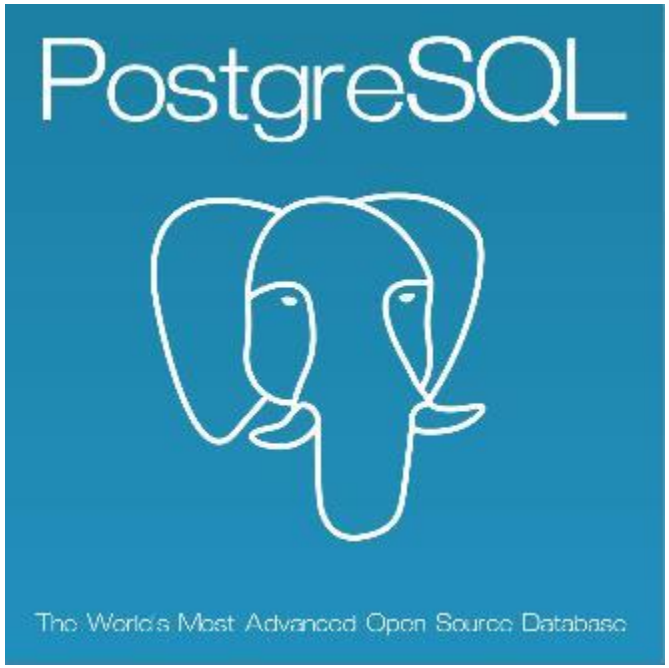
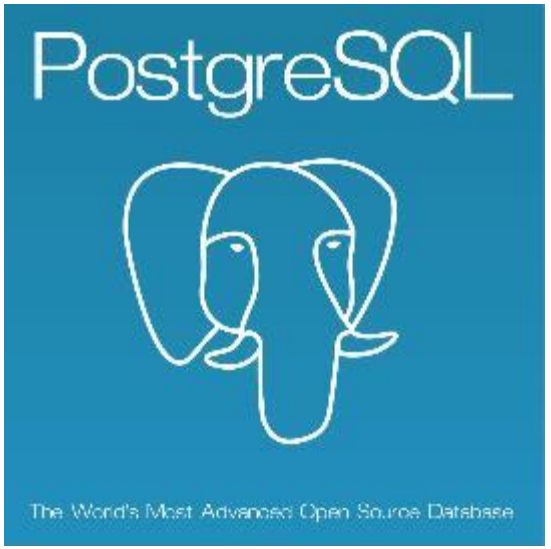
Bruke SQL fra Python

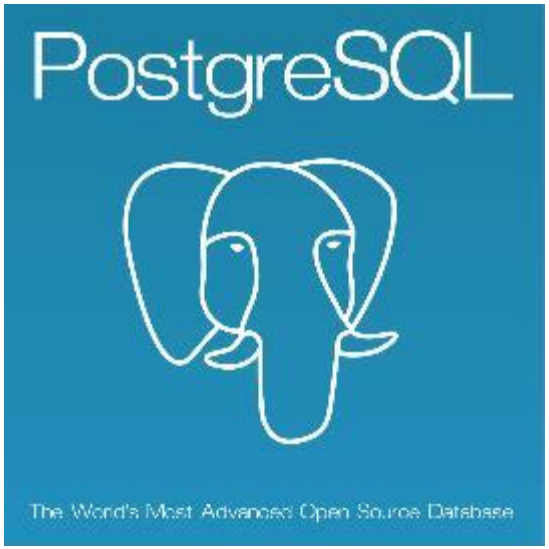
Med Psycopg2


python™


PostgreSQL
The World's Most Advanced Open Source Database

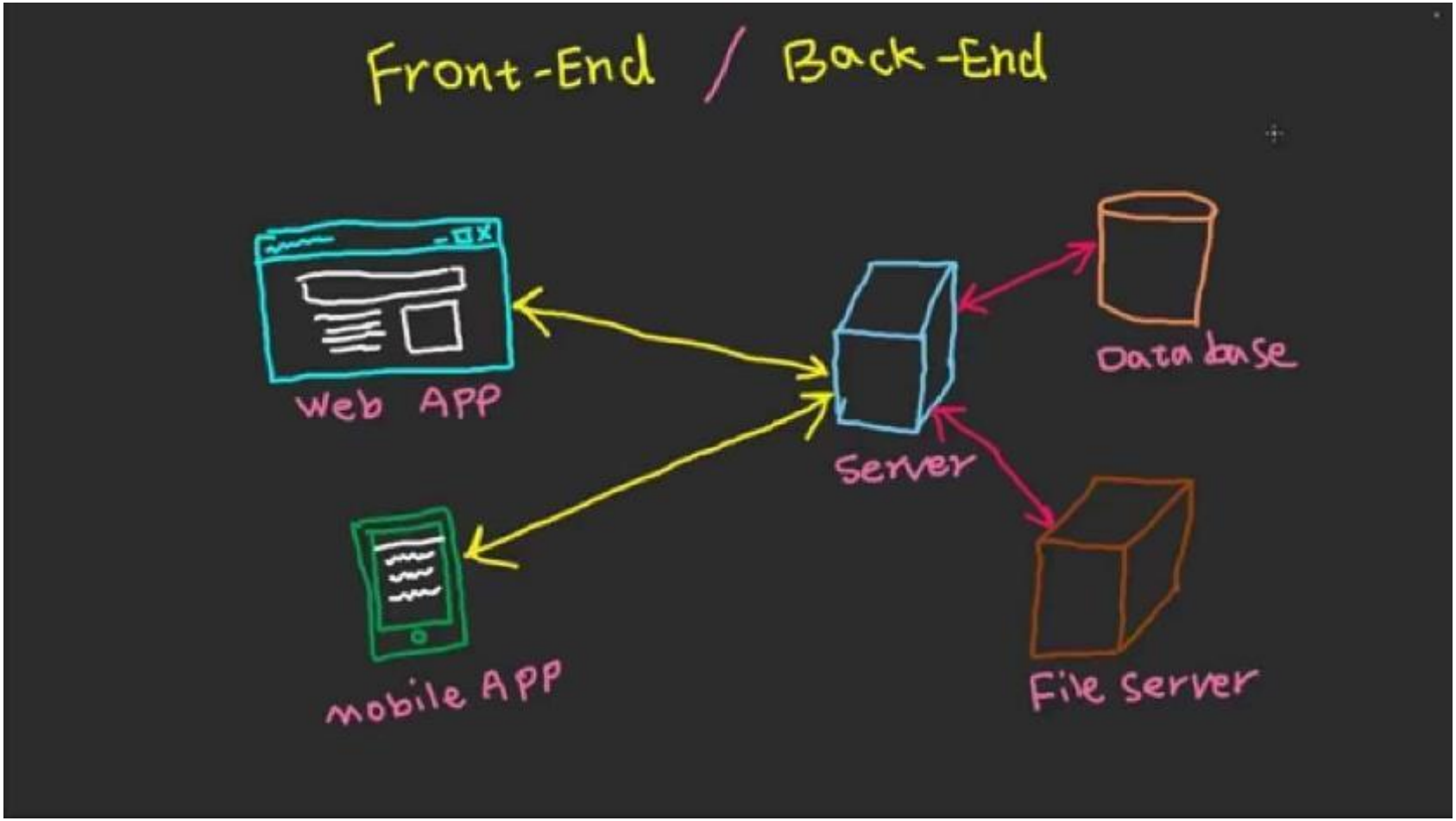

python™


PostgreSQL
The World's Most Advanced Open Source Database



Front-end

Back-end


Front-End / Back-End
Web App
Server
Database
mobile App
File Server

Pakken psycopg2

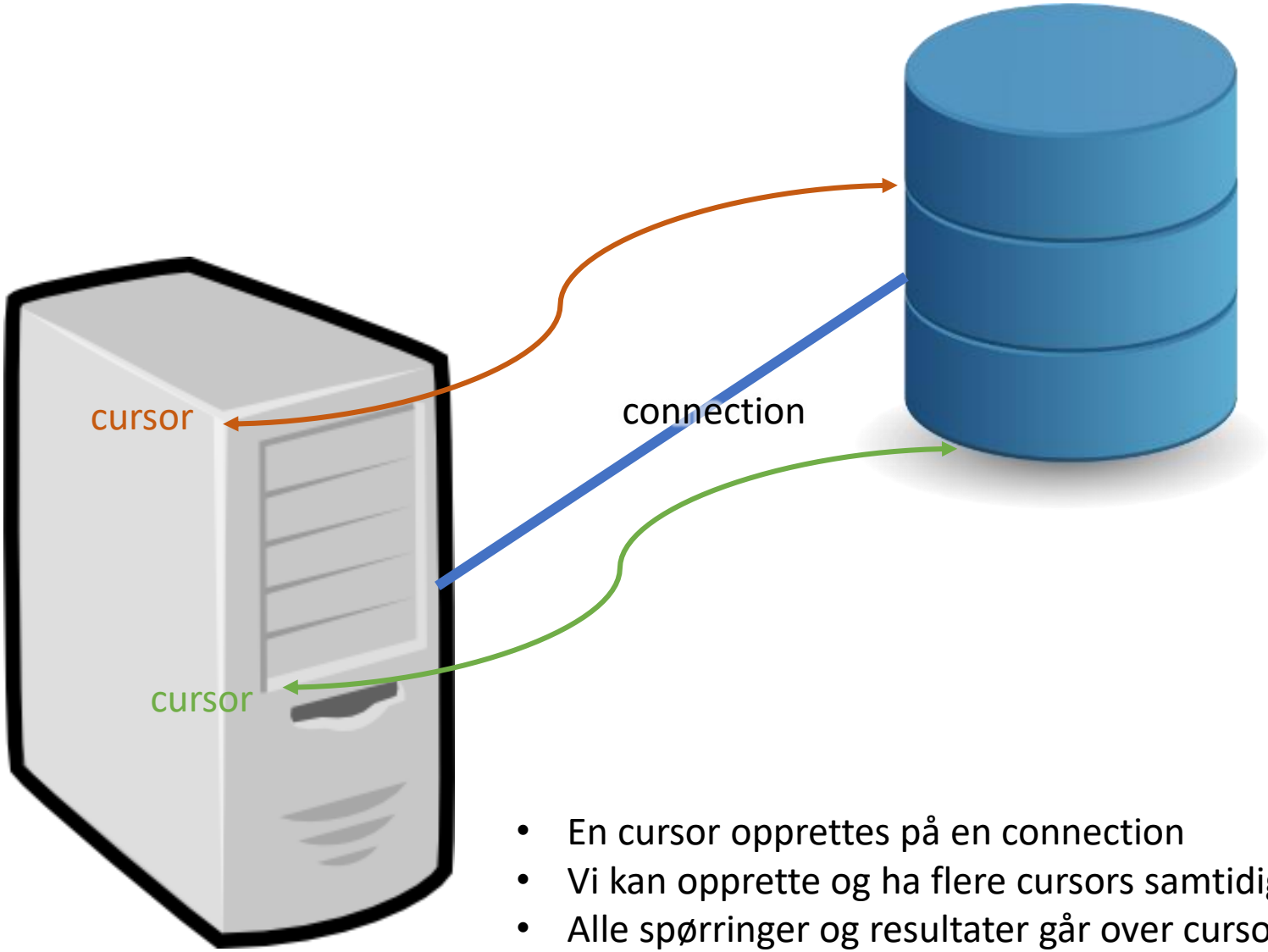
Pakken psycopg2

De viktigste klassene vi trenger i psycopg2:

- connection
  - Håndterer forbindelsen fra Python til PostgreSQL
- cursor
  - Lages fra en connection, brukes for å utføre spørringer over forbindelsen

Pakken kan installeres med `pip`:

```
pip install psycopg2-binary
```



- En cursor opprettes på en connection
- Vi kan opprette og ha flere cursors samtidig
- Alle spørringer og resultater går over cursor-objekter

Håndtere forbindelser mellom Python og PostgreSQL

Klassen `connection` lar oss håndtere forbindelser mellom Python og PostgreSQL.

Forbindelser oppretter vi ved å kalle på funksjonen `connect()`. Denne funksjonen returnerer et `connection`-objekt.

```
conn = psycopg2.connect(
        host="dbpg-ifi-kurs.uio.no",
        dbname="fdb",
        user="brukernavn",
        password="hemmelig")
```

Kjøre SQL-kommandoer: cursor-klassen

Når vi har laget en forbindelse, kan vi bruke forbindelsen for å lage et cursor-objekt. Dette gjør vi ved å kalle på metoden `cursor()` på connection-objektet:

```
conn = psycopg2.connect(…)
cur = conn.cursor()
```

- Det er gjennom cursor-objektet at vi utfører spørringer.
- Man kan lage mange cursor-objekter.
- Alle cursor-objektene er bundet til connection-objektet.

Utføre spørringer: execute

Når vi har en cursor, kan vi bruke denne for å utføre en spørring.

```
conn = psycopg2.connect(…)
cur = conn.cursor()
cur.execute("SELECT title, prodyear FROM film")
```

Du kan bruke «multiline strings» for å skrive lengre spørringer:

```
sporring = """SELECT f.filmid, f.title, f.prodyear
                FROM film f natural join filmcountry c
                WHERE c.country = 'Norway' """
cur.execute(sporring)
```

Hente resultatene (1/3)

Etter å ha kjørt `execute(…)`, kan vi hente resultatene med metoden `fetchall()`

```
conn = psycopg2.connect(…)
cur = conn.cursor()
cur.execute(sporring)
norskeFilmer = cur.fetchall()
```

`fetchall()` returnerer en liste av tupler. Som alle andre lister, kan vi iterere over denne med en for-løkke:

```
for rad in norskeFilmer:
        print(rad)
```

→

```
(230, 'Varis', 2004)
(340, 'Anolit', 2002)
(356, 'Kvinnen i mitt liv', 2003)
(632, 'Syx', 1988)
(664, 'Portrettet', 1954)
(774, '22', 2000)
(792, 'Andre omgang', 2007)
(998, 'Digre daier', 1997)
(1014, 'Stopp', 2001)
...
```

Hente resultatene (2/3)

For å hente enkeltverdier fra hvert tuppel, bruker vi indeksen til kolonnen:

```
cur.execute("""SELECT f.filmid, f.title, f.prodyear
                FROM … """)
norskeFilmer = cur.fetchall()

for rad in norskeFilmer:
        tittel = rad[1]
        prodyear = rad[2]
        print("%s ble produsert i %i" % (tittel, prodyear))
        # fra Python 3.6 kan vi også skrive:
        print(f"{tittel} ble produsert i {prodyear}")
```

→

```
Varis ble produsert i 2004
Anolit ble produsert i 2002
Kvinnen i mitt liv ble produsert i 2003
Syx ble produsert i 1988
Portrettet ble produsert i 1954
22 ble produsert i 2000
...
```

Hente resultatene (3/3)

Det går også an å hente én rad ved å kalle på `fetchone()`.

Merk at det også går an å iterere direkte over `cursor`-objekter, så i stedet for å eksplisitt kalle på `fetchone()` i en løkke, kan `cursor`-objektet brukes slik:

```
cur.execute(sporring)
for rad in cur:
        print(rad)
```

Sette sammen til et lite program

```
import psycopg2

conn = psycopg2.connect(
        host="dbpg-ifi-kurs.uio.no",
        dbname="fdb",
        user="dittBrukernavn", # sett inn ditt brukernavn her
        password=passord)

cur = conn.cursor()
cur.execute("""SELECT f.filmid, f.title, f.prodyear
               FROM film f natural join filmcountry c
               WHERE c.country = 'Norway' limit 10""")

norskeFilmer = cur.fetchall()

for rad in norskeFilmer:
        tittel = rad[1]
        prodyear = rad[2]
        print(f"{tittel} ble produsert i {prodyear}")
```

Tips: Du kan lage et «passord-prompt» ved å bruke getpass.getpass().
Da slipper du å ha UiO-passordet ditt i kildekoden!

Parametriserte spørringer

- Hva om vi ønsker å bruke strings eller annen brukerinput?
- Det er ikke lurt å konkatenere disse på «vanlig» måte!

Om vi f.eks. har en string som inneholder følgende:

```
input = "O'boy"
```

Ser vi kanskje at dette kan føre til problemer i denne spørringen:

```
sporring = f"SELECT * FROM tabell WHERE name = '{input}' "
print(sporring)
```

→

```
SELECT * FROM tabell WHERE name = 'O'boy'
```

Riktig måte å sende med variabler inn i en spørring, er å sende med variablene til execute slik:

```
cur.execute("SELECT * FROM tabell WHERE name = %s",
            (input,))
```

Parametriserte spørringer

Vi kan selvfølgelig sende med flere verdier, også tall:

```
navn = "O'boy"
alder = 20
cur.execute("""SELECT * FROM tabell
                 WHERE name = %s AND age = %s""",
               (navn,alder))
```

Se også dokumentasjonen: Passing parameters to SQL queries

Nyttig SQL til oblig 6

- For datalagring snakker vi gjerne om **CRUD** – Create, Read, Update, Delete
- Create tilsvarer «`INSERT INTO`», Read tilsvarer «`SELECT`»

- I tillegg kan det være nyttig å vite om hvordan vi oppdaterer og sletter data i SQL:

- **`UPDATE`** – oppdaterer rader som tilfredsstiller en betingelse
  - ```
    UPDATE tabellnavn
    SET kolonne1 = verdi1, kolonne2 = verdi2, ...
    WHERE betingelse;
    ```
    f.eks.:
    ```
    UPDATE timeliste
    SET beskrivelse = 'Ny beskrivelse'
    WHERE timelistenr = 3;
    ```
- **`DELETE`** – sletter rader som tilfredsstiller en betingelse
  - ```
    DELETE FROM tabellnavn
    where betingelse
    ```
    f.eks.:
    ```
    DELETE FROM timeliste
    WHERE timelistenr = 8;
    ```

Annet nyttig til oblig 6

- **VIKTIG!** Om du endrer data (legger til, sletter, oppdaterer) fra psycopg, vil ikke endringene være permanente!
- Endringene vil fortsatt kunne sees så lenge `connection`-objektet eksisterer, men når objektet lukkes (eller programmet avsluttes), blir de «glemt»

- Det er derfor viktig at du *committer* alle endringer! Dette gjør du med metoden `connection.commit()`:

```
conn = psycopg2.connect(…)
cur = conn.cursor()
cur.execute("UPDATE tabell SET ...")
conn.commit()
```

Oppgave

Foreslå spørringer som det er vanskelig å gjøre med select-setningen, hvor Java eller Python ville ha vært til (stor) hjelp for å få riktig svar.

Har du opplevd oppgaver som du skulle løse med SQL hvor du har savnet ‘verktøy’ fra Python/Java eller andre programmeringsspråk?

```
create table Gruppelærer (
        brnavn varchar(8),
        år          int,
        vh          varchar(4)
);
```

```
create table Gruppelærer (
      brnavn varchar(8),
      år         int,
      vh         varchar(4)
);
```

```
 brnavn |  år  | vh
--------+------+-----
 mjstang | 2018 | vår
 mjstang | 2017 | høst
```

Representasjon i Python: Liste av tupler?

```
[('mjstang', 2018, 'vår'), ('mjstang', 2017, 'høst')]
```

http://heim.ifi.uio.no/inf1300/oppgaver/tidlEksmOppg/2011-2.pdf

```
create table Gruppelærer (
        brnavn varchar(8),
        år          int,
        vh          varchar(4)
);
```

For hver gruppelærer, finn det lengste antall semestere vedkommende har vært gruppelærer sammenhengende. F.eks. vil en som har vært gruppelærer våren 2008, våren 2009, høsten 2009, våren 2010, våren 2011 og høsten 2011 ha tre sammenhengende semestere på det meste (våren 2009, høsten 2009 og våren 2010) på grunn av brudd høsten 2008 og høsten 2010.

```
create view kodeSem as
      ( select brnavn, år*2 as sk
        from Gruppelærer
              where vh = 'vår' )
   union
      ( select brnavn, år*2 + 1 as sk
        from Gruppelærer
              where vh = 'høst' ) ;

create view diffSem as
      select k2.brnavn, k2.sk, (k2.sk - k1.sk) as diff
      from kodeSem k1, kodeSem k2
      where k1.brnavn = k2.brnavn and
           k2.sk >= k1.sk ;

create view diffHull as
      select d1.brnavn, d1.sk , d1.diff
      from diffSem d1
      where d1.diff+1 not in ( select d2.diff
                                    from diffSem d2
                                          where d2.brnavn = d1.brnavn and
                                          d2.sk = d1.sk and
                                          d2.diff > d1.diff ) ;

select R.brnavn, max(antsmh) as makssammenhengende
from ( select brnavn, sk, ( min(diff) + 1 ) as antsmh
      from diffHull
      group by brnavn, sk ) as R
group by R.brnavn ;
```

```
create table Gruppelærer (
      brnavn varchar(8),
      år         int,
      vh         varchar(4)
);
```