MDS miniHive
Winter Term 2024/25
Lab 2

1 Practicing Relational Algebra with RADB

In this lab session, we practice writing queries in relational algebra. We familiarize ourselves with the specific RADB syntax, a simple relational algebra interpreter. We will rely on this syntax when building miniHive.

1.1 Getting Started

RADB is available in the miniHive Docker container, in the `radb` directory:

```
minihive@container$ cd radb
```

This directory contains the "pizza" scenario by Jennifer Widom. Load the data...

```
minihive@container:~/radb$ radb -i pizza.ra pizza.db
```

...and start the RADB console:

```
minihive@container:~/radb$ radb pizza.db
```

The RADB console wraps SQLite, so SQLite commands are available. To list all available relations, write:

```
radb: A relational algebra interpreter over relational databases
Version 3.0.4 by Jun Yang <junyang@cs.duke.edu>
https://github.com/junyang/radb
ra>\list;
databaserelations:
  Eats(name:string, pizza:string)
  Frequents(name:string, pizzeria:string)
  Person(name:string, age:number, gender:string)
  Serves(pizzeria:string, pizza:string, price:number)
```

To list all tuples in relation `Person`, to only list persons older than 30, and to finally quit the console, write:

```
ra> Person;
(name:string, age:number, gender:string)
----------------------------------------
Amy, 16, female
Ben, 21, male
Cal, 33, male
Dan, 13, male
Eli, 45, male
Fay, 21, female
Gus, 24, male
Hil, 30, female
Ian, 18, male
----------------------------------------
9 tuples returned
ra>
ra> \select_{age>30} Person;
(name:string, age:number, gender:string)
------------------------------------------------------------------------
Cal, 33, male
Eli, 45, male
------------------------------------------------------------------------
2 tuples returned
ra>
ra> \quit;
```

Observe that the RADB syntax mimics the syntax familiar from LaTeX math formulas, writing "`\select_{\phi}`" for the selection operator σ_ϕ.

1.2 Writing RADB Queries

For the following tasks, you are asked to write queries in relational algebra using RADB syntax, and evaluate them against the pizza scenario. The documentation and syntax of RADB can be found here: `https://users.cs.duke.edu/~junyang/radb/`.

1. Ben plans a date with Amy. To which pizzeria can he take her? It should be a pizzeria that she already frequents.
2. Ben considers inviting Amy to Pizza Hut. He wants to know how much money to bring with him. How much do pizzas cost there? List the prices so that Ben can form an idea.
3. Is there a pizzeria that Ben and Amy both frequent? What is its name?
4. Find all pizzas eaten by at least one female over the age of 20.

   Hint: 3 tuples are returned.

5. Find the names of all females who eat at least one pizza served by Straw Hat.

   Note: The pizza need not be eaten at Straw Hat.

   Hint: 2 tuples are returned.

6. Find all pizzerias that serve at least one pizza for less than $10 that either Amy or Fay (or both) eat.

   Hint: 3 tuples are returned.

7. Find the age of the oldest person (or persons) who eats mushroom pizza.

   Hint: This is easiest to do using an aggregation function, but aggregation is not required.

2 Registration with the Praktomat Submission System

We manage submissions for miniHive milestones using the online submission system Praktomat, available here: `https://praktomat-cu.sdbs.fim.uni-passau.de/`. Let's make sure everybody can log on.

2.1 Logging On

Self-registration in Praktomat is not possible. If you are enrolled in the course, an account should have been created for you. For the first login, reset your password, using the e-mail address you used to enroll in the course. Upon login, you are asked to confirm to the Data Privacy Statement.

If you do not receive an e-mail to reset your password within a few minutes, contact technical support: `praktomat-sdbs@uni-passau.de`.

2.2 Uploading a Dummy Submission for Milestone 1

Create an non-empty file named `sql2ra.py` (it can just contain a comment) and upload it as a dummy submission for the first miniHive milestone. The system will run unit tests and should inform you that all tests have failed (as may be expected).

3 Interacting with Luigi

Luigi is a Python-based workflow management system that is used to build complex pipelines of tasks. It helps define tasks, dependencies between tasks, and ensures that tasks are

executed in the correct order, handling issues like failure recovery and reruns. We make use of Luigi in the miniHive project.

Luigi supports different modes of execution:

- Local: Data is loaded from local disk and results are also stored there.
- HDFS: Data is loaded from HDFS and results are also stored there.
- Mock: Data is mocked, i.e. hard-coded dummy data is embedded in the script and used. This mode is used specifically for unit testing.

In the following, different variants for luigi are shown, using the example of the WordCount problem:

```
minihive@container$ cd python-luigi/
```

Inspect the Python implementation of WordCount, the unit test file, and the scripts for invoking `luigi`. Run the scripts and explore their behavior.

```
minihive@container:~/python-luigi$ ./run-local # Local
minihive@container:~/python-luigi$ ./run-hdfs # HDFS
minihive@container:~/python-luigi$ ./run-test # Mock
```

4 Interacting with the miniHive Docker Container

We provide a guided tour of the miniHive Docker container. We refer to our GitHub page at `https://github.com/sdbs-uni-p/minihive-docker` for details.

4.1 Exploring the Directory Structure

Inside the container, the command

```
minihive@container$ ls -l
```

lists directories of the miniHive Docker. The directories `hadoop`, `hive`, `spark`, and `radb` contain sample data and example applications to run on each system. The `tpch-hive` directory contains the TPC-H benchmark that can be run on Hive.

4.2 Developing Inside the Container vs. Outside the Container

You can either develop the miniHive milestones within the Docker container or on your host system, and merely use the container for testing.

If you choose to develop locally, make sure to use *Python 3.10.0*. Also, make sure that you use the same versions of all Python packages as in the miniHive container. You may inspect packages and their versions calling `minihive@container$ pip freeze`.

4.3 Sharing a Directory between Host and Container

To synchronize files on your host machine with the Docker container, you can use directory volumes which map a host directory inside the container.

You may have to stop and remove a previously set up miniHive Docker container to map a directory. This can be done by executing:

```
user@host$ docker stop minihive && docker rm minihive
```

1. Start the miniHive Docker container with a mapping between a directory on your host and the container:

   ```
   user@host$ docker run -d --name minihive -p 2222:22 \
   -v /home/user/minihive:/home/minihive/minihive minihive-docker
   ```

   We assume that the host directory `/home/user/minihive` is mapped to the container (this directory may differ on your system).

2. Create a Python file called `example.py` inside `/home/user/minihive` (on your host machine) with the following content:

   ```
   def hello_world():
       return "Hello World"

   if __name__ == "__main__":
       print(hello_world())
   ```

   Watch out for whitespaces and indentation during copy-pasting.

3. Execute the Python file locally to make sure that it works as expected:

   ```
   user@host$ python example.py
   ```

   Note: You may have to use `python3` instead of `python`, depending on your environment.

   If successful, the program should write "Hello World" to standard output.

4. Connect to the container:

   ```
   user@host$ ssh -p 2222 minihive@localhost
   ```

5. Change directory:

   ```
   minihive@container$ cd ~/minihive
   ```

6. Check whether the Python file is present:

   `minihive@container:~/minihive$ ls -l`

   It should be listed as a file in this directory.

7. Execute the Python file from inside the container:

   `minihive@container:~/minihive$ python example.py`

   Again, the program should write "Hello World" to standard output.

4.4 Transferring Files via `scp`

To transfer files from your host to the Docker container, you can also use `scp`.

We provide further instructions at at `https://github.com/sdbs-uni-p/minihive-docker/blob/main/README.md#copy-files-tofrom-your-local-machine`.

4.5 Running Unit Tests

We describe the process of testing the code you have developed on your host machine. You will copy your code into the miniHive Docker container and run tests within the container environment.

1. In your shared host directory (e.g. `/home/user/minihive`), create a file `test_example.py` with the following content:

```
import unittest
import example

'''
Unit tests for the 'example' module, focusing on 'hello_world()'
to familiarize with milestone testing.
'''
class ExampleTests:

    def test_example(self):
        self._check("Hello World", example.hello_world())

class TestExample(unittest.TestCase, ExampleTests):

    def _check(self, expected, current):
        self.assertEqual(expected, current)

if __name__ == '__main__':
    unittest.main()
```

2. Switch to the miniHive directory (inside the container):

   ```
   minihive@container$ cd ~/minihive
   ```

3. Execute the unit test:

   ```
   minihive@container:~/minihive$ python test_example.py
   ```

   You should see a message indicating that one test has been successfully run.