🧩 Multi-Database Migration and Management Tool

Overview

This project is a **modular, command-line-based Python application** that facilitates **schema migration and data transfer** between MySQL, PostgreSQL, and Oracle databases. Designed for developers and DBAs, the tool automates **schema fetching**, **table creation**, and **data movement** from one database to another — all while ensuring **robust error handling**, **connection abstraction**, and **dynamic data type mapping**. It automates the process of:

- Extracting schema information
- Creating tables
- Inserting data

-It can serve as a lightweight ETL utility or database structure replication assistant.

🛠️ Features

- **Cross-Database Compatibility:** Supports MySQL, PostgreSQL, and Oracle databases.
- **Schema Extraction:** Automatically retrieves table schemas from source databases.
- **Dynamic Table Creation:** Generates CREATE TABLE statements compatible with the destination database.
- **Data Insertion**: Efficiently inserts data into destination tables, handling data type mappings.
- **Data Selection:** Retrieves data from source tables for inspection or transfer.
- **Logging and Error Handling:** Comprehensive logging for operations and robust error handling with transaction rollbacks.
- **Configure Database Connections:**

  Navigate to the configs/ directory and configure the connection settings for each supported database (mysql_config.py, oracle_config.py, postgres_confi).

- **Utility Modules** for extensibility and readability

Folder-wise Module Structure

->configs/ – DB Configuration

Contains database connection dictionaries for modular and reusable access.

- **mysql_config.py**
- **postgres_config.py**
- **oracl_config.py**

Manages DB credentials and DSN configurations cleanly, enabling secure and centralized updates.

->database/ – DB Core Functionality

available_tables.py

- Lists tables available in the selected source database.
- Used to prompt user for table selection.

schema.py

- Fetches column names and data types from a table across supported databases.
- Handles compatibility variations across MySQL (`DESCRIBE`), Oracle, and PostgreSQL (`information_schema`).

datatype_mapper.py

- Converts data types from source to destination DB.
- Handles DB-specific quirks like `VARCHAR2`, `TEXT`, `NUMBER`, etc.

connection.py

- Centralized logic for establishing and closing database connections.
- Returns the required cursor and connection objects for DB operations.

operations.py

- Executes table creation queries on target databases.
- Dynamically constructs column definitions using `datatype_mapper`.

data_transfer.py

- Fetches data from source.
- Inserts into destination with placeholder handling based on DB type.
- Verifies by selecting from the target table post-transfer.

->utils/ – Supporting Utilities

input_utils.py

- Handles user prompts like:
  - Selecting DB types
  - Choosing actions
  - Selecting tables
- Ensures valid input and prompts until correct values are given.

logger.py

- Maintains logs of:
  - Table creation
  - Data transfer status
  - Errors and exceptions
- Useful for auditing and debugging.

->main.py

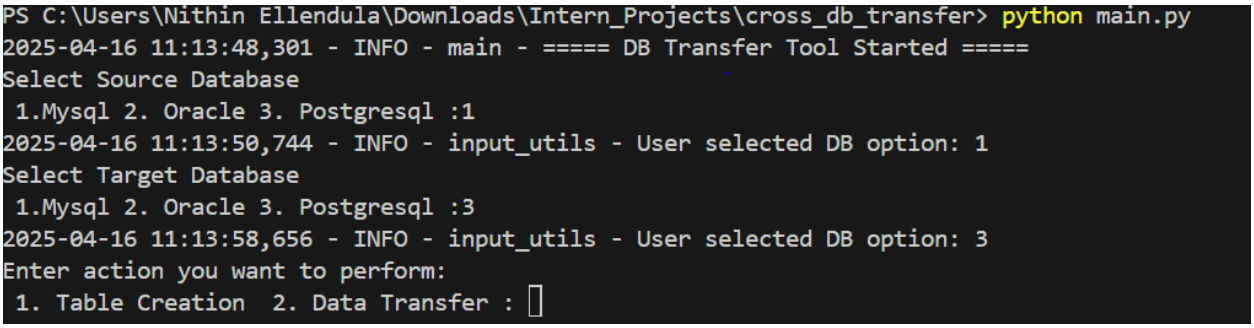
- Entry point of the application.
- Interacts with user for:
  - DB selection
  - Action execution (1: Table Creation, 2: Data Transfer)
- Routes control to corresponding modules.
- Performs final execution of schema transfer and data population

🎯 1. Supported Actions

The system performs two primary actions:

- **Creation** - Extracts the schema from the source DB and creates an equivalent table in the destination DB.
- **Data Transfer** - Fetches data from the source table and inserts it into the matching table in the destination DB.

Output:



Functional Highlights

1. Input Management

The tool uses a **centralized input system** to collect and validate user selections like **database type**, **table name**, and **desired actions** (e.g., table creation or data transfer). This process is abstracted in the `utils/input_utils.py` module to promote **clean code** and **reusability**.

- Dynamic input capture using `input_utils`
- Prevents invalid selections with friendly re-prompts

Output:

```
PS C:\Users\Nithin Ellendula\Downloads\Intern_Projects\cross_db_transfer> python main.py
2025-04-16 11:54:58,624 - INFO - main - ===== DB Transfer Tool Started =====
Select Source Database
 1.Mysql 2. Oracle 3. Postgresql :1
2025-04-16 11:55:00,534 - INFO - input_utils - User selected DB option: 1
Select Target Database
 1.Mysql 2. Oracle 3. Postgresql :2
2025-04-16 11:55:01,596 - INFO - input_utils - User selected DB option: 2
Enter action you want to perform:
 1. Table Creation  2. Data Transfer : 1
2025-04-16 11:55:04,432 - INFO - input_utils - User selected action: 1
2025-04-16 11:55:04,432 - INFO - connection - Connecting to MySQL...
2025-04-16 11:55:04,634 - INFO - connection - MySQL connection established.
2025-04-16 11:55:04,635 - INFO - Tables - Fetching available tables from MySQL...
2025-04-16 11:55:04,639 - DEBUG - Tables - Fetched 3 tables from MySQL.
['employee', 'student1', 'user_table']
Enter the table name: student1
2025-04-16 11:55:09,697 - INFO - input_utils - User selected table: student1
2025-04-16 11:55:09,698 - INFO - connection - Connecting to Oracle...
2025-04-16 11:55:11,081 - INFO - connection - Oracle connection established.
2025-04-16 11:55:11,081 - INFO - schema - Fetching schema for table 'student1' from DB 1
2025-04-16 11:55:11,083 - DEBUG - schema - Schema for 'student1': [('sid', 'int'), ('sname', 'varchar(50)'), ('
saddress', 'varchar(100)'), ('marks', 'int')]
2025-04-16 11:55:11,083 - INFO - main - Action: Data Transfer
2025-04-16 11:55:11,083 - INFO - operations - Creating table 'student1' in DB 2
2025-04-16 11:55:11,084 - DEBUG - operations - Executing query: CREATE TABLE student1 (sid NUMBER, sname VARCHA
R2(255), saddress VARCHAR2(255), marks NUMBER)
2025-04-16 11:55:11,099 - INFO - operations - Table 'student1' created successfully.
2025-04-16 11:55:11,100 - INFO - main - ===== DB Transfer Tool Ended =====
PS C:\Users\Nithin Ellendula\Downloads\Intern_Projects\cross_db_transfer>
```

2. Schema Fetching

The **schema fetching process** is a critical foundational component within the data pipeline of this system. It dynamically extracts metadata (i.e., column names and data types) from a specified table in the **source database**, enabling downstream operations such as **type mapping**, **target table creation**, and **data transfer alignment**.

This functionality is encapsulated in the `get_table_schema()` method, which abstracts the underlying query logic per database dialect (MySQL, Oracle, PostgreSQL) while returning a unified schema definition format.

- Auto-detects schema based on database dialect
- Uses appropriate SQL (e.g., `DESCRIBE`, `information_schema`, `user_table_columns`)

Output:

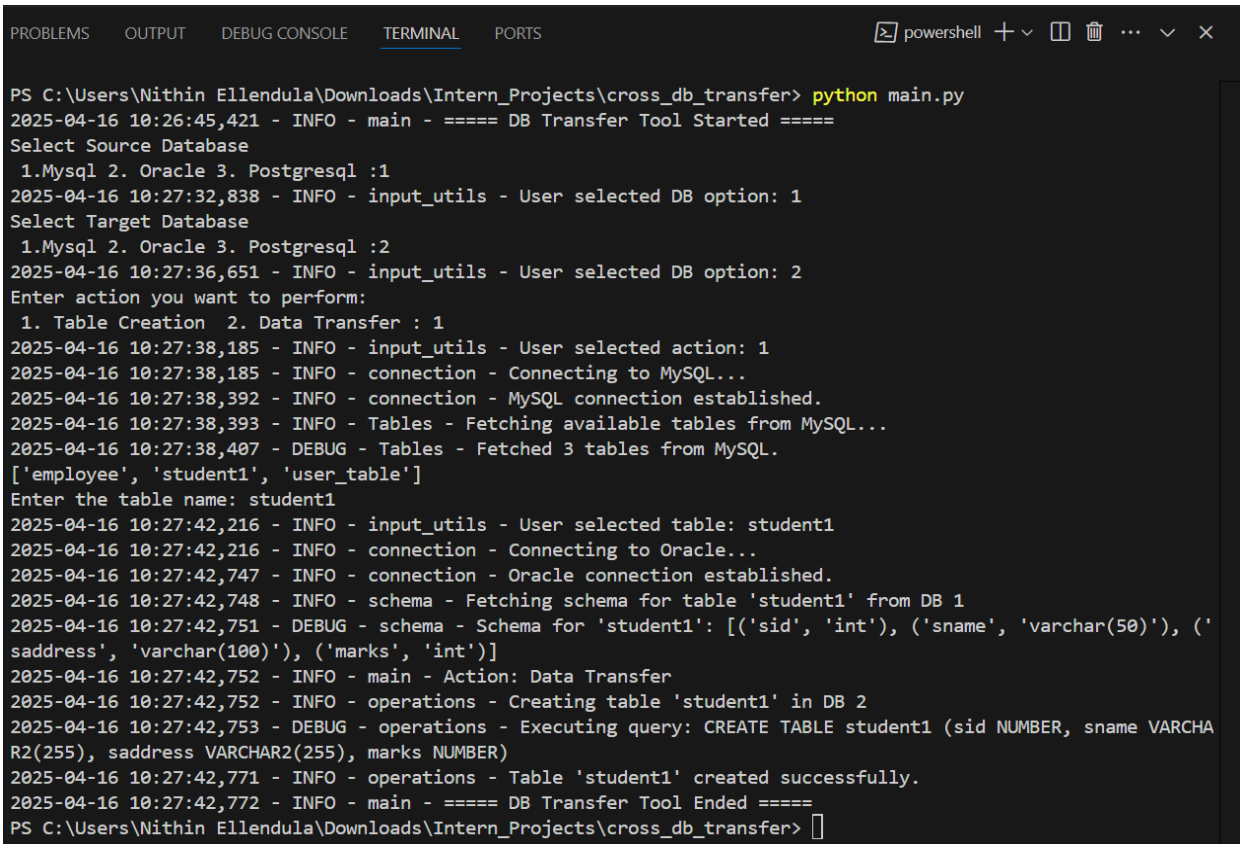
```
Enter the table name: student1
2025-04-16 11:55:09,697 - INFO - input_utils - User selected table: student1
2025-04-16 11:55:09,698 - INFO - connection - Connecting to Oracle...
2025-04-16 11:55:11,081 - INFO - connection - Oracle connection established.
2025-04-16 11:55:11,081 - INFO - schema - Fetching schema for table 'student1' from DB 1
2025-04-16 11:55:11,083 - DEBUG - schema - Schema for 'student1': [('sid', 'int'), ('sname', 'varchar(50)'), ('
saddress', 'varchar(100)'), ('marks', 'int')]
2025-04-16 11:55:11,083 - INFO - main - Action: Data Transfer
2025-04-16 11:55:11,083 - INFO - operations - Creating table 'student1' in DB 2
2025-04-16 11:55:11,084 - DEBUG - operations - Executing query: CREATE TABLE student1 (sid NUMBER, sname VARCHA
R2(255), saddress VARCHAR2(255), marks NUMBER)
2025-04-16 11:55:11,099 - INFO - operations - Table 'student1' created successfully.
2025-04-16 11:55:11,100 - INFO - main - ===== DB Transfer Tool Ended =====
```

3. Table Creation

The **Table Creation Process** is a core procedural component within the data migration lifecycle. It facilitates the **automated generation of destination tables** based on the schema extracted from a specified source table. This process ensures that the destination environment mirrors the structural integrity of the source, while accounting for **data type compatibility** between heterogeneous database engines.

This functionality is orchestrated through the `table_creation()` function defined in `operations.py`, and is executed in coordination with the schema inspection and data type mapping modules.

Output:



4. Data Transfer

The **Data Transfer Process** is a critical operation within this system, responsible for migrating row-level data from a source table to its structurally compatible counterpart in the target database. This process assumes that the target table has been created (manually or via the Table Creation module) and that both source and destination schemas are structurally aligned.

The process is orchestrated by the `data_transfer()` function defined in `data_transfer.py`, and is composed of three phases: **data extraction**, **data insertion**, and **verification**.

- Fetches all rows from the source table
- Inserts into destination using parameterized queries
- Verifies data integrity with SELECT

Output:

```
PS C:\Users\Nithin Ellendula\Downloads\Intern_Projects\cross_db_transfer> python main.py
2025-04-16 12:09:00,124 - INFO - main - ===== DB Transfer Tool Started =====
Select Source Database
 1.Mysql 2. Oracle 3. Postgresql :1
2025-04-16 12:09:07,882 - INFO - input_utils - User selected DB option: 1
Select Target Database
 1.Mysql 2. Oracle 3. Postgresql :2
2025-04-16 12:09:09,395 - INFO - input_utils - User selected DB option: 2
Enter action you want to perform:
 1. Table Creation  2. Data Transfer : 2
2025-04-16 12:09:11,359 - INFO - input_utils - User selected action: 2
2025-04-16 12:09:11,359 - INFO - connection - Connecting to MySQL...
2025-04-16 12:09:11,666 - INFO - connection - MySQL connection established.
2025-04-16 12:09:11,667 - INFO - Tables - Fetching available tables from MySQL...
2025-04-16 12:09:11,669 - DEBUG - Tables - Fetched 3 tables from MySQL.
['employee', 'student1', 'user_table']
Enter the table name: student1
2025-04-16 12:09:16,586 - INFO - input_utils - User selected table: student1
2025-04-16 12:09:16,586 - INFO - connection - Connecting to Oracle...
2025-04-16 12:09:17,875 - INFO - connection - Oracle connection established.
2025-04-16 12:09:17,875 - INFO - schema - Fetching schema for table 'student1' from DB 1
2025-04-16 12:09:17,888 - DEBUG - schema - Schema for 'student1': [('sid', 'int'), ('sname', 'varchar(50)'), ('
saddress', 'varchar(100)'), ('marks', 'int')]
2025-04-16 12:09:17,888 - INFO - main - Action: Data Transfer
Enter the table name: student1
2025-04-16 12:09:22,519 - INFO - input_utils - User selected table: student1
2025-04-16 12:09:22,519 - INFO - transfer - Starting data transfer: student1 from DB 1 to DB 2
2025-04-16 12:09:22,520 - INFO - operations - Selecting data from table 'student1'
2025-04-16 12:09:22,532 - DEBUG - operations - Fetched 10 rows from 'student1'
2025-04-16 12:09:22,533 - DEBUG - transfer - Fetched data: [(101, 'Joe', 'hyd', 89), (102, 'Bob', 'hyd', 90), (
103, 'Henry', 'hyd', 99), (104, 'JAY', 'Bnglr', 98), (105, 'Alex', 'Mum', 78), (106, 'Tom', 'Bnglr', 88), (107,
 'Ankit', 'Mum', 94), (108, 'Varun', 'Del', 85), (109, 'Raj', 'Bnglr', 70), (110, 'nick', 'pune', 100)]
2025-04-16 12:09:22,534 - INFO - schema - Fetching schema for table 'student1' from DB 2
2025-04-16 12:09:22,562 - DEBUG - schema - Schema for 'student1': [('SID', 'NUMBER'), ('SNAME', 'VARCHAR2'), ('SADDRESS', 'VARCHAR2'), ('MARKS', 'NUMBER
')]
2025-04-16 12:09:22,562 - INFO - operations - Detected columns for table 'student1': ['SID', 'SNAME', 'SADDRESS', 'MARKS']
2025-04-16 12:09:22,563 - DEBUG - operations - Constructed INSERT query: INSERT INTO student1 (SID, SNAME, SADDRESS, MARKS) VALUES (:1, :2, :3, :4)
2025-04-16 12:09:22,563 - DEBUG - operations - Inserting row: (101, 'Joe', 'hyd', 89)
2025-04-16 12:09:22,568 - DEBUG - operations - Inserting row: (102, 'Bob', 'hyd', 90)
2025-04-16 12:09:22,569 - DEBUG - operations - Inserting row: (103, 'Henry', 'hyd', 99)
2025-04-16 12:09:22,570 - DEBUG - operations - Inserting row: (104, 'JAY', 'Bnglr', 98)
2025-04-16 12:09:22,570 - DEBUG - operations - Inserting row: (105, 'Alex', 'Mum', 78)
2025-04-16 12:09:22,571 - DEBUG - operations - Inserting row: (106, 'Tom', 'Bnglr', 88)
2025-04-16 12:09:22,572 - DEBUG - operations - Inserting row: (107, 'Ankit', 'Mum', 94)
2025-04-16 12:09:22,573 - DEBUG - operations - Inserting row: (108, 'Varun', 'Del', 85)
2025-04-16 12:09:22,574 - DEBUG - operations - Inserting row: (109, 'Raj', 'Bnglr', 70)
2025-04-16 12:09:22,574 - DEBUG - operations - Inserting row: (110, 'nick', 'pune', 100)
2025-04-16 12:09:22,576 - INFO - operations - Successfully inserted 10 rows into table 'student1'.
2025-04-16 12:09:22,576 - INFO - operations - Selecting data from table 'student1'
2025-04-16 12:09:22,581 - DEBUG - operations - Fetched 10 rows from 'student1'
Transferred Data from student1:
(101, 'Joe', 'hyd', 89)
(102, 'Bob', 'hyd', 90)
(103, 'Henry', 'hyd', 99)
(104, 'JAY', 'Bnglr', 98)
(105, 'Alex', 'Mum', 78)
(106, 'Tom', 'Bnglr', 88)
(107, 'Ankit', 'Mum', 94)
(108, 'Varun', 'Del', 85)
(109, 'Raj', 'Bnglr', 70)
(110, 'nick', 'pune', 100)
2025-04-16 12:09:22,584 - INFO - main - ===== DB Transfer Tool Ended =====
```

5. Logger Utility

The **Logger Process** is an essential component of the system, responsible for providing insights into the execution flow of the application, tracking system events, and managing error reporting. It captures critical information, such as **user actions**, **system status**, and **error conditions**, allowing developers and administrators to monitor and troubleshoot the system more effectively.

The logging process is handled by the `logger.py` module it uses get_logger() and follows structured logging best practices to ensure consistency, scalability, and ease of analysis. It ensures that all major operations, including schema fetching, table creation, and data transfer, are logged with appropriate severity levels.

- Captures:
  - Connection errors
  - Schema mismatches
  - Success messages
- Output to console/log file (extensible)

Output:

```
2025-04-16 10:26:45,421 - INFO - main - ===== DB Transfer Tool Started =====
2025-04-16 10:27:32,838 - INFO - input_utils - User selected DB option: 1
2025-04-16 10:27:36,651 - INFO - input_utils - User selected DB option: 2
2025-04-16 10:27:38,185 - INFO - input_utils - User selected action: 1
2025-04-16 10:27:38,185 - INFO - connection - Connecting to MySQL...
2025-04-16 10:27:38,392 - INFO - connection - MySQL connection established.
2025-04-16 10:27:38,393 - INFO - Tables - Fetching available tables from MySQL...
2025-04-16 10:27:38,407 - DEBUG - Tables - Fetched 3 tables from MySQL.
2025-04-16 10:27:42,216 - INFO - input_utils - User selected table: student1
2025-04-16 10:27:42,216 - INFO - connection - Connecting to Oracle...
2025-04-16 10:27:42,747 - INFO - connection - Oracle connection established.
2025-04-16 10:27:42,748 - INFO - schema - Fetching schema for table 'student1' from DB 1
2025-04-16 10:27:42,751 - DEBUG - schema - Schema for 'student1': [('sid', 'int'), ('sname', 'varchar(50)'), ('saddress', 'varchar(100)'), ('marks',
'int')]
2025-04-16 10:27:42,752 - INFO - main - Action: Data Transfer
2025-04-16 10:27:42,752 - INFO - operations - Creating table 'student1' in DB 2
2025-04-16 10:27:42,753 - DEBUG - operations - Executing query: CREATE TABLE student1 (sid NUMBER, sname VARCHAR2(255), saddress VARCHAR2(255),
marks NUMBER)
2025-04-16 10:27:42,771 - INFO - operations - Table 'student1' created successfully.
2025-04-16 10:27:42,772 - INFO - main - ===== DB Transfer Tool Ended =====
```

==>How to Run

Command: `python main.py`
Follow the CLI prompts:

- Choose source and destination databases
- Choose action:
  -1 for Table Creation
  -2 for Data Transfer
  Select the table name to operate on

Interactive Graphical User Interface (GUI):

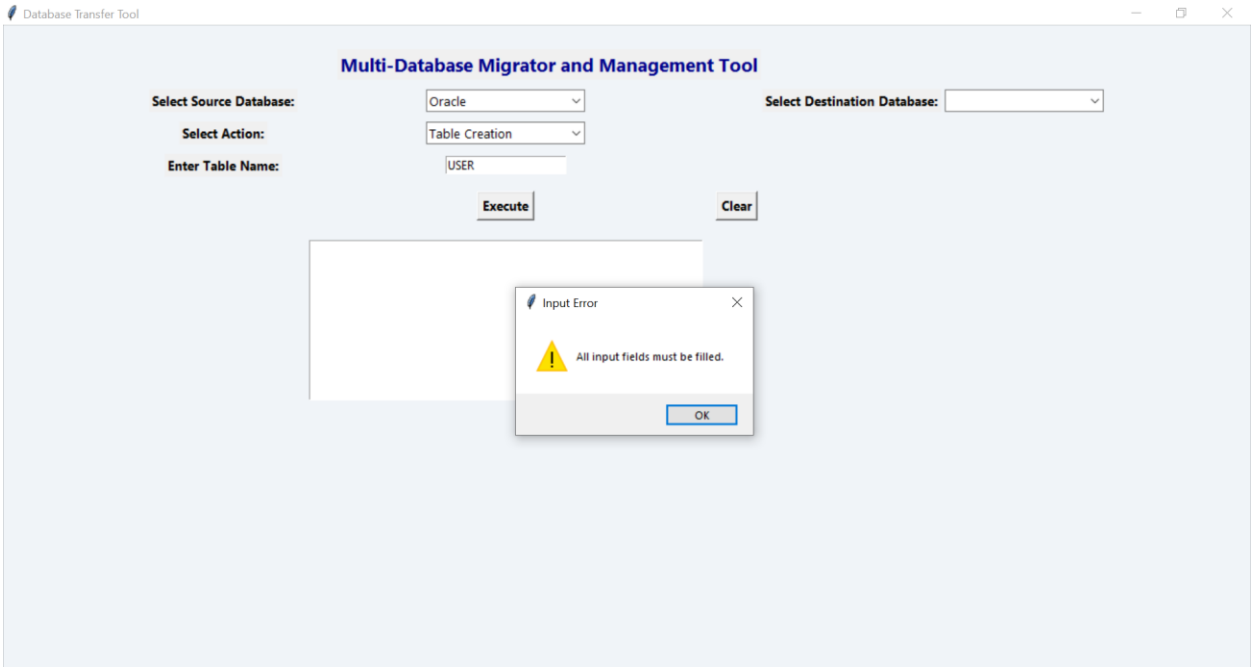
The **Multi-Database Migrator and Management Tool** is designed with an easy-to-use interactive graphical user interface (GUI) built using Python's **Tkinter** library. The interface is clean, organized, and user-friendly, making it simple for anyone, whether a beginner or

expert, to use the tool for database migration and management tasks. Every part of the design is made to help users easily complete the task without confusion.

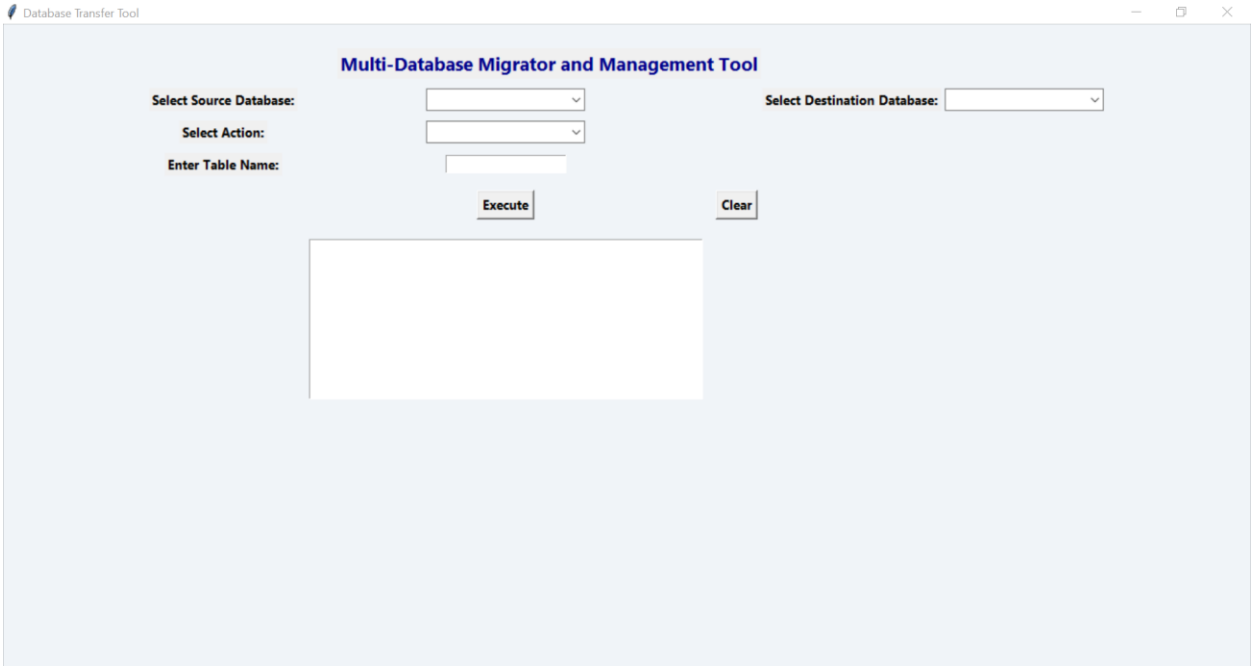
The interface features intuitive dropdowns, entry fields, and clearly labeled buttons, allowing users to select source and destination databases, choose actions, and enter table names with ease. With just a few clicks, users can perform tasks like databases migration through the **“Execute”** button or reset the interface using “**Clear”-** all within a smooth, interactive environment.

This GUI-focused design makes the tool approachable for all users, regardless of technical background, offering amore visual and efficient alternative to traditional command-line workflows.

Input Validation:



User Interface :



Result