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 ($\sigma$)

**Selection:** $\text{select\_}\{\text{condition}\} \text{ input\_relation}$

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

```plaintext
\text{select\_}\{\text{name}='Amy' \text{ or name}='Ben'\} \text{ Drinker;}
```

String literals should be enclosed in single quotes. Comparison operators $\leq$, $<$, $=$, $>$, $\geq$, and $\neq$ (inequality) work as expected on strings, numbers, and dates. For string match you can use the \textit{like} operator; e.g.:

```plaintext
\text{select\_}\{\text{name} \text{ like} 'A%'\} \text{ 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 \texttt{and}, \texttt{or}, and \texttt{not} to construct more complex conditions. More features are available; see Data Types and Operators for details.

**TABLE SCHEMAS**

- \texttt{drinker(name, address)}
- \texttt{bar(name, address)}
- \texttt{beer(name, brewer)}
- \texttt{frequents(drinker, bar, times\_a\_week)}
- \texttt{likes(drinker, beer)}
- \texttt{serves(bar, beer, price)}
RAKB Language Usage -- Projection (Π)

**Projection**: `\text{project\_}{attr\_list} \text{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:

```sql
\text{project\_}{\text{bar, beer}} \text{select\_}{\text{bar='Talk of the Town'}} \text{Serves};
```

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

```sql
\text{project\_}{\text{bar, 'Special Edition' || beer, price+1}} \text{Serves};
```

Note that `||` concatenates two strings.
**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)`
**RAADB 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:** \(input\_relation\_1 \times input\_relation\_2\)

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

\[\text{Drinker} \times \text{Frequents};\]

In fact, the following two queries are equivalent:

\[\text{\select\_}\{\text{Drinker}\.name=\text{Frequents}\.drinker\}\]
\[
\text{(Drinker} \times \text{Frequents});
\]
\[\text{Drinker} \text{\join\_}\{\text{Drinker}\.name=\text{Frequents}\.drinker\} \text{Frequents};\]

**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:

- \[ \text{input	herelation}_1 \cup \text{input	herelation}_2 \]
- \[ \text{input	herelation}_1 \setminus \text{input	herelation}_2 \]
- \[ \text{input	herelation}_1 \cap \text{input	herelation}_2 \]

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\((\text{name, address})\)
- bar\((\text{name, address})\)
- beer\((\text{name, brewer})\)
- frequents\((\text{drinker, bar, times\_a\_week})\)
- likes\((\text{drinker, beer})\)
- serves\((\text{bar, beer, price})\)
RA DB Language Usage -- Rename \( (p) \)

**Rename:**

\[
\text{\texttt{rename\{new\_attr\_names\} input\_relation}}
\]

This form of the rename operator renames the attributes of its input relation to those in \( \text{new\_attr\_names} \), a comma-separated list of names.

\[
\text{\texttt{rename\{new\_rel\_name: \_*\} input\_relation}}
\]

**TABLE SCHEMAS**

- \( \text{drinker(name, address)} \)
- \( \text{bar(name, address)} \)
- \( \text{beer(name, brewer)} \)
- \( \text{frequents(drinker, bar, times\_a\_week)} \)
- \( \text{likes(drinker, beer)} \)
- \( \text{serves(bar, beer, price)} \)
RA DB Language Usage -- Aggregation

Aggregation and grouping:

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

\texttt{aggr\{aggr\_attr\_list\} input\_relation}

This simple form of aggregation computes a single tuple, aggregated over the input relation. Here, \texttt{aggr\_attr\_list} is a comma-separated list of aggregate involving functions such as \texttt{sum}, \texttt{count}, \texttt{avg}, \texttt{min}, and \texttt{max}. For example:

\texttt{aggr\{sum(price), avg(price)\} Serves;}

TABLE SCHEMAS

- \texttt{drinker(name, address)}
- \texttt{bar(name, address)}
- \texttt{beer(name, brewer)}
- \texttt{frequents(drinker, bar, times\_a\_week)}
- \texttt{likes(drinker, beer)}
- \texttt{serves(bar, beer, price)}

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

\texttt{aggr\(beer: avg\{price\}, count\{\}\) 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!)
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!
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
Relation Algebra Debugger

Q(a)
Find names of all bars that Eve frequents.

Input your RA query here

Run your Q(a)

RA Test Status: Incorrect
RA Test Result:
Your query returns different number of columns from the correct query!
RATest Interface - Explaining wrong queries by a small example

A small database instance

Your Output

Correct Output

Sample input database:

In relation `drinker`:

<table>
<thead>
<tr>
<th>#</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aaron</td>
<td>10330 Richardson Place Apt. 664</td>
</tr>
</tbody>
</table>

In relation `frequents`:

<table>
<thead>
<tr>
<th>#</th>
<th>drinker</th>
<th>bar</th>
<th>times_a_week</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In relation `bar`:

<table>
<thead>
<tr>
<th>#</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In relation `likes`:

<table>
<thead>
<tr>
<th>#</th>
<th>drinker</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In relation `serves`:

<table>
<thead>
<tr>
<th>#</th>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Your output:

<table>
<thead>
<tr>
<th>#</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aaron</td>
</tr>
</tbody>
</table>

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!