DBMS PROJECT REPORT

Myers Briggs Personality Test Indicator



**Arnav Jalan (2210110192)  
Arnav Aditya (2210110189)  
Rachit Sen (2210110489)**

20 - 04 - 2024

B.Tech Cse’ 26

**ABSTRACT**The Myers Briggs Personality Test, based on Carl Jung's theory of psychological types, provides insight into an individual's personality preferences and tendencies. This project focuses on the implementation of a Myers Briggs Personality Test system using MySQL, a widely-used relational database management system. The project aims to create a robust database schema capable of storing test responses efficiently while ensuring data integrity and security. Additionally, it involves developing SQL queries and stored procedures to facilitate the administration of the personality test, scoring of responses, and generation of personalized personality profiles. Through this implementation, the project seeks to provide a scalable and user-friendly platform for conducting and analyzing Myers Briggs Personality Tests, enabling individuals and organizations to gain valuable insights into personality dynamics.

**OUR TECH STACK:**

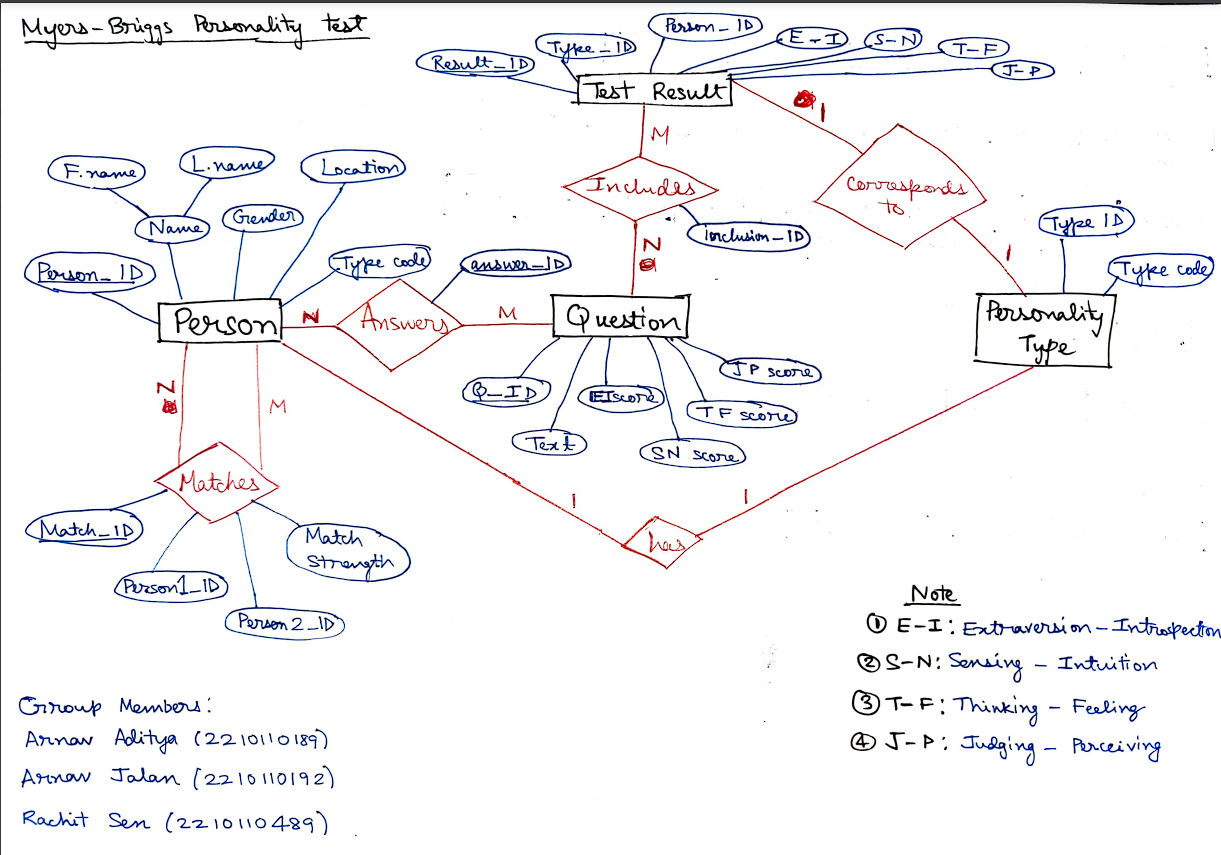
1. Python - MySql connector
2. Python QT
3. MySql

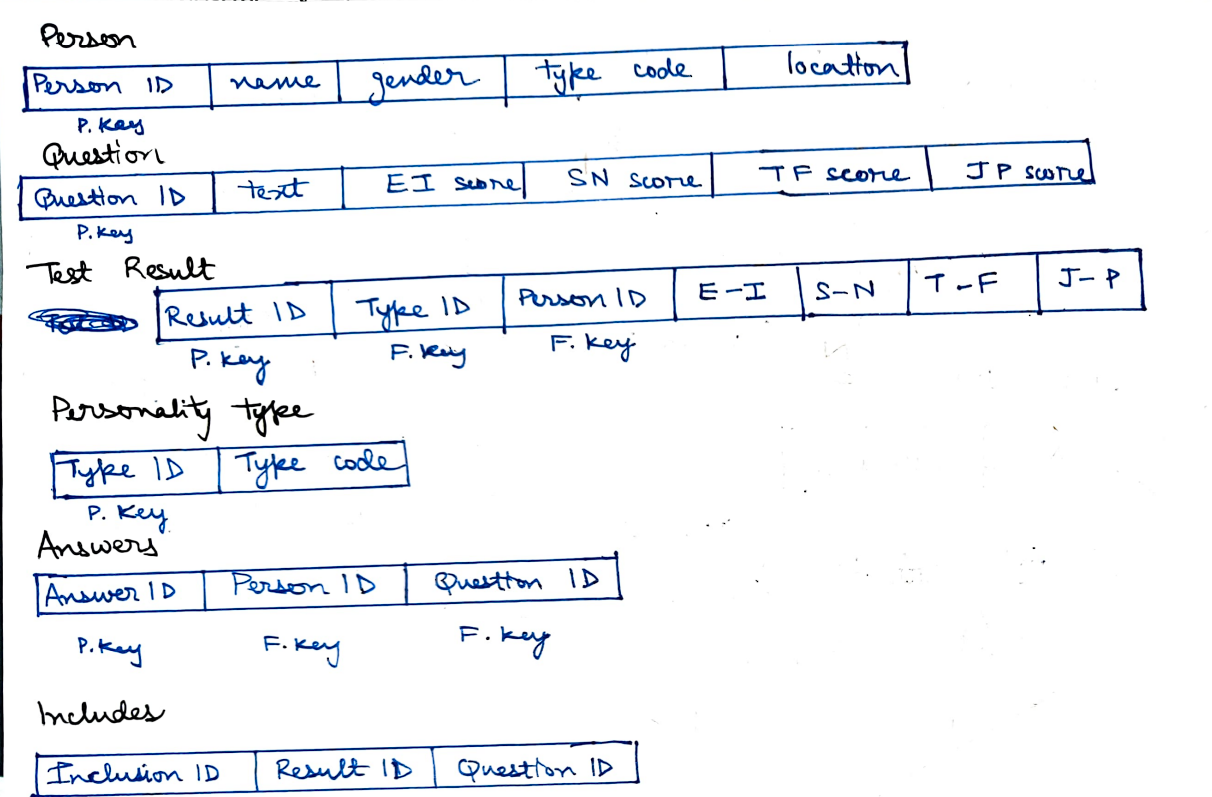
**OVERALL BRIEF**

**OBJECTIVES :**

* Database Implementation: Designing and implementing a MySQL database schema to efficiently store Myers Briggs Personality Test data, including user responses and associated metadata.
* Frontend Design: Creating a user friendly test taking experience where users can give the test smoothly.
* Test Administration: Developing functionalities to administer the Myers Briggs Personality Test, allowing users to input their responses and ensuring a user-friendly interface for test completion.
* Scoring Mechanism: Implementing algorithms to score test responses accurately, following the guidelines of the Myers Briggs Type Indicator (MBTI), and assigning personality types based on the calculated scores.
* Profile Generation: Generating personalized personality profiles for users based on their test results, providing insights into their dominant personality traits, preferences, and tendencies.

**FEATURES:**The test features a list of questions which have been classified based on the various personality types they encapsulate.  
There are 4 types :   
There will be 4 option provided to the test taker which are as follows :   
Based on the options the user picks, their value for the particular personality type will fluctuate and in the end based on the personality which has the most points will be their particular dominant personality.  
To make it more interesting, unique and suited to a college setting we have created an algorithm which matches similar personality tests takers to each other which would help make friends or find like minded people for those who struggle to be social.

**E-R MODEL :   
**

**RELATIONAL MODEL :   
**

**RELATED WORK:**There are many Myers Briggs Personality Tests which exist online but none of them use MySql to store the data which in itself created a problem which needed to be solved. We used the concepts and teaching we learnt in this course to apply this problem and integrate the test with MySql.  
MySql offers features like creating tables, stored procedures, triggers and views which help in making and storing data related to any web application.  
  
**OUR SQL SCRIPT:**

* **Database Creation**:
  + CREATE DATABASE **final\_proj**: Creates a new database named **final\_proj**.
* **Table Creation**:
  + CREATE TABLE **personality\_questions**: Defines a table to store personality questions with columns for **questionid**, **questiondesc**, and **personal\_type**.
* CREATE TABLE **questionoptions**:
  + Defines a table to store options for each personality question with columns for **questionoptionid**, **optiondescription**, optionmarks, and **questionid**.
* CREATE TABLE **persona**:
  + Defines a table to store responses to personality questions with columns for **person qaid**, **person\_name**, **question optionid**, **questionid**, age, and phone\_no.
* **Foreign Key Constraints**:
  + CONSTRAINT **question\_fk**: Defines a foreign key constraint on the **questionid** column of the **personqa** table referencing the **questionid** column of the personality\_questions table.
* **CONSTRAINT questionoptionid\_fk**:
  + Defines a foreign key constraint on the **questionoptionid** column of the **personqa** table referencing the **questionoptionid** column of the **questionoptions** table.
* **Views Creation**:
  + CREATE VIEW **vw\_getperson**: Defines a view to retrieve distinct combinations of **person\_name** and **phone\_no** from the **personqa** table.
* CREATE VIEW **vw\_getquestionoption**:
  + Defines a view to retrieve concatenated question descriptions along with their options from the **personality\_questions** and **questionoptions** tables.
* **Data Insertion**:
  + INSERT INTO **personality\_questions**: Inserts data into the **personality\_questions** table, providing values for **questionid**, **questiondesc**, and **personal\_type**.
* INSERT INTO **questionoptions**:
  + Inserts data into the **questionoptions** table, providing values for **questionoptionid**, **optiondescription**, optionmarks, and **questionid**.
* **NORMALIZATION**:  
  Since there is no redundancy in our tables, our tables are in **BCNF - Boyce Codd Normal Form**.
* **CREATE ALGORITHM**:  
  Instead of using triggers and procedures we have used a feature of views which is “**CREATE ALGORITHM**” and integrated it with our python script to ensure the logic and algorithm is working as intended.

The first and second table are the base tables which contain the questions and the options that we can choose from for the questions. The values in these tables have been populated by a simple insert statement given in the SQL script.

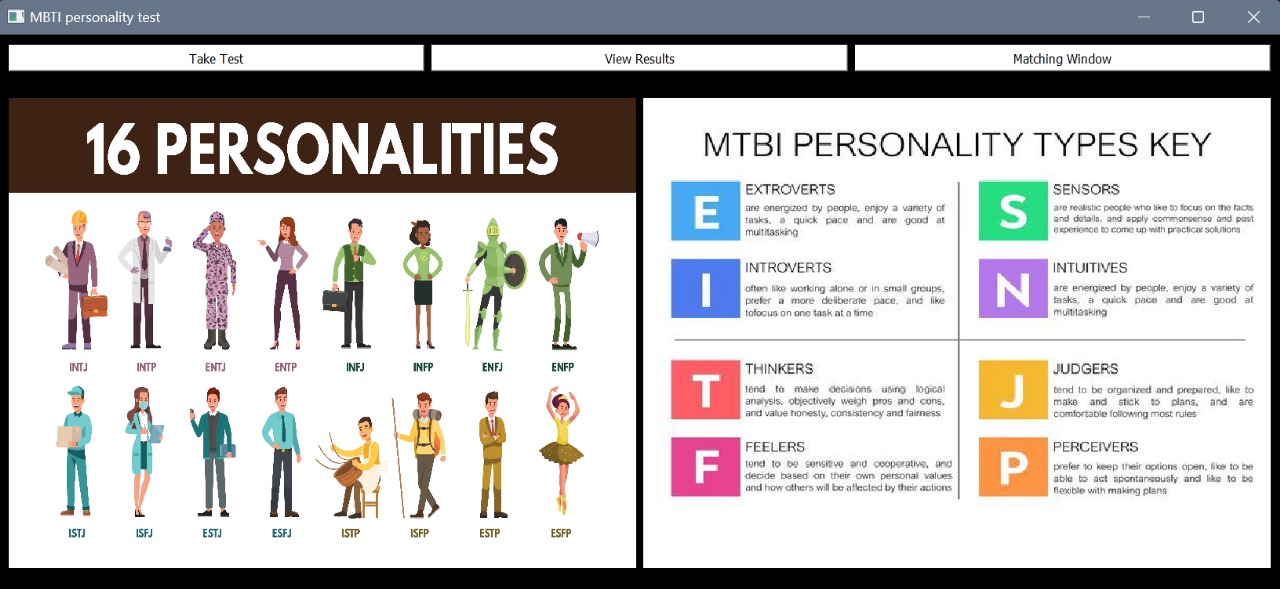
The Third Table, persona is our transaction table, in which we populate the table by taking responses from the user through the front end. The Queries for populating have been defined in the Python Script, which has been connected with MySQL.

Two Different views have also been created in the script, which helps in the Python program, as the first view (vw\_getperson) helps in getting the records in the database in the dropdown menu which comes after clicking the “View Result” button. While the second view (vw\_getquestionoption) helps in keeping a track of the options selected in the “Take Test” window.

**OUR PYTHON SCRIPT:**We have used Python and PyQT for frontend development. It offers a subtle and clean user test taking experience.

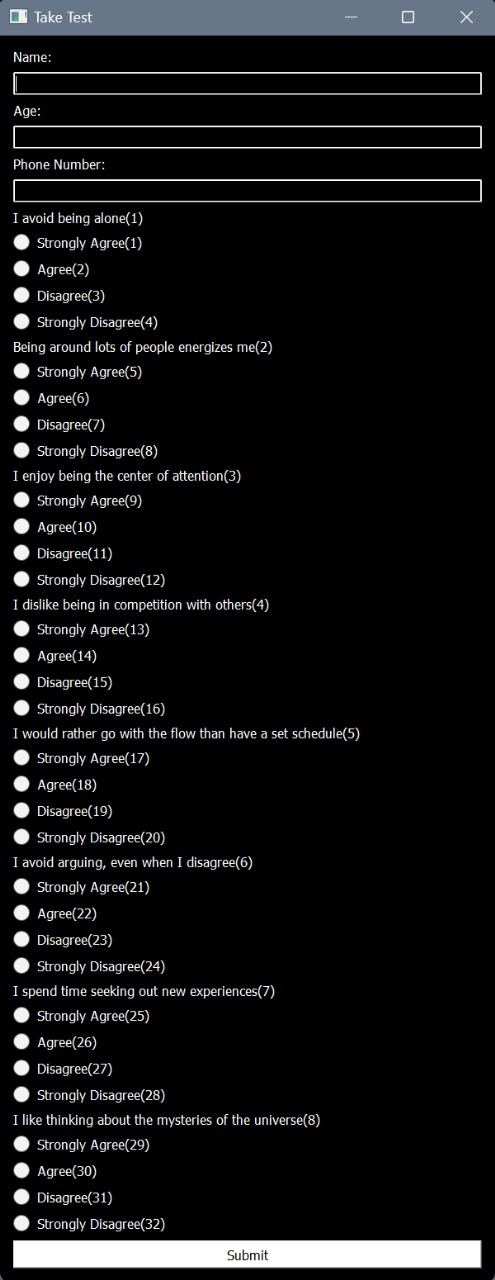
* Database Interaction:
  + The application interacts with a MySQL database (final\_proj) to store questions, options, and user responses
* Main Window:
  + The main window (PersonalityTestApp) displays three buttons: "Take Test", "View Results", and "Matching Window".
  + Clicking on the "Take Test" button opens a window (TakeTestWindow) where users can input their name, age, phone number, and take the personality test.
  + Clicking on the "View Results" button opens a window (ViewResultsWindow) where users can view their test results.
  + Clicking on the "Matching Window" button opens a window (MatchingWindow) where users can select a personality trait and view other users who match that trait.
* Take Test Window (TakeTestWindow):
  + Users input their name, age, and phone number.
  + The window dynamically fetches questions from the database (vw\_getquestionoption) and presents them as labels with corresponding radio buttons for options.
  + Upon submitting the test, the user's responses are stored in the database.
* View Results Window (ViewResultsWindow):
  + Users can select a person from a dropdown menu to view their test results.
  + Test results are fetched from the database and displayed in a messagebox.
* Matching Window (MatchingWindow):
  + Users can select a personality trait from a dropdown menu.
  + The window fetches users from the database who match the selected trait and displays them in a messagebox.
* Database Querying:
  + Various SQL queries are executed to fetch questions, store user responses, fetch test results, and find matching users.
* Error Handling:
  + Error handling is implemented for database connections and queries. If an error occurs, a messagebox displays the error message.
* GUI Styling:
  + GUI elements are styled using CSS-like syntax to set background color, text color, font size, etc.

**WORKING DEMO:**

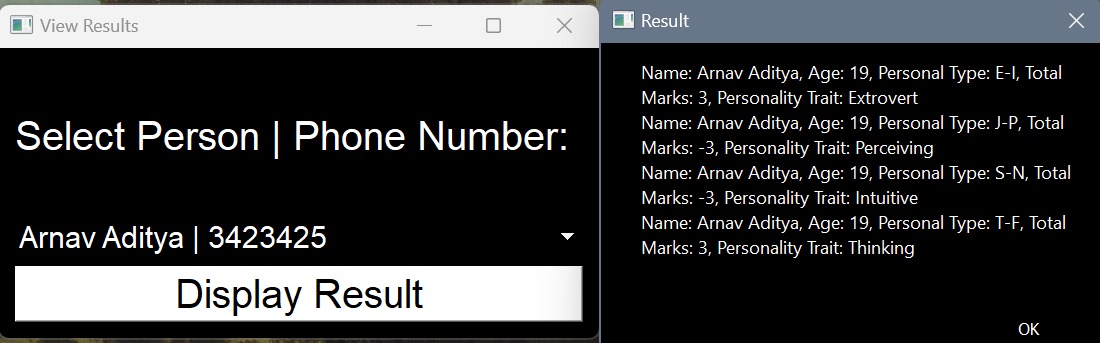
**LANDING PAGE   
**

Guides the user by explaining various personality types so that they can refer to this after viewing their result.

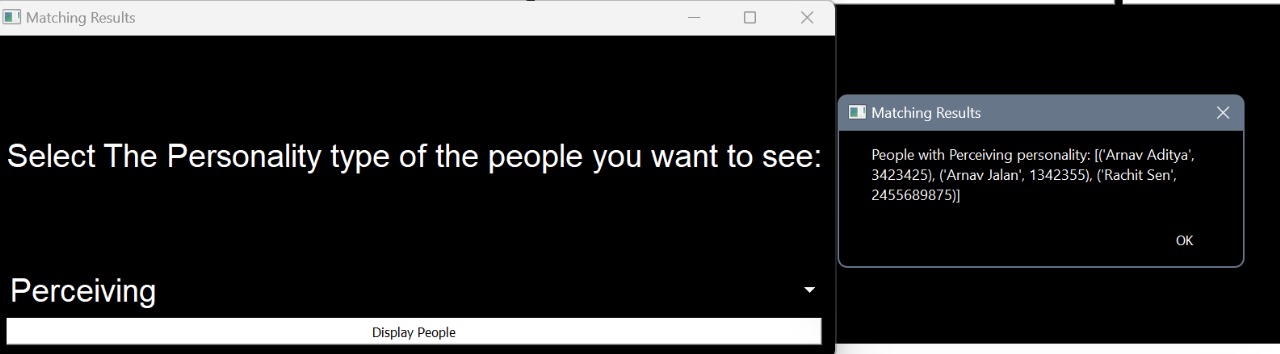
**TEST TAKING EXPERIENCE:**

****

**RESULT VIEWING:**

****

**FINDING MATCHING PEOPLE:**

****

**ACKNOWLEDGEMENT:**We would like to extend our gratitude to our professor Dr. Sonia Khetarpaul for teaching us this course in formidable fashion without which this project would never have reached its completion. We would also like to thank our Lab TA’s for constantly being there to help us with our. We faced a lot of challenges making this project but it was a fun and learning experience for us. We declare that all the information mentioned in the report and the project is honest and authentic to the best of our knowledge.  
  
**LIMITATIONS:**We faced a lot of problems and challenges during this project and this is not perfect as it has some limitations:  
1. The frontend is created in PyQT which is not the most beautiful looking front end out there but it was primarily due to none of us being well versed with HTML/CSS and JS frameworks.  
2. Due to time constraints our matching algorithm is not how it should ideally perform.  
3. Since the tables are only 3 in number most of the tables are already normalized and do not require normalization, hence this project does not show our knowledge of normalization (which is good).  
  
**REFERENCES -**- Stack Overflow

- Github

-<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9909077>

- <https://dev.mysql.com/doc/refman/8.0/en/>

- <https://doc.qt.io/qtforpython-6/>