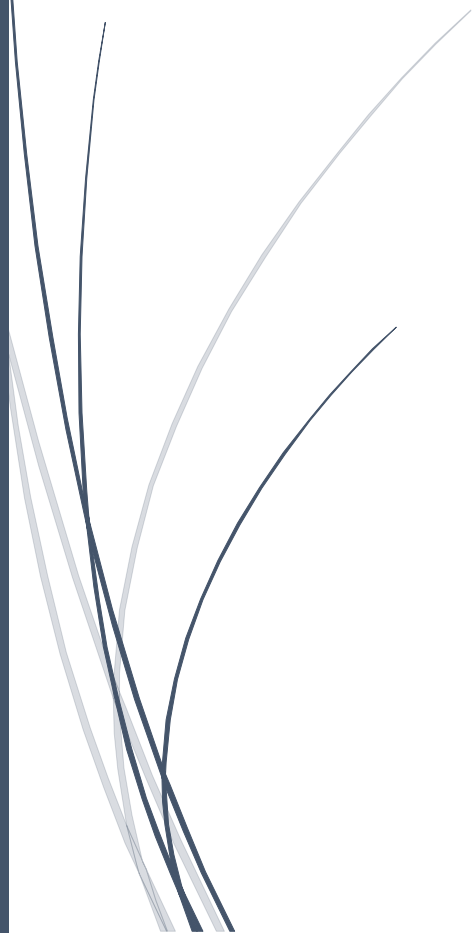
20-03-2016

Grundlæggende Database

Projektopgave



Lars Kramer Kristensen, larskk@gmail.com,
tlf. 2670 6010
CPHBUSINESS
AKADEMIUDDANNELSEN I INFORMATIONSTEKNOLOGI

Indholdsfortegnelse

Problemformulering

Jeg har valgt at lave en database relateret til mit arbejde, som er i Københavns Pakkecenter, Post Danmark. Det er kort fortalt en produktionsvirkssomhed hvor vi med hjælp fra et stort sorteringsanlæg registrerer og udsorterer al pakkepost til Sjælland og Bornholm, ca. 100.000 pakker pr. dag. Der er knap 300 ansatte, primært fordelt på et aften og et nathold med i alt ca. 15 produktionsteams med forskellige opgaver.

Jeg har overvejet en del forskellige muligheder, men er nået frem til at jeg vil lave en database der kan støtte mig i en rolle jeg udfylder som frivillig uddannelsesambassadør- og -vejleder. De fleste af mine kolleger har ligesom jeg mulighed for hvert år at tage på 10 dages selvvalgt uddannelse eller kurser med støtte fra det der hedder Industriens KompetenceUdviklingsFond (www.ikuf.dk). Det er i princippet en selv der skal ansøge, men processen er så indviklet og bureaukratisk, at de fleste sætter pris på at få nogen til at hjælpe sig igennem.

Jeg har valgt at kalde databasen for ”ikuf”, efter navnet på kompentencefonden.

Databasen skal først og fremmest hjælpe mig og evt. andre vejledere, samt ledelsen, med at få et overblik under selve ansøgningsprocessen og til kurset er afsluttet og der er udbetalt kompensation til den pågældende medarbejder. Desuden tænker jeg at den mere fremadrettet kan fungere som værktøj til at hente oplysninger om hvilke medarbejdere der har været på hvilke

kurser. Hvilket kan være interessant både for ledelsen, som gerne vil vide hvilke medarbejdere der har hvilke kompetencer, og også for kolleger som søger inspiration til at finde kurser som er relevante for dem.

Det er ikke en del af faget og opgaven her at udvikle et bruger-interface til databasen. Men et sådant kunne programmeres fx i C Sharp og Windows Forms.

Databasen indeholder en del personoplysninger, i form af CPR-numre, tlf.-numre osv., som jeg vil henstille til ikke videregives til uvedkommende.

Databaseteori

Jeg har valgt at lave databasen med Microsoft SQL Server 2014. Det er en softwarepakke som ud over selve databaseserveren og en række værktøjsprogrammer indeholder et såkalt DBMS: et Database Management System, eller Management Studio, som Microsoft kalder det, hvorfra man både kan designe, udvikle og interagere med en eller flere forskellige databaser. Den er dog ikke tænkt som en brugerapplikation, men udelukkende som et udviklingsværktøj. Slutbrugeren vil typisk arbejde med databasen via en hjemmeside eller desktop-applikation.

Mange af opgaverne med at udviklen databasen kan udføres vha. forskellige menuer og dialogbokse hvor man klikker sig frem med musen. Alternativt kan man, også via Management Studio, arbejde med SQL-kommandoer som skrives i et Query-vindue og eksekveres. SQL er et deklarativt programmeringssprog som man anvender til at arbejde med relationelle databaser. Det kan bruges både til at oprette og designe strukturen i databasen, til at arbejde med indholdet: CRUD (Create, Read, Update and Delete Data), og til at styre brugere og adgangsrettigheder. Disse forskellige dele af SQL-sproget kaldes henholdsvis DDL, DML og DCL. SQL er en international standard, som dog eksisterer i forskellige varianter og implementationer. Microsofts version kaldes Transact-SQL, eller T-SQL.

En database består først og fremmest af en række tabeller indbyrdes forbundne via nøgler, deraf navnet relationsdatabase. Desuden kan der være tilknyttet andre objekter: Views, Stored Procedures, Funktioner, Triggers osv. Som også gemmes i selve databasefilen (MDF).

Beskrivelse af databasen

Se venligst appendiks 1 med entitetsdiagram. For de fleste af tabellernes vedkommende siger det sig selv hvad de dækker over. Nogle enkelte kræver dog en lille forklaring.

”Pakke” er relevant for AMU-kurser, og er specielle kursusforløb hvor man tager en række kurser i eet samlet forløb, inden for det samme område, fx ”Grundlæggende Regnskab”, som har en samlet varighed på 30 dage, stykket sammen af 10-15 enkeltkurser. Det kan være en enkelt skole der har strikket en pakke sammen, eller det kan være mere standardiseret. Et AMU-kursus indgår ikke nødvendigvis i nogen pakke, og et kursus kan i princippet godt optræde både selvstændigt og som en del af en eller flere pakker.

”Kvikkode” er en kode som man kun bruge til hurtigt at fremsøge et specifikt kursushold på efteruddannelse.dk

”KursusType” er groft sagt hvilket niveau kurset er på, gående fra Grundskole til Diplomuddannelse, hvor AMU dog er lidt i en kategori for sig, idet det primært er ens faglige forudsætninger som afgør om man kan deltage et givent kursus. For yderligere information om kurser og og om IKUF-ordningen generelt, se venligst appendiks 2, som er en A4-info jeg benytter til kollegerne på min arbejdsplads.

Jeg vil i det følgende starte med at beskrive de forskellige normalformer: hvad de er, og hvordan jeg har fulgt dem med min database. Dernæst vil jeg gå mere i detaljer med nogle af mine tabeller og forklare om valg datatyper m.m.

Normalisering

I databasedesign benytter man begrebet ”normalisering” eller normalformer som udtryk for forskellige krav en database bør opfylde før den kan betragtes som godt designet. I faget har vi arbejdet med tre normalformer, samt den såkaldte Boyce-Codd normalform, der i virkeligheden er en slags udvidet 3. normalform. Det primære mål med normaliseringen er at undgå at samme data optræder flere steder, fordi det øger risikoen for fejl i forbindelse med indtastning, og gør det svært at holde databasen opdateret hvis man hver gang skal rette samme oplysning mange steder.

En relationsdatabase består af forbundne tabeller, og i en helt unormaliseret database vil alle data ligge i den samme tabel, og jo mere man normaliserer, jo flere tabeller får man. I særlige tilfælde kan man også vælge at gå den anden vej og lave det der kaldes denormalisering, for at gøre databasen hurtigere.

I praksis er det sjældent sådan at man starter med en helt unormaliseret database, og så følger de forskellige trin slavisk. Det har heller ikke været tilfældet med den foreliggende database. Jeg er startet med på papir at lave et design som jeg fandt hensigtsmæssigt, hvorefter jeg har oprettet tabeller og kolonner i SQL Server, derpå indsat data, og i den proces har jeg løbende justeret på designet og tilføjet tabeller efter behov. I praksis ville der også forekomme dialog med en kunde og en testperiode, hvorunder der også kunne opstå behov for at ændre på strukturen af tabellerne.

Da min database under denne proces har nået en vis størrelse (13 tabeller), vurderer jeg at det ville blive for omfattende at vise en tænkt normaliseringsproces fuldt ud med eksempeldata. I stedet vil jeg begrænse mig til at forklare hvad jeg gør i de forskellige normalformer, ledsaget af lidt generelle kommentarer, og i slutningen af hvert afsnit opliste de resulterende tabeller, men kun med kolonne-overskrifterne, uden data, som således kun vil optræde i den færdige, normaliserede database.

I helt **unormaliseret** form ville databasen bestå af én tabel med følgende kolonner:

Medarbejdernavn, CPR, SAP, Email, mobil, Land, Landetlf.kode, Team, Leder
Kursusnavn, Fagkode, Pakkenavn, Kursustype, Kvikkode, Uddannelse, Skole, Startdato, Slutdato, Fraværstimer, Varighed

Alene pga. de mange kolonner ville det altså være ret besværligt at arbejde med de forskellige data uden at have en rigtig database.

1. Normalform (1. NF)

Kravet til 1. NF er at hver tabel har en unik nøgle (enkel eller kombineret), at hvert felt kun indeholder én oplysning (er atomic), og at der ikke forekommer gentagende kolonner (fx kursus 1, kursus 2 osv.). Da den samme medarbejder godt kan tænkes at deltage i flere forskellige kurser, ville det være fornuftigt til at starte med at splitte data ud i en medarbejder- og en kursustabel. Man kunne bruge CPR-nr. eller SAP-nr. som primærnøgle for medarbejdertabellen, da disse begge er unikke. Det er dog at foretrække med et automatisk genereret ID-nummer af databasen, da man aldrig kan være helt sikker på om eksterne data med tiden skiftes eller får et andet format. Der er også Persondataloven at tage hensyn til.

Kursustabellen ville tilsvarende skulle have et kursus-id. Da flere medarbejdere godt kan tage det samme kursus, vil det være nødvendigt med en kombineret primærnøgle, bestående af et automatisk genereret kursusID samt et medarbejderID (fremmednøgle som refererer til medarbejdertabellen). Denne primærnøgle tager dog ikke højde for den mulighed at en medarbejder vælger at tage det samme kursus mere end én gang (måske fordi vedkommende dumper prøven). Derfor ville det være nødvendigt også at inkludere fx startdatoen i primærnøglen, så vi får: MedarbejderID, KursusID og StartDato.

Ved at følge ovenstående ender vi med følgende to tabeller:

Medarbejder:
MedarbejderID (PK), Medarbejdernavn, CPR, SAP, Email, mobil, Land, Landetlf.kode, Team, Leder

Kursus:
MedarbejderID (FK, PK), KursusID (PK), Kursusnavn, Fagkode, Pakkenavn, Kursustype, Kvikkode, Uddannelse, Skole, Startdato (PK), Slutdato, Fraværstimer, Varighed

2. Normalform

Kravet til 2. NF er at for tabeller med en kombineret primærnøgle må der ikke være nogen kolonner som kun afhænger af en del af primærnøglen. Derfor vil det være nødvendigt at splitte kursustabellen yderligere op. I virkeligheden er der mellem kursus- og medarbejdertabellen tale om en mange-til-mange relation: den samme medarbejder kan deltage på flere kurser, og der kan være flere medarbjedere på samme kursus. For at kunne opfylde nævnte krav til 2. NF er vi nødt til at oprette en såkaldt Junction tabel, også kaldet en bridge tabel idet den bygger bro mellem to tabeller. I dette tilfælde vil den indeholde MedarbejderID og KursusID, og desuden oplysningen Fraværstimer, der netop af afhænger af den kombinerede primærnøgle og ikke af kurset alene (forskellige medarbejdere kan til samme kursus søge om et forskelligt antal fraværstimer, af mange årsager). De to kolonner der udgør primærnøglen, vil på samme tid også være fremmednøgler (FK) til hver deres tabel.

Ved at følge ovenstående vil Kursus-tabellen blive opslittet i følgende, idet Medarbejder-tabellen forbliver uændret indtil videre. Bemærk at den kombinerede primærnøgle i Kursus erstattes med en enkelt primærnøgle:

Kursus:
KursusID (PK), Kursusnavn, Fagkode, Pakkenavn, Kursustype, Kvikkode, Uddannelse, Skole, Startdato, Slutdato, Varighed

KursusMedarbejder:
KursusID (PK, FK), MedarbejderID (PK, FK), Fraværstimer

3. Normalform

Kravet for at bringe en database i 3. NF er at ingen af de normale kolonner må afhænge af en anden kolonne end den/de der udgør primærnøglen. I Medarbejder-tabellen har vi Leder, som afhænger af Team, og Tlf.Landekode, som afhænger af Land. I Kursustabellen har vi nogle oplysninger der beskriver kurset mere generelt: KursusNavn, KursusType, Fagkode, Uddannelse og Varighed. Og nogle andre oplysninger der beskriver den tidsmæssige instans af et givent kursus: Datoer, Skole og kvikkode. For at kunne skelne mellem de to, vælger jeg at kalde sidstnævnte ”Hold”, selvom man jo i daglig tale bruger ordet kursus om begge dele. I nogle tilfælde bruges også ordet fag eller modul, men lad os ikke gøre det mere indviklet end højst nødvendigt… Så er der oplysningen Pakkenavn, der er relevant for nogle AMU-kurser. Jeg vil tale mere om denne attribut senere. Pt. vælger jeg at placere den i Hold-tabellen.

Med ovenstående får vi følgende 6 tabeller:

Team:
TeamID (PK), TeamNavn, LederNavn

Land:
LandeID (PK), LandeNavn, Tlf.LandeKode

Medarbejder:
MedarbejderID (PK), Medarbejdernavn, CPR, SAP, Email, mobil, LandeID (FK), TeamID (FK)

HoldMedarbejder:
HoldID (PK, FK), MedarbejderID (PK, FK), Fraværstimer

Hold:
HoldID (PK), Pakkenavn, KursusID (FK), Kvikkode, Skole, Startdato, Slutdato

Kursus:
KursusID (PK), Kursusnavn, Fagkode, Kursustype, Uddannelse, Varighed

Boyce-Codd normalform

For at opfylde BC NF skal vi udskille de entiteter/attributter som skrives med et navn og lægge dem i selvstændige tabeller som vi refererer til med et ID. Disse tabeller kaldes look-up eller reference-tabeller. Det giver en række fordele. Først og fremmest sikrer vi os mod at brugeren kan komme til at skrive et navn forkert, fx navnet på en skole i dette tilfælde. I bruger-applikationen kan man i stedet for et tekstfelt uden validering have fx en drop-down-liste med mulige skoler. Er der brug for at rette et navn på en bestemt skole, kan man nøjes med at gøre det i look-up tabellen, i stedet for alle de poster hvor værdien forekommer. Og slutteligt er det hurtigere for en database at søge og sortere i integer-værdier end varchar-værdier. Dette giver umiddelbart anledning til yderligere 5 tabeller: Leder, Pakke, Skole, KursusType og Uddannelse.

Herudover har jeg valgt at oprette yderligere to Junction-tabeller:

SkoleKursus skal sikre at kun de skoler man har tilknyttet det valgte kursusID, umiddelbart vil kunne vælges når der oprettes en ny post i tabellen Hold. Af samme grund ville jeg gerne have lavet en kombineret fremmednøgle:

```
FOREIGN KEY (SkoleID, KursusID) REFERENCES SkoleKursus (SkoleID, KursusID)
```

Imidlertid har jeg udkommenteret den, da KursusID jo også kan være NULL, og i så fald fungerer det ikke. Det er muligvis selve konceptet med at man enten skal angive kursusID eller PakkeID, der skal laves på en

anderledes måde i tabeldesignet. Et andet alternativ kunne være at binde KursusTypeID op på skole, i stedet for alle de enkelte kurser, eller en kombination af de to.

PakkeKursus er tænkt som en tabel hvor man kan indtaste hvilke kurser der indgår/kan indgå i en given pakke. Et kursus kan som tidligere nævnt indgå i 0, 1 eller flere pakker, og hvis det indgår i en pakke, kan det stadig tages selvstændigt. Jeg har valgt at benytte PakkeID som fremmednøgle i Hold-tabellen på den måde at hvis PakkeID har en værdi, bør KursusId være NULL, og omvendt. Alternativt kunne man måske have valgt at droppe PakkeID som kolonne og i stedet have en kolonne af Boolsk datatype (bit i T-SQL) for at angive om det pågældende kursushold indgår i en pakke eller ej. Det ville så betyde at der skulle oprettes et nyt "hold" for hvert kursus i pakken, hvilket ville blive lidt besværligt fordi der i nogle pakker indgår rigtig mange små kurser.

Tabeller

Jeg vil her fortælle lidt mere om de datatyper, constraints m.m. jeg har valgt.

Til alle ikke-tal-felter har jeg konsekvent valgt NVARCHAR og NCHAR, da disse datatyper tillader brug af Unicode tegn, og dermed skulle være mere fremtidssikrede, både i forhold til mængden af tilladte tegn, og fordi Unicode som standard nok efterhånden tager mere og mere over. Det kan dog ved større databaser have betydning for performance om man vælger den ene eller anden type. Hvor stor betydning, er der delte meninger om. Jeg har valgt NCHAR til LandeID og CPR da der er tale om at de pågældende attributter altid har et fast antal tegn. Man kunne yderligere lave en CHECK constraint på at de indeholder det korrekte antal tegn. For CPR's vedkommende kunne jeg også have valgt en INT datatype og så via en TRIGGER, eller i brugerapplikationen, sikre at en evt. indtastet bindestreg fjernes, og i visningen (i SELECT sætningen) kunne man ved hjælp af LEFT og RIGHT funktionen udtrække de to dele af nummeret og lægge en bindestreg imellem for at øge læsbarheden.

Til talfelter har jeg valgt INT (heltal) i alle tilfælde pånær nogle få, hvor jeg er absolut sikker på at SMALLINT er mere end rigeligt.

På CPR- og SAP-kolonnerne i tabellen Medarbejder har jeg valgt at sætte den CONSTRAINT at de skal være UNIQUE, så man ikke risikerer at indtaste den samme medarbejder flere gange, eller kommer til at give en medarbejder et nummer der tilhører en anden medarbejder (ved fejlindtastning). Det samme har jeg gjort for LandeNavn og LandeTlf.kode. I medarbejder-tabellen, kunne man også overveje en CHECK-constraint på at email-adressen indeholder '@' m.m., men da mulighederne med de indbyggede SQL Wildcards er noget begrænsede, ville det nok være bedre at lave denne validering i interface-applikationen.

I Hold-tabellen har jeg lavet en CHECK constraint på at slutdato skal være større end (=senere) eller lig med startdato.:

```
61 CREATE TABLE Hold (
62 HoldID INT PRIMARY KEY IDENTITY (1, 1),
63 Startdato DATE NOT NULL,
64 Slutdato DATE NOT NULL,
65 SkoleID INT FOREIGN KEY REFERENCES Skole (SkoleID) NOT NULL,
66 Kvikkode NVARCHAR(25), -- henviser til beskrivelse på www.efteruddannelse.dk
67 KursusID INT FOREIGN KEY REFERENCES Kursus (KursusID),
68 PakkeID INT FOREIGN KEY REFERENCES Pakke (PakkeID),
69 --FOREIGN KEY (SkoleID, KursusID) REFERENCES SkoleKursus (SkoleID, KursusID),
70 CONSTRAINT CHK_HOLD_DATOER CHECK (Slutdato >= Startdato),
71 CONSTRAINT CHK_HOLD_PAKKE CHECK ((PakkeID IS NULL OR KursusID IS NULL) AND NOT (PakkeID IS NULL AND KursusID IS NULL))
72 )
```

I samme tabel har jeg også lavet en CHECK på at enten KursusID eller PakkeId skal være udfyldt, men ikke dem begge.

Jeg har valgt at oprette index på både MedarbejderNavn og CPR, da jeg forestiller mig at det vil være dem der hyppigst søges på. Man kan via værktøjet "Query analyzer", som er en del SQL Servers prof-pakke, undersøge hvilke kolonner der ofte søges på, og som det kan betale sig af oprette index på.

Ekstra

Views

Et view er en Select statement som er gemt som et objekt I databasen, og som kan kaldes på samme måde som en tabel, så man slipper for at skrive hele den bagvedliggende kode hver gang. Man kan endda kalde et view fra et view eller fra en Stored Procedure, og på den måde også skabe en mere overskuelig, modulær kode.

Jeg har oprettet I alt 4 views, og gennemgår her det mest komplekse: MKView, der viser oversigt over hvilke medarbejdere der har været (eller skal) på hvilke kurser. I Column-listen bruges der en CASE-expression og correlated subquery, og i FROM-delen benytter jeg både INNER JOIN og OUTER LEFT JOIN til at kæde tabellerne sammen.

```
228  -- Medarbejdere + kurser
229  CREATE VIEW MKView AS
230  SELECT
231  Medarbejder.MedarbejderID,
232  MedarbejderNavn,
233  CASE -- Searched CASE format
234      WHEN Hold.KursusID IS NOT NULL
235      THEN KursusNavn
236      WHEN Hold.PakkeID IS NOT NULL
237      THEN (SELECT --Correlated subquery
238      PakkeNavn
239      FROM Pakke
240      WHERE Hold.PakkeID = Pakke.PakkeID)
241      ELSE 'Intet kursus eller pakke'
242  END AS 'Navn på kursus eller pakke',
243  SkoleNavn,
244  ISNULL(UddannelsesNavn, '') AS Uddannelse,
245  Startdato,
246  Slutdato,
247  Fraværstimer
248  FROM Medarbejder
249  INNER JOIN HoldMedarbejder
250  ON Medarbejder.MedarbejderID = HoldMedarbejder.MedarbejderID
251  INNER JOIN Hold
252  ON HoldMedarbejder.HoldID = Hold.HoldID
253  LEFT JOIN Kursus
254  ON Hold.KursusID = Kursus.KursusID
255  INNER JOIN Skole
256  ON Hold.SkoleID = Skole.SkoleID
257  LEFT JOIN Uddannelse
258  ON Kursus.UddannelsesID = Uddannelse.UddannelsesID
```

Jeg benytter en CASE-struktur til at afgøre om der er tale om et enkeltstående kursus eller en kursuspakke, og afhængigt af tilfældet viser jeg enten kursusnavn eller pakkenavn. I førstnævnte tilfælde bruger jeg en OUTER LEFT JOIN til Kursus-tabellen for at sikre mig at også de poster fra Hold-tabellen som ikke har et kursus-id, kommer med, og i andet tilfælde bruger jeg en correlated subquery i forbindelse med THEN-delen for at forbinde til Pakke-tabellen. Den kunne også være lavet som LEFT JOIN. Jeg bruger ISNULL-funktionen til at vise et blankt felt, i stedet for værdien NULL, når et kursus ikke er tilknyttet nogen uddannelse.

Udbygningsmuligheder – ekstra tabeller

Medarbejderens lønforhold: Det kunne lette ansøgningsprocessen en smule hvis der var nem adgang til lønoplysninger. Noget af lønnen er tilskudsberettiget, og noget ikke, så man skal sidde og tælle sammen ved hver enkelt ansøgning, da en løn er stykket af flere forskellige dele, som i størrelse og antal varierer fra medarbejder til medarbejder.

Bilag: Der skal udfyldes og indsendes en del forskellige skemaer til forskellige steder. Pt. scanner jeg dem ind og har liggende som pdf-filer i forskellige mapper navngivet efter personnavn+kursusnavn, fordelt i to hovedmapper: en med afsluttede kurser og en med kommende/igangværende. Det er måske også den bedste løsning, fordi når først bilagene er indsendt, kurset overstået og alle beløb udbetalt til kursisten, er der egentlig ikke længere behov for at gemme på bilagene. Hvis der var behov for det, kunne man integrere dem i databasen. Der kunne måske også udvikles systemer som interagerer med forskellige skema-skabeloner og automatisk populerer dem med de oplysninger der ligger i databasen.

Fraværsperioder: Der kunne laves en tabel over fraværsperioder/-dage i tilknytning til eller som en del af tabellen HoldMedarbejder, da det kan være forskelligt på det samme hold hvor meget fravær medarbejderne tager.

Det kunne være interessant hvis man kunne koble min database sammen med min virksomheds database, således at den ene kunne drage fordel af den andens oplysninger, fx om fraværsperioder og løn

Kurser som virksomheden anbefaler til bestemte jobfunktioner og –roller.

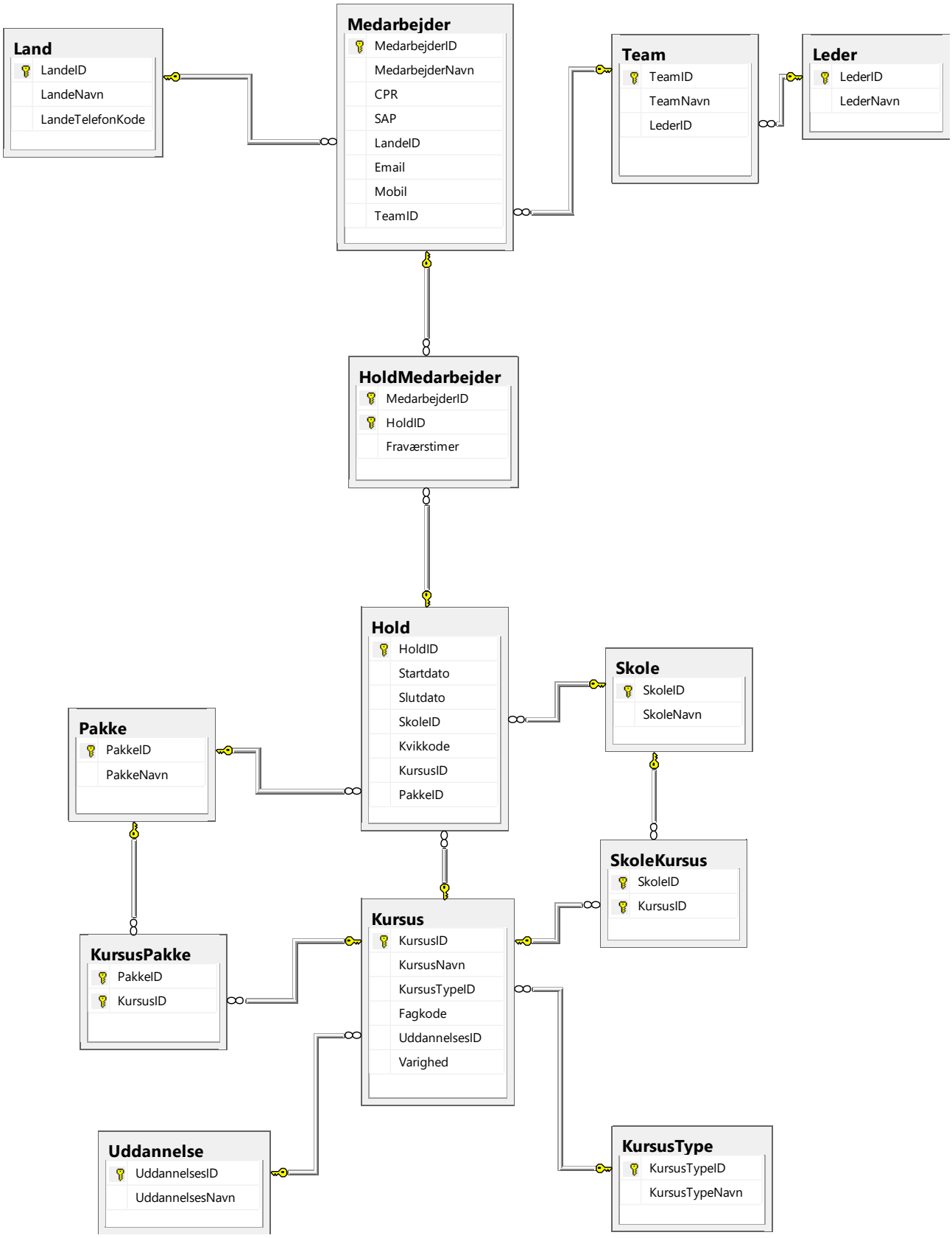
Oplysninger om den enkelte medarbejder: udd. baggrund, interesser: Det kunne evt. blot være i form af et kommentarfelt.

Konklusion

Det er lykkedes at lave en database som umiddelbart afspejler de behov der blev nævnt i problemformuleringen, og som fungerer. Men sålænge den ikke har et bruger-interface, er det dog temmelig besværligt at indtaste data pga. de mange referencetabeller. Som nævnt i afsnittet om normalisering er det vigtigt at have for øje hvad man vil bruge de registrerede data til, og om den tid man bruger på at indtaste data, senere tjener sig ind igen i form af nye muligheder.

Det har været en stor udfordring at designe de forskellige tabeller i databasen. Der er mange forbindelser på kryds og tværs, og det har følgelig givet rig mulighed for at implementere og arbejde med de forskellige teorier og funktioner vi har lært i faget, selvom jeg ikke har været omkring dem alle. Det har været meget lærerigt både at følge undervisningen og arbejde med denne rapport, og de andre opgaver der har været.

Appendiks 1: ER-diagram



Appendiks 2: Ikuf-opslag

Lær noget nyt!

Tag på kursus 10 dage hvert år

Alle ansatte i BRC Produktion kan hvert år tage på kursus i 10 dage med tilskud fra IKUF (**I**ndustriens **K**ompetence**U**dviklings**F**ond). Du kan også spare op fra tidligere år, og på den måde nå helt op på 30 dage inden for samme år.

- Du kan vælge at tage kurser hvor du lærer noget du kan bruge her på BRC.
- Du kan også vælge kurser som forbedrer dine chancer for at finde drømmejobbet.
- Og endelig kan du jo bare tage et kursus om et emne du synes er spændende ☺

Dit kursus skal være på IKUF's liste over godkendte kurser. Se mere på www.ikuf.dk. Dit fravær skal også altid godkendes af din leder, så vi sikrer at det ikke går ud over produktionen eller dine kollegers mulighed for at søge frihed.

Der findes 4 typer kurser du kan deltage i:

1. AMU (ArbejdsMarkedsUddannelse). Tusindvis af kurser. Se mere på www.amukurs.dk og www.efteruddannelse.dk og på de enkelte skolers hjemmesider, fx www.kts.dk, www.brock.dk og www.tec.dk for København.

2. Grundskoleniveau: Danskuddannelse, Regning, ordblindeundervisning.

3. Enkeltfag på Studenterkursus eller HF, fx sprog, naturvidenskab, IT, økonomi osv. Se alle fag på uddannelsesguiden: www.ug.dk

4. Enkeltfag på akademi- og diplomuddannelser: Læs om de forskellige uddannelser og fag på uddannelsesguiden, www.cphbusiness.dk og www.bygovenpaa.dk

---

Ønsker du hjælp til at finde det rigtige kursus, eller få lavet ansøgningen? Du er velkommen til at kontakte Lars Kramer fra Team LL2 (tlf. 2670 6010), samt produktionsleder Tiffanie Sjøting. Du kan også hente hjælp hos din fagforening eller tillidsmand.