JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll zur INSY-Übung „Prepared Statements“

INSY
4AHITM 2015/16

Eren Sefer – Yunus Sari

Version 1.0

Note:

Betreuer: Michael Borko

Begonnen am 30. März 2016

Beendet am 27. Mai 2016

INHALTSVERZEICHNIS

1 Einführung in die Übung

1.1 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.

1.2 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.

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 sinnfrei mittels geeigneten Methoden in Java erstellt werden.

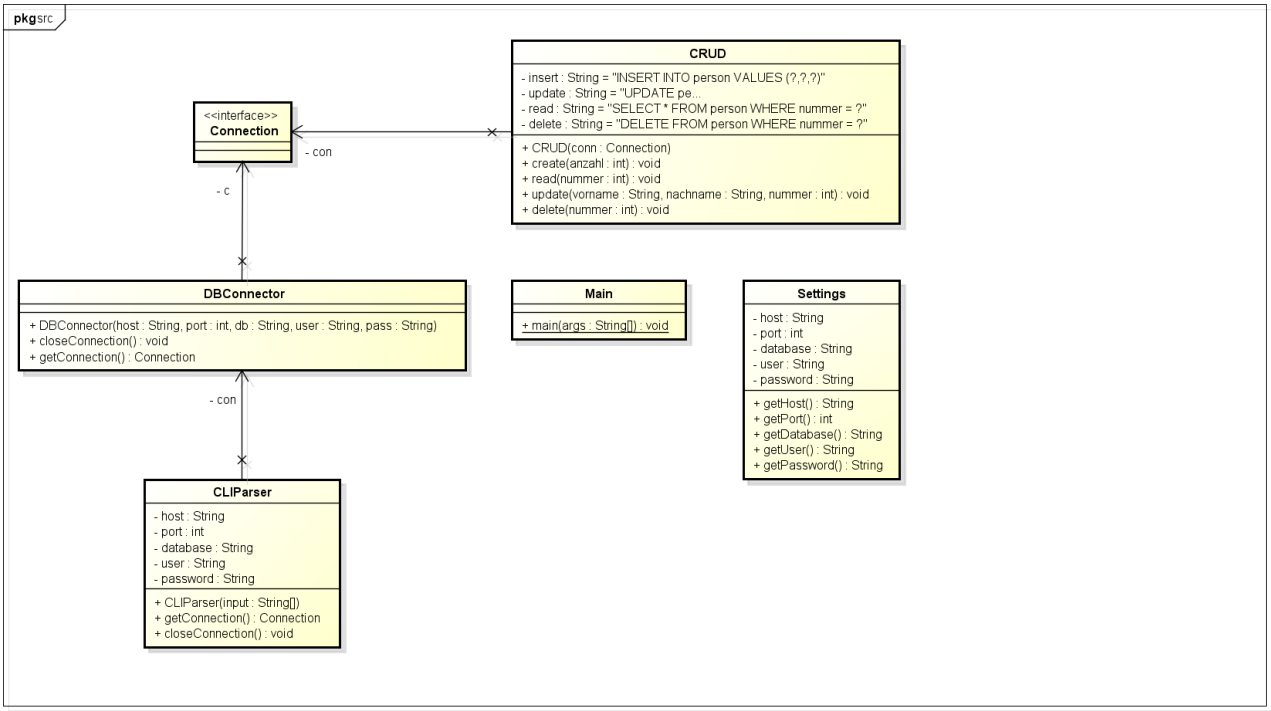
Die Properties sollen dabei folgende Keys beinhalten:
host, port, database, user, password

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.

1.4 Quelle

[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

2 DESIGNÜBERLEGUNG



Um die Aufgabe sinnvoll zu lösen und in sinngemäße Klassen aufzuteilen, wurden Überlegungen notiert und Rechercheuren durchgeführt. Dabei ist man auf folgendem Entschluss gekommen:

Die Aufgabe wurde in 4 Hauptklassen unterteilt (DBConnector, CRUD, CLIParser, Main).

- DBConnector

  Diese Klasse dient dazu die Verbindung zur Datenbank handzuhaben. Eine statische Version dieser Klasse war aufgrund der Übung „Simple Database Connection" vorhanden. Es wurde lediglich die Klasse aus dieser Übung erweitert und angepasst.

  Anfangs gab es bei beidem Teammitgliedern das Problem, dass sie auf die Datenbank nicht zugreifen konnten, obwohl richtige Eingaben getätigt wurden. Dieses Problem wurde gelöst, indem man in der pg_hba.conf Datei ein paar Änderungen durchgeführt wurde.

  Quelle:
  http://www.cyberciti.biz/tips/postgres-allow-remote-access-tcp-connection.html

- CRUD

  Wie der Name schon verrät, enthält diese Klasse die CRUD-Befehle und jene Methoden, die für die Prepared Statements notwendig sind.

- CLIParser

  Durch den CLIParser werden die properties (Parameter, die für die Verbindung zur Datenbank notwendig sind) gesetzt. Notwendig, um die dynamische Eingabe durch die Konsole zu ermöglichen.

  Es gab zwei Möglichkeiten, diesen Teil der Aufgabe zu implementieren. Zur Auswahl standen also zwei sinnvolle Libraries zur Verfügung. So begann die Anforderungsanalyse:

  Diese Libraries wurden analysiert:
  - Commons CLI

    Bei dieser Library ist der Schreibaufwand größer und die Weiterentwicklung der Applikation deswegen aufwändiger. Zusätzlich muss man bei dieser Library vieles noch selber implementieren, was beim Library "JCommander" fast alle zu benutzenden Funktionen schon vorhanden sind.

  - JCommander

    Da bei JCommander der Schreibaufwand weniger ist und diese Library einfacher zu implementieren ist, wurde diese Library als Schnittstelle verwendet. Zusätzlich ist die Dokumentation einfacher zu verstehen. Damit JCommander benutzt werden kann, musste der Source-Code aus der offiziellen GitHub-Seite in den Projektordner hinzugefügt werden (Zu finden unter der Ordnerhierarchie: jcommander). Danach werden die Packages im eigenen Sourcecode mit dem import-Befehl verwendet.

- Main

  In dieser Klasse werden alle (Teil-)Klassen zusammengefügt, sodass die Applikation verwendet werden kann.

- Settings

  Diese Klasse wird verwendet, damit die einzelnen Parameter, die in der Konsole übergeben werden können, definiert werden. Wird allein von CLIParser beansprucht.

3 Ergebnisse

3.1 Settings

```
public class Settings {

	@Parameter(names = "-h", required = true)
	private String host;

	@Parameter(names = "-p", required = false)
	private int port;

	@Parameter(names = "-d", required = true)
	private String database;

	@Parameter(names = "-u", required = true)
	private String user;

	@Parameter(names = "-pw", required = true)
	private String password;

	...
}
```

In dieser Klasse werden die properties gesetzt. Der Parameter port ist nicht zwingend anzugeben. Falls der Benutzer nichts angibt, wird der Defaultwert für PostgreSQL verwendet (5432). Zusätzlich gibt es für die privaten Klassenattribute noch getter-Klassen, um sie dem DBConnector übergeben zu können.

3.2 CRUD

3.2.1 Prepared Statement

```
private String insert = "INSERT INTO person VALUES (?,?,?)";
private String update = "UPDATE person SET vorname = ?, nachname = ? WHERE nummer = ?";
private String read = "SELECT * FROM person WHERE nummer = ?";
private String delete = "DELETE FROM person WHERE nummer = ?";
```

So werden in der CRUD-Klasse die Statements jeweils definiert. Die Fragezeichen werden dann mit Methoden (die für die CRUD-Befehle geschrieben wurden) ersetzt. Um es besser verstehen zu können, hier der Code zur read-Methode:

3.2.1.1 Read

```
public void read(int nummer) {
	PreparedStatement ps;
	try {
		ps = con.prepareStatement(read);
		ps.setInt(1, nummer);
		ResultSet tmp = ps.executeQuery();
		while (tmp.next()) {
			System.out.print("Nummer: " + tmp.getInt(1));
			System.out.print(", Vorname: " + tmp.getString(2));
			System.out.print(", Nachname: " + tmp.getString(3) + "\n");
		}
		ps.close();
	} catch (SQLException e) {
		System.out.println("Fehler beim Lesevorgang.");
		e.printStackTrace();
	}
}
```

3.2.2 Main

```
public class Main {
	public static void main(String[] args) {
		CLIParser cli = new CLIParser(args);
		Connection con = cli.getConnection();
		CRUD crud = new CRUD(con);
		crud.read(1);
		crud.update("Yunus", "Sari", 2);
		crud.delete(2);
		crud.create(12000);
		cli.closeConnection();
	}
}
```

Hier werden (wie vorher beschrieben) alle Teilklassen zusammengeführt und die CRUD-Befehle ausgetestet, wobei natürlich klarerweise für diese Befehle Prepared Statements verwendet werden.

4 Zeitaufzeichnung

4.1 Geschätzt

Arbeitsteil	Aufwand
Designüberlegung und Anforderungsanalyse	2 h
Übung	6 h 15 min
Statischer Code für Testung (DBConnecter)	15 min
DBConnector dynamisch	1 h
CLI-Parser	1 h
Main	1 h
Create	45 min
Read	45 min
Update	45 min
Delete	45 min
Protokoll	2 h
Gesamt	***10 h 15 min***

4.2 Tatsächlich

Arbeitsteil	Durchgeführt von	Aufwand
Designüberlegung und Anforderungsanalyse	Sari, Sefer	1 h 30 min / 1 h 30 min
Übung	Sari, Sefer	6 h
Statischer Code für Testung (DBConnecter)	Sefer	15 min
DBConnector dynamisch	Sari, Sefer	30 min / 30 min
CLI-Parser	Sari	30 min
Main	Sari, Sefer	15 min
Create	Sari	1 h
Read	Sari	1 h
Update	Sari	1 h
Delete	Sari	1 h
Protokoll	Sari, Sefer	1 h / 4 h
Summe		**14 h 15 min**
Sari		7 h 45 min
Sefer		6 h 30 min

4.3 Versionierung

GitHub-Link: https://github.com/ysari-tgm/preparedstatement

5 Quellen

5.1 Troubleshooting

http://www.cyberciti.biz/tips/postgres-allow-remote-access-tcp-connection.html

letzter Zugriff: 31.03.2016

5.2 JCommander

Source Code: https://github.com/cbeust/jcommander

Dokumentation: http://jcommander.org/

Letzter Zugriff: 20.05.2016

5.3 Commons CLI

https://commons.apache.org/proper/commons-cli/

Letzter Zugriff: 20.05.2016