Frankfurt Big Data Lab | Database & Information Systems

SQLChecker

User Documentation

Version 1.0
5-8-2017

Contents

1. Overview of SQLChecker

The SQLChecker[1] is an open source Java program developed to automatically check SQL (Structured Query Language) homework submissions against predefined solution and generate a summary report as a result. It uses internally the DbFit2 testing framework and MySQL or MariaDB as database.
The SQLChecker uses three types of files: ***raw.sql***, ***solution.txt*** and ***submission*** files. The ***raw.sql*** is used to generate the ***solution.txt*** file, which is then used by the SQLchecker for evaluating and correcting the submitted ***submission*** files (text files with strict syntax). After executing the SQLchecker to correct the submission, multiple files are generated (summary_[TIMESTAMP].csv, PlagiatReport_[TIMESTAMP].csv, mistages_[TIMESTAMP].log and CommentReport_[TIMESTAMP].csv). They show the results of the automated correction process.

2. Importing the VM

For using the VM and checking the submissions a few steps have to be done:

1. Install VirtualBox on your system.
   Get the corresponding version for you OS here: https://www.virtualbox.org/wiki/Downloads
2. Restart your system.
3. Download and unpack the db1 VM.
   Link to download VM:
   https://drive.google.com/file/d/0B0tHFST2iVzMMEpSZG0yYUU4NkU/view?usp=sharing
4. Import the VM in VirtualBox. There are two options to do that:
   1. Open VirtualBox → Machine → Add → choose the db1.vbox file
   2. Open VirtualBox → double click on db1.vbox file
5. Before starting the VM you can go to Settings and adjust the Base Memory and CPU if you want to use more resources for faster execution.
6. Start the VM db1 in VirtualBox.
7. Log in with the password: ***start***

3. Accessing MySQL

There are two ways to access the MySQL database, which is already installed in the VM:

- You can open the LXTerminal (for example using the Desktop icon) and then directly type:

$mysql –u root –p
The root user password is ***start***

- Install MySQL Workbench by executing the command in the LXTerminal:

$sudo apt-get install mysql-workbench
Then you can start the MySQL Workbench from the start menu. Once it is started you can click on the + sign near the MySQL Connections and enter the connection credentials. You should just add Connection Name and Password (***start***). Then you can use MySQL Workbench to manage your databases. sudo apt-get install mysql-workbench

[1] https://github.com/mxhdev/SQLChecker
[2] https://github.com/dbfit/dbfit

4. Generating a Solution File

4.1. Writing a raw.sql file

At first the ***raw.sql*** file has to be written. This file consist of all solutions of the exercises.

Before each submission of a student is checked a reset script is executed. The reset script could contain certain tables, data, constraints, procedures and transaction. All this can be used in the submissions and the solution file.

Generally there are three different tags, which can be used: the exercise tag e.g.: `/*1a*/`, the static tag: `/*static*/` and the static-error tag: `/*static.error*/`. The static tags are used to check statements which don't produce any output on their execution (like: create/alter a procedure).

An exercise tag is used like this, the exercise here is a simple SELECT statement:

```
/*1a*/
SELECT
     Name, Ort
FROM
     mitarbeiter
WHERE
 Ort = 'Frankfurt';
```

Also it can be possible that the students can write /*static*/ tags. For example if they have to write multiple INSERT statements. In the following exercise the students should insert a new value in the table products (**this is how it should look like in the submission, in the raw.sql file this isn't needed!**):

```
/*static*/
INSERT INTO mitarbeiter
(Geschlecht, Name, GebDat, Strasse, Hausnummer, Ort, PLZ)
VALUES
('m',  'Albert  Zweistein',  '1879-03-14',  'Some  Street',  1,
'Frankfurt', 12345);
```

By using a static tag in the solution file, one can check, if the student has correctly added the required records to the corresponding table:

```
/*static*/
SELECT count(*) FROM mitarbeiter;
```

In the same way functions, procedures or views can be created and checked.

```
/*2a*/
CREATE FUNCTION warenwert(bestand int, preis decimal(8,2)) RETURNS
decimal(20,2)
BEGIN
  declare gesamt decimal(20,2);
  if bestand <= 0 then set gesamt = 0;
  elseif preis <= 0 then set gesamt = 0;
```

```
  else
    set gesamt = bestand * preis;
  end if;
RETURN gesamt;
END
```

By using static tags, one can also check in the solution file, if the student submission creates the function correctly. (Note: Only function and procedure calls can be called multiple times in one static tag. The only condition here is, that the functions/procedures called in one static tags are all using the same function or procedure)

```
/*static*/
warenwert(2, -2.0)
warenwert(2, 2.0)
warenwert(0, 1)
warenwert(1, 0)
warenwert(0, 0)
warenwert(10, 1.45)
```

The most important difference between functions and procedures here is the fact that that procedures can take multiple IN and OUT as well as INOUT parameters.

```
/*2b*/
CREATE PROCEDURE CalcLength(IN name varchar(100), OUT strlength
int)
     set strlength = length(name);
```

The check of procedures is the same as for functions, you only have to add a parameter for the output of the procedure (e.g. @strlength):

```
/*static*/
CalcLength('abc', @strlength)
CalcLength('FiveLetters', @strlength)
```

And for a procedure with an INOUT parameter it looks like this:

```
/*2c*/
CREATE PROCEDURE PlusEins(INOUT val int)
     set val = val + 1;
```

And the static tag to check the submissions:

```
/*static*/
PlusEins(41)
PlusEins(887)
```

By using the static.error tag of SQLChecker, one can also check, if students have created the correct constraints.

```
/*3a*/
-- Adding the constraint
ALTER TABLE krankenhaus.pfleger
ADD CONSTRAINT pfleger_mitarbeiter
  FOREIGN KEY (PNR)
  REFERENCES krankenhaus.mitarbeiter (PNR)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
ADD CONSTRAINT pfleger_station
  FOREIGN KEY (StationID)
  REFERENCES krankenhaus.station (StationID)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

/*static.error*/

/* failing pnr */

INSERT INTO pfleger
(PNR, StationID)
VALUES
(-99, -2);
```

This statement should fail, when the student created the correct constraint in an exercise tag (The static.error tag should be used AFTER the corresponding exercise tag!).

4.2. Generating the final solution file

Now that we have the ***raw.sql*** file set up, we can generate the ***solution.txt*** file. The following steps will create this file:

1. Create a properties file with the following values (values of tutorial.properties under ~/DB1/conf/):
   - Path to reset script (db.resetscript=data/tutorial/reset.sql)
   - Database user name (gen.db.user=root)
   - Password of the user (gen.db.pw=start)
   - Used host (gen.db.host=localhost)
   - Used database (gen.db.name=krankenhaus)
   - Path to raw.sql file (gen.infile=data/tutorial/raw.sql)
   - The path at which the output solution file should reside (gen.outfile=data/tutorial/solution.txt)
   - The path at which the sample submission file should reside (gen.samplefile=data/tutorial/ftsample.sql)
2. Open the terminal on the desktop.
3. Change directory to Schreibtisch/DB1: `$cd Schreibtisch/DB1`
4. Start generating the files with: `java -jar sqlchecker.jar gen conf/tutorial.properties`

5. Writing a submission

There are three different types of tags you can use in your submission: /*authors*/, exercise tag e.g. /*1a*/ and the static tag: /*static*/. The static tag can only be used, if this is mentioned in your exercise!

Define the author(s) of the submission by using the /*authors*/ tag. Your name and the student id are separated by a ';':

```
/*authors*/
Max Mustermann;5234233
```

An exercise is answered this way, for example a simple SELECT statement:

```
/*1a*/

SELECT
     Name, Ort
FROM
     mitarbeiter
WHERE
     Ort = 'Frankfurt';
```

Attention: You don't need to write something like 'USE databaseXYZ;' the application will take care of that! Also only **ONE** statement for each tag is accepted!

In the same manner FUNCTIONs, PROCEDUREs, VIEWs or CONSTRAINTs can be created:

```
/*2a*/

CREATE FUNCTION warenwert(bestand int, preis decimal(8,2)) RETURNS
decimal(20,2)
BEGIN
declare gesamt decimal(20,2);  if bestand <= 0 then set gesamt = 0;
elseif preis <= 0 then set gesamt = 0;
else
set gesamt = bestand * preis;
end if;
RETURN gesamt;
END



/*2b*/

CREATE PROCEDURE CalcLength(IN name varchar(100), OUT strlength
int)
set strlength = length(name);
```

```
/*2c*/

CREATE PROCEDURE PlusEins(INOUT val int)
set val = val + 1;



/*3a*/

-- Adding the constraint to an existing table
ALTER TABLE krankenhaus.pfleger
ADD CONSTRAINT pfleger_mitarbeiter
FOREIGN KEY (PNR)
REFERENCES krankenhaus.mitarbeiter (PNR)
ON DELETE NO ACTION
ON UPDATE NO ACTION,
ADD CONSTRAINT pfleger_station
FOREIGN KEY (StationID)
REFERENCES krankenhaus.station (StationID)
ON DELETE NO ACTION
ON UPDATE NO ACTION;
```

An exception are the static tags. These can be useful for executing statements, which change the data somehow (e.g. CREATE/INSERT).
It will be mentioned in the exercise, if you should use static tags!

6. Checking your submissions

To check if you wrote a correct submission just do the following:

1. Put your submission in the folder: Schreibtisch/DB1/data/tutorial/submissions
2. Open the terminal on the desktop.
3. Change directory to Schreibtisch/DB1: $`cd Schreibtisch/DB1`
4. Execute the command: `java -jar sqlchecker.jar exec conf/tutorial.properties`
5. Check in the new 'summary_[TIMESTAMP].csv' file if everything worked well.