SCRIPT EVALUATION ASSISTANT

A PROJECT REPORT

Submitted by

N.AJAY RAJ	316126510034
M.RAGHU	316126510031
N.AMRUTHA LAKSHMI	316126510032
M.SUSHMA	316126510027

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Affiliated to Andhra University)

SANGIVALASA, VISAKHAPATNAM - 531162

2018-2019

ABSTRACT

It is an usual practice in Engineering colleges to award marks to a student on the basis on his weekly performance. However, most of these marks are awarded for the observation and record that a student maintains. In CSE and IT departments, the lab tasks are usually programs and should be awarded based on their functionality. This is difficult for a faculty to manually go through all the students' programs and award them. Also, the evaluation is done and stored on paper, thereby reducing the accessibility of it relative to the current world standards.

Our project puts forward a system wherein the faculty members can execute, evaluate and generate spreadsheets on the students' performance.

Table of Contents

List of Figures

1. Introduction

Our project titled “Script Evaluation Assistant”, aims at assisting the faculty members and instructors to run each and every program of the student with various inputs and view the outputs. Then they can graded accordingly.

In the existing system, these lab programs are not executed for weekly basis. Instead the observations are scrutinised. It is quite difficult for the faculty member to go through each and every student’s program and understand his implementation. Also, this paper based approach doesn’t allow the code to be tested.

The marks are also recorded in books. In the modern world where the everything is accessible, having something still only on paper shows that the system is outdated. We need it to be in electronic format so that it is easier to share and grant access to other required members.

The current system also lacks to maintain a record of all the programs a student has submitted. Our project will be used alongside Google Drive’s Backup and Sync client to maintain a file system with individual directories for every student. This is lot easier because the students can submit their files using the Google Drive web or desktop client.

The end users is clear in this case, faculty and students. Ideal for weekly assessment of students’ performance in the lab. This can also be used for grading of students’ assignment in each semester. On the other hand, this can used for conducting coding events where the complexities are humongous and cannot be held on online coding platforms. So this can also be used by companies and job applicants.

2. Design

ER - Diagram

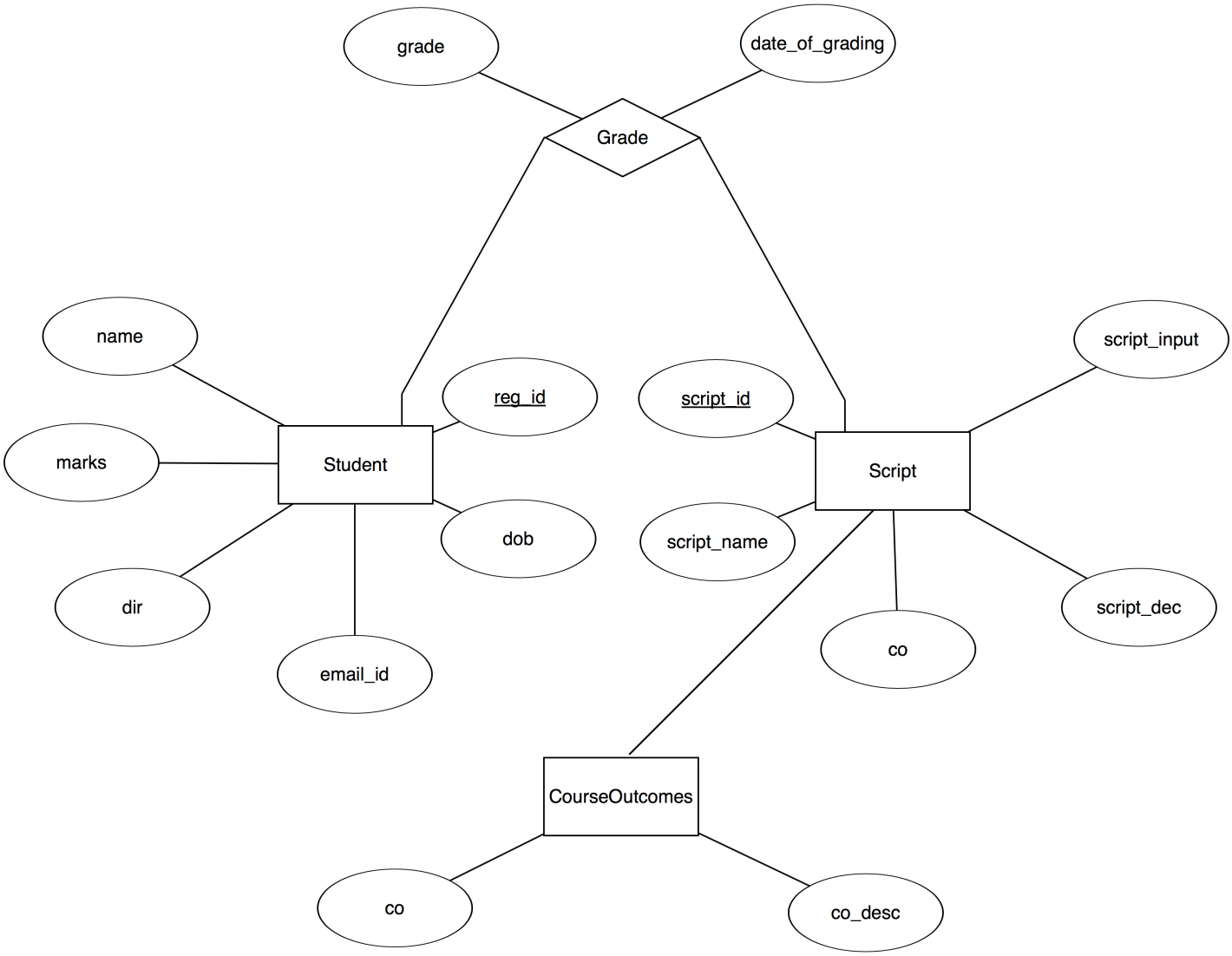


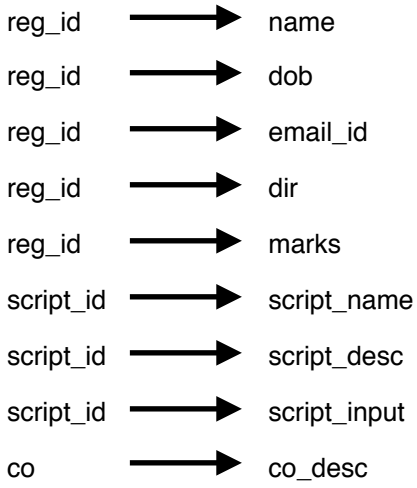
Fig 2.1: ER Diagram for Script Evaluation Assistant

Normalisation

The normalisation followed here is Boyce Codd Normal Form (BCNF). The database seen in the figure 2.1 is a very simple one. All the functional dependencies are such that it satisfies BCNF because for every dependency X —> a, X is a super key.

Functional Dependencies

The functional dependencies observed in the above database are:



There exists no more dependencies in the table. Since all the dependants are of primary keys, it is in Boyce Codd Normal Form. The database itself is fairly simple for any redundancy in data to occur.

Explanation

All the students' data will be present in the Student table, the program scripts' data in Script table and the Course Outcomes' data in CourseOutcomes table. Whenever a student is graded, he will be graded for a particular program script on a particular day. So we need four attributes, the student data, script data, grade and the date of grading. Hence we can form a relation between the student and script table with two more attributes for grade and date of grading. Every program done is mapped to a course outcome, hence we store the CO number along with the script data. This CO has a particular description which can be redundant if we mention it in the script table. Hence, we add a foreign key to CourseOutcome table and put the CO details in the new table.

Triggers are added on the Student table to check for valid inputs. A function is added to compute the average marks of a student. This function will be called by a trigger written on the grade table. Two triggers are present. One is when we insert new data into the grade table and the other trigger is invoked when we update the grade table. This might happen when a student produces a better program and his marks are increased.

3. System Requirements

Hardware

- Intel Core i3 or above
- 2 GB of RAM
- 5 GB of HDD / SSD (helper softwares included)
- Resolution of 1366 x 768 pixels
- LAN / WLAN card

Software

- PC with the required Runtime Environment
- Python 3.6.2 or higher
- MySQL 8.0 or higher
- PyMySQL API for Python - Database Integration
- Google Drive’s Backup & Sync Client
- Linux / MAC (change file paths for windows)
- Internet Access

4. Implementation

```
database = pymysql.connect('localhost', 'root', [redacted])
database.autocommit(True)
db = database.cursor()

def initialise_database():
    '''
    :return: None
    Pareses CreateDatabase.sql and Operations.sql to fetch SQL statements and
    creates database
    creates tables
    adds constraints
    creates triggers and funcs
    '''
    db.execute("SHOW DATABASES")
    databases = db.fetchall()
    if ('SEA',) in databases:
        db.execute('USE SEA')
        return

    try:
        for file in ('CreateDatabase.sql', 'Operations.sql'):
            with open(file, 'r') as queries_file:
                queries = queries_file.read()
                query = queries.split('--')
                for i in range(0, len(query), 2):
                    new_query = query[i]
                    new_query = new_query.replace('\n', ' ')
                    new_query = new_query.replace('\t', ' ')
                    if new_query == '': continue
                    db.execute(new_query)
    except:
        db.execute('DROP DATABASE SEA')
```

Fig 4.1: API for Script Evaluation Assistant

The snapshot show in figure 4.1 is the first set of lines to be executed apart from the GUI. It is responsible for parsing “CreateDatabase.sql” and “Operations.sql”. Then all the queries from the file are retrieved and executed. This is a necessary function because whenever the application is installed on a new system or when the database doesn’t exist, it will be created automatically. All the functions on the backend and properly commented and can be viewed using <function name>.__doc__ .

```
def execute(script, path, input_data):
    '''
    :param script: A string with the script to be executed
    :param path: A string og the path in which the script is present
    :param input_data: A string of the data to be passed as input to the script
    :return: 0 if exec successfully, 1 if not.

    Executes the given script as a seperate subprocess and stores its output in a file
    '''
    file = open(path + '/op.txt', 'r+')
    file.truncate(0)
    file.close()

    with open(path + '/op.txt', 'w') as output_file:
        start = time.time()
        status = subprocess.run(['python3', script],
                                cwd=path,
                                input=input_data,
                                encoding='ascii',
                                stdout=output_file)
        stop = time.time()
        output_file.write('Time Taken: ' + str(stop - start) + '\n\n')
    return status.returncode
```

Fig 4.2: Executing Program Scripts

The execute() function as shown in fig 4.2 will create a subprocess and run the student's program script in it. The output is written to a file and the function returns the return value of the subprocess. The file will run a script from the dir as passed in the path argument and gives the input data to it in the form of ASCII text. The output data from the file is read by the parent process and is displayed in the GUI window for the evaluation process.

```
def generate_spreadsheet():
    '''
    generate_spreadsheet()
    :return: None

    Generates a spreadsheet for given data
    '''
    script =  script_spinbox.get()
    grade = grade_spinbox.get()
    bound = grade_bound_spinbox.get()
    data = mysql.get_query_data(script, grade, bound)

    now = datetime.datetime.now()
    file_name = str(now.year) + '_' + str(now.month) + '_' + str(now.day) + '_'
    file_name += str(now.hour) + '_' + str(now.minute) + '_' + str(now.second)

    with open('/Users/ajayraj/Documents/ScriptEvaluator/' + file_name + '.csv', 'w') as sheet:
        writer = csv.writer(sheet, delimiter = ',')
        for row in data:
            writer.writerow(row)
```

Fig 4.3: Generating Spreadsheets

All the outputs generated from this application are to be used by the faculty. For this purpose, the application is written so as to generate Comma Separated Values(CSV) files as outputs. These files when clicked upon will be open by programs such as MS Excel or Numbers depending on the Operating System. Hence, can be modified according to the discretion of the faculty member.

```
def award_grade(reg_id, script_id, grade):
    '''
    :param roll_no: The roll no of the student
    :param script_id: The script to be awarded
    :param grade: The grade to be awarded
    :return: None
    '''
    check_query = "SELECT * FROM Grade WHERE reg_id = '%s' and script_id = '%s'" % (reg_id, script_id)
    if db.execute(check_query) == 1:
        query = "UPDATE Grade SET grade = %s WHERE reg_id = '%s' and script_id = '%s'" % (grade, reg_id, script_id)
    else:
        query = "INSERT INTO Grade VALUES ('%s', '%s', %s, curdate())" % (reg_id, script_id, grade)
    db.execute(query)
```

Fig 4.4: Queries

A sample of queries execution functions is show in fig 4.4. All the queries in the project is written like an API. The python file is just imported and the respective functions are called for executing the specific queries. Here we award the grade. If a particular grade record exists, we update it. Else we insert it into the grade table.

Modules

- Mysql: The database creation and the all the queries are written modularly in different functions in API like style format with in function documentation comments. One such function is show in fig 4.4. It is implemented as a separate file which is imported and the functions are called when required.
- Setup: This module is run after the mysql.py file. It takes CSV files as input and parses the data. The data thus fetched is put into the database by using INSERT statements in mysql.
- Runtime: The runtime module uses subprocess module of python to create a new subprocess and run the particular program file. This will return the status code of the program, i.e the return value. Additionally the outputs are written to a separate text file.
- Spreadsheets: The spreadsheets module will generate CSV files which contain the data from the queries that the user has selected. Since the Operating System detects the CSV file like a spreadsheet, it opens the spreadsheet with the default program providing ease of work.
- UI: The entire UI is implemented using Python's Tkinter module. It is fairly simple with the main window being the runtime tool while having two other sub-windows. These sub-windows are setup window and spreadsheets window.

5. Results

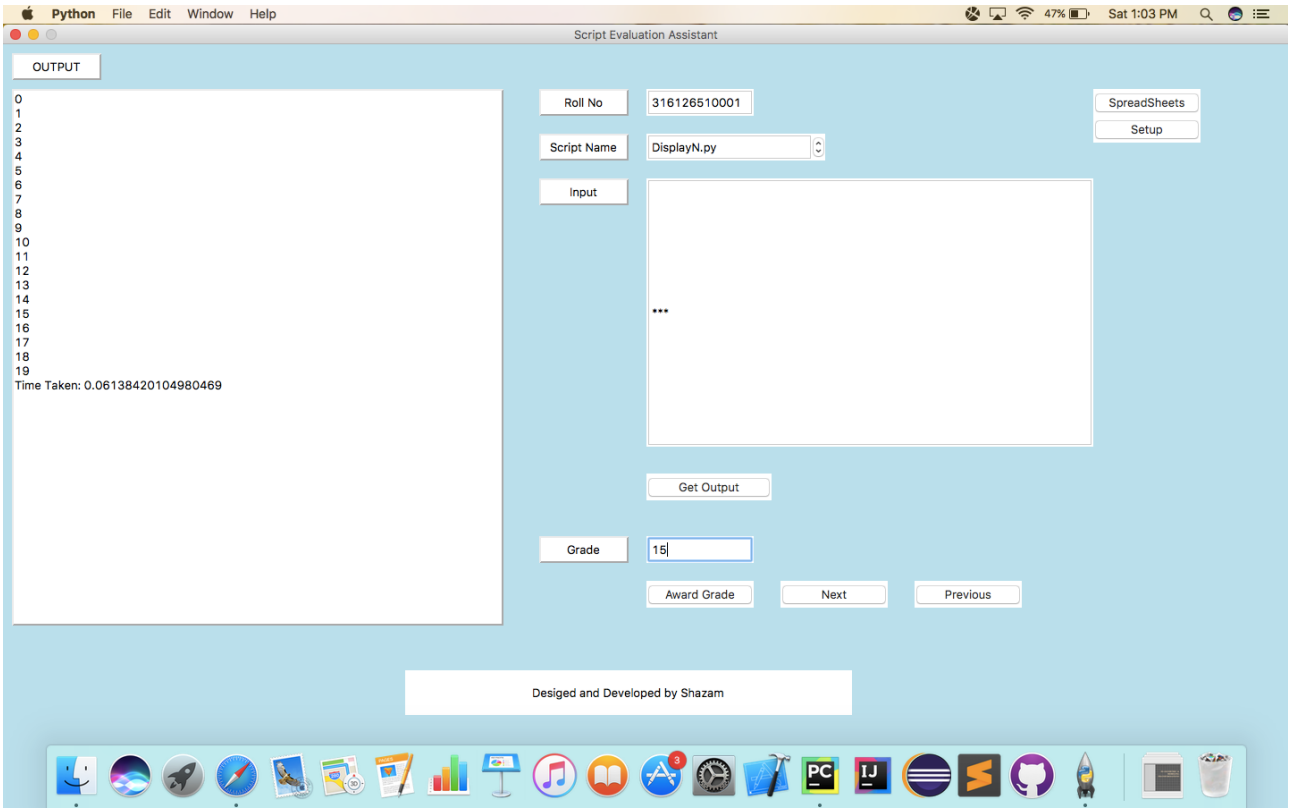


Fig 5.1: Main Window of Application

When the user opens the application, the main window appears as in fig 5.1. They will have to enter the roll no of the students and select the script. They can either specify their own input or enter "***" without quotes. The latter input will tell the application the fetch the default input data from the database. The get output button calls the execute function as shown in fig 4.2. This output is read from the file and displayed on the left side window along with the time taken to execute the program. Depending on the output and time, the user can grade the student and press the "Award Grade" button. When done so, the data is entered into the database by calling the function shown in fig 4.4. The "Next" and "Previous" buttons will increment and decrement the roll no respectively.

It should noted that the marks can be updated just by re awarding the marks and there is no hassle in doing so. The application automatically decides whether to insert or update the data. As and when the data is inserted or updated, the average marks of that student is computed and updated into the Student Table.

However, the data first has to be inserted into the database and it has to be done with ease. The solution to this is shown in fig 5.2.

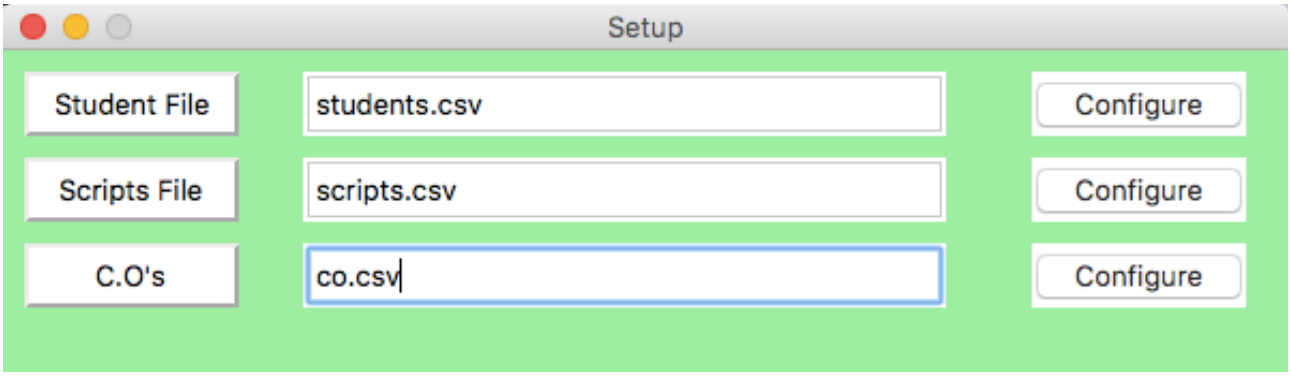


Fig 5.2: Setup Window

The setup window provides an interface to insert the required data. All the user has to do is type data into a spreadsheet or collect it from a Google Form and export it to CSV format. These exported files can be given as inputs and the application will automatically read this data and insert into the specific tables.

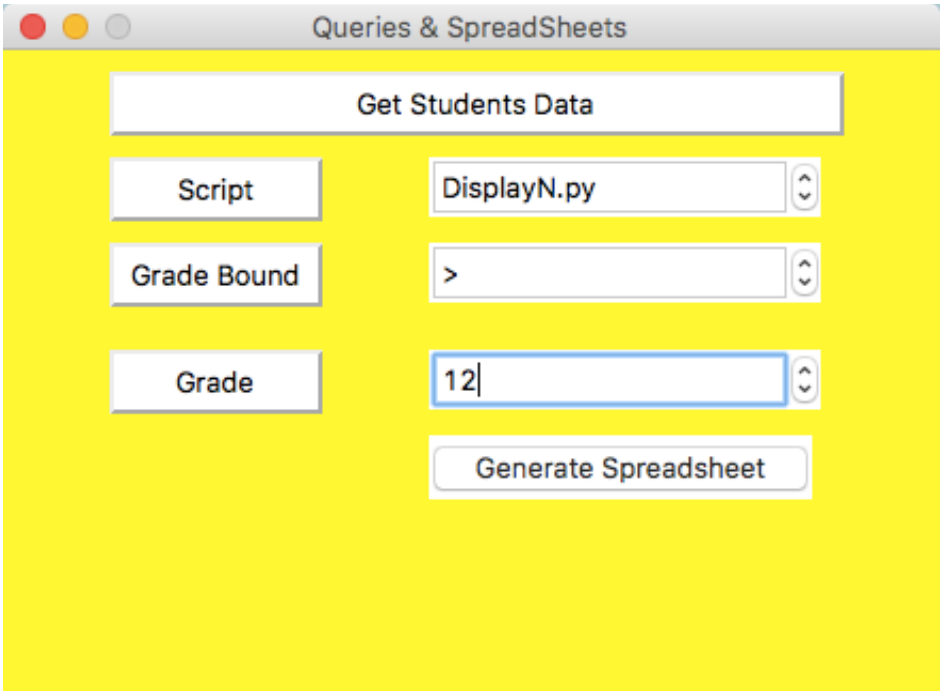


Fig 5.3: Queries & Spreadsheets Window

The window show in fig 5.3 is fairly simple. It translates user inputs to queries. Instead of just printing them on screen, a spreadsheet is generated in the same folder. The uses of this functionality is already mentioned multiple times above in this document. Specific data can be retrieved with ease and can be used for any purpose.

6. Conclusion and Future Scope

This application will save a considerable amount of time for the faculty members while assessing the students performance. Since, we support Open Source, we have open sourced our project at github.com/AjayRajNelapudi/Script-Evaluation-Assistant with a MIT license.

By implementing this project, we will have data in electronic format regarding students performance. This means sharing this data is quite easy when compared to data stored on paper. We can either share the generated spreadsheets or grant access to the database for another user.

We believe there is room for improvement. But to improve, either us or someone else who is taking up this project has to understand the code. We have given importance to code readability to make sure that it is really sensible to read the code. Modularity is maintained for easier understanding. Documentation comments are added to the functions so that developers can easily understand which function call has to be made when necessary.

This project will be taken forward from DBMS Lab to OST Lab. Since, this project is unique and provides a solution for effective grading and assessing the performance of the student, we decided to make it as a product by looking into all the possible details. Few of the new features to be added are a better UI for easier interaction, automating the grading using test-cases and removing the ‘Google Drive BackUp and Sync Client’ and replacing it with Google Drive API which is more sensible to store data in the cloud and access or modify it in need.

References

- Python Documentation
- PyMySQL Documentation
- MySQL Documentation
- Stack Overflow and other online forums