



NEW HORIZON
COLLEGE OF ENGINEERING



A MINI PROJECT

REPORT

For

MINI PROJECT USING PYTHON (20CSE59)

EXPENSE TRACKER

submitted by

J S JAYASREE

1NH18CS077

05/A

In partial fulfillment for the award of

the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



NEW HORIZON
COLLEGE OF ENGINEERING



Certificate

This is to certify that the mini project work titled

EXPENSE TRACKER

*Submitted in partial fulfillment of the degree of Bachelor of Engineering
in Computer Science and Engineering*

Submitted by

J S JAYASREE

1NH18CS077

DURING

ODD SEMESTER 2020-2021

For

20CSE59

Signature of Reviewer

Signature of HOD

SEMESTER END EXAMINATION

Name of the Examiner

Signature with date

1. _____

2. _____

ABSTRACT

“EXPENSE TRACKER” project illustrates the designing and implementation of an Expense Monitoring System. My project portrays itself as a record managing application which provides an explicit support for user in querying the expense incurred on various categories.

The primary aim is to improve accuracy and enhance safety and efficiency in the organization of voluminous data. Today management is one of the most essential features of all form. Management provides sophistication to perform any kind of task in a particular form. My project is used to manage expense related activities.

The exclusive use of the database allows the administrator to may even include any new expense category in the record; and enabling for visualizing data by means of graphical analysis.

This report depicts an inspiring voyage down the road of expense tracking application. From requirements to use cases, to database design, to component frameworks, to user interfaces, we will cover each and every aspect of the project design required to build an application.

The reason why I selected this project is the current expense managing system is mostly based on manual work; however, there may be some exceptions to this as some industries, corporate do not rely on fully furnished computerized system but generate reports for the cost incurred. There is a high need of revolution in the financial managing system in India. Also, raising of awareness is of utmost importance as it plays a vital role in making financial decisions. So, I believe in pragmatic approach and that encouraged me to promote raising awareness about the expenses incurred by an individual in my project.

The use of database improves the efficiency of the financial agency by reducing the tedious job of the administrator and thus making it more responsible for the well-being of the citizen of a nation.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman of New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha**, Principal NHCE, for his constant support and encouragement.

I am grateful to **Dr. Prashanth C.S.R**, Dean Academics, for his unfailing encouragement and suggestions, given to me in the course of my project work.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and Head, Department of Computer Science and Engineering, for her constant support.

I express my gratitude to **Ms. Rajitha Nair**, Senior Assistant Professor, my project guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

J S JAYASREE

1NH18CS077

TABLE OF CONTENTS

<u>Chapter. No</u>	<u>Chapter</u>	<u>Page No</u>
1.	INTRODUCTION	1-18
	1.1 PROBLEM DEFINITION	1
	1.2 COURSE OBJECTIVES	2-16
	1.3 METHODOLOGY TO BE FOLLOWED	17
	1.4 EXPECTED OUTCOMES	18
2.	REQUIREMENT SPECIFICATION	19
	2.1 HARDWARE REQUIREMENT	19
	2.2 SOFTWARE REQUIREMENT	19
3.	ER MODEL	20-26
	3.1 ENTITIES AND ATTRIBUTES	20-22
	3.2 KEYS, RELATIONSHIP AND PARTICIPATION	23-26
4.	IMPLEMENTATION	27-29
	5.1 GUI PROGRAMMING	27
	5.2 SQL	28
	5.3 DATA VISUALISATION USING PYTHON	29
5.	RESULTS	30-33
6.	CONCLUSION	34

APPENDIX A:	PROJECT CODE	35-45
-------------	--------------	-------

	REFERENCES	46
--	------------	----

LIST OF FIGURES

<u>Figure no.</u>	<u>Figure description</u>	<u>Page no.</u>
1.2a	Python Data Structures	6
1.2b	Depicts an example of Class “Dog”	9
1.2c	Depicts a class named “StrangerThings”	10
1.2d	Single Inheritance	11
1.2e	Multiple Inheritance	11
1.2f	Multilevel Inheritance	12
1.2g	Hierarchical Inheritance	12
1.2h	Hybrid Inheritance	13
1.2i	Three tier Architecture of DBMS	16
3.1a	ER - Model	20
3.1b	ER Diagram of “Expense Tracker”	22
3.2a	Participation Constraint	24
3.2b	Mapping of ER Diagram to Relational Schema	26
4.1a	Workflow of Tkinter	27
5.1	GUI asking for new User and Existing User	30
5.2	Registration of New User	31

5.3	Validating Credentials of Existing User	31
5.4	Asking the expenses for the user	32
6.5	Grouped Bar Plot and Scatter Plot for the User	33

LIST OF ABBREVIATIONS

OOP	OBJECTED ORIENTED PROGRAMMING
GUI	GRAPHICAL USER INTERFACE
SQL	STRUCTURED QUERY LANGUAGE
ER	ENTITY RELATIONSHIP
CR	CARDINALITY RATIO
DBMS	DATA BASE MANAGEMENT SYSTEM
ANSI	AMERICAN NATIONAL STANDARD INSTITUTE
DDL	DATA DEFINITION LANGUAGE
DML	DATA MANIPULATION LANGUAGE
DCL	DATA CONTROL LANGUAGE
DQL	DATA QUERY LANGUAGE

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

The main purpose is to build an application program as manual copies of managing expenses tend to cause human error thus making the working of an organisation inefficient. Usually the shelf management system is a tiresome job for an individual in maintaining all the details of amount spent on various categories for a period of month, his credentials and many more wherein he or she searches for the records preserved physically. In the process he probably might get disappointed if there is a lack of the neat records.

Also, Expense management has kept paper record in filing cabinets. If it proves that monitoring individual expense on a monthly basis is difficult then managing a very large corporate, business agency with records on papers will be tedious and difficult to keep track of inventories with regards to the cost incurred and their details. The individual has to look for manual records for every operation or action to be performed.

Significant amount of time is allocated for writing the expense and category details as he/she needs to go through the records available and make rough estimate of the amount spent for current and previous month. Also, the details of already availed user have to be maintained to raise awareness about the unnecessary amount spent.

Thus, Expense Tracker allows for database server application to efficiently manage the data to ensure an elevated performance of the organisation.

Also, data security cannot be fortified by the manual copies.

Thus, the objective of project is to generate a clear database and to present a comparative analysis by visualizing data with the help of database. It proves to be a great help for an individual in making the most effective financial decisions.

1.2 COURSE OBJECTIVES

At the end of the Course, the Student will be able to:

- CO1 Understand the technological needs and/ or societal needs.
- CO2 Design and develop an algorithm by applying PYTHON-programming features.
- CO3 Analyze and evaluate the algorithm performance metrics or database applications.
- CO4 Test, validate and communicate the identified solutions in a structured way.

The aim of the Project could be summarized as:

- Provide for mass storage of relevant data.
- Make access to the data easy for the user.
- Provide prompt response to user requests for data.
- Making modifications to the database available immediately.
- Improves Accuracy
- Enhances Security
- Time Saving
- Perform repetitive Task very well
- Low cost
- Easy Maintenance
- Customer friendly

PYTHON AND DBMS FUNDAMENTALS

INTRODUCTION TO PYTHON:

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently. There are two major Python versions: Python 2 and Python 3. Both are quite different. Python is named after the comedy television show Monty Python's Flying Circus. It is not named after the Python snake.

Features of Python Programming Language:

- 1.Readable:** Python is a high-level programming language and its degree of readability is certainly highly compared to other programming languages.
- 2.Easy to Learn:** Learning Python is interesting and easy since the language is quite expressive and its syntax is not too complicated.
- 3.Cross Platform:** Python is a platform independent and portable Language.
- 4.Open Source:** Python is an open source programming language.
- 5.Exception Handling:** An exception is a Programming error which occurs at runtime and disrupt the normal flow of execution. Python allows us to test various scenarios and the exceptions can be effectively handled without affecting the normal execution of the program.
- 6Automatic Memory Management:** Python supports Dynamic memory management and frees up the unused space.

APPLICATIONS OF PYTHON

Python supports cross-platform operating systems which makes building applications with it all the more convenient. Some of the globally known applications such as YouTube, BitTorrent, Dropbox etc., use Python to achieve their functionality.

1. Web Development

Python can be used to make web-applications at a rapid rate. It is because of the frameworks Python uses to create these applications. There is common-backend logic that goes into making these frameworks and a number of libraries that can help integrate protocols such as HTTPS, FTP, SSL etc. and even help in the processing of JSON, XML, E-Mail and so much more. Some of the most well-known frameworks are Django, Flask, Pyramid. The security, scalability, convenience that they provide is commendable if we compare it to starting the development of a website from scratch.

2. Game Development

Python is also used in the development of interactive games. There are libraries such as PySoy which is a 3D game engine supporting Python 3, PyGame which provides functionality and a library for game development. Games such as Civilization-IV, Disney's Toon town Online, Vega Strike etc. have been built using Python.

3. Machine Learning and Artificial Intelligence

Machine Learning and Artificial Intelligence are the talks of the town as they yield the most promising careers for the future. We make the computer learn based on past experiences through the data stored or better yet, create algorithms which makes the computer learn by itself. The programming language that mostly everyone chooses is Python. Support for these domains with the libraries that exist already such as Pandas, Scikit-Learn, NumPy and so many more.

4. Data Science and Data Visualization

Data is cost if you know how to extract relevant information which can help you take calculated risks and increase profits.

PYTHON BASICS

Python Variable: The concept of naming a variable follows the same idea as other Programming Languages. But in Python we do not pre-define a variable with its datatype. The data type is decided dynamically when the value is initialized to a variable.

Python Datatypes: Python has built-in data types which are of seven categories. They are text type (String), Numeric (float, complex), Sequence Types (list, tuple, range), Mapping Type (Dictionary), Set Type (set, frozen set), Boolean Type(bool), Binary byte array, memory view). Ex: x=5 #when you initialize x with 5 then dynamically it is known x is of Integer Type

Python Operators: Operators are used to perform operations on variables and values. Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Identity and Membership operators are quite interesting with respect to Python. The operators follow the same strategy as with respect to other programming languages Identity operators (is, is not) are used to compare the objects, not checking the value of objects instead it checks if it points to same memory location. If so, it returns True if not it returns False. Membership operators (in, not in) are used to test if a sequence is presented in an object. Ex: Let's take a sequence of integers [1,2,3] Membership Operator checks if 2 occurs in the sequence and returns true if not it returns false.

PYTHON DATA STRUCTURES

What is a Data Structure?

Organizing, managing and storing data is important as it enables easier access and efficient modifications. Data Structures allows you to organize your data in such a way that enables you to store collections of data, relate them and perform operations on them accordingly.

Types of Data Structures in Python

Python has implicit support for Data Structures which enable you to store and access data. These structures are called List, Dictionary, Tuple and Set. Python allows its users to create their own Data Structures enabling them to have full control over their functionality. The most prominent Data Structures are Stack, Queue, Tree, Linked List and so on which are also available to you in other programming languages.

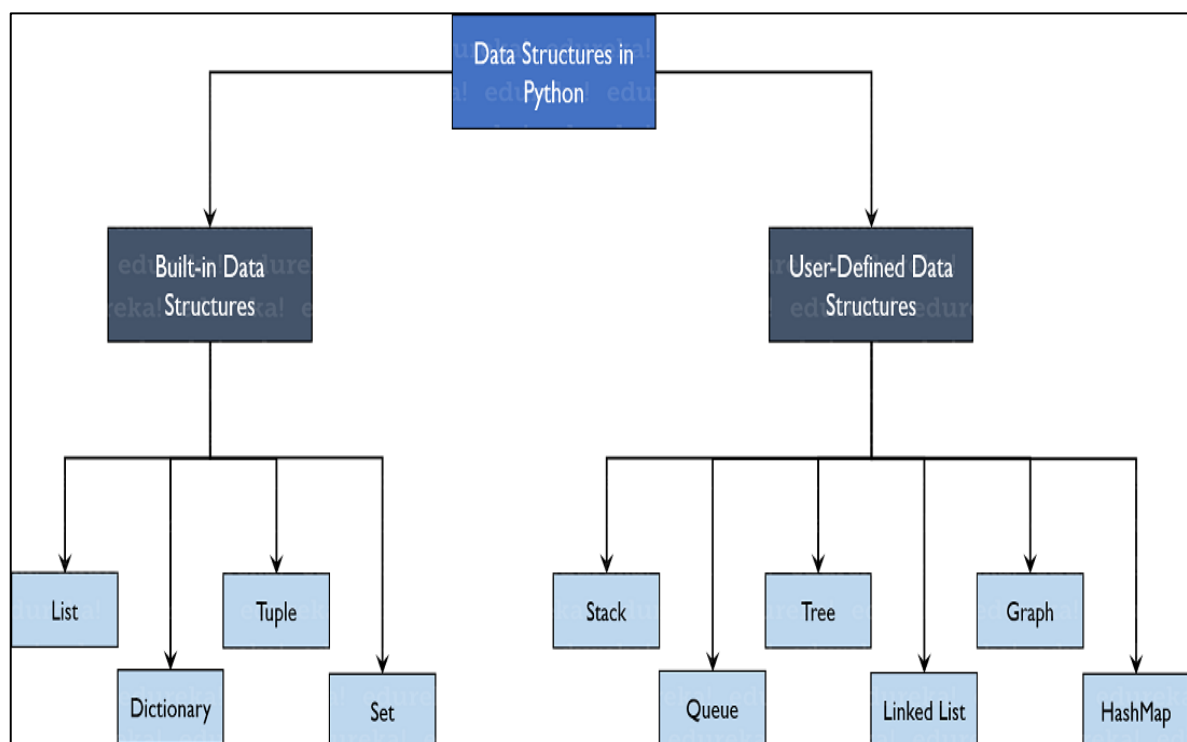


Fig 1.2a Python Data Structures

Built-in Data Structures

Data Structures are built-in with Python which makes programming easier and helps programmers use them to obtain solutions faster.

Lists

Lists are used to store data of different data types in a sequential manner. There are addresses assigned to every element of the list, which is called as Index. The index value starts from 0 and goes on until the last element called the positive index. There is also negative indexing which starts from -1 enabling you to access elements from the last to first. Let us now understand lists better with the help of an example program.

Tuple

Tuples are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed.

Sets

Sets are a collection of unordered elements that are unique. Meaning that even if the data is repeated more than one time, it would be entered into the set only once. It resembles the sets that you have learnt in arithmetic. The operations also are the same as is with the arithmetic sets. An example program would help you understand better.

Dictionary

Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value. We can define a dictionary by enclosing a comma-separated list of key-value pairs in curly braces ({}). A colon (:) separates each key from its associated value.

PYTHON -OBJECT ORIENTED CONCEPTS

Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy.

Overview of OOP Terminology

- **Class** – A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable** – A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member** – A class variable or instance variable that holds data associated with a class and its objects.
- **Function overloading** – The assignment of more than one behaviour to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Instance variable** – A variable that is defined inside a method and belongs only to the current instance of a class.
- **Inheritance** – The transfer of the characteristics of a class to other classes that are derived from it.
- **Instance** – An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation** – The creation of an instance of a class.
- **Method** – A special kind of function that is defined in a class definition.
- **Object** – A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- **Operator overloading** – The assignment of more than one function to a particular operator.

Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows –

```
class ClassName:  
    'Optional class documentation string'  
    class_suite
```

- The class has a documentation string, which can be accessed via *ClassName.__doc__*.
- The *class_suite* consists of all the component statements defining class members, data attributes and functions.

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its *__init__* method accepts.

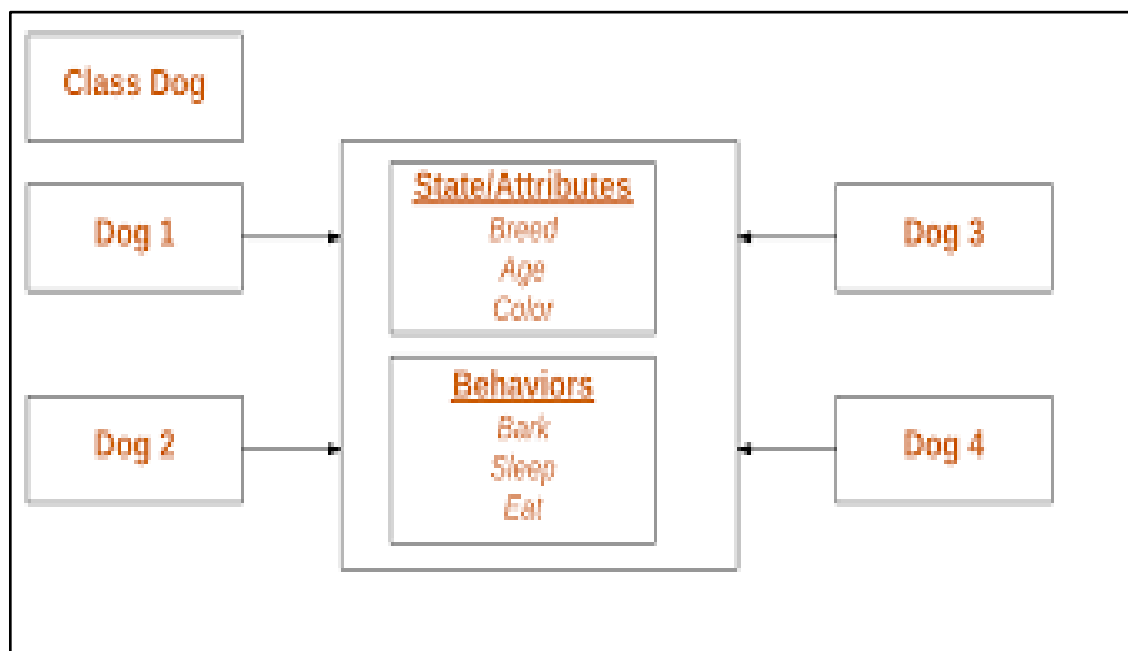


Fig 1.2b Depicts an example of Class “Dog”


```
class StrangerThings:
    eleven = 11

    def oneLiner(self):
        print('Friends Don\'t Lie')

print(StrangerThings.eleven)

print(StrangerThings.oneLiner)
```

Fig 1.2c Depicts a class named “StrangerThings”

In the above figure “StrangerThings” is a user defined class with class variable and a method “oneLiner”. Using the class name, we access the variable and method using dot operator.

INHERITANCE

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order. It is basically used for reusing of existing classes.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Types of inheritance:

1.Single inheritance

Class A is the super class and class B is the child class. B inherits all the features of A and add its own, unique elements like variable, methods etc.

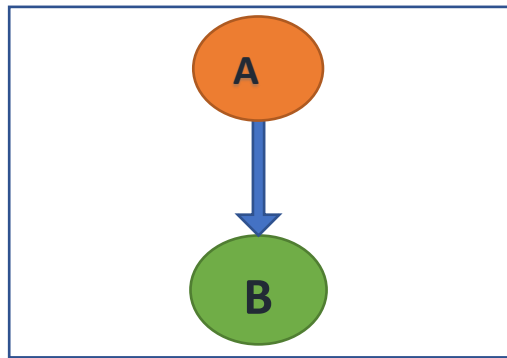


Fig 1.2d Single Inheritance

2. Multiple inheritance

Classes A, B and C are the super classes and D is the sub class or child class. D inherits all the properties of A, B and C.

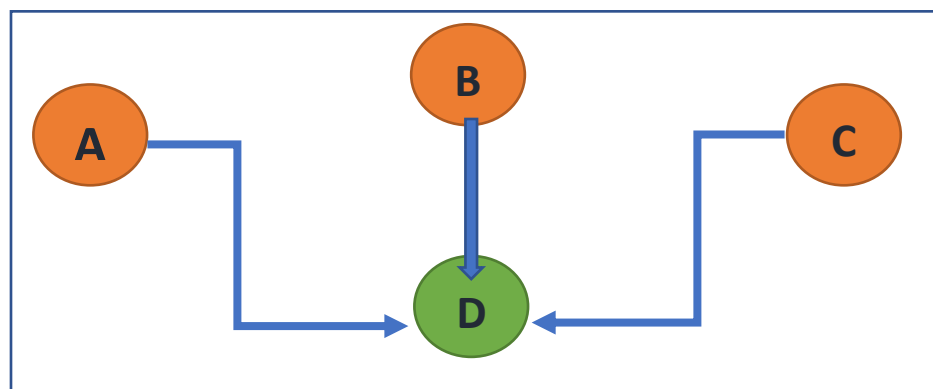


Fig 1.2e Multiple Inheritance

3. Multilevel inheritance

In this type of inheritance, we have different levels of inheritance. A is the parent of B which is level one inheritance and B is the parent of C which is level two inheritance imbibes all the features of A whereas C imbibes the features of both A and B , but the immediate parent of C is Here the class B acts as both parent and child depending on the level of inheritance.

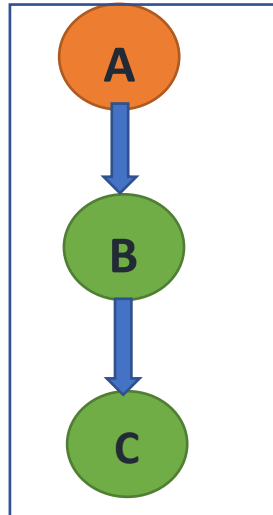


Fig 1.2f Multilevel Inheritance

4.Hierarchical inheritance

In this type of inheritance, we have one superclass and many sub classes. All the sub classes inherit the features of one parent class. Here B, C and D inherit all the features of A.

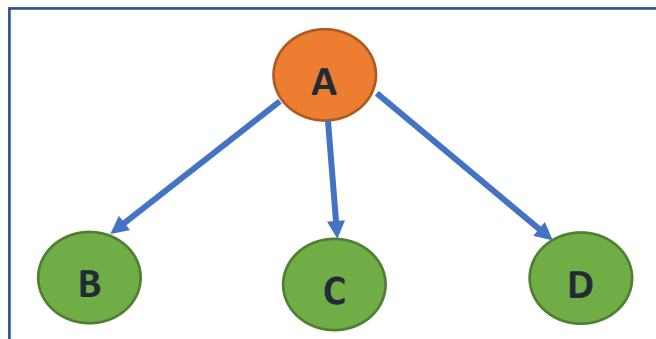


Fig 1.2g Hierarchical Inheritance

5.Hybrid inheritance

It is a combination of two or more inheritance. It is also known as Diamond inheritance. In the above below, blue arrow indicates hierarchical inheritance, dotted line indicated multilevel inheritance and red arrow indicates multiple inheritance. Classes B, C and D inherits the properties of A and Class E inherits all the features of B, C, D and A.

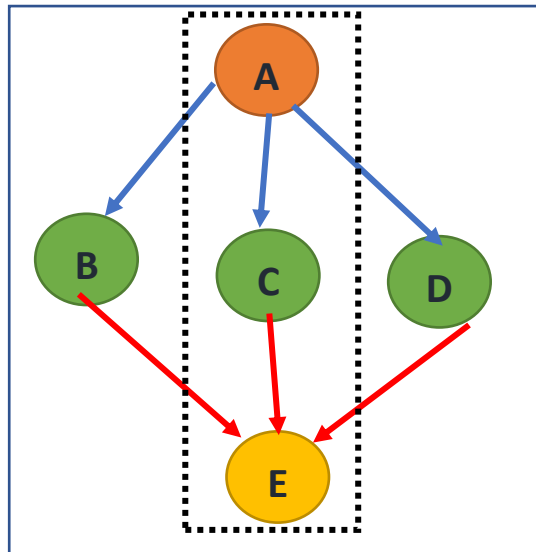


Fig 1.2h Hybrid Inheritance

POLYMORPHISM

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. As we have seen in the below example that we have defined the method `sound ()` and have the multiple implementations of it in the different-2 sub classes. **Types of polymorphism:** Run-time and Compile-time polymorphism.

The polymorphism is supported as:

1. Method Overloading – This is an example of compile time (or static polymorphism)
2. Method Overriding – This is an example of runtime time (or dynamic polymorphism)

ENCAPSULATION :Encapsulation is also an essential aspect of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

DATA ABSTRACTION :Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonyms because data abstraction is achieved through encapsulation. Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does.

DBMS FUNDAMENTALS

Database Management System or **DBMS** in short refers to the technology of storing and retrieving user's data with utmost efficiency along with appropriate security measures. This tutorial explains the basics of DBMS such as its architecture, data models, data schemas, data independence, E-R model, relation model, relational database design, and storage and file structure and much more.

ADVANTAGES OF DBMS

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics –

- **Real-world entity** – A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behaviour and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Relation-based tables** – DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- **Isolation of data and application** – A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.

APPLICATIONS OF DBMS

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information. Following are the important characteristics and applications of DBMS.

- **ACID Properties** – DBMS follows the concepts of **Atomicity**, **Consistency**, **Isolation**, and **Durability** (normally shortened as **ACID**). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- **Multuser and Concurrent Access** – DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- **Multiple views** – DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- **Security** – Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features.

DBMS ARCHITECTURE

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

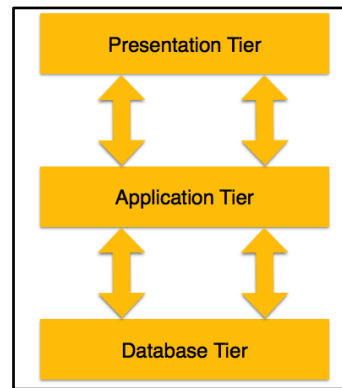


Fig 1.2i Three tier Architecture of DBMS

- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
- **User (Presentation) Tier** – End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

1.3 METHODOLOGY TO BE FOLLOWED

The “user” and the “individual” are synonymous. In any programming environment the terminology “user” is much preferred.

As stated in the introduction of the project Expense Tracker methodology is as follows:

- **Interactive bot:**

An Interactive Bot named “Pybot” allows the user to interact with the bot proving to be the most efficient guide for the user in explaining the process of expense tracking and answering to the query of the user. It is achieved using natural language processing tool kit which Python provides.” Pybot” is based on a text-based communication. However, there are advanced bots in many real-life applications proving to be the most efficient in reducing tedious job of humans.

- **User entering relevant data**

The project eminently implements the graphical user interface in mesmerizing the user with the Expense Tracking Application. Python supports with inbuilt modules and functionalities with which it is simple for a programmer to utilize the graphic features and can be efficiently implemented. Later it incepts with the window asking for a new or existing user. If new user then he/she should proceed for registration if not directly entering the expenses after validating the credentials will work for generating a report.

- **Data Visualizing and Data Organization:**

The user is asked to enter his personal details which in the backend are efficiently maintained in a SQL database and files. Proceeding with this, he is queried for entering the expense incurred on various categories. After which the graph (bar chart) is displayed. For a new user, graphical analysis is shown for current month and for existing user it is shown for previous and current months. Also, it is important to note that every user is given with auto-generated password which are different for all and it is to be remembered to go for a comparative analysis.

1.4 EXPECTED OUTCOMES

The expected outcomes are:

- The project “Expense Tracker” begins with the interactive bot named “Pybot” and it plays a major role in guiding through the process of expense tracking.
- Provides registration facilities to the new user and is queried to enter the expense for a month. For an existing user after validating the necessity credentials he can see the comparative analysis of expenses for the current and previous month.
- It allows only the system administrator in editing (adding and deleting), updating records and identifying a package in the database which results in the proper resource management.
- As it is application software it can be used by a wide variety of outlets (Financial agent, Corporate’s tracking of their employees’ expense) to automate the process of maintaining the records related to the subject of expense details and cash flows.
- The Expense Tracker is basically updating the manual expense inventory system to automated inventory system, so that organization can manage their record in efficient and organized form.
- Lastly, I believe that it is necessary for us to be aware of the money spent on various activities which supports us in making vital decisions and planning of investments in future.
- **Project Characteristics**
 - a) Record management
 - b) Data Security
 - c) Data Visualization
 - d) Data Analysing

CHAPTER 2

REQUIREMENT SPECIFICATION

2.1 HARDWARE REQUIREMENT

- **Hardware Requirement:**
 1. Processor: Pentium processor, i3 or i5(sixth generation) (any one)
 2. Ram :4 GB or 8 GB
 3. Other support(monitor(1080p), camera, etc)

2.2 SOFTWARE REQUIREMENT

- **Software Support:**
 1. Operating System: Windows 10 Home
 2. Developing Tool:
PyCharm Community Edition IDE (Integrated Development Environment) (Version :2020.2.1) or any other platform
Python Interpreter (Version :3.8.5)
 3. Database Tool:
MySQL Community Edition (Version :8.0)

CHAPTER 3

ER MODEL

3.1 ENTITY AND ATTRIBUTES

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints. ER Model is best used for the conceptual design of a database. ER Model is based on – **Entities** and their *attributes* and **Relationships** among entities.

These concepts are explained below.

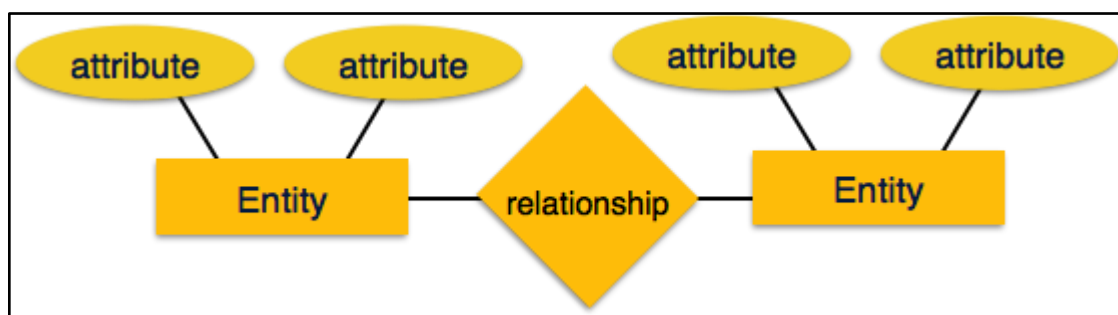


Fig 3.1a ER Model

- **Entity** – An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** – The logical association among entities is called **relationship**. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

Mapping cardinalities –

- one to one
- one to many

- many to one
- many to many

Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.

The main highlights of this model are –

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

ER DIAGRAM REPRESENTATION

Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).

If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse. **Multivalued** attributes are depicted by double ellipse. **Derived** attributes are depicted by dashed ellipse.

In “**Expense Tracker**” four strong entities are identified:

- **ADMIN** : It has attributes Userid, User name, User phone and others.
- **CATEGORY** : It has attributes idCategory(Primary Key), Category_name

- **CUMULATIVE EXPENSE** :It has Usercount(Primary Key),Userid,Usermonth,Current year and others.
- **MONTHLY EXPENSE** : It has Usercount(Primary Key),Userid,month and others.

ER DIAGRAM

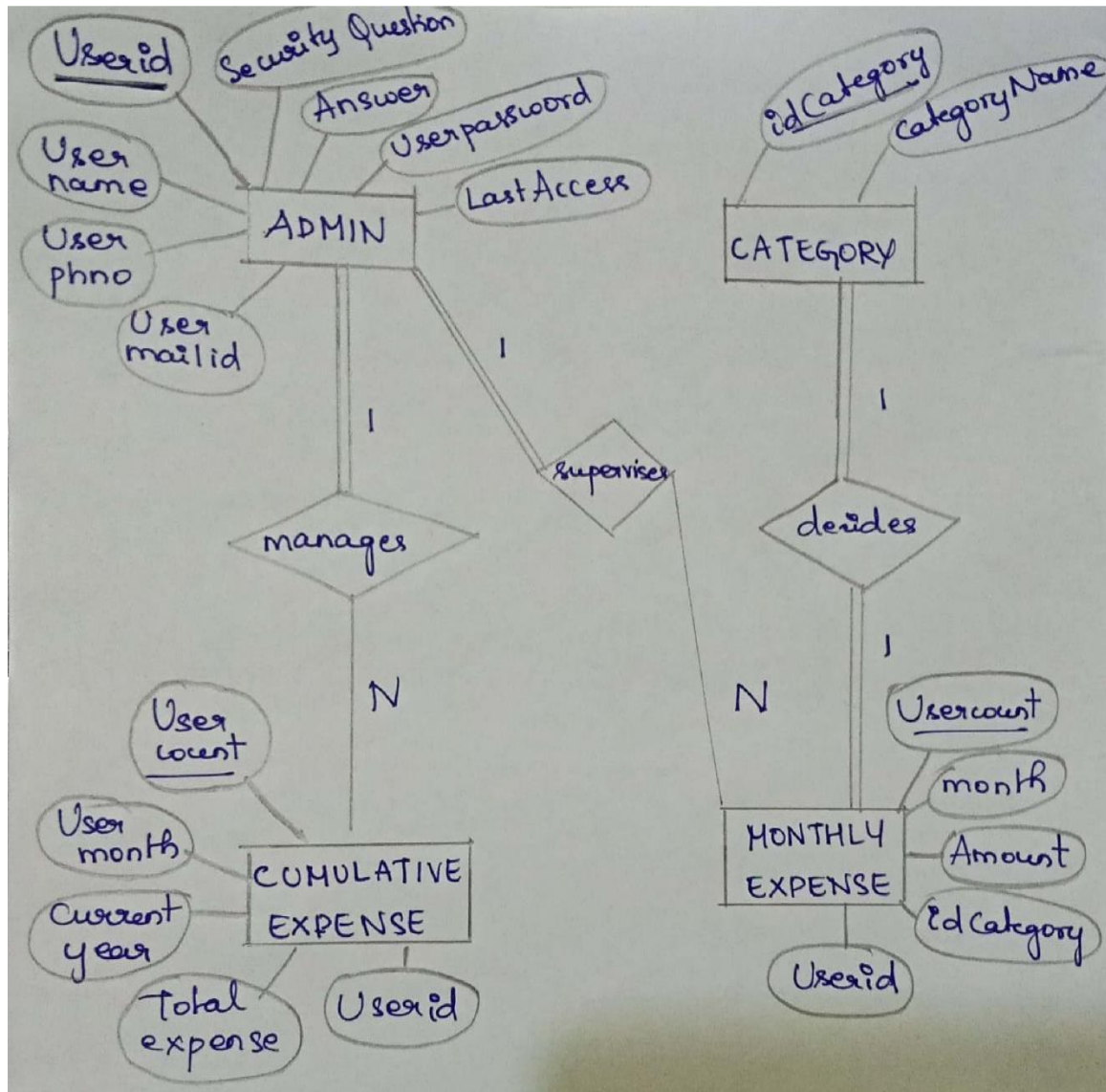


Fig 3.1b ER Diagram of "Expense Tracker"

3.2 KEYS, RELATIONSHIP AND PARTICIPATION

The different types of keys in DBMS are –

- **Candidate Key** - The candidate keys in a table are defined as the set of keys that is minimal and can uniquely identify any data row in the table.
- **Primary Key** - The primary key is selected from one of the candidate keys and becomes the identifying key of a table. It can uniquely identify any data row of the table.
- **Super Key** - Super Key is the superset of primary key. The super key contains a set of attributes, including the primary key, which can uniquely identify any data row in the table.
- **Composite Key** - If any single attribute of a table is not capable of being the key i.e. it cannot identify a row uniquely, then we combine two or more attributes to form a key. This is known as a composite key.
- **Secondary Key** - Only one of the candidate keys is selected as the primary key. The rest of them are known as secondary keys.
- **Foreign Key** - A foreign key is an attribute value in a table that acts as the primary key in another. Hence, the foreign key is useful in linking together two tables. Data should be entered in the foreign key column with great care, as wrongly entered data can invalidate the relationship between the two tables.

Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

Binary Relationship and Cardinality

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instances of an entity from a relation that can be associated with the relation.

- **One-to-one** – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.
- **One-to-many** – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.
- **Many-to-one** – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.
- **Many-to-many** – The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.

Participation Constraints

- **Total Participation** – Each entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation** – Not all entities are involved in the relationship. Partial participation is represented by single lines.

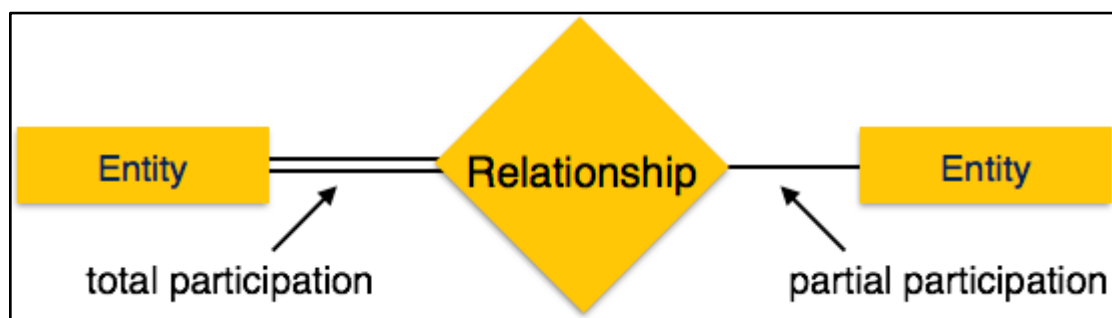


Fig 3.2a Participation Constraint

In “**Expense Tracker**” the scenario is as:

ADMIN to CUMULATIVE EXPENSE has 1: N cardinality ratio with partial on cumulative expense side as it is assumed that one administrator manages many cumulative expenses but vice-versa is not true.

ADMIN to MONTHLY EXPENSE has 1: N cardinality ratio with partial on monthly expense side as it is assumed that one administrator supervises many monthly expenses but vice-versa is not true.

CATEGORY to MONTHLY EXPENSE has 1:1 cardinality ratio meaning that at least one category should have been spent for one monthly expense.

Steps to map ER Diagram from Relational Schema

1) Mapping Strong Entities:

Here there are four strong entities which are mapped to relations directly.

2) Mapping Weak Entity:

Here there is no weak entity. Hence this step is skipped.

3) Mapping of 1:1 Relationship:

Here there exist one relation decides with 1:1 Cardinality ratio and participation constraint total on both the sides. This means that the Primary Key from any one side can be moved as Foreign Key to other side. Here the Primary Key idCategory is moved as Foreign Key to MONTHLY EXPENSE.

4) Mapping of 1: N Relationship:

Here there exists two relations with 1: N Cardinality ratio. Here the Primary Key is moved from ‘1’ side to ‘N’ side. Hence the Primary Key of ADMIN is moved to MONTHLY EXPENSE and CUMULATIVE EXPENSE as Foreign Key.

5) Mapping of M: N Relationship:

There is no relation with M: N cardinality ratio. Hence this step is skipped.

6) Mapping of Multi Valued Attribute:

Here there is no multi valued attribute. Hence this step is skipped.

7) Mapping of Ternary Relationship:

Here there is no Ternary Relationship. Hence this step is skipped.

RELATIONAL SCHEMA

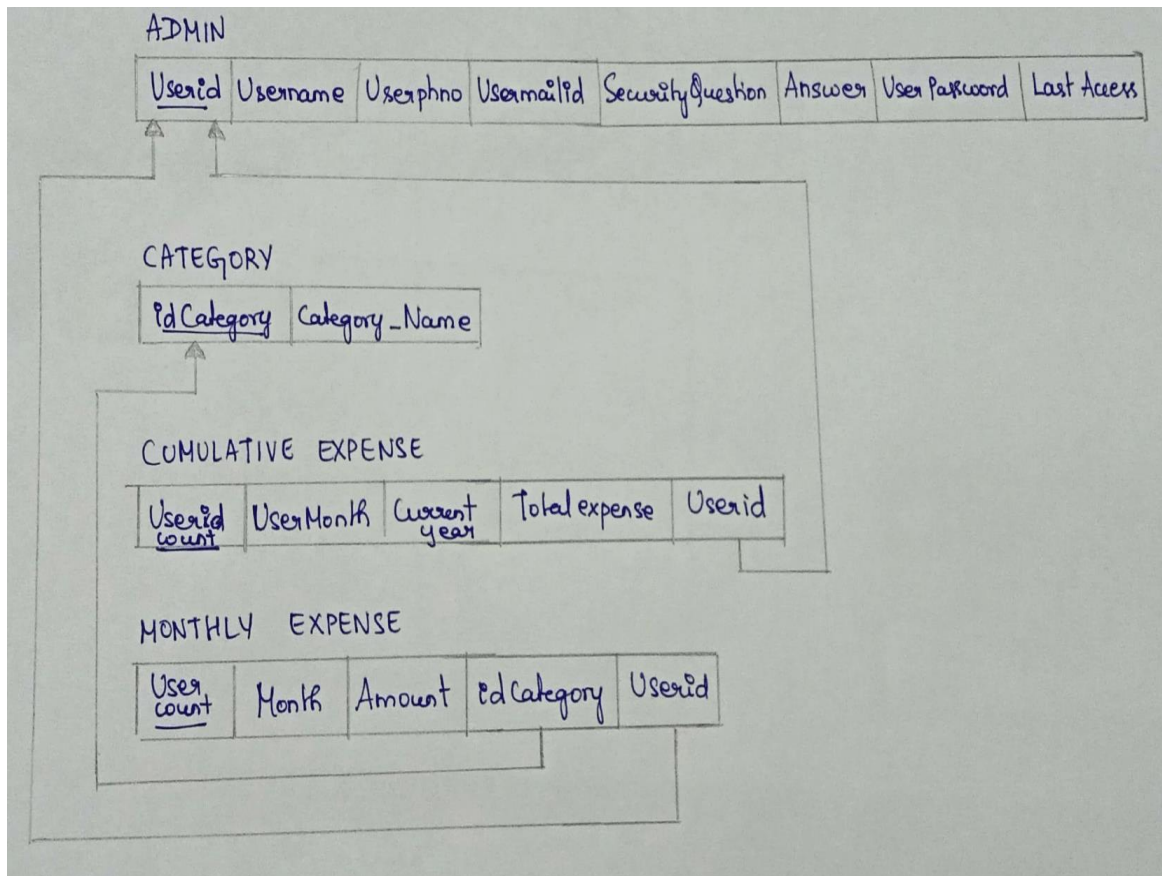


Fig 3.2b Mapping of ER Diagram to Relational Schema

CHAPTER 4

IMPLEMENTATION

4.1 GUI PROGRAMMING

Graphical User Interface is nothing but which provides an interface to interact with the computer. Python supports for GUI majorly with the help of four modules: Kivy, Python, wxPython, Tkinter.

Expense Tracker uses extensively Tkinter to interact with the users allowing them to enter the expense based on different category, validating the credentials in terms of existing user and many others are effectively implemented using GUI.

Tkinter module is available in default and it is simple to implement.

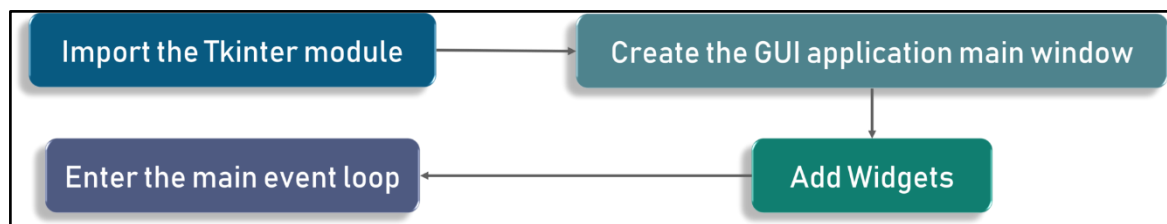


Fig 4.1a: Workflow of Tkinter

The Figure shown above describes that after the importing of Tkinter module in a Python application, a window has to be created so that all widgets can be added and the event should enter into main loop whenever the user triggers it. The basic and frequently used widgets are: Button, Checkbox, Choice, Frame, Panel, Label, List, Scrollbar, TextArea, TextField. For each of the above widgets, there are corresponding classes which has its own constructors for initialization and methods. Using this class, an object of desired type can be created. When an object is created and initialized, it then can be placed on a method, which is defined in their corresponding classes.

4.2 SQL

SQL (Structured Query Language) is a language of Relational Database System, it includes database creation, deletion, fetching rows and modifying rows etc. SQL is an ANSI (American National Standards Institute) standard but there are many different versions of the SQL language.

SQL Architecture

When you are executing an SQL command for any RDBMS (Relational Database Management System), the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc.

SQL Commands

The standard SQL commands to interact are classified into groups based on their nature:

- **Data Definition Language**: DDL defines the conceptual schema providing a link between the logical and the physical structures of the database. Commands under this include: **CREATE, ALTER, DROP**.
- **Data manipulation Language (DML)**: DML provides the data manipulation techniques and facilitates use of relationship between the records. Commands under this include: **INSERT, UPDATE, DELETE**.
- **Data Control Language (DCL)**: DCL provides the facility for controlling the access to the user. Commands include: **GRANT, REVOKE**.
- **Data Query Language (DQL)**: DQL provides the querying facilities, like fetching certain or all the records from the database. Commands include: **SELECT**.

4.3 DATA VISUALISATION USING PYTHON

Data visualization is the technique to present the data in a pictorial or graphical format. “Expense Tracker” uses data visualization to raise awareness to the user about the unnecessary expense spent on various categories. However, visualizing proves to be more useful to the stake holders and financial experts.

The main benefits of data visualization are as follows:

- It simplifies the complex quantitative information
- It helps analyse and explore big data easily
- It identifies the areas that need attention or improvement
- It identifies the relationship between data points and variables
- It explores new patterns and reveals hidden patterns in the data

Three major considerations for Data Visualization:

- Clarity
- Accuracy
- Efficiency

Python Libraries such as matplotlib, Vispy, bokeh, Seabor, pygal, folium are used for data visualization. The matplotlib has emerged as the main data visualization library. “Expense Tracker” uses matplotlib along with NumPy and Pandas library to interpret the data efficiently and is successful in its motive. Matplotlib is a python two-dimensional plotting library for data visualization and creating interactive graphics or plots. Using python's matplotlib, the data visualization of large and complex data becomes easy. The different types of plots are: Histogram, Scatter Plot, Heat Map, Pie Chart, Error Bar. The plots used in the project are Scatter Plot, Bar Plot, Grouped Bar Plot. NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the NumPy package and its key data structure is called the Data Frame. Data Frames allow you to store and manipulate tabular data in rows of observations and columns of variables.

CHAPTER 5

RESULTS

The results of the Project entitled “Expense Tracker” are:

It allows the user to register first using GUI for a new user followed by querying of expense spent on various categories and displaying the graph of the same. For an existing user the credentials are validated after which the grouped bar plot and scatter plot are shown presenting a comparative analysis. An interactive chatbot (text based one) at the beginning guides through the process of entire application.

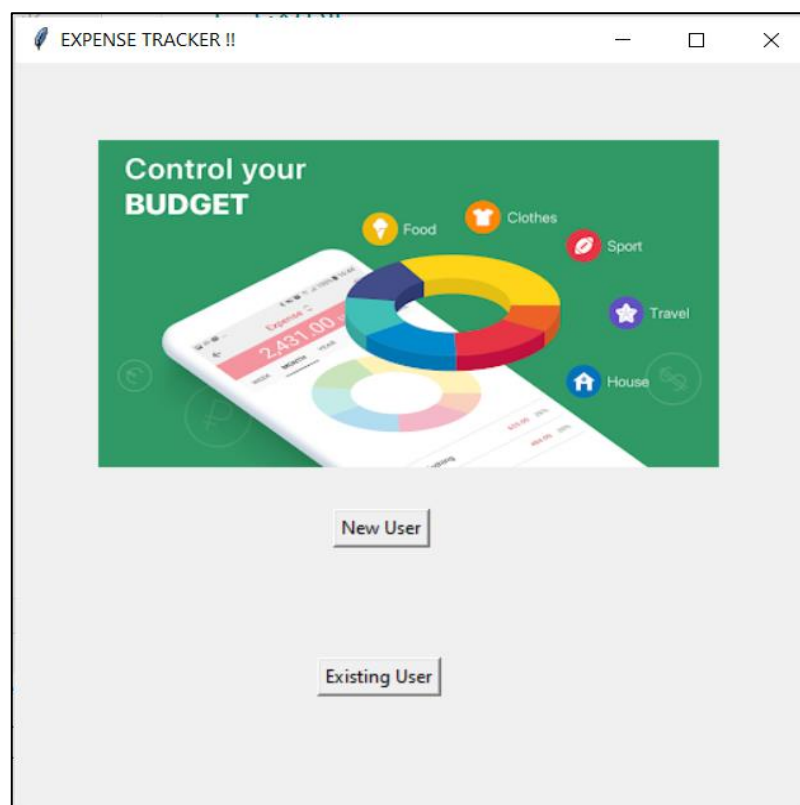
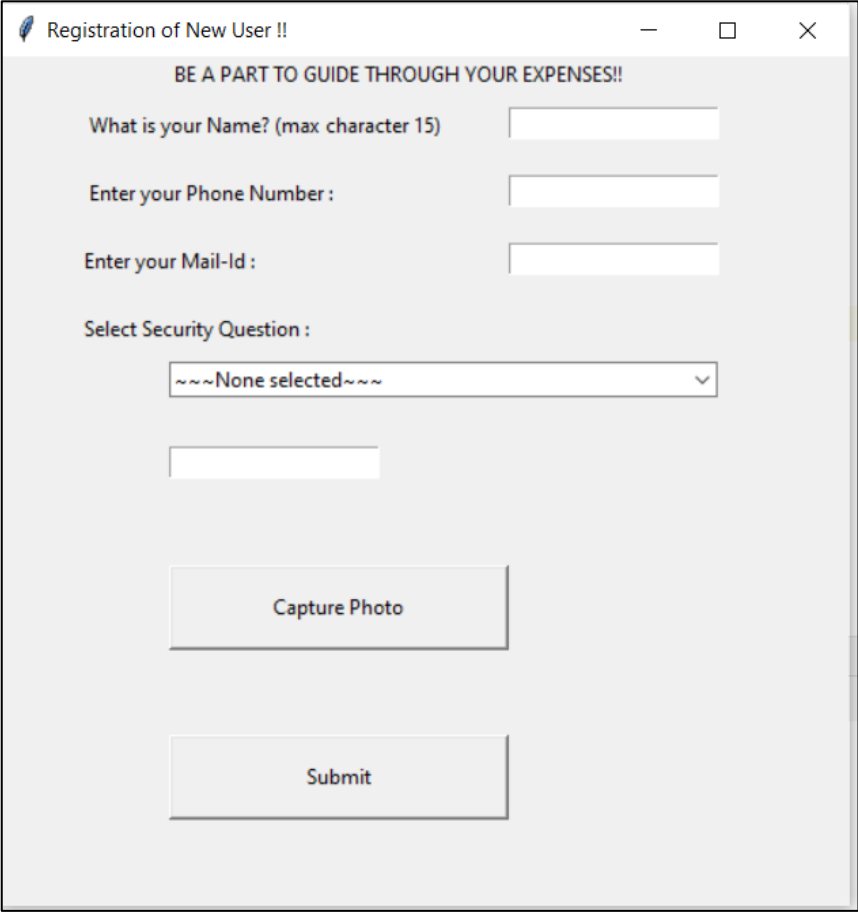


Fig 5.1 GUI asking for new User and Existing User



Registration of New User !!

BE A PART TO GUIDE THROUGH YOUR EXPENSES!!

What is your Name? (max character 15)

Enter your Phone Number :

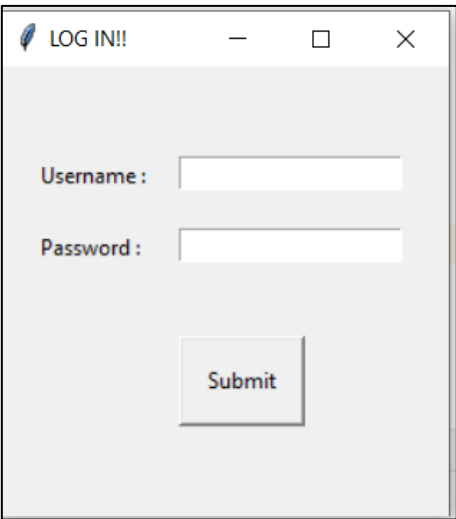
Enter your Mail-Id :

Select Security Question :

~~~None selected~~~

Capture Photo

Submit

**Fig 5.2 Registration of New User**

LOG IN!!

Username :

Password :

Submit

**Fig 5.3 Validating Credentials for Existing User**

EXPENSE DETAILS !!

— □ ×

RECORDING YOUR EXPENSES !!

Select month :

Current year 2020

\* Max no. of characters allowed 9 for expense field \*

Enter the amount spent on Grocery :

Enter Medical expenses :

Enter Shopping expenses :

Enter Entertainment expenses :

Enter Tax expenses :

Enter House-rent expenses :

Enter House-EB expenses :

Enter House-WaterBill expenses :

Enter House-Furniture expenses :

Enter House-Electronic expenses :

Enter House-Gas expenses :

Enter Personal expenses :

Enter Loan expenses :

Enter Insurance expenses :

Enter Educational expenses :

Submit

Fig 5.4 Asking Expenses for the existing User

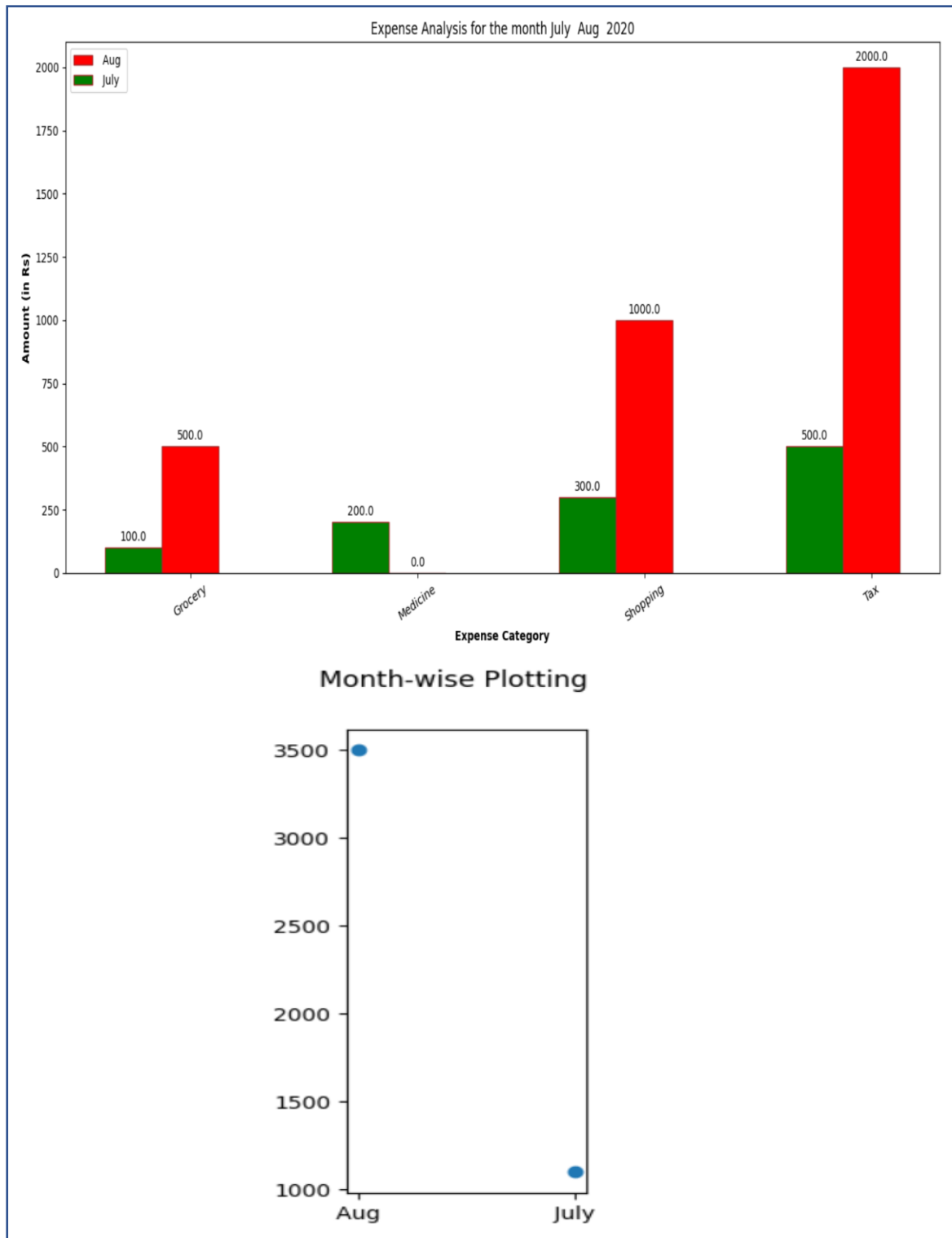


Fig 5.5 Grouped Bar Plot and Scatter Plot for a User



## CHAPTER 6

### CONCLUSION

Thus, the following conclusions can be drawn from the Project "Expense Tracker":

- Expense management system was developed to ensure the security of information and reliability of agency records when accessing and providing services to the customers.
- The information gathered during the data collection was properly analyzed and the results provided the basis for the new system.
- The system was tested and found to be functional and the outputs produced by this system were encouraging.
- The application will hence reduce the loss of information unlike the existing system and also information will be processed fast.
- Effective implementation of this software will take care of the basic requirements of the expense management system because it is capable of providing easy and effective storage of information related to activities happening in the stipulated area. With these, the objectives of the system design will be achieved.
- In order to allow for future expansion, the system has been designed in such a way that will allow possible modification as it may deem necessary by the user system, whenever the idea arises.

## APPENDIX A: PROJECT CODE

### ExpenseTracer.py

```

from tkinter import ttk
from AskExisting import existing
from LanguageChatBot import *
import time
import string
import random
from AskExpenseApp import *
from PIL import ImageTk, Image
# SINGLE LEVEL INHERITANCE
class SampleApp(tk.Tk):
# FUNCTION 1 : SHOWS GUI TO REGISTER THE USER
    def __init__(self):
        tk.Tk.__init__(self)
        self.geometry("500x500")
        self.title("Registration of New User !! ")
        self.label8 = tk.Label(self, text="BE A PART TO GUIDE THROUGH YOUR EXPENSES!!").place(x=100)
        self.label = tk.Label(self, text="What is your Name? (max character 15)").place(x=50, y=30)
        self.entry = tk.Entry(self)
        self.entry.place(x=300, y=30)
        self.label2 = tk.Label(self, text="Enter your Phone Number :").place(x=50, y=70)
        self.entry2 = tk.Entry(self)
        self.entry2.place(x=300, y=70)
        self.label4 = tk.Label(self, text="Enter your Mail-Id :", width=20).place(x=28, y=110)
        self.entry4 = tk.Entry(self)
        self.entry4.place(x=300, y=110)
        self.label6 = tk.Label(self, text="Select Security Question :").place(x=47, y=150)
        self.questions = tk.ttk.Combobox(self, width=50, state="readonly")
        self.questions.place(x=100, y=180)
        self.questions['values'] = ('~~~None selected~~~', ' What is your Nick-Name ?',
            ' What is your Grandmother\'s name?(Mother side)',
            ' Which is Your Favourite place?',
            ' What is your Best Friend\'s name?',
            ' What is your Grandfather\'s Occupation?(Father side)',
            ' Who is the Internet Service Provider(at home)?',
            ' Which is your favourite book?',
            ' What is the name of your First School?',
        )
        self.questions.current(0)
        self.entry6 = tk.Entry(self)
        self.entry6.place(x=100, y=230)
        self.button = tk.Button(self, text="Submit", command=self.on_button)
        self.button.place(bordermode=OUTSIDE, x=100, y=400, width=200, height=50)

```

# FUNCTION 2: CALLED WHEN SUBMIT IS CLICKED TO CHECK THE VALID DETAILS IS ENTERED BY USER OR NOT

```
def on_button(self):
    flag = 1
    regex = '^([a-z0-9]+[\._]?[a-z0-9]+@[a-z0-9]+\w{2,3})$'
    name = self.entry.get()
    phone = self.entry2.get()
    email = self.entry4.get()
    option1 = self.entry6.get()
    current = str(self.questions.get());
    if len(name) <= 0 or len(name) > 15:
        flag = 0
        self.label1 = tk.Label(self, text='* Invalid Name! Entered *', fg='red').place(x=300, y=50)
    if len(phone) > 10 or len(phone) < 10:
        flag = 0
        self.label3 = tk.Label(self, text='* Invalid Phone Number! *', fg='red').place(x=300, y=90)
    if (not re.search(regex, email)):
        flag = 0
        self.label5 = tk.Label(self, text='* Invalid Email *', fg='red').place(x=300, y=130)
    if len(option1) == 0:
        flag = 0
        self.label7 = tk.Label(self, text='* Enter the Answer for the Question Chosen *',
fg='red').place(x=100,y=250)
    if (flag == 1):
        temp = self.password()
        messagebox.showinfo("Password is:", temp)
        time.sleep(2)
        messagebox.showinfo("Welcome", "Successful registration")
        self.destroy()
        top.destroy()
        s1 = sql()
        s1.insert(name,phone,email,current,option1,temp)
        s1.insertexpenseintoApp()
```

# FUNCTION 3: TO AUTO GENERATE PASSWORD

```
def password(self):
    combinedstring = string.ascii_letters + string.digits + string.punctuation
    print(type(combinedstring))
    userpassword = ""
    for i in range(8):
        userpassword = userpassword + random.choice(combinedstring)
    return userpassword

top = Tk()
top.title("EXPENSE TRACKER !!")
top.geometry("500x500")
photo = ImageTk.PhotoImage(Image.open("C:/Python/ExpenseTracker/Picture/ControlyourExpense.webp"))
logo = Label(top, image=photo).place(x=50, y=50)
B = Button(top, text="New User", command=SampleApp, activebackground="yellow",
activeforeground="red",
highlightcolor='green')
```

```
B.place(x=200, y=300)
# FUNCTION OF EXPENSE TRACER MODULE CALLED WHEN EXISTING USER VISITS
def check():
    def nameandpass():
        print('Hrlllo')
        un=Existing.entry2.get()
        up=Existing.entry3.get()
        if len(un)==0 or len(up)==0:
            alert = Tk()
            alert.geometry("100x100")
            messagebox.showwarning("Warning", "Please enter !!")
            alert.mainloop()
            alert.destroy()

        print(un,up)
        mydb1 = mysql.connector.connect(
            host="127.0.0.1",
            user="root",
            password="indira@123"
        )
        mycursor1 = mydb1.cursor()
        mycursor1.execute("use expense_tracker_app;")
        mycursor1.execute("SELECT * FROM `v1` WHERE Username='%s';" % (un))
        userrecord = mycursor1.fetchall()
        b=userrecord[0]
        print(b)
        if not userrecord:
            alert = Tk()
            alert.geometry("100x100")
            messagebox.showwarning("Warning", " Please Register")
            alert.mainloop()
        else:
            a = userrecord[0]
            userid=a[0]
            username=a[1]
            userpassword=a[2]
            userq=a[3]
            usera=a[4]
            if userpassword==up:
                msg2 = messagebox.showinfo("Welcome", "Successful Login")
                Existing.destroy()

            def securityquestion():
                answer = Existing1.entry2.get()
                if len(answer) == 0 or answer != usera:
                    alert = Tk()
                    alert.geometry("100x100")
                    messagebox.showwarning("Warning", "Please enter valid answer!!!")
                    alert.mainloop()
                    alert.destroy()
                elif answer == usera:
                    msg2 = messagebox.showinfo("Welcome", "Successful Login")
```

```
Existing1.destroy()
e1=existing(username,userpassword,userq,usera,userid)
e1.update()
Existing1=Tk()
Existing1.title("SECURITY QUESTION!!")
Existing1.geometry("400x400")
Existing1.label1 = tk.Label(Existing1, text=" Please enter Answer for the question: !!",
font=('Helvetica', 10, 'bold')).place(x=100, y=10)

Existing1.label2 = tk.Label(Existing1, text=userq).place(x=20, y=50)
Existing1.entry2 = tk.Entry(Existing1)
Existing1.entry2.place(x=100, y=100)

Existing1.button1 = tk.Button(Existing1, text="Login", command=securityquestion)
Existing1.button1.place(bordermode=OUTSIDE, x=100, y=200, width=70, height=50)
Existing1.mainloop()
else:
    alert = Tk()
    alert.geometry("100x100")
    messagebox.showwarning("Warning", "Password Wrong !!")
    alert.mainloop()

Existing=Tk()
Existing.title("LOG IN!!")
Existing.geometry("250x250")
Existing.label1=tk.Label(Existing,text="LOGIN !!",font=('Helvetica',10,'bold')).place(x=250,y=10)

Existing.label2=tk.Label(Existing,text="Username :").place(x=20,y=50)
Existing.entry2 = tk.Entry(Existing)
Existing.entry2.place(x=100, y=50)

Existing.label3=tk.Label(Existing,text="Password :").place(x=20,y=90)
Existing.entry3 = tk.Entry(Existing,show='*')
Existing.entry3.place(x=100, y=90)

Existing.button1 = tk.Button(Existing, text="Submit", command=nameandpass)
Existing.button1.place(bordermode=OUTSIDE, x=100, y=150, width=70, height=50)
Existing.mainloop()

B1 = Button(top, text="Existing User", command=check)
B1.place(x=190, y=400)
chatty()
top.mainloop()
app = SampleApp()
app.mainloop()
```

## LanguageChatBot.py

```

from nltk.chat.util import Chat, reflections
def chatty():
    print("Hi, I'm Pybot and I chat alot ;) \nPlease type lowercase English language to start a conversation.
    Type quit to leave \nHow can I help you? ") #default message at the start
    chat = Chat(pairs, reflections)
    chat.converse()
pairs = [
    [
        r"What is the process|What is the Procedure|What are the steps(.*) are involved?|done?",
        ["First of all,you need to register if you are the new user.\nThen you need to enter all expense spent
        by you to get a report",
        "If you are a new user please register,if you are existing user please enter valid credentials."],
    ],
    [
        r"Can you help me|Can you guide me through the process?",
        ["Sure first please register by selecting on new user button and get a graphical interpretation of the
        expense entered by you..\nThen the next time you visit us you "+
        "can see the previous and current month report ."]],
    ],
    [
        r"\n",
        ["Sorry I didnt understand .."]],
    ],
    [
        r"What can you do?(.*)",
        ["I can guide you through the process of Expense Tracking"]],
    ],
    [
        r"How do you(.*)track expense?|How is it done?",
        ["You need to enter expense for current month for a new user.\nBased on the expense,you can see
        a graphical analysis,thereby\n"+
        "creating awareness to you.", "For a new user graphical analysis of current month is shown,\n for
        current month existing and previous "+
        "month is depicted"]],
    ],
    [
        r"(.*)\n",
        ["Sorry,I didnt get you .."]],
    ],
    [
        r"my name is (.*)",
        ["Hello %1, How are you today ?"]],
    ],
    [
        r"what is your name ?",
        ["My name is PyBot and I'm a chatbot "]],
    ],
    [
        r"how are you ?",
        ["I'm doing good\nHow about You ?"]],
    ],

```

```

    ],
    [
        r"I am good|I am fine|Awesome|Great|fantastic|fine|yes fine",
        ["Good to hear that"]
    ],
    [
        r"sorry (.*)",
        ["Its alright","Its OK, never mind"]
    ],
    [
        r"I'm (.*) doing good",
        ["Nice to hear that","Alright :)"]
    ],
    [
        r"hi|hey|hello",
        ["Hello", "Hey there"]
    ],
    [
        r"(.*) age?",
        ["I'm a computer program dude\nSeriously you are asking me this?"]
    ],
    [
        r"what (.*) want ?",
        ["Make me an offer I can't refuse"]
    ],
    [
        r"(.*) created ?",
        ["Jayasree created me using Python's NLTK library ","top secret ;)"]
    ],
    [
        r"(.*) (location|city) ?",
        ['Chennai, Tamil Nadu']
    ],
    [
        r"how is weather in (.*)?",
        ["Weather in %1 is awesome like always","Too hot man here in %1","Too cold man here in %1","Never even heard about %1"]
    ],
    [
        r"I work in (.*)?",
        ["%1 is an Amazing company, I have heard about it. But they are in huge loss these days."]
    ],
    [
        r"(.*)raining in (.*)",
        ["No rain since last week here in %2","Damn its raining too much here in %2"]
    ],
    [
        r"how (.*) health(.*)",
        ["I'm a computer program, so I'm always healthy "]
    ],
    ],

```

```
[
    r"(.*) (sports|game) ?",
    ["I'm a very big fan of Football"]
],
[
    r"who (.*) sportsperson ?",
    ["Messy", "Ronaldo", "Roony"]
],
[
    r"who (.*) (moviestar|actor)?",
    ["Allu Arjun"]
],
[
    r"quit",
    ["Bye take care. See you soon :) ", "It was nice talking to you. See you soon :)"]
],
]
```

### AskExpenseApp.py

```
import mysql as mysql
import mysql.connector
import tkinter as tk
from tkinter import *
from tkinter import ttk, messagebox
import pandas as pd
import matplotlib.pyplot as plt

class sql:
    def __init__(self):
        pass
    def insert(self, name, phone, email, question, answer, password):
        mydb = mysql.connector.connect(
            host="127.0.0.1",
            user="root",
            password="indira@123"
        )
        mycursor = mydb.cursor()
        mycursor.execute("use expense_tracker_app;")
        sql = """INSERT INTO `expense_tracker_app`.`admin`(`Username`, `Userphno`, `Usermailid`, `Securityquestion`, `Answer`, `Userpassword`)
VALUES(%s,%s,%s,%s,%s,%s)"""
        values=(name, phone, email, question, answer, password)
        mycursor.execute(sql, values)
        mydb.commit()
        print(mycursor.lastrowid, "record inserted.")
        print(mycursor.rowcount)

    def insertexpenseintoApp(self):
        category_expense = dict()
        def submit():
            def check(k,n):
                try:
```



```

        d=float(n)
        category_expense[k]=n
    except ValueError:
        category_expense[k]=0.00
num=App1.entry3.get()
num1=App1.entry4.get()
num2=App1.entry5.get()
num3=App1.entry6.get()
num4=App1.entry7.get()
num5=App1.entry8.get()
num6=App1.entry9.get()
num7=App1.entry10.get()
num8=App1.entry11.get()
num9=App1.entry12.get()
num10=App1.entry13.get()
num11=App1.entry14.get()
num12=App1.entry15.get()
num13=App1.entry16.get()
num14=App1.entry17.get()
if len(num) > 9 or len(num1) > 9 or len(num3) > 9 or len(num4) > 9 or len(num5) > 9 or len(num6) >
9 or len(num7) > 9 or len(num8) > 9 or len(num9) > 9 or len(num10) > 9 or len(num11) > 9 or len(num12)
> 9 or len(num13) > 9 or len(num14) > 9:
    alert= Tk()
    alert.geometry("100x100")
    messagebox.showwarning("Warning", " Max 9 digits")
    alert.mainloop()
if App1.month.current()==0:
    alert1= Tk()
    alert1.geometry("100x100")
    messagebox.showwarning("Warning", " Choose a month ")
    alert1.mainloop()
check('Grocery', num)
check('Medicine',num1)
check('Shopping',num2)
check('Entertainment',num3)
check('Tax',num4)
check('Houserent',num5)
check('Houseeb',num6)
check('Housewb',num7)
check('Housefurniture',num8)
check('Houseelectronic',num9)
check('Housegas',num10)
check('Personal',num11)
check('Loan',num12)
check('Insurance',num13)
check('Education',num14)
print(category_expense)
self.insertexpenseintodb(category_expense, App1.month.get())
App1=Tk()
App1.title("EXPENSE DETAILS !!")
App1.geometry("700x1100")

```

```

App1.label = tk.Label(App1, text="RECORDING YOUR EXPENSES !!",font=("Helvetica",
10,'bold')).place(x=250,y=10)

App1.label1=tk.Label(App1,text='Select month :').place(x=10,y=50)
App1.month = tk.ttk.Combobox(App1, width=40, state="readonly")
App1.month.place(x=400,y=50)
App1.month['values'] = ('~~~None selected~~~',' Jan',' Feb', ' Mar', ' Apr',' May ', ' June ', ' July ',
' Aug ', ' Sep ', ' Oct ', ' Nov ', ' Dec ')
App1.month.current(0)
App1.label2=tk.Label(App1,text='Current year').place(x=10,y=90)
App1.label18=tk.Label(App1,text="* Max no. of characters allowed 9 for expense field
*",fg='red').place(x=400,y=110)
import datetime
x = datetime.datetime.now()
App1.label2=tk.Label(App1,text=x.year).place(x=400,y=90)
App1.label3=tk.Label(App1,text="Enter the amount spent on Grocery
:",font=("Helvetica",10)).place(x=10,y=130)
App1.entry3=tk.Entry(App1)
App1.entry3.place(x=400, y=130)
App1.label4 = tk.Label(App1, text="Enter Medical expenses :",font=("Helvetica",10)).place(x=10,
y=170)
App1.entry4 = tk.Entry(App1)
App1.entry4.place(x=400, y=170)
App1.label5 = tk.Label(App1, text="Enter Shopping expenses :",font=("Helvetica",10)).place(x=10,
y=210)
App1.entry5 = tk.Entry(App1)
App1.entry5.place(x=400, y=210)
App1.label6= tk.Label(App1, text="Enter Entertainment expenses
:",font=("Helvetica",10)).place(x=10, y=250)
App1.entry6 = tk.Entry(App1)
App1.entry6.place(x=400, y=250)
App1.label7 = tk.Label(App1, text="Enter Tax expenses :", font=("Helvetica",10)).place(x=10, y=290)
App1.entry7 = tk.Entry(App1)
App1.entry7.place(x=400, y=290)
App1.label8 = tk.Label(App1, text="Enter House-rent expenses :", font=("Helvetica",10)).place(x=10,
y=330)
App1.entry8 = tk.Entry(App1)
App1.entry8.place(x=400, y=330)
App1.label9= tk.Label(App1, text="Enter House-EB expenses :", font=("Helvetica",10)).place(x=10,
y=370)
App1.entry9= tk.Entry(App1)
App1.entry9.place(x=400, y=370)
App1.label10 = tk.Label(App1, text="Enter House-WaterBill expenses :",
font=("Helvetica",10)).place(x=10, y=410)
App1.entry10 = tk.Entry(App1)
App1.entry10.place(x=400, y=410)
App1.label11 = tk.Label(App1, text="Enter House-Furniture expenses :",
font=("Helvetica",10)).place(x=10, y=450)
App1.entry11 = tk.Entry(App1)
App1.entry11.place(x=400, y=450)
App1.label12 = tk.Label(App1, text="Enter House-Electronic expenses :", font=("Helvetica",
10)).place(x=10,y=490)

```

```

App1.entry12 = tk.Entry(App1)
App1.entry12.place(x=400, y=490)
App1.label13= tk.Label(App1, text="Enter House-Gas expenses :", font=("Helvetica",10)).place(x=10,
y=530)
App1.entry13= tk.Entry(App1)
App1.entry13.place(x=400, y=530)
App1.label14 = tk.Label(App1, text="Enter Personal expenses :", font=("Helvetica",10)).place(x=10,
y=570)
App1.entry14 = tk.Entry(App1)
App1.entry14.place(x=400, y=570)
App1.label15= tk.Label(App1, text="Enter Loan expenses :", font=("Helvetica",10)).place(x=10,
y=610)
App1.entry15 = tk.Entry(App1)
App1.entry15.place(x=400, y=610)
App1.label16= tk.Label(App1, text="Enter Insurance expenses :", font=("Helvetica",10)).place(x=10,
y=650)
App1.entry16 = tk.Entry(App1)
App1.entry16.place(x=400, y=650)
App1.label17= tk.Label(App1, text="Enter Educational expenses :",
font=("Helvetica",10)).place(x=10, y=690)
App1.entry17 = tk.Entry(App1)
App1.entry17.place(x=400, y=690)
App1.button = tk.Button(App1, text="Submit", command=submit)
App1.button.place(bordermode=OUTSIDE, x=250, y=730, width=200, height=50)
App1.mainloop()

def insertexpenseintodb(self,d,m):
    import datetime
    x = datetime.datetime.now()
    def graph(d,m):
        c1=[]
        a1=[]
        for i in d:
            a1.append(float(d[i]))
            c1.append(i)
        plotdata = pd.DataFrame(
            {"Amount in (Rs.)": a1},
            index=c1)
        # Plot a bar chart
        plotdata.plot(kind="bar",figsize=(10,10))
        plt.xticks(rotation=30, horizontalalignment="center")
        plt.xlabel("Expense Category")
        plt.ylabel("Amount Spent(in Rs.)")
        plt.title("Expense Analysis for the month"+m+str(x.year))
        plt.show()
    final_dict=dict()
    for i in d:
        if float(d[i])>0.00:
            final_dict[i]=d[i]
    total=0.0
    for i in final_dict:
        total+=float(final_dict[i])

```

```
print(total)
mydb1 = mysql.connector.connect(
    host="127.0.0.1",
    user="root",
    password="indira@123"
)
mycursor1 = mydb1.cursor()
mycursor1.execute("use expense_tracker_app;")
userid=mycursor1.execute("SELECT Userid FROM `admin` WHERE Userid=(SELECT MAX(Userid)
FROM `admin`);")
userid=mycursor1.fetchone()
a=userid[0]
sql1 = """INSERT INTO `expense_tracker_app`.`cumulativeexpense` (`Userid`, `UserMonth`,
`Currentyear`, `Totalexpanse`) VALUES (%s, %s, %s,%s);"""
values1 = (a,m,x.year,total)
mycursor1.execute(sql1, values1)
mydb1.commit()
for i in final_dict:
    mycursor1 = mydb1.cursor()
    mycursor1.execute("use expense_tracker_app;")
    mycursor1.execute("SELECT idCategory FROM `category` WHERE Category_name='%s';" % (i))
    userid = mycursor1.fetchone()
    print(userid)
    print(type(userid))
    sql1 = """INSERT INTO `expense_tracker_app`.`monthlyexpense` (`Userid`, `Month`,
`idCategory`, `Amount`) VALUES (%s, %s, %s,%s);"""
    values1 = (a, m, userid[0], float(final_dict[i]))
    mycursor1.execute(sql1, values1)
    mydb1.commit()
print(mycursor1.lastrowid)
print(mycursor1.rowcount)
graph(final_dict,m)
```

## REFERENCES

- [1] <https://towardsdatascience.com/introduction-to-data-visualization-in-python-89a54c97fbed>
- [2] <https://www.edureka.co/blog/tkinter-tutorial/#z2>
- [3] [https://matplotlib.org/3.1.0/gallery/statistics/barchart\\_demo.html](https://matplotlib.org/3.1.0/gallery/statistics/barchart_demo.html)
- [4] [https://matplotlib.org/3.1.0/api/\\_as\\_gen/matplotlib.pyplot.bar.html](https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.bar.html)
- [5] <https://www.mysqltutorial.org/mysql-limit.aspx>