**Table of Contents**

1. Description --------------------------------------------------------------------------------- 2
2. Flow Chart:
   1. Sequential Flow Chart ----------------------------------------------------------- 2
   2. Flow chart 1 ----------------------------------------------------------------------- 2
   3. Flow chart 2 ----------------------------------------------------------------------- 3
3. Installation -------------------------------------------------------------------------------- 3
4. Step by Step Explanation -------------------------------------------------------------- 5
5. Command to connect the PostgreSQL DB ---------------------------------------- 6
6. Screen Shots of code execution ------------------------------------------------------- 7
7. Code Explanation ------------------------------------------------------------------------ 9
8. Error Handling -------------------------------------------------------------------------- 17

**CSV/Excel to PostgreSQL dB table**

**A blue squares with white text

Description automatically generated**

***The detailed flow chart in two steps:***

***Flow Chart 1:***

This flow chart explains the initial process where a file is processed for any noise or unwanted data and takes users confirmation and cleans the data. The data is then saved to a processed folder which will be then used to create a SQL table.

A diagram of a computer program

Description automatically generated

**CSV or Excel Pre-processing**

***Flow chart 2:***

This flow chart follows the first flow chart and mainly covers the SQL table creation or insertion/updation in detail. The data types of the columns are decided by script automatically considering first 100 rows which can be changed if needed. We take user input for selecting the primary key based on the column nos. If there is any archive file created, we update the SQL table since the data is already inserted else, we create a new table and insert the data. While updating the records If there are any conflicting records, we ask for user input to update them or ignore them.

A diagram of a diagram

Description automatically generated

**SQL table creation and data insertion**

**Installation and setup:**

Open the ubuntu server or instance and run the following commands to setup the server.

apt-get update

apt-get install nano

apt-get install python3

apt-get install python3-pip

apt-get install postgresql

You might be asked to select region (select 2 America)

service postgresql start

su - postgres -c "createuser alok"

su - postgres -c "psql -c \"ALTER USER alok WITH PASSWORD 'reddy';\""

su - postgres -c "createdb meddb"

su - postgres -c "psql -c \"GRANT ALL PRIVILEGES ON DATABASE meddb TO alok;\""

Alternate commands for a sudo user:

sudo apt-get update

sudo apt-get install nano

sudo apt-get install python3

sudo apt-get install python3-pip

sudo apt-get install postgresql

You might be asked to select region (select 2 America)

sudo service postgresql start

sudo -u postgres createuser alok

sudo -u postgres psql -c "ALTER USER alok WITH PASSWORD 'reddy';"

sudo -u postgres createdb meddb

sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE meddb TO alok;"

requirements.txt

pandas, psycopg2-binary, lxml

There are three python libraries that are needed for the data insertion. We need to run the above requirements.txt file using below command. This command installs all required python packages.

# pip3 install -r requirements.txt

# python3 csv2sql.py enrollment.xls

**Step by Step Explanation:**

1. Have the csv or Excel files which are to be processed and saved to the database in the same directory of csv2sql.py file.
2. Run the command as below for sql table creation and insertion.

This command is to be run in terminal with python and necessary packages installed.

Command for csv: python3 csv2sql.py filename.csv

Command for excel: python3 csv2sql.py filename.xls

python3 csv2sql.py filename.xlsx

Ex: python3 make\_trimmedcsv.py publications.csv

1. **html to csv:**

If the data file (csv or excel) provided has any html content it will be parsed using beautifulsoup and table is saved with same name as csv file.

1. **Filter csv rows with a given percentage of nonnull fields:**

The csv will be processed to remove the rows that have fields less than the given threshold (hardcode to 3%) can be changed.

1. **Duplicate rows:**

The script looks for any duplicate rows and if found the user will be asked to confirm to delete the repeating records.

1. **Process the new csv file:**

We now process the new csv file and check for any rows having more than 4 fields (this can be changed) and save it to processed folder.

1. **Check for archived folder:**

We then check for the archived folder with the filename that is processed. If there are any files, we check the headers and if headers don’t match, we exit the program. If there are no files in archived folder for the processed file, we go ahead with new SQL table creation.

1. **Create Primary Key:**

The first 100 rows in the csv file are used to determine the data of the columns. The columns are displayed with numbers for the user to select the primary key.

1. **Table Creation:**

Once the columns are selected a create query is generated and executed upon user confirmation.

1. **Insert Records:**

The user is then asked for the confirmation to insert the records.

1. **Update Records:**

If there are files in the archived folder for the given processed file, we update the records from the csv file.

We also check for any conflicting rows and insert them upon user confirmation.

**Command to connect the PostgreSQL DB:**

The data pushed can be verified by connecting the postgres database and running few select queries.

psql -d dbname -U username -W -h hostname -p portnumber

psql -d meddb -U alok -W -h localhost -p 5432

password: reddy

**Screen shots:**

A screenshot of a computer screen

Description automatically generated

**steps 1 to steps 7**

A screen shot of a computer

Description automatically generated

**Step 8**

A screenshot of a computer screen

Description automatically generated

**steps 9 to steps 11**

**Code Explanation**

The csv2sql.py file performs various operations on a CSV file.

**Importing necessary modules:**

A screenshot of a computer

Description automatically generated with low confidence

This section imports the required modules for file manipulation, date and time operations, CSV handling, system-related functions, and regular expressions.

**Defining the trimcsv function:**

This function reads a file and removes a specified number of lines from the beginning. It then writes the remaining lines back to the same file.

A screen shot of a computer code

Description automatically generated with low confidence

**Defining the remove\_duplicates function:**

This function uses the pandas library to read a CSV file into a DataFrame. It identifies duplicate rows in the DataFrame, displays them, and prompts the user to confirm whether they want to delete those rows. If confirmed, the duplicate rows are removed, and the modified DataFrame is saved back to the CSV file.

A screenshot of a computer code

Description automatically generated with low confidence

**Defining the increment\_duplicate\_columns function:**

def increment\_duplicate\_columns(csv\_filename):

This function reads a CSV file and performs the following operations:

* Reads the file and stores the data rows in a list.
* Retrieves the header row.
* Counts the occurrences of each processed column name by removing non-alphanumeric characters and extra spaces.
* Modifies column names to make them unique by appending a number to duplicates.
* Writes the modified data (including the updated header row) back to the CSV file.

**Defining the process\_csv\_file function:**

def process\_csv\_file(filename, n=4):

    # Check if file exists

    if not os.path.isfile(filename):

        print(f"Error: File '{filename}' does not exist")

        return

    # Rest of the function...

This function processes a CSV file based on the provided filename. It takes an optional argument n (defaulted to 4) which represents the number of commas to consider for data processing.

The function performs the following steps:

* Reads the input file to find the maximum number of columns (max\_columns) and the rows with that count (max\_rows).
* Generates a timestamp for the processed file name.
* Creates a directory structure for the processed files.
* Creates the processed filename based on the original filename and the timestamp.
* Writes a new CSV file with cleaned rows (removes extra spaces and double quotes) based on the specified n and max\_columns.
* Reads the new file, cleans the first row (header row) by removing extra spaces, symbols, and converting to lowercase.
* Writes the modified data back to the CSV file.
* Calls the increment\_duplicate\_columns function to handle duplicate column names.
* Calls the remove\_duplicates function to remove duplicate rows.
* Prints a confirmation message.

**Handling command-line arguments:**

if len(sys.argv) < 2:

    print("Error: Please provide the CSV filename as an argument.")

    sys.exit(1)

filename = sys.argv[1]

n = 5  # Default value for n if not provided in sys arguments

num\_lines = 0  # Default value for num\_lines if not provided in sys arguments

if len(sys.argv) >= 3:

    n = int(sys.argv[2])

if len(sys.argv) >= 4:

    num\_lines = int(sys.argv[3])

This section handles command-line arguments. The script expects at least one argument, which is the CSV filename. The optional second argument sets the value of n, representing the number of commas to consider for processing. The optional third argument sets the number of lines to remove using the trimcsv function.

**Calling the filter\_csv\_rows function:**

def filter\_csv\_rows(filename, output\_filename, threshold):

This function filters rows in a CSV file based on the threshold of non-null fields percentage. It reads the input file, checks each row's non-null fields percentage, and either includes or skips the row based on the threshold.

**Calling the create\_table function:**

We first get the most recent processed file which is used for table creation.

We use psycopgs connection object to connect to database.

The connection object needs database name, username, password and port number.

conn = psycopg2.connect(database="meddb", user="alok", password="reddy", host="localhost", port="5432")

Since the table columns shouldn’t start with numbers. We use a regex and key mappings for ordinal numbers. They will be replaced with text.

def replace\_ordinal\_numbers(column\_name):

    # Replace ordinal numbers

    ordinal\_mapping = {

        "1st": "first",

        "2nd": "second",

        "3rd": "third",

        "4th": "fourth",

        "5th": "fifth",

        "6th": "sixth",

        "7th": "seventh",

        "8th": "eighth",

        "9th": "ninth"

    }

    # Check if the first character is a digit and then replace numbers without ordinal indicators

    if column\_name[0].isdigit():

        column\_name = re.sub(r'^(\d+)([a-zA-Z]+)', lambda m: {\*\*{"1": "first", "2": "second", "3": "third", "4": "fourth", "5": "fifth", "6": "sixth", "7": "seventh", "8": "eighth", "9": "ninth"}, \*\*ordinal\_mapping}[m.group(1)] + m.group(2), column\_name)

    else:

        # Check if the first three letters match an ordinal mapping and replace them

        first\_three\_letters = column\_name[:3]

        if first\_three\_letters in ordinal\_mapping:

            column\_name = ordinal\_mapping[first\_three\_letters] + column\_name[3:]

    return column\_name

**Defining the data types of the columns:**

We use the first 100 rows in the csv file to determine the data type of the column. It can be Decimal, Integer, Text.

data\_types = get\_column\_data\_types(headers, csvreader)

This information will also be displayed in proper format and the user will be asked for primary key selection.

print\_column\_data\_types(headers, data\_types)

# Determine the primary key columns

pk\_columns = get\_primary\_key\_columns(headers)

**Defining the Create table query:**

create\_table\_query = f"CREATE TABLE IF NOT EXISTS {table\_name} ({', '.join([header.replace('(', '').replace(')', '') + ' ' + data\_types[i] for i, header in enumerate(headers)])}, PRIMARY KEY ({pk\_constraint}));"

The information like primary key constraints and column names which are set in previous steps will be used for create table query creation and then will be executed.

cur.execute(create\_table\_query)

**Insert data into table:**

insert\_data\_into\_table(cur, conn, processed\_filepath, headers, data\_types, table\_name)

We call the function insert data to push the data from csv to the table created.

        insert\_query = sql.SQL("INSERT INTO {table} ({columns}) VALUES %s ON CONFLICT DO NOTHING;").format(

            table=sql.Identifier(table\_name),

            columns=sql.SQL(', ').join(map(sql.Identifier, headers\_lower))

        )

The insert query is created and will be executed as per user confirmation.

After successful insertion the data from the table will be used to create an archive file with the same file name.

                # Create archived folder if it doesn't exist

                if not os.path.exists('archived'):

                    os.makedirs('archived')

                # Create a folder with the name of the CSV file in the archived folder if it doesn't exist

                foldername = os.path.splitext(os.path.basename(filename))[0]

                foldername = foldername.split('\_')[0]

                if not os.path.exists(f'archived/{foldername}'):

                    os.makedirs(f'archived/{foldername}')

                # Write the SQL table to a CSV file in the archived folder

                with open(f'archived/{foldername}/{table\_name}\_{timestamp}.csv', 'w') as csvfile:

                    csvwriter = csv.writer(csvfile)

                    csvwriter.writerow(headers\_lower)  # Write lower case column names

                    cur.execute(f"SELECT \* FROM {table\_name}")

                    rows = cur.fetchall()

                    for row in rows:

                        csvwriter.writerow(row)

**Call update function if archive file is not empty:**

We create a query from the csv and execute as per user confirmation.

            query = sql.SQL("INSERT INTO {table} ({columns}) VALUES ({values}) ON CONFLICT ({conflict\_columns}) DO NOTHING RETURNING \*;").format(

                table=sql.Identifier(table\_name),

                columns=sql.SQL(', ').join(map(sql.Identifier, headers)),

                values=sql.SQL(', ').join([sql.Placeholder()] \* len(headers)),

                conflict\_columns=sql.SQL(', ').join(map(sql.Identifier, pk\_columns))

            )

**Checking for any conflicting records:**

If there any conflicting records we add them to a separate list and user will be asked for confirmation to push them to database.

        if conflicting\_records:

            print(f"{len(conflicting\_records)} conflicting records:")

            for record in conflicting\_records:

                print(record)

            print(f"{len(conflicting\_records)} conflicting records:")

            # Ask for confirmation before pushing conflicting records

            confirmation = input("Do you want to push the conflicting records to the database? (yes/no): ")

            if confirmation.lower() == "yes":

                # Push conflicting records to the database

                for record in conflicting\_records:

                    try:

                        cur.execute(query, record)

                        rows\_inserted += 1

                    except psycopg2.IntegrityError:

                        print(f"Failed to insert conflicting record: {record}")

                    else:

                        rows\_updated += 1

                print("Conflicting records inserted.")

            else:

                print("Conflicting records not pushed to the database.")

We use the same logic as in insert function to save the updated or pushed file to the archived folder.

**ERROR Handling:**

**1)Error while updating the Table:**

The following error can occur when the archived folder of the specific file is not empty. This means the table has already been created but deleted later.

Error can occur because of two scenarios.

1. Table is not present in Database, but we are trying to push or update the table.
2. There is an error while processing the csv file.

processed/enrollment/enrollment\_2023-07-12\_19-08-29.csv

Archive file is not empty. Table updation is done

The primary columns are :

[]

Error occurred: syntax error at or near ")"

LINE 1: ...53\_level\_1', 'Unnamed: 154\_level\_1') ON CONFLICT () DO NOTHI...

**Example of scenario 1**

***How to debug more***

We can connect to the sql db and check if table exists.

1. Connect to postgresql db

psql -d dbname -U username -W -h hostname -p portnumber

psql -d meddb -U alok -W -h localhost -p 5432

password: reddy

1. Check the existing tables in the db

\d

The above command shows all the existing tables

1. Check specific table (In our case we are checking the specific table enrollment)

\d tablename

\d enrollment

***How to fix the problem from local:***

A simple fix for the above issue is to delete the local files for the specific table in archived folder. This means we are telling the script there is no table created earlier for the specific table.

Move and remove the files pushed earlier in the archived folder.

1. cd archived
2. Remove the files of the specific folder

rm -r tablename/

rm -r enrollment/

Here we are removing the folder enrolment recursively so that the script can start with table creation and insertion. (Since we found out there is no table in early steps, the archive file needs to empty so that the data inserting starts with creation).

Now we can rerun the python csv2sql.py enrolment.xls command and everything works fine.