## PRISTUP BAZI PODATAKA IZ JAVA PROGRAMSKOG KODA

* Spajanje na MS SQL SERVER i rad s CRUD operacijama

### Spajanje na Microsoft SQL Server i rad s CRUD operacijama

#### SQL JDBC Driver

Da bismo se iz Java aplikacije mogli spojiti na Microsoft SQL Server, potreban nam je JDBC *driver (skr. Java Database Connectivity).* Moguće je preuzeti ga na sljedećoj poveznici:

* + <https://www.microsoft.com/en-us/download/details.aspx?id=54670>

Nakon preuzimanja, uključujemo .jar datoteku kao dio projekta u kojem će se pisati kod za rad s bazom, ali ako se aplikacija sastoji od više projekata (na primjer, projekt tipa class library i glavni web projekt), potrebno je *driver* uključiti u oba projekta, pod pretpostavkom da ćemo kod za rad s podacima pisati u *class library* projektu.

##### Microsoft SQL Server konfiguracija

Preduvjet za pristup SQL Serveru je ispravno konfigurirani protokoli i PORT 1433. U Windowsima je potrebno otvoriti postavke:

* Computer Management -> Services And Applications -> SQL Server Configuration -> SQL Server Network Configuraration -> Protocols for SQL
  + Uključiti opciju TCP/IP protokol
* U postavkama TCP/IP pod karticom IP Addresses, opcija IPALL mora imati sljedeće opcije:
  + TCP port: 1433
  + TCP Dynamic ports: prazno

Nakon izmjena otići pod opciju SQL Server Services i napraviti restart instance SQL Servera. Time smo omogućili JDBC driveru pristup MS SQL Serveru.

#### Sučelje DataSource

Za pristup bazi podataka iz Java koda, potrebno je kreirati klasu SqlDataSource, koja ima metodu povratnog tipa DataSource (paket javax.sql.DataSource). Možemo metodu nazvati createDataSource. U metodi je potrebno upisati točne podatke za pristup MS SQL Serveru, što možemo vidjeti na sljedećem primjeru (najprikladnije je napraviti statičku metodu):

import com.microsoft.sqlserver.jdbc.SQLServerDataSource;

import javax.sql.DataSource;

public class SqlDataSource

{

public static DataSource createDataSource()

{

SQLServerDataSource ds = new SQLServerDataSource();

ds.setServerName("localhost");

ds.setDatabaseName("NekaBaza");

ds.setUser("korisnik");

ds.setPassword("lozinka");

ds.setPortNumber(1433);

return ds;

}

}

Prethodna metoda ima povratni tip DataSource, a ovo je moguće jer SQLServerDataSource kojeg instanciramo i vraćamo kao rezultat implementira sučelje DataSource.

#### Priprema modela

Uzmimo za primjer bazu koja sadrži podatke s tablicama Drzava i Kupac, te jedna proceduru koja vraća podatke određenih kolona iz obje tablice. U tom slučaju, potrebno je napraviti dva modela koji imaju ista svojstva kao tablice u bazi, te jedan ViewModel sa svim svojstvima iz procedure. Ove modele koristit ćemo kao objekte u koje ćemo učitati podatke iz baze u memoriju.

public class Kupac {

int iDKupac, idGrad;

String ime, prezime, email;

public Kupac() { }

public Kupac(int iDKupac, String ime, String prezime, String email) {

this.iDKupac = iDKupac;

this.ime = ime;

this.prezime = prezime;

this.email = email;

}

public int getIdGrad() { return idGrad; }

public void setIdGrad(int idGrad) { this.idGrad = idGrad; }

public int getiDKupac() { return iDKupac; }

public void setiDKupac(int iDKupac) { this.iDKupac = iDKupac; }

public String getIme() { return ime; }

public void setIme(String ime) { this.ime = ime; }

public String getPunoImePrezime() { return ime + " " + prezime; }

public String getPrezime() { return prezime; }

public void setPrezime(String prezime) { this.prezime = prezime; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

}

public class Drzava {

int iDDrzava;

String naziv;

public Drzava() { }

public Drzava(int iDDrzava, String naziv) {

this.iDDrzava = iDDrzava;

this.naziv = naziv;

}

public int getiDDrzava() {

return iDDrzava;

}

public void setiDDrzava(int iDDrzava) {

this.iDDrzava = iDDrzava;

}

public String getNaziv() {

return naziv;

}

public void setNaziv(String naziv) {

this.naziv = naziv;

}

}

public class KupciZaDrzavuViewModel {

String ime, prezime, email, grad, drzava;

int kupacId;

public KupciZaDrzavuViewModel() {}

public KupciZaDrzavuViewModel(String ime, String prezime, String email, String grad, String drzava, int kupacId) {

this.ime = ime;

this.prezime = prezime;

this.email = email;

this.grad = grad;

this.drzava = drzava;

this.kupacId = kupacId;

}

public String getIme() { return ime; }

public void setIme(String ime) { this.ime = ime; }

public String getPrezime() { return prezime; }

public void setPrezime(String prezime) { this.prezime = prezime; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public String getGrad() { return grad; }

public void setGrad(String grad) { this.grad = grad; }

public String getDrzava() { return drzava; }

public void setDrzava(String drzava) { this.drzava = drzava; }

public int getKupacId() { return kupacId; }

public void setKupacId(int kupacId) { this.kupacId = kupacId; }

}

Na kraju, potrebno je napraviti proceduru koja će vratiti podatke iz spojenih tablica:

create proc KupciZaDrzavu

@drzavaId int

as

select top 50 k.IDKupac, k.Ime, k.Prezime, k.Email, g.Naziv as 'Grad', d.Naziv as 'Drzava'

from Kupac k

inner join Grad g on k.GradID = g.IDGrad

inner join Drzava d on g.DrzavaID = d.IDDrzava

where d.IDDrzava = @drzavaId

order by k.IDKupac

#### Sučelje CallableStatement

Imamo sve što je potrebno za pristup bazi i možemo napraviti prvu komunikaciju. Za taj slučaj ćemo koristiti CallableStatement, koji služi za pozivanje pohranjenih procedura u bazi - s ulaznim ili izlaznim parametrima, kao i bez njih. Za naredne slučajeve, koristimo bazu koja ima tablice za prethodno definirane modele.

Dvije najvažnije metode sučelja CallableStatement:

* executeUpdate() – poziva operacije INSERT, UPDATE, DELETE
* executeQuery() – poziva operaciju SELECT

Sljedeći primjer koda vrši poziv prema SQL bazi i poziva proceduru DodajNovuDrzavu s dva parametra, te vraća novi id zapisa iz tablice koristeći izlazni parametar.

* Ako ne trebamo parametre, uklonimo dio iza naziva procedure - (?,?)

Primjer metode executeUpdate()

import javax.sql.DataSource;

public class SqlRepo

{

public int dodajNovuDrzavu(String naziv) {

//Dohvaćamo naš DataSource objekt

DataSource ds = SqlDataSource.createDataSource();

//Varijabla za izlazni parametar

int noviId = 0;

//Dohvaćamo vezu tipa Connection iz DataSource objekta

//Radimo novi objekt tipa CallableStatement i vraćamo njegovu instancu pozivom metode conn.PrepareCall()

try (Connection conn = ds.getConnection();

CallableStatement stmt = conn.prepareCall("{CALL DodajNovuDrzavu(?,?)}")) {

// registriramo jedan ulazni i jedan izlazni parametar

stmt.setString("naziv", naziv);

stmt.registerOutParameter(2, Types.INTEGER);

stmt.executeUpdate();

// Dohvaćamo izlazni parameter nakon izvršavanja

noviId = stmt.getInt(2);

} catch (Exception e) { }

return noviId;

}

}

Primjer metode executeQuery():

public List<KupciZaDrzavuViewModel> dohvatiKupceZaDrzavu(int drzavaId)

{

DataSource ds = SqlDataSource.createDataSource();

List<KupciZaDrzavuViewModel> data = new ArrayList<KupciZaDrzavuViewModel>();

try (Connection conn = ds.getConnection();

CallableStatement stmt = conn.prepareCall("{CALL KupciZaDrzavu(?)}")) {

stmt.setInt(1, drzavaId);// vrijednost 1. parametra

//executeQuery() ima povratni tip ResultSet s podacima koje vrati procedura // prolazimo kroz svaki zapis u ResultSet-u u stvaramo novi objekt tipa KupciZaDrzavuViewModel i dodajemo ga u kolekciju

ResultSet rs = stmt.executeQuery();

while (rs.next())

{

KupciZaDrzavuViewModel vm = new KupciZaDrzavuViewModel();

// rs.get kao parametar prima naziv kolone koju procedura sadrži u rezultatu

vm.setDrzava(rs.getString("Drzava"));

vm.setGrad(rs.getString("Grad"));

vm.setIme(rs.getString("Ime"));

vm.setPrezime(rs.getString("Prezime"));

vm.setEmail(rs.getString("Email"));

vm.setKupacId(rs.getInt("IDKupac"));

data.add(vm);

}

} catch (Exception e) { }

return data;

}

Identično prethodnom primjeru, dohvaćamo DataSource, Connection, te radimo instancu CallableStatement-a pozivom metode prepareCall kojoj smo proslijedili naziv procedure KupciZaDrzavu.

#### Sučelje PreparedStatement

Isto kao i kod sučelja CallableStatement, PreparedStatement ima metode:

* executeUpdate() – poziva operacije INSERT, UPDATE, DELETE
* executeQuery() – poziva operaciju SELECT

Razlike između CallableStatement i PrepearedStatement su:

* Radi se instanca PreparedStatement – pozivom conn.prepareStatement() metode
  + Metoda kao parametar prima SQL upit umjesto naziva procedure
  + Nema ključne riječi CALL
  + Nema vitičastih zagrada {}
  + Ne treba pisati proceduru

Primjer SELECT operacije korištenjem PreparedStatement-a (metoda executeQuery() ):

public List<Kupac> dohvatiKupce()

{

success = true;

List<Kupac> kupci = new ArrayList<>();

try (Connection conn = ds.getConnection();

PreparedStatement stmt = conn.prepareStatement("SELECT \* FROM Kupac")) {

ResultSet rs = stmt.executeQuery();

while (rs.next())

{

Kupac k = new Kupac();

k.setEmail(rs.getString("Email"));

k.setIdGrad(rs.getInt("GradID"));

k.setiDKupac(rs.getInt("IDKupac"));

k.setIme(rs.getString("Ime"));

k.setPrezime(rs.getString("Prezime"));

kupci.add(k);

}

} catch (SQLException ex)

{

success = false;

error = ex.getMessage();

}

return kupci;

}

* Metoda executeUpdate() koristi se na sličan način
  + Npr. INSERT INTO Drzava(Naziv) VALUES(?), nakon čega slijedi registriranje ulaznih parametara statement.setString(1, "Nova država"); - pri čemu je broj 1 redni broj parametra kojeg registriramo
    - Ovakav upit dodao bi jedan novi zapis u tablicu Drzava

Primjer brisanja zapisa s jednim parametrom:

public void obrisiDrzavu(int id)

{

try (Connection conn = ds.getConnection();

PreparedStatement stmt = conn.prepareCall("DELETE FROM Drzava WHERE IDDrzava = ?")) {

// Pažnja: Postavljanje parametara napravljeno je korištenjem rednog broja parametra, a ne naziva kolone. Treba paziti na redoslijed kojim se navode u SQL izjavi

stmt.setInt(1, id);

stmt.executeUpdate();

} catch (Exception e)

{

// Obrada iznimke

}

}