Skjema

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

- *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'   -- seleksjonsbetingelse
      and (     T.AId = A.AId -- joinbetingelse 1
           and P.PId = T.PId -- joinbetingelse 2
          );
```

- Her består join-betingelsen av to ledd. Den binder sammen en forekomst fra hver av de tre tabellene Ansatt, Timeliste og Prosjekt

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

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

```
select distinct Navn, Tittel
from            Ansatt A
           join Timeliste T on T.AId = A.AId -- joinb. 1
 natural join Prosjekt P                     -- joinb. 2
where      Pnavn = 'Vintersalg' ; -- seleksjonsbetingelse
```

- Her består join-betingelsen av to ledd. Den binder sammen en forekomst fra hver av de tre tabellene Ansatt, Timeliste og Prosjekt

Skjema

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

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

```
select distinct Navn, Tittel
from            Ansatt A
     natural join Timeliste T  -- joinb. 1
     natural join Prosjekt P   -- joinb. 2
where     Pnavn = 'Vintersalg' ; -- seleksjonsbetingelse
```

- Her består join-betingelsen av to ledd. Den binder sammen en forekomst fra hver av de tre tabellene Ansatt, Timeliste og Prosjekt

SQLs DML (Data manipulation language

- **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

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';
```

- Merk at det *ikke* er lov å ha regneuttrykk som parameter i min() og max()

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)

- 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 resultatattributt-listen.
- Merk: Attributter som *ikke* er grupperingsattributter, må være funksjonelt avhengige av grupperingsattributtene.

Eksempel

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

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

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

Dette forutsetter at a-navn er en kandidatnøkkel.
Hvis ikke måtte vi hatt **group by** avdnr.
I så fall må vi muligens ta med avdnr i select-listen

Eksempel med eksplisitt join

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

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

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

Dette forutsetter at a-navn er en kandidatnøkkel.
Hvis ikke måtte vi hatt **group by** avdnr.
I så fall må vi muligens ta med avdnr i select-listen

Eksempel

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

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

```
select    pnr as prosjekt, avdnr as avdnummer,
          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, a-navn
order by  pnr, sum(timer) desc;
```

Siste linje kan erstattes med

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

Eksempel

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

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

```
select    pnr as prosjekt, avdnr as avdnummer,
          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, a-navn
having    sum(timer) > 99
order by  prosjekt, innsats desc;
```

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

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):

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

```
select    pnr as prosjekt, avdnr as avdnummer,
          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, a-navn
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

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 distinc**
7. Sorter i henhold til **order by**