Databaser

ER-diagrammer, databaser og lidt SQL

Malene Cramer Engebjerg

Version 1

Indhold

ER-diagrammer

Verden er fyldt med data. Tænk bare på alle de data som f.eks. gemmes af Facebook, Instagram, Lectio, din yndlingstøjbutik og offentlige myndigheder. Har du nogensinde tænkt over, hvordan alle disse data gemmes? Mon Facebook gemmer alle data i et stort Excel-ark? Svaret er nok "næppe"! I stedet vil data som oftest være gemt i det, man kalder for en ***database***. En database består typisk af flere tabeller, som på en eller anden måde hænger sammen. Man tilgår databasen via et ***databasesystem***, hvor I allerede har stiftet bekendtskab med SQLite.

I denne note vil vi se på, hvordan man designer en database, så man får gemt de oplysninger, man har brug for på den mest hensigtsmæssige måde.

Vi vil tage udgangspunkt i et eksempel, hvor vi vil forestille os, at gymnasiet har et festudvalg, som gerne vil holde styr på deres medlemmer, de forskellige fester og hvilke elever, der deltager i festerne. Festudvalget består både af elever og ansatte på skolen, og der er altid ét medlem fra festudvalget, som har hovedansvaret for en given fest. Hvem, det præcis er, går på skift.

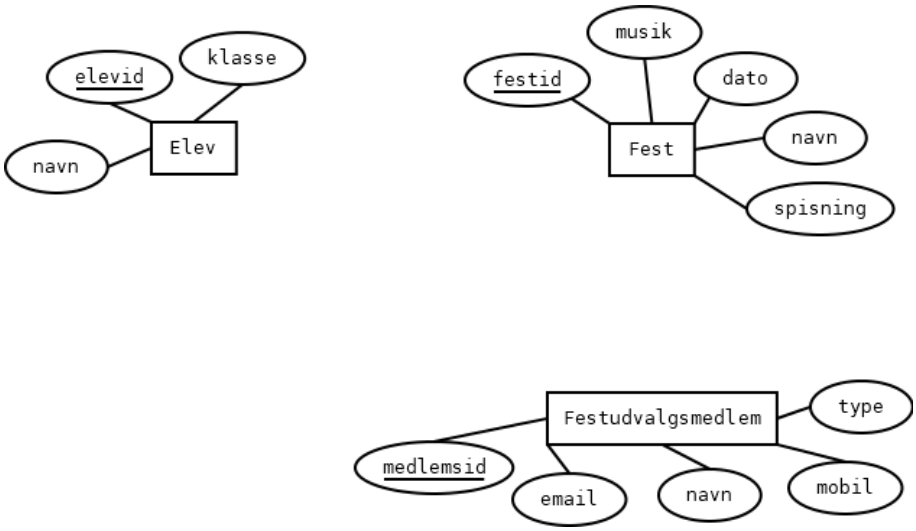
For at designe en database, som kan holde styr på gymnasiets fester, vil vi starte med at lave et ***ER-diagram***. E'et i ER-diagram, står får ***entitet***. En entitet[1] er en enhed, som vi gerne vil gemme oplysninger om. Ofte er entiteter givet ved et navneord f.eks. hund, bil, kunde, vare osv. I vores eksempel er der tre entiteter, vi gerne vil gemme oplysninger om. Nemlig elev, fest og festudvalgs-medlem. For hver af disse entiteter vil vi gerne gemme forskellige oplysninger. For eleverne kunne det f.eks. være navn og klasse, for hver fest kunne det være dato, festens navn, om der er spisning eller ej og hvilken musikform (f.eks. om det er live band eller DJ) og for festmedlemmerne vil man gerne gemme navn, email, mobil og type (om det er en elev eller en ansat). Alle disse oplysninger som vi gerne vil gemme om hver entitet (man siger også, at det er egenskaber ved entiteten) kaldes for ***attributter***.

Nu vil det være sådan, at man har brug for unikt at kunne identificere hver entitet i hver entitetsklasse. Forestil dig f.eks. har vi har to elever i klassen 2023x, som begge hedder "Mette Andersen". Hvis vi ikke ved andet om eleverne, vil vi ikke kunne kende forskel på de to. Det er noget rod! Derfor er det et krav, at der til hver entitet knyttes en ***primærnøgle***. Det er en (eller evt. flere) attributter, som unikt kan identificere hver entitet. Det er et krav, at nøglen består af så få attributter som muligt. For elev-entiteten kunne man f.eks. vælge elevens CPR-nummer. Det vil unikt kunne identificere hver elev. Ofte vil man dog til lejligheden vælge et "kunstigt" id forstået på den måde, at id'et ikke har nogen betydning uden for den database, som vi er i gang med at designe. Sådanne id'er vil ofte være et helt tal. Hvis du er medlem af en sportsklub, har du f.eks. sikkert et medlemsid, som er et sådant helt tal, der ikke betyder noget uden for klubben. Der er flere fordele ved et sådant "kunstigt" nummer fremfor et CPR-nummer - én af dem er, at vi ikke kommer i problemer med GDPR. For hver af de tre entiteter vælger vi derfor følgende primærnøgler: elevid (Elev), festid (Fest), medlemsid (Festudvalgsmedlem).

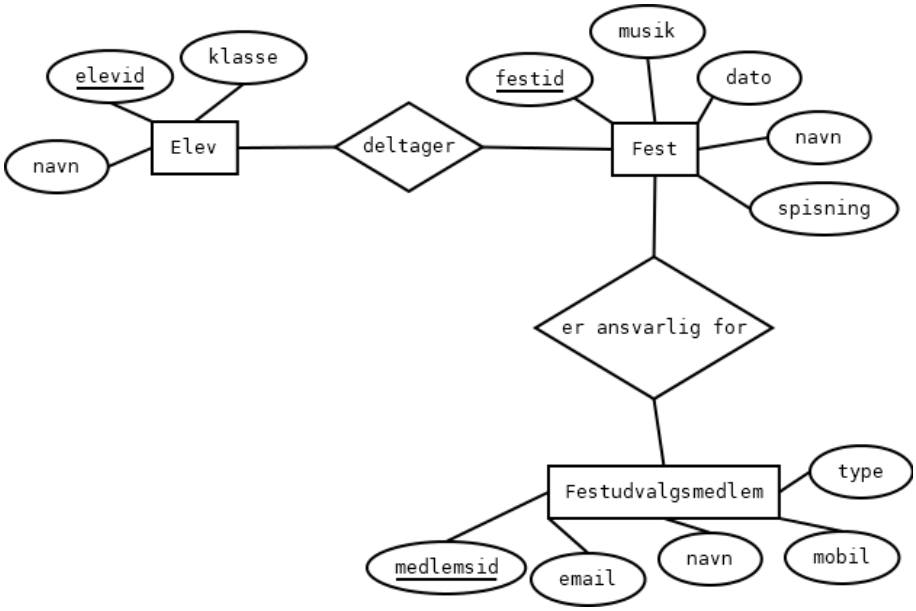
Når man skal lave et ER-diagram er konventionen, at entiteter skrives i firkantede kasser, og de tilhørende attributter skrives i ovaler med en streg hen til den tilhørende entitet. Samtidig viser man

[1] Hvis man skal være meget præcis, er det *entitetsklasse*, men som regel vil vi ikke her skelne mellem en entitet og en entitetsklasse.

hvad der er primærnøglen ved at understrege denne/de attribut(ter). Indtil videre vil det se sådan her ud:



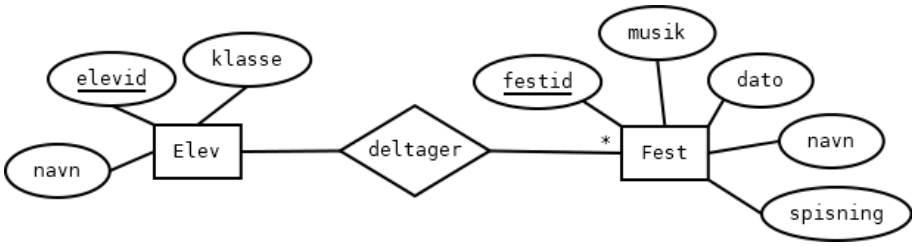
Det er fint nok, men indtil videre hænger de tre entiteter ikke sammen på nogen måde. Men det gør de i virkelighedens verden! Eleverne deltager i festerne, og et festudvalgsmedlem er ansvarlig for hver fest. Disse ***relationer*** (dvs. sammenhænge) mellem entiteterne - elev og fest samt fest og festudvalgsmedlem - er netop hvad R'et i ER-diagram står for. En relation er således en sammenhæng eller en kobling mellem to entiteter. Relationen mellem elev og fest kunne vi kalde for "deltager", fordi en elev deltager i en fest og relationen mellem fest og festudvalgsmedlem kunne vi kalde for "er ansvarlig for", fordi et festudvalgsmedlem er ansvarlig for en fest. I ER-diagrammer tegnes en relation i en diamantform og forbindes med streger til de entiteter, som relationen sammenkobler. I vores eksempel ser det sådan her ud:



For at kunne designe databasen mangler vi at tilføje en enkelt ting til vores ER-diagram, nemlig det man kalder for ***kardinaliteter*** eller ***relationsgrad***. Lad os starte med at se på deltager-relationen mellem elev og fest. Her vil det være sådan, at

én elev kan deltage i mange fester

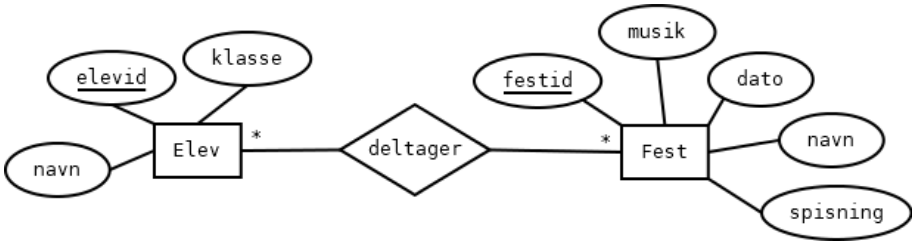
Derfor sætter man en stjerne (*) på stregen mellem deltager og fest:



Omvendt kan man også sige, at til

én fest kan der deltage mange elever.

Derfor sætter man også en stjerne (*) på stregen mellem deltager og elev:

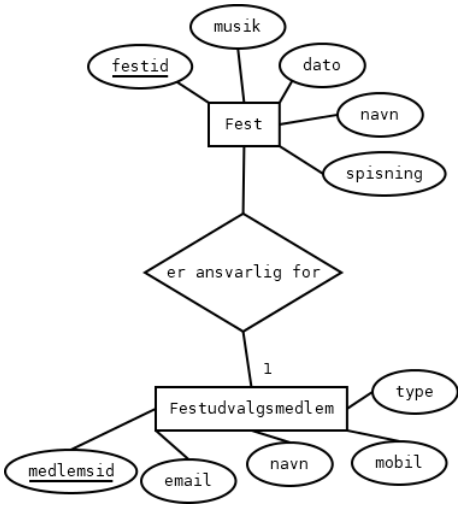


Man siger her, at der er tale om en ***mange-mange relation***.

Ser vi nu på "er ansvarlig for"-relationen mellem fest og festudvalgsmedlem, så vil det være sådan, at til

én fest vil der være ét festudvalgsmedlem, som er ansvarlig.

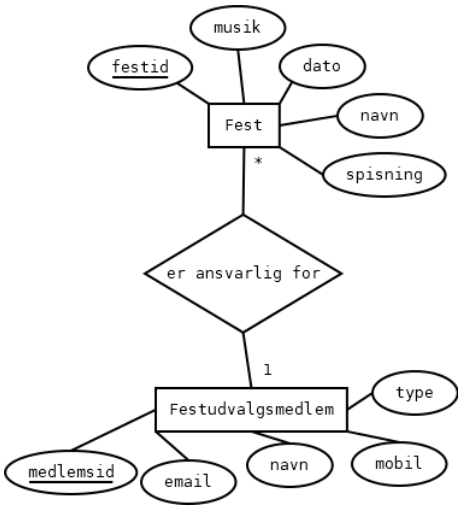
Derfor sætter man et 1-tal på stregen mellem "er ansvarlig for" og festudvalgsmedlem:



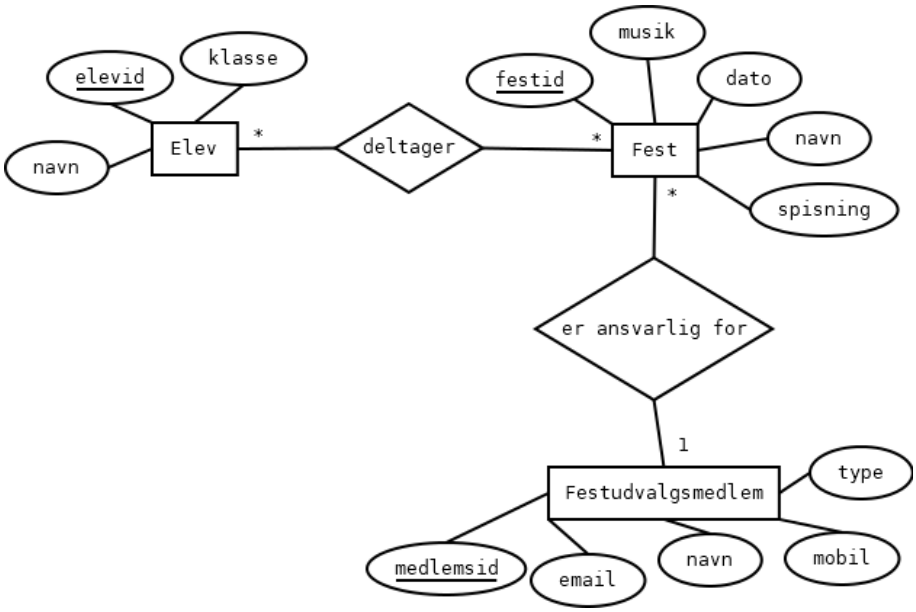
Omvendt kan

ét festudvalgsmedlem være ansvarlig for mange fester

og derfor sætter man en stjerne (*) på stregen mellem ”er ansvarlig for” og fest:



Her er der således tale om en ***1-mange*** relation. Alt i alt ender vi med følgende ER-diagram:



Man kan også forestille sig ***1-1*** relationer, det ses dog ikke så tit.

Det er vigtigt at lægge mærke til, at for hver relation, så er der to veje at forklare, når man skal redegøre for relationsgraden, som det er gjort ovenfor. Man starter altid med at sige ”én … ” og så læser man tegnet længst væk (1 eller *). De to veje at forklare i deltager-relationen bliver altså:

én elev kan deltage i mange fester

og til

én fest kan der deltage mange elever.

Man kan f.eks. ikke bare sige, at "mange elever deltager i mange fester" - det kan man nemlig altid sige! F.eks. vil mange festudvalgsmedlemmer også være ansvarlig for mange fester, så det udsagn, er der ikke nogen brugbar information i.

Tabelskitser

Så langt så godt. Nu har vi et ER-diagram, men hvordan kommer vi fra et ER-diagram til at oprette tabeller i en database? Inden vi hopper direkte over i databasesystemet og opretter tabeller, vil vi lave tabelskitser. En ***tabelskitse*** er præcis hvad navnet antyder: en skitse af en tabel! En tabelskitse er derfor på formen:

tabelnavn(primærnøgle, attribut1, attribut2, …)

Det giver nok umiddelbart god mening, at vi skal have en tabel for hver entitet. Så i første omgang får vi derfor følgende tabelskitser:

elev(elevid, navn, klasse)

fest(festid, navn, dato, musik, spisning)

festudvalgsmedlem(medlemsid, email, navn, mobil, type)

Det vil betyde, at vi i databasen skal oprette tre tabeller, hvor f.eks. tabellen "elev" skal have tre kolonner nemlig: "elevid" (som skal være primærnøgle), "navn" og "klasse". Hver række i "elev"-tabellen, vil så svare til en konkret elev (f.eks. "Mette Andersen" som går i klassen 2023x, og som har elevid 175). Tabellerne kunne f.eks. se sådan her ud:

Elev

elevid	navn	klasse
175	Mette Andersen	2023x
193	Søren Jokumsen	2022y
64	Niels Malling	2021a

Fest

festid	navn	dato	musik	Spisning
1	Julefest	10-12-2022	live	nej
2	Fastelavnsfest	18-02-2023	DJ	nej
3	Galla	15-04-2023	live	ja

Festudvalgsmedlem

medlemsid	navn	email	mobil	type
1	Søren Jokumsen	sj@gmail.com	43430192	Elev
2	Hanne Hermansen	hh@skole.dk	48391045	ansat
3	Peter Gotfredsen	peter_gotfather@mail.dk	94059381	ansat

Men de tre tabeller alene vil ikke være nok. De indeholder ingen information om, hvilke elever, der deltager i hvilke fester og heller ikke hvilket festudvalgsmedlem, som er ansvarlig for en given fest. Vi er derfor nødt til at se på de to relationer også og overveje, hvordan vi kan gemme netop disse informationer.

Vi starter med at se på 1-mange relationen "er ansvarlig for". Vi kan jo starte med at overveje, om vi kan gemme informationen om hvilket festudvalgsmedlem, der er ansvar for en given fest i én af tabellerne "fest" eller "festudvalgsmedlem". Hvis vi skulle gemme informationen i tabellen "festudvalgsmedlem", ville der opstå et problem, fordi et festudvalgsmedlem kan være ansvarlig for mange fester. Så for hvert festudvalgsmedlem vil vi få brug for et ukendt antal kolonner til at gemme informationen om, hvilke fester han eller hun er ansvarlig for. Det er dur ikke. Omvendt kan et festudvalgsmedlem kun være ansvar for én fest. Derfor kan vi gennem denne oplysning i "fest"-tabellen ved for hver fest (som vil svarer til en række i "fest"-tabellen), at gemme medlemsid'et på det festudvalgsmedlem, som er ansvarlig for netop denne fest. Tabelskitsen for fest, skal derfor ændres til:

fest(festid, musik, dato, navn, spisning, medlemsid → festudvalgsmedlem)

Notationen "medlemsid → festudvalgsmedlem" betyder, at medlemsid er primærnøgle i tabellen "festudvalgsmedlem", og når denne optræder i en anden tabel (her tabellen "fest"), så kalder man den for en ***fremmednøgle***.

Med den nye opdatering kan "fest" tabellen derfor se sådan her ud:

Fest

festid	navn	dato	Musik	Spisning	medlemsid
1	Julefest	10-12-2022	Live	Nej	1
2	Fastelavnsfest	18-02-2023	DJ	Nej	2
3	Galla	15-04-2023	Live	Ja	2

Det betyder, at det er Søren Jokumsen (med medlemsid 1), der er ansvarlig for julefesten, mens Hanne Hermansen (med medlemsid 2) er ansvarlig for både fastelavns- og gallafesten.

Nu mangler vi kun at se på "deltager"-relationen. Man kan igen overveje, om vi kan gemme informationen om, hvilke elever, der deltager i hvilke fester i enten "fest"- eller "elev"-tabellen. Men vi får samme problem, som før. Hvis vi gemmer informationen i "fest"-tabellen, så ved vi ikke, hvor mange kolonner, der skal bruges for at, fordi vi ikke kender det maksimale antal deltagere til hver fest. Og selv hvis vi gjorde - lad os sige, at der højst kan deltage 500 elever i hver fest, så vil vi skulle oprette 500 kolonner (dvs. søjler) i "fest"-tabellen. Er der så en fest, hvor der kun deltager 300 elever, så vil vi få en række i "fest"-tabellen, hvor der ikke står noget i 200 af kolonnerne. Det er spild af plads! Et helt tilsvarende argument fås, hvis vi forestiller os, at vi vil gemme oplysningen om hvilke elever, der deltager i hvilke fester i "elev"-tabellen: Vi ved ikke, hvor mange fester en elev vil deltage i, så vi ved ikke på forhånd, hvor mange kolonner vi skal lave. Derfor gør man det, at man for hver mange-mange relation opretter en ny tabel. Attributterne i denne tabel skal være primærnøglerne fra de to tabeller, som indgår i relationen - dvs. at de begge bliver fremmednøgler i den nye tabel. Men samtidig er det også de to attributter, der tilsammen unikt kan identificere en række i tabellen og derfor kommer de til at udgøre primærnøglen i den nye tabel. Når primærnøglen består af mere end én attribut kalder man

den for en ***sammensat nøgle***. Tabelskitsen for den nye tabel (som vi kalder for "deltager") kommer til at se sådan her ud:

deltager(<u>elevid</u> → elev, <u>festid</u> → fest)

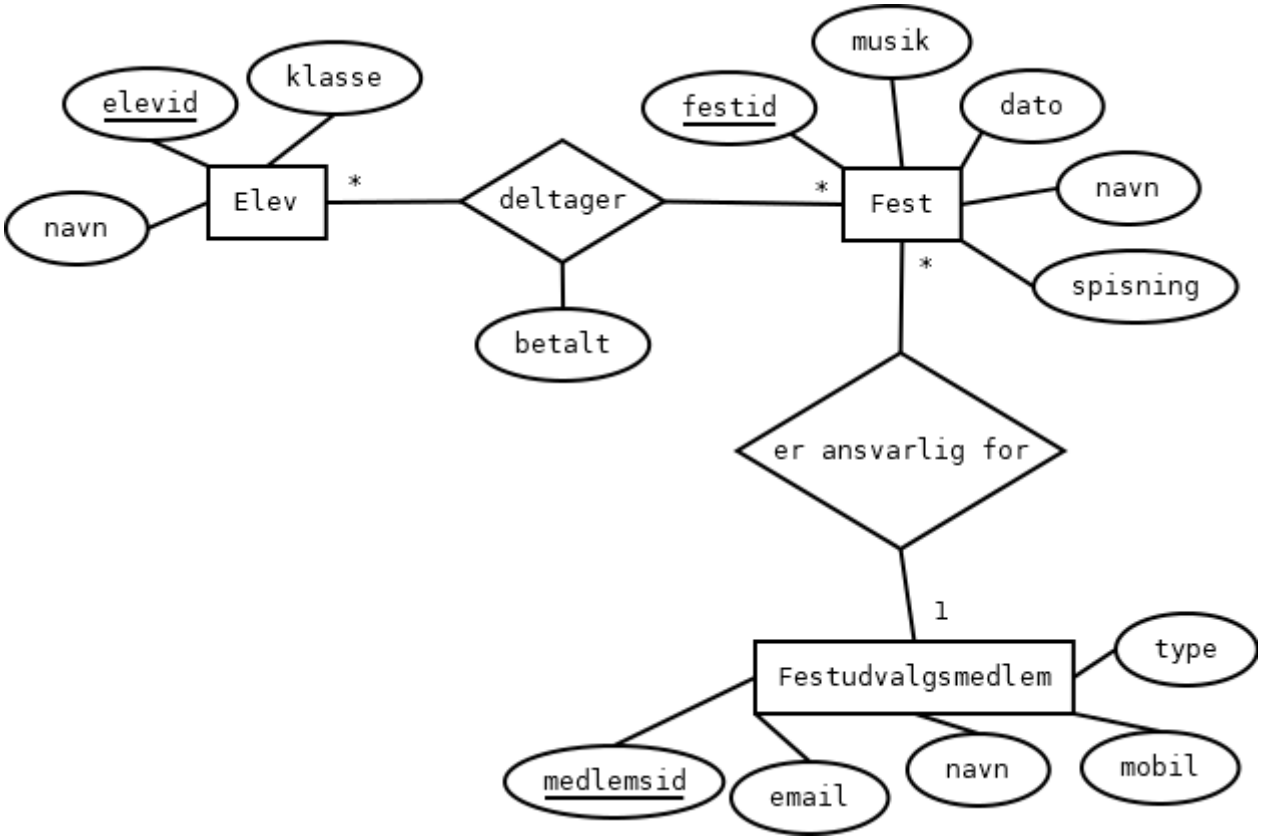
"Deltager"-tabellen kunne f.eks. se sådan her ud:

<u>elevid</u>	<u>festid</u>
175	1
175	2
175	3
64	3
193	2
193	3

Denne tabel indeholder bare en masse tal, og sådan er det faktisk også i den virkelige verden. Tabeller som kobler to tabeller med hinanden indeholder typisk bare tal, som repræsenterer primærnøgler fra andre tabeller.

Ud fra tabellen kan vi se, at Mette Andersen (med elevid 175) er en rigtig festabe. Hun deltager i alle tre fester (med festid 1, 2 og 3). Niels Malling (elevid 64) skal kun med til gallafesten (festid 3) og Søren Jokumsen (elevid 193) skal med til fastelavns- og gallafesten. Nu tænker den opmærksomme læser nok "Nej, hov hov! Søren Jokumsen er jo ansvarlig for julefesten, så den kommer han da også til!!". Svaret er: Ja, det kan godt være. Men som vores database er designet, så ved vi faktisk ikke, om der er tale om den samme Søren Jokumsen. Det er ikke sikkert - og måske er det den samme person, der er tale om. Faktum er, at det kan vi ikke se ud fra vores database. Hvis det er vigtigt at holde styr på, så skal databasen designes anderledes. Det er ikke fordi, at vi nødvendigvis har gjort noget forkert. Det er et designvalg, og det vil altid være op til den konkrete anvendelse at afgøre, om det er det rigtige valg eller ej. I vores lille eksempel vil vi antage, at vores valg er korrekt i forhold til den specifikke anvendelse.

Nu har vi planlagt, hvordan vores database skal se ud, og man kunne forestille sig, at vi som et led i en iterativ arbejdsproces vil fremlægge vores ER-diagram og designovervejelser for de fremtidige brugere af databasen. Så kunne det jo være, at der var en kvik bruger, som siger "Jamen, hvordan registrerer man, om elever har betalt for festen?". Nå for katten! Det har vi faktisk ikke taget højde for. Så må vi tilbage til skrivebordet og overveje, hvordan vi kan modellere det. Om, en elev har betalt for en given fest, er hverken knyttet til eleven alene eller til festen alene, men derimod til "deltager"-relationen mellem elev og fest. Vi beslutter os derfor for at udvide ER-diagrammet med en ny attribut på "deltager"-relationen:



Vi må derfor opdatere tabelskitsen for ”deltager” til:

deltager(elevid → elev, festid → fest, betalt)

”Deltager”-tabellen kunne så f.eks. se sådan her ud:

elevid	festid	betalt
175	1	ja
175	2	ja
175	3	nej
64	3	nej
193	2	ja
193	3	nej

Tanken er nu, at værdien af attributten ”betalt” bliver opdateret, når en elev har betalt for den fest, han eller hun har tilmeldt sig.

Vi ender altså alt i alt med disse fire tabelskitser:

elev(elevid, navn, klasse)

fest(festid, musik, dato, navn, spisning, medlemsid → festudvalgsmedlem)

festudvalgsmedlem(medlemsid, email, navn, mobil, type)

deltager(elevid → elev, festid → fest, betalt)

For at opsummere kommer vi altså fra ER-diagram til tabelskitser på denne måde:

- ✓ Lav en tabel for hver entitet
- ✓ For hver 1-mange relation gemmes oplysningen i tabellen på mange siden
- ✓ For hver mange-mange relation laves en ny tabel.

SQL

SQL står for Structured Query Language, og det er det sprog, som vi skal bruge for at oprette de fire tabeller i et databasesystem. I denne note er det vist, hvordan man gør i SQLite, men syntaksen er tilsvarende i andre databasesystemer, som er baseret på SQL.

For at oprette de fire tabeller, skal man starte med at oprette en tom database (tænk på databasen som den "beholder", hvor alle tabellerne opbevares). Herefter opretter man en tabel ved at bruge kommandoen `create table` (se mere her). Tabelskitsen for "elev"-tabellen er:

elev(elevid, navn, klasse)

Og i SQLite oprettes denne tabel på denne måde:

```
create table elev (
elevid int primary key,
navn text,
klasse text
);
```

Her står `int` for integer, som betyder heltal. Dvs. at datatypen for kolonnen "elevid" er hele tal. Derimod er datatypen for kolonnerne "navn" og "klasse" af typen `text` - altså tekststrenge. Efter `elevid` står der primary key - det betyder selvfølgelig, at vi angiver, at `elevid` skal være primærnøgle i tabellen.

Hvis man har fået oprettet en tabel, hvor man er kommet til at lave en fejl, kan man skrive

```
drop table elev;
```

Køres denne kommando slettes tabellen (incl. alle data - så pas godt på!), og den kan nu oprettes på ny.

Tabelskitsen for "festudvalgsmedlem"-tabellen er:

festudvalgsmedlem(medlemsid, email, navn, mobil, type)

Og tabellen oprettes på (næsten) tilsvarende vis som ovenfor:

```
create table festudvalgsmedlem (
medlemsid int primary key,
```

```
email text,
navn text,
mobil int,
type text check(type in ('elev','ansat')));
```

Bemærk her syntaksen `check(type in ('elev','ansat'))`. Det betyder, at vi får databasesystemet til at kontrollere, at der kun indsættes værdien "elev" eller "ansat" i kolonnen "type". Hvis man f.eks. prøver at indsætte værdien "rektor", vil man få en fejl.

Når vi skal oprette "fest"-tabellen med tabelskitse

fest(festid, musik, dato, navn, spisning, medlemsid → festudvalgsmedlem)

skal vi huske at angive, at "medlemsid" er en fremmednøgle, som refererer til primærnøglen i "festudvalgsmedlem"-tabellen. Det gøres på denne måde:

```
create table fest (
festid int primary key,
navn text,
dato text,
musik text check (musik in ('live','DJ')),
spisning text check(spisning in ('ja','nej')),
medlemsid int references festudvalgsmedlem(medlemsid)
);
```

Bemærk for det første, at vi begrænser de tilladte værdier i "musik"- og "spisning"-kolonnen. For det andet fremgår det, at man med kommandoen

```
medlemsid int references festudvalgsmedlem(medlemsid)
```

angiver, at kolonnen "medlemsid" skal være en fremmednøgle, som refererer til primærnøglen fra "festudvalgsmedlem"-tabellen. Kolonnen behøver for øvrigt ikke at have navnet "medlemsid" - man kan kalde den lige det, man har lyst til! Denne fremmednøgle reference betyder også, at vi først kan oprette "fest"-tabellen, når "festudvalgsmedlem" er oprettet, fordi vi kun kan få lov til at indsætte medlemsid'er, som allerede findes i "festudvalgsmedlem"-tabellen.

For det tredje kan man se, at datatype til kolonnen "dato" er tekst. Nogle databasesystemer stiller en egentlig datotype til rådighed (de hedder typisk noget med "date"), men det er altså ikke tilfældet i SQLite.

Endelig mangler vi tabellen for "deltager"-relationen med tabelskitse:

deltager(elevid → elev, festid → fest, betalt)

Den oprettes sådan her:

```
create table deltager (
elevid int references elev(elevid),
festid int references fest(festid),
betalt text check(betalt in ('ja','nej')),
primary key (elevid, festid)
);
```

Udover at angive to fremmednøgler og sætte begrænsninger på værdierne af "betalt", så er det også her vist, hvordan man opretter en sammensat primærnøgle. Det gøres ved til sidst at skrive:

```
primary key (elevid, festid)
```

Nu mangler vi kun at indsætte værdier i tabellerne. Det gør man ved bruge kommandoen `insert into` (se mere her). Vi indsætter i "elev"-tabellen på denne måde:

```
insert into elev (elevid, navn, klasse) values
(175, 'Mette Andersen','2023x'),
(193, 'Søren Jokumsen','2022y'),
(64, 'Niels Malling','2021a');
```

Efter `insert into` skriver man tabellens navn efterfulgt at navnet på de kolonner, hvor man gerne vil indsætte værdier. Herefter skriver man værdierne for hver række i parenteser. Hvis man indsætter værdier i alle kolonner, er det faktisk ikke nødvendigt at skrive kolonnernes navn, så længe man skriver værdierne i den rigtige rækkefølge. Derfor kan man indsætte i "festudvalgsmedlem"-tabellen på denne måde:

```
insert into festudvalgsmedlem values
(1, 'Søren Jokumsen','sj@gmail.com',43430192,'elev'),
(2,'Hanne Hermansen','hh@skole.dk',48391045,'ansat'),
(3,'Peter Gotfredsen','peter_gotfather@mail.dk',94059381,'ansat');
```

Vi indsætter i "fest"-tabellen:

```
insert into fest values
(1,'Julefest','2022-12-10','live','nej',1),
(2,'Fastelavnsfest','2023-02-18','DJ','nej',2),
```

```
(3,'Galla','2023-04-15','live','ja',2);
```

Bemærk, at datoerne er skrevet i formatet *YYYY-MM-DD* (*year-month-day*). Selvom SQLite ikke har en egentligt datatype til datoer, stilles der alligevel en række funktioner til rådighed, som kan håndtere datoer, hvis de er skrevet i dette format (se mere her). Derfor kan det være en fordel.

Endelig kan vi indsætte i "deltager"-tabellen:

```
insert into deltager values

(175,1,'ja'),(175,2,'ja'),(175,3,'nej'),

(64,3,'nej'),(193,2,'ja'),(193,3,'nej');
```

Nu er databasen klar til brug, og vi kan lave de sædvanlige SQL forespørgsler til tabellerne (se SQL Tutorial (w3schools.com)). Vi vil blot her give et enkelt eksempel. Når f.eks. Mette med "elevid" 175 på et tidspunkt betaler for gallafesten (med "festid" 3), så kan vi ændre værdien af "betalt" kolonnen fra 'nej' til 'ja'. Det gøres med `update` kommandoen (se mere her):

```
update deltager set betalt='ja' where elevid=175 and festid=3;
```

Her får vi netop opdateret kolonnen "betalt" til 'ja' i den række, hvor elevid'et er 175 og festid'et er 3.