-

Faculty of Computer Science

Dalhousie University

CSCI 5408 – Data Management, Warehousing, Analytics

PROJECT REPORT

Building a simple distributed database and its management system

Team Members

Ananthi Thiagarajan Subashini
B00789128
ananthi.ts@dal.ca

Johanan Abhishek Prabhurai
B00869532
jh416787@dal.ca

Sri Sai Bhargav Nuthakki
B00867627
sr995902@dal.ca

Table of Contents

Problem statement

To build a simple distributed database and its management system with the help of Data Structures in Java and no other libraries. It should be able to handle at least 2 users.

Project Overview

A simple distributed database management system designed with the following functionalities:

- Handle multiple users (at least 2).
- Has encrypted (with the help of AES) passwords for users to login.
- Each user can have their own transactions.
- Parse standard SQL queries.
- Follow ACID properties.
- Table level locks (exclusive locks).
- General logs like execution status, processing, number of rows inserted/updated will be available.
- Transactions and their details available through event logs.
- SQL dumps created based on databases and tables present.
- Create a simple ERD based on the databases and tables present.

The Distributed Database functionality was achieved through a Client-Server architecture, where two or more Clients connect to a Server and communicate with each other through the server.

Work distribution and Team coordination

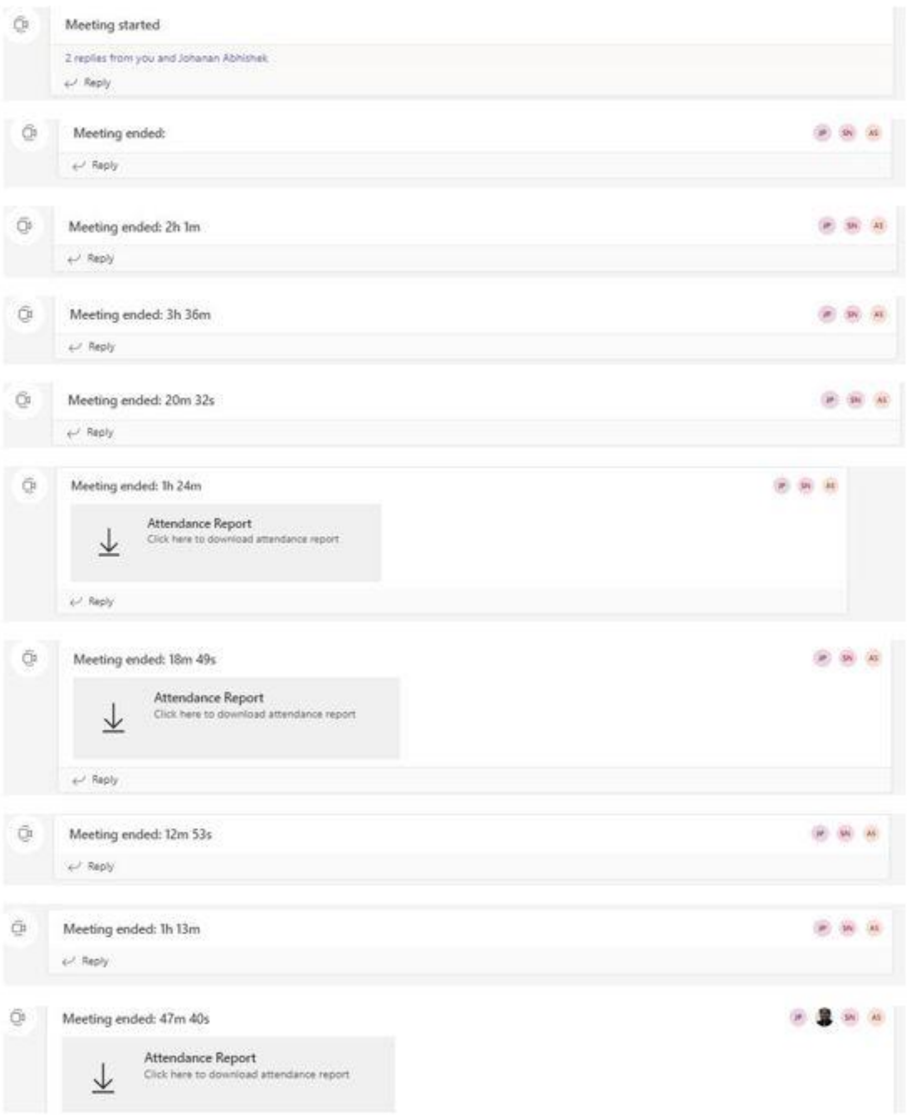
Team Coordination - The overall basic data structure was created initially and then we all took individual queries and worked on that and later we integrated the queries and tested it on the distributed environment. Apart from query processing, the remaining works like GDD, distributed setup, login, ERD, dump and logs we split these tasks, worked on them and again integrated and performed end to end testing.

Individual Contribution –

1. **Ananthi Thiagarajan Subashini (B00789128)**:
   - Login and Encryption
   - Table and column data structures
   - SELECT Query
   - CREATE Query
   - ERD
   - DUMP
2. **Johanan Abhishek Prabhurai (B00869532):**
   - INSERT Query
   - DROP Query
   - Client
   - Listener
   - Server
3. **Sri Sai Bhargav Nuthakki (B00867627):**
   - UPDATE Query

- Event Logs

Group Contribution –



Meeting started

2 replies from you and Johanan Abhishek

Reply

Meeting ended:

Reply

Meeting ended: 2h 1m

Reply

Meeting ended: 3h 36m

Reply

Meeting ended: 20m 32s

Reply

Meeting ended: 1h 24m

Attendance Report
Click here to download attendance report

Reply

Meeting ended: 18m 49s

Attendance Report
Click here to download attendance report

Reply

Meeting ended: 12m 53s

Reply

Meeting ended: 1h 13m

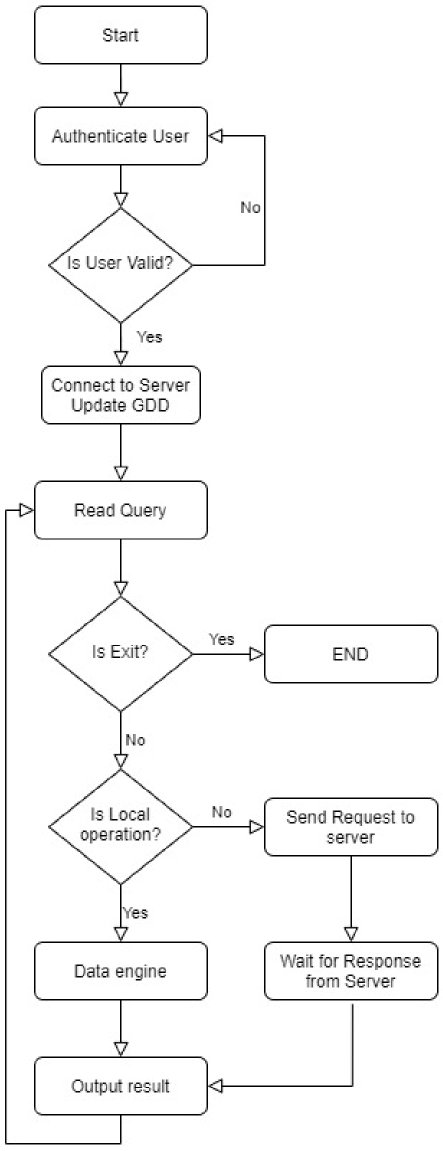
Reply

Meeting ended: 47m 40s

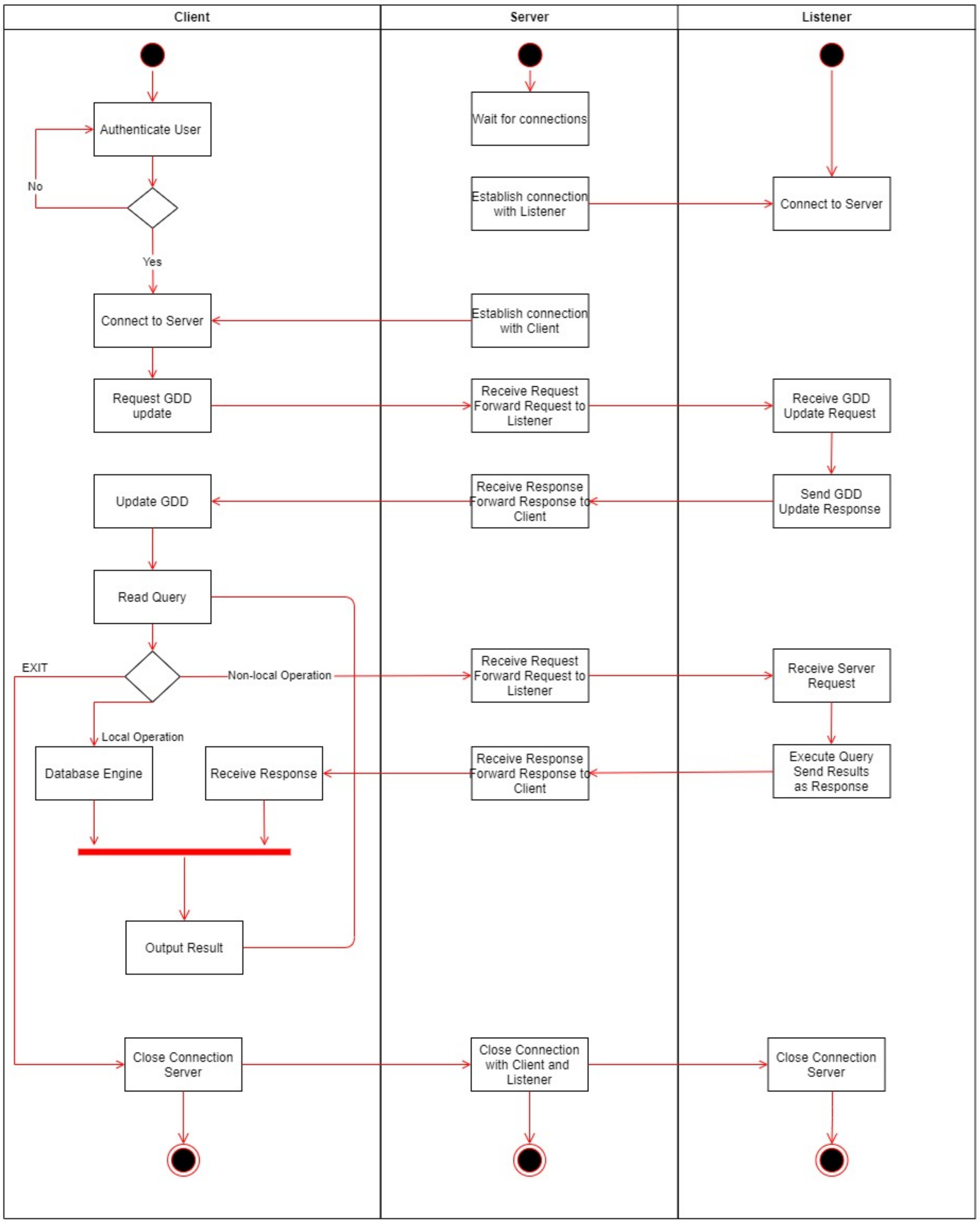
Attendance Report
Click here to download attendance report

Implementation Plan

Flow Chart



Activity Diagram



Client
Server
Listener
Authenticate User
No
Yes
Wait for connections
Establish connection with Listener
Connect to Server
Connect to Server
Establish connection with Client
Request GDD update
Receive Request Forward Request to Listener
Receive GDD Update Request
Update GDD
Receive Response Forward Response to Client
Send GDD Update Response
Read Query
EXIT
Non-local Operation
Receive Request Forward Request to Listener
Receive Server Request
Local Operation
Database Engine
Receive Response
Receive Response Forward Response to Client
Execute Query Send Results as Response
Output Result
Close Connection Server
Close Connection with Client and Listener
Close Connection Server

Project Design

Distributed Database Architecture

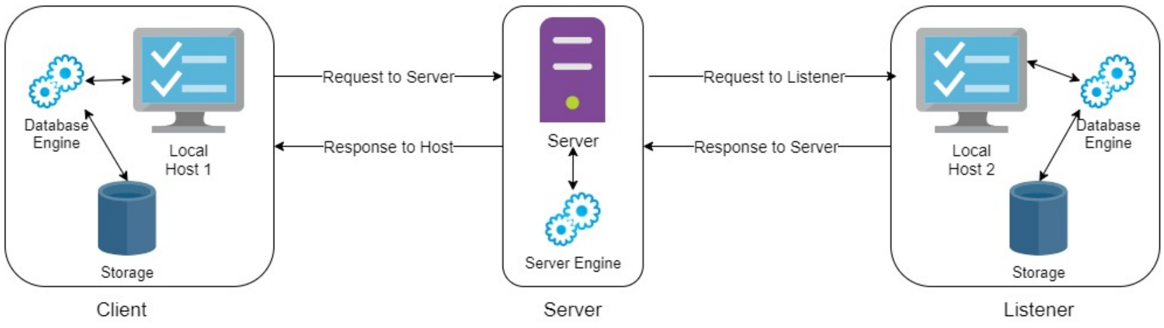


Figure 2.a: Distributed Database Architecture

Figure 2.a depicts the architecture of the Distributed Database. There are two Local hosts and one Server. Each local host acts both as a “Client” and a “Listener”. These hosts communicate with each other through the Server. Both the hosts have the database engine running on their respective machines. Query parser, Query Executor, Storage manager, Access methods, etc… together is the database engine. Along with the database engine, each host has its respective storage in which data is stored under Databases and Tables in the form of files.

Client: The Client is a program through which the user gets to interact with the database. All the queries a user executes happens through the Client program. Following are the tasks that the Client performs:

- Lets the user login to the database if the username and password are valid.
- Establishes a connection with the Server.
- Synchronizes the GDD with the help of the server.
- Takes in queries from the user that are to be executed.
- Decides if the database and table are present at local site or a different site (host) based on the updated GDD.
- If the table is local, it parses, validates, executes the query, and displays the required information to the user.
- If the table is on a different site, it sends a request to the server and waits for a response from the server.
- Once response is received from the server, it displays the required information to the user.

Listener: The Listener is a background program which continuously listens for request coming from the server. It is used when operations related to data that is local (host on which listener is running) is requested by a Client to be performed. Following are the tasks that the Listener performs:

- Receives requests from the server. A request is usually a query that is to be executed on the host which has the table.
- After receiving a request, it parses, validates, executes the query, and gives a response to the server with the relevant information.
- The response starts with a “BEGIN” message followed by the output of the query that ran and the response completes with a “END” message.

Note: Both the Client and Listener use the same database engine (Query parser, Query Executor, Storage manager, Access methods, etc…)

Server: The server is a program that acts as a mediator or communicator between two local hosts. The Server does the following tasks:

- Continuously listens for requests from the Client (Local Host 1)
- Forwards these requests to the Listener (Local Host 2) where the respective operations are to be performed.
- After sending the request to the respective Listener, the server continuously waits for a response from the Listener.
- The server then forwards this response from the Listener to the respective Client that initially requested.
- Once the response is given back to the Client, it gets back to continuously listening for requests from the Client.

We get to see a one-way access from Local Host 1 as a client to Local Host 2 as a Listener in figure 2.a, if we just swap Local Host 1 with 2, i.e., Local Host 2 becomes the Client and Local Host 1 becomes the Listener and establish this connection along with the first connection it becomes two-way access. Using this approach, we form a Distributed Database and its management system. Based on the location at which the table is present, the query is going to be executed on that site. For example, if Local Host 1 is doing a select operation from a table present in Local Host 2, the select query fired by Local Host 1 is executed at Local Host 2 and the result is sent back to Local Host 1. This way, the queries executed are always at the respective sites.

The advantage with this approach to design is,

- Each Local host can act as a standalone database with a management system.
- When you connect to the server, it becomes a distributed database.

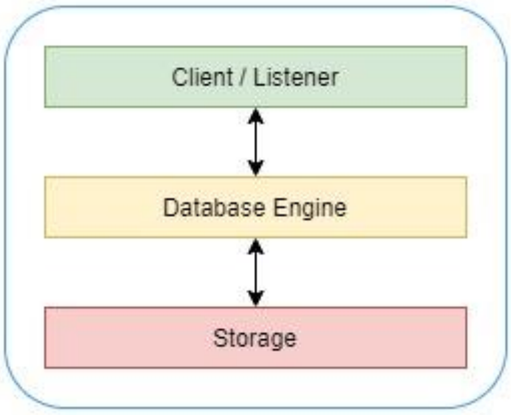


Figure 2.b: Host Layered Architecture of Database

Database Engine: The Database Engine is the backbone of the database system and acts as the Database Management System. Fig 2.b shows the layered architecture of the database system. Each Host (Client/Listener) has the database engine running in the background. The Host internally uses the database engine to perform all the operations related to a database. The tasks performed by the database engine are:

- Parses the query/statement given by the user.
- Validates the query, checks for things such as a valid database/table name, depending on the type of query, it performs the next set of validations.
- Executes the query if it is valid. Based on the type of operation, provides the access to the storage layer.
- Uses Locking mechanisms for concurrency controls.
- Manages the GDD, Metadata related to tables and the files that store the data itself.

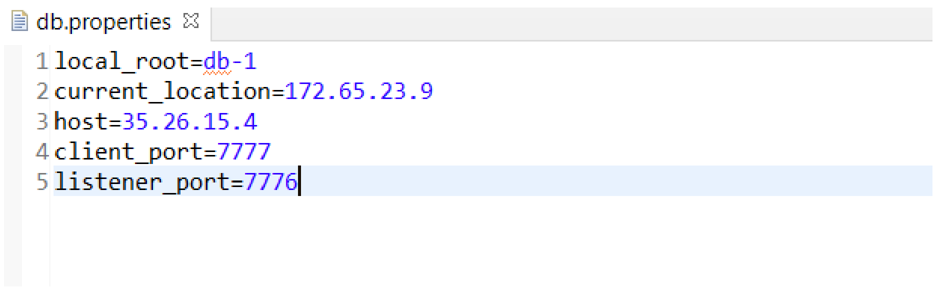
Storage: The traditional file system is used for storage. All the data is stored under directories and text files based on the requirements.

Implementation Details

Data Storage

Application properties

The application has a property file named **db.properties** where the local directory path is mentioned. The GDD, schema and table information are stored in this directory.


db.properties

```
local_root=db-1
current_location=172.65.23.9
host=35.26.15.4
client_port=7777
listener_port=7776
```

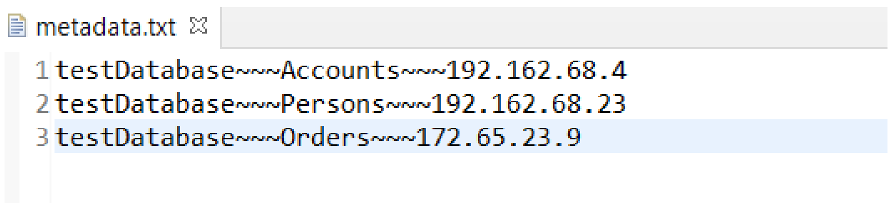
Figure 3: Application properties of the distributed system

The figure 3 shows the property file referred by the distributed database system. The local directory path where the database files are stored is mentioned as local_root and the Ip address of the system is mentioned as current_location. Host denotes the public Ip address of the server. Client port and listener port specifies the port to communicate with server.

Global dictionary Structure

GDD folder has a file called metadata.txt, which is used to store and retrieve the table information. Metadata.txt file stores three values,

- Table name
- Database name
- Location


metadata.txt

```
testDatabase~~~Accounts~~~192.162.68.4
testDatabase~~~Persons~~~192.162.68.23
testDatabase~~~Orders~~~172.65.23.9
```

Figure 4: Global data dictionary

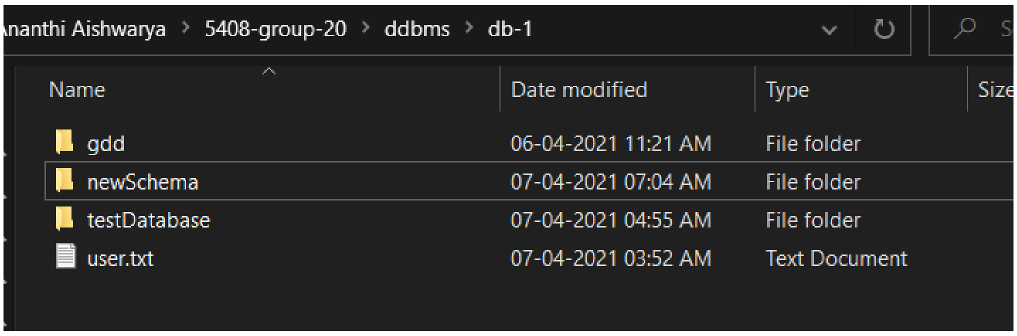
The figure 4 represents how the table information is stored in the data dictionary. Each line in the metadata.txt represents a single table information and the information is stored in the following format,

<database_name>~~~<table_name>~~~<location>

where ~~~ is the delimiter used in the application.

Database

The databases are stored inside the local root directory mentioned in the application property file. Each folder inside the local root directory represents a database and each database will have multiple sub folders inside them which represents the tables present in that database.


nanthi Aishwarya > 5408-group-20 > ddbms > db-1

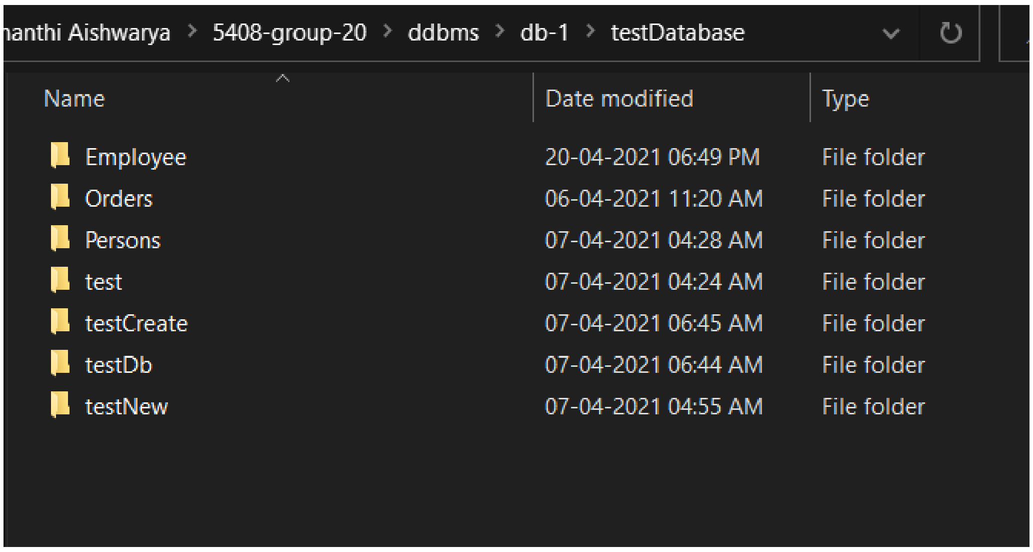
Name	Date modified	Type	Size
gdd	06-04-2021 11:21 AM	File folder	
newSchema	07-04-2021 07:04 AM	File folder	
testDatabase	07-04-2021 04:55 AM	File folder	
user.txt	07-04-2021 03:52 AM	Text Document	

Figure 5: Databases present in the local root directory

The figure 5 shows the databases present in the local machine and a separate folder is created for each database.

Table

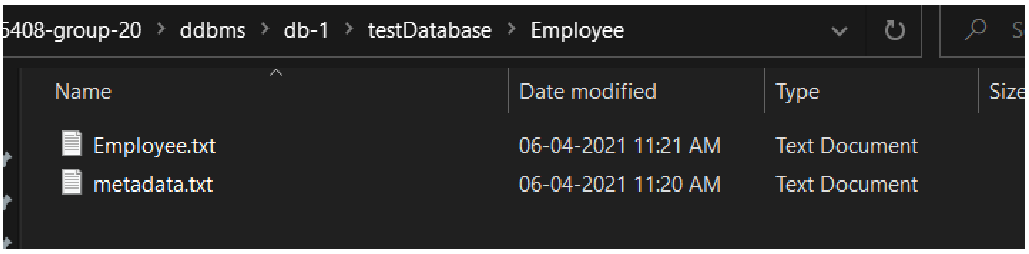
Tables are stored inside the corresponding database directory and a separate folder is created for each table. Each table will have a metadata.txt and table_name.txt where the metadata file contains all the table information like column name, column size, column type, primary key and foreign key values.


anthi Aishwarya > 5408-group-20 > ddbms > db-1 > testDatabase

Name	Date modified	Type
Employee	20-04-2021 06:49 PM	File folder
Orders	06-04-2021 11:20 AM	File folder
Persons	07-04-2021 04:28 AM	File folder
test	07-04-2021 04:24 AM	File folder
testCreate	07-04-2021 06:45 AM	File folder
testDb	07-04-2021 06:44 AM	File folder
testNew	07-04-2021 04:55 AM	File folder

Figure 6: Table directory

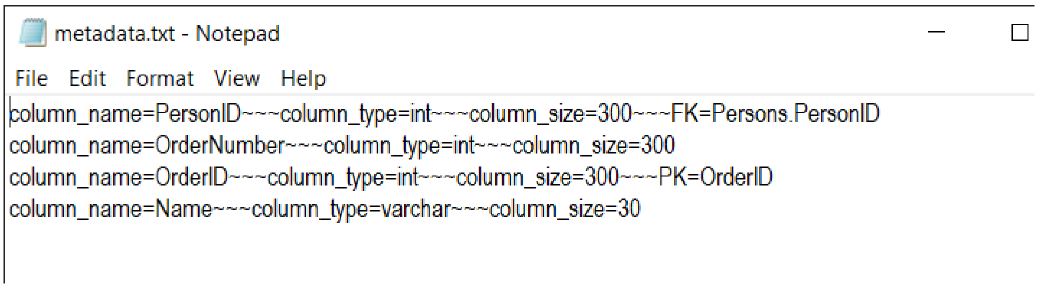
Figure 6 shows all the tables present inside the database called testDatabase.



Name	Date modified	Type	Size
Employee.txt	06-04-2021 11:21 AM	Text Document	
metadata.txt	06-04-2021 11:20 AM	Text Document	

Figure 7: Table structure

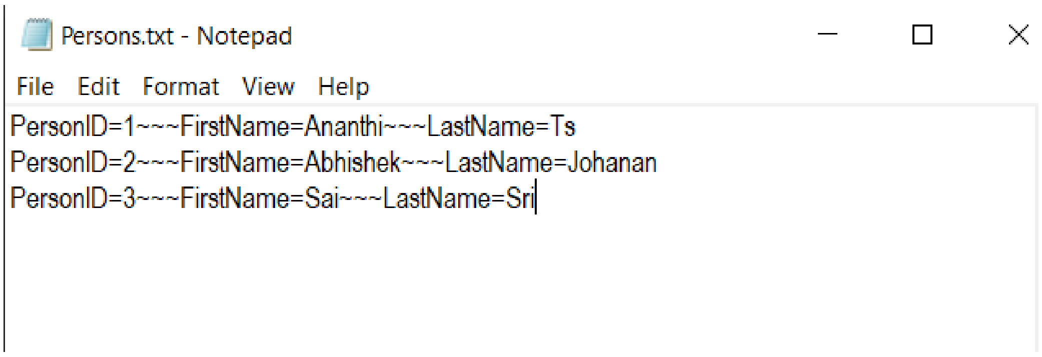
The metadata file and table information of a table Employee is shown in Fig 7.



```
column_name=PersonID~~~column_type=int~~~column_size=300~~~FK=Persons.PersonID
column_name=OrderNumber~~~column_type=int~~~column_size=300
column_name=OrderID~~~column_type=int~~~column_size=300~~~PK=OrderID
column_name=Name~~~column_type=varchar~~~column_size=30
```

Figure 8: Metadata structure

The structure of the metadata file is shown in Fig 8 which contains column name, column type, column size, primary key and foreign key values.



```
PersonID=1~~~FirstName=Ananthi~~~LastName=Ts
PersonID=2~~~FirstName=Abhishek~~~LastName=Johanan
PersonID=3~~~FirstName=Sai~~~LastName=Sri
```

Figure 9: Table content

Fig 9 shows how the data is stored in a separate file and each line in the file denotes the single table row. Each row contains the data in following format,

<column_name>=<column_value>~~~<column_name>=<column_value>……….

The columns are separated using delimiter (~~~).

Data structures

A Table class is created which is used to save the table information like table name, database name, column metadata and the table values.

Each column information is stored in a **Map<String,Column>,** which stores column name as key and Column data structure (Which contains column specific values like column name, type, size, primary and foreign key, etc..).

The table data is stored in a **List<Map<String,Object>>** where in each row is represented as a map, with key as column name and value as corresponding column values.

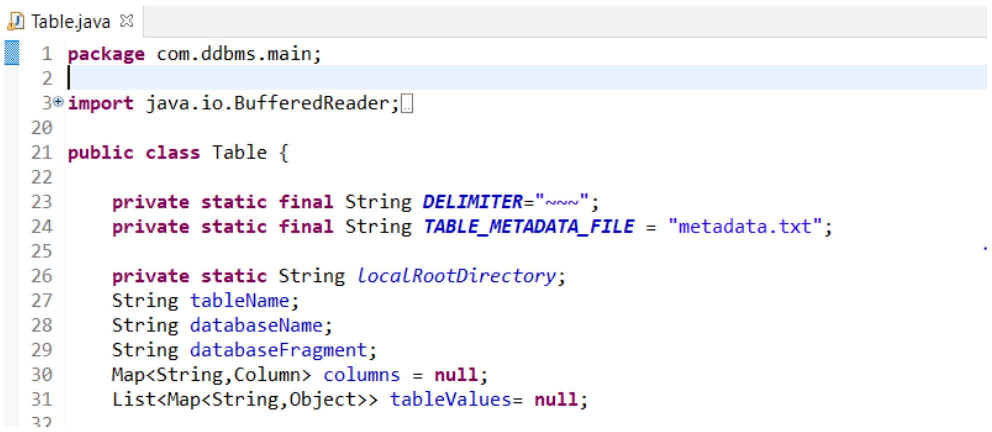
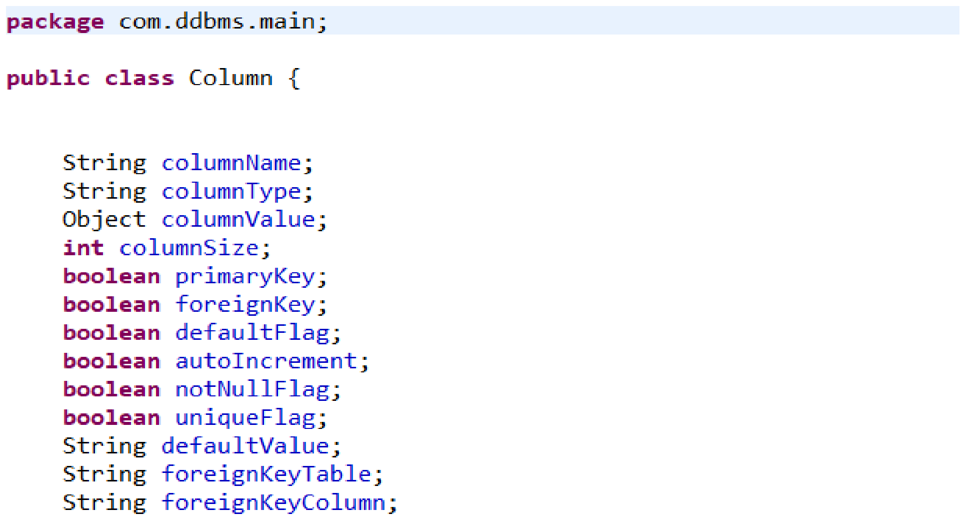


Table.java

```java
package com.ddbms.main;

import java.io.BufferedReader;

public class Table {

    private static final String DELIMITER="~~~";
    private static final String TABLE_METADATA_FILE = "metadata.txt";

    private static String localRootDirectory;
    String tableName;
    String databaseName;
    String databaseFragment;
    Map<String,Column> columns = null;
    List<Map<String,Object>> tableValues= null;
```

Figure 10: Table data structure

Figure 10 shows the Table data structure which shows how the table information are stored. The columns map contains the table metadata information and the table values list contains the overall table data.



```java
package com.ddbms.main;

public class Column {


    String columnName;
    String columnType;
    Object columnValue;
    int columnSize;
    boolean primaryKey;
    boolean foreignKey;
    boolean defaultFlag;
    boolean autoIncrement;
    boolean notNullFlag;
    boolean uniqueFlag;
    String defaultValue;
    String foreignKeyTable;
    String foreignKeyColumn;
```

Figure 11: Column data structure

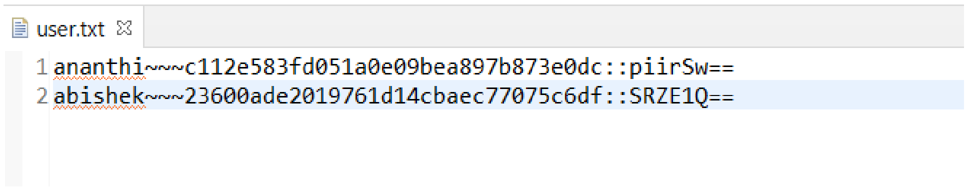
Figure 11 shows the Column data structure where all the details about a single column is stored in the Column class. The column name, column size, column type, column value is mentioned. If the column is primary key the corresponding flag is updated and similarly if the column is referencing to any other foreign key, then that value is also stored inside the class.

Delimiter: ~~~

The application uses three '~' symbol as delimiter in all the property files, metadata and table files.

User Authentication

The user passwords are encrypted and decrypted using AES encryption algorithm.



```
ananthi~~~c112e583fd051a0e09bea897b873e0dc::piirSw==
abishek~~~23600ade2019761d14cbaec77075c6df::SRZE1Q==
```

Distributed Database Communication

Client

1. START
2. Authenticate user.
3. If valid user, connect to server, else go to step 2.
4. Take query input from user.
5. Check if query is to be processed on local or another site based on GDD.
6. If local, pass query to Data Engine.
7. If non-local, send request to server and pass query to Server.
8. Wait for response from Server
9. Update response from server to User
10. If User gives "EXIT" query, go to step 11. Else go to step 4.
11. Close connection with server.
12. END

Listener

1. START
2. Connect to server.
3. Wait for request from Server.
4. Pass the request (query) to the Database Engine.
5. Send response to Server, response starts with "BEGIN" and ends with "END"
6. If request from Server is "EXIT" go to Step 7. Else Go to step3.
7. END

Server

1. START
2. Get connections from Client and Listener.
3. Wait for request from Client.
4. Forward request to appropriate Listener.
5. Wait for response from Listener
6. Forward Listener response to respective Client.
7. If request from Client is "EXIT" go to Step 8. Else Go to step3.
8. END

In this project, the server was deployed on to a Google Cloud Platform (GCP), Virtual Machine (VM). The local hosts (both client and listener) connected to the server through the Public IP and respective ports of the VM.

Database Engine

Create

1. START
2. Load GDD.
3. Split the Query based on spaces.
4. Check if query is complete and has valid words (such as Database.TableName, parenthesis match).
5. Check if database exists.
6. Check if table does not exist.
7. If any foreign keys, it will check if the referenced table and column exists.
8. Create metadata.txt file with all the column data.
9. END

Insert

1. START
2. Load GDD.
3. Split the Query based on spaces.
4. Check if query is complete and has valid words (such as Database.TableName, INTO, VALUES ...).
5. Check if database exists.
6. Check if table exists.
7. If table exists, Load table metadata.
8. Check if column names exists and the values data types match
9. If Step 8. is true, Check if the table is locked.
10. If table is not locked, create table file if it does not exist.
11. Lock the table.
12. For each Column and its value
13. If it is a primary key, check for primary key constraint.
14. If it is a foreign key, check for foreign key constraint,
15. If normal column, insert into file
16. END FOR
17. Unlock table.
18. END

Select

1. START
2. Load GDD.
3. Split the Query based on spaces.
4. Check if query is complete and has valid words (such as Database.TableName, parenthesis match).
5. Check if database exists.
6. Check if table exists.
7. If table exists, Load table metadata.

8. Check if column names exists and the values data types match
9. If Step 8. is true, fetches the table values
10. If any conditions are mentioned, the table values will be filtered.
11. Print the table values
12. END

Update

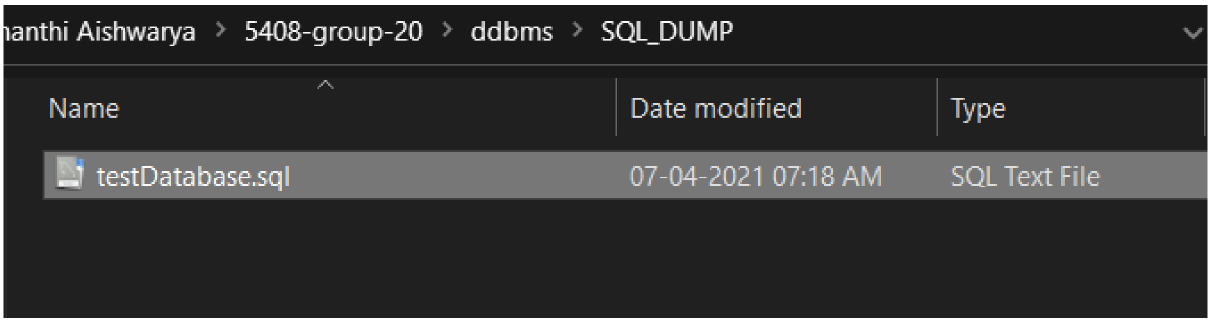
1. START
2. Load GDD
3. Split the Query based on spaces.
4. Check if query is complete and has valid words (such as Database. TableName , WHERE).
5. Check if database exists.
6. Check if table exists.
7. If table exists, Load table metadata.
8. Check if all the column names exists or not
9. If Step 8. is true, Check if the table is locked.
10. If table is not locked, lock the table.
11. Update the specified column values.
12. Unlock table.
13. END

Drop

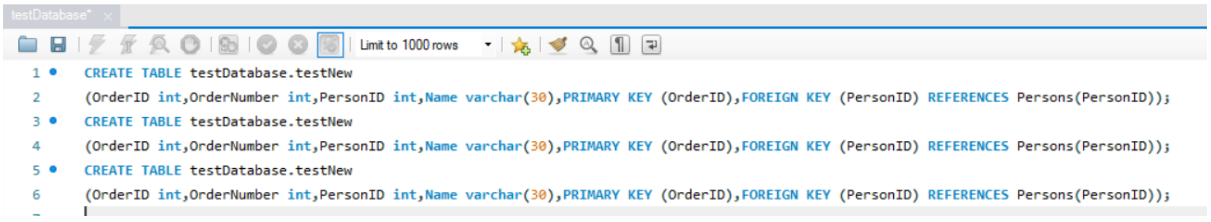
1. START
2. Load GDD
3. Split the Query based on spaces.
4. Check if query is complete and has valid words (such as Database.TableName, TABLE, DROP).
5. Check if database exists.
6. Check if table exists.
7. If step 6 is true, Check if the table is locked.
8. If table is not locked, check if the table has a column which is referenced as a foreign key in other tables.
9. If no columns are referred, delete file data, metadata.
10. Update GDD
11. END

Dump

Syntax format - create dump testDatabase



Name	Date modified	Type
testDatabase.sql	07-04-2021 07:18 AM	SQL Text File

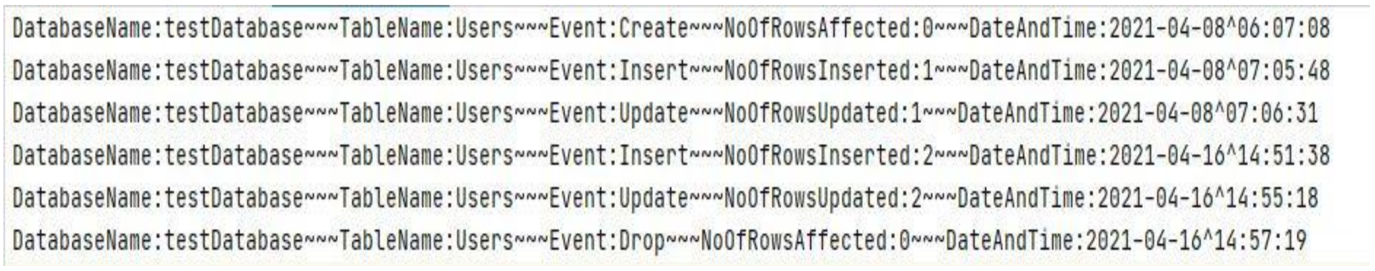

testDatabase*

Limit to 1000 rows

```
CREATE TABLE testDatabase.testNew
(OrderID int,OrderNumber int,PersonID int,Name varchar(30),PRIMARY KEY (OrderID),FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));
CREATE TABLE testDatabase.testNew
(OrderID int,OrderNumber int,PersonID int,Name varchar(30),PRIMARY KEY (OrderID),FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));
CREATE TABLE testDatabase.testNew
(OrderID int,OrderNumber int,PersonID int,Name varchar(30),PRIMARY KEY (OrderID),FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));
```

Event Logs

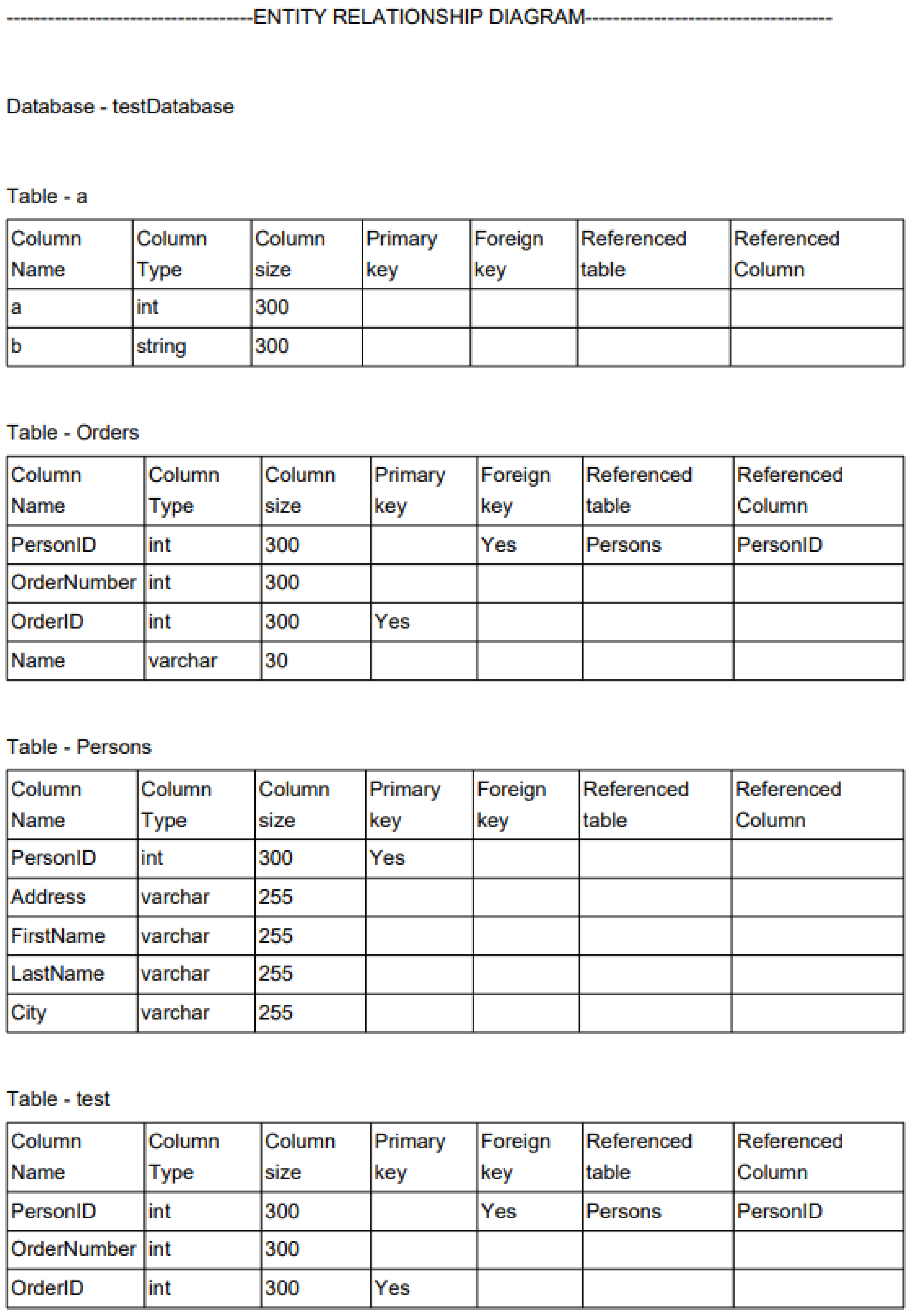
Whenever an operation (insert, update, create etc) is performed it gets updated in the event logs file. It contains Information like database name and table name on which the operation is performed, the event name, the number of rows affected by the operation and the timestamp of the operation.



```
DatabaseName:testDatabase~~~TableName:Users~~~Event:Create~~~NoOfRowsAffected:0~~~DateAndTime:2021-04-08^06:07:08
DatabaseName:testDatabase~~~TableName:Users~~~Event:Insert~~~NoOfRowsInserted:1~~~DateAndTime:2021-04-08^07:05:48
DatabaseName:testDatabase~~~TableName:Users~~~Event:Update~~~NoOfRowsUpdated:1~~~DateAndTime:2021-04-08^07:06:31
DatabaseName:testDatabase~~~TableName:Users~~~Event:Insert~~~NoOfRowsInserted:2~~~DateAndTime:2021-04-16^14:51:38
DatabaseName:testDatabase~~~TableName:Users~~~Event:Update~~~NoOfRowsUpdated:2~~~DateAndTime:2021-04-16^14:55:18
DatabaseName:testDatabase~~~TableName:Users~~~Event:Drop~~~NoOfRowsAffected:0~~~DateAndTime:2021-04-16^14:57:19
```

ERD

Syntax format - create erd testDatabase



-----------------------------------ENTITY RELATIONSHIP DIAGRAM-----------------------------------

Database - testDatabase

Table - a

Column Name	Column Type	Column size	Primary key	Foreign key	Referenced table	Referenced Column
a	int	300				
b	string	300				

Table - Orders

Column Name	Column Type	Column size	Primary key	Foreign key	Referenced table	Referenced Column
PersonID	int	300		Yes	Persons	PersonID
OrderNumber	int	300				
OrderID	int	300	Yes			
Name	varchar	30				

Table - Persons

Column Name	Column Type	Column size	Primary key	Foreign key	Referenced table	Referenced Column
PersonID	int	300	Yes			
Address	varchar	255				
FirstName	varchar	255				
LastName	varchar	255				
City	varchar	255				

Table - test

Column Name	Column Type	Column size	Primary key	Foreign key	Referenced table	Referenced Column
PersonID	int	300		Yes	Persons	PersonID
OrderNumber	int	300				
OrderID	int	300	Yes			

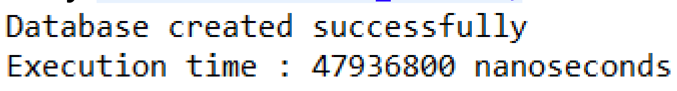
Use Cases

- User can login using his/her credentials in which password is encrypted.
- User after successfully logging in, can perform operations like create, select, update, drop etc.
- User can create a database and add tables to the database.
- Each database can have multiple tables and each table can have multiple records.
- User can insert records to a specified table in a database by giving the details of database
- User can select specific values from a table using select query. We can select multiple values from a table or select all the values at a time. We can also select specific values by providing single or multiple conditions.
- User can update the existing values in table by using update query. We can update multiple values in a table by giving single or multiple conditions.
- User can also drop a table by using the drop query.
- User can create an ERD of a specific database.
- User can also create a dump of a specific database.
- Whenever an operation is performed it gets updated to the eventlogs.
- All the operations are performed using locking mechanism. That means if one user is performing operation on a table another user cannot perform operation on the same table at a time.
- All the above specified operations can be done locally as well as remotely.

Test Cases

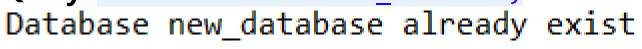
Create

1. **Query:** `create database new_database;`



```
Database created successfully
Execution time : 47936800 nanoseconds
```

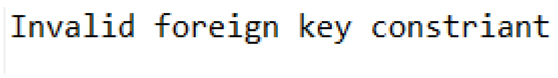
2. **Query:** `create database new_database;`



```
Database new_database already exist
```

3. **Query:**

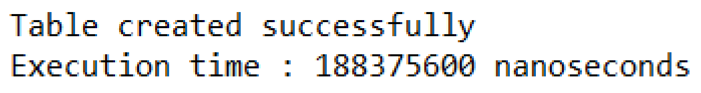
```
CREATE TABLE new_database.testDb (OrderID int,OrderNumber int,PersonID
int,Name varchar(30),PRIMARY KEY (OrderID),FOREIGN KEY (PersonID)
REFERENCES UsersNew(PersonID));
```



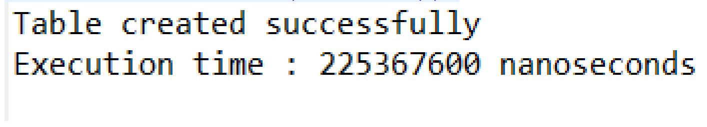
```
Invalid foreign key constriant
```

4. **Query:**

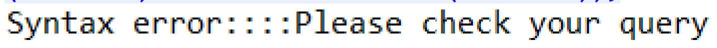
```
CREATE TABLE new_database.UsersNew (PersonID int, LastName varchar(255),
FirstName varchar(255), Address varchar(255), City varchar(255), PRIMARY
KEY (PersonID) );
```



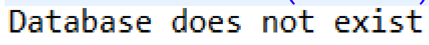
```
Table created successfully
Execution time : 188375600 nanoseconds
```

5. **Query:**
```
CREATE TABLE new_database.testDb (OrderID int,OrderNumber int,PersonID
int,Name varchar(30),PRIMARY KEY (OrderID),FOREIGN KEY (PersonID)
REFERENCES UsersNew(PersonID));
```


```
Table created successfully
Execution time : 225367600 nanoseconds
```

6. Query:
```
CREATE TABLE new_database.testDb asdv (OrderID as int,OrderNumber
int,PersonID int,Name varchar(30),PRIMARY KEY (OrderID),FOREIGN KEY
(PersonID) REFERENCES UsersNew(PersonID));
```


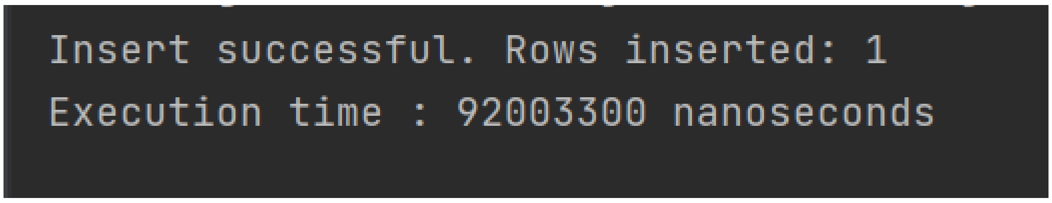
```
Syntax error::::Please check your query
```

7. Query:
```
CREATE TABLE new_database2.testDb (OrderID int,OrderNumber int,PersonID
int,Name varchar(30),PRIMARY KEY (OrderID),FOREIGN KEY (PersonID)
REFERENCES UsersNew(PersonID));
```


```
Database does not exist
```

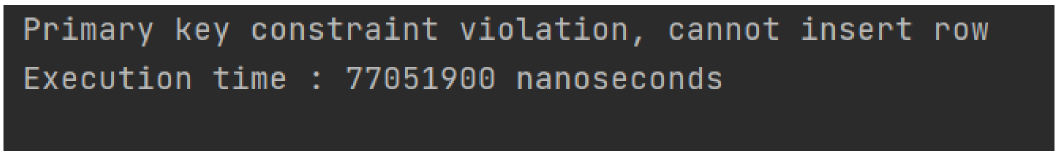
Insert

1. **Query:** INSERT INTO testDatabase.Persons (PersonID,FirstName,Address) VALUES (8,'bhargav','hyderabad');



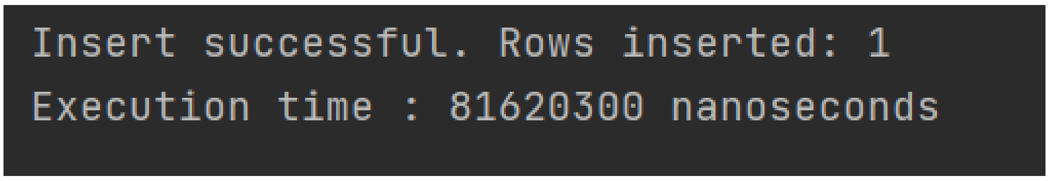
```
Insert successful. Rows inserted: 1
Execution time : 92003300 nanoseconds
```

2. **Query:** INSERT INTO testDatabase.Persons (PersonID,FirstName,Address) VALUES (8,'bhargav','hyderabad');



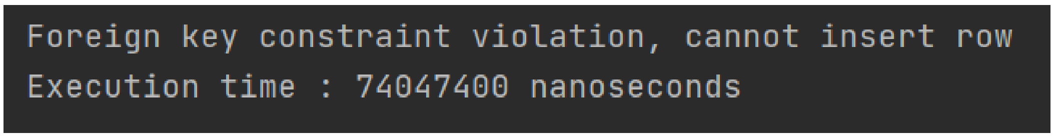
```
Primary key constraint violation, cannot insert row
Execution time : 77051900 nanoseconds
```

3. **Query:** INSERT INTO testDatabase.userOrders (OrderID,OrderNumber,PersonID,Name) VALUES (12312,99999,8,'Soda');



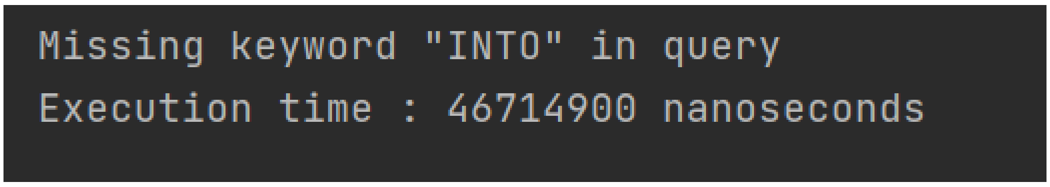
```
Insert successful. Rows inserted: 1
Execution time : 81620300 nanoseconds
```

4. **Query:** INSERT INTO testDatabase.userOrders (OrderID,OrderNumber,PersonID,Name) VALUES (12312,99999,2,’Soda’);



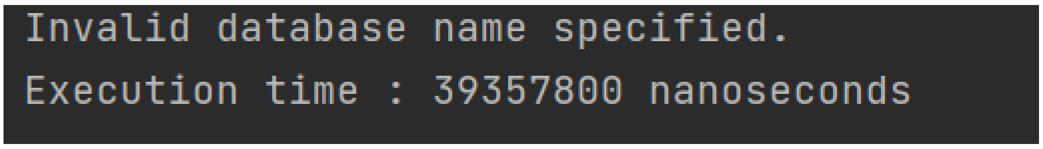
```
Foreign key constraint violation, cannot insert row
Execution time : 74047400 nanoseconds
```

5. **Query:** INTO INSERT INTOOOO testDatabase.Users (PersonID,FirstName,Address) VALUES (2,'bhargav','hyderabad');



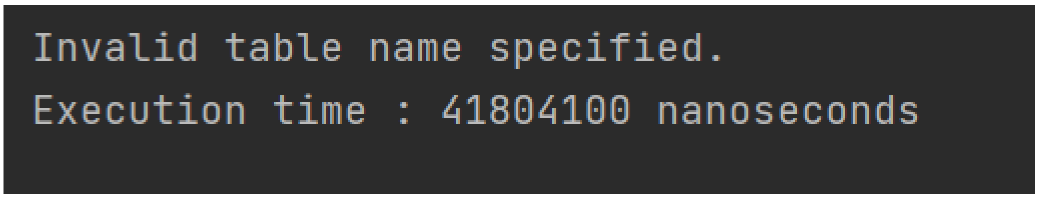
```
Missing keyword "INTO" in query
Execution time : 46714900 nanoseconds
```

6. **Query:** INSERT INTO testDatbase.Users (PersonID,FirstName,Address) VALUES (2,'bhargav','hyderabad');



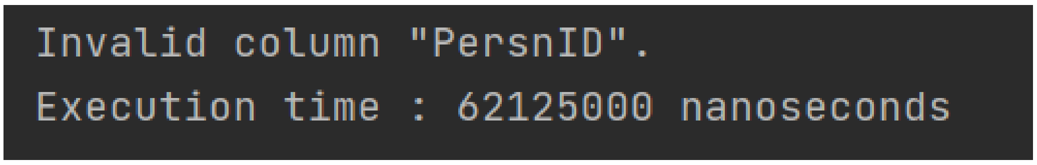
```
Invalid database name specified.
Execution time : 39357800 nanoseconds
```

7. **Query:** INSERT INTO testDatabase.Uers (PersonID,FirstName,Address) VALUES (2,'bhargav','hyderabad');



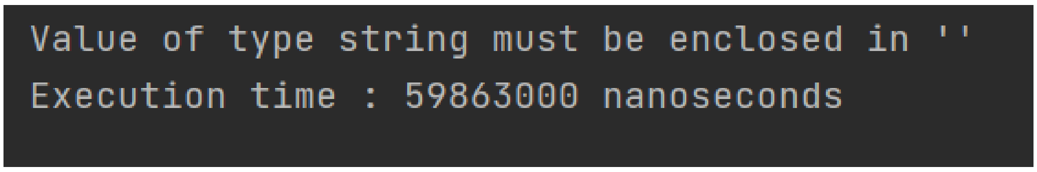
```
Invalid table name specified.
Execution time : 41804100 nanoseconds
```

8. **Query:** INSERT INTO testDatabase.Persons (PersnID,FirstName,Address) VALUES (2,'bhargav','hyderabad');



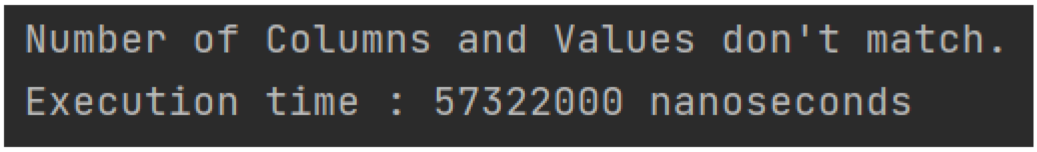
```
Invalid column "PersnID".
Execution time : 62125000 nanoseconds
```

9. **Query:** INSERT INTO testDatabase.Persons (PersonID,FirstName,Address) VALUES (2,2342342,'hyderabad');



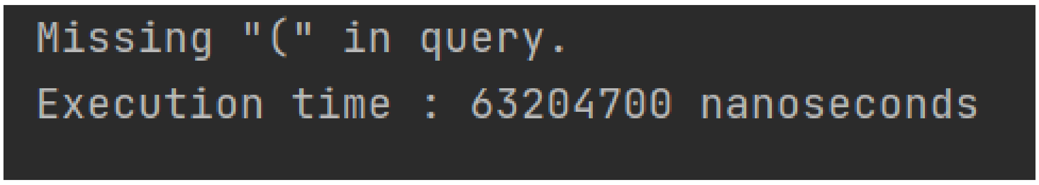
```
Value of type string must be enclosed in ''
Execution time : 59863000 nanoseconds
```

10. **Query:** INSERT INTO testDatabase.Persons (PersonID,FirstName,Address) VALUES (2,'hyderabad');



```
Number of Columns and Values don't match.
Execution time : 57322000 nanoseconds
```

11. **Query:** INSERT INTO testDatabase.Persons (PersonID,FirstName,Address) VALUES 2,'bhargav','hyderabad');



```
Missing "(" in query.
Execution time : 63204700 nanoseconds
```

Select

1. **Query:** `select * from testDatabase.Persons;`

```
{PersonID=1, Address=NULL, FirstName=Ananthi, LastName=Ts, City=NULL}
{PersonID=2, Address=NULL, FirstName=Abhishek, LastName=Johanan, City=NULL}
{PersonID=3, Address=NULL, FirstName=Sai, LastName=Sri, City=NULL}
3 rows returned
Execution time : 245485299 nanoseconds
```

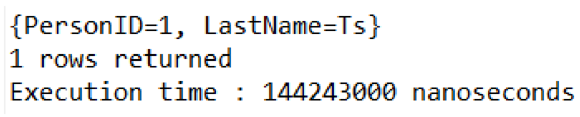
2. **Query:**

```
select * from testDatabase.Persons where PersonID=1 and FirstName=Ananthi;
{PersonID=1, Address=NULL, FirstName=Ananthi, LastName=Ts, City=NULL}
1 rows returned
Execution time : 189945100 nanoseconds
```

3. **Query:** `select PersonID from testDatabase.Persons;`

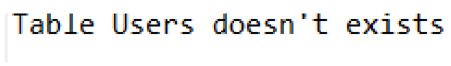
```
{PersonID=1}
{PersonID=2}
{PersonID=3}
3 rows returned
Execution time : 93217600 nanoseconds
```

4. **Query:** `select PersonID,LastName from testDatabase.Persons where PersonID=1 and FirstName=Ananthi;`

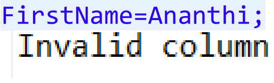


```
{PersonID=1, LastName=Ts}
1 rows returned
Execution time : 144243000 nanoseconds
```

5. **Query:** `select * from testDatabase.Users;`



```
Table Users doesn't exists
```

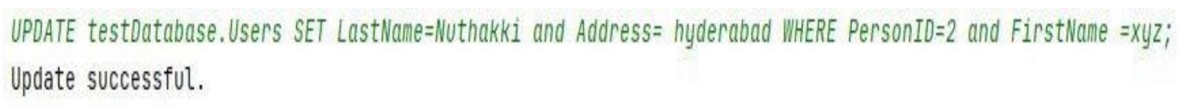
6. **Query:**
```
select PersonID,La from testDatabase.Persons where PersonID=1 and
FirstName=Ananthi;
```


```
Invalid column
```

Update

1. **Query:** UPDATE testDatabase.Users SET LastName=Nuthakki and Address = hyderabad WHERE PersonID=2 and FirstName = xyz;

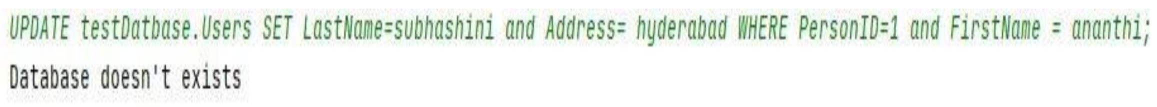
This update query updates the specified multiple values in a given table for the specified multiple conditions.



```
UPDATE testDatabase.Users SET LastName=Nuthakki and Address= hyderabad WHERE PersonID=2 and FirstName =xyz;
Update successful.
```

2. **Query:** UPDATE testDatbase.Users SET LastName=subhashini and Address= hyderabad WHERE PersonID=1 and FirstName = ananthi;

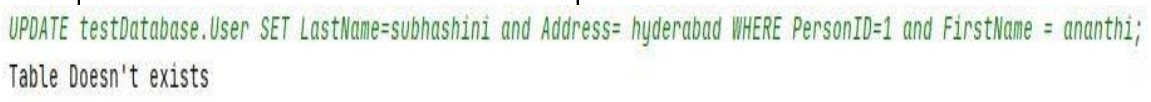
This update statement is not executed as the specified database does not exists.



```
UPDATE testDatbase.Users SET LastName=subhashini and Address= hyderabad WHERE PersonID=1 and FirstName = ananthi;
Database doesn't exists
```

3. **Query:** UPDATE testDatabase.User SET LastName=subhashini and Address= hyderabad WHERE PersonID=1 and FirstName = ananthi;

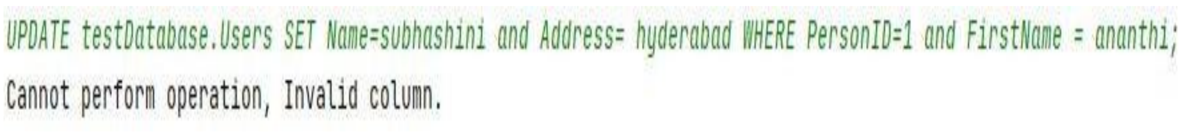
This update statement is not executed as the specified table does not exists.



```
UPDATE testDatabase.User SET LastName=subhashini and Address= hyderabad WHERE PersonID=1 and FirstName = ananthi;
Table Doesn't exists
```

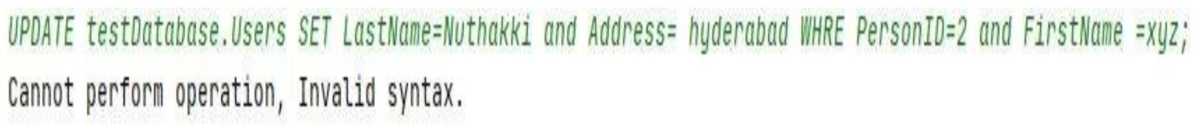
4. **Query:** UPDATE testDatabase.Users SET FirstName=ananthi and Adress= hyderabad WHERE PersonID=1;

This update statement is not executed as the specified column name (i.e. Adress) does not exists.



```
UPDATE testDatabase.Users SET Name=subhashini and Address= hyderabad WHERE PersonID=1 and FirstName = ananthi;
Cannot perform operation, Invalid column.
```

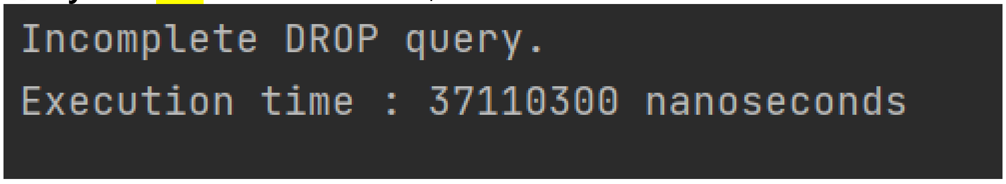
5. **Query:** UPDATE testDatabase.Users SET LastName = Nuthakki and Address= hyderabad WHRE PersonID=2 and FirstName = xyz;



```
UPDATE testDatabase.Users SET LastName=Nuthakki and Address= hyderabad WHRE PersonID=2 and FirstName =xyz;
Cannot perform operation, Invalid syntax.
```

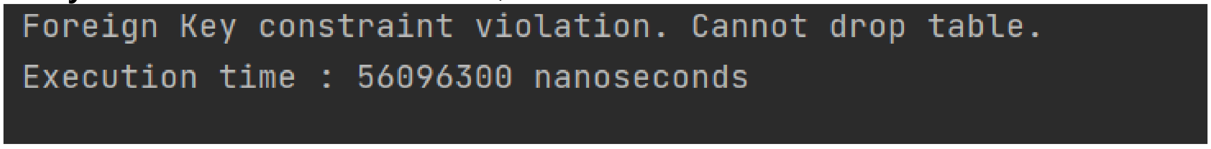
Drop

Query: DROP testDatabase.Orders;



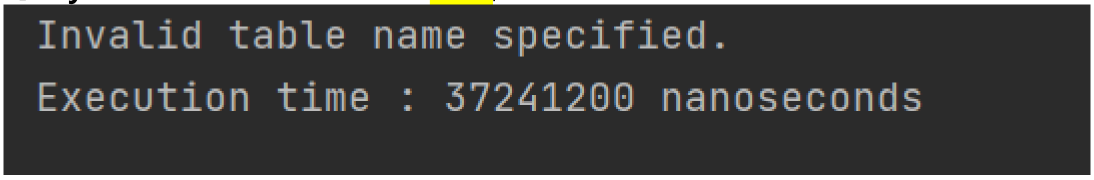
```
Incomplete DROP query.
Execution time : 37110300 nanoseconds
```

Query: DROP table testDatabase.Orders;



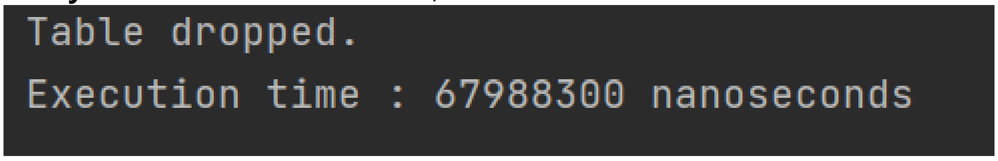
```
Foreign Key constraint violation. Cannot drop table.
Execution time : 56096300 nanoseconds
```

Query: DROP TABLE testDatabase.rders;



```
Invalid table name specified.
Execution time : 37241200 nanoseconds
```

Query: DROP TABLE testDatabase.a;



```
Table dropped.
Execution time : 67988300 nanoseconds
```

Limitations and Future Work

- The current architecture supports only 2 local hosts. Expanding this to multiple hosts communicating through the server is a future work.
- Each query requires the database followed by the table to be specified, the functionality to select the database at the beginning itself and making queries more flexible is another future work.
- Cardinality relationships such as one-one, one-many etc… to be specified in the ERD is another enhancement that can be added.