# Value Measurement Database Application

## Overview

The **Value Measurement Database Application** is a Python-based graphical user interface (GUI) that allows users to interact with a MySQL database for tracking transformation initiatives.

Built using Tkinter, it provides a tabbed interface for managing multiple database tables with CRUD (Create, Read, Update, Delete) functionality.

## Features

* **UI Generation**: Automatically creates input fields and displays data for multiple tables.
* **CRUD Operations**: Supports inserting, updating, deleting, and viewing records.
* **Scrollable Table View**: Displays data using a Treeview with vertical scrolling.
* **Automatic NULL Handling**: Ensures empty fields are correctly handled in the database.
* **Database Connectivity**: Interfaces with a MySQL database using mysql.connector and configuration file to store credentials.
* **Error Handling**: Use of try-except blocks that allows errors to be raised and presented in the GUI.
* **Query Storage, Execution and Download**: Storage of complex queries via table with execution in the GUI.

## Installation

### Prerequisites

* Python 3.x
* MySQL Server
* Required Python Packages: {bash} configparser datetime logging mysql-connector os re tkcalendar tkinter uuid ### Database Setup

1. Create a MySQL database named value.
2. Define the required tables:
   * initiative
   * event
   * metric
   * plan
   * event\_plan
   * global\_metric\_value
   * plan\_metric\_value
   * user\_query
3. Update the database connection details in config.ini.

## Usage

### Running the Application

Execute the following command:

python app.py

### GUI Functionality

1. **Navigate Tabs**: Each tab represents a database table.
2. **Add a Record**: Fill in the input fields and click **Add**.
3. **Update a Record**: Select a row, modify fields, and click **Update**.
4. **Delete a Record**: Select a row and click **Delete**.
5. **Refresh Data**: Click **Refresh** to reload data from the database.
6. **Query Execution**: Store, run and download results from complex SQL queries.

## File Structure

* config.ini - Stores MySQL login credentials and connection details.
* app.py - The main GUI application.
* database.py - Database interaction layer.
* downloader.py - Utility class; handles the downloading of query results to a csv file.
* messenger.py - Utility class; displays messages and logging output in application.
* widget\_binder.py - Utility class; manages and synchronizes values across multiple Tkinter widgets.

## Config.ini

This file stores MySQL login credentials and connection details in a structured format under a section named [value]. Each line represents a key-value pair used to establish a connection to a MySQL database:

host = localhost: Specifies that the MySQL server is running on the local machine.

user = root: The username used to connect to the MySQL server.

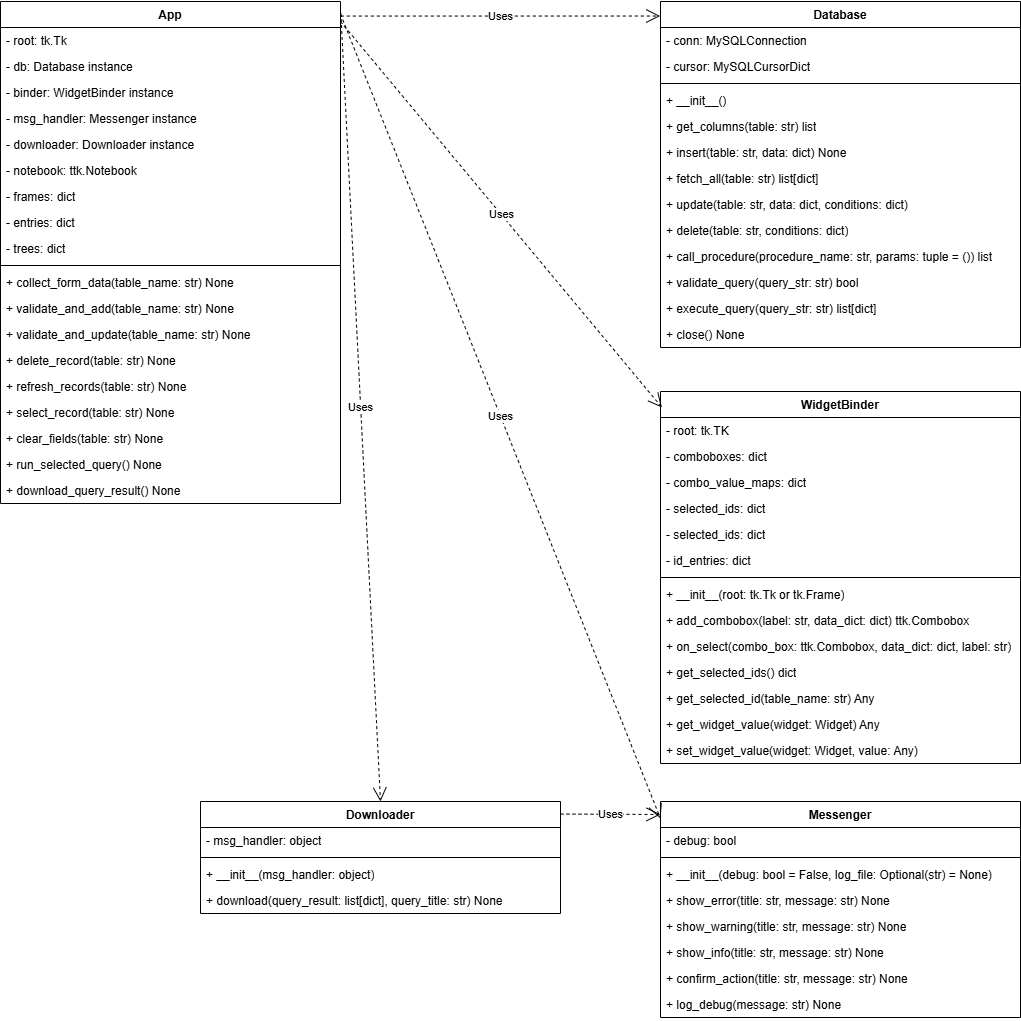
password = XXXXXXXX: Placeholder for the actual password associated with the user.

database = value: The name of the database to connect to.

This file can be securely read by the database.py codes (e.g., using configparser) to centralize and separate configuration from code.

## Class Structure Diagram

The following diagram provides a clear visual map of the system’s structure: its classes, attributes, methods, and relationships.



Class Structure Diagram

## Error Handling

Exception messages are raised from database.py and other classes, and displayed in the application using the functionality of messenger.py.

To accomplish this, database calls and other functions are wrapped in try-except blocks and any error messages are displayed via message boxes.

Here’s a comprehensive README section for SQL integration, detailing the database connection, functions, and usage for value\_db.py. This documentation will help developers understand how to work with the MySQL database in your project.

## SQL Integration Guidelines

### Overview

This project includes database.py, which provides a Python-based interface to a MySQL database. The database handles value measurement operations, allowing users to insert, fetch, update, and delete records efficiently.

### Database Connection

The Database class in value\_db.py establishes a connection to a MySQL database using the following credentials (configured in the config.ini file):

* **Host:** localhost
* **User:** root
* **Password:** XXXXXXXX
* **Database:** value

To modify the connection settings, update the configuration file.

### Methods Overview

The Database class provides several methods to interact with MySQL tables:

#### 1. **get\_columns(table)**

* Retrieves column names for a given table.
* **Usage:** db.get\_columns("table\_name")
* **Returns:** A list of column names.

#### 2. **insert(table, data)**

* Inserts a new record into a specified table, generating primary keys if not provided.
* **Usage:** python data = {"column1": "value1", "column2": "value2"} db.insert("table\_name", data)
* **Returns:** None.

#### 3. **fetch\_all(table)**

* Fetches all records from a given table.
* **Usage:** records = db.fetch\_all("table\_name")
* **Returns:** A list of dictionaries representing rows.

#### 4. **update(table, data, conditions)**

* Updates existing records in a table based on conditions.
* **Usage:** python data = {"column1": "new\_value"} conditions = {"id": 1} db.update("table\_name", data, conditions)
* **Returns:** None.

#### 5. **delete(table, conditions)**

* Deletes records from a table based on conditions.
* **Usage:** python conditions = {"id": 1} db.delete("table\_name", conditions)
* **Returns:** None.

#### 6. **call\_procedure(procedure\_name, params=())**

* Calls a stored procedure.
* **Usage:** db.call\_procedure("procedure\_name", (param1, param2))
* **Returns:** Procedure output (if applicable).

#### 7. **close()**

* Closes the database connection.
* **Usage:** db.close()

## Error Handling

All SQL queries are wrapped in try-except blocks to catch mysql.connector.Error. If an error occurs, an exception is raised with a relevant message.

## Security Considerations

* **Do not hardcode credentials in production.** Use environment variables or a configuration file.
* **Validate user input and use parameterized queries** to prevent SQL injection.
* **Implement role-based access control (RBAC)** in MySQL for secure operations.

## Example Usage

Here’s how you can use the database class in a script:

```{python} from database import Database

db = Database()

# Insert a record

db.insert(“users”, {“name”: “Alice”, “email”: “[alice@example.com](mailto:alice@example.com)”})

# Fetch records

records = db.fetch\_all(“users”)

# Update a record

db.update(“users”, {“email”: “[alice@newdomain.com](mailto:alice@newdomain.com)”}, {“name”: “Alice”})

# Delete a record

db.delete(“users”, {“name”: “Alice”})

# Close connection

db.close() ```

## Future Enhancements

* User authentication and access control.
* Advanced filtering and search functionality.

## License

This project is intended for internal use. Contact the author for usage permissions.

## Author

Developed by Donnie Minnick to satisfy the requirements for Deliverable 5 in the CS727 Relational Database Implementation and Applications course in the IIT MDS program.