

# CSCI5408 – Data Management Warehousing Analytics (Sec01) – 2021 Winter

Group Project Proposal

Group Number - 16

**Team Members:**

Fenil Shah(B00871894)

Hashik Donthineni (B00868314)

Sravani Pinninti(B00884260)

## Project Proposal

### Programming Language

We opted for Java as the programming language to build the DBMS and the main reasons that lead to this decision are:

- Better performance, platform Independent and support for concurrency are the main features that we preferred Java.
- As the majority of the team members are well versed with Java, we figured it would be much simpler

### Assumptions

- User can use this system using only the below functionalities with the Standard SQL structure
  - CREATE
  - INSERT
  - UPDATE
  - DELETE
- User must use the following data types: Integer, Real Number & Text.

### Algorithms

These are the data structures and their corresponding purpose in our project:

#### HashMap

To store metadata. For faster access time of the data  $O(1)$  and the linear-search would take  $O(N)$

#### B+ Tree

For indexing and faster query processing.

- It has  $O(\log N)$  complexity for insertion. Construction of the tree from the file with  $N$  record would take  $O(N \log N)$  time.
- Searching a tree would take  $O(\log N)$  and so does deletion.

#### Why B+ Tree why not B Tree?

- B+ Trees are much easier for finding the range of records as they form LinkedList (or DLL) with the record at the leaf nodes. If we want to find a range in the Tree it would take  $O(\log N)$  for the search of the first element and  $O(R)$  for traversing through the range- $R$ .  $O(\log N + R)$
- B+ Trees are faster to convert into a file as we can traverse from node-to-node  $O(N)$ . We only have to find the first leaf.

## Basic Implementation

We assume that we had a class with a B+ tree implemented using LinkedList at the leaf nodes. Since it's a standard B+ tree with tweak and as it's very big, we didn't add the pseudo-code for that. We are assuming 1 file is 1 table.

```
/**
 * Key can be the primary key or generated ID
 * Row is the values list we want to enter.
 */
addRow(key, row){
/* At this point we assume that the user inputted "row"
values are checked for errors. */
btree.insert(key, row); //O(logN)
}
```

```
/**
 * Making a B+ tree from the file
 */
insertRow(FileObjectItr){
    // We can implement an iterable on file-object.
    while(FileObjectItr.hasNext()){ // O(N logN)
        btree.insert(FileObjectItr.row().key(),
            FileObjectItr.row()); // O(logN)
    }
}
```

```
/**
 * User commits the changes. Writing to the file from the BTree.
 */
commit(file){
    // ID 1 or with a primary-key. We store this when we read the file.
    //Gives the first node of the left-most leaf.
    node = btree.find(table.initialID());
    while(node != null){ // Traversing through Linked-list
        file.write(node.values())
        node = node.next;
    }
}
```

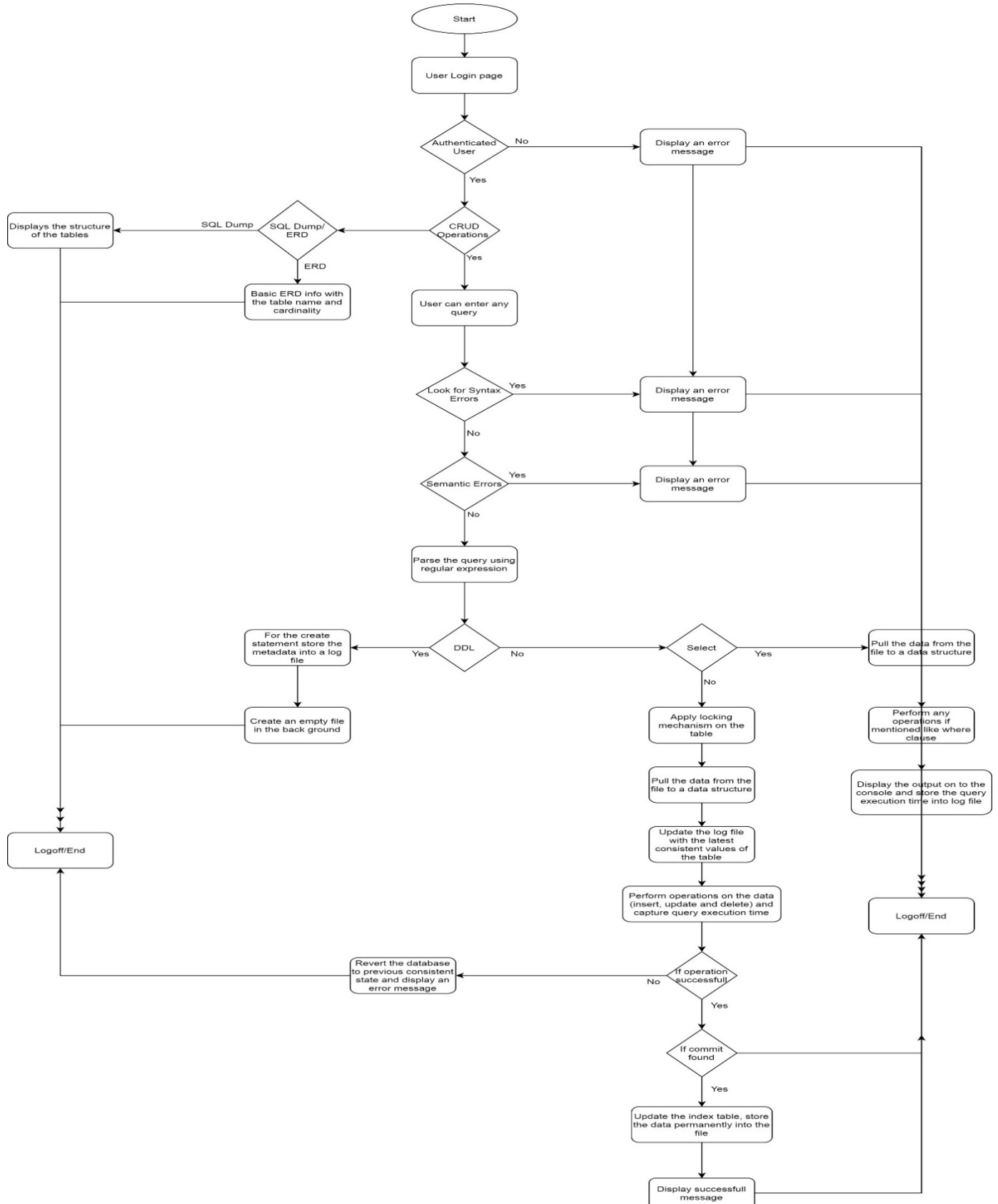
For **SELECT** this is not the efficient way to fetch the values, takes  $O(N)$  with indexing from B tree to B+ tree we can bring down the time to  $O(\log N)$  but indexing is not in the scope of the project, so implemented the naïve approach.

```
/**
 * SELECT with WHERE
 */
select(Conditions){
    node = btree.find(table.initialID()); //Explained above
    ArrayList rows //List to store all the matching rows

    while(node != null){ //For every row.
        satisfy = true;
        // For every condition in conditions
        for(condition : Conditions){
            // If a particular condition fails on that row
            if(checkCondition(node.values(), condition) == false){
                false;
                break;
            }
        }
        if(satisfy)
            rows.add(node.values())
    }
    return rows;
}
```

**UPDATE** will be similar to **SELECT** with the condition. We find all the rows that match that condition and update the Node with the values are given using the Node pointer.

## Flow Chart:



## Test Cases for Unit Testing

### Login

- The user needs to provide the correct username and password for login.
- If the credentials are incorrect, then the user will be shown the error message as follows:

```
>> Enter Username: <uname>
>> Enter Password: <password>
Incorrect username / password!
```

### Syntax Errors

- The syntax of the user input SQL query will be checked which should follow the Standard SQL structure.
- If the syntax of the query is not as per the standards, then an error message will be shown as below:

```
>> CREATE table1(Col1 datatype1, Col2 datatype2, Col3 datatype3);
Incorrect Syntax near keyword CREATE!
>> CREATE TABLE table1(Col1 datatype1, Col2 datatype2, Col3 datatype3);
Command executed successfully.
```

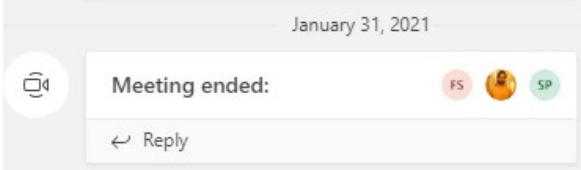
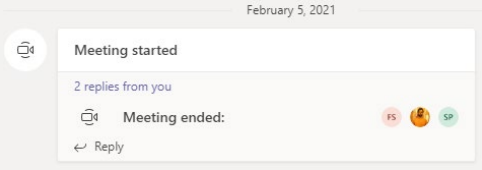
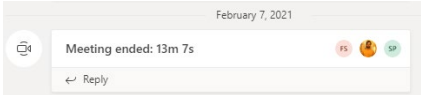
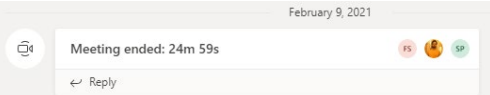
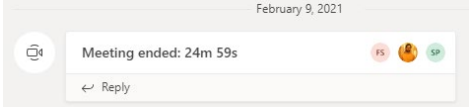
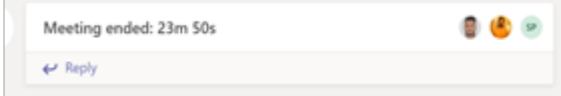
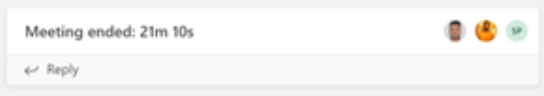
- If the user query follows the standard SQL structure, the query will be executed and the relevant success message will be displayed.

### Semantic Errors

- After checking the syntax of the user query, the system will check for the semantics of the query and if there are some errors then the user will be shown the below errors:

DML Statement	Semantic errors	Error message to be shown
Inserting data records	The table does not exist	Invalid object name " <i>tablename</i> "
	Less values provided than no. of columns	Less values provided than expected!
	Check for Primary key constraint	Primary key column cannot have duplicate /Null values.
Selecting Data Records	Incorrect column name	Invalid column name " <i>col_name</i> "
	Incorrect table name	Invalid object name " <i>tablename</i> "
Updating Data Records	Incorrect table name	Invalid object name " <i>tablename</i> "
	Incorrect column name	Invalid column name " <i>col_name</i> "
Deleting Data Records	Incorrect table name	Invalid object name " <i>tablename</i> "
	Incorrect condition	Invalid column name " <i>col_name</i> "

## Our Discussions and the Agenda

<p><b>Meeting 1 – Jan 31st</b></p>  <ul style="list-style-type: none"> <li>• We introduced ourselves, our background, and the technical skills we are aware of. And then we discussed the requirement of the project in detail so that we are all on the same page about the outcome of the project.</li> </ul>	<p><b>Meeting 2 – Feb 5<sup>th</sup></b></p>  <ul style="list-style-type: none"> <li>• We brainstormed more on identifying and dividing the project into modules</li> </ul>
<p><b>Meeting 3- Feb 7th</b></p>  <ul style="list-style-type: none"> <li>• Our main intention for that meeting was to finalize the programming language that we are going to use for the project. And we also took an action item to gather information about data structures that we feel would suit the requirement.</li> </ul>	<p><b>Meeting 4 -- Feb 9<sup>th</sup></b></p>  <ul style="list-style-type: none"> <li>• We presented all the information we gathered about the data structures, we eliminated a few of the data structures after discussing pros and cons but ended up with a couple of them.</li> </ul>
<p><b>Meeting 5 – Feb 9<sup>th</sup></b></p>  <ul style="list-style-type: none"> <li>• Exchanged our views on parsing logic and about implementing ACID properties.</li> </ul>	<p><b>Meeting 6 – Feb 10<sup>th</sup></b></p>  <ul style="list-style-type: none"> <li>• Discussed the approach for storing the data permanently.</li> </ul>
<p><b>Meeting 7 – Feb 10<sup>th</sup></b></p>  <ul style="list-style-type: none"> <li>• Finalized the data structure and distributed the documentation work.</li> </ul>	

### References:

- <https://www.postgresql.org/docs/9.2/indexes-types.html>
- [https://docs.oracle.com/cd/E11882\\_01/server.112/e40540/indexiot.htm#CNCPT721](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CNCPT721)