Assignment 1: Simulating a Distributed RDBMS

January 29, 2026

Overview

In this assignment, you will design and implement **SimuFragDB**, a simulated distributed database system that runs on a single physical machine. The objective is to emulate a multi-node distributed database environment by executing multiple instances of a relational database management system, preferably PostgreSQL. Each database instance represents an independent node and stores a fragment of the overall relation.

In distributed database systems, relations can be fragmented using two primary strategies: *vertical fragmentation* and *horizontal fragmentation*. In vertical fragmentation, a relation R is decomposed into multiple sub-relations such that

$$R = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n,$$

where each fragment contains a subset of the attributes and the original relation can be reconstructed using natural joins. In contrast, horizontal fragmentation partitions a relation into subsets of tuples such that

$$R = R_1 \cup R_2 \cup \cdots \cup R_n.$$

In this assignment, you will exclusively employ **horizontal fragmentation**. A deterministic routing function will be provided to direct queries to the appropriate fragment (database instance) based on the defined fragmentation scheme. Queries must be explicitly routed to the corresponding node using this routing function.

The primary objective of this assignment is **not** to guarantee complete correctness, consistency, or fault tolerance while storing or retrieving data from the relation. Instead, the focus is on gaining practical insights into data fragmentation, query routing, and the architectural principles underlying distributed database systems.

Database Schema

Initialize each database instance with the following schema:

```sql
CREATE TABLE Student (
    student_id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    email VARCHAR(100)
);

CREATE TABLE Grade (
    student_id VARCHAR(50),
    course_id VARCHAR(20),
    score INT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

```sql

CREATE TABLE Course (
    course_id VARCHAR(20) PRIMARY KEY,
    course_name VARCHAR(100),
    department VARCHAR(50)
);
```

Listing 1: Database Schema (scripts.sql)

Provided Files

You are provided with the following files:

File	Description
`Driver.java`	Main program that reads the workload file and invokes FragmentClient methods
`Router.java`	Deterministic routing function (already implemented)
`FragmentClient.java`	Skeleton class with TODO methods for you to complete
`scripts.sql`	SQL schema to initialize each fragment
`workload.txt`	Input file containing operations to execute

Your Tasks

Complete the following methods in `FragmentClient.java`:

1. **setupConnections()** - Initialize JDBC connections to all N database fragments. Store connections in the `connectionPool` map with Fragment ID as the key.

2. **insertStudent(String studentId, String name, int age, String email)** - Route the student to the fragment using the router and execute an INSERT statement.

3. **insertGrade(String studentId, String courseId, int score)** - Insert a grade record into the fragment using the router.

4. **updateGrade(String studentId, String courseId, int newScore)** - Update an existing grade for a student-course combination.

5. **deleteStudentFromCourse(String studentId, String courseId)** - Remove a grade entry for the specified student and course.

6. **getStudentProfile(String studentId)** - Return the student's name and email in format: `"name,email"`

7. **getAvgScoreByDept()**[1]- Calculate and return the average score per department. Format: `"CS:75.5;Math:82.3;Physics:78.0"`

8. **getAllStudentsWithMostCourses()**[1] - Find students enrolled in the maximum number of courses.

[1]These queries must be executed on a randomly selected fragment.

Workload File Format

The `workload.txt` file contains operations in the following formats:

```
INSERT_STUDENT,<student_id>,<name>,<age>,<email>
INSERT_GRADE,<student_id>,<course_id>,<score>
UPDATE_GRADE,<student_id>,<course_id>,<new_score>
DELETE_STUDENT_COURSE,<student_id>,<course_id>
READ_PROFILE,<student_id>
READ_SCORE
READ_ALL
```

Routing Function

The routing function is already implemented in `Router.java`:

```
public int getFragmentId(String key) {
    return Math.abs(key.hashCode()) % numfragments;
}
```

This ensures deterministic routing. Use the primary key of the table as the key for insertion and updation.

Setup Instructions

1. Install Java 17 and Maven.
2. Install PostgreSQL and create N different databases (fragments).
3. Run `scripts.sql` on each fragment to create the schema
4. Configure your JDBC connection strings in `setupConnections()`
5. Using the attached 'project' folder as the root folder, Compile and run:

```
mvn clean install
mvn dependency:copy-dependencies
java -cp "target/classes:target/dependency/*" Driver

```

6. Your output will be written to `output.txt`

Evaluation Metric

To verify the correctness of your distributed implementation:

1. **Generate expected output:** Run the workload against a **single database** containing all data. Save this as `expected_output.txt`. This constitutes the control part of the experiment.
2. **Generate actual output:** Run the same workload against your SimuFragDB implementation. This produces `output.txt`.
3. **Report accuracy:** In your report, include:
   - Total number of query output lines
   - Number of lines that differ between expected and actual output
   - Accuracy percentage: $\frac{matchinglines}{totallines} \times 100$

Submission Requirements

1. `FragmentClient.java` — Your completed implementation
2. `output.txt` — Generated output from running the workload
3. `expected_output.txt` — Output from single-instance baseline
4. `report.pdf` — Brief report (1-2 pages) containing:
    - Your approach for aggregating data across fragments
    - Any challenges faced and how you resolved them
    - Execution time reported by the driver
    - Accuracy metric (number of differing lines and percentage)

GitHub Submission

Each assignment must be submitted to GitHub:

- Create a **private repository** for your submission
- Add all TAs (Shreyas529, Vishruth23, Sathvik21S21Rao and Ullas-0-1) and `bda-lab` as **collaborators**
- Commits made by individual group members will be evaluated to assess **proper division of work**
- The GitHub repository link must be submitted along with the final report

Repository structure:

```
/
|-- src/
|   |-- fragment
|   |   |-- FragmentClient.java
|       |-- Router.java
|   |-- Driver.java

|-- output/
|   |-- output.txt
|   |-- expected_output.txt
|-- report.pdf
|-- README.md
```

Important Information
The group-wise demo with the TAs will be conducted as per the announced schedule. All group members must be present. **Absent members will be awarded zero marks**. No rescheduling will be permitted, except for genuine medical reasons. In such cases, a formal medical certificate must be submitted to the Registrar's Office and approved.
The evaluation will consist of: (i) assignment demonstration and (ii) viva voce based on the project and relevant lectures. Marks awarded may differ among group members.