387 - Lab 1 - Simulate a DB server in Python

Your task is to write a Python application that will process data in a set of files given to you (you may hardcode these file names -), accept, parse and answer some queries that users supply. The set of files corresponds to data that can potentially be stored in a DB but is stored as CSV files instead and your application effectively behaves like a DB server. File names correspond to relations and column names correspond to headers within the files. Your application should read all of these files on startup and support the following queries where the values are parametrized by user input.

Assume that all keywords (those in bold font) are supplied in lowercase only.

1. **Query type 1:** Retrieve data from a "relation" - each relation's data is captured in a single file. This can take one of 3 forms.
    a. **select** * **from** <relation>; // outputs values of all columns for all rows
    b. **select** * **from** <relation> **where** <column> **=** <value>;
    // support only =. Outputs values of all columns in the relation for a subset of rows defined by the where condition.
    c. **select** <col1>, <col2> …. , <coln> **from** <relation> **where** <colX> **=** <value>;
    // Outputs select columns in the relation for a subset of rows defined by the where condition. colX is a column of the relation but may not correspond to the columns being output.

2. **Query type 2:** Join data from 2 tables
    select * from <relation1>, <relation2> **where** <relation1>.<column1> **=** <relation2>.<column2>;
    // outputs all columns of both relations for those rows in the cross product of the two relations where the condition specified is satisfied.

3. **Query type 3:** Aggregates
    select count from <relation> **where** <column> **=** <value>;
    // outputs an integer value of the count of rows in the relation where the condition is satisfied.

4. **Query type 4:** Set operations (union and intersection only)
    a. (**select** <col1> **from** <relation1> **where** <colX> **=** <value1>) **intersect** (**select** <col1> **from** <relation2> **where** <colY> **=** <value2>)
    b. As above, but replace intersect by **union**

    // outputs the set of all values of <col1> from rows in <relation1> where the condition is satisfied on <relation1> INTERSECTED/UNIONED with the set of values of <col1> from the rows in <relation2> where the condition is satisfied on relation2. Note that <col1> should be the same across both relations.

Formatting your output: For each row, output the columns in the order supplied in the CSV files comma separated with NO additional spaces anywhere. Each row must be on a new line. In the case of Joins output all the columns of the first relation followed by the columns of the second relation in that order.

The Interface: Executing your application

Upon running the python app, it should take the input in the format below in a while loop and return the results . The Query Type = 0 is a signal to exit the application.

```
~$  python3 <rollnumber>-a1.py
Query Type? 1b
Enter your query: select * from student where dept_name = Physics;
output
Query Type? 3
Enter your query: select count from course where credits = 4;
output
.
.
output
Query Type? 0
exiting...
~$
```

Set of data files included

Each file is a csv with a header that corresponds to column names). The name of the file is the name of the relation.

`advisor, classroom, course, department, instructor, prereq, section, student, takes, teaches, time_slot`

Testing your application

We have included test cases for each type of query here. Note that we will be testing with cases other than these (but conforming to the same structure) for grading.

Submission Instructions

Upload a single Python file named **<rollnumber>-a1.py** to Moodle.

Grading Rubric

Query Type	Marks
1a	5
1b	15
1c	15
2	25
3	10
4a	15
4b	15
Total	100