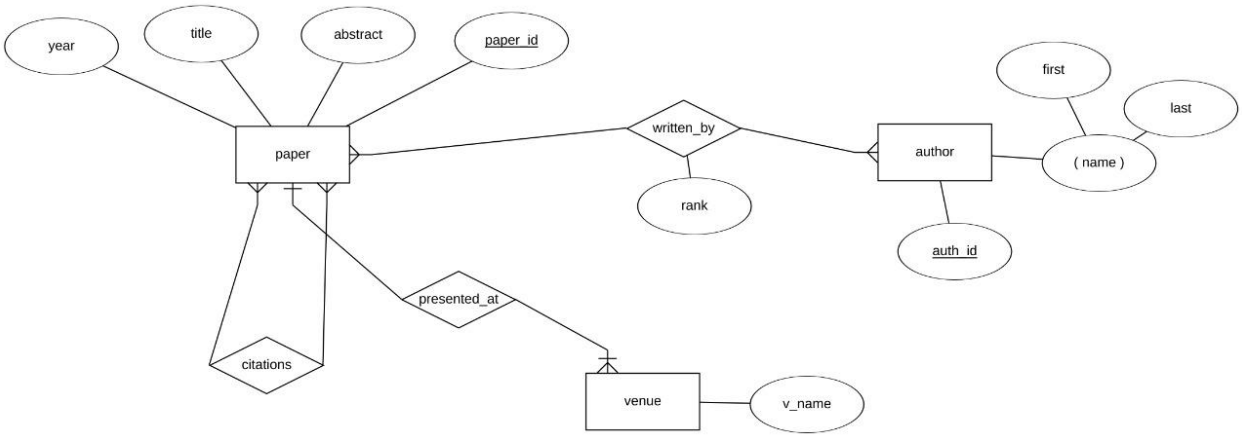
Assignment-2

Group-7

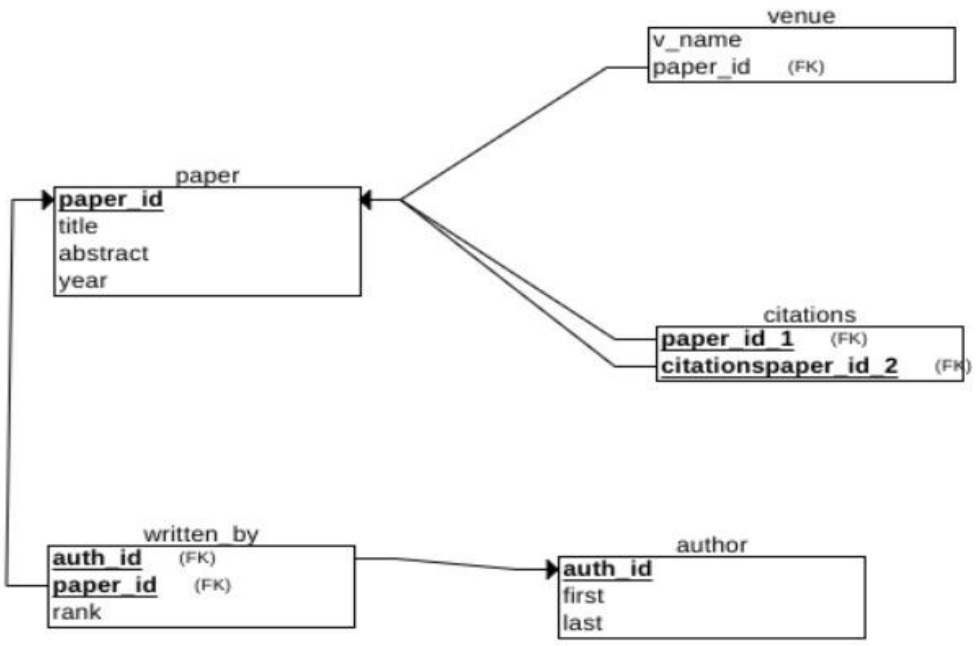
Group members: AI20BTECH11004, AI20BTECH11006, AI20BTECH11011, AI20BTECH11027

The ER diagram was adapted to suit the data provided for the assignment.

The modified ER diagram is given below



And it's relational schema is as follows



Our Parser

We are reading the data from the text file using a C++ program and parsing it into 5 separate csv files, one for each table in the relational schema. The data has a wide range of characters which prohibit us from using any common delimiters for parsing (such as “,”, ”#”, ”$” etc). Hence, we used the unicode character “\u0007” as the delimiter, the reason being that it is one of few single byte characters which can be used as delimiters in Postgresql and as this character didn’t appear anywhere in the data unlike other allowed delimiters.

Some crucial design choices:

- Every author was assigned a unique id based on their appearance in the source file.
- We used C++ STL map, to map author_name to author_id. This helps in faster search/access of author_id, whenever we encounter author_name.
- We dealt with the appearance of “ character in title, abstract by doubling it up.
- To ensure that a paper does not cite itself, a database level Check was created on the citations table.
- We used a unique delimiter for parsing which never occurred in the data
- We removed the NOT NULL constraint from the author name and venue. This was done because there were several instances where the given column had NULL value in the data. Putting a placeholder in place of NULL would be as good as not having the constraint to begin with.
- We splitted the author name by using ‘,’ as the delimiter, the first word being the first name and whatever follows as the last name. This was the only feasible choice because of a lack of standardization while storing names in the original database.
- We considered names like A. Turing as different from Alan Turing because there is a lack of information here, A. could mean anything, like Alexander or Adam, since there is no unique parameter to distinguish between such authors, we decided to consider them as separate authors.
- There are papers which have all the same information except for the paper_id, we considered them as separate papers because there can be other papers which may cite either of the papers, removing such a paper would further cause dependency issues in citations relation.

We have tried our best to parse the data in as elegant a manner as possible. HoweverW we do realize that there are still many shortcomings because of a lack of standardization in the way the data was originally stored. These shortcomings could have been overcome had the data been originally stored in a relational database.

Our Loader

We chose to write the processed data into csv files and then populate the database (using copy command), rather than processing and pushing the database line by line (using API like pqxx), as it's far more efficient and faster this way. Although we have to store the same data (after processing), as the size isn’t very large, it is feasible in this case.

We were able to parse and load the entire data within 90-100 seconds when logged into postgresql database through a unix domain socket on a machine running on Ryzen 5 4600 processor with 16GB DDR4 RAM. We observed that it takes above 20 minutes to parse and load the data sequentially using pqxx.

We can clearly see that the benefits outweigh the storage aspect of this method.

How to Reproduce the Results

1. Compile and run parser.cpp : This will make 5 csv files
2. Then run create_schema.sql to create tables in the database
3. Next run the loader.sql, **change the paths appropriately** in order to execute this program.