JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll

Prepared Statements

Informationssyssteme
4CHIT 2015/16, Gruppe A

Borsos Robert, Frassl Gabriel

Version 1.0

Note:

Betreuer: M. Borko

Begonnen am 23. Februar 2016

Beendet am 24. Februar 2016

Inhalt

1 Einleitung

PreparedStatements sind in JDBC eine Möglichkeit SQL-Befehle vorzubereiten um SQL-Injections zu vermeiden. Die Typüberprüfung kann somit schon bei der Hochsprache abgehandelt werden und kann so das DBMS entlasten und Fehler in der Businesslogic behandelbar machen.

2 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.

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 sinnfrei 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.

4 Ergebnisse

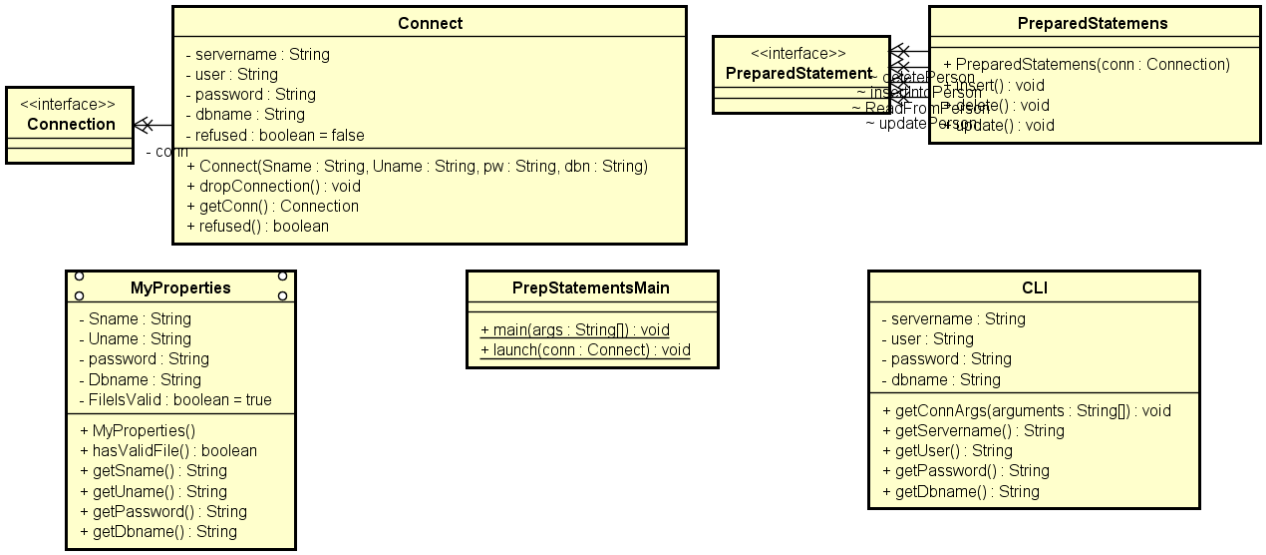
4.1 Prepared Statements

Prepared Statements sind vorbereitete Anweisungen an die Datenbank. Es wird eine Anweisung mit verschiedenen Parametern erstellt, welche mit einem „?“ gekennzeichnet sind. Diese Anweisungen können dann auch häufiger mit unterschiedlichen Werten (Parametern) ausgeführt werden.

Wenn die Datenbank auf eine Abfrage vorbereitet wird, kann die Datenbank die Vorgehensweise zur Ausführung der Abfrage analysieren, kompilieren und optimieren. Da der Zyklus der Analyse/Kompilierung/Optimierung nun früher geschieht wie sonst, benötigen Prepared Statements weniger Ressourcen und funktionieren dadurch schneller. Zudem können somit SQL-Injections verhindert werden.

```
INSERT INTO REGISTRY (name, value) VALUES (?, ?);
```

4.2 UML Designüberlegung



Das UML Klassendiagramm des Java Codes.

4.3 Lösungsweg

Es wurden 6 Klassen mit konkreten Aufgaben erstellt.

PrepStatementsMain: In der Main Methode wird geprüft, ob Daten zur Verbindung zur Datenbank in „properties.properties“ befinden. Wenn dort keine

vorhanden sind, werden die Daten aus der CLI genommen. Weiteres werden dann die Inserts, Updates und Deletes durchgeführt.

Connect: Es wird eine Verbindung zur Datenbank aufgebaut. Diese Klasse, welche schon in einem vergangen Beispiel (Simple JDBC Connection) gemacht wurde, wurde als Grundlage für das neue Beispiel verwendet.

MyProperties: Diese Klasse sucht nach dem „properties.properties"-File und speichert dann den gefundenen Inhalt in eigenen Variablen.

PreparedStatements: Hier werden die Prepared Statements erstellt und anschließend mit Daten gefüllt.

CLI: In der Klasse CLI werden die Daten aus der CLI gespeichert.

4.4 Code

Der Code selber ist schon im Java Programm bereits dokumentiert. Hier werden bloß die wichtigen Teile nochmal erklärt.

PrepStatementsMain.java (Bild 1)

```
MyProperties props = new MyProperties();
if (props.hasValidFile()) { // if a file was found
    Connect conn = new Connect(props.getSname(), props.getUname(),
            props.getPassword(), props.getDbname());// trying to
                                                    // connect
```

Es wird von MyProperties ein Objekt erstellt. In MyProperties werden die Daten aus dem Ressourcen File „properties.properties" ausgelesen in Variablen gespeichert. Für den Zugriff auf diese Informationen werden dann getter-Methoden erstellt. (Bereits in dem Beispiel „Simple JDBC Connection" gemacht) Anschließend wird ein Objekt von Connect erzeugt. Diese stellt eine Verbindung zur Datenbank her. Diese Klassenmethode benötigt dann folgende Parameterwerte um eine Verbindung aufzubauen:

Connect.java (Bild 2)

```
public Connect(String Sname, String Uname, String pw, String dbn){
```

Nun wird dann bei *Bild 1* die getter mit den entsprechenden Informationen aus der Klasse *MyProperties.java* an *Connect.java* als Parameterwerte übergeben.

PrepStatementsMain.java (Bild 3)

```
CLI cli = new CLI();
cli.getConnArgs(args);
conn = new Connect(cli.getServername(),cli.getUser(),
        cli.getPassword(), cli.getDbname()); // trying to connect with cli args
```

Es wird wieder ein Objekt von der Klasse CLI erstellt. Wenn keine Daten in dem Propertie File gefunden werden, nimmt das Programm die Eingaben der CLI (Bild 3).

CLI.java (Bild 4)

```
34      public void getConnArgs(String[] arguments) throws ParseException {
35
36          // Options are made
37          Options options = new Options();
38
39          options.addOption("s", "serveraddress", true, "serveraddress");
40          options.addOption("u", "user name", true, "username");
41          options.addOption("p", "password", true, "password");
42          options.addOption("d", "database name", true, "database name");
43
44          HelpFormatter formatter = new HelpFormatter();
45          formatter.printHelp("ant", options);
46
47
48          CommandLineParser parser = new DefaultParser();
49          CommandLine line;
50          line = parser.parse(options, arguments);
51
52
53          // arguments are saved into classvariables
54          this.servername = line.getOptionValue("s");
55          this.user = line.getOptionValue("u");
56          this.password = line.getOptionValue("p");
57          this.dbname = line.getOptionValue("d");
```

In der Klasse CLI werden „Options" erstellt, welche die Eingabe definieren. Nun wird dann in der Zeile 50 mit dem CommandLineParser die Eingabe der CLI („arguments") mit den Options (Options - Definition der Syntax) abgeglichen. *„arguments"* sind die CLI Eingaben. Die CLI Eingaben können entweder über die Run Konfiguration oder über die CLI eingeben werden (genauere Erklärung noch bei 4.5 Test).
Zum Schluss werden dann ab Zeile 54 Bild 4 die Eingaben in Variablen im Programm gespeichert. Diese Variablen haben getter Methoden die dann in Bild 3 in „Connect" eingetragen werden.

PreparedStatementsMain.java (Bild 5)

```
public static void launch(Connect conn) {
    PreparedStatemens myPrepStats;
    try {
        myPrepStats = new PreparedStatemens(conn.getConn());
        myPrepStats.insert(); // adds Data into Person
        myPrepStats.update(); // updates a column
        myPrepStats.delete(); // deletes the Data that has been added in
                              // Person
        conn.dropConnection(); // drops connection to database
    } catch (SQLException e) {
        // TODO Auto-generated catch block

    }

}
```

Die Methode „launch" aus der Main Klasse setzt dann alles zusammen. „Launch" wird ebenso in der Main Klasse aufgerufen und es wird die Connection übergeben (Bild 2).

Nun wird ein Objekt von „PreparedStatements" erzeugt.

PreparedStatements.java (Bild 6)

```
public PreparedStatemens(Connection conn) throws SQLException {
    insertIntoPerson = conn
            .prepareStatement("INSERT Into person Values (?,?,?)");
    ReadFromPerson = conn
            .prepareStatement("Select count(*) From Person WHERE vorname LIKE (?)");
    updatePerson = conn
            .prepareStatement("UPDATE person SET nachname = ? WHERE nummer = ?");
    deletePerson = conn
            .prepareStatement("DELETE FROM person WHERE nummer = ?");
}
```

In der Klasse „PrepairedStatements" werden die Aussagen (Statements) dann vorbereitet. Diese werden dann in derselben Klasse von verschiedenen Methoden aufgerufen und mit Daten gefüllt.

Nun werden dann bei Bild 5 die verschiedenen Methoden von der Klasse „PreparedStatments" aufgerufen und ausgeführt. Zuletzt wird dann noch die Verbindung getrennt.

4.5 Test

Hier werden Tests durchgeführt um die Funktionalität zu testen.

4.5.1 Property File

Inhalt des Property Files (Bild 7):

properties.properties (Bild 7)

```
server=10.0.105.142
database=schokofabrik
user=schokouser
password=schokoUser
```

Im Property File werden in der oben gezeigten Syntax die Information der Datenbank angegeben.

Wenn nun das dann funktioniert hat, sollte folgendes sichtbar sein (Bild 8):

(Bild 8)

```
propertiesfile correct
Read: Nach dem inserts exestieren 10000 Personen deren Vorname mit Max anfängt
Read: Nach dem delete exestieren 0 Personen deren Vorname mit Max anfängt
```

Das löschen (drop/delete) der eingefügten Daten ist wichtig, damit es beim nächsten Programmstart alles wieder einwandfrei funktioniert.

Test des Tests:

Um nochmal wirklich sicher zu gehen, dass das Programm funktioniert, kommentieren wir die folgende Zeile, wie vorgezeigt, aus der Methode „launch" raus.

```
//myPrepStats.delete();
```

Nun sollten die eingetragenen Daten nicht wieder gelöscht werden und somit sollten die Daten in der Datenbank noch sichtbar sein. Dies überprüfen wir indem wir das Programm neu starten.

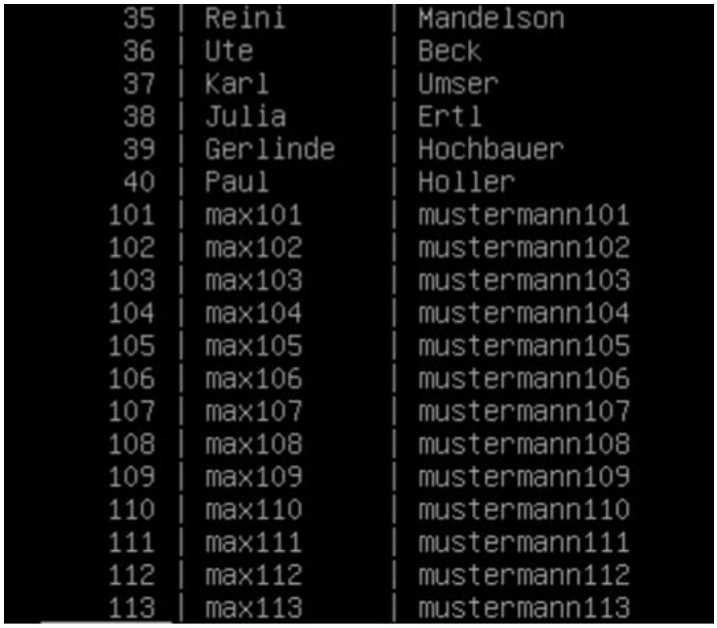
Danach verbinden wir uns mit der Datenbank mit dem folgenden Befehl:

psql -h localhost -U schokouser -W -d schokofabrik

Um nun zu schauen ob in „Person" noch die Daten vorhanden tippen wir folgendes:

SELECT * FROM person;

Wenn alles richtig gemacht wurde, sollte folgendes sichtbar sein:



35	Reini	Mandelson
36	Ute	Beck
37	Karl	Umser
38	Julia	Ertl
39	Gerlinde	Hochbauer
40	Paul	Holler
101	max101	mustermann101
102	max102	mustermann102
103	max103	mustermann103
104	max104	mustermann104
105	max105	mustermann105
106	max106	mustermann106
107	max107	mustermann107
108	max108	mustermann108
109	max109	mustermann109
110	max110	mustermann110
111	max111	mustermann111
112	max112	mustermann112
113	max113	mustermann113

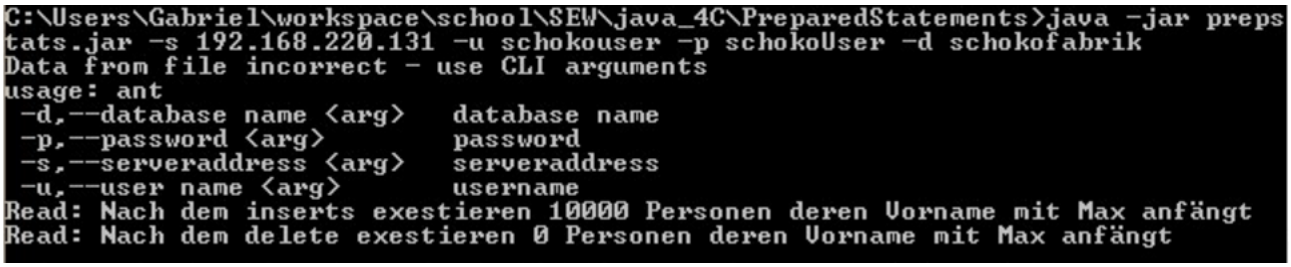
Ab 101 sind die Personen vom Programm erstellt worden.

4.5.2 CLI

Öffnen des CMD unter Windows. Dort das File *prepstats.jar* suchen. Nun mit dem Befehl

Java –jar prepstat.jar -s 10.0.105.142 -u schokouser -p schokoUser -d schokofabrik

das File starten. Nun sollte wieder eine Bestätigung kommen und somit hat das Programm funktioniert.



```
C:\Users\Gabriel\workspace\school\SEW\java_4C\PreparedStatements>java -jar preps
tats.jar -s 192.168.220.131 -u schokouser -p schokoUser -d schokofabrik
Data from file incorrect - use CLI arguments
usage: ant
 -d,--database name <arg>   database name
 -p,--password <arg>        password
 -s,--serveraddress <arg>   serveraddress
 -u,--user name <arg>       username
Read: Nach dem inserts exestieren 10000 Personen deren Vorname mit Max anfängt
Read: Nach dem delete exestieren 0 Personen deren Vorname mit Max anfängt
```

5 Arbeitsteilung und Zeitschätzung

Borsos: Erstellen des Protokolls und testen des Programmes
Zeitaufwand geschätzt: 270 Minuten
Zeitaufwand real: 250 Minuten

Frassl: Erstellen des Java Programmes
Zeitaufwand geschätzt: 300 Minuten
Zeitaufwand real: 340 Minuten

6 Quellen

[1] Apache Commons CLI; Online: http://commons.apache.org/proper/commons-cli/
[2] Java Tutorial JDBC "Prepared Statements"; Online: https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html
[3] Java Tutorial Properties; Online: https://docs.oracle.com/javase/tutorial/essential/environment/properties.html
[4] Overview of Java CLI Libraries; Online: http://stackoverflow.com/questions/1200054/java-library-for-parsing-command-line-parameters