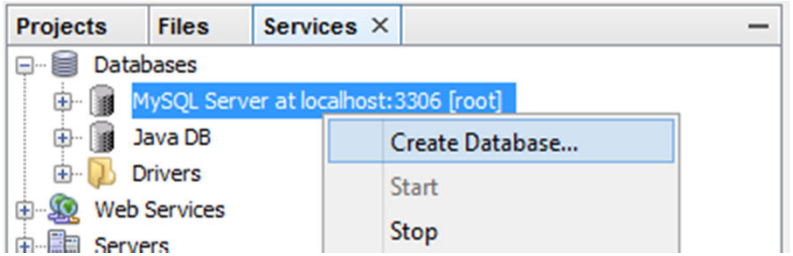
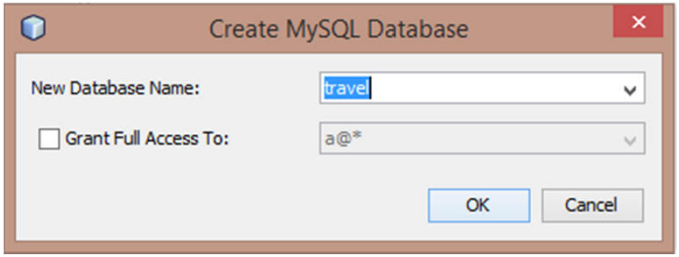
NetBeans i MySql

Netbeans ima ugrađen sistem za upravaljanje bazama, pa čak i nekoliko primera baza podataka

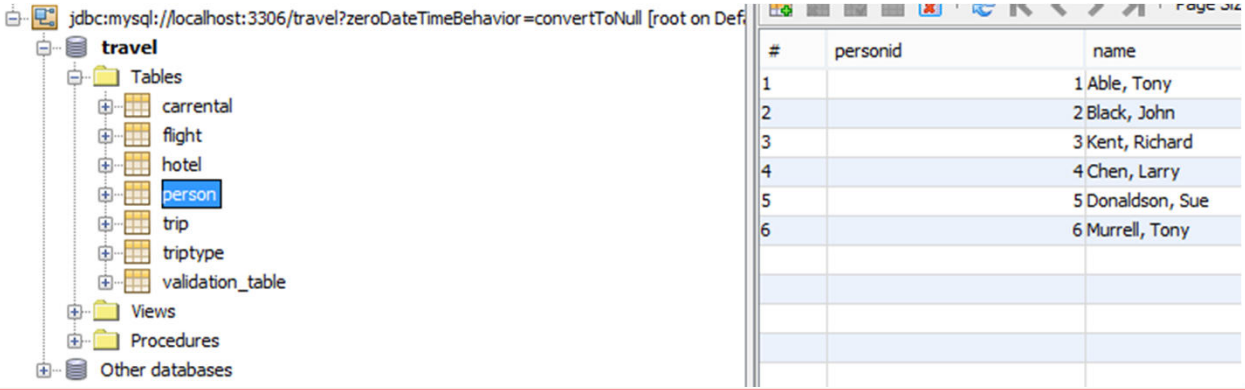
1



2

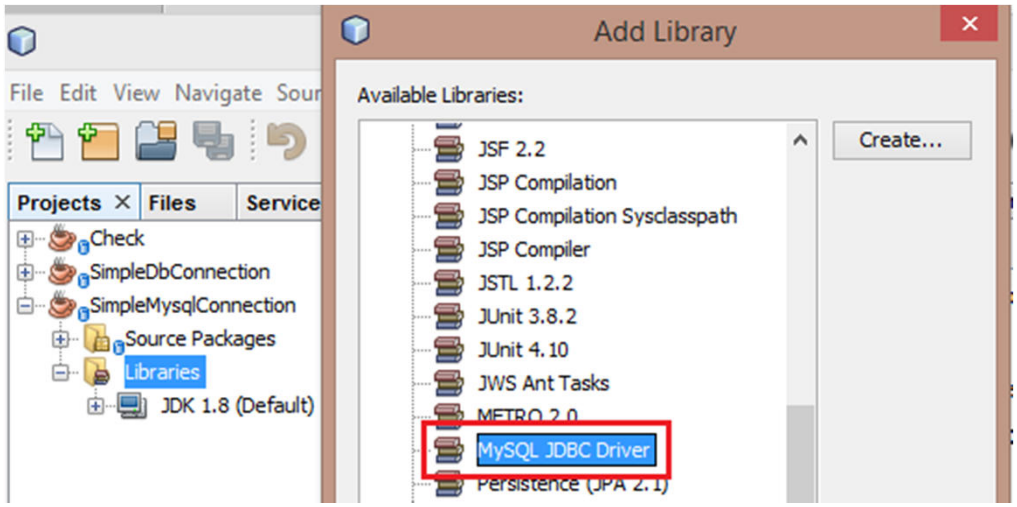


3



Uvođenje MySql drajvera u projekat

- Da bi Java mogla da koristi MySql pomoću JDBC API-ja, mora biti učitan odgovarajući drajver (konektor)
- Ovaj drajver već postoji u NetBeans-u, samo ga treba pridružiti projektu,
  a moguće je preuzeti
  ga eksterno i dodati kao
  Java biblioteku



Konektovanje na MySql server

- Zahvaljujući transparentnosti JDBC-a, konektovanje na bazu podataka obavlja se na isti način kao i na druge baze podataka
- Samo je potrebno obratiti pažnju na konekcioni string

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/travel","root","");
if(!conn.isClosed()){
    System.out.println("Connection is established");
} else {
    System.out.println("Connection is not established");
}
```

Izvršavanje upita nad MySql serverom

- Takođe, i upiti se koriste na isti način za sve baze povezane pomoću JDBC-a
- U sledećem primeru preuzima se lista korisnika sistema (test baza travel)

```
Statement st_persons = conn.createStatement();
ResultSet res = st_persons.executeQuery("select * from person");
String[] columns = new String[res.getMetaData().getColumnCount()];
for(int i=0;i<columns.length;i++) {
    System.out.print((columns[i] = res.getMetaData().getColumnName(i+1))+"\t");
}
System.out.println();
while(res.next()){
    for(int i=0;i<columns.length;i++) {
        System.out.print(res.getString(columns[i])+"\t");
    }
    System.out.println();
}
```

- U ovom primeru smo koristili i dodatni metod klase ResultSet - **getMetaData**

Rukovanje meta podacima

- Većina ključnih klasa JDBC-a, dozvoljavaju pristup metapodacima koji se tiču entiteta koji predstavljaju
- Tako za Connection možemo dobiti meta podatke konekcije, za upit možemo dobiti meta podatke tabele nad kojom je izvršen, i slično

```
System.out.println("CONNECTION METADATA");
DatabaseMetaData db_metadata = conn.getMetaData();
ResultSet databases = db_metadata.getCatalogs();
while(databases.next())
    System.out.println("Database: " + databases.getString(1));
System.out.println("TABLE METADATA");
ResultSetMetaData rs_metadata = res.getMetaData();
System.out.println("Database: " + rs_metadata.getCatalogName(1));
System.out.println("Table: " + rs_metadata.getTableName(1));
```

Rukovanje meta podacima (last insert id)

- Jedan od veoma važnih meta podataka tabele jeste poslednji uneti ID
- Da bi dobili ovaj podataka moramo prilikom izvršavanja upita naglasiti da hoćemo da nam bude vraćen, konstantom **RETURN_GENERATED_KEYS**

```
st_persons.execute("insert into users (firstname,lastname) values ('peter','jackson')",RETURN_GENERATED_KEYS);
long last_id = -1;
ResultSet generated_keys = st_persons.getGeneratedKeys();
if(generated_keys.next()){
    last_id = generated_keys.getLong(1);
}
System.out.println("Last generated key: " + last_id);
```

Vežba

- Kreirati program za upravljanje bazom podataka
- Program treba da ima naredbe za prikaz svih baza na serveru kao i komande za brisanje i kreiranje baze

Vežba

- Potrebno je kreirati program web crawler
- Crawler treba da ima inicijalnu vrednost (html dokument)
- Crawler treba da pretraži kompletan html dokument u potrazi za linkovima
- Za svaki pronađeni link, program treba da izvrši istu proceduru pretrage
- Sve pretražene strane moraju biti snimljene u bazu

Vežba Pomoć

- Preuzimanje sadržaja strane sa Interneta:

```
URL page = new URL(url);
BufferedReader in = new BufferedReader(new
InputStreamReader(page.openStream()));
String inputLine;
StringBuffer bfr = new StringBuffer();
            while ((inputLine = in.readLine()) != null) {
                bfr.append(inputLine);
            }
in.close();
return bfr.toString();
```

Vežba Pomoć

- Preuzimanje sadržaja strane sa Interneta:

```
URL page = new URL(url);
BufferedReader in = new BufferedReader(new
InputStreamReader(page.openStream()));
String inputLine;
StringBuffer bfr = new StringBuffer();
            while ((inputLine = in.readLine()) != null) {
                bfr.append(inputLine);
            }
in.close();
return bfr.toString();
```

Vežba Pomoć

- Preuzimanje pronalaženje hiperlinkova na strani:

```
String parseHlink(String page,int pos){
        StringBuilder output = new StringBuilder();
        pos+=6;
        char ch = page.charAt(pos);
        while((ch=page.charAt(pos++))!='"'){
            output.append(ch);
        }
        return output.toString();
    }
    List parse(String page){
        List res = new ArrayList();
        int lastPos = page.length();
        while(lastPos!=-1){
            lastPos = page.lastIndexOf("href",lastPos-1);
            String adr = parseHlink(page, lastPos);
            System.out.println(adr);
        }
        return res;
    }
```

Upravljanje transakcijama
(Eksplicitne transakcije)

- Transakcija omogućava izvršavanje serije sql naredbi po sistemu sve ili ništa
- Transakcije se dele na **implicitne** i **eksplicitne**
- Implicitne transakcije već nesvesno koristimo korišćenjem standardnih DML sql naredbi (insert, update,delete).
- Eksplicitne transakcije je potrebno specijalno naznačiti
- Svaka eksplicitna transakacija mora se potvrditi (**commit**) ili otkazati (**rollback**)
- JDBC API poznaje pojam transakcija, i omogućava njihovo korišćenje
- *Za praćenje primera bi bilo dobro imati tabelu **accounts** u bazi podataka **travel***

```
create table accounts
(
id int primary key auto_increment,
username varchar(256),
balance decimal(9,2)
)
```

```
insert into accounts values
(null,'Peter',100),(null,'Sally',0)
```

Upravljanje transakcijama
(Eksplicitne transakcije)

- Svaka DML komanda se izvršava transakciono, i ovo ponašanje možemo kontrolisati metodom **setAutoCommit**.
- Analizirajmo sledeći primer:

Isključuje se autcommit

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/travel", "root", "");
conn.setAutoCommit(false);
Statement st = conn.createStatement();
st.execute("insert into accounts values (null,'John',500)");
ResultSet res = st.executeQuery("select * from accounts");
while(res.next()){
    System.out.println(res.getString(1)+" "+res.getString(2)+" "+res.getString(3));
}
```

Izvršava se regularno insert upit

Select prikazuje očekivani razultat

```
run:
1 Peter 100.00
2 Sally 0.00
3 John 500.00
BUILD SUCCESSFUL (total time: 0 seconds)
```

Da li je stvarno sve tako kako izgleda?

Upravljanje transakcijama
(Eksplicitne transakcije)

- Startovanje programa sa prethodnog slajda, dovodi nas do zanimljivog rezultata na izlazu:

```
run:
1 Peter 100.00
2 Sally 0.00
10 John 500.00
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Broj redova je svaki put 3
  - Logično, zato što svaki put pokrećemo transakciju eksplicitnu koju nikada ne potvrđujemo
- Takođe, John prilikom svakog novog startovanja ima novi id
  - Ovo je stvar MySql-a više nego transakcija (mada se većina RDBMS-a isto ponaša). Jednostavno – generacija auto increment kolone ne izvršava se transakciono
- Ponašanje programa u primeru je očekivano, jer smo transakciju naredbe insert iz implicitnog, prebacili u eksplicitni mod. To znači da se sami moramo nositi sa kompletnim tokom njene transakcije.

Upravljanje transakcijama
(Eksplicitne transakcije)

- Transakciju potvrđujemo naredbom **commit**, a otkazujemo naredbom **rollback**. U JDBC-u, ove naredbe se realizuju istoimenim metodama

```
Connection conn = DriverManager.getConnection("jdbc:mysql://l
, "");
conn.setAutoCommit(false);
Statement st = conn.createStatement();
st.execute("insert into accounts values (null,'John',500)");
ResultSet res = st.executeQuery("select * from accounts");
while(res.next()){
    System.out.println(res.getString(1)+" "+res.getString(2)+
}
conn.commit();
```

commit – potvrđuje transakciju

Transkcija je sada potvrđena i izmene u bazi su trajne

```
run:
1 Peter 100.00
2 Sally 0.00
11 John 500.00
BUILD SUCCESSFUL (total time: 0 seconds)
```

Rollback i checkpont

- Da bi otkazali transakciju koristimo naredbu **rollback**
- Za ozbiljnije transakcije, moguće je formirati tačke u izvršavanju do kojih je moguće izvršiti rollback (checkpointovi)
  - U primeru desno, prvi unos je izvršen, ali drugi unos nije jer je izvršen rollback do prvog checkpointa

```
conn.setAutoCommit(false);
Statement st = conn.createStatement();
st.execute("insert into accounts values (null,'John',500)");
ResultSet res = st.executeQuery("select * from accounts");
while(res.next()){
    System.out.println(res.getString(1)+" "+res.getString(2)+
}
Savepoint tran_checkpoint = conn.setSavepoint();
st.execute("insert into accounts values (null,'John',500)");
res = st.executeQuery("select * from accounts");
while(res.next()){
    System.out.println(res.getString(1)+" "+res.getString(2)+
}
conn.rollback(tran_checkpoint);
conn.commit();
```

Vežba 2 (jcex142014 AccountManagement)

- Potrebno je kreirati aplikaciju koja će prebacivati novac sa računa jednog korisnika na račun drugog korisnika
- Kao izvor podataka koristiti tabelu account iz prethodnog primera
- Prebacivanje novca sa računa na račun mora biti obavljeno transakciono, tako da nije moguće doći u situaciju da novac bude skinut sa jednog računa ali ne i dodat na drugi račun, i obrnuto

DataSource interfejs

- U slučaju enterprise Java aplikacija, najčešće se koristi DataSource umesto DriverManager klase
- DataSource radi isti posao kao i DriverManager, ali sadrži i dodatne mogućnosti
  - Eksternu konfiguraciju i automatski pooling konekcije

```
MysqlDataSource ds = new MysqlDataSource();
ds.setURL("jdbc:mysql://localhost");
ds.setDatabaseName("travel");
ds.setUser("root");
ds.setPassword("");
Connection conn = ds.getConnection();

System.out.println(!conn.isClosed()?"Successfully connected to database":"Failed to connect");

conn.close();
```

http://pastie.org/pastes/9938393/text

Rad sa uskladištenim procedurama

- Uskladištene procedure koristimo kako bi razdvojili poslovnu odgovornost aplikacije i prebacili bazi podataka deo posla koji joj odgovara
- Na primer, prebacivanje novca sa jednog računa na drugi (iz prethodnog primera), mogla bi bez problema uraditi i sama baza, bez aplikacije, kroz, na primer, sledeću proceduru:

Napravite proceduru kako bi mogli da ispratite primer

```
delimiter //
create procedure changeBalance(in source int, in destination int, in amount decimal)
begin
    start transaction;
        update accounts set balance = balance + amount where id = destination;
        update accounts set balance = balance - amount where id = source;
    commit;
end//
```

Rad sa uskladištenim procedurama

- Iako je uskladištenu proceduru moguće aktivirati i pomoću Statement i PreparedStatement objekata, ispravan način je korišćenje objekta klase **CallableStatement**
- CalableStatement je moguće instancirati metodom **prepareCall**, klase Connection

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/travel", "root","");
CallableStatement cs = conn.prepareCall("{call changeBalance(?,?,?)}");
cs.setInt(1,1);
cs.setInt(2,2);
cs.setBigDecimal(3, new BigDecimal(75));
cs.execute();
```

Napredno korišćenje ResultSet objekta

- **ResultSet**, u kombinaciji sa Statement objektima može uraditi više od običnog kretanja kroz podatke u fast forward / read only režimu
- Ovaj objekat (ukoliko to omogućava njegova implementacija), dozvoljava kretanje kroz podatke nelinearno, pa čak i unos izmenu ili brisanje podataka, ali se takav tretman mora nagovestiti prilikom kreiranja statement objekta (parametrizacijom metode createStatement)
- Ovakvim pristupom, ResultSet se ponaša kao mala baza podataka u memoriji

Kretanje kroz ResultSet
(jcex142014 ResultSetNavigation)

- Kada je ResultSet jednom formiran, možemo se kretati kroz njega, pomoću sledećih metoda:

• **first( )**	pomera marker na prvi red
• **last( )**	pomera marker na zadnji red
• **next( )**	pomera marker na sledeći red
• **previous( )**	pomera marker na prethodni red
• **beforeFirst( )**	vraća marker na početak (ispred svih redova)
• **afterLast( )**	pomera marker na kraj rezultata
• **absolute(int)**	pomera marker na određeni red (po indeksu)
• **relative(int)**	pomera marker na određeni red, u odnosu na aktuelni red

Manipulacija podacima pomoću ResultSet-a (jcex142014 ResultSetManipulation)

- ResultSet se može ponašati kao mala baza podataka u memoriji, i tako biti u stanju da delimično transparentno upravlja podacima u bazi
- Da bi ovo funkcionisalo, moramo prilikom poziva metode createStatement navesti da želimo takvo ponašanje

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/travel","root","");
Statement st = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet res = st.executeQuery("select * from accounts");

        res.absolute(3);
        res.updateString(2, "Some other user");
        res.updateRow();
        res.absolute(1);
        res.updateBigDecimal(3, res.getBigDecimal(3).add(new BigDecimal(50)));
        res.updateRow();
        res.absolute(2);
        res.updateBigDecimal(3, res.getBigDecimal(3).subtract(new BigDecimal(50)));
        res.updateRow();
```

Manipulacija podacima pomoću ResultSet-a (jcex142014 ResultSetManipulation)

- U prethodnom primeru, bio je prikazan update. Sledeći primer prikazuje insert podataka pomoću ResultSet objekta i njegovih metoda

```
res.absolute(3);
res.moveToInsertRow();
res.updateString(2, "Brand new user");
res.updateBigDecimal(3, new BigDecimal(500));
res.insertRow();
res.moveToCurrentRow();
System.out.println(res.getString(2));
```

- Po istom principu, moguće je obrisati red

```
res.absolute(4);
res.deleteRow();
```

Batchevi sql upita

- Ponekad nam je potrebno da izvršimo veliki broj upita odjednom. Iako u toj situaciji možemo određeni broj puta aktivirati metod **execute**, **executeUpdate** ili **executeQuery**, smatra se boljom praksom kreiranje neke vrste bafera sql naredbi, a zatim njegove aktivacije metodom **executeBatch** ili **executeLargeBatch**

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/travel","root","");
Statement st = conn.createStatement();
for(int i=0;i<1000;i++){
    st.addBatch("update accounts set balance = 200 where id = " + i);
}
st.executeBatch();
```

Vežba 3 (jcex142014 Auction)

- Data je tabela sledeće strukture

```
CREATE TABLE IF NOT EXISTS `auctions` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `product` varchar(256) NOT NULL,
  `price` decimal(9,2) NOT NULL,
  `starttime` timestamp NOT NULL,
  `endtime` timestamp NOT NULL,
  `active` bit(1) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

- Potrebno je kreirati aplikaciju koja će prilikom startovanja preuzeti listu aukcija i smestiti je u memoriju
- Aplikacija će zatim svake sekunde proveravati da li je vreme za startovanje aukcije i ako jeste, promeniće joj status active svojstva na true (1) u bazi podataka
- Ako je vreme za prekid neke aukcije, biće joj izmenjen status active svojstva na false (0)
- Svakih nekoliko (10) sekundi, aplikacija će preuzeti nove podatke iz baze i ažurirati postojeću listu aukcija
- Vreme za startovanje ili prekid, određuje se na osnovu početnog i završnog vremena aukcije. Ako je trenutno vreme veće od početnog i status je 0, on postaje 1, a ako je vreme kraja manje od trenutnog i status je 1, on postaje 0

Vežba 3 (jcex142014 Auction)

- Sledeći kod bi mogao ispuniti gotovo sve uslove tražene aplikacije (ali hoćemo da aplikacija ne komunicira sa bazom podataka baš svake sekunde)

```
Connection conn;
while(true){
    conn = DriverManager.getConnection("jdbc:mysql://localhost/travel","root","");
    Statement st = conn.createStatement();
    st.executeUpdate("UPDATE auctions SET active = CASE "
            + "WHEN starttime<now() and endtime>now() THEN 1 "
            + "WHEN endtime<now() and active = 1 THEN 0 "
            + "else 0 END");
    Thread.sleep(1000);
    conn.close();
```