



inf

INF1300—Det meste av resten av SQL

Utleddsark v. 1.0



UNIVERSITETET
I OSLO

Dagens temaer

- ▶ Sammenligning med tekstmønstre
- ▶ Aggregeringsfunksjoner
- ▶ Nestede spørsmål
- ▶ Gruppering
- ▶ Relasjonssammenligninger:
 - ▶ exists
 - ▶ in
 - ▶ any
 - ▶ all
- ▶ View
- ▶ JDBC

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 navn som begynner
med «O» og slutter med «a», som Ola,
Olga, Othilia, Oda, Ofjhwsjkfhkxxa

Tekstmønstre

► Eksempel 3:

```
select firstname || ' ' || lastname as navn,  
       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 én blank

Aggregeringsfunksjoner

SQL har fem aggregeringsfunksjoner:

<i>navn</i>	<i>virkning (returnerer)</i>
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, lønn, avd). Finn den største lønnsforskjellen i salgsavdelingen:

```
select max(lønn) - min(lønn)
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, lønn, avd)`. Finn sum lønnsutgifter og gjennomsnittslønn for salgsavdelingen:

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


Nestede spørsmål

- ▶ Gitt tabellen Ansatt(anr, navn, lønn, avd)
Finn antall selgere som tjener *mer enn det dobbelte* av markedsførernes gjennomsnittslønn

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

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

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, lønn, avd)

Avdeling(avdnr, a-navn, leder)

Prosjektplan(pnr, anr, timer)

```
select  a-navn, count(*), avg(lønn)
from    Ansatt, Avdeling
where   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, lønn, avd)

Avdeling(avdnr, a-navn, leder)

Prosjektplan(pnr, anr, timer)

```
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;
```

Eksempel

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

Ansatt(anr, navn, lønn, avd)

Avdeling(avdnr, a-navn, leder)

Prosjektplan(pnr, anr, timer)

```
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;
```

Samme oppgave, men ta bare med avdelinger som har minst 10 ansatte

Ansatt(anr, navn, lønn, avd)

Avdeling(avdnr, a-navn, leder)

Prosjektplan(pnr, anr, timer)

```
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 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. Seleker 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**

Relasjonssammenligninger

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

<i>i SQL-2</i>	<i>betyr</i>
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
- Eksempel:
Finn antall selgere som tjener mer enn samtlige markedsførere

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

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

IN og NOT IN

- [not] in kan brukes på ett attributt eller på en liste av attributter
- Eksempel:
Finn navn på ansatte som ikke har ført noen prosjekttimer

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

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

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

- ▶ $F \Rightarrow G \equiv \text{not } F \text{ or } G$
- ▶ $\text{not } (F \text{ and } G) \equiv \text{not } F \text{ or } \text{not } G$
- ▶ $\text{not } (F \text{ or } G) \equiv \text{not } F \text{ and } \text{not } G$
- ▶ $\forall u.F \equiv \text{not } (\exists u.\text{not } F)$
- ▶ $\exists u.F \equiv \text{not } (\forall u.\text{not } F)$

Eksempel

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

Ansatt(anr, navn, lønn, avd)

Avdeling(avdnr, a-navn, leder)

Prosjektplan(pnr, anr, timer)

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

Eksempel

Finn navn på ansatte som skal delta på alle prosjekter

Ansatt(anr, navn, lønn, avd)

Avdeling(avdnr, a-navn, leder)

Prosjektplan(pnr, anr, timer)

```
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));
```


Eksempel

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

Ansatt(anr, navn, lønn, avd)

Avdeling(avdnr, a-navn, leder)

Prosjekt(pnr, leder) Prosjektplan(pnr, anr, timer)

```
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

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

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**

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

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

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**

JDBC (Java Data Base Connectivity)

- Tre nødvendige java-biblioteker: :

```
import java.sql.*; import java.util.*; import java.io.*;
```

- To hjelpemetoder: :

```
static void warn( String msg, Exception e ) {  
    System.err.println( msg );  
    if ( e != null ) {  
        System.err.println( "Fikk unntak: " + e.toString() );  
        System.err.println( "Melding: " + e.getMessage() );  
        System.err.println( "Traceback: " );  
        e.printStackTrace( System.err );  
    }  
}
```

```
static void die( String msg, Exception e ) {  
    warn( msg, e );  
    System.exit( 1 );  
}
```

JDBC

- Metode for å opprette forbindelse med filmdatabasen:

```
public static Connection openConnection() {  
    Connection con = null;  
    try { Class.forName("org.postgresql.Driver"); }  
    catch(Exception e) {  
        die("Fant ingen JDBC-driver", e);  
    }  
    Properties p = new Properties();  
    p.put("user", "<ditt Postgres-brukeravn>");  
    p.put("password", "<ditt Postgres-passord>");  
    String url = "jdbc:postgresql://kurspg/fdb";  
    try { con = DriverManager.getConnection( url, p ); }  
    catch(Exception e) {  
        die("Feil DB-URL, brukeravn eller passord", null);  
    }  
    return con;  
}
```

JDBC

- ▶ I hovedprogrammet (f.eks. i main) opprettes kontakten med filmdatabasen slik:

```
Connection con = openConnection();
```

- ▶ Deretter kan man legge SQL-koden i en String sqlkode, opprette to objektpekere

```
Statement stm = null;  
ResultSet svar = null;
```

```
stm = con.createStatement();  
svar = stm.executeQuery(sqlkode);
```

JDBC-eksempel

Ansatt(anr, navn, lønn, avd)
Avdeling(avdnr, a-navn, leder)

```
Connection con = openConnection();  
Statement stm = con.createStatement();  
ResultSet svar =  
    stm.executeQuery("select a-navn, count(*) as ant,  
                      avg(lønn) as gjsnitt  
                      from   Ansatt, Avdeling  
                      where  avd = avdnr  
                      group  by a-navn");
```

```
String a-navn; int ant; float gjsnitt;  
while (svar.next()) {  
    a-navn = svar.getString("a-navn");  
    ant = svar.getInt("ant");  
    gjsnitt = svar.getFloat("gjsnitt");  
    System.out.println("RAD = " + a-navn + " "  
                       + ant + " " + gjsnitt);  
}
```