Birla Vidya Mandir, Nainital

****

**ACADEMIC YEAR: 2024-25**

**PROJECT REPORT ON**

**DRUG MANAGEMENT SYSTEM**

**FOLIO NO : 22111284 , 2238361**

**NAME : Chitransh Bhatnagar, Siddhant Pathak**

**CLASS : XII-A**

**SUBJECT : INFORMATICS PRACTICES**

**SUB CODE : 065**

**PROJECT GUIDE: MR. KIRTI KAMAL BHARDWAJ**

**PGT (CS)**

**BIRLA VIDYA MANDIR, NAINITAL**

**UTTARAKHAND**

**CERTIFICATE:**

This is to certify that Master **Chitransh Bhatnagar** & **Siddhant Pathak** have successfully completed the project Work entitled:

**Drug Management system** in the subject: Informatics Practices (065) laid down in the regulations of CBSE for the purpose of Practical Examination in Class XII to be held in Birla Vidya Mandir, Nainital on\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

(**Mr.Kirti Kamal Bharadwaj)**

**PGT Informatics Practices**

**Master IC**

**Signature of External Examiner:-**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **TABLE OF CONTENTS:** | | |
| **S. No.** | **Titles Of Content:** | **Pg. No.** |
| 01 | SPECIFICATIONS | **01** |
| 02 | CERTIFICATE | **02** |
| 03 | TABLE OF CONTENTS | **03** |
| 04 | ACKNOWLDGEMENT | **05** |
| 05 | INTRODUCTION | **07** |
| 06 | OBJECTIVES OF THE PROJECT | **08** |
| 07 | PROPOSED SYSTEM | **09** |
| 08 | SYSTEM DEVELOPMENT LIFE CYCLE (SDLC) | **10** |
| 09 | PHASES OF SYSTEM DEVELOPMENT LIFE CYCLE | **12** |
| 10 | BLOCK DIAGRAM | **25** |
| 11 | LOGO | **26** |
| 12 | FUNCTIONALITY SCREENSHOTS | **26** |
| 13 | REPORT AND DATA ANALYSIS SCREENSHOTS | **34** |
| 14 | INSTALLATION PROCEDURE | **43** |
| 15 | TESTING | **44** |
| 16 | HARDWARE AND SOFTWARE REQUIREMENTS | **47** |
| 17 | THANKS PAGE | **48** |
| 18 | SOURCE CODE attachment… | **49** onwards... |

**Total Pages**: 148

**ACKNOWLEDGEMENT:**

I would like to express my heartfelt gratitude to several individuals who have significantly contributed to the completion of my project for the subject "Informatics Practices."

First and foremost, I extend my sincere thanks to **Hon’ble Kirti Kamal Bhardwaj**, my subject teacher. Your unwavering support, guidance, and encouragement have been invaluable throughout this project. Your passion for teaching and your commitment to helping students understand the intricacies of informatics practices have inspired me to push my boundaries and excel in my work. The knowledge and skills I have gained under your tutelage will undoubtedly aid me in my future endeavors.

I would also like to thank **Hon’ble Anil Sharma**, the Principal of BVM School. Your leadership and vision have created an environment where students can thrive academically and personally. Your constant encouragement to explore new ideas and challenges has motivated me to delve deeper into my studies and pursue excellence. Your belief in my abilities has played a crucial role in the successful completion of this project.

Additionally, I am grateful to **Hon’ble. Rakesh Mulasi**, our Vice Principal, for his support and guidance. Your administrative insight and encouragement to participate in various activities have enriched my educational experience. You have fostered a culture of learning and innovation within the school, which has significantly influenced my approach to this project.

I would also like to acknowledge my classmates and friends for their collaboration and support. Our discussions and brainstorming sessions have not only made this project more enjoyable but have also enhanced my understanding of the subject matter. The camaraderie we shared during this process has made the experience both enriching and memorable.

Furthermore, I appreciate the resources and facilities provided by BVM School, which have been instrumental in the successful completion of this project. Access to the computer lab, library, and other academic resources has allowed me to conduct thorough research and enhance the quality of my work.

Finally, I would like to thank my family for their continuous support and encouragement. Their belief in my abilities has motivated me to strive for excellence and has been a source of strength throughout my academic journey.

In conclusion, this project would not have been possible without the collective support of all the individuals mentioned above. I am truly grateful for their contributions, and I hope to carry forward the knowledge and experiences I have gained into my future academic and professional endeavors.

Thank you all for being an integral part of this journey.

**PROJECT ON DRUG MANAGEMENT SYSTEM-**

**INTRODUCTION:**

This project is all about software for Drug Management System, It helps to have a full-fledged control over his/her employees. The project is divided into 2 sections to make the programme easy to understand.

It receives inputs from the user and stores them in the SQL Database and is also capable of reporting the same, through the help panda’s data anylasis functions are also made available to the end-use.

­­­

**OBJECTIVES OF THE PROJECT:**

The objective of this project is to let the students apply the programming knowledge into a real- world situation/problem and exposed the students how programming skills helps in developing a good software.

* Write programs utilizing modern software tools.
* Apply object oriented programming principles effectively when developing small to medium sized projects.
* Write effective procedural code to solve small to medium sized problems.
* Students will demonstrate a breadth of knowledge in computer science, as exemplified in the areas of systems, theory and software development.
* Students will demonstrate ability to conduct a research or applied Computer Science project, requiring writing and presentation skills which exemplify scholarly style in computer science.

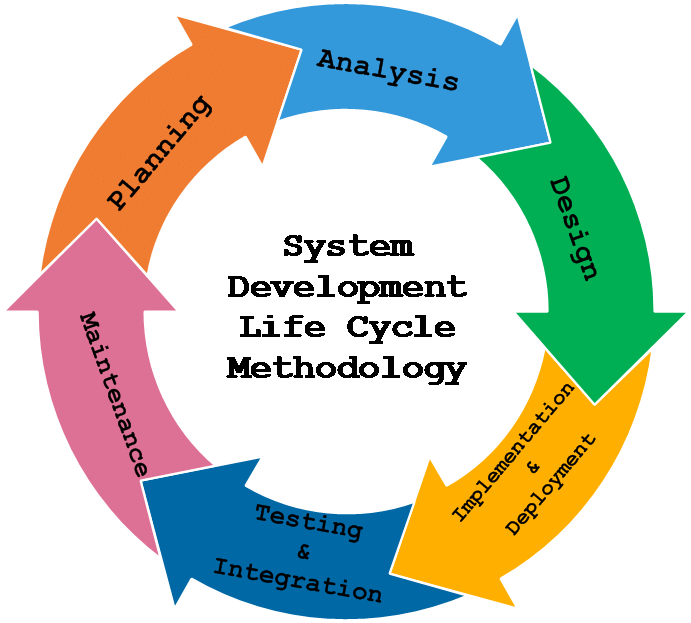
**PROPOSED SYSTEM:**

Today one cannot afford to rely on the fallible human beings of be really wants to stand against today’s merciless competition where not to wise saying **“to err is human”** no longer valid, it’s outdated to rationalize your mistake. So, to keep pace with time, to bring about the best result without malfunctioning and greater efficiency so to replace the unending heaps of flies with a much sophisticated hard disk of the computer.

One has to use the data management software. Software has been an ascent in atomization various organisations. Many software products working are now in markets, which have helped in making the organizations work easier and efficiently. Data management initially had to maintain a lot of ledgers and a lot of paper work has to be done but now software product on this organization has made their work faster and easier. Now only this software has to be loaded on the computer and work can be done.

This prevents a lot of time and money. The work becomes fully automated and any information regarding the organization can be obtained by clicking the button. Moreover, now it’s an age of computers of and automating such an organization gives the better look.

**SYSTEM DEVELOPMENT LIFE CYCLE (SDLC):**

****

The systems development life cycle is a project management technique that divides complex projects into smaller, more easily managed segments or phases. Segmenting projects allows managers to verify the successful completion of project phases before allocating resources to subsequent phases.

Software development projects typically include initiation, planning, design, development, testing, implementation, and maintenance phases. However, the phases may be divided differently depending on the organization involved.

For example, initial project activities might be designated as request, requirements-definition, and planning phases, or initiation, concept-development, and planning phases. End users of the system under development should be involved in reviewing the output of each phase to ensure the system is being built to deliver the needed functionality.

**PHASES OF SYSTEM DEVELOPMENT LIFE CYCLE-**

INITIATION PHASE:

The Initiation Phase begins when a business sponsor identifies a need or an opportunity.

The purpose of the Initiation Phase is to:

* Identify and validate an opportunity to improve business accomplishments of the organization or a deficiency related to a business need.
* Identify significant assumptions and constraints on solutions to that need.
* Recommend the exploration of alternative concepts and methods to satisfy the need including questioning the need for technology, i.e., will a change in the business process offer a solution?
* Assure executive business and executive technical sponsorship. The Sponsor designates a Project Manager and the business need is documented in a Concept Proposal. The Concept Proposal includes information about the business process andthe relationship to the Agency/Organization.
* Infrastructure and the Strategic Plan. A successful Concept Proposal results in a Project Management Charter which outlines the authority of the project manager to begin the project.

Careful oversight is required to ensure projects support strategic business objectives and resources are effectively implemented into an organization's enterprise architecture. The initiation phase begins when an opportunity to add, improve, or correct a system is identified and formally requested through the presentation of a business case. The business case should, at a minimum, describe a proposal’s purpose, identify expected benefits, and explain how the proposed system supports one of the organization’s business strategies. The business case should also identify alternative solutions and detail as many informational, functional, and network requirements as possible.

**SYSTEM CONCEPT DEVELOPMENT PHASE:**

The System Concept Development Phase begins after a business need or opportunity is validated by the Agency/Organization Program Leadership and the Agency/Organization CIO.

The purpose of the System Concept Development Phase is to:

* Determine the feasibility and appropriateness of the alternatives.
* Identify system interfaces.
* Identify basic functional and data requirements to satisfy the business need.
* Establish system boundaries; identify goals, objectives, critical success factors, and performance measures.
* Evaluate costs and benefits of alternative approaches to satisfy the basic functional requirements
* Assess project risks
* Identify and initiate risk mitigation actions, andDevelop high-level technical architecture, process models, data models, and a concept of operations. This phase explores potential technical solutions within the context of the business need.
* It may include several trade-off decisions such as the decision to use COTS software products as opposed to developing custom software or reusing software components, or the decision to use an incremental delivery versus a complete, onetime deployment.
* Construction of executable prototypes is encouraged to evaluate technology to support the business process. The System Boundary Document serves as an important reference document to support the Information Technology Project Request (ITPR) process.
* The ITPR must be approved by the State CIO before the project can move forward.

**PICTORIAL REPRESENTATION OF SDLC:**

**PLANNING PHASE:**

The planning phase is the most critical step in completing development, acquisition, and maintenance projects. Careful planning, particularly in the early stages of a project, is necessary to coordinate activities and manage project risks effectively. The depth and formality of project plans should be commensurate with the characteristics and risks of a given project. Project plans refine the information gathered during the initiation phase by further identifying the specific activities and resources required to complete a project.

A critical part of a project manager’ sjob is to coordinate discussions between user, audit, security, design, development, and network personnel to identify and document as many functional, security, and network requirements as possible. During this phase, a plan is developed that documents the approach to be used and includes a discussion of methods, tools, tasks, resources, project schedules, and user input. Personnel assignments, costs, project schedule, and target dates are established.

A Project Management Plan is created with components related to acquisition planning, configuration management planning, quality assurance planning, concept of operations, system security, verification and validation, and systems engineering management planning.

**REQUIREMENTS ANALYSIS PHASE:**

This phase formally defines the detailed functional user requirements using high-level requirements identified in the Initiation, System Concept, and Planning phases. It also delineates the requirements in terms of data, system performance, security, and maintainability requirements for the system. The requirements are defined in this phase to alevel of detail sufficient for systems design to proceed. They need to be measurable, testable, and relate to the business need or opportunity identified in the Initiation Phase. The requirements that will be used to determine acceptance of the system are captured in the Test and Evaluation MasterPlan.

The purposes of this phase are to:

* Further define and refine the functional and data requirements and document them in the Requirements Document,
* Complete business process reengineering of the functions to be supported (i.e., verify what information drives the business process, what information is generated, who generates it, where does the information go, and who processes it),
* Develop detailed data and process models (system inputs, outputs, and the process.
* Develop the test and evaluation requirements that will be used to determine acceptable system performance**.**

**DESIGN PHASE:**

The design phase involves converting the informational, functional, and network requirements identified during the initiation and planning phases into unified design specifications that developers use to scriptprograms during the development phase. Program designs are constructed in various ways. Using a top-down approach, designers first identify and link major program components and interfaces, then expand design layouts as they identify and link smaller subsystems and connections. Using a bottom-up approach, designers first identify and link minor program components and interfaces, then expand design layouts as they identify and link larger systems and connections. Contemporary design techniques often use prototyping tools that build mock-up designs of items such as application screens, database layouts, and system architectures. End users, designers, developers, database managers, and network administrators should review and refine the prototyped designs in an iterative process until they agree on an acceptable design. Audit, security, and quality assurance personnel should be involved in the review and approval process. During this phase, the system is designed to satisfy the functional requirements identified in the previous phase. Since problems in the design phase could be very

expensive to solve in the later stage of the software development, a variety of elements are considered in the design to mitigate risk. These include:

* Identifying potential risks and defining mitigating design features.
* Performing a security risk assessment.
* Developing a conversion plan to migrate current data to the new system.
* Determining the operating environment.
* Defining major subsystems and their inputs and outputs.
* Allocating processes to resources.
* Preparing detailed logic specifications for each software module. The result is a draft System Design Document which captures the preliminary design for the system.
* Everything requiring user input or approval is documented and reviewed by the user. Once these documents have been approved by the Agency CIO and Business Sponsor, the final System Design Document is created to serve as the Critical/Detailed Design for the system.
* This document receives rigorous review by Agency technical and functional representatives to ensure that it satisfies the business requirements. Concurrent with the development of the system design, the Agency Project Manager begins development of the Implementation Plan, Operations and Maintenance Manual, and the Training Plan.

**DEVELOPMENT PHASE:**

The development phase involves converting design specifications into executable programs. Effective development standards include requirements that programmers and other project participants discuss design specifications before programming begins. The procedures help ensure programmers clearly understand program designs and functional requirements. Programmers use various techniques to develop computer programs. The large transaction oriented programs associated with financial institutions have traditionally been developed using procedural programming techniques. Procedural programming involves the line-by-line scripting of logical instructions that are combined to form a program.Effective completion of the previous stages is a key factor in the success of the Development phase. The Development phase consists of:

* Translating the detailed requirements and design into system components.
* Testing individual elements (units) for usability.
* Preparing for integration and testing of the IT system.

**INTEGRATION AND TEST PHASE:**

* Subsystem integration, system, security, and user acceptance testing is conducted during the integration and test phase. The user, with those responsible for quality assurance, validates that the functional requirements, as defined in the functional requirements document, are satisfied by the developed or modified system. OIT Security staff assess the system security and issue a security certification and accreditation prior to installation/implementation.

***Multiple levels of testing are performed, including***:

* Testing at the development facility by the contractor and possibly supported by end users
* Testing as a deployed system with end users working together with contract personnel
* Operational testing by the end user alone performing all functions. Requirements are traced throughout testing, a final Independent Verification & Validation evaluation is performed and all documentation is reviewedand accepted prior to acceptance of the system.

**IMPLEMENTATION PHASE:**

This phase is initiated after the system has been tested and accepted by the user. In this phase, the system is installed to support the intended business functions. System performance is compared to performance objectives established during the planning phase. Implementation includes user notification, user training, installation of hardware, installation of software onto production computers, and integration of the system into daily work processes. This phase continues until the system is operating in production in accordance with the defined userrequirements.

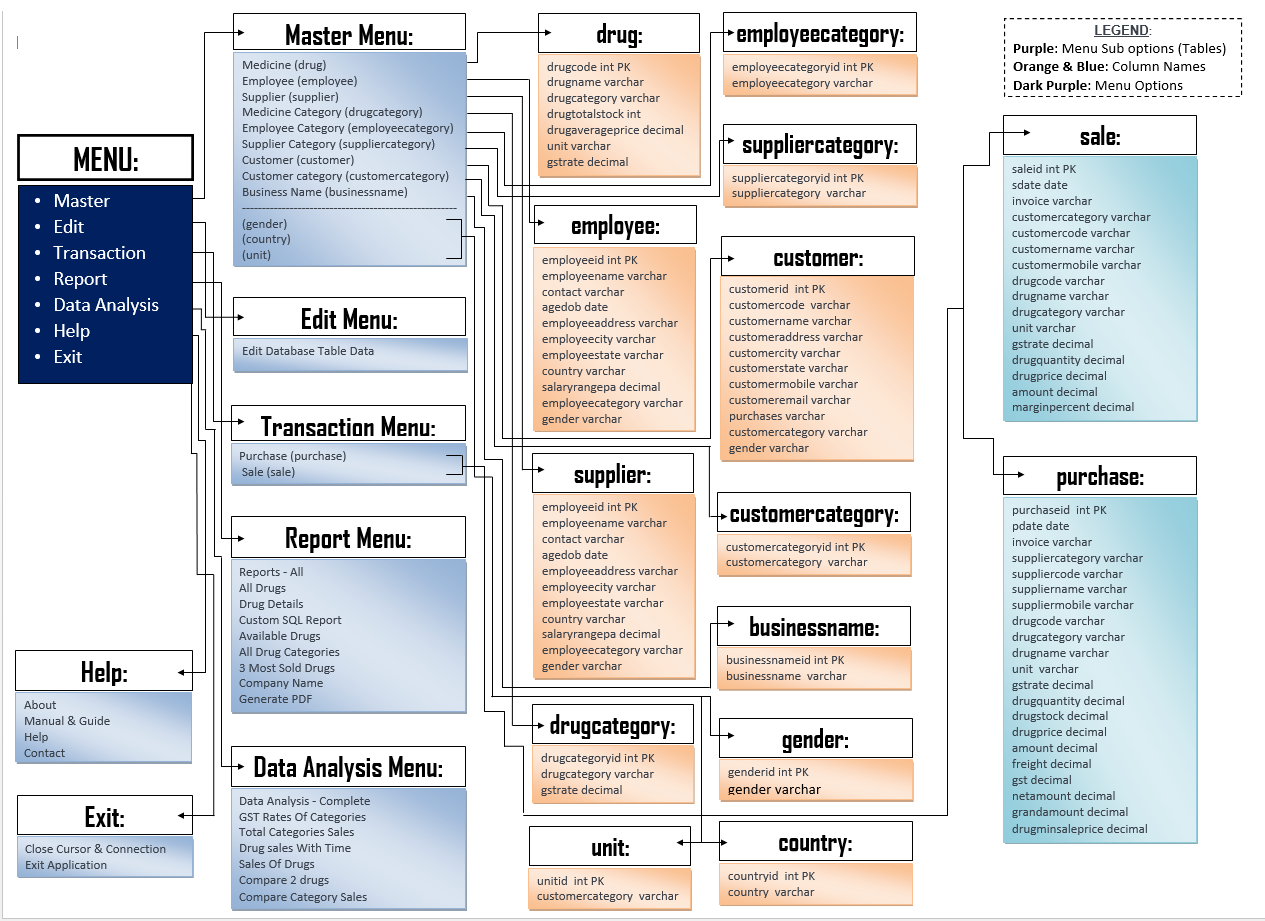
**OPERATIONS AND MAINTENANCE PHASE:**

The system operation is ongoing. The system is monitored for continued performance in accordance with user requirements and needed system modifications are incorporated. Operations continue as long as the system can be effectively adapted to respond to the organization’s needs. When modifications or changes are identified, the system may reenter the planning phase.

***The purpose of this phase is to:***

* Operate, maintain, and enhance the system.
* Certify that the system can process sensitive information.
* Conduct periodic assessments of the system to ensure the functional requirements continue to be satisfied.
* Determine when the system needs to be modernized, replaced, or retired.

**BLOCK DIAGRAM:-**

****

This block diagram was intended to be displayed on an A3 Sheet, which is why it could not be clearly portrayed on this A4 sized page, which you are currently viewing…

**Opulent Pathway Logo:-**

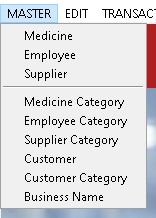


**Functionality Screenshots:-**

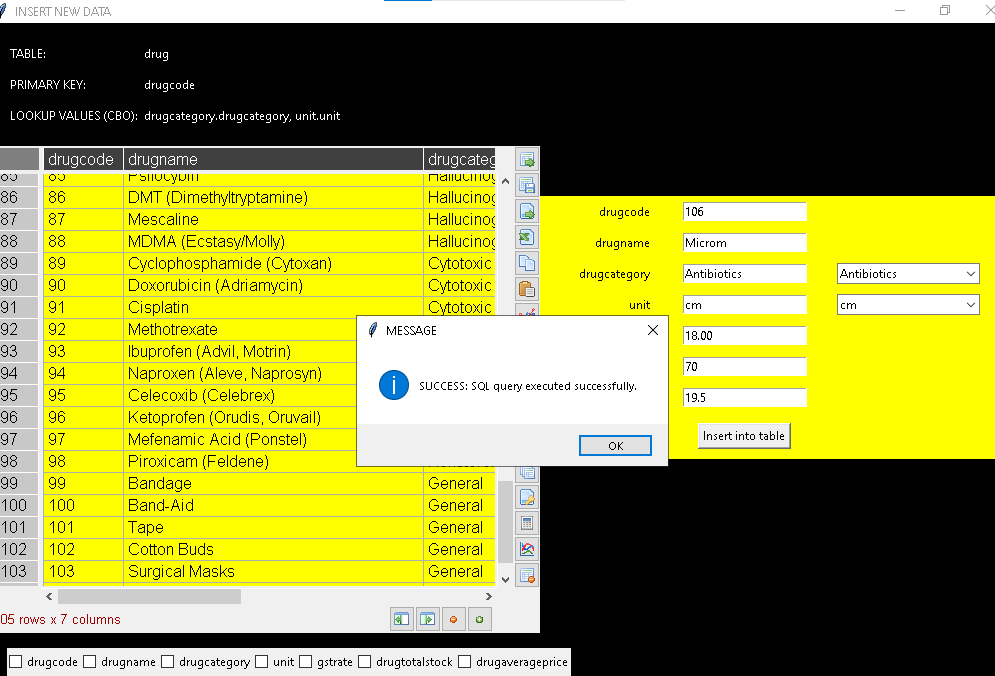
All Menu Options:-



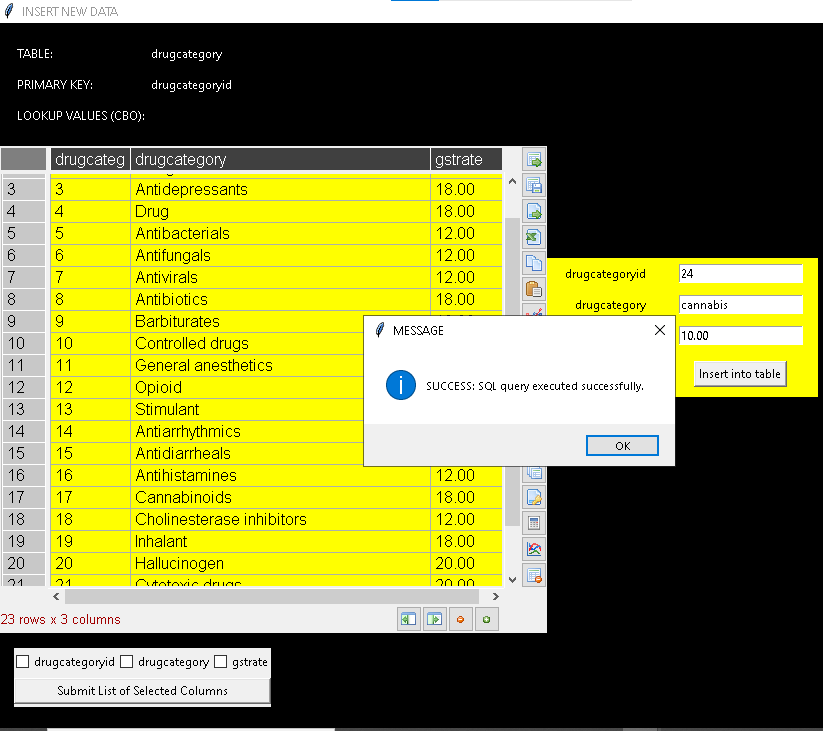
Master Menu:-



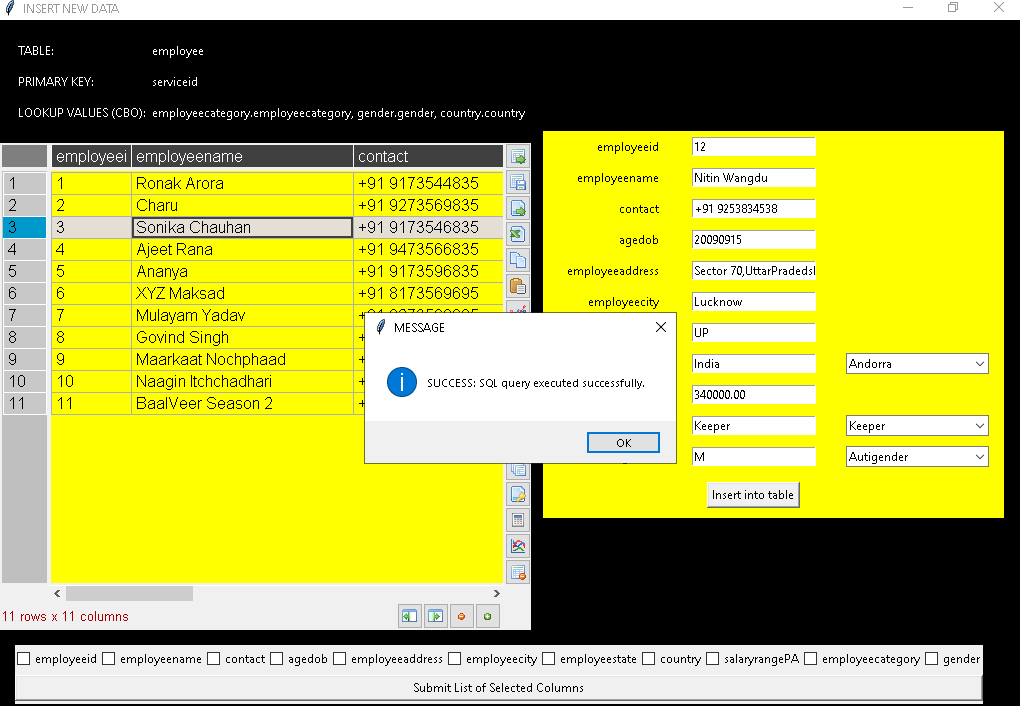
Medicine (Table: drug):-



Medicine Category (table: drugcategory):-



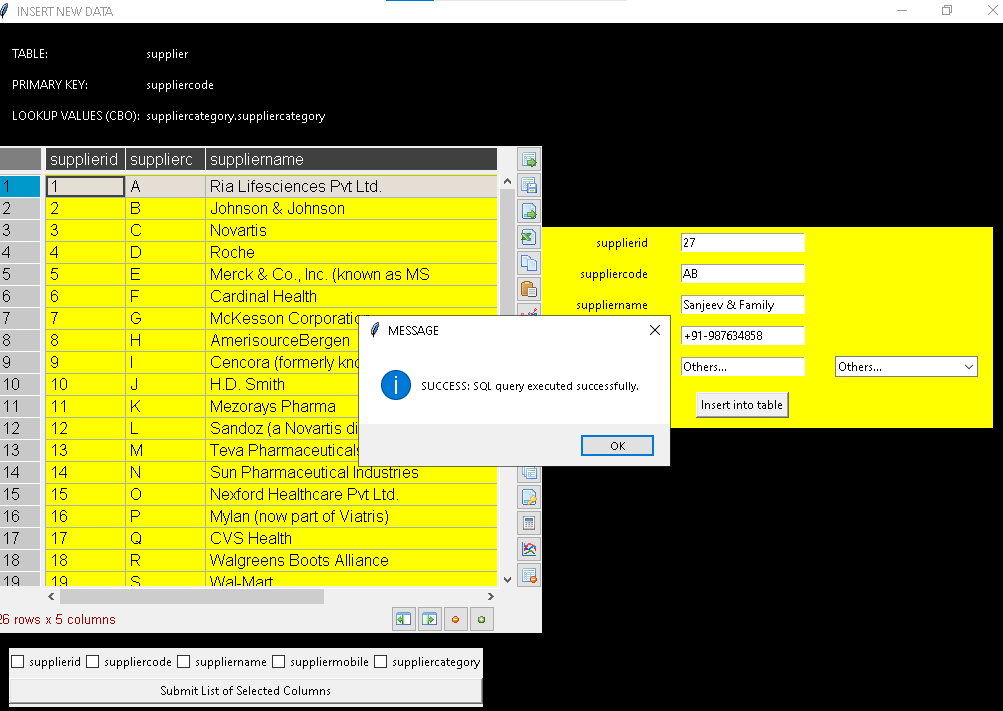
Employee (Table: employee):-



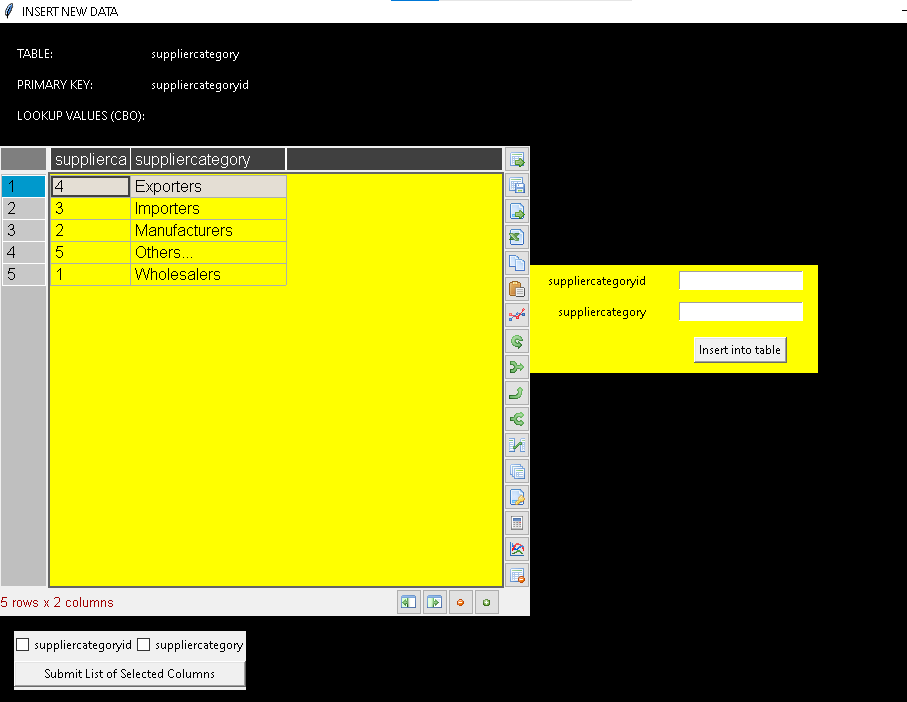
Employee Category (Table: employeecategory):-



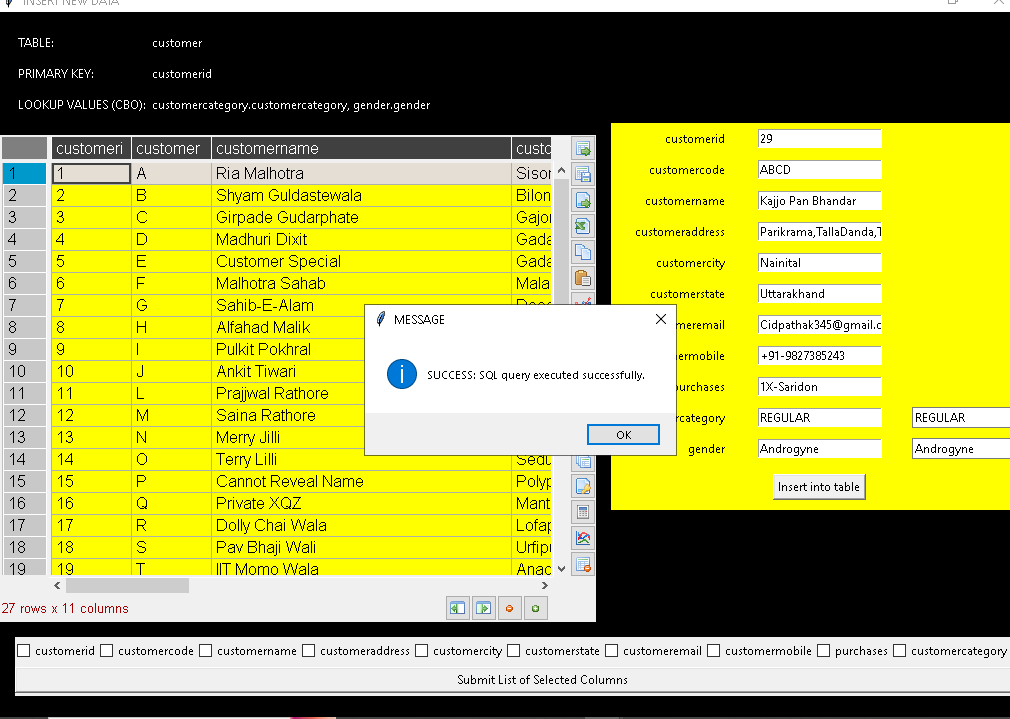
Supplier (Table: supplier):-

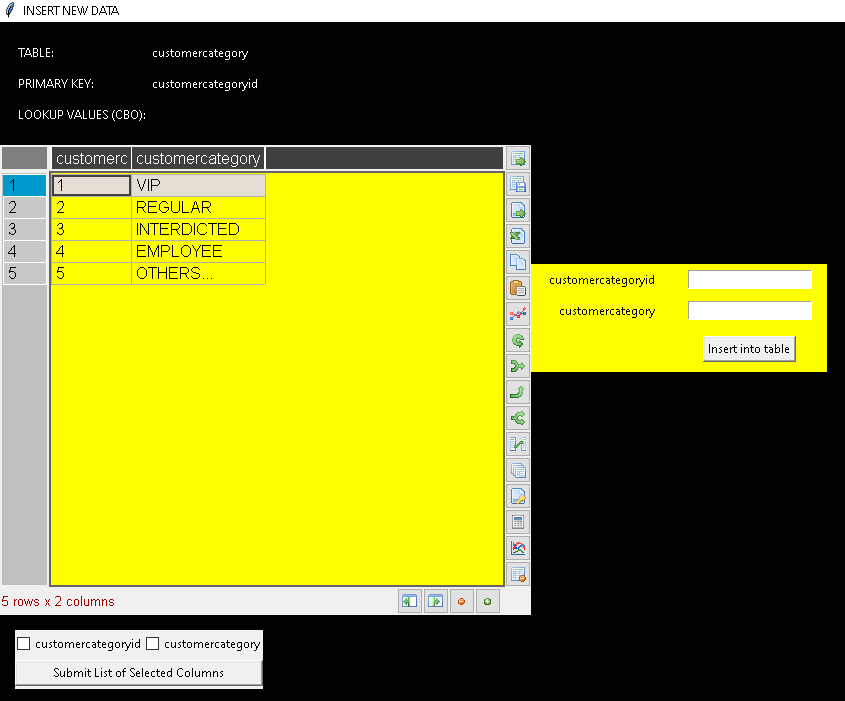


Supplier Category (Table: suppliercategory):-

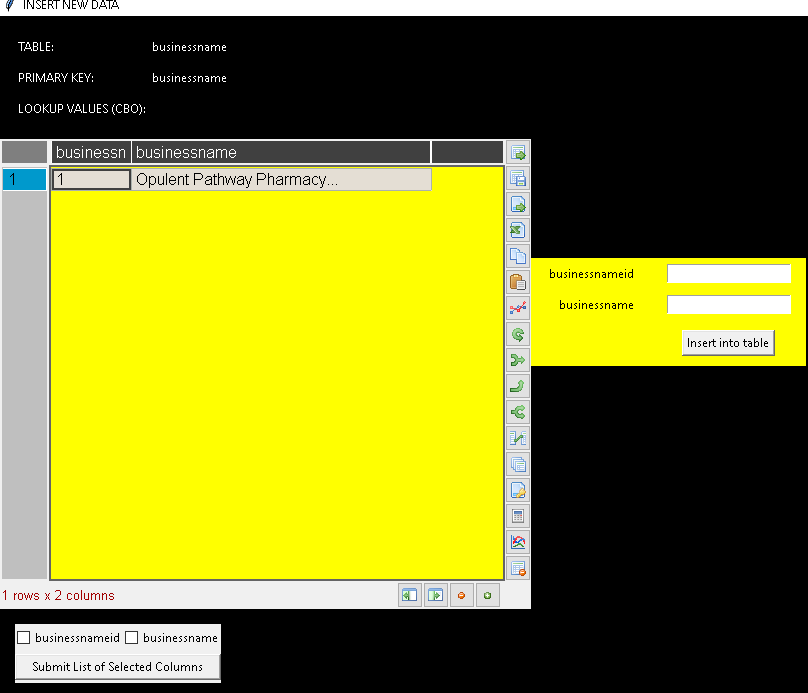


Customer (Table: customer):-



Customer Category (Table: customercategory):-

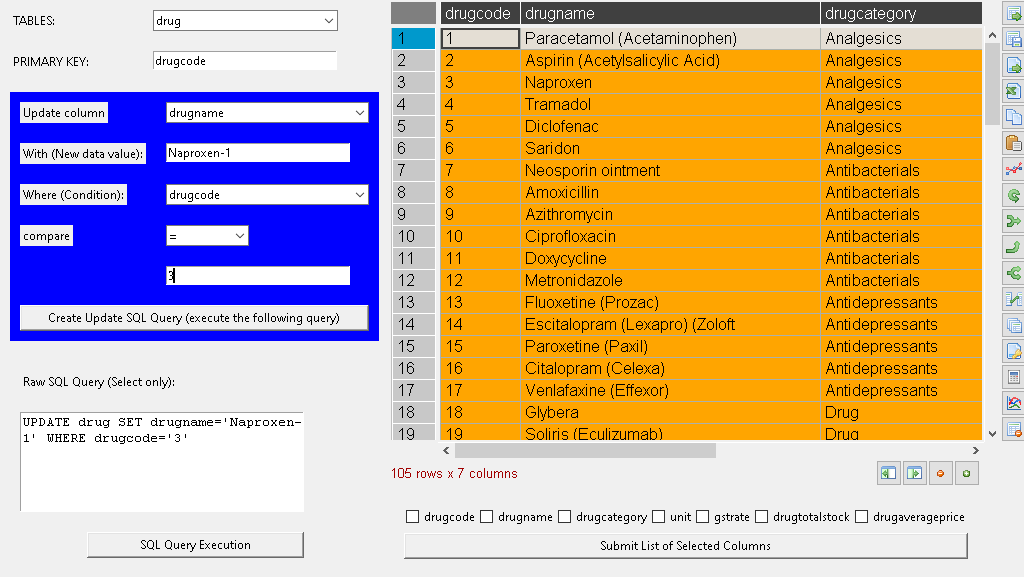
Business Name (Table: businessname):-



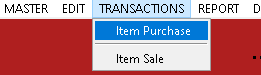
Edit Menu:-



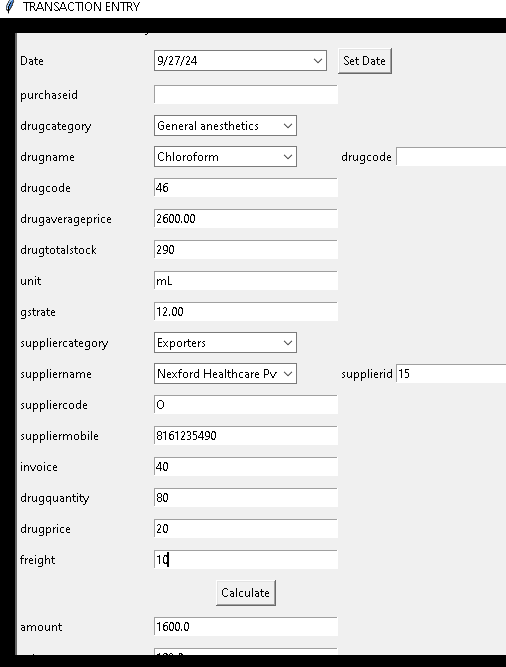
Edit Database Table Data (Table: All tables):-



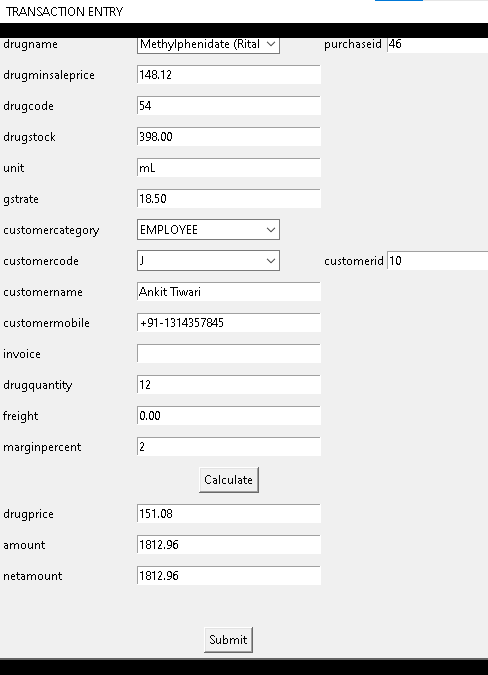
Transaction Menu:-



Item Purchase:-



Item Sale:-

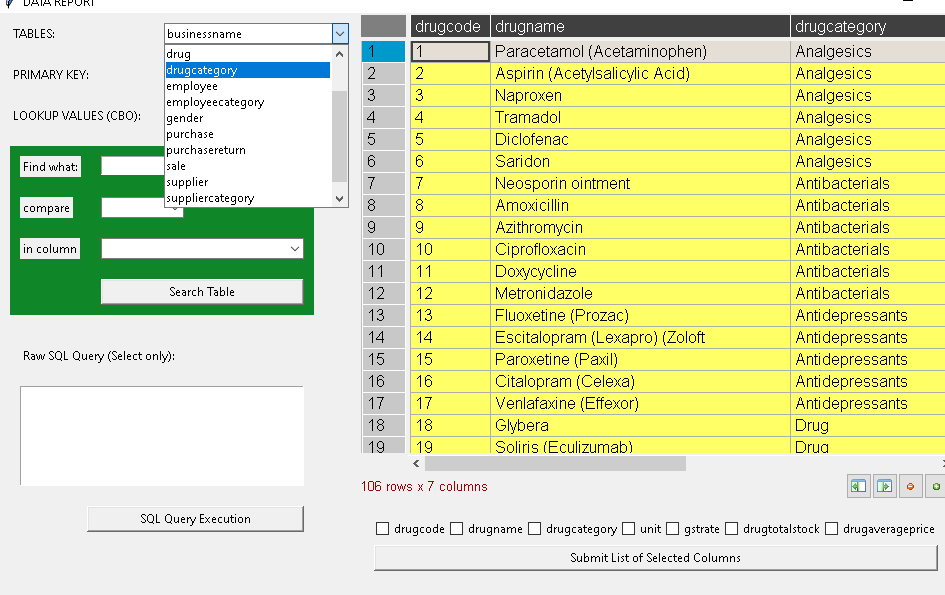


**Report & Data Analysis Screenshots:-**

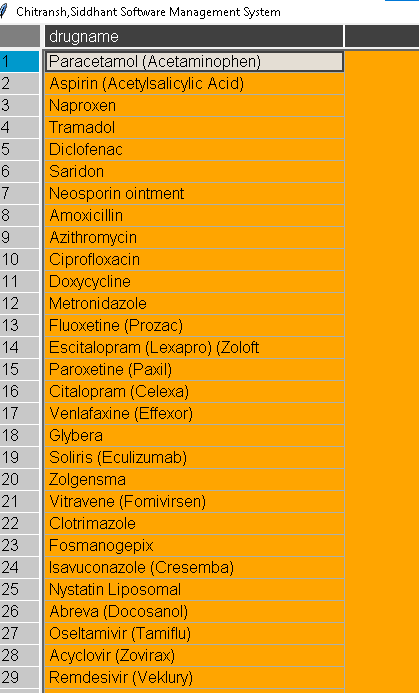
Report Menu:-



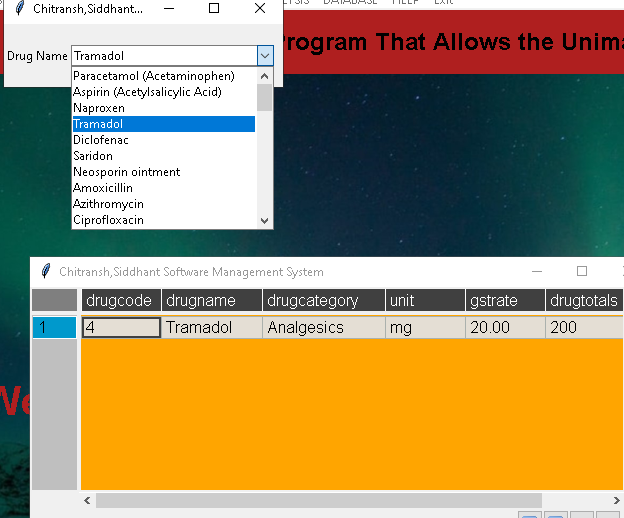
Reports - All:-

****

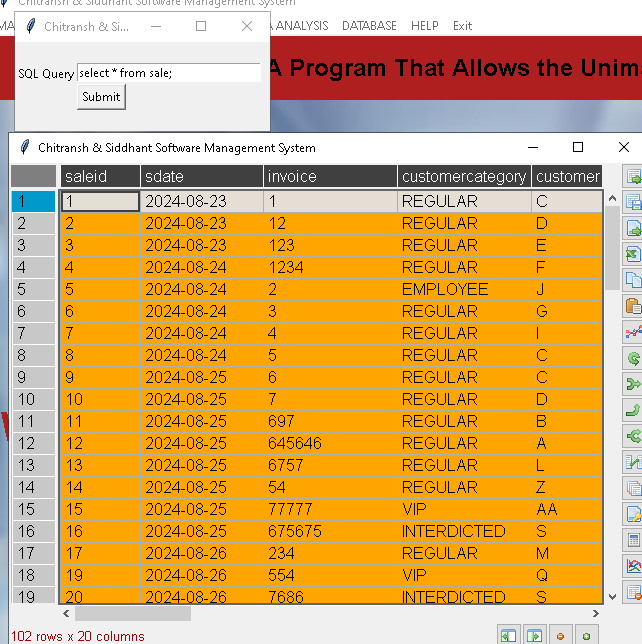
All Drugs:-

****

Drug Details:-

****

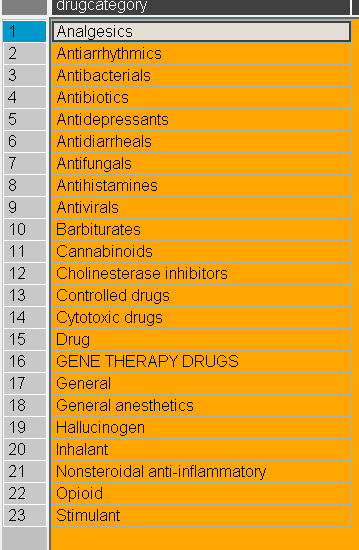
Custom SQL Report:-

****

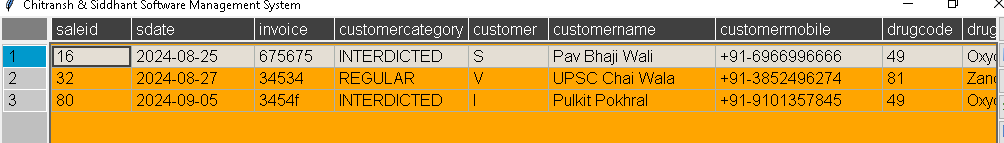
Available Drugs:-

****

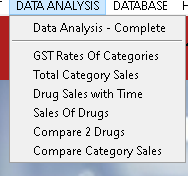
All Drug Catagories:-

****

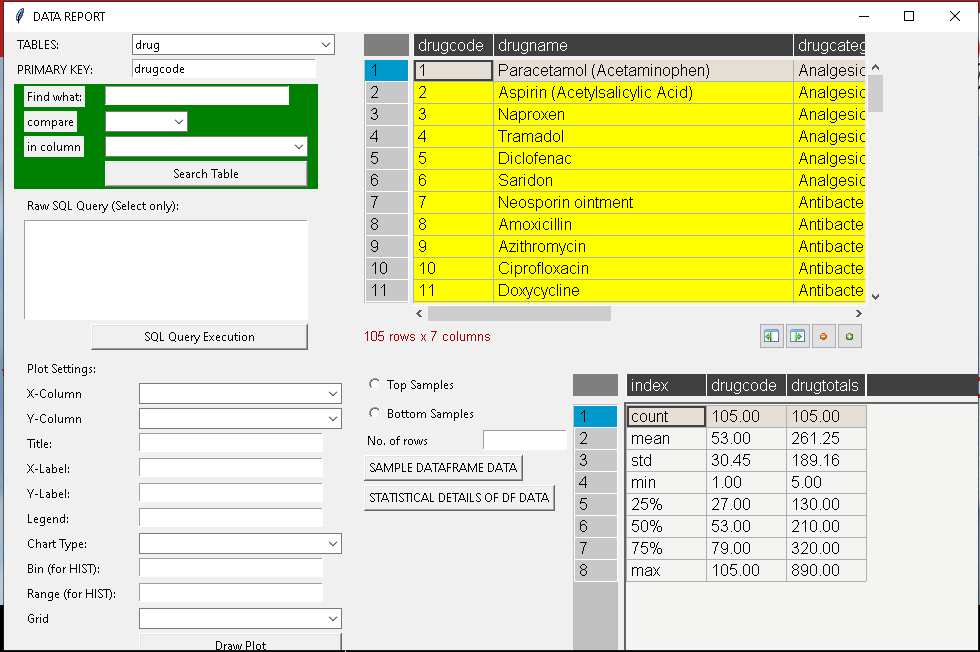
Three most Sold Drugs:-

****

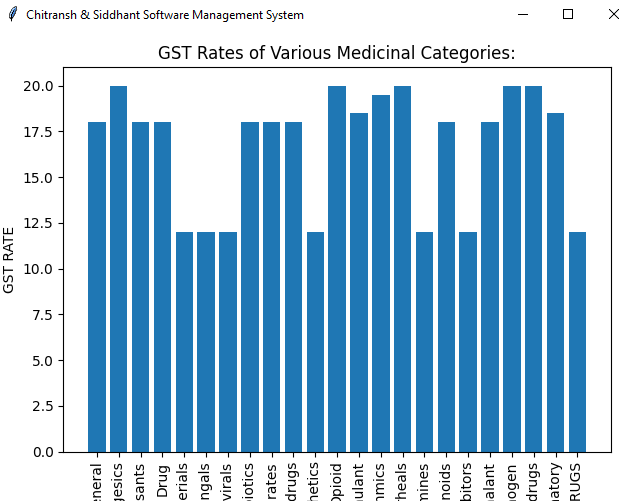
Data Analysis:-

****

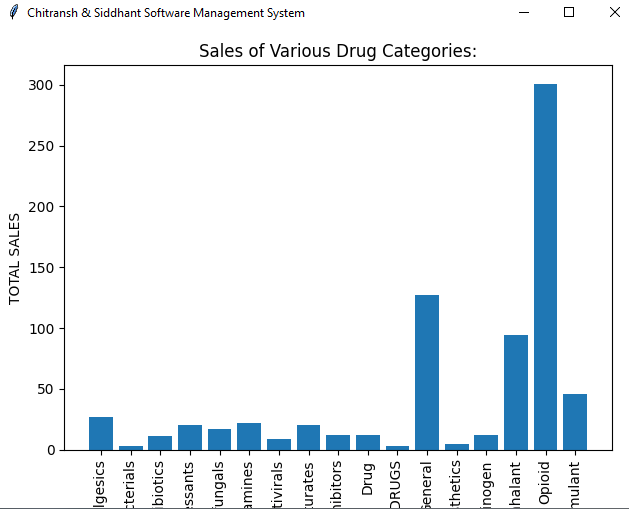
**Data Analysis – Complete:-**

****

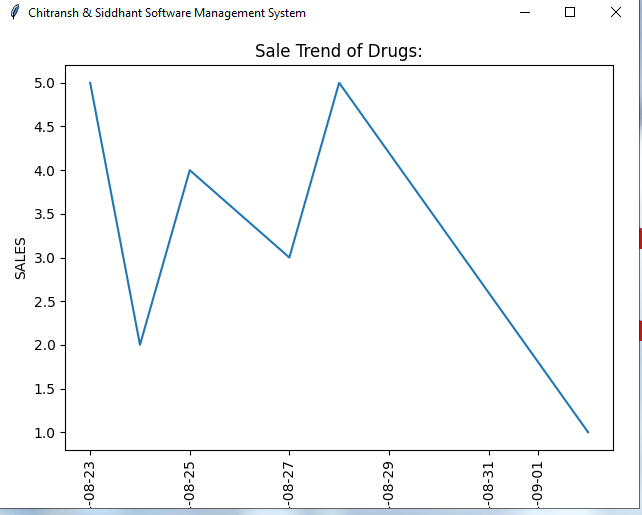
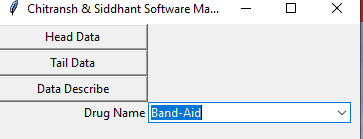
**GST Rates Of Categories:-**

****

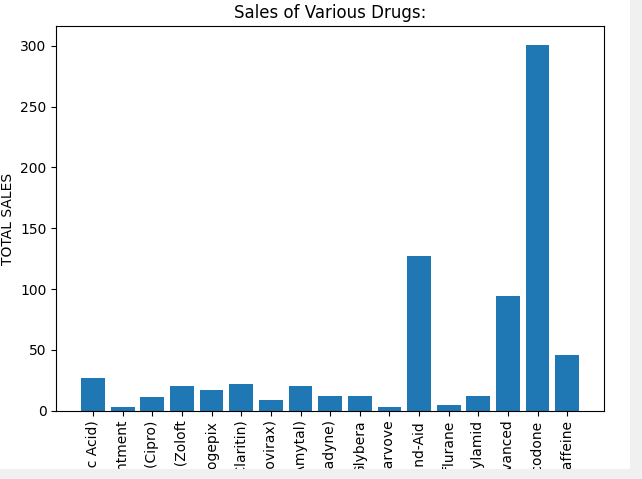
**Total Category Sales:-**

****

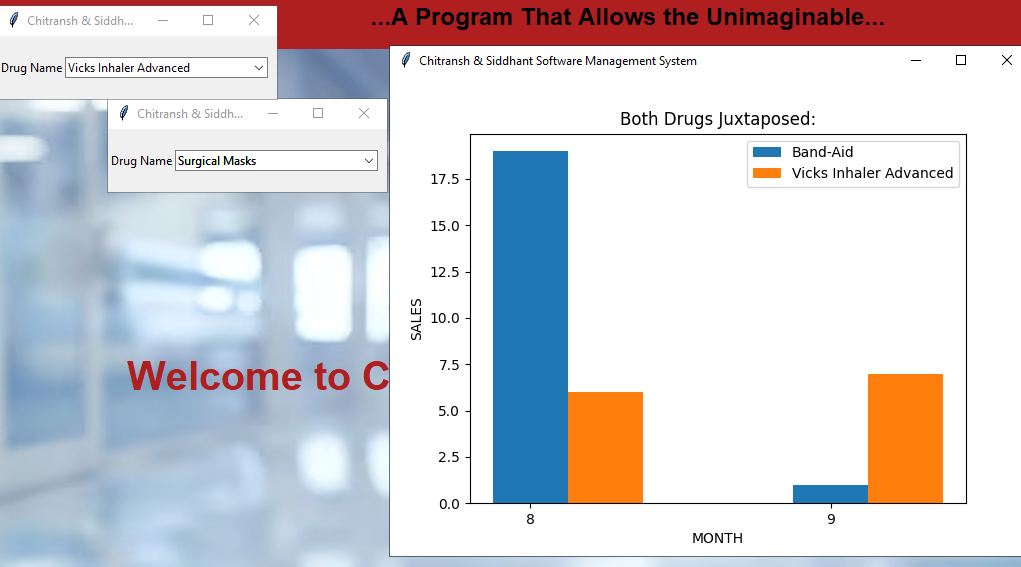
**Drug Sales With Time:-**

****

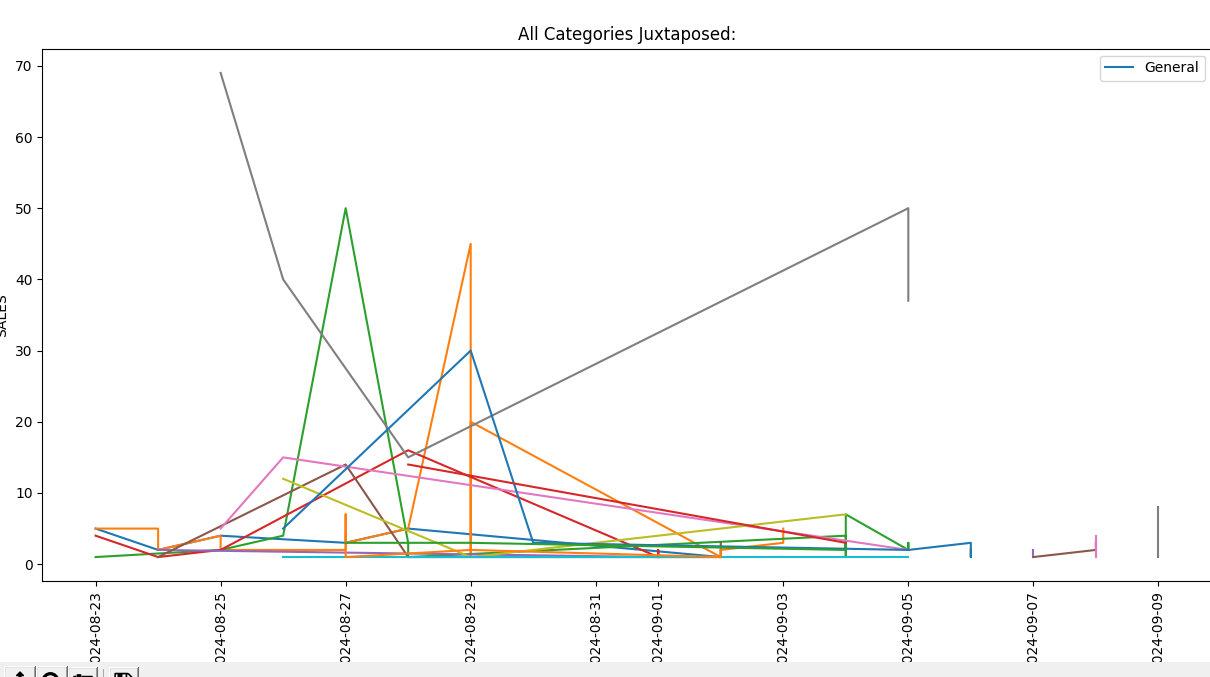
**Sales Of Drugs:-**

****

**Compare 2 Drugs:-**



**Compare Category Sales:-**

****

**INSTALLATION PROCEDURE:**

------------------------------------------------------------------------------------------------

Drug Management system:-

------------------------------------------------------------------------------------------------

**Pre-requisites:-**

* You have to have the following softwares for the successful running of this software; which are as follows:

I) Python (Only for the First time), which can be downloaded from “*https://www.python.org/downloads/*”

II) MySQL (Only for the First time), which can be run via Command Prompt which you already probably have on your Desktop Device.

**Running The Software:-**

* Simply run the file “*index.py*” File from the folder named:

“*d22111284 12A Chitransh's Opulent Pathway Pharmacy*” and use the software!

**TESTING:-**

Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which it is intended to operate. Software Testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

It can also be stated as the process of validating and verifying that a software program/application/product meets the business and technical requirements that guided its design and development, so that it works as expected and can be implemented with the same characteristics. Software Testing, depending on the testing method employed, can be implemented at any time in the development process, however the most test effort is employed after the requirements have been defined and coding process has been completed.

**TESTING METHODS:**

Software testing methods are traditionally divided into black box testing and white box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

**BLACK BOX TESTING:**

Black box testing treats the software as a "black box," without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

**SPECIFICATION-BASED TESTING:**

Specification-based testing aims to test the functionality of software according to the applicable requirements.Thus, the tester inputs data into, and only sees the output from, the test object. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behaviour), either "is" or "is not" the same as the expected value specified in the test case. Specification-based testing is necessary, but it is insufficient to guard against certain risks

**ADVANTAGES AND DISADVANTAGES:**

The black box tester has no "bonds" with the code, and a tester's perception is very simple: a code must have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers don't. But, on the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed.

That's why there are situations when (1) a black box tester writes many test cases to check something that can be tested by only one test case, and/or (2) some parts of the back end are not tested at all. Therefore, black box testing has the advantage of "an unaffiliated opinion," on the one hand, and the disadvantage of "blind exploring," on the other.

**WHITE BOX TESTING:**

White box testing, by contrast to black box testing, is when the tester has access to the internal data structures and algorithms (and the code that implement these)

***Types of white box testing:-***

The following types of white box testing exist:

* API testing - Testing of the application using Public and Private APIs.
* Code coverage - creating tests to satisfy some criteria of code coverage.

For example, the test designer can create tests to cause all statements in the program to be executed at least once.

* Fault injection methods.
* Mutation testing methods.
* Static testing - White box testing includes all static testing.

**CODE COMPLETENESS EVALUATION:**

White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.

***Two common forms of code coverage are:***

* Function Coverage: Which reports on functions executed and
* Statement Coverage: Which reports on the number of lines executed to complete the test?

***They both return coverage metric, measured as a percentage.***

**HARDWARE AND SOFTWARE REQUIREMENTS:-**

HARDWARE REQUIREMENTS**:**

I. OPERATING SYSTEM : WINDOWS 7 or Above

II. PROCESSOR : AMD RYZEN – ANY or

INTEL CORE i3 or i5, above 10th GEN

III. MOTHERBOARD : 1.845 or 915,995 FOR PENTIUM or MSI K9MM-V VIA K8M800+8237R

PLUS CHIPSET FOR AMD ATHALON or supposedly any...

IV. RAM : 512 MB+

V. Hard disk : 40 GB OR ABOVE (SSD is recommended)

VI. CD/DVD/PD/SD+OTG/other: (If Data Backup required)

VII. FLOPPY DISK 1.44 MB : (If Data Backup required)

VIII. MONITOR : Anywhere between or around 13 to 16

Inches, IPS/TN/VA/OLED Panel, 50-200Hz Refresh Rate Monitor Display

IX. KEYBOARD and MOUSE : ANY

SOFTWARE REQUIREMENTS:

1. Windows OS / Any Other OS
2. Python Compiler (IDE)…like IDLE, Spyder IDE…etc.



**By:**  Chitransh Bhatnagar

& Siddhant Pathak

**Source code** of this python program is mentioned below in forms of files (.py) as under:

\_about.py

import tkinter as tk

from tkinter import \*

from tkinter import ttk

import tkinter.messagebox as tkmsgbox

import pymysql

import pandas as pd

import numpy as np

def manual():

frameHelp = Tk()

frameHelp.title('Manual by Chitransh:')

r=0

Label(frameHelp, height=30, width=100, text='"MASTER:" This allows you to input data\nif you want to, for example you saw a customer like never before, you immediately\nthen can go to customercategory ioption in this menu and insert one...\n\n"EDIT:" And then comes this, where you can easily handle your mistakes\nby editing them in a few seconds...\n\n"TRANSACTIONS:" Here you can insert data that you need to insert everyday\nlike if any customer comes, then you can enter his/her name\nin MASTER, then come to transaction to enter data like what did he/she buy,\nhow much/ when/ who bought the stuff and so on...\n\n"REPORT:" And then if you want to see the data you just entered,\njust click on REPORT and then Reports-All to find out what data you have...\n\n"DATA ANALYSIS:" This is self-explanatory about what this does, it allows you see the trend,\npredict future sales trend with a certain probability, make business decisions and so on...\n\n"DATABASE:" This option helps you manage your database,\ni.e. a place where all your tables exist, for example-\nyou can take a backup of your database if you think you might lose it...\n\n"HELP:" This is the menu which you are using right now\nto see the manual that corresponds with the software, this also\nhas other options that allow you to search for the query you have regarding running of program...\n\n"EXIT:" This way you can close the program...').grid(row=r, column=0)

frameHelp.mainloop()

def help():

frameHelp = Tk()

frameHelp.title('Help by Chitransh:')

r=0

Label(frameHelp, height=10, width=50, text='If tables do not appear in the "Reports-All" report Menu suboption,\n Then click on Database Option\nand Click on the Create Tables Option.\nIf you are worried about losing your Data,\nClick on Database option,\nand then Click on Backup Database, and you will nver lose it...').grid(row=r, column=0)

frameHelp.mainloop()

def about():

frameHelp = Tk()

frameHelp.title('About This Software:')

r=0

Label(frameHelp, height=10, width=50, text="This Software is designed to help enter the data and easily manage it using variety of options including those that could potentially save you a ton of time, i.e. options that will help you analyse your data using graphs like line, bar and other charts, therefore showcasing thousands of data values within a fraction of second rapidly decreasing the time you would normally take to make a business decision. Therefore this is not just a storage hub, but is like a friend who is always available for your help in the time when you need it the most...").grid(row=r,column=5)

frameHelp.mainloop()

def contact():

frameHelp = Tk()

frameHelp.title('Contact Chitransh At:')

r=0

Label(frameHelp, height=5, width=50, text="Tele. No.: +91-9453622857\nAddress: Near Nanital, UK, 262001\nEmail: smartestdeveloper69@mydomain.com").grid(row=r, column=0)

frameHelp.mainloop()

\_createMySQLTablesWithTestData.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#=============================================================================

def createTablesWithTestData(frame):

cursor = conn.cursor()

#Creating Tables with Test Data: INVENTORY

#---- 1. drug

#---- 2. drugcategory

#---- 3. employee

#---- 4. employeecategory

#---- 5. supplier

#---- 6. suppliercategory

#---- 7. gender

#---- 8. country

#---- 9. unit

#---- 10. customer

#---- 11. customercategory

#---- 12. businessname

#---- 13. purchase

#---- 14. sale

tbl = ''

try:

tkmsgbox.showinfo("MESSAGE!", "Creating tables with test data...", parent=frame)

#================================================================

# !IMPORTANT: date column should be named something

# as 'transdate', 'pdate', 'sdate', 'doj' etc.

# BUT NOT AS 'date'

#================================================================

tbl = 'drug'

try:

cursor.execute('DROP TABLE drug')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS drug(

drugcode INT PRIMARY KEY AUTO\_INCREMENT,

drugname VARCHAR(30) NOT NULL,

drugcategory VARCHAR(30),

unit VARCHAR(10),

gstrate DECIMAL(10,2) default 18.00,

drugtotalstock INT(5) default 0,

drugaverageprice DECIMAL(12,2) default 0

)"""

cursor.execute(sql)

sql = """INSERT INTO drug

(drugcode,drugname,drugcategory,unit,gstrate,drugtotalstock,drugaverageprice)

VALUES

(1,'Paracetamol (Acetaminophen)','Analgesics','mg',20,120,90.00),

(2,'Aspirin (Acetylsalicylic Acid)','Analgesics','mg',20,600,20.00),

(3,'Naproxen','Analgesics','mg',20,100,30.00),

(4,'Tramadol','Analgesics','mg',20,200,70.00),

(5,'Diclofenac','Analgesics','mg',20,500,100.00),

(6,'Saridon','Analgesics','mg',20,200,150.00),

(7,'Neosporin ointment','Antibacterials','ml',12,330,60.00),

(8,'Amoxicillin','Antibacterials','ml',12,350,80.00),

(9,'Azithromycin','Antibacterials','ml',12,130,100.00),

(10,'Ciprofloxacin','Antibacterials','ml',12,530,60.00),

(11,'Doxycycline','Antibacterials','ml',12,130,120.00),

(12,'Metronidazole','Antibacterials','ml',12,290,310.00),

(13,'Fluoxetine (Prozac)','Antidepressants','mL',18,400,450.00),

(14,'Escitalopram (Lexapro) (Zoloft)','Antidepressants','mL',18,110,120.00),

(15,'Paroxetine (Paxil)','Antidepressants','mL',18,210,50.00),

(16,'Citalopram (Celexa)','Antidepressants','mL',18,280,50.00),

(17,'Venlafaxine (Effexor)','Antidepressants','mL',18,60,150.00),

(18,'Glybera','Drug','mL',18,140,25.00),

(19,'Soliris (Eculizumab)','Drug','mL',18,130,95.00),

(20,'Zolgensma','Drug','mL',18,120,55.00),

(21,'Vitravene (Fomivirsen)','Drug','mL',18,120,375.00),

(22,'Clotrimazole','Antifungals','mg',12,80,980.00),

(23,'Fosmanogepix','Antifungals','mg',12,100,70.00),

(24,'Isavuconazole (Cresemba)','Antifungals','mg',12,450,620.00),

(25,'Nystatin Liposomal','Antifungals','mL',12,150,50.00),

(26,'Abreva (Docosanol)','Antivirals','mL',12,650,29.00),

(27,'Oseltamivir (Tamiflu)','Antivirals','mL',12,60,219.00),

(28,'Acyclovir (Zovirax)','Antivirals','mL',12,70,229.00),

(29,'Remdesivir (Veklury)','Antivirals','mL',12,150,374.00),

(30,'Amoxicillin','Antibiotics','mL',18,440,34.00),

(31,'Ciprofloxacin (Cipro)','Antibiotics','mL',18,210,134.00),

(32,'Azithromycin (Zithromax)','Antibiotics','mL',18,40,324.00),

(33,'Doxycycline (Vibramycin)','Antibiotics','mL',18,320,234.00),

(34,'Clindamycin (Cleocin)','Antibiotics','mL',18,142,214.00),

(35,'Phenobarbital (Luminal)','Barbiturates','mL',18,320,87.00),

(36,'Pentobarbital (Nembutal)','Barbiturates','mg',18,220,187.00),

(37,'Secobarbital (Seconal)','Barbiturates','mL',18,120,233.00),

(38,'Amobarbital (Amytal)','Barbiturates','mg',18,310,247.00),

(39,'Thiopental (Pentothal)','Barbiturates','mL',18,240,149.00),

(40,'Adderall','Controlled drugs','mg',18,290,95.00),

(41,'Fentanyl','Controlled drugs','mg',18,290,195.00),

(42,'Diazepam (Valium)','Controlled drugs','mg',18,290,235.00),

(43,'Hydrocodone (Vicodin, Norco)','Controlled drugs','mg',18,290,134.00),

(44,'Methadone','Controlled drugs','mg',18,290,64.33),

(45,'Sevoflurane','General anesthetics','mL',12,40,1000.00),

(46,'Chloroform','General anesthetics','mL',12,290,2600.00),

(47,'Isoflurane','General anesthetics','mL',12,254,2500.00),

(48,'Morphine','Opioid','mL',20,201,83.00),

(49,'Oxycodone','Opioid','mL',20,212,25.00),

(50,'Codeine','Opioid','mL',20,150,145.00),

(51,'Fentanyl','Opioid','mL',20,130,665.00),

(52,'Caffeine','Stimulant','mL',18.5,830,24.00),

(53,'Amphetamine (Adderall)','Stimulant','mL',18.5,840,215.00),

(54,'Methylphenidate (Ritalin, Concerta)','Stimulant','mL',18.5,70,125.00),

(55,'Nicotine','Stimulant','g',18.5,470,999.99),

(56,'Amiodarone','Antiarrhythmics','mg',19.5,650,185.00),

(57,'Lidocaine','Antiarrhythmics','mL',19.5,220,125.00),

(58,'Flecainide','Antiarrhythmics','mg',19.5,150,112.00),

(59,'Amiodarone','Antiarrhythmics','mL',19.5,410,65.00),

(60,'Sotalol','Antiarrhythmics','mg',19.5,130,95.00),

(61,'Procainamide','Antidiarrheals','mL',20,130,625.00),

(62,'Diphenhydramine (Benadryl)','Antihistamines','mL',12,120,85.00),

(63,'Cetirizine (Zyrtec)','Antihistamines','mL',12,150,105.00),

(64,'Loratadine (Claritin)','Antihistamines','mL',12,630,45.00),

(65,'Fexofenadine (Allegra)','Antihistamines','mL',12,230,375.00),

(66,'Delta-9-Tetrahydrocannabinol (THC)','Cannabinoids','mg',18,130,125.00),

(67,'Cannabidiol (CBD)','Cannabinoids','mL',18,160,95.00),

(68,'Cannabinol (CBN)','Cannabinoids','mg',18,172,235.00),

(69,'Cannabigerol (CBG)','Cannabinoids','mL',18,151,62.00),

(70,'Cannabichromene (CBC)','Cannabinoids','mg',18,450,155.00),

(71,'Donepezil (Aricept)','Cholinesterase inhibitors','mg',12,130,21.50),

(72,'Rivastigmine (Exelon)','Cholinesterase inhibitors','mL',12,170,123.50),

(73,'Galantamine (Razadyne)','Cholinesterase inhibitors','mg',12,181,261.50),

(74,'Tacrine (Cognex)','Cholinesterase inhibitors','mL',12,370,291.50),

(75,'Physostigmine','Cholinesterase inhibitors','mg',12,210,521.50),

(76,'Vicks Vaporub','Inhalant','g',18,180,16.00),

(77,'Vicks Inhaler','Inhalant','mL',18,190,61.00),

(78,'Vicks Inhaler Advanced','Inhalant','mL',18,120,52.00),

(79,'Vicks Balm','Inhalant','mL',18,156,84.00),

(80,'Zandu Balm Red','Inhalant','mL',18,220,36.00),

(81,'Zandu Balm Blue','Inhalant','qty',18,890,44.00),

(82,'Zandu Balm Green','Inhalant','qty',18,640,77.00),

(83,'Nepali Balm','Inhalant','qty',18,510,42.00),

(84,'LSD (Lysergic Acid Diethylamide)','Hallucinogen','mL',20,320,97.00),

(85,'Psilocybin','Hallucinogen','mL',20,736,57.00),

(86,'DMT (Dimethyltryptamine)','Hallucinogen','mL',20,120,997.00),

(87,'Mescaline','Hallucinogen','mL',20,220,297.00),

(88,'MDMA (Ecstasy/Molly)','Hallucinogen','mL',20,370,425.00),

(89,'Cyclophosphamide (Cytoxan)','Cytotoxic drugs','mL',20,140,525.00),

(90,'Doxorubicin (Adriamycin)','Cytotoxic drugs','mL',20,240,55.00),

(91,'Cisplatin','Cytotoxic drugs','mL',20,130,185.00),

(92,'Methotrexate','Cytotoxic drugs','mL',20,190,265.00),

(93,'Ibuprofen (Advil, Motrin)','Nonsteroidal anti-inflammatory drugs','mL',18.5,750,50.00),

(94,'Naproxen (Aleve, Naprosyn)','Nonsteroidal anti-inflammatory drugs','mL',18.5,230,250.00),

(95,'Celecoxib (Celebrex)','Nonsteroidal anti-inflammatory drugs','mL',18.5,150,520.00),

(96,'Ketoprofen (Orudis, Oruvail)','Nonsteroidal anti-inflammatory drugs','mL',18.5,170,510.00),

(97,'Mefenamic Acid (Ponstel)','Nonsteroidal anti-inflammatory drugs','mL',18.5,250,1150.00),

(98,'Piroxicam (Feldene)','Nonsteroidal anti-inflammatory drugs','mL',18.5,255,550.00),

(99,'Bandage','General','m',18,320,85.00),

(100,'Band-Aid','General','qty',18,320,15.00),

(101,'Tape','General','cm',18,120,20.50),

(102,'Cotton Buds','General','kg',18,120,4.99),

(103,'Surgical Masks','General','qty',18,360,10.00),

(104,'Zolgensma','GENE THERAPY DRUGS','Others...',12,5,175000000),

(105,'Luxturna (voretigene neparvovec)','GENE THERAPY DRUGS','Others...',12,6,70000000)

"""

cursor.execute(sql)

tkmsgbox.showinfo("CONGRATS!", "Drug table created with a lot of test data.", parent=frame)

#------------------------------------------------

tbl = 'drugcategory'

try:

cursor.execute('DROP TABLE drugcategory')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS drugcategory(

drugcategoryid INT PRIMARY KEY AUTO\_INCREMENT,

drugcategory VARCHAR(30) UNIQUE,

gstrate decimal(10,2)

)"""

cursor.execute(sql)

sql = """INSERT INTO drugcategory

(drugcategory,gstrate)

VALUES

('General',18),

('Analgesics',20),

('Antidepressants',18),

('Drug',18),

('Antibacterials',12),

('Antifungals',12),

('Antivirals',12),

('Antibiotics',18),

('Barbiturates',18),

('Controlled drugs',18),

('General anesthetics',12),

('Opioid',20),

('Stimulant',18.5),

('Antiarrhythmics',19.5),

('Antidiarrheals',20),

('Antihistamines',12),

('Cannabinoids',18),

('Cholinesterase inhibitors',12),

('Inhalant',18),

('Hallucinogen',20),

('Cytotoxic drugs',20),

('Nonsteroidal anti-inflammatory drugs',18.5),

('GENE THERAPY DRUGS',12)

"""

cursor.execute(sql)

tkmsgbox.showinfo("KUDOS!", "drugcategory table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'employee'

try:

cursor.execute('DROP TABLE employee')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS employee(

employeeid INT PRIMARY KEY AUTO\_INCREMENT,

employeename VARCHAR(30) UNIQUE,

contact VARCHAR(30),

agedob date,

employeeaddress varchar(50),

employeecity VARCHAR(30),

employeestate VARCHAR(30),

country VARCHAR(30),

salaryrangePA DECIMAL(10,3),

employeecategory VARCHAR(30),

gender varchar(30)

)"""

cursor.execute(sql)

sql = """INSERT INTO employee

(employeeid,employeename,contact,agedob,employeeaddress,employeecity,employeestate,country,salaryrangePA,employeecategory,gender)

VALUES

('1', 'Ronak Arora','+91 9173544835',19921226,'Sector 09 Flat 4B Gokuldham Society','Patna','Bihar','India',300000.99,'Sweeper','F'),

('2', 'Charu','+91 9273569835',20001204,'Sector 08 Flat 4A Meragaow Society','Rai Bareilly','Uttar Pradesh','India',480000.01,'Keeper','F'),

('3', 'Sonika Chauhan','+91 9173546835',19990101,'Sector 08 Flat 3B Wild Crest','Vidisha','Madhya Pradesh','India',544000.00,'Keeper','F'),

('4', 'Ajeet Rana','+91 9473566835',20010202,'Sector 06 Flat 1C World Green Society','Dispur','Assam','India',696969.69,'Keeper','M'),

('5', 'Ananya','+91 9173596835',20040404,'Sector 04 Flat 5Z Protestant Society','Nainital','Uttarakhand','India',6500000.00,'Owner','F'),

('6', 'XYZ Maksad','+91 8173569695',20001209,'Sector 03 Flat 2F Revolutionary Societyy','New Delhi','Delhi','India',800000.66,'Miscellaneous','F'),

('7', 'Mulayam Yadav','+91 8273569695',20001109,'Sector 03 Flat 9J Madanmohan Society','Mumbai','Maharashtra','India',200000.66,'Keeper','M'),

('8', 'Govind Singh','+91 8374499695',19941009,'Mall Road Calisthenics Society','Bengaluru','Karnataka','India',100000.66,'Keeper','M'),

('9', 'Maarkaat Nochphaad','+91 9153015695',20000902,'Sector 02 Flat 1K Catholics Society','Jaipur','Rajasthan','India',300000.66,'Keeper','M'),

('10', 'Naagin Itchchadhari','+91 8178452395',20000906,'GB Road Secret Underworld Society','Dehradun','Uttarakhand','India',500000.66,'Keeper','F'),

('11', 'BaalVeer Season 2','+91 7645199695',19991201,'Sector 01 Flat 4S Preist Society','Nanakmatta','Uttarakhand','India',600000.66,'Keeper','M')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "employee table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'employeecategory'

try:

cursor.execute('DROP TABLE employeecategory')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS employeecategory(

employeecategoryid INT PRIMARY KEY AUTO\_INCREMENT,

employeecategory VARCHAR(30) UNIQUE

)"""

cursor.execute(sql)

sql = """INSERT INTO employeecategory

(employeecategory)

VALUES

('Owner'),

('Manager'),

('Sweeper'),

('Keeper'),

('Miscellaneous')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "employeecategory table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'supplier'

try:

cursor.execute('DROP TABLE supplier')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS supplier(

supplierid INT PRIMARY KEY AUTO\_INCREMENT,

suppliercode VARCHAR(30) UNIQUE,

suppliername VARCHAR(30),

suppliermobile VARCHAR(15),

suppliercategory VARCHAR(30)

)"""

cursor.execute(sql)

sql = """INSERT INTO supplier

(suppliercode,suppliername,suppliercategory,suppliermobile)

VALUES

('A','Ria Lifesciences Pvt Ltd.','Manufacturers','9411107750'),

('B','Johnson & Johnson','Manufacturers','8211107750'),

('C','Novartis','Manufacturers','8172107750'),

('D','Roche','Manufacturers','5135607750'),

('E','Merck & Co., Inc. (known as MSD outside the U.S. and Canada)','Manufacturers','9371107750'),

('F','Cardinal Health','Wholesalers','8217612569'),

('G','McKesson Corporation','Wholesalers','7163612569'),

('H','AmerisourceBergen','Wholesalers','7272612569'),

('I','Cencora (formerly known as Alliance Healthcare)','Wholesalers','8162512569'),

('J','H.D. Smith','Wholesalers','7117612569'),

('K','Mezorays Pharma','Wholesalers','7272189011'),

('L','Sandoz (a Novartis division)','Exporters','7897262490'),

('M','Teva Pharmaceuticals','Exporters','8291235490'),

('N','Sun Pharmaceutical Industries','Exporters','8162235490'),

('O','Nexford Healthcare Pvt Ltd.','Exporters','8161235490'),

('P','Mylan (now part of Viatris)','Exporters','5226235493'),

('Q','CVS Health','Importers','9412189928'),

('R','Walgreens Boots Alliance','Importers','7412189011'),

('S','Wal-Mart','Importers','9412189075'),

('T','Mezorays Pharma','Importers','8172789011'),

('U','Rite Aid','Importers','7262618901'),

('V','Thermo Fisher Scientific','Others...','8272636627'),

('W','Charles River Laboratories','Others...','7262636278'),

('X','WuXi AppTec','Others...','9173638227'),

('Y','Labcorp','Others...','7183363937'),

('Z','Syneos Health','Others...','9172737283')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "supplier table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'suppliercategory'

try:

cursor.execute('DROP TABLE suppliercategory')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS suppliercategory(

suppliercategoryid INT PRIMARY KEY AUTO\_INCREMENT,

suppliercategory VARCHAR(30) UNIQUE

)"""

cursor.execute(sql)

sql = """INSERT INTO suppliercategory

(suppliercategory)

VALUES

('Wholesalers'),

('Manufacturers'),

('Importers'),

('Exporters'),

('Others...')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "suppliercategory table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'gender'

try:

cursor.execute('DROP TABLE gender')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS gender(

genderid INT PRIMARY KEY AUTO\_INCREMENT,

gender VARCHAR(30) UNIQUE

)"""

cursor.execute(sql)

sql = """INSERT INTO gender

(gender)

VALUES

('M'),

('F'),

('Transgender Male'),

('Transgender Female'),

('Non-Binary'),

('Genderqueer'),

('Genderfluid'),

('Agender'),

('Bigender'),

('Pangender'),

('Lesbian'),

('Androgynous'),

('Two-Spirit'),

('Demiboy'),

('Demigirl'),

('Neutrois'),

('Gay'),

('Xenogender'),

('Krigar'),

('Intergender'),

('Polygender'),

('Omnigender'),

('Bisexual'),

('Third Gender'),

('Cisgender'),

('Graygender'),

('Autigender'),

('Aliagender'),

('Maverique'),

('Novigender'),

('Gendervoid'),

('Trixic'),

('Trixic/Polygender'),

('Feminine of Center'),

('Masculine of Center'),

('Multigender'),

('Fluidgender'),

('Gender Apathetic'),

('Gender Non-Conforming'),

('Gender Questioning'),

('Boi'),

('Gendermon'),

('Genderless'),

('Genderflux'),

('Demiflux'),

('Demigender'),

('Androgyne'),

('Intersex'),

('Gendervague'),

('Queer')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "gender table created with nice data.", parent=frame)

#------------------------------------------------

tbl = 'country'

try:

cursor.execute('DROP TABLE country')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS country(

countyryid INT PRIMARY KEY AUTO\_INCREMENT,

country varchar(50) UNIQUE

)"""

cursor.execute(sql)

sql = """INSERT INTO country

(country)

VALUES

('Afghanistan'),

('Albania'),

('Algeria'),

('Andorra'),

('Angola'),

('Antigua and Barbuda'),

('Argentina'),

('Armenia'),

('Australia'),

('Austria'),

('Azerbaijan'),

('Bahamas'),

('Bahrain'),

('Bangladesh'),

('Barbados'),

('Belarus'),

('Belgium'),

('Belize'),

('Benin'),

('Bhutan'),

('Bolivia'),

('Bosnia and Herzegovina'),

('Botswana'),

('Brazil'),

('Brunei'),

('Bulgaria'),

('Burkina Faso'),

('Burundi'),

('Cabo Verde'),

('Cambodia'),

('Cameroon'),

('Canada'),

('Central African Republic'),

('Chad'),

('Chile'),

('China'),

('Colombia'),

('Comoros'),

('Congo, Democratic Republic of the'),

('Congo, Republic of the'),

('Costa Rica'),

('Croatia'),

('Cuba'),

('Cyprus'),

('Czechia'),

('Denmark'),

('Djibouti'),

('Dominica'),

('Dominican Republic'),

('Ecuador'),

('Egypt'),

('El Salvador'),

('Equatorial Guinea'),

('Eritrea'),

('Estonia'),

('Eswatini'),

('Ethiopia'),

('Fiji'),

('Finland'),

('France'),

('Gabon'),

('Gambia'),

('Georgia'),

('Germany'),

('Ghana'),

('Greece'),

('Grenada'),

('Guatemala'),

('Guinea'),

('Guinea-Bissau'),

('Guyana'),

('Haiti'),

('Honduras'),

('Hungary'),

('Iceland'),

('India'),

('Indonesia'),

('Iran'),

('Iraq'),

('Ireland'),

('Israel'),

('Italy'),

('Jamaica'),

('Japan'),

('Jordan'),

('Kazakhstan'),

('Kenya'),

('Kiribati'),

('Korea, North'),

('Korea, South'),

('Kosovo'),

('Kuwait'),

('Kyrgyzstan'),

('Laos'),

('Latvia'),

('Lebanon'),

('Lesotho'),

('Liberia'),

('Libya'),

('Liechtenstein'),

('Lithuania'),

('Luxembourg'),

('Madagascar'),

('Malawi'),

('Malaysia'),

('Maldives'),

('Mali'),

('Malta'),

('Marshall Islands'),

('Mauritania'),

('Mauritius'),

('Mexico'),

('Micronesia'),

('Moldova'),

('Monaco'),

('Mongolia'),

('Montenegro'),

('Morocco'),

('Mozambique'),

('Myanmar'),

('Namibia'),

('Nauru'),

('Nepal'),

('Netherlands'),

('New Zealand'),

('Nicaragua'),

('Niger'),

('Nigeria'),

('North Macedonia'),

('Norway'),

('Oman'),

('Pakistan'),

('Palau'),

('Panama'),

('Papua New Guinea'),

('Paraguay'),

('Peru'),

('Philippines'),

('Poland'),

('Portugal'),

('Qatar'),

('Romania'),

('Russia'),

('Rwanda'),

('Saint Kitts and Nevis'),

('Saint Lucia'),

('Saint Vincent and the Grenadines'),

('Samoa'),

('San Marino'),

('Sao Tome and Principe'),

('Saudi Arabia'),

('Senegal'),

('Serbia'),

('Seychelles'),

('Sierra Leone'),

('Singapore'),

('Slovakia'),

('Slovenia'),

('Solomon Islands'),

('Somalia'),

('South Africa'),

('South Sudan'),

('Spain'),

('Sri Lanka'),

('Sudan'),

('Suriname'),

('Sweden'),

('Switzerland'),

('Syria'),

('Taiwan'),

('Tajikistan'),

('Tanzania'),

('Thailand'),

('Timor-Leste'),

('Turkey'),

('Turkmenistan'),

('Tuvalu'),

('Uganda'),

('Ukraine'),

('United Arab Emirates'),

('United Kingdom'),

('United States'),

('Uruguay'),

('Uzbekistan'),

('Vanuatu'),

('Vatican City'),

('Venezuela'),

('Vietnam'),

('Yemen'),

('Zambia'),

('Zimbabwe')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "Country table created with Proper data of 195 countries...", parent=frame)

#------------------------------------------------

tbl = 'unit'

try:

cursor.execute('DROP TABLE unit')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS unit(

unitid INT PRIMARY KEY AUTO\_INCREMENT,

unit varchar(30) UNIQUE

)"""

cursor.execute(sql)

sql = """INSERT INTO unit

(unit)

VALUES

('mL'),

('mg'),

('g'),

('L'),

('kg'),

('m'),

('cm'),

('qty'),

('Others...')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "unit table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'customer'

try:

cursor.execute('DROP TABLE customer')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS customer(

customerid INT PRIMARY KEY AUTO\_INCREMENT,

customercode VARCHAR(10),

customername VARCHAR(60) UNIQUE,

customeraddress varchar(50),

customercity VARCHAR(30),

customerstate VARCHAR(30),

customeremail VARCHAR(30),

customermobile VARCHAR(16),

purchases varchar(60),

customercategory VARCHAR(30),

gender varchar(30)

)"""

cursor.execute(sql)

sql = """INSERT INTO customer

(customercode,customername,customermobile,customeraddress,customercity,customerstate,customeremail,purchases,customercategory,gender)

VALUES

('A','Ria Malhotra','+91-9365453385',"Sisona, Rajput Nagar, Bareilly",'Nagpur','Uttarakhand','uilote@birlavidyamandir.com','9 \* Saridon, 4 \* Heat Bag','REGULAR','F'),

('B','Shyam Guldastewala','+91-9369997345',"Bilona, Gamjam Nagar, Rai bareilly",'Sitarpur','Jaipur','lweigm@gmail.com','2 \* Surgucal Mask, 2 \* Corex','REGULAR','M'),

('C','Girpade Gudarphate','+91-7834643753',"Gajona, Gadarpham Nagar, Hai bareilly",'Gajiapur','Telangana','shflque@gmail.com','1 \* Torex, 2 \* Corex','REGULAR','M'),

('D','Madhuri Dixit','+91-8984357845',"Gadarphona, Gamjam Nagar, Rai bareilly",'Sitarpur','Uttar Pradesh','lweikm@gmail.com','5 \* Chloroform, 2 \* Zhandu Balm Red','REGULAR','F'),

('E','Customer Special','+91-2984677845',"Gadarpur, Rajput Nagar, Kanpur",'Shahjahanpur','Uttar Pradesh','ldfsjhjm@yahoo.com','1 \* Paracetamol, 3 \* Cofsils','REGULAR','M'),

('F','Malhotra Sahab','+91-8952678845',"Malaipur, Anaconda Nagar, Gai bareilly",'Hamjampur','Uttar Pradesh','fij@gmail.com','1 \* Paracetamol, 2 \* Tofsils','REGULAR','M'),

('G','Sahib-E-Alam','+91-1234357845',"Doodhnagar, Malai Nagar, Zai Shillong",'Sitarpur','Rajasthan','qphfjm@yahoo.com','6 \* Contact lens solution, 5 \* Cleaning Cloth','REGULAR','M'),

('H','Alfahad Malik','+91-5678357845',"Manipur, Rohit Nagar, Pilibhit",'Sitarganj','Madhya Pradesh','tgrfwjm@gmail.com','1 \* Opium, 1 \* Drug','OTHERS...','F'),

('I','Pulkit Pokhral','+91-9101357845',"Patnapur, Mithu Nagar, Satellite",'Meenapur','Arunachal Pradesh','hnytrijm@hotmail.com','10 \* Indigo, 22 \* Opium','INTERDICTED','M'),

('J','Ankit Tiwari','+91-1314357845',"Rajapur, Gethiya Nagar, Space Station",'Mainpur','Andhra Pradesh','aeaaaaaaijm@gmail.com','1 \* N-95 Mask, 6 \* Surgical Masks','EMPLOYEE','F'),

('L','Prajjwal Rathore','+91-4465357845',"Gajapur, Banyan Nagar, Turka Tisor",'Gajapur','Telangana','jjjjweijm@hotmail.com','4 \* Saridon, 2 \* Chloroform','REGULAR','M'),

('M','Saina Rathore','+91-1258795684',"Hodapur, Baliyan Nagar, Mantriscity",'Hodapur','Telangana','yujo@gmail.com','4 \* Abc, 1 \* Diclophenac','REGULAR','M'),

('N','Merry Jilli','+91-3598757848',"Pointpur, Galiyan, Hellomyf",'Pointpur','Rajasthan','etyhnjuny@gmail.com','1 \* Xyz, 3 \* Iodoform','REGULAR','M'),

('O','Terry Lilli','+91-1654584654',"Seduntant, Helios Nagar, Turka Tisor",'Koinpur','Uttarakhand','ewrtgrg@gmail.com','2 \* Pqr, 2 \* Norepinephrine','REGULAR','F'),

('P','Cannot Reveal Name','+XX-XXXXXXXXXX','Polypur, Kondapur, Joint Furram','Ploypur','Uttarakhand','tgrfjy7j56@gmail.com','3 \* Stu, 2 \* Acetylcholine','VIP','M'),

('Q','Private XQZ','+XX-XXXXXXXXXX',"Mantispur, Lava Nagar, Mallitalpur",'Mantispur','JnK','6657h6@gmail.com','4 \* Vwx, 2 \* Stamina Pills','VIP','M'),

('R','Dolly Chai Wala','+91-6969696969',"Lofapur, Urmila Nagar, Logdgdugfepur",'Lofapur','Uttarakhand','7htyh6@gmail.com','6 \* Yza, 2 \* Nashe ki Dawa','REGULAR','M'),

('S','Pav Bhaji Wali','+91-6966996666',"Urfipur, Konda Pur, Place Nagar",'Urfipur','Bihar','gsdg5t5@gmail.com','4 \* Saridon, 2 \* Jkl','INTERDICTED','F'),

('T','IIT Momo Wala','+91-7544464466',"Anacondapur, Handersky Nagar, GG",'Anacondapur','Uttarakhand','ytj76u5@gmail.com','3 \* Mno, 2 \* Neend ki goli','REGULAR','F'),

('U','MIT Vada Pav Wala','+91-6589745821',"Dkpur, Kasperskyy Nagar, BpColony",'Dkpur','Uttarakhand','gtrgaek677@gmail.com','2 \* Fgo, 2 \* Khasi ki dawa','REGULAR','M'),

('V','UPSC Chai Wala','+91-3852496274',"Henrypur, HolyCow Nagar, Android Fallam",'Henrypur','Madhya Pradesh','gnhjjuyjh@hotmail.com','1 \* Yfe, 2 \* Positive Feedback Loop','REGULAR','M'),

('W','MBBS International Bhikhari','+23-5784962105',"Follyupur, Crap Nagar, Bareilly",'Follypur','Andhra Pradesh','esgtytjuyb@hotmail.com','5 \* Dawai, 2 \* Chloroform','INTERDICTED','M'),

('X','AIIMS Malai Chaap Wala','+91-6236549875',"Jhutpur, HolyCrap Nagar, Pbt.",'Jhutpur','Arunachal Pradesh','ertyhbgv@gmail.com','4 \* Hellew, 2 \* Xenon Poison','REGULAR','M'),

('Y','Yusuf Dadar Haveli','+91-7539516482',"Sachpur, Dorm Nagar, Stg.",'Sachpur','Uttarakhand','565555534yg@gmail.com','4 \* Medicine, 6 \* Bromoform','REGULAR','F'),

('Z','Melantina Getasdrool','+91-4563218972',"Kalapur, Room Nagar, Jaipur",'Kalapur','Uttar Pradesh','uuyjgye4@gmail.com','4 \* Drug, 2 \* Iodoform','REGULAR','F'),

('AA','Hon. Kirti Kamal Bhardwaj','+91-6498810023',"Leelapur, House Nagar, Leelapur",'Subsidiarypur','Uttarakhand','54rtvbh78@hotmail.com','4 \* Jailop, 2 \* Hentyop','VIP','M'),

('AB','Yadhirubhai Yambani','+91-6929012345',"Neelapur, Country Nagar, Neelapur",'Subsidiarypur','Uttarakhand','45trgreg@yahoo.com','3 \* Saridon, 2 \* Chloroform','VIP','M')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "customer table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'customercategory'

try:

cursor.execute('DROP TABLE customercategory')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS customercategory(

customercategoryid INT PRIMARY KEY AUTO\_INCREMENT,

customercategory VARCHAR(30) UNIQUE

)"""

cursor.execute(sql)

sql = """INSERT INTO customercategory

(customercategory)

VALUES

('VIP'),

('REGULAR'),

('INTERDICTED'),

('EMPLOYEE'),

('OTHERS...')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "customercategory table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'businessname'

try:

cursor.execute('DROP TABLE businessname')

except conn.Error as e:

pass

sql = """CREATE TABLE IF NOT EXISTS businessname(

businessnameid INT PRIMARY KEY AUTO\_INCREMENT,

businessname VARCHAR(30) UNIQUE

)"""

cursor.execute(sql)

sql = """INSERT INTO businessname

(businessname)

VALUES

('Opulent Pathway Pharmacy...')

"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "businessname table created with test data.", parent=frame)

#------------------------------------------------

'''tbl = 'purchase'

#try:

# cursor.execute('DROP TABLE purchase')

#except conn.Error as e:

# pass

sql = """CREATE TABLE IF NOT EXISTS purchase(

purchaseid INT PRIMARY KEY AUTO\_INCREMENT,

pdate DATE,

invoice VARCHAR(30),

suppliercategory VARCHAR(30),

suppliercode VARCHAR(30),

suppliername VARCHAR(30),

suppliermobile VARCHAR(30),

drugcode VARCHAR(30),

drugcategory VARCHAR(30),

drugname VARCHAR(30),

unit VARCHAR(30),

gstrate DECIMAL(5,2),

drugquantity DECIMAL(10,2),

drugstock DECIMAL(10,2),

drugprice DECIMAL(10,2),

amount DECIMAL(10,2),

freight DECIMAL(10,2),

gst DECIMAL(10,2),

netamount DECIMAL(10,2),

grandamount DECIMAL(10,2),

drugminsaleprice DECIMAL(10,2)

)"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "purchase table created with test data.", parent=frame)

#------------------------------------------------

tbl = 'sale'

#try:

# cursor.execute('DROP TABLE sale')

#except conn.Error as e:

# pass

sql = """CREATE TABLE IF NOT EXISTS sale(

saleid INT PRIMARY KEY AUTO\_INCREMENT,

sdate DATE,

invoice VARCHAR(30),

customercategory VARCHAR(30),

customercode VARCHAR(30),

customername VARCHAR(30),

customermobile VARCHAR(30),

drugcode VARCHAR(30),

drugname VARCHAR(30),

drugcategory VARCHAR(30),

unit VARCHAR(30),

gstrate DECIMAL(5,2),

drugquantity DECIMAL(10,2),

drugprice DECIMAL(10,2),

amount DECIMAL(10,2),

marginpercent DECIMAL(10,2),

gst DECIMAL(10,2),

netamount DECIMAL(10,2),

freight DECIMAL(10,2),

grandamount DECIMAL(10,2)

)"""

cursor.execute(sql)

tkmsgbox.showinfo("MESSAGE!", "sale table created with test data.", parent=frame)'''

#------------------------------------------------

tkmsgbox.showinfo("SUCCESS MESSAGE!","ALL tables created successfully with dummy values.", parent=frame)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", "TABLE: "+tbl+"\n"+msg, parent=frame)

#=============================================================================

# standalone start

#=============================================================================

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

createDatabaseFrame(rootframe)

#=============================================================================

\_dataAnalysisAndPlot.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#-----------------------------------------------------------------------------

# pandastable frame r=1 c=0

# colcheckboxes frame r=2 c=0

# insertDataInput frame r=0 c=1

#-----------------------------------------------------------------------------

global tablesCBO

global pandasTableFRM

global colCheckboxesFRM

global plotFRM

global statisticalDetailsFRM

#-----------------------------------------------------------------------------

def sqlQueryExecution(frame, sql): #all SQL queries except SELECT & DESC

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute(sql)

conn.commit()

msg = "SUCCESS: SQL query executed successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#return msg

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

#return msg

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def showTablesInADatabase(frame):

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute("SHOW TABLES")

result = cursor.fetchall() #result: list of dictionary

tables=[]

for i in result:

tables.append(\*i.values()) #use \* to explode the dictionary so as to enlist values only

return tables

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def descTable(frame, table):

try:

sql = "desc "+table

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall() #list of dict with one common key 'Field'

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def getPrimaryKeyColumn(frame, table):

#mydb = \_libraryAndDBConnection.mydatabase

mydb = mydatabase

try:

sql = "SELECT COLUMN\_NAME FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_SCHEMA = '" + mydb + \

"' AND TABLE\_NAME = '" + table + "' AND COLUMN\_KEY = 'PRI'"

#print(mydb, ' ',table, ' ',sql)

print("STARTED: ",mydb)

cursor.execute(sql)

result = cursor.fetchall() #result: list of dictionary

#print("result: ",result)

col = []

for d in result:

col.append(d['COLUMN\_NAME'])

#print(col)

return col

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def table2df(frame='', table='', columns='\*', condition='1=1'):

try:

sql = "select "+columns+" from "+table+" where "+condition

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

print(msg)

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def table2dfSQLQuery(frame='', sql=''):

try:

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def searchTableButtonClickEvent(frame,table,column,value):

try:

sql = "select \* from "+table+" where "+column+"='"+value+"'"

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def updateARow(rowDataList): #rowDataList is a series

try:

frameUpdateTable = Toplevel() #create a pop-up window

frameUpdateTable.title("Update Table")

frameUpdateTable.geometry()

frameUpdateTable.configure(background='orange')

r = 0

for colname in rowDataList.index:

#print(i,' -- ',rowDataList[colname])

Label(frameUpdateTable, text=colname, bg='yellow') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

e = Entry(frameUpdateTable)

e.grid(row=r, column=2, padx=15, pady=5)

e.insert(INSERT, rowDataList[colname])

r += 1

def updateTableSubmitButtonClickEvent():

pass

updateTableSubmitButton = Button(frameUpdateTable, text="Update", command=updateTableSubmitButtonClickEvent)

updateTableSubmitButton.grid(row=r, column=2, padx=10, pady=10)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frameUpdateTable)

#-----------------------------------------------------------------------------

def statisticalDetailsFRM(selectrootframe,table,df,r=1,c=1):

global tablesCBO

global plotFRM

global statisticalDetailsFRM

'''

print(table, df)

df = table2df(table)

'''

#sample data (head/tail)from dataframe

w, h = selectrootframe.winfo\_screenwidth(), selectrootframe.winfo\_screenheight()

h=h\*20/100

w=w\*30/100

#frame4 = Frame(selectrootframe, width=w\*40/100, bg='black') #!!!!!! selectrootframe and not any other frame

frame4 = Frame(selectrootframe, width=w, height=h)#, bg='black') #!!!!!! selectrootframe and not any other frame

frame4.grid(row=1, column=1, sticky='NW')

w, h = frame4.winfo\_screenwidth(), frame4.winfo\_screenheight()

table = tablesCBO.get()

#print(table)

df = table2df(frame4,table)

#print(df)

#subframe41 = Frame(frame4, width=w\*10/100, bg='red') #!!!!!! selectrootframe and not any other frame

subframe41 = Frame(frame4, width=w\*2/100) #, bg='red')#!!!!!! selectrootframe and not any other frame

subframe41.grid(row=0, column=0, sticky='NW')

#subframe42 = Frame(frame4, width=w\*40/100, bg='green') #!!!!!! selectrootframe and not any other frame

subframe42 = Frame(frame4, width=40)#, bg='green') #!!!!!! selectrootframe and not any other frame

subframe42.grid(row=0, column=1, sticky='NW')

#Label(subframe42, text='TABLES:').grid(row=0, column=0, sticky='NW')

d=df.describe().round(2)

d.reset\_index(level=0, inplace=True) #set index as a column in the dataframe for display in pt

pt = Table(subframe42, dataframe=d, showstatusbar=False) #width=w

pt.show()

#frameHeadingLBL = LabelFrame(frame4, text='frame4', font=('Times', 12), bd=5, relief=RIDGE, bg='red')

#r4=0

#Label(subframe41, text="SAMPLE DATAFRAME DATA").grid(row=r4, column=0, columnspan=2, sticky='NW')

r4 = 0

def sampleDataBTNClickEvent():

if df.empty==False:

numofrows = numofrowstxt.get('1.0','end-1c')

if varHeadTail.get()=='head':

if len(numofrows)>0:

d=df.head(int(numofrows))

else:

d=df.head()

else:

if len(numofrows)>0:

d=df.tail(int(numofrows))

else:

d=df.tail()

#displayPandasTable(subframe42, d)

#pt = Table(subframe42, dataframe=d, width=50, showstatusbar=False)

pt = Table(subframe42, dataframe=d, showstatusbar=False)

pt.show()

else:

tkmsgbox.showinfo("ATTENTION PLEASE!","The '"+cboTables.get().upper()+"' table is empty.",parent=frame4)

varHeadTail = StringVar()

values = {"Top Samples":"head","Bottom Samples":"tail"}

#loop to create multiple Radiobuttons - don't create each button separately

c=0

for (text, value) in values.items():

#rbt = Radiobutton(frame3, text=text, variable=var, value=value, command=sampleDataBtnClickEvent)

rbt = Radiobutton(subframe41, text=text, variable=varHeadTail, value=value, tristatevalue=0)

rbt.grid(row=r4, column=c, sticky='NW')

#c+=1

r4 += 1

Label(subframe41, text="No. of rows").grid(row=r4, column=0, sticky=W)

numofrowstxt = Text(subframe41, height=1, width=10)

numofrowstxt.grid(row=r4, column=1, sticky='NW')

r4 += 1

sampleDataBTN = Button(subframe41, text='SAMPLE DATAFRAME DATA', command=sampleDataBTNClickEvent)

sampleDataBTN.grid(row=r4, column=0, columnspan=2, sticky='NW')

r4 += 1

'''

def displayPandasTable(df):

#width=rootDA.winfo\_screenwidth()-50

global globalDF

globalDF = df

pt = Table(framePandasTable, dataframe=globalDF)

pt.show()

'''

def statDetailBTNClickEvent():

table = tablesCBO.get()

df = table2df(subframe41, table)

#df=df.describe()

d=df.describe().round(2)

#df['index']=df.index

d.reset\_index(level=0, inplace=True) #set index as a column in the dataframe for display in pt

#displayPandasTable(d)

#w, h = subframe42.winfo\_screenwidth()-120, subframe42.winfo\_screenheight()-160

#w = w\*70/100

#pt = Table(subframe42, dataframe=d, width=w, showstatusbar=False)

pt = Table(subframe42, dataframe=d, showstatusbar=False)

pt.show()

statDetailBTN = Button(subframe41, text='STATISTICAL DETAILS OF DF DATA', command=statDetailBTNClickEvent)

statDetailBTN.grid(row=r4, column=0, columnspan=2, sticky='NW')

for widget in subframe41.winfo\_children():

widget.grid(padx=2, pady=2)

for widget in subframe42.winfo\_children():

widget.grid(padx=2, pady=2)

'''

#fire tablescbo combobox click-event programmatically

idx = lookupvals.index(table)

tablesCBO.current(idx)

#print(idx, tablesCBO.current(idx), lookupvals[idx])

tablesCBO.event\_generate('<<ComboboxSelected>>')

'''

#sampleDataBTN.event\_generate('<<ComboboxSelected>>')

#statDetailBTN.event\_generate('<<ComboboxSelected>>')

#-----------------------------------------------------------------------------

def MatplotlibGUI(chartFRM, df, xcol, ycol, charttype):

#for widget in chartFRM.winfo\_children():

# widget.destroy()

import matplotlib

matplotlib.use('TkAgg')

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

plt.clf()

fig = plt.figure(1)

x = df[xcol].tolist()

y = df[ycol].tolist()

if charttype=='plot':

plt.plot(x,y)

elif charttype=='bar':

plt.bar(x,y)

#elif charttype=='plot':

# plt.hist(x,y)

canvas = FigureCanvasTkAgg(fig, master=chartFRM)

plot\_widget = canvas.get\_tk\_widget()

plot\_widget.grid(row=0, column=0)

#-----------------------------------------------------------------------------

#---------------------------------------

def createChartFRM(root, df, xcol, ycol, charttype):

global chartFRM

w, h = root.winfo\_screenwidth()-120, root.winfo\_screenheight()-160

w = w\*70/100

#h = h\*5/100

r = 0

chartFRM = Frame(root, width=w, bg='red')#, height=h)

chartFRM.grid(row=0, column=1, rowspan=2, sticky='NW')

chartLBL = Table(chartFRM, text="xxxx")

chartLBL.grid(row=r, column=0, sticky=W)

MatplotlibGUI(chartFRM, df, xcol, ycol, charttype)

#=============================================================================

def plottry():

x = [2012,2013,2014,2015,2016]

y = [45,56,23,78,42]

plt.bar(x,y)

plt.xlabel('X-Axis')

plt.ylabel('Y-Axis')

plt.xticks(x)

plt.yticks(np.arange(0,101,10))

plt.show()

#--------------------------------------------------------------------------

def plottry1(plotparams):

import matplotlib

matplotlib.use('TkAgg')

from matplotlib.figure import Figure

from matplotlib.backends.backend\_tkagg import (FigureCanvasTkAgg, NavigationToolbar2Tk)

plotFRM = Toplevel()

table = plotparams['table']

df = table2df(plotFRM,table)

x = df[plotparams['x']].tolist()

y = df[plotparams['y']].tolist()

title = plotparams['title']

#xticks = plotdata['xticks']

#yticks = plotdata['yticks']

xlabel = plotparams['xlabel']

ylabel = plotparams['ylabel']

legend = plotparams['legend']

grid = plotparams['grid']

charttype = plotparams['charttype']

figure = Figure(figsize=(6, 4), dpi=100)

figure\_canvas = FigureCanvasTkAgg(figure, plotFRM)

NavigationToolbar2Tk(figure\_canvas, plotFRM)

axes = figure.add\_subplot()

if charttype=='plot':

axes.plot(x,y)

elif charttype=='bar':

axes.bar(x,y)

elif charttype=='hist':

mybin = plotparams['mybin']

myrange =plotparams['myrange']

axes.hist(data=x, bin=mybin, range=myrange)

axes.set\_title(title)

axes.set\_xlabel(xlabel)

axes.set\_ylabel(ylabel)

figure\_canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

'''

plt.title = plotparams['title']

#plt.xticks = plotdata['xticks']

#plt.yticks = plotdata['yticks']

plt.xlabel = plotparams['xlabel']

plt.ylabel = plotparams['ylabel']

plt.legend = plotparams['legend']

plt.grid = plotparams['grid']

plt.show()

'''

#=============================================================================

def createPlotFRM(frame, plotparams):

global plotFRM

#import ctypes

#ctypes.windll.shcore.SetProcessDpiAwareness(1)

'''

import win32con, win32gui, win32print

def get\_dpi():

hDC = win32gui.GetDC(0)

HORZRES = win32print.GetDeviceCaps(hDC, win32con.DESKTOPHORZRES)

VERTRES = win32print.GetDeviceCaps(hDC, win32con.DESKTOPVERTRES)

return HORZRES,VERTRES

wp,hp = get\_dpi()

'''

#global plotFRM

plotFRM = Toplevel()

plotFRM.title('My Plot')

wp, hp = frame.winfo\_screenwidth()-20, frame.winfo\_screenheight()-100

'''

import win32con, win32gui, win32print

def get\_dpi():

hDC = win32gui.GetDC(0)

HORZRES = win32print.GetDeviceCaps(hDC, win32con.DESKTOPHORZRES)

VERTRES = win32print.GetDeviceCaps(hDC, win32con.DESKTOPVERTRES)

return HORZRES,VERTRES

wp,hp = get\_dpi()

wp = wp\*40/100

hp = hp\*40/100

'''

plotFRM.geometry("%dx%d+0+0" % (wp, hp))

plotFRM.configure(background="#88cffa") #"light grey"

#print(plotFRM.winfo\_width(), plotFRM.winfo\_height())

#frame.tk.call('tk', 'scaling', 1.25) #Zoom out 2 times

######## --------- PLOTTING ON DEFAULT MATPLOTLIB POPUP WINDOW

table = plotparams['table']

df = table2df(frame,table)

x = df[plotparams['x']].tolist()

y = df[plotparams['y']].tolist()

title = plotparams['title']

#xticks = plotdata['xticks']

#yticks = plotdata['yticks']

xlabel = plotparams['xlabel']

ylabel = plotparams['ylabel']

legend = plotparams['legend']

grid = plotparams['grid']

charttype = plotparams['charttype']

if charttype=='plot':

plt.plot(x,y)

elif charttype=='bar':

plt.bar(x,y)

elif charttype=='hist':

mybin = plotparams['mybin']

myrange =plotparams['myrange']

plt.hist(data=x, bin=mybin, range=myrange)

plt.title = plotparams['title']

#plt.xticks = plotdata['xticks']

#plt.yticks = plotdata['yticks']

plt.xlabel = plotparams['xlabel']

plt.ylabel = plotparams['ylabel']

plt.legend = plotparams['legend']

plt.grid = plotparams['grid']

plt.show()

'''

######## --------- PLOTTING ON A FRAME

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import matplotlib

from matplotlib.figure import Figure

matplotlib.use("TkAgg")

table = plotparams['table']

df = table2df(frame,table)

x = df[plotparams['x']].tolist()

y = df[plotparams['y']].tolist()

charttype = plotparams['charttype']

plt.clf()

fig = plt.figure(1)

if charttype=='plot':

plt.plot(x,y)

elif charttype=='bar':

plt.bar(x,y)

#elif charttype=='hist':

# plt.hist(x,y)

canvas = FigureCanvasTkAgg(fig, master=plotFRM)

plot\_widget = canvas.get\_tk\_widget()

plot\_widget.grid(row=0, column=0)

def onsize(event):

print(plotFRM.winfo\_width(), plotFRM.winfo\_height())

plotFRM.bind("<Configure>", onsize)

'''

#=============================================================================

def createColumnCheckBoxesFRM(selectrootframe, table, df, r=1, col=1):

global tablesCBO

global colCheckboxesFRM

w, h = selectrootframe.winfo\_screenwidth()-120, selectrootframe.winfo\_screenheight()-160

w = w\*20/100

numOfCols = len(df.columns.tolist())

colCheckboxesFRM = Frame(selectrootframe, width=w)

#colCheckboxesFRM.grid(row=r, column=0, columnspan=numOfCols, sticky='NW', padx=15, pady=15) #NW-top left

colCheckboxesFRM.grid(row=r, column=col, sticky='NW', padx=15, pady=15) #NW-top left

r = 1

col = 0

for c in df.columns.tolist():

globals()[c+'VAR'] = IntVar()

globals()[c+'VAR'].set(0)

globals()[c+'CHK'] = ttk.Checkbutton(colCheckboxesFRM,variable=globals()[c+'VAR'],text=c, onvalue=1, offvalue=0)

globals()[c+'CHK'].grid(column=col, row=r, sticky='W')

#col+=1

r+=1

r+=1

def getCheckedBoxesButtonClickEvent(df):

checkedColumnsLST = []

for c in df.columns.tolist():

if globals()[c+'VAR'].get()==1:

checkedColumnsLST.append(c)

cols=''

for c in checkedColumnsLST:

cols += c + ","

cols = cols[:-1]

df = table2df(colCheckboxesFRM, table, cols)

createPandasTableFRM(selectrootframe, df) #!IMPORTANT - use selectrootframe

getCheckedBoxesButton = Button(colCheckboxesFRM, text='Submit List of Selected Columns', \

command=lambda: getCheckedBoxesButtonClickEvent(df))

getCheckedBoxesButton.grid(row=r, column=0, columnspan=numOfCols, sticky='EW')

for widget in colCheckboxesFRM.winfo\_children():

widget.grid(padx=0, pady=3)

#=============================================================================

def createPandasTableFRM(frame, df, r=0, c=1):

global pandasTableFRM

w, h = frame.winfo\_screenwidth()-120, frame.winfo\_screenheight()-160

w = w\*50/100

h = h\*40/100

pandasTableFRM = Frame(frame, height=h, width=w)

pandasTableFRM.grid(row=r, column=c, sticky='NW')

#pt = Table(pandasTableFRM, dataframe=df, showtoolbar=True, width=w, showstatusbar=True)

pt = Table(pandasTableFRM, dataframe=df, width=w, height=h, showstatusbar=True)

pt.cellbackgr = 'yellow'

pt.grid()

pt.show()

#for insert disable the following event handling; enable it for update only

def leftButtonClickEvent(event): #left-button click event handling

rowclicked = pt.get\_row\_clicked(event)

rowDataList = pt.model.df.loc[rowclicked] #Series

updateARow(rowDataList)

pt.rowheader.bind('<Button-1>',leftButtonClickEvent)

#=============================================================================

def inputFrame(selectrootframe, param, r=0, c=0):

global tablesCBO

global colCheckboxesFRM

global pandasTableFRM

table = param['table'][0]

pk = param['pk'][0]

cbo = param['cbo'][0]

w, h = selectrootframe.winfo\_screenwidth()-120, selectrootframe.winfo\_screenheight()-160

w = w\*30/100

#frame = Frame(selectrootframe, height=h, width=w)

frame = Frame(selectrootframe, width=w)

frame.grid(row=r, column=c, rowspan=2, sticky='NW')#W-left, E-right, N-top, S-bottom

#frame.tk.call('tk', 'scaling', 1.25) #Zoom out 1.25 times which is equivalent of 125% default display on Windows10

df = table2df(frame,table)

dfColumns = descTable(frame,table)

r=0

Label(frame, text='TABLES:').grid(row=r, column=0, sticky='NW')

tables = StringVar()

tablesCBO = Combobox(frame, name='tablecbo', width=30)#, textvariable=tablesCBOvar) #,width=30)

lookupvals = showTablesInADatabase(frame)

tablesCBO['values'] = lookupvals

tablesCBO.grid(row=r, column=1, sticky='NW')

idx = lookupvals.index(table)

tablesCBO.current(idx)

df = table2df(frame,lookupvals[0])

dfColumns = descTable(frame,lookupvals[0])

'''

if len(lookupvals)>0:

tablesCBO.current(0)

df = table2df(frame,lookupvals[0])

dfColumns = descTable(frame,lookupvals[0])

'''

def tablesCBOSelectedEvent(event):

# !Important - use "selectrootframe" frame and not this "frame" #pass row and col for child frames

table = event.widget.get()

pk = getPrimaryKeyColumn(frame, table)

#print(table, event.widget.\_name)

df = table2df(frame,table)

columnFindCBO['values'] = df.columns.tolist()

xcolCBO['values'] = df.columns.tolist()

ycolCBO['values'] = df.columns.tolist()

dfColumns = descTable(selectrootframe,table)

if pkTXT.winfo\_exists()== 1:

#pkTXT.delete('1.0', END)

#pkTXT.delete()

pkTXT.delete(0, END)

pkTXT.insert(INSERT,"")

pkTXT.insert(INSERT,pk)

if pandasTableFRM.winfo\_exists()== 1:

pandasTableFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

if colCheckboxesFRM.winfo\_exists()== 1:

colCheckboxesFRM.destroy()

createColumnCheckBoxesFRM(selectrootframe, table, df, 0, 2)

tablesCBO.bind("<<ComboboxSelected>>", tablesCBOSelectedEvent)

r += 1

Label(frame, text='PRIMARY KEY:').grid(row=r, column=0, sticky='NW')

pkTXT = Entry(frame, width=30)

pkTXT.grid(row=r, column=1, sticky='NW')

#pkTXT.delete('1.0', END)

pkTXT.delete(0,END) #for Entry (0,END)

pkTXT.insert(INSERT,pk)

'''

Label(frame, text=pk).grid(row=r, column=1, sticky='NW')

r += 1

Label(frame, text='LOOKUP VALUES (CBO):').grid(row=r, column=0, sticky='NW')

Label(frame, text=pk).grid(row=r, column=1, sticky='NW')

'''

r += 1

#sub frame #1

frame1 = Frame(frame, bg="green")

frame1.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

# search / Find What

r1=0

Label(frame1, text='Find what:').grid(row=r1, column=0, sticky='NW')

findWhatTXT = Entry(frame1, width=30)

findWhatTXT.grid(row=r1, column=1, sticky='NW')

r1 += 1

Label(frame1, text='compare').grid(row=r1, column=0, sticky='NW')

operatorfind = StringVar()

operatorFindCBO = Combobox(frame1, name='operatorFindCBO', width=10, textvariable=operatorfind)

operatorFindCBO['values'] = ['=','<>','>','<','>=','<=','between','like','in']

operatorFindCBO.grid(row=r1, column=1, sticky='NW')

operatorFindCBO.current(0)

r1 += 1

Label(frame1, text='in column').grid(row=r1, column=0, sticky='NW')

columnfind = StringVar()

columnFindCBO = Combobox(frame1, name='columnFindCBO', width=30, textvariable=columnfind)

#columnFindCBO['values'] = lookupvals

columnFindCBO.grid(row=r1, column=1, sticky='NW')

r1 += 1

def searchTableButtonClickEventStart():

table = tablesCBO.get()

column = columnFindCBO.get()

value = findWhatTXT.get()

df = searchTableButtonClickEvent(frame1,table,column,value)

createPandasTableFRM(selectrootframe, df)

'''

df = table2dfSQLQuery(frame2, sql)

pandasTableFRM.destroy()

colCheckboxesFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

createColumnCheckBoxesFRM(selectrootframe, table, df, 1, 1)

df = table2dfSQLQuery(frame2, sql)

pandasTableFRM.destroy()

colCheckboxesFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

createColumnCheckBoxesFRM(selectrootframe, table, df, 1, 1)

'''

searchTableButton = Button(frame1, text='Search Table', width=20, command=searchTableButtonClickEventStart)

searchTableButton.grid(row=r1, column=1, sticky='EW')

r += 1

#sub frame #2

# General SQL Query Execution

frame2 = Frame(frame)

frame2.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

Label(frame2, text='Raw SQL Query (Select only):').grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

sqlQueryTXT = Text(frame2, width=35, height=6)

sqlQueryTXT.grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

def sqlQueryExecutionButtonClickEventStart():

#####For SELECT query

#for sql query use raw string

sql = sqlQueryTXT.get("1.0",'end-1c') #for TEXT use - textbox1.get("1.0",'end-1c') #For entry txt.get()

#sqlQueryExecution(frame2, sql)

#print("SQL: ",sql)

#TO GET THE TABLE NAME FROM A SQL QUERY!!!!!!!!!!!!!!!

#sql="select \* from item where a=1";

idx1 = sql.find("from");

idx2 = sql[idx1+4:].strip().find(" ");

idx3 = sql.find(" ",idx1+4+idx2-1);

idx4=sql[idx1+4+idx2-1:].strip().find(" ");

table = sql[idx1+4+idx2-1:idx3]

#print(table)

df = table2dfSQLQuery(frame2, sql)

pandasTableFRM.destroy()

colCheckboxesFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

createColumnCheckBoxesFRM(selectrootframe, table, df, 0, 2)

'''

#for queries other than SELECT

sql = sqlQueryTXT.get("1.0",'end-1c') #for TEXT use - textbox1.get("1.0",'end-1c') #For entry txt.get()

sqlQueryExecution(frame, sql)

'''

sqlQueryExecutionButton = Button(frame2, text='SQL Query Execution', width=20, command=sqlQueryExecutionButtonClickEventStart)

sqlQueryExecutionButton.grid(row=r, column=1, sticky='EW')

r += 1

#sub frame #3

# Plot Settings

frame3 = Frame(frame)

frame3.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

r2 = 0

Label(frame3, text='Plot Settings:').grid(row=r2, column=0, columnspan=2, sticky='NW')

'''

r2 += 1

Label(frame3, text='Plot Settings:').grid(row=r2, column=0, columnspan=2, sticky='NW')

sqlQueryTXT = Text(frame3, width=35, height=6)

sqlQueryTXT.grid(row=r2, column=0, columnspan=2, sticky='NW')

'''

r2 += 1

Label(frame3, text='X-Column').grid(row=r2, column=0, sticky='NW')

xcol = StringVar()

xcolCBO = Combobox(frame3, name='xcolcbo', width=30, textvariable=xcol)

xcolCBO.grid(row=r2, column=1, sticky='NW')

#xcolCBO.current(0)

r2 += 1

Label(frame3, text='Y-Column').grid(row=r2, column=0, sticky='NW')

ycol = StringVar()

ycolCBO = Combobox(frame3, name='ycolcbo', width=30, textvariable=ycol)

ycolCBO.grid(row=r2, column=1, sticky='NW')

#ycolCBO.current(0)

r2 += 1

Label(frame3, text='Title:').grid(row=r2, column=0, sticky='NW')

titleTXT = Entry(frame3, width=30)

titleTXT.grid(row=r2, column=1, sticky='NW')

r2 += 1

Label(frame3, text='X-Label:').grid(row=r2, column=0, sticky='NW')

xlabelTXT = Entry(frame3, width=30)

xlabelTXT.grid(row=r2, column=1, sticky='NW')

r2 += 1

Label(frame3, text='Y-Label:').grid(row=r2, column=0, sticky='NW')

ylabelTXT = Entry(frame3, width=30)

ylabelTXT.grid(row=r2, column=1, sticky='NW')

r2 += 1

Label(frame3, text='Legend:').grid(row=r2, column=0, sticky='NW')

legendTXT = Entry(frame3, width=30)

legendTXT.grid(row=r2, column=1, sticky='NW')

r2 += 1

Label(frame3, text='Chart Type:').grid(row=r2, column=0, sticky='NW')

chartTypeVar = StringVar()

chartTypeCBO = Combobox(frame3, name='charttype', width=30, textvariable=chartTypeVar)

chartTypeCBO.grid(row=r2, column=1, sticky='NW')

chartTypeCBO['values'] = ['plot','bar','hist']

r2 += 1

Label(frame3, text='Bin (for HIST):').grid(row=r2, column=0, sticky='NW')

binTXT = Entry(frame3, width=30)

binTXT.grid(row=r2, column=1, sticky='NW')

r2 += 1

Label(frame3, text='Range (for HIST):').grid(row=r2, column=0, sticky='NW')

rangeTXT = Entry(frame3, width=30)

rangeTXT.grid(row=r2, column=1, sticky='NW')

r2 += 1

Label(frame3, text='Grid').grid(row=r2, column=0, sticky='NW')

gridvar = StringVar()

gridCBO = Combobox(frame3, name='grid', width=30, textvariable=gridvar)

gridCBO.grid(row=r2, column=1, sticky='NW')

gridCBO['values'] = ['True','False']

#gridCBO.current(0)

r2 += 1

def drawPlotEvent():

#global tablesCBO

table = tablesCBO.get()

x = xcolCBO.get()

y = ycolCBO.get()

title = titleTXT.get()

#xticks = plotdata['xticks']

#yticks = plotdata['yticks']

xlabel = xlabelTXT.get()

ylabel = ylabelTXT.get()

legend = legendTXT.get()

grid = gridCBO.get()

charttype = chartTypeCBO.get()

mybin = legendTXT.get()

myrange = legendTXT.get()

#print(table)

plotparams = {'table':table,'x':x,'y':y,'title':title,'xlabel':xlabel,'ylabel':ylabel,'legend':legend,

'grid':grid,'charttype':charttype,'mybin':mybin,'myrange':myrange}

#createPlotFRM(selectrootframe, plotparams) #!!!!!!selectrootframe and frame

#createPlotFRM(frame, plotparams) #!!!!!!selectrootframe and frame

plottry1(plotparams)

drawPlotBTN = Button(frame3, text='Draw Plot', width=20, command=drawPlotEvent)

#drawPlotBTN = Button(frame3, text='Draw Plot', width=20, command=plottry1)

drawPlotBTN.grid(row=r2, column=1, sticky='EW')

#----------------------------------------

r += 1

#sub frame #5

frame5 = Frame(frame)

frame5.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

r4 = 0

def savecsvfile():

#df=createDataFrame(cboTables.get())

filetypes = [('CSV Files','\*.csv')]

saveatfilepath = asksaveasfile(mode='w', filetypes = filetypes, defaultextension=filetypes)

df.to\_csv(saveatfilepath)

Button(frame5, text='Save DF as CSV file', command=lambda:savecsvfile()).grid(row=r, column=1, sticky='W')

#----------------------------------------

for widget in frame.winfo\_children():

widget.grid(padx=10, pady=2)

for widget in frame1.winfo\_children():

widget.grid(padx=10, pady=2)

for widget in frame2.winfo\_children():

widget.grid(padx=10, pady=2)

for widget in frame3.winfo\_children():

widget.grid(padx=10, pady=2)

#----------------------------------------

#fire tablescbo combobox click-event programmatically

idx = lookupvals.index(table)

tablesCBO.current(idx)

#print(idx, tablesCBO.current(idx), lookupvals[idx])

tablesCBO.event\_generate('<<ComboboxSelected>>')

#=============================================================================

# root frame

#=============================================================================

def createDataAnalysisRootFrame(rootframe, param):

#decalre here before creating root frame as their existance is to be verified during startup

global pandasTableFRM

global colCheckboxesFRM

global statisticalDetailsFRM

global plotFRM

pandasTableFRM = Frame()

colCheckboxesFRM = Frame()

w, h = rootframe.winfo\_screenwidth()-50, rootframe.winfo\_screenheight()-150

selectrootframe = Toplevel(rootframe)

selectrootframe.geometry("%dx%d+15+60" % (w, h))

selectrootframe.title("DATA REPORT")

table = param['table'][0]

df = table2df(selectrootframe,table)

r=0;c=0

inputFrame(selectrootframe,param,r,c)

r=0;c=1

createPandasTableFRM(selectrootframe,df,r,c)

r=0;c=2

createColumnCheckBoxesFRM(selectrootframe,table,df,r,c)

r=1;c=1

statisticalDetailsFRM(selectrootframe,table,df,r,c)

'''

r=1;c=1

plotdata={}

createPlotFRM(selectrootframe,plotdata)

'''

#=============================================================================

# standalone start

#=============================================================================

param = {'table':['item'],'pk':['itemcode'],'cbo':['itemcategory.itemcategory']}

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

'''

w,h = rootframe.winfo\_screenwidth()-120, rootframe.winfo\_screenheight()-160

rootframe.geometry("%dx%d+0+0" % (w, h)) #root window size 'wxh' at left top coordinates 0,0

rootframe.maxsize(w,h)

rootframe.minsize(w,h)

'''

'''

import win32con, win32gui, win32print

def get\_dpi():

hDC = win32gui.GetDC(0)

HORZRES = win32print.GetDeviceCaps(hDC, win32con.DESKTOPHORZRES)

VERTRES = win32print.GetDeviceCaps(hDC, win32con.DESKTOPVERTRES)

return HORZRES,VERTRES

w,h = get\_dpi()

rootframe.geometry("%dx%d+0+0" % (w, h))

'''

createDataAnalysisRootFrame(rootframe, param)

#=============================================================================

\_dataAnalysisAndPlotCustom.py

from \_reportDataAnalysisPlotCustomFunctions import \*

#-----------------------------------------------------------------------------

# User Defined Plots and Data Analysis

#-----------------------------------------------------------------------------

def plotCategoryGSTRate(frame):

try:

sql = "select drugcategory, gstrate from drugcategory"

charttype = 'bar' #'plot','bar','hist'

xcol = 'drugcategory'

ycol = 'gstrate'

title = 'GST Rates of Various Medicinal Categories:'

xlabel = 'CATEGORY of DRUG'

ylabel = 'GST RATE'

xticks = ''

yticks = ''

executeSelectQueryForPlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def plotCategorySales(frame):

try:

sql = "select drugcategory, sum(drugquantity) from sale group by drugcategory"

charttype = 'bar'

xcol = 'drugcategory'

ycol = 'sum(drugquantity)'

title = 'Sales of Various Drug Categories:'

xlabel = 'CATEGORY of DRUG'

ylabel = 'TOTAL SALES'

xticks = ''

yticks = ''

executeSelectQueryForPlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def plotTimeSales(frame):

try:

frame = Toplevel()

sql = "select drugname from sale"

df = executeSelectQueryAndReturnDF(frame, sql)

# GUI

# drop-down/pick-up list

Label(frame, text=" ").grid(row=1, column=1, sticky='NE')

namevar = StringVar()

namelookupvalues = df['drugname'].tolist()

Label(frame, text="Drug Name").grid(row=2, column=1, sticky='NE')

nameCBO = Combobox(frame, name='namecbo', width=30, textvariable=namevar)

nameCBO.grid(row=2, column=2)

nameCBO['values'] = namelookupvalues

nameCBO.current(0)

#COPY#

Label(frame, text=" ").grid(row=1, column=1, sticky='NE')

namevar = StringVar()

namelookupvalues = df['drugname'].tolist()

Label(frame, text="Drug Name").grid(row=2, column=1, sticky='NE')

nameCBO = Combobox(frame, name='namecbo', width=30, textvariable=namevar)

nameCBO.grid(row=2, column=2)

nameCBO['values'] = namelookupvalues

nameCBO.current(0)

def nameCBOSelectedEvent(event):

name = event.widget.get()

sql = "SELECT drugquantity, sdate FROM sale where drugname='"+name+"'"

charttype = 'plot'

xcol = 'sdate'

ycol = 'drugquantity'

title = 'Sale Trend of Drugs:'

xlabel = 'DATE'

ylabel = 'SALES'

xticks = ''

yticks = ''

executeSelectQueryForPlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol)

nameCBO.bind("<<ComboboxSelected>>", nameCBOSelectedEvent)

Label(frame, text=" ").grid(row=3, column=3, sticky='NE')

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def plotDrugSales(frame):

try:

sql = "select drugname, sum(drugquantity) from sale group by drugcategory"

charttype = 'bar'

xcol = 'drugname'

ycol = 'sum(drugquantity)'

title = 'Sales of Various Drugs:'

xlabel = 'NAME of DRUG'

ylabel = 'TOTAL SALES'

xticks = ''

yticks = ''

executeSelectQueryForPlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def plot2TimeDrugSales(frame):

try:

sql = []

ycol = []

legend = []

charttype = ''

title = ''

xlabel = ''

ylabel = ''

xticks = ''

yticks = ''

xcol = ''

sql1 = "select drugname from sale"

df = executeSelectQueryAndReturnDF(frame, sql1)

for i in range(2):

frame = Toplevel()

# drop-down/pick-up list

Label(frame, text=" ").grid(row=1, column=1, sticky='NE')

namevar = StringVar()

namelookupvalues = df['drugname'].tolist()

Label(frame, text="Drug Name").grid(row=2, column=1, sticky='NE')

nameCBO = Combobox(frame, name='namecbo', width=30, textvariable=namevar)

nameCBO.grid(row=2, column=2)

nameCBO['values'] = namelookupvalues

nameCBO.current(0)

def nameCBOSelectedEvent(event):

name = event.widget.get()

s="SELECT sum(drugquantity), month(sdate) FROM sale WHERE drugname='"+name+"' GROUP BY month(sdate)"

#print("S: ",s)

sql.append(s)

#print("SQL: ",sql)

legend.append(name)

charttype = 'bar'

xcol = 'month(sdate)'

ycol.append('sum(drugquantity)')

title = 'Both Drugs Juxtaposed:'

xlabel = 'MONTH'

ylabel = 'SALES'

xticks = ''

yticks = ''

#executeSelectQueryForPlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol1)

if len(sql)==2:

print("SQL Outside: ",sql)

executeSelectQueryForMultiplePlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol, legend)

nameCBO.bind("<<ComboboxSelected>>", nameCBOSelectedEvent)

Label(frame, text=" ").grid(row=3, column=3, sticky='NE')

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

''' VERY DIFFICULT WORK IN PROGRESS!!!!!!!!!

def plotMultipleTimeCategorySales(frame):

try:

sql = []

ycol = []

legend = []

charttype = ''

title = ''

xlabel = ''

ylabel = ''

xticks = ''

yticks = ''

xcol = ''

sql1 = "SELECT drugquantity, sdate FROM sale WHERE GROUP BY sdate"

sql2 = 'select count(distinct(drugcategory)) from sale'

sql3 = 'select distinct(drugcategory) from sale'

df1 = executeSelectQueryAndReturnDF(frame, sql1)

df2 = executeSelectQueryAndReturnDF(frame, sql2)

df3 = executeSelectQueryAndReturnDF(frame, sql3)

print(df2.iloc[0,0])

numrows = df2.iloc[0,0]

print('NUM ROWS: ',numrows)

frame = Toplevel()

for i in range(numrows):

name = event.widget.get()

s="SELECT drugquantity, sdate FROM sale WHERE GROUP BY sdate"

#print("S: ",s)

sql.append(s)

#print("SQL: ",sql)

legend.append(name)

charttype = 'plot'

xcol = 'sdate'

ycol.append('sum(drugquantity)')

title = 'Title Here:'

xlabel = 'DATE'

ylabel = 'SALES'

xticks = ''

yticks = ''

#executeSelectQueryForPlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol1)

if len(sql)==numrows:

print("SQL Outside: ",sql)

executeSelectQueryForMultiplePlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol, legend)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

'''

#=============================================================================

def plotMultipleTimeCategorySales(frame):

try:

import matplotlib.pyplot as plt

import numpy as np

sql1 = "SELECT DISTINCT drugcategory FROM sale"

df1 = executeSelectQueryAndReturnDF(frame, sql1)

allcategories = list(df1['drugcategory'])

sql = []

df = []

for i in range(len(allcategories)):

name = allcategories[i]

sql = "SELECT sdate, drugquantity FROM sale WHERE drugcategory = '"+name+"' ORDER BY sdate"

df.append(executeSelectQueryAndReturnDF(frame, sql))

print('\n\ndf1:\n\n',df1,'\n\ndf:\n\n',df)

plt.title('All Categories Juxtaposed:')

plt.xlabel('DATE')

plt.ylabel('SALES')

plt.yticks(np.arange(0,101,10))

print('LEGEND:',allcategories,type(allcategories))

plt.legend(['General', 'Inhalant', 'Analgesics', 'Antibacterials', 'Antidepressants', 'Antihistamines', 'Opioid', 'Barbiturates', 'GENE THERAPY DRUGS', 'Stimulant', 'Drug', 'Antibiotics', 'Antifungals', 'General anesthetics', 'Cholinesterase inhibitors', 'Hallucinogen', 'Antivirals'])

for j in range(len(allcategories)):

plt.plot(df[j]['sdate'],df[j]['drugquantity'])

plt.show()

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def donothing():

pass

#=============================================================================

param = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory']}

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

plotCategoryGSTRate(rootframe)

#=============================================================================

\_database.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#drop database mastertrans2022

#-----------------------------------------------------------------------------

'''

Use options here to:

- fetch database information

- to create tables

- to alter table structures

in the specified database.

'''

#-----------------------------------------------------------------------------

#=============================================================================

def sqlQueryExecution(frame, sql): #all SQL queries except SELECT & DESC

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute(sql)

conn.commit()

msg = "SUCCESS: SQL query executed successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#return msg

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

#return msg

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def showTablesInADatabase(frame):

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute("SHOW TABLES")

result = cursor.fetchall() #result: list of dictionary

tables=[]

for i in result:

tables.append(\*i.values()) #use \* to explode the dictionary so as to enlist values only

return tables

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def table2df(frame='', table='', columns='\*', condition='1=1'):

try:

sql = "select "+columns+" from "+table+" where "+condition

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def backupDatabase(frame):

try:

dbuser = myuser

dbpassword = mypassword

dbname = mydatabase

dbhost = myhost

#backuppath = 'd:\\' #by default create backup in the app folder

datetime = time.strftime('%d-%m-%Y\_%H-%M-%S') #space and symbols in filename does not work! So, use '\_' or '-'

filename = dbname+datetime+".sql" #space in filename does not work! So, use '\_'

f = "C:\\xampp\\mysql\\bin\\mysqldump -h "+dbhost+" -u "+dbuser+" -p "+dbpassword+" "+dbname+" > "+dbname+datetime+".sql"

os.system(f)

tkmsgbox.showinfo("IMPORTANT!", "MySQL Database "+dbname+" backed up successfully as "+filename+".", parent=frame)

except Exception as e:

#tkmsgbox.showinfo("Oops!","Error: %s" % e, parent=databaserootframe)

tkmsgbox.showinfo("Oops!","Error: %s" % e, parent=frame)

#=============================================================================

def restoreDatabase(frame):

#Restoring from a mysql database backup file

#mysql -u yourusername -h yourusername.mysql.pythonanywhere-services.com 'yourusername$dbname' < db-backup.sql

try:

db = mydatabase+'\_restored'

dbhost = myhost

dbuser = myuser

dbpassword = mypassword

dbname = mydatabase

dbRestoreFRM = Toplevel(bg='red')

dbRestoreFRM.title('Restore MySQL Database')

dbRestoreFRM.configure(background="#00ff22")

#dbRestoreFRM.transient()

w, h = dbRestoreFRM.winfo\_screenwidth()-20, 200

dbRestoreFRM.geometry("%dx%d+0+0" % (w, h))

labl = Label(dbRestoreFRM, text="MySQL Database Backup SQL File")

labl.grid(row=1, column=1)

dbbackupsqlfiletxt = Text(dbRestoreFRM, height=3, width=100)

Label(dbRestoreFRM, text="").grid(row=0, column=0) #empty label for padding

Label(dbRestoreFRM, text="").grid(row=2, column=0) #empty label for padding

Label(dbRestoreFRM, text="").grid(row=4, column=0) #empty label for padding

dbbackupsqlfiletxt.grid(row=1, column=2, columnspan=2)

def selectsqlfilebtnevent():

filetypes = (('sql files','\*.sql'),('All files','\*.\*'))

filename = filedialog.askopenfilename(parent=dbRestoreFRM, title='Select sql file',initialdir='/',filetypes=filetypes)

#tkmsgbox.showinfo(title='Selected File',message=filename)

dbbackupsqlfiletxt.delete('1.0', END)

dbbackupsqlfiletxt.insert(INSERT, filename)

selectsqlfilebtn = Button(dbRestoreFRM,text='Select sql File',command=selectsqlfilebtnevent)

selectsqlfilebtn.grid(row=3, column=2)

paddinglbl = Label(dbRestoreFRM, text="").grid(row=0, column=0)

def dbRestorebtnevent():

#labl.configure(text="...restoring mysql database")

dbbackupsqlfile = dbbackupsqlfiletxt.get('1.0','end-1c')

#tkmsgbox.showinfo(title='SQL File to be restored ',message=dbbackupsqlfile)

#f = "C:\\xampp\\mysql\\bin\\mysql -h "+dbhost+" -u "+dbuser+" -p "+dbpassword+" "+dbname+" < "+dbbackupsqlfile

#f = "C:\\xampp\\mysql\\bin\\mysql -u root -p -h localhost 'root$dbname' < "+dbbackupsqlfile

f = 'C:\\xampp\\mysql\\bin\\mysql -h localhost -u root -p '+dbname+' < "'+dbbackupsqlfile+'"'

#IMPORTANT! - use " " and not ' ' to enclose sql filename as ' ' does not work on CMD

os.system(f)

#print("MySQL Database Backup file created as "+filename+".")

tkmsgbox.showinfo("IMPORTANT!", "MySQL Database "+db+" restored.", parent=frame)

dbRestorebtn = Button(dbRestoreFRM, text ="Restore MySQL Database", command=dbRestorebtnevent)

dbRestorebtn.grid(row=5, column=2)

except Exception as e:

#tkmsgbox.showinfo("Oops!","Error: %s" % e, parent=rootmain)

tkmsgbox.showinfo("Oops!","Error: %s" % e, parent=frame)

#=============================================================================

def resetDatabase(frame, db):

try:

#open mysql database, if exists

#conn = pymysql.connect( host='localhost',user='root',port='',password='',db=db,cursorclass=pymysql.cursors.DictCursor)

#cursor=conn.cursor()

#tkmsgbox.showinfo("MESSAGE", "Database "+mydb+" is all set and open.")

#-------------

#drop database - disable after testing

cursor.execute("drop database if exists "+db)

conn.commit()

createNewDatabase(frame, db)

tkmsgbox.showinfo("Oops!","Error: %s" % e, parent=frame)

except Exception as e:

#tkmsgbox.showinfo("Oops!","Error: %s" % e, parent=rootmain)

tkmsgbox.showinfo("Oops!","Error: %s" % e, parent=frame)

#=============================================================================

def createNewDatabase(frame,db):

try:

#create mysql database, if does not exist

#tkmsgbox.showinfo("IMPORTANT!", "Setting up MySQL Database.", parent=rootmain)

#tkmsgbox.showinfo("IMPORTANT!", "Setting up MySQL Database.")

#conn = pymysql.connect(host='localhost', user='root', password='')

#cursor = conn.cursor()

#create mysql database

cursor.execute("CREATE DATABASE IF NOT EXISTS "+db)

#open mysql database

conn = pymysql.connect( host="localhost",user="root",password="",database= db,cursorclass=pymysql.cursors.DictCursor)

cursor = conn.cursor()

msg = "SUCCESS: Database "+ db +" created successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def descTable(frame,table):

try:

sql = "desc "+table

cursor.execute(sql)

data = cursor.fetchall() #list of dict with one common key 'Field'

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

def displayPandasTable(frame,df,r=0,c=0,w=400,h=400,bkcolor='orange'):

#global pandasTableFRM

pandasTableFRM = Frame(frame, width=w, height=h)

pandasTableFRM.grid(row=r, column=c, sticky='NW')

#pt = Table(pandasTableFRM1, dataframe=df, showtoolbar=True, width=w, height=h, showstatusbar=True)

pt = Table(pandasTableFRM, dataframe=df, width=w, height=h)

pt.cellbackgr = bkcolor

pt.grid()

pt.show()

#for insert disable the following event handling; enable it for update only

def leftButtonClickEvent(event): #left-button click event handling

rowclicked = pt.get\_row\_clicked(event)

rowDataList = pt.model.df.loc[rowclicked] #Series

updateARow(rowDataList)

pt.rowheader.bind('<Button-1>',leftButtonClickEvent)

#=============================================================================

# child frames

#=============================================================================

def createDatabaseChildFrames(databaserootframe):

frame = Frame(databaserootframe)

frame.grid(row=0, column=0, rowspan=2, sticky='NW', padx=15, pady=15) #W-left, E-right, N-top, S-bottom

r=0

Label(frame, text='Database:').grid(row=r, column=0, sticky='NW')

databaseTXT = Entry(frame, width=35)

databaseTXT.grid(row=r, column=1, sticky='NW')

#databaseTXT.delete('1.0', END)

databaseTXT.insert(INSERT, mydatabase)

r += 1

Label(frame, text='TABLES:').grid(row=r, column=0, sticky='NW')

tablesVar = StringVar()

lookupvals = showTablesInADatabase(frame)

tablesCBO = Combobox(frame, name='tablecbo', width=35, textvariable=tablesVar) #,width=30)

tablesCBO['values'] = lookupvals

tablesCBO.grid(row=r, column=1, sticky='NW')

def tablesCBOSelectedEvent(event):

table = event.widget.get()

w, h = databaserootframe.winfo\_screenwidth()-50, databaserootframe.winfo\_screenheight()-150

w = w\*60/100

h = h\*40/100

r=0; c=1; color='orange'

df = table2df(databaserootframe,table)

displayPandasTable(databaserootframe,df,r,c,w,h,color)

r=1; c=1; color='blue'

df = descTable(databaserootframe,table)

displayPandasTable(databaserootframe,df,r,c,w,h,color)

tablesCBO.bind("<<ComboboxSelected>>", tablesCBOSelectedEvent)

r += 1

Label(frame, text='Execute Raw SQL Query (Only DDL):').grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

# Raw SQL Query Execution

rawDDLSQLQueryTXT = Text(frame, width=50, height=15)

rawDDLSQLQueryTXT.grid(row=r, column=0, columnspan=2, sticky='NW')

def rawDDLSQLQueryBTNEvent():

sql = rawDDLSQLQueryTXT.get("1.0",'end-1c') #for TEXT widget use parameters

sqlQueryExecution(frame, sql)

#fire tablescbo combobox click-event programmatically

#tablesCBO.invoke()

idx = lookupvals.index(table)

tablesCBO.current(idx)

tablesCBO.event\_generate('<<ComboboxSelected>>')

r += 1

rawDDLSQLQueryBTN = Button(frame, text='Execute Raw SQL Query (Only DDL)', command=rawDDLSQLQueryBTNEvent)

rawDDLSQLQueryBTN.grid(row=r, column=0, columnspan=2, padx=20, pady=5, sticky='NE')

r += 1

Label(frame, text='CREATE NEW DATABASE').grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

Label(frame, text='New Database: ').grid(row=r, column=0, sticky='NW')

createNewDatabaseTXT = Entry(frame, width=35)

createNewDatabaseTXT.grid(row=r, column=1, sticky='NW')

#createNewDatabaseTXT.delete('1.0', END)

createNewDatabaseTXT.insert(INSERT, mydatabase)

r += 1

def createNewDatabaseBTNEvent():

db = createNewDatabaseTXT.get()

createNewDatabase(db)

createNewDatabaseBTN = Button(frame, text='Create Database', command=createNewDatabaseBTNEvent)

createNewDatabaseBTN.grid(row=r, column=0, columnspan=2, padx=20, pady=5, sticky='NE')

r += 1

Label(frame, text='Create Tables With Test Data: ').grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

Label(frame, text='(As per the detail in createTablesWithTestData() function.').grid(row=r, column=0, \

columnspan=2, sticky='NW')

r += 1

def createTablesWithTestDataBTNEvent():

db = databaseTXT.get()

createTablesWithTestData(db)

createTablesWithTestDataBTN = Button(frame, text='Create Tables with Test Data', command=createTablesWithTestDataBTNEvent)

createTablesWithTestDataBTN.grid(row=r, column=0, columnspan=2, padx=20, pady=5, sticky='NE')

r += 1

def backupDatabaseBTNEvent():

backupDatabase()

backupDatabaseBTN = Button(frame, text='Backup Database', command=backupDatabaseBTNEvent)

backupDatabaseBTN.grid(row=r, column=0, columnspan=2, padx=20, pady=5, sticky='NW')

r += 1

def restoreDatabaseBTNEvent():

restoreDatabase()

restoreDatabaseBTN = Button(frame, text='Restore Database', command=restoreDatabaseBTNEvent)

restoreDatabaseBTN.grid(row=r, column=0, columnspan=2, padx=20, pady=5, sticky='NW')

'''

r += 1

Label(frame, text='Reset Database').grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

def resetDatabaseBTNEvent():

db = databaseTXT.get()

resetDatabase(db)

resetDatabaseBTN = Button(frame, text='Reset Database', command=resetDatabaseBTNEvent)

resetDatabaseBTN.grid(row=r, column=0, columnspan=2, padx=20, pady=5, sticky='NW')

'''

for widget in frame.winfo\_children():

widget.grid(padx=0, pady=5)

#=============================================================================

# root frame

#=============================================================================

def createDatabaseFrame(rootframe):

w, h = rootframe.winfo\_screenwidth()-50, rootframe.winfo\_screenheight()-150

databaserootframe = Toplevel(rootframe)

databaserootframe.geometry("%dx%d+15+60" % (w,h))

databaserootframe.title("MANAGE DATABASE")

createDatabaseChildFrames(databaserootframe)

#createDatabase()

#createTablesWithTestData()

#=============================================================================

# standalone start

#=============================================================================

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

createDatabaseFrame(rootframe)

#=============================================================================

\_dataExportImport.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#------------------------------------------------------------------------------------------------

global globaldf #for reference from within many functions

#------------------------------------------------------------------------------------------------

def savedisplayedDFtoMySQLTableSQLEngine(rootframe, globaldf, cbotableselected):

'''

#PREPARING DATAFRAME

#===================

dftemp = pd.read\_csv("meet\_logs\_1627694691222 31July2021.csv")

df.rename(columns=mymapper, inplace=True)

df.loc[:,'Date'] = df['Date'].str[:12]

df.loc[:,'Duration']=df['Duration'].fillna(0)

df.loc[:,'Duration'] = round(df['Duration']/60)

df.Duration.astype(int)

'''

'''

#dataframe to SQL

#================

sqlEngine = create\_engine("mysql+pymysql://{user}:{pw}@localhost/{db}".format(user="root",pw="",db=mydatabase))

connection = sqlEngine.connect()

try:

frame = globaldf.to\_sql(cbotableselected, connection, index=False) #if\_exists='replace', if\_exists='append'

except pymysql.Error as e:

print("ERROR: %d: %s" %(e.args[0], e.args[1]))

else:

print("Table %s created successfully."%cbotableselected)

finally:

connection.close()

'''

#------------------------------------------------------------------------------------------------

def savedisplayedDFtoMySQLTableInsert(rootframe, globaldf, cbotableselected):

# INSERT

try:

cols = ''

for c in globaldf.columns:

cols += c + ','

cols = cols[:-1]

#for i in globaldf.iterrows:

for i in range(len(globaldf.index)):

#for i in range(len(globaldf.columns)):

row = globaldf.loc[i].tolist()

vals = ''

for v in row:

vals += "'"+str(v)+"'"+','

vals = vals[:-1]

sql = "INSERT INTO " + cbotableselected + "(" + cols + ") VALUES(" + vals + ")"

cursor.execute(sql)

conn.commit()

msg = "Records inserted into MySQL table "+cbotableselected

tk.messagebox.showinfo("MESSAGE", msg, parent=rootframe)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=rootframe)

#------------------------------------------------------------------------------------------------

def savedisplayedDFtoMySQLTableUpdate(rootframe, globaldf, cbotableselected):

# UPDATE

try:

#sql = "SHOW INDEX FROM "+mydatabase+"."+cbotableselected+" WHERE Key\_name = 'PRIMARY'"

sql = '''SELECT COLUMN\_NAME FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_SCHEMA = "'''+mydatabase+'''"

AND TABLE\_NAME = "'''+cbotableselected+'''" AND COLUMN\_KEY = "PRI"'''

cursor.execute(sql)

pk = cursor.fetchone()

#print("Table: ",cbotableselected," PK: ",pk," ",pk['COLUMN\_NAME']) #outputs a dictionary e.g. {'COLUMN\_NAME': 'customercode'}

#print("PK Value: ",globaldf.iloc[0][pk['COLUMN\_NAME']])

colnames = globaldf.columns

#for i in globaldf.iterrows:

for i in range(len(globaldf.index)):

rowdata = globaldf.loc[i].tolist()

sql = "UPDATE " + cbotableselected + " SET "

for j in range(len(globaldf.columns)):

if colnames[j]==pk['COLUMN\_NAME']:

pass

else:

sql += colnames[j] + "='" + str(rowdata[j]) + "',"

sql = sql[:-1]

sql += " WHERE "+pk['COLUMN\_NAME']+" = "+str(globaldf.iloc[i][pk['COLUMN\_NAME']])

#print(sql)

cursor.execute(sql)

conn.commit()

msg = "Records updated in MySQL table "+cbotableselected

tk.messagebox.showinfo("MESSAGE", msg, parent=rootframe)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=rootframe)

#------------------------------------------------------------------------------------------------

def savedisplayedDFasCSVFile(df):

filetypes = [('CSV Files','\*.csv')]

saveatfilepath = asksaveasfile(mode='w', filetypes = filetypes, defaultextension=filetypes)

df.to\_csv(saveatfilepath, index=False, line\_terminator='\n')

# header=False

#use lineterminator to avoid blank lines in csv file

#------------------------------------------------------------------------------------------------

def getNumOfRowsOfMySQLTable(mysqltable):

cursor.execute("select count(\*) as numofrows from "+mysqltable)

numofrows = cursor.fetchone()['numofrows']

return numofrows

#------------------------------------------------------------------------------------------------

def getDataFrameFromCSVFile():

filetypes = [('CSV Files','\*.csv')]

csvfilepath = askopenfilename(filetypes = filetypes, defaultextension = filetypes)

#dfcsv=pd.read\_csv(file\_path,usecols=['Name','Email','Gender'])

df = pd.read\_csv(csvfilepath)

return df

#------------------------------------------------------------------------------------------------

def getDataFrameFromSQLQuery(sql):

cursor.execute(sql)

data = cursor.fetchall()

df=pd.DataFrame(data)

return df

#------------------------------------------------------------------------------------------------

def getDataFrameFromMySQLTable(mysqltable):

cursor.execute("select \* from "+mysqltable)

data = cursor.fetchall()

df=pd.DataFrame(data)

return df

#------------------------------------------------------------------------------------------------

def getTablenamesOfMySQLDatabase(dbname):

sql = '''SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_TYPE = 'BASE TABLE' AND TABLE\_SCHEMA="'''+dbname+'''"'''

cursor.execute(sql)

tablesdict = cursor.fetchall()

tables=[]

for i in range(len(tablesdict)):

tables.append(tablesdict[i]['TABLE\_NAME'])

return tables

#------------------------------------------------------------------------------------------------

def getColumnNamesOfMySQLTable(mysqltable):

cursor.execute("DESC "+mysqltable)

colsdict = cursor.fetchall()

cols=[]

for i in range(len(colsdict)):

cols.append(colsdict[i]['Field']) #show column names of the specified table

return cols

#------------------------------------------------------------------------------------------------

def displayPandasTable(frame,r,df):

pandasTableFRM = Frame(frame)

pandasTableFRM.grid(row=r, column=1, sticky='NW')

#pt = Table(pandasTableFRM, dataframe=df, showtoolbar=True, width=w, showstatusbar=True)

pt = Table(pandasTableFRM, dataframe=df, showtoolbar=True, showstatusbar=True)

pt.cellbackgr = 'orange'

pt.grid()

pt.show()

global globaldf

globaldf = df #for reference from within other functions

#------------------------------------------------------------------------------------------------

def index(rootframe):

topframe = Toplevel()

'''

'''

r=0

#display tablenames of the given mysql database in combo box

tables = getTablenamesOfMySQLDatabase(mydatabase)

Label(topframe, text="Select Table").grid(row=r, column=0, sticky='W')

cboTables = ttk.Combobox(topframe, values=tables)

cboTables.current(0)

cboTables.grid(row=r, column=1, sticky='W')

def cboTablesSelectedEvent(event):#display columns of the selected table

cols = getColumnNamesOfMySQLTable(cboTables.get())

cboColumns['values'] = cols

cboColumns.current(0)

#df=createDataFrame(cboTables.get())

numofrows = getNumOfRowsOfMySQLTable(cboTables.get())

numofrowstxt.delete(0,END)

numofrowstxt.insert(INSERT, numofrows)

#numofrowstxt.get('1.0', 'end-1c'))

df = getDataFrameFromMySQLTable(cboTables.get())

displayPandasTable(topframe, r+1, df)

cboTables.bind("<<ComboboxSelected>>", cboTablesSelectedEvent)

#display No. of rows in the selected table

r+=1

Label(topframe, text="No. of rows in table").grid(row=r, column=0, sticky=W)

numofrowstxt = Entry(topframe, width=30)

numofrowstxt.grid(row=r, column=1, sticky='W')

r+=1

#display column names of the given mysql table in combo box

r+=1

Label(topframe, text="Columns in the selected table").grid(row=r, column=0, sticky='E')

cboColumns=ttk.Combobox(topframe, values=tables)

cboColumns.grid(row=r, column=1, sticky='W')

r+=1

Label(topframe, text="SQL Query").grid(row=r, column=0, sticky='W')

#sqlTXT.delete('1.0', END)

#sqlTXT.insert(INSERT, txt)

sqlTXT = Text(topframe, height=3, width=80)

sqlTXT.grid(row=r, column=1, sticky='W')

def sqlQuerySubmitBTNClickEvent():

sql = sqlTXT.get()

df = getDataFrameFromSQLQuery(sql)

displayPandasTable(topframe, r+1, df)

r+=1

Button(topframe, text="Submit SQL Query", command=sqlQuerySubmitBTNClickEvent).grid(row=r, column=1, sticky='E')

r+=1

def openCSVFileBtnClickEvent():

df = getDataFrameFromCSVFile()

displayPandasTable(topframe, r+1, df)

Button(topframe, text='Open CSV file', command=openCSVFileBtnClickEvent).grid(row=r, column=1, sticky='W')

r+=1

def saveDFasCSVFileBtnClickEvent():

global globaldf

savedisplayedDFasCSVFile(globaldf)

msg = "CSV File saved."

tk.messagebox.showinfo("MESSAGE", msg, parent=topframe)

Button(topframe, text='Save DF as CSV file', command=saveDFasCSVFileBtnClickEvent).grid(row=r, column=1, sticky='W')

r+=1

def saveDFtoMySQLTableSQLEngineBtnClickEvent():

global globaldf

savedisplayedDFtoMySQLTableSQLEngine(topframe, globaldf, cboTables.get())

msg = "Data saved in selected MySQL table."

tk.messagebox.showinfo("MESSAGE", msg, parent=topframe)

Button(topframe, text='Save displayed DF to selected MySQL Table using SQLEngine', command=saveDFtoMySQLTableSQLEngineBtnClickEvent).grid(row=r, column=1, sticky='W')

r+=1

def saveDFtoMySQLTableInsertBtnClickEvent():

global globaldf

savedisplayedDFtoMySQLTableInsert(topframe, globaldf, cboTables.get())

msg = "Data saved in selected MySQL table."

tk.messagebox.showinfo("MESSAGE", msg, parent=topframe)

Button(topframe, text='INSERT displayed DF to selected MySQL Table', command=saveDFtoMySQLTableInsertBtnClickEvent).grid(row=r, column=1, sticky='W')

r+=1

def saveDFtoMySQLTableUpdateBtnClickEvent():

global globaldf

savedisplayedDFtoMySQLTableUpdate(topframe, globaldf, cboTables.get())

msg = "Data saved in selected MySQL table."

tk.messagebox.showinfo("MESSAGE", msg, parent=topframe)

Button(topframe, text='UPDATE displayed DF to selected MySQL Table', command=saveDFtoMySQLTableUpdateBtnClickEvent).grid(row=r, column=1, sticky='W')

#------------------------------------------------------------------------------------------------

#=============================================================================

# standalone start for code testing - to run this file independently

#=============================================================================

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

index(rootframe)

#=============================================================================

\_edit.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#-----------------------------------------------------------------------------

# pandastable frame r=1 c=0

# colcheckboxes frame r=2 c=0

# insertDataInput frame r=0 c=1

#-----------------------------------------------------------------------------

global tablesCBO

global pandasTableFRM

global colCheckboxesFRM

#-----------------------------------------------------------------------------

def sqlQueryExecution(frame, sql): #all SQL queries except SELECT & DESC

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute(sql)

conn.commit()

msg = "SUCCESS: SQL query executed successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#return msg

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

#return msg

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def showTablesInADatabase(frame):

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute("SHOW TABLES")

result = cursor.fetchall() #result: list of dictionary

tables=[]

for i in result:

tables.append(\*i.values()) #use \* to explode the dictionary so as to enlist values only

return tables

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def descTable(frame, table):

try:

sql = "desc "+table

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall() #list of dict with one common key 'Field'

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def getPrimaryKeyColumn(frame, table):

#mydb = \_libraryAndDBConnection.mydatabase

mydb = mydatabase

try:

sql = "SELECT COLUMN\_NAME FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_SCHEMA = '" + mydb + \

"' AND TABLE\_NAME = '" + table + "' AND COLUMN\_KEY = 'PRI'"

#print(mydb, ' ',table, ' ',sql)

print("STARTED: ",mydb)

cursor.execute(sql)

result = cursor.fetchall() #result: list of dictionary

#print("result: ",result)

col = []

for d in result:

col.append(d['COLUMN\_NAME'])

#print(col)

return col

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def table2df(frame='', table='', columns='\*', condition='1=1'):

try:

sql = "select "+columns+" from "+table+" where "+condition

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def table2dfSQLQuery(frame='', sql=''):

try:

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def searchTableButtonClickEvent(frame,table,column,value):

try:

sql = "select \* from "+table+" where "+column+"='"+value+"'"

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def updateARow(rowDataList): #rowDataList is a series

try:

frameUpdateTable = Toplevel() #create a pop-up window

frameUpdateTable.title("Update Table")

frameUpdateTable.geometry()

frameUpdateTable.configure(background='yellow')

r = 0

for colname in rowDataList.index:

#print(i,' -- ',rowDataList[colname])

Label(frameUpdateTable, text=colname, bg='yellow') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

e = Entry(frameUpdateTable)

e.grid(row=r, column=2, padx=15, pady=5)

e.insert(INSERT, rowDataList[colname])

r += 1

def updateTableSubmitButtonClickEvent():

pass

updateTableSubmitButton = Button(frameUpdateTable, text="Update", command=updateTableSubmitButtonClickEvent)

updateTableSubmitButton.grid(row=r, column=2, padx=10, pady=10)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frameUpdateTable)

#-----------------------------------------------------------------------------

def createColumnCheckBoxesFRM(selectrootframe, table, df, r=1, col=1):

global tablesCBO

global colCheckboxesFRM

w, h = selectrootframe.winfo\_screenwidth()-120, selectrootframe.winfo\_screenheight()-160

w = w\*20/100

numOfCols = len(df.columns.tolist())

colCheckboxesFRM = Frame(selectrootframe, width=w)

#colCheckboxesFRM.grid(row=r, column=0, columnspan=numOfCols, sticky='NW', padx=15, pady=15) #NW-top left

colCheckboxesFRM.grid(row=r, column=col, sticky='NW', padx=15, pady=15) #NW-top left

r = 1

col = 0

for c in df.columns.tolist():

globals()[c+'VAR'] = IntVar()

globals()[c+'VAR'].set(0)

globals()[c+'CHK'] = ttk.Checkbutton(colCheckboxesFRM,variable=globals()[c+'VAR'],text=c, onvalue=1, offvalue=0)

globals()[c+'CHK'].grid(column=col, row=r, sticky='W')

col+=1

r+=1

def getCheckedBoxesButtonClickEvent(df):

checkedColumnsLST = []

for c in df.columns.tolist():

if globals()[c+'VAR'].get()==1:

checkedColumnsLST.append(c)

cols=''

for c in checkedColumnsLST:

cols += c + ","

cols = cols[:-1]

df = table2df(colCheckboxesFRM, table, cols)

createPandasTableFRM(selectrootframe, df) #!IMPORTANT - use selectrootframe

getCheckedBoxesButton = Button(colCheckboxesFRM, text='Submit List of Selected Columns', \

command=lambda: getCheckedBoxesButtonClickEvent(df))

getCheckedBoxesButton.grid(row=r, column=0, columnspan=numOfCols, sticky='EW')

for widget in colCheckboxesFRM.winfo\_children():

widget.grid(padx=0, pady=3)

#=============================================================================

def createPandasTableFRM(frame, df, r=0, c=1):

global pandasTableFRM

w, h = frame.winfo\_screenwidth()-120, frame.winfo\_screenheight()-160

w = w\*60/100

pandasTableFRM = Frame(frame, height=h, width=w)

pandasTableFRM.grid(row=r, column=c, sticky='NW')

pt = Table(pandasTableFRM, dataframe=df, showtoolbar=True, width=w, showstatusbar=True)

pt.cellbackgr = 'orange'

pt.grid()

pt.show()

#for insert disable the following event handling; enable it for update only

def leftButtonClickEvent(event): #left-button click event handling

rowclicked = pt.get\_row\_clicked(event)

rowDataList = pt.model.df.loc[rowclicked] #Series

updateARow(rowDataList)

pt.rowheader.bind('<Button-1>',leftButtonClickEvent)

#=============================================================================

def inputFrame(selectrootframe, param, r=0, c=0):

global tablesCBO

global colCheckboxesFRM

global pandasTableFRM

table = param['table'][0]

pk = param['pk'][0]

cbo = param['cbo'][0]

w, h = selectrootframe.winfo\_screenwidth()-120, selectrootframe.winfo\_screenheight()-160

w = w\*30/100

#frame = Frame(selectrootframe, height=h, width=w)

frame = Frame(selectrootframe, width=w)

frame.grid(row=r, column=c, rowspan=2, sticky='NW')#W-left, E-right, N-top, S-bottom

df = table2df(frame,table)

dfColumns = descTable(frame,table)

r=0

Label(frame, text='TABLES:').grid(row=r, column=0, sticky='NW')

tables = StringVar()

tablesCBO = Combobox(frame, name='tablecbo', width=27)#, textvariable=tablesCBOvar) #,width=30)

lookupvals = showTablesInADatabase(frame)

tablesCBO['values'] = lookupvals

tablesCBO.grid(row=r, column=1, sticky='NW')

idx = lookupvals.index(table)

tablesCBO.current(idx)

df = table2df(frame,lookupvals[0])

dfColumns = descTable(frame,lookupvals[0])

'''

if len(lookupvals)>0:

tablesCBO.current(0)

df = table2df(frame,lookupvals[0])

dfColumns = descTable(frame,lookupvals[0])

'''

def tablesCBOSelectedEvent(event):

# !Important - use "selectrootframe" frame and not this "frame" #pass row and col for child frames

table = event.widget.get()

pk = getPrimaryKeyColumn(frame, table)

#print(table, event.widget.\_name)

df = table2df(frame,table)

updateColCBO['values'] = df.columns.tolist()

whereColCBO['values'] = df.columns.tolist()

dfColumns = descTable(selectrootframe,table)

if pkTXT.winfo\_exists()== 1:

#pkTXT.delete('1.0', END)

#pkTXT.delete()

pkTXT.delete(0, END)

pkTXT.insert(INSERT,"")

pkTXT.insert(INSERT,pk)

if pandasTableFRM.winfo\_exists()== 1:

pandasTableFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

if colCheckboxesFRM.winfo\_exists()== 1:

colCheckboxesFRM.destroy()

createColumnCheckBoxesFRM(selectrootframe, table, df, 1, 1)

tablesCBO.bind("<<ComboboxSelected>>", tablesCBOSelectedEvent)

r += 1

Label(frame, text='PRIMARY KEY:').grid(row=r, column=0, sticky='NW')

pkTXT = Entry(frame, width=30)

pkTXT.grid(row=r, column=1, sticky='NW')

#pkTXT.delete('1.0', END)

pkTXT.delete(0,END) #for Entry (0,END)

pkTXT.insert(INSERT,pk)

'''

Label(frame, text=pk).grid(row=r, column=1, sticky='NW')

r += 1

Label(frame, text='LOOKUP VALUES (CBO):').grid(row=r, column=0, sticky='NW')

Label(frame, text=pk).grid(row=r, column=1, sticky='NW')

'''

r += 1

#sub frame #1

frame1 = Frame(frame, bg="blue")

frame1.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

# Update what & with what

r=0

Label(frame1, text='Update column').grid(row=r, column=0, sticky='NW')

updateColvar = StringVar()

updateColCBO = Combobox(frame1, name='updateColCBO', width=30, textvariable=updateColvar)

updateColCBO.grid(row=r, column=1, sticky='NW')

table = tablesCBO.get()

df = table2df(frame1,table)

updateColCBO['values'] = df.columns.tolist()

updateColCBO.current(0)

r += 1

Label(frame1, text='With (New data value):').grid(row=r, column=0, sticky='NW')

updateValTXT = Entry(frame1, width=30)

updateValTXT.grid(row=r, column=1, sticky='NW')

r += 1

# Update Condition

Label(frame1, text='Where (Condition):').grid(row=r, column=0, sticky='NW')

whereCol = StringVar()

whereColCBO = Combobox(frame1, name='whereColCBO', width=30, textvariable=whereCol)

whereColCBO.grid(row=r, column=1, sticky='NW')

df = table2df(frame1,table)

whereColCBO['values'] = df.columns.tolist()

r += 1

Label(frame1, text='compare').grid(row=r, column=0, sticky='NW')

whereOp = StringVar()

whereOpCBO = Combobox(frame1, name='whereOpCBO', width=10, textvariable=whereOp)

whereOpCBO['values'] = ['=','<>','>','<','>=','<=','between','like','in']

whereOpCBO.grid(row=r, column=1, sticky='NW')

whereOpCBO.current(0)

r += 1

whereValTXT = Entry(frame1, width=30)

whereValTXT.grid(row=r, column=1, sticky='NW')

r += 1

def createUpdateQueryBTNEvent():

table = tablesCBO.get()

column = updateColCBO.get()

value = updateValTXT.get()

condition = whereColCBO.get()+whereOpCBO.get()+"'"+whereValTXT.get()+"'"

sql = "UPDATE "+table+" SET "+column+"='"+value+"' WHERE "+condition

print(sql)

sqlQueryTXT.delete('1.0', END) #for Text ('1.0', END)

sqlQueryTXT.insert(INSERT, sql)

#df = searchTableButtonClickEvent(frame1,table,column,value)

#createPandasTableFRM(selectrootframe, df)

createUpdateQueryBTN = Button(frame1, text='Create Update SQL Query (execute the following query)', \

width=20, command=createUpdateQueryBTNEvent)

createUpdateQueryBTN.grid(row=r, column=0, columnspan=2, sticky='EW')

r += 1

#sub frame #2

# General SQL Query Execution

frame2 = Frame(frame)

frame2.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

Label(frame2, text='Raw SQL Query (Select only):').grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

sqlQueryTXT = Text(frame2, width=35, height=6)

sqlQueryTXT.grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

def sqlQueryExecutionButtonClickEventStart():

'''

#####For SELECT query

#for sql query use raw string

sql = sqlQueryTXT.get("1.0",'end-1c') #for TEXT use - textbox1.get("1.0",'end-1c') #For entry txt.get()

#sqlQueryExecution(frame2, sql)

#print("SQL: ",sql)

#TO GET THE TABLE NAME FROM A SQL QUERY!!!!!!!!!!!!!!!

#sql="select \* from item where a=1";

idx1 = sql.find("from");

idx2 = sql[idx1+4:].strip().find(" ");

idx3 = sql.find(" ",idx1+4+idx2-1);

idx4=sql[idx1+4+idx2-1:].strip().find(" ");

table = sql[idx1+4+idx2-1:idx3]

#print(table)

df = table2dfSQLQuery(frame2, sql)

pandasTableFRM.destroy()

colCheckboxesFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

createColumnCheckBoxesFRM(selectrootframe, table, df, 1, 1)

df = table2dfSQLQuery(frame2, sql)

pandasTableFRM.destroy()

colCheckboxesFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

createColumnCheckBoxesFRM(selectrootframe, table, df, 1, 1)

'''

sql = sqlQueryTXT.get("1.0",'end-1c') #for TEXT use - textbox1.get("1.0",'end-1c') #For entry txt.get()

sqlQueryExecution(frame, sql)

#refresh df and then pandastable to show updated values

table = tablesCBO.get()

df = table2df(frame, table)

#createPandasTableFRM(frameUpdateTable, df, param)

global pandasTableFRM

pt = Table(pandasTableFRM, dataframe=df)

pt.cellbackgr = 'orange'

pt.grid()

pt.show()

sqlQueryExecutionButton = Button(frame2, text='SQL Query Execution', width=20, command=sqlQueryExecutionButtonClickEventStart)

sqlQueryExecutionButton.grid(row=r, column=1, sticky='EW')

for widget in frame.winfo\_children():

widget.grid(padx=10, pady=10)

for widget in frame1.winfo\_children():

widget.grid(padx=10, pady=10)

for widget in frame2.winfo\_children():

widget.grid(padx=10, pady=10)

'''

#fire tablescbo combobox click-event programmatically

idx = lookupvals.index(table)

tablesCBO.current(idx)

#print(idx, tablesCBO.current(idx), lookupvals[idx])

tablesCBO.event\_generate('<<ComboboxSelected>>')

'''

#=============================================================================

# root frame

#=============================================================================

def createEditRootFrame(rootframe, param):

#decalre here before creating root frame as their existance is to be verified during startup

global pandasTableFRM

global colCheckboxesFRM

pandasTableFRM = Frame()

colCheckboxesFRM = Frame()

w, h = rootframe.winfo\_screenwidth()-50, rootframe.winfo\_screenheight()-150

selectrootframe = Toplevel(rootframe)

selectrootframe.geometry("%dx%d+15+60" % (w, h))

selectrootframe.title("DATA REPORT")

table = param['table'][0]

df = table2df(selectrootframe,table)

r=0;c=0

inputFrame(selectrootframe,param,r,c)

r=0;c=1

createPandasTableFRM(selectrootframe,df,r,c)

r=1;c=1

createColumnCheckBoxesFRM(selectrootframe,table,df,r,c)

#=============================================================================

# standalone start

#=============================================================================

param = {'table':['item'],'pk':['itemcode'],'cbo':['itemcategory.itemcategory']}

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

createEditRootFrame(rootframe, param)

#=============================================================================

\_libraryAndDBConnection.py

'''

Import this "libraryImports.py" in other python files as "import libraryImports"

and reference its content as with "libraryImports.function"

and to avoid dot notation, import this library of libarires as "from libraryImports import \*"

'''

############## supressing future warnings

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

##############

#GUI library

import tkinter as tk

from tkinter import \*

from tkinter import ttk

from tkinter import Tk, Label

import tkinter.messagebox as tkmsgbox #for messagebox

from tkinter.messagebox import askokcancel, showinfo, WARNING

from tkinter import filedialog

#from tkinter import filedialog as fd

from tkinter import messagebox

from tkinter.ttk import Combobox

from tkcalendar import Calendar

from tkcalendar import DateEntry

import datetime

from datetime import date

from datetime import datetime

from datetime import timedelta

import PIL.Image #to avoid namespace conflicts as image is a common name

from PIL import Image, ImageTk #for image resize

#for file upload and calender

import os, shutil

from pathlib import Path

import time

#MySQL connectivity

import pymysql

from pandastable import Table

#Data & Visualization

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from tkinter.filedialog import asksaveasfile #for save as dialog box

from tkinter.filedialog import askopenfilename #for filename dialog box

from sqlalchemy import create\_engine #for dataframe to MySQL

# Install FPDF2 ver. 2.7.0+ (for table, page background etc.)

# pip install fpdf2==2.7.0

'''

from fpdf import FPDF

from fpdf.fonts import FontFace

from fpdf.enums import XPos, YPos

'''

from fpdf import \*

#to split a string with multiple delimiters

#e.g. mathematical expression as string to be splitted at + - \* / ....

import re

#====================================================================================

#MySQL Connectivity

myhost = 'localhost'

myuser = 'root'

mypassword = ''

mydatabase = 'd22111284'

#keep following conn and cursor outside any scope incl. try and except for global access

conn=''

cursor=''

try:

conn = pymysql.connect( host=myhost,

user=myuser,

password=mypassword,

db=mydatabase,

cursorclass=pymysql.cursors.DictCursor

)

#msg = "Connection established successfully."

#tk.messagebox.showinfo("SUCCESS: ", msg)

except:

conn = pymysql.connect(host='localhost', user='root', password='')

cursor = conn.cursor()

#create mysql database

cursor.execute("CREATE DATABASE IF NOT EXISTS "+mydatabase)

msg = "SUCCESS: Database "+ mydatabase +" created successfully."

print(msg)

#open mysql database

#if conn is not None and conn.is\_connected():

conn = pymysql.connect( host=myhost, user=myuser, password=mypassword, db=mydatabase,

cursorclass=pymysql.cursors.DictCursor

)

cursor = conn.cursor()

#print('Connection established.')

#====================================================================================

\_master.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#-----------------------------------------------------------------------------

#Create global widget for their easy access across the frames

global tablesCBO

global pandasTableFRM

#pandastable frame r=1 c=0

#colcheckboxes frame r=2 c=0

#insertDataInput frame r=0 c=1

#-----------------------------------------------------------------------------

def sqlQueryExecution(frame, sql): #all SQL queries except SELECT & DESC

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute(sql)

conn.commit()

msg = "SUCCESS: SQL query executed successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#return msg

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

#return msg

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def getPrimaryKeyColumn(frame, table):

#mydb = \_libraryAndDBConnection.mydatabase

mydb = mydatabase

try:

sql = "SELECT COLUMN\_NAME FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_SCHEMA = '" + mydb + \

"' AND TABLE\_NAME = '" + table + "' AND COLUMN\_KEY = 'PRI'"

#print(mydb, ' ',table, ' ',sql)

print("STARTED: ",mydb)

cursor.execute(sql)

result = cursor.fetchall() #result: list of dictionary

#print("result: ",result)

col = []

for d in result:

col.append(d['COLUMN\_NAME'])

#print(col)

return col

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def showTablesInADatabase(frame):

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute("SHOW TABLES")

result = cursor.fetchall() #result: list of dictionary

tables=[]

for i in result:

tables.append(\*i.values()) #use \* to explode the dictionary so as to enlist values only

return tables

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def descTable(frame, table):

try:

sql = "desc "+table

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall() #list of dict with one common key 'Field'

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def table2df(frame='', table='', columns='\*', condition='1=1'):

try:

sql = "select "+columns+" from "+table+" where "+condition

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def searchTableButtonClickEvent(frame,table,column,value):

try:

sql = "select \* from "+table+" where "+column+"='"+value+"'"

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def updateARow(rowDataList,param): #rowDataList is a series

#if there are multiple comboboxes for a table

cbocolslstupdate = []

for i in range(len(param['cbo'])):

v = param['cbo'][i]

v = v[v.find('.')+1:]

cbocolslstupdate.append(v)

try:

frameUpdateTable = Toplevel() #create a pop-up window

frameUpdateTable.title("Update Table")

frameUpdateTable.geometry()

frameUpdateTable.configure(background='yellow')

r = 0

for colname in rowDataList.index:

pk = param['pk'][0]

#print(i,' -- ',rowDataList[colname])

Label(frameUpdateTable, text=colname, bg='yellow', fg='black') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

if colname==pk:

globals()[colname] = Label(frameUpdateTable,text=rowDataList[colname])

else:

globals()[colname] = Entry(frameUpdateTable)

globals()[colname].insert(INSERT, rowDataList[colname])

globals()[colname].grid(row=r, column=2, padx=15, pady=5)

#if control is a combobox dropdown pickup

#----------------------------------------

# cbotbl = cbo[:cbo.find('.')]

# cbocol = cbo[cbo.find('.')+1:]

#if cbocol == colname:

if colname in cbocolslstupdate:

i = cbocolslstupdate.index(colname)

cboupdate = param['cbo'][i]

cbotblupdate = cboupdate[:cboupdate.find('.')]

cbocolupdate = cboupdate[cboupdate.find('.')+1:]

colvalsupdate = []

try:

sql = "select "+cbocolupdate+" from "+cbotblupdate+" order by "+cbocolupdate

cursor.execute(sql)

data = cursor.fetchall()

dfcboupdate = pd.DataFrame(data)

#print(dfcbo)

colvalsupdate = dfcboupdate.iloc[:,0].tolist()

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#globals()[colname+'CBO'] = Combobox(insertDataInputFRM)

globals()[colname+'updateCBO'] = Combobox(frameUpdateTable, name=cbocolupdate) # !important to name the combobox for accessing it later

globals()[colname+'updateCBO'].grid(row=r, column=4, padx=15, pady=5)

globals()[colname+'updateCBO']['values'] = colvalsupdate

def funcSelectedEvent(event):

'''

#print(event.widget)

print(event.widget.\_name) # use name assigned to the combobox

'''

#val = globals()[colname+'CBO'].get()

val = globals()[event.widget.\_name+'updateCBO'].get()

globals()[event.widget.\_name].delete(0, END)

globals()[event.widget.\_name].insert(INSERT, val)

globals()[colname+'updateCBO'].bind("<<ComboboxSelected>>", funcSelectedEvent)

r += 1

def updateTableSubmitButtonClickEvent():

#pass

table = param['table'][0]

pk = param['pk'][0]

sql = "UPDATE "+table+" SET "

for colname in rowDataList.index:

if colname==pk:

pkval = globals()[colname].cget("text") #to read the text of a label

condition = " where "+colname+"='"+pkval+"'"

else:

sql += colname+"='"+globals()[colname].get()+"'," #to read the text of an entry

sql = sql[:-1]

sql += " "+condition

print(sql)

sqlQueryExecution(frameUpdateTable, sql) #execute update query

#refresh df and then pandastable to show updated values

df = table2df(frameUpdateTable, table)

#createPandasTableFRM(frameUpdateTable, df, param)

global pandasTableFRM

pt = Table(pandasTableFRM, dataframe=df)

pt.cellbackgr = 'yellow'

pt.grid()

pt.show()

updateTableSubmitButton = Button(frameUpdateTable, text="Update", command=updateTableSubmitButtonClickEvent)

updateTableSubmitButton.grid(row=r, column=2, padx=10, pady=10)

def deleteRowSubmitButtonClickEvent():

#pass

table = param['table'][0]

pk = param['pk'][0]

sql = "DELETE FROM "+table

for colname in rowDataList.index:

if colname==pk:

pkval = globals()[colname].cget("text") #to read the text of a label

condition = " where "+colname+"='"+pkval+"'"

sql += " "+condition

print(sql)

sqlQueryExecution(frameUpdateTable, sql) #execute update query

#refresh df and then pandastable to show updated values

df = table2df(frameUpdateTable, table)

#createPandasTableFRM(frameUpdateTable, df, param)

global pandasTableFRM

pt = Table(pandasTableFRM, dataframe=df)

pt.cellbackgr = 'yellow'

pt.grid()

pt.show()

deleteRowSubmitButton = Button(frameUpdateTable, text="DELETE", command=deleteRowSubmitButtonClickEvent)

deleteRowSubmitButton.grid(row=r, column=3, padx=10, pady=10)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frameUpdateTable)

#-----------------------------------------------------------------------------

def createColumnCheckBoxesFRM(insertframe, table, df, r=2, col=0):

#pandastable frame r=1 c=0

#colcheckboxes frame r=2 c=0

#insertDataInput frame r=0 c=1

global tablesCBO

w, h = insertframe.winfo\_screenwidth()-120, insertframe.winfo\_screenheight()-160

w = w\*20/100

numOfCols = len(df.columns.tolist())

frame = Frame(insertframe, width=w)

#frame.grid(row=r, column=0, columnspan=numOfCols, sticky='NW', padx=15, pady=15) #NW-top left

#frame.grid(row=r, column=0, sticky='NW', padx=15, pady=15) #NW-top left

# r=1 c=0 colspan=2

frame.grid(row=r, column=0, columnspan=2, sticky='NW', padx=15, pady=15) #NW-top left

r = 1

col = 0

for c in df.columns.tolist():

globals()[c+'VAR'] = IntVar()

globals()[c+'VAR'].set(0)

globals()[c+'CHK'] = ttk.Checkbutton(frame,variable=globals()[c+'VAR'],text=c, onvalue=1, offvalue=0)

globals()[c+'CHK'].grid(column=col, row=r, sticky='W')

col+=1

r+=1

def getCheckedBoxesButtonClickEvent(df):

checkedColumnsLST = []

for c in df.columns.tolist():

if globals()[c+'VAR'].get()==1:

checkedColumnsLST.append(c)

cols=''

for c in checkedColumnsLST:

cols += c + ","

cols = cols[:-1]

df = table2df(insertframe, table, cols)

createPandasTableFRM(insertframe, df)

getCheckedBoxesButton = Button(frame, text='Submit List of Selected Columns', \

command=lambda: getCheckedBoxesButtonClickEvent(df))

getCheckedBoxesButton.grid(row=r, column=0, columnspan=numOfCols, sticky='EW')

for widget in frame.winfo\_children():

widget.grid(padx=0, pady=3)

#=============================================================================

def createPandasTableFRM(insertframe, df, param={}, r=1, c=0):

global pandasTableFRM

original\_r = r

original\_c = c

w, h = insertframe.winfo\_screenwidth()-120, insertframe.winfo\_screenheight()-160

w = w\*50/100

pandasTableFRM = Frame(insertframe, height=h, width=w)

pandasTableFRM.grid(row=r, column=c, sticky='NW')

pt = Table(pandasTableFRM, dataframe=df, showtoolbar=True, width=w, showstatusbar=True)

pt.cellbackgr = 'yellow'

pt.grid()

pt.show()

#for insert disable the following event handling; enable it for update only

def leftButtonClickEvent(event): #left-button click event handling

rowclicked = pt.get\_row\_clicked(event)

rowDataList = pt.model.df.loc[rowclicked] #Series

updateARow(rowDataList,param)

pt.rowheader.bind('<Button-1>',leftButtonClickEvent)

#=============================================================================

def extra():

'''

if len(args)>0:

print("Parameters #2: ",args," Total: ",len(args))

print("Table #2: ",args[0][1])

'''

def tablesCBOSelectedEvent(event):

table = event.widget.get()

df = table2df(root,table)

columnFindCBO['values'] = df.columns.tolist()

columnReplaceCBO['values'] = df.columns.tolist()

dfColumns = descTable(root,table)

#setPrimaryKey(dfColumns)

setPrimaryKey(table)

pandasTableFRM.destroy()

createPandasTableFRM(root, df)

createColumnCheckBoxesFRM(root, df)

tablesCBO.bind("<<ComboboxSelected>>", tablesCBOSelectedEvent)

#create two more frames within root frame

createPandasTableFRM(root, df)

createColumnCheckBoxesFRM(root, df)

#-------------------------start a new frame here

#CRUD controls

r = 0

def insertIntoTableButtonClickEvent():

print("START #1")

try:

table = tablesCBO.get()

df = descTable(root,table)

df = df['Field']

frameInsertIntoTable = Toplevel()

frameInsertIntoTable.title("Insert Row Into Table")

frameInsertIntoTable.geometry()

frameInsertIntoTable.configure(background='yellow')

total\_rows = df.shape[0]

r = 0

for i in range(total\_rows):

'''

colname = df[i]

Label(frameInsertIntoTable, text=colname, bg='yellow', fg='black') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

e = Entry(frameInsertIntoTable)

e.grid(row=r, column=3, padx=15, pady=5)

e.insert(INSERT, '')

'''

colname = df[i]

Label(frameInsertIntoTable, text=colname, bg='yellow', fg='black') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

globals()[colname+'TXT'] = Entry(frameInsertIntoTable)

globals()[colname+'TXT'].grid(row=r, column=3, padx=15, pady=5)

globals()[colname+'TXT'].insert(INSERT, '')

r += 1

def insertIntoTableSubmitButtonClickEvent():

print("START #2")

vals = ''

for i in range(total\_rows):

colname = df[i]

vals += globals()[colname+'TXT'].get('1.0','end-1c')

msg = vals

tk.messagebox.showinfo("MESSAGE", msg, parent=frameInsertIntoTable)

'''

def parentCBOSelectedEvent(event):

#substring the combobox name to exclude the leading '.' to get its corresponding text field name

txt = str(event.widget)[1:]

globals()[txt+'TXT'].delete('1.0', END)

globals()[txt+'TXT'].insert(INSERT, event.widget.get())

vals.append(globals()[df.loc[i,'mysqlcolumnname'].strip()+'TXT'].get('1.0','end-1c'))

cc1 = childdropdown.split('.')

sql = "SELECT "+cc1[i+1].strip()+" as v FROM "+cc1[i].strip()

sql += " WHERE "+txt+"='"+globals()[cc[i+1].strip()+'TXT'].get('1.0','end-1c')+"'"

globals()[colname+'TXT'].delete('1.0', END)

globals()[colname+'TXT'].insert(INSERT, newid)

if ctrl=='dropdown':

cursor.execute("SELECT DISTINCT "+lookupcol+" FROM "+lookuptbl)

lookupvals=[]

for val in cursor.fetchall():

lookupvals.append(val[lookupcol])

n = tk.StringVar()

globals()[colname+'CBO'] = ttk.Combobox(frameInsertData, name=colname, width=30, textvariable = n)

globals()[colname+'CBO']['values'] = lookupvals

globals()[colname+'CBO'].grid(row=r, column = 4)

globals()[colname+'CBO'].name=colname

def cboSelectedEvent(event):

txt = str(event.widget)[1:]

globals()[txt+'TXT'].delete('1.0', END)

globals()[txt+'TXT'].insert(INSERT, event.widget.get())

#child columns, if any

d = df[df['mysqlcolumnname']==txt]

d.index=np.arange(0,d.iloc[:,0].count())

childcols = str(d.loc[0,'childcolumn']).strip()

lookuptbl = str(d.loc[0,'lookupmysqltable']).strip()

lookupcol = str(d.loc[0,'lookupmysqlcolumn']).strip()

lookupcolcond = str(d.loc[0,'lookupmysqlcolumncondition']).strip()

globals()[colname+'CBO'].bind("<<ComboboxSelected>>", cboSelectedEvent)

'''

pass

insertIntoTableSubmitButton = Button(frameInsertIntoTable, text="Insert into table", command=insertIntoTableSubmitButtonClickEvent)

insertIntoTableSubmitButton.grid(row=r, column=3, padx=10, pady=10)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frameInsertIntoTable)

insertIntoTableButton = Button(frame, text='Insert', width=20, command=insertIntoTableButtonClickEvent)

insertIntoTableButton.grid(row=r, column=5, padx=20, pady=5, sticky=E)

r += 2

def searchTableButtonClickEventStart():

table = tablesCBO.get()

column = columnFindCBO.get()

value = findWhatTXT.get()

df = searchTableButtonClickEvent(root,table,column,value)

createPandasTableFRM(root, df)

searchTableButton = Button(frame, text='Search Table', width=20, command=searchTableButtonClickEventStart)

searchTableButton.grid(row=r, column=5, padx=20, pady=5, sticky=E)

def deleteRowsButtonClickEventStart():

table = tablesCBO.get()

column = columnFindCBO.get()

value = findWhatTXT.get()

df = searchTableButtonClickEvent(root,table,column,value)

createPandasTableFRM(root, df)

deleteRowsButton = Button(frame, text='Delete Rows', width=20, command=deleteRowsButtonClickEventStart)

deleteRowsButton.grid(row=r, column=6, padx=20, pady=5, sticky=E)

r += 1

def updateMultipleRowsButtonClickEventStart():

table = tablesCBO.get()

column = columnFindCBO.get()

find = findWhatTXT.get()

replace = replaceWithTXT.get()

opfind = operatorFindCBO.get()

#sql = "UPDATE "+table+" SET "+columnreplace+"='"+replace+"' WHERE "+columnfind+"='"+find+"'"

#['=','<>','>','<','>=','<=','between','like','in']

#if opfind in ['=','<>','>','<','>=','<=']:

# sql = "UPDATE "+table+" SET "+columnreplace+"='"+replace+"' WHERE "+columnfind+"='"+find+"'"

sql = "UPDATE "+table+" SET "+columnreplace+"='"+replace+"' WHERE "+columnfind+" "+opfind+" "+find

sqlQueryExecution(root, sql)

updateMultipleRowsButton = Button(frame, text='Update Multiple Rows', width=20, command=updateMultipleRowsButtonClickEventStart)

updateMultipleRowsButton.grid(row=r, column=5, padx=20, pady=5, sticky=E)

'''

sqlupdate += " UPDATE "+cc[i].strip()+" SET "+cc[i+1].strip()+"="

sqlupdate += globals()[expupdateothertblcol+'TXT'].get('1.0','end-1c')

sqlupdate += " WHERE "+keyupdateothertblcol+"="+globals()[keyupdateothertblcol+'TXT'].get('1.0','end-1c')

#print(sqlupdate)

#cursor.execute(sql)

'''

r += 1

# General SQL Query Execution

Label(frame, text='SQL Query:').grid(row=r, column=0, sticky=W)

sqlQueryTXT = Entry(frame)

sqlQueryTXT.grid(row=r, column=1, columnspan=8, sticky=EW)

r += 1

def sqlQueryExecutionButtonClickEventStart():

sql = sqlQueryTXT.get()

sqlQueryExecution(root, sql)

sqlQueryExecutionButton = Button(frame, text='SQL Query Execution', width=20, command=sqlQueryExecutionButtonClickEventStart)

sqlQueryExecutionButton.grid(row=r, column=8, padx=20, pady=5, sticky=E)

#=============================================================================

def createInsertDataInputFRM(insertframe, table, param={}, r=0, c=1):

#if there are multiple comboboxes for a table

cbocolslst = []

for i in range(len(param['cbo'])):

v = param['cbo'][i]

v = v[v.find('.')+1:]

cbocolslst.append(v)

#print(v)

#print(cbocolslst)

original\_r = r

original\_c = c

w, h = insertframe.winfo\_screenwidth()-120, insertframe.winfo\_screenheight()-160

w = w\*40/100

insertDataInputFRM = Frame(insertframe, height=h, width=w)

insertDataInputFRM.grid(row=r, column=c, rowspan=2, sticky=W)

insertDataInputFRM.configure(background='yellow')

try:

df = descTable(insertDataInputFRM,table)

df = df['Field']

total\_rows = df.shape[0]

r = 1

for i in range(total\_rows):

'''

colname = df[i]

Label(insertDataInputFRM, text=colname, bg='yellow', fg='black') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

e = Entry(insertDataInputFRM)

e.grid(row=r, column=3, padx=15, pady=5)

e.insert(INSERT, '')

r += 1

'''

colname = df[i]

if colname == 'id':

pass

else:

Label(insertDataInputFRM, text=colname, bg='yellow', fg='black') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

globals()[colname+'TXT'] = Entry(insertDataInputFRM)

globals()[colname+'TXT'].grid(row=r, column=3, padx=15, pady=5)

globals()[colname+'TXT'].insert(INSERT, '')

#if control is a combobox dropdown pickup

#----------------------------------------

# cbotbl = cbo[:cbo.find('.')]

# cbocol = cbo[cbo.find('.')+1:]

#if cbocol == colname:

if colname in cbocolslst:

i = cbocolslst.index(colname)

cbo = param['cbo'][i]

cbotbl = cbo[:cbo.find('.')]

cbocol = cbo[cbo.find('.')+1:]

print("cbo: ",cbo," cbotbl: ",cbotbl," cbocol: ",cbocol)

colvals = []

try:

sql = "select "+cbocol+" from "+cbotbl+" order by "+cbocol

print(sql)

cursor.execute(sql)

data = cursor.fetchall()

dfcbo = pd.DataFrame(data)

#print(dfcbo)

colvals = dfcbo.iloc[:,0].tolist()

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=insertDataInputFRM)

#globals()[colname+'CBO'] = Combobox(insertDataInputFRM)

globals()[colname+'CBO'] = Combobox(insertDataInputFRM, name=cbocol) # !important to name the combobox for accessing it later

globals()[colname+'CBO'].grid(row=r, column=4, padx=15, pady=5)

globals()[colname+'CBO']['values'] = colvals

def funcSelectedEvent(event):

'''

#print(event.widget)

print(event.widget.\_name) # use name assigned to the combobox

'''

#val = globals()[colname+'CBO'].get()

val = globals()[event.widget.\_name+'CBO'].get()

globals()[event.widget.\_name+'TXT'].delete(0, END)

globals()[event.widget.\_name+'TXT'].insert(INSERT, val)

# autofill

afv = ''

for afvi in range(len(param['autofill'])):

afv = param['autofill'][afvi]

cboname = afv[:afv.find('#')]

#print("event.widget.\_name: ",event.widget.\_name," cboname+'CBO': ",cboname+'CBO')

if event.widget.\_name == cboname:

autofill = afv[afv.find('#')+1:]

autofilltbl = autofill[:autofill.find('.')]

autofillcol = autofill[autofill.find('.')+1:]

try:

sql = "select "+autofillcol+" from "+autofilltbl

sql += " where "+event.widget.\_name

sql += " = '"+val+"'"

print(sql)

cursor.execute(sql)

autofillval = cursor.fetchone()[autofillcol]

print(autofillval)

globals()[autofillcol+'TXT'].delete(0, END)

globals()[autofillcol+'TXT'].insert(INSERT, autofillval)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=insertDataInputFRM)

globals()[colname+'CBO'].bind("<<ComboboxSelected>>", funcSelectedEvent)

r += 1

def insertIntoTableSubmitButtonClickEvent():

#pass

vals = ''

cols = ''

for i in range(total\_rows):

colname = df[i]

if colname == 'id':

pass

else:

cols += df[i]+","

vals += "'"+globals()[colname+'TXT'].get()+"',"

cols = cols[:-1]

vals = vals[:-1]

sql = " INSERT INTO "+ table +"("+ cols +") VALUES("+ vals +")"

'''

print(sql)

cursor.execute(sql)

conn.commit()

print(vals)

msg = "New row inserted into the table: (" + vals + ")"

tk.messagebox.showinfo("MESSAGE", msg, parent=insertDataInputFRM)

'''

sqlQueryExecution(insertDataInputFRM, sql)

#refresh df and then pandastable to show updated values

dfpt = table2df(insertDataInputFRM, table)

#createPandasTableFRM(frameUpdateTable, df, param)

global pandasTableFRM

pt = Table(pandasTableFRM, dataframe=dfpt)

pt.cellbackgr = 'yellow'

pt.grid()

pt.show()

insertIntoTableSubmitButton = Button(insertDataInputFRM, text="Insert into table", command=insertIntoTableSubmitButtonClickEvent)

insertIntoTableSubmitButton.grid(row=r, column=3, padx=10, pady=10)

except conn.Error as e:

print("ERROR in createInsertDataInputFRM")

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=insertDataInputFRM)

#=============================================================================

# child frames

#=============================================================================

def createInsertChildFrames(insertrootframe, param):

table = param['table'][0]

pk = param['pk'][0]

cbos = ', '.join(param['cbo']) #explode list to comma separated strings

#common frame r=0 c=0

r=0

col = 0

frame = Frame(insertrootframe)

frame.grid(row=r, column=col, sticky='W', padx=15, pady=15) #W-left, E-right, N-top, S-bottom

Label(frame, text='TABLE:', bg='black', fg='white').grid(row=0, column=0, sticky=W)

Label(frame, text=table, bg='black', fg='white').grid(row=0, column=1, sticky=W)

Label(frame, text='PRIMARY KEY:', bg='black', fg='white').grid(row=1, column=0, sticky=W)

Label(frame, text=pk, bg='black', fg='white').grid(row=1, column=1, sticky=W)

Label(frame, text='LOOKUP VALUES (CBO):', bg='black', fg='white').grid(row=2, column=0, sticky=W)

Label(frame, text=cbos, bg='black', fg='white').grid(row=2, column=1, sticky=W)

frame.configure(background="black",bd=-10)

df = table2df(frame,table)

dfColumns = descTable(frame,table)

#pandastable frame r=1 c=0

r+=1

createPandasTableFRM(insertrootframe, df, param, r, col)

#insertDataInput frame r=0 c=1

r=0

col = 1

createInsertDataInputFRM(insertrootframe, table, param, r, col)

#colcheckboxes frame r=2 c=0

#r+=1

r = 2

col = 0

colspan = 2

createColumnCheckBoxesFRM(insertrootframe, table, df, r, col)

for widget in frame.winfo\_children():

widget.grid(padx=0, pady=5)

#=============================================================================

# root frame

#=============================================================================

def createInsertRootFrame(rootframe, param):

w, h = rootframe.winfo\_screenwidth()-50, rootframe.winfo\_screenheight()-150

#insertrootframe = Frame(rootframe)

#insertrootframe.grid(row=0, column=0, sticky='W', padx=15, pady=15) #W-left, E-right, N-top, S-bottom

insertrootframe = Toplevel(rootframe)

insertrootframe.geometry("%dx%d+15+60" % (w, h))

insertrootframe.configure(background="black",bd=-10)

insertrootframe.title("INSERT NEW DATA")

table = param['table'][0]

df = table2df(insertrootframe,table)

createInsertChildFrames(insertrootframe, param)

#=============================================================================

# standalone start

#=============================================================================

param = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory']}

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

createInsertRootFrame(rootframe,param)

#=============================================================================

\_menu.py

#====================================================================================

#import standard Python library files

#-------------------------------------

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#import custom library files

#-------------------------------------

import \_master #for master table data handling

import \_edit #for updating tables data

import \_report #for selecting tables data

import \_transaction #for transaction table data handling

import \_dataAnalysisAndPlot #for data analysis & plotting

import \_database #for database management

import \_createMySQLTablesWithTestData #for creating MySQL tables with test data

import \_dataExportImport #

import \_reportCustom #

import \_dataAnalysisAndPlotCustom #

import \_pdfFileGeneration #

import \_about #

#====================================================================================

def donothing():

pass

#====================================================================================

def menu(rootframe):

#--------------------------------------------------------------------

menubar = Menu(rootframe)

#--------------------------------------------------------------------

newmenu = Menu(menubar, tearoff=0)

newmnudrug = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory',

'unit.unit'],'autofill':['drugcategory#drugcategory.gstrate']}

newmenu.add\_command(label="Medicine",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnudrug))

newmnuemployee = {'table':['employee'],'pk':['serviceid'],'cbo':['employeecategory.employeecategory','gender.gender','country.country'],'autofill':[]}

newmenu.add\_command(label="Employee",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnuemployee))

newmnusupplier = {'table':['supplier'],'pk':['suppliercode'],'cbo':['suppliercategory.suppliercategory'],'autofill':[]}

newmenu.add\_command(label="Supplier",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnusupplier))

newmenu.add\_separator()

newmnudrugcat = {'table':['drugcategory'],'pk':['drugcategoryid'],'cbo':[''],'autofill':[]}

newmenu.add\_command(label="Medicine Category",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnudrugcat))

newmnuemployeecat = {'table':['employeecategory'],'pk':['employeecategoryid'],'cbo':[''],'autofill':[]}

newmenu.add\_command(label="Employee Category",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnuemployeecat))

newmnusuppliercat = {'table':['suppliercategory'],'pk':['suppliercategoryid'],'cbo':[''],'autofill':[]}

newmenu.add\_command(label="Supplier Category",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnusuppliercat))

newmnucustomer = {'table':['customer'],'pk':['customerid'],'cbo':['customercategory.customercategory','gender.gender'],'autofill':[]}

newmenu.add\_command(label="Customer",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnucustomer))

newmnucustomercat = {'table':['customercategory'],'pk':['customercategoryid'],'cbo':[''],'autofill':[]}

newmenu.add\_command(label="Customer Category",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnucustomercat))

newmnubusinessname = {'table':['businessname'],'pk':['businessname'],'cbo':[''],'autofill':[]}

newmenu.add\_command(label="Business Name",

command=lambda: \_master.createInsertRootFrame(rootframe, newmnubusinessname))

menubar.add\_cascade(label="MASTER", menu=newmenu)

#--------------------------------------------------------------------

editmenu = Menu(menubar, tearoff=0)

editmenutabledata = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory'],'autofill':['']}

editmenu.add\_command(label="Edit Database Table Data",

command=lambda: \_edit.createEditRootFrame(rootframe, editmenutabledata))

menubar.add\_cascade(label="EDIT", menu=editmenu)

#--------------------------------------------------------------------

transmenu = Menu(menubar, tearoff=0)

paramitempurchase = {

'table' : ['purchase'],

'pk' : ['purchaseid'],

'dateColumn' : 'pdate',

'pickup' : [

{

'masterCBO' : ['drug.drugcategory#drug.drugname'],

'masterLookupItems' : ['drug.drugname'],

'getvalfromothertableonlookupcboevent' : [],

'masterPrimaryKeys' : ['drug.drugcode'],

'masterAutofillValues' : ['drug.drugcode','drug.drugaverageprice','drug.drugtotalstock',

'drug.unit','drug.gstrate'],

'condition' : [],

},

{

'masterCBO' : ['supplier.suppliercategory#supplier.suppliername'],

'masterLookupItems' : ['supplier.suppliername'],

'getvalfromothertableonlookupcboevent' : [],

'masterPrimaryKeys' : ['supplier.supplierid'],

'masterAutofillValues' : ['supplier.suppliercode','supplier.suppliermobile'],

'condition' : [],

},

],

'transItems' : ['invoice','drugquantity','drugprice','freight'],

'expressions' : [

'amount = float(drugquantity.get()) \* float(drugprice.get())',

'gst = round(float(amount.get()) \* float(gstrate.get()) / 100.00, 2)',

'netamount = float(amount.get()) + float(gst.get())',

'grandamount = float(netamount.get()) + float(freight.get())',

'drugminsaleprice = round(float(grandamount.get()) / float(drugquantity.get()),2)',

'drugstock = float(drugquantity.get())',

],

'displayonlynoinsert' : [],

'invisible' : [

'suppliercategory = suppliercategory.get()',

'suppliercode = suppliercode.get()',

'suppliername = suppliername.get()',

'suppliermobile = suppliermobile.get()',

'drugcategory = drugcategory.get()',

'drugcode = drugcode.get()',

'drugname = drugname.get()',

'unit = unit.get()',

'gstrate = gstrate.get()',

],

'prefilled' : ['freight=\"0.00\"'], #to set 0.00 as default inital value in the textfield 'freight'

'masterUpdates' : [

'drug.drugtotalstock=float(drugtotalstock.get())+float(drugquantity.get())',

# to reset average stock price - if not FIFO/LIFO

# 'masterUpdates'

# 'item.itemaverageprice=round((float(itemaverageprice.get())+float(itemminsaleprice.get()))/2, 2)',

],

}

transmenu.add\_command(label="Item Purchase", command=lambda: \_transaction.createTransRootFrame(rootframe, paramitempurchase))

transmenu.add\_separator()

# use 'masterLookupItems' : ['item.itemname']

# so as to update the total quantity in items table

# in addition to the stock updation in itempurchase table

paramitemsale = {

'table' : ['sale'],

'pk' : ['saleid'],

'dateColumn' : 'sdate',

'pickup' : [

{

'masterCBO' : ['purchase.drugcategory#purchase.drugname'],

'masterLookupItems' : ['purchase.drugname'],

'getvalfromothertableonlookupcboevent' : [],

'masterPrimaryKeys' : ['purchase.purchaseid'],

'masterAutofillValues' : ['purchase.drugminsaleprice','purchase.drugcode',

'purchase.drugstock','purchase.unit',

'purchase.gstrate'],

'condition' : ['"purchase.drugstock > 0"'],

},

{

'masterCBO' : ['customer.customercategory#customer.customercode'],

'masterLookupItems' : ['customer.customercode'],

'getvalfromothertableonlookupcboevent' : [],

'masterPrimaryKeys' : ['customer.customerid'],

'masterAutofillValues' : ['customer.customername','customer.customermobile'],

'condition' : [],

},

],

'transItems' : ['invoice','drugquantity','freight','marginpercent'],

'expressions' : [

'drugprice = round(float(drugminsaleprice.get()) + float(drugminsaleprice.get())\*float(marginpercent.get())/100,2)',

'amount = round(float(drugprice.get())\*float(drugquantity.get()),2)',

'netamount = float(amount.get())+float(freight.get())'

],

'displayonlynoinsert' : [],

'invisible' : [

'customercategory = customercategory.get()',

'customercode = customercode.get()',

'customername = customername.get()',

'customermobile = customermobile.get()',

'drugcategory = drugcategory.get()',

'drugcode = drugcode.get()',

'drugname = drugname.get()',

'unit = unit.get()',

],

'prefilled' : ['freight=\"0.00\"','marginpercent=\"2\"'],

'masterUpdates' : [

'purchase.drugstock=float(drugstock.get())-float(drugquantity.get())',

'drug.drugtotalstock=float(drugtotalstock.get())-float(drugquantity.get())',

],

}

transmenu.add\_command(label="Item Sale", command=lambda: \_transaction.createTransRootFrame(rootframe, paramitemsale))

menubar.add\_cascade(label="TRANSACTIONS", menu=transmenu)

#====================================================================

reportmenu = Menu(menubar, tearoff=0)

rptparam1 = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory']}

reportmenu.add\_command(label="Reports - All", command=lambda: \_report.createSelectRootFrame(rootframe, rptparam1)) #use lambda to pass arguments to the function

reportmenu.add\_separator()

reportmenu.add\_command(label="All Drugs", command=lambda: \_reportCustom.alldrugs(rootframe))

reportmenu.add\_command(label="Drug Details", command=lambda: \_reportCustom.drugdetails(rootframe))

reportmenu.add\_command(label="Custom SQL Report", command=lambda: \_reportCustom.sql(rootframe))

reportmenu.add\_command(label="Available Drugs", command=lambda: \_reportCustom.availabledrugs(rootframe))

reportmenu.add\_command(label="All Drug Categories", command=lambda: \_reportCustom.alldrugcategories(rootframe))

reportmenu.add\_command(label="3 Most Sold Drugs", command=lambda: \_reportCustom.threemostsolddrug(rootframe))

reportmenu.add\_command(label="Company Name", command=lambda: \_reportCustom.companyname(rootframe))

reportmenu.add\_separator()

reportmenu.add\_command(label="Generate PDF", command=lambda: \_pdfFileGeneration.invoicePurchase(rootframe))

menubar.add\_cascade(label="REPORT", menu=reportmenu)

#====================================================================

damenu = Menu(menubar, tearoff=0)

daparam1 = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory']}

damenu.add\_command(label="Data Analysis - Complete",

command=lambda: \_dataAnalysisAndPlot.createDataAnalysisRootFrame(rootframe, daparam1))

damenu.add\_separator()

damenu.add\_command(label="GST Rates Of Categories",command=lambda:

\_dataAnalysisAndPlotCustom.plotCategoryGSTRate(rootframe))

damenu.add\_command(label="Total Category Sales",command=lambda:

\_dataAnalysisAndPlotCustom.plotCategorySales(rootframe))

damenu.add\_command(label="Drug Sales with Time",command=lambda:

\_dataAnalysisAndPlotCustom.plotTimeSales(rootframe))

damenu.add\_command(label="Sales Of Drugs",command=lambda:

\_dataAnalysisAndPlotCustom.plotDrugSales(rootframe))

damenu.add\_command(label="Compare 2 Drugs",command=lambda:

\_dataAnalysisAndPlotCustom.plot2TimeDrugSales(rootframe))

damenu.add\_command(label="Compare Category Sales",command=lambda:

\_dataAnalysisAndPlotCustom.plotMultipleTimeCategorySales(rootframe))

#damenu.add\_command(label="Data Analysis - Filter (Boolean Indexing)", command=lambda: dataAnalysisFilterData.dataAnalysis(db))

menubar.add\_cascade(label="DATA ANALYSIS", menu=damenu)

#====================================================================

dbmenu = Menu(menubar, tearoff=0)

dbmenu.add\_command(label="Export-Import Data", command=lambda: \_dataExportImport.index(rootframe))

dbmenu.add\_separator()

#newdatabase = "d1234"

#dbmenu.add\_command(label="Create Database", command=lambda: \_database.createNewDatabase(rootframe,newdatabase))

dbmenu.add\_command(label="Create Table", command=lambda: \_createMySQLTablesWithTestData.createTablesWithTestData(rootframe))

dbmenu.add\_command(label="Backup Database", command=lambda: \_database.backupDatabase(rootframe))

dbmenu.add\_command(label="Alter Table", command=lambda: \_database.alterTable(rootframe))

'''

dbmenu.add\_separator()

dbmenu.add\_command(label="Restore Database", command=lambda: \_database.restoreDatabase(rootframe))

dbmenu.add\_command(label="Reset Database", command=lambda: \_database.resetDatabase(rootframe))

dbmenu.add\_command(label="Drop Table", command=donothing)

'''

menubar.add\_cascade(label="DATABASE", menu=dbmenu)

#====================================================================

helpmenu = Menu(menubar, tearoff=0)

helpmenu.add\_command(label="About", command= \_about.manual)

helpmenu.add\_command(label="Manual & Guide", command= \_about.manual)

helpmenu.add\_command(label="Help", command= \_about.help)

helpmenu.add\_command(label="Contact", command= \_about.contact)

menubar.add\_cascade(label="HELP", menu=helpmenu)

#====================================================================

def clearcursorandconnection():

#if cursor.open:

# cursor.close()

if conn.open:

conn.close() #it will close its dependent cursor on its own

def exitapp():

#rootframe.quit() #NOT RECOMMENDED

clearcursorandconnection()

rootframe.destroy()

exitmenu = Menu(menubar, tearoff=0)

exitmenu.add\_command(label="Close Cursor & Connection", command=clearcursorandconnection)

exitmenu.add\_command(label="Exit Application", command=exitapp) #rootframe.destroy

menubar.add\_cascade(label="Exit", menu=exitmenu)

#====================================================================

return menubar

#====================================================================================

\_report.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

#-----------------------------------------------------------------------------

# pandastable frame r=1 c=0

# colcheckboxes frame r=2 c=0

# insertDataInput frame r=0 c=1

#-----------------------------------------------------------------------------

global tablesCBO

global pandasTableFRM

global colCheckboxesFRM

#-----------------------------------------------------------------------------

def sqlQueryExecution(frame, sql): #all SQL queries except SELECT & DESC

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute(sql)

conn.commit()

msg = "SUCCESS: SQL query executed successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#return msg

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

#return msg

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def showTablesInADatabase(frame):

#database name is globally accessible, so need not pass it on to this function

try:

cursor.execute("SHOW TABLES")

result = cursor.fetchall() #result: list of dictionary

tables=[]

for i in result:

tables.append(\*i.values()) #use \* to explode the dictionary so as to enlist values only

return tables

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def descTable(frame, table):

try:

sql = "desc "+table

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall() #list of dict with one common key 'Field'

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def table2df(frame='', table='', columns='\*', condition='1=1'):

try:

sql = "select "+columns+" from "+table+" where "+condition

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def table2dfSQLQuery(frame='', sql=''):

try:

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def searchTableButtonClickEvent(frame,table,column,value):

try:

sql = "select \* from "+table+" where "+column+"='"+value+"'"

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def updateARow(rowDataList): #rowDataList is a series

try:

frameUpdateTable = Toplevel() #create a pop-up window

frameUpdateTable.title("Update Table")

frameUpdateTable.geometry()

frameUpdateTable.configure(background='yellow')

r = 0

for colname in rowDataList.index:

#print(i,' -- ',rowDataList[colname])

Label(frameUpdateTable, text=colname, bg='yellow') \

.grid(row=r, column=1, padx=15, pady=5, sticky='E')

e = Entry(frameUpdateTable)

e.grid(row=r, column=2, padx=15, pady=5)

e.insert(INSERT, rowDataList[colname])

r += 1

def updateTableSubmitButtonClickEvent():

pass

updateTableSubmitButton = Button(frameUpdateTable, text="Update", command=updateTableSubmitButtonClickEvent)

updateTableSubmitButton.grid(row=r, column=2, padx=10, pady=10)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frameUpdateTable)

#-----------------------------------------------------------------------------

def createColumnCheckBoxesFRM(selectrootframe, table, df, r=1, col=1):

global tablesCBO

global colCheckboxesFRM

w, h = selectrootframe.winfo\_screenwidth()-120, selectrootframe.winfo\_screenheight()-160

w = w\*20/100

numOfCols = len(df.columns.tolist())

colCheckboxesFRM = Frame(selectrootframe, width=w)

#colCheckboxesFRM.grid(row=r, column=0, columnspan=numOfCols, sticky='NW', padx=15, pady=15) #NW-top left

colCheckboxesFRM.grid(row=r, column=col, sticky='NW', padx=15, pady=15) #NW-top left

r = 1

col = 0

for c in df.columns.tolist():

globals()[c+'VAR'] = IntVar()

globals()[c+'VAR'].set(0)

globals()[c+'CHK'] = ttk.Checkbutton(colCheckboxesFRM,variable=globals()[c+'VAR'],text=c, onvalue=1, offvalue=0)

globals()[c+'CHK'].grid(column=col, row=r, sticky='W')

col+=1

r+=1

def getCheckedBoxesButtonClickEvent(df):

checkedColumnsLST = []

for c in df.columns.tolist():

if globals()[c+'VAR'].get()==1:

checkedColumnsLST.append(c)

cols=''

for c in checkedColumnsLST:

cols += c + ","

cols = cols[:-1]

df = table2df(colCheckboxesFRM, table, cols)

createPandasTableFRM(selectrootframe, df) #!IMPORTANT - use selectrootframe

getCheckedBoxesButton = Button(colCheckboxesFRM, text='Submit List of Selected Columns', \

command=lambda: getCheckedBoxesButtonClickEvent(df))

getCheckedBoxesButton.grid(row=r, column=0, columnspan=numOfCols, sticky='EW')

for widget in colCheckboxesFRM.winfo\_children():

widget.grid(padx=0, pady=3)

#=============================================================================

def createPandasTableFRM(frame, df, r=0, c=1):

global pandasTableFRM

w, h = frame.winfo\_screenwidth()-120, frame.winfo\_screenheight()-160

w = w\*60/100

pandasTableFRM = Frame(frame, height=h, width=w)

pandasTableFRM.grid(row=r, column=c, sticky='NW')

pt = Table(pandasTableFRM, dataframe=df, showtoolbar=True, width=w, showstatusbar=True)

pt.cellbackgr = '#ffff66'

pt.grid()

pt.show()

#for insert disable the following event handling; enable it for update only

def leftButtonClickEvent(event): #left-button click event handling

rowclicked = pt.get\_row\_clicked(event)

rowDataList = pt.model.df.loc[rowclicked] #Series

updateARow(rowDataList)

pt.rowheader.bind('<Button-1>',leftButtonClickEvent)

#=============================================================================

def inputFrame(selectrootframe, param, r=0, c=0):

global tablesCBO

global colCheckboxesFRM

global pandasTableFRM

table = param['table'][0]

pk = param['pk'][0]

cbo = param['cbo'][0]

w, h = selectrootframe.winfo\_screenwidth()-120, selectrootframe.winfo\_screenheight()-160

w = w\*30/100

#frame = Frame(selectrootframe, height=h, width=w)

frame = Frame(selectrootframe, width=w)

frame.grid(row=r, column=c, rowspan=2, sticky='NW')#W-left, E-right, N-top, S-bottom

df = table2df(frame,table)

dfColumns = descTable(frame,table)

r=0

Label(frame, text='TABLES:').grid(row=r, column=0, sticky='NW')

tables = StringVar()

tablesCBO = Combobox(frame, name='tablecbo', width=27)#, textvariable=tablesCBOvar) #,width=30)

lookupvals = showTablesInADatabase(frame)

tablesCBO['values'] = lookupvals

tablesCBO.grid(row=r, column=1, sticky='NW')

if len(lookupvals)>0:

tablesCBO.current(0)

df = table2df(frame,lookupvals[0])

dfColumns = descTable(frame,lookupvals[0])

def tablesCBOSelectedEvent(event):

'''

r=0;c=0

inputFrame(selectrootframe,param,r,c)

r=0;c=1

createPandasTableFRM(selectrootframe,df,r,c)

r=1;c=1

createColumnCheckBoxesFRM(selectrootframe,table,df,r,c)

'''

# !Important - use "selectrootframe" frame and not this "frame"

table = event.widget.get()

df = table2df(selectrootframe,table)

columnFindCBO['values'] = df.columns.tolist()

dfColumns = descTable(selectrootframe,table)

pandasTableFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

colCheckboxesFRM.destroy()

createColumnCheckBoxesFRM(selectrootframe, table, df, 1, 1)

tablesCBO.bind("<<ComboboxSelected>>", tablesCBOSelectedEvent)

r += 1

Label(frame, text='PRIMARY KEY:').grid(row=r, column=0, sticky='NW')

Label(frame, text=pk).grid(row=r, column=1, sticky='NW')

r += 1

Label(frame, text='LOOKUP VALUES (CBO):').grid(row=r, column=0, sticky='NW')

Label(frame, text=pk).grid(row=r, column=1, sticky='NW')

r += 1

#sub frame #1

frame1 = Frame(frame, bg="#0f8729")

frame1.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

# search / Find What

r=0

Label(frame1, text='Find what:').grid(row=r, column=0, sticky='NW')

findWhatTXT = Entry(frame1, width=30)

findWhatTXT.grid(row=r, column=1, sticky='NW')

r += 1

Label(frame1, text='compare').grid(row=r, column=0, sticky='NW')

operatorfind = StringVar()

operatorFindCBO = Combobox(frame1, name='operatorFindCBO', width=10, textvariable=operatorfind)

operatorFindCBO['values'] = ['=','<>','>','<','>=','<=','between','like','in']

operatorFindCBO.grid(row=r, column=1, sticky='NW')

operatorFindCBO.current(0)

r += 1

Label(frame1, text='in column').grid(row=r, column=0, sticky='NW')

columnfind = StringVar()

columnFindCBO = Combobox(frame1, name='columnFindCBO', width=30, textvariable=columnfind)

#columnFindCBO['values'] = lookupvals

columnFindCBO.grid(row=r, column=1, sticky='NW')

r += 1

def searchTableButtonClickEventStart():

table = tablesCBO.get()

column = columnFindCBO.get()

value = findWhatTXT.get()

df = searchTableButtonClickEvent(frame1,table,column,value)

createPandasTableFRM(selectrootframe, df)

searchTableButton = Button(frame1, text='Search Table', width=20, command=searchTableButtonClickEventStart)

searchTableButton.grid(row=r, column=1, sticky='EW')

r += 1

#sub frame #2

# General SQL Query Execution

frame2 = Frame(frame)

frame2.grid(row=r, column=0, columnspan=2, sticky='NW') #W-left, E-right, N-top, S-bottom

Label(frame2, text='Raw SQL Query (Select only):').grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

sqlQueryTXT = Text(frame2, width=35, height=6)

sqlQueryTXT.grid(row=r, column=0, columnspan=2, sticky='NW')

r += 1

def sqlQueryExecutionButtonClickEventStart():

#for sql query use raw string

sql = sqlQueryTXT.get("1.0",'end-1c') #for TEXT use - textbox1.get("1.0",'end-1c')

#sqlQueryExecution(frame2, sql)

#print("SQL: ",sql)

#TO GET THE TABLE NAME FROM A SQL QUERY!!!!!!!!!!!!!!!

#sql="select \* from item where a=1";

idx1 = sql.find("from");

idx2 = sql[idx1+4:].strip().find(" ");

idx3 = sql.find(" ",idx1+4+idx2-1);

idx4=sql[idx1+4+idx2-1:].strip().find(" ");

table = sql[idx1+4+idx2-1:idx3]

#print(table)

df = table2dfSQLQuery(frame2, sql)

pandasTableFRM.destroy()

colCheckboxesFRM.destroy()

createPandasTableFRM(selectrootframe, df, 0, 1)

createColumnCheckBoxesFRM(selectrootframe, table, df, 1, 1)

sqlQueryExecutionButton = Button(frame2, text='SQL Query Execution', width=20, command=sqlQueryExecutionButtonClickEventStart)

sqlQueryExecutionButton.grid(row=r, column=1, sticky='EW')

for widget in frame.winfo\_children():

widget.grid(padx=10, pady=10)

for widget in frame1.winfo\_children():

widget.grid(padx=10, pady=10)

for widget in frame2.winfo\_children():

widget.grid(padx=10, pady=10)

#=============================================================================

# root frame

#=============================================================================

def createSelectRootFrame(rootframe, param):

w, h = rootframe.winfo\_screenwidth()-50, rootframe.winfo\_screenheight()-150

selectrootframe = Toplevel(rootframe)

selectrootframe.geometry("%dx%d+15+60" % (w, h))

selectrootframe.title("DATA REPORT")

table = param['table'][0]

df = table2df(selectrootframe,table)

r=0;c=0

inputFrame(selectrootframe,param,r,c)

r=0;c=1

createPandasTableFRM(selectrootframe,df,r,c)

r=1;c=1

createColumnCheckBoxesFRM(selectrootframe,table,df,r,c)

'''

#fire tablescbo combobox click-event programmatically

idx = lookupvals.index(table)

tablesCBO.current(idx)

tablesCBO.event\_generate('<<ComboboxSelected>>')

'''

#=============================================================================

# standalone start

#=============================================================================

param = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory']}

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

createSelectRootFrame(rootframe, param)

#=============================================================================

\_reportCustom.py

from \_reportDataAnalysisPlotCustomFunctions import \*

#-----------------------------------------------------------------------------

# User Defined Reports

#-----------------------------------------------------------------------------

# report 1

#-----------------------------------------------------------------------------

def alldrugs(rootframe):

frame = Toplevel()

sql = "select drugname from drug"

executeSelectQuery(frame, sql)

#-----------------------------------------------------------------------------

def availabledrugs(rootframe):

frame = Toplevel()

sql = "select distinct drugname from purchase"

executeSelectQuery(frame, sql)

#-----------------------------------------------------------------------------

def companyname(rootframe):

frame = Toplevel()

sql = "select businessname from businessname"

executeSelectQuery(frame, sql)

#-----------------------------------------------------------------------------

def alldrugcategories(rootframe):

frame = Toplevel()

sql = "select drugcategory from drugcategory"

executeSelectQuery(frame, sql)

#-----------------------------------------------------------------------------

def threemostsolddrug(rootframe):

frame = Toplevel()

sql = "select \* from sale order by drugquantity desc limit 3"

executeSelectQuery(frame, sql)

#-----------------------------------------------------------------------------

def drugdetails(rootframe):

frame = Toplevel()

sql = "select drugname from drug"

df = executeSelectQueryAndReturnDF(frame, sql)

# GUI

# drop-down/pick-up list

Label(frame, text=" ").grid(row=1, column=1, sticky='NE')

namevar = StringVar()

namelookupvalues = df['drugname'].tolist()

Label(frame, text="Drug Name").grid(row=2, column=1, sticky='NE')

nameCBO = Combobox(frame, name='namecbo', width=30, textvariable=namevar)

nameCBO.grid(row=2, column=2)

nameCBO['values'] = namelookupvalues

nameCBO.current(0)

def nameCBOSelectedEvent(event):

name = event.widget.get()

sql = "SELECT \* FROM drug where drugname='"+name+"'"

executeSelectQuery(frame, sql)

nameCBO.bind("<<ComboboxSelected>>", nameCBOSelectedEvent)

Label(frame, text=" ").grid(row=3, column=3, sticky='NE')

#-----------------------------------------------------------------------------

# report 3

#-----------------------------------------------------------------------------

def sql(rootframe):

frame = Toplevel()

Label(frame, text=" ").grid(row=1, column=1, sticky='NE')

Label(frame, text="SQL Query").grid(row=2, column=1, sticky='NE')

sqlTXT = Entry(frame, width=30)

sqlTXT.grid(row=2, column=2, sticky='NW')

def clickButtonEvent():

sql = sqlTXT.get()

executeSelectQuery(frame, sql)

Button(frame, text="Submit", command=clickButtonEvent).grid(row=3, column=2, sticky='NW')

Label(frame, text=" ").grid(row=4, column=3, sticky='NE')

#=============================================================================

# standalone start for code testing - to run this file independently

#=============================================================================

param = {'table':['drug'],'pk':['drugcode'],'cbo':['drugcategory.drugcategory']}

if \_\_name\_\_ == "\_\_main\_\_":

rootframe = Tk()

#createSelectRootFrame(rootframe, param)

#rptItemDetail(rootframe) # report 1

#rptItemDetailConditionalPickupList(rootframe) # report 2

rptItemDetailConditionalTextbox(rootframe) # report 3

#=============================================================================

\_reportDataAnalysisPlotCustomFunctions.py

# general functions for custom report, data analysis and plot files

from \_libraryAndDBConnection import \*

#-----------------------------------------------------------------------------

'''

def createPandasTableFRM(frame, df):

pandasTableFRM = Frame(frame)

pandasTableFRM.grid(row=0, column=0, sticky='NW')

pt = Table(pandasTableFRM, dataframe=df, showstatusbar=True) #width=w, height=h,

pt.cellbackgr = 'orange'

pt.grid()

pt.show()

'''

#-----------------------------------------------------------------------------

def createPandasTableFRM(frame, df):

w, h = frame.winfo\_screenwidth()-120, frame.winfo\_screenheight()-160

w = w\*60/100

'''

topFRM = Toplevel(height=h, width=w)

pandasTableFRM = Frame(topFRM)

'''

pandasTableFRM = Toplevel()

#pandasTableFRM.grid(row=1, column=1, sticky='NW')

pt = Table(pandasTableFRM, dataframe=df, showtoolbar=True, width=w, showstatusbar=True)

pt.cellbackgr = 'orange'

pt.grid()

pt.show()

#-----------------------------------------------------------------------------

def executeSelectQuery(frame, sql):

try:

#sql = "select \* from item, customer"

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

createPandasTableFRM(frame, df)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def executeSelectQueryAndReturnDF(frame, sql):

try:

cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df=pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frameDisplayData)

#-----------------------------------------------------------------------------

def executeSelectQueryForPlotAndDA(frame, sql, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol):

try:

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

#createPandasTableFRM(frame, df)

createPlotFRM(frame, df, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol) #ycol not required for 'hist'

createStatsFRM(frame, df)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def executeSelectQueryForMultiplePlotAndDA(frame, sql, charttype, title,

xlabel, ylabel, xticks, yticks, xcol, ycol, legend):

#Sql, legend and ycol are 3 lists...

print(sql[0])

print(sql[1])

df = []

try:

for i in range(2):

cursor.execute(sql[i])

data = cursor.fetchall()

df.append(pd.DataFrame(data))

#createPandasTableFRM(frame, df)

#print('\n\nDF: ',df[i])

createMultiplePlotFRM(frame, df, charttype, title, xlabel, ylabel, xticks, yticks, xcol, ycol, legend) #ycol not required for 'hist'

#createStatsFRM(frame, df)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#-----------------------------------------------------------------------------

def createPlotFRM(frame, df, charttype, title='', xlabel='', ylabel='', xticks='', yticks='', xcol='', ycol=''):

#plotFRM = Frame(frame)

#plotFRM.grid(row=0, column=1, sticky='NW')

plotFRM = Toplevel()

plt.clf()

fig = plt.figure(1)

x = df[xcol].tolist()

y = df[ycol].tolist()

print("X:",x," ",len(x))

print("Y:",y," ",len(y))

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.title(title)

plt.xticks(rotation = 90)

plt.tight\_layout()

if charttype=='plot':

#plt.figure(figsize=(50,60))

plt.plot(x,y)

elif charttype=='bar':

#plt.figure(figsize=(100,600))

plt.bar(x,y)

'''elif charttype=='hist':

plt.xticks(rotation = 90)

plt.hist(x,y)'''

canvas = FigureCanvasTkAgg(fig, master=plotFRM)

plot\_widget = canvas.get\_tk\_widget()

plot\_widget.grid(row=0, column=0)

#-----------------------------------------------------------------------------

def createMultiplePlotFRM(frame, df, charttype, title='', xlabel='', ylabel='', xticks='', yticks='', xcol='', ycol='', legend=''):

#plotFRM = Frame(frame)

#plotFRM.grid(row=0, column=1, sticky='NW')

plotFRM = Toplevel()

plt.clf()

fig = plt.figure(1)

firstycol = ycol[0]

secondycol = ycol[1]

x = df[0][xcol].tolist()

y1 = df[0][firstycol].tolist()

y2 = df[1][secondycol].tolist()

print("X:",x)

print("Y1:",y1)

print("Y2:",y2)

#plt.bar(x,y1)

#plt.bar(x,y2)

#print("PLOT\n")

#plt.show()

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.title(title)

plt.xticks(rotation = 90)

plt.tight\_layout()

if charttype=='plot':

plt.plot(x,y1)

plt.plot(x,y2)

#plt.tight\_layout() #yeeeeeee

elif charttype=='bar':

xaxis = np.arange(x[0],x[len(x)-1]+1)

plt.bar(xaxis+0.00,y1,width=0.25)

plt.bar(xaxis+0.25,y2,width=0.25)

plt.xticks(xaxis)

plt.legend(legend)

#plt.tight\_layout() #yeeeeeee

canvas = FigureCanvasTkAgg(fig, master=plotFRM)

plot\_widget = canvas.get\_tk\_widget()

plot\_widget.grid(row=0, column=0)

#-----------------------------------------------------------------------------

def createBarFRM(frame, df, xcol, ycol):

pass

#-----------------------------------------------------------------------------

def createHistFRM(frame, df, datalist):

pass

#-----------------------------------------------------------------------------

def createStatsFRM(frame, df):

statsFRM = Frame(frame)

statsFRM.grid(row=1, column=1, sticky='NW')

r1=0

c1=0

# sample data - head

def headButtonEvent():

topframe = Toplevel()

d=df.head(2)

pt = Table(topframe, dataframe=d, showstatusbar=True)

pt.show()

Button(statsFRM, text="Head Data", width=20, command=headButtonEvent).grid(row=r1, column=c1, sticky='NW')

# sample data - tail

r1 = r1+1

def tailButtonEvent():

topframe = Toplevel()

d=df.tail(2)

pt = Table(topframe, dataframe=d, showstatusbar=True)

pt.show()

Button(statsFRM, text="Tail Data", width=20, command=tailButtonEvent).grid(row=r1, column=c1, sticky='NW')

# describe

r1 = r1+1

def describeButtonEvent():

topframe = Toplevel()

d=df.describe().round(2)

d.reset\_index(level=0, inplace=True)

#set index as a column in the dataframe so that it gets displayed in the pt table

pt = Table(topframe, dataframe=d, showstatusbar=True)

pt.show()

Button(statsFRM, text="Data Describe", width=20, command=describeButtonEvent).grid(row=r1, column=c1, sticky='NW')

\_transaction.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

submitbtnrow = 0

#-----------------------------------------------------------------------------

def table2df(frame, table, columns='\*', condition='1=1'):

try:

sql = "select "+columns+" from "+table+" where "+condition

#cursor = conn.cursor()

cursor.execute(sql)

data = cursor.fetchall()

df = pd.DataFrame(data)

return df

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#=============================================================================

# child frame

#=============================================================================

def createTransChildFrames(root, param):

w, h = root.winfo\_screenwidth()-50, root.winfo\_screenheight()-50

parentframe = Frame(root)#, height=h-100, width=w/2)

canvas = Canvas(parentframe, height=h-100, width=w/2)

scrollbar = Scrollbar(parentframe, orient="vertical", command=canvas.yview)

frame = Frame(canvas, width=w/2)

frame.bind(

"<Configure>",

lambda e: canvas.configure(scrollregion=canvas.bbox("all")

)

)

canvas.create\_window((0, 0), window=frame, anchor=W)

canvas.configure(yscrollcommand=scrollbar.set)

parentframe.grid(row=0, column=0, padx=15, pady=15)#, sticky=NSEW)

canvas.grid(row=0, column=0, sticky=NS)#EW) #fill all the area available

scrollbar.grid(row=0, column=1, sticky=NS) #EW) #fill top to bottom

#frame = Frame(root) #, xscrollcommand=hsb.set, yscrollcommand=vsb.set)

#frame.grid(row=0, column=0, sticky='W', padx=15, pady=15)

###frame.configure(background='white')

r = 0

Label(frame, text="Transaction Record Entry").grid(row=r, column=0, sticky=W)

# SET TRANSACTION TABLE

table = param['table']

r += 1

# SET DATE USING CALENDAR

Label(frame, text='Date').grid(row=r, column=0, sticky=W)

def getDate():

dt = cal.get\_date()

sqlDate = dt.strftime("%Y%m%d") #20210418(For Database query) #displayDate = dt.strftime("%d-%B-%Y") # date string 18-April-2021

globals()[param['dateColumn']].delete(0,END)

globals()[param['dateColumn']].insert(INSERT,sqlDate)

cal = DateEntry(frame, selectmode='day', width=25)

cal.grid(row=r,column=1,sticky=W) #,padx=15)

cal.set\_date(date.today())

setDateBTN = tk.Button(frame,text='Set Date',command=getDate)

setDateBTN.grid(row=r,column=2,sticky=W,padx=10)

globals()[param['dateColumn']] = Entry(frame)

globals()[param['dateColumn']].grid(row=r,column=4,sticky=W)

r += 1

# SET PRIMARY KEY OF TRANSACTION TABLE e.g. invoice for sale or purchase

for i in param['pk']:

globals()[i+'LBL'] = Label(frame, text=i[i.find('.')+1:])

globals()[i+'LBL'].grid(row=r, column=0, sticky=W)

globals()[i] = Entry(frame, width=30)

globals()[i].grid(row=r, column=1, sticky=W)

r += 1

'''

# see - last most setting

# SET PREFILLED - non-database expression values

for prefill in param['prefilled']:

prefillLeft = prefill[:prefill.find('=')].strip()

prefillRight = prefill[prefill.find('=')+1:]

globals()[prefillLeft+'LBL'] = Label(frame, text=prefillLeft)

globals()[prefillLeft+'LBL'].grid(row=r, column=0, sticky=W)

globals()[prefillLeft] = Entry(frame, width=30)

globals()[prefillLeft].grid(row=r, column=1, sticky=W)

globals()[prefillLeft].delete(0,END)

globals()[prefillLeft].insert(INSERT, eval(prefillRight))

r += 1

'''

# SET PICKUP LISTS - single combo or parent-child combo and other affected GUI controls

#for pickups in param['pickup']:

for pickupcounter in range(len(param['pickup'])):

# SET MASTER PICKUP DROP-DOWNS

#'masterCBO' : ['item.itemcategory>item.itemname'] # masterCBO > childCBO ... select event of masterCBO changes lookup values in childCBO

#for mcbo in param['masterCBO']:

#print("PICKUP ITEM - ",param['pickup'][pickupcounter])

for mcbo in param['pickup'][pickupcounter]['masterCBO']:

#set masterCBO

if len(param['pickup'][pickupcounter]['masterCBO'])>0:

masterCBO = mcbo[:mcbo.find('#')]

childCBO = mcbo[mcbo.find('#')+1:]

masterCBOtbl = masterCBO[:masterCBO.find('.')]

masterCBOcol = masterCBO[masterCBO.find('.')+1:]

childCBOtbl = childCBO[:masterCBO.find('.')]

childCBOcol = childCBO[masterCBO.find('.')+1:]

#conditionlist = param['pickup'][pickupcounter]['masterCBO'] #list

'''

#list to string

condition = ''

for i in conditionlist:

condition += i+','

condition = condition[:-1]

'''

'''

Label(frame, text=masterCBOcol).grid(row=r, column=0, sticky=W)

globals()[masterCBOtbl+'\_'+masterCBOcol+'\_\_'+childCBOtbl+'\_'+childCBOcol+'\_\_\_'+condition] = StringVar()

globals()[masterCBOcol] = Combobox(frame, name=masterCBOtbl+'\_'+masterCBOcol+'\_\_'+childCBOtbl+'\_'+childCBOcol+'\_\_\_'+condition) #use combobox name to get the table name column name and condition specified in \_menu.py

globals()[masterCBOcol].grid(row=r, column=1, sticky=W)

'''

Label(frame, text=masterCBOcol).grid(row=r, column=0, sticky=W)

globals()[masterCBOtbl+'\_'+masterCBOcol+'\_\_'+childCBOtbl+'\_'+childCBOcol+'\_\_\_'+str(pickupcounter)] = StringVar()

#use combobox name to get the table name column name and its index in pickup list for checking condition later on as specified in \_menu.py

globals()[masterCBOcol] = Combobox(frame, name=masterCBOtbl+'\_'+masterCBOcol+'\_\_'+childCBOtbl+'\_'+childCBOcol+'\_\_\_'+str(pickupcounter))

globals()[masterCBOcol].grid(row=r, column=1, sticky=W)

#cols = '\*'

#condition = '1=1'

#globals()[masterCBOtbl+'DF'] = table2df(root, masterCBOtbl, cols, cond)

#print('globals()[masterCBOcol]: ',globals()[masterCBOcol])

'''

# condition to filter child combobox upon selection in master combobox

for conds in param['pickup'][pickupcounter]['condition']:

if masterlookuptbl in conds:

condition = condition[condition.find('.')+1:]

print(condition)

'''

#sql = "select distinct "+masterCBOcol+" from "+masterCBOtbl+" where "+condition

#print(sql)

sql = "select distinct "+masterCBOcol+" from "+masterCBOtbl

cursor.execute(sql)

data = cursor.fetchall()

globals()[masterCBOtbl+'DF'] = pd.DataFrame(data)

#print(globals()[masterCBOtbl+'DF'])

lookupvals = globals()[masterCBOtbl+'DF'][masterCBOcol].tolist()

globals()[masterCBOcol]['values'] = lookupvals

# masterCBO event handling

def funcMasterCBOSelectedEvent(event):

# name of combo widget is in format

# masterCBOtbl+'\_'+masterCBOcol+'\_\_'+childCBOtbl+'\_'+childCBOcol

mpk = event.widget.\_name

mpkmaster = mpk[:mpk.find('\_\_')]

mpkchildcond = mpk[mpk.find('\_\_')+2:]

mpkchild = mpkchildcond[:mpkchildcond.find('\_\_\_')]

mpkconditioncounter = mpkchildcond[mpkchildcond.find('\_\_\_')+3:]

mCBOtbl = mpkmaster[:mpkmaster.find('\_')]

mCBOcol = mpkmaster[mpkmaster.find('\_')+1:]

mCBOval = globals()[mCBOcol].get()

cCBOtbl = mpkchild[:mpkchild.find('\_')]

cCBOcol = mpkchild[mpkchild.find('\_')+1:]

#print('mCBOcol: ',mCBOcol,' mpkmaster: ',mpkmaster)

#print('cCBOcol: ',cCBOcol,' mpkchild: ',mpkchild)

#cCBOval = globals()[mCBOcol].get()

'''

mCBOtbl = mpk[:mpk.find('\_')]

mCBOcol = mpk[mpk.find('\_')+1:]

mCBOval = globals()[mCBOcol].get()

'''

'''

masterCBO = mcbo[:mcbo.find('#')]

mCBOtbl = masterCBO[:masterCBO.find('.')]

mCBOcol = masterCBO[masterCBO.find('.')+1:]

mCBOval = globals()[mCBOcol].get()

'''

'''

childCBO = mcbo[mcbo.find('#')+1:]

childCBOcol = childCBO[childCBO.find('.')+1:]

'''

#childCBOcol = globals()[childCBO][globals()[childCBO].find('.')+1:]

#print('mpk: ',mpk,' mCBOtbl: ',mCBOtbl,' mCBOcol: ',mCBOcol,' mCBOval: ',mCBOval)

#print("invoice.get(): ",invoice.get())

#condition: e.g. 'item.itemcode < + int(price.get())' where '+' separates colname and value of a form control

'''

condition = '1=1'

for conds in param['pickup'][int(mpkconditioncounter)]['condition']:

print("conds: ",conds)

if mCBOtbl in conds:

#condition = conds[conds.find('.')+1:]

conditionleft = conds[:conds.find('+')]

conditionright = conds[conds.find('+')+1:]

#globals()[mCBOtbl+'DF'] = table2df(root, mCBOtbl, cols, cond)

'''

'''

condition = '1=1'

for conds in param['pickup'][int(mpkconditioncounter)]['condition']:

condition = conds

#if (condition != '1=1') or (condition is not None):

'''

if len(param['pickup'][int(mpkconditioncounter)]['condition'])>0:

condition = param['pickup'][int(mpkconditioncounter)]['condition'][0]

#sql = "select \* from "+mCBOtbl+" where "+mCBOcol+"='"+mCBOval+"' and "+conditionleft+str(eval(conditionright))

sql = "select \* from "+mCBOtbl+" where "+mCBOcol+"='"+mCBOval+"' and "+str(eval(condition))

else:

sql = "select \* from "+mCBOtbl+" where "+mCBOcol+"='"+mCBOval+"'"

print(sql)

#print(sql)

#print('masterCBO: ',masterCBO, ' childCBO: ',childCBO, ' ',mCBOtbl, ' ',mCBOcol, ' ',mCBOval)

cursor.execute(sql)

data = cursor.fetchall()

globals()[mCBOtbl+'DF'] = pd.DataFrame(data)

#print(globals()[mCBOtbl+'DF'])

#lookupvals = globals()[mCBOtbl+'DF'][childCBOcol].tolist()

#print(globals()[mCBOtbl+'DF'][cCBOcol].tolist())

if globals()[mCBOtbl+'DF'].empty:

lookupvals = ''

else:

lookupvals = globals()[mCBOtbl+'DF'][cCBOcol].tolist()

print("lookupvals: ",lookupvals)

print("globals()[mCBOtbl+'DF']: \n",globals()[mCBOtbl+'DF'])

print("globals()[cCBOcol]: ",globals()[cCBOcol])

#childCBO = mcbo[mcbo.find('>')+1:]

#childCBOtbl = childCBO[:childCBO.find('.')]

#globals()[childCBOcol]['values'] = lookupvals

globals()[cCBOcol]['values'] = lookupvals

globals()[masterCBOcol].bind("<<ComboboxSelected>>", funcMasterCBOSelectedEvent)

#print("Combobox #",str(pickupcounter+1),globals()[masterCBOcol])

#print("Combobox #",str(pickupcounter+1),globals()[masterCBOcol])

#print("FIRST ROUND")

r += 1

#for i in param['masterLookupItems']: #DICT ITEM: 'masterLookupItems':['table1.pk1','table2.pk2',...]

for i in param['pickup'][pickupcounter]['masterLookupItems']:

# Pick one master table along with its primary key, lookup and autofills at a time

masterlookuptbl = i[:i.find('.')]

masterlookupcol = i[i.find('.')+1:]

#'condition' : ['item.itemstock > 0']

#default condition='1=1'

cols = '\*'

'''

#need not check condition upon selection in lookup table - already checked upon selection in master CBO

cond = '1=1'

for conds in param['pickup'][pickupcounter]['condition']:

if masterlookuptbl in conds:

cond = conds[conds.find('.')+1:]

'''

#print(cond)

#print(masterlookuptbl, " ", masterlookupcol, " ",cond)

############ PERFECTELY WORKING STATEMENT

############ globals()[masterlookuptbl+'DF'] = table2df(root, masterlookuptbl, cols, cond)

#use sql query rather than table name to use condition

#sql = "select \* from "+masterlookuptbl+" where "+cond

sql = "select \* from "+masterlookuptbl

#print(sql)

cursor.execute(sql)

data = cursor.fetchall()

globals()[masterlookuptbl+'DF'] = pd.DataFrame(data)

#print(masterlookuptbl, " ", masterlookupcol, " ",cond)

#print(globals()[masterlookuptbl+'DF'])

#print(globals()[masterlookuptbl+'DF'])

lookupvals = globals()[masterlookuptbl+'DF'][masterlookupcol].tolist()

#print(lookupvals)

# set combobox or drop-down with master table lookup values

Label(frame, text=masterlookupcol).grid(row=r, column=0, sticky=W)

globals()[masterlookuptbl+'\_'+masterlookupcol] = StringVar()

globals()[masterlookupcol] = Combobox(frame, name=masterlookuptbl+'\_'+masterlookupcol) #use combobox name to get the table name

globals()[masterlookupcol].grid(row=r, column=1, sticky=W)

globals()[masterlookupcol]['values'] = lookupvals

# master lookup table PKs

for mpk in param['pickup'][pickupcounter]['masterPrimaryKeys']:

if masterlookuptbl in mpk:

masterlookuppkcol = mpk[mpk.find('.')+1:]

globals()[masterlookuptbl+'PK']=masterlookuppkcol

Label(frame, text=masterlookuppkcol).grid(row=r, column=2, sticky=W)

globals()[masterlookuppkcol] = Entry(frame, width=30)

globals()[masterlookuppkcol].grid(row=r, column=3, sticky=W)

#print('globals()[masterlookuppkcol]: ',masterlookuppkcol)

r += 1

# master lookup table autofills

globals()[masterlookuptbl+'AF'] = []

for afv in param['pickup'][pickupcounter]['masterAutofillValues']:

if masterlookuptbl in afv:

masterautofillcol = afv[afv.find('.')+1:]

Label(frame, text=masterautofillcol).grid(row=r, column=0, sticky=W)

globals()[masterautofillcol] = Entry(frame, width=30)

globals()[masterautofillcol].grid(row=r, column=1, sticky=W)

r += 1

globals()[masterlookuptbl+'AF'].append(masterautofillcol)

#print(masterautofillcol)

# master - 'getvalfromothertableoncboevent' handling upon masterlookuptbl cbo selection

# master lookup event handling

def funcSelectedEvent(event):

#combobox name => mastertable+'\_'+masterlookupcol

#Get label text

#print("label text:", event.widget.cget("text"))

mpk = event.widget.\_name

masterlookuptbl = mpk[:mpk.find('\_')]

masterlookupcol = mpk[mpk.find('\_')+1:]

masterlookupval = globals()[masterlookupcol].get()

mastertbleDF = globals()[masterlookuptbl+'DF']

#fill PK and autofill values from master pickup table

#print('mpk: ',mpk)

#print(param['pickup'][pickupcounter]['masterPrimaryKeys'])

#print(masterlookuptbl)

#print(mastertbleDF.columns)

for c in mastertbleDF.columns:

#print(masterlookuptbl+'.'+c,' ',globals()[masterlookuptbl+'PK'])

#if masterlookuptbl+'.'+c in param['pickup'][pickupcounter]['masterPrimaryKeys']:

if c == globals()[masterlookuptbl+'PK']:

#print("GOT PK: ",masterlookuptbl+'.'+c)

#masterlookuppkval = mastertbleDF[mastertbleDF[masterlookupcol]==masterlookupval][c].tolist()[0]

'''

print("mastertbleDF: ",mastertbleDF)

print("masterlookuptbl: ",masterlookuptbl)

print("masterlookupcol: ",masterlookupcol)

print("masterlookupval: ",masterlookupval)

print("c: ",c)

print("mastertbleDF[masterlookupcol]: \n",mastertbleDF[masterlookupcol])

#print("mastertbleDF[masterlookupcol]==masterlookupval: \n",mastertbleDF[masterlookupcol]==int(masterlookupval)) #use int(masterlookupval) if column type is int

print("mastertbleDF[masterlookupcol]==masterlookupval: \n",mastertbleDF[masterlookupcol]==masterlookupval) #use int(masterlookupval) if column type is int

print("mastertbleDF[mastertbleDF[masterlookupcol]==masterlookupval]: \n",mastertbleDF[mastertbleDF[masterlookupcol]==masterlookupval])

print("mastertbleDF[mastertbleDF['membercode']=='G2']: \n",mastertbleDF[mastertbleDF['membercode']=='G2'])

print("mastertbleDF[mastertbleDF['membercode']=='G2'][c]: \n",mastertbleDF[mastertbleDF['membercode']=='G2'][c])

print("mastertbleDF[mastertbleDF['membercode']=='G2'][c].tolist()[0]: \n",mastertbleDF[mastertbleDF['membercode']=='G2'][c].tolist()[0])

'''

masterlookuppkval = mastertbleDF[mastertbleDF[masterlookupcol]==masterlookupval][c].tolist()[0]

# use '==int(masterlookupval)' if lookup column type is int

#print("masterlookuppkval: ",masterlookuppkval)

globals()[c].delete(0,END)

globals()[c].insert(INSERT,masterlookuppkval)

#if masterlookuptbl+'.'+c in param['pickup'][pickupcounter]['masterAutofillValues']:

if c in globals()[masterlookuptbl+'AF']:

autofillVal = mastertbleDF[mastertbleDF[masterlookupcol].astype(str)==str(masterlookupval)][c].tolist()[0]

if isinstance(autofillVal, int)==True:

autofillVal = round(autofillVal,2)

globals()[c].delete(0,END)

#globals()[c].insert(INSERT,round(autofillVal,2))

if autofillVal is not None:

globals()[c].insert(INSERT,autofillVal)

else:

globals()[c].insert(INSERT,"")

#print(masterlookuptbl+'.'+c, ' ' ,autofillVal )

for gvfot in param['pickup'][pickupcounter]['getvalfromothertableonlookupcboevent']:

'''

gvfotcol = gvfot[:gvfot.find('=')]

gvfotquery = gvfot[gvfot.find('=')+1:]

print("gvfotcol: ",gvfotcol)

print("gvfotquery: ",gvfotquery)

print("studentclass.get(): ",globals()['studentclass'].get()+"xxxx")

#print("studentclass.get(): ",globals()[masterlookupcol].get())

cursor.execute(gvfotquery)

gvfotval = cursor.fetchall()

print("gvfotcol: ",gvfotval)

'''

if c in gvfot['triggercol']:

global submitbtnrow

Label(frame, text=gvfot['targetcontrol']).grid(row=submitbtnrow, column=0, sticky=W)

globals()[gvfot['targetcontrol']] = Entry(frame, width=30)

globals()[gvfot['targetcontrol']].grid(row=submitbtnrow, column=1, sticky=W)

print("gvfot['triggercol']: ",gvfot['triggercol'])

print("gvfot['targetcontrol']: ",gvfot['targetcontrol'])

print("gvfot['othertablecol']: ",gvfot['othertablecol'])

print("gvfot['othertable']: ",gvfot['othertable'])

print("gvfot['where']: ",gvfot['where'])

sql = ' select '+gvfot['othertablecol']

sql += ' from '+gvfot['othertable']

sql += ' where '

for gvfoti in gvfot['where']:

sql += gvfoti[:gvfoti.find('=')]+'="'

sql += globals()[gvfoti[gvfoti.find('=')+1:]].get()+'"'

sql += ' and '

sql = sql[:-5]

print(sql)

cursor.execute(sql)

gvfotval = cursor.fetchone()

print("gvfotval: ",gvfotval)

print("gvfotval[gvfot['targetcontrol']]: ",gvfotval[gvfot['targetcontrol']])

globals()[gvfot['targetcontrol']].delete(0,'end')

#globals()[gvfot['targetcontrol']].insert(INSERT,gvfotval[gvfot['targetcontrol']])

#globals()[gvfot['targetcontrol']].insert(0,gvfotval[gvfot['targetcontrol']])

#if globals()[gvfot['targetcontrol']] is not None:

globals()[gvfot['targetcontrol']].insert(INSERT,gvfotval[gvfot['targetcontrol']])

#entry.insert(tk.END, " Hello World")

globals()[masterlookupcol].bind("<<ComboboxSelected>>", funcSelectedEvent)

r += 1

for i in param['transItems']:

globals()[i+'LBL'] = Label(frame, text=i[i.find('.')+1:])

globals()[i+'LBL'].grid(row=r, column=0, sticky=W)

globals()[i] = Entry(frame, width=30)

globals()[i].grid(row=r, column=1, sticky=W)

r += 1

if len(param['expressions'])<=0:

pass

else:

def expressionBTNClickEvent():

for exp in param['expressions']:

expLeft = exp[:exp.find('=')].strip()

expRight = exp[exp.find('=')+1:]

#print("expLeft: ",expLeft)

#print("expRight: ",expRight)

#print("expRight: ",eval(expRight))

#globals()[expLeft].insert(INSERT, round(eval(expRight),2))

globals()[expLeft].delete(0,END)

globals()[expLeft].insert(INSERT, eval(expRight))

expressionBTN = Button(frame, text="Calculate", command=expressionBTNClickEvent)

expressionBTN.grid(row=r, column=1)

r += 1

for i in param['expressions']:

i = i[:i.find('=')].strip()

globals()[i+'LBL'] = Label(frame, text=i)

globals()[i+'LBL'].grid(row=r, column=0, sticky=W)

globals()[i] = Entry(frame, width=30)

globals()[i].grid(row=r, column=1, sticky=W)

r += 1

# SET PREFILLED - non-database expression values

# prefill textfields with default initial values

for prefill in param['prefilled']:

prefillLeft = prefill[:prefill.find('=')].strip()

prefillRight = prefill[prefill.find('=')+1:]

#globals()[prefillLeft+'LBL'] = Label(frame, text=prefillLeft)

#globals()[prefillLeft+'LBL'].grid(row=r, column=0, sticky=W)

#globals()[prefillLeft] = Entry(frame, width=30)

#globals()[prefillLeft].grid(row=r, column=1, sticky=W)

globals()[prefillLeft].delete(0,END)

globals()[prefillLeft].insert(INSERT, eval(prefillRight))

#r += 1

#for inv in param['invisible']:

# globals()[inv].grid\_remove()

def submitButtonClickEvent():

# columns to be sent to trans 'table' - dateColumn, pk, transItems, expressions

# master values not to be sent to trans table unless assigned to the columnnames in expression

# masterUpdates to be set in master tables only

#!IMPORTANT: date column should be named something as 'transdate' not as 'date'

#Do not change 'keys' of the following dictionaries

##use column names only on right hand side of an expression

#'invisible' - is the simple assignment which is to be saved in trans table

# BUT NOT to be displayed on screen e.g. comboboxes without textfields inside PICKUP items

cols = []

vals = []

table = param['table']

cols.append(param['dateColumn'])

for p in param['pk']:

cols.append(p[p.find('.')+1:])

'''

for mcb in param['masterCBO']:

cols.append(mcb[mcb.find('.')+1:mcb.find('#')])

for mpi in param['masterLookupItems']:

cols.append(mpi[mpi.find('.')+1:])

for mpk in param['masterPrimaryKeys']:

#cols.append(mpk[mpk.find('.')+1:])

#cols.append(globals()[mpk[:mpk.find('.')]+'PK']) #equivalent of globals()[table+'PK']

#t = globals()[mpk[mpk.find('.')+1:]].get()

cols.append(mpk[mpk.find('.')+1:])

'''

'''

for pickupcounter in range(len(param['pickup'])):

for mcb in param['pickup'][pickupcounter]['masterCBO']:

cols.append(mcb[mcb.find('.')+1:mcb.find('#')])

for mpi in param['pickup'][pickupcounter]['masterLookupItems']:

cols.append(mpi[mpi.find('.')+1:])

for mpk in param['pickup'][pickupcounter]['masterPrimaryKeys']:

#cols.append(mpk[mpk.find('.')+1:])

#cols.append(globals()[mpk[:mpk.find('.')]+'PK']) #equivalent of globals()[table+'PK']

#t = globals()[mpk[mpk.find('.')+1:]].get()

cols.append(mpk[mpk.find('.')+1:])

'''

for ti in param['transItems']:

cols.append(ti[ti.find('.')+1:])

for ex in param['expressions']:

#cols.append(ex[ex.find('.')+1:])

cols.append(ex[:ex.find('=')].strip())

'''

if 'id' in cols:

cols.remove('id') #do not insert any value for id column of type int auto\_increment

'''

# primary keys in a table to be named as tblname+'id' of type int and auto\_increment

#print(cols)

for c in cols:

if c[-2:]=='id':

cols.remove(c) #remove auto\_increment primary key columns from insert sql query

if c in param['displayonlynoinsert']:

cols.remove(c) #remove display only columns from insert sql query

for c in cols:

#print(cols," ",c)

vals.append(globals()[c].get())

print("cols -1 : ",cols)

print("vals -1 : ",vals)

for inv in param['invisible']: #simple assignments, same as expressions

#print("INV: ",inv)

cols.append(inv[:inv.find('=')].strip())

vals.append(eval(inv[inv.find('=')+1:].strip()))

print("cols -2 : ",cols)

print("vals -2 : ",vals)

if (len(param['transItems'])<=0) and (len(param['expressions'])<=0):

pass

else:

sql = "INSERT INTO "+ table[0] +"("

for c in cols:

sql += c+","

sql = sql[:-1]

sql += ") VALUES("

for v in vals:

sql += "'"+v+"',"

sql = sql[:-1]

sql += ")"

print("INSERT SQL QUERY: ",sql)

try:

cursor.execute(sql)

#cursor.execute(sql1)

conn.commit()

msg = "SUCCESS: TRANSACTION (INSERT SQL query) executed successfully."

#tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

#'masterUpdates' : ['item.stock=float(item.stock)+float(quantity)']

for t in param['masterUpdates']:

left = t[:t.find('=')]

table = left[:left.find('.')]

col = left[left.find('.')+1:]

exp = t[t.find('=')+1:] #right

pkCol=''

print('before pkc loop - table: ',table,' col: ',col,' exp: ',exp)

for i in range(len(param['pickup'])):

for pkc in param['pickup'][i]['masterPrimaryKeys']:

tbl = pkc[:pkc.find('.')]

if tbl==table:

pkCol = pkc[pkc.find('.')+1:]

print('inside pkc loop - tbl: ',tbl,' pkCol: ',pkCol)

#exp = str(round(eval(exp),2))

exp = str(eval(exp))

#print(exp)

sql1 = "UPDATE "+ table +" SET "+ col +"='"+ exp + \

"' WHERE "+ pkCol +"='"+ globals()[pkCol].get() +"'"

print(sql1)

try:

#####cursor.execute(sql)

cursor.execute(sql1)

conn.commit()

msg = "SUCCESS: Master table updated successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

except conn.Error as e:

msg = "ERROR: Master table could not be updated.\n"+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

try:

#cursor.execute(sql)

#cursor.execute(sql1)

#conn.commit() #commit both INSERT and UPDATE queries together

msg = "SUCCESS: Transaction (insert) and Master update (update) executed successfully."

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

except conn.Error as e:

msg = "ERROR: "+str(e.args[0])+e.args[1]

tk.messagebox.showinfo("MESSAGE", msg, parent=frame)

submitButton = Button(frame, text="Submit", command=submitButtonClickEvent)

submitButton.grid(row=r+2, column=1) #leave some rows for late generated controls

global submitbtnrow

submitbtnrow = r

r += 1

Label(frame, text='').grid(row=r, column=0, sticky=W) #for padding

#-----------------------------------------

for widget in frame.winfo\_children():

widget.grid(padx=0, pady=5)

#=============================================================================

# root frame

#=============================================================================

def createTransRootFrame(rootframe, p):

#print(p)

w, h = rootframe.winfo\_screenwidth()-50, rootframe.winfo\_screenheight()-150

transrootframe = Toplevel(rootframe)

transrootframe.geometry("%dx%d+15+60" % (w, h))

transrootframe.title("TRANSACTION ENTRY")

transrootframe.config(bg="black")

table = p['table'][0]

df = table2df(transrootframe,table)

createTransChildFrames(transrootframe, p)

#=============================================================================

\_index.py

from \_libraryAndDBConnection import \* #includes database connection and cursor setting strings

import \_menu

#----------------------------

'''

Instructions:

-------------

1. Set database name in \_libraryAndDBConnection.py file

2. Customize 'DROP TABLE', 'CREATE TABLE' and 'INSERT INTO' statements

in \_createMySQLTablesWithTestData.py file

3. Set menu options in \_menu.py file with

(1) tablename

(2) primary key name

(3) combobox/drop-down source table & column names

(4) date column name [for calendar]

(5) image column name [...to be implemented in future versions]

'''

#----------------------------

#====================================================================================

# rootframe (parent window) to be Tk ()

# rest others (child windows) to be Toplevel() or Frame()

#====================================================================================

#ROOT OR MAIN OR PARENT FRAME

rootframe = Tk()

rootframe.title('Chitransh & Siddhant Software Management System')

welcometext = "Hello There!\n\nWelcome to Chitransh & Siddhant Pharmacy Soft Inc."

#maximize root window with title

w, h = rootframe.winfo\_screenwidth(), rootframe.winfo\_screenheight()-100

rootframe.geometry("%dx%d+0+0" % (w, h)) #root window size 'wxh' at left top coordinates 0,0

rootframe.configure(background="light grey",bd=-10)

#add background image to main root window using label after resizing the image to fit well within it

mywidth = rootframe.winfo\_screenwidth() #-20

myheight = rootframe.winfo\_screenheight()-80 #-100

#BACKGROUND IMAGE

img = Image.open("hospitalbg.jpg") #read the image

resizeimg = img.resize((mywidth, myheight)) #resize the image

bgimg = ImageTk.PhotoImage(resizeimg) #set resized images as bg

#bgimglbl = Label(rootframe, image=bgimg) #place bg image in the label instead of text as in Label(frame, text="text")

bgimglbl = Label(rootframe, compound=tk.CENTER, text=welcometext, bd=-10, font='Helvetica 30 bold', fg='#af1f1f', image=bgimg)

bgimglbl.grid(row=0, column=0, sticky=tk.E+tk.W) #place the label in the first row, first column of the main root window; centered by default

#place text field over the bg image at the top centered horizontally

#logo

'''

#logoimg = Image.open("bvmlogo.png") #logo image

#resizeimg = logoimg.resize((100, 100)) #resize the image

logoimg=PhotoImage(file="bvmlogo.png") #logo image

logoimglbl=Label(rootframe, height=2, image=logoimg, bg= "grey")

logoimglbl.grid(row=0, column=0, sticky='NEW')

'''

'''

topframe = Frame(rootframe, background='')

topframe.grid(row=0, column=0, sticky='NEW')

'''

welcometext1 = "...A Program That Allows the Unimaginable..."

welcomelbl1 = Label(rootframe, height=2, text=welcometext1, font='Helvetica 18 bold', bg= "#af1f1f")

welcomelbl1.grid(row=0, column=0, sticky='NEW') #place it in first row, first column of the main root window; centered by default

'''

from tkinter import Canvas

#import ImageTk

t = Tk()

t.title("Transparency")

#frame = Frame(t)

frame = Frame(rootframe)

#frame.pack()

frame.grid(row=0, column=0, padx=(20,20), pady=(20,20), sticky='NE')

#canvas = Canvas(frame, bg="black", width=500, height=500)

#canvas.pack()

canvas = Canvas(rootframe, bg="black", width=500, height=500)

canvas.grid(row=0, column=0, padx=(20,20), pady=(20,20), sticky='NE')

photoimage = ImageTk.PhotoImage(file="bvmlogo.png")

canvas.create\_image(150, 150, image=photoimage)

'''

#lbl.grid( padx=(padleft, padright), pady=(padtop, padbottom))

logoimg=Image.open('opulent\_logo.jpg')# Load the image

logoresizedimg=logoimg.resize((75, 100))# Resize the image in the given (width, height)

logofinalimg=ImageTk.PhotoImage(logoresizedimg)# Conver the image in TkImage

logoimglbl=Label(rootframe, image=logofinalimg)# Display the image with label

logoimglbl.grid(row=0, column=0, padx=(20,20), pady=(20,20), sticky='NE') #set the label

'''

canvas = Canvas(rootframe, bg="black", width=75, height=100)

canvas.grid(row=0, column=0, padx=(20,20), pady=(20,20), sticky='NW')

canvas.create\_image(75, 100, image=logofinalimg)

'''

#place text field over the bg image at the bottom centered horizontally

welcometext2 = "Contact: +91-9564733916\nBuy more of Chitransh's courses on abc.com\n"

welcomelbl2 = Label(rootframe, height=3

, text=welcometext2, justify="left", font='"Bodoni MT" 15', bg= "black")

welcomelbl2.config(fg="white", justify="left")

welcomelbl2.grid(row=0, column=0, sticky='SEW') #place it in first row, first column of the main root window; centered by default

#---------------------------------------------

#GUI - App Menu

menubar = \_menu.menu(rootframe)

#---------------------------------------------

#GUI - infinite main root loop

rootframe.config(menu=menubar)

rootframe.mainloop()

**===x=====x=====x=====x=====x=====x=====x=====x=====x===**

***THIS IS THE END OF SOURCE CODE!***

**By:**  Chitransh Bhatnagar

& Siddhant Pathak