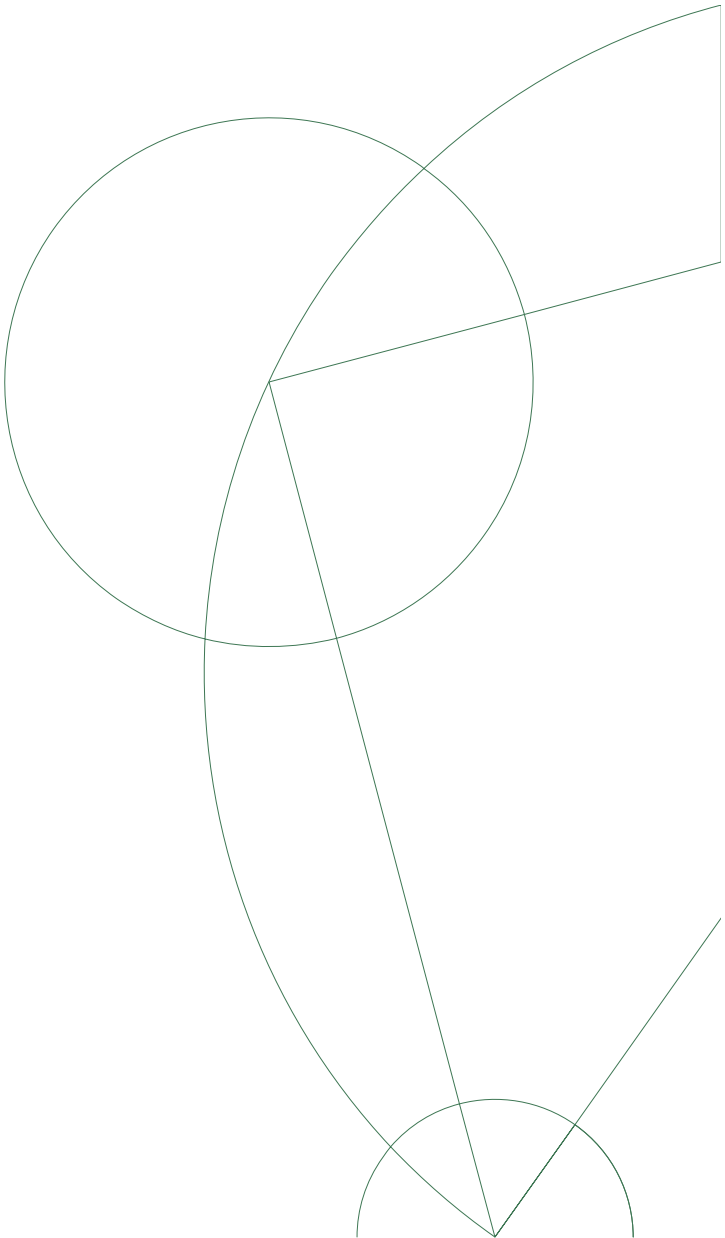
KØBENHAVNS UNIVERSITET

Kandidatspeciale

Søren Gade

Slutbrugerudvikling:

Tilgængelighed og manipulation af data inden for kontaktallergiforskning



Vejleder: Jyrki Katajainen

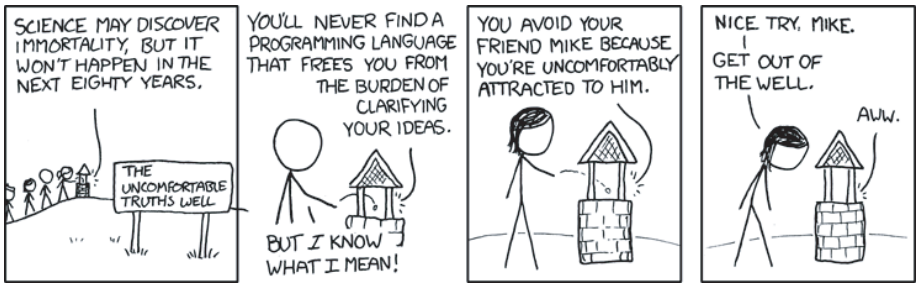
Afleveret april 2010

Tak

Først og fremmest skal lyde en stor tak til min chef professor Jeanne Duus Johansen, leder af Videncenter for Allergi, der dels har ladet mig bruge min arbejdstid på specialet og dels har ladet mig anvende maskinel, programmel, data og kolleger. Uden disse privilegier var nærværende speciale — og dermed min cand.scient. — ikke kommet i stand.

Dernæst ønsker jeg at takke min specialevejleder Jyrki Katajainen. Jyrki har med sin store vejledningserfaring og indsigt i aktuelle og relevante forskningfelter inden for datalogien på samme tid været en god ven og en kompetent og idérig vejleder. Uden Jyrki var hverken specialet eller processen omkring udformningen af det blevet så spændende og frugtbar, som det er tilfældet. Retrospektivt kan siges, at det har været en meget anbefalelsesværdig proces at skrive speciale i datalogi med Jyrki som vejleder.

Jeg takker også de tre testpersoner — mine kolleger — ingeniør Kåre Engkilde, læge Maria Vølund Heisterberg og biolog Michael Dyrgaard Lundov. Uden deres tålmodighed og velvilje i forbindelse med tests, var specialets løsning aldrig blevet så virkeligshedsnær og anvendelig, som den er blevet. At betragte deres interaktion med databaser og tilhørende programmel har været en meget lærerig proces.



Resumé

Inden for kontaktallergiforskning findes en kvalitetsdatabase, der er central for forskning inden for dette domæne. Forskerne, der anvender databasen, har utilstrækkelig viden om databaser og afrapportering til at udvikle ad hoc-forespørgsler på egen hånd.

I dette speciale tilvejebringes et system, der tillader slutbrugerudvikling af ad hoc-forespørgsler. Systemet består af en ny begrænset datamodel i kombination med en professionel forespørgselsapplikation. Modellen anvender denormalisering og indeholder prækalkuleringer og aggregerede kolonner og er opstået som effekt af en iterativ test- og udviklingsstrategi.

Resultatet af specialet er et system, der tillader udvikling af forespørgsler, der leverer valide data. Nogle af testpersonerne har dog stadig problemer med at udforme korrekte forespørgsler, hvilket primært skyldes manglende forståelse for mængdelære og logiske udtryk. Specialet har også resulteret i en række gode råd til udviklere af datamodeller og forespørgselsapplikationer.

Abstract

A national clinical database of contact allergy maintained by the National Allergy Research Centre is central for the research done in this field in Denmark. Even though researchers using the database are experts in their own field, they often have insufficient knowledge of the manipulation of databases and have difficulties in formulating ad hoc queries on their own.

In this thesis the question of end-user development is discussed, and a system is provided that helps end-users, in this particular case contact allergy researchers, to formulate ad hoc queries. The kernel of the system is a new limited data model which is combined with a professional database engine. The data model was an outcome of an iterative test and development strategy where feedback received from end-users had a considerable impact on the final design. To make the interaction straightforward, the model uses denormalization and contains precalculated aggregated columns.

The resulting system provides easy access to the contact allergy data and, as validated, generates the same output as the original system. Some of the test persons still experience difficulties when formulating complicated queries, which is primarily due to lack of understanding of set theory and logical expressions. The thesis also resulted in a series of tips for developers of data models and database engines.

Indhold

Kapitel 1

Introduktion

I løbet af de sidste 20–25 år har IT vundet stadig større indpas i vores hverdag. Et støt stigende antal mennesker anvender dagligt computere og tilhørende software til at løse forskelligartede opgaver — såvel privat som arbejdsrelateret. Tidligere har professionelle softwareudviklere stået for at designe og udvikle de applikationer, slutbrugere anvender, mens slutbrugere typisk kun kan manipulere forskellige data via applikationerne eller tilpasse disse applikationers virkemåde. I takt med den stigende anvendelse af IT i hverdagen er slutbrugerne blevet mere IT-kyndige. Samtidig er der sket en stor udvikling i forhold til at gøre applikationerne mere overskuelige og intuitive at anvende. Dette baner vejen for en større grad af inddragelse af slutbrugere i applikationsudviklingsprocessen. Forskningsfeltet slutbrugerudvikling beskæftiger sig blandt andet med redskaber og metoder, der tillader slutbrugere at udvikle applikationer eller øvrige softwareartefakter.

Inden for lægevidenskaben findes en række databaser og registre, der er centrale for forskning og kvalitetsudvikling inden for de respektive lægespecialer. En del af disse databaser er karakteriseret ved, at forskerne ofte har begrænset viden om IT til støtte af forskningen, datamodeller og hensigtsmæssig afrapportering af data. Dette bevirker, at forskerne ofte spilder kostbar tid, enten fordi de ikke anvender IT optimalt i forhold til de konkrete behov, eller fordi de venter på, at en IT-afdeling eller et IT-firma kan bistå dem i at udforme og foretage de ønskede dataudtræk fra deres databaser. Anvendes eksterne firmaer kan der endvidere opstå økonomiske omkostninger i den forbindelse.

Der findes inden for kontaktallergiforskningen en national klinisk kvalitetsdatabase kaldet *Allergen*, der aktuelt[1] indsamler patientdata fra tre dermatologiske hospitalsafdelinger samt ni dermatologiske speciallægepraksisser. Databasen baserer sig på data vedrørende patienternes lappetests og strækker

[1] Ved udgangen af 2009. Der foreligger planer for national udrulning af databasen, hvilket sammenlagt vil indebære indberetning fra fem dermatologiske hospitalsafdelinger og cirka 90 dermatogiske speciallægepraksisser.

sig for én kliniks vedkommende tilbage til 1977. Denne nationale kliniske kvalitetsdatabase er unik, idet den implicerer data fra såvel primær- som sekundærsektoren; databasen bestyres af Videncenter for Allergi[2]. Videncentret beskæftiger en række ph.d.-studerende samt seniorforskere, der bedriver forskning blandt andet på baggrund af de til databasen indberettede patientdata.

IT-funktionen i Videncenter for Allergi står blandt andet for at udvikle, eksekvere og levere dataudtræk fra databasen til forskerne til brug i deres arbejde. Disse dataudtræk leveres typisk på to måder:

- faste rapporter, der eksekveres via rapportapplikation, og som kan parametriseres på kørselstidpunkt;
- ad hoc-dataudtræk på case-niveau, som udarbejdes og eksekveres på databasen og siden importeres i statistikprogram (SPSS) hos den enkelte forsker.

De faste rapporter bruges hyppigt til mindre opgørelser og statistikker og eksekveres med jævne mellemrum for at skabe overordnet overblik over databasens indhold. Ad hoc-dataudtræk er forskellige fra gang til gang, men adskiller sig i grunden alligevel ikke det store fra hinanden, idet de ofte kun implicerer en begrænset mængde tabeller (fem–ti) fra databasen og stort set altid implicerer de samme tabeller. Sammenfaldet er dog ikke stort nok til at berettige selvstændige faste rapporter.

Der blev fra IT-funktionens side iagttaget en stor del repetition i arbejdet med at udvikle og levere meget ensartede ad hoc-dataudtræk til forskellige forskere. Derfor øjnedes en mulighed for en ressourcebesparelse ved at tilvejebringe et system, der giver forskerne mulighed for selv at udvikle og foretage de fleste relevante ad hoc-dataudtræk fra databasen.

En begrænset udgave af den eksisterende datamodel er indeholdt i dette system, hvor brugerne har adgang til data fra de respektive tabeller og kolonner, der hyppigst anvendes i forbindelse med udvikling af førnævnte ad hoc-dataudtræk. Begrænsningen er således baseret på viden om, hvilke tabeller der typisk anvendes, og søger samtidig dels at skabe et bedre overblik over data via frasortering og dels at minimere sandsynligheden for, at brugerne udvikler fejlbehæftede dataudtræk. Brugerne mister ved begrænsningen af datamodellen adgang til en del af de data, der findes i den fulde database. Det er dog tilstræbt, at brugerne med den nye datamodel som hovedregel kan tilgå samtlige data, der er relevante for deres dataudtræk.

Systemet er visuelt orienteret og intuitivt i brug, således at forskerne med stor sandsynlighed selvstændigt kan hente de ønskede data.

[2]Videncenter for Allergi er et nationalt center, som varetager opgaver vedrørende forskning, overvågning, information og forebyggelse af allergi over for kemiske stoffer. Kilde: `http://www.videncenterforallergi.dk`

1.1 Målsætning

I nærværende projekt satte jeg mig følgende konkrete mål:

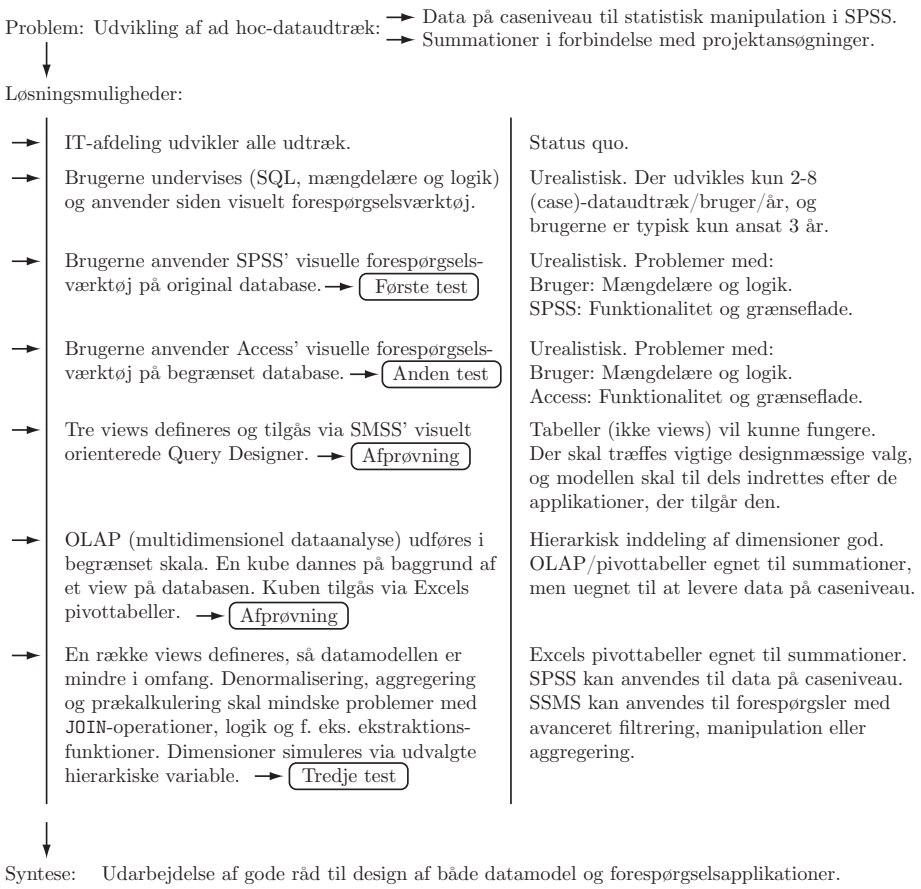
- At oprette en begrænset datamodel, der indeholder de mest relevante og hyppigst anvendte data fra den samlede database.
- At stille en visuelt orienteret grænseflade, der tillader udvikling og afvikling af relevante ad hoc-forespørgsler, til rådighed for slutbrugerne.
- At teste og evaluere systemet ved hjælp af involvering af slutbrugere.
- At validere data genereret af forespørgsler udviklet på baggrund af den nye datamodel. Som reference anvendes den originale datamodel.

1.2 Løsningsmuligheder

Specialet involvererede løsning af et konkret problem af relevans for både nuværende og kommende forskere i Videncentret. Mere generelt kan siges om problemet, at der er tale om en organisation, der indsamler og vedligeholder informationer i en database, hvor disse informationer ikke er repræsenteret på en tilgængelig eller hensigtsmæssig måde i forhold til den videre bearbejdning.

Der er en stor sandsynlighed for, at samme problem gør sig gældende i andre organisationer, der indsamler og bearbejder data, og som ønsker at inddrage slutbrugerne i udvikling og eksekvering af ad hoc-forespørgsler. Det var af denne grund vigtigt at afsøge flere forskellige løsningsmuligheder, der enten helt eller delvist kunne bidrage til den endelige løsning. En væsentlig del af projektet var derfor en iterativ læringsproces, hvor de forskellige løsningsmuligheder afprøvedes én ad gangen, og hvor erfaringer fra tidligere afprøvninger om muligt blev inddraget i designet og afprøvningen af næste løsningsmulighed. Hele denne proces søges afbildet i figur 1.1.

Da det grundlæggende problem som nævnt må findes i mange andre organisationer, kan de erfaringer og løsningsmuligheder, som projektet afføder, sandsynligvis anvendes mere generelt. Resultatet af dette projekt er således også en række gode råd, der kan ses som vejledende retningslinier til lignende organisationer, der indsamler og bearbejder data, og som ønsker at inddrage slutbrugere i udvikling og eksekvering af ad hoc-forespørgsler. Endvidere opstilles en række gode råd til udviklere af forespørgselsapplikationer på baggrund af erfaringer gjort i de udførte tests.



Figur 1.1: Oversigt over udviklingsforløb for løsning af specialets problemstilling.

1.3 Teststrategi

Brugerinddragelse var vigtig i forhold til nærværende speciale, idet et for Videncentret eksisterende problem skulle løses. Der var derfor lagt en strategi, hvor brugerne løbende involveredes i de tests, der blev udført i takt med den iterative udviklingsproces. Disse tests var baseret på en række testscenarier, der blev anvendt gentagne gange. Der blev også testet med flere forskere, der havde forskellige forudsætninger for at foretage databaseudtræk på baggrund af scenarierne.

For at påpege det i nærværende introduktion postulerede problem udførtes først en test med programmel, som forskerne var vant til at anvende, og som umiddelbart var tilgængeligt for dem. Dette programmel havde — omend i begrænset omfang — en visuelt orienteret brugergrænseflade til udvikling af databasefor-

espørgsler, men kunne rent funktionelt ikke løse scenarierne ud fra den aktuelle databasemodel.

Siden blev forskerne inddraget i tests med mere databaseorienteret programmel, der havde større fokus på den visuelle grænseflade og tillod mere avanceret brugerredigering af de forespørgsler, som blev udviklet. Der blev testet med samme scenarier og med en begrænset — men ikke omstruktureret — version af den eksisterende databasemodel.

Senere blev et professionelt databasehåndteringssystem (DBMS)[3] — Microsoft SQL Server 2008 Management Studio (SSMS) — der tilbyder effektiv visuel brugergrænseflade til udvikling af forespørgsler, afprøvet. Forespørgslerne blev udviklet på baggrund af tre views defineret med henblik på at eliminere en stor del af de problemer, som viste sig ved første og anden test. Afprøvningen fandt sted uden inddragelse af testpersonerne, da den alene havde til formål at afdække, hvorvidt en omstrukturering af datamodellen kombineret med præaggregering og -kalkulering kunne være en plausibel løsning.

Derefter udførtes forsøg med inddragelse af elementer fra multidimensionel dataanalyse, hvor en afprøvning søgte at afdække, hvorvidt denne type analyse kunne anvendes til at løse det skitserede problem. Et kontorprograms funktionalitet, *pivottabeller*, blev anvendt til at gennemføre denne afprøvning, der heller ikke inddrog slutbrugerne.

Slutteligt afvikledes tests på baggrund af en nyudviklet datamodel, der bestod af en række views, der trak på de forskellige erfaringer indhentet ved de tidligere tests og afprøvninger.

Som nævnt blev forskerne ikke inddraget under afprøvninger, mens alle tests foregik individuelt og blev afviklet således, at testpersonen efter en kort introduktion til det konkrete programmel skulle udvikle forespørgsler på baggrund af i forvejen fastlagte testscenarier. Personerne observeredes under hele sceancen, og der blev taget skærmdumps af resultater, der blev anskuet for interessante. Hvis der var scenarier, som var mulige at realisere med det forhåndenværende programmel, men de respektive personer ikke på egen hånd lykkedes med dette, blev personerne efter endt forsøg hjulpet i den rigtige retning. Dette skete af hensyn til at sikre så bred en evaluering af det respektive programmel som muligt.

1.3.1 Testscenarier

Der er udviklet syv scenarier af varierende størrelse og kompleksitet, der var designet til at teste såvel brugernes evner som programmernes funktionalitet:

1. Simpelt udtræk, en tabel.
2. Simpelt udtræk, en tabel.

[3] Database Management System.

3. Simpelt udtræk, en tabel, simpel filtrering.
4. Simpelt udtræk, to tabeller, `INNER JOIN` og `DISTINCT`/`GROUP BY`.
5. Middelsvært udtræk, to tabeller, `LEFT OUTER JOIN`.
6. Svært udtræk, fire tabeller, `INNER JOIN`, svær filtrering.
7. Svært udtræk, seks tabeller, tre `INNER JOIN` og to `LEFT OUTER JOIN`, simpel filtrering.

De syv scenarier er beskrevet i både tekst og kode i bilag B. I bilag A findes en gennemgang af de forskellige `JOIN`-operationer.

1.3.2 Testpersoner

Videncenter for Allergi beskæftiger cirka 10–14 medarbejdere. Antallet varierer over tid, idet flere af medarbejderne er ph.d.-studerende eller ansat tidsbegrænset i forbindelse med konkrete projekter. Omtrent halvdelen af disse medarbejdere er forskere, der i varierende grad har behov for data fra Allergendatabasen.

Til afvikling af tests involveredes tre forskere med hver deres baggrund og forudsætninger:

- **Person 1:** Kvindelig lægeuddannet ph.d.-studerende midt i trediverne. Vant til databehandling i anerkendt statistikprogram og eksekvering af faste rapporter i rapportapplikation.
- **Person 2:** Mandlig ingeniøruddannet seniorforsker (ph.d.) i starten af trediverne. Vant til avanceret databehandling i anerkendt statistikprogram med udvikling af programmeringsscript. En del programmeringserfaring.
- **Person 3:** Mandlig biologuddannet ph.d.-studerende i slutningen af tyverne. Har arbejdet minimalt med statistikprogrammer og databaser i forbindelse med kurser.

Ved valget af testpersoner blev der forsøgt taget højde for et repræsentativt udsnit af Videncentrets nuværende og kommende forskeres erfaring med

- databaser, herunder specifikt kontaktallergidatabasen,
- statistisk databehandling,
- matematisk modenhed; mængdelære og logiske udtryk,
- lettere programmering, for eksempel mindre script.

Alle ovenstående faktorer tænktes at have indflydelse på, hvor godt personerne præstererede ved tests.

1.4 Forudsigelser

Det var forventet, at testpersonerne ville opleve en del problemer i forbindelse med at udvikle de dataudtræk, der beskrives i testscenarierne. Nogle af disse testscenarier tænktes at være umulige at realisere for nogle personers eller programmers vedkommende. Der blev identificeret fire mulige årsager til, at dette ville gøre sig gældende:

1. en svært tilgængelig datamodel,
2. testpersonernes manglende viden om eller forståelse for data og manipulation hermed; herunder mængdelære og logiske udtryk,
3. programmer der er uintuitive eller uigennemskuelige for testpersonerne,
4. programmer der er utilstrækkelige i funktionalitet — altså ikke fuldt ud understøtter realisation af testscenariet.

Det skal her nævnes, at et af hovedmålene med dette projekt var at gøre datamodellen mere overskuelig og tilgængelig samt at skabe en mere intuitiv grænseflade til at udføre ad hoc-dataudtræk fra databasen. Ligeledes var det et mål at give mulighed for al den funktionalitet, der er tilgængelig i ren SQL-kode.

Endvidere bør bemærkes, at det blev antaget, at testpersonerne formodentlig blev bedre til at forstå data og udforme udtræk i takt med, at de deltog i tests. Det var derfor forventet, at personerne præsterede bedre i senere tests end i de første. Samtidig var antagelsen, at brugerne i de senere tests kunne huske dele af deres løsninger fra tidligere tests. Første og anden test udførtes med 1–2 ugers mellemrum — begge i november 2009, mens tredje test udførtes i marts 2010.

Kapitel 2

Slutbrugerudvikling

Slutbrugerudvikling (SBU) — end-user development — defineres i [20] på følgende vis:

> *EUD can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact.*

Betegnelsen dækker således over et større felt, der potentielt spænder lige fra simpel konfiguration af programmers opførsel til udvikling af programmer. SBU dækker for eksempel optagelse af makroer i kontorprogrammer, udvikling af formler i regneark eller konfigurering af filtrering af e-mails på baggrund af eksempelvis størrelse eller afsender. Feltet dækker blandt andre også slutbrugeres udvikling af mindre, webbaserede applikationer eller medarbejdere i forskellige firmaer, der agerer ikke-professionelle softwareudviklere, og bidrager til ny- eller videreudvikling af virksomhedens IT-applikationer.

Den øgede brug af IT i hverdagen og adgang til internet giver SBU et stort udviklingspotentiale inden for den nærmeste fremtid. Politisk har feltet også stor betydning, idet det er en essentiel komponent i at få fuld udbredelse af det digitale informationssamfund blandt borgere.

Slutbrugerudvikling er som nævnt et stort forskningsfelt og inddrager både dele fra HCI[1], klassisk programmering, systemdesign og -udvikling samt psykologiske og sociale aspekter ved brugerinddragelse. I det efterfølgende gennemgås en række aktuelle områder inden for feltet samt en vurdering af anvendelighed og relevans i forhold til nærværende speciale.

[1] Human-computer interaction. Studier af interaktionen mellem mennesker og computere.

2.1 Programmering ved eksempel (PBE)

Programmering ved eksempel (eng: *programming by example*) refererer til en teknik, hvor en bruger ved sin interaktion med et system via eksempler demonstrerer, hvordan systemet skal opføre sig i fremtidige situationer [21]. En *agent* overvåger alle brugerens interaktioner med systemet og skaber på baggrund af disse ved hjælp af maskinlæring en række generaliserede rutiner, der repræsenterer brugerens aktioner.

Der findes en række eksempler på PBE i praksis; herunder applikationerne *AgentSheets*[2] og *StageCast Creator*[3]. PBE synes ikke at have fundet større generel udbredelse og tænkes ikke umiddelbart at kunne anvendes i forbindelse med nærværende projekt.

2.2 SBU i organisationer

Forskning inden for SBU fokuserer typisk på kognitive eller tekniske udfordringer, mens det også er relevant at fastlægge de organisatoriske, økonomiske og sociale aspekter ved udviklingen. Et studie [22] blandt forskere og ledere inden for både industri og statslige organisationer, der belyste disse aspekter, viste blandt andet,

- at SBU kan bidrage til udvikling af mere effektive applikationer.
- at værktøjer via domænespecificitet skal tilpasses brugerne, så fokus er på opgaven frem for den kognitive udfordring ved anvendelse af værktøjet.
- at slutbrugerne skal motiveres ved tydeliggørelse af, hvordan SBU kan effektivisere arbejdsgange, samt at brugernes incitament til deltagelse skal øges ved at planlægge videndeling.

SBU i organisationer er et område, der er meget relevant for specialet, idet netop viden og deling af denne viden inden for et specifikt domæne er omdrejningspunkt for Videncentret.

2.3 Participatorisk programmering (PP)

Participatorisk programmering (eng: *participatory programming*) kan beskrives som den samlede proces, der implicerer slutbrugere i alle faser af software; design, programmering, brug og skræddersyning. PP er for eksempel relevant i

[2] AgentSheets er en applikation, der tillader brugere at kreere og eksekvere simuleringer af fysiske objekters opførsel og interaktion. `http://www.agentsheets.com`

[3] StageCast Creator er et kommercielt produkt, der tillader brugeren at kreere små banebaserede spil og derefter spille dem. `http://www.stagecast.com`

forhold til grupper af domæneeksperter, og man kan i fagligt stærke miljøer bestående af domæneeksperter og IT-professionelle med fordel afholde workshops, hvor brugerne i højere grad involveres i design- og programmeringsfasen og derved får større indflydelse på funktionaliteten af og grænsefladen i de værktøjer, de anvender i deres arbejde [18].

Et studie [17] af processen omkring udvikling og integration af et analyse-, udviklings- og visualiseringsmiljø for biologer viste blandt andet, at der trods stor fokus på en grafisk orienteret grænseflade, der gjorde programmeringen mere tilgængelig, var brug for en vis grundlæggende forståelse for programmering blandt de deltagende biologer. Studiet viste også, at biologerne foretrak enten at udføre programmeringsaktiviteter som del af almindelig softwarebrug eller helt at undgå at programmere. Biologernes problemer med at programmere skyldtes overraskende nok ikke så meget selve programmeringen, men i højere grad hvorledes den software, der tillod programmeringen, var konstrueret.

Participatorisk programmering er relevant for specialet, da det netop implicerer udvikling blandt en gruppe domæneeksperter. Fundene i [17] er relevante i forhold til den endelige løsning på specialets problemstilling.

2.4 Naturlige sprog

Gennem tiden er der gjort forsøg på at muliggøre programmering for ikke-professionelle brugere via anvendelse af naturligt sprog i stedet for tradionelle programmeringssprog, der både er krævende og formelle af karakter.

Der er blandt andet ved hjælp af observationer af ikke-programmørers løsninger på programmeringsopgaver gjort forsøg på at udvikle værktøjer, der tillader udvikling af mindre applikationer møntet på børn [25]. Der synes blandt forskere inden for naturlige sprog at herske en enighed om, at udvikling udtrykt i naturlige sprog på sigt er plausibel [19] og vil yde vigtige bidrag til den videre udvikling af brugbare og effektive udviklingsværktøjer og applikationer [24].

Anvendelse af naturlige sprog skønnes ikke relevant i forhold til løsning af det forhåndenværende problem. Tidlige studier [23] har påvist, at udvikling af forespørgsler til databaser er kompliceret, lige meget om der anvendes tradionelle forespørgselssprog eller naturlige sprog. Dette berøres grundigere i kapitel 6 i relation til forespørgselssprog og -applikationer til databaser.

2.5 Softwareformningsworkshopper (SSW)

Softwareformningsworkshopper (eng: *software shaping workshops*) stiller et rammeværk for slutbrugerinddragelse inden for specialiserede domæner til rådighed og er opstået som et forsøg på at muliggøre udvikling af softwareartefakter for brugere, uden at de skal programmere i traditionel forstand [10]. Dette

foregår ved seminarer, der søger at minimere den kommunikationsmæssige afstand mellem brugere, designere og udviklere. Ved disse seminarer samarbejder domæneeksperter med professionelle softwaredesignere om at udvikle softwareartefakter, der er skræddersyet til domænets behov, og som siden kan formes af domæneeksperterne selv efter domænets behov. Denne formning foregår ved hjælp af visuelt orienterede højniveausprog.

SSW tænkes ikke i sin fulde udstrækning at kunne anvendes i forbindelse med dette speciale, da midlet ville overstige målet. Dog kan siges, at forskerne samarbejder med IT-sektionen om at udvikle faste rapporter, der kan parametriseres på kørselstidspunkt. Således bidrager forskerne altså i forvejen til både udvikling og formning af softwareartefakter.

2.6 Samarbejde om tilpasning (CT)

Samarbejde om tilpasning (eng: *collaborative tailoring*)[4] bygger på den præmis, at en softwareapplikation meget sjældent er designet til, at én person anvender den netop én gang. Tværtimod er der ofte flere personer, der anvender applikationen sideløbende med hinanden og ad flere omgange. Der er potentielt store gevinster ved, at personer, der udfører lignende opgaver, samarbejder om at tilpasse pågældende applikation til fælles behov, hvilket belyses og underbygges nærmere i [14]. Et analyseværktøj og en taksonomi for systemer, der anvender CT, beskrives endvidere i [27].

Det skønnes ikke, at CT uden videre kan benyttes til løsning af problemet i nærværende speciale, da der er tale om softwareartefakter — og ikke selvstændige applikationer — der tilpasses. Grundprincippet i CT kan dog ikke helt forkastes, idet forskerne for eksempel potentielt kan samarbejde om at gemme, dele og tilpasse de dataudtræk, der udvikles.

2.7 Visuelle grænseflader

Der er i takt med den stigende udbredelse af computere og anvendelse af software gjort mange forsøg på at gøre applikationer mere brugbare og brugervenlige via indførelse af forskellige visuelle grænseflader; herunder tekst-, ikon-, formular- eller diagrambaserede. Emnet berøres nærmere i kapitel 6 i relation til forespørgselssprog og -applikationer til databaser.

En systematisk gennemgang af en række forsøg på systemer, der muliggør slutbrugerudvikling, har resulteret i 13 retningslinier for design af visuelle grænseflader [29]. Disse retningslinier søges anvendt i løsningen af specialet, hvor det skønnes plausibelt.

[4] *"De tekniske og menneskelige aspekter ved at ændre softwarens funktionalitet, mens softwaren er i brug, og at gøre dette i samarbejde med andre"* [14].

Kapitel 3

Baggrund for projektet

I dette kapitel beskrives den domænespecifikke baggrund for specialeprojektet. Først introduceres kort sygdomsområdet kontaktallergi, herefter forskning inden for området og derefter beskrives eksisterende databaser og for projektet relevante etablerede arbejdsgange inden for forskningen. En mere detaljeret beskrivelse af kontaktallergi findes i [11].

3.1 Kontaktallergi

Kontaktallergi er en specifik overfølsomhed for et eller flere veldefinerede stoffer, man er eksponeret for i miljøet, og kan vise sig ved eksem. Kontaktallergiske reaktioner kan fremprovokeres af mange forskellige stoffer, for eksempel metaller, parfumestoffer eller konserveringsmidler. Kontaktallergi påvises ved en lappetest, hvor plastre påføres huden. Plastrene består af nogle kamre, der hver især indeholder et af de stoffer, der kan være allergi overfor. Efter at plastret har siddet på huden nogle døgn, aflæses eventuelle allergiske reaktioner. Der testes også hyppigt med produkter eller materialer, som patienten er eksponeret overfor og selv har medbragt.

Dermatologen vil på baggrund af de eventuelle positive reaktioner på lappetesten ud fra patientens anamnese[1] og erhverv forsøge at finde den konkrete eksposition i patientens omgivelser, så denne fremdeles kan minimere eller eventuelt helt undgå at blive eksponeret for stoffet eller stofferne.

[1] Anamnese er betegnelsen for patientens egen redegørelse for sin sygehistorie eller generelt for en patients sygehistorie. Kilde:
`http://da.wikipedia.org/wiki/Anamnese_(sygehistorie)`

3.2 Aktuel forskning

Videncenter for Allergi er et nationalt center, som varetager opgaver vedrørende forskning, overvågning, information og forebyggelse af allergi over for kemiske stoffer. Videncentrets arbejdsmetode er projektbaseret vidensgenerering, der har forebyggelse som mål og tager udgangspunkt i kliniske problemstillinger. Centret er etableret i tæt kontakt med de enheder i hospitalsvæsnet, hvor personer med allergisk sygdom behandles, da man herved sikrer den optimale forskning og vidensformidling, idet der går kortest mulig tid fra ny viden opstår om forebyggelse af allergi, til denne er formidlet til for eksempel industri og administrative myndigheder, som skal udmønte forskningen i praksis.

Projekterne beskæftiger sig blandt andet med allergi over for metaller, konserveringsmidler og hårfarver. Desuden har der været mere tværgående ph.d.-projekter om etablering af en ny risikovurderingsmodel for kontaktallergi, sygdomsmønstre, allergirisiko ved nanopartikler, karakteristik af patienter med flere allergier samt sundhedstjenesteforskning med kortlægning af patientforløb og disses indflydelse på sygdommens prognose.

I praksis udmøntes Videncentrets forskning blandt andet gennem deltagelse i nationale og internationale ekspertråd i for eksempel EU-regi, samt ved fremsendelse af relevante resultater til Europa-Kommissionen med henblik på forebyggende reguleringer af miljøet. Forskningen understøtter også praktiske anbefalinger og søger at forbedre information til patienter med allergi.

Resultater af forskning bedrevet ved Videncentret publiceres i anerkendte relevante tidsskrifter. Årsrapport for 2008 for Videncenter for Allergi [13] beskriver i detaljer Videncentrets arbejde.

3.3 Eksisterende databaser

I 2002 etableredes en national klinisk kvalitetsdatabase[2] for kontaktallergi, kaldet Allergen. I databasen[3] indberettes p.t. patientdata fra tre dermatologiske hospitalsafdelinger samt ni dermatologiske speciallægepraksisser. For en enkelt kliniks vedkommende er der indberettet data helt tilbage fra 1977, mens der siden 2003 på landsplan konsistent er indberettet data fra mere end 3500 patienter.

De indberettede data inkluderer en række baggrundsvariable om blandt andet

[2] En klinisk database er et register, der med udgangspunkt i patientforløb søger at belyse og bidrage til forbedring af kvaliteten af Sundhedsvæsenets indsats og resultater for en afgrænset gruppe af patienter. Afgrænsningen af relevante patienter kan for eksempel basere sig på en bestemt sygdomsgruppe, diagnose eller behandlingsmetode. Man ønsker således via vedvarende oplysninger om bestemte patienter at kunne sammenligne data over tid og på tværs af tilsvarende afdelinger.

[3] Reelt findes der én database per klinik, altså sammenlagt 12 databaser. Disse samles i én national database, der således udgør den nationale kliniske kvalitetsdatabase.

alder, køn, stilling og eksemlokalisation(er). Desuden registreres alle de allergener[4], patienterne testes med, samt eventuelle reaktioner. Er der ved lappetesten fremprovokeret positive allergiske reaktioner, registreres endvidere for reaktionen eventuelt nuværende og tidligere relevans for det aktuelle forløb. I fald der er tale om en aktuel relevans, registreres endvidere nuværende konkrete eksposition(er), hvis det er muligt at fastlægge. Endelig registreres en eller flere slutdiagnoser for patienten, samt hvorvidt patienten har fået udleveret skriftligt informationsmateriale om sin lidelse.

Der registreres i databasen også en lang række andre informationer om patienterne, men de bliver dels ikke registreret i alle klinikker og dels anvendes de sjældent i ad hoc-dataudtræk. Disse er ofte tilknyttet projekter, der for eksempel løber i en kortere periode og er knyttet til bestemte klinikker. Der vil derfor være fokus på de væsentligste og hyppigst anvendte tabeller fra databasen.

3.4 Arbejdsmetoder

I forbindelse med levering af dataudtræk fra databasen, findes grundlæggende to arbejdsgange, der afhænger af, hvorvidt der er tale om faste rapporter eller om ad hoc-udtræk.

3.4.1 Faste rapporter

Der knytter sig til databasen en række faste rapporter, der kan parametriseres på eksekveringstidspunktet. Disse rapporter tjener hovedsagligt tre formål:

- Overvågning og dokumentation af kvaliteten i form af indikatorer[5] i henhold til *Bekendtgørelse om godkendelse af landsdækkende og regionale kliniske kvalitetsdatabaser*[6]. Indikatorerne anvendes også til løbende validering af kvaliteten af behandlingen i klinikkerne samt kvaliteten af de indberettede data. En mere udførlig beskrivelse af indikatorerne findes i [11].
- Udarbejdelse af statistikker til brug i forbindelse med oversigter og grafmateriale i projektansøgninger stilet mod fonde, styrelser og øvrige myndigheder.
- Overvågning af udviklingen af allergifrekvenser nationalt.

Rapporterne leverer altså data på opsummerende niveau og ikke caseniveau og er en integreret del af Videncentrets daglige arbejdsgang. Forskerne kan selv få

[4] Substanser, der genkendes af immunsystemet og forårsager en allergisk reaktion.

[5] En indikator er en målbar variabel, der alene eller sammen med andre indikatorer anvendes til at belyse behandlingskvaliteten og udviklingen af denne.

[6] Kilde: `https://www.retsinformation.dk/Forms/R0710.aspx?id=10263`

adgang til en rapportapplikation, der kan eksekvere de faste rapporter. Resultater af rapporter kan enten udskrives eller eksporteres til en flad fil.

Der ny- og videreudvikles løbende faste rapporter i takt med, at behovet opstår. Dette foregår hyppigt i samarbejde mellem IT-funktionen og forskningslederen. IT-funktionen udvikler også ofte faste rapporter på eget initiativ eller på opfordring fra samarbejdspartnere inden for både forskning og patientbehandling.

3.4.2 Ad hoc-dataudtræk

Ad hoc-dataudtræk udvikles primært i forbindelse med konkrete forskningsprojekter og anvendes i flere af projekternes forskellige faser. Dels udvikles og eksekveres dataudtræk i forbindelse med, at idéen til et projekt opstår. Disse dataudtræk er af oversigtsmæssig karakter og tjener det formål at undersøge grundlaget for det potentielle projekt og bliver ofte inddraget i eventuelle projektprotokoller og i ansøgninger til forskellige myndigheder og fonde.

Andre ad hoc-dataudtræk udvikles i forbindelse med konkrete igangværende projekter, hvor disse udtræk er det bærende element. Ofte udvikles og eksekveres også udtræk, der skal samkøres med øvrige registre og databaser, for eksempel ATP, hoftedatabasen eller dødsårsagsregisteret.

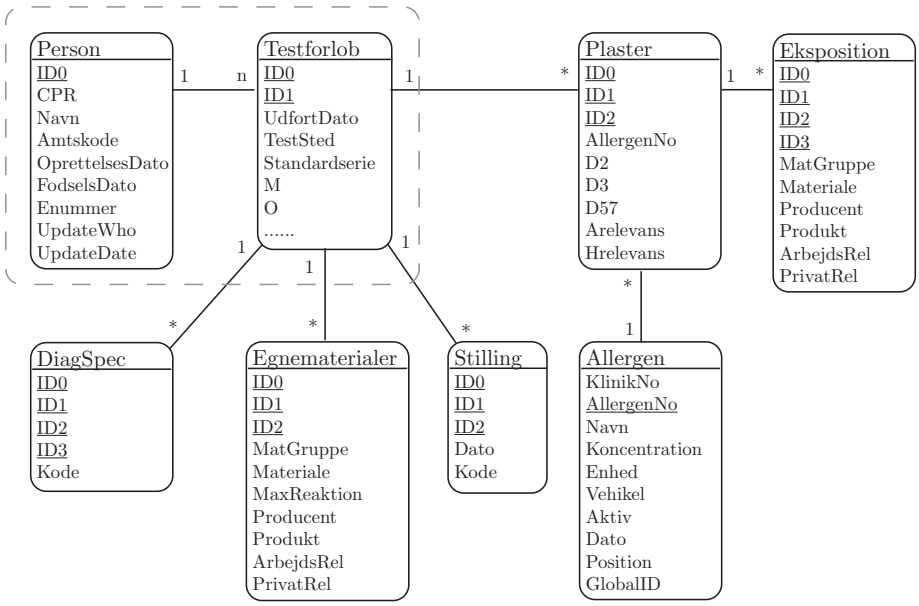
Fælles for ad hoc-dataudtræk er, at de udvikles i samarbejde mellem en eller flere forskere og IT-funktionen. Det fastslås, hvilke data der ønskes leveret, og hvad inklusionskriterierne skal være. Der træffes aftaler om håndtering af fejl og delvist manglende data. Afhængig af kompleksitet og størrelse udvikles og leveres dataudtræk i løbet af en-fem dage. Det hænder med jævne mellemrum, at forskerne kontakter IT-funktionen for at videreudvikle de leverede udtræk, hvis de har vist sig at være utilstrækkelige. Dette skønnes at skyldes forskernes manglende overblik over de ønskede data på tidspunktet for beskrivelsen af de berørte dataudtræk.

3.5 Basal datamodel

Den fulde database består af mere end 150 tabeller (eksklusiv system- og temporære tabeller). Mange af disse anvendes i forbindelse med registrering og overvågning af klinikspecifikke behandlingstiltag og bruges derfor sjældent af forskerne. En del tabeller relaterer sig endvidere til tidligere projekter, der har kørt i en begrænset periode og derfor ikke længere er aktuelle. Der findes også en del tabeller, der kan karakteriseres som historiske, og som er blevet overflødige i takt med den trinvise udvikling i databasens design. Tabellerne er dog ikke slettet, da man principielt gerne vil gemme al data.

I figur 3.1 ses et diagram over de centrale tabeller i databasen og relationerne i mellem dem. Primærnøgler er markeret med understregning.

For hver person findes mindst ét testforløb, hvilket er den minimale datamængde for en patient i databasen. For hvert testforløb kan personen testes med et variabelt antal egnematerialer[7] og pålægges et variabelt antal plastre. Hvert plaster indeholder netop ét allergen, og for hvert plaster kan registreres et variabelt antal ekspositioner. Personen kan tildeles et variabelt antal stillinger og et variabelt antal diagnoser.



Figur 3.1: Oversigt over de centrale tabeller i den eksisterende datamodel.

ID0 er del af primærnøgle i alle tabeller, der er *personbundne*[8], lige som ID1 er del af primærnøglen i alle tabeller, der er bundet til testforløb. Det er værd at bemærke, at ID2 i tabellerne **Diagspec**, **Stilling**, **Egnematerialer** og **Eksposition** ikke har relation til hinanden, lige som Dato i tabellerne **Stilling** og **Allergen** ikke er relaterede.

[7] Egnematerialer er en betegnelse for effekter, patienten selv medbringer og bliver testet med i klinikken. Der kan for eksempel være tale om private kosmetiske produkter eller materialer, som patienten eksponeres for på sin arbejdsplads. Konstateres både allergiske reaktioner over for et allergen og et egnemateriale, der indeholder førstnævnte allergen, vil egnematerialet også hyppigt være en eksposition.

[8] Med personbunden forstås her tabeller, der relaterer sig direkte til specifikke testforløb. Specifikke rækker kan således identificeres på testforløbniveau. Diverse forklaringstabeller er altså at opfatte som *ikke-personbundne*.

Kapitel 4

Første test

4.1 SPSS

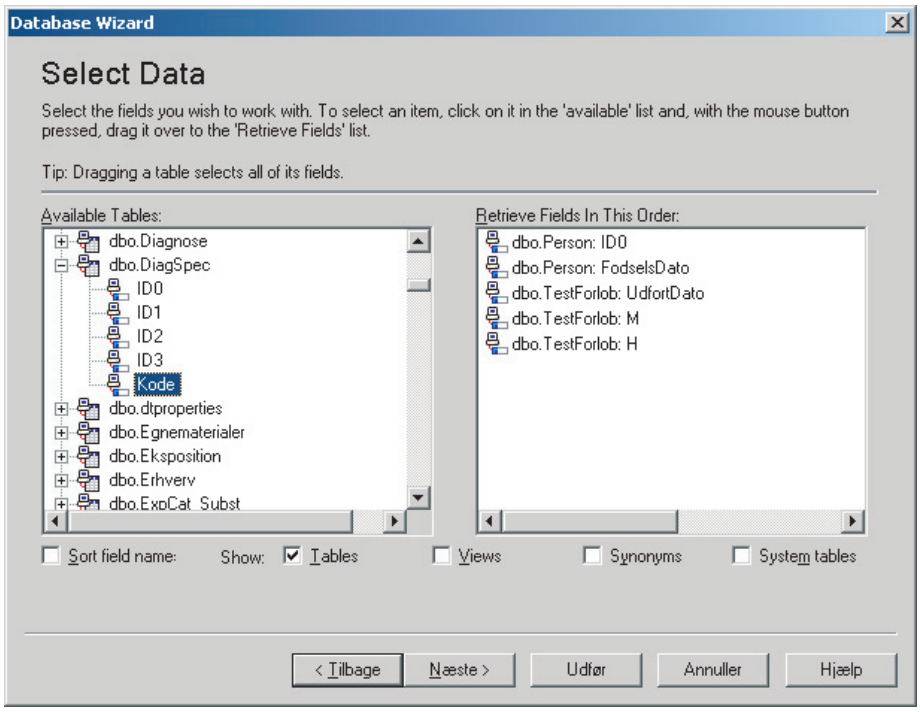
I den første test blev testpersonerne efter en kort introduktion (cirka 15 minutter) til relationelle databaser bedt om at udforme dataudtræk baseret på de tidligere beskrevne testscenarier. Testpersonerne fik udleveret en oversigt over databasen, som afbildet i figur 3.1. Personerne anvendte programmet *SPSS 15.0 for Windows*, som er et af mange blandt forskere anerkendte programmer til statistisk databehandling.

Programmet har en wizard, der via en visuel grænseflade tillader brugeren at udvikle SQL-forespørgsler, hvis resulterende data bliver importeret til programmet. I denne wizard vælges først datakilde. Derefter kan vælges, hvilke felter man ønsker at importere fra alle tabeller og views i databasen; dette er afbildet i figur 4.1.

Brugeren præsenteres for et vindue, hvor man kan se, hvordan wizarden vil udføre eventuelle `JOIN`-operationer. Brugeren kan lade programmet foretage disse `JOIN`-operationer automatisk (programmet udfører `NATURAL JOIN`) eller vælge selv at definere dem. Der kan oprettes og slettes relationer, og der kan vælges mellem `INNER JOIN`, `LEFT OUTER JOIN` eller `RIGHT OUTER JOIN`. `FULL OUTER JOIN` er således ikke understøttet. Vinduet vises selvfølgelig kun, hvis der hentes data fra mere end en tabel.

Brugeren kan derefter vælge at filtrere de data, der hentes, idet man kan stille betingelser op. Der stilles fra programmet en række funktioner til rådighed, blandt andre typekonverterings-, tekstekstraktions- og tekstmanipulationsfunktioner; dette er afbildet i figur 4.2.

Brugeren præsenteres derefter for et vindue, der viser den SQL-kode, som wizarden vil eksekvere på databasen. Der er mulighed for at ændre i koden, lige som man kan gemme koden til senere brug.

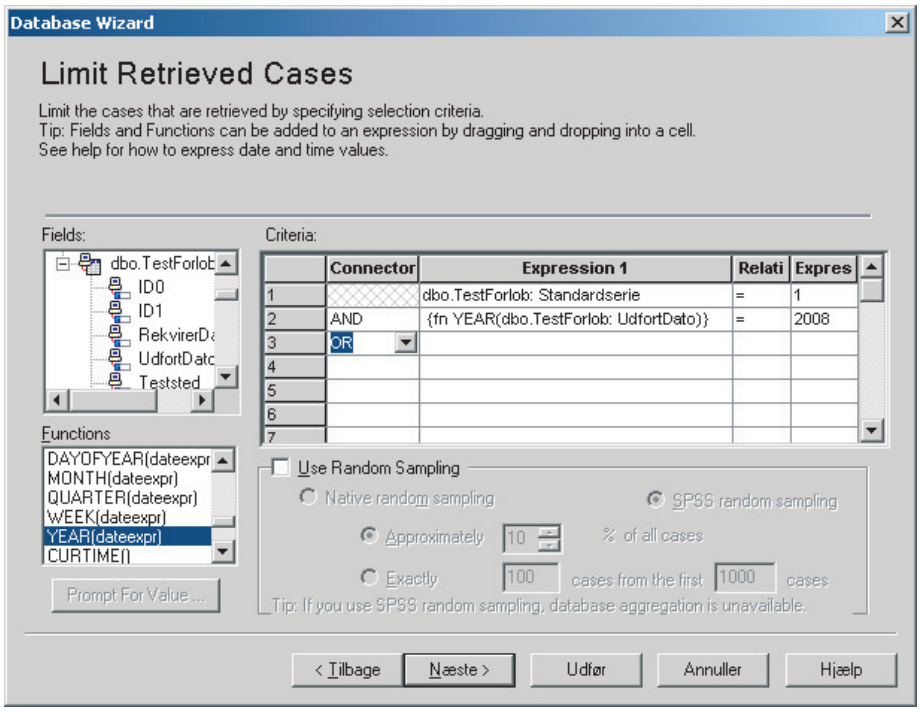


Figur 4.1: Valg af felter fra databasens tabeller og views i wizard.

4.2 Resultater

Der var ingen problemer med at udvikle dataudtræk svarende til testscenarie 1 og 2. I scenarie 3 gik det galt for testperson 1, da denne skulle vælge alle testforløb med testdato i 2008. Personen fandt den indbyggede `YEAR`-funktion, men begik to fejl, idet denne ville filtrere på baggrund af, at teståret skulle være $\geq$ 2008 og samtidig $\leq$ 2009, i stedet for blot $=$ 2008. For det første ville resultatet have været alle testforløb fra både 2008 og 2009, og for det andet kunne testpersonen ikke få programmet til at tillade flere kriterier samtidigt (`AND`/`OR`). Testperson 3 lykkedes ikke med scenariet, idet denne ikke fandt `YEAR`-funktionen, men i stedet forsøgte at angive datoer med formatet `DD-MM-ÅÅÅÅ`. Dette gav problemer, da datoer i databasen er angivet ved hjælp af typen `DATETIME`, hvor formen for en dato er `ÅÅÅÅ-MM-DD TT:MM:SS.MMM`. Programmet konverterer altså — modsat mange øvrige applikationer — ikke automatisk mellem forskellige datoformater baseret på for eksempel styresystemets landespecifikke indstillinger.

Testscenarie 4 voldte testpersonerne problemer, idet programmet tilsyneladende ikke understøtter `DISTINCT`-funktionen. Testperson 2 forsøgte at løse problemet



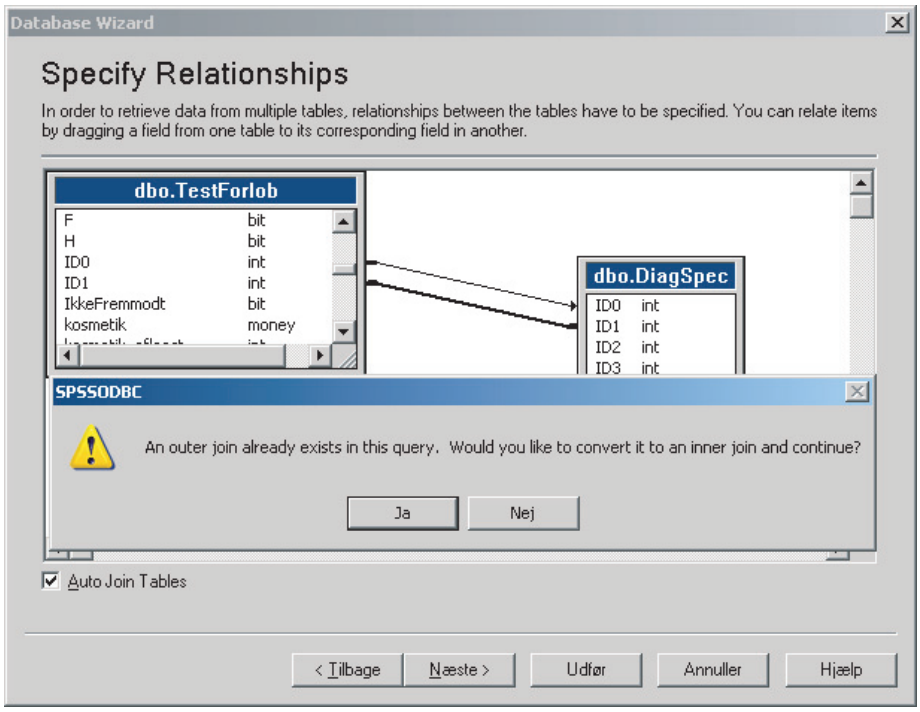
Figur 4.2: Filtrering af cases via logiske udtryk samt forskellige funktioner.

ved at filtrere på ID1. Personen antog, at ID1 blev tildelt iterativt, således at en persons første testforløb ville have ID1 = 1 og dennes andet testforløb ville have ID1 = 2. Tanken var god, men gav dog ikke det ønskede resultat, da der er to problemer:

1. Tidligere tildeltes ID1 iterativt per testforløb i databasen og ikke person-testforløb, det vil sige, at første persons testforløb havde ID1 = 1, anden persons første testforløb havde ID1 = 2 et cetera. Tildelingsproceduren er siden rettet, men de historiske data er uændrede.

2. Hvis en person har to testforløb, og det første slettes, vil det andet testforløbs ID1 ikke blive talt ned, og denne person ville således ikke længere have et testforløb med ID1 = 1.

I testscenarie 5 blev automatisk valgt `INNER JOIN` på både ID0 og ID1. Relationen mellem ID0 i de to tabeller rettede testpersonerne korrekt til `LEFT OUTER JOIN`. Da de forsøgte at ændre relationen mellem ID1 i de to tabeller til `LEFT OUTER JOIN`, mislykkedes det, idet programmet tilsyneladende ikke tillader at definere `OUTER JOIN` på sammensatte nøgler (figur 4.3).



Figur 4.3: Programmet tillader ikke `OUTER JOIN` på sammensatte nøgler.

Testpersonerne opnåede således følgende SQL-forespørgsel:

```
SELECT  T120.ID0,  T120.UdfortDato,  T120.M,  T41.Kode
FROM { oj dbo.TestForlob T120 LEFT OUTER JOIN
dbo.DiagSpec T41 ON T120.ID0 = T41.ID0 }  WHERE T41.ID1 = T120.ID1
```

Den korrekte SQL-forespørgsel ses nedenfor:

```
SELECT  T.ID0, T.UdfortDato, T.M, D.Kode
FROM    dbo.TestForlob T LEFT OUTER JOIN
        dbo.DiagSpec D ON T.ID0 = D.ID0 AND T.ID1 = D.ID1
```

I testscenarie 6 blev dato-kolonnerne i tabellerne **Allergen** og **Stilling** fejlagtigt relateret, selvom de ingen relation har; dette var også tilfældet for ID2 i tabellerne **Plaster** og **Stilling**. Testpersonerne slettede dog ganske korrekt disse relationer.

Der var dog et for testpersonerne svært løseligt problem i scenarie 6, idet man i SPSS tilsyneladende ikke kan anvende parenteser ved filtrering for at angive præcedens i logik. Vil man således angive udsagnet:

bet1 `AND` ((bet2a `AND` bet2b) `OR` (bet3a `AND` bet3b) `OR` (bet4a `AND` bet4b))

skal man skrive:

bet1 `AND` bet2a `AND` bet2b `OR`
bet1 `AND` bet3a `AND` bet3b `OR`
bet1 `AND` bet4a `AND` bet4b

Programmeringsvante vil hurtigt indse dette problem og imødese det, men det må forventes, at det ikke er klart for enhver, at `AND` har højere præcedens end `OR`, og at udsagnet derfor skal konstrueres som angivet ovenfor. Ingen af testpersonerne lykkedes med angivelse af betingelser i filtreringen.

Testscenarie 7 viste sig umuligt at realisere, idet man som tidligere nævnt i programmet ikke kan lave `OUTER JOIN` på sammensatte nøgler. Testperson 1 glemte at ændre automatiske `INNER JOIN` til `LEFT OUTER JOIN` og kom til at slette en automatisk genereret relation for meget. Testperson 2 forsøgte korrekt at ændre `JOIN`-operationerne men opnåede som afbildet i figur 4.4 en fejl under forsøget.

Figur 4.4: SPSS tillader ikke `OUTER JOIN` på flere end to tabeller.

Det kan altså ikke lade sig gøre at anvende `OUTER JOIN` i udtræk, der involverer mere end to tabeller.

4.3 Diskussion

Generelt kan siges, at testpersonerne viste sig at være meget kompetente i forhold til at lave udtræk på baggrund af scenarierne. Testperson 1 skal fremhæves for sin metodiske tilgang til at gennemse alle relationer mellem tabellerne i den visuelle oversigt i programmet. En del af de fejl, testpersonerne begik, skyldtes deres manglende viden eller forståelse, lige som en del skyldtes den svært tilgængelige datamodel.

Mange af fejlene skyldtes dog funktionelle aspekter af programmet. Det faktum, at programmet ikke kan foretage en `OUTER JOIN` på en sammensat nøgle er en stor mangel. Den manglende mulighed for at anvende `OUTER JOIN` i udtræk, der involverer mere end to tabeller, må betegnes som utilfredsstillende. At man ikke kan anvende parenteser ved filtrering, men i stedet på forhånd skal vide, at `AND` præcederer over `OR` er ikke umiddelbart intuitivt. Det kan ligeledes undre, at programmet ikke understøtter anvendelse af `DISTINCT` eller `GROUP BY`. Førnævnte skyldes dog sandsynligvis, at programmet i sin opbygning fungerer som en database bestående af netop én tabel. Udviklerne har sandsynligvis

tænkt, at det vigtigste er at kunne hente data, hvorefter man kan foretage mere avancerede filtreringer eller sletning af dubletter.

En stor begrænsning ved programmet var størrelsen på wizardvinduet. Vinduet var meget småt og kunne ikke umiddelbart forøges i størrelse. Dette bidrog til at mindske overblikket for testpersonerne. Programmet var desuden ikke særligt selvforklarende.

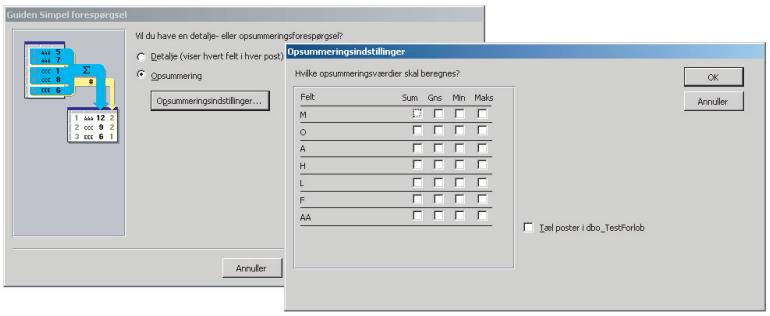
Nogle af forskerne i Videncenter for Allergi anvender *SPSS 17.0 for Windows* i stedet for den i testen anvendte version 15.0. En grundig inspektion af databasewizarden har dog vist, at der ikke er sket en udvikling af det rent funktionelle. Det vurderes således, at en anvendelse af denne version i forbindelse med de afholdte tests ikke ville give andre resultater end de, den anvendte version affødte.

Kapitel 5

Anden test

5.1 Access

I anden test anvendte testpersonerne applikationen *Microsoft Access 2002* til at udforme de tidligere beskrevne testscenarier. Applikationen er del af en kommerciel pakke af kontorprogrammer og anvendes oftest til at oprette og vedligeholde simple databaseløsninger, for eksempel medlemskartoteker eller musiksamlinger. Access stiller både visuelt og tekstbaseret forespørgselsgrænseflade til rådighed samt formularer til indtastning af data og mulighed for at udskrive rapporter. Den visuelle grænseflade består af to dele; dels en *guide* og dels en såkaldt *designvisning* til oprettelse af forespørgsler.

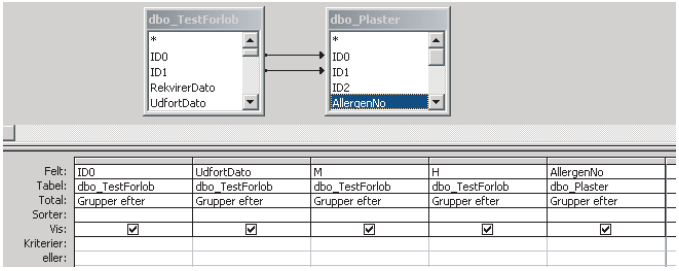


Figur 5.1: Anvendelse af aggregeringsfunktioner i *Guiden Simpel forespørgsel*.

I førstnævnte guide kan brugeren vælge mellem to forskellige slags forespørgsler; detalje- eller opsummeringsforespørgsel. Detaljeforespørgslen resulterer i en tabel, der indeholder de valgte kolonner fra de tabeller, der indgår i forespørgslen. Der kan tilsyneladende ikke filtreres i data; brugeren skal i stedet

oprette og gemme forespørgslen og kan siden via designvisning ændre forespørgslen, således at der filreres.

Opsummeringsforespørgslen giver brugeren mulighed for at anvende et par simple aggregeringsfunktioner (`SUM`, `AVG`, `MIN` og `MAX`) til at opsummere data fra de valgte kolonner, som afbildet i figur 5.1. Denne guide vil dog ikke blive brugt i tests, da brugeren som nævnt ikke kan filtrere på udviklingstidspunktet. Desuden fordrer brug af guiden, at indbyrdes relationer mellem tabellerne defineres én gang for alle, hvilket berøres senere i dette afsnit.



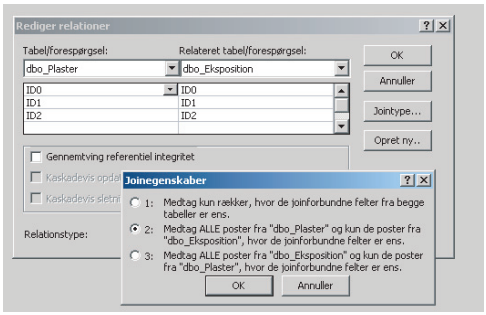
Figur 5.2: Eksempel på forespørgselsudformning via *designvisning*.

Anden del af den visuelle grænseflade består som nævnt af en designvisning til oprettelse af forespørgsler. Denne designvisning anvendtes i anden test til at udvikle dataudtræk svarende til scenarierne. Et eksempel på denne ses afbildet i figur 5.2

I designvisningen tilføjer brugeren først de relevante tabeller. Hvis der ikke først er defineret relationer mellem tabellerne appliceres automatisk `CROSS JOIN`. Brugeren kan eksplicit tilføje relationer, hvilke alle som udgangspunkt er `EQUI JOIN`, men senere kan ændres til de enten `LEFT` eller `RIGHT OUTER JOIN`, men ikke `FULL OUTER JOIN`. Derefter vælges kolonner — både de, de skal være del af det resulterende dataudtræk og de, der alene skal danne basis for opstilling af kriterier for udtrækket. Opstilling af kriterier foregår uden brug af parenteser og fungerer således, at Access automatisk indsætter `AND` mellem alle de betingelser, der angives på samme række horisontalt. Findes der flere rækker, vil programmet automatisk indsætte `OR` mellem rækkerne i den resulterende SQL-forespørgsel. Brugeren kan endvidere vælge at anvende aggregeringsfunktioner. Denne funktionalitet er dog som standard ikke aktiveret. Hvis den vælges, anvender programmet automatisk `GROUP BY` som aggregeringsfunktion, men brugeren kan blandt andre vælge funktionerne: `MIN`, `MAX`, `AVG`, `STDDEV`, `COUNT`, `SUM`, `FIRST` og `LAST`.

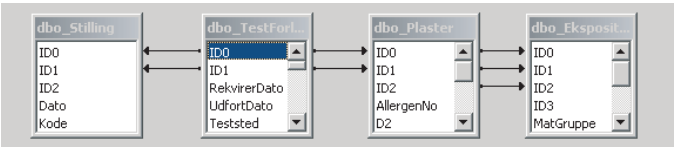
Den tidligere nævnte funktionalitet omkring forhåndsdefinition af relationer er principielt ganske praktisk. En udvikler kan således på forhånd fastlægge, hvilke tabeller der skal relateres via for eksempel `LEFT OUTER JOIN` i stedet for en `INNER JOIN`. Dette er meget relevant i forhold til den aktuelle data-

model, hvor brugerne hyppigt vil være nødt til at anvende netop `LEFT` eller `RIGHT OUTER JOIN`, da mange af tabellerne er knyttet sammen via en-til-nul-eller-flere-relationer. Således vil man i mange tilfælde på forhånd kunne fjerne behovet for, at brugerne skal skifte `JOIN`-operation under udvikling af en konkret forespørgsel.



Figur 5.3: I dette eksempel forhåndsdefineres relation (`LEFT OUTER JOIN`) mellem plastre og ekspositioner baseret på en sammensat nøgle.

Det er dog ikke helt problemfrit at udvikle forespørgsler, når relationerne mellem tabellerne er defineret på forhånd. Inddrager en given forespørgsel en række tabeller, der indbefatter to forskellige typer `JOIN`-operationer — for eksempel `INNER JOIN` og `LEFT OUTER JOIN` – fejler forespørgslen. Brugeren får en fejlmeddelelse, der opfordrer denne til at dele forespørgslen op i to. Dette vil potentielt være et betydeligt problem for brugerne.



Figur 5.4: Forespørgselsformulering ved brug af forhåndsdefinerede relationer, hvor de fire tabeller automatisk relateres korrekt med `OUTER JOIN`.

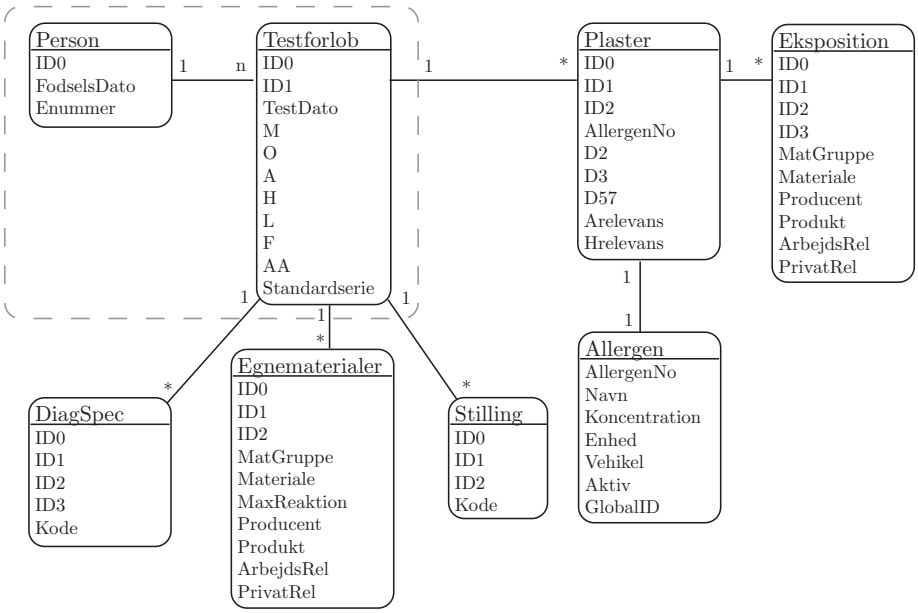
Et andet problem ved forhåndsdefiniton af relationer, der vil gøre sig gældende i forhold til nærværende datamodel, er de tilfælde, hvor brugerne skal ændre relationerne fra for eksempel `LEFT OUTER JOIN` til `INNER JOIN` på sammensatte nøgler. I tilfælde, hvor nøglerne er sammensat af for eksempel tre kolonner, skal brugeren eksplicit ændre hver enkelt del af relationen. Applikationen opfatter i dette tilfælde tilsyneladende ikke relationen som et hele, men snarere som tre enkeltstående `INNER JOIN`-operationer, og tvinger som nævnt brugeren til at ændre relationen en del ad gangen.

Dette er umiddelbart uforståeligt. Det giver ikke nogen mening at udføre forskel-

lige grundlæggende typer `JOIN`-operationer på forskellige dele af samme sammensatte nøgle. Forsøger brugeren at eksekvere forespørgslen uden at have ændret relationen for alle dele af den sammensatte nøgle, vil forespørgslen fejle. Dette må forventes at være en betydelig fejlkilde for brugerne.

På baggrund af ovenstående er forhåndsdefinition således fravalgt i forbindelse med anden test. Dette betyder i praksis, at brugerne selv skal tilføje alle relevante relationer mellem tabeller, når de udvikler forespørgsler.

I anden test blev datamodellen ændret i forhold til den originale model, der lå til grund for første test, idet en del for forsøgene irrelevante kolonner blev fjernet fra de tabeller, der indgik i scenarierne. Endvidere tilgik testpersonerne ikke databasen direkte på serveren, men et anonymiseret lokalt replikat, der bestod udelukkende af de for forsøgene relevante tabeller. Modellen er afbildet i sin helhed i figur 5.5. Der er ikke defineret primærnøgler for tabellerne, idet nøglerne ikke har nogen effekt på, hvordan tabeller relateres ved forespørgselsudformning i Access, sålænge der ikke er foretaget forhåndsdefinition af relationer.



Figur 5.5: Begrænset datamodel brugt i anden test.

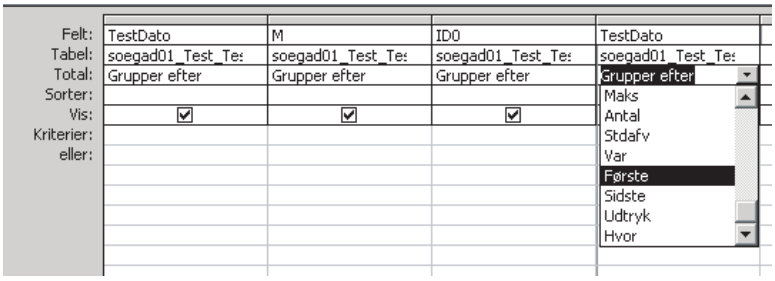
Ved import af data fra Microsoft SQL-databasen til Microsoft Access blev kolonner af typen `BIT` automatisk konverteret til Access' datatype, *Ja/Nej*. Som et kuriosum kan nævnes, at alle celler med værdien 1 i de originale data fejlagtigt blev konverteret til -1, hvilket ikke er en gyldig værdi i Microsoft Access, hvor 0 = *nej* og 1 = *ja*. Dette blev testpersonerne dog gjort opmærksom på, før anden test blev gennemført. Problemet er selvfølgelig ikke større, end at man

kan vælge at konvertere kolonner af typen `BIT` til `INTEGER` i en ny datamodel.

5.2 Resultater

Der var ingen større problemer med at udvikle dataudtræk svarende til testscenarier 1 og 2. Eneste undtagelse var, at testperson 1 valgte også at inkludere tabellen **Person**, som var overflødig for forespørgslen. Da personen samtidig glemte at angive relationen mellem de to tabeller, blev resultatet, at programmet automatisk foretog en `CROSS JOIN`. I testscenarie 3 kom testperson 1 til at anvende `OR` i stedet for `AND`, hvilket betød, at det resulterende dataudtræk indeholdt samtlige testforløb fremfor blot 2008-testforløb.

Testscenarie 4 blev ikke i første omgang løst med succes af nogen af testpersonerne. `DISTINCT`-funktionen var ikke umiddelbart tilgængelig, og selv om testperson 2 efter nogen søgen fandt `GROUP BY`-funktionaliteten, lykkedes det ikke denne at opnå et korrekt dataudtræk. Testperson 1 overraskede postitivt, idet hun ligeledes fandt `GROUP BY`-funktionaliteten, og valgte at forsøge at anvende aggregeringsfunktionen `FIRST` baseret på testforløbenes testdatoer, se figur 5.6.



Figur 5.6: Anvendelse af gruppér efter/aggregater i designvisning.

Testpersonen opnåede således at få et dubletfrit udtræk af personernes ID0, fødselsdato, køn samt første testdato, hvilket er tilfredsstillende. En anden sag er, at testpersonen ikke havde behøvet at anvende `FIRST`-funktionen, men i stedet blot havde anvendt `GROUP BY` baseret på de tre relevante kolonner, hvilket ville have resulteret i følgende SQL-forespørgsel:

```
SELECT    P.id0, P.fodselsdato, T.M
FROM      soegad01_Test_Person AS P INNER JOIN
          soegad01_Test_TestForlob AS T ON P.id0 = T.ID0
GROUP BY  P.id0, P.fodselsdato, T.M
```

Alle testpersonerne havde tilsyneladende problemer med at udføre korrekte `JOIN`-operationer i scenarie 5. Testperson 1 udførte korrekt en `LEFT OUTER JOIN` mellem de to tabeller, men glemte at betinge den af både ID0 og ID1 og opnåede således ikke det ønskede resultat. Testperson 2 glemte ligeledes, at

primærnøglerne i tabellerne **TestForlob** og **DiagSpec** er sammensatte, ligesom han glemte at anvende (`LEFT`) `OUTER JOIN`. Testperson 3 glemte først at relatere tabellerne, hvilket automatisk resulterede i en `CROSS JOIN`. Derefter anvendte han en `INNER JOIN` på kun ID0; derefter `INNER JOIN` på både ID0 og ID1.

Testscenarie 6, hvor der skal anvendes relativt svær filtrering, voldte også testpersonerne nogle problemer. Testperson 1 og 2 udførte dog efter en del *trial and error* korrekt filtrering i dataudtrækket. Der viste sig umiddelbart at være en del problemer forbundet med, at opstillede kriterier i applikationen — som tidligere nævnt — logisk forbindes med `AND`, når de står på samme linie og linierne som hele derefter logisk forbindes med `OR`.

Første forsøg på løsning af scenarie 6[1] udført af testperson 1 resulterede således i følgende konstellation:

		Stilling.Kode	Plaster.D2	Plaster.D3	Plaster.D57
	`AND`→	$= 51411$	≥ 1	≥ 1	≥ 1
`OR` ↓			< 4	< 4	< 4

Testpersonen indså, at dette resulterede i, at udtrækket indeholdt alle frisører, der havde en reaktionskode ≥ 1 på alle tre testdage samt personer fra alle erhverv, der havde reaktionskoder < 4 på alle tre testdage. Næste forsøg ses nedenfor:

		Stilling.Kode	Plaster.D2	Plaster.D3	Plaster.D57
	`AND`→	$= 51411$	≥ 1 `AND` < 4	≥ 1 `AND` < 4	≥ 1 `AND` < 4
`OR` ↓					

Dette resulterede i, at udtrækket indeholdt alle frisører, der havde positive reaktioner på samtlige testdage, hvilket også var forkert. Testpersonen omformede derfor til nedenstående:

		Stilling.Kode	Plaster.D2	Plaster.D3	Plaster.D57
	`AND`→	$= 51411$	≥ 1 `AND` < 4		
`OR` ↓				≥ 1 `AND` < 4	
					≥ 1 `AND` < 4

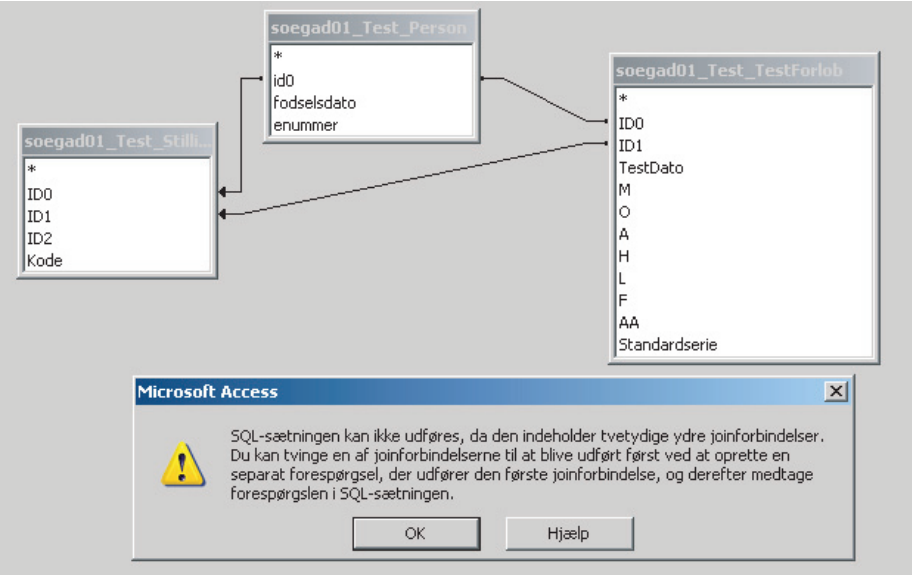
Testpersonen fik således et udtræk, der indeholdt alle frisører, der havde positiv reaktion på dag 2 samt personer fra alle erhverv, der havde positive reaktioner på dag 3 eller dag 5-7. Dette var også forkert, hvorfor testpersonen indså, at kriterierne måtte opstilles som følger:

[1] I testscenariet skulle filtereres således, at alle frisører (stillingskode = 51411) med positiv reaktion (reaktionskode = 1, 2 eller 3) på mindst én af de tre reaktionsdage D2, D3, D57 blev valgt.

		Stilling.Kode	Plaster.D2	Plaster.D3	Plaster.D57
	AND→	= 51411	≥ 1 AND < 4		
OR ↓		= 51411		≥ 1 AND < 4	
		= 51411			≥ 1 AND < 4

Testperson 2 var den eneste, der lykkedes med at foretage et korrekt dataudtræk på baggrund af testscenarie 7. De to øvrige havde store problemer; primært i forbindelse med at definere relationerne mellem tabellerne, hvor der både skulle bruges `INNER JOIN` og (`LEFT`) `OUTER JOIN`. Ligeledes var der problemer med at huske at betinge relationerne af hele nøglen i de tilfælde, hvor tabeller med sammensatte nøgler skulle forbindes.

Testperson 1, der i dette scenarie benyttede trial and error til trinvist at opbygge sit dataudtræk, kom allerede ved forsøg på forbindelse af tredje tabel galt afsted, idet hun som afbildet i figur 5.7 forsøgte at lave en `LEFT OUTER JOIN`, der var betinget af sammensatte relationer til to forskellige tabeller.

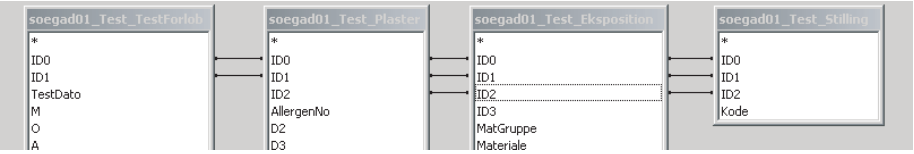


Figur 5.7: Mislykket forsøg på `JOIN`-operation betinget af sammensatte relationer til forskellige tabeller.

Access tillader tilsyneladende ikke denne måde at forbinde tabeller på, omend det — til trods for, at det ikke ser kønt ud — ikke formelt er forkert. Nedenstående udtræk vil således kunne valideres og afvikles med succes på flere af Microsofts databaseprodukter:

```
SELECT  P.ID0, T.UdfortDato, S.Kode
FROM    dbo.Person P INNER JOIN
        dbo.TestForlob T ON P.ID0 = T.ID0 LEFT OUTER JOIN
        dbo.Stilling S ON P.ID0 = S.ID0 AND T.ID1 = S.ID1
```

Testperson 1 havde endvidere problemer med at forbinde tabellerne **Plaster** og **Eksposition** korrekt, idet hun anvendte en `RIGHT OUTER JOIN`, og således kun fik de rækker med i sit udtræk, hvor der fandtes en række i tabellen eksposition. Testperson 3 stillede sine tabeller op på en visuelt uhensigtsmæssig måde, hvilket affødte, at han fejlagtigt kom til at relatere kolonnerne ID2 i tabellerne Eksposition og Stilling (figur 5.8).



Figur 5.8: Forkert relation mellem tabeller grundet visuelt uhensigtsmæssig opstilling af tabeller.

5.3 Diskussion

Overordnet gik anden serie af tests godt, men der viste sig flere tydelige problemer. Alle testpersonerne oplevede — lige som i første test — på den ene eller anden måde og i varierende grad at have svært ved at

1. tilføje korrekte relationer mellem tabeller med `JOIN`-operationer.
2. foretage svær filtrering af data ved hjælp af simpel logik.
3. anvende aggregeringsfunktioner.

Dette kommer for så vidt ikke som en overraskelse, da det i et historisk perspektiv tradionelt har vist sig at være problematiske emner for ikke-professionelle slutbrugere af databasesystemer.

Testpersonerne syntes at have større problemer med at anvende `JOIN`-operationer i anden test end i første test. Dette skyldes formodentlig, at applikationen (SPSS) i første test selv automatisk oprettede relationer mellem tabellerne, mens Access' funktionalitet omkring forhåndsdefinition af relationer i anden test blev udeladt, hvorfor brugerne selv eksplicit måtte tilføje alle relationer. Hvor SPSS' anvendelse af `NATURAL JOIN` tilføjede for mange relationer mellem enslydende kolonner i forskellige tabeller, der ikke skulle relateres, sikrede SPSS omvendt, at brugerne ikke glemte at relatere dele af sammensatte

nøgle, hvilket var tilfældet i nærværende test. Det kan altså umiddelbart tyde på, at det giver bedre resultater, at en applikation selv foreslår relationer — selv om nogle af disse er fejlbehæftede — frem for ikke at tilføje nogen.

I udviklingen af en endelig datamodel, der understøtter korrekt forespørgselsudformning, er det således vigtigt at tage højde for, hvordan den applikation, der skal bruges til at udforme forespørgslerne, automatisk relaterer tabeller. Anvender applikationen for eksempel `NATURAL JOIN`, vil det for flere tabellers vedkommende være relevant at ombenævne kolonner, der ikke indbyrdes er relateret og derfor ikke skal danne basis for en `NATURAL JOIN`. Omvendt vil de i de tilfælde, hvor de anvendte applikationer automatisk tilføjer `INNER JOIN` eller `EQUI JOIN` på dele af — men ikke hele – den sammensatte nøgle, være relevant at søge at samle den sammensatte nøgle i én kolonne.

Det er endvidere vigtigt at holde sig for øje, at applikationerne kan have problemer med forespørgsler, der involverer samtidig anvendelse af flere forskellige typer `JOIN`-operationer; for eksempel `INNER JOIN` mellem to tabeller og `LEFT OUTER JOIN` mellem to andre tabeller i samme forespørgsel. For Access' vedkommende opstod problemer med `JOIN`-operationer betinget af sammensatte relationer til forskellige tabeller, lige som der ved test med SPSS opstod problemer med `OUTER JOIN`-operationer på sammensatte nøgler.

I Access, hvor der er større fokus på den visuelle brugergrænseflade end i SPSS, er det stadig relativt problematisk for testpersonerne at definere kriterier for filtrering korrekt ved brug af `AND`, `OR` og parenteser. Dette til trods for, at kriterierne indtastes i en overskuelig tabel. Resultater synes mest drevet af trial and error og tilfældigheder. Det kan derfor tyde på, at en plausibel løsning vil være at forsøge at definere sig ud af de mest oplagte faldgruber via en opdateret datamodel.

Kapitel 6

Databaseforespørgsler

I dette kapitel afdækkes, hvorvidt en ny datamodel, der inddrager erfaringer fra de tidligere tests og som består af en række views på databasen, vil være en plausibel løsning i forhold til slutbrugerudvikling af databaseforespørgsler.

Først beskrives vigtige dele af udviklingen inden for manipulations- og forespørgselssprog samt forsøg på at gøre sprogene mere tilgængelige for brugere via visuelt orienterede grænseflader. Siden diskuteres relevans og anvendelsesmuligheder af de beskrevne områder for nærværende specialeprojekt. Slutteligt udvikles tre views, der danner basis for en afprøvning af forespørgselsudvikling ved hjælp af det professionelle værktøj, Microsoft SQL Server 2008 Management Studio. Afprøvningen skal således afdække, hvorvidt en kombination af en ny datamodel og et professionelt visuelt orienteret værktøj vil være en brugbar løsning.

6.1 Forespørgselssprog

6.1.1 Query-by-Example

Query-by-Example (QBE) er et databasesprog — udviklet ved IBM i 1970'erne af Zloof [34] — der som flere øvrige databasesprog tillader oprettelse, opdatering og forespørgsel af informationer i relationelle databaser. QBE har en grafisk grænseflade, der forsyner brugeren med en slags tomt forespørgselsskelet, som brugeren kan fylde ud med forskellige værdier. Sproget adskiller sig fra andre databasesprog, idet brugeren således ikke selv behøver at kende til den konkrete sproglige syntaks, men blot skal "udfylde blanke felter" med *eksempler*. På baggrund af de udfyldte felter genereres fra systemets side en forespørgsel, hvor eksemplerne er betingelser.

Query-by-Example anses af mange for at være et af de tidligste eksempler

på slutbrugerudvikling, og nævnes derfor hyppigt i den sammenhæng. Selve Query-by-Example er et IBM-trademark, men mange applikationer anvender i dag QBE-lignende grafiske brugergrænseflader i forsøg på at gøre udformningen af forespørgsler mere intuitiv [28]. Stort set alle visuelt orienterede forespørgselsapplikationer eller -wizards anvender dette til overskuelig angivelse af betingelser til den forespørgsel, der er under udformning.

QBE kan endvidere bruges som et effektivt søgeværktøj ved hjælp af domænerelationel kalkyle (eng: *domain relational calculus*[1]) for eksempel til tekstsøgning. Senest er også set eksempler på søgning i billeder baseret på deres grafiske komposition, kaldet Query-by-Visual-Example (QBVE) [1].

6.1.2 Structured Query Language

Structured Query Language (SQL) er et deklarativt forespørgselssprog, der lige som for Query-by-Examples vedkommende primært bygger på forskning og udvikling foretaget ved IBM i 1970'erne. Sproget var i sin originale form designet til at understøtte forespørgsler og manipulation af data i IBM's eget databasesystem, men er i dag et af de mest anvendte forespørgselssprog til databaser. Sproget er baseret på relationel algebra og var et af de første til at understøtte Codds relationelle model [8].

SQL har siden vundet stor udbredelse og understøttes af stort set alle relationelle DBMS'er. Der findes af samme grund flere forskellige dialekter af SQL, hvilket har affødt en række standarder, hvor ANSI[2] SQL-86 var den første og SQL:2008 den seneste. På trods af standardiseringen kritiseres SQL ofte for ikke at være fuldt platformuafhængig, idet de forskellige DBMS'er kun i varierende grad understøtter den fulde — og meget komplekse — standard. Data håndteres ikke ensartet på tværs af de forskellige platforme, hvilket for eksempel gør sig gældende i forbindelse med syntaks for dato og tid lige som `NULL`[3] håndteres uensartet.

Den hyppigst anvendte funktionalitet i SQL er forespørgslen, der henter data fra en eller flere tabeller eller matematiske udtryk, er opbygget som følger:

[1] Domænerelationel kalkyle er en deklarativ metode til formulering af forespørgsler inden for den relationelle databasemodel. DRC skal ses i kontrast til relationel algebra, som er en mere procedurel metode til at formulere databaseforespørgsler. Kilde: `http://en.wikipedia.org/wiki/Relational_calculus`

[2] American National Standards Institute. Amerikansk non-profitorganisation, der står for udviklingen af frivillig enighed omkring standarder for blandt andet processer, produkter og services.

[3] `NULL` er en pseudoværdi, der bruges i databaser til at indikere, at en given værdi er uudfyldt. Hvis en kolonne i en tabel for eksempel kan have værdien *ja* eller *nej*, vil en forekomst af værdien `NULL` i kolonnen indikere, at kolonnen for den aktuelle rækkes vedkommende hverken er *ja* eller *nej* men altså uudfyldt.

`SELECT`: *Felter eller værdier, der skal hentes.*

`FROM`: *Den eller de tabeller, der skal hentes data fra. Ved flere tabeller anvendes endvidere* `JOIN`*-operatoren som del af* `FROM`*.*

`WHERE`: *Kriterier for de data, som forespørgslen resulterer i ved hjælp af sammenligninger.*

`GROUP BY`: *Samling af rækker med ens værdier i et mindre antal rækker. Anvendes ofte sammen med aggregeringsfunktioner eller til fjernelse af dubletter.*

`HAVING`: *Kriterier for filtrering af de data, som* `GROUP BY` *resulterer i. Aggregeringsfunktionerne anvendes her.*

`ORDER BY`: *Rækkefølge for sortering af de resulterende data.*

Ud over forespørgsler, der anvendes til at læse i eksisterende data, kan SQL groft inddeles i følgende underkategorier:

- Data Definition Language (DDL): *Dette anvendes til at styre tabeller og indices, det vil sige for eksempel oprettelse, ændring, omdøbning og sletning af dele af eller hele tabeller eller indices (kolonnevis).*
- Data Manipulation Language (DML): *Dette anvendes til at tilføje, opdatere eller slette data i tabeller eller indices (rækkevis).*
- Data Control Language (DCL): *Dette anvendes til at styre brugeres eller brugergruppers adgang til at læse og skrive data.*

6.1.3 Andre forespørgsselssprog

Gennem tiden er der set mange forsøg på at udvikle manipulations- og forespørgselssprog til databaser, hvoraf en stor del — nogle domænespecifikke sprog undtaget — er udsprunget af forsøg på at gøre manipulation og forespørgsler på baggrund af databaser mere intuitive og tilgængelige for en bredere skare af brugere end blot professionelle databaseudviklere.

Således er der blandt andet udviklet sporg, der baserer sig på anvendelse af naturlige sprog, såsom ERROL[4], der er et deklarativt manipulations- og forespørgselssprog skræddersyet til relationelle databaser. ERROL er baseret på, at forespørgeren i forbindelse med formulering af forespørgslen anvender *Structured English*[5] til at beskrive de data, denne ønsker at hente fra databasen. Ved

[4] **E**ntity **R**elationsship **R**ole **O**riented **L**anguage.

[5] Begrebet Structured English henviser til brugen af det engelske sprog under anvendelse af syntaks kendt fra struktureret programmering. Structured English tilstræber således at drage fordel både af struktur og logik kendt fra programmeringssprog og naturligt sprog for tilgængeligheden.

hjælp af kendskab til ER-modellen[6] for den pågældende database kan en sætning udtrykt i naturligt sprog af ERROL automatisk omdannes til en formaliseret forespørgsel.

Et andet interessant sprog er Object Query Language (OQL), der — som navnet antyder — bygger på en objektorienteret tilgang til data. Forespørgsler defineres ved hjælp af objekter og attributter, lige som resultater af forespørgslerne returneres som objekter. OQL kan enten anvendes indlejret på en eksisterende relationel database til at formulere forespørgsler eller som enkeltstående komplet sprog til design og oprettelse af databaser samt efterfølgende manipulation og forespørgselsformulering. I OQL kan brugeren ved hjælp af et semantisk diagram over databasens objekter og klasser af objekter formulere en forespørgsel. Relevante objekter vælges ved at definere betingelser for det enkelte objekts attributter og betingelser for samling på tværs af objekter ved at sammenligne de pågældende objekters attributter.

Empiriske studier af nybegynderes evne til at formulere forespørgsler samt at forstå eksisterende forespørgsler indikerer, at det er mere intuitivt og mere effektivt at anvende OQL frem for SQL samt at definere en datamodel ved hjælp af objektorienteret modellering end ved hjælp af den relationelle model [5]. Dette sprog vil potentielt kunne bidrage til at gøre databaser mere tilgængelige for slutbrugere, men er grundet sin kompleksitet endnu ikke fuldt ud understøttet af noget DBMS.

XQuery er endnu et af de nyere interessante forespørgselssprog. Sproget er designet med henblik på at foretage forespørgsler i data lagret i XML-format[7]. XQuery, der understøttes af de fleste større databasehåndteringssystemer, kan således anvendes til at ekstrahere entiteter og attributter fra XML-dokumenter. Sproget minder for så vidt en del om SQL, idet det bygger på sproget XPath[8] kombineret med forespørgsler udtrykt på FLWOR-formen: `for`, `let`, `where`, `order by` og `return`.

[6] Entitet-relationsmodellen. En meget anvendt model til design af databaser beskrevet i 1976 af PPS Chen [7]. Modellen bygger på, at der findes en række entiteter (tabeller) indeholdende en række attributter (kolonner). Entiteterne er forbundet via relationer af forskellig kardinalitet for eksempel én-til-én eller én-til-mange.

[7] E**x**tensible **M**arkup **L**anguage er et struktureret opmærkningssprog, der blandt andet anvendes på websider og til udveksling af information mellem computere. Mange databasehåndteringssystemer kan gemme eller eksportere databaser til XML med henblik på at opnå platformuafhængig kommunikation af data.

[8] XPath (XML Path Language) er et forespørgselssprog beregnet på at ekstrahere data fra XML-dokumenter og kan endvidere anvendes til at foretage beregninger på baggrund af disse data. Kilde: `http://en.wikipedia.org/wiki/XPath`

6.2 Visuelle grænseflader

Siden de første forespørgselssprog til relationelle databaser så dagens lys, er der sket en stor udvikling inden for udbredelsen af databaser, visuelle brugergrænseflader og slutbrugeres anvendelse af begge førnævnte. Dette har — sammen med en stigende erkendelse af, at databasemodeller som den relationelle og sprog som SQL er utilgængelige for de fleste slutbrugere — affødt en del forsøg på at gøre databasemodellering og forespørgsler mere tilgængelige via grafisk orienterede visuelle grænseflader.

I [4] findes en grundig gennemgang af de forskellige tilgange til visuelle grænseflader, som er set gennem tiden, baseret på en klassifikation af systemer i henholdsvis formular-, diagram-, ikonbaserede eller hybrider heraf. De formularbaserede grænseflader var det første skridt væk fra den tekstbaserede tankegang og er baseret på en todimensionel grafisk repræsentation af tabeller. Query-by-Example er et eksempel på dette, mens entitet-relationsdiagrammer (ER) er eksempel på en diagrambaseret grænseflade.

	Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...	Or...
▶	ID0		TestForlob...	☑	Ascending	2			
	ID1		TestForlob...	☑					
	UdfortDato		TestForlob...	☑	Descending	1	>= '01-01-2002'	>= '01-01-2002'	
	Teststed		TestForlob...	☑					
	M		TestForlob...	☑			= 1	= 1	
	O		TestForlob...	☑					
	H		TestForlob...	☑			= 1		
	F		TestForlob...	☑				= 1	

```
SELECT    TOP 100 PERCENT ID0, ID1, UdfortDato, Teststed, M, O, H, F
FROM      dbo.TestForlob
WHERE     (M = 1) AND (H = 1) AND (UdfortDato >= CONVERT(DATETIME, '2002-01-01 00:00:00', 102)) OR
                (M = 1) AND (UdfortDato >= CONVERT(DATETIME, '2002-01-01 00:00:00', 102)) AND (F = 1)
ORDER BY UdfortDato DESC, ID0
```

Figur 6.1: Eksempel på QBE-lignende grænseflade i Microsoft SMSS.

Ikonbaserede grænseflader er baseret på, at brugeren under både design af datamodel og formulering af forespørgsler flytter forskellige ikoner, der repræsenterer datamodellens tabeller, rundt på skærmen, uden egentlig at kende til selve datamodellen. Funktioner er ligeledes repræsenteret ved ikoner; disse funktioner anvendes ved, at ikoner kombineres, ligesom relationer repræsenteres ved kombination af ikoner.

Førnævnte type grænseflade synes ikke at have fundet særlig udbredelse, mens idéer af den formularbaserede tilgang har vundet bredt indpas i mange nutidige databaseapplikationer; et eksempel på dette er afbildet i figur 6.1, hvor brugeren under en forespørgselsformulering har mulighed for at opstille betingelser for forespørgslen op i en slags formular. Ligeledes kan i de fleste applikationer anvendes formularer som grafisk værktøj ved oprettelse af tabeller, hvor brugeren for eksempel kan indtaste datatype for tabellens kolonner frem for at angive det i for eksempel SQL.

De diagrambaserede grænseflader findes ligeledes i mange databaseapplikationer, både under design af datamodel og formulering af forespørgsler. Brugeren vil typisk i begge tilfælde oprette et diagram over alle tabeller, der skal indgå, og derefter relatere tabellerne indbyrdes.

6.3 Diskussion

Som beskrevet i afsnit 6.1 er der gennem tiden udviklet mange forskellige forespørgselssprog beregnet på databaser. Query-by-Example anvendt som selvstændigt forespørgselssprog har ikke vundet stor udbredelse. Studier i forespørgselsudformning har vist, at brugere af QBE yder bedre end brugere af SQL ved anvendelse af papir og blyant, mens der ikke er nogen forskel, når forespørgslerne udformes online [33]. Sidstnævnte er påvist i tidligere studier, blandt andet i [2]. Det et dog værd at bemærke, at QBE stiller en grafisk grænseflade til rådighed, hvilket *har* vundet bred udbredelse, idet en sådan grænseflade er integreret i stort set alle grafiske forespørgselsværktøjer.

Senere studier, der sammenligner forskellen på evnen til at formulere forespørgsler og forstå eksisterende forespørgsler, indikerer, at nybegyndere yder bedre ved anvendelse af OQL end SQL [5], men dette sprog er som tidligere nævnt endnu ikke fuldt understøttet af noget DBMS og er derfor ikke umiddelbart tilgængeligt til anvendelse i dette projekt.

Anvendelse af naturlige sprog til formulering af forespørgsler er heller ikke plausibel i nærværende projekt. Et studie, der sammenligner forskelle på brugeres evner til at formulere forespørgsler i henholdsvis formelle og naturlige sprog, konkluderer, at det for lavfrekvente brugere generelt er svært at formulere korrekte forespørgsler, uagtet om der anvendes formelle eller naturlige sprog [23]. Samtidig bemærkes det, at en væsentlig del af de forsøg, der gennem tiden er gjort på at tilvejebringe systemer, der tillader udformning af forespørgsler via naturlige sprog, synes at være løbet ud i sandet.

Sproget XQuery, der finder bred anvendelse i både dokumenter og databaser inden for mange domæner og på flere platforme, skønnes heller ikke umiddelbart oplagt at anvende i forbindelse med dette specialeprojekt, da løsningen ikke antages at ligge i introduktionen af endnu et forespørgselssprog, men i stedet i at anvende en forespørgselsapplikation med en visuel grænseflade. Samtidig skal en del af løsningen findes i en omstrukturering af datamodellen, da brugernes kendskab til datamodellen spiller en stor rolle for deres evne til at formulere korrekte forespørgsler. Hvis datamodellen ikke er, som brugeren antager, vil brugeren fejle i sine forsøg på at formulere forespørgsler [30].

Tilbage står således, at en mulig løsning på problemet vil være at mindske og omstrukturere den eksisterende datamodel, så den ligner det, brugerne forventer. Den database, der arbejdes med i dette projekt, opfattes af brugerne som én stor tabel, der indeholder alle data på casebasis — altså én række i tabellen

per testforløb. Dette stemmer dårligt overens med en normaliseret relationel database, hvilket understreger vigtigheden af at indrette datamodellen, så den i højere grad korresponderer med brugernes opfattelse og behov.

Resultaterne fra første og anden test (kapitel 4 og 5 henholdsvis) understreger, at testpersonerne specielt har problemer med at anvende `JOIN`-operationer og logik i filtrering korrekt. Dette er for så vidt ingen overraskelse, idet talrige studier har udpeget netop disse; blandt andre [31, 32, 25].

En essentiel del af løsningen er således at forsøge at minimere antallet af `JOIN`-operationer og dermed nedbringe det potentielle antal fejl. En prækalkulering af visse data er ligeledes nødvendig for at mindske antallet af fejl ved anvendelse af logik til filtrering. Samtidig vil introduktionen af en applikation, der ikke har de indbyggede fejl, som applikationerne anvendt i første og anden test havde, med stor sandsynlighed bidrage til, at brugerne med succes kan foretage korrekte forespørgsler på baggrund af testscenarierne.

6.4 Afprøvning

I denne afprøvning søges — uden inddragelse af testpersonerne — belyst, hvorvidt en datamodel baseret på overvejelserne i ovenstående diskussion kombineret med erfaringer fra første og anden test kan danne rammer for en løsning på specialets problemstilling. Afprøvningens formål er således at afklare, om brugerne potentielt kan formulere korrekte forespørgsler under forudsætning af følgende tiltag:

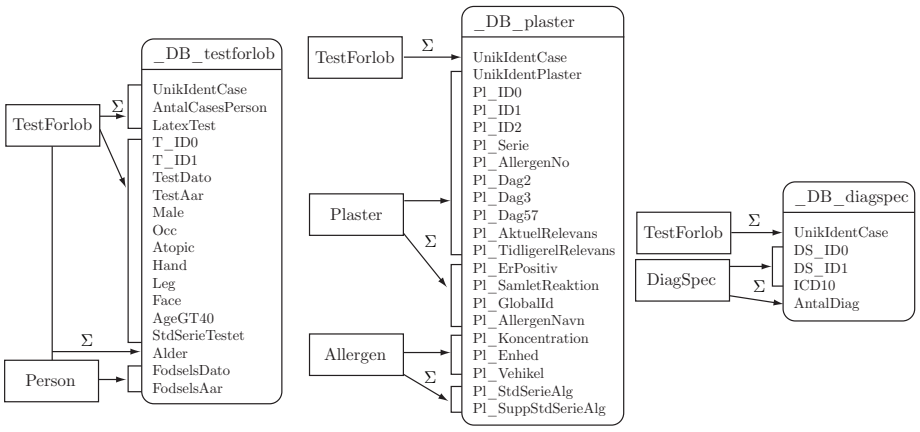
- Samling af tabeller minimerer antallet af `JOIN`-operationer.
- Antallet af sammensatte primærnøgler minimeres.
- Behov for `OUTER JOIN` mindskes ved oprettelse af nye *dummy-rækker*, der kun indeholder unik nøgle og `NULL`-kolonner.
- Kolonner, der typisk skal filtreres på, aggregeres/prækalkuleres.
- Enslydende kolonner i forskellige tabeller omdøbes.
- Programmel, der stiller QBE-lignende grænseflade til rådighed, anvendes.

Til dette formål formuleres tre views på databasen. Det ene view samler de to tabeller **Person** og **TestForlob** samt nogle beregninger og aggregeringer i én tabel. Viewet har en kolonne, UnikIdentCase, der som navnet antyder unikt identificerer det enkelte testforlob. Kolonnen er en konkatenering af kolonnerne ID0 og ID1 fra tabellen **TestForlob** og svarer altså til denne tabels primærnøgle.

Det andet view er en samling af tabellerne **Plaster** og **Allergen**, nogle aggregeringer heraf samt en kopi af kolonnen UnikIdentCase fra førstnævnte view.

Desuden findes kolonnen UnikIdentPlaster, som er en konkatenering af kolonnerne ID0, ID1 og ID2 fra tabellen **Plaster** og altså modsvarer denne tabels primærnøgle.

Det tredje view indeholder nogle kolonner fra den originale tabel **DiagSpec**, den tidligere omtalte kolonne UnikIdentCase samt en optælling af antallet af diagnoser per testforløb. Kildekoden til disse views findes i bilag D.



Figur 6.2: Oversigt over de tre views, der blev udviklet til afprøvningen.

I den originale datamodel er den minimale datamængde én række i hver af tabellerne **Person** og **TestForlob**. Der vil for et givent testforløb derfor ikke nødvendigvis findes korresponderende rækker i hverken **Plaster** eller **DiagSpec**, men da begge sidstnævnte views som tidligere nævnt indeholder kolonnen UnikIdentCase, der er oprettet på baggrund af tabellen **TestForlob**, vil en del af rækkerne — én per berørt testforløb — være dummy-rækker og således kun bestå af kolonnen UnikIdentCase og `NULL`-kolonner. Dette er et bevidst designvalg, da de tidligere tests viste, at brugerne havde problemer med at udføre korrekte `OUTER JOIN`-operationer.

I afprøvningen anvendes applikationen Microsoft SQL Server Management Studio (SSMS), som beskrives nærmere i afsnit 8.2. Applikationen anvender diagramvisning af tabeller ved forespørgselsformulering og stiller som tidligere påkrævet en QBE-lignende grænseflade til rådighed i forbindelse med blandt andet filtrering og sortering.

6.4.1 Resultater

Testscenarie 1, 2 og 3 er umiddelbart mulige at realisere alene ved hjælp af viewet _DB_testforlob. Dog opnår man i scenarie 1 de relevante kolonner for alle testforløb i databasen og ikke for kun for alle personer. Dermed får man

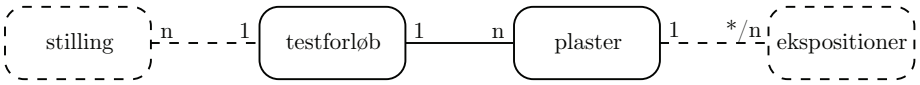
en række dubletter. I scenarie 3 kan man vælge data fra et bestemt år, da der i viewet er oprettet kolonnen TestAar, som kan fri brugerne for at anvende datoekstraktionsfunktioner eller anvende filtrering, der kræver kendskab til databasespecifikt format for datoangivelser.

Testscenarie 4 kan ligeledes realiseres, men også i dette tilfælde får man nogle dubletter. Dubletter i de første fire scenarier kan sorteres enten ved hjælp af `DISTINCT` — som brugerne ikke kan forventes at anvende — eller `GROUP BY`, hvilket vil blive berørt nærmere i diskussionen af denne afprøvnings resultater (afsnit 6.4.2).

Scenarie 5 kan udføres komplet ved hjælp af de to views _DB_testforlob og _DB_diagspec, som korrekt samles via en `INNER JOIN`, hvilket er muliggjort via oprettelse af de tidligere omtalte dummy-rækker.

Man kan ved hjælp af de tre views ikke skabe en korrekt forespørgsel svarende til testscenarie 6. Dog kan de tre views sandsynliggøre, at en korrekt forespørgsel kan formuleres, hvis endnu et view indeholdende stillinger var oprettet. Man kan uden videre sidestille filtrering af stillingskoder med filtrering af diagnosekoder, såfremt et givent stillingsview indeholder mindst én række per testforløb, hvoraf flere af disse potentielt vil være dummy-rækker. Filtrering af positive reaktioner er gjort simpel, idet kolonnen Pl_ErPositiv opsummerer alle tre reaktionsdage. En bruger skal således blot stille som kriterie, at Pl_ErPostiv = 1 frem for at undersøge, hvorvidt $1 \leq D2 \leq 3$ eller $1 \leq D3 \leq 3$ eller $1 \leq D57 \leq 3$, hvilket viste sig at være et problem i de første tests.

Testscenarie 7, der er det mest komplicerede, kan heller ikke umiddelbart realiseres ved hjælp af de tre udviklede views. For det første er der ikke udviklet et view indeholdende stillinger. Det anses dog i henhold til overvejelserne i forbindelse med testscenarie 6 sandsynliggjort, at et sådant view vil virke efter hensigten. For det andet er der heller ikke oprettet et view, der indeholder ekspositioner.



Figur 6.3: Oversigt over implicerede tabeller/views og relationer ved formulering af forespørgsel svarende til testscenarie 7.

I den originale datamodel og den virkelighed, den beskriver, findes en én-til-nul-eller-flere-relation mellem plastre og ekspositioner, hvilket afføder nogle væsentlige designmæssige overvejelser i forbindelse med oprettelse af et view over ekspositioner. Disse overvejelser beskrives nærmere i afsnit 6.4.2. Hvad angår filtreringen af data i scenariet, vil denne kunne udføres simpelt ved hjælp af to kolonner.

6.4.2 Diskussion af resultater

Afprøvningen af forespørgselsudformning på baggrund af de udviklede views har været meget frugtbar, idet den dels har sandsynliggjort, at en nydesignet datamodel kan udgøre en brugbar løsning på specialets problemstilling og dels har påvist fejl/mangler ved de views, der er udviklet i forbindelse med denne afprøvning. Samtidig giver afprøvningen anledning til en række væsentlige designmæssige overvejelser, der er nødvendige at tage stilling til, inden en given løsning kan bringes i drift.

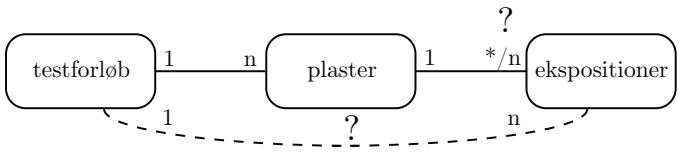
Første væsentlige erfaring ved afprøvningen var, at det ikke er plausibelt at anvende views i en given implementering, idet de er for langsomme — alt afhængig af filtreringsgrad. Det mest beregningstunge og omfangsrige view, _DB_plaster, indeholdt over 1,1 millioner rækker og var ufiltreret meget langsomt i anvendelse. I praksis blev alle views i denne afprøvning derfor gemt i tabeller. Tabellen svarende til _DB_plaster tog over seks minutter at bygge. En øvrig grund til, at en given implementering med fordel kan bestå af tabeller frem for views, er, at en del programmer kun udfører automatiske `JOIN`-operationer, hvis der findes mindst én primærnøgle i de valgte views/tabeller. Det anses derfor for væsentligt, at den endelige testforløb-tabel indeholder primærnøglen UnikIdentCase. Alle øvrige personbundne tabeller skal ligeledes indeholde kolonnen UnikIdentCase, hvor den dog ikke vil være primærnøgle. Da testforløb-tabellen er central for hele databasen, er det naturligt, at tabellen vil indgå i stort set samtlige forespørgsler, brugerne vil udvikle. I de tilfælde, hvor tabellen ikke inkluderes i en forespørgsel, vil tilstedeværelsen af UnikIdentCase i samtlige personbundne tabeller sikre, at al data, der logisk er knyttet til hinanden, stadig kan relateres korrekt.

Et andet problem, der viste sig ved afprøvningen, var håndteringen af dubletter. Dette problem kan enten løses ved anvendelse af `DISTINCT` — hvilket brugerne antages ikke at have kendskab til — eller `GROUP BY`-funktionaliteten, som dog erfaringsmæssigt kan være svær at anvende. Dette dels på grund af forståelsesmæssige barrierer og dels på grund af, at SSMS foretager automatisk valg af aggregeringsfunktion, som i bestemte tilfælde håndteres uhensigtsmæssigt, hvilket kan skabe unødige og for brugerne potentielt uløselige problemer. Dette berøres nærmere i afsnit 8.2.

Det væsentligste spørgsmål, der rejste sig ved afprøvningen var, hvordan modellen vil fungere ved forespørgsler, der implicerer tabeller relateret via én-til-nul-eller-flere-relationer, hvor brugerne af databasen altså skal anvende `OUTER JOIN` for meningsfyldt at kunne samle tabellerne uden tab af data. I den centrale del af den originale datamodel (figur 3.1) findes fem personbundne én-til-nul-eller-flere-relationer, hvoraf den eneste tabel, der er berørt af disse relationer, og som ikke er relateret direkte til tabellen **Testforlob**, er tabellen **Eksposition**, som er relateret via tabellen **Plaster**.

I nærværende afprøvning blev tabellerne indeholdende data fra **Testforlob**- og **Plaster**-tabellerne relateret via en én-til-flere-relation, idet der blev oprettet

dummy-rækker i viewet/tabellen _DB_Plaster. Ligeledes er det sandsynliggjort, at data fra tabellerne **DiagSpec**, **Stilling** og **Egnematerialer** kan relateres meningsfyldt, således at brugerne slipper for problemer med `INNER/OUTER JOIN` uden at miste data.



Figur 6.4: Mulige relationer mellem testforløb, plastre og ekspositioner.

Tilbage står overvejelser omkring, hvordan data fra tabellen **Eksposition** skal relateres til de eksisterende views/tabeller. Logisk set kan der for et givent plaster være registreret en eller flere ekspositioner, men i de fleste tilfælde, vil der ikke være registreret nogen, da der ikke har været en positiv reaktion. Der findes aktuelt over 1,1 millioner rækker i **Plaster**-tabellen, mens der findes omkring 3.500 rækker i tabellen **Eksposition**. Tabellerne skal dog nødvendigvis relateres på plaster-/allergenniveau, hvis brugerne skal opnå forespørgsler, hvor ekspositionerne er knyttet til de rigtige plastre. Der findes flere potentielle løsninger på dette (figur 6.4):

1. **Ekspositioner integreres i tabellen indeholdende plastre.** Dette er en dårlig løsning, idet der for testforløbene som nævnt findes et variabelt antal ekspositioner. For hver eventuel eksposition vil der samtidig skulle oprettes omkring 10 kolonner vedrørende den enkelte eksposition. Antallet af kolonner i den endelige tabel indeholdende plastre og ekspositioner vil således potentielt være variabel over tid og i sidste ende afgøres af et multiplum af det maksimale antal ekspositioner registreret for et givent plaster for samtlige testforløb i hele databasen.

2. **Oprettelse af én-til-flere-relation ved hjælp af dummy-rækker.** Denne løsning indebærer, at der oprettes mindst én række i ekspositionstabellen, for hver række der findes i plastertabellen. Således vil der i ekspositionstabellen skulle oprettes over 1,1 millioner rækker, hvoraf over 99 % vil være dummy-rækker, der altså reelt ikke indeholder data. Løsningen vil sikre, at der ikke opstår problemer med `INNER/OUTER JOIN`. Dog vil datamængden potentielt blive for stor og overskuelig for brugerne, og de fleste rækker vil som nævnt reelt være tomme.

3. **Bibeholdelse af én-til-nul-eller-flere-relation.** Endnu en løsning vil være at bibeholde den nuværende én-til-nul-eller-flere-relation. Dette betyder, at brugerne ved samling af plaster- og ekspositionstabeller bliver nødt til at anvende `OUTER JOIN`, hvis de ønsker, at forespørgslen skal omfatte plasterdata (og øvrige af forespørgslen indbefattede tabellers data) i

de tilfælde, hvor der ikke er registreret nogen eksposition for det enkelte plaster.

4. **Oprettelse af én-til-flere-relation mellem testforløb og ekspositioner.** Alternativt kan oprettes en én-til-flere-relation mellem testforløb og ekspositioner, der kan eksistere sideløbende med den nuværende én-til-nul-eller-flere-relation mellem plastre og ekspositioner. Denne løsning vil betyde, at brugere for eksempel kan forespørge alle ekspositioner for et givent testforløb via testforløb-eksposition-relationen. Samtidig vil brugerne kunne finde bestemte ekspositioner for de enkelte plastre ved anvendelse af en `INNER JOIN` på plaster-eksposition-relationen. Denne løsning indebærer altså, at det kan blive nødvendigt for brugere at anvende `OUTER JOIN`, som beskrevet i punkt 3.

Sidste væsentlige — men indlysende — observation var, at brugerne ved udvikling af forespørgsler, der implicerer flere tabeller med mere end én række per testforløb, ufiltreret vil opnå alle kombinationer af disse rækker (for hvert enkelt testforløb) i deres udtræk. Dette skønnes ikke at være det store problem i forbindelse med diagnoser og stillinger, men kan vise sig at være svært at håndtere for brugerne, specielt hvad angår plastre. Der er i gennemsnit 45 rækker i tabellen **Plaster** per række i tabellen **Testforlob** — svarende til, at der for hvert testforløb gennem tiden gennemsnitligt er pålagt 45 plastre.

Et testforløb, hvor der er pålagt 50 plastre, og der er registreret én stilling, to diagnoser og fem egnematerialer, vil således ufiltreret give brugeren 500 rækker i en given forespørgsel, hvilket hurtigt bliver uoverskueligt for det utrænede øje. Det kan diskuteres, hvor godt dette harmonerer med både brugernes behov og deres generelle opfattelse af, at al data i databasen meningsfyldt kan repræsenteres som én stor tabel, hvor et testforløb fylder netop én række.

Problemet med de mange rækker vil være mest udpræget i de tilfælde, hvor brugerne anvender flere af de tabeller, der kan indeholde mange rækker per testforløb og ikke på nogen måde filtrerer. Tabeller som **Plaster**, **Egnematerialer** og **Eksposition** kan indeholde mange rækker per testforløb, mens der sjældent er registreret mere end én stilling og maksimalt to diagnoser per testforløb. Erfaringsmæssigt vil brugerne dog sjældent efterspørge data vedrørende egnematerialer samtidigt med data fra andre tabeller end **Testforlob**. Ligeledes vil brugerne traditionelt efterspørge data, der er meget specifikke med henblik på allergener, stillinger eller diagnoser, hvilket sandsynligvis vil begrænse omfanget af forespørgselsresultaterne. Typiske eksempler på forespørgsler fra brugerne kan være:

- Hvilke stoffer i standardserien har frisører testet i 2009 reageret på?
- Hvilke ekspositioner er gennem de seneste fem år registreret for læger, sygeplejersker og SOSU'er med håndeksem og gummikemikalieallergi?
- Hvilke erhverv reagerer hyppigst positivt på epoxy?

Det anslås derfor baseret på erfaringer, at problemet ikke vil være så omfangsrigt. Dog vil det være hensigtsmæssigt at håndtere i en eventuel implementering.

Opsummerende kan siges, at denne afprøvning har vist, at en ny datamodel under de rette tiltag kan danne basis for, at brugerne selv på meningsfyldt vis udformer korrekte databaseforespørgsler. Afprøvningen har endvidere påpeget en række potentielle problemer, der er blevet diskuteret i ovenstående.

Kapitel 7

Multidimensionel dataanalyse

I dette kapitel beskrives en række centrale elementer inden for multidimensionel dataanalyse. Først beskrives data warehousing — en databasearkitektur til lagring af data, der danner grundlag for dataanalysen. Der beskrives to grundlæggende forskellige tilgange til og modeller for opbygningen af data warehouses.

Siden beskrives OLAP — en teknologi, der understøtter beslutningstagen ved hjælp af multidimensionel dataanalyse og som hører til under den bredere kategori *business intelligence*[1], der blandt andet også omfatter *predictive analysis*, *data mining*, *performance management* og *benchmarking*.

Endelig defineres på baggrund af et view på databasen en kube og ved hjælp af funktionaliteten *pivottabeller* i kontorprogrammet Microsoft Excel udføres en afprøvning, der afdækker hvorvidt multidimensionel dataanalyse eller elementer heraf kan anvendes til at foretage de relevante databaseudtræk.

7.1 Data warehousing

Data warehousing er en teknik til lagring af ofte store mængder data fra potentielt heterogene datakilder på en ensartet struktureret vis. Virksomheder bruger ofte data warehouses til samling og integration af data fra flere forskellige systemer, afdelinger og filialer og danner således grundlag for at skabe en "fælles sandhed" for den givne virksomhed.

I begyndelsen af 1980'erne grundlagdes principperne bag data warehousing, idet

[1] Business intelligence refererer til en række processer, teknikker og applikationer, beslutningstagere i organisationer kan anvende, når de skal træffe strategiske valg.

man forsøgte at tilvejebringe en model for arkitekturer, der tillod en bedre strøm af informationer fra operationelle systemer og databaser til software til understøttelse af beslutningstagen (eng: *decision support software*). Siden har specielt to personer — William Harvey Inmon og Ralph Kimball — ydet deres bidrag til udviklingen af data warehouses. Førnævnte foreslår hver deres grundlæggende forskellige modeller for opbygning og indhold af data warehouses, hvilket belyses nærmere i 7.1.1–7.1.3.

Et data warehouse er af oprindelse beregnet på at understøtte beslutningstagen i virksomheder, hvilket indebærer, at det indeholder både historiske og aktuelle data på både aggregeret og transaktionsniveau. Det skal tillade brugerne at tilgå og forespørge de indeholdte data behændigt og vil således ofte blive brugt i forbindelse med traditionel dataanalyse, hvor for eksempel salgsdata fra forskellige filialer sammenholdes kvantificeret på baggrund af tid, produkt og lokalitet. Samtidig åbner et data warehouse også mulighed for — via sin evne til at gemme enorme mængder tidsstemplede data — at gennemskue ikke-trivielle sammenhænge mellem forskellige data eller at foretage kvalificerede forudsigelser.

7.1.1 Inmons model

William Harvey Inmon fastlagde som den første en komplet model for indhold og struktur af et data warehouse [12]. Inmon anlægger i sin model et oppefra-og-ned-syn på tværs af virksomheders datasystemer, idet han advokerer for, at virksomhederne tilpasser alle deres datasystemer, således at data kan samles i ét data warehouse. Data fra operationelle systemer kopieres således til et "atomisk data warehouse", som man kan bygge forskellige forespørgsler og rapporter ovenpå i forskellige aggregeringsniveauer.

Kongstanken er altså, at alle virksomhedens forskellige forespørgsler og rapporter kan tilpasses den enkelte enheds behov samtidig med, at data på tværs af systemerne er konsistente, idet de stammer fra samme atomiske datakilde. Informationerne i denne datakilde er grupperet emneorienteret (for eksempel efter forretningsområder). Da den atomiske datakilde er tæt på de operationelle (database)systemer, er data warehouses efter Inmons model som mange øvrige databaser normaliseret i henhold til tredje normalform.

7.1.2 Kimballs model

Ralph Kimball har foreslået en model for data warehouses, der adskiller sig meget fra den traditionelle relationelle databasemodel [16], idet modellen ofte refereres til som dimensionel datamodellering. Kimballs model anvender en nedefra-og-op-tilgang til virksomheders data, og der opereres i modellen med flere sameksisterende databaser (data marts), der svarer til forskellige forretningsprocesser, for eksempel salg, lager, personale eller markedsføring.

Datatabeller kan enten være faktatabeller (eng: *fact tables*) eller dimensionsta-

beller (eng: *dimension tables*), hvor førstnævnte indeholder numerisk målbare værdier, og sidstnævnte indeholder de attributter, der knytter sig til faktatabellernes værdier — i stærkt denormaliseret form. Faktatabellerne består typisk af få kolonner, men indeholder mange rækker, mens dimensionstabellerne omvendt består af mange kolonner og få rækker.

I Kimballs model kopieres data fra operationelle databasesystemer via et "opsætningsområde" til data marts, der svarer til de tidligere nævnte forretningsprocesser. Kimball definerer fire trin i opbygningen af et data mart:

1. Vælg forretningsproces.
2. Deklarér detaljeringsgrad.
3. Vælg dimensioner.
4. Identificér fakta.

Når samtlige forretningsprocesser er kortlagt og tilsvarende data marts er bygget, kan disse samles til et data warehouse via en *data bus*, hvilket fordrer stor konformitet dimensionerne imellem, som er essentiel for succesfuld integration af Kimballs model. Forestiller man sig således, at der for en given virksomheds data warehouse findes en dimension *Produkt*, vil et produkt i hver instans af denne dimension have samme primærnøgle på tværs af samtlige data marts. Dette sikrer, at der ikke opstår konflikter mellem resultater, når man eksekverer forespørgsler på tværs af flere data marts.

7.1.3 Sammenligning af modeller

Som det fremgår af 7.1.1 og 7.1.2 er der flere grundlæggende forskelle på de to tilgange til opbygning af et data warehouse. I det efterfølgende sammenlignes de to modeller. For en mere grundig gennemgang af modellerne henvises til [3].

Inmon anlægger et oppefra-og-ned-syn på data warehouses, hvor virksomheden betragtes som et hele, og operationelle systemer og data warehouse må tilpasses hinanden. Arkitekturen baserer sig på relationelle databasemodeller, anvender normalisering og atomitet af data og er datastyret. En succesfuld implementering af et data warehouse i henhold til denne arkitektur er ressourcetung og ikke-triviel, og den henvender sig derfor primært til IT-professionelle som del af større virksomheder.

Kimball derimod ser virksomheden nedefra-og-op. Arkitekturen er processtyret, og et data warehouse er inddelt i flere data marts, der hver repræsenterer en forretningsproces. Data kopieres fra eksisterende databaser og inddeles i stærkt denormaliseret form i fakta og dimensioner. Dimensionskonformitet sikrer integritet mellem de forskellige data marts. Et data warehouse kan implementeres med relativt få ressourcer, idet en virksomhed kan fokusere på enkelte forretningsprocesser én ad gangen.

Fælles træk for Inmons og Kimballs modeller findes i en proces, der skal udføres hver gang et data warehouse skal opdateres, og som er kendt som ETL — *extract*, *transform*, *load*.

Extract refererer til den fase, hvor data hentes fra flere forskellige kilder — for eksempel relationelle databaser, flade filer og web services. Transform henviser til udvælgelse, ensretning og sikring af integriteten af de data, der skal indeholdes i det pågældende data warehouse. Eksempler på transform-operationer er:

- Valg af bestemte kolonner/rækker fra kildetabellerne.
- Filtrering af data.
- Sortering af data.
- Aggregering — for eksempel summering eller optælling.
- Beregning af nye værdier på baggrund af eksisterende data.
- Opsplitning af kolonner — en datokolonne kan splittes op i flere kolonner: for eksempel dato, år, kvartal, måned, uge, dag og tid.
- Ensretning af værdier mellem systemer — for eksempel kan et system anvende 0/1 til kodning af en kønsvariabel, mens et andet anvender M/K.
- Omkodning af værdier gemt i fritekst til heltallige værdier.
- Transponering af tabeller.
- Samling af tabeller fra forskellige kilder.

Load refererer til en fase, hvor de transformerede data kopieres til et data warehouse, hvilket ikke er trivielt, idet der ofte skal defineres regelsæt i forbindelse med opdateringer. Kompleksiteten af opdateringer afhænger endvidere af, hvorvidt historiske data er gemt i de operationelle databaser eller kun gemmes i virksomhedens data warehouse. Ved anvendelse af Kimball-modellen er det i load-fasen[2], at fakta- og dimensionstabeller oprettes [15].

[2] I [15] kaldes fasen for *deliver*.

7.2 OLAP

Online analytical processing (OLAP) er en betegnelse både for en arkitektur og teknologi, der via multidimensionel modellering er beregnet på at understøtte virksomheders evne til at træffe beslutninger på baggrund af historiske såvel som aktuelle data.

7.2.1 Baggrund for OLAP

OLAP-termen hører til under den bredere business intelligence-kategori og blev første gang brugt af E. F. Codd og andre [9], der definerede 12 kriterier til evaluering af programmel i forhold til OLAP-kompatibilitet.

Der er siden definitionen af OLAP rejst tvivl om både navngivningen af OLAP-konceptet og såvel tilstrækkeligheden som rimeligheden af de 12 tilkyttede kriterier. Artiklen af Codd og andre blev originalt publiceret i ComputerWorld, men blev siden trukket tilbage, da det viste sig, at artiklen var bestillingsarbejde for virksomheden Arbor Software[3], der året før var gået på markedet med deres multidimensionelle DBMS *EssBase*. Codd og andre evaluerer i artiklen netop EssBase i forhold til de 12 kriterier, hvoraf flere hævdes ikke at være matematisk funderet men at være som skræddersyet i forhold til EssBases funktionalitet.

Der er siden fra forskellige sider rejst tvivl om, hvorvidt det er Codd selv, hans hustru eller en forskningsassistent, der har stået for artiklens indhold. Der har formodentlig fra Arbors side været tale om et meget effektivt pr-trick, der har har trukket på Codds store troværdighed siden hans banebrydende arbejde med den relationelle databasemodel [8], der blandt andet var årsag til, at han i 1981 modtog ACMs *Turing Award*.

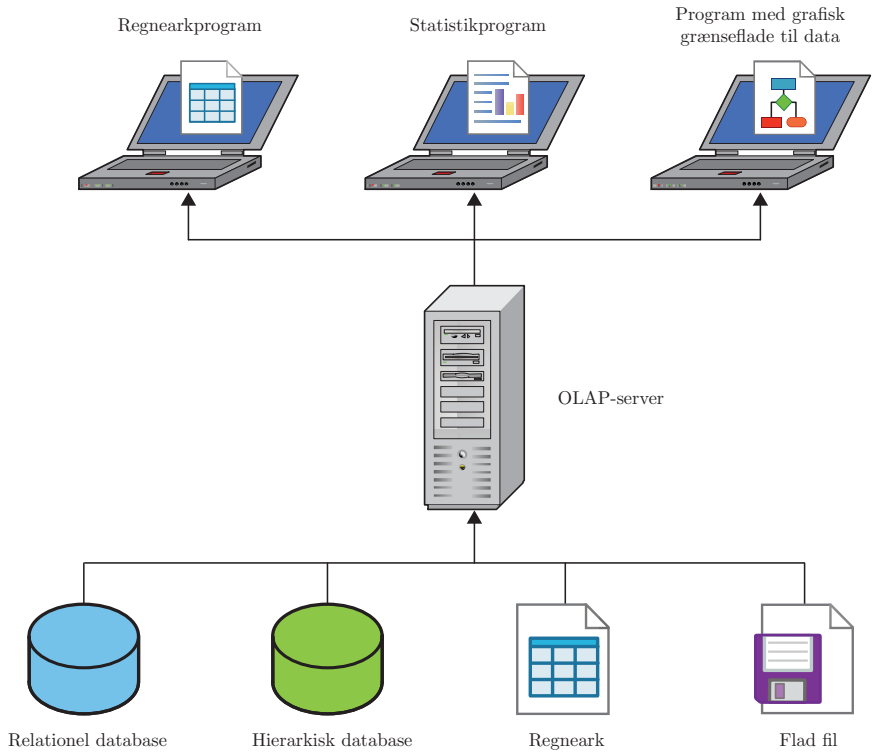
Der er blandt udviklere og anvendere af systemer til understøttelse af virksomheders beslutningstagen dog næppe nogen tvivl om, at Codd og andres arbejde har været med til at sætte gang i en rivende udvikling inden for det felt, der i dag samlet kaldes business intelligence.

Den uafhængige britiske analytiker Nigel Pendse, der er ophavsmand til *The OLAP Report*[4] har i stedet for Codd og andres 12 kriterier foreslået 18 kriterier til bestemmelse af OLAP-kompatibilitet og har endvidere foreslået, at man anvender navnet FASMI — *Fast Analysis of Shared Multidimensional Information*, hvilket ganske rigtigt siger mere om emnet.

Kort fortalt går OLAP eller FASMI — som øvrige business intelligence-teknikker — ud på at samle data fra potentielt flere forskellige heterogene datakilder i én

[3] Arbor Software fusionerede i 1997 med Hyperion, der i 2007 blev opkøbt af Oracle.

[4] Siden 2009 kendt som *The BI Verdict*. Den er nu en del af de services, der udbydes af en uafhængig analysevirksomhed inden for softwareindustrien, BARC — *Business Application Research Center*. Kilde: `http://www.bi-verdict.com/`.



Figur 7.1: Oversigt over OLAP-serverens rolle i henhold til [9].

struktur, hvor data er aggregeret ensartet med henblik på eksekvering af hurtige forespørgsler til flere samtidige brugere via forskelligt programmel, der via standardiseret kaldgrænseflade tilgår strukturen, for på den måde at understøtte beslutningstagning. Dette er afbildet i simpel form i figur 7.1 i henhold til [9].

7.2.2 Essensen af OLAP

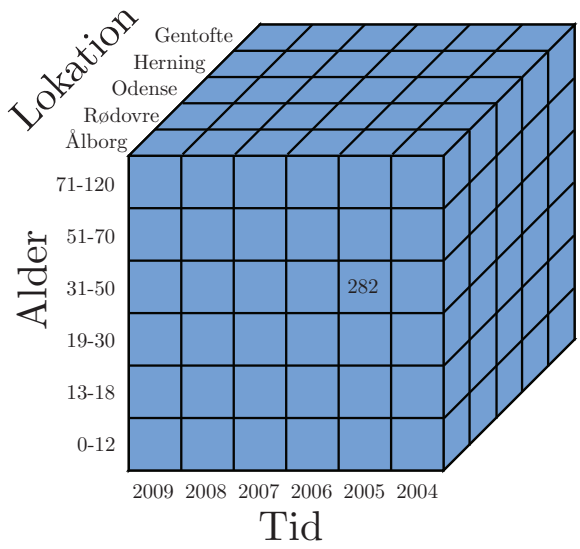
Centralt for OLAP eller FASMI er opfattelsen af en flerdimensionel datastruktur, der hyppigt refereres til som en *cube* eller *hypercube*, idet strukturen teoretisk kan bestå af et arbitrært antal dimensioner.

Kuben er dannet på baggrund af data fra én eller flere potentielt forskelligartede datakilder, og disse data er aggregerede af hensyn til effektiviteten af de forespørgsler, der skal anvende kuben. Modsat en OLTP[5]-database — for eksempel

[5] **O**n-**L**ine **T**ransactional **P**rocessing refererer til en klasse systemer, der tillader og styrer transaktionsorienterede applikationer, typisk transaktioner, der involverer skrivning eller læsning af data. *Oversat fra engelsk.* Kilde:

en produktionsdatabase — er OLAP altså beregnet på effektive forespørgsler [6].

Kuben er inddelt i dimensioner og numeriske mål (eng: *measures*). Dimensionerne kan for eksempel være tid, alder og lokation og inddeles hyppigt i hierarkier af niveauer af forskellig granularitet; for eksempel kan tid inddeles i alt fra år til sekunder, alder kan angives absolut eller som del af et eller flere intervaller, ligesom lokation kan inddeles i alt fra verdensdel til lokalenummer.



Figur 7.2: Eksempel på tredimensionel kube.

I figur 7.2 ses et eksempel på en tredimensionel kube, med dimensionerne tid, lokation og alder. Denne kube indeholder patientdata fra fem klinikker over en årrække på seks år, stratificeret på alder. Af kuben fremgår det, at der i 2005 i klinikken i Ålborg sammenlagt blev testet 282 patienter i alderen 31–50.

7.2.3 Tilblivelsen af en kube

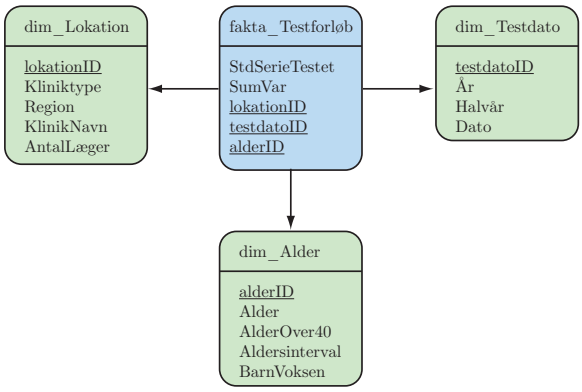
Datagrundlaget for en kube vil oftest forefindes i et data warehouse, som beskrevet i afsnit 7.1, men kan principielt også udgøres af en simpel tabel, et view eller et regneark. Ved implementering af en kube kan overordnet vælges mellem to forskellige tilgange — MOLAP (multidimensionel) eller ROLAP (relationel) — svarende til de i afsnit 7.1 beskrevne data warehouse-modeller.

Til trods for navnene understøtter både MOLAP og ROLAP multidimensionel

`http://en.wikipedia.org/wiki/Online_transaction_processing`

dataanalyse. Den store forskel består i de underliggende datastrukturer. MOLAP lagrer data denormaliseret, mens ROLAP lagrer i en normaliseret (tredje normalform) relationel database og først introducerer multidimensionaliteten på det tidspunkt, hvor en slutbruger eksekverer en forespørgsel. En nærmere sammenligning af de to modeller findes i [26] samt [6].

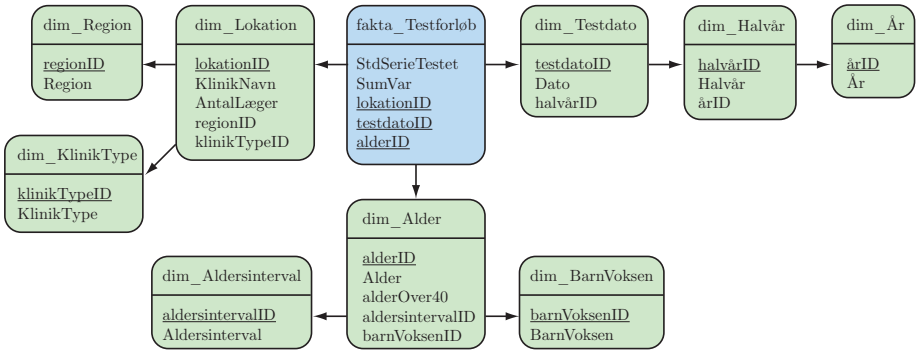
Ved oprettelse af en kube skal fastlægges, hvilke tabeller fra det underliggende data warehouse, der skal anvendes. En faktatabel indeholdende de numeriske mål og de til faktatabellen hørende dimensionstabeller skal vælges. En n-dimensionel kube vil således inkludere data fra n dimensionstabeller i det aktuelle data warehouse. Valget af data warehouse-arkitektur er afgørende for, hvilke tabeller der skal inkluderes. Grundlæggende findes to modeller for, hvordan faktatabellen og dimensionstabellerne er relateret; disse benævnes henholdsvis stjerneskema (eng: *star schema*) og snefnugskema (eng: *snowflake schema*), hvor førstnævnte er denormaliseret og sidstnævnte normaliseret i henhold til tredje normalform. Et stjerneskema kan dermed betragtes som et specialtilfælde af et snefnugskema.



Figur 7.3: Eksempel på stjerneskema.

En faktatabel indeholder således ud over de numeriske mål en række referencer (fremmednøgler) til hver af dimensionstabellerne, og faktatabellens primærnøgle er altså sammensat af dimensionstabellernes primærnøgler.

I praksis vil et komplet data warehouse indeholde flere kuber/faktatabeller — repræsenteret ved relationelle databaser, der enten tager form som snefnug- eller stjerneskema — der er relateret til hver deres unikke dimensionstabeller, men mange af disse dimensionstabeller vil også være delt mellem kuberne. På baggrund af disse fakta- og dimensionstabeller samt de indbefattede numeriske mål kan en kube defineres og fremdeles anvendes til datanaalyse.



Figur 7.4: Eksempel på snefnugskema.

7.2.4 OLAP-software

Der findes en række kommercielle serverbaserede systemer, der tilbyder komplette data warehouse- og OLAP-løsninger, hvoraf de markedsdominerende og mest anerkendte er:

- Microsoft SQL Server Enterprise Edition (SSMS[6], SSIS[7], SSAS[8], SSRS[9]).
- Oracle Olap som del af Oracle Database 11g.
- SAP BusinessObjects.
- IBM Cognos.
- MicroStrategy OLAP Services.

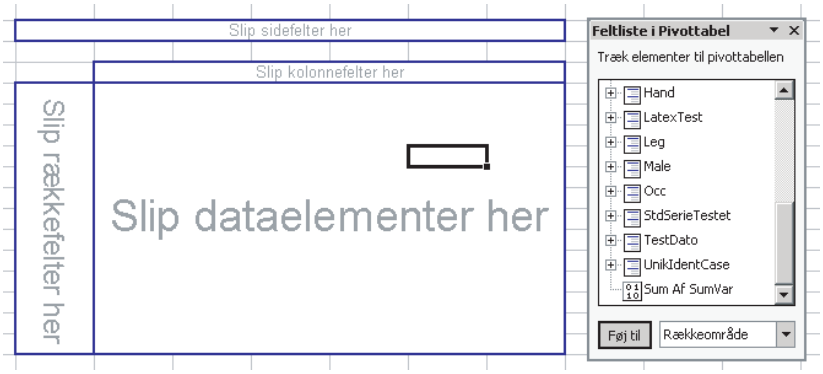
Ovennævnte systemer indeholder værktøjer til blandt andet oprettelse og vedligeholdelse af data warehouses og OLAP-systemer, der primært er rettet mod IT-professionelle. De indeholder også applikationer til multidimensionel dataanalyse, som er møntet på brugere, der ikke er IT-professionelle. Applikationerne giver brugerne mulighed for at gennemse data indeholdt i kuber, definere og eksekvere rapporter og udarbejde forskellige statistikker og indikatorer.

Der findes endvidere en række mindre database-, kontor-, og statistikapplikationer, der muliggør oprettelse af kuber og giver ikke IT-professionelle brugere mulighed for at gennemse kuberne og udarbejde statistikker på baggrund af dem. Eksempler på disse er SPSS og Microsoft Excel, som begge tillader gennemsyn af eksisterende kuber eller oprettelse af nye på baggrund af for eksempel

[6]SQL Server Management Studio (DBMS)
[7]SQL Server Integration Services (ETL)
[8]SQL Server Analysis Services (OLAP)
[9]SQL Server Reporting Services (Rapporter)

databasetabeller eller -views. Datagrundlaget behøver således ikke at være multidimensionelt i sin struktur.

Fælles for både de større professionelle systemer og de mindre applikationer er, at de stiller et interaktivt værktøj tilgennemsyn af data til rådighed for brugerne. Værktøjet vil ofte tage form som en tabel, hvor brugeren ved hjælp af træk og slip (eng: *drag and drop*) kan definere, hvordan tabellen inddeles horisontalt og vertikalt baseret på dimensioner i kuben. I midten af tabellen findes de numeriske mål og summeringsvariable, der aggregeres på baggrund af. Disse aggregeringer opdateres automatisk i takt med, at brugeren ændrer på opsætningen af dimensionerne.



Figur 7.5: Microsoft Excel tillader gennemsyn af kuber eller views via det interaktive værktøj pivottabeller.

Hvis man alene ser på brugerens grænseflade til interaktivt gennemsyn af en kube eller et view via ovennævnte værktøj, minder de store professionelle systemer og de mindre applikationer om hinanden. En ikke IT-professionel slutbruger vil ved hensigtsmæssigt design næppe mærke forskel på at gennemse en kube, der er defineret som del af en større datawarehouse- og OLAP-løsning, og en mindre kube defineret i kontor- eller statistikprogrammel.

7.3 Diskussion

På trods af den store udbredelse, data warehouse- og OLAP-programmel har fundet, er det ikke givet, at det fyldestgørende vil kunne bruges til at foretage de relevante udtræk af databasen. Multidimensionel datananalyse er skabt med henblik på at samle heterogene data fra flere kilder i én central datastruktur og tillade effektive forespørgsler i denne. I sin natur er teknologien beregnet på at samle store mængder data, foretage aggregeringer og levere hurtige svar på brugeres forespørgsler. Disse forespørgsler vil ofte danne grundlag for forskellige

indikatorer (for eksempel KPI[10]) eller faste rapporter bestående af summationer, der tillader sammenligning af eksempelvis salgstal for enheder på tværs af tid, produktkategori og fysisk lokation. Ligeledes er anvendelsesmulighederne mange inden for den finansielle sektor. I denne sammenhæng er multidimensionel dataanalyse et utroligt virkningsfuldt værktøj.

Multidimensionel dataanalyse er dog ikke særegent effektiv til at levere data på caseniveau, da en forespørgsel i mange tilfælde blot vil svare til at foretage udtræk af en del af kolonnerne i en stærkt denormaliseret databasetabel eller -view. I forhold til nærværende specialeprojekt tænkes det således, at multidimensionel dataanalyse vil være meget anvendelig i forbindelse med mindre forespørgsler, der ikke fordrer, at data leveres på caseniveau, men snarere stiller krav om interaktiv og dynamisk leverance af intuitive statistikker, der kan give overblik og ligge til grund for ansøgninger i forbindelse med mulige projekter. For nuværende drages der dog tvivl om gevinsten ved at anvende multidimensionel datanaalyse frem for eksempelvis veldefinerede, højt denormaliserede views/tabeller, der indeholder prækalkuleringer og aggregeringer.

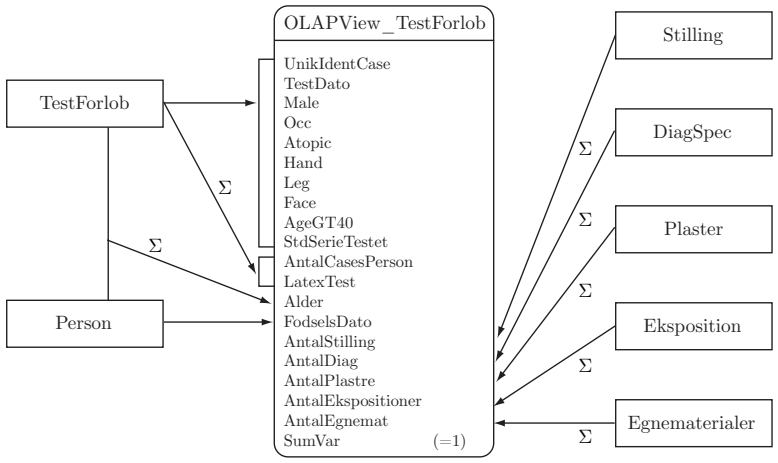
Da en komplet implementering af en data warehouse- og OLAP-løsning, der potentielt vil understøtte samtlige forespørgsler svarende til alle testscenarier, er meget ressourcekrævende, og gevinsten af en sådan implementering i forhold til views/tabeller kan drages i tvivl, udføres en simpel afprøvning, hvor en mindre kube udvikles på baggrund af et view på databasen. Denne afprøvning skal afdække, hvorvidt det vil være plausibelt at foretage en komplet implementering.

7.4 Afprøvning

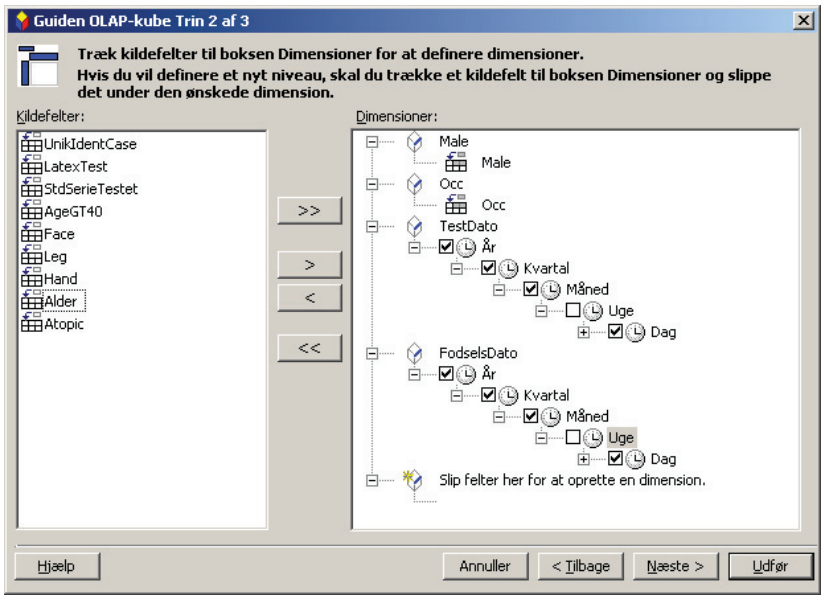
Det søges i denne afprøvning belyst, hvorvidt multidimensionel dataanalyse eller elementer heraf kan bruges til at udvikle forespørgsler svarende til de behov, brugerne af databasen har. Til dette formål formuleres et view på databasen, som danner grundlag for en kube, der oprettes via *Guiden OLAP-kube*, der er tilgængelig i applikationen Microsoft Query. Denne kube afprøves siden ved hjælp af Microsoft Excels funktionalitet pivottabeller.

Til afprøvningen formuleres viewet OLAPView_Testforlob (figur 7.6), der er en videreudvikling af tabellen **_DB_testforlob** kendt fra kapitel 6. Viewet indeholder ud over de tidligere beskrevne kolonner, en række aggregerede kolonner; herunder optællinger af antal stillinger, diagnoser og plastre, der er registreret for hvert enkelt testforløb. Endvidere indeholder viewet også kolonnen SumVar, der er en konstant med værdien 1. Sidstnævnte kolonne skal sammen med de andre optællinger anvendes som summeringsvariabel i kuben, mens de øvrige kolonner danner grundlag for dimensioner. Kildekoden til dette view findes i bilag D.

[10] **K**ey **P**erformance **I**ndicator. Et målepunkt, som for eksempel virksomheder kan definere og bruge til at overvåge, hvor succesfuld den er udi at opfylde forskellige processer eller diverse kvantificérbare mål.



Figur 7.6: OLAPView_Testforlob som anvendes til dataanalyse i Excel.



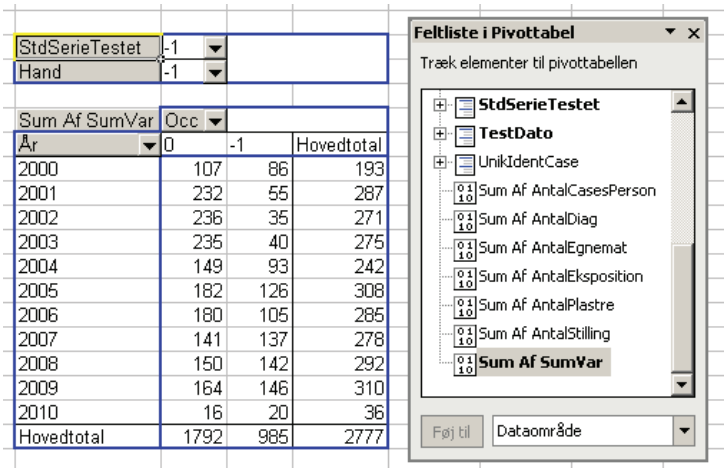
Figur 7.7: Oprettelse af kube ved hjælp af Guiden OLAP-kube.

Via Guiden OLAP-kube defineres en kube med viewet OLAPView_Testforlob som datakilde. Dimensioner for kuben defineres som afbildet i figur 7.7. Kolonnerne TestDato og FodselsDato er begge af datatypen `DATETIME`, og guiden opretter for disse to automatisk et dimensionshierarki, hvor man kan vælge gran-

ularitet — det vil sige hvilke niveauer af hierarkiet, der skal indgå i den endelige dimension. For de øvrige kolonner, der alle er af andre datatyper, oprettes dimensioner i ét niveau.

Efter fastlæggelse af summeringsvariable og dimensioner gemmes kuben og er klar til anvendelse. I figur 7.5 ses en tom pivottabel, der anvender den definerede kube som datakilde.

Brugeren trækker og slipper dimensionerne ind i pivottabellen, der dermed oprettes som enten kolonne-, række- eller sidefelter. De to førstnævnte bestemmer den horisontale og vertikale inddeling af tabellen. Denne inddeling kan være i flere niveauer for den enkelte dimension, lige som der samtidigt kan anvendes flere dimensioner både horisontalt og vertikalt. Endvidere kan brugeren vælge at udelade visning af dele af dimensionen, lige som man kan vælge, at dele af dimensionen inddeles i ét niveau, mens resten af dimensionen inddeles i et andet niveau. Dimensioner, der oprettes som sidefelter, kan bruges til at filtrere data. Summeringsvariable trækkes ind i midten af pivottabellen og danner dermed grundlag for optællinger og andre aggregeringer



Figur 7.8: Eksempel på anvendelse af Excels Pivot-tabeller til statistik på baggrund af kube baseret på viewet OLAPView_Testforlob (figur 7.6).

Samlet set er pivottabellen et meget brugervenligt og intuitivt værktøj, der giver brugeren mulighed for at gennemse data interaktivt. De summerede data i tabellen opdateres automatisk, så snart brugeren ændrer den måde, tabellen er inddelt i dimensioner på. Hvis brugeren ændrer kriterierne for filtrering, vil det også straks afspejles i data. I figur 7.8 ses et eksempel, hvor dimensionen testdato er valgt som rækkeelement og Occ (arbejdsbetinget lidelse) er valgt som kolonneelement. For testdato er valgt år som inddelingsenhed og alle testforløb før år 2000 er udeladt. Endvidere filtreres på baggrund af dimensionerne StdSerieTestet og Hand, der begge skal være opfyldt. Af figuren kan således aflæses,

at der i 2008 i alt var 292 testforløb, hvor patienten havde håndeksem og blev testet med standardserien. Af disse blev 142 bedømt som arbejdsbetingede.

7.4.1 Resultater

I nærværende afprøvning har det på baggrund af det udviklede view og den udviklede kube ikke været muligt at realisere et eneste af scenarierne. Pivottabellen har vist sig at være et effektfuldt værktøj til at skabe statistiske overblik over data indeholdt i kuben, mens værktøjet som berørt i afsnit 7.3 ikke egner sig til at levere data på caseniveau. Da pivottabellen minder meget om de værktøjer, som øvrige leverandører af OLAP-systemer stiller brugeren til rådighed til gennemsyn af kuben, anses det for usandsynliggjort, at multidimensionel datanalyse vil kunne leve op til de krav, der konkret findes i forbindelse med udvikling af forespørgsler i databasen, hvor data ønskes på caseniveau.

Generelt må siges, at multidimensionel dataanlyse er utroligt effektiv til at udarbejde og se indikatorer og andre faste rapporter baseret på store mængder let aggregérbare data som for eksempel finansielle data eller salgsdata. Data på caseniveau, der eventuelt implicerer kolonner fra flere relaterede tabeller samtidigt, og hvor man ønsker, at mange kolonner skal indgå i de endelige forespørgsler, egner multidimensionel dataanalyse sig dog dårligt til. Hvis værktøjerne til gennemsyn af kuben tillader det, vil kuben i bedste fald være at betragte som et stort denormaliseret view på databasen.

Der hersker dog ingen tvivl om, at kuber kan bruges i forbindelse med opstart af nye projekter, hvor der ønskes statistiske udtræk, som kan give hurtige overblik over data. Pivottabeller er endvidere intuitive i brug og kan således baseret på veldefinerede views på databasen danne grundlag for, at brugerne selv udvikler og eksekverer indikatorer og andre faste rapporter til at overvåge databasens indhold. Samtidig kan pivottabeller bidrage til at gennemskue sammenhænge mellem forskellige data, der ikke tidligere var kendt, og dermed danne grundlag for formuleringen af nye projekter.

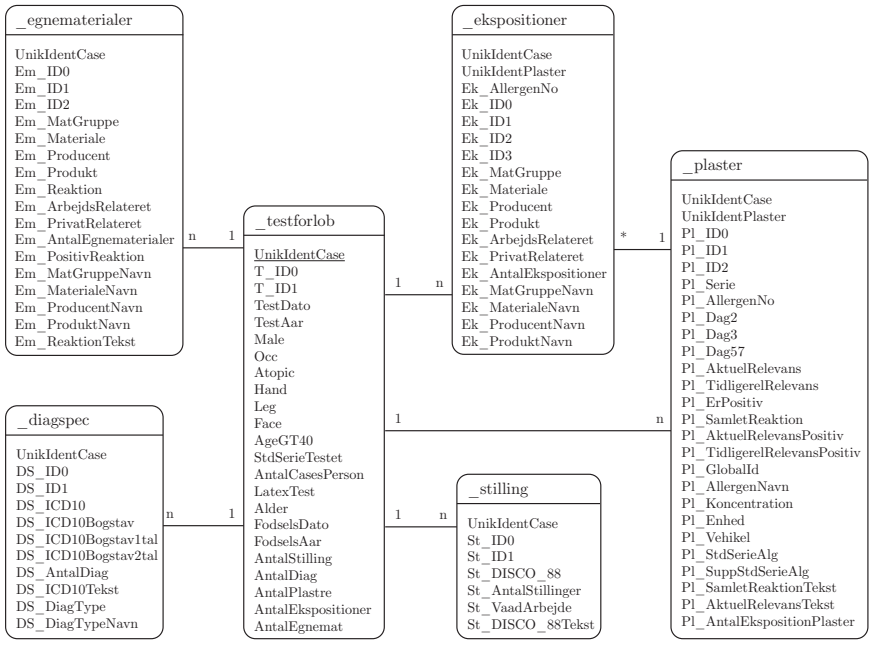
Et konkret bidrag, multidimensionel dataanalyse kan gøre i forhold til den endelige løsning, er en indførelse af den hierarkiske inddeling af dimensioner. Denne inddeling skønnes uanset løsningens karakter at kunne anvendes til effektiv filtrering af data under samtidig hensyntagen til, at antallet af fejl, der skyldes brugernes interaktion med systemet, minimeres. Ved dimensioner/kolonner, der indeholder datoer eller tekststrenge, hvis data i forvejen er af en hierarkisk karakter — for eksempel diagnose- og stillingskoder — vil det være lige for at indføre hierarkisk dimensionsinddeling, mens det i andre tilfælde ikke vil give mening. Endvidere vil dimensionens granularitet afhænge af den enkelte dimension og brugernes behov.

Kapitel 8

Tredje test

8.1 Den nye datamodel

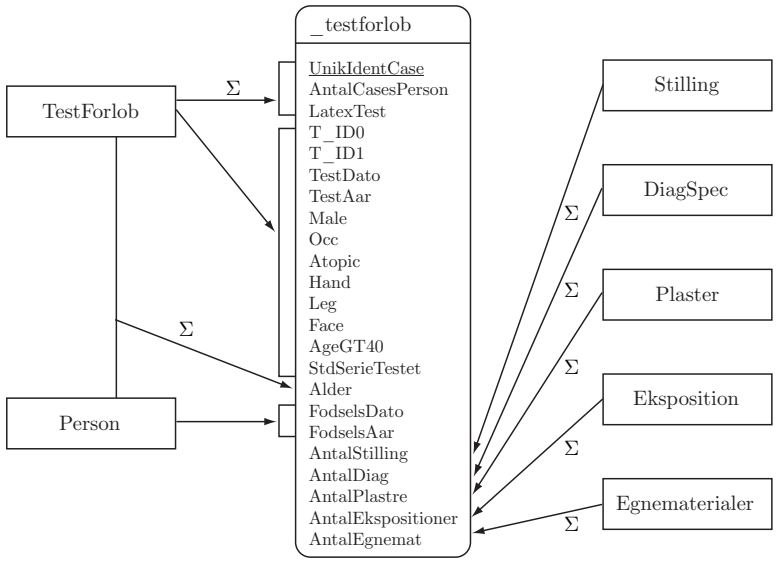
Den nye datamodel (figur 8.1) består af kun seks tabeller og implicerer sammenlagt 21 tabeller fra den originale datamodel; heraf syv personbundne tabeller og 14 ikke-personbundne tabeller.



Figur 8.1: Oversigt over den nye datamodel.

Modellen er centreret omkring tabellen **_testforlob** (figur 8.2), der er en kombination af de originale **Person**- og **TestForlob**-tabeller samt en række aggregater af øvrige tabeller, som dels skal bruges til optælling ved simple forespørgsler og dels skal give brugeren en mulighed for at validere resultaterne af de udviklede forespørgsler. Brugerne skal altså kunne anvende disse aggregater og summationer som kontrolinstans, hvilket forventes at minimere fejl i de udviklede forespørgsler.

Det er værd at bemærke, at tabellen indeholder kolonnen TestAar, der indeholder årstallet for, hvornår testforløbet er udført. Dette årstal er reelt allerede indeholdt i kolonnen TestDato, men er dels inkluderet for at imødekomme brugernes problemer med de indbyggede datoekstraktionsfunktioner og dels for at efterligne den hierarkiske dimensionsinddeling, der er kendt fra den multidimensionelle dataanalyse (kapitel 7). Det samme gør sig gældende for kolonnerne FodselsDato og FodselsAar.

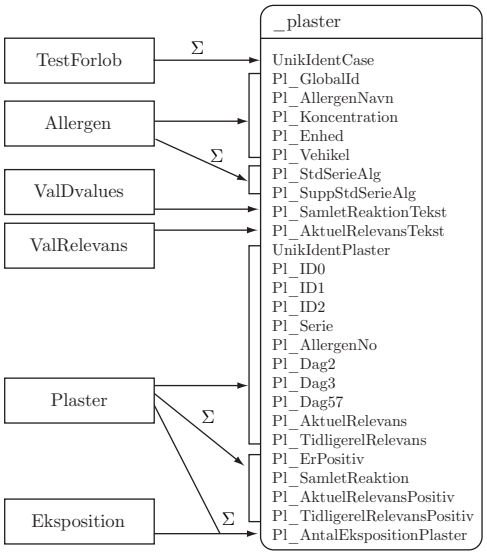


Figur 8.2: Oversigt over tabellen **_testforlob**.

Tabellen **_testforlob** har en primærnøgle, UnikIdentCase, der er en konkatenering af de to kolonner ID0 og ID1, der konstituerer den sammensatte primærnøgle for tabellen **TestForlob** i den originale datamodel. Dermed anvendes i nærværende datamodel den samme metode til fjernelse af den sammensatte primærnøgle, som den der blev anvendt i første afprøvning (afsnit 6.4) og som bekendt effektivt eliminerede en del af de problemer, testpersonerne havde med at udføre korrekte `JOIN`-operationer på sammensatte nøgler i de to første tests. Kolonner, der i den originale **TestForlob** var af typen `BIT` er i nærværende model konverteret til `INTEGER`.

Inklusionen af data fra tabellen **Person** i **_testforlob** er reelt en denormalisering, der overflødiggør førstnævnte tabel. Eneste omkostning i forbindelse med forespørgsler er, at der opstår nogle dubletter. Man kan altså ved hjælp af UnikIdentCase, der indgår i alle øvrige tabeller i modellen, for enhver række i enhver tabel unikt identificere det pågældende testforløb.

Modellen er således baseret på, at der for hvert testforløb findes mindst én række i hver af de øvrige fem tabeller, hvorved den minimale datamængde for et givent testforløb består af mindst én række i samtlige af de seks tabeller. I de tilfælde, hvor der i et testforløb for eksempel ikke er pålagt nogle plastre, vil der således alligevel findes netop én række i tabellen **_plaster** (figur 8.3) indeholdende **_testforlob**s primærnøgle og værdien `NULL` i samtlige øvrige kolonner.



Figur 8.3: Oversigt over **_plaster**.

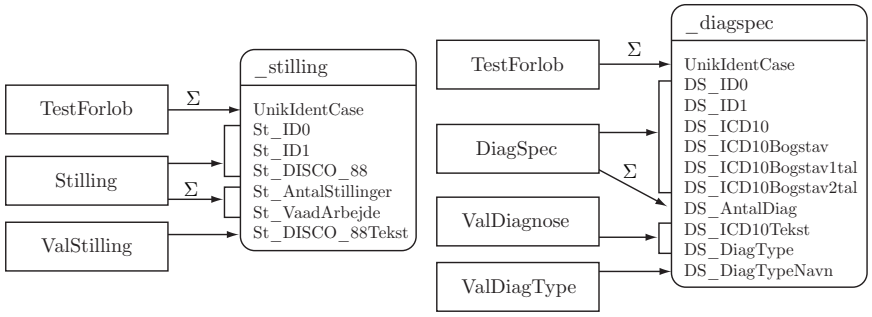
Dette er et bevidst valg i design af datamodellen for at imødekomme, at de hidtil anvendte applikationer ved tilføjelse af relationer mellem tabeller automatisk anvender `INNER JOIN`-typer, hvilket gav testpersonerne en del problemer i de tilfælde, hvor der ikke fandtes korresponderende rækker i de øvrige tabeller, da de ofte ikke opdagede problemet og derfor ikke ændrede til en korrekt `OUTER JOIN`. Med det nye design undgås denne situation. Omvendt kan der være situationer, hvor brugeren reelt ønskede at foretage en `INNER JOIN`, i hvilke tilfælde det antages, at brugeren vil opdage de rækker, for hvilke nogle kolonner er tomme. Samtidig vil en sådan situation kunne imødekommes, hvis brugeren filtrerer på baggrund af de kolonner i **_testforlob**, der optæller rækker i de fem øvrige tabeller. Brugeren kan for eksempel stille som krav for en filtrering, at kolonnen AntalPlastre ≥ 1, og vil dermed opnå samme resultat som en `INNER`

`JOIN` anvendt på **TestForlob** og **Plaster** ville afføde i den originale datamodel. I den forbindelse skal det selvfølgelig nævnes, at AntalPlastre indeholder en optælling af antallet af rækker i tabellen **Plaster** og ikke antallet af rækker i **_plaster**, da der i sidstnævnte findes dummy-rækker.

Tabellen **_plaster**, der primært er en kombination af tabellerne **Plaster** og **Allergen** fra den originale datamodel samt nogle aggregeringer heraf, indeholder også nogle tekstfelter fra ikke-personbundne tabeller. Desuden er som nævnt inkluderet nøglen UnikIdentCase fra tabellen **_testforlob** og kolonnen UnikIdentPlaster, der, som navnet antyder, unikt identificerer det enkelte plaster — eller række i tabellen — og dermed er kandidatnøgle. I de tilfælde, hvor der for et givent testforløb ikke er pålagt plastre, vil der alligevel findes netop én række (en dummy-række) i nærværende tabel.

Kolonnen UnikIdentPlaster er ikke primærnøgle for tabellen, hvilket berøres nærmere ved gennemgangen af tabellen **_ekspositioner**. **_plaster** indeholder endvidere kolonnen Pl_AntalEkspositionPlaster, der er en optælling af antal registrerede ekspositioner, der hidrører fra netop det plaster, der er indeholdt i den enkelte række (i tabellen **_plaster**).

Tabellen **_stilling** (figur 8.4) er primært dannet ud fra den originale tabel **Stilling** og indeholder ud over UnikIdentCase en stillingskode og den dertil hørende tekstlige stillingsbeskrivelse. Desuden findes en optælling af antal stillinger registreret for det pågældende testforløb samt en kolonne, der indikerer, hvorvidt det enkelte erhverv er at kategorisere som våderhverv.

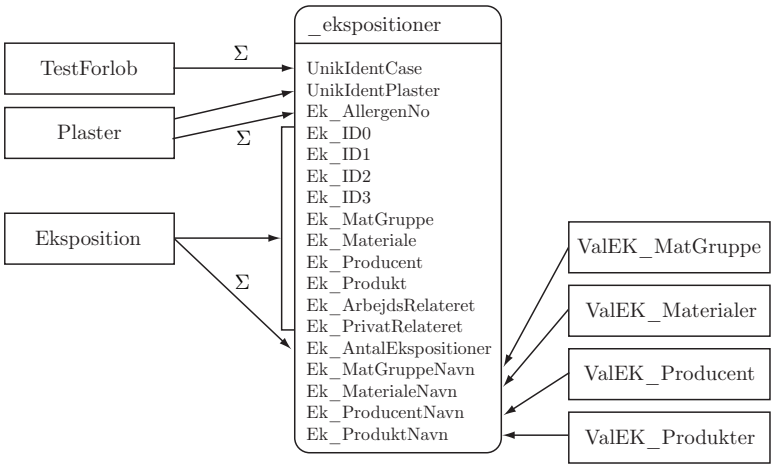


Figur 8.4: Oversigt over tabellerne **_stilling** og **_diagspec**.

_diagspec (figur 8.4) er dannet ud fra den originale **DiagSpec**-tabel og indeholder derfor hovedsagligt data fra denne. Samtidig indeholder den ud over UnikIdentCase nogle forklarende kolonner fra ikke-personbundne tabeller. I forbindelse med tabellen er det værd at bemærke kolonnerne DS_ICD10, DS_ICD10Bogstav, DS_ICD10Bogstav1tal og DS_ICD10Bogstav2tal. Reelt er de tre sidstnævnte indeholdt i førstnævnte, idet DS_ICD10 indeholder hele diagnosekoden (som firecifret tekststreng), mens de andre indeholder henholdsvis en, to og tre karakterer af tekststrengen. Denne indretning af tabellen er indført

for at efterligne den hierarkiske dimensionsinddeling fra den multidimensionelle dataanalyse.

Brugerne af databasen kender til opbygningen af ICD10-diagnoser[1], men kan ikke forventes at have kendskab til SQL-funktionen `SUBSTRING`, der ekstraherer en del af en tekststreng. Det kan være nyttigt for brugerne at foretage udtræk, hvor der for eksempel filtreres på, hvorvidt der i det enkelte testforløb er givet en dermatologisk relevant diagnose (DS_ICD10Bostav = 'L'), eller hvor der ønskes fundet samtlige toksiske kontakteksem-diagnoser (DS_ICD10Bostav2tal = 'L24'), men hvor man ikke ønsker at filtrere på det specifikke stof, der har forårsaget reaktionen (L240–L249).

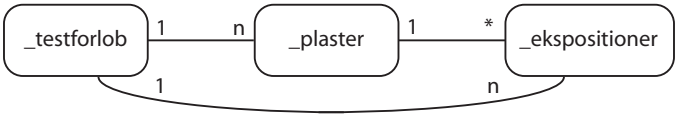


Figur 8.5: Oversigt over **_ekspositioner**.

Tabellen **_ekspositioner** (afbildet i figur 8.5) indeholder primært data fra tabellerne **Plaster** og **Eksposition** fra den originale datamodel samt nogle kolonner, der stammer fra ikke-personbundne tabeller. Disse kolonner indeholder forklarende tekstlige beskrivelser af koder for eksempelvis materialer og producenter for de registrerede ekspositioner. **_ekspositioner** indeholder som alle øvrige tabeller i modellen kolonnen UnikIdentCase, hvorved der er etableret en én-til-flere-relation mellem testforløb og ekspositioner ved hjælp af dummy-rækker.

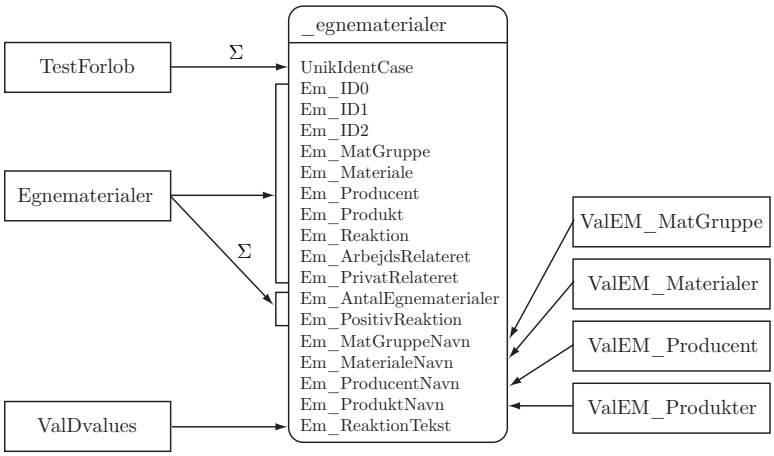
_ekspositioner indeholder endvidere kolonnen UnikIdentPlaster, der korresponderer med kolonnen af samme navn i tabellen **_plaster**. Der er ikke oprettet dummy-rækker i **_ekspositioner**, og således er oprettet en én-til-nul-eller-flere-relation mellem plastre og ekspositioner.

[1]En dansk oversættelse af WHO's ICD-10-diagnoser (The International Statistical Classification of Diseases and Related Health Problems 10^{th} Revision) findes i Sygehusvæsnets Klassifikationssystem under Sundhedsstyrelsen (SKS-browseren). Link: `http://www.medinfo.dk/sks/brows.php`



Figur 8.6: Oversigt over relationer mellem tabellerne **_testforlob**, **_plaster** og **_ekspositioner**.

Som beskrevet i afsnit 6.4.2 indebærer dette valg, at brugerne ved udvikling af forespørgsler, der implicerer både **_plaster**- og **_ekspositioner**-tabellerne, og hvor ekspositionerne skal være relateret til bestemte plastre, vil være tvunget til at anvende `OUTER JOIN`, der i tidligere tests var kilde til mange fejl. For at undgå, at den applikation, der anvendes i nærværende test, udfører automatisk `INNER JOIN` — hvilket brugerne skønnes at skulle bruge i de færreste tilfælde — er UnikIdentPlaster ikke gjort til primærnøgle i tabellen **_plaster**. Brugerne er således nødt til selvstændigt korrekt at relatere de to tabeller til hinanden, når de udvikler forespørgsler. For at øge sandsynligheden for dette er som tidligere nævnt oprettet kolonnen Pl_AntalEkspositionPlaster i tabellen **_plaster**. Denne kolonne, der er en optælling af antal registrerede ekspositioner for det enkelte plaster, søger altså at bidrage til korrekthed i de udviklede forespørgsler. Endvidere findes i **_ekspositioner** en kolonne, der indeholder nummeret for det allergen, der findes i det plaster, som den enkelte eksposition knytter sig til. Dette tænkes også at kunne bidrage til, at brugerne relaterer tabellerne korrekt ved forespørgselsudformning.



Figur 8.7: Oversigt over **_egnematerialer**.

Tabellen **_egnematerialer** indeholder ud over UnikIdentCase primært en lang række kolonner fra den originale tabel **Egnematerialer** samt nogle kolonner

fra ikke-personbundne tabeller, der hidrører fra tekstlig beskrivelse af blandt andet materiale og producent for de enkelte egnematerialer. Desuden findes to aggregeringer; en optælling af totalt antal egnematerialer for testforløbet samt en entydig indikation af, hvorvidt personen har reageret positivt på testen med det pågældende egnemateriale.

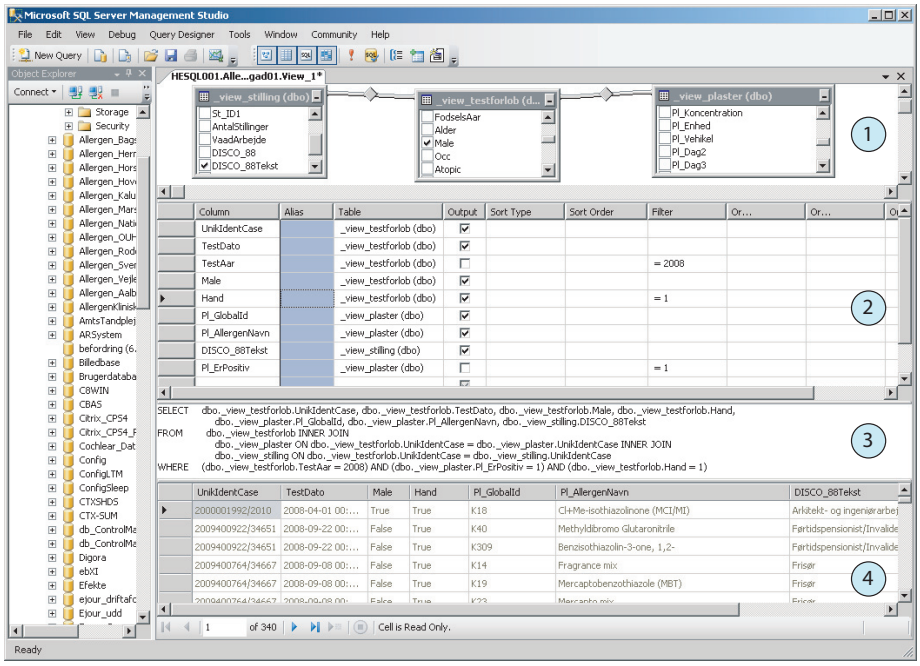
Opsummerende kan siges om den nye datamodel, at den søger at

- undgå problemer i forbindelse med `NATURAL JOIN`, idet kolonner navngives med præfix svarende til den tabel, de optræder i.
- minimere antallet af fejl i forbindelse med `JOIN`-operationer på sammensatte nøgler ved i så høj grad som muligt at samle disse nøgler i én kolonne.
- fjerne behovet for, at brugerne skal anvende `OUTER JOIN` ved indførelse af dummy-rækker i den udstrækning, det skønnes profitabelt.
- prækalkulere værdier, der typisk vil indgå i de fleste forespørgsler og ofte vil blive anvendt som filtreringsgrundlag.
- udnytte den hierarkiske inddeling af dimensioner kendt fra multidimensionel dataanalyse, hvor det skønnes frugtbart.
- minimere antallet af fejl ved anvendelse af datoekstraktionsfunktioner og `SUBSTRING` i forbindelse med værdier, der skal anvendes til filtrering eller indgå i forespørgsler.
- foretage optælling af ikke-tomme forekomster (rækker) i øvrige tabeller som en kontrolinstans, der kan minimere fejl.
- minimere fejl ved brug af `GROUP BY` ved at konvertere kolonner af typen `BIT` til typen `INTEGER`.
- mindske antallet af tabeller ved denormalisering. Tekstbaserede kolonner i ikke-personbundne tabeller inkluderes i personbundne tabeller. Samtidig minimeres antallet af `JOIN`-operationer, der kan slå fejl.

Den komplette kildekode, der danner de overfor beskrevne tabeller, findes i bilag D.

8.2 Microsoft SSMS

I den tredje test fik brugerne udleveret en oversigt over den nye datamodel, som afbildet i figur 8.1 og blev bedt om at udforme dataudtræk svarende til de tidligere beskrevne testscenarier ved hjælp af applikationen Microsoft SQL Server 2008 Management Studio (SSMS). SSMS er et komplet DBMS og stiller et visuelt værktøj til udvikling af forespørgsler — kaldet Query Designer —



Figur 8.8: Eksempel på udformning af forespørgsel i Microsoft SMSS.

til rådighed for brugeren. I figur 8.8 ses et skærmdump af dette værktøj, hvis grafiske grænseflade beskrives i det efterfølgende.

Vertikalt er Query Designer delt op i fire ruder (eng: *panes*):

1. Diagramrude: *Diagrammatisk afbildning af de views eller tabeller, der aktuelt anvendes, samt deres respektive kolonner og de indbyrdes relationer, brugeren definerer. Primærnøgler er markeret med fed og valgte kolonner markeret med flueben. Eventuelle filtreringer eller sorteringer er markeret ud for hver kolonne.*

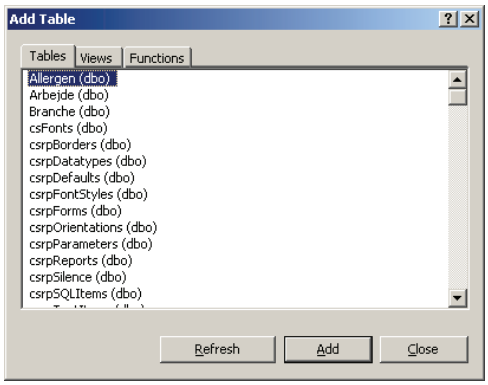
2. Kriterierude: *QBE-lignende grænseflade, der afbilder alle de kolonner, der ligger til grund for forespørgslen; valgte kolonner, kolonner, der filtreres eller sorteres på baggrund af samt kolonner, der enten er aggregater eller resultater af funktionskald.*

3. SQL-rude: *Tekstuel angivelse af SQL-koden svarende til forespørgslen.*

4. Resultatrude: *Angivelse af resultatet af forespørgslen i tabelform. Tabellen opdateres hver gang forespørgslen eksekveres.*

Ved åbning af Query Designer ses en oversigt over tilgængelige tabeller, views og funktioner (figur 8.9). Brugeren dobbeltklikker på de ønskede komponenter,

som derefter tilføjes i diagramvinduet. Applikationen forsøger automatisk at samle tabellerne ved hjælp af en `INNER JOIN` på første sammenfald mellem kolonner i tabellerne, hvis én af disse er del af eller hele primærnøglen. Det er dog værd at bemærke, at der udføres automatisk `EQUI JOIN` og ikke `NATURAL JOIN`, hvilket applikationen ikke understøtter eksplicit brug af. Brugeren skal således — for eksempel i de tilfælde, hvor primærnøglen er sammensat — selv ændre den automatisk oprettede `EQUI JOIN`, så denne omfatter alle relevante kolonner. Hvis der blandt tabellerne ikke er primærnøgler, eller der ikke er noget navnesammenfald mellem en tabels primærnøgle og øvrige tabeller udføres automatisk `CROSS JOIN`.

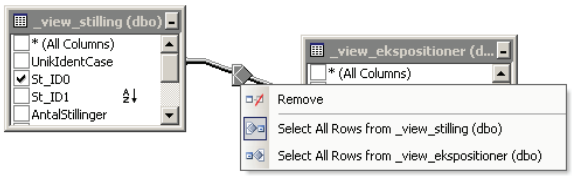


Figur 8.9: Liste over tilgængelige tabeller, views og funktioner ved forespørgselsudformning.

I diagramvinduet kan brugeren derefter tilføje, ændre eller slette `JOIN`-operationer, hvilket er afbildet i figur 8.10. Hvis der ønskes anvendt en `INNER JOIN`-operation, der ikke er `EQUI JOIN`, skal dette gøres direkte i SQL-koden, der vises i SQL-ruden. Brugeren kan i diagramvinduet også markere hvilke kolonner, der skal tilføjes forespørgslen, og som dermed vil optræde i kriterieruden. Endvidere kan brugeren ved at føre musepilen hen over specifikke kolonner eller forbindelser se art og kriterier for disse.

I kriterieruden kan brugeren definere kriterer for filtrering af forespørgslen og sortering (rækkefølge samt præcedens) af forespørgselsresultatet samt bestemme, hvorvidt en kolonne skal udskrives i den endelige forespørgsel. Desuden kan tilføjes kolonner fra de originale tabeller eller defineres nye kolonner baseret på for eksempel funktionskald. Hvis brugeren har valgt at anvende `GROUP BY`-funktionaliteten (vælges ved klik på knap i en værktøjslinie), kan denne endvidere anvende alle de aggregeringsfunktioner, SQL stiller til rådighed. Disse vælges for hver kolonne/kriterie i ruden.

I forbindelse med anvendelse af `GROUP BY`-funktionaliteten er applikationen tilsyneladende noget uhensigtsmæssigt indrettet. Der tilføjes automatisk ag-



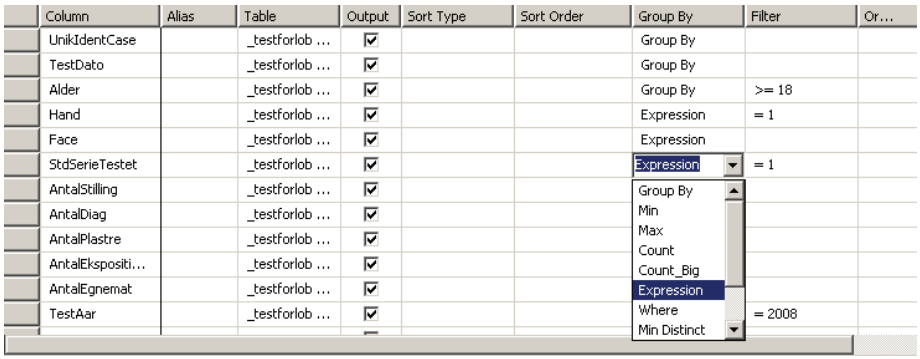
Figur 8.10: Eksempel på ændring af eksisterende `JOIN`-operation mellem to tabeller. Brugeren kan enten slette forbindelsen eller vælge mellem `INNER JOIN` og de tre typer `OUTER JOIN`.

Figur 8.11: Fejlmeddelelse ved anvendelse af automatisk aggregering, der implicerer kolonner af typen `BIT`.

gregeringsfunktioner for samtlige rækker i kriterieruden; i de fleste tilfælde tilføjes `GROUP BY`, men i de tilfælde hvor datatypen for den korresponderende kolonne i kildetabellen eller -viewet er `BIT`, tilføjer applikationen funktionen `EXPRESSION`. Hvis brugeren uden videre forsøger at eksekvere den udviklede SQL-kode, vil dette resultere i en fejlmeddelelse, som den afbildet i figur 8.11.

Fejlmeddelelsen skyldes selvfølgelig, at de førnævnte kriterier af datatypen `BIT` ikke er indeholdt i den samlede forespørgsels `GROUP BY` (eller `HAVING`), hvilket alle øvrige kriterier, for hvilke der automatisk tilføjes `GROUP BY` som aggregeringsfunktion, er. Fejlen rettes ved at ændre samtlige fejlbehæftede kriterier til enten `GROUP BY`, hvis der blot skal grupperes, eller `WHERE` hvis der skal filtreres på baggrund af det enkelte kriterie. Dette er afbildet i figur 8.12. Rent praktisk kan dette problem selvfølgelig omgås ved at konvertere berørte kolonner til typen `INTEGER` (heltal).

Mens brugeren definerer forespørgslen genereres den korresponderende SQL-kode automatisk i SQL-ruden. Brugeren kan til enhver tid vælge at eksekvere



Figur 8.12: Eksempel på rettelse af automatiske aggregeringer, der fejler, når kolonnetypen er BIT. Ved gruppering skal aggregeringsfunktionen ændres til `GROUP BY` og ved filtrering til `WHERE`.

forespørgslen og se den resulterende tabel i resultatruden. I SQL-ruden kan brugeren endvidere redigere SQL-koden, eksekvere forespørgslen og se eventuelle ændringer afspejles i de øvrige ruder i Query Designer. Brugeren kan altså ved hjælp af trial and error opbygge sin forespørgsel trin for trin.

8.3 Resultater

Ingen af testpersonerne havde problemer med at udvikle forespørgsler svarende til testscenarie 1–3. I scenarie 3, hvor forespørgslen kun skal indeholde testforløb fra 2008, opdagede testperson 1 ikke kolonnen TestAar, men filtrerede i stedet på 31-12-2007 $<$ TestDato $<$ 01-01-2009, hvor testperson 2 og 3 som ventet filtrerede på TestAar $=$ 2008.

Testpersonerne opnåede med varierende succes at udforme en forespørgsel svarende til testscenarie 4. Alle testpersonerne formulerede grundlæggende forespørgslen korrekt, men det lykkedes kun for testperson 1 at fjerne dubletter i det resulterende dataudtræk, idet hun — lige som ved anden test — fandt `GROUP BY`-funktionaliteten. I dette tilfælde anvendte hun `COUNT DISTINCT` på kolonnen TestDato og opnåede det ønskede resultat. Testperson 2 forsøgte lige som i første test at at filtrere på baggrund af kolonnen ID1, denne gang ligeledes uden succes.

Det viste sig, at alle personerne med succes kunne realisere testscenarie 5. Applikationen udførte automatisk en `EQUI JOIN`, så snart testpersonerne havde tilføjet de to i udtrækket implicerede tabeller. Person 3 godtog uden videre resultatet, mens de andre personer brugte tid på at overveje og efterprøve, om relationen var rigtig. Person 1 ændrede den automatiske `EQUI JOIN` til en `LEFT OUTER JOIN`, hvilket ikke gjorde nogen forskel, idet der netop var indført dummy-

rækker i tabellen **_diagspec**.

Testscenarie 6 kunne både testperson 1 og 2 grundlæggende udføre med succes, i modsætning til i de to tidligere tests, hvor samtidig filtrering på baggrund af stilling og de tre reaktionsdage voldte problemer. De filtrerede begge korrekt efter stilling på baggrund af kolonnen St_DISCO-88 og positive reaktioner efter kolonnen Pl_ErPositiv. Dog glemte de begge at udskrive selve reaktionerne for de tre reaktionsdage, hvilket var krævet i scenariet. Testperson 3 fejlede, idet han ikke opdagede kolonnen Pl_ErPositiv i tabellen **_plaster**, og dermed forsøgte at filtrere på baggrund af alle tre reaktionsdage samtidigt med stillingskoden. Trods den overskuelige QBE-lignende grænseflade lykkedes det ham ikke at udforme kriteriet. Figur 8.13 illustrerer hans forsøg, mens det i figur 8.14 ses, hvordan dette udføres korrekt.

	Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...	Or...
	T_ID0		_testforlob (dbo)	☑					
▶	TestDato		_testforlob (dbo)	☑					
	Male		_testforlob (dbo)	☑					
	Hand		_testforlob (dbo)	☑					
	St_DISCO_88		_stilling (dbo)	☑			= '51411'		
	Pl_AllergenNo		_plaster (dbo)	☑					
	Pl_AllergenNavn		_plaster (dbo)	☑					
	Pl_Dag2		_plaster (dbo)	☑			>= '1' AND <= '3'		
	Pl_Dag3		_plaster (dbo)	☑				>= '1' AND <= '3'	
	Pl_Dag57		_plaster (dbo)	☑					>= '1' AND <= '3'

Figur 8.13: Mislykket forsøg på filtrering på baggrund af stillingskode og de tre reaktionsdage.

Det skal dog siges, at alle testpersonerne lykkedes med at udføre korrekte `JOIN`-operationer på de tre indbefattede tabeller, idet applikationen automatisk tilføjede `EQUI JOIN`.

	Hand		_testforlob (dbo)	☑			
	St_DISCO_88		_stilling (dbo)	☑			= '51411'
	Pl_AllergenNo		_plaster (dbo)	☑			
	Pl_AllergenNavn		_plaster (dbo)	☑			
	Pl_ErPositiv		_plaster (dbo)	☐			= 1
	Pl_Dag2		_plaster (dbo)	☑			

Figur 8.14: Korrekt filtrering på baggrund af stillingskode og kolonnen Pl_ErPositiv.

Ingen af testpersonerne lykkedes med selvstændigt at formulere en forespørgsel svarende til scenarie 7, som det var tiltænkt. Med udgangspunkt i den nye datamodel skal anvendes en `OUTER JOIN`, når **_eksspositioner** skal relateres til de øvrige tabeller. Denne relation skal endvidere være sammensat, idet den enkelte eksposition skal relateres til et bestemt plaster.

Testperson 2 fejlede, idet han kom til at anvende tabellen **_egnematerialer** i stedet for **_eksspositioner**, men efter et vink, kom han i den rigtige retning og lykkedes med scenariet. Testperson 1 og 3 misforstod intentionen med scenariet,

hvis tekstlige ordlyd gennem alle tests og afprøvninger har været:

> *Hent ID0, fødselsdato, testdato, køn, eventuelle stillingskoder, alle reaktioner (også negative), nuværende og tidligere relevans samt materialegruppekoder for samtlige eventuelle ekspositioner for alle standardserietestede personer i databasen, der er testet med et allergen med globalid = K7.*

Intentionen med scenariet er, at de eventuelle ekspositioner, som en tilsvarende forespørgsel leverer, alle skal være relateret til plastre indeholdende et allergen med globbalid = K7. De to testpersoner forstod — hvilket set retrospektivt er ganske berettiget, men erfaringsmæssigt ikke er meningsfyldt i forhold til dataudtræk i den aktuelle database — scenariet således, at de skulle finde alle eventuelle ekspositioner — uanset om disse var relateret til et allergen med globalid = K7. Testperson 1 ændrede relationen mellem **_ekspositioner** og de øvrige tabeller til en `OUTER JOIN` og leverede således reelt en forespørgsel svarende til hendes fortolkning af den tekstlige ordlyd af scenariet. Relationen blev dog ikke sammensat, og der var således ingen korrekt forbindelse mellem de valgte plastre og ekspositioner.

8.4 Diskussion

Overordnet kan siges om resultaterne af tredje test, at der er sket store forbedringer i forhold til de tidligere tests. I store træk lykkedes testpersonerne godt med at udvikle forespørgsler svarende til testscenarierne. Dog udestår stadig nogle problemer.

Designet af den nye datamodel og valget af applikation er sket på baggrund af erfaringer gjort i de tidligere tests og afprøvninger. Som konkret eksempel kan nævnes scenarie 5, der slet ikke kunne realiseres i første test, idet applikationen ikke tillod `OUTER JOIN` på sammensatte nøgler. I anden test lykkedes testpersonerne heller ikke på egen hånd med at udforme korrekte forespørgsler, idet testpersonerne enten glemte at anvende `OUTER JOIN`, glemte at betinge `JOIN`-operationen af hele den sammensatte nøgle eller helt glemte at relatere tabellerne, hvilket medførte en `CROSS JOIN`. I nærværende test havde ingen af testpersonerne problemer med at udforme korrekte forespørgsler.

Generelt gælder for de udviklede forespørgsler, at der ikke længere er problemer med at forbinde tabeller, hvor der tidligere var sammensatte primærnøgler, hvilket både applikationer og brugere havde problemer med. Ligeledes gælder, at tabeller der før skulle relateres ved hjælp af `OUTER JOIN` nu menings- og succesfuldt kan relateres ved hjælp af en automatisk genereret `EQUI JOIN`. Det eneste udestående problem, der var med at udføre korrekte `JOIN`-operationer, var i scenarie 7, hvor plastre og ekspositioner skal relateres. Kun en af testpersonerne lykkedes med at formulere en korrekt `OUTER JOIN`. Det skønnes dog, at

den i nærværende datamodel anvendte løsning på relationsproblemet (beskrevet i afsnit 6.4.2) er den bedste.

Erfaringsmæssigt vil brugerne af databasen hyppigst forespørge samtlige reaktioner for alle testforløb i databasen, mens de oftest kun er interesseret i de konkrete ekspositioner for testforløb, hvor der er positive reaktioner på bestemte stoffer. Dette er oplagt, da der kun vil være registreret ekspositioner for plastre, der har været en positiv reaktion overfor. Det vurderes, at arbejdsgangen for brugerne vil være således, at de først vil udforme forespørgsler, der indbefatter alle personer, der er testet for et bestemt stof inden for en given tidsramme. Siden vil brugerne på baggrund af de fundne positive reaktioner — hvoraf lægen har bedømt, at reaktionen var relevant for det pågældende eksemudbrud — forespørge de registrerede ekspositioner. Problemets omfang skønnes således at være begrænset. Samtidig er i tabellen **_plaster** foretaget optælling af antallet af registrerede ekspositioner for hvert enkelt plaster, hvilket kan bruges som kontrol for brugerne.

Tredje test viste også, at nye de aggregater og prækalkuleringer bidrog til en effektiv filtrering, omend testscenarierne langt fra var udtømmende i forhold til at afprøve disse i deres fulde potentiale. Det er værd at bemærke, at testpersonerne ikke i alle tilfælde udnyttede de nye kolonner til filtrering, men i stedet benyttede samme filtrering, som de benyttede — eller forsøgte at benytte — i første og anden test under anvendelse af den originale datamodel. Dette kan dels skyldes, at testpersonerne af forskellige årsager ikke var opmærksomme på de nye kolonner. Det tænkes endvidere, at der kan være tale om en genkendelseseffekt, hvor testpersonerne forsøger at løse scenarierne ensartet i forhold til de to tidligere tests; dette til trods for, at tredje test blev udført fire måneder efter, at anden test blev udført. Den mulige genkendelseseffekt kan også være det fænomen, at testpersonerne havde en bestemt grundlæggende tilgang til at løse de samme problemer uanset datamodel eller anvendt applikation.

Endnu et udestående emne, der viste sig i testen, var eksistensen af dubletter (helt identiske rækker) og brugernes forsøg på fjernelse af disse. Dubletterne kan som tidligere berørt fjernes effektivt med `DISTINCT` eller en af aggregeringsfunktionerne i `GROUP BY`-funktionaliteten. Førstnævnte var kun tilgængelig i SSMS ved redigering af SQL-koden, mens sidstnævnte er tilgængelig via tryk på en knap i en værktøjslinie. Selv om aggregeringsfunktioner kan være svære at forklare og forstå til fulde, må den mest plausible løsning være at lære brugerne at trykke på denne knap, hvis deres udtræk skal være dubletfri. Omvendt kan diskuteres, hvorvidt dubletter i praksis vil vise sig at være et problem. Forskerne, der arbejder med udtræk fra databasen, er vant til at manipulere data i SPSS; herunder at bruge den i applikationen indbyggede funktion *Identify duplicate cases*, der for en datatabel effektivt identificerer dubletter.

Med indførelsen af den nye datamodel er der opstået mange nye kolonner, der skal understøtte brugerne i deres udformning af forespørgsler. Nærværende test kan dog tyde på, at testpersonerne havde svært ved at overskue de mange kolonner, deres indhold og anvendelsesmuligheder. Selv om det under navngivningen

af kolonner er tilstræbt, at datamodellen i så høj grad som muligt skal være intuitiv og selvforklarende i brug, må det erkendes, at der kan være behov for en beskrivelse af kolonnerne og deres indhold.

Generelt kan siges, at den nye datamodel kan virke i praksis, omend der i nærværende test viste sig nogle problemer. Modellen er dog også indrettet således, at den i vid udstrækning vil kunne bruges sammen med de applikationer, der blev anvendt i første og anden test, idet modellen tager højde for de fleste af de problemer, der har vist sig ved de tidligere tests og afprøvninger. Således vil brugerne selv kunne vælge, hvilken af de tre applikationer de vil bruge til at tilgå tabellerne i modellen. Samtidig vil tabellerne også kunne anvendes til at danne kuber, og brugerne vil dermed også kunne bruge modellen til simpel multidimensionel dataanalyse. Hvis dette skal anvendes i større stil, skal der dog udvikles nogle større OLAP-tabeller, der bedre knytter tabellerne i nærværende model sammen.

Kapitel 9

Sammenfatning

Dette speciale er opstået på baggrund af en konkret observation foretaget i IT-funktionen i Videncenter for Allergi: En del forskere henvender sig hyppigt med henblik på udvikling af databaseforespørgsler, der strukturelt er ensartede og ofte implicerer data fra de samme fem–ti tabeller fra databasen. Forespørgslerne — og de præmisser eller filtreringskriterier, der ligger til grund for dem — er dog alligevel så forskellige, at der ikke umiddelbart kan oprettes faste rapporter eller generiske modeller, der vil opfylde brugernes behov. Samtidig havde mange af forskerne ikke fra start et klart billede af de data, de forespurgte, og således måtte en del af deres dataudtræk revideres løbende i takt med, at deres forståelse for de data, de modtog, steg.

Fra IT-funktionens side blev iagttaget en del repetition i forbindelse med udviklingen af forespørgslerne, og der blev øjnet en mulighed for i højere grad at inddrage forskere i udviklingen af egne forespørgsler og dermed øge deres forståelse for data, hvilket på sigt kunne føre til, at forskerne selv foretog dataudtræk.

I forbindelse med specialet blev afsøgt litteratur inden for forskningsfeltet slutbrugerudvikling, der tænktes at kunne bidrage med løsninger på en højere grad af inddragelse af slutbrugere i udviklingen af databaseforespørgsler. Emnerne programmering ved eksempel, softwareformningsworkshopper, samarbejde om tilpasning og naturlige sprog blev belyst, men skønnedes ikke umiddelbart at være anvendelige i forhold til specialet. Emnerne participatorisk programmering, SBU i organisationer og visuelle grænseflader skønnedes relevante, og resultater og pointer inden for disse emner blev fremhævet og er søgt tilgodeset og implementeret i den endelige datamodel, der blev udviklet i forbindelse med tredje test. En effekt af dette er dog ikke umiddelbart påvist, da det skønnes at være uden for specialets kerneområde.

Siden blev afholdt brugertests, der involverede applikationer møntet på slutbrugere. Disse tests bidrog til at påpege svagheder ved den eksisterende data-

model, problemer med applikationernes virkemåde og håndtering af bestemte situationer samt de forståelsesmæssige barrierer, der eksisterede blandt brugerne i forhold til at udforme korrekte forespørgsler. Endvidere blev afholdt to afprøvninger, hvoraf den første søgte at sandsynliggøre, at en omstrukturering af den originale datamodel kunne bidrage til specialets løsning, mens anden afprøvning søgte at belyse, hvorvidt multidimensionel dataanalyse eller elementer herfra kunne appliceres.

Med baggrund i erfaringer fra de afholdte tests og afprøvninger blev udarbejdet en ny datamodel, der sidenhen blev genstand for en tredje brugertest under anvendelse af en professionel og tidssvarende forespørgselsapplikation. Den nye datamodel var indrettet under hensyntagen til de erfaringer, der var gjort under tidligere tests og afprøvninger, og tredje test viste, at det var muligt for testpersonerne at udvikle forespørgsler svarende til de beskrevne testscenarier via den nye datamodel. Testen viste dog også, at der blandt testpersonerne stadig var nogle forståelsesmæssige problemer i forhold til data, lige som der var problemer og uhensigtsmæssigheder i forhold til forespørgselsapplikationens håndtering af datamodellen og de scenarier, testpersonerne forsøgte at realisere. Forløbet med tests og afprøvninger har sammenlagt udgjort en meget erfaringsgivende og frugtbar proces, der har affødt en række konkrete gode råd til udviklere af databasemodeller og forespørgselsapplikationer.

9.1 Gode råd

På baggrund af de tests og afprøvninger, der er udført i relation til specialet, er gjort en del væsentlige erfaringer, både hvad angår datamodeller, brugere og forespørgselsapplikationer. Da disse erfaringer tænkes at være relevante for organisationer, der lige som Videncenter for Allergi indsamler casedata beregnet på efterfølgende statistisk bearbejdning, og hvor brugerne søges inddraget i udviklingen af databaseforespørgsler, opstilles i det efterfølgende en række gode råd, der baserer sig på de vundne erfaringer.

9.1.1 Databasemodeller

I punktform findes her en række gode råd til udviklere af datamodeller, der skal understøtte slutbrugerinddragelse i udviklingen af korrekte forespørgsler baseret på data på caseniveau:

- Opret en casebundet model centreret om en tabel, der indeholder karakteristika for den enkelte case samt optællinger af casespecifikke forekomster (ikke dummy-rækker) i de øvrige tabeller. Opret en primærnøgle, der også findes i de øvrige tabeller.
- Anvend denormalisering til at begrænse antallet af tabeller og eliminere ikke-casebundne tabeller.

- Navngiv kolonner (ikke casespecifikke nøgler) unikt på tværs af tabeller.
- Saml sammensatte nøgler i én kolonne.
- Opret dummy-rækker i tabeller, hvor andelen af dummy-rækker ikke bliver for stor.
- Prækalkulér/ekstrahér værdier og logiske udtryk, der typisk vil indgå i forespørgsler og blive anvendt som filtreringsgrundlag.
- Simulér hierarkisk dimensionsinddeling af kolonner, der indeholder datoer eller tekstlige klassifikationsstrukturer ved tilføjelse af kolonner.
- Konvertér kolonner af typen `BIT` til `INTEGER`.

Ovenstående råd sigter mod, at den udviklede model skal understøtte så mange forskellige forespørgselsapplikationer som muligt. Visse af ovenstående råd knytter sig særligt til bestemte af de i projektet involverede applikationers håndtering af forskellige situationer. Ovenstående råd skal derfor ses i relation til den eller de applikationer, der skal anvendes i forbindelse med en eventuel forestående implementering af nye datamodeller. Nogle råd vil være frugtbare, mens andre sandsynligvis ikke vil være applicérbare i den konkrete situation.

9.1.2 Forespørgselsapplikationer

I forbindelse med afprøvninger og afvikling af tests er gjort væsentlige erfaringer med de implicerede forespørgselsapplikationer. Disse erfaringer er dels opnået ved at observere, hvordan brugerne interagerer med systemerne, dels ved at betragte, hvorledes applikationerne håndterer bestemte situationer, og hvordan man i applikationerne konkret kan løse eventuelle fejl og problemer.

- Stil en simpel funktion til fjernelse af dubletter til rådighed, for eksempel en separat `DISTINCT`-knap.
- Giv brugeren mulighed for at vælge, hvorvidt tabeller automatisk skal relateres.
- Giv brugeren mulighed for at vælge, hvilken type `JOIN`-operation, der som udgangspunkt skal forsøges anvendt.
- Hvis automatiske relationer er valgt, forsøg da først at tilføje `NATURAL JOIN`, siden `CROSS JOIN`.
- Tillad `OUTER JOIN` på sammensatte nøgler i flere tabeller samtidigt.
- Tillad `FULL OUTER JOIN`.

9.2 Validering

Der er i forbindelse med specialet ikke udført en udtømmende validering af de data, den nye datamodel måtte generere, idet dette skønnes at være et meget omfangsrigt selvstændigt projekt. I stedet er taget udgangspunkt i de syv testscenarier. Således er udviklet forespørgsler svarende til de syv scenarier med udgangspunkt i både den originale og den nye datamodel. Rækkerne i de resulterende dataudtræk genereret af førnævnte forespørgsler er siden blevet talt op, og en sorteret stikprøve (fem-ti rækker) er taget for på rækkeniveau at kunne validere samtlige kolonner. I tilfælde af dubletter i disse udtræk er anvendt funktionen `DISTINCT`. Skærmdump af eksekvering af forespørgslerne er at finde bilag C.

Valideringen viser, at en eksekvering af forespørgsler genereret ved hjælp af de to datamodeller leverer de samme dataudtræk for testscenarie 1, 3, 4, 6 og 7. For scenarie 2 og 5 gælder, at der findes 29 rækker mere i de dataudtræk, der er genereret via den originale datamodel. Dette skyldes, at der i tabellen **TestForlob** findes 29 rækker, der ikke findes i tabellen **_testforlob**. Disse 29 rækker er *herreløse data* idet der ikke findes data med tilsvarende ID0 i nogen andre tabeller i databasen, heller ikke i tabellen **Person**. Rækkerne er ikke oprettet i den nye model, idet der ved dannelsen af tabellen **_testforlob** ved hjælp af en `INNER JOIN` blev inkluderet data fra tabellen **Person**, der som bekendt blev overflødig grundet indførelsen af denormalisering.

Valideringen af data viser altså overordnet, at modellen er blevet en succes, hvad angår korrektheden af de genererede data. Som nævnt er der ikke foretaget en udtømmende validering, men mens projektet er pågået, har en del medarbejdere henvendt sig til IT-funktionen med henblik på at få udarbejdet forespørgsler. I alle de tilfælde, hvor den nye datamodel indeholdt de for forespørgslerne tilstrækkelige kolonner, blev der udarbejdet to forskellige forespørgsler. Den ene baserede sig på den nye datamodel, mens den anden havde grundlag i den originale datamodel. I alle disse tilfælde viste det sig, at forespørgslerne leverede identiske dataudtræk.

9.3 Konklusion

Følgende mål for specialeprojektet blev defineret (i kort form):

1. At oprette en begrænset datamodel.
2. At stille en visuelt orienteret grænseflade til rådighed for slutbrugerne.
3. At teste og evaluere systemet ved involvering af slutbrugere.
4. At validere data genereret via den nye datamodel.

I projektet er oprettet en ny begrænset datamodel, der i højere grad understøtter udvikling af korrekte forespørgsler inden for det specifikke domæne. En visuelt orienteret grænseflade er via applikationen Microsoft SQL Server 2008 Management Studio stillet til rådighed for slutbrugerne. Endvidere er førstnævnte datamodel testet af slutbrugere, og resultaterne af disse tests er siden evalueret og diskuteret. Siden er validiteten af de data, som de udviklede forespørgsler genererede, blevet efterprøvet og godkendt i forhold til den originale datamodel. Målene med specialet vurderes således sammenlagt at være nået.

Under udførelsen af specialet er der endvidere gjort vigtige erfaringer, og der er opstået interessant viden om testpersonerne og de applikationer og teknikker, der anvendtes i tests og afprøvninger. Således står det klart, at mængdelære og udformning af (tilsyneladende simple) logiske udtryk stadig er en udfordring i forhold til at gøre det muligt for slutbrugere på egen hånd at udvikle korrekte ad hoc-databaseforspørgsler. Specialet har konkret vist, at en omsktrukturering af datamodellen baseret på erfaringer i de afholdte tests kan minimere sandsynligheden for, at slutbrugerne udformer fejlbehæftede databaseforespørgsler, men ikke eliminere alle fejl. Samtidig har specialet affødt en række gode råd, der kan anvendes i lignende situationer.

Før tests blev udført, blev der gjort en række forudsigelser omkring mulige årsager til, at testpersonerne ikke ville lykkes med at udforme korrekte forespørgsler. Disse forudsigelser blev delvis honoreret ved tests, men i takt med, at mere professionelt programmel belv taget i brug, og datamodellen blev indrettet ud fra viden om tidligere fejl, minimeredes førnævnte årsagers betydning dog. Modellering, der indebar en høj grad af denormalisering, skønnes at være det mest betydningsfulde bidrag, men også prækalkulering og aggregeringer bidrog. Blandt forudsigelserne står således brugernes manglende forståelse for mængdelære samt uintuitive eller uigennemskuelige applikationer som væsentligste kilder til, at brugerne potentielt mislykkes med at udforme forespørgsler. Det er interessant at konstatere, at det for selv domænevante motiverede IT-brugere — til trods for den stigende udbredelse af IT i samfundet og den generelle høje grad af slutbrugerudvikling — reelt stadig er forståelse af basal mængdelære og simpel logik, der er afgørende for, hvorvidt der kan udformes korrekte forespørgsler. Applikationerne har udviklet sig meget gennem de sidste 20–30 år, men brugernes problemer i forhold til forespørgselsformulering er stadig de samme, som er beskrevet i litteraturen i samme periode.

Afrundende kan siges om nærværende speciale, at der er oprettet en datamodel, der muliggør udvikling af ad hoc-forespørgsler. Denne model kan tilgås via flere forskellige applikationer, idet den tager højde for de problemer, der viste sig ved tests og afprøvninger. Samtidig søger den at integrere de elementer fra multidimensionel dataanalyse, der blev vurderet som værende anvendelige, lige som det er beskrevet, hvordan multidimensionel dataanalyse i simpel form kan bringes i drift og fremadrettet blive brugt til at skabe hurtige overblik i forbindelse med mindre statistikker.

9.4 Videre arbejde

Konkret er i projektet implementeret en simplere datamodel, der muliggør enklere og hurtigere udvikling af ad hoc-databaseforespørgsler. Modellen er i produktion og anvendes med succes af IT-funktionen som foretrukkent redskab ved udvikling af forespørgsler grundet sin overskuelighed og effektivitet. Denormalisering og prækalkulering/aggregering skønnes at være væsentligste årsager til dette. Under projektets udførelse er flere gange opstået behov for at udarbejde ad hoc-forespørgsler til forskerne. I de tilfælde, hvor forespørgeren var en af de tre testpersoner fra projektet, foretog denne i så høj grad som muligt selv sin forespørgsel. Testpersonerne syntes trygge ved både datamodellen og forespørgselsapplikationen.

Datamodellen er aktuelt kun implementeret for én kliniks vedkommende, men bredes ud, så den fremadrettet inkluderer data fra alle indberettende klinikker. Endvidere forestår et stort arbejde med at dokumentere modellen og udbrede kendskab til og anvendelse af modellen blandt forskerne i Videncenter for Allergi. Dette skønnes blandt andet at indebære introduktions- og undervisningssessioner — både hvad angår datamodel og forespørgselsapplikation. Fremadrettet skal endvidere udføres en række brugerundersøgelser, der skal afdække hvilke views, der er behov for, for at integrere multidimensionel dataanalyse/pivottabeller som en naturlig del af arbejdsprocessen omkring udarbejdelse af statistikker til brug i for eksempel projektansøgninger.

Litteratur

[1] N. Boujemaa, J. Fauqueur & V. Gouet, What's beyond query by example?, *Trends and Advances in Content-Based Image and Video Retrieval*, Springer-Verlag (2004).

[2] J. Boyle, K. Bury & R. Evey, Two studies evaluating learning and use of QBE and SQL, *Proceedings of the Human Factors Society 27th Annual Meeting* (1983), 663–667.

[3] M. Breslin, Data warehousing battle of the giants: Comparing the basics of the Kimball and Inmon models, *Business Intelligence Journal* **9**, 1 (2004).

[4] T. Catarci, M. F. Costabile, S. Levialdi & C. Batini, Visual query systems for databases: A survey, *Journal of Visual Languages and Computing* **8** (1997), 215–260.

[5] H. C. Chan, H. H. Teo & X. H. Zeng, An evaluation of novice end-user computing performance: Data modeling, query writing, and comprehension: Research articles, *Journal of the American Society for Information Science and Technology* **56**, 8 (2005), 843–853.

[6] S. Chaudhuri & U. Dayal, An overview of data warehousing and OLAP technology, *ACM SIGMOD Record* **26**, 1 (1997), 65–74.

[7] P. P.-S. Chen, The entity-relationship model — toward a unified view of data, *ACM Transactions on Database Systems* **1**, 1 (1976), 9–36.

[8] E. F. Codd, A relational model of data for large shared data banks, *Communications of the ACM* **13**, 6 (1970), 377–387.

[9] E. F. Codd, S. B. Codd & C. T. Salley, Providing OLAP (on-line analytical processing) to user-analysis: An IT mandate, Hvidpapir, E. F. Codd Associates (1993).

[10] M. F. Costabile, D. Fogli, P. Mussio & A. Piccinno, End-user development: The software shaping workshop approach, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 183–205.

[11] S. Gade, Weballergen: idriftsættelse, oplæring og feedback — et usabilitystudie, Skriftligt arbejde, Datalogisk Institut Københavns Universitet (2007).

[12] W. H. Inmon, *Building the Data Warehouse*, 3. udgave, Wiley (2002).

[13] J. D. Johansen, Årsrapport 2008, Statusrapport, Videncenter for Allergi (2009).

[14] H. Kahler, Supporting Collaborative Tailoring, Ph.d.-afhandling, Institut for Kommunikation, Journalistik og Datalogi, Roskilde Universistetscenter (2001).

[15] R. Kimball & J. Caserta, *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*, 1. udgave, Wiley (2004).

[16] R. Kimball & M. Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, 2. udgave, Wiley (2002).

[17] C. Letondal, Participatory programming: Developing programmable bioinformatics tools for end-users, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 207–242.

[18] C. Letondal & W. E. Mackay, Participatory programming and the scope of mutual responsibility: balancing scientific, design and software commitment, *Proceedings of the 8th Conference on Participatory Design*, ACM (2004), 31–41.

[19] H. Lieberman & H. Liu, Feasibility studies for programming in natural language, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 459–473.

[20] H. Lieberman, F. Paternò, M. Klann & V. Wulf, End-user development: An emerging paradigm, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 1–8.

[21] H. Liebermann (Redaktør), *Your Wish is My Command: Programming by Example*, Morgan Kaufman (2001).

[22] N. Mehandjiev, A. Sutcliffe & D. Lee, Organizational view of end-user development, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 371–399.

[23] W. C. Ogden & S. R. Brooks, Query languages for the casual user: Exploring the middle ground between formal and natural languages, *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, ACM (1983), 161–165.

[24] J. F. Pane & B. A. Myers, More natural programming languages and environments, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 31–50.

[25] J. F. Pane, B. A. Myers & C. A. Ratanamahatana, Studying the language and structure in non-programmers' solutions to programming problems, *International Journal of Human-Computer Studies* **54**, 2 (2001), 237–264.

[26] T. B. Pedersen & C. S. Jensen, Multidimensional database technology, *Computer* **34**, 12 (2001), 40–46.

[27] V. Pipek & H. Kahler, Supporting collaborative tailoring. Issues and approaches, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 315–345.

[28] R. Ramakrishnan & J. Gehrke, *Database Management Systems*, 3. udgave, McGraw-Hill (2002) Kapitel 6.

[29] A. Repenning & A. Ioannidou, What makes end-user development tick? 13 design guidelines, *End-User Development*, *Human-Computer Interaction Series* **9**, Springer-Verlag (2006), 51–85.

[30] J. B. Smelcer, Database query composition and the role of user view of data, *ACM SIGCHI Bulletin* **18**, 3 (1987), 74–76.

[31] J. B. Smelcer, User errors in the use of the structured query language (SQL), *Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting* (1993), 382–386.

[32] J. B. Smelcer, User errors in database query composition, *International Journal of Human-Computer Studies* **42**, 4 (1995), 353–381.

[33] M. Y.-M. Yen & R. W. Scamell, A human factors experimental comparison of SQL and QBE, *IEEE Transactions on Software Engineering* **19**, 4 (1993), 390–409.

[34] M. M. Zloof, Query by example, *AFIPS National Computer Conference* (1975), 431–438.

Bilag A

JOIN-operationer

I nærværende oversigt over de forskellige typer JOIN-operationer anvendes nedenstående to tabeller til at visualisere konkrete eksempler:

Bogstav	Tal
a	1
b	2
c	3
d	4
e	2

Tal	Tegn
1	!
2	?
5	*

JOIN-operationer bruges til at oprette forbindelser mellem to eller flere tabeller i SQL-forespørgsler ved at sammenligne værdierne indeholdt i bestemte kolonner i disse tabeller. Overordnet findes to slags JOIN-operationer; INNER JOIN og OUTER JOIN, som hver især kan inddeles i nogle underkategorier. Endvidere anvendes af og til en tredje type JOIN-operation, der kaldes SELF JOIN, som kan bruges i de tilfælde, hvor man ønsker at samle en tabel med sig selv. Denne type JOIN-operation vil dog ikke blive berørt nærmere i nærværende oversigt.

INNER JOIN

INNER JOIN bygger grundlæggende på, at rækker (eller dele deraf) i to tabeller kombineres ved rækkevis at sammenligne kolonner. I de fleste tilfælde vil denne kombination være baseret på et prædikat (for eksempel $\leq$, $=$, $\neq$ eller $>$) for JOIN-operationen. Forespørgslen

```
SELECT   Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM     Tabel1 INNER JOIN Tabel2 ON Tabel1.Tal < Tabel2.Tal
```

resulterer således i følgende tabel:

Bogstav	Tal	Tal_2	Tegn
a	1	2	?
a	1	5	*
b	2	5	*
c	3	5	*
d	4	5	*
e	2	5	*

CROSS JOIN

I en `CROSS JOIN` kombineres to tabeller uden prædikat, hvilket betyder, at alle kombinationer af værdierne i de for `JOIN`-operationen relevante kolonner for alle rækker i de to tabeller. Operationen svarer således til det kartesiske produkt. Alle øvrige typer `INNER JOIN` kan således anskues som delmængder af en `CROSS JOIN` for to tabeller. Nedenfor ses et eksempel, hvor der udføres `CROSS JOIN` (implicit på baggrund af kolonnen Tal):

```
SELECT    Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM      Tabel1 CROSS JOIN Tabel2
```

Bogstav	Tal	Tal_2	Tegn
a	1	1	!
b	2	1	!
c	3	1	!
d	4	1	!
e	2	1	!
a	1	2	?
b	2	2	?
c	3	2	?
d	4	2	?
e	2	2	?
a	1	5	*
b	2	5	*
c	3	5	*
d	4	5	*
e	2	5	*

EQUI JOIN

`EQUI JOIN` er en `INNER JOIN`, hvor der specifikt anvendes prædikatet =. Det vil med andre ord sige, at der i resultatet af `JOIN`-operationen kun vil optræde kombinationer, hvor værdierne i de kolonner, der sammenlignes, er ens. Et eksempel på `EQUI JOIN` ses her:

```
SELECT    Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM      Tabel1 INNER JOIN Tabel2 ON Tabel1.Tal = Tabel2.Tal
```

Bogstav	Tal	Tal_2	Tegn
a	1	1	!
b	2	2	?
e	2	2	?

I følge SQL-standarden kan ovenstående forespørgsel omskrives til:

```
SELECT   Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM     Tabel1 INNER JOIN Tabel2 USING (Tal)
```

Ikke alle forespørgselsprogrammer understøtter denne omskrivning.

NATURAL JOIN

`NATURAL JOIN` kan anskues som en yderligere specialisering af `EQUI JOIN`, hvor det kræves, at værdierne i alle ensbenævnte kolonner i de to tabeller skal være ens. Findes således i (begge) to tabeller tre kolonner med det eksakt samme navn, vil resultattabellen af `JOIN`-operationen kun være præcis de rækker, hvor værdierne i kolonnerne for hver enkelt kombination indbyrdes er ens.

I nærværende eksempel med to simple tabeller, vil der ikke være nogen forskel på `EQUI JOIN` og `NATURAL JOIN`. En `NATURAL JOIN` formuleres i henhold til SQL-standarden således:

```
SELECT   Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM     Tabel1 NATURAL JOIN Tabel2
```

En del forespørgselsprogrammer tillader dog ikke brugen af `NATURAL JOIN`, men kræver i stedet, at brugeren formulerer forespørgslen som en `EQUI JOIN`/`INNER JOIN`, hvor der eksplicit sammenlignes på alle relevante kolonner.

OUTER JOIN

`OUTER JOIN` bygger modsat `INNER JOIN` ikke på, at der skal findes et match mellem de kolonner i de to tabeller, der baseret på prædikatet danner basis for kombinationen af rækker i den resulterende tabel. Med andre ord vil en række fra den ene tabel, der ikke umiddelbart finder match i en ellere flere rækker i den anden tabel alligevel indgå i resultatet og vice versa. Der kan anvendes samme typer prædikater som for `INNER JOIN`.

FULL OUTER JOIN

I `FULL OUTER JOIN` kan begge tabeller, der indgår i operationen bidrage med rækker, der ikke findes match til. Forspørgslen

```
SELECT   Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM     Tabel1 FULL OUTER JOIN Tabel2 ON Tabel1.Tal = Tabel2.Tal
```

resulterer således i følgende tabel:

Bogstav	Tal	Tal	Tegn
a	1	1	!
b	2	2	?
c	3	NULL	NULL
d	4	NULL	NULL
e	2	2	?
NULL	NULL	5	*

`LEFT OUTER JOIN`

I `LEFT OUTER JOIN` sorteres umatchede resultater fra, således at kun umatchede rækker i den tabel, der er nævnt før (eller til venstre for) `JOIN`-operationen, bibeholdes. Eksekveres følgende forespørgsel

```
SELECT    Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM      Tabel1 LEFT OUTER JOIN Tabel2 ON Tabel1.Tal = Tabel2.Tal
```

opnås nedenstående resultat:

Bogstav	Tal	Tal	Tegn
a	1	1	!
b	2	2	?
c	3	NULL	NULL
d	4	NULL	NULL
e	2	2	?

`RIGHT OUTER JOIN`

I `RIGHT OUTER JOIN` sorteres umatchede resultater fra, således at kun umatchede rækker i den tabel, der er nævnt efter (eller til højre for) `JOIN`-operationen bibeholdes. Eksekveres følgende forespørgsel

```
SELECT    Tabel1.Bogstav, Tabel1.Tal, Tabel2.Tal AS Tal_2, Tabel2.Tegn
FROM      Tabel1 RIGHT OUTER JOIN Tabel2 ON Tabel1.Tal = Tabel2.Tal
```

opnås nedenstående resultat:

Bogstav	Tal	Tal	Tegn
a	1	1	!
b	2	2	?
e	2	2	?
NULL	NULL	5	*

Bilag B

Testscenarier

Tekstlig beskrivelse

1. Hent ID0 og fødselsdato på alle personer i databasen
2. Hent ID0, testdato, køn og standardseriebit for alle testforløb i databasen
3. Hent ID0, testdato, køn og standardseriebit for alle testforløb foretaget i 2008
4. Hent ID0, fødselsdato og køn på alle personer i databasen. Ingen dubletter
5. Hent ID0, testdato, køn og eventuelle diagnosekoder for alle testforløb i databasen
6. Hent ID0, testdato, køn, håndeksembit og allergennummer, allergennavn, reaktioner for eventuelle positive reaktioner på samtlige stoffer for alle frisører i databasen
7. Hent ID0, fødselsdato, testdato, køn, eventuelle stillingskoder, alle reaktioner (også negativ), nuværende og tidligere relevans samt materialegruppekoder for samtlige eventuelle ekspositioner for alle standardserietestede personer i databasen, der er testet med et allergen med globalid = K7

SQL-kode

1. ```
   SELECT  ID0, FodselsDato
   FROM    dbo.Person
   ```

2. ```
   SELECT  ID0, UdfortDato, M, Standardserie
   FROM    dbo.TestForlob
   ```

3. ```
   SELECT  ID0, UdfortDato, M, Standardserie
   FROM    dbo.TestForlob
   WHERE   (YEAR(UdfortDato) = 2008)
   ```

4. ```
   SELECT DISTINCT Pe.ID0, Pe.FodselsDato, TF.M
   FROM   dbo.Person Pe INNER JOIN
          dbo.TestForlob TF ON Pe.ID0 = TF.ID0
   ```

5. ```
   SELECT  TF.ID0, TF.UdfortDato, TF.M, DS.Kode
   FROM    dbo.TestForlob TF LEFT OUTER JOIN
           dbo.DiagSpec DS ON TF.ID0 = DS.ID0 AND TF.ID1 = DS.ID1
   ```

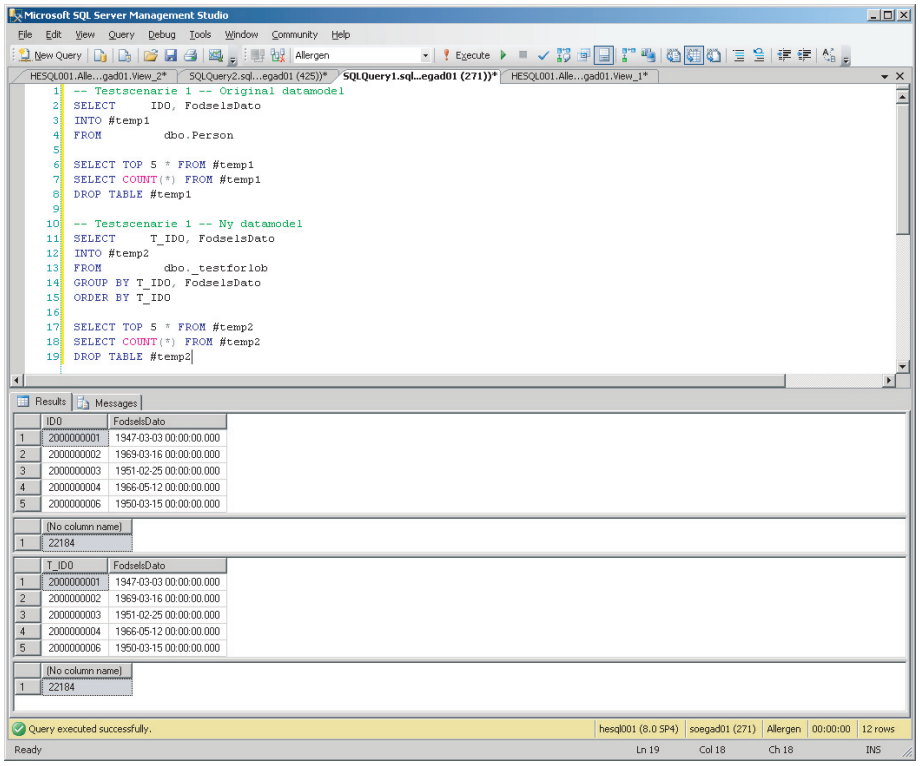
6. ```
   SELECT  TF.ID0, TF.UdfortDato, TF.M, Pl.AllergenNo, Al.Navn, Pl.D2, Pl.D3, Pl.D57
   FROM    dbo.TestForlob TF INNER JOIN
           dbo.Plaster Pl ON TF.ID0 = Pl.ID0 AND TF.ID1 = Pl.ID1 INNER JOIN
           dbo.Allergen Al ON Pl.AllergenNo = Al.AllergenNo INNER JOIN
           dbo.Stilling St ON TF.ID0 = St.ID0 AND TF.ID1 = St.ID1
   WHERE   (St.Kode = '51411') AND
               ((Pl.D2 >= '1') AND (Pl.D2 <= '3') OR
                (Pl.D3 >= '1') AND (Pl.D3 <= '3') OR
               (Pl.D57 >= '1') AND (Pl.D57 <= '3'))
   ```

7. ```
   SELECT  Pe.ID0, Pe.FodselsDato, TF.UdfortDato, St.Kode, Pl.D2, Pl.D3, Pl.D57,
           Pl.ARelevans, Pl.HRelevans, Ek.MatGruppe
   FROM    dbo.Person Pe INNER JOIN
           dbo.TestForlob TF ON Pe.ID0 = TF.ID0 INNER JOIN
           dbo.Plaster Pl ON Pe.ID0 = Pl.ID0 AND TF.ID1 = Pl.ID1 INNER JOIN
           dbo.Allergen Al ON Pl.AllergenNo = Al.AllergenNo LEFT OUTER JOIN
           dbo.Eksposition Ek ON Pe.ID0 = Ek.ID0 AND Pl.ID1 = Ek.ID1
                             AND Pl.ID2 = Ek.ID2 LEFT OUTER JOIN
           dbo.Stilling St ON Pe.ID0 = St.ID0 AND TF.ID1 = St.ID1
   WHERE   (TF.Standardserie = 1) AND (Al.GlobalID = 'K7')
   ```
   ```

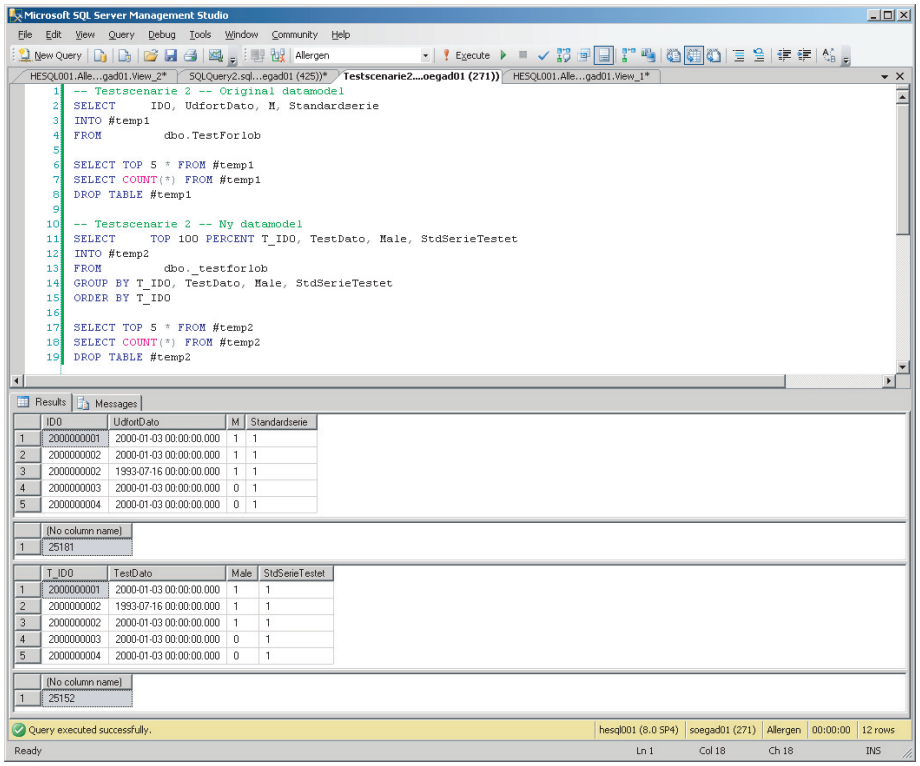
Bilag C

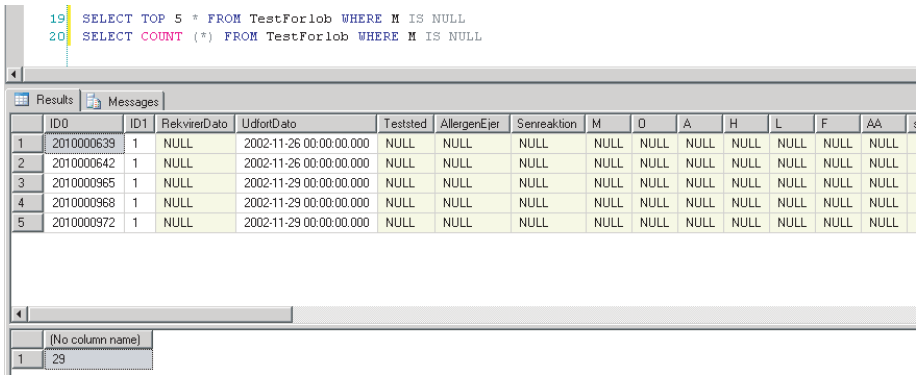
Validering

Testscenarie 1

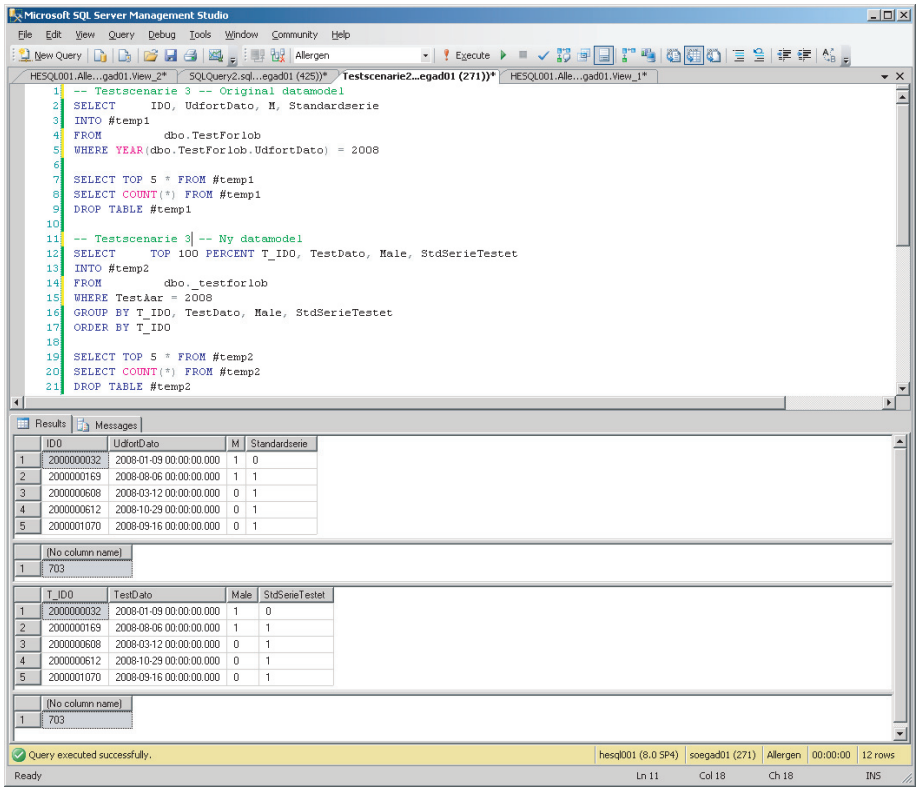


Testscenarie 2

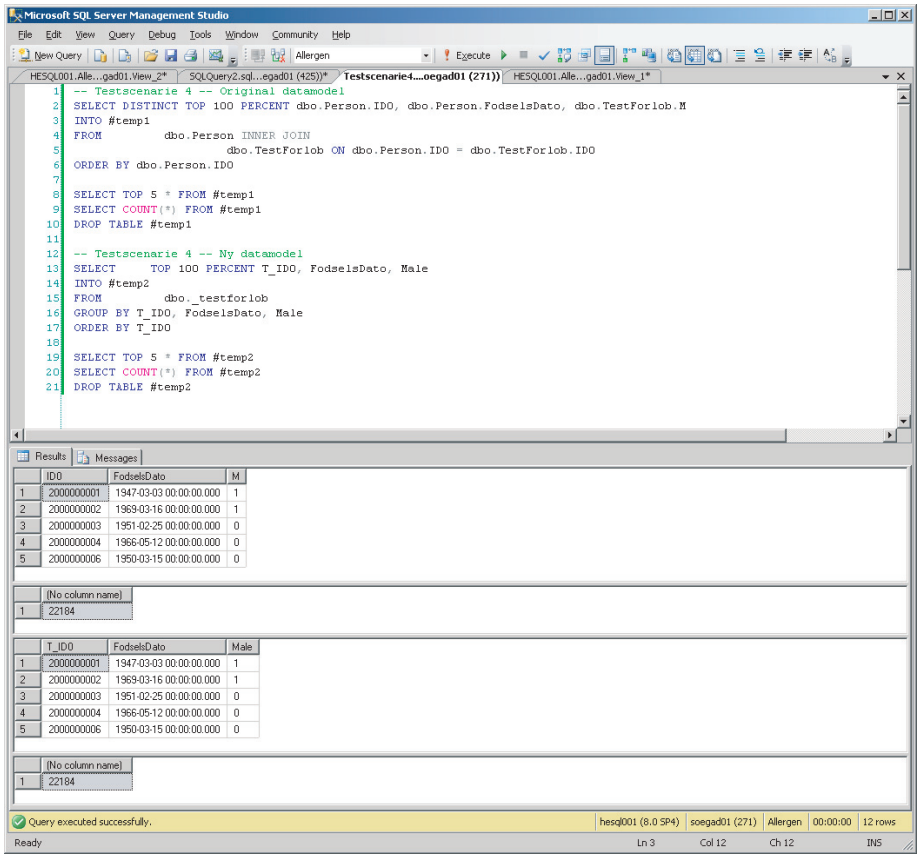




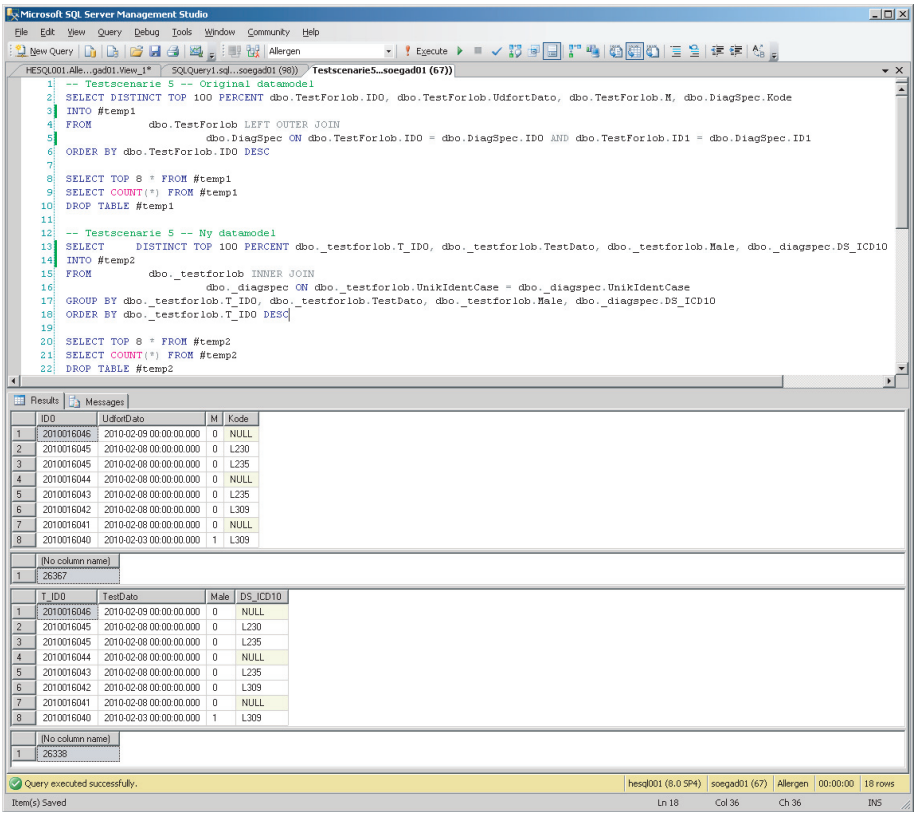
Testscenarie 3



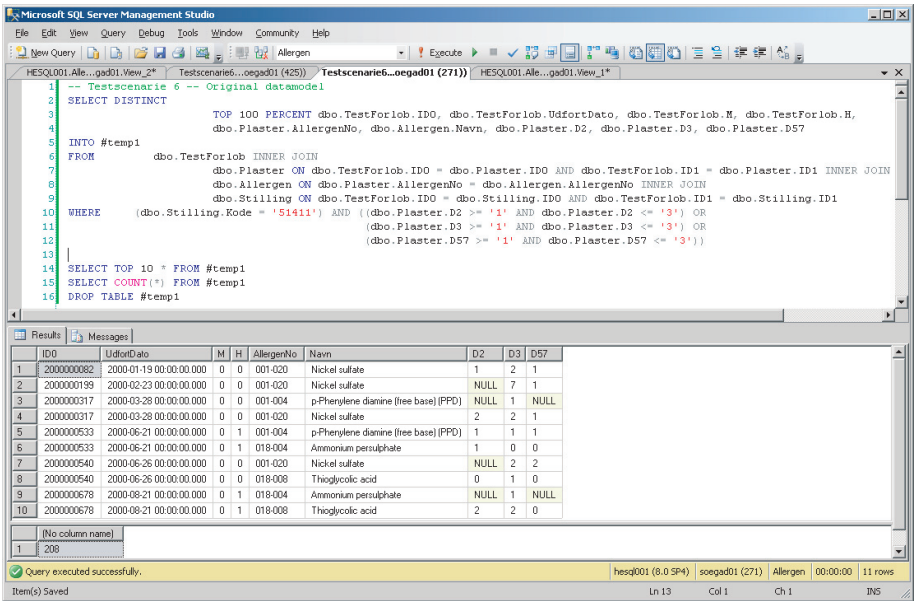
Testscenarie 4

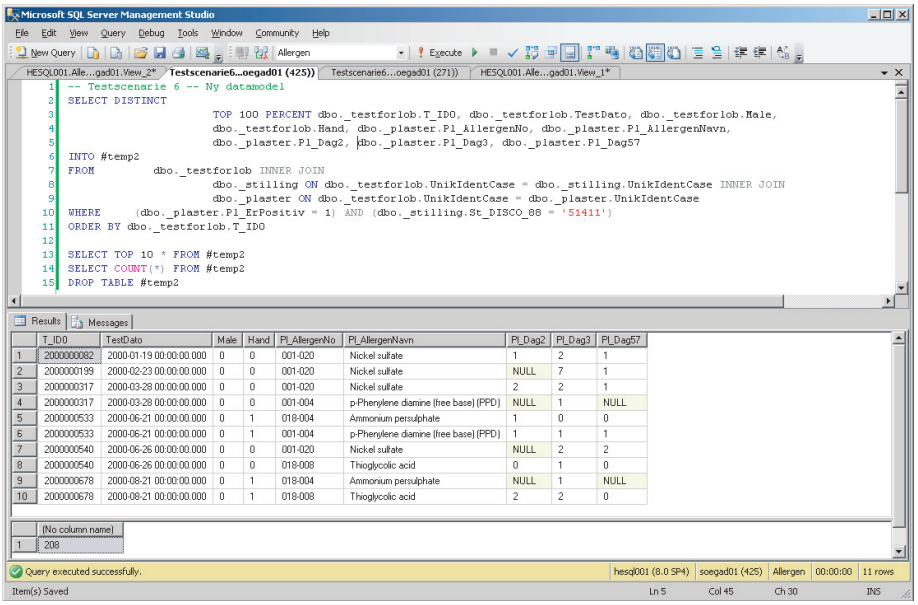


Testscenarie 5

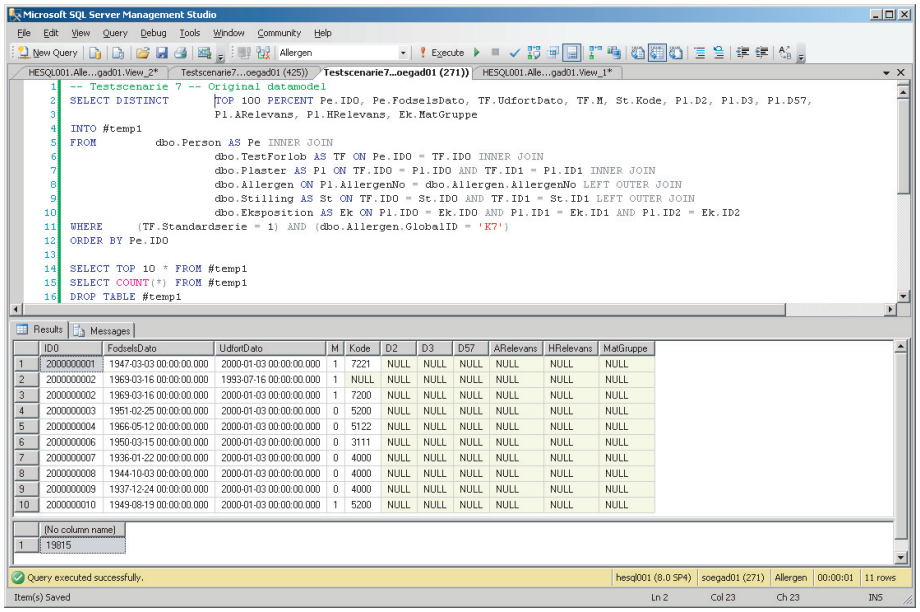


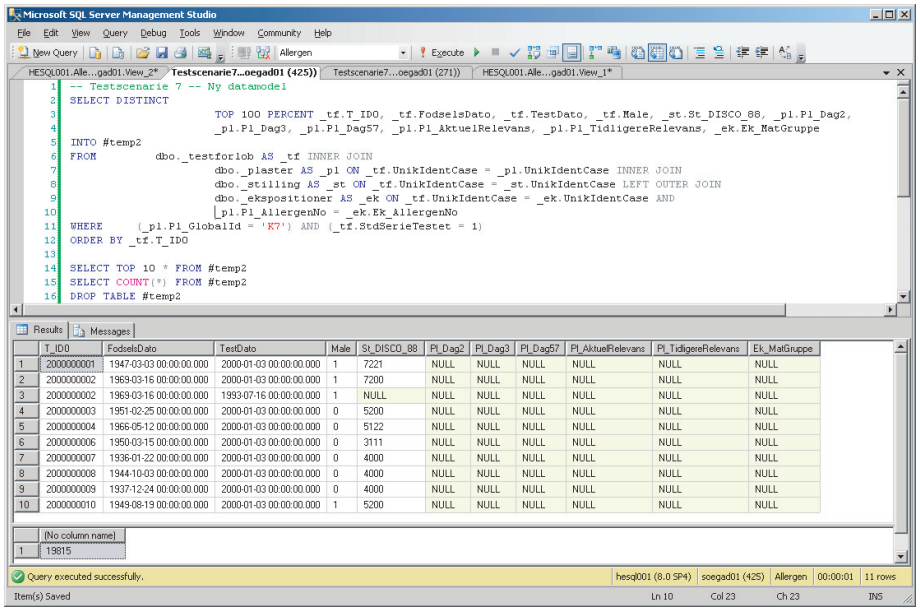
Testscenarie 6





Testscenarie 7





Bilag D

Kildekode

D.1 Fælles optællingsviews

```
CREATE VIEW [dbo].[CT_DiagSpec] AS
SELECT    CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)) AS UnikIdentCase,
          ID0 AS DS_ID0, ID1 AS DS_ID1, COUNT(DISTINCT Kode) AS AntalDiag
FRO       dbo.DiagSpec
GROUP BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)), ID0, ID1
ORDER BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10))


CREATE VIEW [dbo].[CT_Egnemat] AS
SELECT    CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)) AS UnikIdentCase,
          ID0 AS EM_ID0, ID1 AS EM_ID1, COUNT(ID2) AS AntalEgnemat
FROM      dbo.Egnematerialer
GROUP BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)), ID0, ID1
ORDER BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10))


CREATE VIEW [dbo].[CT_Ekspositioner] AS
SELECT    CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)) AS UnikIdentCase,
          ID0 AS Ek_ID0, ID1 AS Ek_ID1, COUNT(ID3) AS AntalEkspositioner
FROM      dbo.Eksposition
GROUP BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)), ID0, ID1
ORDER BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10))


CREATE VIEW [dbo].[CT_Plaster] AS
SELECT    CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)) AS UnikIdentCase,
          ID0 AS Pl_ID0, ID1 AS Pl_ID1, COUNT(ID2) AS AntalPlastre
FROM      dbo.Plaster
GROUP BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)), ID0, ID1
ORDER BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10))


CREATE VIEW [dbo].[CT_Plaster_Ekspo] AS
SELECT    dbo.Plaster.ID0, dbo.Plaster.ID1, dbo.Plaster.ID2,
          COUNT(*) AS AntalEkspositionPlaster
FROM      dbo.Plaster INNER JOIN
          dbo.Eksposition ON dbo.Plaster.ID0 = dbo.Eksposition.ID0 AND
          dbo.Plaster.ID1 = dbo.Eksposition.ID1 AND
          dbo.Plaster.ID2 = dbo.Eksposition.ID2
GROUP BY dbo.Plaster.ID0, dbo.Plaster.ID1, dbo.Plaster.ID2
```

```
CREATE VIEW [dbo].[CT_TestForlob] AS
SELECT   ID0, COUNT(ID1) AS AntalCasesPerson
FROM     dbo.TestForlob
GROUP BY ID0


CREATE VIEW [dbo].[CT_Latex_TestForlob] AS
SELECT   ID0, ID1, dbo.fnc_SOEGAD01_Latex(latexprik_udfoert, latexprik,
         latexrasthr_udfoert, latexrasthr) AS LatexTest
FROM     dbo.TestForlob


CREATE VIEW [dbo].[CT_Stilling] AS
SELECT   CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)) AS UnikIdentCase,
         ID0 AS St_ID0, ID1 AS St_ID1, COUNT(DISTINCT Kode) AS AntalStilling
FROM     dbo.Stilling
GROUP BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10)), ID0, ID1
ORDER BY CAST(ID0 AS VARCHAR(10)) + '/' + CAST(ID1 AS VARCHAR(10))
```

D.2 Første afprøvning

D.2.1 Viewet/tabellen _DB_testforlob

```
SELECT   CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
         TF.ID0 AS T_ID0, TF.ID1 AS T_ID1, TF.UdfortDato AS TestDato,
         YEAR(TF.UdfortDato) AS TestAar, Pe.FodselsDato, YEAR(Pe.FodselsDato) AS FodselsAar,
         dbo.fnc_calcAge(Pe.FodselsDato, TF.UdfortDato) AS Alder, TF.M AS Male, TF.O AS Occ,
         TF.A AS Atopic, TF.H AS Hand, TF.L AS Leg, TF.F AS Face, TF.AA AS AgeGT40,
         TF.Standardserie AS StdSerieTestet, CTT.AntalCasesPerson, CTL.LatexTest
INTO     dbo._DB_testforlob
FROM     dbo.TestForlob AS TF INNER JOIN
         dbo.Person AS Pe ON TF.ID0 = Pe.ID0 INNER JOIN
         dbo.CT_TestForlob AS CTT ON TF.ID0 = CTT.ID0 INNER JOIN
         dbo.CT_Latex_TestForlob AS CTL ON TF.ID0 = CTL.ID0 AND TF.ID1 = CTL.ID1
GROUP BY CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)), TF.ID0, TF.UdfortDato,
         YEAR(TF.UdfortDato), Pe.FodselsDato, YEAR(Pe.FodselsDato),
         dbo.fnc_calcAge(Pe.FodselsDato, TF.UdfortDato), TF.M, TF.O, TF.A, TF.H, TF.L, TF.F,
         TF.AA, TF.Standardserie, CTT.AntalCasesPerson, CTL.LatexTest, TF.ID1
```

D.2.2 Viewet/tabellen _DB_diagspec

```
SELECT   CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
         DS.ID0 AS DS_ID0, DS.ID1 AS DS_ID1, CTD.AntalDiag, DS.Kode
INTO     dbo._DB_diagspec
FROM     dbo.DiagSpec AS DS INNER JOIN
         dbo.CT_DiagSpec AS CTD ON DS.ID0 = CTD.DS_ID0 AND DS.ID1 = CTD.DS_ID1 RIGHT OUTER JOIN
         dbo.TestForlob AS TF ON DS.ID0 = TF.ID0 AND DS.ID1 = TF.ID1
ORDER BY CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10))
```

D.2.3 Viewet/tabellen _DB_plaster

```
SELECT   CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
         CAST(Pl.ID0 AS VARCHAR(10)) + '/' + CAST(Pl.ID1 AS VARCHAR(10)) + '/' +
         CAST(Pl.ID2 AS VARCHAR(10)) AS UnikIdentPlaster, Pl.ID0 AS Pl_ID0, Pl.ID1 AS Pl_ID1,
         Pl.ID2 AS Pl_ID2, SUBSTRING(Pl.AllergenNo, 1, 3) AS Pl_Serie,
         dbo.fnc_AllergenTypeSerie(Pl.AllergenNo, 1) AS Pl_StdSerieAlg,
         dbo.fnc_AllergenTypeSerie(Pl.AllergenNo,2) AS Pl_SuppStdSerieAlg,
         Al.GlobalID AS Pl_GlobalId, Pl.AllergenNo AS Pl_AllergenNo, Al.Navn AS Pl_AllergenNavn,
         Al.Koncentration AS Pl_Koncentration, Al.Enhed AS Pl_Enhed, Al.Vehikel AS Pl_Vehikel,
         Pl.D2 AS Pl_Dag2, Pl.D3 AS Pl_Dag3, Pl.D57 AS Pl_Dag57,
         dbo.IsPos(Pl.D2, Pl.D3, Pl.D57) AS Pl_ErPositiv,
         dbo.allergen_reaktionsresultat(Pl.D2, Pl.D3, Pl.D57) AS Pl_SamletReaktion,
         Pl.ARelevans AS Pl_AktuelRelevans, Pl.HRelevans AS Pl_TidligereRelevans
INTO     dbo._DB_plaster
FROM     dbo.Plaster AS Pl INNER JOIN
         dbo.Allergen AS Al ON Pl.AllergenNo = Al.AllergenNo RIGHT OUTER JOIN
         dbo.TestForlob AS TF ON Pl.ID0 = TF.ID0 AND Pl.ID1 = TF.ID1
ORDER BY CAST(Pl.ID0 AS VARCHAR(10)) + '/' + CAST(Pl.ID1 AS VARCHAR(10)) + '/' +
         CAST(Pl.ID2 AS VARCHAR(10))
```

D.3 Anden afprøvning

```
CREATE VIEW [dbo].[OLAPView_TestForlob]
AS
SELECT    CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
          TF.UdfortDato AS TestDato, Pe.FodselsDato,
          dbo.fnc_calcAge(Pe.FodselsDato, TF.UdfortDato) AS Alder, TF.M AS Male, TF.O AS Occ,
          TF.A AS Atopic, TF.H AS Hand, TF.L AS Leg, TF.F AS Face, TF.AA AS AgeGT40,
          TF.Standardserie AS StdSerieTestet, CT_TF.AntalCasesPerson, CT_L.LatexTest,
          CT_St.AntalStilling, CT_DS.AntalDiag, CT_Pl.AntalPlastre, CT_Ek.AntalEkspositioner,
          CT_EM.AntalEgnemat, 1 AS SumVar
FROM      dbo.TestForlob AS TF INNER JOIN
          dbo.Person AS Pe ON TF.ID0 = Pe.ID0 INNER JOIN
          dbo.CT_TestForlob AS CT_TF ON TF.ID0 = CT_TF.ID0 INNER JOIN
          dbo.CT_Latex_TestForlob AS CT_L ON TF.ID0 = CT_L.ID0 AND TF.ID1 = CT_L.ID1
          LEFT OUTER JOIN
          dbo.CT_Egnemat AS CT_EM ON TF.ID0 = CT_EM.EM_ID0 AND TF.ID1 = CT_EM.EM_ID1
          LEFT OUTER JOIN
          dbo.CT_Ekspositioner AS CT_Ek ON TF.ID0 = CT_Ek.Ek_ID0 AND TF.ID1 = CT_Ek.Ek_ID1
          LEFT OUTER JOIN
          dbo.CT_Plaster AS CT_Pl ON TF.ID0 = CT_Pl.Pl_ID0 AND TF.ID1 = CT_Pl.Pl_ID1
          LEFT OUTER JOIN
          dbo.CT_DiagSpec AS CT_DS ON TF.ID0 = CT_DS.DS_ID0 AND TF.ID1 = CT_DS.DS_ID1
          LEFT OUTER JOIN
          dbo.CT_Stilling AS CT_St ON TF.ID0 = CT_St.St_ID0 AND TF.ID1 = CT_St.St_ID1
GROUP BY  CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)),
          TF.UdfortDato, Pe.FodselsDato, dbo.fnc_calcAge(Pe.FodselsDato, TF.UdfortDato),
          TF.M, TF.O, TF.A, TF.H, TF.L, TF.F, TF.AA, TF.Standardserie,
          CT_TF.AntalCasesPerson, CT_L.LatexTest, CT_St.AntalStilling, CT_DS.AntalDiag,
          CT_Pl.AntalPlastre, CT_Ek.AntalEkspositioner, CT_EM.AntalEgnemat
```

D.4 Tredje test

D.4.1 Tabellen _testforlob

```
SELECT   CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
         TF.ID0 AS T_ID0, TF.ID1 AS T_ID1, TF.UdfortDato AS TestDato,
         YEAR(TF.UdfortDato) AS TestAar, Pe.FodselsDato, YEAR(Pe.FodselsDato) AS FodselsAar,
         dbo.fnc_calcAge(Pe.FodselsDato, TF.UdfortDato) AS Alder, CAST(TF.M AS INT) AS Male,
         CAST(TF.O AS INT) AS Occ, CAST(TF.A AS INT) AS Atopic, CAST(TF.H AS INT) AS Hand,
         CAST(TF.L AS INT) AS Leg, CAST(TF.F AS INT) AS Face, CAST(TF.AA AS INT) AS AgeGT40,
         CAST(TF.Standardserie AS INT) AS StdSerieTestet, CT_TF.AntalCasesPerson,
         CT_L.LatexTest, CT_St.AntalStilling, CT_DS.AntalDiag, CT_Pl.AntalPlastre,
         CT_Ek.AntalEkspositioner, CT_EM.AntalEgnemat
INTO     dbo._testforlob
FROM     dbo.TestForlob AS TF INNER JOIN
         dbo.Person AS Pe ON TF.ID0 = Pe.ID0 INNER JOIN
         dbo.CT_TestForlob AS CT_TF ON TF.ID0 = CT_TF.ID0 INNER JOIN
         dbo.CT_Latex_TestForlob AS CT_L ON TF.ID0 = CT_L.ID0 AND TF.ID1 = CT_L.ID1
         LEFT OUTER JOIN
         dbo.CT_Egnemat AS CT_EM ON TF.ID0 = CT_EM.EM_ID0 AND TF.ID1 = CT_EM.EM_ID1
         LEFT OUTER JOIN
         dbo.CT_Ekspositioner AS CT_Ek ON TF.ID0 = CT_Ek.Ek_ID0 AND TF.ID1 = CT_Ek.Ek_ID1
         LEFT OUTER JOIN
         dbo.CT_Plaster AS CT_Pl ON TF.ID0 = CT_Pl.Pl_ID0 AND TF.ID1 = CT_Pl.Pl_ID1
         LEFT OUTER JOIN
         dbo.CT_DiagSpec AS CT_DS ON TF.ID0 = CT_DS.DS_ID0 AND TF.ID1 = CT_DS.DS_ID1
         LEFT OUTER JOIN
         dbo.CT_Stilling AS CT_St ON TF.ID0 = CT_St.St_ID0 AND TF.ID1 = CT_St.St_ID1
GROUP BY CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)), TF.ID0,
         TF.UdfortDato, YEAR(TF.UdfortDato), Pe.FodselsDato, YEAR(Pe.FodselsDato),
         dbo.fnc_calcAge(Pe.FodselsDato, TF.UdfortDato), TF.M, TF.O, TF.A, TF.H, TF.L,
         TF.F, TF.AA, TF.Standardserie, CT_TF.AntalCasesPerson, CT_L.LatexTest,
         CT_St.AntalStilling, CT_DS.AntalDiag, CT_Pl.AntalPlastre, CT_Ek.AntalEkspositioner,
         TF.ID1, CT_EM.AntalEgnemat
```

D.4.2 Tabellen _egnematerialer

```
SELECT   CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
         EM.ID0 AS Em_ID0, EM.ID1 AS Em_ID1, EM.ID2 AS Em_ID2,
         CTE.AntalEgnemat AS Em_AntalEgnematerialer, EM.MatGruppe AS Em_MatGruppe,
         ValMg.GruppeNavn AS Em_MatGruppeNavn, EM.Materiale AS Em_Materiale,
         ValMa.MaterialeNavn AS Em_MaterialeNavn, EM.Producent AS Em_Producent,
         ValPc.ProducentNavn AS Em_ProducentNavn, EM.Produkt AS Em_Produkt,
         ValPk.ProduktNavn AS Em_ProduktNavn, dbo.IsPosEM(EM.MaxReaktion) AS Em_PositivReaktion,
         EM.MaxReaktion AS Em_Reaktion, ValD.Tekst AS Em_ReaktionTekst,
         EM.ArbejdsRel AS Em_ArbejdsRelateret, EM.PrivatRel AS Em_PrivatRelateret
INTO     dbo._egnematerialer
FROM     dbo.CT_Egnemat AS CTE RIGHT OUTER JOIN
         dbo.TestForlob AS TF ON CTE.EM_ID0 = TF.ID0 AND CTE.EM_ID1 = TF.ID1 LEFT OUTER JOIN
         dbo.ValEM_Materialer AS ValMa INNER JOIN
         dbo.Egnematerialer AS EM ON ValMa.Kode = EM.Materiale ON TF.ID1 = EM.ID1 AND
         TF.ID0 = EM.ID0 LEFT OUTER JOIN
         dbo.ValEM_Producent AS ValPc ON EM.Producent = ValPc.Kode LEFT OUTER JOIN
         dbo.ValEM_Produkter AS ValPk ON EM.Produkt = ValPk.Kode LEFT OUTER JOIN
         dbo.ValEM_Matgruppe AS ValMg ON EM.MatGruppe = ValMg.Kode LEFT OUTER JOIN
         dbo.ValDvalues AS ValD ON EM.MaxReaktion = ValD.Kode
ORDER BY CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10))
```

D.4.3 Tabellen _plaster

```
SELECT    CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
          CAST(Pl.ID0 AS VARCHAR(10)) + '/' + CAST(Pl.ID1 AS VARCHAR(10)) + '/' +
          CAST(Pl.ID2 AS VARCHAR(10)) AS UnikIdentPlaster, Pl.ID0 AS Pl_ID0, Pl.ID1 AS Pl_ID1,
          Pl.ID2 AS Pl_ID2, SUBSTRING(Pl.AllergenNo, 1, 3) AS Pl_Serie,
          dbo.fnc_AllergenTypeSerie(Pl.AllergenNo, 1) AS Pl_StdSerieAlg,
          dbo.fnc_AllergenTypeSerie(Pl.AllergenNo, 2) AS Pl_SuppStdSerieAlg,
          Al.GlobalID AS Pl_GlobalId, Pl.AllergenNo AS Pl_AllergenNo, Al.Navn AS Pl_AllergenNavn,
          Al.Koncentration AS Pl_Koncentration, Al.Enhed AS Pl_Enhed, Al.Vehikel AS Pl_Vehikel,
          Pl.D2 AS Pl_Dag2, Pl.D3 AS Pl_Dag3, Pl.D57 AS Pl_Dag57,
          dbo.IsPos(Pl.D2, Pl.D3, Pl.D57) AS Pl_ErPositiv,
          dbo.allergen_reaktionsresultat(Pl.D2, Pl.D3, Pl.D57) AS Pl_SamletReaktion,
          ValD.Tekst AS Pl_SamletReaktionTekst,
          dbo.IsPosRel(Pl.ARelevans) AS Pl_AktuelRelevansPositiv,
          Pl.ARelevans AS Pl_AktuelRelevans, ValR.Tekst AS Pl_AktuelRelevansTekst,
          dbo.IsPosRel(Pl.HRelevans) AS Pl_TidligerelRelevansPositiv,
          Pl.HRelevans AS Pl_TidligereRelevans,
          CT_PE.AntalEkspositionPlaster AS Pl_AntalEkspositionPlaster
INTO      dbo._plaster
FROM      dbo.Plaster AS Pl INNER JOIN
          dbo.Allergen AS Al ON Pl.AllergenNo = Al.AllergenNo LEFT OUTER JOIN
          dbo.CT_Plaster_Ekspo AS CT_PE ON Pl.ID0 = CT_PE.ID0 AND Pl.ID1 = CT_PE.ID1
          AND Pl.ID2 = CT_PE.ID2 RIGHT OUTER JOIN
          dbo.TestForlob AS TF ON Pl.ID0 = TF.ID0 AND Pl.ID1 = TF.ID1 LEFT OUTER JOIN
          dbo.ValDvalues AS ValD ON
          dbo.allergen_reaktionsresultat(Pl.D2, Pl.D3, Pl.D57) = ValD.Kode LEFT OUTER JOIN
          dbo.ValRelevans AS ValR ON Pl.ARelevans = ValR.Kode
GO

UPDATE dbo._plaster SET UnikIdentPlaster = UnikIdentCase + '/NULL' WHERE
       UnikIdentPlaster IS NULL
```

D.4.4 Tabellen _ekspositioner

```
SELECT    CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
          CAST(Pl.ID0 AS VARCHAR(10)) + '/' + CAST(Pl.ID1 AS VARCHAR(10)) + '/' +
          CAST(Pl.ID2 AS VARCHAR(10)) AS UnikIdentPlaster, EK.ID0 AS Ek_ID0, EK.ID1 AS Ek_ID1,
          EK.ID2 AS Ek_ID2, EK.ID3 AS Ek_ID3, CTE.AntalEkspositioner AS Ek_AntalEkspositioner,
          SUBSTRING(Pl.AllergenNo, 1, 8) AS Ek_AllergenNo, EK.MatGruppe AS Ek_MatGruppe,
          ValMG.GruppeNavn AS Ek_MatGruppeNavn, EK.Materiale AS Ek_Materiale,
          ValMa.MaterialeNavn AS Ek_MaterialeNavn, EK.Producent AS Ek_Producent,
          ValPc.ProducentNavn AS Ek_ProducentNavn, EK.Produkt AS Ek_Produkt,
          ValPk.ProduktNavn AS Ek_ProduktNavn, EK.ArbejdsRel AS Ek_ArbejdsRelateret,
          EK.PrivatRel AS Ek_PrivatRelateret
INTO      dbo._ekspositioner
FROM      dbo.CT_Eksposition AS CTE RIGHT OUTER JOIN
          dbo.TestForlob AS TF ON CTE.Ek_ID0 = TF.ID0 AND CTE.Ek_ID1 = TF.ID1 LEFT OUTER JOIN
          dbo.Eksposition AS EK INNER JOIN
          dbo.Plaster AS Pl ON EK.ID0 = Pl.ID0 AND EK.ID1 = Pl.ID1 AND EK.ID2 = Pl.ID2
          LEFT OUTER JOIN
          dbo.ValEK_Produkter AS ValPk ON EK.Produkt = ValPk.Kode LEFT OUTER JOIN
          dbo.ValEK_Producent AS ValPc ON EK.Producent = ValPc.Kode LEFT OUTER JOIN
          dbo.ValEK_Materialer AS ValMa ON EK.Materiale = ValMa.Kode LEFT OUTER JOIN
          dbo.ValEK_MatGruppe AS ValMG ON EK.MatGruppe = ValMG.Kode ON TF.ID0 = Pl.ID0 AND
          TF.ID1 = Pl.ID1
ORDER BY CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10))
GO

UPDATE dbo._ekspositioner SET UnikIdentPlaster = UnikIdentCase + '/NULL' WHERE
       UnikIdentPlaster IS NULL
```

D.4.5 Tabellen _stilling

```
SELECT   CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
         St.ID0 AS St_ID0, St.ID1 AS St_ID1, CTS.AntalStilling AS St_AntalStillinger,
         dbo.fnc_IsVaadArb(St.Kode) AS St_VaadArbejde, SUBSTRING(St.Kode, 1, 5) AS St_DISCO_88,
         ValS.Tekst AS St_DISCO_88Tekst
INTO     dbo._stilling
FROM     dbo.CT_Stilling AS CTS RIGHT OUTER JOIN
         dbo.TestForlob AS TF ON CTS.St_ID0 = TF.ID0 AND CTS.St_ID1 = TF.ID1 LEFT OUTER JOIN
         dbo.Stilling AS St INNER JOIN
         dbo.ValStilling AS ValS ON St.Kode = ValS.Kode ON TF.ID0 = St.ID0 AND TF.ID1 = St.ID1
```

D.4.6 Tabellen _diagspec

```
SELECT   CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10)) AS UnikIdentCase,
         DS.ID0 AS DS_ID0, DS.ID1 AS DS_ID1, CTD.AntalDiag AS DS_AntalDiag,
         DS.Kode AS DS_ICD10, SUBSTRING(DS.Kode, 1, 1) AS DS_ICD10Bogstav,
         SUBSTRING(DS.Kode, 1, 2) AS DS_ICD10Bogstav1tal,
         SUBSTRING(DS.Kode, 1, 3) AS DS_ICD10Bogstav2tal, ValD.Tekst AS DS_ICD10Tekst,
         ValD.DiagType AS DS_DiagType, ValDT.Navn AS DS_DiagTypeNavn
INTO     dbo._diagspec
FROM     dbo.ValDiagnose AS ValD INNER JOIN
         dbo.DiagSpec AS DS ON ValD.Kode = DS.Kode INNER JOIN
         dbo.CT_DiagSpec AS CTD ON DS.ID0 = CTD.DS_ID0 AND DS.ID1 = CTD.DS_ID1 INNER JOIN
         dbo.ValDiagType AS ValDT ON ValD.DiagType = ValDT.DiagType RIGHT OUTER JOIN
         dbo.TestForlob AS TF ON DS.ID0 = TF.ID0 AND DS.ID1 = TF.ID1
ORDER BY CAST(TF.ID0 AS VARCHAR(10)) + '/' + CAST(TF.ID1 AS VARCHAR(10))
```