smartOBD

Release 0.0.1

Will Walker

Dec 10, 2019

CONTENTS:

smartOBD is a python module that uses ELM327 OBD-II adapters to write data about a vehicle to a database, either in real-time using *asynco*, or in aggregate using `test_commands`.

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER TWO

INTERFACE (MAIN FUNCTION)

Initialization and interface Simple command line interface, with choices for asynchronous data and a full data query

`smartOBD.main.`**`main`**`()`

This function determines which functionality the user would like to use, and calls it

CHAPTER

THREE

ADDING A NEW CAR

`smartOBD.new_car.`**`new_car`**`()`

Creates new car in database based on username. Collects make, model, model year for car and adds it to the cars table.

Also creates new car and car_temp table for *`smartOBD.test_commands.fullQuery()`* and *`smartOBD.asynco.getAsync()`* respectively.

CHAPTER FOUR

ASYNCHRONOUS CONNECTIONS

Reads data using async functions and writes to a single row of the database to be read by the website

`smartOBD.asynco.`**`getAsync`**`(`*dur*`)`

sets connection for async fucntions Starts connection and waits for key press to stop connection

`smartOBD.asynco.`**`userGet`**`()`

This function gets the user and write the dbtable

Parameters

- **dbconn** (*psycopg2 database connection*) – The database connection class.
- **cur** (*psycopg2 database cursor*) – The cursor from the database.

Returns name of car table (str).

Writes dbtable name to global variable dbtable

`smartOBD.asynco.`**`writeToDB`**`()`

Writes to database Erases data from database and writes new values to be read by the website

Inputs

- Username (str) – username in database
- Car make/model (str, str) – make and model of car desired if user has more than one car in the database

CHAPTER

FIVE

FULL QUERY

Runs every compatible command to query as much data as possible from vehicle and writes to a new row in the database

smartOBD.test_commands.**fullQuery**()

Gets dbtable name, then attempts connection with car. After connection is established, all commands are queried, and the successful ones are written to the database

Parses through all OBDCommands as a dictionary, and queries the car with all commands, appends results to a data array, checks database for all columns and appends new ones, finally, writes to database .. code-block:

```
# dictionary generation
for key, i in test_dict.items():
    # print(key, test_dict[key])
    command.append((key, test_dict[key]))

#basic loop for running commands from dictionary
for i in range(0, len(temp2)):
res = str((car.query(temp2[i])).value)
description = str(temp2[i])
if(res != 'None'):
    columns.append(description.rsplit(': ', 1)[1])
    results.append(str(res).rsplit(' ', 1)[0])
```

After running all queries, final column generation and insertion .. code-block:

```
# * length checking for all arrays
if(len(columns) != len(results)):
    print("Results error")
# *final loop for database access
else:
    print("Parsing success")
    print(len(columns),"=",len(results))
    # * checking all columns for existence
    for i in range(1, len(columns)):
        data = columns[i]
        data = data.replace("'", " ")
        data = data.replace("\"", " ")
        cur.execute("select exists(select 1 from information_schema.columns where␣
↪table_name='%s' and column_name='%s');",
                    (AsIs(dbtable), AsIs(data)))
        test = cur.fetchone()[0]
        if(not test):
            data.replace("'", " ")
            data.replace("\"", " ")
```

(continues on next page)

(continued from previous page)

```
            cur.execute("alter table %s add column \"%s\" VARCHAR(2000)",
                        (AsIs(dbtable), AsIs(data)))
            print("TABLE ALTERED",data)
    # * final insertion
    dbconn.commit()
    q1 = sql.SQL("insert into {0} values ({1})").format(sql.Identifier(dbtable),
                                                         sql.SQL(', ').join(sql.
↪Placeholder() * len(results)))
    # print(results)
    cur.execute(q1, results)
    dbconn.commit()
    print("Successful Read")
```

Runs every compatible command to query as much data as possible from vehicle and writes to a new row in the database

`smartOBD.test_commands.`**`userGet`**(*dbconn*, *cur*)

This function gets the user and write the dbtable

Parameters

- **dbconn** (*psycopg2 database connection*) – The database connection class.
- **cur** (*psycopg2 database cursor*) – The cursor from the database.

Returns name of car table (str).

Writes dbtable name to global variable dbtable

PYTHON MODULE INDEX

INDEX