REPORT - Group 10

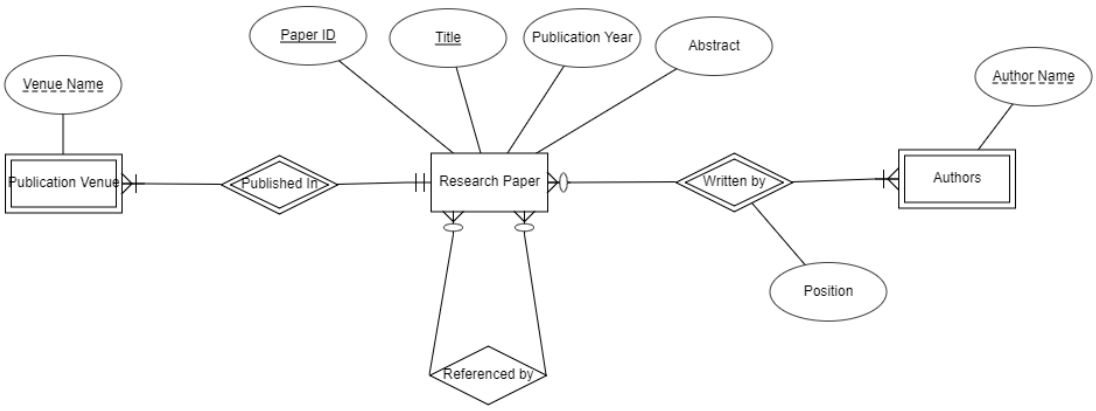
TEAM MEMBERS:

GOKUL - CS19BTECH11048
RASHMITHA - CS19BTECH11019
SRAVYA - CS19BTECH11017
LIKITHA - CS19BTECH11008

Why did we use python?

Python supports databases like PostgreSQL, and also it supports Data Query Statements, Data Manipulation Language, and Data Definition Language. The Postgres database performs better because of its concurrent support for write operations without using read/write locks, and also we are comfortable with the python language. Python has a module named psycopg2 (https://pypi.org/project/psycopg2/) which we can use to connect to a database and perform various queries.

ER-diagram:



This is our modified ER diagram, the entities are the Research paper, Authors, and Publication Venue. The weak entities are Authors and Publication Venue because they can't be defined uniquely by their attributes alone.

The attributes of the Research Paper entity are Paper_ID, Title, Publication Year, and Abstract; attribute of Authors entity is Author Name; attribute of Publication Venue entity is Venue Name; attribute if Written by relationship is Position.

In general, a database can be represented using the notations, and these notations can be reduced to a collection of tables. A table is created for each entity and for many to many relationships. For this ER diagram, we have to create four tables namely Research Paper, Authors, Publication Venue, and Referenced by.

PAPER
PAPER_ID TITLE YEAR ABSTRACT

AUTHORS
PAPER_ID AUTHORS NAME POSITION PRIMARY KEY(PAPER_ID, AUTHOR_NAME)

VENUES
PRIMARY KEY (PAPER_ID) VENUE_NAME

REF
PAPER_ID REFERENCED_ID PRIMARY KEY(PAPER_ID, REFERENCED_ID)

Code Implementation:

Pre-processing:

- We saw that there are some missing entries in the given file source.txt, we have pushed in Null for those missing entries.
- We noticed that there are some repeated comma(,) in the authors line, in the file source.txt, we handled that situation.
- We noticed that there are author names being repeated in the given file source.txt, we only push in unique author names for each paper into the database.
- We have also used a position attribute to denote the position in which a given author appears in the list of authors, which was mentioned to be important in the problem statement of Assignment-I.

Program Outline:

We have implemented 2 different codes - one to read from the given file source.txt, connect to the database and populate it directly and one to create csv files, which can be then loaded into the database.

First, we have to import the library psycopg2. Psycopg2 is the most used PostgreSQL database, it connects the python language and libraries to manipulate the data in the PostgreSQL database.

You will be prompted to enter database name, username, password of the postgres database, so that psycopg can connect to the database and execute the queries.

We have created a paper dictionary as given below in python to read and store the values from the file source.txt.

```
paper = {
   "title" : "",
   "authors" : [],
   "year" : "",
   "venue" : "",
   "index" : "",
   "references" : [],
   "abstract" : ""
}
```

Using psycopg2 library first we use the function **connect()** creates a new database which returns the new connection instance. The class connection encapsulates the created database and creates a cursor instance using the function **cursor()** to execute the queries of the database. The class cursor interacts with the database and we can send the commands like execute.

We have created a database test with the username test and password 1234 with the host address 127.0.0.1 (local host) and port as 5432. Now using the cursor() we have created the connection to the database, and using the execute() we have created the PAPER TABLE using multiline strings i.e using three quotes.

```
conn = psycopg2.connect(database = db, user = us, password = pw, host =
"127.0.0.1", port = "5432")
print ("Opened database successfully")

cur = conn.cursor()
start = time.time()
cur.execute(
    """
    CREATE TABLE PAPER (
        PAPER_ID INTEGER PRIMARY KEY,
        TITLE VARCHAR(1000),
        YEAR INTEGER,
        ABSTRACT VARCHAR(50000)
    );
"""
)
```

Similarly we have created the AUTHORS, VENUES, and REF TABLES.

We created empty lists for paper, paper[“abstract”], paper[“authors”], paper[“references”], paper[“venue”] to store the entries for the tables.

Now we have read the source.txt file an insert the values into the tables. We used f.readlines() to read the contents of the file. Now we will write the queries to insert the values using f-strings because they have an easy syntax to specify the variables in the insert query. After writing the queries we have to execute them using the function **execute().**

Using the INSERT INTO TABLE query we can insert the values after reading from the file. Here we are inserting **NULL** if there are no entries i.e (len(paper[“abstract”]) == 0) in the abstract column.

We keep reading from the file source.txt, and as soon as a ‘\n’ is seen, it means the end of one entry, so we push in whatever we have read till then into the database using the curr.execute() command.

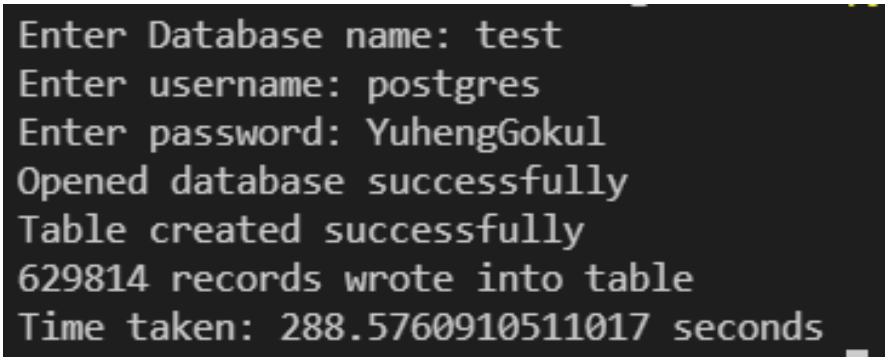
```
query = f"""
            INSERT INTO PAPER (PAPER_ID, TITLE, YEAR, ABSTRACT)
            VALUES ({paper["index"]}, '{paper["title"]}', {paper["year"]},
            """
        if len(paper["abstract"]) == 0:
            query += "NULL);"
        else:
            query += f"""'{paper["abstract"]}');"""

        cur.execute(query)
```

The second program creates 4 csv files - papers.csv, authors.csv, references.csv, venues.csv (i.e. loaders) which can be then used to populate the database.

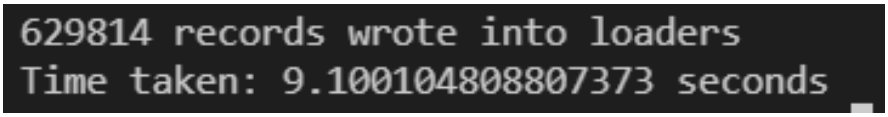
We measured the time taken for each code using the python module time.

Screenshots of Output:

First program (the one that populates the database directly)



Second program (the one that creates 4 csv files) i.e. loaders



Csv files can be loaded into database by using the following command in sql shell, after opening a database:

COPY <table name> (attributes)
FROM 'location of csv file'
DELIMITTER ','
CSV HEADER;

Files Submitted:

Main.py - The file that reads from source.txt and populates the database
Loaders.py - The file that creates the 4 csv files (loaders) for each table
Dump.sql - The pg dump file that we got from using the pg_dump command
Report.pdf - This report