CS516
LAB2-Relational Algebra

09/12/2019

Outline

1. RADB introduction
2. Lab questions
3. RATest (Optional)

1. RADB introduction

What's RADB and how does it work?

- A simple **Relational Algebra (RA) interpreter** written in Python 3
- It implements RA queries by **translating them into SQL** and executing them on the underlying database system through SQLAlchemy.
- RADB is packaged with SQLite, so you can use RADB as a standalone RA database system. Alternatively, you can use RADB as an RA front-end to connect to other database servers from various vendors.

RADB Language Usage -- Selection

Selection: `\select_{`*condition*`}` *input_relation*

For example, to select *Drinker* tuples with name Amy or Ben, we can write:

```
\select_{name='Amy' or name='Ben'} Drinker;
```

String literals should be enclosed in single quotes. Comparison operators `<=`, `<`, `=`, `>`, `>=`, and `<>` (inequality) work as expected on strings, numbers, and dates. For string match you can use the *like* operator; e.g.:

```
\select_{name like 'A%'} Drinker;
```

finds all drinkers whose name start with "A", where `%` is a wildcard character that matches any number of characters. Finally, you can use boolean connectives `and`, `or`, and `not` to construct more complex conditions. More features are available; see Data Types and Operators for details.

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

RADB Language Usage -- Projection

Projection: `\project_{`*attr_list*`}` *input_relation*

Here, *attr_list* is a comma-separated list of expressions that specifies the output attributes. For example, to find out what beers are served by Talk of the Town (but without the price information), you can write:

```
\project_{bar, beer} \select_{bar='Talk of the Town'} Serves;
```

You can also use an expression to compute the value of an output attribute; e.g.:

```
\project_{bar, 'Special Edition '||beer, price+1} Serves;
```

Note that `||` concatenates two strings.

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

RADB Language Usage -- Theta-Join

Theta-Join: *input_relation_1* `\join_{`*cond*`}` *input_relation_2*

For example, to join *Drinker*(*name*, *address*) and *Frequents*(*drinker*, *bar*, *times_a_week*) relations together using drinker name, you can write:

```
Drinker \join_{name=drinker} Frequents;
```

Syntax for *cond* is similar to the case of `\select`.

You can prefix references to attributes with names of the relations that they belong to, which is sometimes useful to avoid confusion (see Relation Schema and Attribute References for more details):

```
Drinker \join_{Drinker.name=Frequents.drinker} Frequents;
```

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

RADB Language Usage -- Natural Join

Natural join: *input_relation_1* `\join` *input_relation_2*

For example, to join *Drinker*(*name*, *address*) and *Frequents*(*drinker*, *bar*, *times_a_week*) relations together using drinker name, we can write `Drinker \join \rename_{name, bar, times_a_week} Frequents;`. Natural join will automatically equate all pairs of identically named attributes from its inputs (in this case, name), and output only one attribute per pair. Here we use `\rename` to create two name attributes for the natural join; see notes on `\rename` below for more details.

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

RADB Language Usage -- Cross Product

Cross product: *input_relation_1* `\cross` *input_relation_2*

For example, to compute the cross product of *Drinker* and *Frequents*, you can write:

```
Drinker \cross Frequents;.
```

In fact, the following two queries are equivalent:

```
\select_{Drinker.name=Frequents.drinker}
  (Drinker \cross Frequents);

Drinker \join_{Drinker.name=Frequents.drinker} Frequents;
```

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

RADB Language Usage -- Set Operations

Set union, difference, and intersection:

input_relation_1 `\union` *input_relation_2*

input_relation_1 `\diff` *input_relation_2*

input_relation_1 `\intersect` *input_relation_2*

For a trivial example, the set union, difference, and intersection between *Drinker* and itself, should return the contents of *Drinker* itself, an empty relation, and again the contents of *Drinker* itself, respectively.

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

RADB Language Usage -- Rename

Rename:

`\rename_{`*new_attr_names*`}` *input_relation*

This form of the rename operator renames the attributes of its input relation to those in *new_attr_names*, a comma-separated list of names.

`\rename_{`*new_rel_name*`: *}` *input_relation*

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

RADB Language Usage -- Aggregation

Aggregation and grouping:

This operator is not in the standard relational algebra. It has two forms:

`\aggr_{`*aggr_attr_list*`}` *input_relation*

This simple form of aggregation computes a single tuple, aggregated over th input relation. Here, *aggr_attr_list* is a comma-separated list of aggregate e involving functions such as `sum`, `count`, `avg`, `min`, and `max`. For example:

```
\aggr_{sum(price), avg(price)} Serves;
```

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

`\aggr_{`*group_by_attrs*`:` *aggr_attr_list*`}` *input_relation*

With this form, the input relation is first partitioned into groups, according to the attributes listed in *group_by_attrs*: all tuples that agree on the values of *group_by_attrs* go into the same group. Then, for each group, one output tuple is produced: it will have the values for *group_by_attrs* (which are shared by all group members), followed by the values of aggregate expressions in *aggr_attr_list*. For example, the following query finds, for each beer, its average price and number of bars serving it:

```
\aggr_{beer: avg(price), count(1)} Serves;
```

RADB Language Documentation

To find more details about this language, and how to use radb, please find this link:

https://users.cs.duke.edu/~junyang/radb/

(RADB is an in-house Duke product developed by Prof. Jun Yang!)

2. Lab Questions

Lab Questions

TABLE SCHEMAS

- drinker(name, address)
- bar(name, address)
- beer(name, brewer)
- frequents(drinker, bar, times_a_week)
- likes(drinker, beer)
- serves(bar, beer, price)

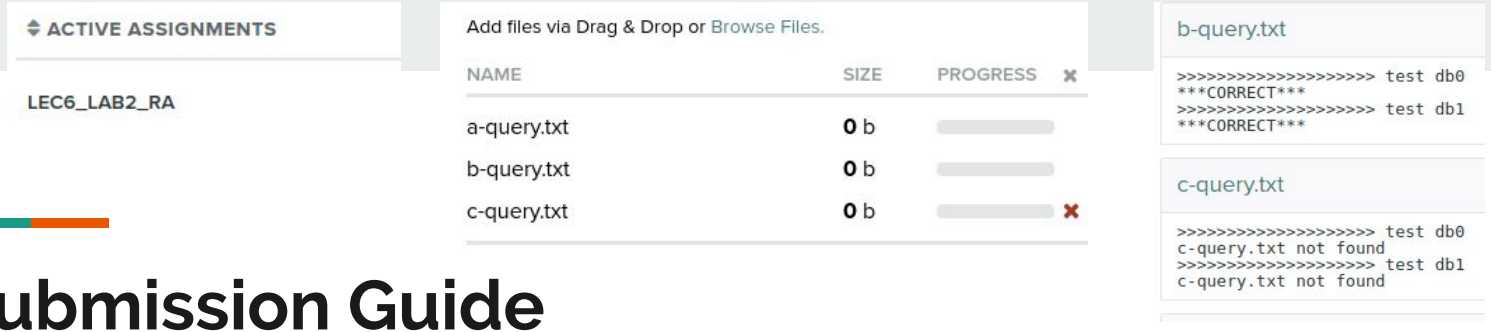
Lab Questions Cont'd

(a) Find names of all bars that Eve frequents
(b) Find names and addresses of drinkers who frequents Satisfaction more than once per week
(c) Find names of bars serving some beer Amy likes for strictly less than $2.75
(d) Find names of all drinkers who like Corona but not Budweiser
(e) For each beer that Eve likes, find the names of bars that serve it at the highest price. Format your output as list of (beer, bar) pairs

NOTE: If the format of your solution is (bar, beer), the autograder will consider it incorrect.

Points

- 20 points for each question. 100 points in total.
  - 10 points for any submission within 24 hours
  - + 10 points for a correct submission within 24 hours
- Extra 10 points for **all** correct submissions within the class



Submission Guide

(a) Once you get a working solution, record your query in a **plain text file** for each part. Take part (a) for example. You should write your query in a plain text file named **"a-query.txt"**(replace "a" with "b", "c", and other parts as appropriate).

(b) Thus, for the whole question, you are gonna submit 5 files in total. They are: "a-query.txt", "b-query.txt", "c-query.txt", "d-query.txt" and "e-query.txt".

(c) Remember to (re)-submit **all your query files at the same time** on Gradescope (Assignment "LEC6_LAB2_RA").

(d) Even though it's an "auto"grader, we are still manually scoring the problem. Hence the autograder component of the score is always set to 0 (which doesn't mean you got nothing right).

How to test your answers: Three options

1. Submit to gradescope. If it is wrong, it will tell you it is wrong. However, it if says true, it doesn't necessarily guarantee that your answer is true
    - your answer may match the correct answer even by a wrong query!
2. You can install RADB (follow the instructions from the RADB link), and run queries there. (optional)
3. Use RATest to debug your queries! (optional)

3. RATest

(optional use)

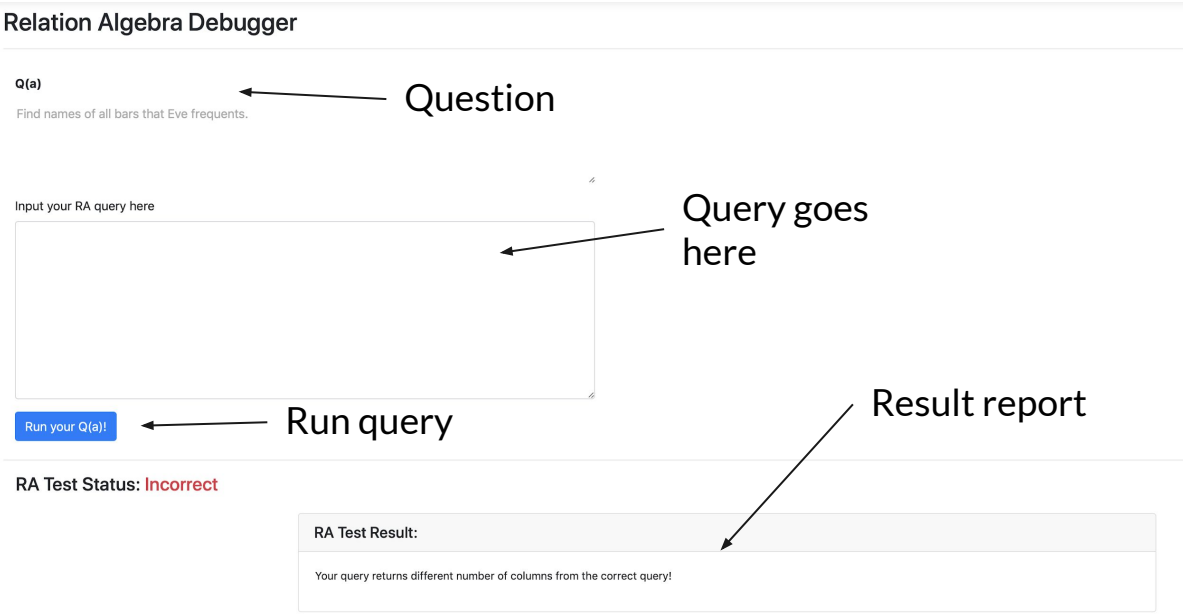
What is RATest

- A tool to help you debug your RA query!
- It will provide you a “small counterexample” if your result does not match the correct result - the counterexample should help you fix the query
    - CAUTION: If your query is incorrect, but the answer somehow matches the correct answer, it won’t be able to catch such errors
- No environment setup required!
- Link: https://ratest516.cs.duke.edu

RATest Consent

- RATest is a research tool that is still under development. Before you start using this debug tool, we need you to accept a **consent form**, so that we can use your anonymized data to evaluate and improve our tool.
- Note that the use of RATest is completely optional, does not affect your grade anyway, and we will completely anonymize the data for analysis purposes
- Alternatively, you can only use the autograder on Gradescope, which will give you the same answer on whether your query is correct/wrong

RATest Interface

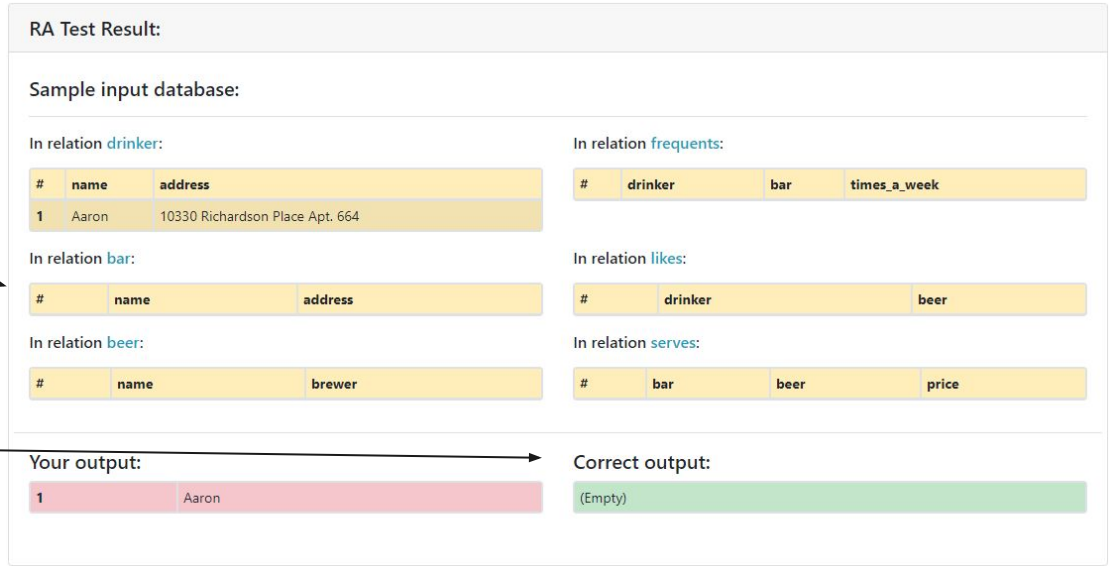


RATest Interface - Explaining wrong queries by a small example

A small database instance

Your Output

Correct Output



RA Test Result:

Sample input database:

In relation drinker:

#	name	address
1	Aaron	10330 Richardson Place Apt. 664

In relation frequents:

#	drinker	bar	times_a_week

In relation bar:

#	name	address

In relation likes:

#	drinker	beer

In relation beer:

#	name	brewer

In relation serves:

#	bar	beer	price

Your output:

1	Aaron

Correct output:

(Empty)

Want to know more about RATest?

We have recent research and demonstration papers in SIGMOD 2019 led by Duke database group PhD student Zhengjie Miao:

- Research paper
- Demonstration paper

Want to join the RATest team? We will have class project ideas based on RATest!