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.

Basic Usage

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*

This form of the rename operator gives a new relation name to its input relation (the attribute names remain the same). For example:

```
\rename_{s1:*} Serves
  \join_{s1.beer=s2.beer and s1.price>s2.price}
\rename_{s2:*} Serves;
```

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

This form of the rename operator allows you to rename both the input relation as well as its attributes.

TABLE SCHEMAS

- drinker(<u>name</u>, address)
- bar(<u>name</u>, address)
- beer(<u>name</u>, brewer)
- frequents(<u>drinker</u>, <u>bar</u>, times_a_week)
- likes(<u>drinker</u>, <u>beer</u>)
- serves(<u>bar</u>, <u>beer</u>, price)

\rename_{drinker, address} drinker

\rename_{d1: drinker, address} drinker

More useful tips

Nested Queries

- Build a complex query by nesting: you can feed a subquery as an input relation to another relational operator (using parentheses to enclose the subquery as necessary to avoid ambiguity) , e.g.: `\select_{`*condition*`} (\project_{`*attr_list*`}` *input_relation_1* `)\join` *input_relation_2* `;`

View

Views

RA lets you define “views,” which may be thought of as temporary, relation-valued variables holding the results of relational algebra expressions. To define a view, use the syntax:

view_name `:-` *view_definition_query*`;`

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!)