

## Database Management System

### Problem Statement:

To create a distributed Database Management System that performs the basic CRUD operations (create, insert, update, delete, select), generates SqlDump, ERD, executes concurrent transactions and performs processing on the machine where the table resides.

### Overview:

- This project represents the multi-user distributed Database Management System which allows operation on data using a basic set of SQL queries such as CREATE, INSERT, UPDATE, DELETE and SELECT.
- Also, a single condition Where clause can be used to perform operations on filtered data.
- This project represents the novelty of a distributed Database Management System using the LinkedHashMap<> data structure to operate on the data until the user commits and to facilitate the faster syntax and semantics check, HashMap<> data-structure is used to store the Metadata of the table.
- Google cloud storage(Bucket) is used to store the centralized Global Data Dictionary to ensure the availability of the dictionary to all the remote instances.
- Also, the system allows the user to create the queries using a SQLDump of the database and a text-based ERD with the structure of the tables in a text format can be generated.

### Team Coordination:

- Before the Submission of our Feasibility Report, we discussed and brainstormed about the requirements like the features to be implemented, the file formats, the data structures, the programming language etc.
- Then we first splitted the basic tasks to get a start, then we met once or twice a week to discuss the status and the blockers if there are any.
- We all supported each other whenever there was an issue during our meets.
- We implement the CRUD operations with conditions, SqlDump, ERD, General and Event Logs, Connecting to GCP and fetching the data, multi user operations, distributed concept etc.
- In our Q&A session with the professor, we understood that we only implemented a partial distributed database concept. The outcome of that meeting is to separate parsing and processing logic, to centralize GDD and to use JSCH to SFTP request and response files between local and remote machines.
- We utilized the time after Q&A to implement these missed out things as suggested accordingly.

### Sample charts of our meetings:

↩ Reply

April 4, 2021

Meeting ended: 49m 17s

↓

Attendance Report

Click here to download attendance report

FS HD SP

↩ Reply

April 5, 2021

Meeting ended:

FS HD SP

↩ Reply

April 7, 2021

Meeting ended:

FS HD SP

↩ Reply

April 9, 2021

Meeting ended: 10m 7s

FS HD SP

↩ Reply

Meeting ended: 22m 2s

↓

Attendance Report

Click here to download attendance report

FS HD SP

↩ Reply

April 12, 2021

Meeting ended: 19m 45s

FS HD SP

↩ Reply

Meeting ended: 45m 17s

↓

Attendance Report

Click here to download attendance report

FS HD SP

↩ Reply

Meeting ended: 10m 28s

↓


Attendance Report

Click here to download attendance report

FS HD SP

↩ Reply

Meeting ended: 31m 7s

FS HD  SP

↩ Reply

Meeting ended: 17m 28s

FS HD SP

2

### Group Coordination:

- Every team-mate worked on their part and provided a fair contribution in the project.
- Team met regularly to decide on the tasks and designing of the solution.
- Team performed the review of the work done and provided inputs on any blockers in the implementation part.
- Everybody on the team collaborated on every module (at least one function in each module).

### Individual Coordination:

#### Hashik Donthineni

##### 1. Table Locking (Multi User Feature)

Table locking and unlocking to enable concurrent access even with distributed databases.

##### 2. File Reading and Writing

File/Table Reading and writing

##### 3. Query processing on data structure (LinkedHashMap)

- a. Update
- b. Select
- c. Delete
- d. insert

on the table/data-structure

##### 4. Data structure serialization

Loading data from file to the data-structure. This module uses the file reading and writing module.

##### 5. Data structure de-serialization

Prepping data structure data to be written into the file. This module uses the file reading and writing module.

#### Sravani Pinninti

##### 6. User Authentication

User is validated against the credentials which are already stored and only the valid user will be able access the database.

##### 7. Syntax validation

The first step of validation is validating the syntax of the queries, if it is a valid syntax then the further steps are executed, else an error message is displayed.

##### 8. SQLDump generation

For all the tables present in local and remote locations, the corresponding metadata files that are created during the table creation are fetched. From these files, metadata is pulled and a create table query is generated for each table. and from those emulating create queries from the metadata stored during table creation.

##### 9. ERD generation

For all the tables present in local and remote locations, the corresponding metadata files that are created during the table creation are fetched and displayed on to the console.

##### 10. Logging

General logs - The current statistics of the database along with the timestamp are stored in this log file.

Event Logs - The query that is being executed will be stored here, along with the timestamp.

These log functions are called before executing every query.

#### **11. Creating database and tables with Metadata**

For a create statement an empty file is created and the metadata from the create statement is stored into a metadata file.

#### **12. Established remote server connection:**

For this module VM configuration is set up using SSH-Key gen and this module will send a request file to GCP if the query is related to a remote table and will pull a response file from GCP after execution of the query to local machine.

### **Fenil Shah**

#### **13. Semantics Validation**

This module will check the semantics of the query which will validate if the table name and columns provided in the query are correct or not and it will also validate the datatype of the value provided while filtering or updating the records. So this will check if the query entered by the user is semantically correct or not.

#### **14. Execution Engine**

This module will convert the query into objects required by the Query processing data structure and then perform the operation based on the query type and return the result or write the data to the file if the user commits.

#### **15. Metadata Data Structure**

This data structure will be used to load the metadata of the tables such as tablename, the column properties that include index of the column, datatype of the column, constraints, etc., location of the table data file, location of the table(local/remote).

#### **16. Metadata Data Structure Serialization/Deserialization**

This module provides the functions to serialize the metadata data structure after new table creation and deserialization after every table deletion.

This module handles the changes done in the table structure.

#### **17. Global Data Dictionary**

This will store all the table names for the database, the name of the data file and the location type (local/remote) and reads/writes/updates gdd whenever the new table is created.

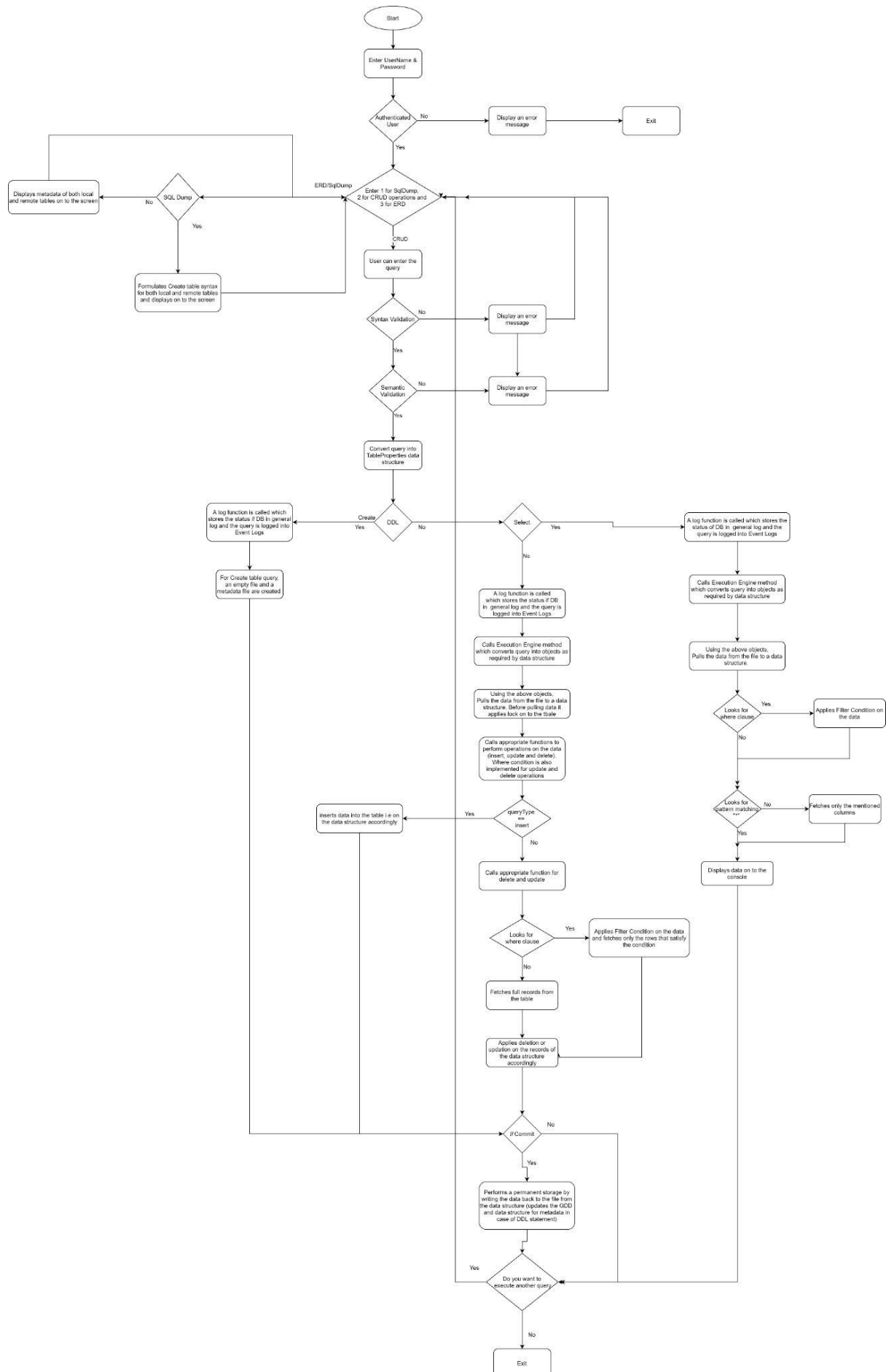
#### **18. Controller Separated from Main module**

This module watches if the query submitted to the system is for a local or remote database, and it is responsible for calling the execution engine for locally processing the query and writing the response to a file. If the query submitted is for the local table, then the controller of the local module will process it and in case of the remote table, the controller module of the

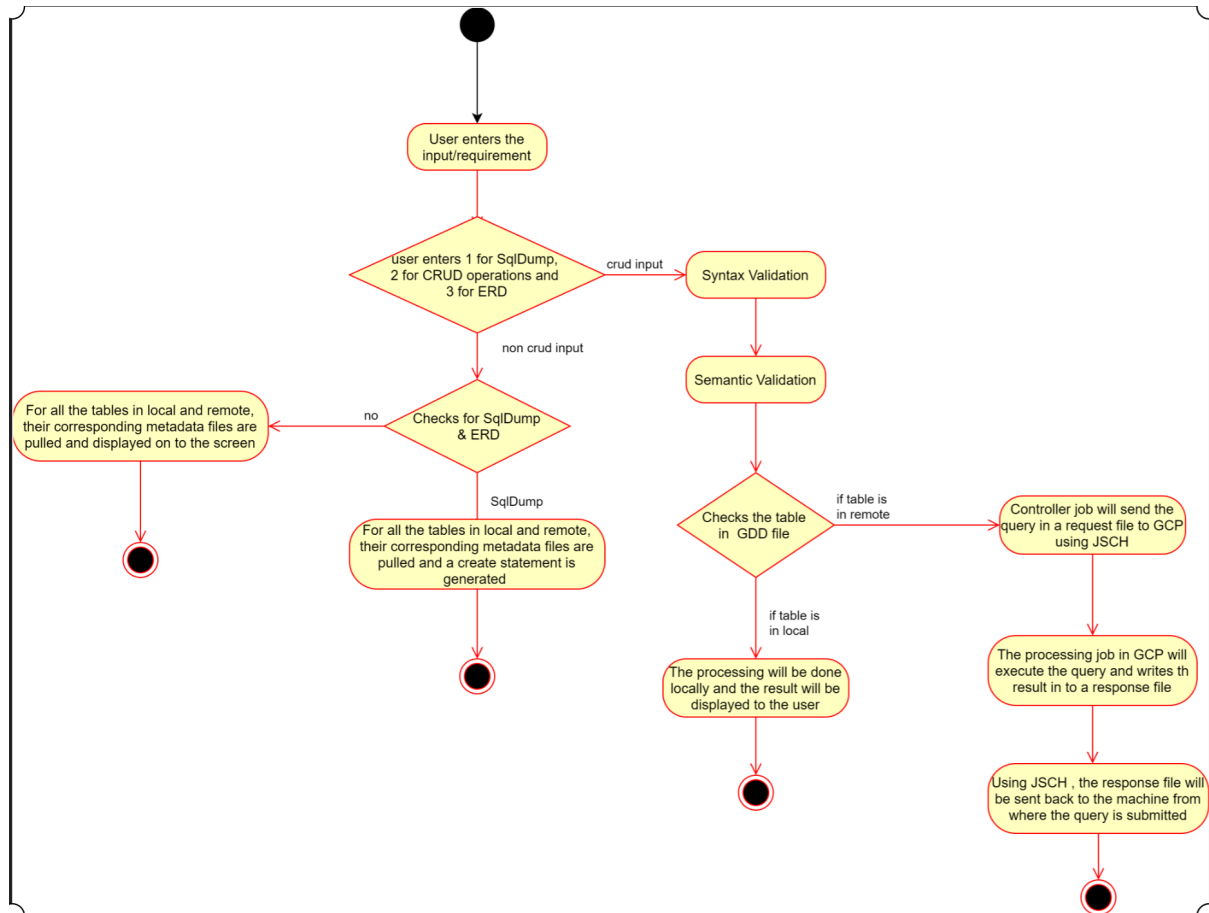
remote DBMS instance will process the query and send the response to the user.

## **Implementation Plan:**

### **Flow Chart**



## Activity Diagram:



## Implementation Details:

### Login Validation pseudo code:

```

while((line = bufferedReader.readLine()) != null)
{
    splitCredentials = line.split( regex: "@" );
    credential.put(splitCredentials[0],splitCredentials[1]);
}
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
if (password.trim().equals(credential.get(userName).trim()))
{
    flag = true;
}

```

### Syntax validation pseudo code:

```
public boolean syntaxValidation (String query, String regex)
{
    Pattern pattern = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
    Matcher match = pattern.matcher(query);
    Boolean validity = match.find();
    return validity;
}
```

## The regular expressions that are used for syntax validation

```
String deleteRegex = "^delete\\sfrom\\s(?:_)?\\w*=\\s(\\s(\\swhere)\\s[a-z0-9]*=([a-z0-9]*)?)?;$";  
String updateRegex = "^update\\s(?:_)?\\w*=\\sset\\s((((([a-z0-9]+)=([a-z0-9]+))(|[,])+)+\\s((\\swhere)\\s[a-z0-9]*=([a-z0-9]*)?)?$";  
String insertRegex = "^insert\\sinto\\s(?:_)?\\w*=\\sVALUES\\s?\\s\\\\\\\\((((([a-z0-9\\\\\\\\]+)([(),*])+) + ^,\\\\\\\\))$";  
String selectRegex = "^select\\s(((+|((([a-z]+)([(),*]+)))\\sfrom\\s(?:_)?\\w*=\\s(\\s(\\swhere)\\s[a-z0-9]*=([a-z0-9]*)?)?$";  
String createDBRegex = "^create\\sdatabase\\s[^;]*;$";  
String createTableRegex = "^create\\stable\\s(?:_)?\\w*=\\s\\\\\\\\((((([a-z0-9]+)\\s(int|varchar\\\\\\\\(\\d+\\\\\\\\))([(),*]+))(?! ,\\\\\\\\))$";  
String useDbRegex = "^use\\s[^;]*;$";  
String commitRegex = "commit;$";
```

### Semantic Validation pseudo code:

Checks if the table already exists

```
if (queryType.equals("create")) {
    if (ds != null) {
        if (ds.keys().contains(TableName.toLowerCase(Locale.ROOT))) {
            System.out.println("Table " + TableName + " already exists.");
            return false;
        }
    }
    return true;
} else {
    if (ds != null) {
        if (ds.keys().contains(TableName.toLowerCase(Locale.ROOT))) {
            return true;
        }
    }
}
```

Checks if the column already exists:



```
public boolean checkIfColumnExists(String tbl_name, String fields) {
    TablePropertiesDS props = this.databaseMap.get(this.databaseName);
    TableProperties tblProps = props.properties.get(tbl_name);
    Set<String> columns = tblProps.getColumns().keys();
    if (fields != null) {
        String[] cols = fields.split(regex: " ");
        if (!cols[0].equals("*")) {
            for (String col : cols) {
                if (!columns.contains(col)) {
                    System.out.println("Table " + tbl_name + " has no column with name " + col);
                    return false;
                }
            }
        }
    }
}
```

Checks for the data type match:

```
if(!values.equals(""))
{
    try {
        index = getColumnIndex(tableName, value[0]);
    } catch (Exception e) {
        index = -1;
    }
    if (index >= 0) {
        TableColumn column = columns.get(index);
        if (column.getDataType().contains("int")) {
            try {
                int intValue = Integer.parseInt(value[1]);
                result = true;
            } catch (NumberFormatException e) {
            }
        }
    }
}
```

**GDD Writer:**

```
public class GDDWriter {
    public static void replaceGDDWithString(String stringToWrite) throws IOException {
        Bucket database = GCPStorage.getStorage();
        database.create(FileConstants.GDD_FILE, stringToWrite.getBytes(StandardCharsets.UTF_8));
    }

    public static void appendGDD(String stringToAppend) throws IOException {
        replaceGDDWithString(stringToWrite: GDDReader.readGDD() + stringToAppend + "\n");
    }

    public static void createNewGDD() throws IOException {
        Bucket database = GCPStorage.getStorage();
        database.create(FileConstants.GDD_FILE, "").getBytes(StandardCharsets.UTF_8));
    }
}
```

**GDD Reader:**

```
public class GDDReader {
    public static String readGDD() throws IOException {
        return new String(GCPStorage.getStorage().get(FileConstants.GDD_FILE).getContent());
    }
}
```

### Loading GDD into Data structure:

```
public Map<String, String[]> getGDD() {
    TablePropertiesDS ds = this.databaseMap.get(this.databaseName);
    File file = new File(this.locationGDD + "\\gdd.txt");
    Map<String, String[]> gdd = new HashMap<>();
    try {
        String fileContent = new String(gddReader.readGDD());
        Scanner sc = new Scanner(fileContent);
        sc.nextLine();
        String[] queryTokens;
        int index = 0;
        do {
            queryTokens = sc.nextLine().split(regex: " ");
            if(queryTokens[0] != "")
            {
                gdd.put(queryTokens[0], new String[]{queryTokens[1], queryTokens[2]});
            }
        }while (sc.hasNextLine());
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return gdd;
}
```

### Writing Metadata Data structure to GDD:

```
public void writeMetadataToGDD() {
    TablePropertiesDS ds = this.databaseMap.get(this.databaseName);
    File file = new File(this.locationGDD + "\\gdd.txt");
    try {
        file.createNewFile();
        gddWriter.createNewGDD();
        FileWriter fileWriter = new FileWriter(this.locationGDD + "\\gdd.txt");
        for (TableProperties prop : ds.values()) {
            gddWriter.appendGDD( stringToAppend: prop.getTableName() + "~" + prop.getLocation() + "~");
            gddWriter.appendGDD("\n");
        }
        fileWriter.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

## Loading Data tables' metadata into a metadata data structure:

```
public TablePropertiesDS loadMetaData() {
    try {
        if (this.checkDBExist(this.databaseName)) {
            File directoryPath = new File(this.location);
            File[] files = directoryPath.listFiles();
            TablePropertiesDS tablePropertiesDS = new TablePropertiesDS();
            for (File file : files) {

                String[] filename = file.getName().split(regex: "\\-");
                if (filename[0].equals("meta")) {
                    TableProperties tableProperties = new TableProperties();
                    tableProperties.setTableName(filename[1].split(regex: "\\-")[0]);
                    tableProperties.setStructureUpdated(false);
                    tableProperties.setLocation(filename[1]);
                    tableProperties.setLocationType("local");
                    TableColDS tableColDS = new TableColDS();
                    BufferedReader br = new BufferedReader(new FileReader(file));
                    String data;
                    int index = 0;
                    while ((data = br.readLine()) != null) {
                        String[] fields = data.split(regex: "\\~");
                        List<String> field = new ArrayList<>();
                        field = Arrays.asList(fields.clone());
                        TableColumn column = new TableColumn();
```

## SqlDump:

```
String[] tablenames;
File directory = new File(FileConstants.LOCAL_DB_LOCATION);
File[] files = directory.listFiles((d, name) -> name.startsWith("meta"));
for (File file : files) {
    String[] fileName = file.getName().split(regex: "\\-");
    FileReader fileReader = new FileReader(FileConstants.LOCAL_DB_LOCATION + "\\- " + file.getName());
    BufferedReader bufferedReader = new BufferedReader(fileReader);
    String line = "", mergedLine = "";
    while ((line = bufferedReader.readLine()) != null) {
        mergedLine += (line + ",");
    }
    tablenames=fileName[0].split(regex: "\\-");
    System.out.println("create table "+tablenames[1]+"("+mergedLine.replaceAll(regex: "\\~", replacement: " ").replaceAll(regex: "\\$", replacement: " "));
}
```

### For Remote Tables:

```
for (Blob blob : blobs.iterateAll()) {
    String[] filename = blob.getName().split( regex: "\\-");
    if (filename[0].equals("meta")) {
        String[] fileName = blob.getName().split( regex: "\\-");
        String fileContent = new String(blob.getContent());
        Scanner sc = new Scanner(fileContent);
        sc.nextLine();
        int index = 0;
        String remoteMerge="";
        while (sc.hasNextLine()) {
            remoteMerge += (sc.nextLine()+"," );
        }
        tablename = fileName[0].split( regex: "\\-");
        System.out.println("create table " + tablename[1] + "(" + remoteMerge.replaceAll( regex: ",", replacement: " ").replaceAll( regex: " ", replacement: " ") + ");");
    }
}
```

### ERD :

### For local tables

```
for (File file : files) {
    String[] fileName = file.getName().split( regex: "\\-");
    FileReader fileReader = new FileReader( fileName: FileConstants.LOCAL_DB_LOCATION + "\\- " + file.getName());
    BufferedReader bufferedReader = new BufferedReader(fileReader);
    String line = "", mergedLine = "";
    tablename = fileName[0].split( regex: "\\-");
    //System.out.println("The metadata for " + tablename[1] + " is:");
    System.out.println("----- " + tablename[1].toUpperCase() + " -----");
    while ((line = bufferedReader.readLine()) != null) {
        System.out.println(line);
    }
}
```

### For Remote Tables:

```
for (Blob blob : blobs.iterateAll()) {
    String[] filename = blob.getName().split( regex: "\\-");
    if (filename[0].equals("meta")) {
        String[] fileNames = blob.getName().split( regex: "\\-");
        String fileContent = new String(blob.getContent());
        Scanner sc = new Scanner(fileContent);
        sc.nextLine();
        tablename = fileNames[0].split( regex: "\\-");
        System.out.println("----- " + tablename[1].toUpperCase() + " -----");
        while (sc.hasNextLine()) {
            System.out.println(sc.nextLine());
        }
    }
}
```

### Controller PseudoCode:

- Controller for each DBMS instance keeps watch on the requestFile and responseFile.
- If there is any request from some other server provided in the "requestFile" file, then the controller will process the query and send the output to the requestor.

- If there is any response from some other server is provided in the “responseFile”, then the controller will display the result on the screen to the user

```
public Controller(ExecutionEngine executionEngine)
{
    this.executionEngine = executionEngine;
    this.filePath = Paths.get(new File( pathname: "." ).getAbsolutePath().replace( target: ".", replacement: "\\QueryRequestResponse"));
    try {
        watchService = FileSystems.getDefault().newWatchService();
        System.out.println(filePath);
        filePath.register(watchService, StandardWatchEventKinds.ENTRY_CREATE, StandardWatchEventKinds.ENTRY_DELETE, StandardWatchEventKinds.ENTRY_MODIFY);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
for (WatchEvent<?> event : key.pollEvents()) {
    WatchEvent<Path> ev = (WatchEvent<Path>) event;
    Path filename = ev.context();
    if(filename.getFileName().equals("RequestedQuery"))
    {
        System.out.println("File modified");
        //Perform Operation
        File file = new File( pathname: filePath + "\\requestFile.txt");
        BufferedReader br = new BufferedReader(new FileReader(file));
        String data;
        while ((data = br.readLine()) != null) {
            this.query = data;
        }
        file.createNewFile();
        //write to the remote file
    }
    else
    {
        //Read response file and show output
        File file = new File( pathname: filePath + "\\responseFile.txt");
        BufferedReader br = new BufferedReader(new FileReader(file));
        String data;
        while ((data = br.readLine()) != null) {
            System.out.println(data);
        }
    }
}
```

## Use Cases:

## Test Cases:

### 1. User Validation

#### i. Valid User

- Proceed with the flow to enter the inputs

#### ii. Invalid User

- Displays an error message as “Invalid User”

### 2. Syntax Validation

#### i. Valid Syntax for all queries

- Proceeds further with semantic validation

#### ii. Invalid Syntax

- Displays an error message as “Invalid Syntax”

#### iii. Accepting where clause for select, update and delete

- Filtering should be performed on the rows using where clause.

### 3. SqlDump

- It should print the create table syntax for all the tables from local and remote.

### 4. ERD

- Prints metadata for all tables from local and remote locations.

### 5. Logs

#### i. Valid Case

- Every query should be stored in an Event Log file.
- Database statistics should be stored into a General log file before execution of every query.

### 6. Semantic Validation

#### i. Correct Tablename

- Proceed the control flow to execute the query.

#### ii. Incorrect Table name

- If the table name given in the query does not exist in the local or remote machine, the error will be shown to the user.

#### iii. Correct Fieldlist

- If the columns specified in the field list provided in the select query belongs to the mentioned table, then the query will be executed successfully.

#### iv. Incorrect Fieldlist

- If any of the columns specified in the field list provided in the select query does not belong to the mentioned table, then the query will not be executed and the error will be shown to the user.

#### v. Correct Where Clause

- If the where clause is written correctly and the data provided in the condition adheres to the datatype of the conditional column then the query will be executed and the results will be displayed.
- vi. **Incorrect Where Clause**
- If the where clause is written incorrectly and the data provided in the condition does not adhere to the datatype of the conditional column or the column name given in the condition is incorrect then the user will be displayed an error message.

## Test Demo:

### 1. User Validation

#### i. Valid User

```
Enter Username
fenil
Enter Password
pass
User Authenticated
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
2
Enter Query for Processing
```

#### ii. Invalid User

### 2. Syntax Validation

#### i. Valid syntax

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
2
Enter Query for Processing
insert into emp values(1,sravani);
Insertion Successful
```

#### ii. Invalid Syntax

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
2
Enter Query for Processing
insert emp(2,abc);
Syntax Error
The expected format is... INSERT INTO <TABLENAME> VALUES(VALUE1,VALUE2,...VALUEN);
```

#### iii. where clause

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
2
Enter Query for Processing
select * from emp where id=1;
[1,sravani
```

### 3. SqlDump:

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
1
create table emp(id int,name varchar(10));
create table student(id int,name varchar(25),country varchar(25));
create table table_name(col1 int,col varchar(100),col varchar(100));
*****SQLDUMP FORE REMOTE TABLES*****
create table student_info(name varchar(20),bannerid varchar(50));
create table table1(address varchar(100),email varchar(100),contact varchar(10));
create table table2(address varchar(100),email varchar(100),contact varchar(10));
Do you want to execute another query? (yes / no)
```

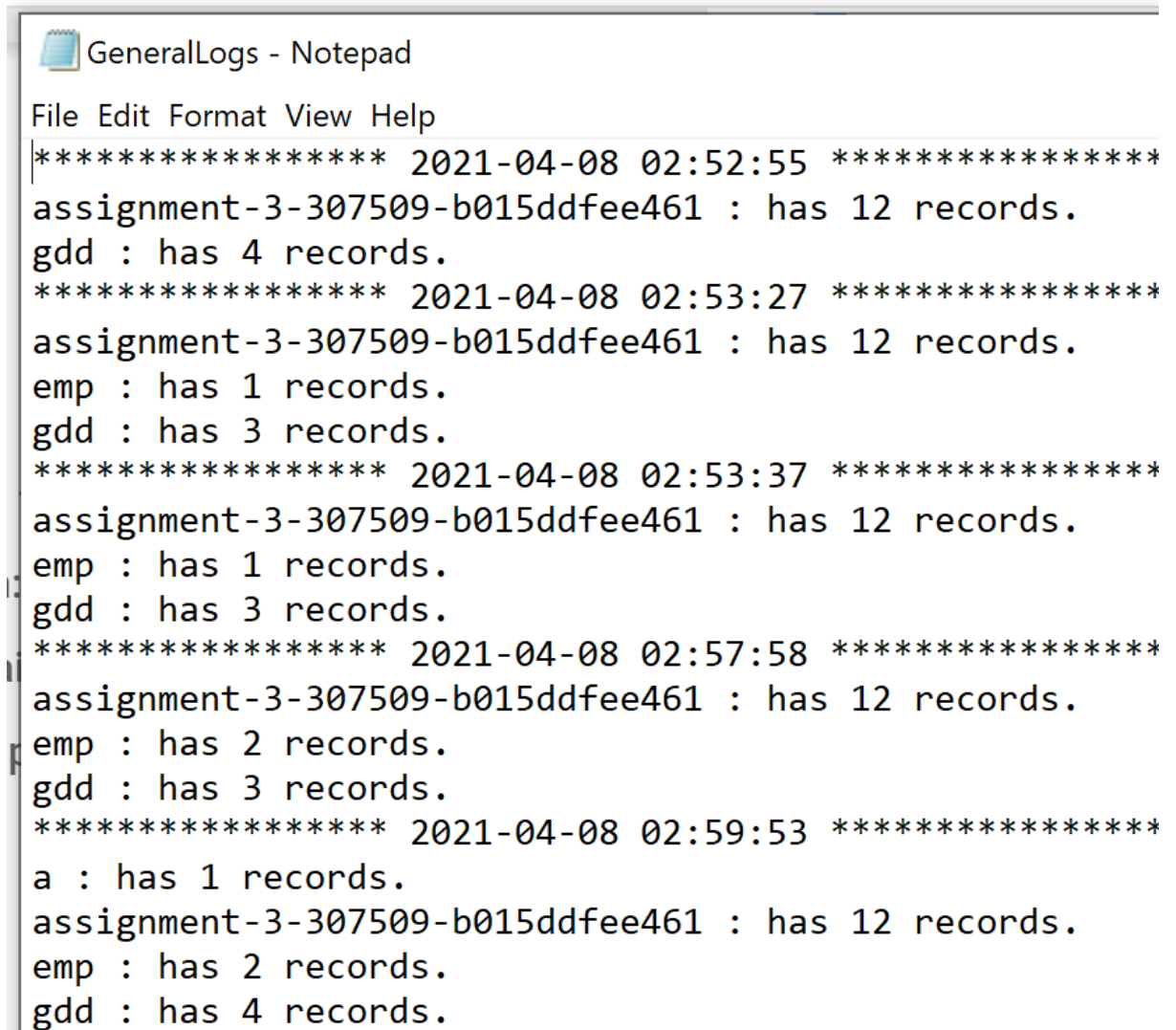
### 4. ERD

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
3
----- EMP -----
id~int
name~varchar(10)
----- STUDENT -----
id~int
name~varchar(25)
country~varchar(25)
----- TABLE_NAME -----
col1~int
col~varchar(100)
col~varchar(100)
*****SQLDUMP FORE REMOTE TABLES*****
----- STUDENT_INFO -----
name~varchar(20)
bannerid~varchar(50)
----- TABLE1 -----
address~varchar(100)
email~varchar(100)
contact~varchar(10)
----- TABLE2 -----
address~varchar(100)
email~varchar(100)
contact~varchar(10)
```




## 5. Logs

### General Logs:



```
GeneralLogs - Notepad
File Edit Format View Help
***** 2021-04-08 02:52:55 *****
assignment-3-307509-b015ddfee461 : has 12 records.
gdd : has 4 records.
***** 2021-04-08 02:53:27 *****
assignment-3-307509-b015ddfee461 : has 12 records.
emp : has 1 records.
gdd : has 3 records.
***** 2021-04-08 02:53:37 *****
assignment-3-307509-b015ddfee461 : has 12 records.
emp : has 1 records.
gdd : has 3 records.
***** 2021-04-08 02:57:58 *****
assignment-3-307509-b015ddfee461 : has 12 records.
emp : has 2 records.
gdd : has 3 records.
***** 2021-04-08 02:59:53 *****
a : has 1 records.
assignment-3-307509-b015ddfee461 : has 12 records.
emp : has 2 records.
gdd : has 4 records.
```

Event Logs:

 EventLogs - Notepad  
File Edit Format View Help  
\*\*\*\*\* 2021-04-08 02:52:55 \*\*\*\*\*:  
create table emp(id int,name varchar(10));  
\*\*\*\*\* 2021-04-08 02:53:27 \*\*\*\*\*:  
insert into emp values(2,b);  
\*\*\*\*\* 2021-04-08 02:53:37 \*\*\*\*\*:  
commit;  
\*\*\*\*\* 2021-04-08 02:57:58 \*\*\*\*\*:  
create table a(id int,name varchar(10));  
\*\*\*\*\* 2021-04-08 02:59:53 \*\*\*\*\*:  
insert into a values(2,b);  
\*\*\*\*\* 2021-04-08 03:00:23 \*\*\*\*\*:  
insert into emp values(3,c);  
\*\*\*\*\* 2021-04-08 03:00:52 \*\*\*\*\*:  
commit;  
\*\*\*\*\* 2021-04-08 03:04:11 \*\*\*\*\*:  
select \* from emp;  
\*\*\*\*\* 2021-04-08 03:09:22 \*\*\*\*\*:

6. Semantic Validation

a. Select Query

i. Correct Tablename

```
Enter Query for Processing
select * from student;
1, fenil, India
2, test, Canada
3, raj, Australia
```

ii. Incorrect Table name

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
2
Enter Query for Processing
select * from stu;
Table stu does not exist.
Do you want to execute another query? (yes / no)
```

```
Enter Query for Processing
update stu set name=fenil where id=1;
Table stu does not exist.
Do you want to execute another query? (yes / no)
|
```

```
Enter Query for Processing
delete from studen;
Table studen does not exist.
Do you want to execute another query? (yes / no)
```

iii. **Correct Fieldlist**

```
Enter Query for Processing
select id,name from student where country=India;
1,fenil
```

iv. **Incorrect Fieldlist**

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
2
Enter Query for Processing
select nam from student;
Table student has no column with name nam
Do you want to execute another query? (yes / no)
```

```
Enter Query for Processing
update student set test=1 where id=1;
Table student has no column with name test
Do you want to execute another query? (yes / no)
```

```
Enter Query for Processing
update student set id=fenil where name=test;
Column : id expects a valid integer.
Do you want to execute another query? (yes / no)
|
```

v. **Correct Where clause**

```
Enter Query for Processing
select name,country from student where id=1;
fenil,India
```

vi. **Incorrect Where clause**

```
Enter 1 for SqlDump, 2 for CRUD operations and 3 for ERD
2
Enter Query for Processing
select * from student where id=fenil;
Column : id expects a valid integer.
Do you want to execute another query? (yes / no)
```

```
Enter Query for Processing
update student set name=test where id=test;
Column : id expects a valid integer.
Do you want to execute another query? (yes / no)
```

```
Enter Query for Processing
update student set name=test where count=5;
Column : count does not exist in table student
Do you want to execute another query? (yes / no)
|
```

```
Enter Query for Processing
delete from student where nam=test;
Column : nam does not exist in table student
Do you want to execute another query? (yes / no)
|
```

```
Enter Query for Processing
delete from student where id=fenil;
Column : id expects a valid integer.
Do you want to execute another query? (yes / no)
```

### Limitations and Future Scope:

- The current system is limited to only 2 systems connected over a network. There's a future scope of extending the number of remote connections.
- Currently, the system allows only a single database on a server to be used , which can be further extended to multi-database and switching to other databases.
- Current system allows users to execute only simple queries such as table creation, data selection, insertion, deletion and updation on a single table. Multi-table operations can be added to this DBMS and complex functions such as aggregation, sum, etc. can be implemented.