CS44800 Project 1
Disk vs In-Memory vs Database
Spring 2024

Total Points: **3 points**

Learning Objectives

1. Appreciate the usefulness of database technologies.
2. Understand the tradeoffs between file-systems, main-memory, and database systems.
3. Understand how to use an embedded database.

Project Description

The goal for the project is to create a simple database application that supports query operation over a relation. You are given a dataset that represents people from the movie industry. The data has the following schema:

- **nconst** (id or key of the person)
- **primaryName** (name)
- **birthYear** (birth year in YYYY format)
- **deathYear** (death year in YYYY format)
- **primaryProfession** (primary roles or professions separated by comma)
- **knownForTitles** (the titles or movies for which the person is known for separated by comma)

The application will perform simple queries over the data using three different backends and will print results for each query and the time it takes to execute the query. The queries will ask for a given ID and the application should report **specific columns of that record** for that ID **separated by tabs**.

Basically, you will mimic the functionality of the following SQL query using three different backends -

SELECT COLUMN_NAMES FROM TABLE_NAME

Example:

> Query: nm0000001
>
> Column Names: {primaryName, deathYear, knownForTitles}
>
> Result: Fred Astaire 1987 tt0072308,tt0053137,tt0043044,tt0050419

There should only be one result for each query as the ID should be unique. Column names will always be a non-empty array and possible values contains values from headers of the columns.

The application should include five methods: `load_mainmemory()`, `load_mapdb()`, `select_file()`, `select_mainmemory()` and `select_mapdb()`. It would also include two summary methods: `fastestLoad()` and `fastestSelect()`. See the specific descriptions of the different parts of the project below for more details.

- The `load_x()` methods will load a dataset into the appropriate backend. This method should only be available for the Part 2 and 3 of the Project.
- The `select_x()` methods will retrieve information from the dataset and will return the result of the query.
- The `fastestx()` methods will return the indices of the fastest backend for the Load and Select operation. In the skeleton file, the indices are commented out for your ease. They are as follows: **0=file system, 1=main memory and 2=MapDB.**
  So if MapDB has the fastest load time, you will return 2 for fastestLoad().

Implementation Guidelines

For the project you have been provided with three TSV files and the skeleton of a Java Project.

1. The TSV files are based from the name.basics dataset available online from IMDB (https://datasets.imdbws.com/):
   - data_test.tsv
   - data_10MB.tsv
   - data_30MB.tsv

   All TSV files have the same structure described above, which also includes a header row. You should not include this header row in the results or in loading your data. The difference between the files is the amount of records: **data_test.tsv** has 10 records, **data_10MB.tsv** is roughly 10MB, while **data_30MB.tsv** is roughly 30MB

2. Skeleton code for the java project will be included to help you with your implementation. **You only need to modify the class: project1.java.**
3. You will control the application by specifying three arguments in the command line. The first argument is the path to the file. The second argument is an integer that specifies which backend to use: **0=file system, 1=main memory and 2=MapDB**. The third argument is the ID of the row you want to retrieve.

Configuring Maven

Part 3 of this project relies on third-party libraries that are managed by Maven. We have made Maven available on the university machines. In order to use Maven, you will have to set the Path variable through the terminal as follows:

```
export PATH=/homes/afiroze/cs448/apache-maven-3.6.0/bin/:$PATH
```

Building with Maven

In the terminal, navigate to your application directory (the one with the pom.xml file). Input the following command:

```
mvn clean compile assembly:single
```

Maven should now build your project. After the build finishes, you will find the compiled class files in the `target` directory. You can run your project with the following command:

```
java -cp target/cs448p1-1.0-SNAPSHOT-jar-with-dependencies.jar
-Xmx128m cs448.App <data_file> <backend> <id>
```

where `<data_file>` is the path to the data file, `<backend>` is the backend selection (0 for file, 1 for main-memory, 2 for MapDB) and `<id>` is the ID of the row to select.

These parameters will be used during grading, so make sure you test your code with these parameters.

Things to remember

1. **The main() method will be replaced during grading. All your work should be done inside the load() and select() methods.**
2. For returning specific columns, you will have to pair each columns' value with the column header when you parse each row in the file and save it as <key, value> pairs. For example, for the following row –

   nm0081055 Simone Bicking \N \N actress tt0806901,tt0115088,tt0103434,tt0248657

   you will save these <key, value> pairs -

   {primaryProfession=actress, birthYear=\N, nconst=nm0081055, knownForTitles=tt0806901,tt0115088,tt0103434,tt0248657, deathYear=\N, primaryName=Simone Bicking}

3. You need to return the desired record in both cases – sorted (by key) or unsorted. Your search algorithm should accommodate it.

The project consists of three parts as follows.

Part 1: Disk Storage

The application will use the filesystem as back-end. The application will parse the input query and access the data directly from the TSV files.

1. Complete the `select_file()` method. This method should retrieve the row directly from the file using Java's built-in file I/O functionality. You may use whichever method you're comfortable with.

As mentioned in Things to remember:2, you will have to parse the retrieved row and return values only for columns in column_names.

Part 2: In-Memory Storage

The application will use an in-memory data structure as back-end. You will use an in-memory data structure (e.g. HashMap) to store the data.

1. Complete the `load_mainmemory()` method. This should attempt to load the data from the given file into a main-memory HashMap. Use the ID field of each row as the key and use the parsed <key, value> pairs (refer to Things to remember:2) as value in the HashMap. The key would be of type **`String`** and the value would be of type **`Dictionary`**.
2. Complete the `select_mainmemory()` method. This should retrieve the row using the main-memory HashMap that you loaded the data into.

Try this with both the 10MB and 30MB datasets and observe what happens.

Part 3: Embedded Database

The application will use an embedded-database as back-end. For this part, you have to use **MapDB**, an embedded Java database engine. Please, check MapDB documentation to become familiar with the database. The provided skeleton code should already have the necessary import statements included and data structures defined.

1. Complete the `load_mapdb()` method. This should attempt to load the data from the given file into a MapDB HashMap. Use the ID field of each row as the key and use the parsed <key, value> pairs (refer to Things to remember:2) as value in the HashMap. The key would be of type **String** and the value would be of type **Dictionary**.
2. Complete the `select_mapdb()` method. This should retrieve the row using the MapDB HashMap that you loaded the data into.

What to Turn in

1. You only need to turn in the following files:
   - **project1.java**
2. You do not have to turn in any other files. Do not put any logic or any changes in **App.java**.
3. Do not change methods signature for: `load_mainmemory(), load_mapdb(), select_file(), select_mainmemory(), select_mapdb(), fastestLoad()` and `fastestSelect()` in **project.java**. However, feel free to add additional methods or helper methods inside **project.java**.

These files should be submitted on Brightspace.