# SQL for INF1300

Stein Michael Storleer

4. november 2016

## SQL—Structured Query Language

- **SQL—Structured Query Language**—er et deklarativt språk for spørringer mot relasjonsdatabaser. Spørringer skjer med **select**-setningen.
- SQL inneholder også konstruksjoner for å definere nye relasjonsdatabaser, for å legge inn og endre data i databasen:

  **create table ...**

  **insert into ... values**

  **update ... set**

  **alter table**

  **drop table ...**

SQL består av en samling konstruksjoner som funksjonelt, men ikke syntaktisk, kan deles opp slik:

- SDL: Storage Definition Language
  3-skjemaarkitekturens[1] fysiske lag
- **DDL: Data Definition Language**[2]
  3-skjemaarkitekturens konseptuelle lag
- VDL: View Definition Language
  3-skjemaarkitekturens presentasjonslag
- **DML: Data Manipulation Language**
  innlegging, endring og sletting av data
- **DQL: Data Query Language—spørrespråk**
- DCL: Data Control Language—integritet og sikkerhet

## SQLs DDL—Data Definition Language

[1] 3-skjemaarkitektur er et viktig prinsipp i alle databasesystemer. Denne innebærer en deling av databaseskjemaet i presentasjonslaget, det konseptuelle (logiske) laget og det fysiske laget. Målet med denne delingen er at man skal kunne endre deler av databaseskjemaet uten at det er nødvendig å endre andre lag. For eksempel kan man endre måten tall lagres på (fysisk lag) uten å endre informasjonen om hva som skal lagres i systemet (logisk lag).

[2] De deler av SQL vi kommer borti i INF1300 er satt med **fete typer**.

**create table**: Opprette tabell/relasjon

**drop table**: Fjerne tabell

**alter table**: Endre tabell, *herunder:*

- Legge til eller fjerne kolonner
- Legge til, fjerne eller endre integritetsregler (constraints)

Med disse tre setningene definerer vi hele databasen vår.

## create table

Med den defineres alle tabellene, og hva de heter. For hver tabell angir vi attributter og hvilken type (domene) det enkelte attributt skal ha. Vi definerer også nøkler og andre viktige skranker/integritetsregler. Hvis vi har laget en ORM-modell, kan tabelldefinisjonene automatiseres, dvs. modelleringsprogrammet (f.eks. ORMLite) lager **create**-setningene for oss. Har vi ikke en modell i ORM (eller et annet modelleringsspråk) må vi skrive **create**-setningene selv. Hvordan vi gjør det har avgjørende betydning for hvor god databasen blir til å gjenspeile den virkelighet og de fakta vi er ute etter.

**create table** $R$ **(**

$A_1$ $D_1$ [$S_1$] **,**

...

$A_n$ $D_n$ [$S_n$]

[ **,** *liste av skranker* ]

**)**;

$R$ er navnet på relasjonen/tabellen
$A_i$ er et attributt
$D_j$ er et domene (datatype)
$S_k$ er en skranke

[ ] betyr at dette leddet er en valgfri del av setningen

## Eksempler på **create**

### Ansatt(AId, Navn, Tittel, Fdato, Pnr, AnsDato)

- Vi ønsker ikke at to ansatte skal kunne ha samme **AId**
- To personer kan aldri ha samme fødselsnummer = **Fdato** + **Pnr**
- Dermed er både **AId** og **(Fdato, Pnr)** kandidatnøkler. Vi velger **AId** som primærnøkkel og markerer **(Fdato, Pnr)** som kandidatnøkkel.

```
create table Ansatt (
    AId         int primary key,
    Navn        varchar(40) not null,
    Tittel      varchar(15) not null,
    Fdato       char(6) not null,
    Pnr         char(5) not null,
    AnsDato     date,
    unique (Fdato, Pnr)
);
```

## Datatyper i PostgreSQL

| *datatype* | *forklaring* |
|---|---|
| `integer` | heltall |
| `int` | heltall |
| `bigint` | store heltall (8 byte) |
| `real` | flyttall |
| `numeric(n,d)` | $n$ desimale sifre, $d$ etter desimalpunktum |
| `char(n)` | tekst med fast lengde |
| `varchar(n)` | tekst med variabel lengde |
| `text` | tekst med ubegrenset variabel lengde |
| `boolean` | boolsk verdi |
| `date` | dato |
| `time` | klokkeslett |
| `timestamp` | dato og klokkeslett |
| `bit(n)` | bitstreng med fast lengde |
| `bit varying(n)` | bitstreng med variabel lengde |

Et eksempel fra filmdatabasen[3]:

```
create table filmparticipation (
 partid int primary key,
 personid int not null references person (personid),
 filmid int not null references filmitem (filmid),
 parttype text not null
);
```

[3]Filmdatabasen er beskrevet i et eget notat av Ellen Munthe-Kaas som finnes under ressurser på semestersidene.

Legg merke til definisjonen av attributtet **parttype** som har datatypen **text**. Denne er ikke standard SQL.[4]

## Nøkler og nøkkelattributter

Nøklene i en relasjon er de viktigste skrankene. En kandidatnøkkel er kort sagt et eller flere attributter som bare kan ha unike verdier. F.eks. (fødselsdato, personnr), eller (brukernavn), eller (filmid), eller (brukernavn, emnekode, semester).

- **Supernøkkel**:
  En kombinasjon (delmengde) $X$ av attributtene $\{A_1, A_2, \ldots, A_n\}$ som er slik at hvis $t$ og $u$ er to tupler hvor $t \neq u$, så er $t[X] \neq u[X]$. Altså at hvis to tupler er forskjellige, er de også forskjellige når vi bare ser på attributtene som er i $X$. Merk: I relasjonsmodellen er samtlige attributter i én tabell/relasjon alltid selv en supernøkkel. Supernøkler benyttes til å uttrykke integritetsregler.
- **Kandidatnøkkel**: En *minimal* supernøkkel.
  Dvs: Fjerning av et hvilket som helst attributt fører til at de gjenværende attributtene ikke lenger utgjør en supernøkkel.
- **Primærnøkkel**: En utvalgt blant kandidatnøklene. I relasjonsmodellen skal alle relasjoner ha nøyaktig én primærnøkkel. Primærnøkkelen blir markert med én strek.
- Hvis det er andre kandidatnøkler, markeres de med én strek, og primærnøkkelen med to.[5]
- **Nøkkelattributt**: Attributt som er med i (minst) en kandidatnøkkel.

Sammenlign kandidatnøkler og entydighetsskranker i ORM: Begge angir at forekomster under skranken bare kan forekomme én gang.

## Primærnøkler

Kan deklareres i **create table** sammen med primærnøkkelattributtet (bare hvis attributtet utgjør primærnøkkelen alene)

```
create table Ansatt (
    AId         int primary key,
... );
```

[4] Datatyper varierer ganske mye fra system til system. Eksempler fra Access, MySQL og SQL Server, se `http://www.w3schools.com/sql/sql_datatypes.asp`

[5] Vi har i INF1300 gjort det omvendt tidligere, men har endret dette fra 2015 for å bli konsistent med læreboka.

Kan deklareres separat i **create table** etter attributtdeklarasjonene

```
create table Ansatt (
    AId         int,
    ...
    primary key (AId)
);
```

- Maks én primærnøkkeldeklarasjon pr. relasjon
- Konsekvenser av deklarasjonen:
  - To tupler i relasjonen får ikke stemme overens i alle attributtene i primærnøkkelen.
  - Attributtene i primærnøkkelen får ikke inneholde **null**
- Dette må sjekkes av systemet ved hver **insert** og hver **update**. Forsøk på brudd ved **insert** eller **update** skal avvises av DBMSet.

## Kandidatnøkler

- Kan deklareres i **create table** *sammen med nøkkelattributtet* (bare hvis attributtet utgjør kandidatnøkkel alene)

```
create table Ansatt (
    ...
    Fnr     char(11) not null unique,
    ... );
```

- Kan deklareres separat i **create table** *etter* attributtdeklarasjonene

```
create table Ansatt (
    ...
    Fdato   char(6) not null,
    Pnr     char(5) not null,
    ...
    unique (Fdato, Pnr)
);
```

- Flere kandidatnøkkeldeklarasjoner er tillatt pr. relasjon
- Konsekvenser av deklarasjonen:
  - To tupler i relasjonen får ikke stemme overens i alle attributtene i kandidatnøkkelen
  - Kan brytes hvis ett eller flere av attributtene i kandidatnøkkelen inneholder **null**
- Dette må sjekkes av systemet ved hver **insert** og hver **update**. Forsøk på brudd ved **insert** eller **update** skal avvises av DBMSet.

## Andre skranker

- Skranke på et attributt eller et tuppel i en relasjon:
  **create table** R ( ... **check** ... );
  - Sjekkes ved **insert** og **update** av R
- Skranke på tvers av relasjoner:
  **create trigger** T ...;
  - Triggere «vekkes» når en hendelse (typisk insert, update, delete på en relasjon) skal til å inntreffe
  - Når triggerne «vekkes», eksekveres en tilhørende metode

## Skranke på ett attributt

- **not null**
  - **create table** Ansatt ( ... Fdato int **not null**, ..);
  - Konsekvenser:
    * Kan ikke sette inn tuppel med verdien null i attributtet
    * Kan ikke endre verdien til null senere
- **check**
  - ```
    create table Ansatt (
        ...
        Tittel varchar(15)
            check (Tittel='Selger' or Tittel='Direktør' or ...),
        ...
    );

    create table InfEmne (
        ...
        emnekode varchar(7) check (emnekode like 'INF____'),
        ...
    );


        ...
        alder integer check (alder between 0 and 99),
        ...
        emnekode varchar(12) check (
                        emnekode in      (
                                'INF1000',
                                'INF1500',
                                'INF1080',
                                'INF1100' )
                        )
        ...
    ```
  - Angir en betingelse på attributtet. Sjekkes ved hver endring av attributtets verdi
    ```

## Fremmednøkler

En fremmednøkkel refererer til en primærnøkkel (ett eller flere attributter) i en annen tabell.

- Deklarasjon av fremmednøkler:

```
create table Timeliste (
    AId      int references Ansatt (AId),
    ...);
```

- Alternativt:

```
create table Timeliste (
    AId      int,
    ...
    foreign key (AId) references Ansatt (AId)
    ...);
```

- Flere attributter i nøkkelen

```
create table Student (
    bnavn varchar(8) primary key,
    fd date,
    pnr varchar(5),
    ...
    foreign key (fd, pnr) references Person (fdato, personnr)
    ...);
```

- Konsekvenser av deklarasjonen:
  - De refererte attributtene må være deklarert **primary key** [6]. De må altså være primærnøkkel i den refererte tabellen.
  - Verdier ($\neq$ **null**) som opptrer i fremmednøkkelens refererende attributter *må* opptre i de refererte attributtene
- Dette må sjekkes av systemet både ved **insert**, **update** og **delete**

Fremmednøkler mot flere tabeller brukes for å implementere et mange-til-mange forhold mellom tabeller:

```
create table Student (
    bnavn char(8) primary key,
    navn varchar(80),
    ...
);

create table Emne (
    ekode char(10) primary key,
    emnenavn varchar(80),
    emneeier varchar(80),
    ...          );
```

[6] Noen databasesystemer tillater at fremmednøkler refererer til kandidatnøkler generelt, men det er dårlig programmeringsskikk.

```
create table Antalleksamensforsøk (
    brukernavn char(8) references Student(bnavn),
    emnek char(10) references Emne(ekode),
    antforsøk integer,
    primary key (brukernavn, emnek)
);
```

God skikk å alltid angi attributtene i den refererte relasjonen, selv om det ikke er nødvendig.

## drop, alter

Disse setningene lar oss slette tabeller, samt legge til og slette attributter i eksisterende relasjoner.

**drop table** $R$;

**alter table** $R$ **add** $A_x$ $D_y$;

**alter table** $R$ **drop** $A_x$;

$R$ er et relasjonsnavn, $A_x$ er et attributt, $D_y$ er et domene

# SQLs DQL—Data Query Language

**select** [ **distinct** ] ATTRIBUTTLISTE
**from** NAVNELISTE
[ **where** WHERE-BETINGELSE ]
[ **group by** GRUPPERINGSATTRIBUTTER
[ **having** HAVING-BETINGELSE ] ]
[ **order by** ATTRIBUTT [ **asc** | **desc** ]
[ , ATTRIBUTT [ **asc** | **desc** ] ] ... ];

[ ] betyr at dette leddet er en valgfri del av setningen

- **select**
  Angir hvilke attributter som skal vises i svaret
- **distinct**
  Fjerner flerforekomster (duplikater) av svartuplene
- **from**
  Navn på de relasjonene spørringen refererer til
- **where**
  Seleksjonsbetingelse (kan inneholde en eller flere join-betingelser)
- **group by**, **having**
  Angir grupperingsattributter til bruk ved aggregering og betingelser på resultatet av grupperingen; disse kommer vi tilbake til i en senere forelesning
- **order by** Ordner tuplene i henhold til angitte kriterier

## select-setningen

- Typisk utseende:

  **select** [**distinct**] $A_1, A_2, \ldots, A_j$
  **from** $R_1, R_2, \ldots, R_k$
  **where** $C$;

  hvor

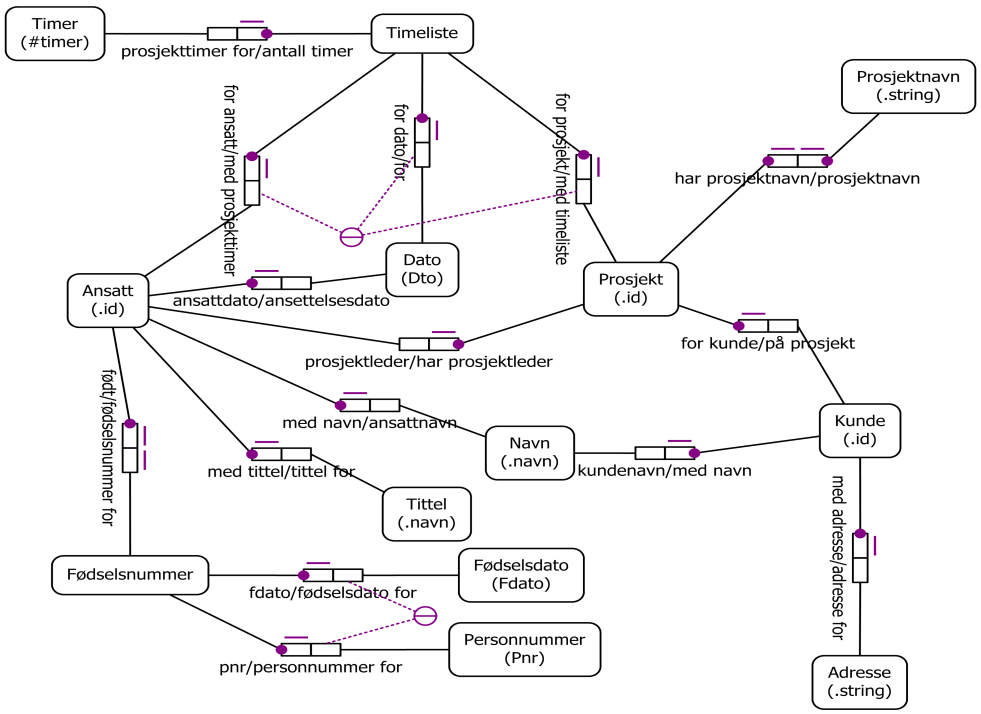
  $R_1, R_2, \ldots, R_k$ er relasjonsnavn

  $A_1, \ldots, A_j$ er attributter fra $R_1, R_2, \ldots, R_k$

  $C$ er en betingelse

## Eksempler med select

Modell av en prosjektdatabase med tilhørende skjema:




**Skjema**
Prosjekt(PId, Pnavn, KId, Pleder, StartDato)
Ansatt(AId, Navn, Tittel, Fdato, Pnr, AnsDato)
Timeliste(AId, Dato, PId, Timer)
Kunde(KId, Knavn, Adresse)

Finn navn på de ansatte som er ansatt etter 2003. (Det kan være flere ansatte som har samme navn.)

```
select distinct Navn
from Ansatt
where AnsDato > date '2003-12-31'
```

Finn navn og startdato for alle prosjekter bestilt av kunden «Pust og pes AS». Sorter dem slik at det nyeste prosjektet kommer først.

```
select   Pnavn, StartDato
from     Kunde K, Prosjekt P
where    Knavn ='Pust og pes AS' and K.KId = P.KId
order by StartDato desc;
```

## Seleksjons- og join-betingelser

La oss se nærmere på eksemplet ovenfor:

where-betingelsen består av to deler:

- Knavn = 'Pust og pes AS'
  Dette leddet kalles en ***seleksjonsbetingelse***
  Det plukker ut forekomster/tupler i Kunde (her trolig bare én)

- K.KId = P.Kid
  Dette leddet kalles en ***join-betingelse***
  Det kobler sammen forekomster/tupler fra Kunde med tupler/forekomster i Prosjekt forutsatt at verdiene i attributtene KId og Kid er like

*Oppgave*: Finn navn og tittel på alle som har arbeidet på prosjektet «Vintersalg»

- *Løsning*

```
select distinct Navn, Tittel
from       Ansatt A, Timeliste T, Prosjekt P
where      Pnavn = 'Vintersalg' and
           P.PId = T.PId and T.AId = A.AId;
```

Her består join-betingelsen av to ledd. Den binder sammen en forekomst fra hver av de tre tabellene Ansatt, Timeliste og Prosjekt. Join-attributtene behøver ikke å ha samme navn. Det holder at de har samme domene.

## Merknader til select

- **select** (SQL) skiller ikke mellom store og små bokstaver, unntatt i tekststrenger

- **select** beregner *bager* (med unntak av noen av operatorene)

En *bag* er en *samling* av tupler der samme tuppel kan forekomme flere ganger. (Like tupler fjernes eventuelt ved å bruke **distinct**.)

## Eksempler fra filmdatabasen

```
select
   x.partid, x.personid, x.filmid, x.parttype
from
   filmparticipation x, person p, film f
where
   p.lastname = 'Besson' and
   p.firstname = 'Luc' and
   f.title = 'Fifth␣Element,␣The' and
   x.personid = p.personid and
   x.filmid = f.filmid
;
```

Resultat:

```
 partid  | personid | filmid | parttype
---------+----------+--------+----------
  781285 |    89222 | 237127 | director
 1009544 |    89222 | 237127 | writer
 1009560 |    89222 | 237127 | writer
 1009576 |    89222 | 665467 | writer
 1009592 |    89222 | 665467 | writer
(5 rows)
```

```
select
    f.title, c.filmcharacter
from
    filmparticipation x, person p, film f, filmcharacter c
where
    x.parttype = 'cast' and
    p.lastname = 'Jovovich' and
    p.firstname = 'Milla' and
    x.personid = p.personid and
    x.filmid = f.filmid and
    x.partid = c.partid
;
```

Resultat:

```
                            title                             |   filmcharacter
--------------------------------------------------------------+--------------------
 .45                                                          | Kate
 AFI's 100 Years... 100 Cheers: America's Most Inspiring Movies | Herself
 Cannes: Through the Eyes of the Hunter                       | Herself
 Chaplin                                                      | Mildred Harris
 Claim, The                                                   | Lucia
 Corporate Malfeasance                                        | Herself
 Dazed and Confused                                           | Michelle Burroughs
 Dummy                                                        | Fangora
 Fifth Element, The                                           | Leeloo
 Fifth Element, The                                           | Leeloo
 Game Babes                                                   | Herself
 Game Over: 'Resident Evil' Reanimated                        | Herself
 He Got Game                                                  | Dakota Burns
 House on Turk Street, The                                    | Erin
 Kuffs                                                        | Maya Carlton
 Making and Meaning of 'We Are Family', The                   | Herself
 Messenger: The Story of Joan of Arc, The                     | Joan of Arc
 Million Dollar Hotel, The                                    | Eloise
 Night Train to Kathmandu, The                                | Lily McLeod
 Playing Dead: 'Resident Evil' from Game to Screen            | Herself
 Resident Evil                                                | Alice
 Resident Evil: Apocalypse                                    | Alice
 Resident Evil: Extinction                                    | Alice
 Return to the Blue Lagoon                                    | Lilli
 Star Element, The                                            | Herself
 Starz on the Set: Ultraviolet                                | Herself
 Teen Vid II                                                  | Herself
 Trailer for a Remake of Gore Vidal's Caligula                | Druscilla
 Two Moon Junction                                            | Samantha Delongpre
 Ultraviolet                                                  | Violet
 VH1/Vogue Fashion Awards                                     | Herself
 You Stupid Man                                               | Nadine
 Zoolander                                                    | Katinka
(33 rows)
```

## Uttrykk i betingelser (**where** og **having**)

**where**-betingelsen er et boolsk uttrykk hvor atomene har en av følgende former:

- Verdisammenlikning: P **op** Q
  - P og Q må ha samme domene, minst en av dem må være et attributt, den andre kan være en konstant
  - **op** $\in \{=, <, >, <=, >=, <>,$ **like**$\}$
    (**like** er bare lov når Q er en konstant tekststreng)
- null-test: P **is null** eller P **is not null**
- Relasjonssammenlikning: **exists**, **in**, **all**, **any**
  (Disse tar vi for oss i en senere forelesning)

Spesialregler for sammenlikning av **strenger** :

- Leksikografisk ordning: $s < t$, $s > t$, $s <= t$, $s >= t$
- Sammenlikning: $s = t$, $s <> t$
- Mønstergjenkjenning: $s$ **like** $p$
  $p$ er et mønster hvor
  **%** matcher en vilkårlig sekvens (null eller flere tegn)
  **_** matcher ett vilkårlig tegn

Datoer og tidspunkter:

- Dato: **date** 'yyyy-mm-dd'
- Tidspunkt: **time** 'hh:mm', **time** 'hh:mm:ss'
- Tidspunkt med finere gradering enn sekund:
  **time** 'hh:mm:ss.ccc'
- Tidspunkt før GMT: **time** 'hh:mm:ss+hh'
- Tidspunkt etter GMT: **time** 'hh:mm:ss-hh'
- Dato og tid: **timestamp** 'yyyy-mm-dd hh:mm:ss'

## select—navnekonflikter

- Kvalifiser attributter med relasjonsnavn: R.A
- Navngi relasjoner med aliaser:
  ...**from** R **as** S... (**as** kan sløyfes)
  S blir en kopi av R med nytt relasjonsnavn
- Gi attributter nytt navn:
  **select** A **as** B **from**...
  A omnavnes til B i resultatrelasjonen

## SQLs DML—Data Manipulation Language

Tilbur setninger for å håndtere tupler og verdier i dem.

- **insert**: Innsetting av nye data
- **update**: Endring av eksisterende data
- **delete**: Sletting av data

## insert

**insert into** $R(A_1, A_2, \ldots, A_k)$
**values** $(v_1, v_2, \ldots, v_k)$;

**insert into** $R(A_1, A_2, \ldots, A_k)$
**select**-setning;

- Attributtlisten kan sløyfes hvis den dekker samtlige attributter i $R$ og følger attributtenes default rekkefølge
- **NB**—optimaliseringer i DBMSet kan medføre at tuplene legges inn etterhvert som de beregnes i **select**-setningen. Dette kan ha sideeffekter på beregningen av **select**-setningen

## update, delete

**update** $R$
**set** $A_1 = E_1, \ldots, A_k = E_k$
[**where** $C$];

**delete from** $R$
[**where** $C$];

$R$ er en relasjon, $A_i$ er attributter (kolonnenavn) og $E_j$ uttrykk. [ ] betyr at dette leddet er en valgfri del av setningen

## Tekstmønstre

- I SQL kan vi bruke **like** for å sammenligne et tekst-attributt med et tekstmønster

- Et *tekstmønster* er en tekstkonstant hvor to tegn, kalt jokertegn, har spesiell betydning:
  - _ (understrekning) passer med *ett* vilkårlig tegn
  - % passer med en vilkårlig tekststreng (null eller flere tegn)

Eksempel 1:

```
select firstname from person
where firstname like 'O_a';
```

passer med Oda og Ola og O4a, men *ikke* med Olga

Eksempel 2:

```
select firstname from person
where firstname like 'O%a';
```

passer med alle tekststrenger som begynner med «O» og slutter med «a», som Oa, Ola, Olga, Othilia, Oda, Ofjhwskjfhkxxa

Eksempel 3:

```
select firstname || '␣' || lastname
            as navn, -- slår sammen 3 tekststrenger
       gender as kjonn
from   person
where firstname like '___' and
       lastname not like '%sen';
```

Resultatet blir en tabell over navn og kjønn på personer som har eksakt tre tegn i fornavnet og et etternavn som ikke slutter på «sen». Navnet består av for- og etternavn adskilt med en blank.

Eksempel fra filmdatabasen. Tittel og produksjonsår for filmer med strengen 'Tintin' i seg:

```
select title, prodyear
from film
where title like '%Tintin%';
```

Resultat:

```
                  title                   | prodyear
------------------------------------------+----------
 Tintin et les oranges bleues             |     1964
 Tintin et moi                            |     2003
 Tintin et le mystère de la Toison d'Or   |     1961
 Tintin et le lac aux requins             |     1972
 Moi, Tintin                              |     1976
 Tintin et le temple du soleil            |     1969
 I, Tintin                                |     1976
 Tintin: Chez les negros                  |
 Tintin and the Lake of Sharks            |     1972
 Tintin and the Mystery of the Golden Fleece |     1961
 Tintin and the Temple of the Sun         |     1969
 Tintin and the Blue Oranges              |     1964
```

```
 Tintin and I                                |     2003
 Tintin and Me                               |     2003
 Tintin och jag                              |     2004
 Tintin og mig                               |     2003
 Tintin                                      |     2009
(17 rows)
```

Legg merke til tuppelet som ikke har noe produksjonsår. Her er verdien null, ikke blank eller tallet 0.

## Aggregeringsfunksjoner

SQL har fem aggregeringsfunksjoner:

| *navn* | *virkning (returnerer)* |
|---|---|
| **count** | teller antall |
| **min** | finner minste verdi |
| **max** | finner største verdi |
| **sum** | summerer verdier |
| **avg** | finner gjennomsnitt av verdier |

## count()

- `select count(*) from person;`

  gir antall tupler i tabellen

- `select count(*) as antTupler from person;`

  Som for alle attributter i select-listen, kan vi gi **count**(*) et nytt navn.

- `select count(gender) from person;`

  gir antall tupler i tabellen hvor attributtet gender ikke er null

- `select count(distinct firstname) from person;`

  gir antall forskjellige verdier i attributtet firstname (**null** telles ikke med)

## min() og max()

- min(attributt) og max(attributt) gir henholdsvis minste og største verdi av attributtet
- Attributtet må være numerisk eller tekstlig
  (date og time håndteres som tekststrenger)
- Eksempel: Gitt tabellen Ansatt(anr, navn, lonn, avd). Finn den største lønnsforskjellen i salgsavdelingen:

```
select max(lonn) - min(lonn)
from Ansatt
where avd = 'Salg';
```

## sum() og avg()

- sum(attributt) og avg(attributt) beregner henholdsvis summen og gjennomsnittet av verdiene i attributtet
- Attributtet må være numerisk
- Tupler hvor attributtet er **null**, blir ignorert. (Dette er viktig for avg())
- Eksempel: Gitt tabellen Ansatt(anr, navn, lonn, avd). Finn sum lønnsutgifter og gjennomsnittslønn for salgsavdelingen:

```
select sum(lonn), avg(lonn)
from Ansatt
where avd = 'salg';
```

# group by

- Gruppering er å dele forekomstene inn i grupper og så gi **en** resultatlinje for hver gruppe
- Syntaksen er slik:

```
select <resultatattributt-liste>
from <tabell-liste>
where <betingelse>
group by <grupperingsattributt-liste>;
```

- Resultatet beregnes slik:
  1. Beregn **select** * **from** <tabell-liste> **where** <betingelse>
  2. Lag grupper av de tuplene som er like i alle grupperingsattributtene
  3. Utfør aggregeringsfunksjonene lokalt innenfor hver gruppe og presenter én resultatlinje for hver gruppe
- En god regel er å inkludere alle grupperingsattributtene i resultatattributtlisten.
- Merk: Attributter som *ikke* er grupperingsattributter, kan bare forekomme som argument til aggregeringsfunksjoner.

## Eksempler

Ansatt(anr, navn, lonn, avd)
Avdeling(avdnr, a-navn, leder)
Prosjektplan(pnr, anr, timer)

Finn antall ansatte i hver avdeling og gjennomsnittlig lønn for disse:

```
select   a-navn, count(*), avg(lonn)
from     Ansatt, Avdeling
where    avd = avdnr
group by a-navn, avdnr;
```

For hvert prosjekt, list opp medvirkende avdelinger og sorter dem etter innsats:

```
select   pnr as prosjekt, avdnr as avdnummer,
         max(a-navn) as avdeling,
         sum(timer) as innsats
from     Ansatt A, Avdeling, Prosjektplan P
where    avd = avdnr and A.anr = P.anr
group by pnr, avdnr
order by pnr, sum(timer) desc;
```

Siste linje kan erstattes med

```
order by prosjekt, innsats desc;
```

Samme oppgave, men ta bare med avdelinger som skal bidra med minst 100 timer på prosjektet:

```
select   pnr as prosjekt, avdnr as avdnummer,
         max(a-navn) as avdeling,
         sum(timer) as innsats
from     Ansatt A, Avdeling, Prosjektplan P
where    avd = avdnr and A.anr = P.anr
group by pnr, avdnr
having   sum(timer) > 99
order by prosjekt, innsats desc;
```

Eksempler med group by og aggregering fra filmadatabasen:

```
select prodyear, count(*) as antall
from film
group by prodyear
order by prodyear;
```

```
 prodyear | antall
----------+--------
     1888 |      2
     1889 |      1
     1890 |      3
     1891 |      7
     1892 |     12
     1893 |      4
     1894 |     82
     1895 |    137

  .....

     2005 |  24089
     2006 |  21796
```

```
     2007 |   9340
     2008 |   2203
     2009 |    696
     2010 |     64
     2011 |      1
          |   1277
(125 rows)
```

```
select genre, count(*)
from filmgenre
group by genre
order by genre ;
```

```
    genre    | count
-------------+--------
 Action      |  22370
 Adult       |  33999
 Adventure   |  15431
 Animation   |  24356
 Biography   |   4426
 Comedy      |  92070
 Crime       |  18420
 Documentary |  69950
 Drama       | 114529
 Family      |  19170
 Fantasy     |   8903
 Film-Noir   |    440
 Game-Show   |   2159
 History     |   3463
 Horror      |  11623
 Music       |  11757
 Musical     |   9248
 Mystery     |   7966
 News        |   1642
 Reality-TV  |   2399
 Romance     |  21551
 Sci-Fi      |   8241
 Short       | 138289
 Sport       |   4310
 Talk-Show   |   2093
 Thriller    |  16830
 War         |   6009
 Western     |  10089
(28 rows)
```

Hvor mange kjønn finnes i tabellen Person?

```
select gender as kjønn,
       count(*) antall
from person
group by gender;
```

```
 kjønn | antall
-------+--------
       | 415520
 M     | 810006
 F     | 483858
(3 rows)
```

```
select country land, count(*) as ant_filmer
from filmcountry
group by country
order by count(*) desc
limit 9; -- bare de første 9 linjene i svaret
```

```
  land   | ant_filmer
---------+------------
 USA     |     206556
 UK      |      41320
 France  |      34626
 Germany |      22882
 Canada  |      21075
 India   |      19614
 Italy   |      17900
 Spain   |      16639
 Mexico  |      15281
(9 rows)
```

```
select  max(lastname) || ', ' || max(firstname) as filmdeltaker,
        count(distinct parttype) as antall_deltakerfunksjoner
from  person p, filmparticipation x
where x.personid = p.personid
group by p.personid
having count(distinct parttype) > 6;
```

```
      filmdeltaker        | antall_deltakerfunksjoner
--------------------------+---------------------------
 Aldridge, Stephanie      |                         7
 Amaral, Henrique         |                         7
 Amero, Lem               |                         7
 Anderson, Fred           |                         7
 Bannatyne, Neamh         |                         7
 Bell, Bevan              |                         7
 Blackburn, Slade         |                         7
 Booth, Christopher Saint |                         7
 Bower, Brett             |                         7
 Bressane, Júlio          |                         7
 Chamorro, Daniel         |                         7
 Culhane, James           |                         7
 Fagundes, Antonio Augusto |                         7
 Gallo, Vincent           |                         7
 Griffith, D.W.           |                         7
 Haslaman, Hakan          |                         7
 Holmes, Al               |                         7
 Holwerda, Gus            |                         7
 Irvine, Travis           |                         7
 Jarzebowski, Katy        |                         7
 Jittlov, Mike            |                         7
 Kondratiuk, Andrzej      |                         7
 Lind, Jay                |                         7
 Luellen, Rick            |                         7
 Myracle, Brice           |                         7
 Necro,                   |                         7
 Odoutan, Jean            |                         7
 Rocca, Vincent           |                         7
 Sganzerla, Rogério       |                         7
 Spadaccini, Anthony      |                         7
 Taylor, Al               |                         7
 Undergaro, Keven         |                         7
 Wascavage, Dave          |                         7
 Wasylkiw, Blaine         |                         7
 Zedd, Nick               |                         7
(35 rows)
```

Dette resultatet er kanskje ikke så rart, siden

```
select distinct parttype
from filmparticipation;
```

```
     parttype
------------------
 writer
 costume designer
 director
 editor
 cast
 composer
 producer
(7 rows)
```

Navn på personer som har hatt en annen rolle enn å være skuespiller i mer enn 777 filmer:

```
select max(firstname) || ' ' || max(lastname) as navn,
-- må aggregere, siden firstname og lastname
-- ikke er blant grupperingsattributtene
       parttype as deltakerfunksjon,
       count(*) as antall_filmer
from  person p, filmparticipation x
where x.personid = p.personid
group by p.personid, parttype
having count(*) > 777
       and parttype not like 'cast';
```

```
       navn       | deltakerfunksjon | antall_filmer
------------------+------------------+---------------
 Bob Cobert       | composer         |          1289
 Bradley Bell     | writer           |          1128
 Ramsey Mostoller | costume designer |          1102
  Ilayaraja       | composer         |           895
 Dan Curtis       | producer         |          1259
 Dan Curtis       | writer           |          1236
 Robert Costello  | producer         |           990
 Hal Roach        | producer         |          1142
 Winston Sharples | composer         |           807
 Anna Home        | producer         |          1074
 Matt Groening    | writer           |           955
 David E. Kelley  | writer           |           825
 Aaron Spelling   | producer         |          1645
 Geoff McQueen    | writer           |          1440
 Paul Terry       | producer         |          1072
 Art Wallace      | writer           |          1231
(16 rows)
```

```
select lastname etternavn,
       firstname fornavn,
       count(*) as antall
from   Person
where firstname like '_%'
group by lastname, firstname
having count(*) > 24
order by antall desc ;
```

```
 etternavn | fornavn | antall
-----------+---------+--------
 Williams  | John    |     30
 Williams  | David   |     30
 Martin    | John    |     29
 Scott     | Michael |     28
 King      | John    |     26
 Lee       | David   |     26
 Smith     | Michael |     26
 James     | David   |     26
 Davis     | John    |     26
 Miller    | John    |     25
 White     | Michael |     25
 Johnson   | Michael |     25
 Smith     | Paul    |     25
 Smith     | John    |     25
 Scott     | David   |     25
 Taylor    | John    |     25
 Wilson    | John    |     25
 Jones     | Chris   |     25
 Murphy    | Michael |     25
l(19 rows)
```

```
select title, count(distinct film.filmid)
from Film, Filmitem
where     filmtype = 'C'
      and film.filmid = filmitem.filmid
group by title
having count(distinct film.filmid) > 25
order by count(distinct film.filmid) desc;
```

```
       title        | count
--------------------+-------
 Popular Science    |    45
 Love               |    42
 Mother             |    41
 Hamlet             |    39
 Desire             |    37
 Stranger, The      |    37
 Unusual Occupations|    35
 Trap, The          |    34
 Carmen             |    34
 Home               |    33
 Destiny            |    31
 Jealousy           |    31
 Honeymoon          |    29
 Cinderella         |    29
 Alone              |    29
 Revenge            |    29
 Dead End           |    28
 Kiss, The          |    28
 Escape             |    27
 Temptation         |    27
 Fear               |    26
 Awakening, The     |    26
(22 rows)
```

## Generelt utseende av SQL-spørsmål

```
  select    [distinct] <resultatattributter>
  from      <tabeller>
[ where     <utvalgbetingelse> ]
[ group by  <grupperingsattributter>
[ having    <resultatbetingelse> ] ]
[ order by  <ordningsattributter> ]
```

Regler:

- Ordningsattributtene har utseendet:
  <attributt> [asc | desc]
- Ordningsattributtene må være blant resultatattributtene
- Ifølge SQL-standarden skal grupperingsattributtene alltid være blant resultatattributtene, men de fleste DBMSer krever ikke dette

## where vs. having

- **where**-betingelsen velger ut de tuplene som skal danne datagrunnlaget for grupperingen
- **having**-betingelsen plukker ut de tuplene fra det ferdig-grupperte resultatet som skal med i det endelige svaret
- **having**-betingelsen kan inneholde aggregatfunksjoner, men det kan ikke **where**-betingelsen
- Siden **having** håndterer en (mye) mindre datamengde enn **where**, er det i kompliserte spørringer lurt å legge så mye som mulig av logikken inn i **having**-betingelsen

## Hvordan SQL-spørsmål med group by evalueres

1. Selekter ifølge seleksjonsbetingelsene i **where**
2. Join relasjonene i **from** i henhold til joinbetingelsene i **where**
3. Grupper resultattuplene i henhold til like verdier i grupperingsattributtene angitt i **group by**-klausulen
4. Fjern de gruppene som ikke oppfyller resultatbetingelsen i **having**-klausulen
5. Projiser ifølge attributtene i **select**
6. Fjern flerforekomster hvis **select distinct**
7. Sorter i henhold til **order by**

## Views

- Et view er en tenkt relasjon som vi bruker som mellomresultat i kompliserte SQL-beregninger
- Det er to måter å lage view på:
  - **create view** navn(attributtliste) **as select** ...
  - **create view** navn **as select** ...
- I det første tilfellet må det være like mange navn i attributtlisten som det er attributter i **select**-setningen
- I det andre tilfellet arver viewet attributtnavnene fra **select**-setningen
- Når man har laget et view, kan det brukes som en vanlig tabell i senere **select**-setninger

## Eksempel på view

Prosjektplan(pnr, anr, timer)

```
create view Innsats as
            select   anr, sum(timer) as timer
            from     Prosjektplan
            group by anr;


create view Bonus(anr, bonusiNOK) as
            (select  anr, 3000
            from     Innsats
            where    timer >= 500 )
       union
            (select  anr, 1500
             from    Innsats
             where   timer >= 300 and timer < 500 );
```

Her har vi brukt mengdeoperatoren union for å slå sammen tuplene fra to tabeller. Mer om mengdeoperatorer i lysarkene om relasjonsalgebra.

Eksempel fra filmdatabasen.

```
/* Lage et view med oversikt over hvor
 * mange funksjoner filmarbeiderne
 * med flere enn 1 funksjon har hatt

 *  Først lager vi selectsetningen */

select p.personid, count(distinct parttype)
from  person p, filmparticipation x
where x.personid = p.personid
group by p.personid
having count(distinct parttype) > 1;
```

```
-- P.g.a. manglende rettigheter kan vi ikke lage views i fdb
-- Dette eksemplet er kjørt mot utdraget vi har i hver brukers DB

create view funksjoner (personid, antroller) as
select p.personid, count(distinct parttype)
from  person p, filmparticipation x
where x.personid = p.personid
group by p.personid
having count(distinct parttype) > 1;
```

Da kan vi bruke viewet som en tabell:

```
select max(firstname) || ' ' || max(lastname) as navn,
       parttype as deltakerfunksjon,
       count(distinct filmid) as antall_filmer
from  person p,
      filmparticipation x,
      funksjoner f
where x.personid = p.personid
      and f.personid = p.personid
      and f.antroller > 5
group by p.personid, parttype
having count(distinct filmid) > 7;
```

```
       navn         | deltakerfunksjon | antall_filmer
--------------------+------------------+---------------
 Dominique Delouche | director         |            16
 Dominique Delouche | writer           |             8
 Philipe Bordier    | cast             |            11
 Philipe Bordier    | director         |            15
 Philipe Bordier    | editor           |             8
 Philipe Bordier    | writer           |            15
 Laurent Boutonnat  | composer         |            10
 Jean-Pierre Bouyxou | cast            |            32
 Jean-Pierre Bouyxou | director        |             8
 Jean-Pierre Bouyxou | writer          |            16
 Vincent Gallo      | cast             |             8
 Luis Buñuel        | cast             |            11
 Luis Buñuel        | director         |            12
 Luis Buñuel        | writer           |            14
 Pierre Clémenti    | cast             |            85
 Julie Delpy        | cast             |            30
 Jean Odoutan       | cast             |             8
 Jesus Franco       | cast             |            26
 Jesus Franco       | director         |            57
 Jesus Franco       | writer           |            53
 Orson Welles       | cast             |            34
 Orson Welles       | director         |            10
 Orson Welles       | writer           |            10
 Joseph Morder      | cast             |            17
 Joseph Morder      | director         |            21
 Joseph Morder      | writer           |            15
 Luc Moullet        | cast             |            39
 Luc Moullet        | director         |            33
 Luc Moullet        | writer           |            24
(29 rows)
```

## Nestede spørsmål

Gitt tabellen Ansatt(anr, navn, lonn, avd)

Finn antall selgere som tjener *mer enn det dobbelte* av markedsførernes gjennomsnittslønn

```
select count *
from  Ansatt
where avd = 'salg' and
    lonn > ( select 2 * avg(lonn)
             from Ansatt
             where avd = 'marketing');
```

Merk: En **select** inne i **where**-betingelsen må være omsluttet av parenteser

Ansatt(anr, navn, lonn, avd)
Avdeling(avdnr, avdelingsnavn, leder)
Prosjektplan(pnr, anr, timer)

For hvert prosjekt, list opp medvirkende avdelinger som har minst 10 ansatte og sorter dem etter innsats (Altså: ta bare med avdelinger som har minst 10 ansatte):

```
select    pnr as prosjekt, avdnr as avdnummer,
          max(avdelingsnavn) as avdeling, sum(timer) as innsats
from      Ansatt A, Avdeling, Prosjektplan P
where     avd = avdnr and A.anr = P.anr
group by pnr, avdnr
having    sum(timer) > 99 and
          9 < (select count(*)
               from  Ansatt A1
               where A1.avd = avdnr)
order by prosjekt, innsats desc;
```

Merk bruken av avdnr i den indre select-setningen! Den gjør at den indre select-setningen må beregnes én gang for hver verdi av avdnr beregnet i den ytre select-setningen. Dette kalles *korrellerte* spørringer.

*— Spørringen fra view-eksemplet over, nå omskrevet uten bruk av view*

```
select max(firstname) || ' ' || max(lastname) as navn,
       parttype as deltakerfunksjon,
       count(distinct filmid) as antall_filmer
from  person p,
      filmparticipation x,
      (select p.personid as personid,
              count(distinct parttype) as antroller
       from  person p, filmparticipation x
       where x.personid = p.personid
       group by p.personid
       having count(distinct parttype) > 1) as f
where x.personid = p.personid
      and f.personid = p.personid
      and f.antroller > 5
group by p.personid, parttype
having count(distinct filmid) > 7 ;
```

Skuespillere som har deltatt i mer enn 20 filmer og hvor ingen filmer har fått en rangering som er dårligere enn gjennomsnittsrankingen til alle filmer + 1:

```
select p.personid,
       max(lastname) || ', ' || max(firstname) as navn,
       count(distinct f.filmid) as antfilmer,
       min(rank) as minimumsvurdering
from film f, filmrating r,
     filmparticipation x, person p,
     filmcountry c
where      r.filmid = f.filmid and rank is not null
      and p.personid = x.personid
      and f.filmid = x.filmid and parttype = 'cast'
      and c.filmid = f.filmid
      and country not like 'India'
group by p.personid
having count(distinct f.filmid) > 20
       and min(rank) > ( 1 + (select avg(rank)
                              from filmitem i, film f,
                              filmrating r
                              where      i.filmid = f.filmid
                              and r.filmid = f.filmid
                              and rank is not null
                             )
                         )
;
```

```
 personid |       navn        | antfilmer | minimumsvurdering
----------+-------------------+-----------+-------------------
   286025 | Barsi, Béla       |        22 |               6.7
   711113 | Caramitru, Ion    |        23 |               6.7
   948697 | Cotescu, Octavian |        23 |               7.6
  1897625 | Hara, Setsuko     |        21 |               6.6
  2157383 | Okada, Mariko     |        26 |               6.5
  2821977 | López, Marga      |        22 |               6.8
  3143273 | Mishima, Masao    |        22 |               6.8
  3287913 | Naniwa, Chieko    |        21 |               6.5
  4196905 | Shpigel, Grigori  |        29 |               6.4
(9 rows)
```

## Relasjonssammenligninger

SQL har fem operatorer som sammenligner med innholdet i en hel relasjon:

| *i SQL-2* | *betyr* |
|---|---|
| **exists R** | at R har minst én forekomst |
| **not exists R** | at R ikke har noen forekomster |
| **in R** | $\in$ R |
| **not in R** | $\notin$ R |
| **any R** | en vilkårlig verdi i R |
| **all R** | alle verdier i R |

## any og all

any og all brukes i praksis bare på relasjoner med ett attributt.

Ansatt(anr, navn, lonn, avd)
Avdeling(avdnr, a-navn, leder)
Prosjektplan(pnr, anr, timer)

Finn antall selgere som tjener mer enn samtlige markedsførere

```
select count(*)
from   Ansatt
where  avd = 'salg'  and
       lonn > all (select lonn
                   from   Ansatt
                   where  avd = 'marketing');
```

For eksempler med any, se eksempler fra filmdatabasen lenger ned.

## in og not in

[ not ] in kan brukes på ett attributt eller på en liste av attributter

Finn navn på ansatte som ikke har ført noen prosjekttimer

```
select navn
from   Ansatt
where  anr not in (select anr
                   from   Prosjektplan);
```

Finn filmer produsert i et nordisk land:

```
select F.filmid
from Film F natural join
     Filmcountry FC
where FC.country in ('Denmark','Finland','Iceland','Norway','Sweden') ;
```

Finn navn på regissører som har resgissert filmer som er kategorisert i mer enn 5 filmsjangre. Først finner vi filmer som har mer enn 5 sjangre:

```
select FG.filmid
from Filmgenre FG
group by FG.filmid
having count(*) > 5
```

Så bruker vi denne tabellen som seleksjonsbetingelse, dvs. vi tar bare med filmider som finnes i denne:

```
select P.lastname, P.firstname
from filmparticipation FP
     natural join filmitem FI
     natural join Person P
where FI.filmtype = 'C' and
      FP.parttype = 'director'
      AND FP.filmid in ( select FG.filmid
                         from filmgenre FG
                         group by FG.filmid
                         having count(*) > 5 );
```

Hvis den indre setningene gir filmidene $f_1, f_2, \ldots f_k$, blir det siste AND-uttrykket det samme som å skrive:

AND (FP.filmid = $f_1$ OR FP.parttype = $f_2$ OR ... OR FP.parttype = $f_k$)

## exists og not exists

**exists** R
er sann hvis tabellen inneholder tupler (ett eller flere)

**not exists** R
er sann hvis tabellen ikke inneholder noen tupler

Merk at SQL ikke har noen egen all-kvantor ($\forall$)

Skulle vi trenge en all-kvantor, må vi uttrykke den ved hjelp av andre SQL-konstruksjoner

## Noen nyttige formler fra logikken

- F $\Rightarrow$ G $\equiv$ **not** F **or** G
- **not** (F **and** G) $\equiv$ **not** F **or not** G
- **not** (F **or** G) $\equiv$ **not** F **and not** G
- $\forall$u.F $\equiv$ **not** ($\exists$u.**not** F)
- $\exists$u.F $\equiv$ **not** ($\forall$u.**not** F)

## Eksempler

Ansatt(anr, navn, lonn, avd)
Avdeling(avdnr, a-navn, leder)
Prosjektplan(pnr, anr, timer)

Finn navn på ansatte som skal arbeide mer enn 10 timer på samtlige av sine prosjekter.

```
select A.navn
from   Ansatt A
where  not exists ( select *
                    from   Prosjektplan P
                    where  P.anr = A.anr
                    and P.timer <= 10 );
```

Finn navn på ansatte som skal delta på alle prosjekter

```
select A.navn
from   Ansatt A
where  not exists (
           select pnr
           from   Prosjektplan P1
           where  not exists (
                      select *
```

```
        from    Prosjektplan P2
        where   P2.pnr = P1.pnr
        and     P2.anr = A.anr));
```

Ansatt(anr, navn, lonn, avd)
Avdeling(avdnr, a-navn, leder)
Prosjekt(pnr, leder)
Prosjektplan(pnr, anr, timer)

Finn navn på de ansatte som ikke skal delta på noe prosjekt ledet av en avdelingsleder

```
select navn
from   Ansatt A
where  not exists (
           select *
           from   Prosjekt P, Prosjektplan PL
           where  PL.anr = A.anr and
                  PL.pnr = P.pnr and
                  P.leder in (select leder
                              from   Avdeling) );
```

## Plassering av sub-queries

- Det er lov å ha sub-queries (indre **select**-setninger) i
  - **from**-klausulen
  - **where**-klausulen
  - **having**-klausulen
- SQL-standarden inneholder ingen øvre grense for antall sub-queries i et query
- Sub-queries skal alltid være omsluttet av parenteser

Eksempler fra filmdatabasen. Merk at mange eksempler kan skrives om å gjøres kortere uten bruk av nestede setninger.

Kinofilmer som bare har én verdi i genre

Antall filmer i filmparticipation som har regissør. *(Om natural join, se lenger ned i notatet.)*

```
select count(distinct filmid)
from filmparticipation fp
     natural join filmitem fi
where filmtype = 'C'
      and exists
                  (select filmid
                   from filmparticipation x
                   where fp.filmid = x.filmid
                         and x.parttype = 'director');
```

```
 count
--------
 330490
(1 row)
```

Antall filmer i filmparticipation som ikke har regissør.

```
select count(distinct filmid)
from filmparticipation fp
     natural join filmitem fi
where filmtype = 'C'
      and not exists
                  (select filmid
                   from filmparticipation x
                   where fp.filmid = x.filmid
                         and x.parttype = 'director');
```

```
 count
-------
 31157
(1 row)
```

Filmer med i Filmparticipation

```
select count(distinct filmid)
-- distinct fordi fp har flere tupler med samme filmid.
from filmparticipation fp;
```

```
 count
--------
 934752
(1 row)
```

```
-- filmer med i Film
-- filmid primærnøkkel distinct overflødig
select count(filmid)
from film;
```

```
 count
--------
 692361
(1 row)
```

```
-- antall filmer i filmparticipation som ikke finnes i film
select count(distinct filmid)
from filmparticipation fp
where filmid not in (select filmid from film);
-- svært tidkrevende fordi indre selectsetning har svært mange tupler

-- BEDRE: det er langt raskere å avgjøre om en relasjon er tom eller ikke
select count(distinct filmid)
from filmparticipation fp
where not exists (select f.filmid
                  from film f
                  where fp.filmid = f.filmid);
```

```
 count
--------
 455223
(1 row)
```

```
-- filmer i Film som ikke er i Filmparticipation
select count(distinct filmid)
from film f
where not exists (select f.filmid
                  from filmparticipation fp
                  where fp.filmid = f.filmid);
```

```
 count
--------
 212832
(1 row)
```

```
-- filmer med i begge
select count(distinct filmid)
from film
     natural join filmparticipation;
```

```
 count
--------
 479529
(1 row)
```

```
-- Filmer og rolle for personer med navnet Michael King
select personid, title, parttype
from          film f
natural join filmparticipation fp
natural join person p
where personid = any ( select personid
                       from Person
                       where lastname like 'King'
                       and firstname like 'Michael' );
```

```
 personid |          title           | parttype
----------+--------------------------+----------
   980485 | Mission of Mercy         | director
  1458440 | Misfit Patrol            | writer
  1458456 | Kiss Remembered, A       | writer
  1458456 | Mission of Mercy         | writer
  1458456 | Kiss Remembered, A       | producer
  1524887 | North End, The           | producer
  6089729 | Perfect Getaway, The     | cast
  6089745 | Witches of the Caribbean | cast
  6089793 | Creative Violence        | cast
  6089793 | Razor Eaters             | cast
  6089825 | Daughters of Discipline  | cast
  2418217 | Cold Front               | cast
  2418217 | Dan's Motel              | cast
  2418217 | Hero and the Terror      | cast
(14 rows)
```

```
-- Det samme for Ingmar Bergman, men bare for filmer med
-- ett-ords titler

select personid, title, parttype
```

```
from          film f
natural join  filmparticipation fp
natural join  person p
where title not like '%␣%'
and personid = any ( select personid
                        from Person
                        where lastname like 'Bergman'
                        and firstname like 'Ingmar' )
order by title;
```

```
 personid |       title       | parttype
----------+-------------------+----------
   155477 | Ansiktet          | writer
   155477 | Ansiktet          | director
   155477 | Backanterna       | director
   155477 | Beröringen        | producer
   155477 | Beröringen        | director
   155477 | Beröringen        | writer
   155477 | Bildmakarna       | director
   155477 | Eva               | writer
   155477 | Frånskild         | writer
   155477 | Fängelse          | writer
   155477 | Fängelse          | director
   155477 | Hamnstad          | director
   155477 | Hamnstad          | writer
   155477 | Hets              | writer
   155477 | Hets              | cast
   155477 | Hustruskolan      | director
   155477 | Höstsonaten       | director
   155477 | Höstsonaten       | writer
   155477 | Jungfrukällan     | producer
   155477 | Jungfrukällan     | director
   155477 | Kris              | writer
   155477 | Kris              | director
   155477 | Kvinnodröm        | writer
   155477 | Kvinnodröm        | director
   155477 | Kvinnodröm        | cast
   155477 | Lustgården        | writer
   155477 | Misantropen       | director
   155477 | Nattvardsgästerna | writer
   155477 | Nattvardsgästerna | director
   155477 | Oväder            | director
   155477 | Paradistorg       | producer
   155477 | Persona           | director
   155477 | Persona           | writer
   155477 | Persona           | producer
   155477 | Persona           | writer
   155477 | Rabies            | director
   155477 | Reservatet        | writer
   155477 | Riten             | director
   155477 | Riten             | writer
   155477 | Riten             | cast
   155477 | Saraband          | director
   155477 | Saraband          | writer
   155477 | Skammen           | director
   155477 | Skammen           | writer
   155477 | Smultronstället   | director
   155477 | Smultronstället   | writer
   155477 | Sommarlek         | director
   155477 | Sommarlek         | writer
   155477 | Stimulantia       | writer
   155477 | Stimulantia       | director
   155477 | Söndagsbarn       | writer
   155477 | Trollflöjten      | cast
   155477 | Trollflöjten      | director
   155477 | Trollflöjten      | writer
   155477 | Trolösa           | writer
   155477 | Trämålning        | writer
   155477 | Tystnaden         | writer
   155477 | Tystnaden         | director
```

```
   155477 | Törst            | cast
   155477 | Törst            | director
   155477 | Vargtimmen       | writer
   155477 | Vargtimmen       | director
   155477 | Venetianskan     | director
(63 rows)
```

```
-- Hvor mange filmer har karakter (rank) mindre enn 3
select rank,
       count(filmid) antall
from filmrating r natural join filmitem fi
where filmtype='C'
group by rank
having rank < 3.0
order by rank;
```

```
 rank | antall
------+--------
    1 |    413
  1.1 |    104
  1.2 |     84
  1.3 |    110
  1.4 |    118
  1.5 |    160
  1.6 |    157
  1.7 |    203
  1.8 |    225
  1.9 |    238
    2 |    289
  2.1 |    247
  2.2 |    276
  2.3 |    323
  2.4 |    322
  2.5 |    323
  2.6 |    604
  2.7 |    441
  2.8 |    548
  2.9 |    486
(20 rows)
```

```
-- Rating av Ingmar Bergmans kinofilmer:
select rank, fr.*
from          filmrating fr
natural join filmitem fi
natural join filmparticipation fp
where filmtype='C'
  and personid = any ( select personid
                       from Person
                       where lastname like 'Bergman'
                       and firstname like 'Ingmar' );
```

```
-- Antall filmer som har bedre rating enn samtlige
-- filmer Ingmar Bergman har vært involvert i
select count(distinct filmid) as antall
from          filmrating fr
natural join filmitem fi
natural join filmparticipation fp
where filmtype ='C'
  and rank > all (
```

```
            select rank
            from          filmrating fr
            natural join filmitem fi
            natural join filmparticipation fp
            where filmtype='C'
              and personid =  any ( select personid
                                    from Person
                                    where lastname like 'Bergman'
                                    and firstname like 'Ingmar' )
            ) ;
```

```
 antall
--------
   8279
(1 row)
```

```
-- Antall filmer som har bedre rating enn en av
-- filmene Ingmar Bergman har vært involvert i
select count(distinct filmid) as antall
from          filmrating fr
natural join filmitem fi
natural join filmparticipation fp
where filmtype ='C'
  and rank > any (
            select rank
            from          filmrating fr
            natural join filmitem fi
            natural join filmparticipation fp
            where filmtype='C'
            and personid  =  any ( select personid
                                    from Person
                                    where lastname like 'Bergman'
                                    and firstname like 'Ingmar' )
            ) ;
```

```
 antall
--------
  90501
(1 row)
```

Den siste spørringen svarer til antall filmer med bedre rating enn den dårligste Ingmar Bergman-filmen.

```
-- film med flest deltakere som ikke er i tabellen Film:

select filmid, count(*) as antdeltakere
from filmparticipation fp
where not exists (select filmid from film f
                  where f.filmid = fp.filmid)
group by filmid
order by antdeltakere desc
limit 10;
```

```
 filmid | antdeltakere
--------+--------------
  66961 |         1266
  27387 |          993
```

```
  24145 |         969
  77595 |         946
  25627 |         939
   7115 |         892
   3634 |         882
 165650 |         833
   5035 |         830
  33195 |         790
(10 rows)
```

```
-- det maksimale ant. deltakere i en film som ikke er i Film-tabellen
select max(ff.antdeltakere)
from (select filmid, count(*) as antdeltakere
      from filmparticipation fp
      where not exists (select filmid from film f
                        where f.filmid = fp.filmid)
      group by filmid) as ff
;
```

```
 max
------
 1266
(1 row)
```

```
-- hvem har spilt (cast) i flest kinofilmer ?
select personid,
       max(lastname) || ', ' || max(firstname) as navn,
       count(distinct filmid) as ant_filmer
from filmparticipation natural join filmitem natural join person
where parttype = 'cast'
      and filmtype = 'C'
group by personid
order by ant_filmer desc    limit 15;
```

```
 personid |      navn        | ant_filmer
----------+------------------+------------
   267736 | Blanc, Mel       |        846
  1487049 | Flowers, Bess    |        716
   872857 | Cobb, Edmund     |        606
  3830649 | Richardson, Jack |        597
  3597785 | Phelps, Lee      |        585
  2760825 | London, Tom      |        571
  3449625 | Osborne, Bud     |        564
   168501 | Bhasi, Adoor     |        554
  3231817 | Mower, Jack      |        516
  3300201 | Nazir, Prem      |        507
  1389589 | O'Connor, Frank  |        497
   422339 | Jeremy, Ron      |        496
   218329 | Bacon, Irving    |        485
  1334761 | Ellis, Frank     |        483
   457081 | Blystone, Stanley |       478
(15 rows)
```

```
-- Hvor mange av filmene Mel Blanc har spilt i, finnes ikke i Film?

select count(*) as uten_tittel
from filmparticipation natural join filmitem fi
     natural join person
where personid = 267736
      and parttype = 'cast'
```

```
      and filmtype = 'C'
and not exists ( select filmid from Film f
                 where fi.filmid = f.filmid )
;
```

```
 uten_tittel
-------------
           0
(1 row)
```

*-- Titler på kinofilmer Mel Blanc har spilt i*

```
select title as filmerSomMelBlancHarSpiltI
from           filmparticipation
   natural join filmitem
   natural join person
   natural join film
where personid = 267736
   and parttype = 'cast'
   and filmtype = 'C' ;
```

```
 ...
 Yankee Dood It
 Yankee Doodle Bugs
 Yankee Doodle Daffy
 Year of the Mouse, The
 You Ought to Be in Pictures
 You Were Never Duckier
 Zip 'N Snort
 Zip Zip Hooray!
 Zipping Along
 Zoom and Bored
 Zoom at the Top
(846 rows)
```

## Sammenskjøting av tabeller—join

- Intuitivt å skjøte sammen to relasjoner
  $R \underset{C}{\bowtie} S$
- Vi kan tenke slik (jf. relasjonsalgebra):
  - 1. Beregn $R \times S$
  - 2. Velg ut de tuplene som tilfredsstiller joinbetingelsen $C$

**Bistro**

| bn | mkat |
|---|---|
| A | kosher |
| A | vegetabilsk |
| B | uten melk |
| B | hallal |
| B | glutenfri |
| B | kosher |
| C | glutenfri |
| C | hallal |
| C | kosher |
| D | vegetabilsk |

**Krav**

| navn | mkat |
|---|---|
| Ali | hallal |
| Liv | kosher |
| Lise | kosher |
| Geir | glutenfri |

**Bistro** $\underset{C}{\bowtie}$ **Krav**

| bn | Bistro.mkat | navn | Krav.mkat |
|---|---|---|---|
| B | hallal | Ali | hallal |
| C | hallal | Ali | hallal |
| A | kosher | Liv | kosher |
| B | kosher | Liv | kosher |
| C | kosher | Liv | kosher |
| A | kosher | Lise | kosher |
| B | kosher | Lise | kosher |
| C | kosher | Lise | kosher |
| B | glutenfri | Geir | glutenfri |
| C | glutenfri | Geir | glutenfri |

$C$: Bistro.mkat = Krav.mkat

## Naturlig join

- $R \bowtie S$er relasjonen som fås fra $R$ og $S$ ved å danne alle mulige sammensmeltinger av ett tuppel fra $R$ med ett fra $S$ der tuplene skal stemme overens i samtlige attributter med sammenfallende navn
  - Fellesattributtene forekommer bare én gang i de sammensmeltede attributtene
  - Resultatskjemaet har attributtene i $R$ etterfulgt av de attributtene i $S$ som ikke også forekommer i $R$

## Naturlig join

**Bistro**

| bn | mkat |
|---|---|
| A | kosher |
| A | vegetabilsk |
| B | uten melk |
| B | hallal |
| B | glutenfri |
| B | kosher |
| C | glutenfri |
| C | hallal |
| C | kosher |
| D | vegetabilsk |

**Krav**

| navn | mkat |
|---|---|
| Ali | hallal |
| Liv | kosher |
| Lise | kosher |
| Geir | glutenfri |

**Bistro⋈Krav**

| bn | mkat | navn |
|---|---|---|
| B | hallal | Ali |
| C | hallal | Ali |
| A | kosher | Liv |
| B | kosher | Liv |
| C | kosher | Liv |
| A | kosher | Lise |
| B | kosher | Lise |
| C | kosher | Lise |
| B | glutenfri | Geir |
| C | glutenfri | Geir |

Når vi skal studere de forskjellige måtene å sammenføye tabeller på, bruker vi denne eksempeldatabasen:

```
select * from Stud;
```

```
 id  |         navn
-----+----------------------
 sss | Siv Sande Smil
 aaa | Ali Ali Ahmed
 bbb | Berit Brur Breiesen
 ccc | Chris C. Carr
 ddd | Dina Dorthea Dahl
 eee | En Som Ikke Tar Kurs
(6 rows)
```

```
select * from Kurs;
```

```
  kkode  |                   knavn
---------+-------------------------------------------
 INF1000 | Grunnkurs i objektorientert programmering
 INF1400 | Digital teknologi
 INF1080 | Logiske metoder for informatikk
 INF1500 | Introduksjon til design, bruk, interaksjon
 INF1010 | Objektorientert programmering
 INF1300 | Introduksjon til databaser
 INF2220 | Algoritmer og datastrukturer
(7 rows)
```

```
select * from KursStud;
```

```
 id  |  kkode
-----+---------
 aaa | INF1400
 aaa | INF1080
 bbb | INF1300
 bbb | INF1060
 bbb | INF2220
 ccc | INF2220
 ccc | INF1060
 ccc | INF1300
 ddd | INF3100
 fff | INF1010
 ggg | INF1000
(11 rows)
```

Det kartesiske produktet mellom alle tre inneholder 462 tupler. Hva med naturlig join?

```
select * from KursStud natural join Kurs;
```

```
 kkode  | id  |                  knavn
--------+-----+-----------------------------------------
 INF1000 | ggg | Grunnkurs i objektorientert programmering
 INF1400 | aaa | Digital teknologi
 INF1080 | aaa | Logiske metoder for informatikk
 INF1010 | fff | Objektorientert programmering
 INF1300 | ccc | Introduksjon til databaser
 INF1300 | bbb | Introduksjon til databaser
 INF2220 | ccc | Algoritmer og datastrukturer
 INF2220 | bbb | Algoritmer og datastrukturer
(8 rows)
```

```
select * from KursStud natural join Kurs natural join Stud;
```

```
 id  |  kkode  |             knavn              |        navn
-----+---------+--------------------------------+---------------------
 aaa | INF1400 | Digital teknologi              | Ali Ali Ahmed
 aaa | INF1080 | Logiske metoder for informatikk | Ali Ali Ahmed
 ccc | INF1300 | Introduksjon til databaser      | Chris C. Carr
 bbb | INF1300 | Introduksjon til databaser      | Berit Brur Breiesen
 ccc | INF2220 | Algoritmer og datastrukturer    | Chris C. Carr
 bbb | INF2220 | Algoritmer og datastrukturer    | Berit Brur Breiesen
(6 rows)
```

Vi ser vi mister informasjon der joinattributtene ikke har en match i den andre relasjonen.

Noen eksempler på natural join i filmdatabasen:

```
select title, count(distinct film.filmid)
from Film, Filmitem
where
  -- seleksjonsbetingelse
     filmtype = 'C'
  -- joinbetingelse:
     and film.filmid = filmitem.filmid
group by title
having count(distinct film.filmid) > 25
order by count(distinct film.filmid) desc;
```

*-- er det samme som:*

```
select title, count(distinct film.filmid)
from film natural join filmitem
where filmtype = 'C'
group by title
having count(distinct filmid) > 25
order by count(distinct filmid) desc;
```

```
       title        | count
--------------------+-------
 Popular Science    |    45
 Love               |    42
 Mother             |    41
 Hamlet             |    39
 Desire             |    37
 Stranger, The      |    37
 Unusual Occupations |    35
 Trap, The          |    34
 Carmen             |    34
 Home               |    33
 Destiny            |    31
```

```
 Jealousy           |   31
 Honeymoon          |   29
 Cinderella         |   29
 Alone              |   29
 Revenge            |   29
 Dead End           |   28
 Kiss, The          |   28
 Escape             |   27
 Temptation         |   27
 Fear               |   26
 Awakening, The     |   26
(22 rows)
```

En spørring uten, og så den samme med naturlig sammenskjøting:

```
select p.personid,
       max(lastname) || ', ' || max(firstname) as navn,
       count(distinct f.filmid) as antfilmer,
       min(rank) as minimumsvurdering
from film f, filmrating r,
     filmparticipation x, person p,
     filmcountry c
where      r.filmid = f.filmid and rank is not null
      and p.personid = x.personid
      and f.filmid = x.filmid and parttype = 'cast'
      and c.filmid = f.filmid
      and country not like 'India'
group by p.personid
having count(distinct f.filmid) > 20
       and min(rank) > ( 1 + (select avg(rank)
                              from filmitem i, film f,
                              filmrating r
                              where      i.filmid = f.filmid
                              and r.filmid = f.filmid
                              and rank is not null
                             )
                         )
;
```

er da det samme som:

```
select p.personid,
       max(lastname) || ', ' || max(firstname) as navn,
       count(distinct fp.filmid) as antfilmer,
       min(rank) as minimumsvurdering
from filmrating r natural join filmparticipation fp
     natural join person p natural join filmcountry c
where rank is not null
      and parttype = 'cast'
      and country not like 'India'
group by p.personid
having count(distinct fp.filmid) > 20
       and min(rank) > ( 1 + (select avg(rank)
                              from filmitem natural join filmrating
                              where rank is not null
                             )
                         )
;
```

Sammenlign med svaret fra samme spørring lenger opp uten join:

```
 personid |        navn         | antfilmer | minimumsvurdering
----------+---------------------+-----------+-------------------
   286025 | Barsi, Béla         |        22 |               6.7
   711113 | Caramitru, Ion      |        23 |               6.7
   750345 | Casey, Daniel       |        27 |               7.1
   948697 | Cotescu, Octavian   |        23 |               7.6
  1443881 | Ferruzca, Homero    |        22 |               9.2
  1540385 | Buckley, Ivan       |        22 |               7.1
  1790776 | Maschio, Robert     |        23 |               7.6
  1897625 | Hara, Setsuko       |        21 |               6.6
  2038865 | Cheng, Pablo        |        23 |               9.2
  3143273 | Mishima, Masao      |        22 |               6.8
  3340953 | Nieto, Evelyn       |        23 |               8.2
  3654681 | Popandov, Pavel     |        23 |               6.7
  3999497 | Sakamoto, Maaya     |        21 |               6.6
  4838105 | Waterhouse, Matthew |        39 |               6.7
  5230273 | Horta, Jose María   |        22 |               9.2
  6111490 | Saldívar, Ana Elena |        22 |               9.2
  7000178 | Torres, Barbara     |        22 |               9.2
  9428513 | Pérez, Sammy        |        29 |               8.8
 12933153 | Zuñiga, Edson       |        22 |               9.2
(19 rows)
```

Finn navn på regissører som har resgissert filmer som er kategorisert i mer enn 5 filmsjangre. (Denne løste vi over under operatoren **in** slik):

```
select P.lastname, P.firstname
from filmparticipation FP
     natural join filmitem FI
     natural join Person P
where FI.filmtype = 'C' and
      FP.parttype = 'director'
      AND FP.filmid in ( select FG.filmid
                         from filmgenre FG
                         group by FG.filmid
                         having count(*) > 7 );
```

Denne naturlige joinen gir samme tabell:

```
select P.lastname, P.firstname
from filmparticipation FP
     natural join filmitem FI
     natural join Person P
     natural join  ( select FG.filmid
                         from filmgenre FG
                         group by FG.filmid
                         having count(*) > 7 ) f5
where FI.filmtype = 'C' and
      FP.parttype = 'director';
```

Her er seleksjonsbetingelsen gjort om til en joinbetingelse (likhet i attributtet filmid), men resultattabellen blir den samme. Men ikke tuplenes rekkefølge, nødvendigvis.

## Hengetupler

Når vi joiner to tabeller, kaller vi et tuppel som ikke har noen match i den andre relasjonen, et *hengetuppel*

Hengetupler blir ikke med i resultatet av en (vanlig) join, også kalt en **inner join**

Hvis vi ønsker å gjøre en join hvor vi beholder hengetuplene fra en eller begge tabellene, bruker vi en **outer join**. Da fylles de resterende attributtene med null.

**Bistro**

| bn | mkat |
|---|---|
| A | kosher |
| A | vegetabilsk |
| B | uten melk |
| B | hallal |
| B | glutenfri |
| B | kosher |
| C | glutenfri |
| C | hallal |
| C | kosher |
| D | vegetabilsk |

**Krav**

| navn | mkat |
|---|---|
| Ali | hallal |
| Liv | kosher |
| Lise | kosher |
| Geir | glutenfri |

Hengetupler satt med grå typer i Bistro-relasjonen.

**Bistro⋈Krav**

| bn | mkat | navn |
|---|---|---|
| B | hallal | Ali |
| C | hallal | Ali |
| A | kosher | Liv |
| B | kosher | Liv |
| C | kosher | Liv |
| A | kosher | Lise |
| B | kosher | Lise |
| C | kosher | Lise |
| B | glutenfri | Geir |
| C | glutenfri | Geir |

I eksemplet med kurs og studenter er det også hengetupler. Ta tabellene Kurs, og KursStud, f.eks.

```
  kkode  |                   knavn
---------+-------------------------------------------
 INF1000 | Grunnkurs i objektorientert programmering
 INF1400 | Digital teknologi
 INF1080 | Logiske metoder for informatikk
 INF1500 | Introduksjon til design, bruk, interaksjon
 INF1010 | Objektorientert programmering
 INF1300 | Introduksjon til databaser
 INF2220 | Algoritmer og datastrukturer
(7 rows)
```

```
 id  |  kkode
-----+---------
 aaa | INF1400
 aaa | INF1080
 bbb | INF1300
 bbb | INF1060
 bbb | INF2220
 ccc | INF2220
 ccc | INF1060
 ccc | INF1300
 ddd | INF3100
 fff | INF1010
 ggg | INF1000
(11 rows)
```

```
select * from KursStud natural join Kurs;
```

```
  kkode  | id  |                   knavn
---------+-----+-------------------------------------------
 INF1000 | ggg | Grunnkurs i objektorientert programmering
 INF1400 | aaa | Digital teknologi
```

```
 INF1080 | aaa | Logiske metoder for informatikk
 INF1010 | fff | Objektorientert programmering
 INF1300 | ccc | Introduksjon til databaser
 INF1300 | bbb | Introduksjon til databaser
 INF2220 | ccc | Algoritmer og datastrukturer
 INF2220 | bbb | Algoritmer og datastrukturer
(8 rows)
```

I sammenføyningstabellen er ikke informasjon om INF1500 med. Det samme gjelder informasjonen om studenter (bbb og ccc) som tar INF1060, og om ddd som tar INF3100. Tuplene som innholder utelatt informasjon er *hengetupler*. Vil vi ha med noe av denne informasjonen, kan vi bruke såkalt outer join. Denne sammenskjøtingsoperatoren kommer i tre varianter, avhengig av om vi vil ha med hengetupler fra den ene (i eksempelet KursStud), fra den andre (Kurs) eller fra begge.

Vil vi ha med hengetupler fra relasjonen som står foran joinoperatoren (til venstre) bruker vi en left outer join.

## left outer join

- Syntaks for en **left outer join** er slik:

```
select <svar-attributter>
from   tabell-1 left outer join tabell-2
                on join-betingelse;
```

- Resultatet blir en join av tabell-1 og tabell-2, pluss en linje for hvert hengetuppel i tabell-1 der alle svar-attributtene fra tabell-2 er **null**
- Eventuelle hengetupler fra tabell-2 blir ikke med i resultatet

```
select * from KursStud ks left outer join Kurs k
              on ks.kkode = k.kkode;
```

```
 id  |  kkode  |  kkode  |                  knavn
-----+---------+---------+-----------------------------------------
 ggg | INF1000 | INF1000 | Grunnkurs i objektorientert programmering
 aaa | INF1400 | INF1400 | Digital teknologi
 aaa | INF1080 | INF1080 | Logiske metoder for informatikk
 fff | INF1010 | INF1010 | Objektorientert programmering
 ccc | INF1300 | INF1300 | Introduksjon til databaser
 bbb | INF1300 | INF1300 | Introduksjon til databaser
 ccc | INF2220 | INF2220 | Algoritmer og datastrukturer
 bbb | INF2220 | INF2220 | Algoritmer og datastrukturer
 ddd | INF3100 |         |
 ccc | INF1060 |         |
 bbb | INF1060 |         |
(11 rows)
```

Her har vi fått med hengetuplene fra KursStud (tabellen til venstre for operatoren), men ikke fra Kurs.

Eksempler fra filmadatabasen:

```
select count(distinct f.filmid)
from film f
     left outer join filmparticipation p
     on f.filmid = p.filmid;
```

```
 count
--------
 692361
(1 row)
```

```
-- antall filmer som kun er i Film
select count(f.filmid)
from film f
     left outer join filmparticipation p
     on f.filmid = p.filmid
        where p.filmid is NULL;
```

```
 count
--------
 212832
(1 row)
```

```
select count(x.*)
from (  (select filmid from film)
     except
      (select distinct filmid from filmparticipation)  )
    as x ;
```

```
 count
--------
 212832
(1 row)
```

except er mengdedifferanse, jf. lysark om relasjonsalgebra.

Vil vi ha med hengetupler fra relasjonen som står bak joinoperatoren (til høyre) bruker vi en right outer join.

## Right outer join

- Syntaks for en **right outer join** er slik:

```
select <svar-attributter>
from   tabell-1 right outer join tabell-2
                on join-betingelse;
```

- Resultatet blir en join av tabell-1 og tabell-2, pluss en linje for hvert hengetuppel i tabell-2 der alle svar-attributtene fra tabell-1 er **null**
- Eventuelle hengetupler fra tabell-1 blir ikke med i resultatet

```
select * from KursStud ks right outer join Kurs k
              on ks.kkode = k.kkode;
```

```
 id  |  kkode  |  kkode  |                  knavn
-----+---------+---------+-------------------------------------------
 ggg | INF1000 | INF1000 | Grunnkurs i objektorientert programmering
 aaa | INF1400 | INF1400 | Digital teknologi
 aaa | INF1080 | INF1080 | Logiske metoder for informatikk
     |         | INF1500 | Introduksjon til design, bruk, interaksjon
 fff | INF1010 | INF1010 | Objektorientert programmering
 ccc | INF1300 | INF1300 | Introduksjon til databaser
 bbb | INF1300 | INF1300 | Introduksjon til databaser
 ccc | INF2220 | INF2220 | Algoritmer og datastrukturer
 bbb | INF2220 | INF2220 | Algoritmer og datastrukturer
(9 rows)
```

Vi konstaterer at vi får med hengetuplet fra den høyre relasjonen, INF1500.

Eksempler fra filmadatabasen:

```
select count(distinct p.filmid)
from film f
     right outer join filmparticipation p
     on f.filmid = p.filmid;
```

*— pass på kvalifikasjonen bak distinct. Ikke likegyldig om p. eller f.*

```
 count
--------
 934752
(1 row)
```

```
— antall filmer som kun er i Filmparticipation
select count(distinct p.filmid)
from film f
     right outer join filmparticipation p
     on f.filmid = p.filmid
        where f.filmid IS NULL;
```

```
 count
--------
 455223
(1 row)
```

Vil vi ha med hengetupler fra begge relasjonene, både de fra den til venstre og den til høyre for joinoperatoren, bruker vi en full outer join.

## Full outer join

Syntaks for en **full outer join** er slik:

```
select <svar-attributter>
from   tabell-1 full outer join tabell-2
                on join-betingelse;
```

Resultatet blir en join av tabell-1 og tabell-2, pluss en linje for hvert hengetuppel i tabell-1 der alle svar-attributtene fra tabell-2 er **null** og en linje for hvert hengetuppel i tabell-2 der alle svar-attributtene fra tabell-1 er **null**.

```
select * from KursStud ks full outer join Kurs k
              on ks.kkode = k.kkode;
```

```
 id  |  kkode  |  kkode  |                  knavn
-----+---------+---------+-----------------------------------------
 ggg | INF1000 | INF1000 | Grunnkurs i objektorientert programmering
 aaa | INF1400 | INF1400 | Digital teknologi
 aaa | INF1080 | INF1080 | Logiske metoder for informatikk
     |         | INF1500 | Introduksjon til design, bruk, interaksjon
 fff | INF1010 | INF1010 | Objektorientert programmering
 ccc | INF1300 | INF1300 | Introduksjon til databaser
 bbb | INF1300 | INF1300 | Introduksjon til databaser
 ccc | INF2220 | INF2220 | Algoritmer og datastrukturer
 bbb | INF2220 | INF2220 | Algoritmer og datastrukturer
 ddd | INF3100 |         |
 ccc | INF1060 |         |
 bbb | INF1060 |         |
(12 rows)
```

En full outer join gir oss hengetuplene fra begge tabellene.

Eksempler fra filmadatabasen:

```
select count(distinct p.filmid) + count(distinct f.filmid) as antall
from film f
     full outer join filmparticipation p
     on f.filmid = p.filmid;
```

```
 antall
---------
 1627113
(1 row)
```

Dette er for mange, vi har fått med snittet to ganger, så trekk fra filmer med i begge

```
select count(distinct filmid)
from film natural join filmparticipation;
```

```
 count
--------
 479529
(1 row)
```

```
-- antall filmid med i en av tabellene, men ikke i begge

select count(distinct p.filmid) + count(distinct f.filmid) as antall
from film f
     full outer join filmparticipation p
     on f.filmid = p.filmid
        where f.filmid is null
        or    p.filmid is null;
```

```
 antall
--------
 668055
(1 row)
```

## Mengdeoperatorer

De vanlige operatorene på mengder, union, snitt, mengdesubstraksjon, kartesisk produkt (multiplikasjon) og divisjon finnes i SQL. Se lysarkene om relasjonsalgebra. Her et eksempel på bruk av dem for å beregne den samme mengden som i siste eksempel over.

```
select count(distinct x.*)
from ( (
          ( select filmid from film )
        union
          ( select distinct filmid from filmparticipation )
       )
     except
       (    ( select distinct filmid from film )
          intersect
            (  select distinct filmid from filmparticipation )
       )
     )
as  x ;
```

```
 count
--------
 668055
(1 row)
```

Svaret fra en select-setning vet vi alltid er en tabell. Vi kan se på tuplene i tabellen som elementene i en mengde. Men siden ett og samme tuppel kan forekomme flere ganger i en slik tabell, kaller vi ikke dette en mengde, men en *multimengde* eller *bag*. De tilsvarende mengdeoperatorene på multimengder som også gir en multimengde som svar, heter **union all, intersect all** og **except all**. Noen eksempler på forskjellen:

UNION [ ALL ]

```
select * from Tab;
```

```
 tekst  | tall
--------+------
 London |    4
 London |    4
 Paris  |    3
 Paris  |    3
 Paris  |    4
 Roma   |    1
 Roma   |    2
 Roma   |    3
 Roma   |    3
 Berlin |    5
(10 rows)
```

```
( select * from Tab ) union ( select * from Tab ) ;
```

```
  tekst  | tall
--------+------
 Roma   |    2
 Roma   |    3
 London |    4
 Paris  |    3
 Roma   |    1
 Paris  |    4
 Berlin |    5
(7 rows)
```

```
( select * from Tab ) union all ( select * from Tab ) ;
```

```
  tekst  | tall
--------+------
 London |    4
 London |    4
 Paris  |    3
 Paris  |    3
 Paris  |    4
 Roma   |    1
 Roma   |    2
 Roma   |    3
 Roma   |    3
 Berlin |    5
 London |    4
 London |    4
 Paris  |    3
 Paris  |    3
 Paris  |    4
 Roma   |    1
 Roma   |    2
 Roma   |    3
 Roma   |    3
 Berlin |    5
(20 rows)
```

INTERSECT [ ALL ]

```
select * from Tab where tall = 3;
```

```
  tekst | tall
-------+------
 Paris |    3
 Paris |    3
 Roma  |    3
 Roma  |    3
(4 rows)
```

```
( select * from Tab ) intersect ( select * from Tab where tall = 3) ;
```

```
  tekst | tall
-------+------
 Paris |    3
 Roma  |    3
(2 rows)
```

```
( select * from Tab ) intersect all ( select * from Tab where tall = 3) ;
```

```
  tekst | tall
-------+------
 Paris |    3
 Paris |    3
 Roma  |    3
 Roma  |    3
(4 rows)
```

Legg speiselt merke til at mengdesnittet eller mengdeunionen av en tabell med seg selv, svarer til select distinct på den samme tabellen. **intersect all** (multimengdesnittet) svarer til select * på den samme tabellen:

```
select distinct * from Tab
```

gir samme tabell som

```
( select * from Tab ) intersect ( select * from Tab ) ;
```

og som

```
( select * from Tab ) union ( select * from Tab ) ;
```

Dette fordi distinct fjerner flerforekomster og derfor gjør en multimengde om til en mengde.

EXCEPT [ ALL ]

```
select * from Tab;
```

```
 tekst  | tall
--------+------
 London |    4
 London |    4
 Paris  |    3
 Paris  |    3
 Paris  |    4
 Roma   |    1
 Roma   |    2
 Roma   |    3
 Roma   |    3
 Berlin |    5
(10 rows)
```

```
( select * from Tab where tekst in ('Roma','Paris'));
```

```
 tekst | tall
-------+------
 Paris |    3
 Paris |    3
 Paris |    4
 Roma  |    1
 Roma  |    2
 Roma  |    3
 Roma  |    3
(7 rows)
```

```
(select * from Tab) except (select * from Tab where tekst in ('Roma','Paris'));
```

```
  tekst  | tall
--------+------
 London |    4
 Berlin |    5
(2 rows)
```

```
(select * from Tab) except all (select * from Tab where tekst in ('Roma','Paris'));
```

```
  tekst  | tall
--------+------
 London |    4
 London |    4
 Berlin |    5
(3 rows)
```

# Tilleggsstoff—ikke pensumrelevant

## Hvordan uttale «SQL»?

- På norsk uttaler vi SQL bokstav for bokstav: «ess-ku-ell»
- På engelsk uttales SQL «'si:kwel»
  Årsaken er historisk:
  - SQL ble utviklet av IBM, og prototypen het SEQUEL—et akronym for «Structured English QUEry Language»
  - Da SEQUEL ble lansert som et produkt i 1976, ble navnet forkortet til SQL, men uttalen ble beholdt og har overlevd til nå

## SQLs SDL: indekser

**create index** $X$ on $R(A_1, \ldots, A_k)$;

**drop index** $X$;

Valg av indekser må gjøres med omhu.
Indekser gjør at

- spørringer mot vedkommende attributt(er) går mye fortere
- innsetting, sletting og oppdatering blir mer komplisert

## Indekser på kandidatnøkler

- DBMSet bygger indekser automatisk på primærnøklene
- For hver kandidatnøkkel må man vurdere spesielt om det bør deklareres indeks på nøkkelen. Syntaks avhenger av DBMSet Noen SQL-implementasjoner tillater deklarasjon av kandidatnøkkel + indeks i en og samme setning:

```
create unique index FnrIndex
on Ansatt (Fdato, Pnr);
```

- **I Postgres bygges automatisk en unique index på kandidatnøkler**
- Hvis det er opprettet indeks på en nøkkel, benyttes denne under sjekk av flerforekomster. Ellers: Må i verste fall søke gjennom hele relasjonen.