# Økt 5 (av 12)

**DB1100 Databaser**

**( Tomas Sandnes / tomas.sandnes@kristiania.no )**

# Dagens temaer

Dagens tema: Avanserte spørringer.

- Dagens pensum: Læreboka, kapittel 5.

- Status arbeidskrav

- Fra forrige økt: Resultater Kahoot

- Nytt innhold: Avanserte spørringer
  - Ny SQL i denne sammenheng: `VIEW` og subqueries.

# Status arbeidskrav

- Alle dere som *ikke* leverte arbeidskrav og ikke er interessert i videomappe:
  - Se bort fra denne sliden. :-)

- Frist for innlevering av 1. arbeidskrav er forbi.
  - 31 stykker leverte videoer, og kan derfor jobbe videre med arbeidskrav og har muligheten til å velge videomappe til eksamen.
  - Dere 31 får tidlig i neste uke invitasjon av meg til fellesmøte rett før neste forelesning (altså om 1 uke) der jeg går gjennom arbeidskravprosessen videre.

- NB: Har du levert video, men ikke fått en mail av meg som bekrefter at jeg har mottatt den?
  - Gi meg beskjed asap om når du leverte, så skal jeg følge det opp!

# Resultat, forrige Kahoot

- Forrige gang måtte vi se oss slått av fjorårets studenter.
  - De klarte 67,5 % riktig svar.

- Dere: 63,4 % riktig svar!
  - Bra innsats, gi dere selv noen anerkjennende nikk!

| DB1100 - økt 4 | |
|---|---|
| Played on | 13 Sep 2018 |
| Hosted by | Tomas_Sandnes |
| Played with | 137 players |
| Played | 10 of 10 questions |

| Overall Performance | |
|---|---|
| Total correct answers (%) | 63,40% |
| Total incorrect answers (%) | 36,60% |
| Average score (points) | 5876,90 points |

# Lærdom fra Kahoot

Spm. 1: Vi har to tabeller R og S som begge har 2 kolonner og 3 rader. Det kartesiske produktet R X S?

A) 2 kolonner, 6 rader.

B) 4 kolonner, 9 rader.

C) 4 kolonner, 6 rader.

D) 2 kolonner, 9 rader.

*Plusse kolonner, gange rader:*

*2 + 2 kolonner = 4 kolonner.*
*3 * 3 rader = 9 rader.*

*Se også læreboka, s. 91! :-)*

# Lærdom fra Kahoot, #2

Spm. 8: Fungerer ikke i MySQL: SELECT a FROM b FULL OUTER JOIN c; Hvorfor ikke?

A) MySQL støtter ikke FULL OUTER JOIN.
B) Du kan ikke kalle en kolonne for "a".
C) Du kan ikke kalle en tabell for "b" eller "c".
D) Det er ingenting i SQL som heter FULL OUTER JOIN.

*FULL OUTER JOIN vil si at kolonnen har innhold i enten venstre eller høyre tabell (men ikke alltid venstre, ikke alltid høyre, og ikke nødvendigvis begge samtidig).*

*MySQL støtter ikke dette, selv om det er en del av SQL syntaksen.*

*Se evt. læreboka, s. 103-104 ("Full ytre kobling").*

# View

**Ligner veldig på en tabell...**

# Hva er et View?

- Et View er en forhåndslaget spørring fra en eller flere tabeller. Eksempel:

| SNR | NAVN | GATE | STED | STILLING | MLOENN | ANSDATO | FNR |
|---|---|---|---|---|---|---|---|
| 3 | JON | BRUVEIEN 7 | STAVANGER | LEDER | 35000 | 08-SEP-95 | 1 |
| 2 | MARIE | STRILEGATEN 8 | BERGEN | LEDER | 30000 | 01-JAN-95 | 2 |
| 1 | SUSANNE | SKRAPLODDVEIEN 62 | OSLO | LEDER | 45000 | 01-JUL-94 | 3 |
| 20 | OLAV | GALMANNSVEIEN 4 | STAVANGER | SENIORMEGLER | 26000 | 07-JUL-97 | 1 |
| 5 | DAVID | GULERLEVEIEN 43 | STAVANGER | SEKRETÆR | 18000 | 14-JUN-95 | 1 |
| 4 | ANNE | STRANDGATEN 5 | [illegible]VANGER | MEGLER | 12000 | 12-DEC-96 | 1 |
| 8 | GUSTAV | NORDLYSVEIEN 78 | | | | 01-JAN-96 | 1 |
| 9 | OLAVA | LOMVIVEIEN 57 | | | | 01-JAN-98 | 1 |
| 11 | LEONORA | RØDVEIEN 6 | OSL[illegible] | | | 01-JUL-94 | 3 |
| 10 | TEODOR | TULIPAN 12 | OSL[illegible] | | | 30-MAY-97 | 3 |
| 6 | JONNAS | KIRKEVEIEN 7 | BÆRU[illegible] | | | 19-APR-96 | 3 |
| 12 | TULLA | BLÅVEIEN 7a | OSL[illegible] | | | 09-SEP-95 | 3 |
| 18 | FREDRIK | LASSOVEIEN 37 | OSL[illegible] | | | 01-JUL-94 | 3 |
| 7 | KARL | OLAVSGATE 7 | OSL[illegible] | | | 10-SEP-95 | 3 |
| 16 | SMUKKA | GRAUTSTIEN 43 | OSL[illegible] | | | 01-JUL-94 | 3 |
| 17 | KARL | BLÅVEISVEIEN 7 | OSLO | MEGLER | 20500 | 12-MAY-96 | 3 |

| SNR | NAVN | GATE |
|---|---|---|
| 3 | JON | BRUVEIEN 7 |
| 20 | OLAV | GALMANNSVEIEN 4 |
| 5 | DAVID | GULERLEVEIEN 43 |
| 4 | ANNE | STRANDGATEN 5 |
| 8 | GUSTAV | NORDLYSVEIEN 78 |
| 9 | OLAVA | LOMVIVEIEN 57 |

# Hvorfor bruke View?

- Sikkerhet: Begrense datatilgangen i databasen.

- Redusere kompleksitet:
  - Gjøre komplekse spørringer enklere.
  - Redusere datamengden for bruker.

- Tilpasning: Oppnå forskjellige syn (views) av datagrunnlaget. (Brukergrupper/applikasjoner)

# Ulemper med View

- Mer kompleksitet knyttet til oppdateringer:
  - Det er begrensninger på hvordan man kan endre underliggende data gjennom et view.
  - Et views struktur blir bestemt når det opprettes, og vil ikke automatisk endre seg senere (VIEW basert på SELECT * FROM...)

- Ytelse:
  - Et view kan joine mange tabeller, og dermed være en relativt tung spørring.
  - Dette kommer ikke alltid tydelig frem for brukeren.

# Definere et View

```
CREATE [OR REPLACE] VIEW navn
        [(alias [,alias ] . . .)]
AS subquery;
```

- OR REPLACE — Overskriver view om det allerede eksisterer
- alias — Kolonnenavn
- subquery — En fullstendig SELECT setning

- Merk! Hvis du angir ett kolonnenavn, så må du oppgi alle.

# Eksempel 1: world databasen

```
CREATE OR REPLACE VIEW EuropeCountry_view
AS SELECT Code, Name, Population
FROM country
WHERE Continent = 'Europe';
```

```
SELECT *
FROM EuropeCountry_view;
```

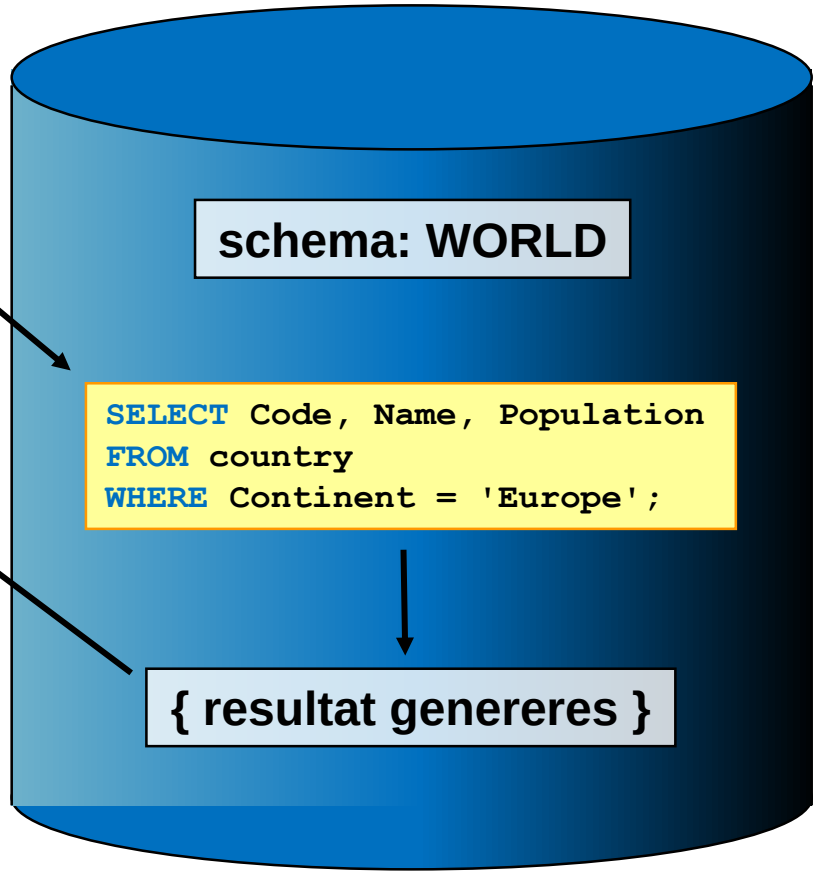
```
Code Name                      Population
---- ----------------------- ----------
ALB  Albania                   3401200
AND  Andorra                   78000
AUT  Austria                   8091800
BEL  Belgium                   10239000
BGR  Bulgaria                  8190900
BIH  Bosnia and Herzegovina    3972000
BLR  Belarus                   10236000
CHE  Switzerland               7160400
CZE  Czech Republic            10278100
DEU  Germany                   82164700
DNK  Denmark                   5330000
ESP  Spain                     39441700
...
```

- Et view behandles på samme måte som en tabell.

# Slik fungerer spørring mot view

```
SELECT *
FROM EuropeCountry_view;
```

```
Code Name            Population
---- ------------    ----------
ALB  Albania         3401200
AND  Andorra         78000
AUT  Austria         8091800
BEL  Belgium         10239000
BGR  Bulgaria        8190900
...
```




- View definisjonen (ikke resultatet!) er lagret i databasen.

# Eksempel 1: noen varianter

- Vi kan gi view'et egne kolonnenavn:

```
CREATE OR REPLACE VIEW EuropeCountry_view
AS SELECT Code AS ID, Name AS Country, Population
FROM country
WHERE Continent = 'Europe';
```

```
CREATE OR REPLACE VIEW EuropeCountry_view
(ID, Country, Population)
AS SELECT Code, Name, Population
FROM country
WHERE Continent = 'Europe';
```

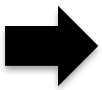
```
SELECT *
FROM EuropeCountry_view;
```

| ID | Country | Population |
|---|---|---|
| ALB | Albania | 3401200 |
| AND | Andorra | 78000 |
| AUT | Austria | 8091800 |
| BEL | Belgium | 10239000 |
| BGR | Bulgaria | 8190900 |
| BIH | Bosnia and Herzegovina | 3972000 |
| BLR | Belarus | 10236000 |
| CHE | Switzerland | 7160400 |
| CZE | Czech Republic | 10278100 |
| DEU | Germany | 82164700 |
| DNK | Denmark | 5330000 |
| ESP | Spain | 39441700 |
| ... | | |

# Summering av kolonner i view

```
CREATE OR REPLACE VIEW ContinentPopulation_view
AS SELECT Continent, SUM(Population) AS Population
   FROM country
   GROUP BY Continent
   ORDER BY Population ASC;
```

```
SELECT *
FROM ContinentPopulation_view;
```



| Continent | Population |
|---|---|
| Antarctica | 0 |
| Oceania | 30401150 |
| South America | 345780000 |
| North America | 482993000 |
| Europe | 730074600 |
| Africa | 784475000 |
| Asia | 3705025700 |

- Gruppefunksjoner kan inngå i et view.
  - Det er en fordel å bruke kolonne-alias.

# Oppgave

- Lag et view som viser alle byer i verden som har en befolkning større enn 5 millioner. Viewet skal vise bynavn, befolkning og hvilket land byen ligger i (navnet på landet), og være sortert synkende på innbyggertall.

```
CREATE OR REPLACE VIEW largeCities
(city, population, country)
AS
SELECT ci.name, ci.population, co.name
FROM city ci LEFT JOIN country co
ON ci.countryCode = co.Code
WHERE ci.Population > 5000000
ORDER BY population DESC;
```

```
SELECT * FROM largeCities LIMIT 7;
```

| city | population | country |
|---|---|---|
| Mumbai (Bombay) | 10500000 | India |
| Seoul | 9981619 | South Korea |
| SÃ£o Paulo | 9968485 | Brazil |
| Shanghai | 9696300 | China |
| Jakarta | 9604900 | Indonesia |
| Karachi | 9269265 | Pakistan |
| Istanbul | 8787958 | Turkey |

# Tips ved CREATE View

- Kjør alltid SELECT først, så vet du om viewet ditt fungerer før du oppretter det.

```
CREATE OR REPLACE VIEW largeCities
(city, population, country)
AS SELECT ci.name, ci.population, co.name
FROM city ci LEFT JOIN country co ON ci.countryCode = co.Code
WHERE ci.Population>5000000
ORDER BY population DESC;
```

# Oppdatering via view

- Et view kan benyttes til å oppdatere data i underliggende tabell(er).

- NB: ISO restriksjoner på hvordan et view er laget m.t.p. oppdateringer. Bl.a.:
  - View'et kan bare referere til én tabell.
  - `DISTINCT` kan ikke være del av view'et.
  - Alle elementer i view'ets select del må være kolonner (ikke konstanter, summeringer, etc. ...)
  - Ingen `GROUP BY` eller `HAVING`.
  - Rad som blir lagt til må følge integritetsreglene for underliggende tabell (not null, etc.).

# Oppdatering via view – forts.

- Merk at et view oppdaterer dataene i tabellen!
  (Ikke bare data for view'et selv.)

```
UPDATE EuropeCountry_view
SET ID = 'A_Z'
WHERE Country = 'Austria';
```

```
SELECT ID, Country
FROM EuropeCountry_view;
```

```
ID         Country
----       ---------
ALB        Albania
AND        Andorra
A_Z        Austria
BEL        Belgium
BGR        Bulgaria
...
```

```
SELECT Code, Name, Population
FROM country
WHERE Name = 'Austria';
```

```
CODE Name       Population
---- --------- ----------
A_Z  Austria   8091800
```

# Definere et View – del 2.

```
CREATE [OR REPLACE] VIEW navn
        [(alias [,alias ] . . .)]
AS subquery
[WITH CHECK OPTION]
```

- WITH CHECK OPTION

  Spesifiserer at rader som et view kan aksessere ikke kan endres gjennom dette view'et om det resulterer i at de forsvinner fra view'et.

# View med check option

- Check option gjør at vi ikke kan oppdatere en rad slik at den forsvinner ut av viewet:

```
CREATE OR REPLACE VIEW EuropeCountry_view
AS SELECT Code, Name, Continent
   FROM country
   WHERE Continent = 'Europe'
WITH CHECK OPTION;
```

```
SELECT *
FROM EuropeCountry_view;
```

| Code | Name | Continent |
|---|---|---|
| ALB | Albania | Europe |
| AND | Andorra | Europe |
| A_Z | Austria | Europe |
| BEL | Belgium | Europe |
| BGR | Bulgaria | Europe |
| ... | | |

```
UPDATE EuropeCountry_view
SET Continent = 'Asia'
WHERE Name = 'Austria';
```

```
Error Code: 1369. CHECK OPTION failed
'world.europecountry_view'
```

# Slette et view

- Syntaks for å slette et view er nesten som for tabell:
  (bytt ut table med view)

```
DROP VIEW EuropeCountry_view;
```

# Materialized views

- I steden for å kun lagre et view som en spørring, og kjøre spørringen når man kjører viewet, så kan vi materialisere viewet vårt.
  - Å materialisere et view betyr å lagre resultatet av spørringen i en egen tabell, slik at vi kan hente dette innholdet når vi kjører viewet.
  - Dette vil være aktuelt når spørringen bak viewet tar lang tid, og underliggende data sjelden endrer seg.

- MySQL tilbyr dessverre ikke slik funksjonalitet.
  - Vi må ved bruk av MySQL lage det selv.

# Delspørringer (sub-queries)

**Spørringer i spørringer...**

# Subqueries

- Som nevnt, er resultatet av en SELECT formatert som en ny tabell:
  - Det danner kolonner og rader på samme måte som databasens eksisterende tabeller.

- Derfor er det ikke noe problem å bruke resultatet av en SELECT som et element i en annen!

- Å putte en SELECT inne i en annen på denne måten kalles en subquery.

# Subqueries II

- Noen ganger trenger vi svaret fra den ene spørringen før vi kan begynne på den andre.

- Eks.: Ønsker å finne hvor mange byer som har et innbyggertall over eller likt gjennomsnittet.
  - Før vi kan fullføre denne spørringen, må vi vite hva gjennomsnittet er!

# Subqueries III

- Løser det med en subquery, på denne måten:
  - Oppgave: «hvor mange byer har et innbyggertall over eller likt gjennomsnittet.»

```
select count(*)
from City
where Population >=
(select avg(Population) from City);
```

- Eller hva med: "Ønsker å se prosentandelen som innbyggerne i et land utgjør av jordas totale befolkning"? (<-Vanskelig)
  - Denne er blant dagens øvingsoppgaver! :-)

- Husk å lese pensumbokas beskrivelse av delspørringer (subqueries, kap. 5.4). Der vil du finne mange flere eksempler på bruk.

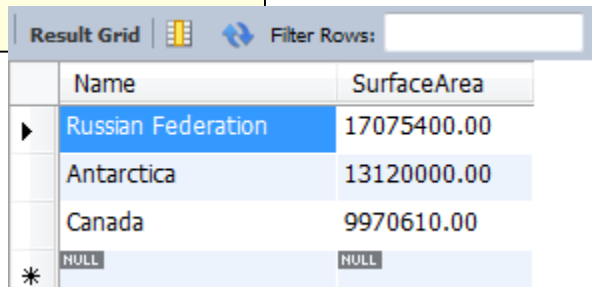
# Limit

- Begrenser antall rader i resultatet ditt.

- Eksempel:

  Jeg vil finne de tre største landene i verden.

```
SELECT Name, SurfaceArea
FROM Country
ORDER BY SurfaceArea DESC
LIMIT 3;
```



# Videre arbeid i dag (og kort om neste gang)

- Neste gang (torsdag om 1 uke), litt spesiell form:
  - "Vanlig" struktur, dvs. teori først og øving etterpå!
  - Pensum: Kapittel 6 – Relasjonsmodellen.
  - Pluss repetisjon av tidligere innhold (med meg).
  - En tidligere multiple choice eksamen er "øvingsoppgave". (Dere kan teste hvordan dere ligger an, resultatet teller *ikke* på karakteren.)

- NÅ: (om du ikke alt har gjort det)
  - Lese kap. 5 i pensumboka.
  - Se videoer til dagens økt.
  - Begynne på øvingsoppgaver!

- Etterpå:
  - Øving (flipped) 14:15 – 17:00, sjekk TimeEdit for rom.
    Teori (flipped) 17:15 – 18:00, auditoriet.