|  |
| --- |
| The exporter  Anzeigen einer RDBMS Tabelle mittels JDBC  12.12.2014  4BHIT  Melanie Göbel |

Inhaltsverzeichnis

[Aufgabenstellung 2](#_Toc406148992)

[Aufwand und Zeit 3](#_Toc406148993)

[Schätzung und Realität 3](#_Toc406148994)

[Designüberlegung 3](#_Toc406148995)

[Arbeitsdurchführung 3](#_Toc406148996)

[Testbericht 3](#_Toc406148997)

# Aufgabenstellung

Erzeuge eine Konsolen-Anwendung (Java), die den Inhalt einer beliebigen Tabelle eines RDBMS als Textfile ausgibt (ein Datensatz pro Zeile, Kopfzeilen sind nicht notwendig)

Die Anwendung soll folgende Parameter entgegennehmen:

-h ... Hostname des DBMS. Standard: localhost  
-u ... Benutzername. Standard: Benutzername des im Betriebssystem angemeldeten Benutzers  
-p ... Passwort. Alternativ kann ein Passwortprompt angezeigt werden. Standard: keins  
-d ... Name der Datenbank  
-s ... Feld, nach dem sortiert werden soll (nur eines möglich, Standard: keines)  
-r ... Sortierrichtung. Standard: ASC  
-w ... eine Bedingung in SQL-Syntax, die um Filtern der Tabelle verwendet wird. Standard: keine  
-t ... Trennzeichen, dass für die Ausgabe verwendet werden soll. Standard: ;   
-f ... Kommagetrennte Liste (ohne Leerzeichen) der Felder, die im Ergebnis enthalten sein sollen. \* soll akzeptiert werden (Pflicht)  
-o ... Name der Ausgabedatei. Standard: keine -> Ausgabe auf der Konsole  
-T ... Tabellenname (Pflicht)

z.B.

java -jar TheExporter.jar -h projekte.tgm.ac.at -u test -p test -d megamarkt -T produkt -f titel,preis -s preis -r DESC -w "preis > 4" -t "$"

Achte darauf, dass die Parameter nur sinnvoll angegeben werden können z.B. -r macht nur Sinn mit -s, wo ein Standardwert vorgegeben ist, darf der Schalter auch weggelassen werden. Die Reihenfolge der Schalter soll keine Rolle spielen, jeder Schalter darf nur 1x vorkommen.

# Aufwand und Zeit

Die Aufgabe scheint nicht sehr einfach zu sein aber machbar. Wenn man eine gute Libary nimmt und bei der Verbindung aufpasst und nicht schlampig wird, wird die Arbeitszeit reduziert. Da es einige vernünftige Libaries gibt und ich schon letztes Jahr JDBC gemacht habe, hoffe ich nicht auf eine zusätzliche Arbeit um Probleme zu lösen.

## Schätzung und Realität

|  |  |  |
| --- | --- | --- |
| Arbeit | Geschätzte Zeit (in min) | Benötigte Zeit (in min) |
| Libary finden und testen für CLI | 30 | 80 |
| Parameterverarbeitung | 60 | 30 |
| Verbindung zur Datenbank | 30 | 15 |
| Abfrage der Tabelle | 30 | 10 |
| Erstellen der Ausgabe | 60 |  |
| Testen | 50 |  |
| Dokumentation | 50 |  |
| Zusätzliche Arbeit (zB. Bug fixinig) | 0 |  |
| Insgesamt | 310 = 5 Stunden 10 Minuten |  |

# Arbeitsdurchführung

## Evaluierung Command-Line-Interpreter Libary

Für die Verwaltung der Parameter soll eine Libary benutzt werden, bevor man eine wählt muss man jedoch wissen welche es gibt. Aus den vielen Libaries 7 frei zugängliche Libaries gefunden die sind [1]:

* Natural CLI
* JOpt Simple
* Jakarte Commons CLI
* Args4j
* jArgs
* argsparse4j

Für die Auswahl müssen mehrere Kriterien erfüllt werden wie zum Beispiel: Gute Dokumentation(Manual), Aktualität, JavaDocs, Testcases, einfache Beispiele und eine gute Community. Durch anschauen der Seiten von den Libaries kam es zu einer engeren Auswahl. Darunter waren Commons CLI, Args4j und argsparse4j. Um genau zu wissen welches von denen bei dem Anwendungsbereich am besten geeignet ist, muss man Beispielcode schreiben.

Von dem Beispielcode hat mir argsparse4j am besten gefallen. Es ist stets aktuell und auch gut Dokumentiert. Es basiert auf das Python-modul argsparse und wurde so gut wie möglich gleich gehalten wie dieses, wenn es die verschiedene Syntax zulässt.

Zu finden ist die Dokumentation sowie einfach Beispiele auf der Seite von argsparse4j[2]. Ebenso ist der Sourcecode auf Github einsehbar[3].

## Parameterverarbeitung

Mit argsparse4j ist die Parameterverarbeitung ganz simple. Einen ArgumentParser fügt man mehreren Argumenten hinzu, dies ist z.B. „-t“. Für dieses Argument kann man einiges festsetzen, benutzt habe ich:

.choices(“ASC”,”DESC”) Wert muss eines der beiden Werte sein

.help(“The name of the user”) Hinzufügen eines Textes für den Hilfetext

.setDefault(“localhost”) Setzt diesen Wert wenn nicht anders angegeben

.requires(true) Dieses Argument MUSS gesetzt werden! Default: false

Mit *Namespace res = parser.parseArgs(args);* kann man den Parser args hinzufügen, Fehlermeldungen werden selbst geschrieben sowie die Hilfe. Wenn man sich minimal vertippt kommt auch ein Vorschlag was man meinen hätte können.

Problem bei dem Beispielcode war, dass die Hilfe mit –h und –help aufrufbar ist, jedoch das –h für den Host verwendet werden soll. Problem dabei war die erste Zeile:

*ArgumentParser parser = ArgumentParsers.newArgumentParser("prog")*

Mit einer kurzen Einsicht in die JavaDoc erkennt man das ArgumentParser ein Interface ist und es sinnvoll ist dass man [ArgumentParserImpl](http://argparse4j.sourceforge.net/apidocs/net/sourceforge/argparse4j/internal/ArgumentParserImpl.html), [SubparserImpl](http://argparse4j.sourceforge.net/apidocs/net/sourceforge/argparse4j/internal/SubparserImpl.html) dazu braucht (implementiert ArgumentParser). In einen Konstruktor von ArgumentParserImpl kann man dann die Hilfe ausstellen. Mit .addArgument() kann man wieder –help hinzufügen und die Hilfe anzeigen.

## Connection

Grundlage für die Verbindung ist eine java.sql.Connection. Diese benötigt Informationen zur Verbindung, die von MysqlDataSource mit der Methode .getConnection() geholt wird. Im Grunde sieht das so aus:

MysqlDataSource ds = **new** MysqlDataSource();

ds.setServerName(hostname);

ds.setUser(user);

ds.setPassword(password);

ds.setDatabaseName(database);

java.sql.Connection con = ds.getConnection();

## Abfrage der Tabelle

Dieser Code ist dazu zuständig, SQL-Befehle abzuschicken und sie zu speichern:

Statement st = con.createStatement();

ResultSet rs = st.executeQuery(command);

Command wäre zB ein select das so aussieht: select id,nachname from Person where vorname=’Melanie’; Wichtig ist nachdem beides nicht mehr benötigt wird zu schließen!

## Erstellen der Ausgabe

„Das ResultSet ist ein Zeiger auf einen Datensatz des Ergebnisses (selbe Vorgehensweise wie ein Iterator). 1. Position ist vor den ersten Datensatz, letzte Position nach den letzten Datensatz.“[4]

Mit Hilfe des .next() und .getString(i) von ResultSet kann man das Ergebnis ausgeben. Nach jedem .getString(i) muss das Trennzeichen hinzugefügt werden und vor den nächsten .next() einen Zeilenumbruch.

String result = ““;

**while**(rs.next()){

**for**(**int** i = 1; i <= anz;i++){

result += rs.getString(i)+delimeter;

}

result += "\n";

}

# Testbericht

# Quellen

[1] Open Source Command Line Interpreters in Java – Java-Source.net

Verfügbar: <http://java-source.net/open-source/command-line> (zuletzt gesehen 12.12.2014)

[2] Argparse4j - The Java command-line argument parser library - Tatsuhiro Tsujikawa

Verfügbar: <http://argparse4j.sourceforge.net/> (zuletzt gesehen 18.12.2014)

[3] Github repository: [argparse4j](https://github.com/tatsuhiro-t/argparse4j) owner: [tatsuhiro-t](https://github.com/tatsuhiro-t)

Verfügbar: <https://github.com/tatsuhiro-t/argparse4j>

[4] Datenzugriff via Cursor am Beispiel JDBC – Erhard List

Verfügbar: <https://elearning.tgm.ac.at/course/view.php?id=846> (zuletzt gesehen 20.12.2014)