1. **Executive Summary:**

This report outlines the development and implementation of a banking system project aimed at addressing inherent limitations in conventional banking systems. The project focused on leveraging database management with MySQL and Python programming to enhance customer management, transaction handling, and user experience within the banking domain.

The report begins with an introduction that delineates the context and challenges faced by existing banking systems, emphasizing limitations in data management, system capabilities, and user experience. A problem statement underscores the necessity for innovation and improvement in these areas, setting the stage for the project's objectives.

The project scope encompassed three primary facets: database management, system functionalities, and user interaction. Through meticulous planning and execution, the methodology involved conceptualizing, designing, and implementing a comprehensive system architecture.

Key steps in the project included the utilization of MySQL for database creation, ER diagram development using an online tool, Python coding for system functionalities, and integration with VS Code and Anaconda for development environment setup.

The project achieved its goals by facilitating user registration, personal information management, account creation, and financial transactions. However, encountered limitations include constraints in altering database structures and foreign key constraints.

Recommendations for future improvements highlight the need for enhanced database flexibility, UI/UX refinements, and addressing limitations within the current system. Additionally, the project's alignment with Sustainable Development Goals underscores its potential societal impact and contributions toward quality education and innovation.

The learning outcomes encompass technical skill enhancement, deeper insights into banking systems, and the fusion of technology for real-world applications. The report concludes by summarizing the project's achievements, emphasizing its role as a foundational learning experience with vast potential for future advancements.

1. **Introduction:**

The **Bank Management System** project represents a focused initiative aimed at designing a streamlined and efficient database-driven solution for managing essential banking operations. It addresses specific challenges encountered in traditional banking systems by implementing an organized database structure and core functionalities.

* 1. **Problem Statement:**

Inadequate data management and operational limitations in smaller-scale banking systems hinder efficient customer service, data accessibility, and essential functionalities. These limitations impact user experience and impede the seamless operation of day-to-day banking tasks. Addressing these challenges is crucial to streamline data organization, enhance customer service, and improve the overall banking experience within a smaller-scale environment.

1. **Objectives:**

* **Database Design and Structuring:** Create organized tables and relationships to efficiently store and manage customer information, account details, and transaction records.
* **Core Banking Operations:** Implement functionalities like account creation, deposit, withdrawal, and information updates to ensure essential banking services are readily accessible.
* **User-Friendly Interface:** Develop a straightforward user interface that facilitates easy navigation and interaction, ensuring a seamless banking experience.
* **CRUD Operations:** Performing Create Read Update and Delete operations in Database.

The Bank Management System project aims to address these specific challenges by providing an effective and accessible banking solution tailored to meet the requirements of a smaller-scale banking environment.

1. **Project’s Scope:**
   1. **Functional Scope:**

The project encompasses fundamental functional aspects, including user registration and login functionalities, comprehensive management of customer information, account details, and transaction records. It facilitates essential banking tasks such as deposits, withdrawals, and balance inquiries. Additionally, robust access control mechanisms ensure data security and user privacy.

* 1. **Technical Scope:**

From a technical standpoint, the project involves the development of a secure database structure to efficiently store and manage customer data. It includes the implementation of a user-friendly interface for seamless interactions and compatibility across various devices. Emphasis is placed on ensuring system security, data integrity, and efficient user experience.

* 1. **Limitations:**

However, the system's scope is limited in certain aspects. It might lack advanced financial functionalities like investment management or complex financial instruments, also there is no (GUI) graphical user interface. Additionally, potential constraints on scalability for a larger user base and the absence of advanced analytics or predictive features are notable limitations within this project's scope.

1. **Methodology:**
   1. **Project Planning and Conceptualization:**

The project commenced with meticulous planning and conceptualization, outlining the required functionalities and system requirements. A detailed layout of the project was initially crafted on paper to ensure a clear understanding of the database structure and application flow.

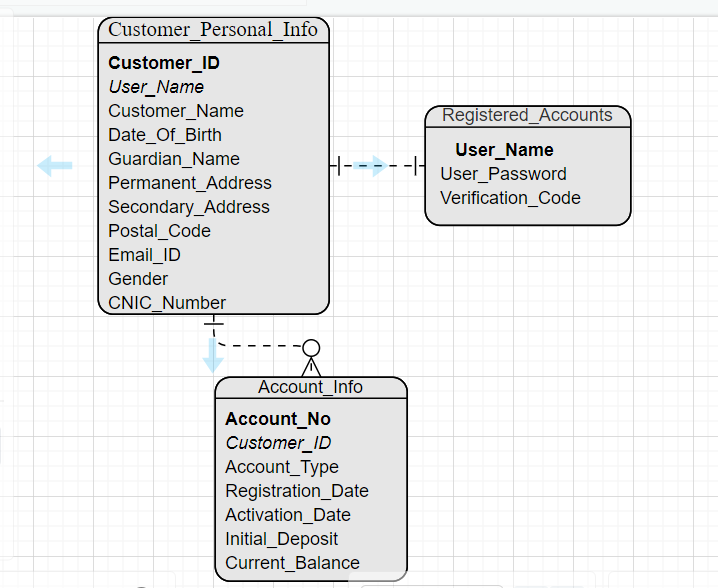
* 1. **Database Management using MySQL:**

MySQL, a robust and widely-used relational database management system, was employed for creating and managing the database. This phase involved defining database schemas, tables, establishing relationships between entities, and implementing constraints to ensure data integrity.

**SQL Queries:**

|  |
| --- |
| -- Create the Bank\_Management\_System database  CREATE DATABASE IF NOT EXISTS Bank\_Management\_System;  USE Bank\_Management\_System;  -- Table for Registered Accounts  CREATE TABLE Registered\_Accounts(  User\_Name CHAR(25) UNIQUE NOT NULL PRIMARY KEY,  User\_Password VARCHAR(25) NOT NULL,  Verification\_Code VARCHAR(10) NOT NULL  );  -- Table for Customer Personal Information  CREATE TABLE Customer\_Personal\_Info(  Customer\_ID INT AUTO\_INCREMENT PRIMARY KEY,  User\_Name VARCHAR(25) UNIQUE NOT NULL, -- Unique username for each customer  Customer\_Name VARCHAR(25) NOT NULL, -- Customer's name  Date\_Of\_Birth DATE, -- Date of birth (format: yyyy-mm-dd)  Guardian\_Name VARCHAR(25) NOT NULL, -- Guardian's name  Permanent\_Address VARCHAR(50) NOT NULL, -- Permanent address  Secondary\_Address VARCHAR(50), -- Secondary address (optional)  Postal\_Code VARCHAR(8), -- Postal code  Email\_ID VARCHAR(50) UNIQUE, -- Email address (format validation)  Gender CHAR(6), -- Gender  CNIC\_Number VARCHAR(15) UNIQUE, -- CNIC number (format: 12345-1234567-1)  CHECK (Customer\_Name REGEXP '^[A-Za-z]+$'), -- Validates Customer\_Name contains only alphabetic characters  CHECK (Guardian\_Name REGEXP '^[A-Za-z]+$'), -- Validates Guardian\_Name contains only alphabetic characters  CHECK (Email\_ID REGEXP '^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$'), -- Validates Email\_ID format  CHECK (CNIC\_Number REGEXP '^[0-9]{5}-[0-9]{7}-[0-9]$'), -- Validates CNIC\_Number format (12345-1234567-1)  CHECK (Date\_Of\_Birth REGEXP '^[0-9]{4}-[0-9]{2}-[0-9]{2}$'), -- Validates Date\_Of\_Birth format (yyyy-mm-dd)  FOREIGN KEY (User\_Name) REFERENCES Registered\_Accounts(User\_Name) -- Foreign key reference to Registered\_Accounts table  );  CREATE TABLE Account\_Info(  Account\_No INT AUTO\_INCREMENT PRIMARY KEY, -- Unique account number  Customer\_ID INT NOT NULL, -- Associated customer's ID  Account\_Type VARCHAR(10), -- Type of account  Registration\_Date DATE, -- Date of registration  Activation\_Date DATE, -- Date of activation  Initial\_Deposit BIGINT(10), -- Initial deposit amount  Current\_Balance BIGINT(20), -- Current balance  CHECK (Activation\_Date >= Registration\_Date), -- Validates Activation\_Date not earlier than Registration\_Date  FOREIGN KEY(Customer\_ID) REFERENCES Customer\_Personal\_Info(Customer\_ID) -- Foreign key reference to Customer\_Personal\_Info table  ); |

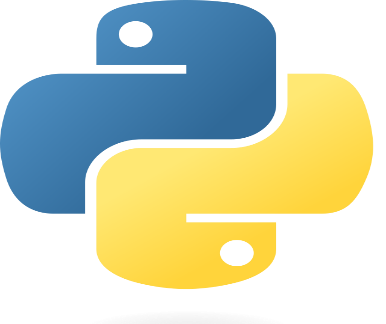
* These queries have different validation rules which are commented to be identified in the above query block.
* There exist a one-to-one relationship between Customer Info and Registered Accounts
* There exist also a one-to-many relation between Customer Info and Accounts Information
  1. Entity-Relationship (ER) Diagram Creation:

An online tool [Visual Paradigm](https://online.visual-paradigm.com/app/diagrams/#diagram:proj=0&type=ERDiagram&width=11&height=8.5&unit=inch) was utilized to craft the Entity-Relationship (ER) diagram. This visual representation offered a clear depiction of the database structure, encompassing entities, their attributes, and the relationships among them. 

(ER Diagram)

* 1. **Development Environment and Setup:**

Visual Studio Code (VS Code) was chosen as the Integrated Development Environment (IDE) for its robust features, aiding in code writing and debugging. Additionally, an environment using Anaconda with Python version 3.7 was set up to ensure compatibility and facilitate the installation of the MySQL Connector/Python library for database connectivity.



* 1. **Python Code Development:**

Python served as the primary programming language for creating the application logic to interact with the MySQL database. The Python code was meticulously crafted to manage user interactions, perform database operations, and ensure seamless data retrieval and manipulation.

In this code the library mysql connector was imported to establish a connection between python and MySQL Database, later on a class was created by the name of **Bank\_DB** in which there is a constructor **def \_\_init\_\_** where the connection of python with specific database is made .

|  |
| --- |
| import mysql.connector as connector  #Bank Database  class Bank\_DB:  def \_\_init\_\_(self):  self.conn=connector.connect(host='localhost',user='root',password='iamshahir1',port='3306',database='Bank\_Management\_System') |

**Methods of class Bank\_DB:**

* This method is used for creating account on portal.

|  |
| --- |
| def create\_portal\_log\_in(self,user\_name,password,verification\_code):    query = "INSERT INTO Registered\_Accounts(User\_Name, User\_Password, Verification\_Code) VALUES (%s, %s, %s)"  cur = self.conn.cursor()  cur.execute(query, (user\_name, password, verification\_code))  self.conn.commit()  print("Account created successfully")  return 1 |

* This method will be used to log into the portal

|  |
| --- |
| def log\_in(self, user\_name, password):  # Check if the username exists  query\_username = "SELECT User\_Name FROM Registered\_Accounts WHERE User\_Name = %s"  cur = self.conn.cursor()  cur.execute(query\_username, (user\_name,))  result\_username = cur.fetchone()  if result\_username:  # If the username exists, check the password  query\_password = "SELECT User\_Password FROM Registered\_Accounts WHERE User\_Name = %s"  cur.execute(query\_password, (user\_name,))  stored\_password = cur.fetchone()[0]  if stored\_password == password:  print("Login successful!")  return 1  else:  print("Incorrect password. Please try again.")  else:  print("Username does not exist.")  return 0 |

* This is use for deleting bank account

|  |
| --- |
| def delete\_account(self, user\_name):  Customer\_ID = self.get\_customer\_id(user\_name)  query = "DELETE FROM Account\_Info WHERE Customer\_ID = %s"  cur = self.conn.cursor()  cur.execute(query, (Customer\_ID,))  self.conn.commit()  print("Account deleted successfully.") |

* This is used for updation purpose if someone wants to update their info

|  |
| --- |
| def update\_personal\_info(self, user\_name):  query\_select = "SELECT \* FROM Customer\_Personal\_Info WHERE User\_name = %s"  query\_update = """  UPDATE Customer\_Personal\_Info  SET Customer\_Name = %s, Guardian\_Name = %s,  Secondary\_Address = %s, Postal\_Code = %s, Email\_ID = %s  WHERE User\_name = %s  """  cur = self.conn.cursor()  cur.execute(query\_select, (user\_name,))  existing\_data = cur.fetchone()  if existing\_data:  print("Enter new personal details:")  Customer\_Name = input("enter customer Name = ").capitalize()  Guardian\_Name = input("enter Guardian Name = ")  Secondary\_Address = input("enter Secondary Address = ")  Postal\_Code = input("enter Postal Code = ")  Email\_ID = input("enter Email Address = ")  # Execute the SQL query to update data in the database  cur.execute(query\_update, ( Customer\_Name, Guardian\_Name,  Secondary\_Address, Postal\_Code,  Email\_ID,user\_name))  self.conn.commit()  print("Personal details updated successfully.")  else:  print("User not found.") |

* This method method is used to insert details in customer info table

|  |
| --- |
| def insert\_personal\_details(self,User\_name):  Customer\_Name = input("enter customer Name = ").capitalize()  Date\_Of\_Birth = input("enter DOB {YYYY-MM-DD} = ")  Guardian\_Name = input("enter Guardian Name = ")  Permanent\_Address = input("enter your Permenant Address = ")  Secondary\_Address = input("enter Secondary Address = ")  Postal\_Code = input("enter Postal Code = ")  Email\_ID = input("enter Email Address = ")  Gender = input("enter your gender male/female = ")  CNIC\_Number = input("enter CNIC {XXXXX-XXXXXXX-X} = ")    query = "INSERT INTO Customer\_Personal\_Info(Customer\_Name,Date\_Of\_Birth,Guardian\_Name,Permanent\_Address,Secondary\_Address,Postal\_Code,Email\_ID,Gender,CNIC\_Number,User\_name)values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"  cur = self.conn.cursor()  cur.execute(query,(Customer\_Name,Date\_Of\_Birth,Guardian\_Name,Permanent\_Address,Secondary\_Address,Postal\_Code,Email\_ID,Gender,CNIC\_Number,User\_name))  self.conn.commit()  print("data entered successfully") |

* This method is used for printing the data

|  |
| --- |
| def print\_personal\_details(self, user\_name):  query = "SELECT \* FROM Customer\_Personal\_Info WHERE User\_name = %s"  cur = self.conn.cursor()  cur.execute(query, (user\_name,))    # Fetching the details of the user  user\_details = cur.fetchone()  if user\_details: # If the user is found  print("User Details:")  print(f"Customer ID: {user\_details[0]}")  print(f"Customer Name: {user\_details[1]}")  print(f"Date of Birth: {user\_details[2]}")  print(f"Guardian Name : {user\_details[3]}")  # Include other details similarly |

* This method is used to find customer id based on user name

|  |
| --- |
| def get\_customer\_id(self, user\_name):  query = "SELECT Customer\_ID FROM Customer\_Personal\_Info WHERE User\_name = %s"  cur = self.conn.cursor()  cur.execute(query, (user\_name,))  result = cur.fetchone()  if result:  return result[0] # Return the Customer\_ID if found  else:  print("Username does not exist.")  return None # Return None if username doesn't exist |

* This method is used for creating a bank account on portal

|  |
| --- |
| def create\_account(self,user\_name):  Customer\_ID = self.get\_customer\_id(user\_name)  if Customer\_ID:  print("enter account details ")  Account\_Type = input("enter account type = ").lower()  Registration\_Date = input("enter registeration date {YYYY-MM-DD} = ")  Activation\_Date = input("enter Activation date {YYYY-MM-DD} = ")  Initial\_Deposit = int(input("enter initial deposit = "))  Current\_Balance = Initial\_Deposit  # Execute the SQL query to insert data into the database  query = "insert into Account\_Info(Customer\_ID,Account\_Type,Registration\_Date,Activation\_Date,Initial\_Deposit,Current\_Balance)values(%s,%s,%s,%s,%s,%s,%s)"  cur = self.conn.cursor()  cur.execute(query,(Customer\_ID,Account\_Type,Registration\_Date,Activation\_Date,Initial\_Deposit,Current\_Balance))  self.conn.commit()  print("account details entered successfully") |

* This method will be used for printing the account details

|  |
| --- |
| def print\_account\_details(self,user\_name):  Customer\_ID = self.get\_customer\_id(user\_name)  query = "SELECT \* from Account\_Info WHERE Customer\_ID = %s"  cur = self.conn.cursor()  cur.execute(query,(Customer\_ID,))  user\_details = cur.fetchone()  if user\_details:  print(f"account number = {user\_details[0]}")  print(f"account type = {user\_details[2]} ")  print(f"current balance = {user\_details[6]}")  else:  print("account does not exist") |

* This method will insert the deposit the amount in data base and also update it

|  |
| --- |
| def deposit\_cash(self, user\_name):  customer\_id = self.get\_customer\_id(user\_name)  if customer\_id:  deposit\_amount = int(input("Enter the amount to deposit: "))    # Assuming the account number is already known or fetched  # Update the Current\_Balance in Account\_Info table  query = "UPDATE Account\_Info SET Current\_Balance = Current\_Balance + %s WHERE Customer\_ID = %s"  cur = self.conn.cursor()  cur.execute(query, (deposit\_amount,customer\_id))  self.conn.commit()  print(f"Deposit of {deposit\_amount} made successfully.")  else:  print("Unable to find the customer ID for the given username.") |

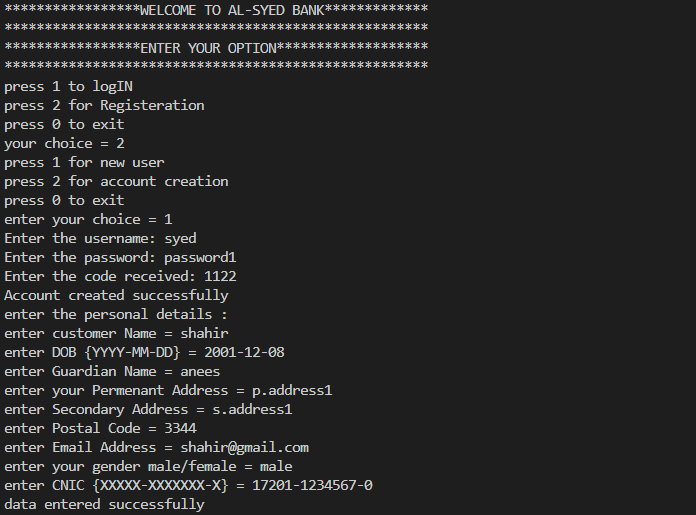
* This method will be used to withdraw cash if available and also update the account info with new amount

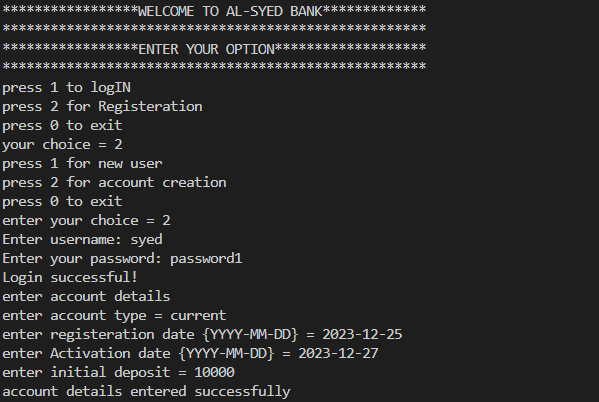
|  |
| --- |
| def withdraw\_cash(self, user\_name):  customer\_id = self.get\_customer\_id(user\_name)  if customer\_id:  query = "SELECT Current\_Balance FROM Account\_Info WHERE Customer\_ID = %s"  cur = self.conn.cursor()  cur.execute(query, (customer\_id,))  check\_current = cur.fetchone()  if check\_current:  current\_balance = check\_current[0]  amount = int(input("Enter amount to withdraw: "))  if amount <= current\_balance:  query = "UPDATE Account\_Info SET Current\_Balance = Current\_Balance - %s WHERE Customer\_ID = %s"  cur.execute(query, (amount, customer\_id))  self.conn.commit()  print(f"Withdrawal of {amount} made successfully.")  else:  print("Insufficient funds.")  else:  print("Failed to retrieve current balance.")  else:  print("Customer ID not found for the given username.") |

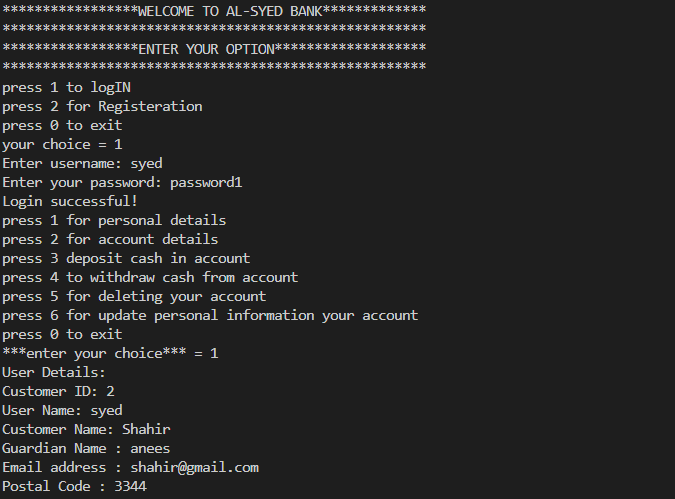
Main function, this code consists of while loop for menu driven interface, it gives the user flexibility of selecting different options from available menu , this code also create and instance of the class created above so that we will be able to use the class and its methods

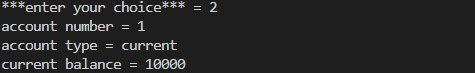
|  |
| --- |
| bank1 = Bank\_DB()  switch = True  while(switch):  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*WELCOME TO AL-SYED BANK\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ENTER YOUR OPTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("press 1 to logIN ")  print("press 2 for Registeration ")  print("press 0 to exit ")  choice = int(input("your choice = "))  if(choice == 1):  #login function  user\_name = input("Enter username: ")  password = input("Enter your password: ")  if(bank1.log\_in(user\_name,password) == 1):  login\_switch = True  while(login\_switch):  print("press 1 for personal details ")  print("press 2 for account details ")  print("press 3 deposit cash in account ")  print("press 4 to withdraw cash from account ")  print("press 5 for deleting your account")  print("press 6 for update personal information your account")  print("press 0 to exit ")  login\_choice = int(input("\*\*\*enter your choice\*\*\* = "))  if(login\_choice == 1):  bank1.print\_personal\_details(user\_name)  elif(login\_choice == 2):  bank1.print\_account\_details(user\_name)  elif(login\_choice == 3):  #cash deposit  bank1.deposit\_cash(user\_name)  elif(login\_choice == 4):  #cash with draw  bank1.withdraw\_cash(user\_name)  elif(login\_choice==5):  bank1.delete\_account(user\_name)  elif(login\_choice==6):  bank1.update\_personal\_info(user\_name)  elif(login\_choice == 0):  login\_switch = False  else:  print("invalid input")  bank1.print\_personal\_details(user\_name)  else:  print("Access Denied")  elif(choice == 2):  #registeration function  registeration\_switch = True  while(registeration\_switch):  print("press 1 for new user")  print("press 2 for account creation")  print("press 0 to exit ")  registeration\_choice = int(input("enter your choice = "))  if(registeration\_choice == 1):  user\_name = input("Enter the username: ")  password = input("Enter the password: ")  verification\_code = input("Enter the code received: ")  bank1.create\_portal\_log\_in(user\_name,password,verification\_code)  print("enter the personal details :")  bank1.insert\_personal\_details(user\_name)  elif(registeration\_choice == 2):  user\_name = input("Enter username: ")  password = input("Enter your password: ")  if(bank1.log\_in(user\_name,password) == 1):  bank1.create\_account(user\_name)  else:  print("account do not exist")  elif(registeration\_choice == 0):  registeration\_switch = False  else:  print("invalid input!!!")  elif(choice == 0):  switch = False  else:  print("invalid input")  print("\*\*\*\*You exited From Bank System\*\*\*\*") |

1. **Program Testing and working:**

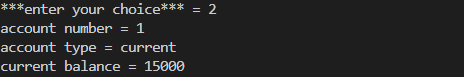










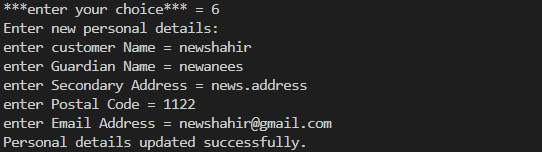


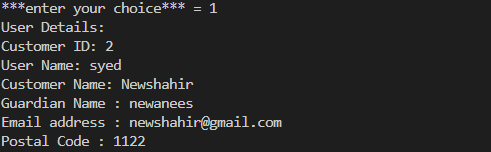






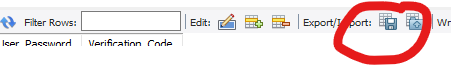




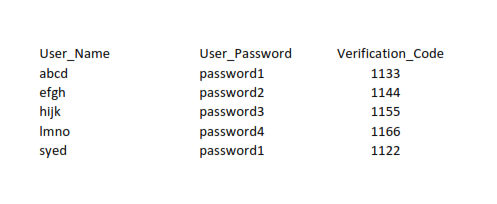


All the above screenshots shows all the CRUD operations of the database as well as gives the idea how the project works and interface looks.

* 1. **Report Generation:** we can export data in comma separated file from MySQL in order to use as report purpose



**Output:**



1. **Learning Outcomes:**
   1. **Technical Proficiency:** Students gain hands-on experience in database design, SQL queries, Python programming, and integrating these technologies for a banking system. This project cultivates technical skills crucial for database management and application development.
   2. **Problem-Solving:** Addressing challenges like data validation, user authentication, and transaction processing hones problem-solving abilities. Debugging errors and optimizing code enhance critical thinking and troubleshooting skills.
   3. **Understanding Banking Systems:** Developing a banking system provides insights into real-world banking processes, including customer management, account handling, and security protocols.
   4. **Teamwork and Collaboration:** If done in a group setting, this project fosters teamwork, collaboration, and division of tasks to achieve common project goals.
2. **SDG Alignment:**
   1. **Quality Education (SDG 4):** This project promotes quality education by providing practical learning experiences in technology, database management, and application development, empowering students with relevant skills for the future workforce.
   2. **Industry, Innovation, and Infrastructure (SDG 9):** Creating a functional banking system contributes to innovation in technology and infrastructure, promoting economic growth and development.
   3. **Sustainable Cities and Communities (SDG 11):** A well-designed banking system can positively impact communities by providing accessible financial services, contributing to economic stability and community development.
   4. **Partnerships for the Goals (SDG 17):** Collaborations between educational institutions and industry partners for project guidance or mentorship foster partnerships crucial for achieving sustainable development objectives.
3. **Conclusion:**

In conclusion, the development of the banking system project has been a remarkable journey that encapsulated various facets of database management, Python programming, and real-world application design. Through meticulous planning, iterative development, and problem-solving, this project addressed fundamental challenges faced in traditional banking systems. It provided a hands-on learning experience that not only enhanced technical skills but also imparted invaluable insights into the complexities of customer management, transaction handling, and data security within the banking sector.

This project highlighted the importance of adaptability and innovation in creating functional solutions. The integration of SQL and Python to manage the database and drive system functionalities showcased the power of technology in streamlining banking operations. While the project achieved its primary objectives, it also unveiled the vast potential for future enhancements, emphasizing the continuous evolution and refinement required in technological applications.

Moreover, the alignment of this project's outcomes with Sustainable Development Goals, particularly in terms of fostering quality education, innovation, and potential societal impact, underscores its relevance in addressing contemporary challenges.

As a learning endeavor, this project not only honed technical skills but also instilled a deeper understanding of the intricacies within banking systems, laying a foundation for future endeavors in technology-driven solutions. It's a testament to the capabilities cultivated through practical application and problem-solving in the realm of database management and software development.

In essence, the banking system project served as an exemplary platform for learning, innovation, and skill development, paving the way for a future where technology and thoughtful design converge to redefine conventional systems.