# End Term Project: Data Synchronization Across Heterogeneous Systems

In this project, you are expected to demonstrate your understanding of data integration and synchronization across heterogeneous systems. Building on the concepts explored in Assignment 3, assume that the file `student_course_grades.csv` (attaching herewith) has been redundantly loaded onto at least **three heterogeneous data systems**. The systems must include **MongoDB** and any two or more from the following: **Pig**, **Hive**, or **PostgreSQL/MySQL**.

You are required to implement methods for reading (`get()`) and updating (`set()`) data independently in each of these systems. Furthermore, each system should define an abstract function named `merge()`, which takes as an argument the name of another system whose state is to be merged with the calling system. The `merge()` function should not directly access the data from the other system; instead, it must rely solely on an **operation log (oplog)** that records the sequence of operations performed in that system. The operation log may be designed in a way that it is generic and applicable to any table, not just `student_course_grades.csv`.

As a first step, identify the possible CRUD operations supported by each chosen system (e.g., MongoDB, Pig, Hive, PostgreSQL/MySQL) and define them as the services offered by each system. From these, **consider only the operations relevant to reading and updating** the `Grade` field for a given student ID, course-ID combination. Note: (student-ID, course-ID) form a composite primary key of the table.

Design the merge functionality specific to each system – if necessary. For example, a command like `PIG.MERGE(SQL)` should merge the current state of the Pig table with the state of the PostgreSQL table (but not vice versa). The merge should be performed based on the operation logs of the two systems involved. These merge commands will be provided as input to your main program, which should execute them in the given order to synchronize the systems. Keep in mind the mathematical properties of the merge operations, such as *associativity*, *commutativity*, and *idempotency*, while implementing your merge function. Reflect on how convergence would occur in these scenarios, given the types of operations each system supports.

You may define the structure of operation logs (oplogs) for each system using the following example formats: ⟨*timestamp or counter value*⟩, ⟨*SET/GET operation with respective arguments*⟩

## Sample Operation Logs

**oplog.hiveql**

```
1, SET((SID103,CSE016), A)
2, GET(SID103,CSE016)
```

**oplog.sql (PostgreSQL or MySQL)**

```
1, GET(SID103,CSE016)
2, GET(SID403,CSE013)
3, SET((SID103,CSE016), B)
```

**oplog.mongo**

```
1, SET((SID101,CSE026), B)
2, GET(SID403,CSE013)
3, SET((SID101,CSE026), A)
```

Prepare a test case file that contains a sequence of `SET`, `GET`, and `MERGE` commands. This test case should serve as an input to your main program. A sample test case file could look like the following:

**testcase.in**

```
1, HIVE.SET((SID103,CSE016), A)
2, HIVE.GET(SID103,CSE016)
1, SQL.GET(SID103,CSE016)
2, SQL.GET(SID403,CSE013)
1, MONGO.SET((SID101,CSE026), B)
3, SQL.SET((SID103,CSE016), B)
2, MONGO.GET(SID403,CSE013)
3, MONGO.SET((SID101,CSE026), A)
HIVE.MERGE(SQL)
SQL.MERGE(HIVE)
SQL.MERGE(MONGO)
MONGO.MERGE(HIVE)
```

**Note:** The `SET` and `GET` operations for a particular server are ordered based on the timestamp. You may also assume that the `MERGE` operations do not carry any timestamps and must be executed strictly in the given order.

# Submission Guidelines

- The project submission deadline is on **April 29th, midnight**.
- By **April 30th**, each student group should meet with the TAs for a demo and evaluation based on mutual convenience. No extensions beyond **April 30th** will be allowed.
- Each student group should submit the source code along with a short report describing their approach.
- In the report, each student should explicitly self-declare their contribution out of 100. Failure to do so will indicate no or partial participation.

- We expect simple and feasible solutions rather than complex and non-intuitive ones.
- You may use any programming language of your choice for the implementation.
- The code should not be multi-threaded or involve parallel processing.
- Having a graphical user interface (GUI) is not mandatory.

Your final submission should include the source code, detailed explanations of the merge logic, the design of the operation log, sample logs and test cases, and a short write-up discussing the behavior and properties of your merge operations. The implementation must use at least three heterogeneous systems and must **include MongoDB**.