Admindokumentation SQLChecker
ACHTUNG VERALTET

Wintersemester 2018/2019

Databases and Information Systems (DBIS)
Johann Wolfgang Goethe-Universität Frankfurt am Main

Inhaltsverzeichnis

Vorwort Diese Dokumentation richtet sich Übungsleiter, welche Aufgaben für den SQLChecker erstellen.

1 Eine Aufgabe erstellen

Aus Sicht eines Übungsgruppenleiters sind vier Dateien erforderlich, um eine Aufgabe erfolgreich durchzuführen: Eine Template-Datei(.sqlt), ein Resetskript(.sql), eine Lösungs-Datei (.sql) und eine Evaluation-Konfigurationsdatei (.ini).

1.1 Die Tag-Annotation

Die Template-Datei, die Studentenabgabe und die Lösungsdatei haben eine spezielle Form. Sie besitzen Tag-Elemente, welche folgende Form haben: `/*<name>*/`. Der Name des Tags darf keine Whitespaces besitzen. Der gesamte Text zwischen zwei Tags gehört zum ersten Tag. Im Weiteren wird dieser Text als Rumpf eines solchen Tags bezeichnet. Diese Darstellung wurde von der ursprünglichen SQL-Checker-Version übernommen. Es gibt zur Zeit drei Schlüsselwort-Tags, welche nicht als Namen von Aufgaben erlaubt sind: `submission_name`, `authors` und `static`.

1.2 Das Template

Das Template ist eine Tag-annotierte Datei. Die Datei sollte einen `submission_name` definieren und keine `static`-Tags verwenden.

Eine Template-Datei wird von Studenten zum Initialisieren einer neuen Übung benötigt. Die Datei definiert, welche Aufgaben im SQLChecker zum Bearbeiten angezeigt werden und welcher Text als Code dieser Aufgaben beim Initialisieren der Aufgabe angezeigt werden soll.

Alle Tags, die keine Schlüsselwörter beinhalten, werden als Aufgaben interpretiert. Kein Name von Aufgaben darf doppelt verwendet werden. Der Rumpf eines Aufgaben-Tags ist der Code, der beim Initialisieren der Aufgabe angezeigt wird. Da der SQL-Checker die Aufgaben sequentiell abarbeitet, ist Reihenfolge der Aufgaben relevant. Ein Beispiel-Template Datei ist 1.

1.3 Das Resetskript

Das Reset-Skript besteht aus einer Reihe vom SQL-Statements. Der SQL-Checker führt dieses Skript vor jedem Test aus. Studenten können das Resetskript in der SQLChecker-GUI ausführen. Es sollte darauf geachtet werden,

```
/*submission_name*/
Blatt1
/*1a*/
/* Kommentar zu Aufgabe 1a
ueber mehrere Zeilen */
CREATE...
/*1b*/
-- Kommentar zu Aufgabe 1b
INSERT ...
```

Listing 1: Template-Beispiel

dass kein Statement Fehler verursacht. Zum Beispiel sollte `DROP DATABASE IF EXISTS <name>;`, statt `DROP DATABASE <name>;` genutzt werden, damit kein Fehler auftritt, wenn noch keine entsprechende Datenbank existiert. Damit der SQL-Checker das Skript bei der Initialisierung der Übung automatisch findet, muss der Name der Datei folgendermaßen aussehen: *<submission_name>*`_reset.sql`. `submission_name` wird in der Template-Datei definiert.

1.4 Die Lösung

Die Lösung ist eine weitere Tag-annotierte Datei. Die Lösung muss den selben `submission_name` und die gleichen Aufgaben-Tags in gleicher Reihenfolge definieren wie das dazugehörige Template. Der SQLChecker bewertet auf der Basis dieser Datei die Studentenabgaben. Der Code im Rumpf eines Aufgaben-Tags sollte eine gültige Lösung der entsprechenden Aufgabe sein. Das Programm überprüft, ob die von Studenten erzeugten Statements die gleichen Resultate erzielen, wie diese Statements der Lösung. Zusätzlich kann diese Datei `static`-Tags definieren. Das SQLStatement im Rumpf dieser Tags wird sowohl bei der Lösung als auch der Abgabe ausgeführt und die Resultate verglichen. Ist das Ergebnis in beiden Fällen gleich, gilt der Test dieses Tags als bestanden.

1.5 Die Konfigurationsdatei

Die Konfigurationsdatei ist eine INI-Datei, welche die nötigen Informationen für die Auswertung der Abgaben enthält. Diese Datei wird nur für die Evaluation und nicht die Erstellung der Abgabe benötigt. Die Schlüssel `database`, `username`, `password`, `hostname` und `port` werden für die Datenbankverbin-

```
/*submission_name*/
Blatt1
/*1a*/
CREATE TABLE tische (name varchar(40), beine int(6));
/*1b*/
INSERT INTO tische VALUES('KNARREVIK', 4);
/*static*/
SELECT COUNT(*) FROM tische;
```

Listing 2: Lösung-Beispiel

```
1  #settings for database connection
2  [db]
3  \database = airport
4  username = airportuser
5  password = airportuser
6  hostname = localhost
7  port = 3306
8
9  #pathes to files
10 [files]
11 resetPath = ~/Dropbox/SQLChecker/aufgaben/Blatt1/Blatt1_reset.sql
12 solutionPaths =
   ↪  ~/Dropbox/SQLChecker/aufgaben/Blatt1/Blatt1_solution.sql,
   ↪  ~/Dropbox/SQLChecker/aufgaben/Blatt1/Blatt1_solution2.sql
13 submissionPath = ~/Dropbox/SQLChecker/aufgaben/Blatt1/submissions/
```

Listing 3: Konfigurations-Beispiel

dung benötigt. `resetPath` gibt die Pfad zum Resetskript an. `solutionPaths` gibt den Pfad zu einer oder mehreren Lösungs-Dateien an. Mehrere Lösungen werden mit Komma separiert. `submissionPath` gibt den Pfad zu Abgaben an. Der Pfad wird bis zu einer Tiefe von zwei nach Abgaben durchsucht. 3 ist ein Beispiel einer Konfigurationsdatei.

2 Die Auswertung

Die Ausführung läuft ebenfalls über die SQLChecker Jar Datei. Das Programm wird über das Terminal gestartet und benötigt die korrekten Argumente.

2.1 Relevante Argumente

`-c,--config <Path>` Der Pfad zur .ini Datei. Muss gesetzt werden.

`-csv,--csv <Path>` Wenn angegeben, wird am Ende das Ergebnis in Form von CSV dargestellt. Bei Angabe eines Pfads, wird versucht das Resultat an diesen Ort zu schreiben. Ohne Angabe der Datei, wird das Ergebnis auf dem Terminal ausgegeben.

`-e,--evaluate` Das Argument, um die Evaluation zu starten. Ohne dieses wird die GUI ausgeführt.

`-onlyBest,--onlyBest` Wenn gesetzt, wird pro Student nur das Ergebnis mit der höchsten Punktzahl vermerkt. Es macht nur Sinn bei Angabe mehrerer Solution-Files.

`-v,--verbose` Verbose-Mode, zusätzliche Information wird ausgegeben.

2.2 Beispiel-Aufruf

Der folgende Aufruf startet den Evaluationprozess mit der Config am Ort `/simple/config.ini` im csv-Mode.

```
1 java -jar SQLChecker-1.0.2.jar -c ~/simple/config.ini -csv -e
```

Die nachfolgende Ausgabe wird erzeugt:

```
1 Submission loaded:
  ↪ /home/xyntek/simple/submission/sample_with_author.sql
2 Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new
  ↪ driver class is `com.mysql.cj.jdbc.Driver'. The driver is
  ↪ automatically registered via the SPI and manual loading of the
  ↪ driver class is generally unnecessary.
3 SUCCESS Path:/home/xyntek/simple/submission/sample_with_author.sql
  ↪ Authors:[Max Mustermann max_mustermann@gmail.de 5727685, Nadine
  ↪ Mustermann nadine_mustermann@gmail.de 1234567]
  ↪ Solution:sample_with_author Evaluation:[1a:pass, 1b:pass,
  ↪ 1c:pass, 1d:pass, 2a:pass, 2b:pass]
4 SUCCESS Path:/home/xyntek/simple/submission/sample_with_author.sql
  ↪ Authors:[Max Mustermann max_mustermann@gmail.de 5727685, Nadine
  ↪ Mustermann nadine_mustermann@gmail.de 1234567]
  ↪ Solution:sample2_with_author Evaluation:[1a:pass, 1b:pass,
  ↪ 1c:pass, 1d:pass, 2a:pass, 2b:fail]
5 "Path", "Authors", "Solution", "1a", "1b", "1c", "1d", "2a", "2b",
  ↪ "#Success", "ErrorMsg"
```

```
6 "sample_with_author.sql", "[Max Mustermann max_mustermann@gmail.de
  ↪ 5727685, Nadine Mustermann nadine_mustermann@gmail.de 1234567]",
  ↪ "sample_with_author", "pass", "pass", "pass", "pass", "pass",
  ↪ "pass", "6", ""
7 "sample_with_author.sql", "[Max Mustermann max_mustermann@gmail.de
  ↪ 5727685, Nadine Mustermann nadine_mustermann@gmail.de 1234567]",
  ↪ "sample2_with_author", "pass", "pass", "pass", "pass", "pass",
  ↪ "fail", "5", ""
```

Zeile 1 gibt an welche Solution geladen wurden. Zeile 2 weißt darauf hin, dass dbfit veraltete Software verwendet. Die Zeilen 3 und 4 geben an, dass jeweils eine Abgabe erfolgreich überprüft werden konnte. Zeile 5 bis 7 sind das Ergebnis-CSV.