Lab 3: pRA: psql-like Interface for Relational Algebra

This assignment should be done in Python using psycopg as the interface to postgresql.
Psycopg helpful links:

- Psycopg docs
- Psycopg tutorial
- PostgreSQL Information Schema
    - Query the tables in this schema, such as information_schema.tables, or information_schema.columns, etc, to get metadata such as relation names, column names etc
- You should have installed postgresql already as explained in lab 1. If you have not set it up already, read up the instructions on Moodle. Here are some extra links for installing postgresql
    - https://www.devart.com/dbforge/postgresql/how-to-install-postgresql-on-linux/
    - https://www.devart.com/dbforge/postgresql/how-to-install-postgresql-on-macos/
- For this lab, it would be beneficial to know the basics of commit and rollback. Refer to the link below for the same
    - https://www.geeksforgeeks.org/difference-between-commit-and-rollback-in-sql/

Part 1 (In Class)

Create a simple relational algebra calculator which has a psql like interface supporting the following commands.

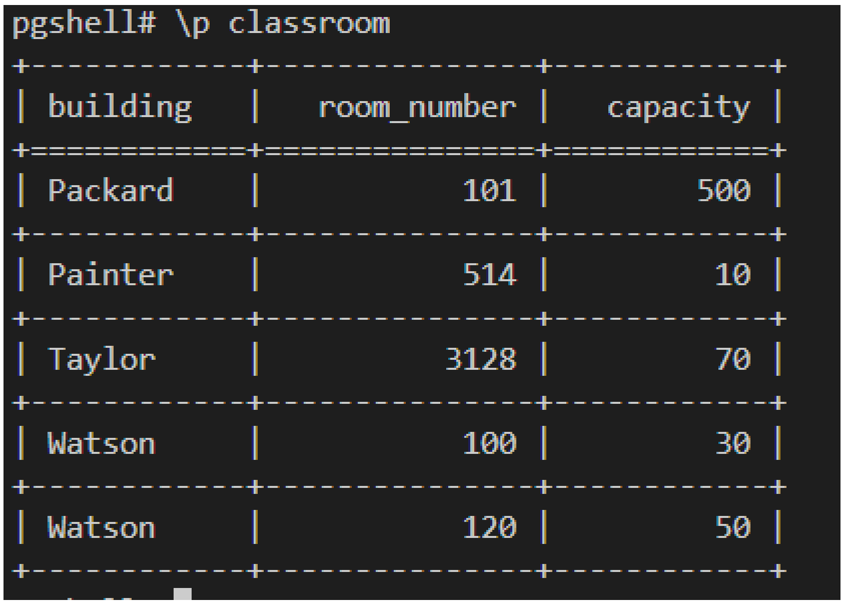
Your code should be in a file lab3-1.py; we have provided a template file, which you need to fill in. Please add your roll number in a comment at the top of the file but don't change the file name.

You will run python lab3-1.py and then enter commands from the following list, which get executed as you enter them.

1. *\connect hostname port database username password*
    a. Connects to postgresql running on *hostname:port* with *database, username and password* as specified.
2. *\ddl filename*
   which loads DDL in SQL format from file *filename*
    a. Create a cursor based on the connection made from previous step
    b. Use execute() method of cursor to pass the ddl file content
    c. Make sure to commit the connection
3. *\data filename*

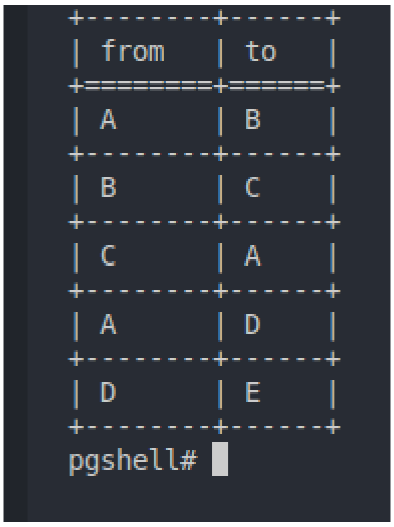
    a. which loads data in SQL format from input file filename, which contains a sequence of INSERT statements

4. = *r sigma [pred] s*
    a. Where *r* and *s* are relation names, and table *r* is created using
       DROP TABLE IF EXISTS r;
       CREATE TABLE r AS SELECT * FROM s WHERE pred
5. = *r pi [cols] s*
    a. Where *cols* is a comma separated list of columns which will be used in the select clause, table *r* is created as result
6. = *r join [pred] s t*
    a. Where *r* is created as *s natural join t WHERE pred*.
       You can assume pred will never be empty, if there is no other predicate use true as the predicate
7. = *r gamma [gbcols] [aggs] s*
    a. Table *r* is created by applying aggregator *aggs* on *gbcols* of s. The result should include all columns in *gbcols* followed by the aggregate operations in *aggs* (e.g. min(A), or count(B)).
    b. Sample input for instructor table: = r gamma [dept_name] [avg(salary)] instructor
        i. If you want to give a name to an aggregate result you can use "avg(salary) as avgsal" for example
8. = *r tc s v*
    a. Relation *s* is a table with columns named from and to, along with perhaps other columns, where each tuple represents an edge. The result is a table with a single column *vertex* which should contain all the nodes reachable from vertex v, including v itself, i.e the transitive closure of the given vertex. Use WITH RECURSIVE to write this query.
    b. The vertex v will be input with quotes, e.g. = r tc edges 'A'
    c. In our test cases we will ensure that 'A' above is present in the "from" attribute of at least one tuple.
9. *\p r*
    a. prints out the data for *r* with column names; You can use a cursor for the SQL query and use psycopg cursor.description field to get the column names. (see https://www.psycopg.org/docs/cursor.html). Then rerun a modified query with ORDER BY on all the column names (e.g. if the columns found were a, b, c, you will ORDER BY r.a, r.b, r.c
    b. Do not print the table name in the column headers
    c. Use the format given in the sample code file. Use tabulate library



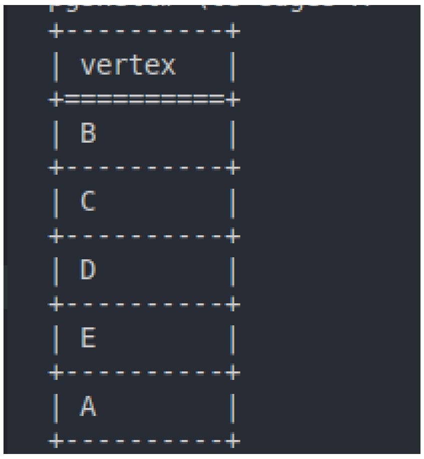
```
pgshell# \p classroom
+------------+---------------+------------+
| building   |   room_number |   capacity |
+============+===============+============+
| Packard    |           101 |        500 |
+------------+---------------+------------+
| Painter    |           514 |         10 |
+------------+---------------+------------+
| Taylor     |          3128 |         70 |
+------------+---------------+------------+
| Watson     |           100 |         30 |
+------------+---------------+------------+
| Watson     |           120 |         50 |
+------------+---------------+------------+
```

Sample Table:



```
+--------+------+
| from   | to   |
+========+======+
| A      | B    |
+--------+------+
| B      | C    |
+--------+------+
| C      | A    |
+--------+------+
| A      | D    |
+--------+------+
| D      | E    |
+--------+------+
pgshell#
```

Sample Input and Output:



```
+----------+
| vertex   |
+==========+
| B        |
+----------+
| C        |
+----------+
| D        |
+----------+
| E        |
+----------+
| A        |
+----------+
```

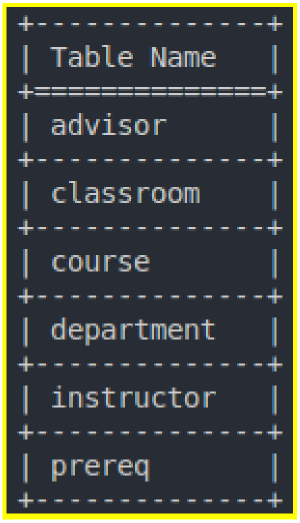
10. \q quit the program
    a. This is already provided in the sample code.

Part 2: Database meta data (Take home):

Add the following further commands which will need postgresql information_schema.
You will submit the file as lab3-2.py, based on the sample file we have provided.
Please add your roll number and name in a comment at the beginning of the file but don't change the name of the file itself.

11. \d
    a. which prints all tables in the default schema (public).. Do not print out anything other than tables. Also do not print the system (pg_) tables; these are in a different schema (pg_catalog) so if you select the schema properly you will not see these tables.
    b. The template file uses print(myRelax.getTables()) where getTables() returns a list; DO NOT USE this, instead write a query to get the tables, one row per table.
    c. Use (a modification of) the printTable() function from Part 1 of the assignment for this to support autograding.   The output should be a table with one table name per row, and the column should be called Table Name.

Sample output:



```
+--------------+
| Table Name   |
+==============+
| advisor      |
+--------------+
| classroom    |
+--------------+
| course       |
+--------------+
| department   |
+--------------+
| instructor   |
+--------------+
| prereq       |
+--------------+
```

12. \clear
    a. Drops all the tables in the default schema (public)
    b. You can output an error message if something is wrong, else don't output anything.

13. \r relation_name
    a. which prints all columns of the relation *r* along with types, nullability, and primary key and foreign keys as illustrated below.
    b. The Length field is the maximum length for characters, and numeric precision for numbers, else 0, computed as below from relevant fields in the columns relation.
    COALESCE(character_maximum_length, numeric_precision, 0)
    c. The column names should be as shown in the sample output, including capitalization
    d. Use WITH clause to subqueries that relate column names with primary key, type, nullability and foreign key, and combine them in the final query using join or outerjoin. For this purpose you can use information_schema.columns, information_schema.table_constraints, and information_schema.key_column_usage
    e. You can use CASE expressions to generate the column values, e.g. YES or NO for Nullable.
    f. Getting the foreign key to matching primary key column mapping is a bit non-trivial. To simplify your life, here's a query that you can use:

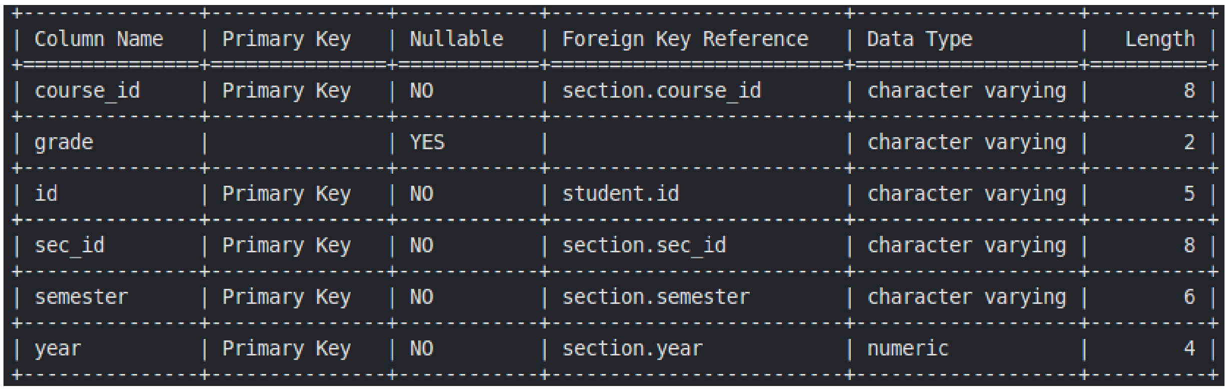
```
select c.constraint_name , x.table_schema as schema_name , x.table_name ,
x.column_name , y.table_schema as foreign_schema_name , y.table_name
as foreign_table_name , y.column_name as foreign_column_name
from information_schema.referential_constraints c
    join information_schema.key_column_usage x
        on x.constraint_name =  c.constraint_name
    join information_schema.key_column_usage y
         on y.ordinal_position = x.position_in_unique_constraint
         and y.constraint_name = c.unique_constraint_name
```

g. You can assume that each column is used in at most 1 foreign key.

h. Use (a minor modification of) the printTable() from the previous part of the assignment to print the final table. It should look like the below.

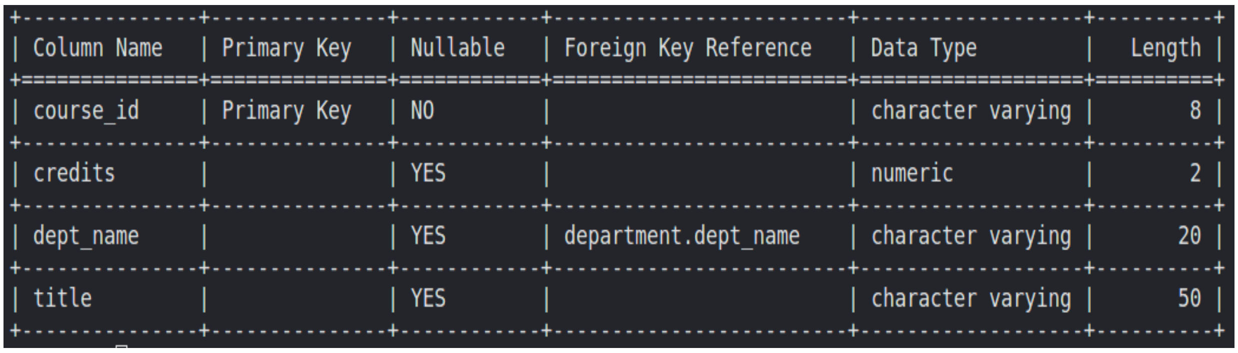
Sample input and output:

\r takes



```
+---------------+---------------+------------+-------------------------+-------------------+----------+
| Column Name   | Primary Key   | Nullable   | Foreign Key Reference   | Data Type         |   Length |
+===============+===============+============+=========================+===================+==========+
| course_id     | Primary Key   | NO         | section.course_id       | character varying |        8 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| grade         |               | YES        |                         | character varying |        2 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| id            | Primary Key   | NO         | student.id              | character varying |        5 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| sec_id        | Primary Key   | NO         | section.sec_id          | character varying |        8 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| semester      | Primary Key   | NO         | section.semester        | character varying |        6 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| year          | Primary Key   | NO         | section.year            | numeric           |        4 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
```

\r course



```
+---------------+---------------+------------+-------------------------+-------------------+----------+
| Column Name   | Primary Key   | Nullable   | Foreign Key Reference   | Data Type         |   Length |
+===============+===============+============+=========================+===================+==========+
| course_id     | Primary Key   | NO         |                         | character varying |        8 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| credits       |               | YES        |                         | numeric           |        2 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| dept_name     |               | YES        | department.dept_name    | character varying |       20 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
| title         |               | YES        |                         | character varying |       50 |
+---------------+---------------+------------+-------------------------+-------------------+----------+
```