Documentation for GenerateTable.py & JSON file Cem Iskir

This document is mostly to process a JSON file which should come from the server side when a client wants to create a table.

If we first start with the format of JSON file:

**Field name:** operation

**Example:** Create, update (functionality is not yet supported in the current version of GenerateTable.py) (*STRING*)

**Explanation:** Type of operation. This is for the whole tables. This can very easily be changed by putting this field inside of each table. Small changes should be made in GenerateTable.py in order to do so

**Field name:** table

**Example:** [{

**"name"**: **"myMaliciousta''=''ble2"**,  
 **"fields"**: [  
 {  
 **"name"**: **"username"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **false**,  
 **"unique"**: **true**,  
 **"null"**: **false**,  
 **"default"**: **null** }, …}] (*ARRAY*)

**Explanation:** Holds an array of tables

**Field name:** table[name]

**Example:** Users *(STRING)*

**Explanation:** Name of the table. Only 0-9 a-z A-Z allowed

**Field name:** table[fields]

**Example:** [{  
 **"name"**: **"username"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **false**,  
 **"unique"**: **true**,  
 **"null"**: **false**,  
 **"default"**: **null** },…] *(ARRAY)*

**Explanation:** Holds the fields of the table

**Field name:** table[fields][name]

**Example:** username (*STRING*)

**Explanation:** Name of the column in the database. Only 0-9 a-z A-Z allowed.

**Field name:** table[fields][type]

**Example:** TEXT, BOOLEAN, REAL, INTEGER (*STRING*)

**Explanation:** Type of the column.

**Field name:** table[fields][primary]

**Example:** true, false (*BOOLEAN*)

**Explanation:** If the column is primary or not.

**Field name:** table[fields][unique]

**Example:** true, false (*BOOLEAN*)

**Explanation:** If the column is unique or not.

**Field name:** table[fields][null]

**Example:** true, false (*BOOLEAN*)

**Explanation:** If the null value is allowed for the specific column or not.

**Field name:** table[fields][default]

**Example:** null, 5, “enrolled” (*STRING, NULL, INTEGER*)

**Explanation:** Default value for that column. It can be null if no default value is wanted.

**Field name:** table[foreign-fields]

**Example:** [{  
 **"foreign-table"**:**"myMalicioustable2"**,  
 **"relation-model"**:**"M-to-1"**},… ]

(*ARRAY*)

**Explanation:** Holds the functional dependencies give by user

**Field name:** table[foreign-fields][foreign-table]

**Example:** username (*STRING*)

**Explanation:** Name of the table which we define the relationship

**Field name:** table[foreign-fields][relation-model]

**Example:** M-to-M M-to-1 (*STRING*)

**Explanation:** Type of the relation. If it is many to one, or many to many

An example could be:

{  
 **"operation"**: **"create"**,  
 **"table"**: [  
 {  
 **"name"**: **"myMaliciousta''=''ble2"**,  
 **"fields"**: [  
 {  
 **"name"**: **"username"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **false**,  
 **"unique"**: **true**,  
 **"null"**: **false**,  
 **"default"**: **null** },  
 {  
 **"name"**: **"password"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **true**,  
 **"unique"**: **false**,  
 **"null"**: **false**,  
 **"default"**: **null** },  
 {  
 **"name"**: **"site"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **false**,  
 **"unique"**: **false**,  
 **"null"**: **false**,  
 **"default"**: **"localhost"** }  
 ],  
 **"foreign-fields"**: [  
 {  
 **"foreign-table"**:**"myMalicioustable2"**,  
 **"relation-model"**:**"M-to-1"** },  
 {  
 **"foreign-table"**:**"myMalicioustable3"**,  
 **"relation-model"**:**"M-to-M"** }  
 ]  
 },  
 {  
 **"name"**: **"myMaliciousta''=''ble4"**,  
 **"fields"**: [  
 {  
 **"name"**: **"username"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **true**,  
 **"unique"**: **true**,  
 **"null"**: **false**,  
 **"default"**: **null** },  
 {  
 **"name"**: **"password"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **false**,  
 **"unique"**: **false**,  
 **"null"**: **false**,  
 **"default"**: **null** },  
 {  
 **"name"**: **"site"**,  
 **"type"**: **"TEXT"**,  
 **"primary"**: **false**,  
 **"unique"**: **false**,  
 **"null"**: **false**,  
 **"default"**: **"localhost"** }  
 ]  
 }  
 ]  
}

**SOME NOTES:** It is expected that this file is created by our web-site. There was no SQL injection library for CREATE TABLE statement; thus SQL injection is avoided manually as much as possible. Aggressive injection tests and some improvements based on those tests might be required before website is live.

GenerateTable.py functions

1. sanitizeMe(string)

Returns the sanitized version of the given string. Doesn’t allow anything other than a-zA-Z0-9

1. sanitizeNumber(stringOrNumber)

Same with sanitizeMe() only difference is it also allows “.”

1. sanitize(givenstr, regex)

Used by sanitizeMe and sanitizeNumber to increase code reusability.

1. checkIfExistedInRealTime(tablename,shouldBeSanitized)

Checks if a given tablename is existed in database by executing command “pragma table\_info(tablename)”

Note that this command is specific to SQLite.

This function returns primary key of that table and the type of the primary key as a list. It returns an empty list if the table is not existed

1. getAllTableNames(username,delimiter)

This function returns all tables directly created by the user. It queries the table\_username table. It doesn’t return intermediate tables (tables starting with inter\_) created by the result of many-to-many relationship. Note that intermediate tables don’t have primary keys. It returns empty string if there is no table for that user. delimiter is the string that will be used in the return value which will separate table names from each other.

1. returnPrimary(username,tableName,data)

This function checks if tableName is existed in database or in the current JSON file (if this is the case it hasn’t been yet created, but should be created soon.)

It returns:

-if existed in database:

Primary key and primary key column type

-if existed in JSON file

If primary key is selected by user, its type and name is returned as a list. Else default primary key is returned as [‘ID’,’INTEGER’]

**NOTE:** To improve performance, we first check the JSON file and reduce the probability to connect to database. However, it is assumed that table creation from JSON file will be successful.

1. processJSON(username,dbname,jsonfilename)

This is the main method which handles the given JSON file jsonfilename. username is the username of the client who makes the request. This parameter is important because all table creation is dependent on it. dbname is the name of db from where the tables are created.

7.1- connect to database and get the JSON file data

7.2- create requested tables by reading the JSON file

We go through each record in the JSON file. While going through a statement is constructed to create tables. This statement is hold in the variable *statement*.

Some of the fields aren’t checked when the statement is constructed.

For instance if a column is primary, we don’t need to check the fields null, default and unique.

If null is true, we don’t need to check uniqueness, etc.

The table’s names are constructed as it is below:

USERNAME\_TABLENAME

Username won’t be displayed to the user; instead they will see the table name as TABLENAME. TABLENAME is checked against malicious characters are removed from the table and column (field) names

As indicated, if a primary key isn’t specified, a column named ID having the type INTEGER is created.

Later functional dependencies are added to the statement. Since SQLite doesn’t support adding functional dependencies separately, they must be specified while creating the table.

At the end, we insert the table named tables\_USERNAME. This table will hold all user tables, including intermediate tables created for many-to-many relation.

ALL OF THESE TABLE WILL ONLY BE CREATED WHEN THEY AREN’T ALREADY EXISTED IN DB.

This file also adds name and primary key of all tables it inserts.