Mini Project 2

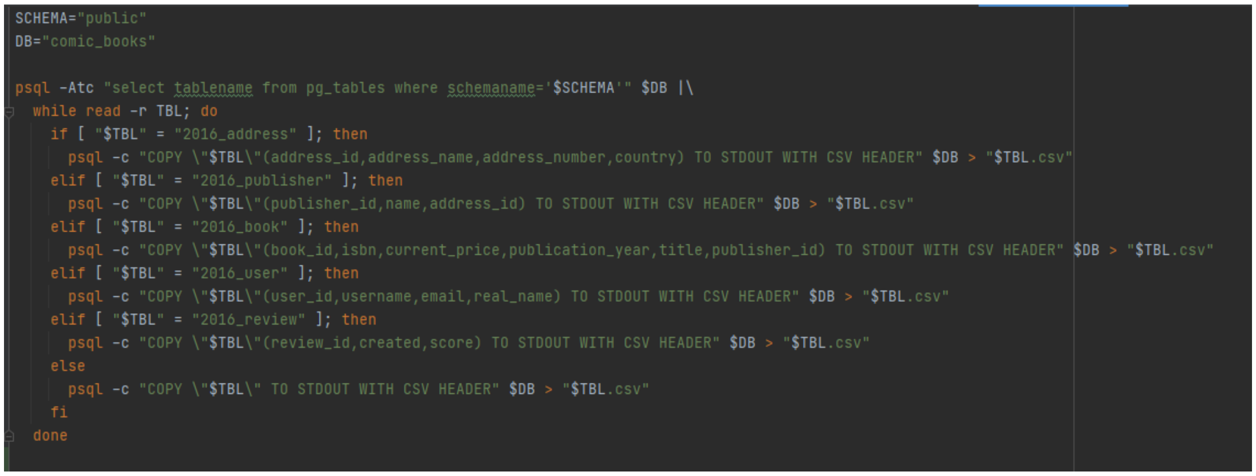
Assignment 2

The database can be loaded as in the previous exercise, using the following 2 commands:

1. *psql -f 2016_schema.sql <database_name>*
2. *psql -f 2016_export.sql <database_name>*

Both files are provided.

To generate the csv files, **provided that the above commands have run**, a shell script was used.



```
SCHEMA="public"
DB="comic_books"

psql -Atc "select tablename from pg_tables where schemaname='$SCHEMA'" $DB |\
  while read -r TBL; do
    if [ "$TBL" = "2016_address" ]; then
      psql -c "COPY \"$TBL\"(address_id,address_name,address_number,country) TO STDOUT WITH CSV HEADER" $DB > "$TBL.csv"
    elif [ "$TBL" = "2016_publisher" ]; then
      psql -c "COPY \"$TBL\"(publisher_id,name,address_id) TO STDOUT WITH CSV HEADER" $DB > "$TBL.csv"
    elif [ "$TBL" = "2016_book" ]; then
      psql -c "COPY \"$TBL\"(book_id,isbn,current_price,publication_year,title,publisher_id) TO STDOUT WITH CSV HEADER" $DB > "$TBL.csv"
    elif [ "$TBL" = "2016_user" ]; then
      psql -c "COPY \"$TBL\"(user_id,username,email,real_name) TO STDOUT WITH CSV HEADER" $DB > "$TBL.csv"
    elif [ "$TBL" = "2016_review" ]; then
      psql -c "COPY \"$TBL\"(review_id,created,score) TO STDOUT WITH CSV HEADER" $DB > "$TBL.csv"
    else
      psql -c "COPY \"$TBL\" TO STDOUT WITH CSV HEADER" $DB > "$TBL.csv"
    fi
  done
```

Figure 1: Bash Script for exporting the csv files from the database

It is quite simple. Given the database name, in this case it is **comic_books,** and the schema name, in this case it is **public,** it loops through all the database tables and executes the copy command.

Some notes:

1. The file can be executed by the command ***<shell> export_tables.sh*** In my case the shell that was used was **bash.** If the command does not work you

may want to change the file’s permissions by the command: ***chmod +x export_tables.sh*** and then run the file as executable ***./export_tables.sh***

2. The files will be generated in the same directory as the script.
3. There are some if conditions for some tables. These are used to export the specific columns that will be useful for the graph database, since not all columns will be used, memory and space will be saved.

The csv files used will also be delivered under the csv_files directory.

Graph design:

For the graph design, nodes and edges were created based on the following requirements:

- Book data: ISBN, title, authors, publisher, publication year, current price
- Publisher data: name, country of headquarters
- Author data: full name, biological gender, nationality
- User data: username, email, real name
- Book orders data: which user made the order, her address of delivery, the order
- placement timestamp, and the order completion timestamp.
- Book reviews data: timestamp, score

Schema:

Node Address:

<id>: id (id field assigned by neo4j)

address_id : integer (unique)

country: string

Node Book:

<id>: id (id field assigned by neo4j)

book_id : integer (unique)

isbn: string unique

current_price: float

publication_year: string

title: string

Node Publisher:

<id>: id (id field assigned by neo4j)

publisher_id : integer (unique)

name: string

Node Author:

<id>: id (id field assigned by neo4j)

author_id : integer (unique)

name: string

gender: string

nationality: string

Node User:

<id>: id (id field assigned by neo4j)

user_id : integer (unique)

username*: string (unique)*

email*: string (unique)*

real_name: string

Node Order:

<id>: id (id field assigned by neo4j)

order_id : integer (unique)

placement: string

completed: string

Node Review:

<id>: id (id field assigned by neo4j)

review_id : integer (unique)

timestamp: string

score: integer

Edge (publisher)-[:HAS_HEADQUARTERS_IN]->(address):

This relationship represents the address in which a publisher has his/her headcounters.

Edge (user)-[:HAS_ADDRESS]->(address):

This relationship represents the address which a user has registered in the system.

Edge (book)-[:PUBLISHED_BY]->(publisher):

Relationship between books and publishers.

Edge (user)-[:HAS_ORDERED]->(order):

Relationship that represents the order that a user has placed.

Edge (order)-[:SHIPPED_TO]->(address):

Relationship that represents the address where an order has been shipped to.

Edge (book)-[:AUTHORED_BY]->(author):

Relationship between books and authors.

Edge (order)-[:INCLUDES_BOOK]->(book):

Relationship that represents the book that is included in an order.

Edge (book)-[:HAS_REVIEW]->(review):

Relationship between books and reviews.

Some notes on the **2016_import.sql** file:

1. The cypher statements where run on Neo4j browser – **version 4.2.1**
2. All the cypher statements can be executed using the multi statement query editor of Neo4j browser. However, since the statements for loading the Author and Review Nodes contain auto transactions which are not supported by the multi statement editor they need to be run separately. Periodic commits were used to avoid memory errors due to large data volume in the author and review csv files.
3. The statements must be run in the same order as in the file. However, they are split into 3 parts: *node creation, constraint creation and edge creation.* Order of statements in these parts does not matter.
4. Unique constraints are added for all the *_id* fields such as *author_id, book_id* etc. Unique constraints where also added for *email, username* and *isbn*. call db.awaitIndexes(300) function with timeout of 300 seconds (about 5 minutes) was used to wait on the indexes coming online.

5. Note that since cypher does not include data types such as *varchar (200), int (4)* etc, which impose constraints on data all the constraints except for the uniqueness should be taken care of by code checks or be handled manually when adding new data.
6. The csv files that were used, were placed directly in the import directory used by Neo4j for importing files, so no path was specified – example: 'file:///2016_review.csv', 'file:///2016_author.csv' etc.
7. Merge statements were tried out for creating the Nodes and Edges as suggested by the documentation tutorial but the execution time took too long to run (5 hours for importing author data since the file had many records, over 800000). Therefore, they were replaced with create statements since we are guaranteed that the data from the first project did not contain any duplicate records, for example id, username etc. fields are unique. Node creation with create statements took only a matter of seconds for all files.
8. There were many values that were null on the data, such as gender, book prices and nationality etc. The code of mini-project 1 was used to generate fake data for all rows. Finally, for the completed timestamp of the orders the *coalesce(row.completed, datetime()) function was used to assign default timestamp values. The current datetime was used as default.*