Lekcija 4

Java programiranje sa bazama podataka

dr Svetlana Cvetanović

JAVA PROGRAMIRANJE SA BAZAMA PODATAKA

Uvod

Uvod

01

JDBC

- *Kreiranje JDBC Aplikacije*
- *JDBC Driver*
- *URL-a za konekciju na bazu*

02

Interfejsi

- *Statement Objekti*
- *PreparedStatement Objekti*
- *ResultSet*

03

Vežba: Preduslovi za pisanje Java koda

- *Vežba: primer Java koda*

04

Zaključak

UVOD

Šta ćemo naučiti u ovoj lekciji?

U ovom predavanju se govori o standardni Java API koji omogućava da se veliki broj baze podataka različitih proizvođača koriste iz Java programskog jezika. JDBC biblioteka omogućava izvršenje svakog posla koji omogućava:

- *Uspostavljenje konekcije sa bazom podataka.*
- *Kreiranje SQL ili MySQL naredbi.*
- *Izvršenje SQL ili MySQL upita u bazi podataka.*
- *Pregledavanje i modifikacija rezultujućih slogova.*

JDBC API se sastoji od dva nivoa:

- *JDBC API*
- *JDBC Driver manager*

***JDBC API** koristi JDBC Driver manager. **JDBC Driver manager** je u stanju da podrži više različitih drivera koji su konektovani na više različitih baza.*

JDBC aplikacija se kreira kroz šest osnovnih koraka:

1. *importovanje paketa,*
2. *registrovanje JDBC drajvera,*
3. *otvaranje konekcije,*
4. *izvršavanje upita,*
5. *izvdajanje podataka iz skupa rezultata,*
6. *čišćenje okruženja*

JDBC

- *Kreiranje JDBC Aplikacije*
- *JDBC Driver*
- *URL-a za konekciju na bazu*

01

ŠTA JE JDBC?

Standardni Java API koji omogućava da se veliki broj baze podataka različitih proizvođača koriste iz Java programskog jezika

JDBC je skraćeno od Java DataBase Connectivity i prestavlja standardni Java API koji omogućava bazama podataka da se nezavisno koriste iz Java programskog jezika sa velikim brojem različitih proizvođača samih baza.

JDBC biblioteka uključuje API koji omogućava izvršenje svakog posla koji se obično povezuje sa radom nad bazama:

1. Uspostavljenje konekcije sa bazom podataka.
2. Kreiranje SQL ili MySQL naredbi.
3. Izvršenje SQL ili MySQL upita u bazi podataka.
4. Pregledavanje i modifikacija rezultujućih slogova.

JDBC je specifikacija obezbeđuje kompletan skup interface-a koji omogućavaju portabilnost nad različitim bazama. Java može da se koristi za različite vrste aplikacija:

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs).

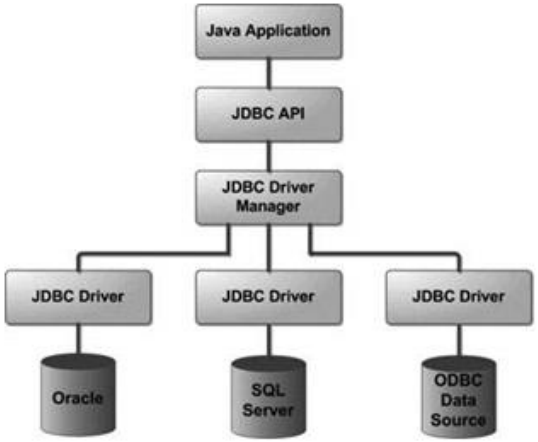
Zahtevi: *Za dalji rad na ovom predmetu neophodno je posedovati osnovno znanje Java programiranja kao i osnovno poznavanje SQL jezika.*

O ČEGA SE SASTOJI JDBC API?

Sastoji se od dva nivoa: JDBC API i JDBC Driver API

JDBC API se sastoji od dva nivoa:

- **JDBC API**
- **JDBC Driver Manager**



Slika 1. Česte JDBC komponente

***JDBC API** koristi JDBC Driver manager i drivere za konkretnu bazu kako bi omogućio heterogenu konekciju na različite RDBMS-e.*

JDBC menager obezbeđuje da driveri za korektnu bazu iskoriste za sve izvore podataka. ***JDBC Driver manager** je u stanju da podrži više različitih drivera koji su konektovani na više različitih baza.*

JDBC API obezbeđuje naredne interface-e i klase:

DriverManager: Ova klasa barata listom drivera za različite baze podataka, povezuje zahteve za konekciju iz java aplikacije sa odgovarajućim driver-om korišćenjem pod protokola. Koristi se onaj driver koji se prvi registruje i prepozna podprotokol.

Driver: Ovaj interface je zadužen za komunikaciju sa serverom baze. Naša interakcija će se retko odvijati direktno sa driver objektima, najčešće ćemo koristiti objekat tipa DriverManager.

Connection: Ovo je interface sa svim metodama za konektovanje sa bazom. Objekat konekcije predstavlja komunikacioni kontekst, sve komunikacije sa bazom idu kroz konekciju.

Statement: Objekti ovog tipa predstavljaju upite ka bazi podataka. Ovi upiti se izvršavaju kroz Connection objekat.

ResultSet: Ovi objekti sadrže podatke iz baze podataka nakon izvršavanja SQL upita korišćenjem Statement objekta i Connection objekta. ResultSet se koristi kao iterator koji nam omogućava da prolazimo kroz skupove podataka.

SQLException: Predstavlja klasu za izuzetke prilokom rada sa JDBC-em.

Kreiranje JDBC Aplikacije

01

KAKO KREIRATI APLIKACIJU KORIŠĆENJEM JDBC-A?

Kroz šest koraka

Šest osnovnih koraka za kreiranje JDBC aplikacije:

1. **Importovanje paketa:** zahteva uključivanje paketa koji sadrže JDBC klase neophodne za programiranje sa bazama podataka. Često je dovoljno iskoristiti *import java.sql.**.
2. **Registrovanje JDBC drajvera:** Zahteva inicijalizaciju drajvera tako da se može otvoriti komunikacioni kanal sa Jave-om.
3. **Otvaranje konekcije:** Zahteva korišćenje metode *DriverManager.getConnection()* kako bi se kreirao Connection object, koji predstavlja fizičku konekciju sa bazom podataka.
4. **Izvršavanje upita:** zahteva korišćenje objekta tipa Statement kako bi se kreirale SQL naredbe nad bazom podataka.
5. **Izvdajanje podataka iz skupa rezultata:** Zhateva korišćenje odgovarajuće metode *ResultSet.getXXX()* za pretraživanje podataka iz skupa rezultata.
6. **Čišćenje okruženja:** Zahteva eksplicitno zatvaranja svih resursa baze podataka.

Primer koda:

Ovo je primer koda koji možemo da koristimo kao template za dalji rad sa JDBC aplikacijama

Ovaj primer se konektuje na bazu sa nazivom EMP na našoj lokalnoj mašini i koristi promenljive USER i PASS da predstavi username i password naše baze.

Pre nego što je ovaj projekat moguće pokrenuti potrebno je da se skine mysql jdbc driver i da se doda u naš projekat, napravi baza u mysql-u pod nazivom EMP i da se u njoj napravi tabela Employees i da se u nju dodaju neki zaposlleni. Kolone tabele Employees treba da budu id, first, last i age. Id i first su Integer polja dok su last i age varchar polja.

KORACI 1. – 4.

Importovanje paketa, Registrovanje JDBC drajvera, Otvaranje konekcije i Izvršavanje upita

```
//STEP 1. Import required packages
import java.sql.*;
public class FirstExample {
// JDBC driver name and database URL
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/EMP";
//  Database credentials
static final String USER = "username";
static final String PASS = "password";
public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;    try{
```

slika 1. Korak 1. – importovanje paketa

```
//STEP 2: Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");
//STEP 3: Open a connection
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);
//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
```

Slika 2. Koraci 2. 3. i 4.: Registrovanje JDBC drajvera, Otvaranje konekcije i Izvršavanje upita

KORAK 5.

Izvdajanje podataka iz skupa rezultata

```
//STEP 5: Extract data from result set
while(rs.next()){
//Retrieve by column name
int id  = rs.getInt("id");
int age = rs.getInt("age");
String first = rs.getString("first");
String last = rs.getString("last");
//Display values
System.out.print("ID: " + id);
System.out.print(", Age: " + age);
System.out.print(", First: " + first);
System.out.println(", Last: " + last);
}
```

Slika 3. Korak 5. - Izvdajanje podataka iz skupa rezultata

KORAK 6.

Čišćenje okruženja

```
//STEP 6: Clean-up environment
rs.close();
stmt.close();
conn.close();
}catch(SQLException se){
//Handle errors for JDBC
se.printStackTrace();
}catch(Exception e){
//Handle errors for Class.forName
e.printStackTrace();
}finally{
//finally block used to close resources
try{
if(stmt!=null)
stmt.close();
```

Slika 4. Čišćenje okruženja – I deo

```
}catch(SQLException se2){
}try{
if(conn!=null)
conn.close();
}catch(SQLException se){
se.printStackTrace();
}//end finally try
}//end try
System.out.println("Goodbye!");
}
//end main
}
//end FirstExample
```

Slika 5. Čišćenje okruženja – II deo

JDBC Driver

01

ŠTA JE JDBC DRIVER?

Implementacija definisanog interface-a u JDBC API-u koja omogućava konkretnu vezu sa konkretnim RDBMS-om.

JDBC driver je implementacija definisanog interface-a u JDBC API-u koja omogućava konkretnu vezu sa konkretnim RDBMS-om.

Na primer: korišćenje JDBC drivera nam omogućava da otvorimo konekcije sa bazom i sa njom budemu u interakciji slanjem SQL komandi ili komandi koje obezbeđuju odgovarajući rezultat u Javi.

Paket java.sql koji dolazi sa standardnim JDK-om sadrži veliku količinu klasa sa predefinisanim ponašanjima i omogućava da proizvođači baze koji trebaju da implementuju interface java.sql.Driver ove klase direktno implementiraju.

Import JDBC Paketa

Potrebno je da primenimo import ključnu reč u javi kako bismo dodali podršku za jdbc u našoj klasi.

```
import java.sql.* ;
// for standard JDBC programs
import java.math.* ;
// for BigDecimal and BigInteger support
```

Slika 1. Import JDBC java.sql paketa

REGISTROVANJE JDBC DRIVERA

Proces kada obezbeđujemo konkretnu implementaciju za određenu vrstu baza; Moguće je na dva načina: korišćenjem metode Class.forName() ili metode DriverManager.registerDriver().

Da bismo koristili driver potrebno je da ga prethodno registrujemo u našem programu. Registracija drivera je proces kada obezbeđujemo konkretnu implementaciju za određenu vrstu baza.

Naš driver je potrebno da registrujemo samo jedan put u našem programu i moguće je to uraditi na dva načina, korišćenjem metode Class.forName() ili korišćenjem metode DriverManager.registerDriver().

Način I - Class.forName()

Najčešći način da se registruje driver je korišćenjem metode forName iz klase Class. Ova metoda dodaje dinamički određenu klasu u memoriju, što je automatski i registruje.

Priemer korišćenja:

```
try {
Class.forName("oracle.jdbc.driver.OracleDriver");
} catch(ClassNotFoundException ex) {
System.out.println("Error: unable to load driver class!");
System.exit(1);
}
```

Slika 2. primer korišćenja Class.forName() metode

Moguće je koristiti getInstance za JVM-e koji ne učitavaju klasu direktno pozivom forName metode. U takvom slučaju je potrebno uhvatiti dva izuzetka:

```
try {
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
} catch(ClassNotFoundException ex) {
System.out.println("Error: unable to load driver class!");
System.exit(1);
catch(IllegalAccessException ex) {
System.out.println("Error: access problem while loading!");
System.exit(2);
catch(InstantiationException ex) {
System.out.println("Error: unable to instantiate driver!");
System.exit(3);
}
```

Slika 3. Hvatanje izuzetaka u slučaju poziva Class.forName metode

REGISTRACIJA KORIŠĆENJEM METODE DRIVERMANAGER.REGISTERDRIVER().

Ovu metoda se koristi ukoliko radimo na JVM-u koji ne prati JDK standard

Način II - DriverManager.registerDriver()

Drugi način bi bio da pozovemo statičku metodu registerDriver() klase DriverManager.

Potrebno je koristiti ovu metodu ukoliko radimo na JVM-u koji ne prati JDK standard, jedan ovakav JVM nudi Microsoft.

.

Evo primera za korišćenje ove metode za registraciju OracleDriver-a

```
try {
Driver myDriver = new oracle.jdbc.driver.OracleDriver();    DriverManager.registerDriver(
myDriver );
} catch(ClassNotFoundException ex) {
System.out.println("Error: unable to load driver class!");
System.exit(1);
}
```

Slika 4. Primer korišćenja DriverManager.registerDriver() metode za registraciju OracleDriver-a

URL-a za konekciju na bazu

01

FORMULACIJA URL-A ZA KONEKCIJU NA BAZU

Za uspostavljanje konekcije koristićemo getConnection metodu klase DriverManager

Nakon što smo učitali driver potrebno je uspostavimo konekciju sa bazom. Za uspostavljanje konekcije koristićemo getConnection metodu klase DriverManager.

Postoje tri varijace ove metode:

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

Svaka od njih zahteva URL baze, prva se koristi ukoliko nemamo password, druga ukoliko imamo password i dodatna podešavanja a treća ukoliko imamo samo username i password.

Primeri URL-a za neke česte RDBMS-e:

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql://**hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

Slika 1. Zadebljane delove URL-a iz ovih primera ne treba manjati.

KORIŠĆENJE URL-A

Postoje tri varijace ove metode: sa korišćenjem URL-a, username i password-a i properties objekta

Korišćenje URL-a sa usernamom i passwordom

Najčešća forma getConnection-a zahteva username i password. Primer poziva ove metode:

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password";
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

Korišćenje samo URL-a

Primer koda za korišćenje getConnection-a sa samo url-om.

```
DriverManager.getConnection(String url);
```

Obično baze nude mogućnost unosa username-a i password-a u samom url-u:

```
jdbc:oracle:driver:username/password@database
```

Putpun primer otvaranja konekcije sa username-om i password-om u URL-u:

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

Korišćenje URL-a i Properties objekta

Primer instanciranja konekcije sa bazom korišćenjem properties objekta:

```
import java.util.*;
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );
Connection conn = DriverManager.getConnection(URL, info);
```

ZATVARANJE JDBC KONEKCIJE

Na kraju našeg JDBC programa potrebno je eksplicitno da zatvorimo konekciju sa bazom.

Ukoliko zaboravimo da to uradimo, Javin garbage collector će to uraditi pre ili kasnije, Ipak, ukoliko sami ugasimo konekciju imaćemo efikasniji kod jer neće biti potrebe da se drži konekcija otvorena kada ne postoji. Oslanjati se na garbage kolektor je loša praksa.

Interfejsi

- *Statement Objekti*
- *PreparedStatement Objekti*
- *ResultSet*

02

KOJI SE INTERFEJSI KORISTE?

To su interfejs Statement, PreparedStatement, CallableStatement

Svaki od interfejsa ima svoju namenu:

Interfaces	Predloženo korišćenje
Statement	Koristi se za pristup bazi. Ovaj interface ne dozvoljava parametre.
PreparedStatement	Koristi se kada planiramo da izvršimo isti upit više puta sa isti ili različitim parametrima. Dozvoljava input parametara za vreme rada programa.
CallableStatement	Koristi se za skladištenje procedura.

Slika 1. Interfejsi

Statement Objekti

02

KREIRANJE STATEMENT OBJEKTA

Pre nego što možemo da koristimo Statement objekat potrebno je da ga napravimo pozivanjem metode createStatement iz klase Connection

```
Statement  stmt = null;
try {
stmt = conn.createStatement( );
. . .
} catch (SQLException e) {
. . .
} finally {
. . .
}
```

Slika 1. Pozivanje metode createStatement iz klase Connection

Kada smo napravili Statement objekat možemo da koristimo jednu od narednih metoda da bismo je izvršili:

- **boolean execute (String SQL)**: Vraća boolean vrednost true ukoliko ResultSet treba da bude vraćen, u suprotnom varaća false. Metodu execute koristimo kada izvršavamo DDL naredbe.
- **int executeUpdate (String SQL)**: Vraća broj redova na koje se naš SQL upit odnosio, koristi se sa INSERT, UPDATE, ili DELETE upitima.
- **ResultSet executeQuery (String SQL)**: Vraća ResultSet objekat. Koristi se kao metoda koja vraća ResultSet i koristi se sa upitima tipa Select.

ZATVARANJE STATEMENT OBJEKTA

Dobra je praksa zatvarati i statement objekte; Prepared stament objekti se koriste kako bismo izbegli SQL injection napade

Ukoliko prvo zatvorimo konekciju stameneti će biti zatvoreni, preporučuje se da ručno zatvaramo statemente pre nego da zatvorimo konekciju.

```
Statement stmt = null;
try {
stmt = conn.createStatement( );
. . .
} catch (SQLException e) {
. . .
} finally {
stmt.close();
}
```

Slika 2. Zatvaranje Statement Objekta

PreparedStatement Objekti

02

KREIRANJE PREPAREDSTATEMENT OBJECT

Koristi se kada planiramo da izvršimo isti upit više puta sa isti ili različitim parametrima.

Prepared stament objekti se koriste kako bismo izbegli SQL injection napade, omogućavaju nam izvršavanje predefinisanih (kompileovanih) upita.

Isto kao i sa Statement objektima, PreparedStatement-e je potrebno zatvoriti.

```
PreparedStatement pstmt = null;
try {
String SQL = "Update Employees SET age = ? WHERE id = ?";
pstmt = conn.prepareStatement(SQL);
. . .
} catch (SQLException e) {
. . .
} finally {
. . .
}
```

Slika 3. Kreiranje PreparedStatement Object

```
PreparedStatement pstmt = null;
try {
String SQL = "Update Employees SET age = ? WHERE id = ?";
pstmt = conn.prepareStatement(SQL);
. . .
} catch (SQLException e) {
. . .
} finally {
pstmt.close();
}
```

Slika 4. Zatvaranje PreparedStatement Object

ResultSet

02

ŠTA JE RESULTSET?

Omogućava da se krećemo kroz rezultat upita i analiziramo ga red po red.

ResultSet predstavlja iterator kroz rezultat našeg upita. Omogućava da se krećemo kroz rezultat upita i analiziramo ga red po red. ResultSet ima metode koje nam omogućavaju da dobijamo vrednosti za određene kolone trenutnog reda.

Tip ResultSet-a

Ukoliko ne preciziramo tip ResultSet-a, dobiće se tip TYPE_FORWARD_ONLY.

Tip	Opis
ResultSet.TYPE_FORWARD_ONLY	Moguće je proći kroz rezultate samo unapred.
ResultSet.TYPE_SCROLL_INSENSITIVE	Moguće je prolaziti kroz rezultate i ka napred i ka nazad.
ResultSet.TYPE_SCROLL_SENSITIVE.	Moguće je proći i ka napred i ka nazad a sve izmene u bazi se preslikavaju u ResultSet-u.

Slika 1. Različiti tipovi ResultSet-a

Konkurentnost ResultSet-a

Konkurentnosti je moguće izmeniti, ukoliko nije eksplicitno rečeno onda je mod konkurentnosti CONCUR_READ_ONLY.

Concurrency	Opis
ResultSet.CONCUR_READ_ONLY	Kreira samo set iz koga se može čitati.
ResultSet.CONCUR_UPDATABLE	Kreira ResultSet koji je može menjati.

Slika 2. Mouće konkuretnosti: čitanje i ažuriranje

Primer:.

```
try {
Statement stmt = conn.createStatement(                ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_READ_ONLY);
} catch(Exception ex) {    .... }
finally {    .... }
```

Slika 3. Primer gde je konkuretnost ResultSet-a čitanje a tip TYPE_FORWARD_ONLY (defaultna vrednost)

METODE RESULTSET-A

Za pomeranje kursora pre prvog reda, posle zadnjeg reda, na prvi, zadnji ili određeni red, metode za pomeranje kursora unazad, unapred i tekući red

S.N.	Methode i opisi
1	**public void beforeFirst() throws SQLException** Pomera kursor na pre prvog reda.
2	**public void afterLast() throws SQLException** Pomera kursor nakon zadnjeg reda
3	**public boolean first() throws SQLException** Pomera kursor na prvi red.
4	**public void last() throws SQLException** Pomera kursor na zadnji red.
5	**public boolean absolute(int row) throws SQLException** Pomera kursor na određeni red.
6	**public boolean relative(int row) throws SQLException** Pomera kursor za određeni broj redova.

Slika 4. Metode za pomeranje kursora pre prvog reda, posle zadnjeg reda, na prvi, zadnji ili određeni red itd.

7	**public boolean previous() throws SQLException** Pomera kursor unazad.
8	**public boolean next() throws SQLException** Pomera kursor unapred.
9	**public int getRow() throws SQLException** Vraća trenutni red.

Slika 5. metode za pomeranje kursora unazad, unapred i tekući red

PREGLEDAVANJE RESULT SET-A

ResultSet interface sadrži puno metoda za izvlačenje podataka iz trenutnog reda.

Svaka od ovih get metoda ima dve verzije:

- Prva uzima ime kolone.
- Druga uzima indeks kolone.

Ukoliko želimo da dobijemo integer vrednost (int) koristimo jednu od dve metode:

S.N.	Methode i opis
1	**public int getInt(String columnName) throws SQLException** Vraća int vrednost iz kolone sa nazivom columnName.
2	**public int getInt(int columnIndex) throws SQLException** Vraća int vrednost iz kolone koja se nalazi na columnIndex-u. Indeksiranje kreće od jedinice.

Slika 6. Metode getint kojima se vraća integer vrednost kolone sa imenom “ColumnName” (1) ili sa indeksom kolone “ColumnIndex” (2)

Updateovanje Result Set-a

ResultSet interface sadrži metode za update isto kao i za čitanje.

Primer metoda za update String vrednosti u ResultSetu:

S.N.	Methoda i opis
1	**public void updateString(int columnIndex, String s) throws SQLException** Menja string sa indeksom columnIndex na vrednost s.
2	**public void updateString(String columnName, String s) throws SQLException** Menja vrednost sa kolone naziva columnName na vrednost String-a s.

Slika 7. Metode updateString kojima se menja string vrednost kolone sa indeksom kolone “ColumnIndex” (1) ili sa imenom kolone “ColumnName” (2)

KONVERZIJA SQL TIPOVA PODATAKA U JAVA TIPOVE

JDBC driver konvertuje podatke i prilagođava ih tipovima baze nad kojom se radi

JDBC driver konvertuje podatke i prilagođava ih tipovima baze nad kojom se radi. Na primer Java int je konvertovan u SQL INTEGER. Kako bi se uvela konzistentnost između različitih driver-a, postoji defaultno mapiranje.

Naredna tabela prikazuje kako se SQL tipovi konvertuju u Java tipove i koje metode je za ovo potrebno koristiti.

SQL	JDBC/Java	setXXX	updateXXX
VARCHAR	java.lang.String	setString	updateString
CHAR	java.lang.String	setString	updateString
LONGVARCHAR	java.lang.String	setString	updateString
BIT	boolean	setBoolean	updateBoolean
ARRAY	java.sql.Array	setARRAY	updateARRAY
REF	java.sql.Ref	SetRef	updateRef
STRUCT	java.sql.Struct	SetStruct	updateStruct
NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
TINYINT	byte	setByte	updateByte
SMALLINT	short	setShort	updateShort
INTEGER	int	setInt	updateInt
BIGINT	long	setLong	updateLong
REAL	float	setFloat	updateFloat
FLOAT	float	setFloat	updateFloat
DOUBLE	double	setDouble	updateDouble
VARBINARY	byte[]	setBytes	updateBytes
BINARY	byte[]	setBytes	updateBytes
DATE	java.sql.Date	setDate	updateDate
TIME	java.sql.Time	setTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp
CLOB	java.sql.Clob	setClob	updateClob
BLOB	java.sql.Blob	setBlob	updateBlob

UPDATEOVANJE RESULT SET-A

ResultSet interface sadrži metode za update-ovanje podataka iz trenutnog reda.

Postoje metode za objektne tipove i String-ove.

Updateovanje promenjljive u ResultSetu ne updateuje direktno u bazi, da bi se uradio update u bazi potrebno je pozvati jednu od narednih metoda:

S.N.	Methoda i opis
1	**public void updateRow()** Vrši update u bazi.
2	**public void deleteRow()** Briše red iz baze
3	**public void refreshRow()** Ponovo učitava red ukoliko ga je neko drugi updateovao.
4	**public void cancelRowUpdates()** Prekida update reda.
5	**public void insertRow()** Unosi novi red u bazu.

Slika 8. Metode za upisivanje promena u bazu, brisanje redova iz baze, ponovno učitavanje, insertovanje redova u bazu

SQL EXCEPTION METODE

SQLException može da se javi u na driveru i na bazi

Kada se desi ovakav izuzetak, objekat tipa SQLException će biti prosleđen catch bloku.

Prosleđeni SQLException objekat ima metode koje nam omogućavaju da detaljnije analiziramo izuzetak, metode su sledeće:

Metod	Opis
getErrorCode( )	Vraća broj greške.
getMessage( )	Vraća poruku greške.
getSQLState( )	Vraća sql stanje, ova metoda može da vrati null vrednost.
getNextException( )	Vraća naredni Exception objekat iz lanca izuzetaka.
printStackTrace( )	Stampa trenutni izuzetak.
printStackTrace(PrintStream s)	Stampa trenutni izuzetak na prosleđeni PrintStream.
printStackTrace(PrintWriter w)	Stampa trenutni izuzetak na prosleđeni PrintWriter.

Slika 4. Metode SQLException objekta koje nam omogućavaju da detaljnije analiziramo razlog izuzetka

```
try {   // Your risky code goes between these curly braces!!! } catch(Exception ex) {
// Your exception handling code goes between these
// curly braces, similar to the exception clause
// in a PL/SQL block.
} finally {
// Your must-always-be-executed code goes between these
// curly braces. Like closing database connection.
}
```

Slika 5. Primer za try catch

PRIMER KODA ZA TRY, CATCH I FINALLY BLOKOVE -I DEO

Deo koda u kojem se poziva operacija try

Kada pokušamo da izvršimo operaciju u try bloku koda, može se pojaviti izuzetak.

Izuzetak može da nastane zbog ili hardverske ili softverske greške ili može da bude izazvan čak i od strane programera. Ukoliko dođe do greške, Catch blok nam omogućava da o tome obavestimo korisnika ili da ugasimo aplikaciju.

Finaly blok je blok koda koji će se zagarantovano izvršiti, u ovom delu koda pokušavamo da zatvorimo sve svoje konekcije.

```
//STEP 1. Import required packages
import java.sql.*;
public class JDBCExample {
// JDBC driver name and database URL
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/EMP";
//  Database credentials
static final String USER = "username";
static final String PASS = "password";
public static void main(String[] args) {
Connection conn = null;   try{
```

Slika 1. Deo koda u kojem se poziva operacija try

```
//STEP 2: Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");
//STEP 3: Open a connection
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);
//STEP 4: Execute a query
System.out.println("Creating statement...");
Statement stmt = conn.createStatement();
String sql;
sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);
//STEP 5: Extract data from result set
while(rs.next()){
//Retrieve by column name
int id  = rs.getInt("id");
int age = rs.getInt("age");
String first = rs.getString("first");
String last = rs.getString("last");
//Display values
System.out.print("ID: " + id);
System.out.print(", Age: " + age);
System.out.print(", First: " + first);
System.out.println(", Last: " + last);
}
```

Slika 2. Deo koda u komen se izvršava upit nad tabelom Emplyees i štampa rezultat

PRIMER KODA ZA TRY, CATCH I FINALLY BLOKOVE -II DEO

Deo koda u kojem se koristie operacija catch i try

```
//STEP 6: Clean-up environment
rs.close();
stmt.close();
conn.close();
}catch(SQLException se){
//Handle errors for JDBC
se.printStackTrace();
}catch(Exception e){
//Handle errors for Class.forName
e.printStackTrace();
}finally{
//finally block used to close resources
 try{
if(conn!=null)
conn.close();
}catch(SQLException se){
se.printStackTrace();
}//end finally try
}//end try
System.out.println("Goodbye!");
}
//end main
}//end JDBCExample
```

Slika 3. Deo koda u kojem se u Catch blok hvata greška ukoliko je do nje došlo (na nivou baze podataka (SQExeption) ili aplikacije (Exeption)). Zatim se finally bloku o gresci obaveštava korisnika ili gasi aplikacija

Vežba: Preduslovi za pisanje Java koda

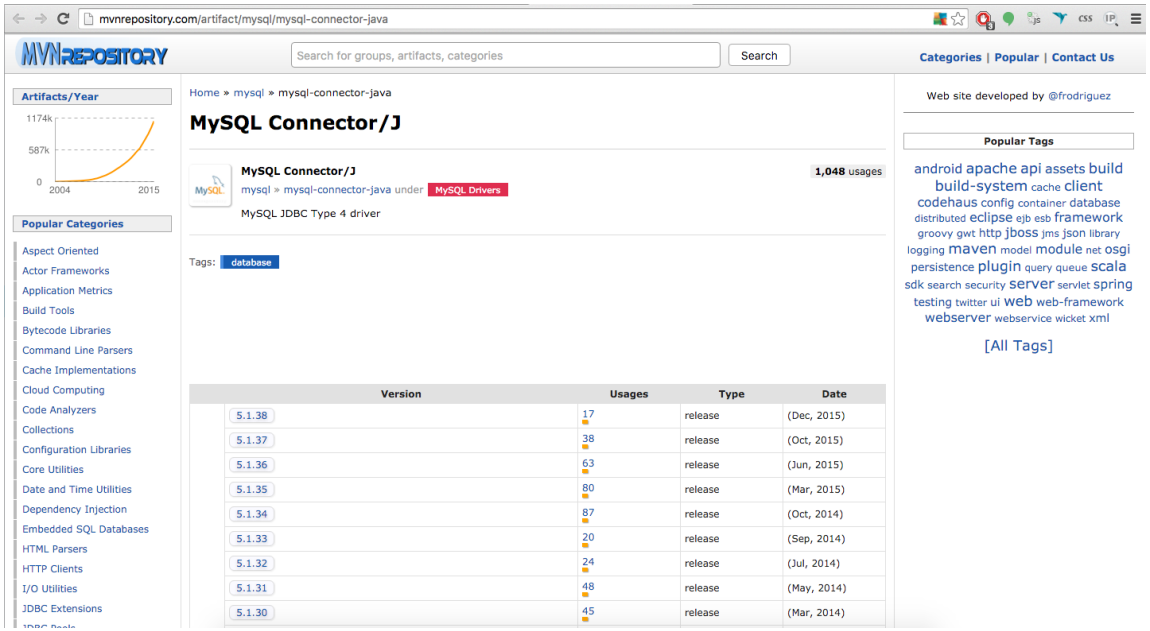
- *Vežba: Primer Java koda*

03

SKIDANJE JDBC DRIVER-A

U konkretnom sluzčaju skida se JDBC driver za MySQL bazu

Najpre treba skinuti JDBC driver za MySQL bazu. U ovom slučaju, drajver je skinut sa sajta Maven.



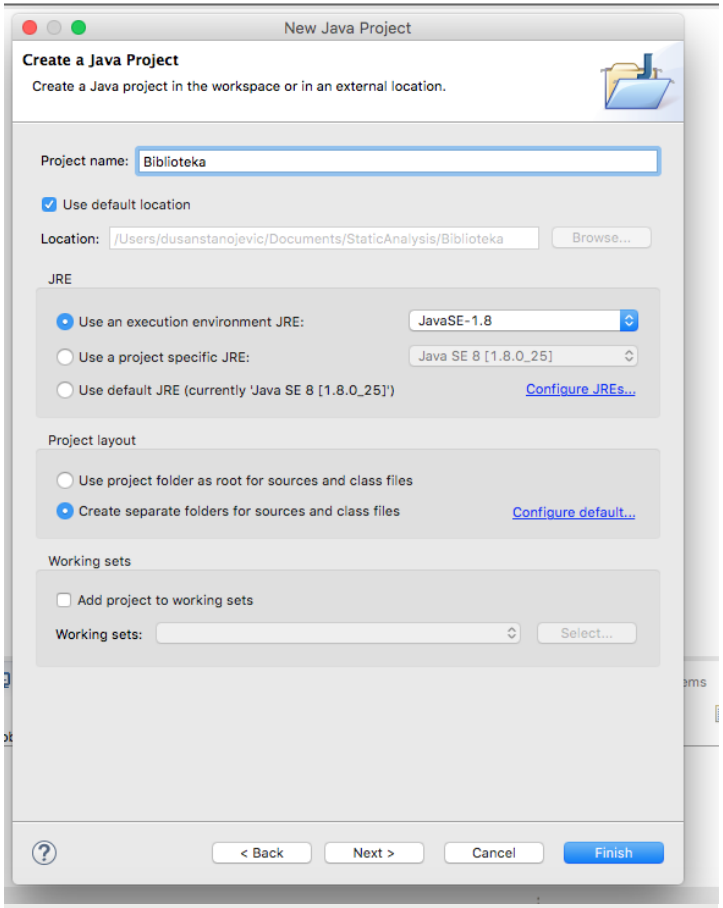
Slika 1. Skidanje JDBC drajvera sa https://maven.apache.org

SKIDANJE ODGOVARAJUĆEG INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

U ovom slučaju se koristi Eclipse

Zatim treba izabrati IDE u kojem želimo da radimo i njemu napraviti jedan projekat. U ovom slučaju se za demonstraciju se koristi Eclipse IDE a projekat je nazvan Biblioteka.

.

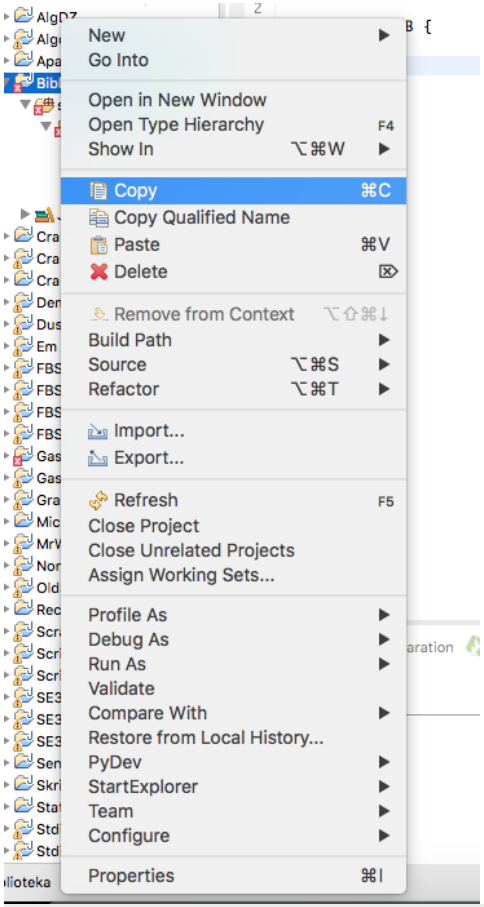


Slika 2. Dodave projekta Biblioteka

DODAVANJE ODGOVARAJUĆEG DRIVER ZA KONEKCIJU NA BAZU NA BUILD PATH

U ovom slučaju se dodaje MySQL driver

Da bismo koristili MySQL bazu iz Java aplikacije, potrebno je dodati MySQL driver na build path, kako bi mogli da ga koristimo. (u Eclipse-u treba desnim klikom kliknuti na naziv projekta Biblioteka, iz opadajućeg menija izabrati opciju Properties.

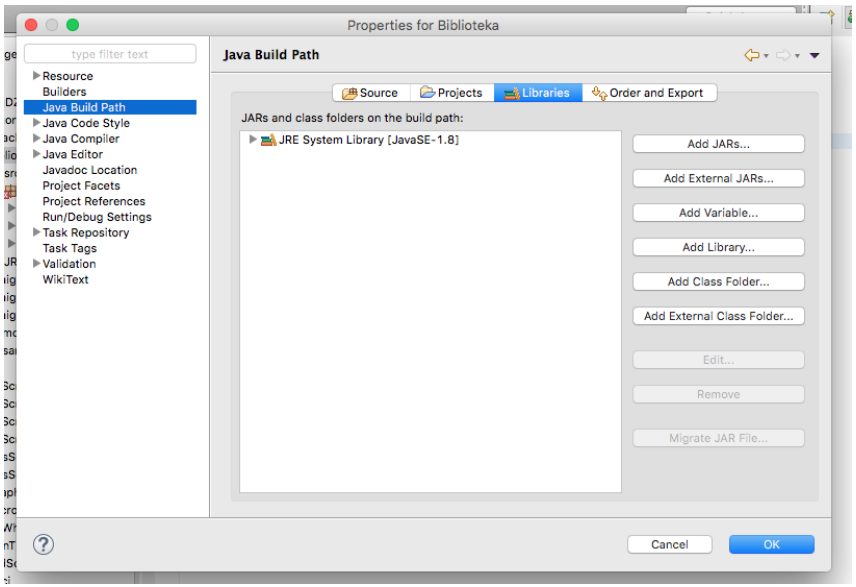


Slika 3. Desnim klikom na ime projekta dobija se meni odakle treba izabrati opciju Properties

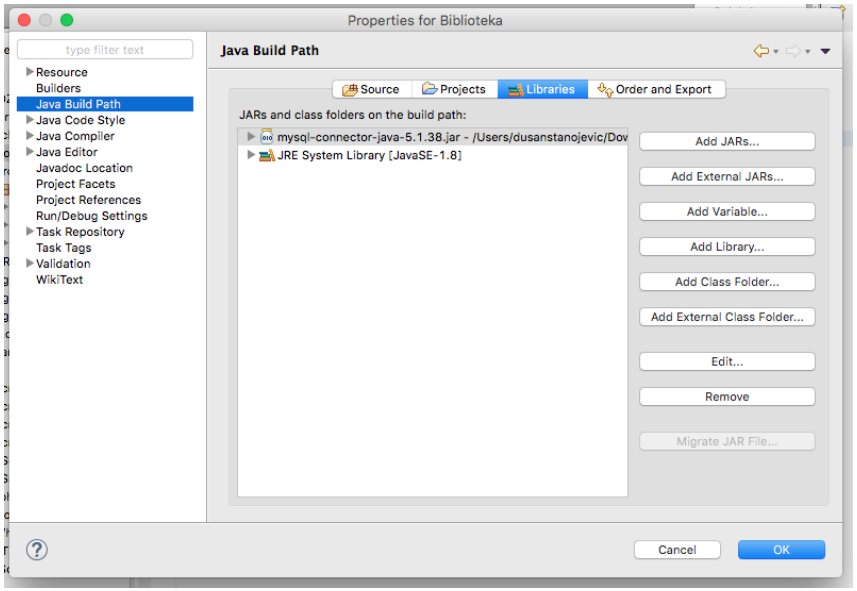
IZBOR OPCIJE JAVA BUILD PATH

Dolaženje do kartica Liberies i izbor MySQL drivera

Nakon klika na Properties dobija se meni sa slike 4. iz kojeg s leve strane treba izabrati opciju Java Build Path. U kartici libraries kliknemo na dugme Add External Jars.. i odaberemo MySQL driver koji smo skinuli (slika 5.).



Slika 4. Izbor opcije Java Build Path



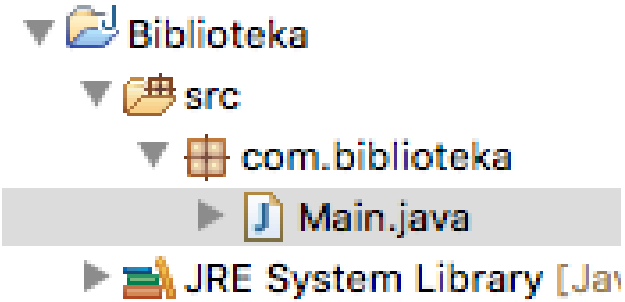
Slika 5. Kartica Liberies i izbor MySQL drivera

PRAVLJENJE PAKETA I ODGOVARAJUĆE KLASE U NJEMU

U ovom slučaju pravimo paket com.biblioteka i u njemu Main klasu sa main metodom

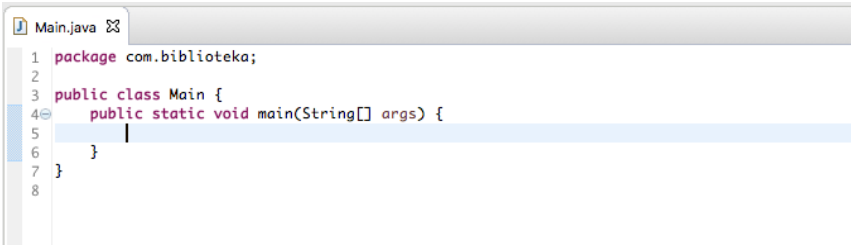
Da bismo pokrenuli primere iz vezbi potrebno je napraviti paket (u ovom slučaju com.biblioteka) i u njemu odgovarajuću klasu (u ovom slučaju Main klasu) sa main metodom.

Kako bi se što jednostavnije demonstrirao rad sa JDBC-em, svi primeri iz predavanja su pisani na ovaj nacin.



Slika 6. Kreiranje main klase Main.java i njene metode

Nakon toga se dobija sledeći kod



Slika 7. Kod koji se dobija nakon kreiranja Main. Java klase i njene metode

Sada se mogu koristiti primeri iz predavanja.

EFIKASNIJE KORISCENJE BAZA U JAVA KODU – I DEO

Efikasnost u pisanju koda se postiže dodavanjem odgovarajućih interfejsa u Javi8

Za malo naprednije koriscenje baza u javi treba kopirati file DB.java iz prilozenog koda i dodajte ga u projekat. U njemu podesite username i password.

Efikasnost u pisanju koda se sada postize dodavanjem odgovarajućih interfejsa u Javi8.

Moze se koristiti kao na slikama 8. i 9.

```java
package com.biblioteka;

import java.sql.Connection;

public class Knjiga {
    public String name;
    public Long id;

    public Knjiga() {}

    private Knjiga(ResultSet rs) throws SQLException {
        this.name = rs.getString("name");
        this.id = rs.getLong("id");
    }

    public static List<Knjiga> loadAll(Connection conn) throws SQLException {
        final String query = "SELECT * FROM books";

        PreparedStatement statement = conn.prepareStatement(query);

        List<Knjiga> knjigaList = new ArrayList<>();

        DB.executeAndParse(statement, rs -> {
            knjigaList.add(new Knjiga(rs));
        });

        return knjigaList;
    }

    public static Long insert(Connection conn, Knjiga knjiga) throws SQLException {
        final String query = "INSERT INTO books (id, name) VALUES(?,?)";

        PreparedStatement statement = conn.prepareStatement(query, PreparedStatement.RETURN_GENERATED_KEYS);

        int i = 1;
        statement.setLong(i++, knjiga.id);
        statement.setString(i++, knjiga.name);

        int affectedRows = statement.executeUpdate();
```

Slika 8. Java kod za definisanje klase Knjiga

EFIKASNIJE KORISCENJE BAZA U JAVA KODU – I DEO

Primer koriscenja Java8 funkcionalnih interface-a u kombinaciji sa DB klasom i klasom Knjiga

Sve metode vezane za rad sa odredjenim Entitetom se pisu u klasi tog entiteta kao static metode. Ovde je dat primer za klasu knjiga koja je mapirana tabelom books koja u sebi sadrzi dve kolone: id i name.

```
package com.biblioteka;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        DB.withConnection(conn -> {
            Knjiga.loadAll(conn).forEach(k -> System.out.println(k));
        });

        Scanner s = new Scanner(System.in);
        System.out.println("Unesite knjigu: ");
        Knjiga k = new Knjiga();
        k.name = s.nextLine();
        s.close();

        DB.withTransaction(conn -> {
            Knjiga.insert(conn, k);

            Knjiga.loadAll(conn).forEach(knjiga -> System.out.println(knjiga));
        });
    }
}
```

Slika 9. Primer koriscenja Java8 funkcionalnih interface-a u kombinaciji sa nasom DB klasom i nasom Knjiga klasom.

Kao što se može videti, efektivni kod je znatno kraci i organizovaniji od koda iz prethodnih primera.
U klasi DB nalaze se dve metode koje se koriste u aplikativnoj logici

Vežba: Primer Java koda

03

PRIMER RADA SA XML-OM U BAZI

Primer se odnosi na čuvanje XML dokumenta u bazi podataka

Ukoliko želimo možemo da koristimo XML file da ga dodamo u bazu.

```
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
</Employee>
```

Slika 1. Primer XML dokumenta koji unosimo u bazu

JAVA PROGRAM KOJIM SE XML DOKUMENT ČUVA U BAZI: KORACI OD 1-3

Deo java koda u kojem se vrši importovanje paketa, registrovanje JDBC drajvera i otvaranje konekcije sa bazom

```
// Import required packages
import java.sql.*;
import java.io.*;
import java.util.*;
public class
JDBCExample {
// JDBC driver name and database URL
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/EMP";
//  Database credentials
static final String USER = "username";
static final String PASS = "password";
public static void main(String[] args) {
Connection conn = null;
PreparedStatement pstmt = null;
Statement stmt = null;
ResultSet rs = null;
try{
// Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");
```

Slika 2. Deo java koda u kojem se izvršava korak 1: importovanje paketa i korak 2:registrovanje JDBC drajvera

```
// Open a connection
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);
//Create a Statement object and build table
stmt = conn.createStatement();
createXMLTable(stmt);
//Open a FileInputStream
File f = new File("XML_Data.xml");
long fileLength = f.length();
FileInputStream fis = new FileInputStream(f);
//Create PreparedStatement and stream data
String SQL = "INSERT INTO XML_Data VALUES (?,?)";
pstmt = conn.prepareStatement(SQL);
pstmt.setInt(1,100);
pstmt.setAsciiStream(2,fis,(int)fileLength);
pstmt.execute();
//Close inputstream
fis.close();
```

Slika 3. Deo java koda u kojem se izvršava korak 3: otvaranje konekcije

JAVA PROGRAM KOJIM SE XML DOKUMENT ČUVA U BAZI: KORACI OD 4-6

Deo java koda u kojem se izvršava upit, izvdajanje podataka iz skupa rezultata, čišćenje okruženja i upravlja greškama

```
// Do a query to get the row
SQL = "SELECT Data FROM XML_Data WHERE id=100";
rs = stmt.executeQuery (SQL);
// Get the first row
if (rs.next ()) {
//Retrieve data from input stream
InputStream xmlInputStream = rs.getAsciiStream (1);
int c;
ByteArrayOutputStream bos = new ByteArrayOutputStream();
while (( c = xmlInputStream.read ()) != -1)
bos.write(c);
//Print results
System.out.println(bos.toString());
}
// Clean-up environment
rs.close();
stmt.close();
pstmt.close();
conn.close();
}catch(SQLException se)
```

Slika 4. deo java koda u kojem se izvršava korak 4: upit, korak 5: izvdajanje podataka iz skupa rezultata i korak 6. čišćenje okruženja

```
{ //Handle errors for JDBC
se.printStackTrace();
}catch(Exception e){
//Handle errors for Class.forName
e.printStackTrace();
}finally{
//finally block used to close resources
try{
if(stmt!=null)
stmt.close();
}catch(SQLException se2){     }
// nothing we can do
try{
if(pstmt!=null)
pstmt.close();
}catch(SQLException se2){     }
// nothing we can do
try{
if(conn!=null)
conn.close();
}catch(SQLException se){
se.printStackTrace();
}//end finally try
}//end try
System.out.println("Goodbye!"); }
```

Slika 5. Deo java koda u kojem se upravlja greškama

JAVA PROGRAM KOJIM SE XML DOKUMENT ČUVA U BAZI: REZULTAT UPITA

```
//end main
public static void createXMLTable(Statement stmt)
throws SQLException{
System.out.println("Creating XML_Data table..." );
//Create SQL Statement
String streamingDataSql = "CREATE TABLE XML_Data " +
"(id INTEGER, Data LONG)";
//Drop table first if it exists.
try{
stmt.executeUpdate("DROP TABLE XML_Data");
}catch(SQLException se){    }
// do nothing
//Build table.
stmt.executeUpdate(streamingDataSql);
}//end createXMLTable
}//end JDBCExample
```

Slika 6. Deo java koda u kojem se upravlja greškama - nastavak

Rezultat programa:

```
C:\>java JDBCExample Connecting to database...
Creating XML_Data table...
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
</Employee> Goodbye! C:\>
```

Slika 7. Rezultat programa: kreirani XML dokument

Zaključak

ZAKLJUČAK

Šta smo naučili u ovoj lekciji?

U ovoj lekciji je kroz nekoliko primera i uz teorijska obajašnjena prikazano na koji se način iz java programa korišćenjem JDBC-a može pristupiti bazi podataka i u njoj izvršiti odgovarajuće transakcije. Podaci uzeti iz baze se mogu koristiti u aplikaciji radi održavanja odgovarajuće poslovne logike.

U prikazanim primerima je korišćena MySQL baza podataka i Eclipse integrated development environment (IDE). JDBC driver za MySQL bazu je skinut sa site-a Maven koji je zatim dodat na build path, kako bi mogli da ga koristimo u Javi.

U Javi je napravljen paket com.biblioteka i u njemu Main klasu sa main metodom.

Savladavanjem ove lekcije, studenti se osposobljavaju za korišćenje bate podataka iz java koda.