Databases
Lab 01: A 'gentle' Introduction to Relational Algebra.

Andrés Oswaldo Calderón Romero, Ph.D.

February 16, 2026

1 Introduction

Welcome to the first laboratory session of the Databases course. This session serves as a practical bridge between theoretical relational algebra and its application in modern database environments. Through this lab, you will practice formulating relational algebra expressions to manipulate and query data effectively.

1.1 Learning Objectives

The primary goals of this laboratory session are to ensure you can:

- **Synthesize Algebraic Queries:** Master the syntax and logic required to implement fundamental operations such as Selection (σ), Projection (Π), Cartesian Product ($\times$), and Joins ($\bowtie$).
- **Software Proficiency:** Gain hands-on experience installing and navigating the `radb` environment using Python.
- **Data Model Analysis:** Interpret and query diverse database schemas, including the Applications, Beer, and Pizza example datasets.
- **Problem Solving:** Translate complex natural language queries into executable relational algebra code to derive specific data.

Go ahead!

2 `radb` Tutorial

The `radb`[1] software is a specialized relational algebra interpreter designed to bridge the gap between theoretical database concepts and practical implementation. Operating within a Python environment, it provides a command-line interface where users can execute algebraic expressions directly against sample databases, such as `.db` files. By supporting fundamental operations like selection (σ), projection (Π), and various joins ($\bowtie$), `radb` allows for the real-time evaluation of queries without the immediate need for complex SQL syntax, making it an essential tool for mastering data manipulation logic. Please, have a look at `radb` documentation page to have more information (see figure 1).

[1] https://users.cs.duke.edu/ junyang/radb

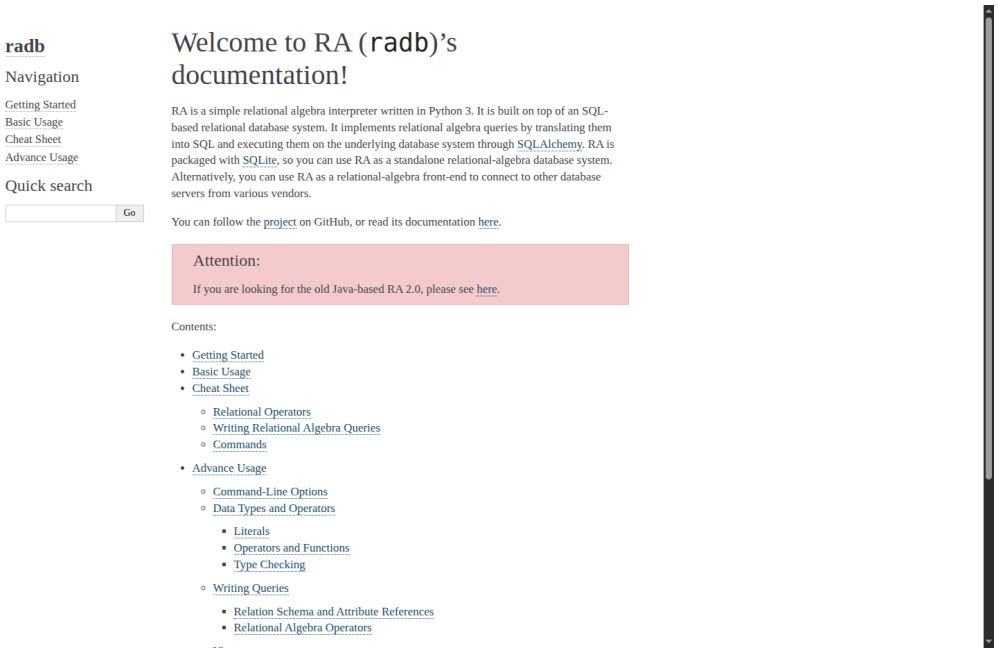

Figure 1: The official `radb` documentation homepage, providing a comprehensive reference for relational operations, software commands, and installation guidelines.

2.1 Verify Python Installation

Ensure that Python 3.5 or a later version is installed on your system. You can check your Python version by running:

```
python --version
```

If Python 3 is not the default version on your system, you may need to use `python3` instead.

```
python3 --version
```

If your system does not have Python installed, we recommend following this tutorial to install Miniconda, a minimal yet powerful Python distribution.

2.2 Install `radb`

Use `pip` to install the `radb` package. If `pip` is not associated with Python 3 on your system, use `pip3` instead.

```
pip install radb
```

Or, if necessary:

```
pip3 install radb
```

3 Set Up a Sample Database

To practice using `radb`, set up a sample database as follows:

- Download the `applications.db` file, which contains the sample database.
- Run the following command in a terminal console to start the `radb` interpreter with the sample database `applications.db`.

  First, navigate to the directory where the database is located (set the full path accordingly with your system):

  ```
  cd /full/path/to/applications.db
  ```

  Then, run the following command:

  ```
  radb applications.db
  ```

If your system cannot find the `radb` command, it might be installed in a directory that is not included in your system's `PATH`. On Linux, for example, it may be located in `~/.local/bin/`. You can run it by specifying the full path:

```
~/.local/bin/radb applications.db
```

Alternatively, you can add the directory to your `PATH` to avoid typing the full path each time. To find out where `pip` installed `radb`, run:

```
pip show -f radb
```

Upon successful initialization, the `ra>` prompt will appear at the beginning of your terminal line. This indicates that the interpreter is connected to the database and is ready to evaluate your relational algebra expressions.

4 Run `radb` commands on the Sample Database

At the `ra>` prompt, you can execute commands such as:

- List all relations in the database. If everything is running correctly, you should see a list of three relations.

  ```
  \list;
  ```

- Exit the `radb` interpreter:

  ```
  \quit;
  ```

For more detailed information and additional commands, refer to the official `radb` documentation section **Advanced Use** at: https://users.cs.duke.edu/~junyang/radb/advance.html.

5 Practice Relational Algebra in a Sample Database

5.1 The Applications Database

The `applications.db` database is extremely simple and will help us understand the format of the relational algebra expressions covered in class, as well as how to use them in the `radb` software for practice.

The `applications.db` database contains dummy data about a group of students applying to different colleges or universities. It consists of only three relations and models the following attributes:

- student(sID, sName, GPA, sizeHS)
- college(cName, state, enrollment)
- apply(sID, cName, major, decision)

5.2 Relational algebra operations

5.2.1 Simplest relational algebra expresion

Let's list the content of the database. Go back to the `radb` interpreter, and when you see the `ra>` prompt again, type the name of each relation and see what happens.

```
student;
```

```
college;
```

```
apply;
```

How many tuples does each relation have?

5.2.2 Selection (σ)

The selection operation retrieves tuples from a relation that satisfy a specified condition.

Syntax:

```
\select_{condition} input_relation;
```

Examples:

- Students with GPA greater than 4.0.
- $\sigma_{GPA>4.0}$ *student*.

```
\select_{GPA > 4.0} student;
```

- Students with GPA greater than 4.0 and the size of their high school greater than 2000.

- $\sigma_{GAP>4.0 \wedge sizeHS>2000}\ student$.

```
\select_{GPA > 4.0 and sizeHS > 2000} student;
```

- Applications to the program of 'Sistemas' at the 'PUJ'.
- $\sigma_{major=Sistemas \wedge cName=PUJ}\ apply$.

```
\select_{major = 'Sistemas' and cName = 'PUJ'} apply;
```

5.2.3 Projection (Π)

Projection creates a new relation by selecting specific attributes from an existing relation.

Syntax:

```
\project_{attr_list} input_relation;
```

Examples:

- Student ID and decision for all applications.
- $\Pi_{sID,decision}\ apply$.

```
\project_{sID, decision} apply;
```

- ID and name of students with GPA greater than 3.5 (it is a composed query).
- $\Pi_{sID,sName}\,(\sigma_{GPA>3.5}\ student)$.

```
\project_{sID, sName}( \select_{GPA > 3.5} student );
```

5.2.4 Cartesian Product ($\times$)

Perform Cartesian product over the tuples of two relations.

Syntax:

```
input_relation_1 \cross input_relation_2;
```

Examples:

- Find the Cartesian product between the students and the applications.
- $student\ \times\ apply$.

```
student \cross apply;
```

- Name and GPA of students which applied to 'Sistemas'.
- $\Pi_{sName,GPA}\ (\sigma_{student.sID=apply.sID \wedge major=Sistemas}\ (student\ \times\ apply))$.

```
\project_{sName, GPA}( \select_{student.sID = apply.sID and major = 'Sistemas'}( student \cross apply ) );
```

5.2.5 Natural Join ($\bowtie$)

Natural join will automatically equate all pairs of identically named attributes from its inputs.

Syntax:

```
input_relation_1 \join input_relation_2;
```

Examples:

- Name and GPA of students which applied to 'Sistemas'.
- $\Pi_{sName,GPA}\ (\sigma_{major=Sistemas}\ (student\ \bowtie\ apply))$.

```
\project_{sName, GPA}( \select_{major = 'Sistemas'}( student \join apply ) );
```

- Name, GPA, and college of students which applied to 'Medicina' at institutions with enrollment greater than 20000.
- $\Pi_{sName,GPA,cName}\ (\sigma_{major=Medicina \wedge enrollment>20000}\ (student\ \bowtie\ apply\ \bowtie\ college))$.

```
\project_{sName, GPA, cName}(
  \select_{major = 'Medicina' and enrollment > 20000}(
    student \join ( apply \join college )
  )
);
```

Note that you can also use a multi-line format to write relational expressions. The expression will be evaluated only when the ; is entered.

5.2.6 Theta Join ($\bowtie_\theta$)

Theta join will equate pairs of attributes from its inputs based on a particular condition. It is particularly useful when the attributes in the relations to be joined have different names.

Syntax:

```
input_relation_1 \join_{cond} input_relation_2;
```

6 Independent Work

6.1 Beer Database

The `beer.db` database models drinkers, bars, beers, and drinking preferences among a group of people. The data model of the database is as follows:

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

You will use this database to practice additional relational algebra queries by following the Basic Use reference available in the `radb` documentation (https://users.cs.duke.edu/~junyang/radb/basic.html).

Your task is to connect to the `beer.db` database in the same way as with `applications.db` and follow the guide by running all the examples provided. Pay particular attention to the last three operations: Set union, difference, and intersection; Rename; and Aggregation and Grouping.

For each expression in the guide (those with a gray background), you will capture the results and present them in a well-formatted report, along with their corresponding relational algebra expressions.

6.2 Pizza Database

The `pizza.db` database has the following data model:

- Person (name, age, gender)
- Frequents (name, pizzeria)
- Eats (name, pizza)
- Serves (pizzeria, pizza, price)

You will use the `pizza.db` database to answer the following queries. For each query, you must provide the corresponding relational algebra expression and the `radb` code, along with your response.

1. Find the unique person who does not frequent pizzerias that sell supreme pizza.
2. Find the pizzerias and pizzas that are more expensive than those sold by Little Caesars.
3. How old is the only woman who eats pizza that costs less than $7.75?
4. What is the most popular pizzeria among people between 20 and 30 years old?
5. What is the average age of men who eat pepperoni pizza?

Finally, you will propose your own query for the `pizza.db` database (just one). Based on the queries above, you will create a new query from scratch, write its corresponding relational algebra expression, and provide the `radb` code to answer it.

We expect you to submit a well-structured report in PDF format, containing the requested information outlined in Sections 6.1 and 6.2. The submission deadline is **February 26, 2026** before **11:00am**.

Happy Hacking 😎!