MAJOR PROJECT 1

END-TERM REPORT

ON

Relational Database With Minimal Functionality

Submitted By

Prakash Tiwari	Akshit Chauhan	Gaurav Singh
500062116	500062444	500062611

Under the guidance of

Amit Singh

Assistant Professor, SoCSE

Department of Cybernetics,

School of Computer Science

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

Dehradun-248007

Abstract

There are many applications out there today that use some sort of local storage format to keep data collocated with the application. However, without a database, most of the proposed solutions cannot handle the atomicity of multiple changes (all or nothing, and the management of transactions that may impact each other), and there are no built-in recovery mechanisms in case of a failure during the transaction. Here we propose a project to design a relational database with minimal functionality.

A relational database is a collection of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows. Tables are used to hold information about the objects to be represented in the database. Advantages of relational model are simplicity, structural independence, ease of use,query capability, data independence, scalability.

Introduction:

A collected information which is in an organized form for easier access, management, and various updating is known as a database. The relational database was invented by E. F. Codd at IBM in 1970. The data in relational databases is present in tabular form, i.e. as a *collection* of tables with each table consisting of a set of rows and columns) and it provides relational operators to manipulate the data in tabular form.

1. Database Engine: A database engine (or storage engine) is the underlying software component that a database management system (DBMS) uses to create, read, update and delete (CRUD) data from a database. Most database management systems include their own application programming interface (API) that allows the user to interact with their underlying engine without going through the user interface of the DBMS.

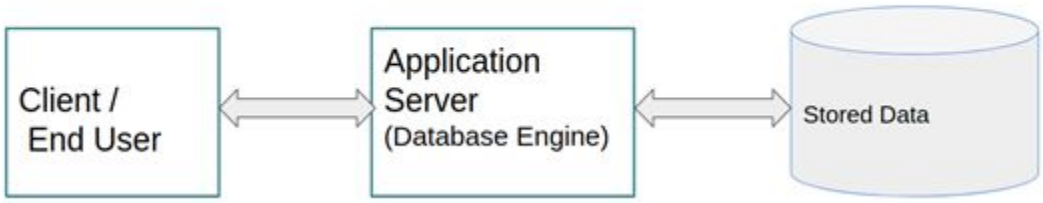

Fig 1.0

2. Database storage structures:

Databases may store data in many data structure types. Common examples are the following: ordered/unordered, flat files, hash tables, B+ trees, ISAM and heaps.

In our project we are using B+ tree. These are the most commonly used in practice. A B+ tree is an m-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children. The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context in particular, file systems. This is primarily because unlike binary search trees, B+ trees have very high fanout (number of pointers to child nodes in a node,typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree. Time taken to access any record is the same because the same number of nodes is searched. Access in B+ tree is fast.

Problem Statement:

There is always a need to store, retrieve and update data efficiently. Also remove data redundancy and inconsistency. In this project Relational Database will solve the problem of redundant data and inconsistency. For performing efficient operation in the database it is designed using B+ tree.

Objective:

Creating A Relational Database to perform CRUD operations.

Sub-Objective:

- Store data internally using B+ tree and creating caching mechanisms for fast retrieval.
- Apply hashing and indexing in the database engine to find data quickly.

Objective Achieved:

· We have achieved validation of query(Create, Delete, Insert, Select) and extraction of information from query that the user enters.

· We are able to store the information that the user enters in a cache from which execution of the query will take place.

. Task of Storing data into B+ tree and creating a caching mechanism for fast retrieval is successfully achieved.

. We have successfully applied hashing and indexing in the database engine for faster searching of data.

Literature Review:

Since the 90's, databases have shown tremendous growth. This growth can be determined in different aspects. Different demands of every era give databases a new bunch of challenges. To achieve those challenges, researchers come up with different ideas and combinations. These various combinations enhance features of databases and this way databases start evolving from one period to another. Database that we had in 1960 is entirely, absolutely different from what we have now.[3] In this phase of evaluation The relational database was invented by E. F. Codd at IBM in 1970. In a relational database, all data is held in tables, which are made up of rows and columns. Each table has one or more columns, and each column is assigned a specific data type, such as an integer number, a sequence of characters (for text), or a date. Each row in the table has a value for each column. A relational database usually contains many relations, with tuples in relations that are related in various ways,A *relational database* is a type of database. It uses a structure that allows us to identify and access data *in relation* to another piece of data in the database[1]. The main advantage of relational databases is that they enable users to easily categorize and store data that can later be queried and filtered to extract specific information for reports. Relational databases are also easy to extend and aren't reliant on physical organization. After the original database creation, a new data category can be added without all existing applications being modified. [2]

Methodology:

Software Development Model:

In our project we are using the Waterfall model to build our project. Waterfall model is the oldest software development model and all other software development models are formed on the basis of the Waterfall model. There are six phases in Waterfall model

1) Feasibility Study: The feasibility study involves understanding the problem and then determines the various possible strategies to solve the problem.

2) Requirement Analysis: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly.

3) Designing: The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

4) Coding and Unit Testing: In coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded.

5) Integration and System Testing: Integration of various modules is carried out incrementally over a number of steps.

6) Maintenance: It includes corrective maintenance, perfective maintenance, Adaptive maintenance.

Implementation:

The query entered by the user is firstly checked if it is valid or not. For that the object of ValidateQuery is created and its method isValid() called. Internally for the parsing of the query we are using Hyrise c++ Sql Parser. Which checks the grammar of the query and generates the parsing tree. If the query is valid then we extract the information of the query entered by the user. That includes query type (READ or WRITE), name of table, table field information and where clauses. That info is stored into a cache for the validation of the name of the table if it is Create command. (If the table name already exists). We are using B+ tree to store data. B+ tree is an algorithm (i.e. a set of rules) allowing not just to quickly find the entries in a database, but also add / edit and delete these entries (assuming you can use an eraser). We manage a separate tree for each table and a master tree which will help to find the table info. The information of the table will be stored in the leaf of the tree. In which each row will be stored in a single node. By using this we can perform all operations quickly.

Architecture Of DBplus:

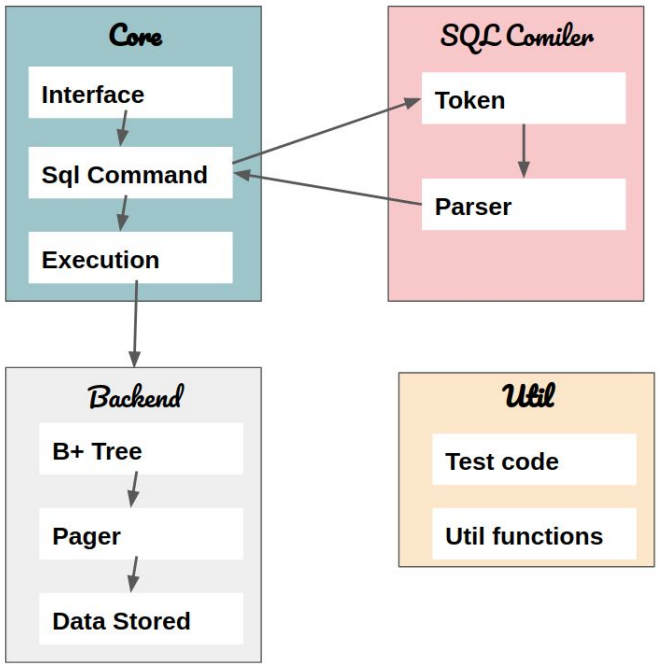

Fig 2.0

Core:-

When database is run then user interact with DBPlus interface where query is entered after that query goes to Sql Compiler where we query is being parsed, if the query is invalid then it prints the error message otherwise if the query is valid it again goes to core section and execution takes place.

Sql compiler:-

This part majorly contains two sections first is parser and other is token. Parsers parse the query into c++ objects that will be used in further processing. Here we have used Hyrise Sql Parser to parse the query. Once a query is parsed then a token is generated for the queries in which the grammar of the query is validated.

Backend:-

If the query is valid and parsed successfully then it forwarded to the backend where all the database interaction operations performs in backend which are as follows:

In the backend, firstly it will check the type of query,

- If the query type is create, then it checks if the table name already existed or not, if the table name is already existed then an error message is printed on the screen otherwise the table will be created.
- If the query type is select then again it checks if the table name already existed or not, if the table name is already existed then the select operations take place and print the result otherwise an error message is printed on the screen
- If the query type is delete, then again it checks if the table name already existed or not, if the table name does not exist then error message is printed otherwise delete operations take place and it will delete table data from database db from the leaf of b+ tree.
- If the query type is insert, then again it checks if the table name already existed or not, if the table name does not exist then error message is printed otherwise insert operations take place and it will insert row into database db into leaf of the b+ tree.
- If the query type is update, then again it checks if the table name already existed or not, if the table name does not exist then an error message is printed otherwise an update operation is being performed.

Util:-

This section of DBPlus contains utility functions that are being used to perform backend operations. The function are like: printTable(), cellStructure(), help(), split(), toUpper(), toLower(), extractInfo() etc are implemented in this. It also contains

File and directory structure:

Directory	Files
src	**Contains Driver and build files:** driver.cpp,make,cmake.txt
src/include	**Contains all interfaces:** Insert.hpp, create.hpp, update.hpp, delete.hpp, select.hpp, execute.hpp, validatequery.hpp, global.hpp
src/executeQuery	**Contains actual cpp files that have functions.** Insert.cpp, create.cpp, update.cpp, delete.cpp, select.cpp,

	execute.cpp, validatequery.cpp, global.cpp
src/util	**Contains all utility functions used to implement.** datatype.hpp, enums.hpp, util.hpp, util.cpp

Files Description:

src/driver.cpp: This is the driver file responsible for running the project and includes all the headers of the project. First query/command is entered in the interface from there it goes to the parser where the query is being validated if the query is valid then command is executed otherwise it prints the error message.

src/executeQuery/validatequery.cpp: In this cpp file **existTable(table_name)** and **isValid(query,err,result)** method is implemented where validation of query takes place. In the existTable method it checks the existence of the table that is present or not and in the isValid method query is being validated.

src/executeQuery/execute.cpp: In this cpp file, **ExecuteQuery** class contains **executeQuery(result**) method that is responsible for the execution of query and all the **operations(create, select, insert, delete & update)** are being executed.

src/executeQuery/create.cpp: In this cpp file, **insertTableIndex(info, table)** and **createStatement(create_stmt)** method are executed that is responsible for the create table.

src/executeQuery/select.cpp: In this cpp file **printIndent(size), printResult(columns, table), printResult(columns,exist_columns,table),fillTableColumns(table,sequence),my_printExpression(expr, columns, all_field), my_printSelectStatementInfo(table, columns, const stmt, all_field), my_printOperatorExpression(expr,numIndent), my_printTableRefInfo(table, tableName)** and **selectStatement(stmt)** method are responsible for select operation in which selectStatement(stmt) method calls the above mentioned method to perform operations and all the operations are being performed at the leaf of b+ tree.

src/executeQuery/delete.hpp: This cpp file is responsible for the delete operation in db-plus database.

src/executeQuery/insert.hpp: In this cpp file **fillTableColumns(t, table, sequence), insertInToDatabase(info), fillstatementcolumns(stmt, t)** and **insertStatement(smrt)** are responsible for insert operation in the db-plus database in which data is inserted at leaf of b+ tree.

Table Index Managing :

To store information about the table, we store all the information in table.db and make a table_index.txt, that stores the table name along with a pointer to the start of the table information by which we can directly access the table information in key value pairs. We are using a special structure to store the information in files. Followings are graphical representation of how data stored in files:

Format of table.db :

Each line follow this format

Table name	column-1	data _type	isNull (0/1)	Primary key	Column -2	…..

Format of table_index.db :

1st line : metadata

total_tables	Last_points to be inserted	Total bytes

Rest of the line store key_value pair :

(Key) table_name	(value) points

Algorithms:

DB Plus Algorithm:

1) Enter query
2) Parse the query and check
   if(!valid)
      return error;
   else if(valid)
      execute query;
   a) It will preprocess the table so that it can be used globally to
      perform efficient operations:-
      Check the type of query:-
      switch()
        case(create) :- If the sql statement is create type
              Perform the operation for create table;
              if(table name already exist)
                return error;

```
        else
          create table;
    case(select) :- Perform operations for select table
        if(table not exist)
          return error;
        else
          check field/column/expression
          & return result;
    case(delete) :- perform operations for delete
        if(table not exist)
          return error;
        else
          delete table data from database db
                  &
          delete table information and update index
    case(insert) :- perform operations for insert statement
        if(table not exist)
          return error;
        else if(column name and database type not valid)
          return error;
        else
          insert row into database db into leaf of the b+ tree
    case(update) :- perform pre-processing for update information
        if(table not exist)
          return error;
        else
          perform update;
```

3) exit;

Indexing Algorithm:

1) Open file as file stream
2) For insert :-
   key = table_name;
   value = (last_poition)+1;
   String str = key + " " +value;
   Write to the file and insert data into table.db

3) For Delete :-
   String metadata: store metadata;
   store all key- value : map<string,int>;

map.remove(table_name);
total_table--;
total_byte = total bytes of (table.db);
last_pointer = last marks of(table.db);
update metadata;
insert map into table insert.txt;
update table.db;

4) For retrieving into :-
store all key value:map;
store metadata in:string
split string to get all fields of metadata
5) exit;

Table Management Algorithm:

1) Open table.db file and table_index.txt
2) Insert

Make str according to format
Pointer : last_pointer
Insert : to last pointer
Update metadata of table_index
Insert table in table_index

3) Delete

Find pointer of table_name from index
Seek g(pointer)
Copy string from pointer -- end of file
Delete file
Append new string to file
Delete(table) from index

4) Select(retrieve) data:

Find pointer of table_name
seekg(pointer)
Store string line in any string
Split the string according to format of table.db

5) exit

B+ Tree Algorithm:

A) Insertion in B+ Tree

1. Every element is inserted into a leaf node. So, go to the appropriate leaf node.
2. Insert the key into the leaf node in increasing order only if there is no overflow. If there is an overflow go ahead with the following steps mentioned below to deal with overflow while maintaining the B+ Tree properties.

Case-1: Overflow in leaf node

1. Split the leaf node into two nodes.
2. First node contains ceil((m-1)/2) values.
3. Second node contains the remaining values.
4. Copy the smallest search key value from the second node to the parent node.

Case-2: Overflow in non-leaf node

1. Split the non leaf node into two nodes.
2. First node contains ceil(m/2)-1 values.
3. Move the smallest among remaining to the parent.
4. Second node contains the remaining keys.

B) Searching in B+ Tree

1. Start from the root node. Compare k with the keys at the root node [k1, k2, k3,.....km - 1].
2. If k < k1, go to the left child of the root node.
3. Else if k == k1, compare k2. If k < k2, k lies between k1 and k2. So, search in the left child of k2.
4. If k > k2, go for k3, k4,...km-1 as in steps 2 and 3.
5. Repeat the above steps until a leaf node is reached.
6. If k exists in the leaf node, return true else return false.

C) Deletion in B+ Tree

1. Find leaf L containing (key,pointer) entry to delete
2. Remove entry from L
   a. If L meets the "half full" criteria, then we're done.
   b. Otherwise, L has too few data entries.
3. If L's right sibling can spare an entry, then move smallest entry in right sibling to L
   a. Else,if L's left sibling can spare an entry then move largest entry in left sibling toL
   b. Else, merge L and a sibling
4. If merging, then recursively deletes the entry (pointing toL or sibling) from the parent.
5. Merge could propagate to root, decreasing height.

Class Diagram:

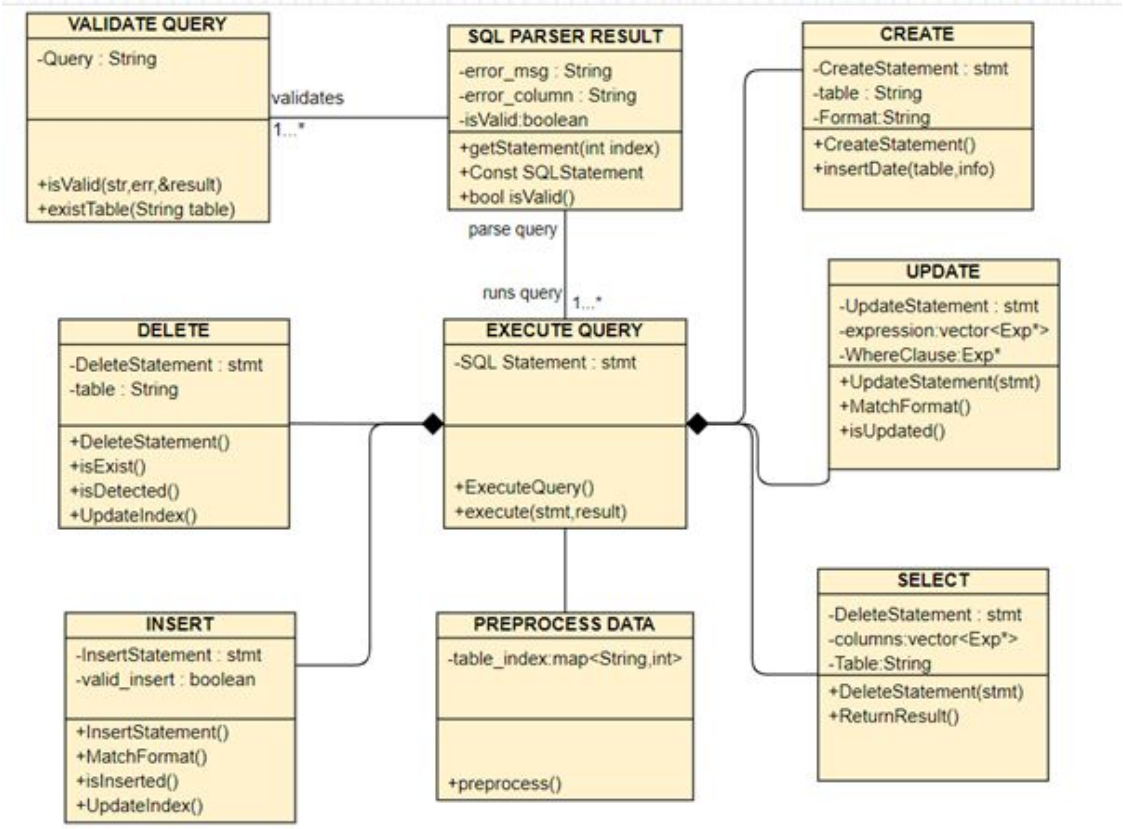

Fig 3.0

Data Flow Diagram:

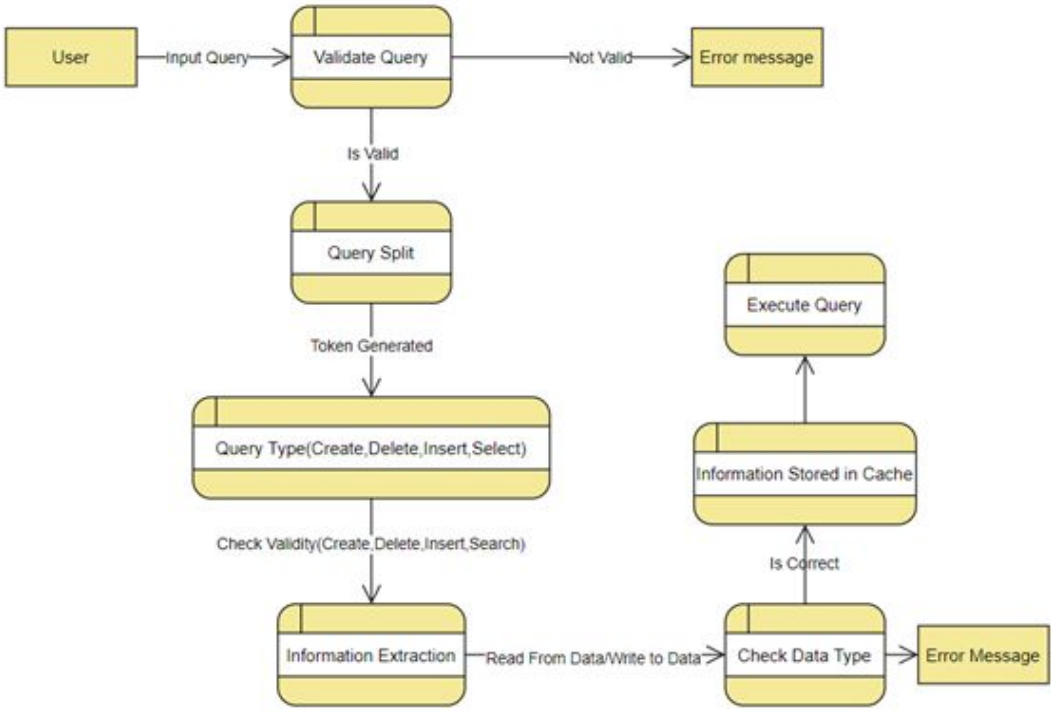

Fig 4.0

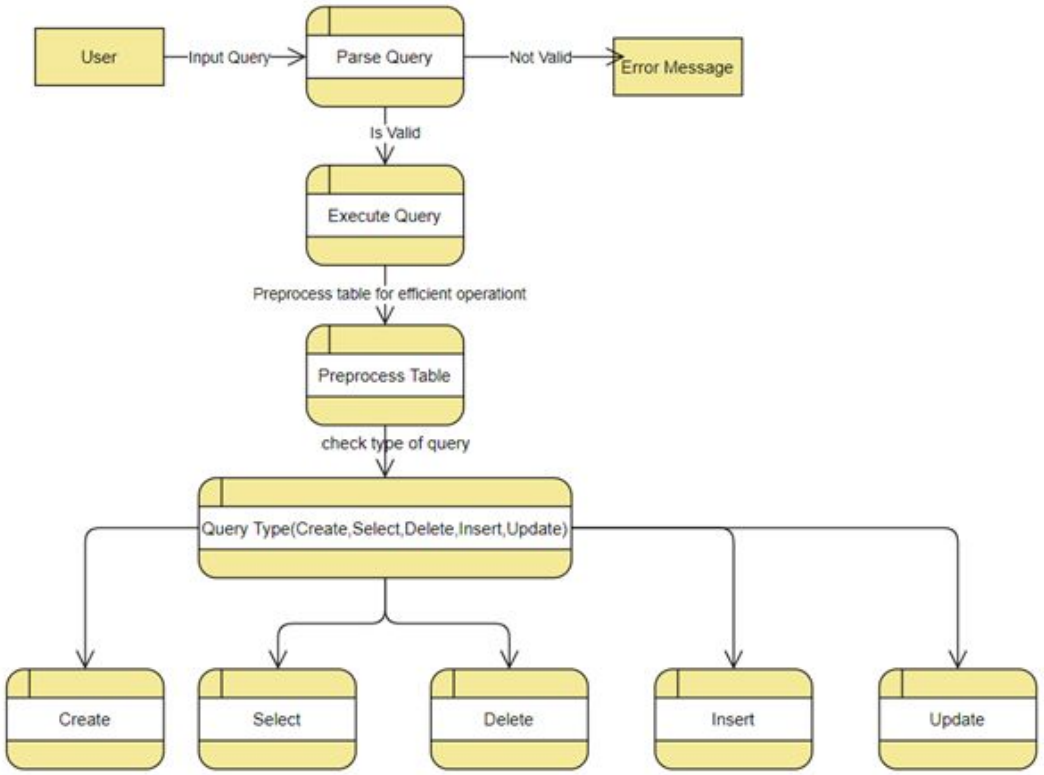
User
Input Query
Parse Query
Not Valid
Error Message
Is Valid
Execute Query
Preprocess table for efficient operationt
Preprocess Table
check type of query
Query Type(Create,Select,Delete,Insert,Update)
Create
Select
Delete
Insert
Update

Fig 4.1

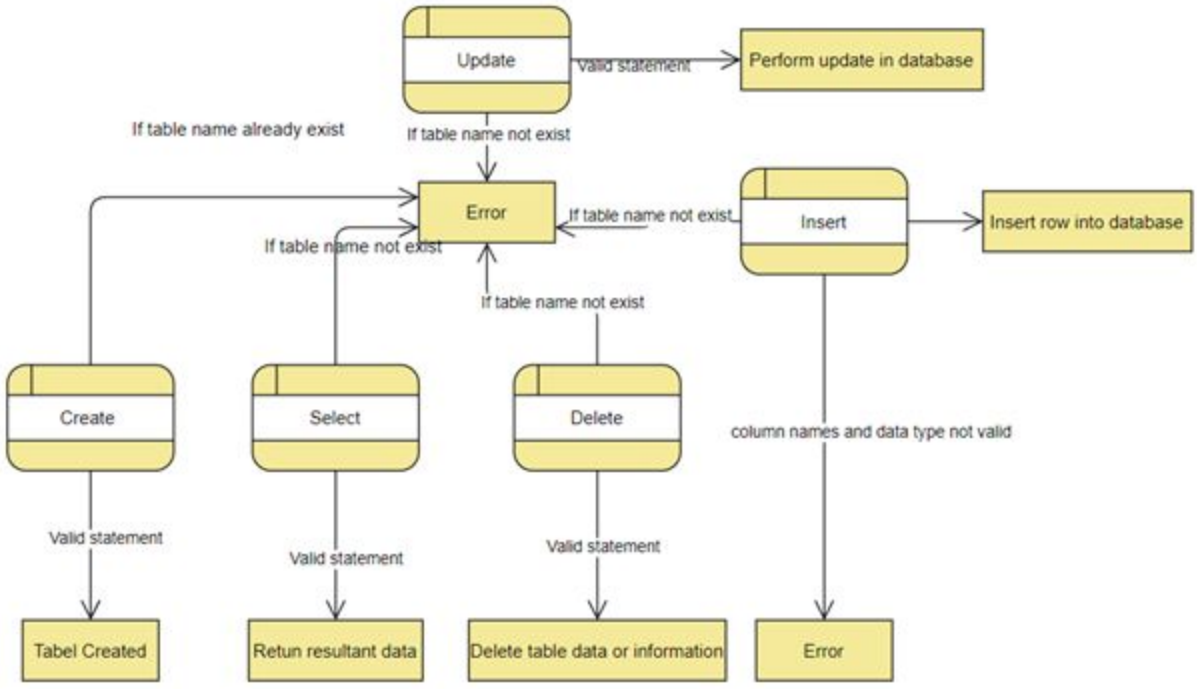
Update
Valid statement
Perform update in database
If table name already exist
If table name not exist
Error
If table name not exist
Insert
Insert row into database
If table name not exist
If table name not exist
Create
Select
Delete
column names and data type not valid
Valid statement
Valid statement
Valid statement
Tabel Created
Retun resultant data
Delete table data or information
Error

Fig 4.2

Use Case Diagram:

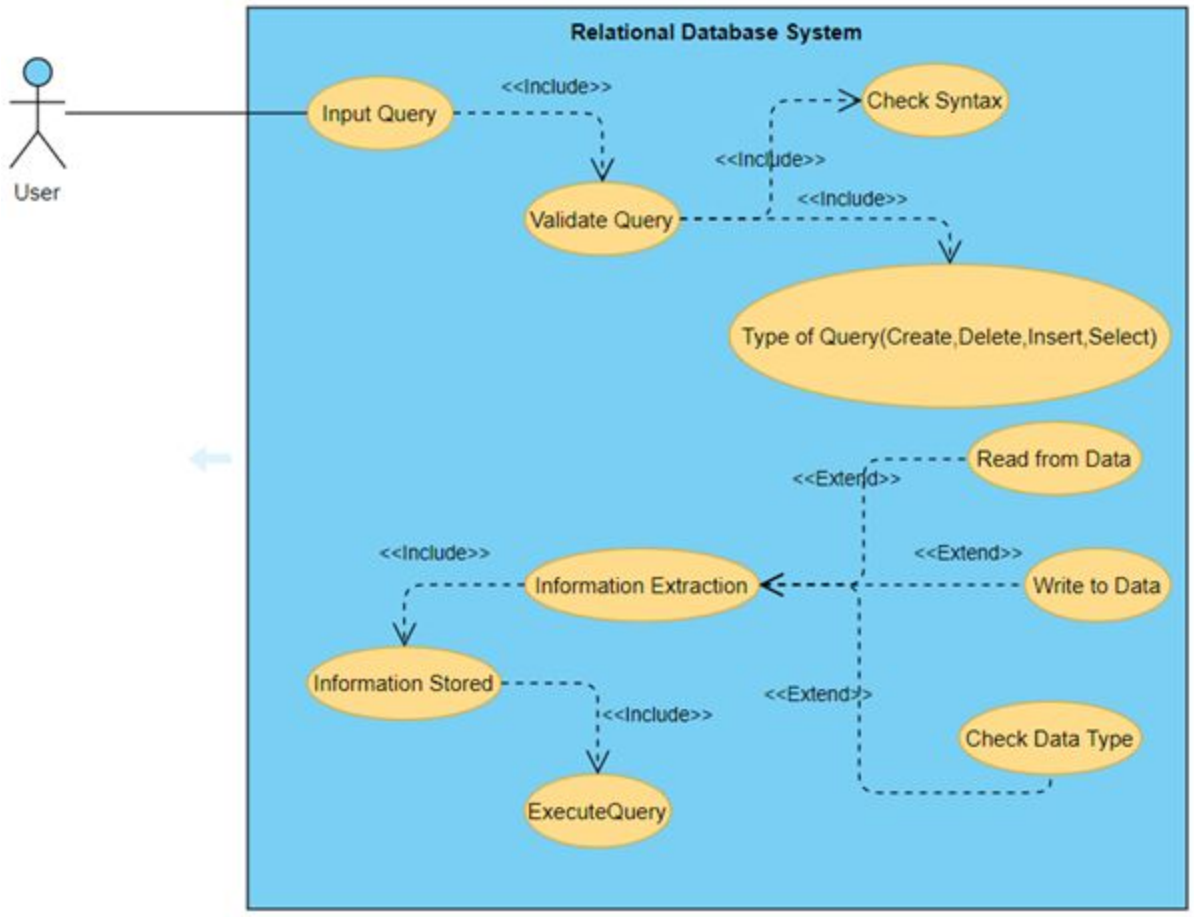

Fig 5.0

Output :

```
prakash@prakash-bs0xx:~/DBPlus-Relational-Database-With-Minimal-Functionality/src$ ./DBPlus
Runnig database DBplus>>
To exit from database type q or quit
For help type help

DBplus-: help
DBPlus: Relation Database with Minimal Functionality ,Open Source Project
GNU-GPL Licensed

./DBPlus        :       connect to global database
-q or -quit     :       quit from database
-help           :       help
-f or -function :       to enter in database shortcuts

    shortcuts:
    -t or -table:  total num of table
    -tl or -list:  list of tables
    -tv or -view:  to view all table structrues


@Support:
All SQL Queries Parsed And Syntax check successfully
Limited Queries Supported for database!!
(Insert,Create,Drop,Delete,Update,Select)

+--------------------------------------------------+
|@developer Contact                          UPES  |
|Prakash,Akshit,Gaurav                             |
|85prakas20172@gmail.com                           |
+--------------------------------------------------+
DBplus-:
```

Fig 6.0

```
prakash@prakash-bs0xx:~/DBPlus-Relational-Database-With-Minimal-Functionality/src$ ./DBPlus
Runnig database DBplus>>
To exit from database type q or quit
For help type help

DBplus-: select * from Customers ;
CustomerName |ContactName  |Address      |City         |PostalCode   |Country      |
-------------+-------------+-------------+-------------+-------------+-------------+
Cardinal     |NULL         |Skagen-21    |Stavanger    |4006         |Norway       |             |
prakash      |NULL         |rewa         |rewa         |486220       |India        |             |
akshit       |NULL         |rewa         |rewa         |486220       |India        |             |
gaurav       |golu         |Skagen-21    |Stavanger    |4006         |Norway       |             |
(4 rows)
DBplus-: select CustomerName, Address from Customers ;
CustomerName |Address      |
-------------+-------------+
Cardinal     |Skagen-21    |
prakash      |rewa         |
akshit       |rewa         |
gaurav       |Skagen-21    |
(4 rows)
DBplus-: select CustomerName from Customers;
CustomerName |
-------------+
Cardinal     |
prakash      |
akshit       |
gaurav       |
(4 rows)
DBplus-:
```

Fig 6.1

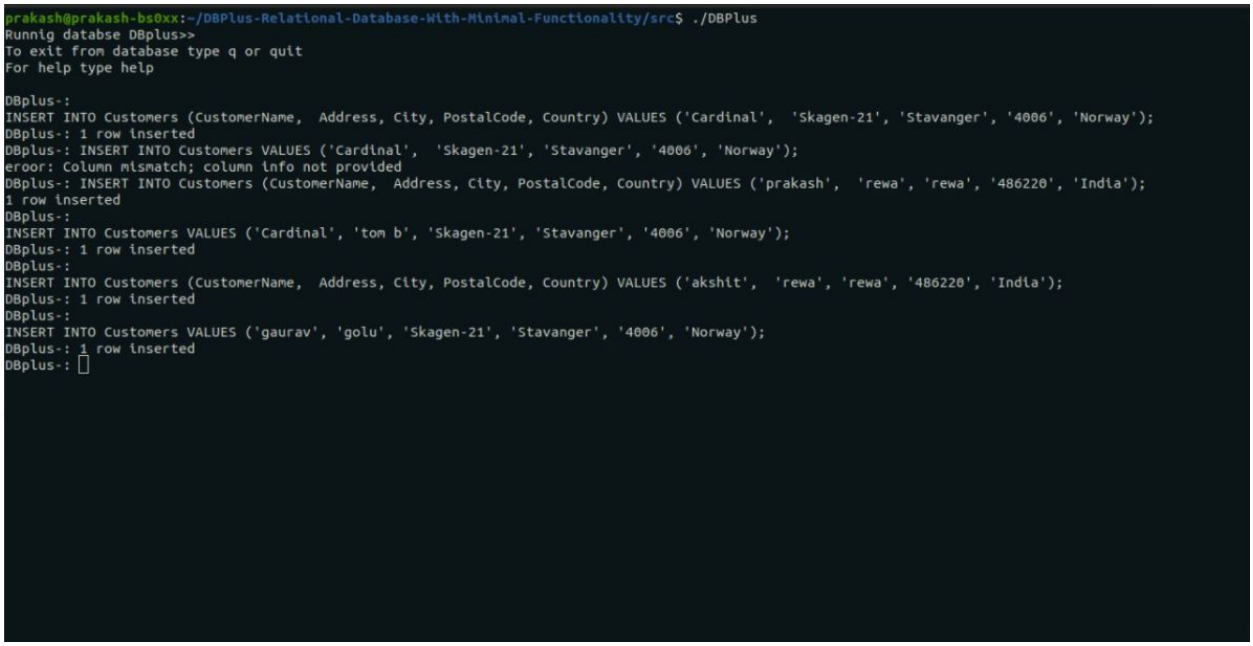
prakash@prakash-bs0xx:~/DBPlus-Relational-Database-With-Minimal-Functionality/src$ ./DBPlus
Runnig database DBplus>>
To exit from database type q or quit
For help type help
DBplus-:
INSERT INTO Customers (CustomerName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Skagen-21', 'Stavanger', '4006', 'Norway');
DBplus-: 1 row inserted
DBplus-: INSERT INTO Customers VALUES ('Cardinal', 'Skagen-21', 'Stavanger', '4006', 'Norway');
eroor: Column mismatch; column info not provided
DBplus-: INSERT INTO Customers (CustomerName, Address, City, PostalCode, Country) VALUES ('prakash', 'rewa', 'rewa', '486220', 'India');
1 row inserted
DBplus-:
INSERT INTO Customers VALUES ('Cardinal', 'tom b', 'Skagen-21', 'Stavanger', '4006', 'Norway');
DBplus-: 1 row inserted
DBplus-:
INSERT INTO Customers (CustomerName, Address, City, PostalCode, Country) VALUES ('akshit', 'rewa', 'rewa', '486220', 'India');
DBplus-: 1 row inserted
DBplus-:
INSERT INTO Customers VALUES ('gaurav', 'golu', 'Skagen-21', 'Stavanger', '4006', 'Norway');
DBplus-: 1 row inserted
DBplus-:

Fig 6.2

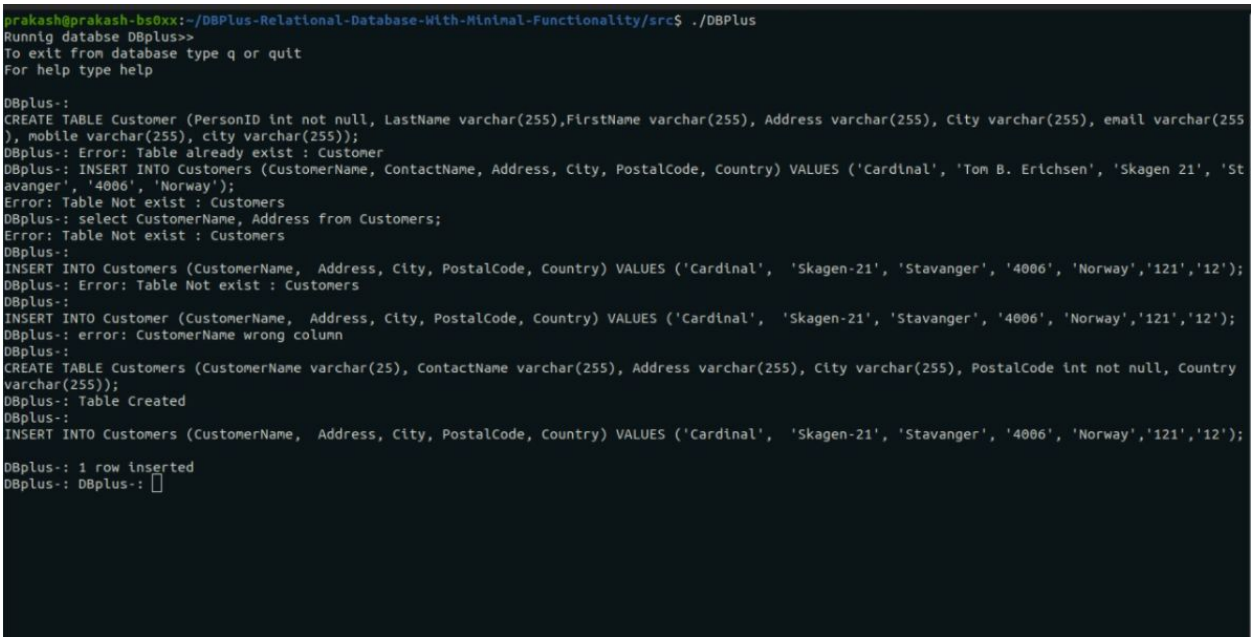
prakash@prakash-bs0xx:~/DBPlus-Relational-Database-With-Minimal-Functionality/src$ ./DBPlus
Runnig database DBplus>>
To exit from database type q or quit
For help type help
DBplus-:
CREATE TABLE Customer (PersonID int not null, LastName varchar(255),FirstName varchar(255), Address varchar(255), City varchar(255), email varchar(255), mobile varchar(255), city varchar(255));
DBplus-: Error: Table already exist : Customer
DBplus-: INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
Error: Table Not exist : Customers
DBplus-: select CustomerName, Address from Customers;
Error: Table Not exist : Customers
DBplus-:
INSERT INTO Customers (CustomerName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Skagen-21', 'Stavanger', '4006', 'Norway','121','12');
DBplus-: Error: Table Not exist : Customers
DBplus-:
INSERT INTO Customer (CustomerName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Skagen-21', 'Stavanger', '4006', 'Norway','121','12');
DBplus-: error: CustomerName wrong column
DBplus-:
CREATE TABLE Customers (CustomerName varchar(25), ContactName varchar(255), Address varchar(255), City varchar(255), PostalCode int not null, Country varchar(255));
DBplus-: Table Created
DBplus-:
INSERT INTO Customers (CustomerName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Skagen-21', 'Stavanger', '4006', 'Norway','121','12');
DBplus-: 1 row inserted
DBplus-: DBplus-:

Fig 6.3

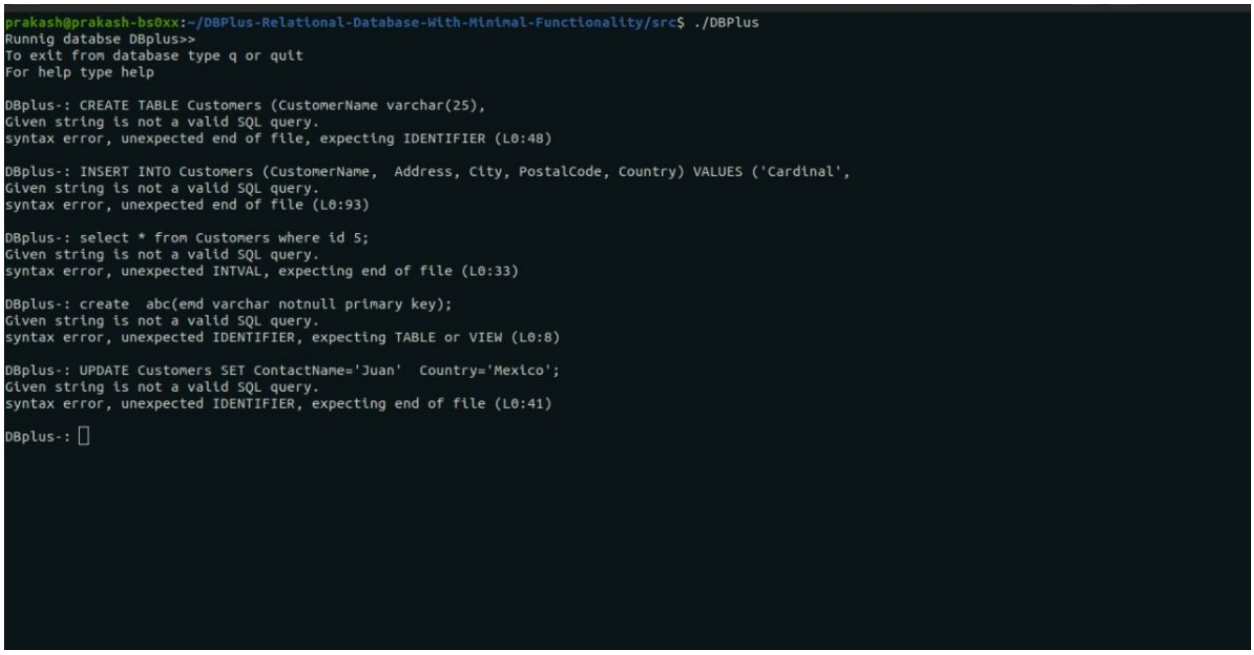

Fig 6.4

Commands:

Commands To Run The Project:

1. mkdir build
2. cd build
3. cmake ..
4. make
5. ./DBPlus

Project Dependencies:

1. Min version of CMAKE 3.0.1 or above cmake --version
2. Must be install Hyrise c++ Sql Parser
3 .g++ 11 or above

System Requirements:

Operating System	:	Ubuntu 18.04/Windows
Programming Language	:	C++
Running Environment	:	Terminal
Processor	:	Pentium IV

Disk Drive	:	Floppy or Hard Disk Drive
RAM	:	512 MB (min)

Schedule(Pert Chart):

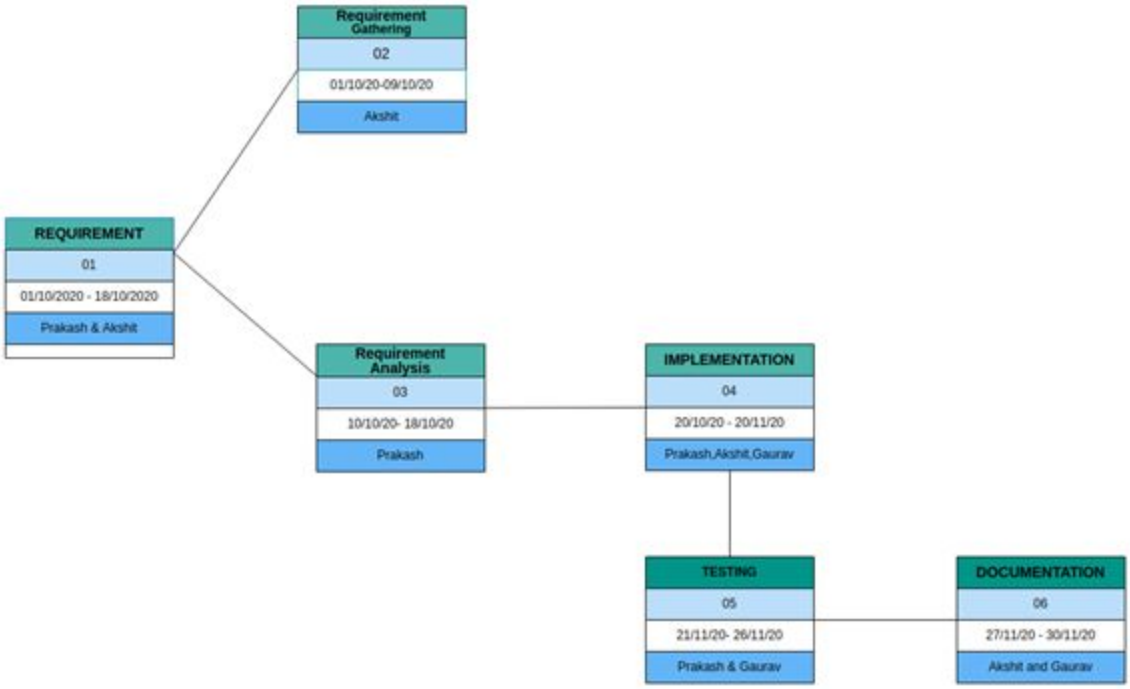

Fig 7.0

References:

[1]. Navathe, Ramez Elmasri, Shamkant B. (2010). *Fundamentals of database systems* (6th ed.). Upper Saddle River, N.J.: Pearson Education. pp. 652–660. ISBN.

[2]. Lightstone, S.; Teorey, T.; Nadeau, T. (2007). *Physical Database Design: the database professional's guide to exploiting indexes, views, storage, and more*. Morgan Kaufmann Press. ISBN .

[3]. S Praveen,U Chandra,*A Literature Review on Evolving Database (March 2017)* International Journal of Computer Applications.

[4]. https://en.wikipedia.org/wiki/Database_engine

[5]. SQLite3 Documentation, *sqile ofiicial doc*

Approved By

Signature	**Signature**
Mr Amit Singh	**Dr. Monit Kapoor**
Mentor	**Head of Department**