# Project Documentation

## 1. Project Overview

**Project Name**: Automated Data Population for Central Bank of Egypt Reports

**Description**:
This project addresses the requirement of populating Excel files sent by the Central Bank of Egypt (CBE) with complex data automatically. These reports often need to be filled with data sourced from internal databases of the bank. The solution automates this process by dynamically extracting data using SQL queries and modifying the corresponding cells in the Excel sheets as per the structure and format required by the CBE.

## 2. Problem Statement

The Central Bank of Egypt regularly sends complex Excel templates that need to be filled with financial, operational, and risk-related data. Manually filling these templates is:

- Time-consuming
- Prone to errors
- Challenging due to the complexity and volume of data

The proposed solution automates data population using a flexible and highly configurable process that reads data directly from the bank's database using SQL queries, ensuring data accuracy and faster report generation.

---

## 3. Key Objectives

- **Automate the Process**: Minimize manual intervention by automating the data extraction and population process.
- **Improve Accuracy**: Reduce human errors by directly querying data from internal databases.
- **Flexibility**: Allow easy addition of new sheets and data configurations by managing input files and SQL scripts.
- **Compliance**: Ensure the data format and structure match the requirements set forth by the Central Bank of Egypt.

---

## 4. How It Works

**Process Flow**:

1. **Folder Structure for SQL Queries**:
    - Each report (Excel sheet) has a dedicated subfolder under the `SQL_Scripts` directory.
    - Each subfolder contains files named after cell references (e.g., `D11.txt`).
    - Each file contains an SQL query that specifies the data required for that cell.
2. **Database Connection**:
    - The system connects to the internal Oracle database using credentials specified in a configuration file (`db_config.txt`).
3. **Query Execution and Data Retrieval**:
    - For each cell reference, the system reads the corresponding SQL file, executes the query, and retrieves the data.
4. **Excel File Modification**:
    - The retrieved data is written to the specified cells in the Excel sheets, matching the format required by the CBE.
    - If cells are part of merged ranges, they are handled appropriately to maintain data integrity.

---

## 5. Project Components

**Folder Structure**:

```
bash
Copy code
Project/
│
├── SQL_Scripts/
│   ├── Report1/  # Subfolder representing a specific Excel sheet
│   │   ├── D11.txt  # SQL query file for cell D11
│   │   ├── E12.txt  # SQL query file for cell E12
│   │   └── ...
│   ├── Report2/
│   │   ├── B5.txt
│   │   └── ...
│   └── ...
│
├── db_config.txt  # Database connection configuration file
├── main.py        # Main execution script
└── README.md      # Documentation
```

**Configuration Files**:

- **`db_config.txt`**: Stores database connection details to ensure secure access to the Oracle database.

**Main Script (`main.py`)**:

- Connects to the database using credentials from `db_config.txt`.

- Iterates over each subfolder (representing an Excel sheet) in `SQL_Scripts`.
- Reads SQL queries from text files, executes them, and retrieves the data.
- Modifies the corresponding cells in the Excel sheet.

---

## 6. Technical Details

**Database Configuration**:

- The `db_config.txt` file contains the following parameters:

```
makefile
Copy code
username=YOUR_USERNAME
password=YOUR_PASSWORD
host=YOUR_HOST
port=1521
service_name=YOUR_SERVICE_NAME
```

**SQL Query Files**:

- Each text file corresponds to a specific cell in an Excel sheet.
- The contents of the file should contain a valid SQL query to retrieve the required data.

**Example SQL Query** (e.g., `D11.txt`):

```
sql
Copy code
SELECT SUM(CRNT_BAL)
FROM BI_DWH.PIO_ACCOUNTS
WHERE DAY_DATE = TO_DATE(:date_param, 'MM/DD/YYYY HH24:MI:SS')
AND SUB_ACCT_CODE BETWEEN 020 AND 029;
```

---

## 7. Usage Instructions

1. **Prepare the SQL Scripts**:
   - Create subfolders under `SQL_Scripts` for each sheet name.
   - Add text files named after cell references, containing SQL queries for each cell.
2. **Configure Database Access**:
   - Edit the `db_config.txt` file to include the appropriate database connection details.
3. **Run the Script**:
   - Execute the main script using:

```
bash
Copy code
python main.py
```

- The script will automatically populate the specified cells in the Excel sheets.

---

## 8. Example Scenario

**Objective**: Populate an Excel report sent by the CBE with data for financial risk metrics.

1. **Create Folder Structure**:
   - `SQL_Scripts/Report1/` contains files like `D11.txt`, `E12.txt`, etc.
2. **Write SQL Queries**:
   - Each file contains a query that retrieves the required metric.
3. **Execute the Script**:
   - The script reads the queries, fetches data from the database, and updates the Excel sheet.

---

## 9. Business Value and Benefits

**For the Business Team**:

- **Time Efficiency**:
  Eliminates manual data entry, allowing staff to focus on higher-value tasks.
- **Accuracy and Consistency**:
  Ensures that all reports are populated with accurate data fetched directly from the database.
- **Scalability**:
  New reports or cells can be easily added by updating the `SQL_Scripts` folder, making the solution highly adaptable.
- **Compliance and Reporting**:
  Helps the bank meet regulatory reporting requirements set by the CBE with minimal effort and high data accuracy.

---

## 10. Potential Enhancements

- **Enhanced Security**:
  Use encrypted storage for database credentials.
- **GUI Interface**:
  Develop a graphical interface for easier management of configurations and SQL queries.
- **Support for Multiple Databases**:
  Extend support to other database types, such as PostgreSQL or MySQL.