



DALHOUSIE
UNIVERSITY

Data Management, Warehousing and Analytics
Assignment 2

Name: Yogish Honnadevipura Gopalakrishna

Banner ID : B00928029

GitLab :

https://git.cs.dal.ca/yogish/csci5408_s23_b00928029_yogish_honna-devipura-gopalakrishna.git

Problem 1

In the given paper, distributed systems and distributed databases has been explained. It starts with talking about computer networks with shared properties, such as the Internet, mobile phone networks, and corporate networks. By distributing or duplicating data over numerous physical locations, distributed systems seek to offer increasing data to various query points.

A single logical database is physically dispersed across multiple computers linked by a data communications network in a distributed database. Data distribution processes include fragmentation, which involves breaking up the data into smaller pieces, replication to make copies of the data, and allocation procedures. Despite sharing a space allocation and a network address, communication between distributed database sites happens over a network rather than shared memory.

Initial design, redesign, and materialisation are the three stages of the distributed database design process. Algorithms for allocation and fragmentation are used in the initial design stage to reduce the cost of transaction processing. New fragmentation and allocation methods are created during the redesign process to account for modifications to the distributed database environment. Implementing the new design through required operations is known as materialisation.

The allocation and fragmentation schemes in distributed database systems are the main topics of the aforementioned research papers. They offer a number of strategies and methods to improve the performance, availability, and dependability of such systems. The research examine clustering strategies, fragmentation methods, vertical and horizontal fragmentation methodologies, and data allocation tactics. The objectives include increasing proximity of query assessment, lowering communication costs, enhancing data distribution, speeding up execution, and enhancing processing effectiveness. The advantages of distributed databases, including parallel processing, availability, dependability, and cost savings, are highlighted in the study.

The main aim of the fragmentation is to improve Reliability, efficient storage mechanisms, lowering communication costs and secure the data. The qualitative and quantitative information such as cardinality, type of data, site of the query are used while deciding the fragmentation.

Fragmentation in distributed databases can be categorised into three main types: horizontal fragmentation, vertical fragmentation, and mixed fragmentation. In horizontal fragmentation, a relation or class is divided into disjoint tuples or instances. Each fragment contains a subset of rows, and each fragment is stored at a different node in the distributed system. The unique rows within each fragment have the same attributes (columns). In vertical fragmentation, a relation or class is partitioned into separate sets of columns or attributes, excluding the primary key. Each set must include the primary key attributes of the table. Mixed fragmentation combines both horizontal and vertical fragmentations. The table is divided into blocks based on specific requirements. Each fragmentation may be allocated to a specific site. Mixed fragmentation is more complex and requires additional management.

The correctness rules of fragmentation in a distributed database system ensure that the process of dividing a relation into fragments is done correctly and maintains the integrity of the data. Completeness ensures that when a relation is divided into fragments in a distributed database system, none of the data is lost during the process. Reconstruction refers to the ability to reconstruct the original relation from its fragments. Disjointness focuses on the distribution of data items among the fragments. It ensures that each data item, whether it's a tuple or an attribute, is assigned to one and only one fragment, avoiding duplication or inconsistency.

The suggested fixes for distributed database systems' fragmentation cover a number of important topics. With the help of effective indexing, query optimisation, and caching techniques, performance optimisation strives to improve query processing and data access. Data distribution can be adjusted based on shifting access patterns and shifting workloads thanks to dynamic fragmentation. Scalability is promoted through load balancing methods, which ensure uniform resource utilisation across fragments and nodes. Replication techniques are used to increase fault tolerance and data availability and resilience. In this way this paper presents an introduction to distributed databases and distributed database design with the explanation about the different types of fragmentation, and also discusses their benefits and drawbacks.

Problem 2

SOLID Principles

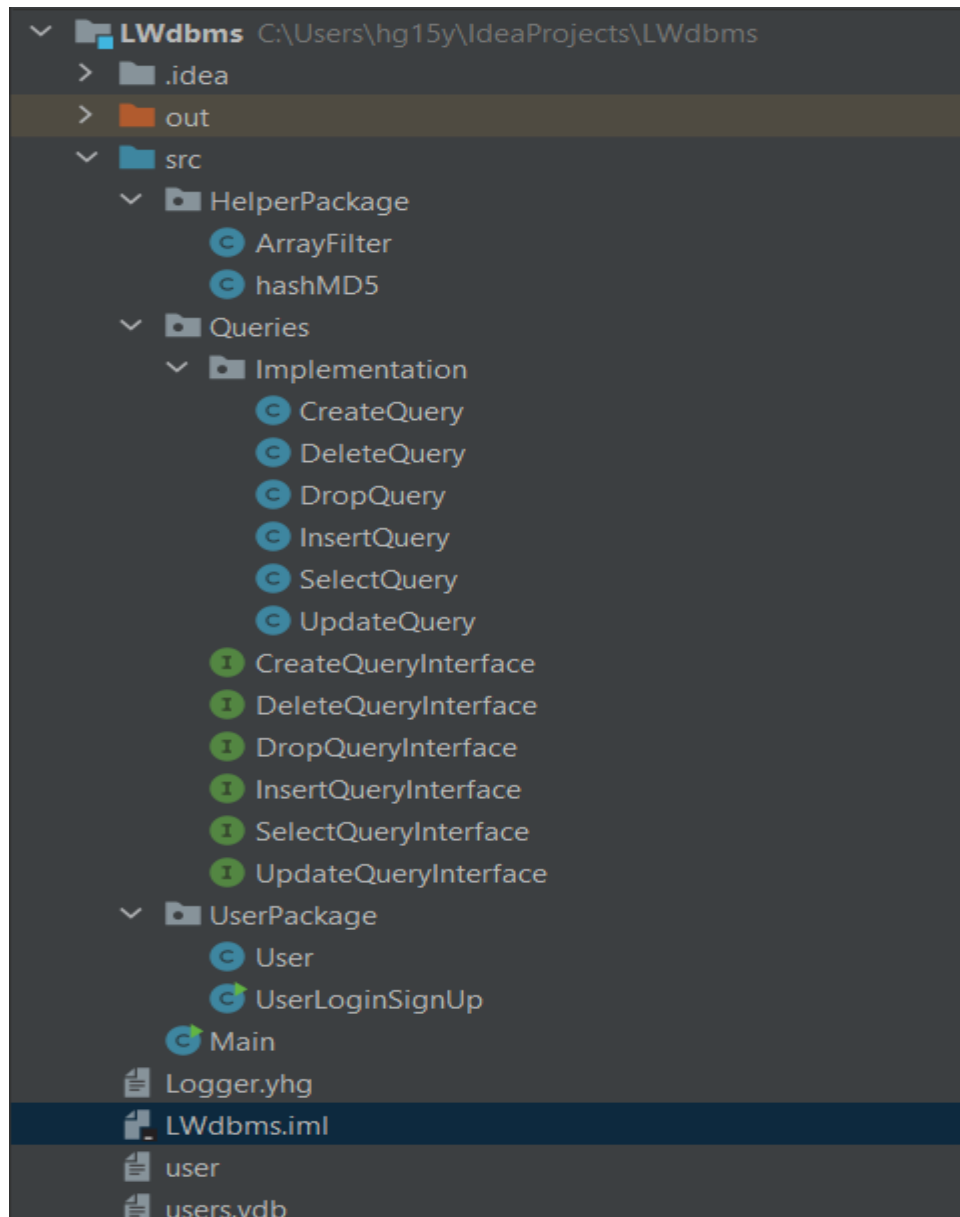


Fig : File Structure

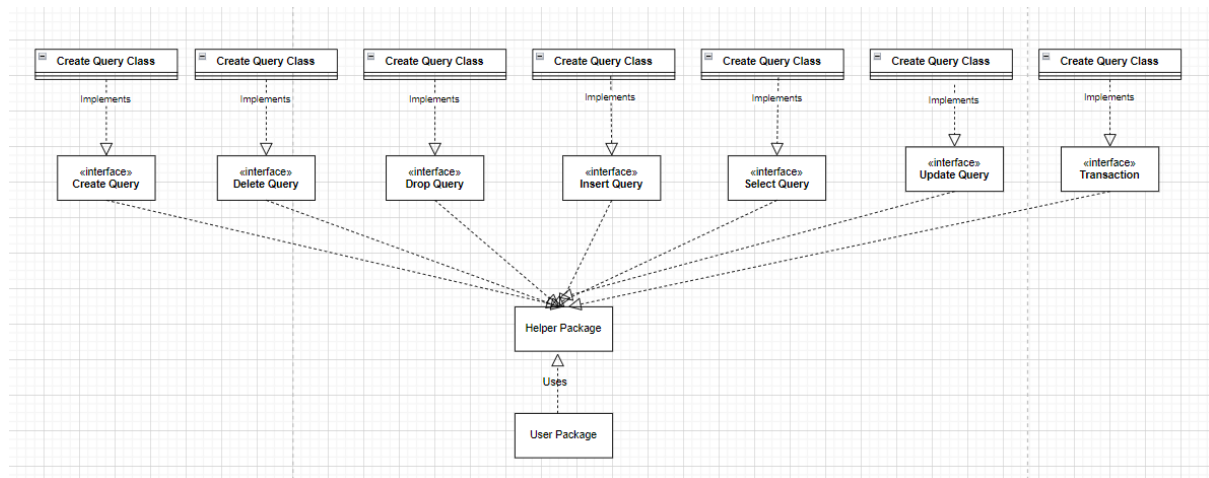


Fig: SOLID Principles Structure

The code adheres to the solid principles in the following ways

- **Single Responsibility** - All the classes in the file do only one thing. Ex: There are separate classes for executing every query so that each class is concerned with a single responsibility.
- **Interface Segregation Principle** - There are multiple interfaces where each interface is concerned with an individual concern so that a class implementing an interface need not implement a method which is of no use.
- **Dependency Inversion** - The main file is dependent on the interface rather than directly depending on the class so that dependency inversion is achieved
- No classes are implementing the methods which does not bring value into the program

Authentication and Logging

The program is asking for UserID and password from the user. If the userID and the password is authenticated from the list of stored credentials in the file, if they match, security question is asked which is also checked. If it matches, only then the user is allowed to query the database. This achieves the two factor authentication

The password is **hashed** using **MD5** so that the password is secure and every time an user does an operation to the table, a log is saved into a file so that every operation is recorded.

```

aa,4124bc0a9335c27f086f24ba207a4912,aa
user1,4124bc0a9335c27f086f24ba207a4912,aa
  
```

Fig : User credentials stored with password hashed

```
2, Virat, 18
3, Dhoni, 7
```

Fig : Table details file

```
1 User: aa has inserted a row to the table user at 2023-07-11 21:21:14.714
2 User: aa has created a new table called employees at 2023-07-11 21:59:11.479
3 User: aa has inserted a row to the table employees at 2023-07-11 21:59:52.77
```

Fig : Log File

Queries

Create Query

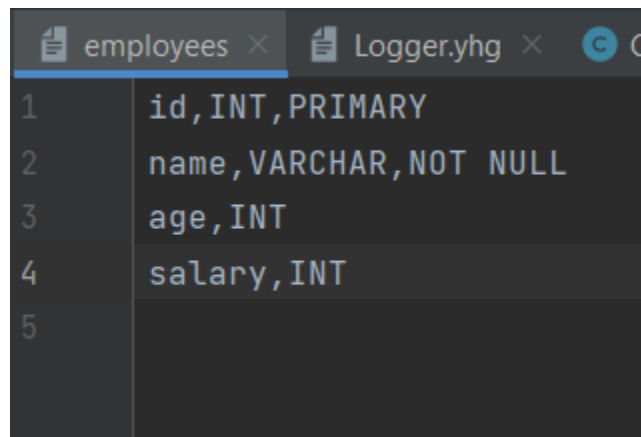
The create query can be used to create tables and the code breakdown is as follows:

- The CreateQuery class is defined and it implements the CreateQueryInterface.
- The createTable method is defined with the purpose of creating a table based on the provided query.
- The method begins by filtering and splitting the input query2 using the ArrayFilter class, storing the result in query3 and query arrays respectively.
- The name of the database is extracted from the query array and stored in the databaseName variable.
- A File object is created with the databaseName.
- If the file already exists, the method returns false indicating that the table creation failed.
- If the file doesn't exist, the method proceeds to create the table by writing the table structure to the file.
- It uses a PrintWriter wrapped around a FileWriter to write to the file in an append mode (the second parameter of FileWriter constructor is set to true).
- The method iterates over the elements of the query array and processes them to generate the table structure.
- Inside the loop, it checks various conditions to determine the type of table column, such as "VARCHAR," "PRIMARY," "DEFAULT," "UNIQUE," and "NOT NULL."
- Based on the conditions, the appropriate information is written to the file using the writer object.
- The loop continues until all the elements of the query array have been processed.
- If any exception occurs during the file writing process, the exception message is printed to the console, and the method returns false indicating that the table creation failed.
- If the table creation is successful, the method returns true.

The create statement can handle Primary Key, Not null, Default, Unique.

```
CREATE TABLE employees ( id INT PRIMARY KEY, name VARCHAR(100) NOT NULL, age INT, salary INT);  
Enter your query
```

Fig: Running the create statement



The screenshot shows a database IDE with two tabs: 'employees' and 'Logger.yhg'. The 'employees' tab is active, displaying the table structure. The table has five columns: 'id' (INT, PRIMARY), 'name' (VARCHAR, NOT NULL), 'age' (INT), 'salary' (INT), and an empty column. The columns are listed in a table with line numbers 1 through 5 on the left.

Line	Column
1	id, INT, PRIMARY
2	name, VARCHAR, NOT NULL
3	age, INT
4	salary, INT
5	

Fig: MetaFile Created

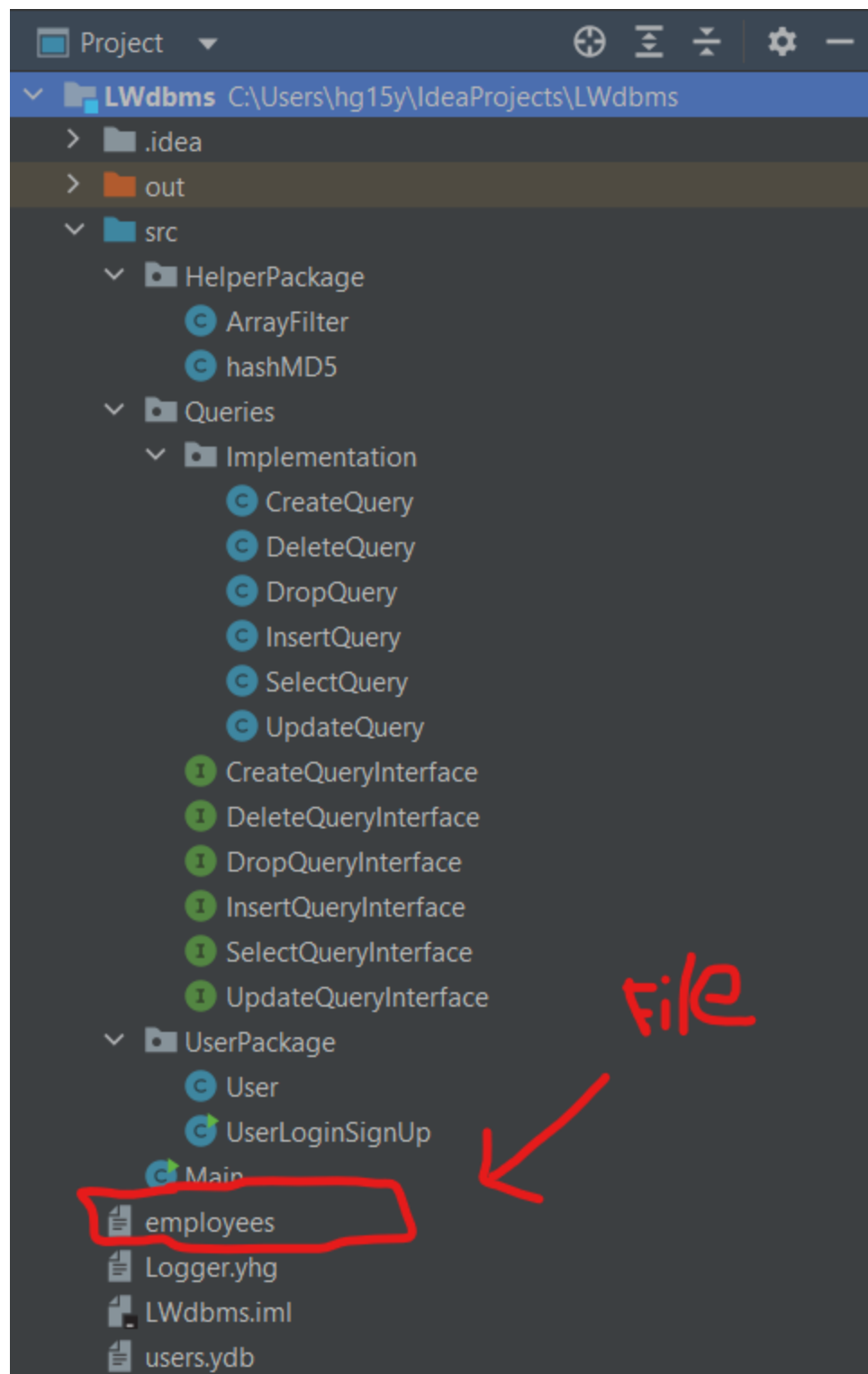


Fig: File created in project structure

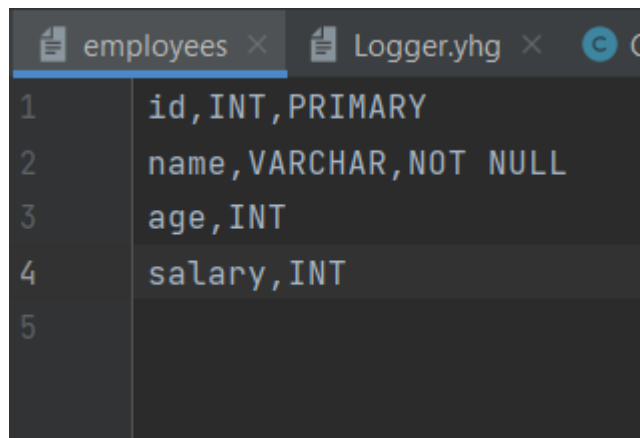
Insert Query:

Breakdown

- The code defines an InsertQuery class that implements the InsertQueryInterface. This class is responsible for inserting values into a table based on the provided query.
- The main method in the InsertQuery class is called Insert. It takes a query as input and performs the insertion operation.
- Inside the Insert method, several variables are initialized, including lineCount, exceptionFlag, and insertionArray. These variables are used to track the number of lines in the table, any exceptions that occur during the insertion process, and the values to be inserted, respectively.
- The input query is filtered and split using the ArrayFilter class, resulting in the query3 and query arrays.
- Two ArrayLists, columns and values, are created to store the column names and the values to be inserted, respectively.
- A File object is created with the name specified in the query, appended with "details". If the file doesn't exist, a new file is created.
- The method then processes the query array to extract the column names and values to be inserted. It iterates through the query array, starting from index 3, until it encounters the keyword "VALUES". It stores the column names in the columns list and counts the number of columns.
- If the keyword "values" is found and the column count is 0, it means that the column names were not explicitly provided. In this case, it reads the column names from the existing table file and adds them to the columns list.
- The index of the values to be inserted is determined by finding the index where "values" occurs in the query array. The values are then extracted from the query array and stored in the values list.
- Next, the method reads the existing table file to determine the line count (number of rows in the table) using a BufferedReader.
- It enters a loop to process each line in the table. Inside the loop, it checks if the line contains keywords such as "PRIMARY" or "UNIQUE" to identify primary key and unique key columns, respectively.
- If the line contains "PRIMARY," it reads the existing primary key values from the table file and stores them in a list called primaryKeyValues. If the line contains "UNIQUE," it reads the existing unique key values from the table file and stores them in a list called uniqueKeyValues.
- The method then retrieves the current column from the line and finds its index in the columns list. If the column is not found in the insert query, it handles different scenarios based on the line keywords (e.g., "PRIMARY," "NOT NULL," "UNIQUE," "DEFAULT") and adds the appropriate value or throws an exception.
- If the column is found and it is a primary key or unique key column, it checks for duplicate values. If a duplicate is found, it throws an exception. Otherwise, it adds the value to the insertionArray.
- After processing all the lines in the table, the size of the insertionArray is determined. If an exception occurred during the process, the size is set to -1, and the method returns false.

- If the size of the insertionArray matches the line count, it means that all the values were processed successfully. It then opens a PrintWriter wrapped around a FileWriter in append mode to write the values to the table file.
- The method iterates through the insertionArray and writes each value to the file, separating them with commas. After writing all the values, a new line is added.
- If an exception occurs during the writing process, the exception message is printed to the console.
- Finally, the method returns true to indicate that the insertion was successful.

Types of Insert queries by syntax



Line	Column Definition
1	id,INT,PRIMARY
2	name,VARCHAR,NOT NULL
3	age,INT
4	salary,INT
5	

Fig : MetaFile for Employees table

```
insert into employees values (1, hg, 22, 1000000);
```

Case 1 - All the values can be given to insert by not specifying the columns name. The constraint is that the values are to be given in the same order as the table created.

```
Insert into employees (id, name) values (111, hgwdasdssay);
```

Case 2 - less values can also be inserted into the database by specifying the columns name, but if the column is not null, a value has to to given and if a column has a default value specified and not given any value while insertion, the default value will be considered

```
Insert into employees (id, age, name) values (12, 32, hello1);
```

Case 3 - The columns name can be given in any order and the program handles the values inserted and will insert into that column with the help of metafile.

Fig : Types of Insert Syntax handled by the program

1	1	hgy	Null	Null
2	111	hgwdasdssay	Null	Null
3	12	hello1	32	Null
4				

Fig: Output for the above insert statements

Select Query

- The SelectQuery class is responsible for executing select queries on a database table. It implements the SelectQueryInterface, which means it provides the necessary implementation for select query functionality.
- The main method in the SelectQuery class is called SelectTable. It takes an array of strings called query2 as input, which represents the select query provided by the user.
- Inside the SelectTable method, two ArrayLists are created: colName to store column names and valueArrayList to store retrieved values from the database table.
- The query2 array is filtered and split into two separate arrays, query3 and query, using the ArrayFilter helper class.
- The code checks if the second element of the query array is a wildcard "*" and the length of the query array is 4. If these conditions are true, it means the user wants to select all columns from the specified table.
- In that case, the code reads data from the specified file (provided in query[3]) assuming it contains comma-separated values. It retrieves the column names and adds them to the colName ArrayList. It also retrieves the values from the file and adds them to the valueArrayList.
- After retrieving the data, the code prints the column names and values in a nicely formatted manner.
- If the conditions mentioned above are not met, the code checks if the length of the query array is greater than 4 and the second element is a wildcard "*", while the fifth element is the keyword "where". If this condition is true, it means the user wants to select specific rows based on a condition.
- In this case, the code reads data from the specified file (provided in query[3]) and retrieves the column names, adding them to the colName ArrayList.
- The code then extracts the condition and value from the query, storing them in variables called cond and val, respectively.
- Next, the code reads a separate details file (created by appending "details" to the original file name) and compares the value at the specified column index with the given condition. If they match, it prints the column names and the corresponding row values.
- Finally, the SelectTable method returns true to indicate that the select query execution was successful.

```
select * from employees;
```

id	name	age	salary
1	hgy	Null	Null
111	hgwdasdssayNull	Null	Null
12	hello1	32	Null

Fig : Selecting all entries

```
select * from employees where id = 1;
```

id	name	age	salary
1	hgy	Null	Null

Fig: Selecting data based on condition

Update Query

Breakdown

- The UpdateQuery class implements the UpdateQueryInterface, indicating that it provides an implementation for the update query functionality.
- The main method in the UpdateQuery class is called Update. It takes an array of strings called query2 as input, which represents the update query provided by the user.
- The method initializes some variables, including query3 and query, which are obtained by filtering and splitting the query2 array using the ArrayFilter helper class. It also creates an ArrayList called colName to store column names.
- The code then attempts to read the table file, details file, and another details file (for reading purposes) using BufferedReader objects. It assumes that the table file and the details file have the same name, but the details file is appended with "details".
- Inside a try-with-resources block, the code reads the table file line by line and splits each line by a comma (assuming comma-separated values). It adds the first element (column name) to the colName ArrayList and counts the total number of lines.
- The code extracts the values needed for the update operation, such as afterWhere (the column to be updated), index (the index of the column to be updated), beforeWhere (the column used as a condition), and index2 (the index of the column used as a condition).

- It creates an empty string array called resArray and initializes a counter variable called countRes.
- The code then reads the details file line by line and splits each line by a comma. If the value in the afterWhere column matches the value provided in the query, it updates the corresponding value in the beforeWhere column to the new value provided in the query. It stores the updated line in the resArray and breaks out of the loop.
- The updated line stored in resArray is converted to a string using String.join method and assigned to the resString variable.
- The code reads the details file again and keeps track of the line count using the resCount variable. It creates a StringBuilder called content to store the updated content.
- Inside the loop, if the line count (resCount) matches the counter of the matched line (countRes), it appends the updated line (resString) to the content. Otherwise, it appends the original line from the details file.
- After the loop, the code creates a BufferedWriter to write the updated content to the details file. It writes the content using the writer.write method, flushes the writer, and closes it.
- If any exception occurs during the process, the exception message is printed to the console, and the method returns false to indicate a failure.
- Finally, if the update operation completes successfully, the method returns true to indicate success.

```
Update employees set name = hello2 where id = 111;
```

Fig : Update query execution

users.ydb × employees × employee	
1	1,hgy,Null,Null
2	111,hello2,Null,Null
3	12,hello1,32,Null
4	

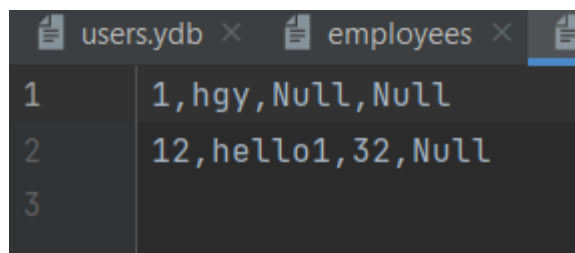
Fig : Result of update query

Delete Query

- The DeleteQuery class implements the DeleteQueryInterface, indicating that it provides an implementation for the delete query functionality.
- The main method in the DeleteQuery class is called delete. It takes an array of strings called query2 as input, which represents the delete query provided by the user.
- The method initializes some variables, including query3 and query, which are obtained by filtering and splitting the query2 array using the ArrayFilter helper class. It also creates an ArrayList called colName to store column names.
- The code then attempts to read the table file and the corresponding details file using BufferedReader objects. It assumes that the table file and the details file have the same name, but the details file is appended with "details".
- Inside a try-with-resources block, the code reads the table file line by line and splits each line by a comma (assuming comma-separated values). It adds the first element (column name) to the colName ArrayList and counts the total number of lines.
- The code extracts the value needed for the delete operation, which is the value to be matched in the column specified in the query.
- It creates a StringBuilder called content to store the updated content.
- The code then reads the details file line by line. For each line, it splits the line by a comma and checks if the value in the specified column matches the value provided in the query. If they match, the line is skipped (essentially deleting it). If they don't match, the line is appended to the content.
- After processing all the lines in the details file, the code creates a BufferedWriter to write the updated content back to the details file. It writes the content using the writer.write method, flushes the writer, and closes it.
- If any exception occurs during the process, the exception message is printed to the console, and the method returns false to indicate a failure.
- Finally, if the delete operation completes successfully, the method returns true to indicate success.

```
Delete from employees where id = 111;
```

Fig : Delete Query



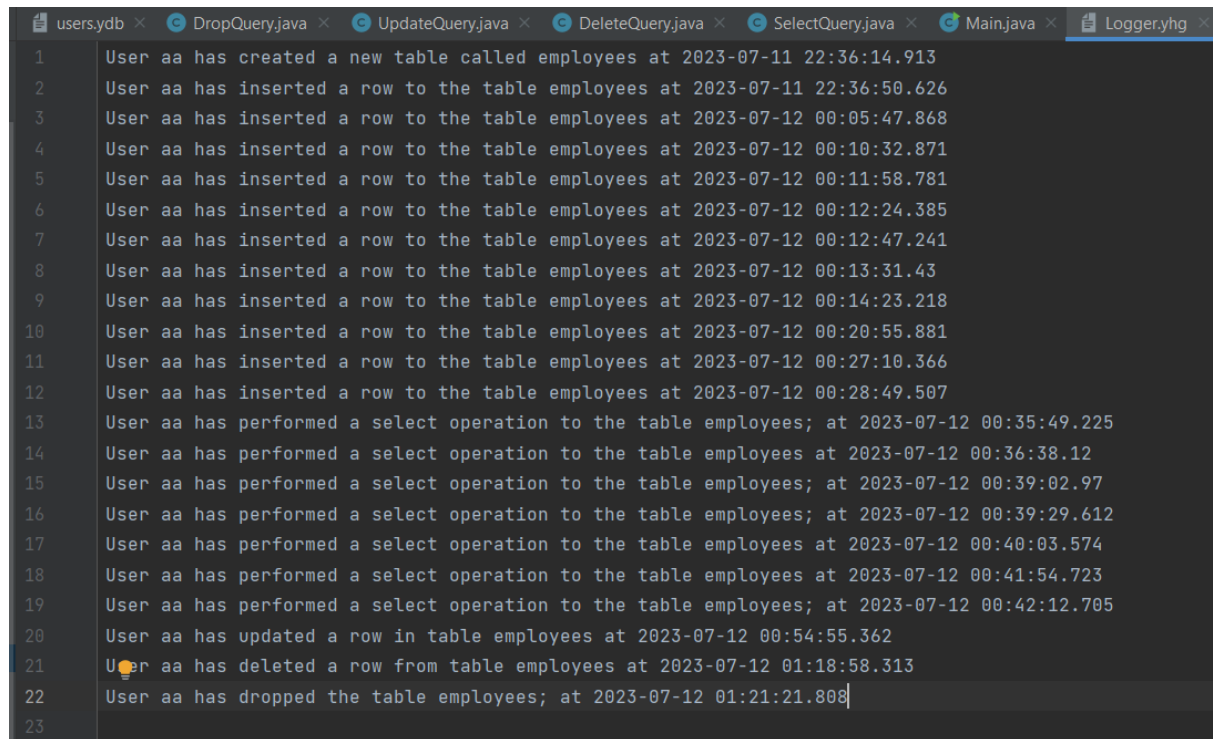
1	1,hgy,Null,Null
2	12,hello1,32,Null
3	

Fig : Result of Delete query

Drop Table:

The drop table drops the whole table meaning it will delete the meta file as well as the details file from the project structure. This operation has to be executed by stopping and running the program again.

The Whole log Operation

A screenshot of a Java IDE with multiple tabs open: users.ydb, DropQuery.java, UpdateQuery.java, DeleteQuery.java, SelectQuery.java, Main.java, and Logger.yhg. The Logger.yhg tab is active, displaying a log of database operations. The log entries are as follows:

```
1 User aa has created a new table called employees at 2023-07-11 22:36:14.913
2 User aa has inserted a row to the table employees at 2023-07-11 22:36:50.626
3 User aa has inserted a row to the table employees at 2023-07-12 00:05:47.868
4 User aa has inserted a row to the table employees at 2023-07-12 00:10:32.871
5 User aa has inserted a row to the table employees at 2023-07-12 00:11:58.781
6 User aa has inserted a row to the table employees at 2023-07-12 00:12:24.385
7 User aa has inserted a row to the table employees at 2023-07-12 00:12:47.241
8 User aa has inserted a row to the table employees at 2023-07-12 00:13:31.43
9 User aa has inserted a row to the table employees at 2023-07-12 00:14:23.218
10 User aa has inserted a row to the table employees at 2023-07-12 00:20:55.881
11 User aa has inserted a row to the table employees at 2023-07-12 00:27:10.366
12 User aa has inserted a row to the table employees at 2023-07-12 00:28:49.507
13 User aa has performed a select operation to the table employees; at 2023-07-12 00:35:49.225
14 User aa has performed a select operation to the table employees at 2023-07-12 00:36:38.12
15 User aa has performed a select operation to the table employees; at 2023-07-12 00:39:02.97
16 User aa has performed a select operation to the table employees; at 2023-07-12 00:39:29.612
17 User aa has performed a select operation to the table employees at 2023-07-12 00:40:03.574
18 User aa has performed a select operation to the table employees at 2023-07-12 00:41:54.723
19 User aa has performed a select operation to the table employees; at 2023-07-12 00:42:12.705
20 User aa has updated a row in table employees at 2023-07-12 00:54:55.362
21 User aa has deleted a row from table employees at 2023-07-12 01:18:58.313
22 User aa has dropped the table employees; at 2023-07-12 01:21:21.808
23
```

Transaction

Working

- It checks if the second element of the queryRes(the actual query) array is equal to "transaction".
- If it is, the code initializes some variables and writes a transaction start message to a file.
- It enters a loop that reads input from the scanner until it encounters a "rollback" command or reaches the end of input.
- Inside the loop, each input line is split into an array of strings using whitespace as a delimiter and added to the TransactionArray.
- If the first element of the query2 array is "rollback", the loop breaks, indicating the end of the transaction.
- If the first element of the query2 array is "commit", it proceeds to process the queries stored in the TransactionArray.
- Inside the transaction processing block, it iterates over each query in the TransactionArray.

- Depending on the type of query (create, delete, drop, insert, select, or update), it calls the corresponding interface implementation to execute the query.
- For each successful query execution, it writes an appropriate message to the file, indicating the user, table, and timestamp.
- After processing all queries, it writes a transaction end message to the file and breaks the loop.
- If the second element of queryRes is not equal to "transaction", it prints an error message indicating incorrect syntax or suggests exiting the program.

```
start transaction
create table t1 (id int primary key, name varchar(100), age int);
insert into t1 values (1, Person1, 22);
commit
Enter your query
```

Fig: Successful transaction

6	Transaction started at 2023-07-12 19:03:06.128
7	User aa has created a new table called t1 at 2023-07-12 19:04:38.572
8	User aa has inserted a row to the table t1 at 2023-07-12 19:04:38.582
9	Transaction ended at 2023-07-12 19:03:06.128

Fig : Log file for commit

```
start transaction
insert into t1 values (3, p2, 22);
rollback
Enter your query
```

Fig : Rolled back transaction

```
Transaction started at 2023-07-12 19:15:01.435
Transaction rolled back
```

Fig : log for rollback

References

- [1] *Indeed.com*. [Online]. Available:
<https://www.indeed.com/career-advice/career-development/data-fragmentation#:~:text=Data%20fragmentation%20is%20data%20that's,File%20shares>.
[Accessed: July 10, 2023].
- [2] *Baeldung.com*. [Online]. Available:
<https://www.baeldung.com/java-regex-text-after-match>.
[Accessed: July 10, 2023].
- [3] *w3schools.com*. [Online]. Available:
https://www.w3schools.com/java/java_regex.asp. [Accessed: July 09, 2023]
- [4] “Flowchart maker & online diagram software,” *Diagrams.net*. [Online].
Available: <https://app.diagrams.net/>. [Accessed: 13-Jul-2023].