

# JDBC

---

# Sadržaj

---

Osnove baza podataka

Java aplikacije i pristup bazi podataka pomoću JDBC-a

JDBC arhitektura

H2 baza podataka i Eclipse DTP *plugin*

Programski kod za spajanje na bazu podataka

Uspostavljanje veze s bazom podataka

Pripremanje i izvršavanje upita

Kreiranje tablice „STUDENT”

Dohvaćanje podataka iz tablice „STUDENT”

Korištenje klase „PreparedStatement”

Zatvaranje veze s bazom podataka

Korištenje transakcija prilikom izvršavanja upita nad bazom podataka

# Osnove baza podataka

---

- Relacijske baze podataka (engl. *relational databases*) predstavljaju organizirane zbirke podataka
- Korištenjem mehanizama iz RDBMS (engl. *Relational Database Management System*) moguće je pouzdano upravljati svim operacijama s bazom podataka, kao što su kreiranje tablica u bazi, spremanje podataka, dohvaćanje, ažuriranje, brisanje, korištenje transakcija itd., uz zadržavanje konzistentnosti stanja baze podataka
- Za rad s bazom podataka koristi se SQL (engl. *Structured Query Language*) jezik koji omogućava izvršavanje operacija nad bazom podataka
- Najčešće korištene baze podataka su Microsoft SQL, Oracle, Sybase, IBM DB2, Informix, PostgreSQL, MySQL, Apache Derby i **H2**

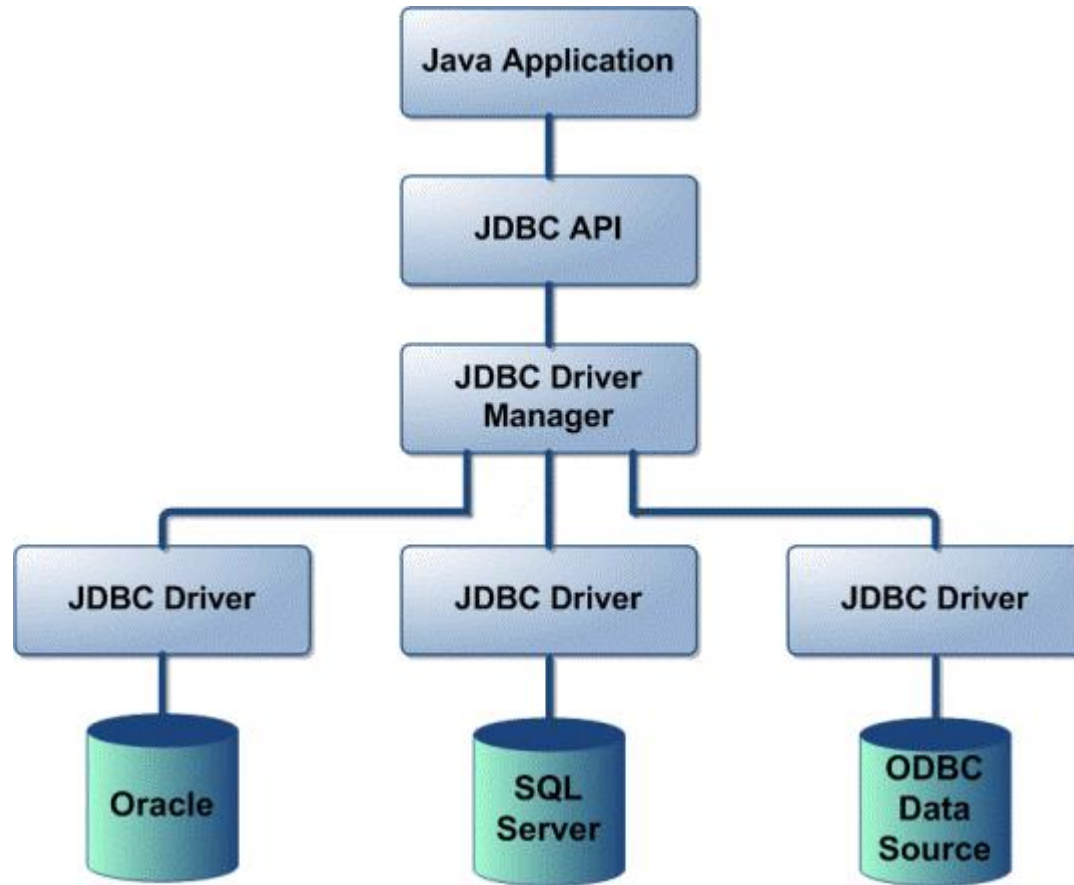
# Java aplikacije i pristup bazi podataka pomoću JDBC-a

---

- Za pristup relacijskim bazama podataka Java SE sadrži JDBC
- JDBC (engl. *Java Database Connectivity*) omogućava spajanje Java aplikacija na sve vrste baze podataka uz korištenje odgovarajućeg JDBC *drivera* (u obliku „JAR” datoteke)
- Omogućava tri programske aktivnosti:
  - Ostvarivanje veze s izvorom podataka (engl. *data source*) – bazom podataka
  - Pripremanje i izvršavanje SQL upita (engl. *queries*) koje uključuju spremanje dohvaćanje, ažuriranje i brisanje podataka iz baze
  - Zatvaranje veze s izvorom podataka

# JDBC arhitektura

---



- Iz Java aplikacije poziva se programski kod iz JDBC biblioteke
- JDBC pomoću *drivera* komunicira s bazom podataka korištenjem SQL upita
- *Driveri* SQL pozive pretvaraju u odgovarajući komunikacijski protokol za određenu bazu podataka

# H2 baza podataka

---

- H2 je *open source* relacijska baza podataka u cijelosti implementirana u Javi
- Vrlo popularna među Java programerima i često se koristi prilikom razvoja i prije prebacivanja na testnu okolinu
- Može se besplatno preuzeti sa stranice „<http://www.h2database.com/html/main.html>”
- Posljednja stabilna verzija je 1.4.195 (objavljena 23.04.2017.)
- Jednostavna za korištenje
- Podržava „embedded” i „klijent-server” način rada
- Lako se integrira u razvojno okruženje Eclipse

# Eclipse DTP *plugin*

---

- engl. *Data Tools Platform*
- Služi za dodavanje nove „perspektive” u Eclipse za rad s bazom podataka: Database Development
- Omogućava dodavanje *drivera* za bazu podataka, kreiranje baza podataka, izvršavanje SQL upita itd.
- Osim DTP *plugin*a za izvršavanje upita nad bazom podataka moguće je koristiti i H2 administracijsku konzolu

# Programski kod za spajanje na bazu podataka

---

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

```
try {  
    Connection veza = DriverManager.getConnection(  
        "jdbc:mojDriver:mojaBaza", "mojLogin", "mojaLozinka");  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

- Statička metoda „getConnection” iz klase „DriverManager” služi za kreiranje „Connection” objekta i prima parametre koji definiraju URL baze podataka, korisničko ime i lozinku za pristupanje bazi
- Navedeni programski kod baca označenu iznimku „SQLException” koja obuhvaća sve potencijalne probleme u radu s bazom podataka



# Programski kod za spajanje na bazu podataka

---

- U konkretnom slučaju podaci za spajanje na H2 bazu podataka mogu biti sljedeći:
  - URL: `jdbc:h2:tcp://localhost/~Java-2018`
  - Korisničko ime: `student`
  - Lozinka: `student`
- URL koji označava bazu podataka sastoji se od sljedećih dijelova:
  - „`jdbc:h2`” – označava tip baze podataka
  - „`tcp://localhost/~`” – lokacija baze podataka
  - „`Java-2018`” – naziv baze podataka

# Pripremanje i izvršavanje upita

---

- Da bi se mogao kreirati upit nad bazom podataka, korištenjem objekta koji predstavlja vezu s bazom potrebno je kreirati objekt klase „Statement”:

```
Statement stmt = veza.createStatement();
```

- Objekt klase „Statement” omogućava pozivanje metode „executeQuery” koja izvršava upit nad bazom podataka i prikuplja rezultate upita:

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tablica");
```

- Objekt tipa „ResultSet” sadrži strukturu podataka koja opisuje rezultat izvođenja upita nad bazom podataka koji se može sastojati i od više zapisa (redaka)
- Navedeni upit **"SELECT a, b, c FROM Tablica"** dohvaća podatke iz stupaca „a”, „b” i „c” svih redaka tablice „Tablica” bez dodatnih kriterija filtriranja

# Pripremanje i izvršavanje upita

---

- Pomoću „while” petlje može se napisati programski kod koji će dohvatiti i ispisati proizvoljan broj redaka koji su dobiveni kao rezultat izvršavanja upita nad bazom podataka i spremljeni u objekt tipa „ResultSet”:

```
while (rs.next()) {  
    int x = rs.getInt("a");  
    String s = rs.getString("b");  
    float f = rs.getFloat("c");  
}
```

- Vrijednost svakog stupca u svakom retku ima svoj tip u bazi podataka pa je kod njezinog dohvaćanja potrebno koristiti „get” metodu koja u svom nazivu koristi i tip podatka koji dohvaća, npr. „getInt”, „getString”, „getFloat” itd.

# Kreiranje tablice „STUDENT”

---

- SQL naredba za kreiranje tablice „STUDENT” može izgledati ovako:

```
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)  
);
```

# Dohvaćanje podataka iz tablice „STUDENT”

---

```
Connection veza = DriverManager
    .getConnection("jdbc:h2:tcp://localhost/~Java-2018", "student", "student");
```

```
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 = (Date) rs.getDate("datum_rodjenja");
    Instant instant = Instant.ofEpochMilli(date.getTime());
    LocalDate localDate = LocalDateTime.ofInstant(instant, ZoneId.systemDefault()).toLocalDate();

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

Pročitani redak: 1 0036374849 Pero Perić  
1994-01-01  
Pročitani redak: 2 0024568238 Ivo Ivić  
1993-01-01

# Korištenje klase „PreparedStatement”

---

- 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** čiji objekt vraća metoda „prepareStatement” sadržana u objektu tipa „Connection”
- Korištenjem klase PreparedStatement upit se ne prevodi više puta (kao što je to u slučaju objekata tipa Statement), već samo jednom
- Moguće je korištenje promjenjivih parametara koji se označavaju s „?” i time omogućavaju višestruko izvođenje istog upita s drugim parametrima, npr.

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

# Korištenje klase „PreparedStatement”

---

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

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

- Izvršavanje samih upita u tom slučaju obavlja se korištenjem metode „executeUpdate”:

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

- Nakon ponovnog pokretanja upita za dohvaćanje podataka iz tablice „STUDENTI” dobivaju se sljedeći rezultati:

```
Pročitani redak: 1 0036374849 Pero Perić 1994-01-01  
Pročitani redak: 2 0024568238 Željko Ivić 1993-01-01
```

# Korištenje klase „PreparedStatement“

---

- Klasa PreparedStatement koristi se i kod spremanja novih redaka u bazu podataka, što je moguće postići korištenjem sljedeće naredbe:

```
PreparedStatement stmt = conn.prepareStatement(  
    "INSERT INTO STUDENTI (JMBAG, IME, PREZIME, DATUM_RODJENJA) VALUES (?, ?, ?, ?)";  
  
    stmt.setString(1, jmbagStudenta);  
    stmt.setString(2, imeStudenta);  
    stmt.setString(3, prezimeStudenta);  
    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 unutar „finally” bloka (ili koristiti „try-with-resources”):

```
Connection veza = null;
Statement stmt = null;
ResultSet rs = null;

try {
    veza = DriverManager.getConnection(...);
    stmt = veza.createStatement();
    rs = stmt.executeQuery("SELECT * FROM STUDENTI");
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    try {
        rs.close();
        stmt.close();
        veza.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

# Korištenje transakcija prilikom izvršavanja upita nad bazom podataka

---

- Transakcija predstavlja skup jedne ili više operacija (engl. *statements*) koje se moraju izvršiti zajedno kao da se radi o jednoj transakciji (atomarno)
- Svaka nova veza s bazom podataka konfigurirana je tako da uvijek automatski sprema sve promjene u bazu podataka (engl. *auto-commit mode*)
- Za promjenu tih predefiniranih postavki potrebno je koristiti metodu „`setAutoCommit(false)`” nad objektom koji predstavlja vezu:

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

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

# Korištenje transakcija prilikom izvršavanja upita nad bazom podataka

---

- Primjer: oba podatka se upisuju zajedno 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);
```

# Pitanja?

---