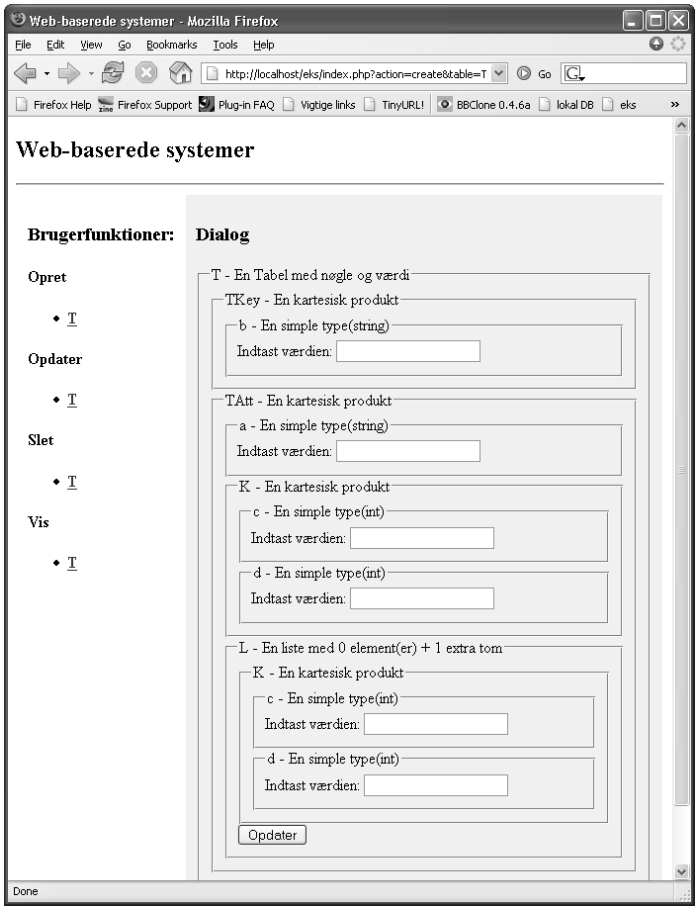
```
create(typetree,valuetree,action) -> string
```

Funktionen updatere bygger indsættelsesformularen på baggrund af typetræet og værditræet. Funktionen er rekursiv og bygger indtastningsformularen afhængig af sammensætningen af typetræet og valgene der er foretaget tidligere i dialogen.

`create` bygger rodelementet i typetræet ved at kalde create på alle de elementer, der indgår i dette. Hvor sammensatte og simple typer har deres egen opbygning. Alle de sammensatte typer kalder `create` med de indgående typer og de simple typer genererer de passende indtastningsfelter, På Figur 6.2 på næste side ses et eksempel på den nestede struktur, indtastningsforman er opbygget på baggrund af det gennemgående eksempel.

Vis

`Vis` virker grundlæggende på samme måde som `indsæt`. Det hele styres fra en hovedfunktion, `view(datatree,keyNames,keyValues)` Denne kaldes på alle de poster der er i den tabel som er argument til visningsmetoden.



Figur 6.2: *Indtastningformularen*

Da visningsmetoden skal kunne vise mere en et element af gangen, er der brug for en metode, der viser værdierne af posterne mere kompakt end det var tilfældet i indtastningsdialogen. Ved at implementere visningen via html konstruktoren `<table>` opnås en struktur der er mere kompakt.

Opdater og Slet

Ved disse to funktioner listes alle poster i tabellen. Ud for hver af posterne laves der en link til ental at slette posten eller opdatere den. Sletning af en post sker ved at posten slettes i den pågældende databasetabel, integritetsbegrænsningerne søger herefter for at tilhørende poster i de andre tabeller også slettes.

Opdatering af en post sker ved, at den valgte post indlæses til et værditræ og herefter kan værdierne i posten ændres via en dialog der er magen til den der fandtes ved `indsæt`. Igen er integriteten af databasen sikret af de integritetsbegrænsninger der er implementeret i databasen.

```
<?php if ($_REQUEST['action']=="create" ||
 $_REQUEST['action']=="Opret"){
    switch($_REQUEST['table'])
    {
        case "T":
            $main.=insertForm($T ,"",$_REQUEST['action']);
        break;
    }
}else if ($_REQUEST['action']=="update"){
    switch($_REQUEST['table'])
    {
        case "T":
            $main.=updateForm($T ,"","");
        break;
    }
}else if ($_REQUEST['action']=="view"){
    switch($_REQUEST['table'])
    {
        case "T":
            $main.=view($T ,"","");
        break;
    }
}else if ($_REQUEST['action']=="delete"){
    switch($_REQUEST['table'])
    {
        case "T":
            $main.=deleteRow($T ,"","");
        break;
    }
} ?>
```

Tabel 6.13: *Switchen for det gennemgående eksempel*

Kapitel 7

Konklusion

Tesen for specialet var, at en stor del af visse typiske web-anvendelser kan beskrives, ved hjælp af en simple model. Modellen skulle være i stand til at modellere sammensatte datatyper. Ud fra beskrivelsen af web-anvendelsen skulle hele infrastrukturen for systemet genereres automatisk.

I specialet er der udviklet et typesystem, der kan modellere størstedelen af de web-baserede systemer, man møder på Internettet. De typekonstruktorer der understøttes af systemet kan kombineres, hvilket gør typesystemet fleksibelt.

Ved udarbejdelse af de funktioner der oversætter typekonstruktorerne til et databaseskema, er en mellemrepræsentation et E/R-diagram. Fordelen ved at en signatur bliver omformet til et E/R diagram er, at det kan oversættes til et databaseskema efter en standard fremgangsmåde. Ønskes en udvidelse af typesystemet er det kun de funktioner, der oversætter typerne til E/R-diagram, der skal udvides. Udvidelse af typesystemet kræver, at den nye typekonstruktor medtages i de funktioner der udfører oversættelserne. Internt er alle transformatorfunktionerne opdelt efter de forskellige typekonstruktioner og de kan derfor let udvides med andre typekonstuktorer.

Relationsdatabaseskemaet er konstrueret efter standardreglerne for oversættelse af E/R-diagram til databaseskema med E/R stil for oversættelse af isa relationships. Der er ikke foretaget valg, der kan virke begrænsende ved en eventuel udvidelse. Det eneste der antages om entitetssættene i E/R diagrammet er, at de er erklæret inden de indgår i et relationship. Denne

antagelse har kun betydning for indsamling af fremmednøgler til svage entitetssæt.

Tesen er eftervist, da værktøjerne der er udviklet i specialet automatisk kan generere en web-anvendelse der anvendes i dagligdagen. Systemet har en høj grad af datamodellering og repræsenterer et typisk system.

Typesystemet er udvidet i forhold til det der blev udviklet i [JeT03], og genereringen af databasen er gjort fuldautomatisk, som grænseflade til databasen er der udviklet generiske funktioner til kommunikation med databasen.

7.1 Bidrag fra dette speciale

Ud fra et case study af et kørende system med en datamodellering der er typisk for mange web-anvendelser, er indholdet af siderne blevet analyseret og der er udviklet et typesystem, der kan beskrive dette.

Ud fra det opstillede typesystem er der udviklet retningslinier for hvorledes disse oversætter typesystemet til et E/R-diagram. Der er blevet udviklet et værktøj, der kan oversætte et web-baseret system, der er beskrevet vha. typesystemet, til et E/R-diagram og derfra til et relationsdatabaseskema.

Udtrykskraften i typesystemet kan bl.a. ses af hvor kompakt et system kan modelleres i forhold til det samme system modelleret vha. et E/R-diagram. Værktøjet er et godt selvstændigt værktøj, der i almindelighed kan bruges til at generere E/R-diagrammer ud fra et typesystem.

Som interaktion med databasen er der implementeret fire generiske funktioner til at oprette, indsætte, opdatere og slette data med.

Databasen er designet med integritetsbegrænsninger for at mindske antallet af forekomsten af null referencer og undgå redundans.

Både brugergrænseflade og databaseskemaet genereres automatisk på baggrund af signaturen.

7.2 Forbedringer

Specialets fokus er på konceptafprøvning, hvor de vigtigste typekonstruktorer er blevet medtaget, og ikke på at lave et færdigt produkt. Hvis sys-

temet skal fungere i praksis, vil det være nødvendigt at lave nogle tilføjelser der kan hjælpe med til præsentationen af det data, der modelleres i systemet.

Signaturen kan udvides til at indeholde oplysninger om i hvilken sammenhæng en sammensat type indgår, på den måde kan opbygningen af grænsefladen gøres mere brugervenlig. Eksempelvis vil det være brugbart at kunne angive, at en type skal være hovedmenu. For det kartesiske produkt ville en mulighed for at angive om visningen skal være vertikal eller horizontal være nyttig. En mulighed for at forbedre den generiske visning af data, kan opnås ved at indføre en ekstra parameter til hver sammensat type. Ved generering af den html kode der skal vise siden, skal denne ekstra parameter bruges som en *class* attribut til det element der indeholder den sammensatte type. Layoutet af siden kan herefter styres med et stylesheet (CSS).

I typesystemet kan `list` konstruktoren forbedres ved at argumentet i erklæringen, kan angives som et typeudtryk i stedet for blot et typenavn. Hvis denne funktionalitet skal implementeres, skal funktionerne der oversætter fra listen af typeerklæringer til entiteter og relationships udvides, dette er en lille udvidelse, og vil ikke påvirke det allerede implementerede system.

I systemet er der ikke mulighed for at definere specifikke funktioner. De fire indbyggede basisfunktioner virker på hele tabellen.

Der vil som regel altid være flere brugere af et web-baseret system, i denne situation er der altså flere brugere, hvor alle ikke skal have mulighed for at ændre i alle informationer, derfor får man brug for at implementere en form for adgangskontrol. En mulig måde at håndtere brugeradgang til systemet på, er at oprette en tabel indeholdende brugerne. Som en udvidelse af værdisættet i tabellen vil være en reference til den nye tabel.

Det skal medføre at en bruger kun kan manipulere en post i tabellen hvis han har rettigheder ifølge brugertabellen. Dette medfører altså at der skal oprettes en post i brugertabellen for hver person der har adgang til at ændre en given post.

Signaturen giver ikke mulighed for at lave cykliske erklæringer, dette har både ulemper og fordele. Fordelen er at E/R diagrammet kan laves automatisk og ulempen er at der er vise situationer der ikke kan modelleres med den foreslåede signatur. På Danmarks Rocenters hjemmeside indeholder tabellen personer et relationship til bådtyper, men der også en forbindelse fra bådtyper til personer. Begge dele kan ikke modelleres med typesystemet da en type skal være erklæret inden den kan indgår i en anden type. Ved

at tillade gensidigt rekursive erklæringer kan denne funktionalitet opnås. Dette medfører dog at indsamlingen af fremmednøgler skal ændres, da der nu ikke længere kan antages noget om erklæringsrækkefølgen.

Der er ikke udført systematiske test af systemet. Under udviklingen af værktøjerne er funktionerne blevet testet på eksemplet med træningsprogrammer og på det eksempel der er givet i implementationafsnittet. Til at observere om de rigtige poster blev sat ind i databasen, er denne overvåget ved brug af et web-baseret administrationsprogram [PMA], under transformationerne fra signatur til databaseskema er de enkelte skridt kontrolleret via udskrift af mellemrepræsentationerne.

Systematisk testning af systemet er ikke blevet gennemført grundet tidsmangel. De funktioner der er implementeret i SML bør testes via enhedstestning. Dette vil sikre at alle grene af koden kan nås, samt at den har den ønskede virkning på input.

7.3 Forslag til fremtidigt arbejde

For at have et færdigt system skal der tilføjes en del af den funktionalitet, der blev udviklet i [JeT03], bla. validering af input samt styring af brugergrupper.

For at de to systemer tilsammen kan bruges til at opbygge et web-baseret system, er der brug for et værktøj til at styre layoutet af web-systemet.

Hvis systemet skal indeholde følsomme data, er det vigtigt at disse sikres. Dvs. kommunikation mellem klient og server skal foregå over en sikker forbindelse og data skal opbevares 'sikkert' i databasen.

I case studiet blev der nævnt en mulighed for at importere data fra resultatdatabasen, der findes hos det internationale roforbund. Import af data fra andre web-baserede systemer er ikke blevet behandlet, men hvis det antages at data er tilgængelig i semistruktureret form, kan typesystemet bruges til at beskrive resultaterne, og databasen kan dermed genereres. Det der mangler i denne proces er en mekanisme til at udveksle data mellem de to systemer.

Litteratur

[HaR99]
Michael R. Hansen og Hans Rischel.
Introduction to Programming using SML.
Addison-Wesley, 1999.
ISBN: 0-201-39820-6

[CLR90]
T.H. Cormen, C.E. Leiserson, og R.L. Rivest.
Introduction to Algorithms.
MIT Press/McGraw-Hill Book Company, 1990.
ISBN: 0-262-53091-0

[GUW02]
Hector Gracia-Molina, Jeffrey D. Ullman og Jennifer Widom.
Database Systems, the complete book
Prentice Hall, 2002
ISBN: 0-130-31995-3

[ABD00]
Serge Abiteboul, Peter Buneman og Dan Suciu
Data on the web. From relations to semistructured Data and XML
Morgan Kaufman, 2000
ISBN: 0-555-860-622-X

[Kru00]
Steve Krug
Don't make me think, a common sense approach to web usability
Circle.com, 2000
ISBN: 0-7897-2310-7

[JeT03]
Tue Becher Jensen og Terkel Kristian Tolstrup
Teoribaseret udvikling af pålidelige webbaserede systemer
Kgs. Lyngby, 2003
IMM-THESIS-2003-19

[MML]
`http://www.dina.dk/~sestoft/mosml.html`
Moscow ML v.2.01
Jan 12, 2004

[PHP]
`http://www.php.net`
PHP Version 4.3.4

[MyS]
`http://www.mysql.com`
MySQL Version 4.0.18

[PMA]
`http://www.phpmyadmin.net`
phpMyAdmin Version 2.6.0-pl3

[GiM01]
A. Ginige and S. Murugesan
Guest Editors introduction: Web ingeneering: An introduction
IEEE MultiMedia, 8(1):14-18, 2001.

[SAC03]
Symposium on Applied Computing 2004
Proceedings og the 2004 ACM
University of Cyprus, 2004.
ISBN: 1-582-13812-1

[GMS94]
Michel Goossens, Frank Mittelbach and Alexander Samarin.
The LaTex Companion
Addison-Wesley, 1994.
ISBN: 0-201-54199-8

[KoD04]
Helmut Kopla og Patrik W. Daly.
Guide To LaTex, 4th edition
Addison-Wesley, 2004.
ISBN: 0-321-17385-6

Appendiks A

Ordliste

Forkortelse	Forklaring
BCNF	Boyce-Codd Normalform
Entitetssæt	En samling af entiteter
Entitet	Et abstrakt objekt der indeholder attributter
Attribut	En atomisk værdi
Relationship	Forholdet mellem to entitetssæt
E/R diagram	Entity Relationship diagram
Relation	Den tabel der skal oprettes i databasen
Webserver	Server der genererer html
DBMS	Database Management System
Browser	Et program der kommunikere med en webserver
html	Hyper text markup language
PHP	PHP hypertext preprocessor
SQL	Structured query language
CSS	Cascading stylesheet

Tabel A.1: *Ordliste*

A.1 Primitive Typer i MySQL

I MySQL er der 29 indbyggede primitive typer incl. de 4 BLOB[1] typer (TINYBLOB, BLOB, MEDIUMBLOB, og LONGBLOB).

Understøttet	MySQL Datatype	Længde
bool	TINYINT	-128 to 127
	TINYINT UNSIGNED	0 to 255
	SMALLINT	-32,768 to 32,767
	SMALLINT UNSIGNED	0 to 65,535
	MEDIUMINT	-8,388,608 to 8,388,607
	MEDIUMINT UNSIGNED	0 to 16,777,215
int	INT	-2,147,483,647 to 2,147,483,647
	INT UNSIGNED	0 to 4,294,967,295
	BIGINT	64 Bit
real	FLOAT	32 Bit Floating Point
	DOUBLE	64 Bit Floating Point
	DECIMAL	Variable Floating Point
	CHAR	1 to 255 Characters
string	VARCHAR	1 to 255 Characters
	TINYTEXT	1 to 255 Characters
	TEXT	1 to 65535 Characters
	MEDIUMTEXT	1 to 16,777,215 Characters
	LONGTEXT	1 to 4,294,967,295 Characters
	all BLOB types	1 to 4,294,967,295 Bytes
	DATE	Date without Time
	DATETIME	Date and Time
	TIMESTAMP	Date and Time
	TIME	Time
	YEAR	Year
	ENUM	Enumeration of Value Set
	SET	Set of Values

Tabel A.2: *Kilde: http://dev.mysql.com/tech-resources/articles/visual-basic-datatypes.html*

[1] binary large object

Appendiks B

Personkartotek

Den tabel der indeholder information om personerne på Danmarks Rocenter.

Tabel B.1: *Structure of table personer*

Field	Type	Null	Default
id	tinyint(4)	No	
fornavn	varchar(50)	Yes	NULL
efternavn	varchar(50)	Yes	NULL
email	varchar(50)	Yes	NULL
emailto	varchar(50)	Yes	NULL
emailtre	varchar(50)	Yes	NULL
homepage	varchar(100)	Yes	NULL
mobil	varchar(16)	Yes	NULL
fastnet	varchar(16)	Yes	NULL
tredie	varchar(16)	Yes	NULL
fjerde	varchar(16)	Yes	NULL
adresse	varchar(50)	Yes	NULL
postnr	varchar(5)	Yes	NULL
byen	varchar(30)	Yes	NULL
dob	date	Yes	NULL
hoejde	varchar(5)	Yes	NULL
vaegt	varchar(5)	Yes	NULL

Tabel B.1: *Structure of table personer (continued)*

Field	Type	Null	Default
kampvaegt	varchar(5)	Yes	NULL
icq	varchar(10)	Yes	NULL
profil	text	Yes	NULL
pasord	varchar(20)	Yes	NULL
dato	varchar(20)	Yes	NULL
infomail	tinyint(4)	No	1
roklub	varchar(40)	No	
kluburl	varchar(100)	No	
center	varchar(50)	No	
centertype	int(11)	No	0
sex	tinyint(4)	No	0
let	tinyint(4)	No	0
slags	varchar(20)	No	0
persontype	int(11)	No	0
sculler	tinyint(4)	No	0
baadtype	int(1)	No	0
billed	varchar(50)	No	
point	int(11)	No	0
erhverv	varchar(255)	No	
uddannelse	varchar(255)	No	
klubid	int(11)	No	0
datostamp	timestamp(14)	Yes	NULL
klasse	tinyint(4)	No	0
roertype	tinyint(4)	No	0
rosted	varchar(100)	No	
confirmprofile	tinyint(4)	No	0
confirmpicture	tinyint(4)	No	0

Appendiks C

Bådtyper

Tabellen der indeholder information om de forskellige bådtyper.

Field	Type	Null	Default
baadid	int(2)	No	
type	varchar(10)	No	
dbnavn	varchar(10)	No	
fisaNavn	varchar(20)	No	
fuld	varchar(50)	No	
sex	tinyint(4)	No	0
letvaegt	tinyint(4)	No	0
beskrivelse	text	No	
traener	int(11)	No	0
vaegt	int(11)	No	0
laengde	varchar(10)	No	0
bredde	varchar(10)	No	0
pris	int(11)	No	0
olympisk	tinyint(4)	No	0
typen	char(2)	No	

Tabel C.1: *Structure of table baadtyper*

C.1 Type erklæringer for Danmarks Rocenter

De begreber der blev introduceret i case-studyet, erklæres nu ved brug af typer og type konstruktorer. De typer der bliver erklæret i de første afsnit antages kendte i de efterfølgende afsnit.

Almene Meddelelser

```
type dag              = int
type maaned           = int
type aar              = int
type overskrift       = string
type meddelelsestekst = string
type dato             = (dag*maaned*aar)
type meddelelse       = overskrift * meddelelsestekst * dato
type meddelelser      = meddelelse list
```

Person Kartotek

```
type navn             = string
type klub             = string
type adresse          = string
type email            = string
type foedselsdato     = dato
type telefon          = int
type tidspunkt        = int
type placering        = int
type sted             = string
type baadnavn         = string
type stoerelse        = int
type olympisk         = bool
type hold             = string
type begivenhed       = string
type beskrivelse      = string
type begivenhedstype  = string
type hoejde           = real
type side             = string
type smig             = real
type ssmig            = real
type bsmig            = real
```

```
type gearing          = real
type hoejde           = real
type hforskel         = real

type roer             = side * smig * hoejde * gearing
type sculler          = ssmig * bsmig * hoejde * hforskel
type begge            = roer * sculler

datatype roertype     = roer | sculler | begge
datatype vaegtklasse  = let | tung

table baadtype        = (baad,baadnavn*stoerelse*olympisk)
tabel begivenheder    = (begivenhed*aarstal,
                                  sted*beskrivelse*begivenhedstype)
type personData       = navn * adresse * email * telefon *
                        foedselsdato * klub * ... * hold *
                        roertype * vaegtklasse
type resultat         = tidspunkt * baadtype * begivenhed * sted *
                        placering * tid
type resultater       = resultat list
type person           = persondata * resultater
table personKartotek  = (navn*foedselsdato,personData*resultater)
```

Kalender

```
type startdato  = int
type slutdato   = int
type navn       = string
type ekstra     = string

type kalenderBegivenhed = startdato * slutdato * navn * ekstra
table kalender = (startdato, kalenderBegivenhed)

indsætAktivitet : kalenderBegivenhed * kalenderBegivenheder -> kalenderBegivenheder
aflysAktivitet : navn * kalenderBegivenheder -> kalenderBegivenheder
```

Træningsprogrammer

```
type beskrivelse = string
```

```
type tid         = int
type afstand     = int
type pause       = int
type tempo       = int
type sted        = string
type formaal     = string
type kommentar   = string

disjoint basal   = afstand | tid
type serie       = basal list
disjoint session = basal | serie

type distance    = afstand * pause * tempo
type interval    = tid * pause * tempo

type distanceT   = distance list
type intervalT   = interval list
type udholdT     = session list
type teknikT     = basal * tempo
type test        = basal * sted * formaal * kommentar * beskrivelse

disjoint program = intervalT | distanceT | udholdT | teknikT | test
table traening   = (beskrivelse, program)
```

Dagbog

```
table dagbog = (dato, traening)
```

Registrering af test

```
type testNavn      = string
type beskrivelse   = string
type effect        = real
type maelkesyre    = real
type iltoptagelse  = real
type testDato      = dato
```

```
type splitlaengder = laengde list
type resultat      = testNavn * splitlaengder *  laengde *
                     effekt * syre * ilt * beskrivelse

table resultater   = (person*testDato,resultat)
```

Ranglister

```
visTestRagliste : testNavn * resultater -> (person * resultat)list
```

Tilmeldinger og ansøgninger

```
tilmeldHold : hold * personKartotek -> tilmelding

ansøgning   : person * personKartotek -> ansøgning
```

Appendiks D

Kildekode

I dette Kapitel listes al koden der er blevet implementeret i forbindelse med genereringen af det web-baserede system. I Afsnit D.2.8 på side 142 listes koden der transformere en signatur til et databaseskema, i Afsnit D.2 på side 123 listes det kode der genererer den grafiske brugergrænseflade og i Afsnit D.2.8 på side 142 listes de forspørgsler der anvendes som interface til databasen.

D.1 Database genereringen

Parseren

Følgende filer udgør tilsammen parseren:

`Absyn.sml Exprlex.lex Exprpar.sml` og `parse.sml`

Absyn.sml

Den abstrakte grammatik.

```
1  (* The abstract syntax for webbased systems with lists,       *)
2  (* cartesian product, disjoint types and tables               *)
3
4  datatype typeexpr =
```

```
5       typeName of string
6     | Ptype of string
7     | Ltype of typeexpr
8     | Ctype of typeexpr * typeexpr
9     | Ttype of typeexpr * typeexpr
10    | Dtype of typeexpr * typeexpr
11
12  datatype decl =
13      Typedecl of string * typeexpr
14    | Tabledecl of string * typeexpr
15    | Disjdecl of string * typeexpr
16
17  type speci = decl list
```

Exprlex.lex

Information om keywords til scanneren.

```
1  {
2   open Lexing Exprpar;
3
4   (*(message, loc1, loc2)*)
5   exception LexicalError of string * int * int
6
7   fun lexerError lexbuf s = raise LexicalError
8                  (s, getLexemeStart lexbuf, getLexemeEnd lexbuf);
9
10 (* Scan keywords as identifiers and use this function to      *)
11 (* distinguish them.                                          *)
12
13  fun keyword s =
14      case s of
15          "type"         => TYPE
16        | "disjoint"     => DISJOINT
17        | "list"         => LIST
18        | "table"        => TABLE
19        | "of"           => OF
20        | "end"          => END
21        | "sig"          => SIG
22        | "signature"    => SIGNATURE
23        | "int"          => PRIMITIVE s
24        | "string"       => PRIMITIVE s
25        | "bool"         => PRIMITIVE s
26        | "real"         => PRIMITIVE s
27        | _              => TYPENAME s;
28 }
29
30 rule Token = parse
31      [` ` `\t` `\n` `\r`]  { Token lexbuf }
32    | [`0`-`9`]+            { case Int.fromString
33                                    (getLexeme lexbuf) of
34                NONE   => lexerError lexbuf "internal error"
35              | SOME i => INT i
36          }
37    | [`a`-`z``A`-`Z`][`a`-`z``A`-`Z``0`-`9`]*
38                          { keyword (getLexeme lexbuf) }
39    | `*`                 { CART  }
40    | `=`                 { EQ    }
41    | `,`                 { COMMA }
42    | `|`                 { PIPE  }
43    | `(`                 { LPAR  }
44    | `)`                 { RPAR  }
45    | eof                 { EOF   }
```

```
46    | _            { lexerError lexbuf "Illegal symbol in input" }
47  ;
```

Exprpar.grm

Grammatikken som parsningen bliver lavet ud fra.

```
%{
  (* Exprpar.grm: parser specification *)
  open Absyn;
%}

%token <int> INT
%token <string> TYPENAME PRIMITIVE
%token TYPE DISJOINT
%token CART LIST TABLE EQ COMMA PIPE OF SIGNATURE
%token SIG END LPAR RPAR
%token EOF

%left CART      /*  lowest precedence  */
%left PIPE
%right LIST
%nonassoc EQ    /*  highest precedence */

%start Main
%type <Absyn.speci> Main Speci
%type <Absyn.decl> Decl
%type <Absyn.typeexpr> Typeexpr Simpexpr

%%

Main:
    SIGNATURE TYPENAME EQ SIG Speci END EOF    { $5 }
;

Speci:
    /* empty */                              { [] }
  | Decl Speci                               { $1 :: $2 }
;

Decl:
    TYPE TYPENAME EQ Typeexpr                { Typedecl($2,$4)  }
  | TABLE TYPENAME EQ Typeexpr               { Tabledecl($2,$4) }
  | DISJOINT TYPENAME EQ Typeexpr            { Disjdecl($2,$4)  }
;

Typeexpr:
    TYPENAME                                 { typeName $1      }
  | PRIMITIVE                                { Ptype $1         }
  | Typeexpr LIST                            { Ltype $1         }
  | Typeexpr CART Typeexpr                   { Ctype($1,$3)     }
  | LPAR Typeexpr COMMA Typeexpr RPAR        { Ttype($2,$4)     }
```

```
46    | Typeexpr PIPE Typeexpr                  { Dtype($1,$3)      }
47  ;
```

parse.sml

Parsing af signatur.

```
app load ["Location", "Nonstdio", "Exprpar", "Exprlex"];
open Absyn;

(*============================================================*)
(* 1.2 *)
(* Fancy parsing from a file; show the offending program piece*)
(* on error                                                   *)
fun parseExprReport file stream lexbuf =
    let val expr =
            Exprpar.Main Exprlex.Token lexbuf
            handle
               Parsing.ParseError f =>
                   let val pos1 = Lexing.getLexemeStart lexbuf
                       val pos2 = Lexing.getLexemeEnd lexbuf
                   in
                       Location.errMsg (file, stream, lexbuf)
                                       (Location.Loc(pos1, pos2))
                                        "Syntax error."
                   end
             | Exprlex.LexicalError(msg, pos1, pos2) =>
                   if pos1 >= 0 andalso pos2 >= 0 then
                       Location.errMsg (file, stream, lexbuf)
                                       (Location.Loc(pos1, pos2))
                                        ("Lexical error: " ^ msg)
                   else
                       (Location.errPrompt ("Lexical error: " ^
                                            msg ^ "\n\n");
                        raise Fail "Lexical error");
    in
        Parsing.clearParser();
        expr
    end
    handle exn => (Parsing.clearParser(); raise exn);

(*============================================================*)
(* 1.1 *)
(* Create lexer from instream *)
fun createLexerStream (is : BasicIO.instream) =
    Lexing.createLexer (fn buff => fn n
                               =>
                Nonstdio.buff_input is buff 0 n)
(*============================================================*)
(* 1 *)
(* Parse a program from a file, with error reporting *)
fun parsef file =
```

```
46      let val is  = Nonstdio.open_in_bin file
47          val expr= parseExprReport file is (createLexerStream is)
48                    handle exn => (BasicIO.close_in is; raise exn)
49      in
50          BasicIO.close_in is;
51          expr
52      end;
```

D.1.1 test.sml

Main filen.

```
1  load "TextIO";
2  use "file.sml" ;      (* create files from lists        *)
3  use "auxiliary.sml"; (* Auxiliary functions            *)
4  use "parse.sml";      (* Parse functions                *)
5  use "er.sml" ;        (* E/R transformation functions   *)
6  use "schema.sml" ;    (* Schema transformation functions*)
7  use "php.sml" ;       (* convert types to php syntax    *)
8
9  (*============================*)
10 (*1 Parse a signature into a list of declarations *)
11 val declList = parsef "test2.sig";
12
13 (*============================*)
14 (*2 Transform into E/R diagram*)
15 val typeNameList = getTypeList(declList,[]);
16   makeFile("deklarations.txt",typeName2txt(typeNameList));
17
18 val entities = getEntities(declList,[],typeNameList)
19 val entTypes = entities;
20   val entNameType = ent2txt(entities);
21   makeFile("entNameType.txt",entNameType);
22
23 val relationships =
24              getRelationships(declList,[],entities,typeNameList)
25   val relNavnType = rel2txt(relationships);
26   makeFile("relationships.txt",relNavnType);
27
28 val ue = updateEntities(entities,relationships);
29   val uet = ent2txt(ue);
30   makeFile("ue.txt",uet);
31
32 val entities =
33     removeDublicates(updateEntities(entities,relationships),[]);
34   val entNavnType = ent2txt(entities);
35   makeFile("entitiesUpdated.txt",entNavnType);
```

```
36
37 (*============================*)
38 (*3 Transform into DBskema    *)
39 val (isaRel, nonISA) = List.partition isISA relationships
40   val isaReltxt = rel2txt(isaRel);
41   makeFile("relISAtxt.txt",isaReltxt);
42   val nonISAtxt = rel2txt(nonISA);
43   makeFile("relNonISAtxt.txt",nonISAtxt);
44
45 val (weakRel,normRel)= List.partition isWeak nonISA
46   val weakReltxt = rel2txt(weakRel);
47   makeFile("relWeaktxt.txt",weakReltxt);
48   val normReltxt = rel2txt(normRel);
49   makeFile("relNormtxt.txt",normReltxt);
50
51 val (weakEnt,normEnt) = List.partition isWeak entities
52   val normEnttxt = ent2txt(normEnt);
53   makeFile("entNormtxt.txt",normEnttxt);
54   val weakEnttxt = ent2txt(weakEnt);
55   makeFile("entWeaktxt.txt",weakEnttxt);
56
57
58 (* Update the weak entities with the foreign keynames *)
59 val updatedEntities =
60            getKeyNames(rev weakEnt,weakRel@isaRel,entities);
61
62 val (weakEnt,normEnt) = List.partition isWeak updatedEntities
63   val normEnttxt = ent2txt(normEnt);
64   makeFile("entNormtxt.txt",normEnttxt);
65   val weakEnttxt = ent2txt(weakEnt);
66   makeFile("entWeaktxt.txt",weakEnttxt);
67
68
69 (* transform the updated entites and relationships into       *)
70 (* relations                                                  *)
71 (*weakEnt + weakRel@isaRel + (allEnt)  ->   Relations*)
72 val RFWE=wEnt2rel(weakEnt, weakRel@isaRel, updatedEntities)
73
74 (*normEnt                        ->   Relations*)
75 val RFE = ent2rel(normEnt)
76
77 (*normRel + (allEnt)            ->   Relations*)
78 val RFR = rel2rel(normRel,updatedEntities)
79
80 val relations = RFWE@RFR@RFE;
81 (* makeFile("relations.tex",rel2tex(relations)); *)
82
83 (* create the database schema *)
84 val schema = cSchema(rev relations,entTypes);
```

```
85
86  (* write the databaseschema to a file *)
87  makeFile("database.sql",schema);
88
89  val phptype = phpTypes(declList);
90  makeFile("typeDecl.php","<?php\n"^phptype^"\n?>");
91
92
93  fun getTableNames [] = []
94    | getTableNames (p::ps) =
95       case p of
96         Tabledecl(s,e) => s::getTableNames(ps)
97    | _               => getTableNames(ps)
98
99  val tableList = getTableNames(declList)
100
101 val phpfunc = phpFun(tableList);
102 makeFile("userFun.php","<?php\n"^phpfunc^"\n?>");
103
104 val phps = phpSwitch(tableList);
105 makeFile("userSwitch.php","<?php\n"^phps^"\n?>");
```

D.1.2 auxiliary.sml

Fælles funktioner.

```
1   (* Common functions used by "er.sml" and "db.sml"             *)
2   (*=============================================================*)
3   (* val makeFile = fn : string * string -> unit               *)
4   (* write a string to file. Arg1 is the filename              *)
5   fun makeFile(filename,s) =
6     let
7       val out = TextIO.openOut(filename)
8     in
9       TextIO.output(out,s);
10      TextIO.closeOut(out)
11    end;
12  (*=============================================================*)
13  (* returns true is the typeName is a primitive type  *)
14  fun isPrim s = s="int" orelse s="string" orelse
15                 s="real"orelse s="bool"
16  (*=============================================================*)
17  fun isCompound m =  m="disj" orelse m="list" orelse m="cart"
18                      orelse m="table" orelse m="sub"
19  (*=============================================================*)
20  (* check if name exist in typeName list                       *)
21  fun isDecl(x,ys) = List.exists (fn y => x=y) ys
```

```
22 (*============================================================*)
23 (*s is simple if s is declared and it's type is primive*)
24 fun isSimp(s,[])        = false (* s not found in list*)
25   | isSimp(s,(n,t)::tn) =
26                      (s=n andalso isPrim(t)) orelse isSimp(s,tn)
27 (*============================================================*)
28 fun knownType m = isPrim(m) orelse isCompound(m)
29 (*============================================================*)
30 (*  *)
31 (*fun isAtt(s,[])          = false *)
32 (*  | isAtt(s,(n,t)::att) = (s=n) orelse isAtt(s,att)*)
33 (*============================================================*)
34 fun getEntityName (n,k,a,m) = n
35 (*============================================================*)
36 (* Check if relationship is an isa relationship *)
37 fun isISA (a,b,c,d) = d="isa"
38 (*============================================================*)
39 (* Find the entities and relationships which are weak*)
40 fun isWeak (a,b,c,d) = d="weak"
41 (*============================================================*)
42 (* Return the primitive type of a simple type *)
43 exception typeNotSimple of string
44 fun getType(a,[]   )     = raise typeNotSimple a (*not an att.*)
45   | getType(a,(n,t)::aa) = if a=n then t else getType(a,aa)
46 (*============================================================*)
47 fun member(x,ys) = List.exists (fn y => x=y) ys
48 (*============================================================*)
49 fun merge (ls,[]) = ls
50   | merge (ls,r::rs) = if member(r,ls) then merge(ls,rs)
51                                        else merge(r::ls,rs)
52 (*============================================================*)
```

D.1.3 er.sml

Transformering til E/R diagram.

```
(*============================================================*)
(* E/R transformation functions                               *)

(*============================================================*)
exception notEntity;
fun getName e =
  case e of
      typeName x   => x                                (* a *)
    | Ptype t      => t                                (* b *)
    | _            => raise notEntity                  (* c *)

(*============================================================*)
exception NameList;
fun getNameList e =
  case e of
      typeName x   => [x]
    | Ptype t      => [t]
    | Ctype(e1,e2) => getName(e2)::getNameList(e1)
    | _            => raise NameList

(*============================================================*)
(* check that all attributes of e are declared *)
exception missingDeclaration of string;
fun isDecl (e, [])         =
     (case e of typeName x => raise missingDeclaration x
      | _          => raise missingDeclaration "Unknown"
     )
  | isDecl (e, (n,t)::att) =
     (case e of
        (* find x among the declared attributes *)
        typeName x        => x=n orelse isDecl(e,att)
      | Ptype t      => true  (* it's just a primitive type *)
      | Ltype(e1)    => isDecl(e1,(n,t)::att)
      | Ctype(e1,e2) => isDecl(e1,(n,t)::att)
                            andalso isDecl(e2,(n,t)::att)
      | Dtype(e1,e2) => isDecl(e1,(n,t)::att)
                            andalso isDecl(e2,(n,t)::att)
      | Ttype(e1,e2) => isDecl(e1,(n,t)::att)
                            andalso isDecl(e2,(n,t)::att)
     )

(*============================================================*)
(* 2.1 *)
(* Find the name and type of an attribute in a declaration    *)
(* and return it                                              *)
```

```
46  exception undefinedAttribute of string;
47  fun addDeclType (p, att) =
48    case p of
49        Typedecl(s,e)     =>
50          (case e of
51                            (* x is a variable, add s to
52                   list of type if x is defined    *)
53             typeName x  => if isDecl(e,att) then
54                             if isSimp(x,att) then
55                   [(s,x)]                   (*a*)
56                else
57                     [(s,"CART")]              (*b*)
58                            else raise undefinedAttribute(x) (*c*)
59                             (* its a primitive type,
60                   add s to list of att. *)
61           | Ptype t     => [(s,t)]                          (*d*)
62           | Ltype e1    => if isDecl(e1,att) then
63                              [(s,"LIST")]                  (*e*)
64                            else
65                  raise undefinedAttribute(s)    (*f*)
66           | Ctype(e1,e2)=> if isDecl(e1,att)
67                             andalso isDecl(e2,att) then
68                              [(s,"CART")]                  (*g*)
69                            else
70                  raise undefinedAttribute(s)    (*h*)
71           | _           => raise undefinedAttribute         (*i*)
72                            ("Untreated branch in 'addDeclType' ")
73          )
74      | Disjdecl(s,e)    => if isDecl(e,att) then
75                              [(s,"DISJ")]                  (*j*)
76                            else
77                raise undefinedAttribute(s)    (*k*)
78      | Tabledecl(s,e)   => if isDecl(e,att) then
79                              [(s,"TAB")]                   (*l*)
80                            else
81                  raise undefinedAttribute(s)    (*m*)
82  (*============================================================*)
83  (* 2 *)
84  (* Add the attributes to the list one by one                  *)
85  (* the type of the first parsetree is added as the last       *)
86  (* attribute The list of attributes is reversed when returned *)
87  (* I use append because I need to be able to create empty pair*)
88  (* in addAttribute                                           *)
89  fun getTypeList ([],att)      = rev att          (* a *)
90    | getTypeList ((d::dl),att) =                (* b *)
91                            getTypeList(dl,addDeclType(d,att)@att)
92  (*============================================================*)
93  (* 4.4 *)
94  exception unexpected of string;
```

```
95  fun getCartAtt(e,tn) =
96    case e of
97        typeName x   => if isSimp(x,tn) then [(x,getType(x,tn))]
98                                        else []
99      | Ctype(e1,e2) => getCartAtt(e2,tn)@getCartAtt(e1,tn)
100                       (*no attribute is added because a
101               Relationship is needed here*)
102     | Ltype e1     => []
103     | Ptype t      => raise unexpected(
104             "Simple type must be declared through a variable")
105     | Dtype t      => raise unexpected("Disjoint type not"^
106         "allowed here, must be declared as type before use")
107     | _            => raise unexpected("getCartAtt");
108
109 (*============================================================*)
110 (* 4.3 *)
111 (* create entity if x is an attibute, otherwise create        *)
112 (* an empty entity                                            *)
113 fun createEntity(s,x,tn) = if isSimp(x,tn)
114                 then[(s^"_"^x,[],[(x,getType(x,tn))],"sub")]
115                 else[] (* create elsewhere *)
116
117 (*============================================================*)
118 (* 4.2 *)
119 exception notAsubEntity;
120 fun createSubEntities(s,e,att) =
121   case e of
122       typeName x   => createEntity(s,x,att)
123     | Dtype(e1,e2) => createEntity(s,getName(e2),att)
124                                   @createSubEntities(s,e1,att)
125     | _            => raise notAsubEntity
126
127
128 (*============================================================*)
129 (* create a new list element if the e is a simple type        *)
130 fun createListEntity(s,e,tn) =
131   case e of
132       typeName x => if isSimp(x,tn) then
133                        [(x^"_Element",[],[(x,getType(x,tn))],s)]
134                     else
135             []
136     | _          => []
137
138 (*============================================================*)
139 (* 4.1 *)
140 (* examine one parse tree for possible occurences of one or   *)
141 (* more entities Create the entity based on the parsetree     *)
142
143 exception entity of string
```

```
144 fun getEntity(d,ent,tn) =
145   case d of
146     Typedecl(s,e)   =>
147       (case e of
148           (* This is a special case of the cartisian produkt*)
149         typeName x   => [(s,[],[],"cart")]
150                         (* A simple type is not an entity    *)
151       | Ptype t      => []
152                         (* If s is simple then createListEntity
153                 will create an list element       *)
154       | Ltype e1     => [(s,[],[],"list")]@
155                                 createListEntity(s,e1,tn)
156
157                         (* e1 holds the Id for the table,    *)
158                         (* e2 holds the values               *)
159       | Ctype(e1,e2) => [(s,[],getCartAtt(e,tn),"cart")]
160       | _            => raise entity(s)
161       )
162   | Disjdecl(s,e)   => createSubEntities(s,e,tn)
163                                  @[(s,[],[],"disj")]
164   | Tabledecl (s,(Ttype(e1,e2)))
165                     => [(s,getCartAtt(e1,tn),
166                                     getCartAtt(e2,tn),"table")]
167   | _               => raise entity("Unkwnow type declaration")
168
169 (*============================================================*)
170 (* 4 *)
171 (* examine the first parsetree and call the function with the *)
172 (* remaining parsetrees getEntity is appended to the tail     *)
173 (* because one call to getEntity can result in more entities  *)
174 fun getEntities([],ent,tn)      = ent
175   | getEntities((d::dl),ent,tn) =
176                       getEntity(d,ent,tn)@getEntities(dl,ent,tn)
177
178 (*============================================================*)
179 (* 5.7 *)
180 (* create entity if x is an attibute, otherwise it is created.*)
181 fun createRelationship(s,x,ant,tn) = if isSimp(x,tn)
182     then [(s,s^"_"^x,"p-1","isa")]
183                 (* create isa relationship to containing entity*)
184       else [(s,x,"p-1","isa")]
185                 (* x is an entity, create both isa relationship*)
186                 (* to the extra entity and a one-one           *)
187                 (* relationship to the containing entity.      *)
188
189 (*============================================================*)
190 (* 5.6 *)
191 exception DisjointRelationShip of string
192 fun getDisjointRelationships(s,e,ent,att) =
```

```
193   case e of
194                        (* The last one *)
195       typeName x         => createRelationship(s,x,ent,att)
196     | Dtype(e1,e2) => createRelationship(s,getName(e2),ent,att)@
197                           getDisjointRelationships(s,e1,ent,att)
198     | _            => raise DisjointRelationShip s
199
200 (*============================================================*)
201 (* 5.5 *)
202 fun createCartRelationship(s,x,tn) =
203   if isSimp(x,tn) then []                    (* x is an attrib.*)
204                   else [(s,x,"p-1","weak")] (* x is an entity *)
205
206 (*============================================================*)
207 (* 5.2 *)
208 exception CartRelationships
209 fun getCartRelationships(s,e,ent,att) =
210   case e of
211       typeName x         => createCartRelationship(s,x,att)
212     | Ctype(e1,e2) => createCartRelationship(s,getName(e2),att)
213                            @getCartRelationships(s,e1,ent,att)
214     | _            => raise CartRelationships
215
216 (*============================================================*)
217 (* 5.3 *)
218 fun createTblRelationship(s,x,tn) =
219   if isSimp(x,tn) then []                    (* x is an attrib.*)
220                   else [(s,x,"n-1","tabel")](* x is an entity *)
221
222 (*============================================================*)
223 (* 5.4 *)
224 exception TabelRelationships
225 fun getTblRelationships(s,e,ent,att) =
226   case e of
227       typeName x         => createTblRelationship(s,x,att)
228     | Ctype(e1,e2) => createTblRelationship(s,getName(e2),att)
229                             @getTblRelationships(s,e1,ent,att)
230     | _            => raise TabelRelationships
231
232 (*============================================================*)
233 (* 5.1 *)
234 (* examine one parse tree for possible occurences of entities *)
235 exception Relationship of string
236 (* p:   a parse tree *)
237 (* rel: The accumulating list of relations *)
238 (* ent: The total list of entities *)
239 (* att: The total list of attributes *)
240 fun getRelationship(p,rel,ent,att) =
241   case p of
```

```
242     Typedecl(s,e)     =>
243       (case e of
244          (* x can both be an attribute and an entity          *)
245          typeName x        => [(s,x,"p-1","weak")]
246          (* An attribute, hence not an entity*)
247        | Ptype t      => []
248          (* The weak Relationship in a list                   *)
249        | Ltype e1     => if isSimp (getName(e1),att) then
250                             [(s,getName(e1)^"_Element",
251                                          "p-n","weak")]
252              else
253                [(s,getName(e1),"p-n","weak")]
254          (* No relationship is needed for the Id, hence call  *)
255          (* the method with e2.                               *)
256        | Ctype(e1,e2) => getCartRelationships(s,e,ent,att)
257        | _            => raise Relationship s
258       )
259   | Disjdecl(s,e)     => getDisjointRelationships(s,e,ent,att)
260   | Tabledecl(s,(Ttype(e1,e2)))
261                       => getTblRelationships(s,e1,ent,att)@
262                 getCartRelationships(s,e2,ent,att)
263   | _                 => raise Relationship
264                          ("Unknows decl. in getRelationship")
265
266 (*============================================================*)
267 (* 5 *)
268 (* Find the Relationships in the different parsetrees         *)
269 fun getRelationships([],rel,ent,att)      = rel
270   | getRelationships((d::dl),rel,ent,att) =
271       getRelationship(d,rel,ent,att)@
272     getRelationships(dl,rel,ent,att)
273
274 (*============================================================*)
275 (* update the marking of one entity, marks can be either      *)
276 (* weak : get key via weak relationships                      *)
277 (* auto : create autoincrementing ID                          *)
278 (* norm : The entity contains the key                         *)
279 fun updateEnt((n,k,a,m),[])               =
280                   if m="table"
281                     then (n,k,a,m)
282          else (n,(n^"ID","auto")::k,a,"auto")
283   | updateEnt((n,k,a,m),(n1,n2,i,m2)::rel)=
284                   if n=n2 then
285             case i of "p-1" => (n,k,a,"weak")
286                             | "p-n" => (n,(n^"count","int")
287                                         ::k,a,"weak")
288               | _     => (n,k,a,m)
289             else
290             updateEnt((n,k,a,m),rel)
```

```
291
292 (*============================================================*)
293 (* The function chages the marking of the entity sets, all    *)
294 (* should have a mark of either weak or auto after this, no   *)
295 (* keys or attributes are added here                          *)
296 fun updateEntities([],relationships) = []
297   | updateEntities(e::ent,relationships)=
298                   updateEnt(e,relationships)::
299           updateEntities(ent,relationships)
```

D.1.4 schema.sml

Transformering til databaseskema.

```
1  (*============================================================*)
2  exception typeNotSimple of string
3  fun getPrimType(a,[]   )     = raise typeNotSimple
4                                  ("Primitive type not found : "^a)
5    | getPrimType(a,(n,t)::aa) = if a=n then t
6                                 else getPrimType(a,aa)
7  (*============================================================*)
8  (* Returns a list containing the keys of the weakly related   *)
9  (* entities to the given entity naed 'n', the list may contain*)
10 (* dublicates                                                 *)
11 fun FindWeakRelEnt(n,[],entities) = []
12   | FindWeakRelEnt(n,(n1,n2,r,m)::rel,entities) =
13     if (n=n1) then
14       if (m="weak") then
15         n2::FindWeakRelEnt(n2,rel,entities)
16            @FindWeakRelEnt(n ,rel,entities)
17       else
18         n2::FindWeakRelEnt(n,rel,entities)
19     else
20     if (n=n2) then
21       if (m="weak") then
22         n1::FindWeakRelEnt(n1,rel,entities)
23            @FindWeakRelEnt(n ,rel,entities)
24       else
25         n1::FindWeakRelEnt(n,rel,entities)
26     else
27       FindWeakRelEnt(n,rel,entities)
28
29 (*============================================================*)
30 (* return the key list of an entity *)
31 (* n: the name of the entity, the list is allEntities*)
32 fun getKeys(n,[])              = []
33   | getKeys(n,(n1,k,a,m)::es) = if n=n1 then k
34                                 else getKeys(n,es)
35
36 (*============================================================*)
37 (* return keys of all entities in the list, *)
38 (* may contain dublicates                                     *)
39 fun getAllKeys([],allEntities)             = []
40    | getAllKeys((n,_,_,_)::ns, allEntities) =
41              getAllKeys(ns, allEntities)@getKeys(n,allEntities)
42
43 (*============================================================*)
44 fun contains((n1,k1,a1,m1),[])                = false
45   | contains((n1,k1,a1,m1),(n2,k2,a2,m2)::ls) =
```

```
46                          n1=n2 orelse contains((n1,k1,a1,m1),ls)
47
48  (*============================================================*)
49  fun relatedTo((n,k,a,m),(n1,n2,_,_)) = n=n1 orelse n=n2
50
51  (*============================================================*)
52  (* u is the untreated entity *)
53
54  fun related(u,[],    untreated,treated) = untreated
55    | related(u,r::rel,untreated,treated) =
56        if relatedTo(u,r) andalso not(contains(u,treated))
57          then related(u,rel,u::untreated,treated)
58          else related(u,rel,untreated,treated)
59  (*============================================================*)
60  fun getEntName(n,k,a,m)=n
61  fun getKeys(n,k,a,m)=k
62  fun getAtts(n,k,a,m)=a
63  (*============================================================*)
64  (* Find the entity from the name                              *)
65  exception NoEntity
66  fun findEnt(n,[])             = raise NoEntity
67    | findEnt(n,(n1,k,a,m)::es) = if n=n1 then (n1,k,a,m)
68                                        else findEnt(n,es)
69
70  (*============================================================*)
71  fun neighboursTO(n,[],allEnt)              = [n]
72    | neighboursTO(n,(n1,n2,_,_)::rs,allEnt) =
73      let val (x,y,z,w) = n
74      in
75        if x=n1 then findEnt(n2,allEnt)::neighboursTO(n,rs,allEnt)
76        else
77        if x=n2 then findEnt(n1,allEnt)::neighboursTO(n,rs,allEnt)
78        else
79        neighboursTO(n,rs,allEnt)
80      end
81
82  (*============================================================*)
83  (* check if the entity given in the first         *)
84  (* argument is in the list in the second argument *)
85  fun isTreated(_,[])   = false
86    | isTreated((n,x,y,z),(m,_,_,_)::rs)=
87                              n=m orelse isTreated((n,x,y,z),rs)
88
89  (*============================================================*)
90  (* input = a weak entitet. output = a list of keys            *)
91  (* The function examines all the untreated entities           *)
92  (* The function returns all the entities which is needed for  *)
93  (* the weak key                                               *)
94  (* tr: the treated entities *)
```

```
95  fun getKeysForWeakEnt(ent,allEnt,allWeakRel)    =
96      let fun visit([],tr)    = tr
97            | visit(e::es,tr) =
98               if isTreated(e,tr) then visit(es,tr)
99               else visit(es@neighboursTO(
100                                 e,allWeakRel,allEnt),e::tr)
101     in
102         getAllKeys(visit([ent],[]),allEnt)
103     end
104
105 (*============================================================*)
106 (* gets the foreign key to en entity regarding one rel.ship   *)
107 exception entityNotFound of string
108 fun keys (name,[])              = raise entityNotFound name
109   | keys (name,(n,k,a,m)::ent) =
110       if name=n
111       then (n,k)
112     else keys(name,ent)
113
114 (*============================================================*)
115 (* gets the foreign key to en entity                          *)
116 (* only name and keys are needed, keytypes are not            *)
117 fun getFkeys(n,[]               ,aEnt) = []
118   | getFkeys(n,(n1,n2,_,_)::rel,aEnt) =
119       if n=n2 then keys(n1,aEnt)::getFkeys(n,rel,aEnt)
120             else getFkeys(n,rel,aEnt)
121
122 (*============================================================*)
123 (* Returns a list of relations containing all keys in the weak*)
124 (* entities, including foreign keys                           *)
125 fun wEnt2rel([]             ,wRel,allEnt) = []
126   | wEnt2rel((n,k,a,m)::ent,wRel,allEnt) =
127      (n,k,a,getFkeys(n,wRel,allEnt))::wEnt2rel(ent,wRel,allEnt)
128 (*============================================================*)
129 fun getTypeName []          = []
130   | getTypeName ((n,t)::tn) = n::getTypeName(tn);
131
132 (*============================================================*)
133 (* convert norm entities to relations                         *)
134 (* just remove the marking and insert enpty list i foreign key*)
135 fun ent2rel             [] = []
136   | ent2rel((n,k,a,m)::ls) = if m="auto" then
137                                 (n,k,a,[])::ent2rel(ls)
138               else
139                 (n,k,a,[])::ent2rel(ls)
140
141 (*============================================================*)
142 (* copy keys in isa hierachy                                  *)
143 exception keysNotFound of string
```

```
144 fun copyRootKeys(n1,[])             = raise keysNotFound n1
145   | copyRootKeys(n1,(n,k,a,m)::ent) = if n=n1
146         then k
147       else copyRootKeys(n1,ent)
148 (*============================================================*)
149 (* copy keys from root(n1) entity to sub entity(n2)           *)
150 (* returns an updated ent list                               *)
151 exception RootEntityNotFound of string
152 fun updateWithRootKeys(n2,n1,[]            ,allEnt) =
153       raise RootEntityNotFound (n1^" - "^n2)
154   | updateWithRootKeys(n2,n1,(n,k,a,m)::ent,allEnt) =
155       if n2=n then (* n2 is located and is updated *)
156         (n,k@copyRootKeys(n1,allEnt),a,m)::ent
157     else          (* find n2 in the tail *)
158       (n,k,a,m)::updateWithRootKeys(n2,n1,ent,allEnt)
159
160 (*============================================================*)
161 (* concatenate entity name to attribute name                  *)
162 fun createFkeyNames(n,[])    = []
163   | createFkeyNames(n,(kn,kt)::ks) = if (kt="auto")
164     then  (n^"_"^kn,"int")::createFkeyNames(n,ks)
165     else  (n^"_"^kn,kt)::createFkeyNames(n,ks)
166
167 (*============================================================*)
168 (* get foreign keys for n2 from n1                            *)
169 exception keysNotFound of string
170 fun ForeignKeys(n2,n1,[]) = raise keysNotFound n2
171   | ForeignKeys(n2,n1,(n,k,a,m)::ent) =
172       if n=n1 then
173       createFkeyNames(n1,k)
174     else
175       ForeignKeys(n2,n1,ent)
176 (*============================================================*)
177 (* This function finds the supporting entity *)
178 exception WeakEntityNotFound of string
179 fun updateWithWeakKeys(n2,n1,[]            ,allEnt) =
180       raise WeakEntityNotFound (n1^" - "^n2)
181   | updateWithWeakKeys(n2,n1,(n,k,a,m)::ent,allEnt) =
182       if n2=n then (* n2 is located and is updated *)
183             (* get foreign keys for n2 from n1  *)
184         (n,k@ForeignKeys(n2,n1,allEnt),a,m)::ent
185     else          (* find n2 in the tail *)
186       (n,k,a,m)::updateWithWeakKeys(n2,n1,ent,allEnt)
187
188 (*============================================================*)
189 (* Find the relationships which affect the entity             *)
190 (* This function update one entity using ALL relations        *)
191 (* returns an entity list *)
192 exception EntityNotFound of string
```

```
193 fun updateList((n,k,a,m),[]               ,ent)  = ent
194   | updateList((n,k,a,m),(n1,n2,i,m2)::rel,ent) =
195       if n=n2 then (* a relationships is pointing at n *)
196       case m2 of "isa" => updateList((n,k,a,m),rel,
197                           updateWithRootKeys(n2,n1,ent,ent))
198           | _          => updateList((n,k,a,m),rel,
199                           updateWithWeakKeys(n2,n1,ent,ent))
200     else (* update using rest of relationships *)
201       updateList((n,k,a,m),rel,ent)  (* update in the tail *)
202
203 (*============================================================*)
204 (* Collect keys from supporting entities                      *)
205 (* Updates one entity using all relationships                 *)
206 (* returns a updated Ent list *)
207 fun getKeyNames([]   ,rel,ent) = ent
208   | getKeyNames(e::we,rel,ent) =
209       getKeyNames(we,rel,updateList(e,rel,ent))
210
211 (*============================================================*)
212 (* find the keys for the given entity *)
213 exception noKeys of string
214     fun getKey(n,[])             = raise noKeys n
215      | getKey(n,(n1,k,a,m)::ls) =
216                               if n=n1 then k else getKey(n,ls)
217
218 (*============================================================*)
219 (* Only normal, e.g. neither Isa nor Weak relatonsships are   *)
220 (* converted to relations.                                    *)
221 fun rel2rel(             [],ent)= []
222   | rel2rel((e1,e2,_,m)::ls,ent)=(e1^"_rel_"^e2,
223                                   getKey(e1,ent)@getKey(e2,ent),
224                                   [],
225                                   []
226                                   )::rel2rel(ls,ent)
227
228 (*============================================================*)
229 (* copy the key from root entity to sub entity                *)
230 fun change (root,sub,[],allEnt)            = []
231   | change (root,sub,(n,k,a,m)::ls,allEnt) =
232             if sub=n then [(n,k@getKey(root,allEnt),a,m)]
233             else change(root,sub,ls,allEnt)  (* call the tail *)
234
235 (*============================================================*)
236 (* The root entity is the first key                           *)
237 (*fun copyRootKeysToSubEntities([],entities)              = []
238   | copyRootKeysToSubEntities((n1,n2,_,_)::ls,entities) =
239       change(n1,n2,entities,entities)@
240                         copyRootKeysToSubEntities(ls,entities)
241 *)
```

```
242 (*============================================================*)
243 (* the second list holds a temporary search list              *)
244 (* In the fist branch the entity is not found and hence added *)
245 (* In the second, if it's found then the keys are appended    *)
246 fun move (e,front,[])                          = e::front
247   | move ((n,k,a,m),front,(n1,k1,a1,m1)::ls) =
248               if n=n1 then front@[(a,k@k1,a,m)]@ls
249               else move((n,k,a,m),front@[(n1,k1,a1,m1)],ls)
250 (*============================================================*)
251 fun nameMember((x,_,_,_),ys) =
252                           List.exists (fn (y,_,_,_) => x=y) ys
253
254 (*============================================================*)
255 fun removeDublicates([],new)   = rev new
256   | removeDublicates(x::xs,new)= if nameMember(x,new) then
257                                    removeDublicates(xs,new)
258                                  else
259                   removeDublicates(xs,x::new)
260
261 (*============================================================*)
262 (* when merging two entities the marking is taken from the    *)
263 (* first if it isn't the empty string else, the second marking*)
264 (* is used                                                    *)
265 (* First argument : the updated list of entities              *)
266 (* Second argument: the list to compare with                  *)
267 (* third argument : the entity which is to be merged or app.  *)
268 fun update(front,[],x) = front@[x]
269   | update(front,(ne,ke,ae,me)::es,(nx,kx,ax,mx)) =
270                           (* merge the keys an the attributes*)
271       if ne = nx then
272       front@[(ne,merge(ke,kx),merge(ae,ax),me)]@es
273       else
274       update(front@[(ne,ke,ae,me)],es,(nx,kx,ax,mx))
275
276 (*============================================================*)
277 (* both lists in the agument are entities*)
278 fun removeDubEntities(reducedList,[])    = reducedList
279   | removeDubEntities(reducedList,x::xs) =
280     (* match x agaist the reducedList *)
281     removeDubEntities(update([],reducedList,x),xs)
282
283 (*============================================================*)
284 (* translate the SML type to an SQL type*)
285 fun getSQLtype t =      if t="real"   then "FLOAT"
286                   else if t="string" then "VARCHAR(30)"
287                   else if t="bool"   then "TINYINT"
288                   else t
289
290 (*============================================================*)
```

```
291 fun createAtt([])          = ""
292   | createAtt((n,t)::ls) =
293     "\n  `"^n^"` "^getSQLtype(t)^" ,"^createAtt(ls)
294 (*============================================================*)
295 fun returnNullValue t = if t = "string" then "''" else "''"
296
297 (*============================================================*)
298 (* *)
299 fun listNulls [] = ""
300   | listNulls ((_,t)::[]) = returnNullValue(t)
301   | listNulls ((_,t)::ks) = returnNullValue(t)^","^listNulls(ks)
302 (*============================================================*)
303 (* *)
304 fun listKeys ([],_) = ""
305   | listKeys ((n,_)::[],tn) = tn^""^n
306   | listKeys ((n,_)::ks,tn) = tn^""^n^",\n  "^listKeys(ks,tn)
307 (*============================================================*)
308 (* creates constrains to one table (tn)                       *)
309 fun createRelCon ((tn,ks),et) = "\n"^
310     "  INDEX ("^listKeys(ks,(tn^"_"))^" ),\n"^
311   "  FOREIGN KEY ("^listKeys(ks,(tn^"_"))^" )\n"^
312   "    REFERENCES "^tn^" ("^listKeys(ks,"")^" )\n"^
313     "      ON DELETE CASCADE\n"^
314     "      ON UPDATE CASCADE,"
315
316 (*============================================================*)
317 (* creates constrains to one table (tn)                       *)
318 fun createDisCon ((tn,ks),et) = "\n"^
319     "  INDEX ("^listKeys(ks,"")^" ),\n"^
320   "  FOREIGN KEY ("^listKeys(ks,"")^" )\n"^
321   "    REFERENCES "^tn^" ("^listKeys(ks,"")^" )\n"^
322     "      ON DELETE CASCADE\n"^
323     "      ON UPDATE CASCADE,"
324
325 (*============================================================*)
326 fun isDisj(tn,[]) = false
327   | isDisj(tn,(n,_,_,t)::et) =
328                 (tn = n andalso t="disj") orelse isDisj(tn,et)
329
330 (*============================================================*)
331 (* Creates constraints in one table from many tables *)
332 fun createCons ([],et) = ""
333   | createCons (((n,l))::[],et) = if isDisj(n,et)
334    then createDisCon((n,l),et)
335    else createRelCon((n,l),et)
336   | createCons (((n,l)::cs),et) = if isDisj(n,et)
337    then createDisCon((n,l),et)^"\n"^createCons(cs,et)
338    else createRelCon((n,l),et)^"\n"^createCons(cs,et)
339 (*============================================================*)
```

```
340 fun createChoice(n,et) =
341             if isDisj(n,et) then "\n  `choice` int ," else ""
342
343 (*============================================================*)
344 fun createDummy(n, att,et) = if isDisj(n,et)
345   then
346     "INSERT INTO `"^n^"` (\n  choice,"^listKeys(att,"")^")\n"^
347     "VALUES ('','"^listNulls(att)^");"
348   else
349     "INSERT INTO `"^n^"` (\n  "^listKeys(att,"")^")\n"^
350     "VALUES ("^listNulls(att)^");"
351
352 (*============================================================*)
353 fun cTable ((n,keys,atts,cons),et) =
354 "CREATE TABLE `"^n^"` ("^
355 createAtt(keys)^
356 createChoice(n,et)^
357 createAtt(atts)^
358 createCons(cons,et)^
359 "\n  PRIMARY KEY ("^listKeys(keys,"")^")\n) TYPE = InnoDB;\n"^
360 createDummy(n,keys@atts,et)^"\n\n";
361
362 (*============================================================*)
363 (* Each relation result in one table in the database          *)
364 fun cSchema ([],et)     = ""
365   | cSchema (r::rel,et) = cTable(r,et)^cSchema(rel,et)
```

D.1.5 php.sml

Transformering til php type repræsentation

```
(* The functions in this file generates the type declarations *)
(* and userfunction in php instead of sml                     *)
(*============================================================*)
exception phpSubTypeException
fun phpSubType (e,s) =
  case e of
      typeName x    => "$"^x
    | Ctype(e1,e2)  => phpSubType(e1,"")^","^phpSubType(e2,"")
    | Dtype(e1,e2)  => phpSubType(e1,"")^","^phpSubType(e2,"")
    | Ttype(e1,e2)  => "$"^s^"Key = array(\""^s^"Key\","^
                "\"cart\", array("^phpSubType(e1,"")^"));\n"^
                  "$"^s^"Att = array(\""^s^"Att\","^
           "\"cart\", array("^phpSubType(e2,"")^"));"
    | _             => raise phpSubTypeException

(*============================================================*)
exception phpTypeException
fun phpType (p) =
  case p of
      Typedecl(s,e)     =>
        (case e of

           typeName x    => "$"^s^"= array(\""^s^"\","^
                   "\"cart\", array("^phpSubType(e,"")^"));"
         | Ptype t       => "$"^s^"= array(\""^s^"\","^
                                          "\""^t^"\");"
         | Ltype e1      => "$"^s^"= array(\""^s^"\","^
                  "\"list\", array("^phpSubType(e1,"")^"));"
         | Ctype(e1,e2)  => "$"^s^"= array(\""^s^"\","^
                  "\"cart\",\narray("^phpSubType(e,"")^"));"
         | _             =>  raise phpTypeException
     )
    | Disjdecl(s,e)    => "$"^s^"= array(\""^s^"\","^
                    "\"disj\",\narray("^phpSubType(e,"")^"));"
    | Tabledecl(s,e)   => phpSubType(e,s)^"\n"^
                     "$"^s^"= array(\""^s^"\","^
           "\"table\",\narray($"^s^"Key, $"^s^"Att));"

(*============================================================*)
(* Add the attributes to the list one by one                  *)
(* the type of the first parsetree is added as the last       *)
(* attribute The list of attributes is reversed when returned *)
(* I use append because I need to be able to create empty pair*)
(* in addAttribute                                            *)
fun phpTypes  []     = ""
```

```
46    | phpTypes (d::dl) = phpType(d)^"\n"^phpTypes(dl)
47  (*============================================================*)
48  fun createFun (s,[])    = ""
49    | createFun (s,t::tl) =
50     "<li>\n<a href=\"index.php?action="^s^"&amp;table="^t^"\">"^
51     "\n<strong>"^t^"</strong></a></li>\n"^createFun (s,tl)
52  (*============================================================*)
53  fun phpFun [] = "LAV EN TOM MENU"
54    | phpFun t = "$usermenu='"^
55        "<h4>Opret</h4><ul>"^createFun("create",t)^"</ul>\n"^
56        "<h4>Opdater</h4><ul>"^createFun("update",t)^"</ul>\n"^
57        "<h4>Slet</h4><ul>"^createFun("delete",t)^"</ul>\n"^
58        "<h4>Vis</h4><ul>"^createFun("view",t)^"</ul>\n';"
59  (*============================================================*)
60  fun createCase (_,[],_)   = ""
61    | createCase (fName,t::tl,y)=
62    "\t\tcase \""^t^"\":\n\t\t\t$main.="^fName^"($"^t^""^
63    " ,\"\","^y^");\n\t\tbreak;\n"^createCase(fName, tl,y)
64  (*============================================================*)
65  fun phpSwitch t =
66      "if ($_REQUEST['action']==\"create\" ||\n"^
67       " $_REQUEST['action']==\"Opret\")"^
68    "{\n\tswitch($_REQUEST['table'])\n\t{\n"^
69        createCase ("insertForm",t,"$_REQUEST['action']") ^
70    "\t}\n}"^
71    "else "^
72      "if ($_REQUEST['action']==\"update\")"^
73    "{\n\tswitch($_REQUEST['table'])\n\t{\n"^
74        createCase ("updateForm",t,"\"\"") ^
75    "\t}\n}"^
76    "else "^
77      "if ($_REQUEST['action']==\"view\")"^
78    "{\n\tswitch($_REQUEST['table'])\n\t{\n"^
79        createCase ("view",t,"\"\"") ^
80    "\t}\n}"^
81    "else "^
82      "if ($_REQUEST['action']==\"delete\")"^
83    "{\n\tswitch($_REQUEST['table'])\n\t{\n"^
84        createCase ("deleteRow",t,"\"\"") ^
85    "\t}\n}"
```

D.2 Generiske funktioner

D.2.1 Hovedsiden - index.php

Brugergrænsefladen til systemet, filen inkludere alle er generiske funktioner samt de autogenererede typeerklæringer og funktioner. Se eksempel på autogenererede typeerklæringer og funktioner i Appendiks D.3 på side 148

```
1  <?php
2  require('html_functions.php');
3  require('validate_functions.php');
4  require('../../eksprojekt/sml/typeDecl.php');
5  require('../../eksprojekt/sml/userFun.php');
6  require('db_functions.php');
7  require('insert.php');
8  require('view.php');
9  require('delete.php');
10 require('update.php');
11 $main='
12 <table width="100%" cellpadding="10">
13  <tr valign="top">
14   <td>
15    <h3>Brugerfunktioner:</h3>
16   '.$usermenu.'  </td>
17   <td width="75%" bgcolor="#f0f0f0">
18    <h3>Dialog</h3>';
19 include('../../eksprojekt/sml/userSwitch.php');
20 $main.='
21 </td>
22  </tr>
23 </table>';
24 buildpage("Web-baserede systemer",$main);
25 ?>
```

D.2.2 Html funktioner - html_functions.php

```
1  <?php
2  // Functions for setting up the basic html
3  function pre($title="Web-baserede systemer"){
4  return '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
5  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
6  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="da">
7  <head>
8    <meta http-equiv="Content-Type" content="text/html;
9          charset=iso-8859-1" />
10   <script src="checkform.js" type="text/javascript"></script>
```

```
11    <title>'.$title.'</title>
12  </head>
13  <body>
14  <div>
15  <h2>'.$title.'</h2>
16  </div>
17  <hr/>
18  ';
19  }
20  function post(){
21  return '
22  </body>
23  </html>
24  ';
25  }
26  function buildpage($title,$main){
27  echo pre($title).$main.post();
28  }
29  ?>
```

D.2.3 Indsæt - insert.php

```
1  <?php
2  // One form for the whole page
3  $data=$_REQUEST['data'];
4  $value=$_REQUEST['value'];
5  $action=$_REQUEST['action'];
6
7  function insertForm($data,$value,$action)
8  {
9    // the args: datatree, value tree, start niveau
10   $value = updatetree($data,"","");
11   if ($action!="" && $action!="create")
12   {
13 //    printdata("$data",$data);   // debug functions
14 //    printdata("$value",$value); // debug functions
15     return
16     'Der æindsttes nu i databasen<hr/>'.
17     insertIntoDB($data,$value,"","");
18   }
19   else
20   {
21     $main.='
22     <form name="start" action="" method="post">
23       '.create($data,$value,"").'
24       <div>';
25     $main.='<input type="submit" name="action" value="Opret"/>
26       </div>
27     </form>';
28     return $main;
29   }
30 }
31
32 // update the values in the data tree with the corresponding
33 // posted values The update function work the following way.
34 // It runs through the structure as defined in the datatree,
35 // but it updates the values in the valuetree, also the length
36 // of the lists and the choices in the disjoint type are read
37 // from the valuetree.
38 function updatetree($data,$value,$niveau){
39 if(is_array($data))
40   switch ($data[1]) // switch on the type
41   {
42   case "table": // update each type in the table
43     for ($i=0; $i<count($data[2]);$i++){
44     $value[$i]=updatetree($data[2][$i],$value[2][$i],
45     $niveau.'-'.$i);
46     }
47     break;
```

```
48    case "cart": // update each type in the cartesian type
49      for ($i=0; $i<count($data[2]);$i++){
50      $value[$i]=updatetree($data[2][$i],$value[2][$i],
51      $niveau.'-'.$i);
52      }
53      break;
54    case "list": // update each member in the list
55      $list_length = $_REQUEST['listcount'.$niveau];
56      // loop through the elements in the list
57      for ($i=0;$i<$list_length;$i++)
58      {
59      // 1'st arg: the structure of the list element
60      // 2'nd arg: the position of the list data
61      // If just one component is notEmpty the element is added
62      if(isNotEmpty($data[2][0],$niveau.'-'.$i))
63      {
64        // renumbering
65        $value[]=updatetree($data[2][0],$value[2][$i],
66        $niveau.'-'.$i);
67      }
68      }
69      break;
70      case "disj":
71    // update the selected member and discard the others
72      if ($_REQUEST['select'.$niveau.'-0']=="")
73        $value[0]="";  // Nothing is selected in select box
74      else
75      { // the choosen item in the select box
76        $valg = $_REQUEST['select'.$niveau.'-0'];
77        // holds the selected branch in the disjoint type
78        $value[0]=$valg;
79        $value[$valg]=
80        updatetree($data[2][$valg-1],$value[2][$valg],
81                   $niveau.'-'.$valg);
82      }
83      break;
84      case "int":
85        // this array only has one element
86        $value[0]=$_REQUEST[$niveau.'-0'];
87      break;
88      case "string":
89        // this array only has one element
90        $value[0]=$_REQUEST[$niveau.'-0'];
91      break;
92      case "float":
93        // this array only has one element
94        $value[0]=$_REQUEST[$niveau.'-0'];
95      break;
96      case "bool":
```

```
97        // this array only has one element
98        $value[0]=$_REQUEST[$niveau.'-0'];
99      break;
100   }
101   return $value;
102 }
103
104 function create($data,$value,$place){
105   switch ($data[1]) // The switch is made on the 'type'
106   {
107   case "string": $choice=createSimple($data,$value,$place);break;
108   case "int"   : $choice=createSimple($data,$value,$place);break;
109   case "disj"  : $choice=createDist($data,$value,$place); break;
110   case "cart"  : $choice=createCart($data,$value,$place); break;
111   case "list"  : $choice=createList($data,$value,$place) ;break;
112   case "table" : $choice=createTable($data,$value,$place) ;break;
113   }
114   return $choice;
115 }
116
117 // when this is called, the type is simple
118 function createSimple($data,$value,$place)
119 {
120   $main='<fieldset>
121   <legend>'.$data[0].' - En simple type('.$data[1].')</legend>
122     Indtast ævrdien: <input type="text"  name="'.$place.'-0"
123                           value="'.$value[0].'"/>
124   </fieldset>';
125   return $main;
126 }
127
128 function createDist($data,$value,$place)
129 {
130   $selectbox='<select name="select'.$place.'-0"
131             onChange="document.start.submit()">';
132   if ($value[0]=="")
133     $selectbox.='
134       <option value=""> ævlg </option>'; // the selected
135   for ($i=0;$i<count($data[2]);$i++)
136     if (($i+1)==($value[0])) // $i er lig med den valgte
137       $selectbox.='
138       <option value="'.($i+1).'" selected >
139       '.$data[2][$i][0].'</option>'; // the selected one
140     else
141       $selectbox.='
142       <option value="'.($i+1).'">
143       '.$data[2][$i][0].'</option>'; // the dis-selected ones
144   $selectbox.='</select>';
145
```

```
146   $main='<fieldset><legend>'.$data[0].' - '.$selectbox.'
147    - En disjunkt type</legend>';
148
149   if ($value[0]!="")
150   {
151     $main.= create(
152     $data[2][$value[0]-1],  // The structure of the data
153     $value[$value[0]],      // The value of the data
154     $place.'-'.($value[0]));// create input field for each val.
155   }
156
157   $main.='</fieldset>';
158   return $main;
159 }
160
161 // when this is called, the type is Cart
162 function createTable($data,$value,$place)
163 {
164   $main='<fieldset>
165   <legend>'.$data[0].' - En Tabel med øngle og ævrdi</legend>';
166   for ($i=0;$i<count($data[2]);$i++)
167   {
168     // create a input field for each value
169     $main.= create($data[2][$i],$value[$i],$place.'-'.$i);
170   }
171   $main.='</fieldset>';
172   return $main;
173 }
174
175 // when this is called, the type is Cart
176 function createCart($data,$value,$place)
177 {
178   $main='<fieldset><legend>'.$data[0].' - En kartesisk produkt
179                   </legend>';
180   for ($i=0;$i<count($data[2]);$i++)
181   {
182     // create a input field for each value
183     $main.= create($data[2][$i],$value[$i],$place.'-'.$i);
184   }
185   $main.='</fieldset>';
186   return $main;
187 }
188
189 // The list type must create a sub niveau to
190 // contain the list elements
191 // when this is called, the type is a list
192 function createList($data,$value,$place)
193 {
194   if (is_array($value)=="") $antal = 1;
```

```
195   else $antal = count($value)+1;
196   $main='<fieldset>
197           <legend>
198         '.$data[0].' - En liste med '.($antal-1).' element(er)
199                                   + 1 extra tom </legend>';
200       // make one more element than there are members in the list
201       for ($i=0;$i<$antal;$i++)
202       {
203         $main.=create(
204         $data[2][0],     // the structure of the list element
205         $value[$i],      // values of previous input
206         $place.'-'.$i);  // the position in the value tree
207       }
208       $main.='
209       <div>
210         <input name="listcount'.$place.'" value="'.$antal.'"
211                type="hidden"/>
212         <input type="submit" name="insertElement"
213                value="Opdater"/>
214       </div>
215       </fieldset>';
216   return $main;
217 }
218
219 // Functions used for validating data
220 function isString()
221 {
222   return "true";
223 }
224 // Functions used for printing datastruktures
225 function printdata($title,$data)
226 {
227   echo'<fieldset><legend>'.$title.'</legend>';
228   echo'<pre>';
229   print_r($data);
230   echo'</pre></fieldset>';
231 }
232 function printdatatree($data)
233 {
234   echo '<h3>'.$data[0].'</h3><hr/>';
235   echo "<pre>";
236   print_r($data);
237   echo "</pre>";
238 }
239
240 ?>
```

D.2.4 Slet - delete.php

```
<?php
function deleteRow($data,$keyNames,$keyValues)
{
  printdata("",$datatree);
  if($val!="")
  {
    // Delete the selected entry
    ECHO "ikke tom";;
  }
  else
  {
    $num_key = count($datatree[2][0][2]);
    for ($i=0;$i<$num_key;$i++){ // number of keys
      // collect the key typeNames and types
      $keyNames[]  = $datatree[2][0][2][$i];
    }
    $sql = "SELECT * FROM `".$datatree[0]."` where
        ".$keyNames[0][0]."!='' " ;
    echo $sql;
    $result = mysql_query($sql);
    if($result)
    {
      $main.='<table cellpadding="3" cellspacing="0" border="1">
          <tr><th colspan="2">'.$datatree[0].'</th></tr>';
      $main.='<tr><td>';
      for ($i=0;$i<count($keyNames);$i++)
        $main.=$keyNames[0][$i]." ";
      $main.='</td><td></td></tr>';
      // Loop through all records
      while($row = mysql_fetch_array($result))
      {
        $keyValues="";
        $keyNames2="";
        for ($i=0;$i<count($keyNames);$i++)
        {
          $keyNames2[] = $keyNames[$i][0];    // key names
          $keyValues[]= $row[$keyNames[$i][0]]; // the values
        }
        $main.='<tr><td>';
        for ($i=0;$i<count($keyNames);$i++)
          $main.=$row[$keyNames[0][$i]].' ';
        $main.='</td><td>
              <form action="delete" method="post">
                <input type="submit" name="slet" value="Slet">
              </form>
             </td></tr>';
      }
```

```
48        $main.='</table>';
49      }
50      else
51        $main.='Tabellen er ikke oprettet i databasen';
52    }
53      return $main;
54  }
55
56  ?>
```

D.2.5 Vis - view.php

```
<?php
function view($data,$keyNames,$keyValues){
//  printdata("DATA",$data);
  switch ($data[1]) // The switch is made on the 'type'
  {
  case "string": $choice=viewSimple($data,$keyNames,$keyValues);
  break;
  case "int"   : $choice=viewSimple($data,$keyNames,$keyValues);
  break;
  case "real"  : $choice=viewSimple($data,$keyNames,$keyValues);
  break;
  case "bool"  : $choice=viewSimple($data,$keyNames,$keyValues);
  break;

  case "disj"  :   $choice=viewDisj($data,$keyNames,$keyValues);
  break;
  case "cart"  :   $choice=viewCart($data,$keyNames,$keyValues);
  break;
  case "list"  :   $choice=viewList($data,$keyNames,$keyValues);
  break;
  case "table" :  $choice=viewTable($data,$keyNames,$keyValues);
  break;
  }
  return $choice;
}

// when this is called, the type is Cart
function viewTable($datatree,$key,$val)
{
  $num_key = count($datatree[2][0][2]);
  for ($i=0;$i<$num_key;$i++){ // number of keys
    // collect the key typeNames and types
    $keyNames[]  = $datatree[2][0][2][$i];
  }
  // pass the att part to view -> viewCart
  $num_att = count($datatree[2][1][2]);
  for ($i=0;$i<$num_att;$i++){ // number of att
    // collect the att typeNames and types
    $attNames[]  = $datatree[2][1][2][$i];
  }
  if($val!=""){ // get the row corresponding to the keysvalues
  $sql = "SELECT * FROM `".$datatree[0]."`
  WHERE ".$key." = ".$val;
  }
  else{ // select all rows
  $sql = "SELECT * FROM `".$datatree[0]."`
  WHERE ".$keyNames[0][0]."!='" ;
```

```
48    }
49    $result = mysql_query($sql);
50    if($result)
51    {
52      $main.='<table cellpadding="3" cellspacing="0" border="1">
53          <tr><th colspan="2">'.$datatree[0].'</th></tr>';
54      // loop through all records
55      while($row = mysql_fetch_array($result))
56      {
57        $keyValues="";
58        $keyNames2="";
59        for ($i=0;$i<count($keyNames);$i++)
60        {
61          // key names
62          $keyNames2[] = $keyNames[$i][0];
63          // the values of the keys
64          $keyValues[]= $row[$keyNames[$i][0]];
65        }
66        // change the table name when looking up keys
67        $datatree[2][0][0]=$datatree[0];
68        // change the table name when looking up keys
69        $datatree[2][1][0]=$datatree[0];
70        // get the attributes for the tabel
71        $main.='<tr><td>'.// print the keys
72        view($datatree[2][0],$keyNames2,$keyValues).
73        '</td><td>'.// print the attributes
74        view($datatree[2][1],$keyNames2,$keyValues).
75         '</td></tr>';
76      }
77      $main.='</table>';
78    }
79    else
80      $main.='Tabellen er ikke oprettet i databasen';
81    return $main;
82  }
83
84  function searchCondition($key,$val){
85    $main.="`".$key[0]."` = '".$val[0]."'";
86    for ($i=1;$i<count($key);$i++)
87      $main.=" AND `".$key[$i]."` = '".$val[$i]."'";
88    return $main;
89  }
90
91  // when this is called, the type is Cart
92  function viewCart($datatree,$key,$val)
93  {
94    if($val!="") // get the row corresponding to the keysvalues
95    {
96    $sql = "SELECT * FROM `".$datatree[0]."`
```

```
97        WHERE ".searchCondition($key,$val);
98  //echo  $sql;
99    $result = mysql_query($sql);
100   if($result)
101   {
102     $row = mysql_fetch_array($result);
103     $conkey = concat($datatree[0],$key);
104     $main="<table border='1' width='100%'><tr>";
105 //    printdata("",$datatree);
106     for($i=0;$i<count($datatree[2]);$i++){
107       if(is_array($datatree[2][$i][2]))
108 //        $main.='<td> : '.$datatree[2][$i].'</td>';
109         $main.='<td>'.view($datatree[2][$i],$conkey,$val).'
110                 </td>';
111       else
112         $main.='<td>'.$row[$datatree[2][$i][0]].'</td>';
113     }
114     $main.="</tr></table>";
115   }
116   }
117   return $main;
118 }
119
120 // when this is called, the type is List
121 function viewList($datatree,$key,$val)
122 {
123   $conkey = concat($datatree[0],$key);
124   // It is a list of catesian products
125   $sql = "SELECT * FROM `".$datatree[2][0][0]."`
126           WHERE ".searchCondition($conkey,$val);
127
128   $result = mysql_query($sql);
129   if($result)
130   {
131     $numrow = mysql_num_rows($result);
132 //      echo '<h2>'.$sql.'</h2>';
133     $main.='<table width="100">
134     <tr>';
135     for ($i=0;$i<$numrow;$i++)
136       $main.='<th>'.$datatree[2][$i][0].'</th>';
137
138     $main.='</tr>
139     <tr><td>';
140
141     for ($i=0;$i<$numrow;$i++)
142       $main.=view(
143       $datatree[2][0],
144       array_merge($conkey,array($datatree[2][0][0]."count")),
145       array_merge($val,array($i+1))
```

```
146     );
147     $main.='</td></tr></table>';
148   }
149   return $main;
150 }
151 // when this is called, the type is Disj
152 function viewDisj($datatree,$key,$val)
153 {
154   if($val!="") // get the row corresponding to the keysvalues
155   $sql = "SELECT choice FROM `".$datatree[0]."`
156           WHERE ".searchCondition($key,$val);
157   else // Hent hele listen
158   return "Ingen key angivet til det katesiske produkt";
159 //  echo '<h2>'.$sql.'</h2>';
160 
161   $result = mysql_query($sql);
162   $row = mysql_fetch_array($result);
163   $choice = $row['choice'];
164   $main='<table>
165     <tr><th>'.$datatree[2][($choice-1)][0].'</th></tr>
166     <tr><td>'.view($datatree[2][($choice-1)],$key,$val).'</td>
167     </tr></table>';
168   return $main;
169 }
170 
171 // when this is called, the type is Simpel
172 function viewSimple($datatree,$key,$val)
173 {
174   $sql = "SELECT * FROM `".$datatree[0]."`
175           WHERE ".searchCondition($key,$val);
176 //  echo '<br/>'.$sql;
177   $main=$val[0];
178   return $main;
179 }
180 
181 ?>
```

D.2.6 Opdater - update.php

```
1  <?php
2  // One form for the whole page
3  $data=$_REQUEST['data'];
4  $value=$_REQUEST['value'];
5  $action=$_REQUEST['action'];
6  
7  function insertForm($data,$value,$action)
8  {
9    // the args: datatree, value tree, start niveau
10   $value = updatetree($data,"","");
```

```
11    if ($action!="" && $action!="create")
12    {
13  //    printdata("$data",$data);   // debug functions
14  //    printdata("$value",$value); // debug functions
15      return
16      'Der æindsttes nu i databasen<hr/>'.
17      insertIntoDB($data,$value,"","");
18    }
19    else
20    {
21      $main.='
22      <form name="start" action="" method="post">
23        '.create($data,$value,"").'
24        <div>';
25      $main.='<input type="submit" name="action" value="Opret"/>
26        </div>
27      </form>';
28      return $main;
29    }
30  }
31
32  // update the values in the data tree with the corresponding
33  // posted values The update function work the following way.
34  // It runs through the structure as defined in the datatree,
35  // but it updates the values in the valuetree, also the length
36  // of the lists and the choices in the disjoint type are read
37  // from the valuetree.
38  function updatetree($data,$value,$niveau){
39  if(is_array($data))
40    switch ($data[1]) // switch on the type
41    {
42    case "table": // update each type in the table
43      for ($i=0; $i<count($data[2]);$i++){
44      $value[$i]=updatetree($data[2][$i],$value[2][$i],
45      $niveau.'-'.$i);
46      }
47      break;
48    case "cart": // update each type in the cartesian type
49      for ($i=0; $i<count($data[2]);$i++){
50      $value[$i]=updatetree($data[2][$i],$value[2][$i],
51      $niveau.'-'.$i);
52      }
53      break;
54    case "list": // update each member in the list
55      $list_length = $_REQUEST['listcount'.$niveau];
56      // loop through the elements in the list
57      for ($i=0;$i<$list_length;$i++)
58      {
59      // 1'st arg: the structure of the list element
```

```
60      // 2'nd arg: the position of the list data
61      // If just one component is notEmpty the element is added
62      if(isNotEmpty($data[2][0],$niveau.'-'.$i))
63      {
64        // renumbering
65        $value[]=updatetree($data[2][0],$value[2][$i],
66        $niveau.'-'.$i);
67      }
68      }
69      break;
70      case "disj":
71    // update the selected member and discard the others
72      if ($_REQUEST['select'.$niveau.'-0']=="")
73        $value[0]="";  // Nothing is selected in select box
74      else
75      { // the choosen item in the select box
76        $valg = $_REQUEST['select'.$niveau.'-0'];
77        // holds the selected branch in the disjoint type
78        $value[0]=$valg;
79        $value[$valg]=
80        updatetree($data[2][$valg-1],$value[2][$valg],
81                   $niveau.'-'.$valg);
82      }
83      break;
84      case "int":
85        // this array only has one element
86        $value[0]=$_REQUEST[$niveau.'-0'];
87      break;
88      case "string":
89        // this array only has one element
90        $value[0]=$_REQUEST[$niveau.'-0'];
91      break;
92      case "float":
93        // this array only has one element
94        $value[0]=$_REQUEST[$niveau.'-0'];
95      break;
96      case "bool":
97        // this array only has one element
98        $value[0]=$_REQUEST[$niveau.'-0'];
99      break;
100   }
101   return $value;
102 }
103
104 function create($data,$value,$place){
105   switch ($data[1]) // The switch is made on the 'type'
106   {
107   case "string": $choice=createSimple($data,$value,$place);break;
108   case "int"   : $choice=createSimple($data,$value,$place);break;
```

```
109   case "disj"  : $choice=createDist($data,$value,$place); break;
110   case "cart"  : $choice=createCart($data,$value,$place); break;
111   case "list"  : $choice=createList($data,$value,$place) ;break;
112   case "table" : $choice=createTable($data,$value,$place) ;break;
113   }
114   return $choice;
115 }
116
117 // when this is called, the type is simple
118 function createSimple($data,$value,$place)
119 {
120   $main='<fieldset>
121   <legend>'.$data[0].' - En simple type('.$data[1].')</legend>
122     Indtast ævrdien: <input type="text"  name="'.$place.'-0"
123                           value="'.$value[0].'"/>
124   </fieldset>';
125   return $main;
126 }
127
128 function createDist($data,$value,$place)
129 {
130   $selectbox='<select name="select'.$place.'-0"
131             onChange="document.start.submit()">';
132   if ($value[0]=="")
133     $selectbox.='
134       <option value=""> ævlg </option>'; // the selected
135   for ($i=0;$i<count($data[2]);$i++)
136     if (($i+1)==($value[0])) // $i er lig med den valgte
137       $selectbox.='
138       <option value="'.($i+1).'" selected >
139       '.$data[2][$i][0].'</option>'; // the selected one
140     else
141       $selectbox.='
142       <option value="'.($i+1).'">
143       '.$data[2][$i][0].'</option>'; // the dis-selected ones
144   $selectbox.='</select>';
145
146   $main='<fieldset><legend>'.$data[0].' - '.$selectbox.'
147    - En disjunkt type</legend>';
148
149   if ($value[0]!="")
150   {
151     $main.= create(
152     $data[2][$value[0]-1],  // The structure of the data
153     $value[$value[0]],      // The value of the data
154     $place.'-'.($value[0]));// create input field for each val.
155   }
156
157   $main.='</fieldset>';
```

```
158   return $main;
159 }
160
161 // when this is called, the type is Cart
162 function createTable($data,$value,$place)
163 {
164   $main='<fieldset>
165   <legend>'.$data[0].' - En Tabel med øngle og ævrdi</legend>';
166   for ($i=0;$i<count($data[2]);$i++)
167   {
168     // create a input field for each value
169     $main.= create($data[2][$i],$value[$i],$place.'-'.$i);
170   }
171   $main.='</fieldset>';
172   return $main;
173 }
174
175 // when this is called, the type is Cart
176 function createCart($data,$value,$place)
177 {
178   $main='<fieldset><legend>'.$data[0].' - En kartesisk produkt
179                   </legend>';
180   for ($i=0;$i<count($data[2]);$i++)
181   {
182     // create a input field for each value
183     $main.= create($data[2][$i],$value[$i],$place.'-'.$i);
184   }
185   $main.='</fieldset>';
186   return $main;
187 }
188
189 // The list type must create a sub niveau to
190 // contain the list elements
191 // when this is called, the type is a list
192 function createList($data,$value,$place)
193 {
194   if (is_array($value)=="") $antal = 1;
195   else $antal = count($value)+1;
196   $main='<fieldset>
197           <legend>
198         '.$data[0].' - En liste med '.($antal-1).' element(er)
199                                   + 1 extra tom </legend>';
200       // make one more element than there are members in the list
201       for ($i=0;$i<$antal;$i++)
202       {
203         $main.=create(
204         $data[2][0],     // the structure of the list element
205         $value[$i],      // values of previous input
206         $place.'-'.$i);  // the position in the value tree
```

```
207       }
208       $main.='
209       <div>
210         <input name="listcount'.$place.'" value="'.$antal.'"
211                type="hidden"/>
212         <input type="submit" name="insertElement"
213                value="Opdater"/>
214       </div>
215       </fieldset>';
216   return $main;
217 }
218
219 // Functions used for validating data
220 function isString()
221 {
222   return "true";
223 }
224 // Functions used for printing datastruktures
225 function printdata($title,$data)
226 {
227   echo'<fieldset><legend>'.$title.'</legend>';
228   echo'<pre>';
229   print_r($data);
230   echo'</pre></fieldset>';
231 }
232 function printdatatree($data)
233 {
234   echo '<h3>'.$data[0].'</h3><hr/>';
235   echo "<pre>";
236   print_r($data);
237   echo "</pre>";
238 }
239
240 ?>
```

D.2.7 Validerings funktioner

```
1  <?php
2  // niveau contains list index information
3  function isNotEmpty($data,$niveau)
4  {
5  //      echo '<h1>'.$data[0].'-----'.$data[1].'</h1>';
6    switch ($data[1]){ // the type of the data
7      case "int":
8        if ($_REQUEST[$niveau.'-0']!="")
9          return true;
10       break;
11
12     case "string":
```

```
13        if ($_REQUEST[$niveau.'-0']!="")
14          return true;
15        break;
16
17      case "cart":
18        for($i=0;$i<count($data[2]);$i++){
19          // is each of the members filled correct?
20          if(isNotEmpty($data[2][$i],$niveau.'-'.$i))
21            return true; // one was not empty
22        }
23        return false;        // they were all empty
24
25      case "disj":
26        if($_REQUEST['select'.$niveau.'-0']!="")
27          return true;  // one was not empty
28        else
29          return false;   // they were all empty
30        break;
31    }
32  }
33
34  ?>
```

```
1   function sendform(){
2     if (document.all) {
3       document.all.start.action.value="";
4       document.all.start.submit();
5     }
6     else{
7       document.start.action="";
8       document.start.submit();
9     }
10  }
11
12  function isInt(input, field)
13  {
14    var re = /^[-]?[0-9]*$/;
15    if (input.match(re) == null)
16    {
17      alert ("\"" + input + "\" er ikke et heltal");
18      var x = document.forms.check
19      x[field].focus();
20      document.check.IntFelt.focus();
21      document.check.IntFelt.value="FUCK YOU";
22      return false;
23    }
24    return true;
25  }
26
```

```
27 function isString(input, field)
28 {
29   var re = /^[a-zA-Z0-9]*$/;
30   if (input.match(re) == null)
31   {
32     alert ("\"" + input + "\" er ikke en streng");
33     var x = document.forms.check
34     x[field].focus();
35     return false;
36   }
37   return true;
38 }
39
40 function isDec(input, field)
41 {
42   var re = /^([-]?[0-9]*(,[0-9]+)+)*$/;
43   if (input.match(re) == null)
44   {
45     alert ("\"" + input + "\" er ikke et decimaltal");
46     var x = document.forms.check
47     x[field].focus();
48     return false;
49   }
50   return true;
51 }
```

D.2.8 Database funktioner - db_functions.php

```
1  <?php
2  // set up the connection to the database
3  $uname = "xxxxxxxx";      // mysql username
4  $pword = "xxxxxxxx";    // mysql password
5  $hname = "localhost";  // mysql hostname
6  //$dbase = "tprog";      // mysql database
7  $dbase = "impeks";      // mysql database
8  // create connection to the database
9  $con = mysql_connect($hname,$uname,$pword);
10 mysql_select_db($dbase);
11
12 // database auxillery functions
13 // $value is considered valid at this point
14 // insert the data into the database and return an id
15 function insertIntoDB($datatree,$value,$fkeys,$fvals)
16 {
17   //$datatree : The strukture of the data
18   //$valuetree: All data entered in the form
19   //$fkeys    : Names of foreign keys
20   //$fvals    : Values of foreign keys
21
```

```
22    //printdata("value",$value);
23    //printdata("datatree",$datatree);
24    // When inserting into the database, the strukture can be one
25    // of four kinds, {cart,disj,table or list} also data can be
26    // af root element without a key. In the latter case(which
27    // only applyes to cart,disj and list), a key must be
28    // genereted automatically.
29    switch($datatree[1]) // swith on the type
30    {
31      case "table":
32        $main.="<p><u>OPRET TABLE:</u></p>";
33        // both keys and attributes are provided for tables.
34        // Get the keys entered in the form
35        // number of keys
36        for ($i=0;$i<count($datatree[2][0][2]);$i++)
37        {
38          // collect the key typeNames
39          $keyNames[]  = $datatree[2][0][2][$i][0];
40          // collect the value of the simple type
41          $keyValues[] = $value[0][$i][0];
42        }
43        // Subtypes are split into simple types and complex types
44        // number of subtypes
45        for ($i=0;$i<count($datatree[2][1][2]);$i++)
46        {
47          // look at the second component to determine if it is
48          // simple or complex
49          if (is_array($datatree[2][1][2][$i][2])) // it's complex
50          {
51            // collect the complex types
52            $complex[] = $datatree[2][1][2][$i];
53            // collect the complex values
54            $cvalues[] = $value[1][$i];
55          }
56          else // collect the attributes
57          {
58            // collect the simple typeNames
59            $typeNames[]  = $datatree[2][1][2][$i][0];
60            // collect the value of the simple type
61            $svalues[]    = $value[1][$i][0];
62          }
63        }
64        $query = 'INSERT INTO ``.$datatree[0].``
65        ('.implode(",", array_merge($keyNames,$typeNames)).')
66                  VALUES
67        (\''.implode("','", array_merge($keyValues, $svalues))
68        .'\');';
69        // Errorhandling are handled in performQuery
70        $main.= performQuery($datatree[0],$query);
```

```
71
72        //Put the name of the table in front of all key names
73  //      echo "<br/>keynames : ".$keyNames[0];
74        $keyNames = concat($datatree[0],$keyNames);
75
76  //      echo "<br/>keynames : ".$keyNames[0];
77
78        //create the sub entries in the database.
79        if(count($complex)>0)
80        {
81          for ($i=0; $i<count($complex);$i++){
82            $main.=insertIntoDB(
83                   $complex[$i],  // datatree
84                   $cvalues[$i],  // valuetree
85                   $keyNames,   // KeyTypeNames
86                   $keyValues   // KeyValues
87                   );
88          }
89        }
90        return $main;
91
92        break;
93      case "cart":
94        $main.="<p><u>OPRET CART:</u></p>";
95        if (count($fkeys)==0)
96          $main.="DEBUG: Ingen øngler
97                                  fundet til kartesiske produkt";
98        else
99            $main.="DEBUG: <u>".count($fkeys)."</u>
100                  fundet til kartesiske produkt";
101       $keyNames  = $fkeys;
102       $keyValues = $fvals;
103
104       // both keys and attributes are provided for tables.
105       // Get the keys entered in the form
106       for ($i=0;$i<count($datatree[2]);$i++) //# subtypes
107       {
108         // look at the second component to determine if
109         // it is simple or complex
110         if (is_array($datatree[2][$i][2])) // it's complex
111         {
112           // collect the complex types
113           $complex[] = $datatree[2][$i];
114           // collect the complex values
115           $cvalues[] = $value[$i];
116         }
117         else // collect the attributes (it's simple)
118         {
119           // collect the simple typeNames
```

```
120             $typeNames[]  = $datatree[2][$i][0];
121             // collect the value of the simple type
122             $svalues[]    = $value[$i][0];
123           }
124         }
125         // A CARTESIAN INSERTION
126         $query = 'INSERT INTO `'.$datatree[0].'`
127         ('.implode(",", array_merge($keyNames,$typeNames)).')
128                 VALUES
129         (\''.implode("','", array_merge($keyValues, $svalues)).
130         '\');';
131         // Errorhandling are handled in performQuery
132         $main.= performQuery($datatree[0],$query);
133
134         //Put the name of the table in front of all key names
135 //        echo "<br/>keynames : ".$keyNames[0];
136         $keyNames = concat($datatree[0],$keyNames);
137
138 //        echo "<br/>keynames : ".$keyNames[0];
139
140         //create the sub entries in the database.
141         if(count($complex)>0)
142         {
143           for ($i=0; $i<count($complex);$i++){
144             $main.=insertIntoDB(
145                    $complex[$i],  // datatree
146                    $cvalues[$i],  // valuetree
147                    $keyNames,   // KeyTypeNames
148                    $keyValues   // KeyValues
149                    );
150           }
151         }
152         return $main;
153           break;
154       case "list":
155         $main.="<p><u>OPRET LIST:</u></p>";
156         // This branch creates the list entry and calls
157         // InsertIntoDB with all list elements
158         if (count($fkeys)==0)
159           $main.="DEBUG: Der er <u>ikke</u> fundet
160               nogen øngle til Listen";
161         else
162             $main.="DEBUG: Der er fundet ".count($fkeys)."ø
163               ngler til listen";
164
165         $keyNames  = $fkeys;
166         $keyValues = $fvals;
167
168         $query = 'INSERT INTO `'.$datatree[0].'`
```

```
169           ('.implode(",", $keyNames).')
170                 VALUES (\''.implode("','",$keyValues).'\');';
171
172       // Errorhandling are handled in performQuery
173       $main.= performQuery($datatree[0],$query);
174
175       $keyNames = concat($datatree[0],$keyNames);
176
177       $main.=" Listen har : ".count($value)." elements";
178       // insert each of the listelements into the database
179       for ($i=0;$i<count($value);$i++)
180       {
181       $main.="<p>DEBUG: Den valgte subtype er : ".($i+1)."</p>";
182       $main.=insertIntoDB(
183         $datatree[2][0], // datatree
184         $value[$i],   // valuetree
185         array_merge($keyNames,// KeyTypeNames
186                     array($datatree[2][0][0]."count")),
187         array_merge($keyValues,// KeyValues
188               array(($i+1)))
189         );
190       }
191
192
193       return $main;
194         break;
195     case "disj":
196       // The parameters for the function
197         // function insertIntoDB($datatree,$value,$fkeys,$fvals)
198       $main.="<p><u>OPRET DISJ:</u></p>";
199       if (count($fkeys)==0)
200         $main.="DEBUG: Der er ikke fundet nogen øngle til
201                 den disjunkte forening";
202       else
203           $main.="DEBUG: Der <u>ER</u> fundet ".count($fkeys).
204                 " øngler til den disjunkte forening";
205
206       $keyNames  = $fkeys;
207       $keyValues = $fvals;
208       // the only value which is needed in the disjoint case
209       // is the choice
210       $choice = $value[0][0]; // the selected subtype
211       $main.="<p>DEBUG: Den valgte subtype er : ".$choice."</p>";
212       $query = 'INSERT INTO `'.$datatree[0].'`
213       ('.implode(",", array_merge($keyNames,
214                   array("choice"))).')
215                 VALUES
216       (\''.implode("','", array_merge($keyValues,
217                   array($choice))).'\');';
```

```
218
219       // Errorhandling are handled in performQuery
220       $main.= performQuery($datatree[0],$query);
221
222       //create the sub entries in the database.
223       $main.=insertIntoDB(
224          $datatree[2][($choice-1)], // datatree
225          $value[$choice],     // valuetree
226          $keyNames,         // KeyTypeNames
227          $keyValues         // KeyValues
228       );
229       printdatatree($keyNames);
230       printdatatree($keyValues);
231       return $main;
232         break;
233   }
234 }
235
236
237
238 function performQuery($tableName,$query){
239   $result = mysql_query($query);
240     if (mysql_error())
241     return "øFlgende fejl opstod ved kommunikation med databasen:
242      <br/><b>".mysql_error()."</b><br/>ø
243             Foresprgslen var: ".$query;
244     else return "Post korrekt indsat i ".$tableName;
245 }
246
247 // Concatenate $a on all elements in $b
248 function concat($a,$b)
249 {
250   for ($i=0;$i<count($b);$i++)
251     $b[$i]=$a."_".$b[$i];
252   return $b;
253 }
254 ?>
```

D.3 Træningsprogrammer

De filer der bliver når signaturen med træningsprogrammer transformeres.

D.3.1 Signaturen

```
signature tprog =
sig
 type beskrivelse = string
 type tid         = int
 type afstand     = int
 type pause       = int
 type tempo       = int
 type sted        = string
 type formaal     = string
 type kommentar   = string

 disjoint basal   = afstand | tid
 type serie       = basal list
 disjoint session = basal | serie

 type distance    = afstand * pause * tempo
 type interval    = tid * pause * tempo

 type distanceT   = distance list
 type intervalT   = interval list
 type udholdT     = session list
 type teknikT     = basal * tempo
 type test        =
              basal * sted * formaal * kommentar * beskrivelse

 disjoint program =
              intervalT | distanceT | udholdT | teknikT | test
 table traening   = (beskrivelse, program)

end
```

D.3.2 PHP typerepræsentationen

```
<?php
$beskrivelse= array("beskrivelse","string");
$tid= array("tid","int");
$afstand= array("afstand","int");
$pause= array("pause","int");
$tempo= array("tempo","int");
$sted= array("sted","string");
$formaal= array("formaal","string");
```

```
$kommentar= array("kommentar","string");
$basal= array("basal","disj",
array($afstand,$tid));
$serie= array("serie","list", array($basal));
$session= array("session","disj",
array($basal,$serie));
$distance= array("distance","cart",
array($afstand,$pause,$tempo));
$interval= array("interval","cart",
array($tid,$pause,$tempo));
$distanceT= array("distanceT","list", array($distance));
$intervalT= array("intervalT","list", array($interval));
$udholdT= array("udholdT","list", array($session));
$teknikT= array("teknikT","cart",
array($basal,$tempo));
$test= array("test","cart",
array($basal,$sted,$formaal,$kommentar,$beskrivelse));
$program= array("program","disj",
array($intervalT,$distanceT,$udholdT,$teknikT,$test));
$traeningKey = array("traeningKey","cart", array($beskrivelse));
$traeningAtt = array("traeningAtt","cart", array($program));
$traening= array("traening","table",
array($traeningKey, $traeningAtt));

?>
```

D.3.3 PHP brugerfunktioner

```
<?php
$usermenu='<h4>Opret</h4><ul><li>
<a href="index.php?action=create&amp;table=traening">
<strong>traening</strong></a></li>
</ul>
<h4>Opdater</h4><ul><li>
<a href="index.php?action=update&amp;table=traening">
<strong>traening</strong></a></li>
</ul>
<h4>Slet</h4><ul><li>
<a href="index.php?action=delete&amp;table=traening">
<strong>traening</strong></a></li>
</ul>
<h4>Vis</h4><ul><li>
<a href="index.php?action=view&amp;table=traening">
<strong>traening</strong></a></li>
</ul>
';
?>
```

```
<?php
```

```
2  if ($_REQUEST['action']=="create" ||
3   $_REQUEST['action']=="Opret"){
4    switch($_REQUEST['table'])
5    {
6      case "traening":
7        $main.=insertForm($traening ,"",$_REQUEST['action']);
8      break;
9    }
10 }else if ($_REQUEST['action']=="update"){
11   switch($_REQUEST['table'])
12   {
13     case "traening":
14       $main.=updateForm($traening ,"","");
15     break;
16   }
17 }else if ($_REQUEST['action']=="view"){
18   switch($_REQUEST['table'])
19   {
20     case "traening":
21       $main.=view($traening ,"","");
22     break;
23   }
24 }else if ($_REQUEST['action']=="delete"){
25   switch($_REQUEST['table'])
26   {
27     case "traening":
28       $main.=deleteRow($traening ,"","");
29     break;
30   }
31 }
32 ?>
```

D.3.4 Database skemaet

```
1  CREATE TABLE `traening` (
2    `beskrivelse` VARCHAR(30) ,
3    PRIMARY KEY (beskrivelse)
4  ) TYPE = InnoDB;
5  INSERT INTO `traening` (
6    beskrivelse)
7  VALUES ('');
8
9  CREATE TABLE `program` (
10   `traening_beskrivelse` VARCHAR(30) ,
11   `choice` int ,
12   INDEX (traening_beskrivelse ),
13   FOREIGN KEY (traening_beskrivelse )
14     REFERENCES traening (beskrivelse )
15       ON DELETE CASCADE
```

```
16        ON UPDATE CASCADE,
17    PRIMARY KEY (traening_beskrivelse)
18  ) TYPE = InnoDB;
19  INSERT INTO `program` (
20    choice,traening_beskrivelse)
21  VALUES ('','');
22
23  CREATE TABLE `test` (
24    `traening_beskrivelse` VARCHAR(30) ,
25    `beskrivelse` VARCHAR(30) ,
26    `kommentar` VARCHAR(30) ,
27    `formaal` VARCHAR(30) ,
28    `sted` VARCHAR(30) ,
29    INDEX (traening_beskrivelse ),
30    FOREIGN KEY (traening_beskrivelse )
31      REFERENCES program (traening_beskrivelse )
32        ON DELETE CASCADE
33        ON UPDATE CASCADE,
34    PRIMARY KEY (traening_beskrivelse)
35  ) TYPE = InnoDB;
36  INSERT INTO `test` (
37    traening_beskrivelse,
38    beskrivelse,
39    kommentar,
40    formaal,
41    sted)
42  VALUES ('','','','','');
43
44  CREATE TABLE `teknikT` (
45    `traening_beskrivelse` VARCHAR(30) ,
46    `tempo` int ,
47    INDEX (traening_beskrivelse ),
48    FOREIGN KEY (traening_beskrivelse )
49      REFERENCES program (traening_beskrivelse )
50        ON DELETE CASCADE
51        ON UPDATE CASCADE,
52    PRIMARY KEY (traening_beskrivelse)
53  ) TYPE = InnoDB;
54  INSERT INTO `teknikT` (
55    traening_beskrivelse,
56    tempo)
57  VALUES ('','');
58
59  CREATE TABLE `udholdT` (
60    `traening_beskrivelse` VARCHAR(30) ,
61    INDEX (traening_beskrivelse ),
62    FOREIGN KEY (traening_beskrivelse )
63      REFERENCES program (traening_beskrivelse )
64        ON DELETE CASCADE
```

```
65        ON UPDATE CASCADE,
66    PRIMARY KEY (traening_beskrivelse)
67  ) TYPE = InnoDB;
68  INSERT INTO `udholdT` (
69    traening_beskrivelse)
70  VALUES ('');
71
72  CREATE TABLE `intervalT` (
73    `traening_beskrivelse` VARCHAR(30) ,
74    INDEX (traening_beskrivelse ),
75    FOREIGN KEY (traening_beskrivelse )
76      REFERENCES program (traening_beskrivelse )
77        ON DELETE CASCADE
78        ON UPDATE CASCADE,
79    PRIMARY KEY (traening_beskrivelse)
80  ) TYPE = InnoDB;
81  INSERT INTO `intervalT` (
82    traening_beskrivelse)
83  VALUES ('');
84
85  CREATE TABLE `distanceT` (
86    `traening_beskrivelse` VARCHAR(30) ,
87    INDEX (traening_beskrivelse ),
88    FOREIGN KEY (traening_beskrivelse )
89      REFERENCES program (traening_beskrivelse )
90        ON DELETE CASCADE
91        ON UPDATE CASCADE,
92    PRIMARY KEY (traening_beskrivelse)
93  ) TYPE = InnoDB;
94  INSERT INTO `distanceT` (
95    traening_beskrivelse)
96  VALUES ('');
97
98  CREATE TABLE `interval` (
99    `intervalcount` int ,
100   `intervalT_traening_beskrivelse` VARCHAR(30) ,
101   `tempo` int ,
102   `pause` int ,
103   `tid` int ,
104   INDEX (intervalT_traening_beskrivelse ),
105   FOREIGN KEY (intervalT_traening_beskrivelse )
106     REFERENCES intervalT (traening_beskrivelse )
107       ON DELETE CASCADE
108       ON UPDATE CASCADE,
109   PRIMARY KEY (intervalcount,
110   intervalT_traening_beskrivelse)
111 ) TYPE = InnoDB;
112 INSERT INTO `interval` (
113   intervalcount,
```

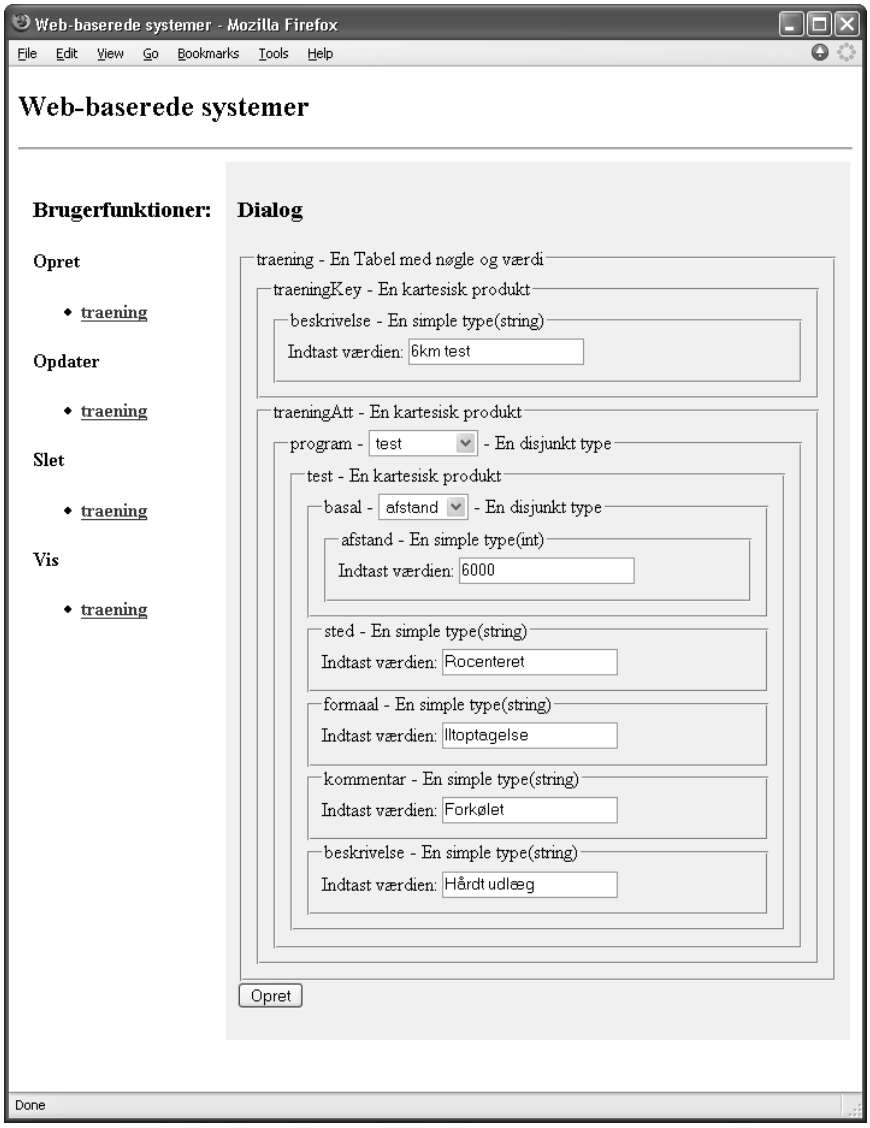
```
114   intervalT_traening_beskrivelse,
115   tempo,
116   pause,
117   tid)
118 VALUES ('','','','','');
119
120 CREATE TABLE `distance` (
121   `distancecount` int ,
122   `distanceT_traening_beskrivelse` VARCHAR(30) ,
123   `tempo` int ,
124   `pause` int ,
125   `afstand` int ,
126   INDEX (distanceT_traening_beskrivelse ),
127   FOREIGN KEY (distanceT_traening_beskrivelse )
128     REFERENCES distanceT (traening_beskrivelse )
129       ON DELETE CASCADE
130       ON UPDATE CASCADE,
131   PRIMARY KEY (distancecount,
132   distanceT_traening_beskrivelse)
133 ) TYPE = InnoDB;
134 INSERT INTO `distance` (
135   distancecount,
136   distanceT_traening_beskrivelse,
137   tempo,
138   pause,
139   afstand)
140 VALUES ('','','','','');
141
142 CREATE TABLE `session` (
143   `sessioncount` int ,
144   `udholdT_traening_beskrivelse` VARCHAR(30) ,
145   `choice` int ,
146   INDEX (udholdT_traening_beskrivelse ),
147   FOREIGN KEY (udholdT_traening_beskrivelse )
148     REFERENCES udholdT (traening_beskrivelse )
149       ON DELETE CASCADE
150       ON UPDATE CASCADE,
151   PRIMARY KEY (sessioncount,
152   udholdT_traening_beskrivelse)
153 ) TYPE = InnoDB;
154 INSERT INTO `session` (
155   choice,sessioncount,
156   udholdT_traening_beskrivelse)
157 VALUES ('','','');
158
159 CREATE TABLE `serie` (
160   `sessioncount` int ,
161   `udholdT_traening_beskrivelse` VARCHAR(30) ,
162   INDEX (sessioncount,
```

```
163   udholdT_traening_beskrivelse ),
164   FOREIGN KEY (sessioncount,
165   udholdT_traening_beskrivelse )
166     REFERENCES session (sessioncount,
167   udholdT_traening_beskrivelse )
168       ON DELETE CASCADE
169       ON UPDATE CASCADE,
170   PRIMARY KEY (sessioncount,
171   udholdT_traening_beskrivelse)
172 ) TYPE = InnoDB;
173 INSERT INTO `serie` (
174   sessioncount,
175   udholdT_traening_beskrivelse)
176 VALUES ('','');
177
178 CREATE TABLE `basal` (
179   `basalcount` int ,
180   `serie_sessioncount` int ,
181   `serie_udholdT_traening_beskrivelse` VARCHAR(30) ,
182   `teknikT_traening_beskrivelse` VARCHAR(30) ,
183   `test_traening_beskrivelse` VARCHAR(30) ,
184   `sessioncount` int ,
185   `udholdT_traening_beskrivelse` VARCHAR(30) ,
186   `choice` int ,
187   INDEX (serie_sessioncount,
188   serie_udholdT_traening_beskrivelse ),
189   FOREIGN KEY (serie_sessioncount,
190   serie_udholdT_traening_beskrivelse )
191     REFERENCES serie (sessioncount,
192   udholdT_traening_beskrivelse )
193       ON DELETE CASCADE
194       ON UPDATE CASCADE,
195
196   INDEX (teknikT_traening_beskrivelse ),
197   FOREIGN KEY (teknikT_traening_beskrivelse )
198     REFERENCES teknikT (traening_beskrivelse )
199       ON DELETE CASCADE
200       ON UPDATE CASCADE,
201
202   INDEX (test_traening_beskrivelse ),
203   FOREIGN KEY (test_traening_beskrivelse )
204     REFERENCES test (traening_beskrivelse )
205       ON DELETE CASCADE
206       ON UPDATE CASCADE,
207
208   INDEX (sessioncount,
209   udholdT_traening_beskrivelse ),
210   FOREIGN KEY (sessioncount,
211   udholdT_traening_beskrivelse )
```

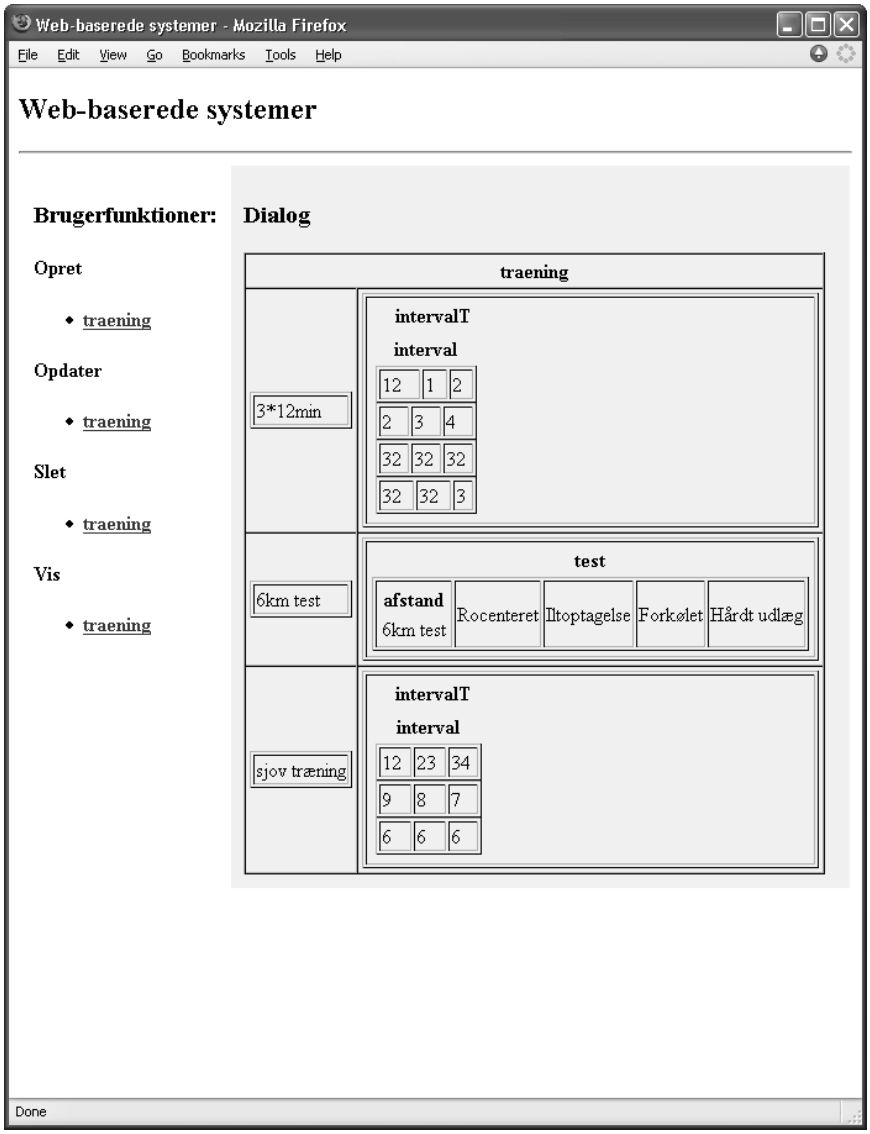
```
212     REFERENCES session (sessioncount,
213   udholdT_traening_beskrivelse )
214       ON DELETE CASCADE
215       ON UPDATE CASCADE,
216   PRIMARY KEY (basalcount,
217   serie_sessioncount,
218   serie_udholdT_traening_beskrivelse,
219   teknikT_traening_beskrivelse,
220   test_traening_beskrivelse,
221   sessioncount,
222   udholdT_traening_beskrivelse)
223 ) TYPE = InnoDB;
224 INSERT INTO `basal` (
225   choice,basalcount,
226   serie_sessioncount,
227   serie_udholdT_traening_beskrivelse,
228   teknikT_traening_beskrivelse,
229   test_traening_beskrivelse,
230   sessioncount,
231   udholdT_traening_beskrivelse)
232 VALUES ('','','','','','','','');
233
234 CREATE TABLE `basal_afstand` (
235   `basalcount` int ,
236   `serie_sessioncount` int ,
237   `serie_udholdT_traening_beskrivelse` VARCHAR(30) ,
238   `teknikT_traening_beskrivelse` VARCHAR(30) ,
239   `test_traening_beskrivelse` VARCHAR(30) ,
240   `sessioncount` int ,
241   `udholdT_traening_beskrivelse` VARCHAR(30) ,
242   `afstand` int ,
243   INDEX (basalcount,
244   serie_sessioncount,
245   serie_udholdT_traening_beskrivelse,
246   teknikT_traening_beskrivelse,
247   test_traening_beskrivelse,
248   sessioncount,
249   udholdT_traening_beskrivelse ),
250   FOREIGN KEY (basalcount,
251   serie_sessioncount,
252   serie_udholdT_traening_beskrivelse,
253   teknikT_traening_beskrivelse,
254   test_traening_beskrivelse,
255   sessioncount,
256   udholdT_traening_beskrivelse )
257     REFERENCES basal (basalcount,
258   serie_sessioncount,
259   serie_udholdT_traening_beskrivelse,
260   teknikT_traening_beskrivelse,
```

```
261   test_traening_beskrivelse,
262   sessioncount,
263   udholdT_traening_beskrivelse )
264       ON DELETE CASCADE
265       ON UPDATE CASCADE,
266   PRIMARY KEY (basalcount,
267   serie_sessioncount,
268   serie_udholdT_traening_beskrivelse,
269   teknikT_traening_beskrivelse,
270   test_traening_beskrivelse,
271   sessioncount,
272   udholdT_traening_beskrivelse)
273 ) TYPE = InnoDB;
274 INSERT INTO `basal_afstand` (
275   basalcount,
276   serie_sessioncount,
277   serie_udholdT_traening_beskrivelse,
278   teknikT_traening_beskrivelse,
279   test_traening_beskrivelse,
280   sessioncount,
281   udholdT_traening_beskrivelse,
282   afstand)
283 VALUES ('','','','','','','','');
284
285 CREATE TABLE `basal_tid` (
286   `basalcount` int ,
287   `serie_sessioncount` int ,
288   `serie_udholdT_traening_beskrivelse` VARCHAR(30) ,
289   `teknikT_traening_beskrivelse` VARCHAR(30) ,
290   `test_traening_beskrivelse` VARCHAR(30) ,
291   `sessioncount` int ,
292   `udholdT_traening_beskrivelse` VARCHAR(30) ,
293   `tid` int ,
294   INDEX (basalcount,
295   serie_sessioncount,
296   serie_udholdT_traening_beskrivelse,
297   teknikT_traening_beskrivelse,
298   test_traening_beskrivelse,
299   sessioncount,
300   udholdT_traening_beskrivelse ),
301   FOREIGN KEY (basalcount,
302   serie_sessioncount,
303   serie_udholdT_traening_beskrivelse,
304   teknikT_traening_beskrivelse,
305   test_traening_beskrivelse,
306   sessioncount,
307   udholdT_traening_beskrivelse )
308     REFERENCES basal (basalcount,
309   serie_sessioncount,
```

```
310   serie_udholdT_traening_beskrivelse,
311   teknikT_traening_beskrivelse,
312   test_traening_beskrivelse,
313   sessioncount,
314   udholdT_traening_beskrivelse )
315       ON DELETE CASCADE
316       ON UPDATE CASCADE,
317   PRIMARY KEY (basalcount,
318   serie_sessioncount,
319   serie_udholdT_traening_beskrivelse,
320   teknikT_traening_beskrivelse,
321   test_traening_beskrivelse,
322   sessioncount,
323   udholdT_traening_beskrivelse)
324 ) TYPE = InnoDB;
325 INSERT INTO `basal_tid` (
326   basalcount,
327   serie_sessioncount,
328   serie_udholdT_traening_beskrivelse,
329   teknikT_traening_beskrivelse,
330   test_traening_beskrivelse,
331   sessioncount,
332   udholdT_traening_beskrivelse,
333   tid)
334 VALUES ('','','','','','','','');
```



Figur D.1: *Oprettelse af et træningsprogram*



Figur D.2: *Visning af træningsprogrammer fra tabellen* `traening`

Indeks