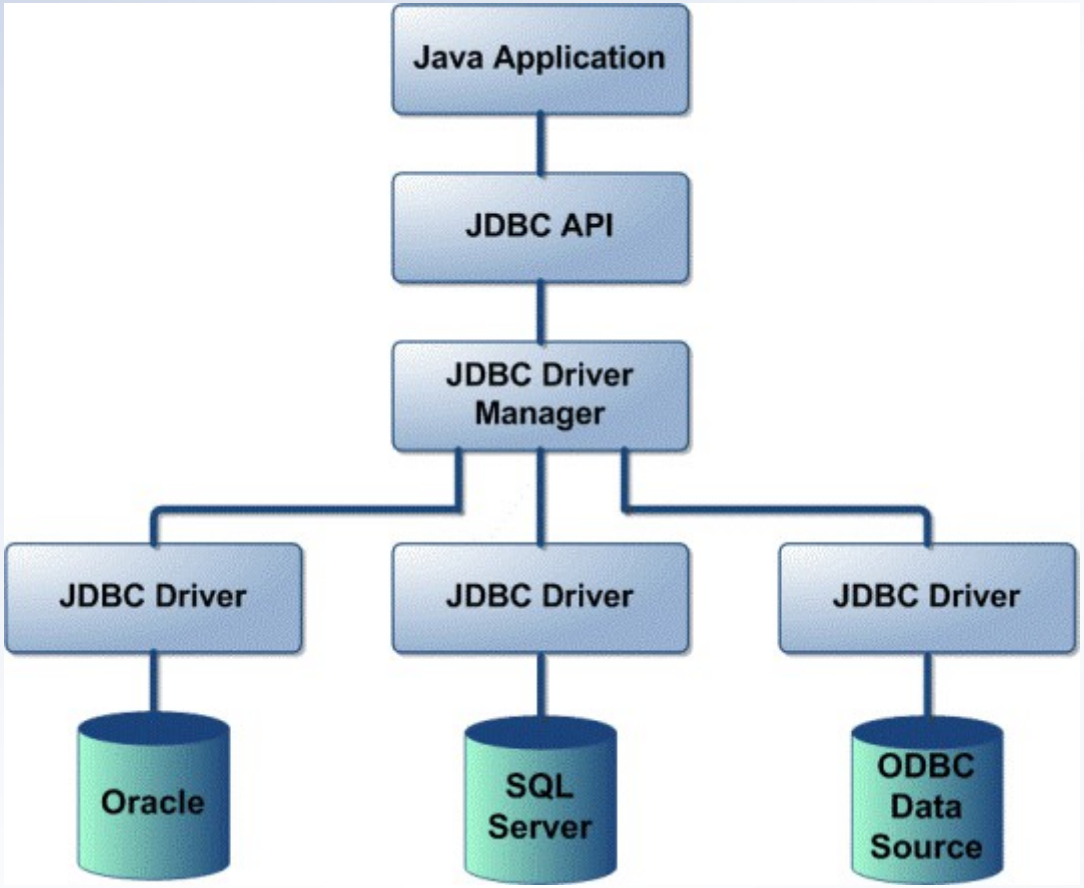
Spajanje Java aplikacija na bazu podataka

Java aplikacije i pristup bazi podataka pomoću JDBC-a

- Za pristup relacijskim bazama podataka Java SE sadrži JDBC (engl. *Java Database Connectivity*) API
- Omogućava spajanje na sve vrste baza podataka, pri čemu svaka od njih mora imati vlastitu "JAR" datoteku
- Omogućava tri programske aktivnosti:
  - Ostvarivanje veze s izvorom podataka (engl. *data source*) – bazom podataka
  - Pripremanje i izvršavanje SQL upita (engl. *queries*), koje uključuju dohvaćanje, ažuriranje i brisanje iz baze podataka
  - Zatvaranje veze s izvorom podataka

JDBC arhitektura



- Iz Java aplikacije poziva se programski kod iz JDBC biblioteke
- JDBC uz pomoć pogonskih programa (engl. *drivera*) komunicira s bazom podataka pomoću SQL upita
- Pogonski programi SQL pozive pretvaraju u odgovarajući komunikacijski protokol za određenu bazu podataka

H2 baza podataka i Eclipse DTP *plugin*

- **H2** je open source relacijska baza podataka u cijelosti implementirana u Javi
- Preuzimanje sa stranice: http://www.h2database.com/html/download.html
- Aktualna verzija 1.3.176 izdana 05.04.2014.
- Jednostavna za korištenje
- Podržava arhitekturu klijent-poslužitelj (engl. *client-server*) i lako se integrira u razvojno okruženje Eclipse
- **Eclipse DTP** (engl. *Data Tools Platform*) *plugin* – služi za prilagođavanje razvojnog okruženja Eclipse za rad s bazama podataka, *open source*
- Preuzimanje sa stranice: http://eclipse.org/datatools/, aktualna verzija 1.11.2

Programski kod za spajanje na bazu podataka (1/2)

- Prvi korak u pisanju Java programskog koda koji pristupa bazi podataka je kreiranje objekta koji predstavlja vezu s bazom podataka:

```
// otvaranje veze (konekcije) s bazom podataka
Connection veza = DriverManager.getConnection(
"jdbc:mojDriver:mojaBaza", "mojLogin", "mojaLozinka");
```

- Statička metoda "**getConnection**" iz klase "**DriverManager**" služi za kreiranje veze s bazom podataka koja je definirana "URL"-om, korisničkim imenom i lozinkom za pristupanje bazi podataka
- Nakon toga potrebno je kreirati objekt pomoću kojeg će se pripremati i izvoditi upiti nad bazom podataka:

```
// kreiranje objekta za izvršavanje upita
Statement stmt = veza.createStatement();
```

Programski kod za spajanje na bazu podataka (2/2)

- Objekti tipa "**Statement**" omogućavaju pozivanje metode "**executeQuery**" koja izvršava zadani SQL upit, npr:

```
// izvođenje upita i dohvaćanje rezultata upita
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
```

- Navedeni primjer dohvaća vrijednosti iz stupaca pod nazivom "a", "b" i "c" iz tablice "Table1", bez dodatnih kriterija za filtriranje
- Rezultat upita predstavljen je pomoću objekta tipa "**ResultSet**" koji može sadržavati nijedan, jedan ili više redaka kao rezultata izvršavanja upita, te pomoću "while" petlje dohvatiti svaki od tih redaka i tipove podataka:

```
// analiza rezultata koji može predstavljen u više redaka (rows)
while (rs.next()) {
    int x = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
}
```

Uspostavljanje veze s bazom podataka

- Za uspostavljanje veze između Java programa i baze podataka koristi se *Database Management System* (DBMS)
- Veza se uspostavlja pomoću poziva metode "getConnection" iz klase "DriverManager", npr.:

```
Connection veza = DriverManager.getConnection(
"jdbc:h2:~studenti", "student", "password");
```

- URL koji označava lokaciju baze podataka (npr. `"jdbc:h2:~studenti"`) sastoji se od nekoliko ključnih dijelova:
  - jdbc:h2 – tip baze podataka
  - `~studenti` – naziv baze podataka
  - "student" i "password" – korisničko ime i lozinka

Kreiranje jednostavne tablice "STUDENTI" u H2 bazi podataka

- Programski kod za kreiranje i punjenje tablice "STUDENTI" koja sadrži podatke o studentima:

```
CREATE TABLE STUDENTI
(
id INT NOT NULL GENERATED ALWAYS AS IDENTITY,
jmbag VARCHAR(20) NOT NULL,
ime VARCHAR(50) NOT NULL,
prezime VARCHAR(50) NOT NULL,
datum_rodjenja DATE NOT NULL,
PRIMARY KEY (id)
);

INSERT INTO STUDENTI (jmbag, ime, prezime, datum_rodjenja)
VALUES ('0036374849', 'Pero', 'Perić', '1984-01-01');
INSERT INTO STUDENTI (jmbag, ime, prezime, datum_rodjenja)
VALUES ('0024568238', 'Ivo', 'Ivić', '1983-01-01');

SELECT * FROM STUDENTI;
```

Dohvaćanje podataka iz tablice "STUDENTI"

- Priprema veze i izvršavanje upita:

```
Connection veza = DriverManager.getConnection(
"jdbc:h2:~studenti", "student", "password");

Statement stmt = veza.createStatement();

ResultSet rs = stmt.executeQuery("SELECT * FROM STUDENTI");

while (rs.next()) {
    int id = rs.getInt("id");
    String jmbag = rs.getString("jmbag");
    String ime = rs.getString("ime");
    String prezime = rs.getString("prezime");
    Date date = rs.getDate("datum_rodjenja");

    System.out.println("Pročitani redak: " + id + " " + jmbag
        + " " + ime + " " + prezime + " " + date);
}
```

Ažuriranje podataka u tablici "STUDENT" (1/2)

```
Statement stmt = veza.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                      ResultSet.CONCUR_UPDATABLE);

ResultSet rs = stmt.executeQuery("SELECT * FROM STUDENTI");

while (rs.next()) {
   int id = rs.getInt("id");
   String jmbag = rs.getString("jmbag");
   String ime = rs.getString("ime");
   String prezime = rs.getString("prezime");
   Date date = rs.getDate("datum_rodjenja");

   System.out.println("Pročitani redak: " + id + " " + jmbag + " "
      + ime + " " + prezime + " " + date);

   if (jmbag.equals("0036374849")) {
      rs.updateString("ime", "Ivan");
      rs.updateString("prezime", "Horvat");
      rs.updateRow();
   }
}
```

Ažuriranje podataka u tablici "STUDENT" (2/2)

- U slučaju da će biti potrebno osim dohvaćanja i ažurirati podatke u tablici, potrebno je kreirati objekt tipa "Statement" koji u metodi "createStatement" prima dva ulazna parametra:

```
Statement stmt = veza.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                      ResultSet.CONCUR_UPDATABLE);
```

- Ulazni parametri definiraju način kretanja "pokazivača" rezultantnih redaka (FORWARD_ONLY), te opciju za konkurentno ažuriranje (CONCUR_UPDATABLE)
- Samo ažuriranje podataka obavlja se pomoću metode "**update**" koja u samom nazivu sadrži tip podataka koji se ažurira (npr. "**updateString**"), te pozivom metode "**updateRow**" poziva završne radnje kod ažuriranja podataka

Korištenje klase PreparedStatement (1/2)

- Objekti tipa "Statement" prilikom svakog izvršavanja prevode SQL upit, bez obzira na to što je identičan kao i kod prvog izvođenja
- Kako bi se izbjeglo nepotrebno prevođenje istog upita više puta, te time uštedilo na vremenu tijekom izvođenja programskog koda, koristi se klasa "**PreparedStatement**"
- Korištenjem klase "PreparedStatement" upit se ne prevodi više puta (kao što je to u slučaju objekata tipa "Statement"), već samo jednom
- Moguće je i korištenje promjenjivih parametara koji se označavaju s "?" i time omogućavaju višestruko izvođenje istog upita s drugim parametrima:

```
PreparedStatement updateStudenti =
 veza.prepareStatement(
 "UPDATE STUDENTI SET IME = ? WHERE JMBAG = ?");
```

Korištenje klase PreparedStatement (2/2)

- U tom slučaju vrijednosti parametara upita mogu se definirati na sljedeći način:

```
updateStudenti.setString(1, "Željko");
updateStudenti.setString(2, "0024568238");
```

- Izvršavanje upita za ažuriranje u tom slučaju izgleda ovako:

```
updateStudenti.executeUpdate();
```

- Rezultati nakon izvođenja upita za ažuriranje izgledaju ovako:

```
Pročitani redak: 3 0036374849 Ivan Horvat 1984-01-01
Pročitani redak: 4 0024568238 Željko Ivić 1983-01-01
```

Spremanje podataka u bazu

- Klasa "PreparedStatement" koristi se i kod dodavanja redaka u bazu podataka, što je moguće postići na sljedeći način:

```
PreparedStatement stmt = conn.prepareStatement(
"INSERT INTO STUDENTI (JMBAG, IME, PREZIME, DATUM_RODJENJA)
VALUES(?, ?, ?, ?)");

stmt.setString(1, jmbagStudenta);
stmt.setString(2, imeStudenta);
stmt.setString(3, prezimeStuddenta);
stmt.setDate(4, datumRodjenjaStudenta);

stmt.executeUpdate();
```

Zatvaranje veze s bazom podataka

- Slično kao i kod datoteka, nakon završetka korištenja baze podataka potrebno je zatvoriti vezu:

```
try {
    con = DriverManager.getConnection(url, "myLogin",
       "myPassword");
    stmt = con.createStatement();
    stmt.executeUpdate(queryString);
} catch (SQLException ex) {
    System.err.println("SQLException: " + ex.getMessage());
} finally {
    try {
       stmt.close();
       con.close();
    }
    catch(SQLException ex) {
       ex.printStackTrace();
    }
}
```

- Java 7 omogućava korištenje ***try* bloka s resursima**, što znatno skraćuje količinu programskog koda

Korištenje transakcija prilikom izvršavanja upita (1/2)

- Transakcija predstavlja skup jedne ili više operacija (engl. *statements*) koje se moraju izvesti zajedno kao da se radi o jednoj operaciji (atomarno)
- Svaka nova veza (engl. *connection*) s bazom podataka konfigurirana je tako da uvijek automatski sprema (engl. *commit*) sve promjene u bazu podataka odmah nakon izvršavanja upita (engl. *auto-commit mode*)
- Za promjenu tih predefiniranih postavki potrebno je koristiti metodu "setAutoCommit(false)":

```
veza.setAutoCommit(false);
```

- Tad je omogućeno „ručno“ spremanje (engl. *commit*) promjena u bazu kada je to potrebno ili vraćanje promjena na staro stanje (engl. *rollback*)

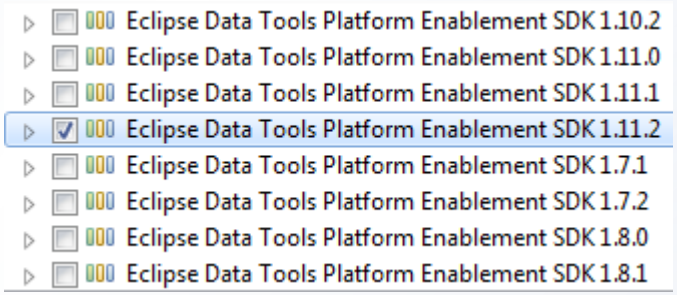
Korištenje transakcija prilikom izvršavanja upita (2/2)

- Primjer: oba ažuriranja podataka se obavljaju atomarno (ili oba budu uspješno izvedena ili nijedan od njih):

```
veza.setAutoCommit(false);

PreparedStatement updateStudenti1 = veza.prepareStatement(
"UPDATE STUDENTI SET IME = ? WHERE JMBAG = ?");

updateStudenti1.setString(1, "Petar");
updateStudenti1.setString(2, "0024568238");
updateStudenti1.executeUpdate();

PreparedStatement updateStudenti2 = veza.prepareStatement(
"UPDATE STUDENTI SET PREZIME = ? WHERE JMBAG = ?");

updateStudenti2.setString(1, "Ivičić");
updateStudenti2.setString(2, "0024568238");
updateStudenti2.executeUpdate();

veza.commit();
veza.setAutoCommit(true);
```

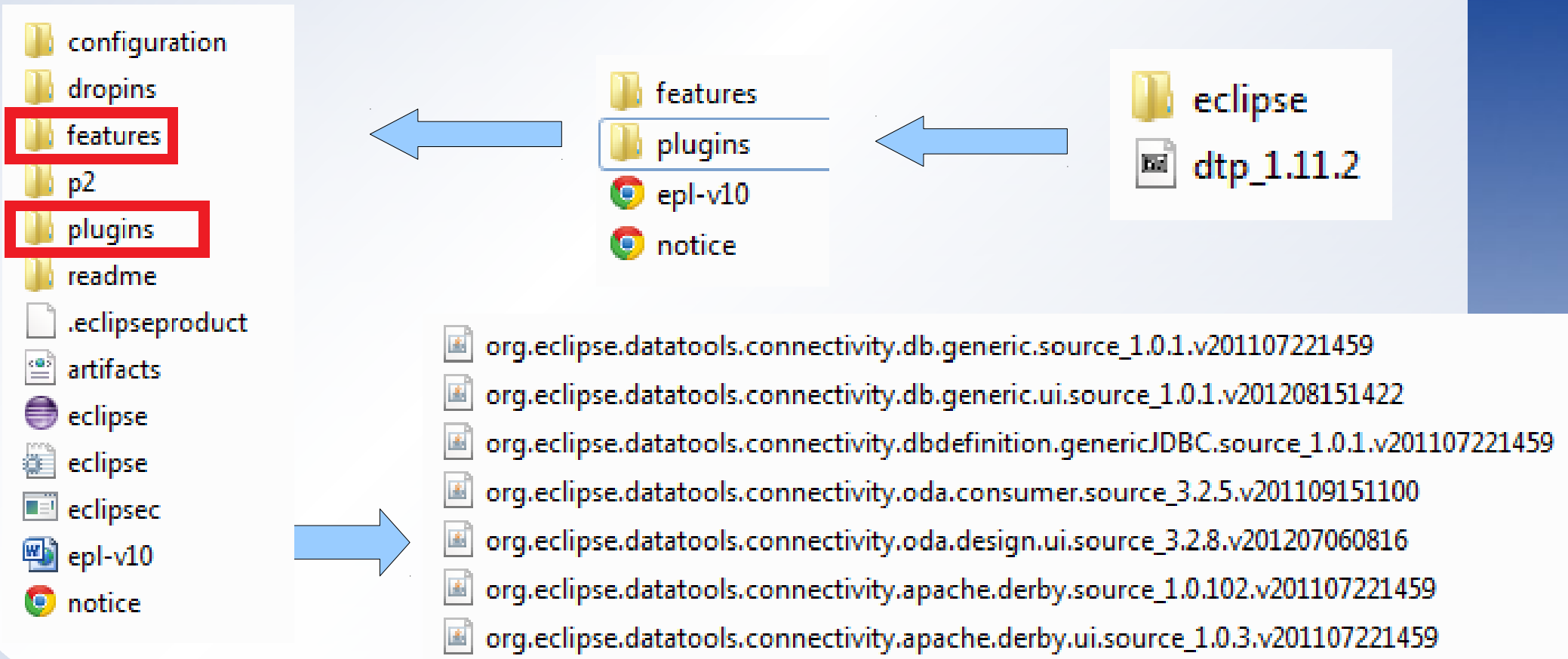
Instaliranje DTP *plugina* (1/2)

- Prije početka rada s bazom podataka potrebno je instalirati *Data Tools Platform plugin* za Eclipse koji se može naći na URL-u (*update site*-u): http://download.eclipse.org/datatools/updates
- Korištenjem opcije “Help->Install New Software...” moguće je odabrati opciju "Data Tools Platform Enablement SDK 1.11.2." i "Data Tools Platform SDK 1.11.2":



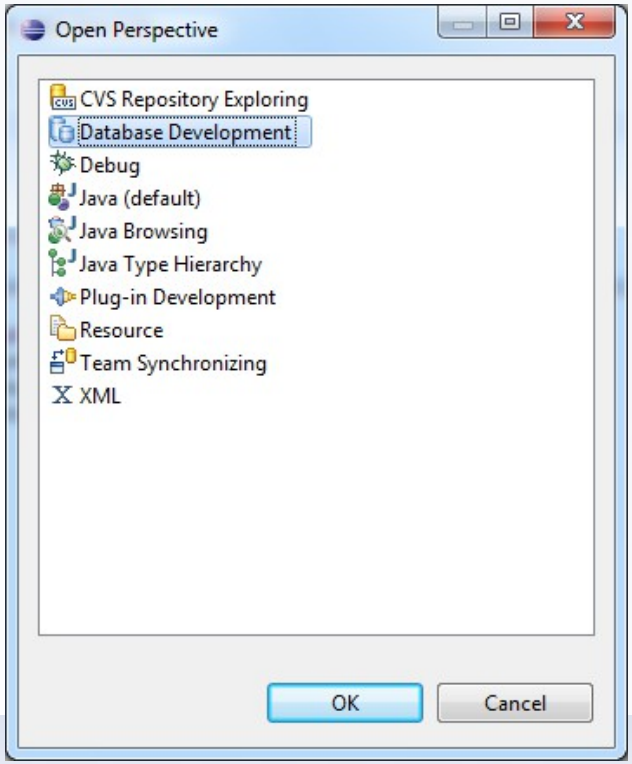
Instaliranje DTP *plugina* (2/2)

- Osim pomoću *update sitea*, pluginove je moguće instalirati i preuzimanjem arhive s mrežnih stranica, raspakiravanjem, te kopiranjem dobivenih mapa u instalacijsku mapu razvojnog okruženja Eclipse:



Otvaranje perspektive “*Database Development*”

- Nakon uspješne instalacije i ponovnog pokretanja razvojnog okruženja “Eclipse”, potrebno je otvoriti odgovarajuću "perspektivu" koja je prilagođena radu s bazom podataka
- Pomoću opcije "Window->Open Perspective->Other..." moguće je odabrati "perspektivu" pod nazivom "Database Development":

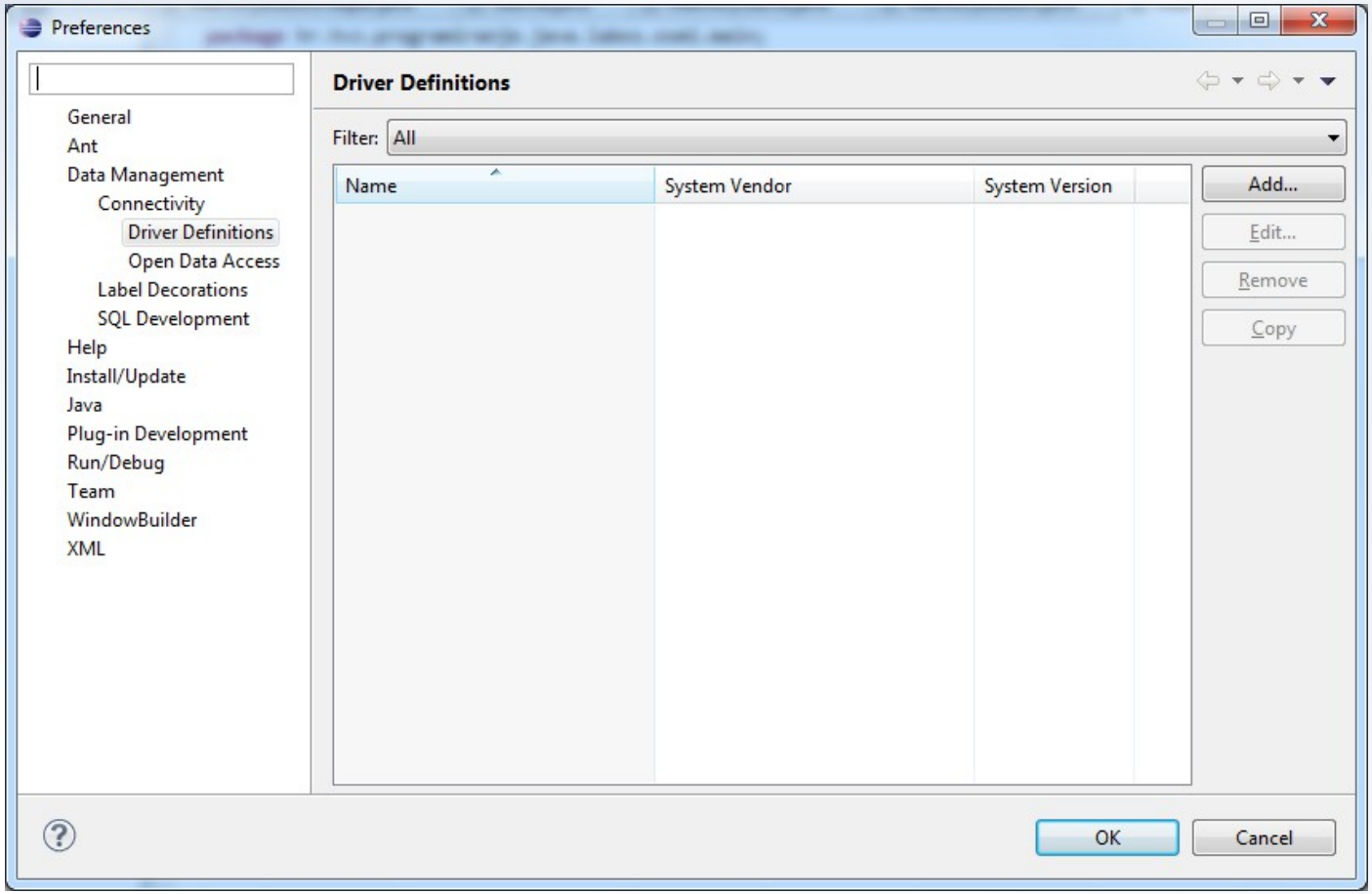


Preuzimanje H2 baze podataka

- Na URL-u
  http://www.h2database.com/html/download.html
  moguće je preuzeti datoteku pod nazivom
  "Platform-Independent Zip" ili "**Windows Installer**"
  koja sadrži distribuciju H2 baze podataka
- Raspakiravanjem "zip" arhive potrebno je locirati datoteku "bin/h2-1.3.176.jar" (ili druge verzije) koja će se koristiti kako bi se pokrenula i koristila baza podataka
- Tu datoteku je potrebno iskoristiti kako bi se definirao "pogonski program" (engl. *driver*) unutar razvojnog okruženja Eclipse i time omogućio Java aplikacijama korištenje H2 baze podataka

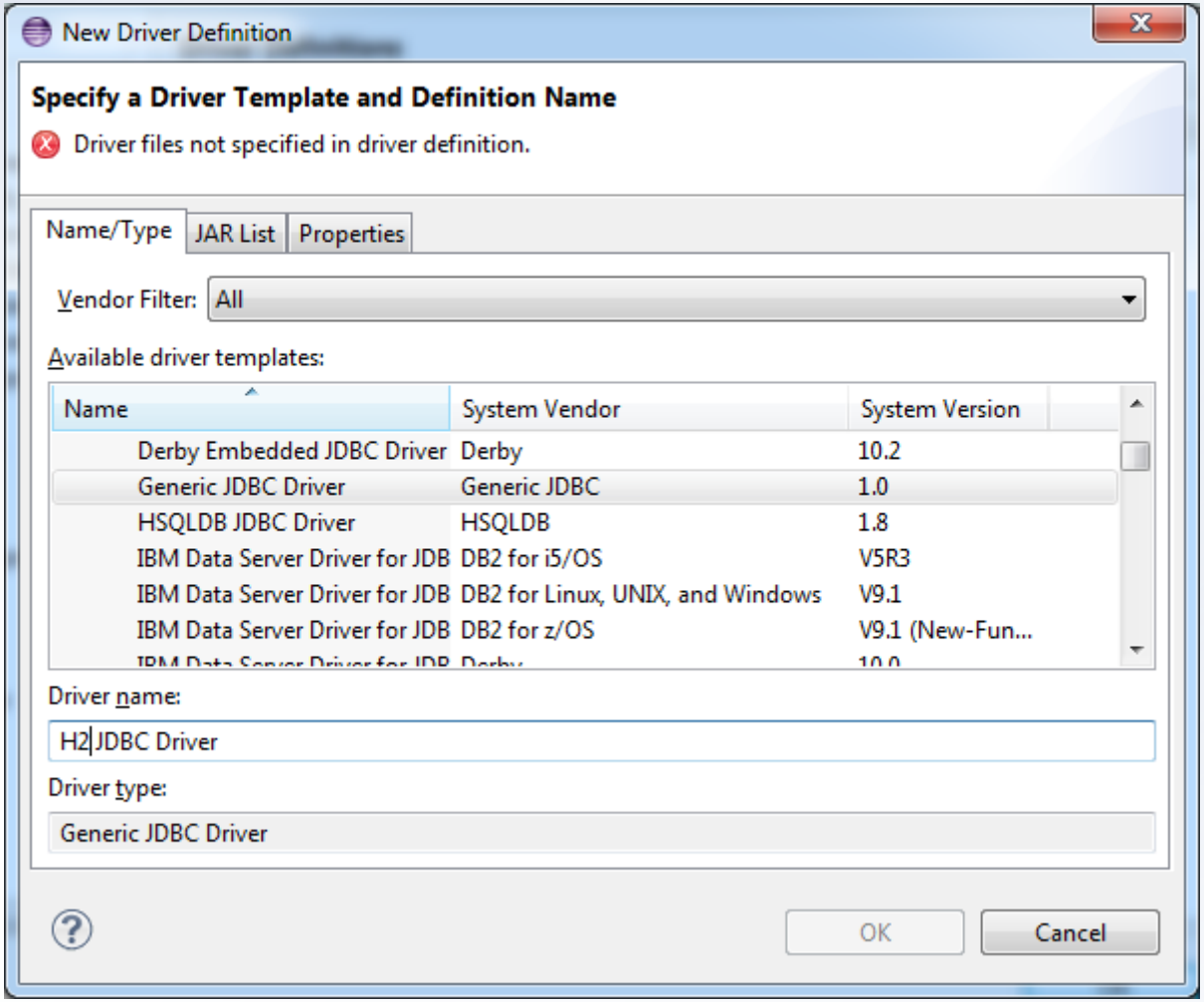
Postavljanje definicije *drivera* za H2 (1/4)

- Pokretanjem opcije "Window->Preferences->Data Management->Connectivity->Driver Definitions" otvara se prozor za dodavanje novog drivera u Eclipse:



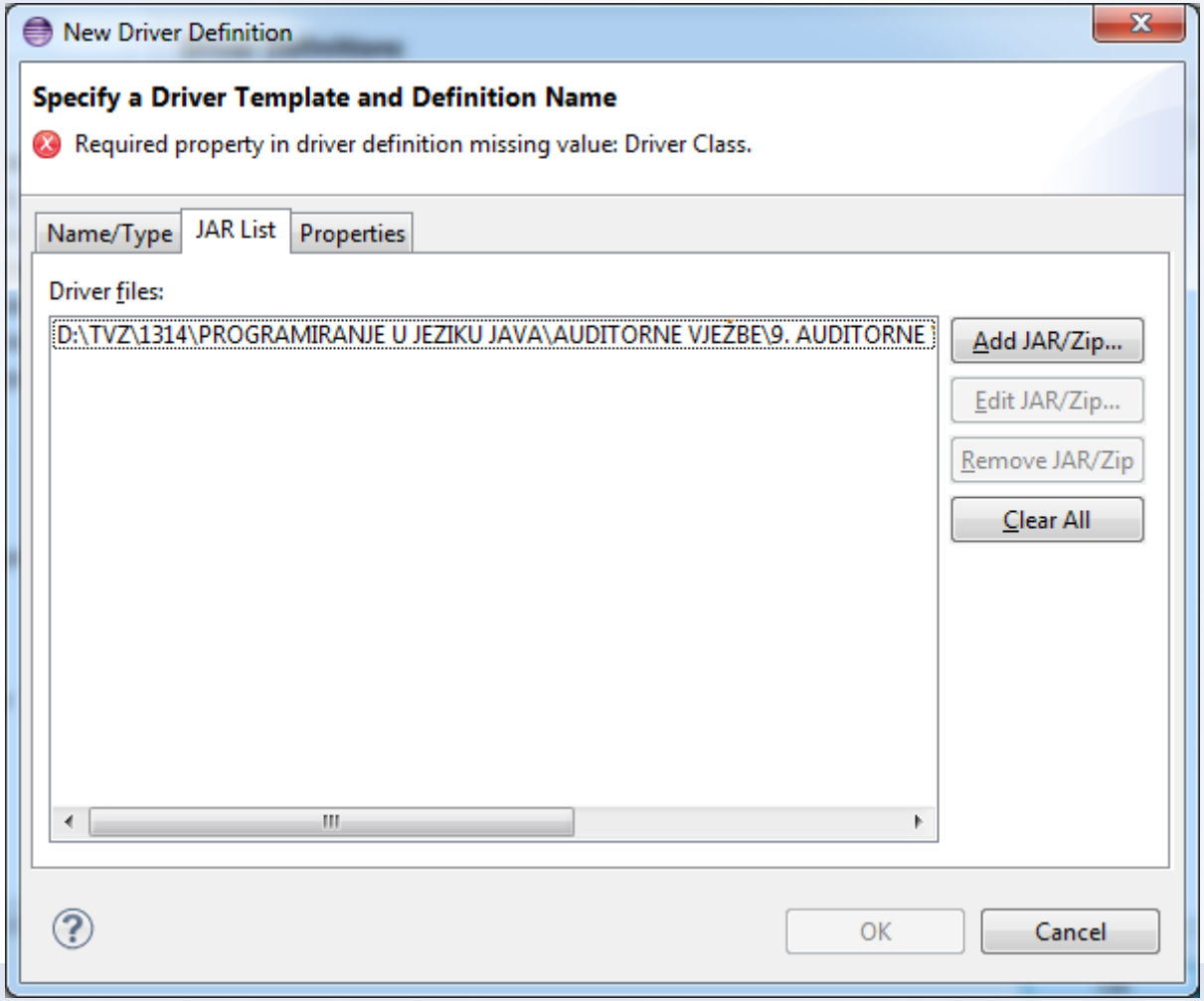
Postavljanje definicije *drivera* za H2 (2/4)

- Pritiskom na gumb "Add..." otvara se dijalog s tri taba na kojima je potrebno izabrati sljedeće postavke:



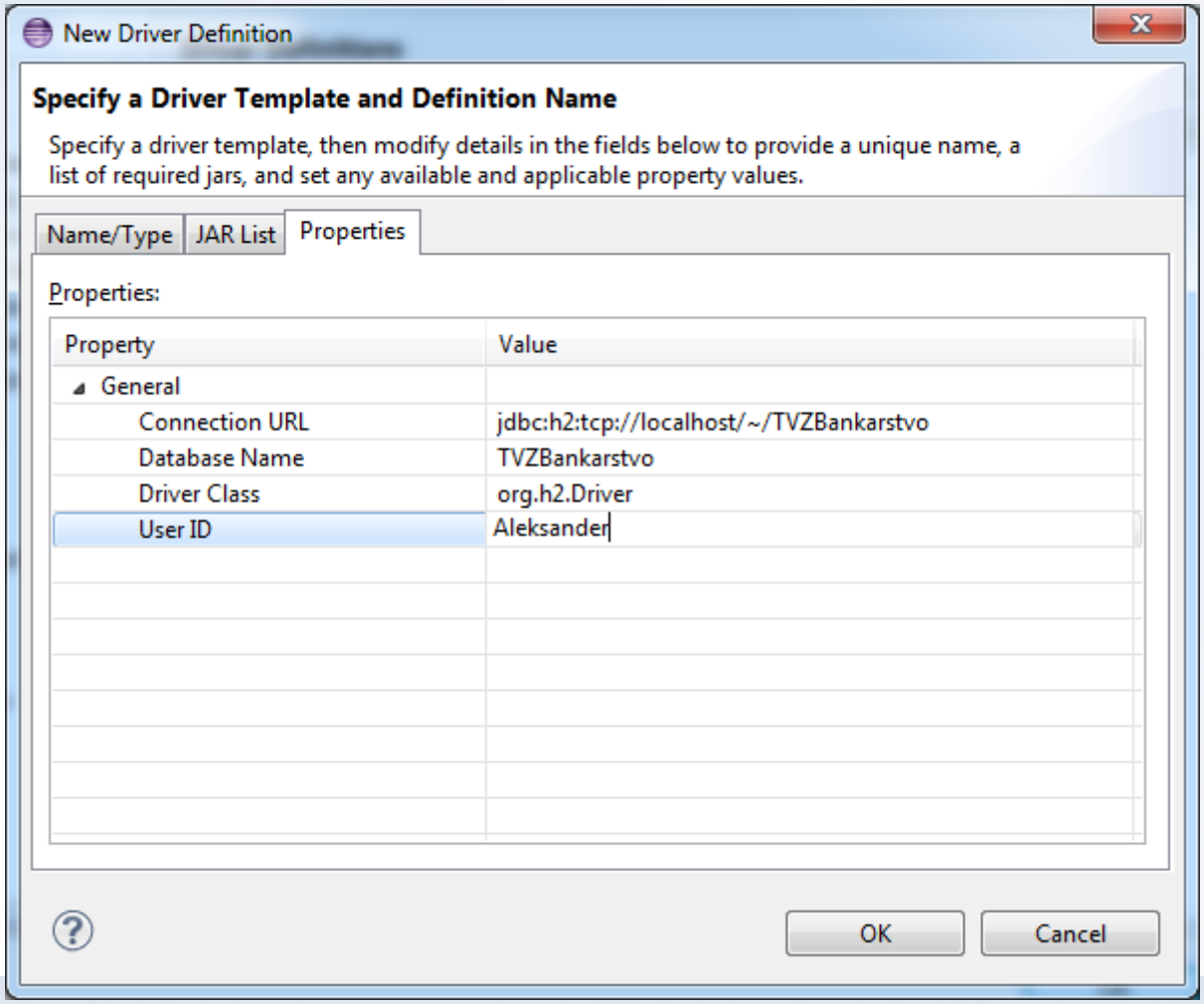
Postavljanje definicije *drivera* za H2 (3/4)

- Na *tabu* "Jar List" potrebno je definirati *driver* pomoću datoteke "h2-1.3.176.jar" koja se treba nalaziti unutar Eclipse projekta u kojem će se nalaziti aplikacija:



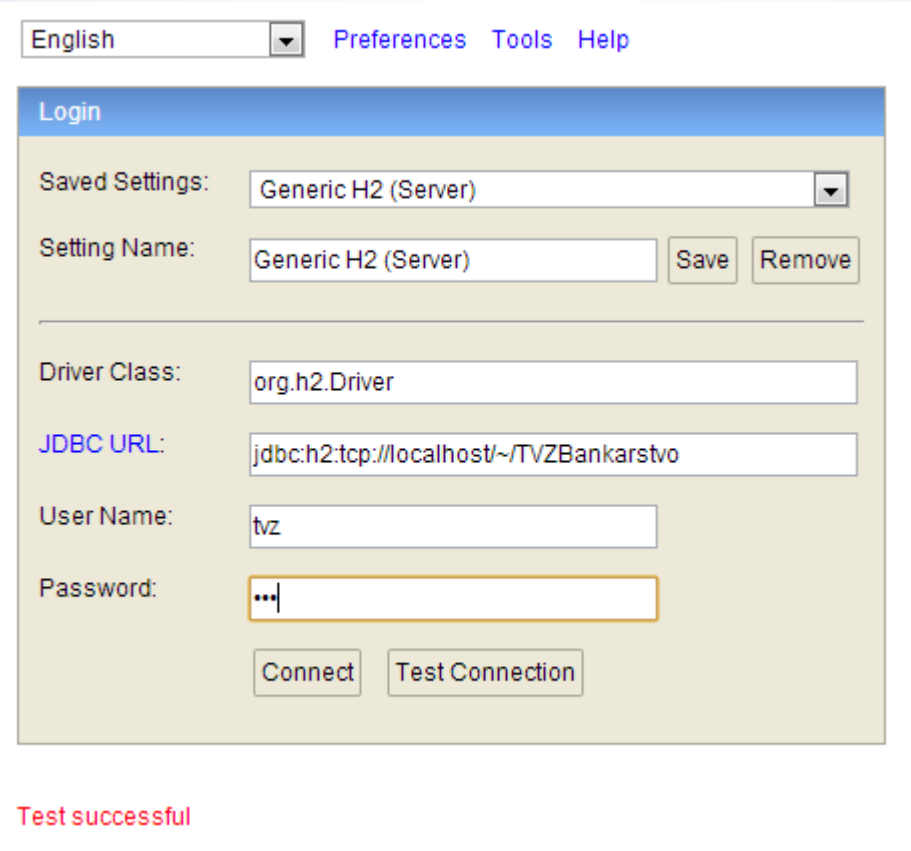
Postavljanje definicije *drivera* za H2 (4/4)

- Na "Properties" tabu potrebno je postaviti URL za bazu podataka, naziv baze podataka, klasu za pogonski program, lozinku te korisničko ime:



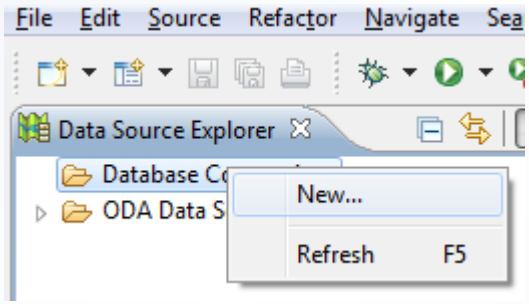
Kreiranje nove H2 baze podataka

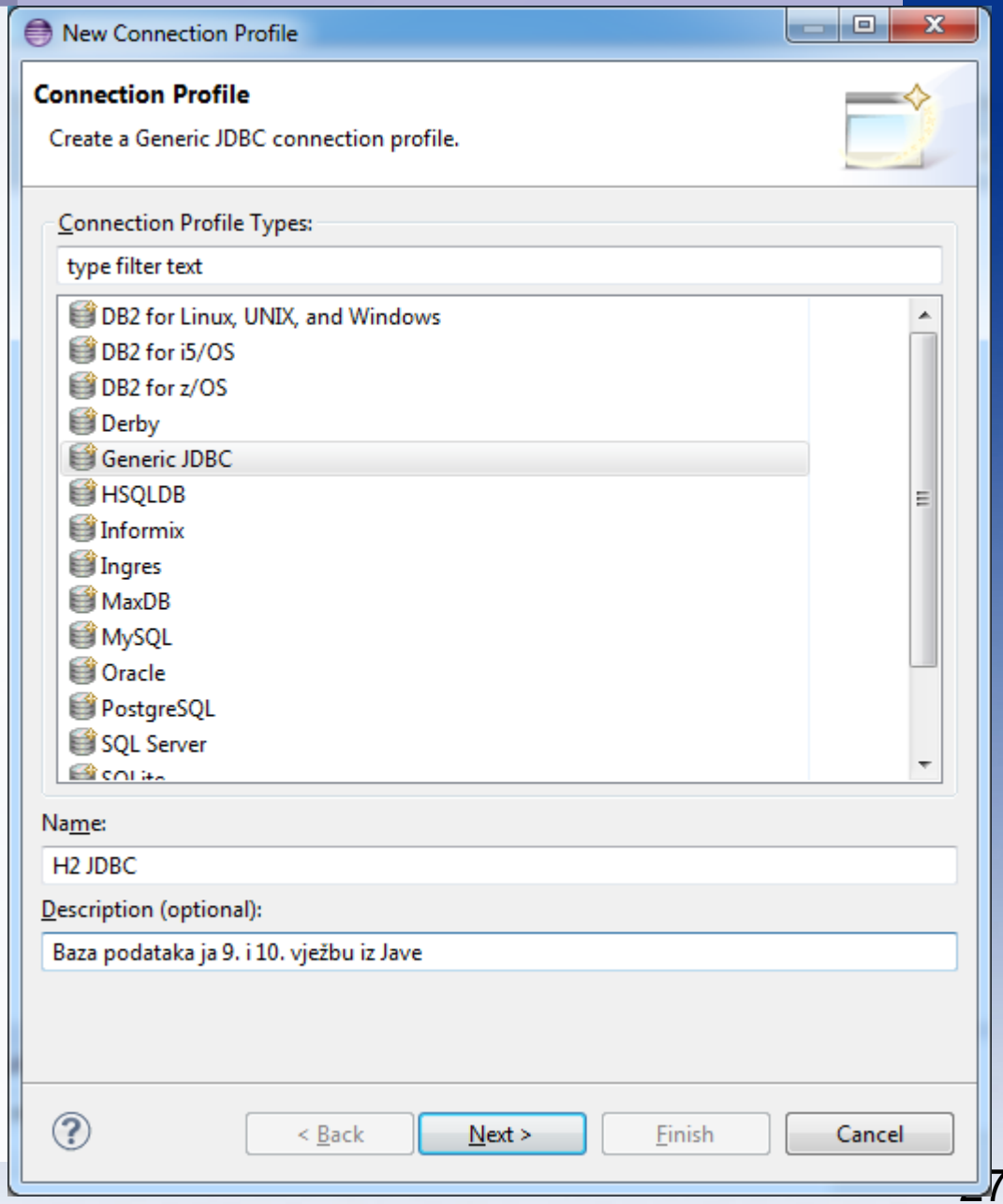
- Da bi se mogla koristiti iz razvojnog okruženja Eclipse, prvo je potrebno kreirati H2 bazu podataka uz pomoć web sučelja koje se otvara nakon pokretanja "h2-1.3.176.jar" datoteke (potrebno je samo upisati podatke, pritisnuti "Connect" i "Test Connection"):



Spajanje na H2 bazu podataka iz Eclipsea (1/2)

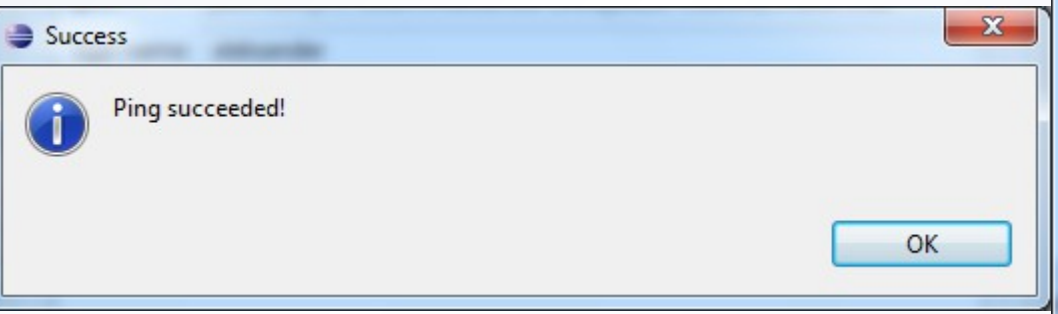
- Unutar "Database Development" perspektive potrebno je u "Data Source Explorer" tabu kreirati novu bazu podataka korištenjem opcije "New..." (koja se dobije klikom desne tipke miša):

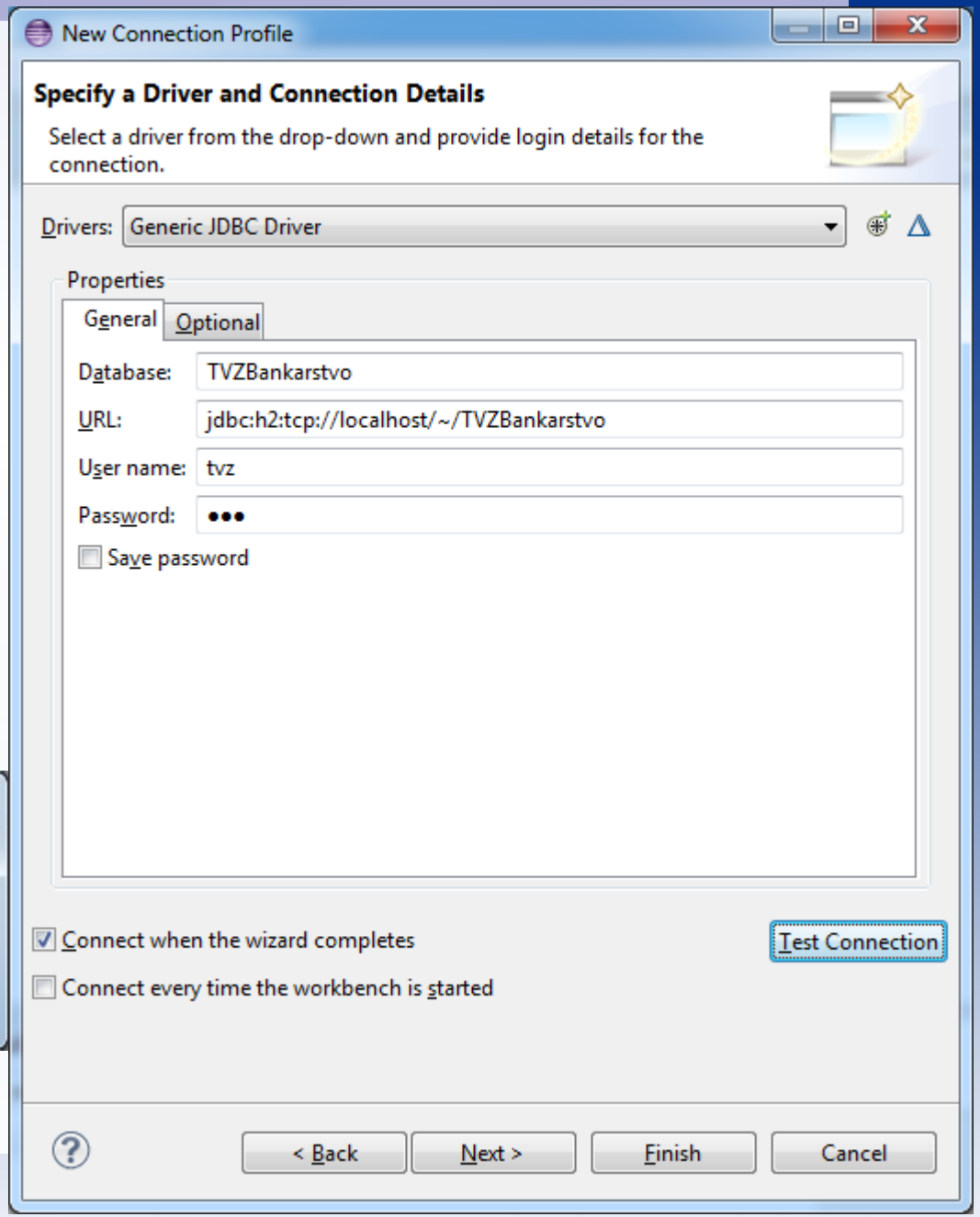




Spajanje na H2 bazu podataka iz Eclipsea (2/2)

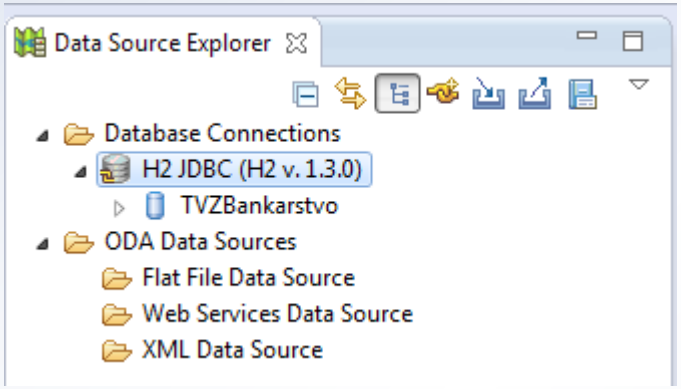
- Kako bi se mogla testirati veza s bazom podataka, potrebno je istu kreirati
- Ako se nakon toga pritisne tipka "Test Connection", razvojno okruženje Eclipse u slučaju da je sve u redu mora prikazati sljedeću poruku:



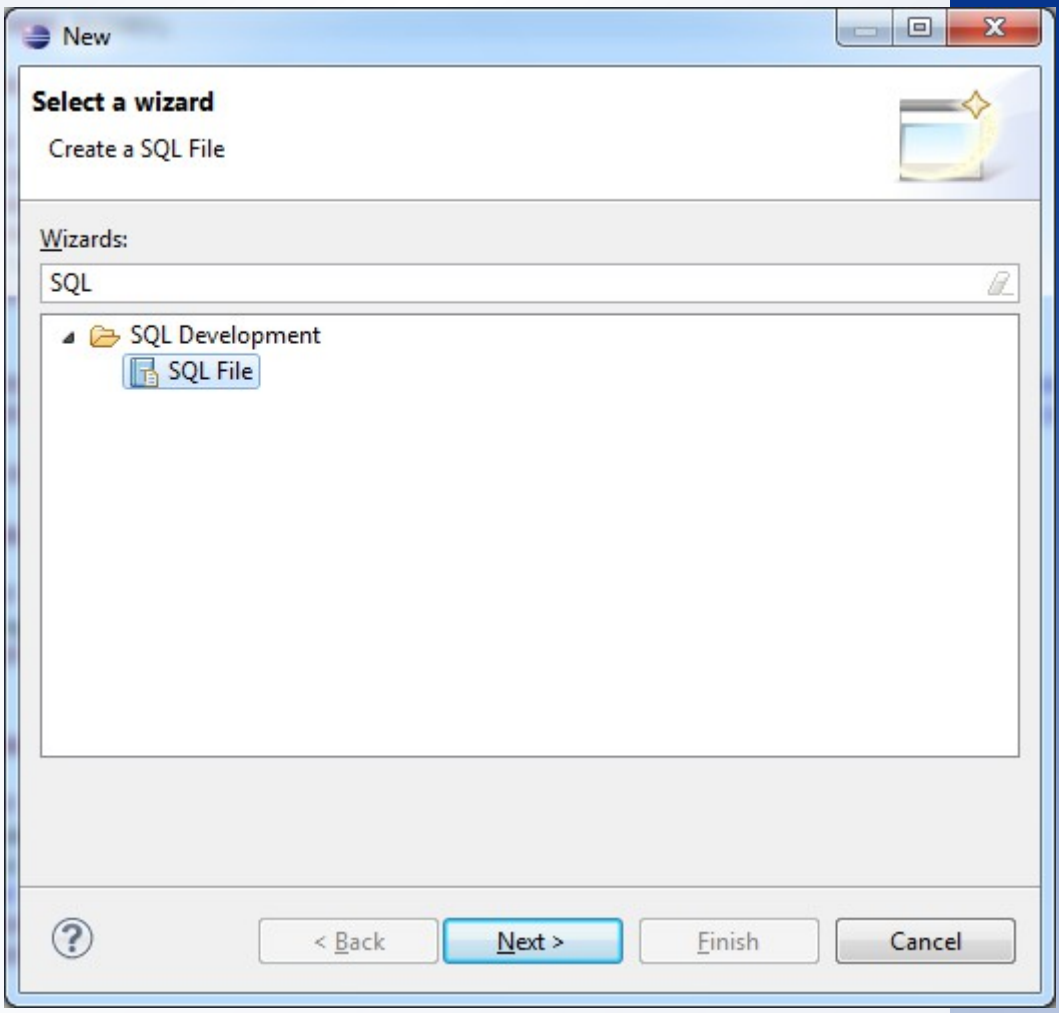


Izvođenje naredbi nad kreiranom bazom podataka (1/3)

- Nakon uspješnog spajanja na bazu podataka unutar "Data Source Explorera" prikazuje se sljedeća oznaka:

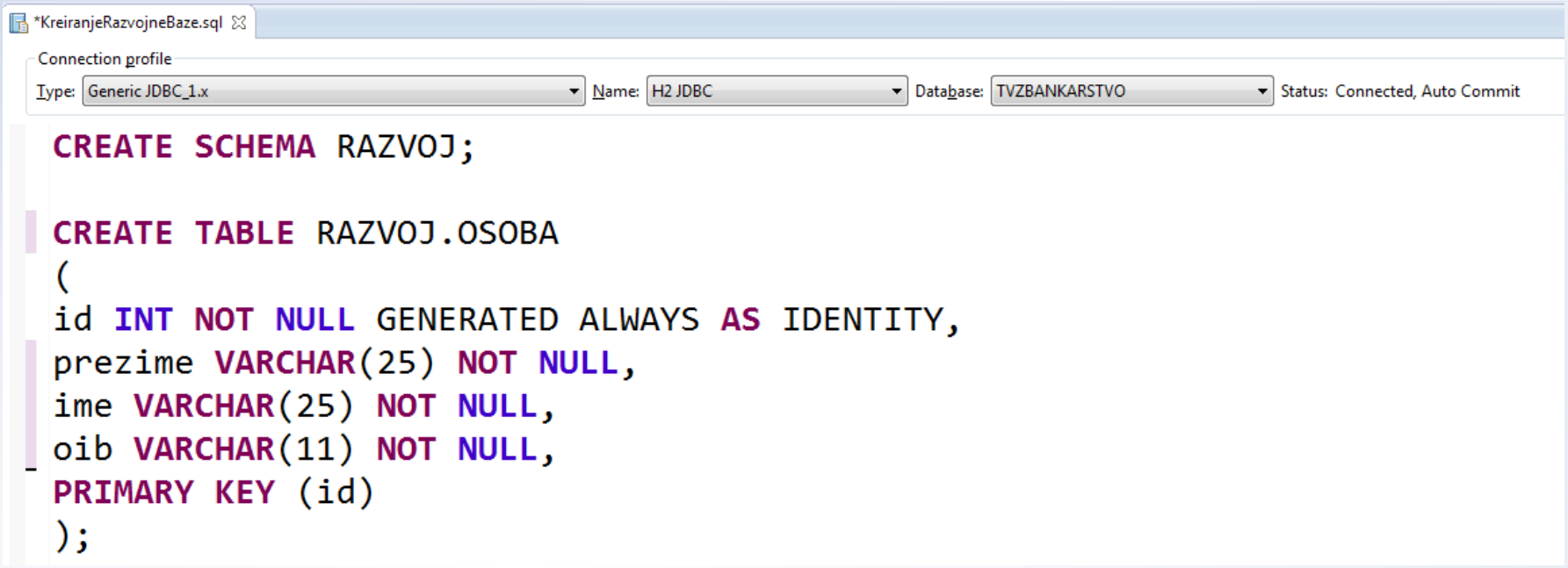


- Sljedeći korak je kreiranje "SQL" datoteke pomoću koje će se izvoditi SQL naredbe:



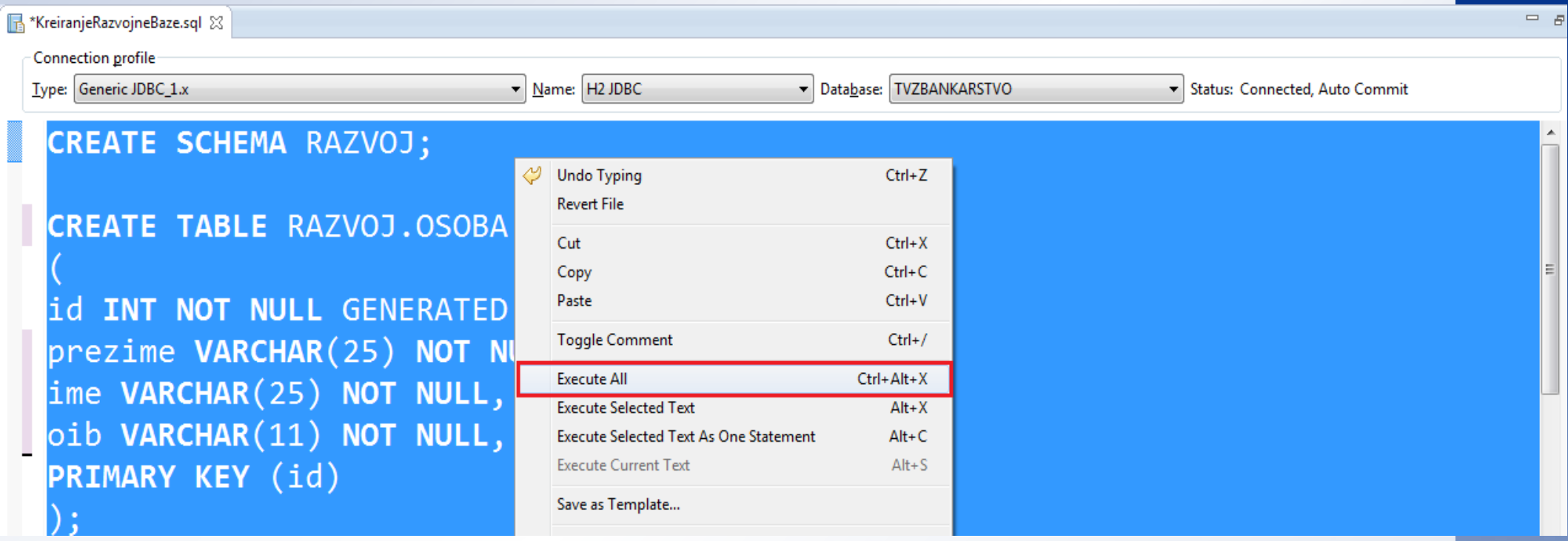
Izvođenje naredbi nad kreiranom bazom podataka (2/3)

- "SQL" datoteku potrebno je "spojiti" s bazom podataka kako bi se naredbe izvršavale na njoj:

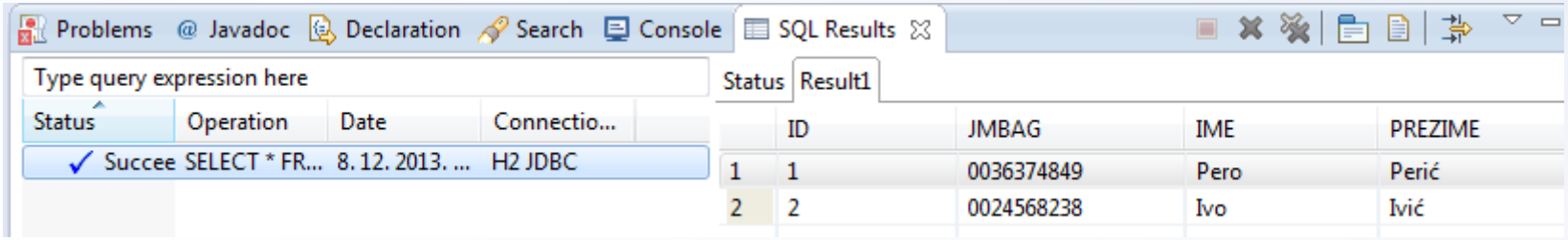


- Nakon toga moguće je izvoditi naredbu po naredbu koje se nalaze unutar SQL datoteke (preporuča se) ili izvoditi više naredbi zajedno (sadržaj datoteke "KreiranjeRazvojneBaze.sql" nalazi se unutar projekta "JDBC" koji se nalazi na službenim stranicama kolegija):

Izvođenje naredbi nad kreiranom bazom podataka (3/3)

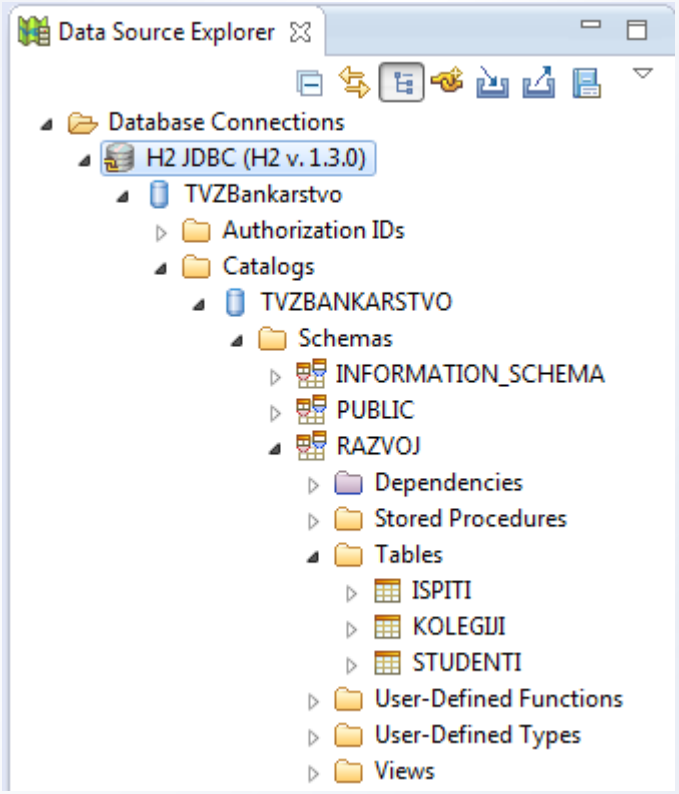


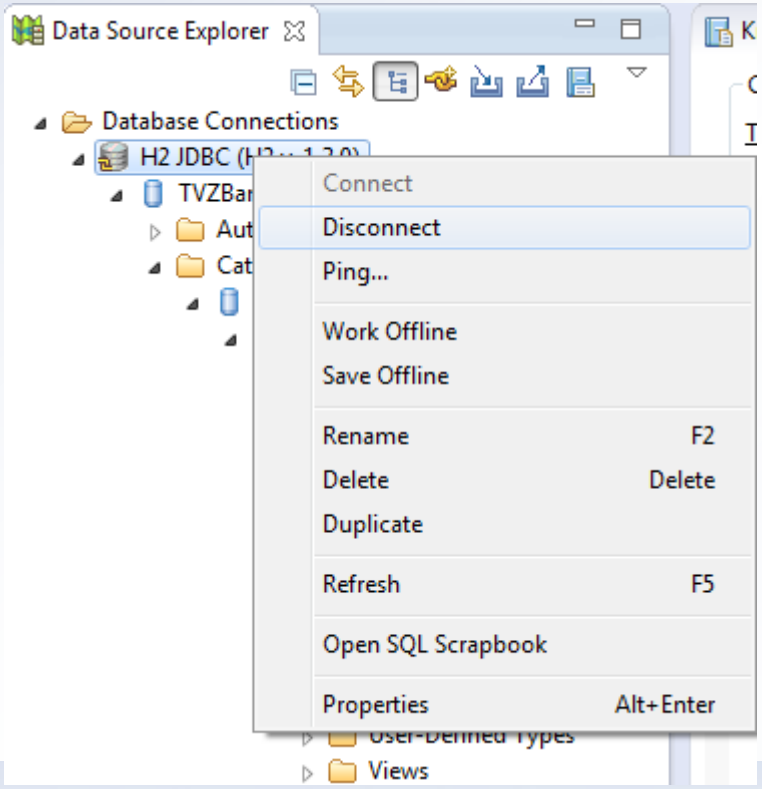
- Rezultate izvođenja upita moguće je vidjeti unutar taba "SQL Results":



Struktura baze podataka i prekidanje veze

- Nakon što se kreira baza podataka, moguće je vidjeti njenu strukturu
- Na kraju je potrebno prekinuti vezu s bazom:





Organizacija programskog koda za korištenje baze podataka

- Dobra praksa je da se sve operacije s bazom podataka izdvoje u zasebnu klasu koja se naziva npr. "DatabaseUtils"
- Na taj način je moguće programski kod zajednički za sve operacije s bazom podataka (spajanje na bazu podataka i zatvaranje konekcije s bazom podataka) držati u zajedničkim metodama koje se pozivaju na različitim mjestima
- Osim toga se na taj način odvajaju "slojevi" aplikacije, pri čemu postoji skup klasa koje kreiraju izgled korisničkog sučelja, skup klasa koje definiraju na koji način se obrađuju događaji na bazi podataka, skup klasa koje služe za komunikaciju s bazom podataka itd.

"Properties" datoteke (1/2)

- U Java programima moguće je koristiti tzv. "Properties" datoteke u kojima se nalaze podaci ključni za rad programa
- Na taj način se izbjegava "hardkodiranje" podataka po programskom kodu, te izmjena tih parametara bez potrebe za ponovnim prevođenjem Java aplikacije
- Takve datoteke sadrže podatke kao što su URL baze podataka, korisničko ime i lozinka, SQL upiti koji se koriste u programu itd., a funkcioniraju na principu "ključ-vrijednost"
- Primjer sadržaja jedne "Properties" datoteke:

```
#Osnovni podaci za spajanje na bazu podataka
bazaPodatakaUrl = jdbc:h2:tcp://localhost/~/TVZBankarstvo

#Podaci za pristupanje bazi podataka
korisnickoIme = tvz
lozinka = tvz
```

"Properties" datoteke (2/2)

- Primjer korištenja "Properties" datoteke u Java programskom kodu:

```
private static final String DATABASE_FILE = "database.properties";

private static Connection connectToDatabase() throws SQLException, IOException
{

   Properties svojstva = new Properties();

   svojstva.load(new FileReader(DATABASE_FILE));

   String urlBazePodataka = svojstva.getProperty("bazaPodatakaUrl");
   String korisnickoIme = svojstva.getProperty("korisnickoIme");
   String lozinka = svojstva.getProperty("lozinka");

   Connection veza = DriverManager.getConnection(urlBazePodataka,
                                                 korisnickoIme,lozinka);

   return veza;
}
```

Primjer metode za spremanje podataka u bazu

- Metoda "saveStudent" prima objekt klase "Student" i odrađuje sve što je potrebno za pripremu i izvršavanje upita za spremanje podataka u bazu:

```
public static void saveStudent(final Student p_student) throws SQLException, IOException
{
   Connection veza = connectToDatabase();

   String queryString =
   "INSERT INTO RAZVOJ.STUDENTI (jmbag, ime, prezime, datum_rodjenja) VALUES (?, ?, ?, ?)";

  PreparedStatement preparedStatement = veza.prepareStatement(queryString);
  preparedStatement.setString(1, p_student.getJmbag());
  preparedStatement.setString(2, p_student.getIme());
  preparedStatement.setString(3, p_student.getPrezime());
  java.sql.Date datum = convertToSQLDate(p_student.getDatumRodjenja());
  preparedStatement.setDate(4, datum);

  preparedStatement.executeUpdate();

  closeConnectionToDatabase(veza);
}
```

Poziv metode za spremanje podataka u bazu

```
public void provjeri() {
   if (isJMBAGUnesen() && isPrezimeUneseno() && isImeUneseno()
                                             && isDatumRodjenjaUneseno()) {

      Student student = new Student();
      student.setJmbag(m_jmbagTextField.getText());
      student.setPrezime(m_prezimeTextField.getText());
      student.setIme(m_imeTextField.getText());
      student.setDatumRodjenja((Date) m_datumRodjenjaTextField.getValue());

      try {
         DatabaseUtils.saveStudent(student);
      } catch (Throwable e) {
          e.printStackTrace();
      }
   }
   else {
      throw new RuntimeException("Niste unijeli sve podatke!")
   }
}
```