



**DALHOUSIE  
UNIVERSITY**

**CSCI 5408**

**Data Management , Warehousing Analytics**

**Project Report Phase – 2**

---

**Group – 4**

Neelkanth Dabhi: B00848532

Samkit Shah: B00852292

Prasant Sarvi: B00870599

## Table of Contents

Team Members .....	4
Introduction .....	6
Conceptual Framework.....	9
Implementation of Main Module .....	11
Implementation of Query Parser .....	12
Implementation of Query Execution Engine .....	16
Overview: .....	16
Data Structures: .....	16
Pseudo Code .....	18
Implementation of Transaction .....	21
Implementation of Data Dictionary .....	22
Project Features .....	23
Test Cases.....	24
Conclusion .....	28
Future Enhancement .....	<b>Error! Bookmark not defined.</b>
References .....	30

## Table of Figures

Figure 1 Block Diagram of the system .....	7
Figure 2 Conceptual Framework.....	9
Figure 3 Query Parser Flow Diagram .....	12
Figure 4 Workflow of query execution engine .....	17

## Team Members

We are three team members. We have developed a database management system using Python.

The GitLab link for the project: <https://git.cs.dal.ca/ssshah/database-management-system>

<b>Name</b>	Samkit Shah
<b>Roles &amp; Responsibilities</b>	Samkit has worked as a backend developer throughout this project. He has successfully planned and developed Query Execution Engine, implemented Data Structures and methods to execute the query. Along with that, he has developed transaction functionality and integrated successfully with the project for multiple concurrent users. Along with that, built a mechanism to store data into the persistent form. He has proactively participated in the team to bring team members on the same page.
<b>New Learning</b>	Getting to know the working of the core engine functionality which is the backbone of the database which has been built in the application. Samkit got a detailed understanding about concept of the database management system and transaction. The most important thing samkit get to learn is that non-technical problems are the most difficult. And one more thing that never assumes that someone will do something just because you've asked them to. Always check for follow-up. And if you don't understand anything and working in a team then ask your teammates. Communication plays an important role when developing a project in a team.
<b>Difficulties</b>	The main difficulty was about the efficiency in executing the query. If we look for the non-technical factors then the limited time and time-zone difference between teammates, no face to face conversation.

<b>Name</b>	Neelkanth Dabhi
<b>Roles &amp; Responsibilities</b>	Neelkanth was mainly responsible for building parsers for Create Table, Create Database, Use Database, Insert into table queries. Also, he has built the main module that communicates between the query console, query parser, and the execution engine, and forwards messages to the correct method in both. He created the Logging system that exports log to a different file. Along with that, he has built a SQL dump feature that helps the user to rebuild the database using the dump. Moreover, he has developed user authentication, prettifying the python dictionary into a table like a format in the console. Color coding all the warnings, errors, and success messages in the console.

<b>New Learning</b>	This project was a huge learning for him as he gets to know about the internal working of the DBMS. For example, as he was building the parser, he got the idea about how a MySQL workbench would parse the queries. Moreover, build a console application was new for him, as he has only worked on web and mobile-based apps.
<b>Difficulties</b>	The greatest difficulty was to build a parser that handles boundary cases and make sure that all the modules can communicate efficiently with each other and functions smoothly.

<b>Name</b>	Prashant Sarvi
<b>Roles &amp; Responsibilities</b>	Creation of select parser which is used to group required keywords from the statement provided by the user. The select parser covers all the conditions which are provided and maintains logs. The parser also handles exceptions when occurred during execution. Providing a Data dictionary that dynamically tracks the metadata files which are generated during the creation of the table and stores all the aggregated metadata in a file. A file is creating dynamically where all the metadata of the application is stored, and it is featured in the form of a table. Creation of update parser in the application is used to analyze the statement and retrieve the required fields from it. Creation of delete parser which covers all the scenarios of the condition type and makes use of model classes in an appropriate way. Description of product features, conclusion & future work, algorithm for the parser and conceptual framework in the documentation.
<b>New Learning</b>	The complete walk through of the implementation side of the database provided more clarity on the core functionality of the database. Generating parser for each query statement helped to get a detail information on how the SQL statements works internally whenever it is executed. Practical implementation of the parser by using regular expressions. The in-depth clarity of the concept of RegEx has been attained by this project. Getting to know the working of the core engine functionality which is the backbone of the database which has been built in the application. Learning the usage of logs and the types of logs used based on the level of severity which are categorized in order to be keep track of the transactions More clarity regarding data dictionary and its practical implementation in the application and observing the aggregation of metadata generating dynamically. Learning about the new coding standards which are implemented in the application.
<b>Difficulties</b>	The virtual learning platform was kind of difficult to cope with the team. The inconvenience faced during meeting schedule and discussion in different time zones issues created disturbance in sleep cycle.

## Introduction

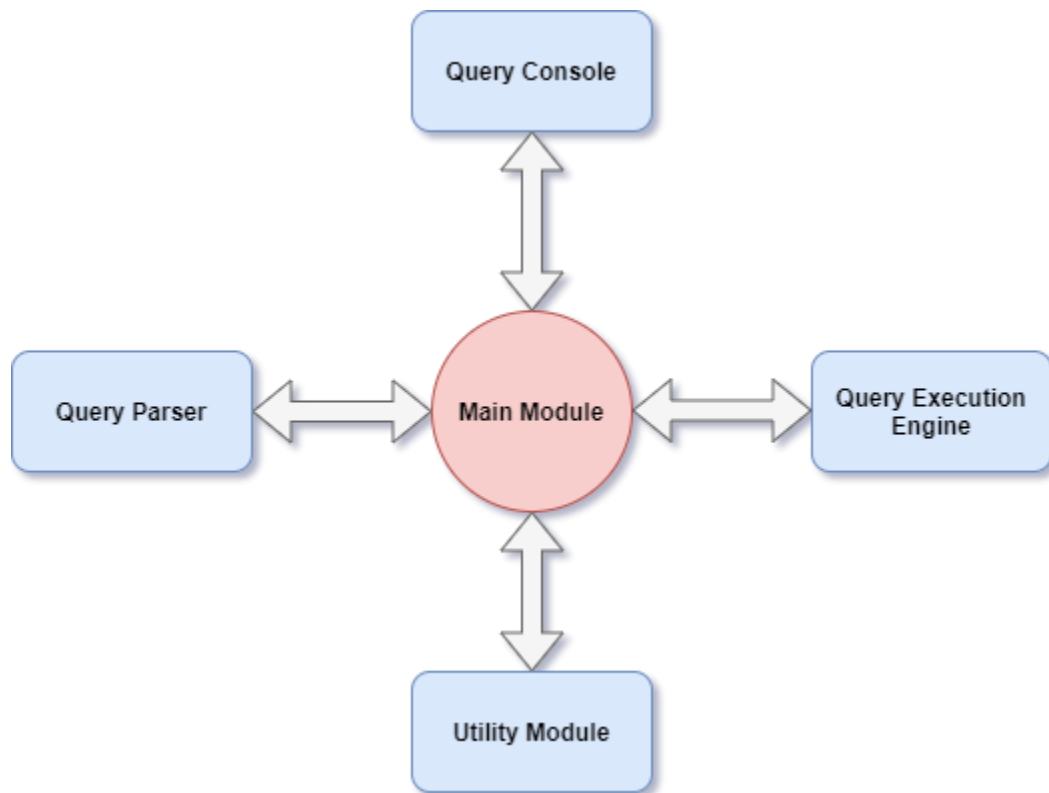
It is very important to get a clear difference between the database and what is a database management system. Yes, in many cases both the terms can be used interchangeably even though they mean two separate things. According to Wikipedia the term “Database” is defined as an organized collection of data, and on the other hand “Database Management System” is a software that enables the user to access, control, and maintain a database.

We were assigned a task to build a “Database Management System” from scratch as a group project, using the skills and knowledge that we have accouraged in this course. Before we start developing the databases, we have one rigorous research on which programming language to use and what are the efficient data structures that we can use in our application for smoother functioning.

As far as a programming language was concerned, we were given two clear choices the first one was JAVA and the other one was Python. Both the languages have their pros and cons Java has more libraries and a lot of developer community support if we get stuck anywhere on the other hand Python is more flexible in terms of boilerplate code. As we were supposed to build the database from the scratch, we were not dependant on libraries to get things done. So, after a lot of research and discussion, we decided to go with Python.

The next important thing that we need to decide is how are we going to save the data. In another way what data structures that we need to use for data manipulation and saving the data. Here we were looking at three different choices which are either we store our data in a CSV, XML, or JSON file. As we have already decided to go with Python, the decision to go with JSON file was easier and logical. As we knew the python dictionary stores data in form of JSON. It will be much easier to read from and write to files in a JSON format with Python.

Moving on with development we first decided on the naming conventions and other best practices that we are going to follow till the end of the development. We have decided to create Three main modules first being parser, the second is query execution engine, and the last but not least is a utility which will hold all the helper classes in the project. We have also decided that we are going to use GitLab as projects version control systems and agreed on pushing individual updates daily.



*Figure 1 Block Diagram of the system*

As you can see in the figure, we have divided our database management system into different modules such as Main Module, Query Parser, Query Execution Engine, and Utility module.

**Query Console:** It is nothing but a terminal where our application runs, using which we allow the user to interact with the database management system. We also show the user any warning, successes, and error messages to the user input.

**Query Parser:** It is responsible for parsing the user input as the name itself suggests. User input from query console will be forwarded to Query parser first, via the main module. Now it is the parser's responsibility to return an object that the query execution engine can read and perform the instructions on the persistent data. To translate the user input to the parsed object the parser uses regular expression. If the user input is in the invalid format it will return the error to the main module, if not the parsed object will be returned. In a nutshell, the parser checks if the syntax of the input is correct or not.

**Query Execution Engine:** The main module forwards the parsed object sent by the parser to the query execution engine. This object consists of all the information engine requires to complete the operation. Now, the execution engine is responsible for checking if the operation is logically correct or not. For example, validating the primary key constraints. After the execution of the query, the engine will return the appropriate message to the main module.

**Utility Module:** The utility module holds the helper classes such as constants and logger. These classes help the other modules to function smoothly.

**Main Module:** As it is apparent from the diagram the main module works as glue, holding all the modules together. It takes the user input from the query console forwards it to the correct parsing program, takes the parsed object from the parser, and sends it to the execution engine. Then it takes the message from the engine and prints it in a query console using utility classes.

Another thing that was very important to identify while developing these modules were handling the boundary cases. We simply can not assume that the user will always insert a correct query. All the grammar level test cases were handled in the query parser on the other hand all the logical test cases were handled in the execution engine. To get some idea on how, to begin with, to handle test cases we opened up the MySQL Workbench and tested some queries with extra spaces, missing parentheses, and more to know how a well-built engine handles it.

The difficult part was to figure out where a user is making a logical error, for example, a user should not be allowed to insert the data into a table that does not exist in the first place, or when a table has an exclusive lock on it another user can not read or write the table until the transaction completes. At the end of the transaction, the user may choose to omit or rollback, or a user can even quit the app without commit or rollback. At that time system needs to perform a rollback before stopping the execution. All the modules, their test cases, and their work are explained in detail further into the report.



## Conceptual Framework

The main objective of the initiative taken by the team is to build a robust database by implementing it using the concepts relating to database taught previously. The framework of the application consists of various objectives that play a specific role, and all are interconnected to each other. The conceptual framework was decided in the team by contributing the components by each team member and arranging the components in a sequential manner.

These components have to be connected with each other and also specifying directions to communicate on a mutual consent. Once the components of the applications and their relationships were ready, the team decided to embed it into the logic model and put it in use.

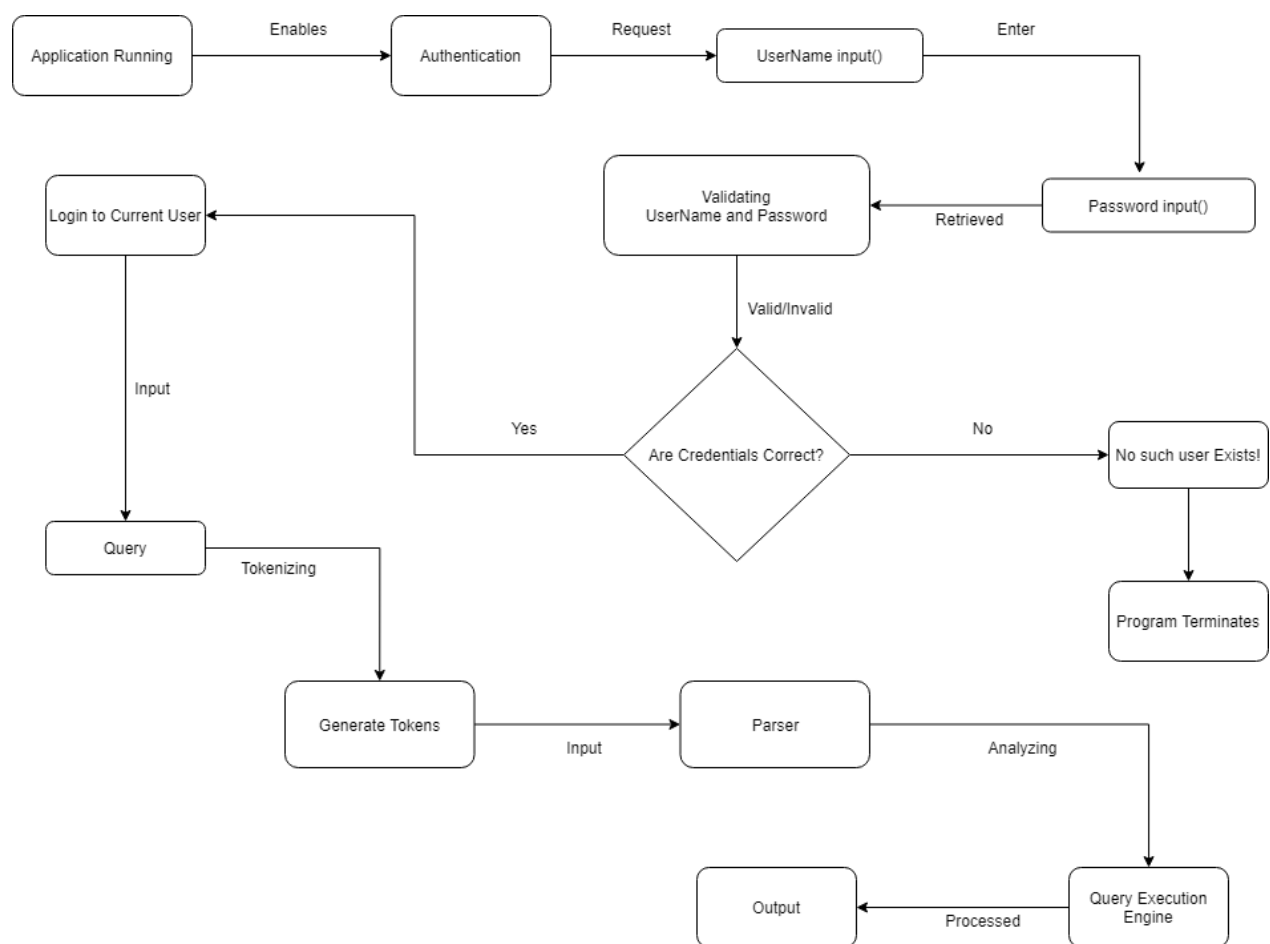


Figure 2 Conceptual Framework

As shown in Figure 2, When the application starts running it enables the first step as authentication process. Here, the user is asked to provide username and password for verification. In our application, the database created is accessible to only two authorized users simultaneously. If the validation is not successful, the application throws a message for login

failed. If the validation is passed the current user is in active state and ready to perform transactions.

Input query is given by the user which is one of the CRUD operations to perform operations on data concurrently. The input is parsed and grouped into different segments and these segments consist of table name, their records and conditions which is optional depending on the statement provided to the parser.

The parser consists of different analysis. The basic semantic analysis takes the input as tokens and convert it into a parse tree. The code optimization and code cleaning is also taken care of various phrases of parser. The symbol table is used to keep the update of all the identifier's names along with their respective database. The Engine takes care of processing of the data and provides the required processed data as output.

## Implementation of Main Module

Main module is like a spine of the application holds all the modules with together and let them work with each other in synchronization smoothly. It has main three responsibility.

### **AUTHENTICATION**

- It will continue to ask the user for credential until and unless user logs in with correct username and password.
- It runs a while loop until a global constant `CURRENT_USER` in Constants file is not None.
- At the start of application `CURRENT_USER` is set to None, and on a successful login it will be set to the username of the user.

### **SET DATABASE:**

- Set database method makes sure that a default database is set before user gives any query to execute.
- Like Authentication it also uses a while loop and `DBNAME` constant to keep on asking for the DB name, until user enters correct DB name.
- As the correct DB is given by the user, it sets the value of `DBNAME` to that DB

### **QUERY PROCESSING:**

- It is a mediator between parser and the query execution engine.
- When a user entry a query, it identifies what kind of a query is it, it can be Select query, Insert query, update query and so on. It identifies the type of it forwards it to the correct parser.
- Parser may return an error message or an object with query grammar parsed.
- If it is an error messages it prints the message in query console, else it will call the appropriate execution engine method with the parsed object in it.
- The engine may return error message or the table to print to the user.
- Another method of Main module called dictionary to table will convert dictionary returned by engine to the table and print it to the user.

## Implementation of Query Parser

The parser solely works on regular expression, to check the grammar of the user inputted SQL statement and parsing the correct data out of it. The main module identifies the correct parser method to call on based of the user input. Here is the parser that we have implemented and their working.

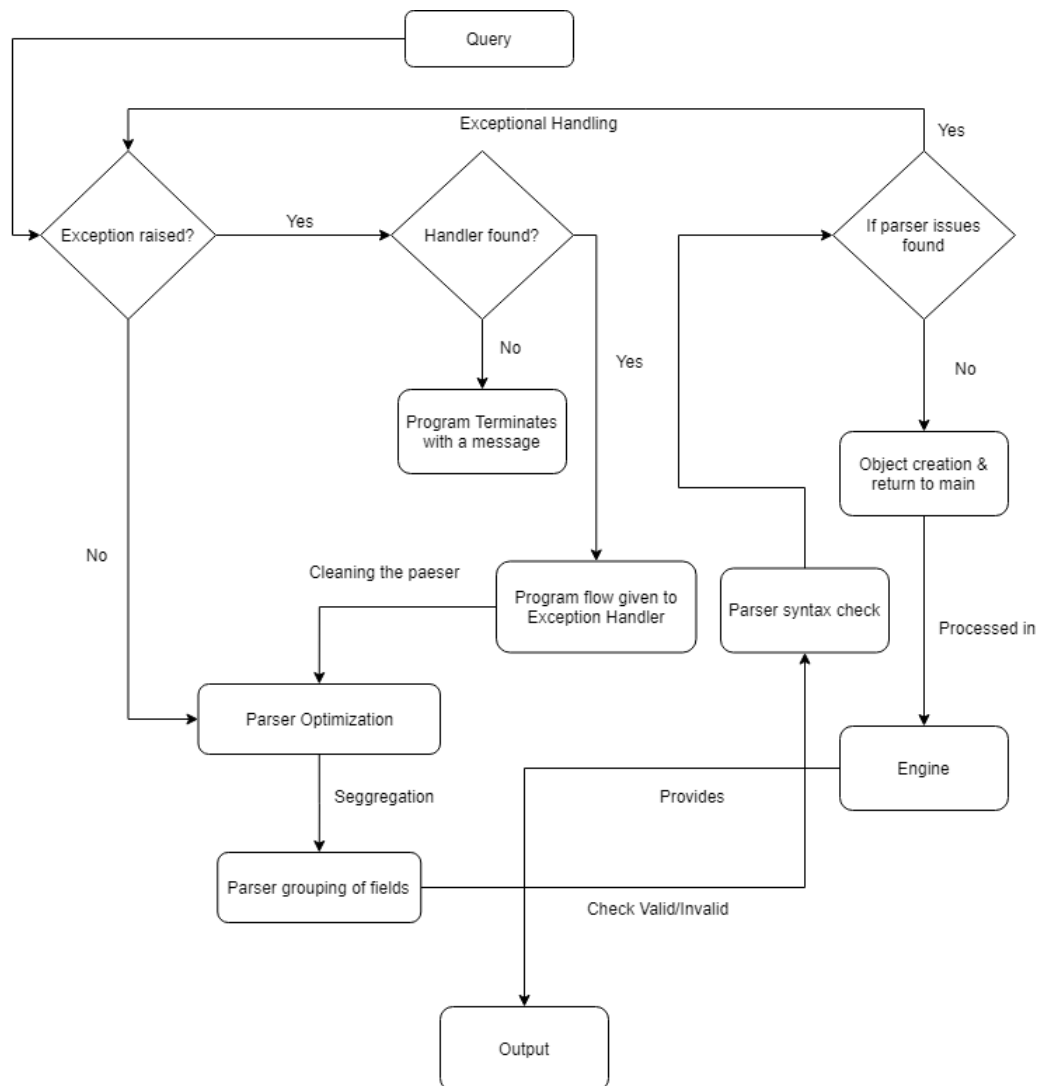


Figure 3 Query Parser Flow Diagram

### CREATE DATABASE:

- Removes extra space from start and end and replace the multiple spaces with a single space.
- Checks the length of string is 3 word, and splits it by space, and makes a list of words.
- Now it checks if the first word in the list is "CREATE", if yes moves ahead, else

generate parsing error.

- Then checks if the second word is "DATABASE", if yes moves ahead, else generate parsing error.
- Third word is the name of database, creates an object with the Database name in it and sends to main module, then main forwards it to the correct execution engine.

#### **SELECT TABEL:**

**Regular Expression:** "SELECT\s((?:\s\*)|(?:(?:\s+)(?:\s?\s+)\*))\sFROM\s(\s+)(?:\sWHERE\s(.\*)?)?"

- Removes extra space from start and end and replace the multiple spaces with a single space.
- If the user input matches the regular expression, moves ahead, else generate parsing error.
- Using the regular expression extracts table name, column name WHERE conditions and its datatypes.
- The conditions are checked for each input given by the user and decide the type of condition to be passed for the model class.
- Creates an object with the data in it and sends to main module, then main forwards it to the correct execution engine.

#### **UPDATE TABEL:**

**Regular Expression:** "UPDATE\s(\s+)\sSET\s(.\*)\sWHERE\s(.\*)?"

- Removes extra space from start and end and replace the multiple spaces with a single space.
- If the user input matches the regular expression, moves ahead, else generate parsing error.
- Using the regular expression extracts table name, column name WHERE conditions and its datatypes.
- The conditions are checked for each input given by the user and decide the type of condition to be passed for the model class.
- Creates an object with the data in it and sends to main module, then main forwards it to the correct execution engine.

## **DELETE TABEL:**

**Regular Expression:** `r''DELETE\sFROM\s(\w+)(?:\sWHERE\s(.*))?''`

- Removes extra space from start and end and replace the multiple spaces with a single space.
- If the user input matches the regular expression, moves ahead, else generate parsing error.
- Using the regular expression extracts table name, column name WHERE conditions and its datatypes.
- The conditions are checked for each input given by the user and decide the type of condition to be passed for the model class.
- Creates an object with the data in it and sends to main module, then main forwards it to the correct execution engine.

## **CREATE TABEL:**

**Regular Expression:** `r''CREATE\s+TABLE\s+(\S+)\s*\(((.*)\)''`

- Removes extra space from start and end and replace the multiple spaces with a single space.
- If the user input matches the regular expression, moves ahead, else generate parsing error.
- Using the regular expression extracts table name, column name and its datatypes.
- Creates an object with the data in it and sends to main module, then main forwards it to the correct execution engine.

## **DELETE TABLE:**

- Removes extra space from start and end and replace the multiple spaces with a single space.
- Checks the length of string is 3 word, and splits it by space, and makes a list of words.
- Now it checks if the first word in the list is "DROP", if yes moves ahead, else generate parsing error.
- Then checks if the second word is "TABLE", if yes moves ahead, else generate parsing error.
- Third word is the name of the table that needs to be deleted, creates an object with the table name in it and sends to main module, then main forwards it to the correct execution engine.

## **INSERT INTO TABLE:**

**Regular Expression:** `r "INSERT INTO (\S+) \((.*)\) VALUES \((.*)\)"`

- Removes extra space from start and end and replace the multiple spaces with a single space.
- If the user input matches the regular expression, moves ahead, else generate parsing error.
- Using the regular expression extracts table name, column name and data to be inserted into column or value of the column.
- Creates an object with the data in it and sends to main module, then main forwards it to the correct execution engine.

## **USE DATABASE:**

- Removes extra space from start and end and replace the multiple spaces with a single space.
- Checks the length of string is 2 word, and splits it by space, and makes a list of words.
- Then it checks if the first word in the list is "USE", if yes moves ahead, else generate parsing error.
- The second word is the name of the database, creates the object with name of database to use and sends it to main module.

## Implementation of Query Execution Engine

### Overview:

Query Execution Engine is a core component of this database management system. It is mainly responsible for executing the input query and perform the task based on input query. The query execution engine has methods to create database, create table, insert data into table, select data from the table, update the value in table, delete records from the table, and drop table. All the implementation of query execution engine is based on the data structure, JSON Data format, and SQL concepts. This database management system stores the data in JSON format. So, it is easy to manageable and it is easy to perform operation on the JSON object in any programming language. Along with core methods , query execution engine is responsible to check primary key constraint, handle concurrent transaction, all validation such as table or database exist or not, columns user want to insert or update exist or not, and many other functionality.

### Data Structures:

Data structures plays a key role while we are dealing with data. As a part of query execution engine, we have used data structure to store data and perform operation on that data. Here we have used Python programming language and JSON data format. The main reason to select these platforms is that it supports all the basic data structures required and also work well with the JSON data format.

The major data structure used are List and Associative Array. Basically, when we are dealing with JSON format then it stores data in the form of Associative array. Python provides that feature as a dictionary. All the data come from JSON directly stored into the dictionary. And after that we perform various operations on the dictionary. Dictionary has the time complexity of  $O(1)$  for insertion and deletion operations.

When it comes to transfer the data between parser to query execution engine or internally transfer, then we have developed custom Model Classes for every operation. That models are used for the data transfer between parser and query execution engine and between the methods of the engine. By developing the custom Model classes, it also provides facility for future enhancement.

The basic flow diagram is shown in the figure 4.



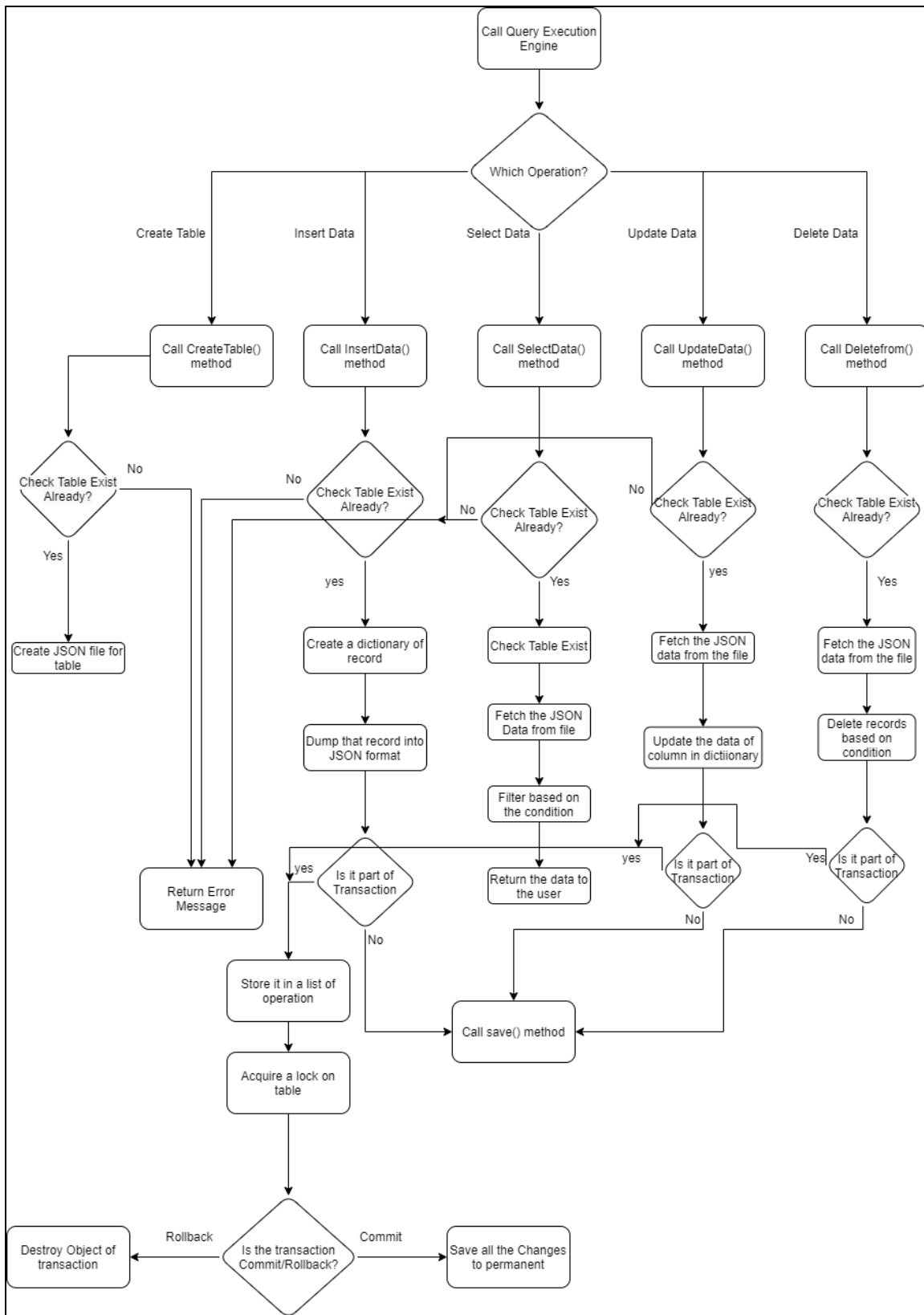


Figure 4 Workflow of query execution engine

## Pseudo Code

1. Take an input from the parser.
2. Call the method based on the input query.

Query execution engine run different methods based on the operation. Every operation is explained further.

### **CREATE DATABASE OPERATION:**

- Object of CreateDatabase Model class will be created by the parser and passed as a parameter to query execution engine.
- Engine will check first that the database already exists or not. If it already exists, then it will return error message.
- If database does not exist, then it will create directory of database in file system and return success message.

### **CREATE TABLE OPERATION:**

- Object of CreateTable Model class will be created by the parser and passed as a parameter to query execution engine.
- Check that table already exists in the database or not. If it already exists, then it will return error message.
- If table does not exist then, it will create two files in the database directory. One file will contain the metadata for the table and other file will store the records of the table. The format for the file will be JSON.
- Engine will return success message once table is created.

### **INSERT INTO TABLE OPERATION:**

- Object of InsertTable Model class will be created by the parser and passed as a parameter to query execution engine.
- Check that table is exist or not, if it does not exist then return error message.
- Check that the data to be insert is complete and does it have data for all the columns present in the table.
- If user try to insert column which does not exist in the table, then return error message.
- Check for the primary key constraint, check that the data user tries to enter already exist or not. If the value of primary key is already present in the table, then return error message.
- Create the dictionary of data to be insert and store it in JSON object.
- Store that JSON object to JSON file of the table to persist data.
- After storing JSON object to file, return success message to user.

### **SELECT OPERATION:**

- Object of SelectFrom Model class will be created by the parser and passed as a parameter to query execution engine.
- Check that table is exist or not, if it does not exist then return error message.
- Fetch the JSON file of required table and store it into the JSON format.
- Convert JSON object to dictionary.
- Check the condition based on the WHERE clause and filter the data based on the condition.
- If user need some columns, then remove the other from the dictionary.
- Return the data after filteration.

### **UPDATE OPERATION:**

- Object of UpdateTable Model class will be created by parser and passed as a parameter to query execution engine.
- Check that table is exist or not, if it does not exist then return error message.
- Fetch the JSON file of required table and store it into the JSON format.
- Convert JSON object to dictionary.
- Check the condition based on the WHERE clause and filter the data based on the condition.
- Return error if the column specified to update does not exist in the database.
- If column exist, then update the value of that.
- Store it in a JSON file and return success message.

### **DELETE FROM TABLE OPERATION:**

- Object of DeleteTable Model class will be created by parser and passed as a parameter to query execution engine.
- Check that table is exist or not, if it does not exist then return error message.
- Fetch the JSON file of required table and store it into the JSON format.
- Convert JSON object to dictionary.
- Check the condition based on the WHERE clause and filter the data based on the condition.
- Check the condition based on the WHERE clause and delete the data based on the condition.
- After deleting data in the JSON object save that json object to table file.
- Return success message that data is deleted.

**DROP TABLE:**

- Object of DropTable Model class will be created by parser and passed as a parameter to query execution engine.
- Check that table is exist or not, if it does not exist then return error message.
- If table exist, then delete the JSON file of table.
- Return the success message that table is deleted.

This is how the operations in the query execution engine implemented.

## Implementation of Transaction

Transaction is implemented in a way where two users can execute two parallel transaction. All the test cases are covered, and the locking mechanism is developed to avoid the conflicts. ACID properties are being followed during the transaction. User will have a two option; one is committing and second is rollback. If user will commit the transaction, then all the changes made during the transaction made permanently. If user enter rollback command, then the changes will not made permanent.

To implement transaction, we developed one class named "Transaction". When user initiate transaction, the object of that class will be initialized. This object contains the details about user, database, list of operation performed and list of tables with lock. When any user try to execute query at that time user will check the lock on the table. There are two types of lock we have used, one is "Shared" – it is for select operation. Second is "Exclusive" – it is for insert, update, delete operation.

The detailed implementation of operation is explained in the flow diagram of query execution engine and explained in demo video.

## Implementation of Data Dictionary

### **DATA DICTIONARY:**

- Iterating through the all the directories of the project and finding the database package by passing the DBName which is database name as a parameter.
- Once the database package is found, iterate the package and filter out the .json extension files.
- Again iterating through the .json extension files to finally Set the metadata files and then opening the files containing the metadata
- The Key, Value pair for the metadata file is provided with column name and table name and appended it to the file.
- Then finally creating a new file named "DataDictionary.json" and aggregating all the metadata of the application and storing it one place accessible by all users.
- Refactoring the final metadata in the form of a table output for readability and user convenience.

## Project Features

### **CRUD OPERATIONS:**

- The database developed provides convenience to perform actions on data like create, read, update and delete.
- These operations are the core for interacting with any database.
- Create keyword is used to create the database and the table, read is used to access the contents of the table.
- Update functionality usually deals with modifying the records by using certain parameters as a condition.
- In our application the delete keyword is used for deleting the existing records in the table. It also considers the condition to remove the records specified by the user.

### **SECURITY:**

- Enhanced security by giving specific access to only authorized user and restricting data visibility of sensitive content from other users.
- In our project only two users are allowed to login to the application concurrently.

### **CONCURRENT TRANSACTION:**

- The transactions which take place concurrently is maintaining data integrity as we have made use of locks in the application.
- The transactions executed does not affect the final result as we are handling it by providing appropriate locks.
- Warnings are passed to the user to notify about the uncommitted transaction and other conflict scenarios.
- The order of transaction if not followed in an appropriate manner will lead to failure of the transaction which is successfully handled by the application.

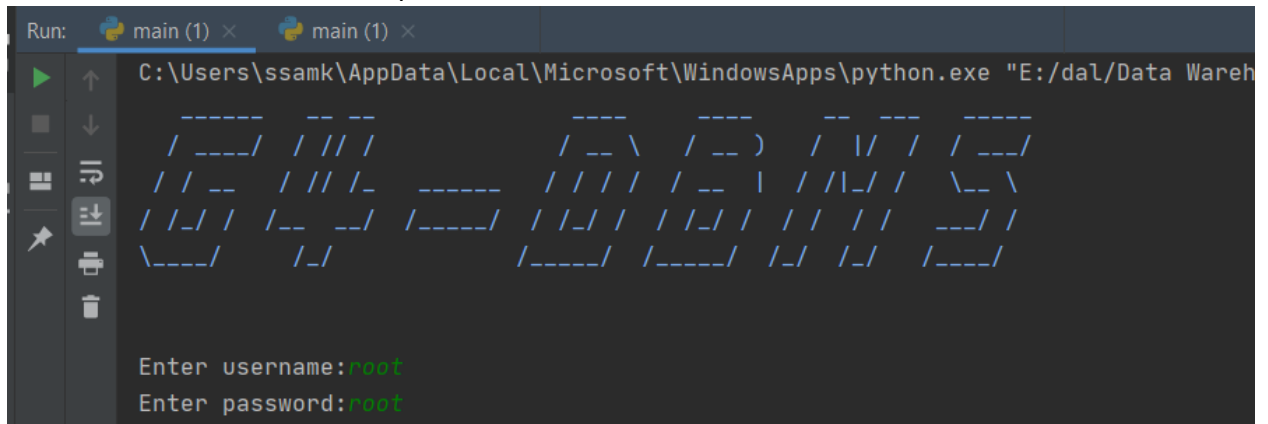
### **DATA DICTIONARY:**

- Data dictionary is an aggregation of tables, fields and the attributes which are present in a database.
- Along with the creation of table, metadata of the table is also generated by our application.
- The holistic view of the complete database has been provided in the application by collecting the metadata provided and portrayed in form of a table for user readability.

## Test Cases

### 1. AUTHENTICATE USER:

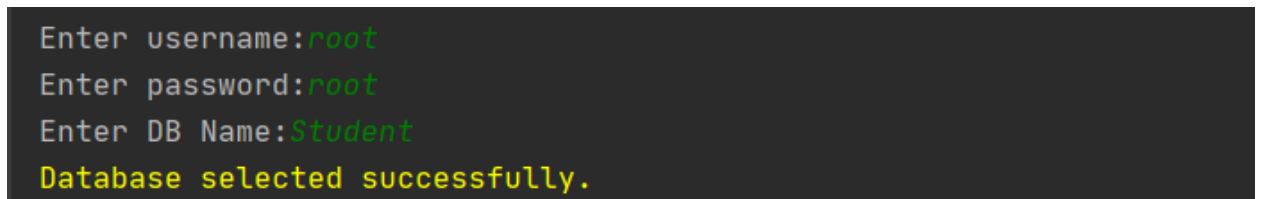
User will enter username and password.



```
Run: main (1) × main (1) ×
C:\Users\ssamk\AppData\Local\Microsoft\WindowsApps\python.exe "E:/dal/Data Wareh
Enter username:root
Enter password:root
```

### 2. SELECT DATABASE:

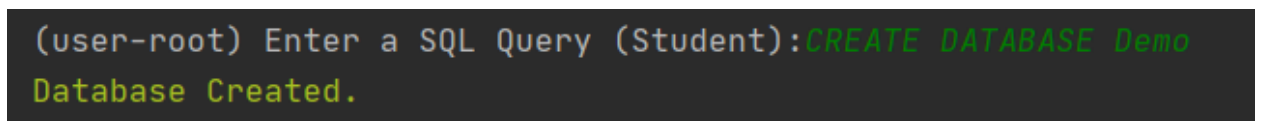
User need to enter the name of the database.



```
Enter username:root
Enter password:root
Enter DB Name:Student
Database selected successfully.
```

### 3. CREATE NEW DATABASE:

User has created database named "Demo"



```
(user-root) Enter a SQL Query (Student):CREATE DATABASE Demo
Database Created.
```

### 4. CREATE NEW TABLE:

User has created new table. The query in image is not clear.

**Input query is** - CREATE TABLE AGENTS (AGENT\_CODE CHAR(6), AGENT\_NAME CHAR(40), WORKING\_AREA CHAR(35), COMMISSION CHAR(10), PHONE\_NO CHAR(15), COUNTRY VARCHAR2(25), PRIMARY KEY (AGENT\_CODE));

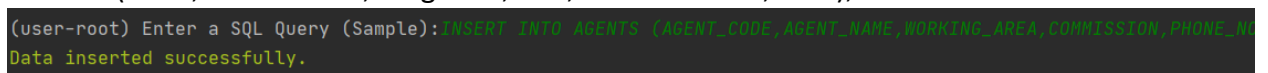


```
(user-root) Enter a SQL Query (Sample):CREATE TABLE AGENTS (AGENT_CODE CHAR(6), AGENT_NAME CHAR(40), WORKING_AREA CHAR(35), COMMISSION CHAR(10), PHONE_NO CHAR(15), COUNTRY VARCHAR2(25), PRIMARY KEY (AGENT_CODE));
Table created successfully.
```

### 5. INSERT DATA INTO TABLE:

**Input query is** - INSERT INTO AGENTS

(AGENT\_CODE,AGENT\_NAME,WORKING\_AREA,COMMISSION,PHONE\_NO,COUNTRY)  
VALUES (A007,Ramasundar,Bangalore,0.15,077-25814763,India);



```
(user-root) Enter a SQL Query (Sample):INSERT INTO AGENTS (AGENT_CODE,AGENT_NAME,WORKING_AREA,COMMISSION,PHONE_NO,COUNTRY) VALUES (A007,Ramasundar,Bangalore,0.15,077-25814763,India);
Data inserted successfully.
```



## 6. INSERT DATA INTO TABLE WITH WRONG COLUMN NAME

User has entered query name which does not exist in the table.

```
(user-root) Enter a SQL Query (Sample):INSERT INTO AGENTS (AGENT_CODE,AGENT_NAME,WORKING_AREA,COMMISSION,PHONE_NO,COUNTRY) VALUES (A007,'Ramasundar','Bangalore',0.15,077-25814763,India)
Please enter valid columns.
```

## 7. INSERT DUPLICATE DATA FOR PRIMARY KEY

User has inserted the entry which already has entry in table.

```
(user-root) Enter a SQL Query (Sample):INSERT INTO AGENTS (AGENT_CODE,AGENT_NAME,WORKING_AREA,COMMISSION,PHONE_NO,COUNTRY) VALUES (A007,'Ramasundar','Bangalore',0.15,077-25814763,India)
Primary key Constraint: Data with primary key already exist in table.
```

## 8. SELECT DATA FROM TABLE

```
(user-root) Enter a SQL Query (Sample):SELECT * FROM AGENTS
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A007	Ramasundar	Bangalore	0.15	077-25814763	India
A009	Benjamin	Hampshair	0.11	008-22536178	Australis

## 9. SELECT DATA FROM TABLE WITH WHERE CLAUSE AND SPECIFIC COLUMN

```
(user-root) Enter a SQL Query (Sample):SELECT AGENT_CODE,AGENT_NAME FROM AGENTS WHERE AGENT_CODE = A007
```

AGENT_CODE	AGENT_NAME
A007	Ramasundar

## 10. UPDATE VALUE OF THE TABLE

Before Update:

```
(user-root) Enter a SQL Query (Sample):SELECT * FROM CUSTOMER
```

CUST_CODE	CUST_NAME	CUST_CITY	WORKING_AREA	CUST_COUNTRY	GRADE	OPENING_AMT	RECEIVE_AMT	PAYMENT_AMT	OUTSTANDING_AMT	PHONE_NO	AGENT_CODE
C00001	Micheal	New York	New York	USA	2	3000	5000	2000	6000	CCCCCCC	A008
C00013	Neelkanth	London	London	UK	2	6000	5000	7000	4000	BBBBBBB	A003

After Update:

```
(user-root) Enter a SQL Query (Sample):UPDATE CUSTOMER SET CUST_NAME = Samkit WHERE CUST_CODE = C00013
Data updated successfully.
(user-root) Enter a SQL Query (Sample):SELECT * FROM CUSTOMER
```

CUST_CODE	CUST_NAME	CUST_CITY	WORKING_AREA	CUST_COUNTRY	GRADE	OPENING_AMT	RECEIVE_AMT	PAYMENT_AMT	OUTSTANDING_AMT	PHONE_NO	AGENT_CODE
C00001	Micheal	New York	New York	USA	2	3000	5000	2000	6000	CCCCCCC	A008
C00013	Samkit	London	London	UK	2	6000	5000	7000	4000	BBBBBBB	A003

## 11. DELETE DATA FROM THE TABLE

```
(user-root) Enter a SQL Query (Sample):DELETE FROM CUSTOMER WHERE CUST_CODE = C00001
Data Deleted successfully.
(user-root) Enter a SQL Query (Sample):SELECT * FROM CUSTOMER
```

CUST_CODE	CUST_NAME	CUST_CITY	WORKING_AREA	CUST_COUNTRY	GRADE	OPENING_AMT	RECEIVE_AMT	PAYMENT_AMT	OUTSTANDING_AMT	PHONE_NO	AGENT_CODE
C00013	Samkit	London	London	UK	2	6000	5000	7000	4000	BBBBBBB	A003

## 12. DELETE TABLE

```
(user-root) Enter a SQL Query (Sample):DROP TABLE CUSTOMER
Table deleted successfully.
(user-root) Enter a SQL Query (Sample):SELECT * FROM CUSTOMER
Select Failed. Table does not exist.
```

### 13. CREATE A PARALLEL INSTANCE OF APPLICATION AND LOGIN ADMIN USER

```
Run: main (1) × main (1) ×
C:\Users\ssamk\AppData\Local\Microsoft\WindowsApps\python.exe "E:/dal/Dat

----- -- --
 / ____/ / // /      / _ \ / _ \ / // / / ____/
 / / __ / // / /      / // / / _ \ / // / / ____/
 / / / / / _ \ /      / / / / / _ \ / // / / ____/
 \ ____/ / /      / ____/ / ____/ / / / / ____/

Enter username:admin
Enter password:admin
Enter DB Name:Sample
Database selected successfully.
```

### 14. BEGIN TRANSACTION FOR ROOT USER

```
(user-root) Enter a SQL Query (Sample):BEGIN TRANSACTION
Transactions Initiated.
```

### 15. SELECT OPERATION DURING TRANSACTION – ROOT USER

```
(user-root) Enter a SQL Query (Sample):SELECT * FROM AGENTS
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A007	Ramasundar	Bangalore	0.15	077-25814763	India
A009	Benjamin	Hampshair	0.11	008-22536178	Australis

### 16. SELECT OPERATION – ADMIN USER

```
Database selected successfully.
(user-admin) Enter a SQL Query (Sample):SELECT * FROM AGENTS
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A007	Ramasundar	Bangalore	0.15	077-25814763	India
A009	Benjamin	Hampshair	0.11	008-22536178	Australis

### 17. INSERT OPERATION DURING TRANSACTION – ROOT USER

```
(user-root) Enter a SQL Query (Sample):INSERT INTO AGENTS (AGENT_CODE,AGENT_NAME,WORKING_AREA,COMMISSION,
Data inserted successfully.
```

### 18. SELECT OPERATION – ADMIN USER

```
(user-admin) Enter a SQL Query (Sample):INSERT INTO AGENTS (AGENT_CODE,AGENT_NAME,WORKING_AREA,COMMISS
Sorry you can not perform any operation because another user has already acquired lock on this table.
```

## 19. ROLLBACK TRANSACTION

```
(user-root) Enter a SQL Query (Sample): ROLLBACK TRANSACTION
Transactions Rolledback Successfully.
(user-root) Enter a SQL Query (Sample): SELECT * FROM AGENTS
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A007	Ramasundar	Bangalore	0.15	077-25814763	India
A009	Benjamin	Hampshair	0.11	008-22536178	Australis

## 20. TRANSACTION EXAMPLE WITH COMMIT

```
(user-admin) Enter a SQL Query (Sample):BEGIN TRANSACTION
Transactions Initiated.
(user-admin) Enter a SQL Query (Sample):INSERT INTO AGENTS (AGENT_CODE,AGENT_NAME,WORKING_AREA,COMM
Data inserted successfully.
(user-admin) Enter a SQL Query (Sample):COMMIT TRANSACTION
Transactions Committed Successfully.
(user-admin) Enter a SQL Query (Sample):SELECT * FROM AGENTS
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A007	Ramasundar	Bangalore	0.15	077-25814763	India
A009	Benjamin	Hampshair	0.11	008-22536178	Australis
A001	PRASHANT	SARVI	0.11	008-22536178	Australis

## 21. GENERATE DATA DICTIONARY

```
(user-root) Enter a SQL Query (Sample): show tables
Data Dictionary generated.


| TableName | primary key | Columns (Datatype)                                                                                                                                    |
|-----------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| AGENTS    | AGENT_CODE  | ['AGENT_CODE(CHAR(6))', 'AGENT_NAME(CHAR(40))', 'WORKING_AREA(CHAR(35))', 'COMMISSION(CHAR(10))', 'PHONE_NO(CHAR(15))', 'COUNTRY(VARCHAR2(25))']      |
| ORDERS    | ORD_NUM     | ['ORD_NUM(VARCHAR2(6))', 'ORD_AMOUNT(VARCHAR2(6))', 'ADVANCE_AMOUNT(VARCHAR2(6))', 'ORD_DATE(DATE)', 'CUST_CODE(VARCHAR2(6))', 'AGENT_CODE(CHAR(6))', |


```

## Conclusion

The database management system being more robust, efficient, and convenient for the user as compared to file system, it is an essential to store the data and perform operations on the database for data manipulation. The database helps in maintaining data consistency unlike file system and maintains data integrity. The database management system is faster to implement and saves time with minimal effort.

The implementation of relational database management systems requires in-depth understanding of all the components used. Although not implemented fully but provides a prototype. The system is efficient and flexible for further add-ons of the module to be integrated. There's a still lot of room for the improvement of the database by making few future enhancements.

## Future Enhancement

Integration of a new module which is Entity-Relationship diagram (ERD) to the existing application to provide a structural view of the database. The application can have several advantages by the introduction of the ERD like data debugging which can reduce the number of challenges. The visualised presentation of the of the layout provides support to the work when developing database design.

Another improving in the application is adding "NOT EQUAL" conditions in the update, delete and select statement when a WHERE condition is specified.

## References

- [1] "Building a New Database Management System in Academia // Blog // Andy Pavlo - Carnegie Mellon University", *Cs.cmu.edu*, 2020. [Online]. Available: <https://www.cs.cmu.edu/~pavlo/blog/2017/03/building-a-new-database-management-system-in-academia.html>. [Accessed: 10- Nov- 2020]
- [2] "Parsing SQL - Strumenta", *Strumenta*, 2020. [Online]. Available: <https://tomassetti.me/parsing-sql/>. [Accessed: 22- Nov- 2020]
- [3] A. Das "Execution Plans in SQL Server", *SQL Shack - articles about database auditing, server performance, data recovery, and more*, 2020. [Online]. Available: <https://www.sqlshack.com/execution-plans-in-sql-server/>. [Accessed: 29- Nov- 2020]
- [4] S. Chambers, "How to extract the table name from a CREATE/UPDATE/INSERT statement in an SQL query?", *Stack Overflow*, 2020. [Online]. Available: <https://stackoverflow.com/questions/62311026/how-to-extract-the-table-name-from-a-create-update-insert-statement-in-an-sql-qu>. [Accessed: 01- Dec- 2020]
- [5] Alberto Lerner, Ioana Manolescu, "The nuts and bolts of DBMS construction: building your own prototype", [Online]. Available: <https://pages.saclay.inria.fr/ioana.manolescu/SLIDES/LernerManolescu-SBBD2003.pdf> [Accessed 20 October 2020].
- [6] Tanmay Chakrabarty, "Draw the database system architecture. DBMS", [Online]. Available: <https://onlineclassnotes.com/draw-database-system-architecture-dbms#.XMhh7IOZsJE.pinterest> [Accessed 24 October 2020].