# IN2090 – Databaser og datamodellering

## 12 – Sikkerhet i programmer som bruker databaser

Leif Harald Karlsen
leifhka@ifi.uio.no


Universitetet i Oslo

# Databasetilkoblinger

- `Connection`-objektene (og de tilhørende `cursor` eller `Statement` og `ResultSet`) er ressurser som kan føre til minnelekasjer
- Alle disse har en egen metode som heter `close()` som stenger ressursen og frigjør den
- For de enkle programmene vi så sist uke var dette ikke veldig viktig, ettersom ressursene blir frigjort når programmet avsluttes
- Men for større programmer og tjenester som skal kjøre over lengre tid bør man alltid sørge for at tilkoblingene blir stengt med en gang man er ferdige med dem

# Stenge tilkoblinger

- I Python har både `Connection` og `Cursor` en `close()` metode
- I Java har både `Connection`, `Statement`/`PreparedStatement` og `ResultSet` egne `close()` metoder
- Generelt blir alle objekter stengt når objektet det er laget fra stenges
- F.eks. vil en `PreparedStatement` stenges dersom dens `Connection` stenges

# Automatisk stenge ressurser

- Java kan bruke `try-with`-blokker (fom. Java 7) for automatisk å stenge tilkoblinger
- F.eks.:

```
try (Connection con = DriverManager.getConnection(conStr);
    PreparedStatement stmt = con.prepareStatement(query)) {
    try (ResultSet res = stmt.executeQuery()) {
        // Do stuff with the result set.
    }
} catch (...) {
    // errors
} // con, stmt, res closed here
```

- Har tilsvarende i Python:

```
with psycopg2.connect(dsn) as conn:
    with conn.cursor() as cur:
        cur.execute(sql)
```

men dette stenger ikke tilkoblingen (kun eventuelle åpne transaksjoner)

# SQL injections

- SQL injection er en type angrep mot en klient/backend hvor en (ondsinnet) bruker får kjørt egen SQL-kode på databasen
- Brukeren kan da forbigå autentisering eller hente ut, endre eller slette data brukeren egentlig ikke skal ha tilgang til
- SQL injection utnytter følgende faktorer:
  - At vi har et program som kjører SQL-spørringer (f.eks. Java eller Python)
  - Programmet tar input fra brukeren som skal være en del av spørringene
  - Programmet ikke skiller mellom data og kommandoer i spørringen

# SQL injections: Grunnleggende prinsipp

- La oss si at en nettbutikk lar brukere søke etter varer på følgende måte:

```
product = input("Product: ");
cur.execute("SELECT * FROM products WHERE navn = '" + product + "';");
```

- Med input `Socks` blir spørringen:

```
SELECT * FROM products WHERE navn = 'Socks';
```

- Med input `O'Boy` får vi error:

```
SELECT * FROM products WHERE navn = 'O'Boy';
```

- Med input `Socks'; DROP TABLE products;--` får vi

```
SELECT * FROM products WHERE navn = 'Socks'; DROP TABLE products; --';
```

# SQL injections: Webshop-eksempel

Live-kode-eksempel!

# Hvorfor er dette viktig?

- En av de vanligste formene for hackerangrep
- Dersom angrepet lykkes vil det gi den ondsinnede brukeren veldig mye makt:
  - Ødelegge/slette data
  - Endre data
  - Hente ut konfidensiell informasjon
  - Hindre tjenesten i å fungere (DOS)
- Veldig enkelt å forhindre med parametriserte spørringer!

# Helhetlig sikkerthet

- Merk at selvom klienten skulle være sårbar for SQL injection-angrep kan databasen likevel forhindre mye skade om man har satt riktige rettigheter
- F.eks. dersom databasebrukeren som klienten benytter ikke har tilgang til å slette eller endre tabeller eller spørre mot vilkårlige tabeller
- Dette viser hvor viktig det er at alle ledd er sikret og at man ikke bør anta for mye av andre komponenter i et system

# Takk for nå!

Neste uke handler om indekser og spørreprossesering.