BUILDING A DISTRIBUTED DATABASE MANAGEMENT SYSTEM FEASIBILITY REPORT

Group 10-

Srikrishnan Sengottai Kasi (B00881083)

Preethi Jeeva (B00861320)

Akshay Garg (B00864880)

Introduction

DBMS is used extensively for storing and optimally retrieving and updating data. It provides many useful features like security, transaction management, and many more.

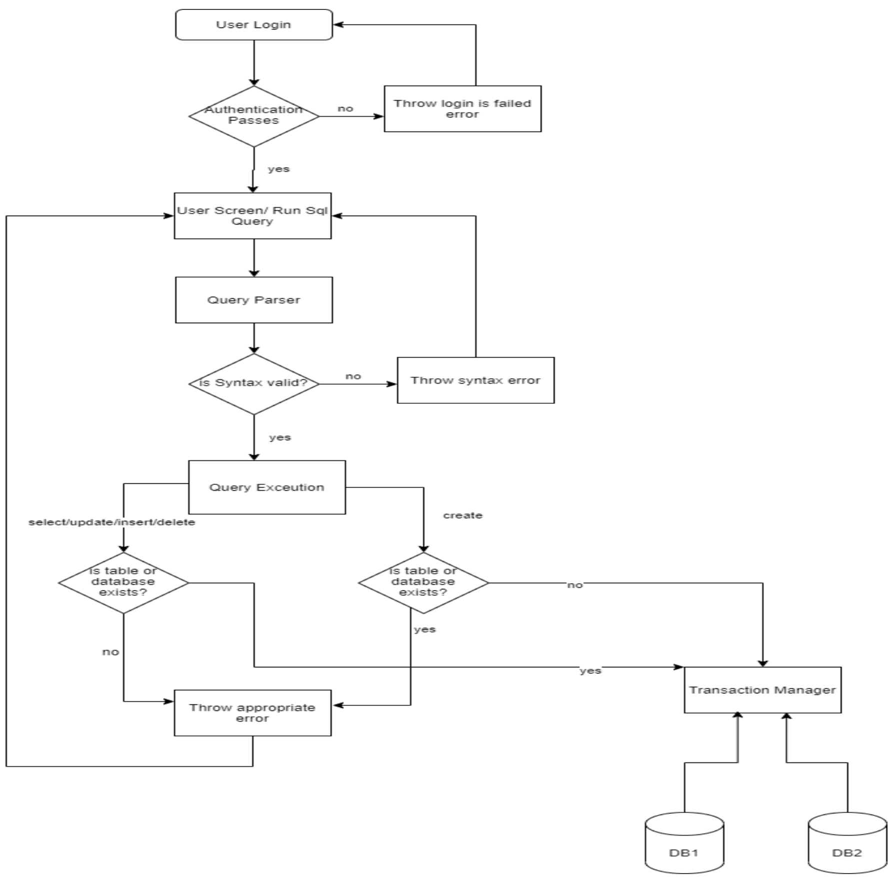
This document provides the feasibility study of Group Project -Building a simple distributed database and its management system by Group 10. It presents the information in sections. The first section shows the required technologies to implement it.

The second mentions how the various Structured Query Language commands are implemented and what kind of data structure is used to handle those data. Mainly we will have a parser and an implementation handler that does the job of communicating between the database client and the physical storage layer. The third section shows the flow chart of the working of DBMS. The fifth section outlines the test cases, sixth contains meeting logs and the seventh one describes our tentative project plan.

Required Technologies

- **Java:** We will use Java programming language as our tool to implement the simple distributed database management system. Also, Java can help us structure our code well. The help and support for Java is humungous and with the team members having strong experience in Java can help us work more on the implementation logic of the project rather than the learning the about the programming language. For a distributed system concurrency and asynchronous nature are needed and Java provides that.
- Other than Java we plan to do regular sprints using JIRA software.
- We will use JetBrains IntelliJ as our preferred IDE to work on the project.
- We will use GitLab as our version control system and submission of the source code.

Flowchart



Implementation Algorithm, Pseudo-code, and Data structures

Login Functionality

- User will enter username and password for login.
- System will fetch already saved username and password from the file storage and check if the user entered correct credentials.
- If user authentication fails, system will throw error.
- If user authentication fails, user will be prompted to run query.

Query Parser

- System stores the query run by user in the String variable.
- Then it will split the string with space. (As mentioned above in assumption that this system expects user to enter query by putting space between each word.)
- Check if the SQL clauses spellings are correct. If fails, throw error to user.
- Check if the SQL clauses in the query are in correct order. If fails, throw error to user.
- Finally, it will extract operation type, table name, column names, condition information from the query and store it in an appropriate class object and pass it to the Query handler.

Query implementation handler

CREATE

Database

- This will have the inputs like database name and the access control associated with it.
- Then when the handler receives the database name, username and password, those details are temporarily stored in a **queue** which acts as a buffer. As first come first served priority need to be maintained.
- Then the items are popped from the queue and a separate folder is created for each database and access permission for that folder are stored as a **meta.csv** file inside that folder because of the tabular structure provided by csv file.

Table

- This command will get the database name, table name, number of columns, constraints (Primary Key, Foreign Key) for which the table must be created.
- This information is then stored in a simple two-dimensional array. As **two-dimensional array** visualises the table data, it is easy to adapt it in this case.
- Since the mapping for the parameters is going to be constant, like 0 will contain the table name and parameters are stored in the subsequent array indexes.
- We will expand the data type and constraints horizontally. For now, this array will have only 3 columns. One for type, one for primary key and another for the foreign key. Each row represents the fields with 0 representing the table name.
- Then the data from array is used to create a csv file under the database. Before creating the csv file for the table folder exist check is performed to maintain the consistency.
- This command also enters a new entry in the meta.csv file present in the folder directory to make is table exist search faster in future.

INSERT:

- This handler will be called from the parser when user tries to add some records to their table.
- Assuming the insert values are in the same order of the fields in the table each value are passed as an element of array.

- An **array** will be a contiguous allocation of data and can be used here as the values are passed in the same order used while table creation.
- A basic retrieval of all the tables is done from the folder structure and stored in a **HashMap** because of its nature to store the String as a key and most of the table name will be String.
- This HashMap searches the filesystem for the database (folder) in which the user is performing the insert operation and keeps the table name as key and Boolean value with it.
- Then first index of the received array will have the table name and it is validated against the HashMap constructed.
- If key exists then we can proceed for the insertion, else will respond a failure object to the parser module which again parses the response and outputs the user proper error message.
- Once validation is done then the values corresponding to fields are written in the new line of CSV file with each field separated by commas.
- While insertion primary key constraint is also checked, and appropriate error handling is performed.

UPDATE

- In this operation the handler receives the table name, attribute name, attribute value and condition arguments.
- The conditional variables are the search indexes to find the appropriate row to do a modification.
- Assuming the table is present and based on the search index, the record is fetched from the csv and populated in a **HashMap** as it supports key to hold String value. This requires HashMap because HashMap are mutable.
- So, the value is modified for the appropriate key and again the values are load into the csv file with the help of a Primary key index if present to fasten up the process.
- If there are multiple conditions are present for the update record, then conditions are looked up sequentially and the records are loaded into the HashMap.

READ

- When the parser calls for the read handler it takes table name, search conditions as an argument.
- Assuming the file with the table name is found inside the directory the contents of the csv file are read into a **two-dimensional array**.
- The columns are in the order which is fetched from the folders meta data. Then in the array string matching algorithms like KMP is performed in case of where condition and LCS is performed in case of LIKE operator.
- When field matching such value is found, the matching rows are put into a new array and sent to the parser which parses the content and displays in a clear manner to the user.

DELETE:

Row

- The handler receives the table name and conditions as argument. Then the table name is looked up in the folder meta file.
- Assuming the table name is present the contents of the csv file is again read into a two-dimensional array (**Matrix**).
- The two-dimensional array is used to hold the data because of the faster time to access the data and perform various search algorithms in o (log n) times on the data structure.
- If such record is present, then the row indices are stored in a one-dimensional array and then those lines are deleted in the file swiftly to perform the delete operation.

Drop a table [Includes truncate case logic too]

- The handler receives the table name as an argument. The meta csv file of the database is parsed and stored in a **HashMap** because of its ability to support String as a key.
- Assuming the table name is present as a key, the file inside the folder is completely deleted and the entry in the meta csv file is also removed.
- The only difference between truncate and drop will be, in case of truncate the table entry in meta csv will not be deleted but the file is deleted. But in case of drop the entry in meta csv and file both are deleted.

Transaction Manager

- This module makes sure Atomicity, consistency, Integrity and Durability properties are kept in intact.
- This is achieved by logging the user action under each database whenever a sequence of database operations is performed. The log will be written in another log.csv file inside the database folder.
- As transaction manager handler fetches those details and flags the tables state in the meta.csv to warn against lost update problem, deadlock prevention, etc.

Connectivity: Socket programming in Java is used to establish network between the databases.

Reverse Engineering: Since the meta.csv file is stored inside each database folder, that can be easily used to construct the ER diagram. The cardinality is derived with the help of Foreign Key constraints mentioned in the meta file.

SQL Dump: Our application zips the folder along with the meta and table csv inside it and creates the dump. This ZIP file can be extracted by our program and restore the schema on any system by the proposed DDBMS.

Test Cases:

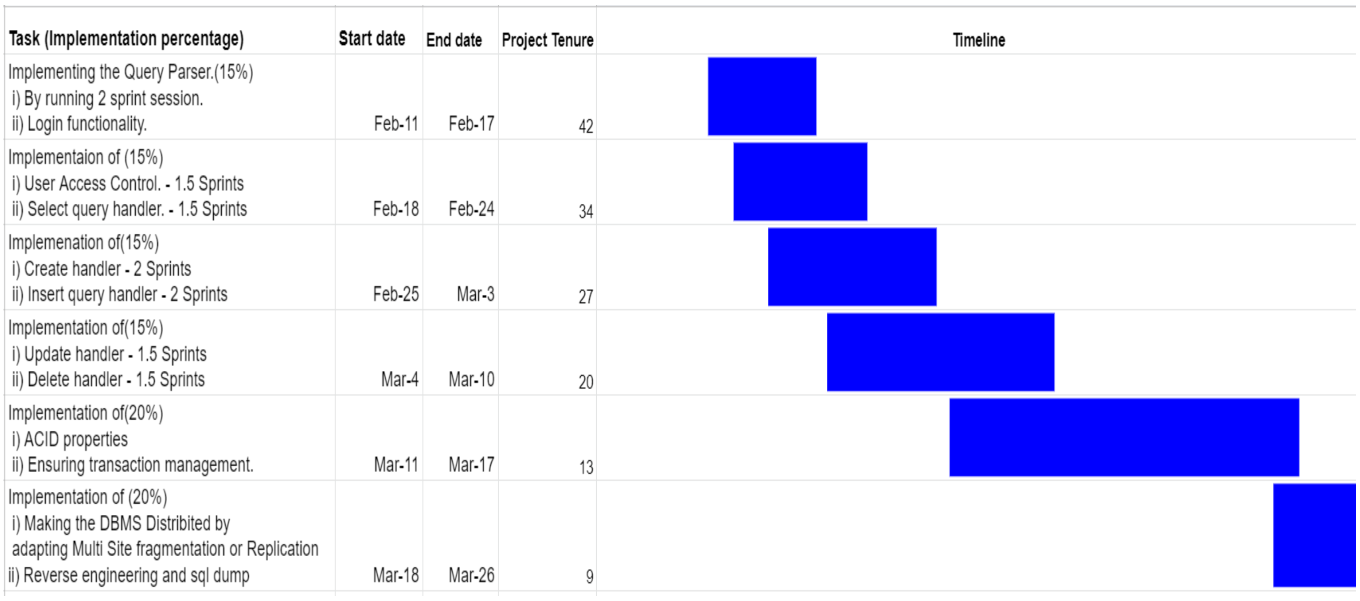
1. Wrong SQL clause spelling. Example – SLCT * FROM table_name.
2. Order of SQL clauses in query are wrong. Example – SELECT * WHERE id =1 FROM table_name.
3. Check user has permission for accessing a database while login.
4. Table name is not present in DB for Select, Delete, Insert and Update Query.
5. Column name is not present in DB for Select, Delete, Insert and Update Query.
6. Table name is present in DB for Create query.
7. Violating primary key constraint of table by inserting or updating.
8. Violating the foreign key constraint of table during inserting or updating.
9. Lost Update Problem in Transaction Management.
10. Uncommitted data problem in Transaction Management.

Meeting Logs

- **Meeting 1** - 05.02.2021 [6:03 P.M to 6:37 P.M IST] - Group Introduction and share our ideas on Database management system.[meeting1.png]
  - **Agenda**: We discussed on things like what we know about DBMS. Discussed on the components like Query parser, Query Optimizer and Storage engine.
- **Meeting 2 -** 08.02.2021[10:34 A.M to 11:29 AM] - Decide on modules and implementation parts.[meeting2.png]
  - **Agenda**: We broke down the implementation part in to 5 components and shared the researched ideas over the past weekend. Then started to work on our rough draft. Parallelly explored the detailed implementation of storage engine.

- **Meeting 3** - 09.02.2021[8:10 P.M to 8:37 P.M] - Check on the status of each one's documentation completion.
    - **Agenda**: discussed and settled on a time to work on our flowchart during the same day. Shared few ideas on what to include on the documentation and cleared our heads on the project requirements and marking rubrics of the idea proposal phase.
- **Meeting 4** - 09.02.2021[1 hour 25 mins] - Completion of the Flowchart.
    - **Agenda:** We discussed on the flowchart of the proposed implementation. Explained each other about their work completed in documentation and finalized the submission timing.
- **Meeting 5** – Feb 10[49 minutes 33 seconds] - Discussion on deliverables & Document peer review
    - **Agenda:** We discussed on the weightage of each implementation and settled on the documented implementation plan as below.

Gantt Chart



Task (Implementation percentage)	Start date	End date	Project Tenure	Timeline
Implementing the Query Parser.(15%) i) By running 2 sprint session. ii) Login functionality.	Feb-11	Feb-17	42	
Implementaion of (15%) i) User Access Control. - 1.5 Sprints ii) Select query handler. - 1.5 Sprints	Feb-18	Feb-24	34	
Implemenation of(15%) i) Create handler - 2 Sprints ii) Insert query handler - 2 Sprints	Feb-25	Mar-3	27	
Implementation of(15%) i) Update handler - 1.5 Sprints ii) Delete handler - 1.5 Sprints	Mar-4	Mar-10	20	
Implementation of(20%) i) ACID properties ii) Ensuring transaction management.	Mar-11	Mar-17	13	
Implementation of (20%) i) Making the DBMS Distribited by adapting Multi Site fragmentation or Replication ii) Reverse engineering and sql dump	Mar-18	Mar-26	9	