CS 387 - Lab 1

Simulate a Simple DB server in Python

Due on 8th Jan 2022, 11:59 pm

You have been given a set of files that contain some data which could have been stored in a DB, but is provided in csv format. You have to write a python application that loads all of the data into memory, accepts SQL queries and processes the data to answer those queries. This would involve parsing the queries in a way that would help you figure out what data is to be extracted from the CSV files. Your application therefore effectively behaves as 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. **You should only use the csv module in python for this assignment.**

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>;
   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. Also, order of columns being output should be the same as mentioned after the select keyword.

      See note below on <value> being a string.

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. Note that the common column would appear twice in the output.

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:** Group by

```
select sum(<colX>),<column> from <relation> group by <column>;
```

//outputs 2 columns, first having the sum of values in <colX> corresponding to <column> , second having that <column> value for each distinct <column> value present in the original relation. For instance, if there are 2 distinct <column> values X and Y in the <relation>, then there should be 2 rows in the output, first containing sum of values of <colX> corresponding to <column> value X, and second containing the same for <column> value Y. **Note that the order of sum(<colX>) and <column> can be interchanged in the query, in which case, the output order is also expected to be different. Also it can be assumed that the group by query outputs rows lexicographically sorted in increasing order according to <column> value. (2 > 10 in lexicographic order when treated as strings)**

5. **Query type 5:** Subqueries

```
select <col1>, <col2> .... , <coln> from (select * from
<relation> where <colX> = value1) where <colY> = value2;
```

//Outputs select columns but now those columns are sampled from a subset of the original relation. The subset is decided by another subquery which outputs all columns but only those rows that satisfy the inner condition. The outer query also selects only some rows satisfying the outer condition. You are expected to follow the parsing and processing of the inner query first and then the outer query.

Note that the <value> field can be of any data type. If it is a string, it would be enclosed within single quotes (‘). And you can assume that there would not exist additional quotes inside the string. (For instance, a string like **Shaw’s** will never be a part of the <value> field. Also, the query may or may not contain brackets.

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. In cases of selective columns, order of output columns should exactly follow the order provided in the queries. The output should not contain the header corresponding to the columns. **Any deviation from this format will result in the**

test case failing since the queries will be autograded. It can be assumed that all test queries will be syntactically correct and you are not expected to implement error handling.

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 match where season_year = 2011;
output
Query Type? 3
Enter your query: select count from player where country_name = 'India';
output
.
.
output
Query Type? 0
exiting...
~$
```

Query assumptions/clarifications:

- Space problems
  - Spaces can be variable. Below is a sample valid query
    `select  *  from  <relation>  ;`
  - Spaces will be given around keywords except for semi-colon(';'). Semi-colon might not have space before it. 'select' will only have space after it since it's the beginning of the query
    `select * from <relation>;` This is valid
    `select *from<relation>  ;` This is invalid
    So there will be space around all other keywords also

- More queries
  - valid queries

```
select player_id, player_name, dob from player where
country_name = 'India';
select player_id,player_name,dob from player where
country_name = 'India';
```

```
select player_id,        player_name,dob from player
where country_name = 'India';
select     player_id, player_name, dob     from player
where country_name = 'India';
select     player_id       , player_name, dob from
player where country_name = 'India';
```

- invalid queries

  `selectplayer_id, player_name, dob from player where country_name = 'India';` (reason: no space after select)

  `select player_id, player_name, dobfrom player where country_name = 'India';` (reason: no space before from)

- Semi-colon problems
  - Semi-colon may or may not be present in queries(all 4 query-types)

    `select * from <relation>;` This is valid

    `select * from <relation>` This is also valid

- Value problems
  - Strings will be surrounded by quotes and numbers will not be surrounded by quotes, just like how you write in other programming languages (in C++/python)

    `... where country_name = 'India';`

    `... where man_of_match = 125;`

    In python

    ```
    a = 'New      Zealand'
    b = 'New Zealand'
    ```

    a is not equal to b. Similarly here also

    `... country_name = 'New      Zealand'`

    `... country_name = 'New Zealand'`

    are not equal

  - Strings won't be surrounded by double quotes

    `... where country_name = 'India';` This is valid

    `... where country_name = "India";` This is invalid
  - Theoretically speaking below both should give the same answers

    `... where win_margin = 2;`

    `... where win_margin = 2.0000;`

    But we won't be testing this case, so you don't have to support this case for this assignment.

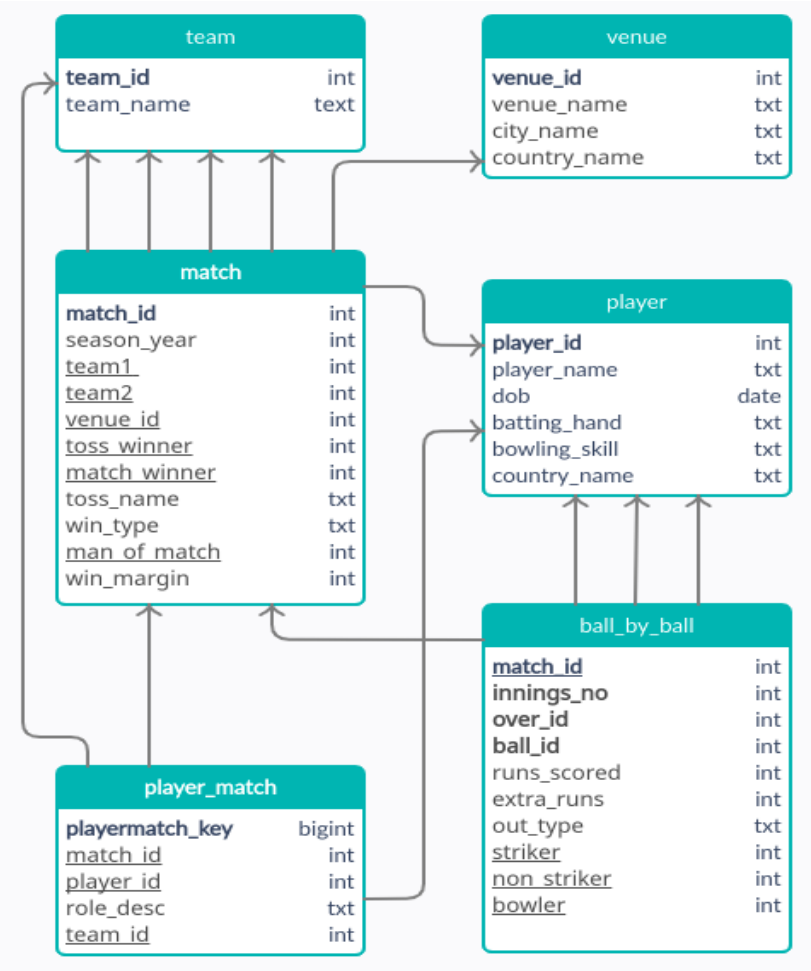
Assume there won't be keywords in the column_names and values and there won't be any spaces, dots, or commas in column_names

P.S: refer strip() function in python

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 : ball_by_ball, player, player_match, team, venue, match

Below schema shows how the data is stored in the csv files.



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. We therefore advise you to test your application on a few test cases of your own.

Your python file will be tested in the below environment. The folder csv_files has all the data files in it.

.

├── **csv-files**

└── **<rollnumber>-a1.py**

Submission Instructions

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

Grading Rubric

Query type	**Marks**
1a	5
1b	7
1c	8
2	15
3	20
4	20
5	25
Total	100