Web- og serverprogrammering

Databasekommunikation - dag 7

Strategier til databaseadgang
JDBC (Java DataBase Connectivity)
Evt: JDBC RowSet
Login-eksempel: Javabønne med JDBC

JPA - Java Persistence API

Læsning: WJSP 5, WJSP 9.5-9.7

JDBC – databaseadgang

Indlæse driveren

Med Java under Windows følger en standard JDBC-ODBC-bro med, så man kan kontakte alle datakilder, der er defineret under ODBC:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Er det en anden database, skal man have en jar-fil med en driver fra producenten. Nyeste drivere kan findes på http://java.sun.com/jdbc/

Driver til en Oracle-database (hedder typisk classes12.zip):

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Driver til en MySQL-database (hentes på http://mysql.com):

```
Class.forName("com.mysql.jdbc.Driver")
```

Etablere forbindelsen

Herefter kan man oprette forbindelsen med (for en ODBC-driver):

```
Connection forb = DriverManager.getConnection("jdbc:odbc:datakilde1");
```

Datakildens navn (her "datakilde1") skal være defineret i Windows.

Oracle-database:

```
Connection forb = DriverManager.getConnection(
  "jdbc:oracle:thin:@ora.javabog.dk:1521:student","jacob","jacob");
```

MySQL-database:

```
DriverManager.getConnection("jdbc:mysql:///jacob","root","xyz");
```

Databasedrivere

JDBC-drivere findes i fire typer:

- Type 1: JDBC-ODBC-broen. Langsomste og kun til Windows.
- Type 2: Drivere skrevet i C eller C++ til den specifikke platform (normalt de hurtigste).
- Type 3: Platformsuafhængig (ren Java-) driver med databaseuafhængig kommunikationsprotokol
- Type 4: Platformsuafhængig (ren Java-) driver skrevet til at kommunikere med en specifik database (mest udbredte og næsten lige så hurtig som type 2).

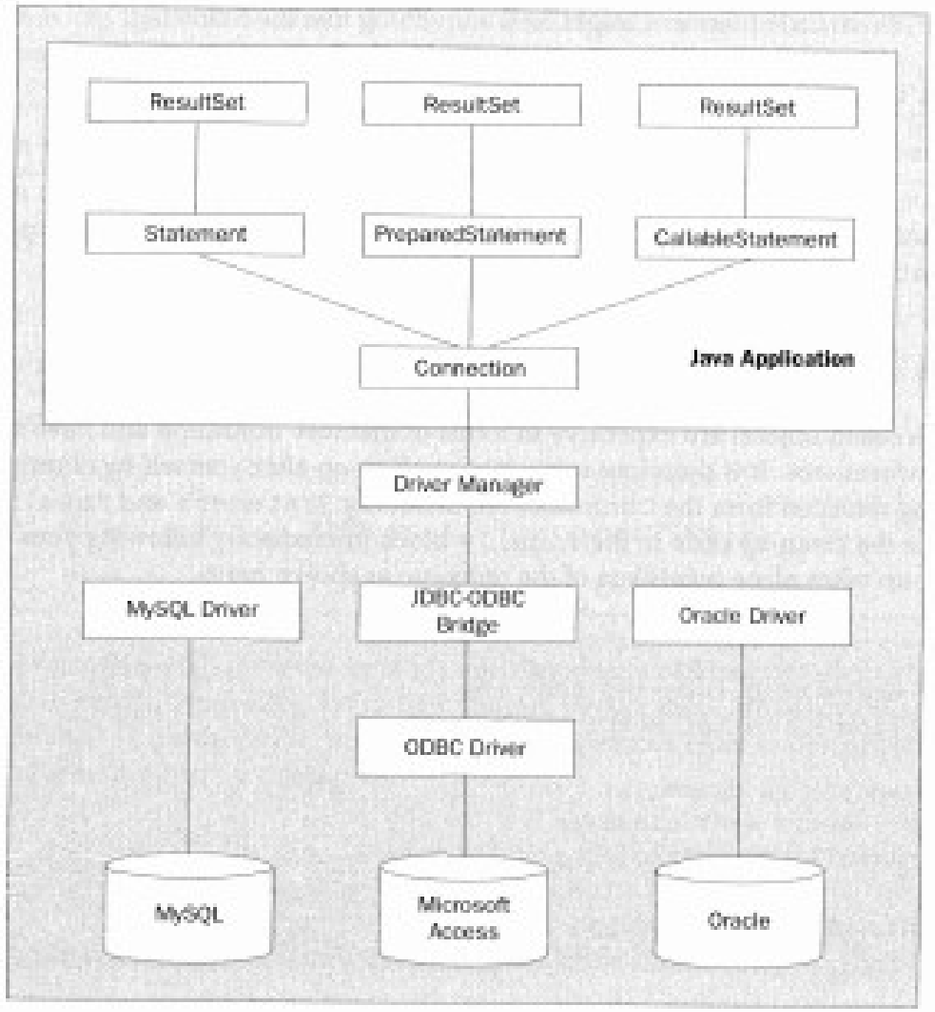
JDBC – databaseadgang

Kommunikere med databasen

```
import java.sql.*;
public class Databasekommunikation {
  public static void main(String[] arg) throws Exception {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection forb = DriverManager.getConnection("jdbc:odbc:datakilde1");
    Statement stmt = forb.createStatement();

    // forsøg at slette tabel - hvis den ikke findes opstår en fejl som fanges
    try { stmt.executeUpdate("drop table KUNDER"); } catch (Exception e) {}

    stmt.executeUpdate("create table KUNDER (NAVN varchar(32), KREDIT number)");

    stmt.executeUpdate("insert into KUNDER values('Jacob', -1799)");

    // indsæt data fra variabler
    String navn = "Hans";
    double kredit = 500;
    stmt.executeUpdate("insert into KUNDER(NAVN,KREDIT) values('"+navn+"', "+kredit+")");

    // forespørgsler
    ResultSet rs = stmt.executeQuery("select NAVN, KREDIT from KUNDER");
    while (rs.next())
    {
      navn = rs.getString("NAVN");
      kredit = rs.getDouble("KREDIT");
      System.out.println(navn+" "+kredit);
    }
  }
}
```

```
Jacob -1799.0
Brian 0.0
Hans 500.0
```

JDBC – databaseadgang



The figure shows how the `Connection` is the root of all objects returned from the `DriverManager`, and also that there are other ways of executing statements and obtaining `ResultSet` objects, namely the `PreparedStatement`, which is precompiled by the database for faster execution, and the `CallableStatement`, which can be used to access stored procedures.

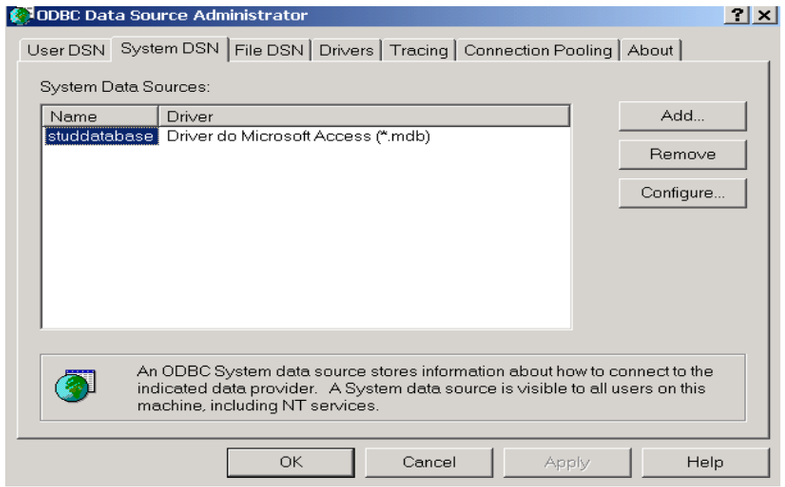
Persistensproblemet
Strategier til databaseadgang

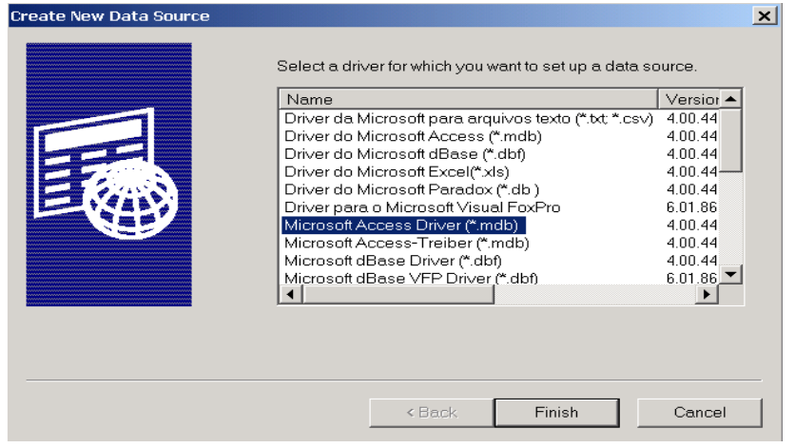
- Rigtig mange muligheder, f.eks.:
  - JDBC-kode blandet sammen med resten af koden
  - JDBC-kode i dedikerede klasser
  - JDBC RowSet - ResultSet, der også opdaterer databasen
    - CachedRowSet - kan eksistere løsrevet fra databasen
    - WebRowSet giver XML, der kan transformeres, f.eks. med XSLT (XSL style sheet), eller Javas XML-behandling
  - EJB - Enterprise JavaBeans - ét serverobjekt pr. række
  - JPA - Java Persistence AP - ét objekt pr. række
  - Proprietære løsninger
    - Giver mulighed for grafiske databasekomponenter
    - Oracle JDeveloper ADF - Application Developer Framework
    - Borland JBuilder: DataModule
    - IBM WebSphere Studio

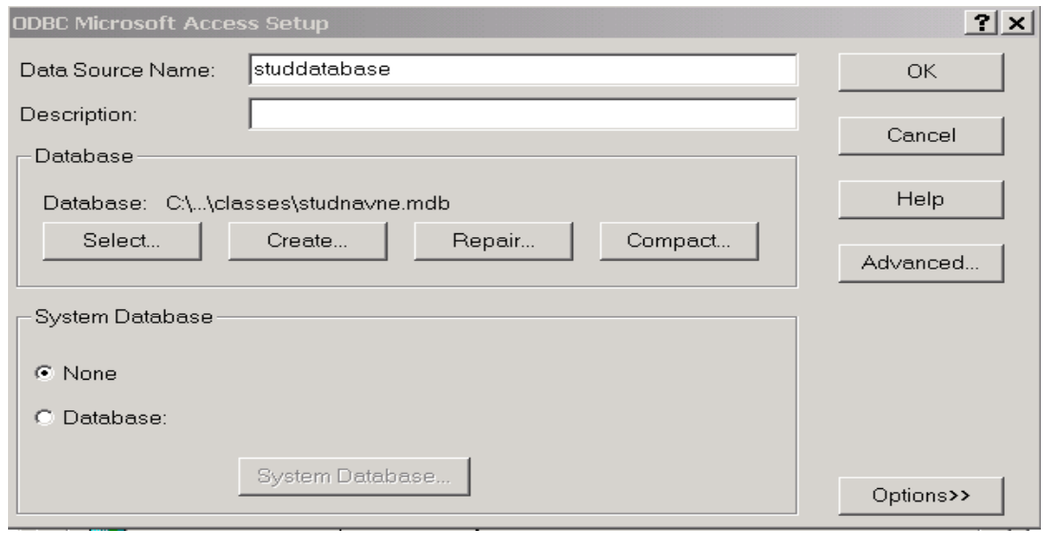
JDBC-ODBC-bro til Access-fil

Eksempel:

1. Denne computer
2. Kontrolpanel
3. Administration
4. 

5. 

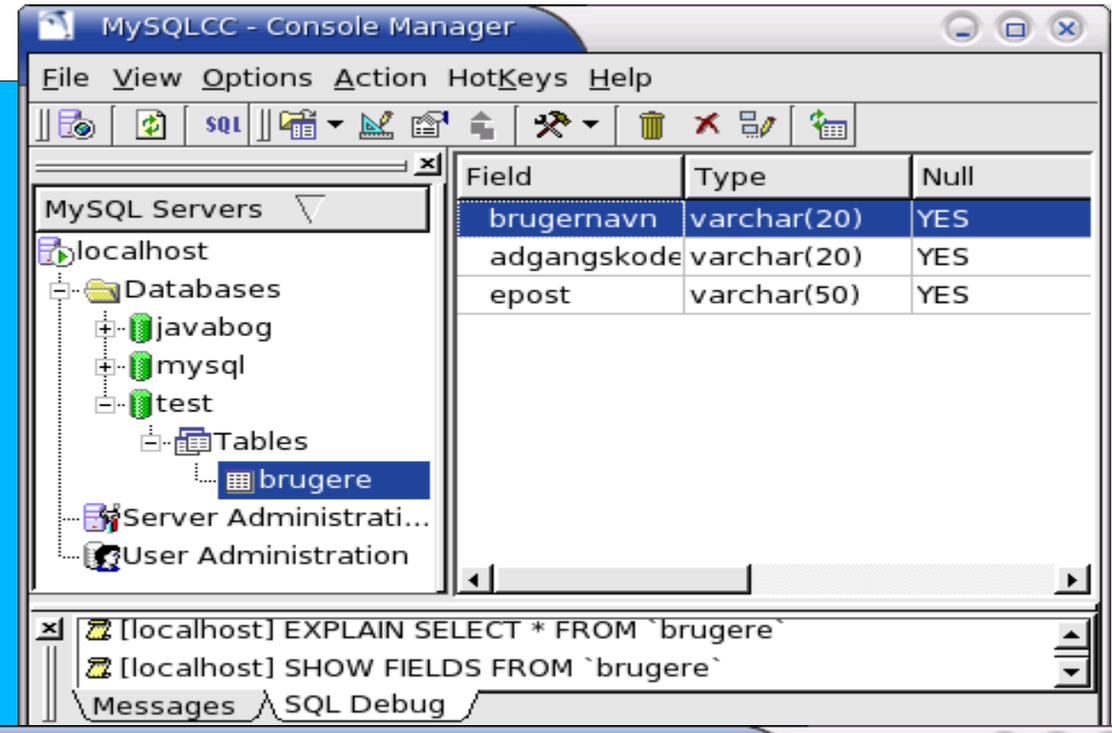
6. 

7. 

MySQL-database

- Installér MySQL
  - Hent fra mysql.com
  - test-database god i starten
  - Grafiske værktøjer
- Installér JDBC-driver
  - Connector/J fra mysql.com
  - Læg JAR-fil i java/jre/lib/ext/
- Kontakt test-database:

```
Class.forName("com.mysql.jdbc.Driver");
Connection forb =
  DriverManager.getConnection("jdbc:mysql:///test
```



Forberedte SQL-kommandoer

```
import java.sql.*;
public class ForberedtSQL {
    public static void main(String[] arg) throws Exception
    {
       Class.forName("com.mysql.jdbc.Driver");
       Connection con = DriverManager.getConnection("jdbc:mysql:///test");

       // Forbered kommandoerne til databasen, f.eks. i starten af programmet:
       PreparedStatement indsætPstm = con.prepareStatement(
                                  "INSERT INTO kunder (navn,kredit) VALUES(?, ?)");

       PreparedStatement hentPstm = con.prepareStatement(
                                  "SELECT navn, kredit FROM kunder WHERE navn=?");

       // under programudførelsen kan de forberedte kommandoer udføres mange gange:
       for (int i=0; i<100; i++)
       {
          indsætPstm.setString(1, "Brian");
          indsætPstm.setInt(2, i);
          indsætPstm.execute();

          indsætPstm.setString(1, "Hans' venner");  // bemærk ' i strengen
          indsætPstm.setInt(2, 1042+i);
          indsætPstm.execute();


          hentPstm.setString(1, "Hans' venner");   // bemærk ' i SQL-forespørgslen
          ResultSet rs = hentPstm.executeQuery();

          // man løber igennem svaret som man plejer
          while (rs.next())
          {
             String navn = rs.getString(1);
             double kredit = rs.getDouble(2);
             System.out.println(navn+" "+kredit);
          }
       }
    }
}
```

Eksempel: Gæstebog

Se WJSP, s. 98

Optimering

- Drivere
- Prepared statement
- Batch (kø)
- Stored procedures
- Forbindelsespujler

Samlede batch-opdateringer

- venter ikke på svar fra DBMS mellem opdateringer
- data-integritet...?

```
import java.sql.*;
public class Batchopdateringer
{
   public static void main(String[] arg) throws Exception
   {
     Class.forName("com.mysql.jdbc.Driver");
     Connection con = DriverManager.getConnection("jdbc:mysql:///test");

     PreparedStatement pstmt = con.prepareStatement(
                         "INSERT INTO kunder (navn,kredit) VALUES(?, ?)");

     pstmt.setString(1, "Hans");
     pstmt.setInt(2, 142);
     pstmt.addBatch();

     pstmt.setString(1, "Grethe");
     pstmt.setInt(2, 242);
     pstmt.addBatch();

     // send ændringer til databasen
     pstmt.executeBatch();
   }
}
```

Transaktioner


```
import java.sql.*;
public class UdenAutocommit
{
    public static void main(String[] arg) throws Exception
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection forb = DriverManager.getConnection(
                "jdbc:oracle:thin:@ora.javabog.dk:1521:student","jacob","jacob");

      try {
        forb.setAutoCommit(false);
        Statement stmt = forb.createStatement();

        stmt.executeUpdate("insert into KUNDER(NAVN,KREDIT) values('Jacob',-17)");
        stmt.executeUpdate("insert into KUNDER(NAVN,KREDIT) values('Brian', 0)");

        // flere transaktioner ...
        System.err.println("Alt gik godt, gør ændringerne forpligtigende");
        forb.commit();
      }
      catch (Exception e)
      {
        e.printStackTrace();
        System.err.println("Noget gik galt! Annullerer ændringerne...");
        forb.rollback();
      }
      finally
      {
        // Husk at sætte auto-commit tilbage, af hensyn til andre transaktioner
        forb.setAutoCommit(true);
      }
    }
}
```

Rydde op med finally

- Finally-blokke bliver altid udført, selv når undtagelsen ikke fanges, eller der hoppes ud af metoden på anden vis

```
public void metode1()
{
   try
   {
      ...
      if (...) return;
      if (...) throw new IllegalArgumentException(...);
      ...
   }
   catch (NullPointerException e)
   {
      System.out.println("Intern fejl:");
      e.printStackTrace();
   }
   finally
   {
      System.out.println("Dette bliver altid udført.");
   }
   System.out.println("Slut på metode1()");
}
```

```
      ResultSetMetaData rsmd = rs.getMetaData();
      int antalKolonner = rsmd.getColumnCount();
      System.out.println();
      System.out.println(" --------- " + titel + " ("+antalKolonner+" kolonner)");

      // udskriv kolonnenavnene
      for (int i=1; i<=antalKolonner; i++) skrivFormateret(rsmd.getColumnName(i));
      System.out.println();

      // udskriv cellerne i hver række
      while (rs.next())
      {
         for (int i=1; i<=antalKolonner; i++) skrivFormateret(""+rs.getString(i));
         System.out.println();
      }
      System.out.println(" ----------------------------");
   }

   public static void main(String[] arg) throws Exception
   {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection forb = DriverManager.getConnection(
                "jdbc:oracle:thin:@ora.javabog.dk:1521:student","jacob","jacob");

      DatabaseMetaData dmd = forb.getMetaData();
      System.out.println("DatabaseProductName = "+dmd.getDatabaseProductName());
      System.out.println("DriverName          = "+dmd.getDriverName());
      System.out.println("MaxRowSize          = "+dmd.getMaxRowSize());

      ResultSet rs = dmd.getTables(null, "JANO", "%", null);
      udskriv("tabeller i databasen", rs);

      Statement stmt = forb.createStatement();
      rs = stmt.executeQuery("select * from KUNDER");
      udskriv("kunder", rs);
   }

   public static void skrivFormateret(String str) {
      System.out.print(str);
      String KOL = "               | ";
      if (str.length()<KOL.length()) System.out.print(KOL.substring(str.length()));
   }
}
```

```
      ResultSetMetaData rsmd = rs.getMetaData();
      int antalKolonner = rsmd.getColumnCount();
      System.out.println();
      System.out.println(" --------- " + titel + " ("+antalKolonner+" kolonner)");

      // udskriv kolonnenavnene
      for (int i=1; i<=antalKolonner; i++) skrivFormateret(rsmd.getColumnName(i));
      System.out.println();

      // udskriv cellerne i hver række
      while (rs.next())
      {
         for (int i=1; i<=antalKolonner; i++) skrivFormateret(""+rs.getString(i));
         System.out.println();
      }
      System.out.println(" ----------------------------");
   }

   public st
   {
      Class.f
      Connect

      Databas
      System.
      System.
      System.

      Results
      udskriv

      Stateme
      rs = st
      udskriv
   }

   public static void skrivFormateret(String str) {
      System.out.print(str);
      String KOL = "              | ";
      if (str.length()<KOL.length()) System.out.print(KOL.substring(str.length()));
   }
}
```

```
DatabaseProductName = Oracle
DriverName          = Oracle JDBC driver
MaxRowSize          = 2000

 --------- tabeller i databasen (5 kolonner)
TABLE_CAT     | TABLE_SCHEM   | TABLE_NAME    | TABLE_TYPE    | TABLE_REMARKS |
null          | JANO          | KUNDER        | TABLE         | null          |
null          | JANO          | PERSONER      | TABLE         | null          |
 ----------------------------

--------- kunder (2 kolonner)
NAVN          | KREDIT        |
Jacob         | -1799         |
Brian         | 0             |
Hans          | 500           |
 ----------------------------
```

Opdatere og navigere i ResultSet

De fleste ResultSet har metoder til at bevæge sig aktivt rundt i svaret og endda opdatere databasen gennem svaret.
Det gøres med f.eks.:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("select NAVN, KREDIT from KUNDER");
```

Derefter kan man navigere rundt i ResultSet-objektet med f.eks.:

```
rs.absolute(3);     // går til 3. række i svaret (regnet fra 1 af)
rs.previous();      // går en række tilbage (modsatte af next())
rs.first();         // går til starten af svaret (svarende til rs.absolute(1))
rs.relative(3);     // går 3 rækker frem, dvs. til 4. række
int r = rs.getRow();// giver hvilken række vi nu er i (her returneres 4)
```

Man kan ændre i data med f.eks.:

```
rs.updateString("NAVN", "Jakob"); // ændrer kundens navn til Jakob
rs.updateRow();                   // opdaterer rækken i databasen

rs.moveToInsertRow();             // flyt til speciel indsættelses-række
rs.updateString("NAVN", "Søren"); // sæt navn
rs.updateDouble("KREDIT", 1000);  // sæt kredit
rs.insertRow();                   // indsæt rækken i databasen
rs.moveToCurrentRow();            // gå væk fra speciel indsættelsesrække
```

Derudover findes cancelRowUpdates(), der annullerer opdateringer i en række, deleteRow(), der sletter den aktuelle række fra både svaret og databasen, refreshRow(), der opfrisker ResultSet-objektet med de nyeste data.

JDBC RowSet

Oracle: ”A thin wrapper around ResultSet object to make a JDBC driver look like a JavaBeans component”

- RowSet = et sæt rækker hentet fra databasen
  - =ResultSet+oprindelse+ændringer
- RowSet-objekter giver mulighed ny arbejdsform
  - Man behøver kun at spørge på data (f.eks. med SELECT) hvorefter man får et RowSet-objekt.
    - Når man skal opdatere nogle rækker, slette eller oprette nye, opererer man blot det eksisterende RowSet-objekt
    - Slut med at rode med INSERT-, DELETE- eller UPDATE-sætninger i SQL.
- Flere undertyper af RowSet
  - JdbcRowSet =ResultSet+oprindelse+ændringer
  - CachedRowSet =rækkerne er gemt i hukommelsen
    - kan altså eksistere afbrudt fra databasen!
      - Undertyper: FilteredRowSet og JoinRowSet
  - WebRowSet =CachedRowSet repræsenteret som XML

JdbcRowSet

```
import java.sql.*;
import javax.sql.*;
import javax.sql.rowset.*;
import com.sun.rowset.*; // Bemærk: rowset.jar fra Sun skal være i CLASSPATH

public class BenytJdbcRowSet
{
   public static void main(String[] arg) throws Exception
   {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection("jdbc:mysql:///test");
      JdbcRowSetImpl jrs = new JdbcRowSetImpl(con);

      // Lav forespørglen
      jrs.setCommand("SELECT * FROM kunder WHERE navn = ?");
      jrs.setString(1,"Jacob");
      jrs.execute();

      // Udskriv resultatet
      while (jrs.next())
      {
         String navn = jrs.getString("navn");
         double kredit = jrs.getDouble("kredit");
         System.out.println(navn+" "+kredit);
      }
   }
}
```

Opdateringer? - skal indstille jrs Type og Concurrency
 - kun én table i SELECT!
Data-integritet...?

CachedRowSet - off-line

```
import java.sql.*;
import javax.sql.*;
import javax.sql.rowset.*;
import com.sun.rowset.*; // Bemærk: rowset.jar fra Sun skal være i CLASSPATH

public class BenytCachedRowSet
{
   public static void main(String[] arg) throws Exception
   {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection("jdbc:mysql:///test");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM kunder");

      CachedRowSetImpl crs = new CachedRowSetImpl();
      crs.populate(rs);
      rs.close();

      // Opdatér første række i CachedRowSet-objektet
      crs.first();
      crs.updateDouble("kredit", -2000);
      crs.updateRow();

      // Indsæt række
      crs.moveToInsertRow();
      crs.updateString("navn", "Poul Nyrup");
      crs.updateDouble("kredit", 100000);
      crs.insertRow();
      crs.moveToCurrentRow();

      // Opdatér data i databasen
      crs.setUrl("jdbc:mysql:///test");
      crs.setUsername("root");
      crs.setPassword("");
      crs.acceptChanges();
   }
}
```

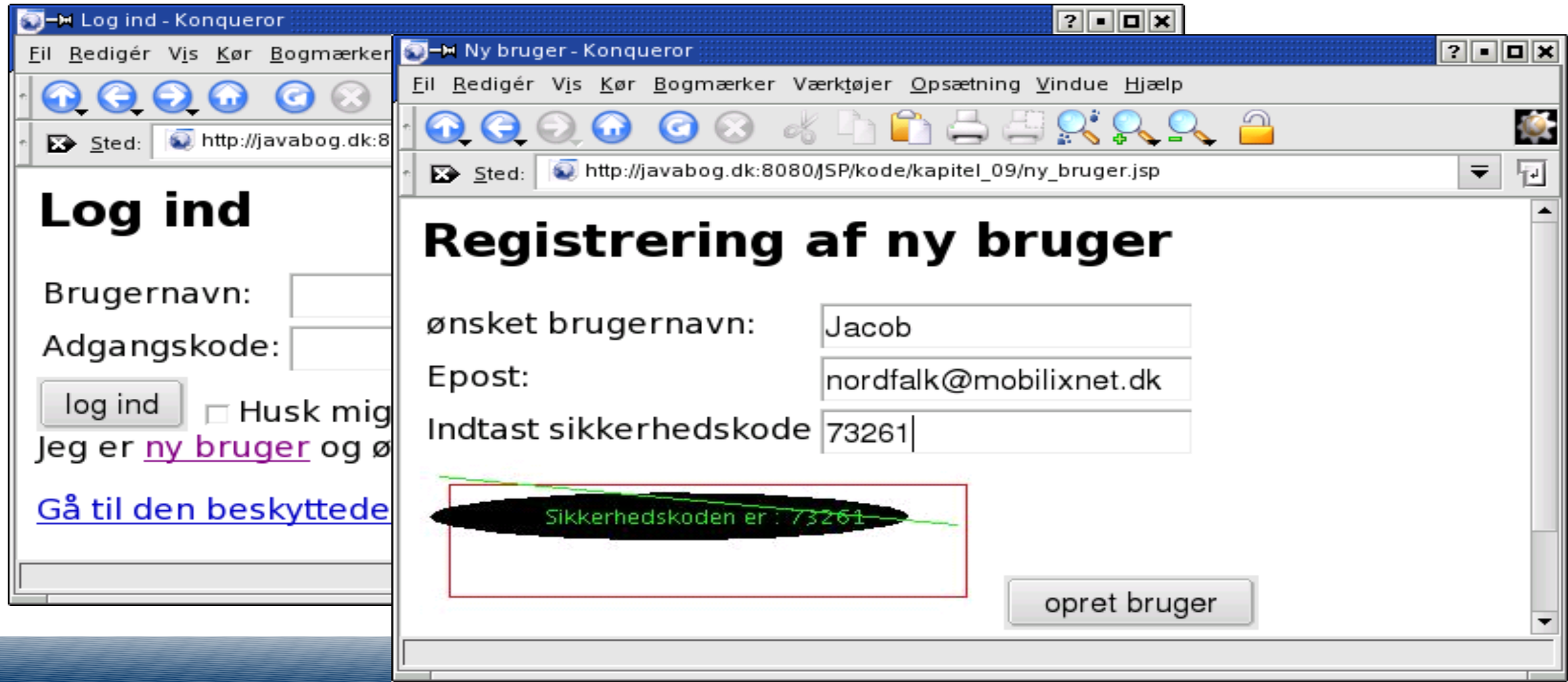
WebRowSet – off-line; XML-output

```
public class Benyt
{
   public static vo
   {
      Class.forName
      Connection co
      Statement stm
      ResultSet rs =

      WebRowSetImpl
      wrs.populate(
      rs.close();

      // Generér XM
      wrs.writeXml(

      // Opdatér fø
      wrs.first();
      wrs.updateDou
      wrs.updateRow

      // Indsæt ræk
      wrs.moveToIns
      wrs.updateStr
      wrs.updateDou
      wrs.insertRow
      wrs.moveToCur

      // Generér XM
      wrs.writeXml(

      // Opdatér da
      wrs.setUrl("j
      wrs.setUsername("root");
      wrs.setPassword("");
      wrs.acceptChanges();
   }
}
```

```
<?xml version="1.0"?>
<webRowSet xmlns="http://java.sun.com/xml/ns/jdbc" ...>
  <metadata>
    <column-count>2</column-count>
    <column-definition>
      <column-index>1</column-index>
      <column-display-size>32</column-display-size>
      <column-name>navn</column-name>
      <table-name>kunder</table-name>
      <column-type-name>VARCHAR</column-type-name>
    </column-definition>
    <column-definition>
      <column-index>2</column-index>
      <column-name>kredit</column-name>
      <schema-name></schema-name>
      <table-name>kunder</table-name>
      <column-type-name>FLOAT</column-type-name>
    </column-definition>
  </metadata>
  <data>
    <currentRow>
      <columnValue>jacob</columnValue>
      <columnValue>-2000.0</columnValue>
    </currentRow>
    <currentRow>
      <columnValue>brian</columnValue>
      <columnValue>0.0</columnValue>
    </currentRow>
    <currentRow>
      <columnValue>Poul Nyrup</columnValue>
      <columnValue>100000.0</columnValue>
    </currentRow>
  </data>
</webRowSet>
```

Login og brugeroprettelse

- Eksempel på javabønne
- Bemærk: Serveren kan klare brugervalidering for dig



JPA / Java Persistence API

- Bibliotek holder styr på binding mellem database og klasser
  - Annotations

```
@Entity
@Table(name="user")
public class User  {

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  @Basic(optional=false)
  @Column(name="id", nullable=false)
  private Integer id;

  @Column(name="username", length=30)
  private String username;

  @Column(name="password", length=50)
  private String password;
```

- Typisk brug
  - EntityManagerFactory emf = javax.persistence.Persistence.createEntity-ManagerFactory("ffproduction");
  - EntityManager emq = emf.createEntityManager();
  - Query allUsersq = emq.createNativeQuery("select * from user",User.class);
  - List<User> allUsers = allUsersq.getResultList();

JPA / Java Persistence API

- Bibliotek holder styr på al opdatering
  - Transaktioner

```
EntityManager emEnqueueJobs = emf.createEntityManager();
emEnqueueJobs.getTransaction().begin();

job.setState(FF.STATE_CANCELLED);

emEnqueueJobs.merge(job);
emEnqueueJobs.getTransaction().commit();
```

- Opsætning (persistence.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="ffproduction" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>ff.db.Job</class>
    <class>ff.db.User</class>
    <properties>
      <property name="eclipselink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="eclipselink.jdbc.url" value="jdbc:mysql://localhost/ff"/>
      <property name="eclipselink.jdbc.user" value="ff"/>
```

Åben Dokumentlicens

- Dette foredragsmateriale er under Åben Dokumentlicens (ÅDL)
  - Du har derfor lov til frit at kopiere dette værk
  - Bruger du dele af værket i et nyt værk, skal de dele, der stammer fra dette værk, igen frigives under ÅDL
  - Den fulde licens kan ses på http://www.sslug.dk/linuxbog/licens.html