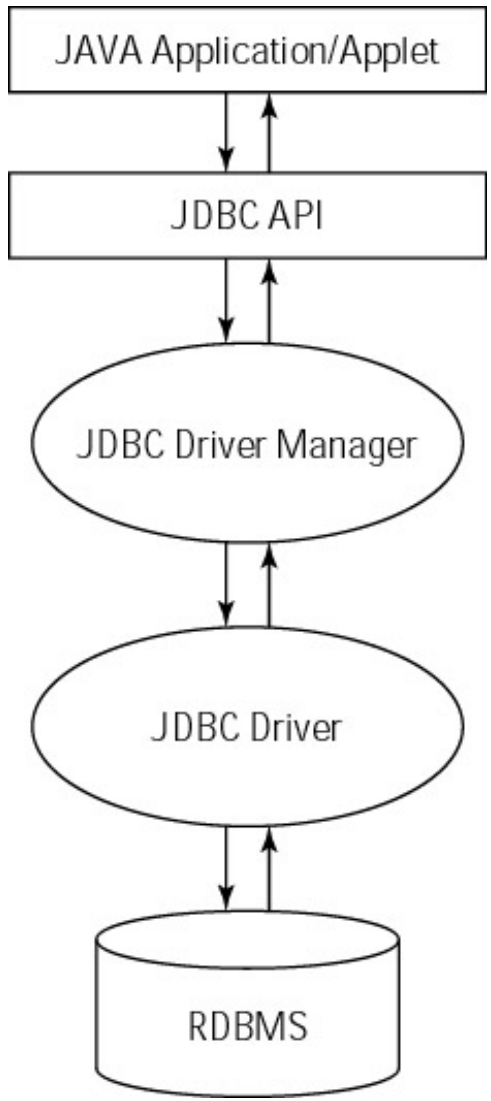
Sedma Nedelja
– Programski SQL –

Autori: Mr. Miloš Cvetanović

- **SQL/CLI standard (call-level interface)**
- **ODBC (Open DataBase Connectivity)**
- **JDBC**
  - java.sql
  - javax.sql
- **Tip 1:**
  JDBC – ODBC
- **Tip 2:**
  JDBC – CLI
- **Tip 3:**
  JDBC – Nezavisni mrežni DB protokol
- **Tip 4:**
  JDBC – Specifični mrežni DB protokol



JDBC – kreiranje veze

- **Sve metode koje će biti opisane nalaze se u paketu koji treba biti uveden sa:**

  import java.sql.*;

- **Uspostavljanje veze sa bazom podataka, što uključuje dva koraka:**
  1. učitavanje drajvera

  Class.forName("jdbc.DriverXYZ");

  Nije potrebno praviti instancu drajvera i registrovati je sa DriverManager-om jer će to obaviti poziv Class.forName automatski.

  2. pravljenje veze (konekcije)

  String url = "jdbc:odbc:XYZ";Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");

JDBC – kreiranje i izvršavanje SQL iskaza

- **Da bi se DBMS prosledio željeni SQL iskaz neophodno je kreirati Statement objekat, za šta će biti iskorišćen prethodno kreiran Connection objekat.**

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

- **Ukoliko SQL iskaz ne vraća rezultat (CREATE, INSERT, UPDATE, DELETE)**

```
String SQLStm = "jdbc:odbc:XYZ";stmt.executeUpdate(SQLStm);
```

(ova metoda vraća celobrojnu vrednost koja predstavlja koliko je redova promenjeno izvršavanjem SQL iskaza. Za DDL naredbe povratna vrednost je 0)

- **Ukoliko SQL iskaz vraća rezultat, što je u slučaju SELECT naredbe, onda je potrebno koristiti objekat za prihvatanje rezultata ResultSet**

```
String SQLStmQuery = "SELECT * ...";
ResultSet rs = stmt.executeQuery(SQLStmQuery);
```

- **Dobijeni rezultat se obrađuje red po red, a za dohvatanje vrednosti pojednih kolona u posmatranom redu se koriste odgovarajuće get metode (getString, getInt, itd.). Obično se obrada vrši u petlji:**

```
while (rs.next()) {
        String s = rs.getString("column1");
        float n = rs.getFloat("column2");
        int m = rs.getInt("column3");
        Boolean b = rs.getBoolean("column4");
}
```

- **U slučaju kada se želi obrada različita od obrade red po red, potrebno je definistai parametre kursora pri kreiranju Statement objekta:**

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
```

- **Za dohvatanje se može koristiti naziv kolone kao "column1" (String), ili pak redni broj kolone (int):**

```
String s = rs.getString(1);
```

- **Upotrebom rezultata moguće je vršiti promene u posmatranoj tabeli (ukoliko se radi u upitu koji odgovara pravilima ažurabilnog pogleda), ali je pre toga neophodno definisati tip rezultata (tip kursora). Za takve potrebe koriste odgovarajuće update metode (updateString, updateFloat, updateInt, itd.)**

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet uprs = stmt.executeQuery(SQlString);
uprs.updateFloat("Column1", 102.45f);
uprs.updateRow();
```

- **U slučaju kada se u rezultat želi dodati novi red**

```
uprs.moveToInsertRow();…uprs.updateFloat("Column1",
102.45f);uprs.insertRow();
```

- **Ukoliko se želi obrisati određeni red**

```
uprs.deleteRow();
```

JDBC – pripremljeni iskazi

- **U nekim slučajevima je potrebno izvršavati neki SQL iskaz više puta. U tim slučajevima je korisno napraviti pripremljeni SQL iskaz, tj. PreparedStatement objekat. Pripremljen iskaz može biti upotrebljen bez parametara, mada je u najvećem broju slučajeva njegova upotreba upravo takva da se paramteri koriste.**

```
PreparedStatement updateTabela1 = con.prepareStatement( "UPDATE Tabela1
SET Atribute1 = ? WHERE Atribute2 LIKE ?");
updateSales.setInt(1, 43);
updateSales.setString(2, "TestString");
updateSales.executeUpdate():
```

- **Parametri su u SQL iskazu obeleženi znakom pitanja, i oni će pre samog izvršavanja iskaza biti prosleđeni odgovarajućim set metodama (setInt, setString, itd.)**

- **Nakon kreiranja Connection objekta, ona se nalazi u auto-commit modu rada, što znači da se svaki SQL iskaz posmatra kao zasebna transakcija. Ukoliko se želi postići da veči broj SQL iskaza predstavljaju jednu transakciju, onda je potrebno eksplicitno isključiti auto-commit režim rada. Međutim u tom slučaju je neophodno eksplicitno izvršiti commit u cilju završavanja transakcije.**

```
con.setAutoCommit(false);
...
con.commit();
con.setAutoCommit(true);
```

- **Transakcijom se smatraju svi SQL iskazi u periodu između dva poziva metode commit. U slučajevima kada je to potrebno moguće je anulirati efekte tekuće transakcije pozivom metode rollback. Nivo izolacije tranksakcija se postavlja sa:**

```
con.setTransactionIsolation(TRANSACTION_READ_COMMITTE);
```

- **Pozivi uskadištenih procedura obavlja se upotrebom CallableStatement objekta, koji poput PreparedStatement objekta može da prihvata ulazne parametre, ali takođe i izlazne odnosno ulazno-izlazne paramtere takođe.**

```
CallableStatement cs = con.prepareCall("{call Procedura1}");
ResultSet rs = cs.execute();
```

- **Pri radu sa JDBC, pozivi metoda u slučaju greške mogu baciti izuzetke tipa SQLException, i detalji mogu biti pročitani getMessage, getSQLState, getErrorCode metodoma.**