JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Prepared Statements

INSY Übung
4AHITM 2015/16

Kanyildiz Muhammedhizir

Version 1.0

Note:

Betreuer: Prof. Borko

Begonnen am 10. April 2016

Beendet am 6. Mai 2016

Inhaltsverzeichnis

1 Einführung

1.1 Ziele

Es ist erwünscht Konfigurationen nicht direkt im Sourcecode zu speichern, daher sollen Property-Files [3] zur Anwendung kommen bzw. CLI-Argumente (Library verwenden) [1,4] verwendet werden. Dabei können natürlich Default-Werte im Code abgelegt werden. Das Hauptaugenmerk in diesem Beispiel liegt auf der Verwendung von PreparedStatements [2]. Dabei sollen alle CRUD-Aktionen durchgeführt werden.

1.2 Voraussetzungen

CLI – Grundkenntnisse
Prepared-Statements – Grundkenntnisse
Java-Umgebung – Grundkenntnisse
JDBC – Grundkenntnisse

1.3 Aufgabenstellung

Verwenden Sie Ihren Code aus der Aufgabenstellung "Simple JDBC Connection" um Zugriff auf die Postgresql Datenbank "Schokofabrik" zur Verfügung zu stellen. Dabei sollen die Befehle (CRUD) auf die Datenbank mittels PreparedStatements ausgeführt werden. Verwenden Sie mindestens 10000 Datensätze bei Ihren SQL-Befehlen. Diese können natürlich sinn frei mittels geeigneten Methoden in Java erstellt werden. Die Properties sollen dabei folgende Keys beinhalten: host, port, database, user, password Vergessen Sie nicht auf die Meta-Regeln (Dokumentation, Jar-File, etc.)! Die Testfälle sind dabei zu ignorieren. Diese Aufgabe ist als Gruppenarbeit (2 Personen) zu lösen.

2 Prepared Statement [2]

2.1 Erklärung

Ein Prepared Statement ist eine sogenannte vorbereitete Anweisung für ein Datenbanksystem. Im Gegensatz zu gewöhnlichen Statements enthält es noch keine Parameterwerte. Stattdessen werden dem Datenbanksystem Platzhalter übergeben. Mittels Prepared Statements können SQL-Injections effektiv verhindert werden, da das Datenbanksystem die Gültigkeit von Parametern prüft, bevor diese verarbeitet werden.

Soll ein Statement mit unterschiedlichen Parametern mehrere Male (z. B. innerhalb einer Schleife) auf dem Datenbanksystem ausgeführt werden, können Prepared Statements einen Geschwindigkeitsvorteil bringen, da das Statement schon vorübersetzt im Datenbanksystem vorliegt und nur noch mit den neuen Parametern ausgeführt werden muss.

2.2 Allgemeines Beispiel

```
String sql = "update people set firstname=? , lastname=? where id=?";

PreparedStatement preparedStatement =
        connection.prepareStatement(sql);

preparedStatement.setString(1, "Gary");
preparedStatement.setString(2, "Larson");
preparedStatement.setLong  (3, 123);

int rowsAffected = preparedStatement.executeUpdate();
```

Bei diesem Beispiel werden mittels Fragezeichen Werte Definiert die eingesetzt werden können. Die Reihenfolge der Werte ist ganz wichtig. Diese eingesetzte Querry muss dann noch ausgeführt werden, dies geht Mittels: executeUpdate ()

2.3 Spezifisches Beispiel

Insert

Hier wird die Entsprechende Insert-Methode definiert. Zuerst werden im Parameter die Werte übergeben, die die Querry braucht. Dann werden durch die Fragezeichen diese Werte erwartet. Durch die .setType Methoden werden diese Werte eingefügt und Executet.

```
public void insert(int nummer, String beschreibung, Connection con) throws SQLException {

    String sql = "INSERT INTO Maschine (nummer, beschreibung) VALUES (?, ?)";

    PreparedStatement statement = con.prepareStatement(sql);
    statement.setInt(1, nummer);
    statement.setString(2, beschreibung);

    statement.executeUpdate();
}
```

Select

Hier wird die Entsprechende Select-Methode definiert. Diese „Funktion" brauch nur die Connection für die Ausgabe der Datensätze. Mit einer while schleife kann man alles möglichen Datensätze ausgeben.

%d: Sind die Durchläufe.
%s: Die Platzhalter für Werte die eingesetzt werden.

Die müssen natürlich gefüllt werden.
Dies muss in der Richtigen Reihenfolge eingesetzt werden!

```
public ResultSet select(Connection con) throws SQLException {

    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT * FROM Maschine");
    int count = 0;
    while (rs.next()) {
        String nummer = rs.getString(1);
        String beschreibung = rs.getString(2);

        String output = "Person #%d: %s - %s";
        System.out.println(String.format(output, ++count, nummer, beschreibung));
        }
    return null;
}
```

Update

Hier wird die Entsprechende Update-Methode definiert.
Zuerst werden im Parameter die Werte übergeben, die die Querry braucht.
Dann werden durch die Fragezeichen diese Werte erwartet. Durch die .setType Methoden werden diese Werte eingefügt und Executet.

```
public void update(Integer nummer, String beschreibung, Connection con) throws SQLException {

        String sql = "UPDATE Maschine SET beschreibung=? WHERE nummer=?";

        PreparedStatement statement = con.prepareStatement(sql);
        statement.setString(1, beschreibung);
        statement.setInt(2, nummer);

        statement.executeUpdate();
}
```

Delete

Hier wird die Entsprechende Delete-Methode definiert.
Zuerst werden im Parameter die Werte übergeben, die die Querry braucht.
Dann werden durch die Fragezeichen diese Werte erwartet. Durch die .setType Methoden werden diese Werte eingefügt und Executet.

```
public void delete(int nummer, Connection con) throws SQLException {

    String sql = "DELETE FROM Maschine WHERE nummer=?";

        PreparedStatement statement = con.prepareStatement(sql);
        statement.setInt(1, nummer);

        statement.executeUpdate();

}
```

3 Command-Line-Interface [1] [3]

3.1 Voreinstellungen

1. JarFile ins Projekt einbinden und die Imports erstellen.
2. (Intellij) Projekt

3.2 Erstellen von Optionen

Als erstes wird ein Options Objekt angelegt. Dann für alle benötigten Optionen jeweils eine Option (ohne s) Objekt das den Options hinzugefügt werden.

```
Options lvOptions = new Options();

lvOptions.addOption("h", "hilfe", false, "zeigt diese Hilfe an.");
```

Hierbei ist der erste Parameter der kurze Name der Option. Hier im Beispiel das h. Auf der Kommandozeile muss dann -hingegeben werden.
Als Alternative ist der 2. Parameter, der den langen Namen der Option enthält. Dies kann von der Kommandozeile mit --Hilfe aufgerufen werden. Der 3. Parameter gibt an, ob die Option ein Argument enthält (=true) oder nicht (=false). Der letzte Parameter gibt einen Text an, der die Option beschreibt. Dieser Text wird dann als Hilfe ausgegeben.

```
Option lvName = new Option("name", true, "der Namen des Nutzers.");

lvOptions.addOption(lvName);
```

3.3 Allgemeines Beispiel

```
public static void main(String[] args) {
    Options options = new Options();
    options.addOption("myCmd", "myCommand", false, "will run myCommand()." );
    options.addOption("helloW", "helloWord", true, "display hello word the number

    try{
        CommandLine line = new BasicParser().parse( options, args );

        if( line.hasOption( "myCommand" ) ) {
            myCommand();
        }

        if(line.hasOption("helloWord")){
            String repeat = line.getOptionValue("helloWord");
            Integer repeatInt = new Integer(repeat);
            for(int i =0; i<repeatInt; i++){
                System.out.println( "Hello word !");
            }
        }

    }catch( ParseException exp ) {
        System.out.println( "Unexpected exception:" + exp.getMessage() );
    }
}

public static void myCommand(){
    System.out.println("myCommand() just get call");
}
```

3.4 Spezifisches Beispiel

Optionen Definieren

Eine Command-Line-Parser erstellen, damit man die Benutzereingaben „einbinden" kann.
Die Options für die Verbindung zur Datenbank werden erstellt und mit dem geeigneten Variablen definiert.

```
CommandLineParser CLP  = new DefaultParser();

Options options = new Options();
options.addOption("h_",true,"IP-Adresse");
// options.addOption("h_","IP-Adresse",true,"IP-Adressen-Feld");
options.addOption("P_",true,"Port");
// options.addOption("P_","Port",true,"Port-Feld");
options.addOption("d_",true, "Datenbank");
options.addOption("u_",true,"Benutzername");
options.addOption("p_",true,"Passwort");
options.addOption("help",true, "Help");
```

Optionen Value Weiterleiten

Die Values werden wie gesagt durch den CLP(Command-Line-Parser), vom Benutzer, geholt und in der PostgreSQLConnection Klasse, in die static variable gespeichert.

```
CommandLine CL =  CLP.parse( options , args );

PostgreSQLConnection.h_ = CL.getOptionValue("h_");
PostgreSQLConnection.P_ = Integer.valueOf(CL.getOptionValue("p_"));
PostgreSQLConnection.d_ = CL.getOptionValue("d_");
PostgreSQLConnection.u_ = CL.getOptionValue("u_");
PostgreSQLConnection.p_ = CL.getOptionValue("u_");
PostgreSQLConnection PSQLC= new PostgreSQLConnection();
```

4 Aufrufen der Methoden [3]

4.1 Voreinstellung

Zuerst muss ein Start und End Timer für jede CURD-Methode erstellt werden. Danach erstellt man für jede CRUD-Methode den Gesamt Timer.

```
public float TStart = Float.parseFloat(null);
public float TEnde = Float.parseFloat(null);

public float ITime = Float.parseFloat(null);
public float STime = Float.parseFloat(null);
public float UTime = Float.parseFloat(null);
public float DTime = Float.parseFloat(null);
```

4.2 Beispiele

Insert

```
Connection con = PGSDS.getConnection();
Statement st = con.createStatement();
st.executeQuery("SELECT * FROM Maschine");

TStart = System.nanoTime();

for(int i=700; i <= 100700; i++) {
    statements.insert(i, "MaschineneBezeichnung_"+i, con);
}

TEnde = System.nanoTime();
ITime = TStart - TEnde;
```

Select

```
TStart = System.nanoTime();
statements.select(con);
TEnde = System.nanoTime();
STime = TEnde - TStart;
```

Update

```
TStart = System.nanoTime();
for(int i=700; i<100700; i++){
    statements.update(i,"MaschineneBezeichnung_"+i,con);
}
TEnde = System.nanoTime();
UTime = TEnde - TStart;
```

Delete

```
TStart = System.nanoTime();
for(int i=700; i<100700; i++){
    statements.delete(i,con);
}
TEnde = System.nanoTime();
DTime = TEnde - TStart;
```

5 CRUD [5]

Das Akronym CRUD [kɹʌd] umfasst die grundlegenden Datenbankoperationen

- Create, Datensatz anlegen,
- Read oder Retrieve, Datensatz lesen,
- Update (Datensatz aktualisieren) und
- Delete oder Destroy (Datensatz löschen).

CRUD-Operation	SQL-92	HTTP (REST)
Create	`INSERT`	`PUT` oder `POST`
Read (Retrieve)	`SELECT`	`GET`
Update	`UPDATE`	`PATCH` oder `PUT`
Delete (Destroy)	`DELETE`	`DELETE`

6 Aufwandsabschätzung

Arbeit	Stundenabschätzung
CLI	2:00 h
Prepared Statement	1:00 h

7 Literatur Verzeichnisse

Information
GitHub: https://github.com/mkanyildiz01/TGM-SEW-INSY.git

8 Quellen

INDEX	Quelltitel + Information
[1]	Wikipedia: http://commons.apache.org/proper/commons-cli/ Kapitel: 3 zuletzt abgerufen am [03.05.2016]
[2]	Wikipedia: https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html Kapitel: 2,5 zuletzt abgerufen am [03.05.2016]
[3]	Wikipedia: http://stackoverflow.com/questions/1200054/java-library-for-parsing-command-line-parameters Kapitel: 3 zuletzt abgerufen am [03.05.2016]