JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll zur Übung

Prepared Statements

INSY - Datenbanken
4CHITT 2015/16

Daniel May
Martin Weber

Version 1.0

Note:

Betreuer: Michael Borko

Begonnen am 18. Februar 2016

Beendet am 25. Februar 2016

Inhaltsverzeichnis

1 Einführung

„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.“ [BOR]

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.“ [BOR]

1.2 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.“ [BOR]

1.3 Quellen der Angabe

„[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“ [BOR]

2 Designüberlegung

Die Applikation wird für diese Aufgabenstellung in mehrere Klassen unterteilt.

- DBConnection: um die Verbindung mit der Datenbank handzuhaben
- CLParser: um die Kommandozeilenargumente und das Property File zu verwalten
- PSMain: um die Teilaufgaben zusammenzufügen
- CRUD: um die Prepared Statements durchzuführen. CRUD wird ebenfalls noch unterteilt, sodass die eigentlichen Klassen wie folgt heißen:
    - PSCreate
    - PSRead
    - PSUpdate
    - PSDelete

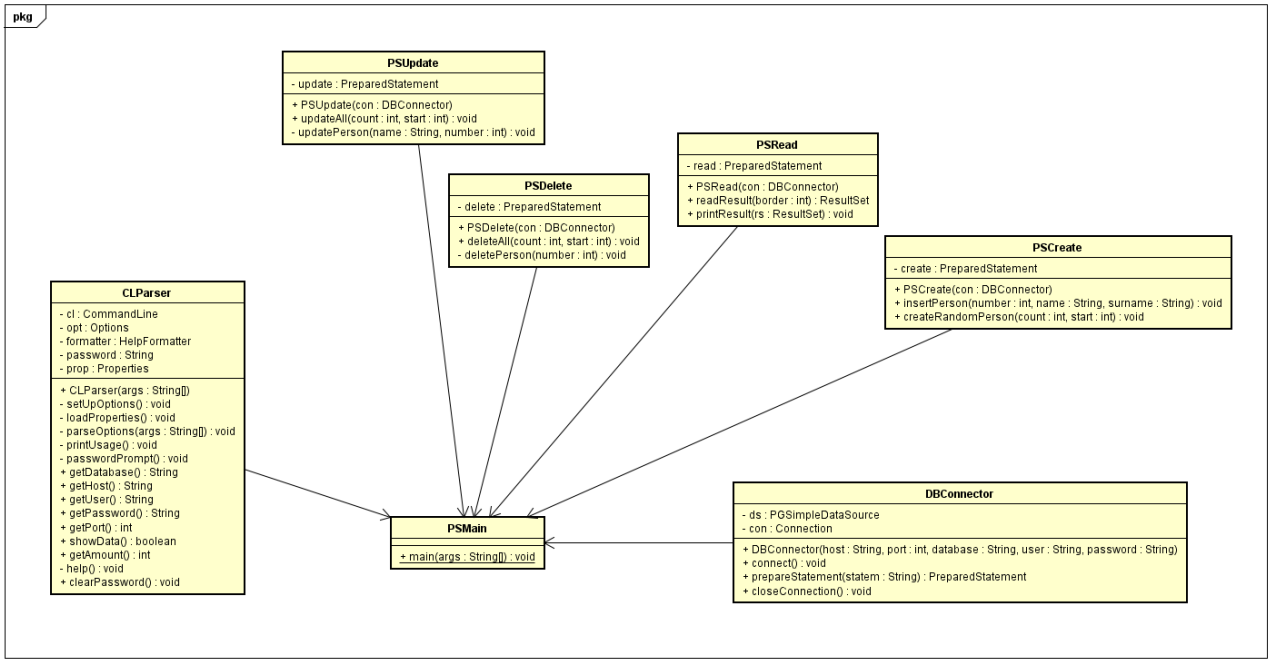

Abbildung 1: UML Diagramm

3 Ergebnisse

3.1 Apache Commons CLI

Apache Commons CLI ist eine Bibliothek um Kommandozeilenargumente auszulesen und zu verarbeiten. Dies geschieht in 3 Schritten. Zuerst müssen die erlaubten Optionen definiert werden. Dazu muss ein Options Objekt erstellt werden, welchem später die Optionen hinzugefügt werden. [APA]

```
opt = new Options();
opt.addOption(Option.builder("d").argName("database-name").desc("database name to connect to").hasArg()
        .longOpt("database").numberOfArgs(1).required().build());
```

Abbildung 2: Optionen definieren

Mittel sogenannten OptionGroups kann man Optionen die sich gegenseitig ausschließen definieren.
Danach werden die Argumente aus der Main Methode geparsed. [APA]

```
DefaultParser parser = new DefaultParser();
try {
    cl = parser.parse(opt, args);
```

Abbildung 3: Parsen der Argumente

Als Ergebnis erhält man ein CommandLine Objekt, welches die Informationen aus der CLI Eingabe enthält. Zuletzt muss man dann noch im eigenen Code auf die Eingabe reagieren. [APA]

```
cl.hasOption('W')
```

Abbildung 4: Überprüfen ob ein Switch gesetzt wurde

3.2 Property Files

Mithilfe des Property Files können Konfigurationen einfach abgespeichert und eingelesen werden. Die einzelnen Werte sind folgendermaßen im File abzuspeichern: key=value.
Zuerst wird ein Properties Objekt erstellt. In dieses können dann die einzelnen Properties aus dem File mit einem Stream geladen werden.

```
prop = new Properties();
prop.load(is);
```

Abbildung 5: Erstellen des Objekts und laden des Files

Falls ein Wert nicht über die CLI angegeben wurde, wird dieser, falls vorhanden, aus dem Property File nachgeladen.

```
if (cl.hasOption("H"))
    return cl.getOptionValue('H');
else {
    if (!prop.containsKey("host")) {
        System.err.println("Missing option host");
        System.exit(-1);
        return "";
    } else {
        return prop.getProperty("host");
    }
}
```

Abbildung 6: Beispiel Nachladen der Werte für den Hostnamen

3.3 Prepared Statements

Create

Erstellen eines Prepared Statements.

```
public PSCreate(DBConnector con) {
    create = con.prepareStatement("INSERT INTO person VALUES(?,?,?)");
}
```

Abbildung 7: Erstellen des Prepared Statements

```
private void insertPerson(int number, String name, String surname) {
    try {
        create.setInt(1, number);
        create.setString(2, name);
        create.setString(3, surname);
        create.execute();
    } catch (SQLException e) {
        System.err.println("Inserting a new Person failed");
        System.err.println(e.getMessage());
    }
}
```

Abbildung 8: Create Implementierung

Hier wird eine Person eingefügt. Die einzelnen Werte im Prepared Statement werden gefüllt und dann wird die Query ausgeführt.

```
public void createRandomPerson(int count, int start) {
    for (int i = start; i < count + start; i++) {
        /*
         * Auf Stackoverflow gefunden generiert random ID
         * http://stackoverflow.com/questions/2863852/how-to-generate-a-
         * random-string-in-java
         */
        String s = UUID.randomUUID().toString().substring(0, 20);
        insertPerson(i, s, s);
    }
}
```

Abbildung 9: Erstellen von random Inserts

Hier werden <count> Personen in die Datenbank mit zufälligen sinnfreien Werten eingefügt.

Read

Das Prepared Statement.

```
public PSRead(DBConnector con) {
    read = con.prepareStatement("SELECT * FROM person WHERE nummer > ?");
}
```

Abbildung 10: Prepared Statement read

Auslesen der DB mithilfe des Prepared Statements

```
public ResultSet readResult(int border) throws SQLException {
    read.setInt(1, border);
    return read.executeQuery();
}
```

Abbildung 11: Read implementierung

Ausgeben des Result Sets.

```
public void printResult(ResultSet rs) {
    try {
        while (rs.next()) {
            int number = rs.getInt(1);
            String name = rs.getString(2);
            String surname = rs.getString(3);
            System.out.println(number + " " + name + " " + surname);
        }
    } catch (SQLException e) {
        System.err.println("printing the ResultSet Failed");
    }
}
```

Abbildung 12: Ausgabe Read

Update

Das Update Prepared Statement.

```
public PSUpdate(DBConnector con) {
    update = con.prepareStatement("UPDATE person SET vorname = ? WHERE nummer = ?;");
}
```

Abbildung 13: Prepared Statement Update

```
private void updatePerson(String name, int number) {
    try {
        update.setString(1, name);
        update.setInt(2, number);
        update.execute();
    } catch (SQLException e) {
        System.err.println("Updating a Person failed.");
        System.err.println(e.getMessage());
    }
}
```

Abbildung 14: Update auf eine Person

```
public void updateAll(int count, int start) {
    for (int i = start; i < count + start; i++) {
        updatePerson("Vorname" + i, i);
    }
}
```

Abbildung 15: Update auf mehrere Personen

Delete

Das Delete Prepared Statement.

```
public PSDelete(DBConnector con) {
    delete = con.prepareStatement("DELETE FROM person WHERE nummer = ?;");
}
```

Abbildung 16: Delete Statement

```
private void deletePerson(int number) {
    try {
        delete.setInt(1, number);
        delete.execute();
    } catch (SQLException e) {
        System.err.println("Deleting a Person failed.");
        System.err.println(e.getMessage());
    }
}
```

Abbildung 17: Delete Implementierung

```
public void deleteAll(int count, int start) {
    for (int i = start; i < count + start; i++) {
        deletePerson(i);
    }
}
```

Abbildung 18: löschen mehrerer Personen

GitHub Repository:

https://github.com/mweber-tgm/PreparedStatements

4 Teamarbeit

Daniel May übernimmt die Main, CLI Parser, Connector, Update und Delete Klasse. Martin Weber übernimmt das Property File, sowie die Create und Read Klassen. Am Protokoll wird gemeinsam gearbeitet.

Der Aufwand wird wie folgt geschätzt:

Arbeitsteil	**Durchführender**	**geschätzter Aufwand**
CLI Parser	Daniel May	2 h
Main	Daniel May	0.25 h
Connection	Daniel May	0.25 h
Update	Daniel May	0.5 h
Delete	Daniel May	0.5 h
Create	Martin Weber	0.5 h
Read	Martin Weber	0.5 h
Property File	Martin Weber	1.5 h
Protokoll	D. May & M. Weber	0.5 h/Person
Gesamt	**D. May & M. Weber**	**7 h**

Der tatsächliche Aufwand ist wie folgt aufgeschlüsselt:

Daniel May:

Durchgeführte Arbeit	**Datum**	**tatsächlicher Aufwand**
CLI Parser	23. -25.02.2016	3.5 h
Main	23.02.2016	0.1 h
Connector	23.02.2016	0.1 h
Update	24.02.2016	0.25 h
Delete	24.02.2016	0.25 h
Protokoll	24. – 25.02.2016	1 h
Gesamt	**23. – 25.02.2016**	**5.2 h**

Martin Weber:

Durchgeführte Arbeit	**Datum**	**tatsächlicher Aufwand**
Property File	24.-25.02.2016	2.5 h
Create	24.-25.02.2016	0.5 h
Read	24.-25.02.2016	0.6 h
Protokoll	24.-25.02.2016	1 h
Gesamt	**24.-25.02.2016**	**4.6 h**

Gesamt:

Durchführender	**tatsächlicher Aufwand**
Daniel May	5.2 h
Martin Weber	4.6 h
Gesamt	**9.8 h**

5 Literaturverzeichnis

[BOR] Michael Borko. (2016, February). Prepared Statements [Online]. Available at: https://elearning.tgm.ac.at/mod/assign/view.php?id=47181 [abgerufen am 25.02.2016]

[APA] Apache. (2015, June). User Guide: Getting started [Online]. Available at: http://commons.apache.org/proper/commons-cli/introduction.html [abgerufen am 25.02.2016]