Jaypee Institute of Information Technology

Sector - 62, Noida

SDF Lab Project Report

Database Management System Using C++

Submitted to:
Dr. Ashish Mishra
Dr. Kapil Madan
Department of Computer Science

Submitted by:
Amolik Agarwal (2401030193)
Vansh Garg (2401030202)
Aarnya Jain (2401030209)

Letter of Transmittal

Date: April 30, 2025

To: Dr. Ashish Mishra and Dr. Kapil Madan
Department of Computer Science
Jaypee Institute of Information Technology
Sector-62, Noida

Subject: Submission of Project Report on "Database Management System Using C++"

Dear Sir,

We are pleased to submit our project report titled "Database Management System Using C++" undertaken as part of the SDF Lab course. This project explores the implementation of a comprehensive database management system built entirely using C++, demonstrating the application of object-oriented programming principles and advanced C++ features.

The report details our design methodology, the implementation of key database features, the C++ concepts utilized, and a thorough evaluation of the system's performance. We have focused on creating a robust, scalable solution that showcases the power of C++ for backend development while maintaining a user-friendly interface.

Throughout this project, we have gained valuable insights into database design principles, memory management, file handling, and the application of object-oriented programming in solving real-world problems. The system we've developed demonstrates practical applications of the theoretical concepts covered in our course.

We would appreciate your feedback on our work and are available to discuss any aspects of the project you find interesting or that warrant further exploration.

Sincerely,

Amolik Agarwal	Vansh Garg	Aarnya Jain
2401030193	2401030202	2401030209

Contents

1 Abstract

This project presents the design and implementation of a comprehensive Database Management System (DBMS) developed entirely in C++. The system leverages object-oriented programming principles to create a robust framework for data storage, retrieval, manipulation, and management. Our implementation provides essential DBMS functionalities including table creation, data insertion, querying, updating, and deletion operations, along with basic transaction management and data persistence.

The project demonstrates practical applications of advanced C++ features such as classes, inheritance, polymorphism, templates, file handling, and exception handling. By focusing on memory efficiency and performance optimization, we have created a lightweight yet powerful database solution suitable for educational purposes and small-scale applications.

Throughout the development process, we confronted challenges related to data consistency, concurrent access, and efficient indexing structures, applying theoretical database concepts while working within the constraints of a procedural language. The resulting system showcases how C++ can be effectively utilized to implement fundamental database concepts, providing valuable insights into both database architecture and advanced programming techniques.

2 Introduction

2.1 Background and Motivation

Database Management Systems form the backbone of modern information technology infrastructure, enabling the storage, organization, and retrieval of vast amounts of data. While industrial-strength DBMS like Oracle, SQL Server, and PostgreSQL are implemented using complex technologies and optimized algorithms, developing a simplified DBMS from scratch provides invaluable insights into the underlying principles of database design and implementation.

This project was motivated by the desire to apply our C++ programming knowledge to a complex real-world problem, gaining deeper understanding of both database concepts and advanced C++ features. By building a DBMS from the ground up, we aimed to explore the challenges of data organization, memory management, file operations, and query processing while implementing practical solutions using object-oriented design principles.

2.2 Problem Statement

Modern applications require efficient data management solutions that can handle persistent storage, provide data integrity, and support complex queries. While professional DBMS solutions exist, there is educational value in developing a simplified system that demonstrates core functionality. This project addresses the challenge of creating a functional, object-oriented database management system in C++ that incorporates fundamental DBMS features while maintaining code readability and extensibility.

The key problems we aimed to solve include:

- Designing an intuitive, object-oriented architecture for database operations

- Implementing efficient data storage and retrieval mechanisms
- Providing a query processing system that supports basic SQL-like operations
- Ensuring data persistence through file system integration
- Handling concurrent access and maintaining data integrity
- Creating a user-friendly interface for database administration

2.3 Objectives

The primary objectives of this project were:

1. To design and implement a functional database management system using C++ that supports:
    - Creation and management of database tables with defined schemas
    - Basic CRUD operations (Create, Read, Update, Delete)
    - Data persistence through file-based storage
    - Simple query capabilities with filtering and sorting
    - Basic transaction management with commit and rollback functionality
2. To apply and demonstrate advanced C++ programming concepts including:
    - Object-oriented design principles (encapsulation, inheritance, polymorphism)
    - Template metaprogramming for type-safe operations
    - Smart pointers and memory management
    - Exception handling for robust error management
    - Standard Template Library (STL) data structures and algorithms
3. To create a well-structured, maintainable, and extensible codebase that:
    - Follows good software engineering practices
    - Is well-documented and easy to understand
    - Can be extended with additional features in the future
    - Demonstrates practical application of theoretical database concepts

2.4 Significance and Applications

The significance of this project extends beyond academic exercise:

- **Educational value:** The implementation provides a concrete understanding of database internals and C++ capabilities.
- **Practical utility:** The resulting system can be used for small-scale applications and prototypes.

- **Skill development:** The project enhances programming skills in system design, memory management, and algorithm implementation.
- **Foundation for advanced features:** The modular design provides a base for implementing more complex database concepts like indexing, joins, and advanced query optimization.

By developing this simplified DBMS, we contribute to the understanding of both database principles and C++ programming capabilities, creating a bridge between theoretical concepts and practical implementation.

2.5 C++ Features Utilized

- **Classes and Objects**: Encapsulation of database entities and operations through custom classes
- **File I/O Operations**: Using `fstream` for persistent data storage and retrieval
- **STL Containers**: Leveraging `vector` and `map` for efficient data management
- **Exception Handling**: Implementing try-catch blocks for robust error management
- **Templates**: Creating generic functions for handling different data types
- **Operator Overloading**: Customizing operations for database objects
- **Namespaces**: Organizing code into logical compartments

3 Implementation Details

3.1 Database and Table Management

```cpp
# include <iostream>
# include <filesystem>
# include <fstream>
# include <sstream>
# include <variant>
# include <string>
# include <vector>
# include <iomanip>
# include <algorithm>
# include <unordered_set>


#define GREEN   "\033[32m"
#define RESET   "\033[0m"

using namespace std;

void write_table(vector<vector<string>> data, string owner_database,string table_name) {
    string path = "./data/"+owner_database+"/"+table_name+".csv";
    ofstream file(path);

    if (!file.is_open()) {
        cerr << "Error: Could not open file " << path << " for writing." << endl;
        return;
    }

    for (const auto& row : data) {
        for (size_t i = 0; i < row.size(); ++i) {
            file << row[i];
            if (i != row.size() - 1)
                file << ",";
        }
        file << "\n";
    }

    file.close();
}

class tables{
private:
```

```cpp
private:
    vector<vector<string>> data;
    string name;
public:

    vector<vector<string>> read(string owner_database,string table_name);
    void create_table(string owner_database,string table_name);
    void delete_table(string owner_database,string table_name);
    void addcol(vector<string> col);
    void addrow(vector<string> rows,vector<vector<string>> data,string owner_database,string table_name);
    vector<vector<string>> sort_asc(vector<vector<string>> data , string col);
    vector<vector<string>> sort_desc(vector<vector<string>> data , string col);
    vector<vector<string>> where(vector<vector<string>> &vec, string column, string op, string val );
    vector<vector<string>> select_by_col(vector<vector<string>> vec,unordered_set<string> col);
    void update_table(string owner_database,string table_name, string c1name,string c1value, string c2name,string c2value,string op);
    vector<vector<string>> refine(vector<vector<string>> data);

};

void tables::create_table(string owner_database,string table_name){
    string path = "./data/"+owner_database+"/"+table_name+".csv";
    ofstream table(path);
}

void tables::delete_table(string owner_database, string table_name) {
    string path = "./data/" + owner_database + "/" + table_name + ".csv";
    if (filesystem::exists(path)) {
        filesystem::remove(path);
        cout << "Table '" << table_name << "' deleted successfully.\n";
    } else {
        cout << "Table '" << table_name << "' does not exist.\n";
    }
}

vector<vector<string>> tables::read(string owner_database,string table_name){
    data.clear();
    string path = "./data/"+owner_database+"/"+table_name+".csv";
    ifstream file(path);
    string line;

      while(getline(file,line)){
```

```
        while(getline(file,line)){
         stringstream ss(line);
         string cell;
         vector<string> row;
       while (getline(ss, cell, ',')) {
            row.push_back(cell);
         }
         data.push_back(row);
      }
     return data;
  }
  void displaytable(vector<vector<string>> data) {
     if (data.empty() || data[0].empty()) return;
     int rows = data.size();
     int cols = data[0].size();
     int cellWidth = 0;
     for (int j = 0; j < cols; j++) {
         int len = data[0][j].length();
         if (len > cellWidth) cellWidth = len;
     }
     cellWidth += 6;
     for (int j = 0; j < cellWidth * cols; j++) cout <<GREEN<< "=" <<RESET;
     cout << endl;
     for (int j = 0; j < cols; j++) {
         cout << setw(cellWidth - 2) << data[0][j] << GREEN<<" |"<<RESET;
     }
     cout << endl;
     for (int j = 0; j < cellWidth * cols; j++) cout <<GREEN<< "="<<RESET;
     cout << endl;
     for (int i = 1; i < rows; i++) {
         for (int j = 0; j < cols; j++) {
             cout << setw(cellWidth - 2) << data[i][j] << GREEN<<" |"<<RESET;
         }
         cout << endl;
         for (int j = 0; j < cellWidth * cols; j++) cout <<GREEN <<"-"<<RESET;
         cout << endl ;
     }
  }
```

```
void tables::addcol(vector<string> col){
       int index = data[0].size()-1;
       for(int i=0;i<data.size();i++){
           data[i].push_back(col[i]);
       }
}

vector<vector<string>> tables::refine(vector<vector<string>> data){
    vector<vector<string>> refined_data;
    int max = -1;
    for(int i=0;i<data.size();i++){
        if(data[i].size()>max){
            max = data[i].size();
        }
    }

    for(int i=0;i<data.size();i++){
        while(data[i].size() != max){
            data[i].push_back("-");
        }
    }

    refined_data = data;
    return refined_data;
}

void tables::addrow(vector<string> rows,vector<vector<string>> data,string owner_database,string table_name){
    int max = -1;
    for(int i=0;i<data.size();i++){
        if(data[i].size()>max){
            max = data[i].size();
        }
    }

    if(rows[0].size()>max){
        cout << "Values exceed the no. of columns ..." <<endl;
        return;
    }

    for(int i=0;i<rows.size();++i){
        data.push_back(rows);
    }

    data = refine(data);
    write_table(data,owner_database,table_name);
    return;

}
```

```cpp
vector<vector<string>> tables::sort_asc(vector<vector<string>> vec, string column){
    int idx=-1;
    for(int i=0;i<vec[0].size();i++){
        if(vec[0][i]==column){
            idx=i;
            break;
        }
    }
    for(int i=1;i<vec.size();i++){
        for(int j=1;j<vec.size()-i;j++){
            if(vec[j][idx]>vec[j+1][idx]){
                swap(vec[j],vec[j+1]);
            }
        }
    }
    return vec;
}
vector<vector<string>> tables::sort_desc(vector<vector<string>> vec, string column){
    int idx=-1;
    for(int i=0;i<vec[0].size();i++){
        if(vec[0][i]==column){
            idx=i;
            break;
        }
    }
    for(int i=1;i<vec.size();i++){
        for(int j=1;j<vec.size()-i;j++){
            if(vec[j][idx]<vec[j+1][idx]){
                swap(vec[j],vec[j+1]);
            }
        }
    }
    return vec;
}
vector<vector<string>> tables::where(vector<vector<string>> &vec, string column, string op, string val ){
    vector<vector<string>> ans;
    ans.push_back(vec[0]);
    int idx=-1;
    for(int i=0;i<vec[0].size();i++){
        if(vec[0][i]==column){
            idx=i;break;
        }
    }
    if(idx==-1){
        return ans;
    }
    if(op=="="){
        for(int i=1;i<vec.size();i++){
            if(vec[i][idx]==val){
```

```cpp
                if(vec[i][idx]==val){
                    ans.push_back(vec[i]);
                }
            }
        }
        else if(op=="<"){
            for(int i=1;i<vec.size();i++){
                if(vec[i][idx]<val){
                    ans.push_back(vec[i]);
                }
            }
        }
        else if(op=="<="){
            for(int i=1;i<vec.size();i++){
                if(vec[i][idx]<=val){
                    ans.push_back(vec[i]);
                }
            }
        }
        else if(op==">"){
            for(int i=1;i<vec.size();i++){
                if(vec[i][idx]>val){
                    ans.push_back(vec[i]);
                }
            }
        }
        else if(op==">="){
            for(int i=1;i<vec.size();i++){
                if(vec[i][idx]>=val){
                    ans.push_back(vec[i]);
                }
            }
        }
        else if(op=="!="){
            for(int i=1;i<vec.size();i++){
                if(vec[i][idx]!=val){
                    ans.push_back(vec[i]);
                }
            }
        }
        else{
            return ans;
        }
        return ans;
    }
```

```cpp
vector<vector<string>> tables::select_by_col(vector<vector<string>> vec,
    unordered_set<string> col){
    vector<vector<string>>ans;
    for(int i=0;i<vec.size();i++){
        vector<string>temp;
        for(int j=0;j<vec[i].size();j++){
            if(col.find(vec[0][j])!=col.end()){
                temp.push_back(vec[i][j]);
            }
        }
        ans.push_back(temp);
    }
    return ans;
}

void tables::update_table(string owner_database,
    string table_name, string c1name,string c1value,
    string c2name,string c2value,string op){
    tables t;
    vector<vector<string>> vec = t.read(owner_database,table_name);
    int c1idx=-1;
    int c2idx=-1;
    for(int i=0;i<vec[0].size();i++){
        if(vec[0][i]==c1name){
            c1idx=i;
        }
        if(vec[0][i]==c2name){
            c2idx=i;
        }
    }

    if (c1idx == -1 || (op != "" && c2idx == -1)) {
        cout << "Not a valid column name.\n";
        return;
    }

    else{
        for(int i=1;i<vec.size();i++){
            if(op=="="){
                if(vec[i][c2idx] ==c2value){
                    vec[i][c1idx]=c1value;
                }
            }
            if(op==">"){
                if(vec[i][c2idx] >c2value){
                    vec[i][c1idx]=c1value;
                }
            }
```

```cpp
                if(vec[i][c2idx] >=c2value){
                    vec[i][c1idx]=c1value;
                }
            }
            if(op=="<"){
                if(vec[i][c2idx] <c2value){
                    vec[i][c1idx]=c1value;
                }
            }
            if(op=="<="){
                if(vec[i][c2idx] <=c2value){
                    vec[i][c1idx]=c1value;
                }
            }
            if(op=="!="){
                if(vec[i][c2idx] !=c2value){
                    vec[i][c1idx]=c1value;
                }
            }
            if (op == ""){
                vec[i][c1idx] = c1value;
            }
        }
    }

    write_table(vec,owner_database,table_name);
    cout << "Update successful on matching rows." << endl;
    return;
}
```

3.1.1 Main Function

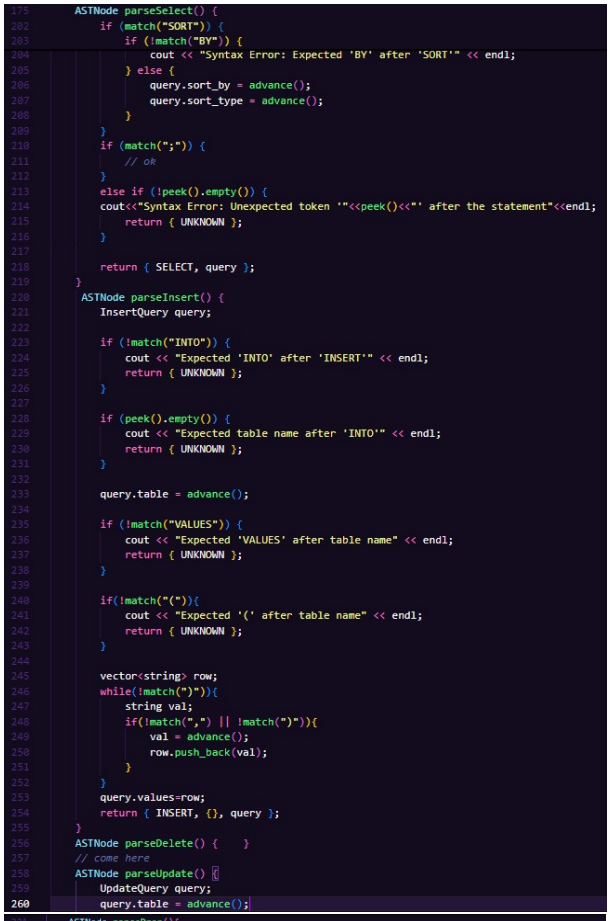
```cpp
#include <iostream>
#include <string.h>
#include <vector>
#include <stdlib.h>
#include "parsing.h"
#include "lexer.h"
using namespace std;
#define RESET   "\033[0m"
#define RED     "\033[31m"
#define GREEN   "\033[32m"
#define YELLOW  "\033[33m"
#define BLUE    "\033[34m"
#define BOLD    "\033[1m"
bool check_general(string &cmd){
    if(!(strcasecmp(cmd.c_str(),"CLEAR")) || !(strcasecmp(cmd.c_str(),"CLS"))){
        system("CLS");
        return true;
    }
    if(!(strcasecmp(cmd.c_str(),"SHOW DATABASES"))){
        // THESE ONES TAKEN FROM THE fetch.h HEADER FILE LOADED IN parsing.h
        show_tree(fetch_structure());
        return true; }
    return false;}
int main(){
    while(1)
    {
       string currentDB = getCurrentDatabase();
        cout<<endl;
        cout << BOLD << RED << currentDB << " " << BOLD << GREEN << "> " << RESET;

        string cmd;
        getline(cin, cmd);
        cmd = normalizeSpaces(cmd);

        if (cmd.empty()) {
            continue;
        }
        if(!strcasecmp(cmd.c_str(),"EXIT")){
            cout << "Exiting the running instance ... \n\n";
            break;
        }
        if(check_general(cmd)) continue;
        lex l;
        vector<string> tokens = l.tokenize(cmd);
        Parser p(tokens);
        ASTNode ast = p.parse();
        ExecutionEngine engine;
            engine.execute(ast);
        }
}
```

3.1.2 Parsing Logic for SQL Statements

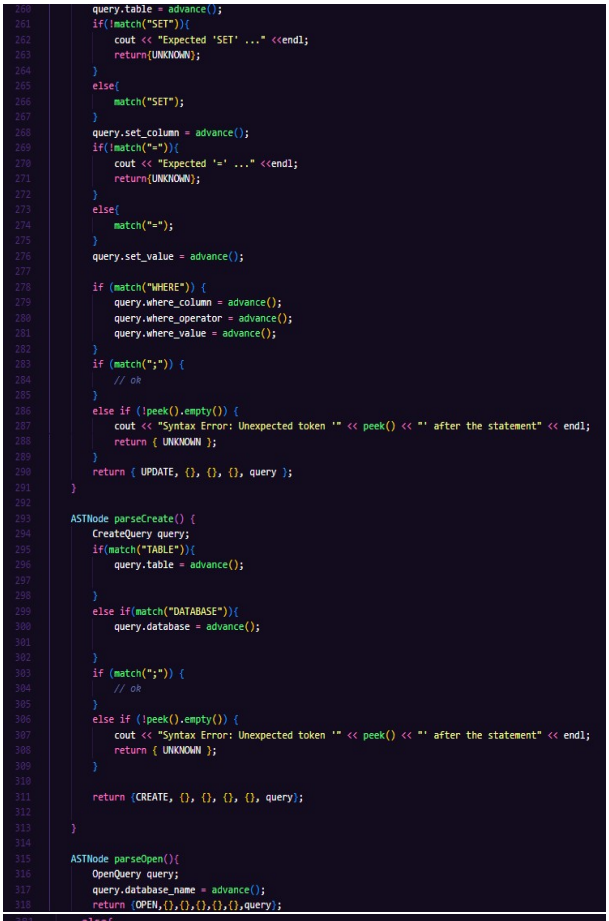
```cpp
#include <stdio.h>
#include <iostream>
#include <vector>
#include <string>
#include <unordered_set>
#include <unordered_map>
#include <algorithm>
#include <filesystem>
#include <cctype>
#include <sstream>
#include "database.h"
#include "fetch.h"

string normalizeSpaces(const std::string& input) {
    stringstream ss(input);
    string word, result;

    while (ss >> word) {
        if (!result.empty())
            result += " ";
        result += word;
    }

    return result;
}

using namespace std;

enum QueryType {
    SELECT,
    INSERT,
    DELETE,
    UPDATE,
    UNKNOWN,
    CREATE,
    DROP,
    OPEN,
};

/// THE SORT ONE IS OF THE LEAST PRIORITY
/// SHOW , CLEAR , EXIT ----> ARE THE GENERAL COMMANDS

struct SelectQuery {
    string all_operator;
    unordered_set<string> columns;
    string table;
    string where_column;
    string where_operator;
    string where_value;
    string sort_by;
    string sort_type;
};

struct DropQuery{
    string table;
    string database;
};

struct DeleteQuery{
    string table;
    vector<string> rows;
    vector<string> cols;
};

struct InsertQuery {
    string table;
    vector<string> values;
};


struct CreateQuery{
    string table;
    string database;
};

struct UpdateQuery {
    string table;
    string set_column;
    string set_value;
    string where_column;
    string where_operator;
    string where_value;
};

struct OpenQuery {
    string database_name;
};

struct ASTNode {
    QueryType type;
    SelectQuery selectQuery;
    InsertQuery insertQuery;
    DeleteQuery deleteQuery;
    UpdateQuery updateQuery;
    CreateQuery createQuery;
    OpenQuery openQuery;
    DropQuery dropQuery;
};
```

```cpp
class DatabaseManager {
    private:
        static DatabaseManager* instance;
        string current_database;

        DatabaseManager() {}

    public:
        static DatabaseManager* getInstance() {
            if (!instance) {
                instance = new DatabaseManager();
            }
            return instance;
        }

        void setCurrentDatabase(const string& dbName) {
            current_database = dbName;
        }

        string getCurrentDatabase() const {
            return current_database;
        }
    };

    DatabaseManager* DatabaseManager::instance = nullptr;

    inline string getCurrentDatabase() {

        string path = "./data/" +
        DatabaseManager::getInstance()->getCurrentDatabase();
        if (filesystem::exists(path)){
            return DatabaseManager::getInstance()->getCurrentDatabase();
        }

        return "";
    }


class Parser {
    vector<string> tokens;
    int current = 0;

public:
    Parser(const vector<string>& toks) : tokens(toks) {}

    string peek() {
        return current < tokens.size() ? tokens[current] : "";
    }
```

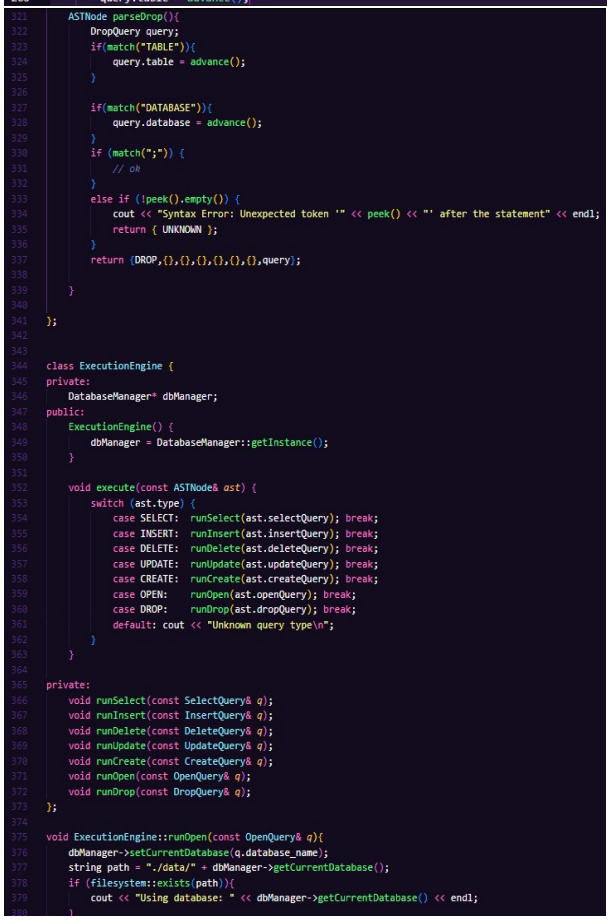
```cpp
    bool match(const string& expected) {
        string s = peek();
        transform(s.begin(), s.end(), s.begin(),::toupper);

        if (s == expected) {
            advance();
            return true;
        }
        return false;
    }
    ASTNode parse() {
        if (match("SELECT")) return parseSelect();
        if (match("INSERT")) return parseInsert();
        if (match("DELETE")) return parseDelete();
        if (match("UPDATE")) return parseUpdate();
        if (match("CREATE")) return parseCreate();
        if (match("OPEN")) return parseOpen();
        if (match("DROP")) return parseDrop();
        return { UNKNOWN };
    }
private:
    ASTNode parseSelect() {
        SelectQuery query;
        if(peek().empty()){
            return{UNKNOWN};
        }
        if (peek() == "*") {
            query.all_operator = advance();
            query.columns = {};
            if (!match("FROM")) {
                cout << "Syntax Error: Expected 'FROM' after '*'" << endl;
                return { UNKNOWN };
            }
            query.table = advance();
        }
        else {
            while (!match("FROM")) {
                string col = advance();
                if (col != ",") query.columns.insert(col);
            }

            query.table = advance();
        }
        if (match("WHERE")) {
            query.where_column = advance();
            query.where_operator = advance();
            query.where_value = advance();
        }
```



```
    ASTNode parseSelect() {
        if (match("SORT")) {
            if (!match("BY")) {
                cout << "Syntax Error: Expected 'BY' after 'SORT'" << endl;
            } else {
                query.sort_by = advance();
                query.sort_type = advance();
            }
        }
        if (match(";")) {
            // ok
        }
        else if (!peek().empty()) {
        cout<<"Syntax Error: Unexpected token '"<<peek()<<"' after the statement"<<endl;
            return { UNKNOWN };
        }

        return { SELECT, query };
    }
     ASTNode parseInsert() {
        InsertQuery query;

        if (!match("INTO")) {
            cout << "Expected 'INTO' after 'INSERT'" << endl;
            return { UNKNOWN };
        }

        if (peek().empty()) {
            cout << "Expected table name after 'INTO'" << endl;
            return { UNKNOWN };
        }

        query.table = advance();

        if (!match("VALUES")) {
            cout << "Expected 'VALUES' after table name" << endl;
            return { UNKNOWN };
        }

        if(!match("(")){
            cout << "Expected '(' after table name" << endl;
            return { UNKNOWN };
        }

        vector<string> row;
        while(!match(")")){
            string val;
            if(!match(",") || !match(")")){
                val = advance();
                row.push_back(val);
            }
        }
        query.values=row;
        return { INSERT, {}, query };
    }
    ASTNode parseDelete() {    }
    // come here
    ASTNode parseUpdate() {
        UpdateQuery query;
        query.table = advance();
```



```
        query.table = advance();
        if(!match("SET")){
            cout << "Expected 'SET' ..." <<endl;
            return{UNKNOWN};
        }
        else{
            match("SET");
        }
        query.set_column = advance();
        if(!match("=")){
            cout << "Expected '=' ..." <<endl;
            return{UNKNOWN};
        }
        else{
            match("=");
        }
        query.set_value = advance();

        if (match("WHERE")) {
            query.where_column = advance();
            query.where_operator = advance();
            query.where_value = advance();
        }
        if (match(";")) {
            // ok
        }
        else if (!peek().empty()) {
            cout << "Syntax Error: Unexpected token '" << peek() << "' after the statement" << endl;
            return { UNKNOWN };
        }
        return { UPDATE, {}, {}, {}, query };
    }

    ASTNode parseCreate() {
        CreateQuery query;
        if(match("TABLE")){
            query.table = advance();

        }
        else if(match("DATABASE")){
            query.database = advance();

        }
        if (match(";")) {
            // ok
        }
        else if (!peek().empty()) {
            cout << "Syntax Error: Unexpected token '" << peek() << "' after the statement" << endl;
            return { UNKNOWN };
        }

        return {CREATE, {}, {}, {}, {}, query};

    }

    ASTNode parseOpen(){
        OpenQuery query;
        query.database_name = advance();
        return {OPEN,{},{},{},{},{},query};
```



```
    ASTNode parseDrop(){
        DropQuery query;
        if(match("TABLE")){
            query.table = advance();
        }

        if(match("DATABASE")){
            query.database = advance();
        }
        if (match(";")) {
            // ok
        }
        else if (!peek().empty()) {
            cout << "Syntax Error: Unexpected token '" << peek() << "' after the statement" << endl;
            return { UNKNOWN };
        }
        return {DROP,{},{},{},{},{},{},query};


    }

};


class ExecutionEngine {
private:
    DatabaseManager* dbManager;
public:
    ExecutionEngine() {
        dbManager = DatabaseManager::getInstance();
    }

    void execute(const ASTNode& ast) {
        switch (ast.type) {
            case SELECT:  runSelect(ast.selectQuery); break;
            case INSERT:  runInsert(ast.insertQuery); break;
            case DELETE:  runDelete(ast.deleteQuery); break;
            case UPDATE:  runUpdate(ast.updateQuery); break;
            case CREATE:  runCreate(ast.createQuery); break;
            case OPEN:    runOpen(ast.openQuery); break;
            case DROP:    runDrop(ast.dropQuery); break;
            default: cout << "Unknown query type\n";
        }
    }

private:
    void runSelect(const SelectQuery& q);
    void runInsert(const InsertQuery& q);
    void runDelete(const DeleteQuery& q);
    void runUpdate(const UpdateQuery& q);
    void runCreate(const CreateQuery& q);
    void runOpen(const OpenQuery& q);
    void runDrop(const DropQuery& q);
};

void ExecutionEngine::runOpen(const OpenQuery& q){
    dbManager->setCurrentDatabase(q.database_name);
    string path = "./data/" + dbManager->getCurrentDatabase();
    if (filesystem::exists(path)){
        cout << "Using database: " << dbManager->getCurrentDatabase() << endl;
    }
```

```
    else{
        cout << "Database not found" << endl;
        dbManager->setCurrentDatabase("");
    }
}

void ExecutionEngine::runDrop(const DropQuery& q){
    tables t;
    databases d;
    if(!q.database.empty()){
        d.delete_database(q.database);
        remove_database(fetch_structure(),q.database);
    }

    if(!q.table.empty()){
        if(dbManager->getCurrentDatabase() == ""){
            cout << "Please Open a Database to drop a Table ..." << endl;
        }
        else{
            t.delete_table(dbManager->getCurrentDatabase(),q.table);
            remove_table(fetch_structure(),dbManager->getCurrentDatabase(),q.table);
        }

    }

    else if(q.database.empty() && q.table.empty()){
        cout << "Specify what to delete ... " << endl;
    }
}

void ExecutionEngine::runInsert(const InsertQuery& q) {
    tables t;
    if (dbManager->getCurrentDatabase().empty()) {
        cout << "Please open a database to insert into a table ..." << endl;
        return;
    }

    vector<vector<string>> data = t.read(dbManager->getCurrentDatabase(),q.table);

    if (data.empty()) {
        cout << "Table is empty or does not exist ..." << endl;
        return;
    }

    t.addrow(q.values,data,dbManager->getCurrentDatabase(),q.table);
    cout << "Data inserted successfully ..." << endl;



}

void ExecutionEngine::runSelect(const SelectQuery& q) {
    tables t;

    // OPEN
    if (dbManager->getCurrentDatabase().empty()) {
        cout << "Please open a database to select a table ..." << endl;
        return;
    }
```

```
    // CHECK TABLE EXIST
    vector<vector<string>> data = t.read(dbManager->getCurrentDatabase(), q.table);
    if (data.empty()) {
        cout << "Table is empty or does not exist ..." << endl;
        return;
    }

    // CHECK WHERE
    if (!q.where_column.empty() && !q.where_operator.empty() && !q.where_value.empty()) {
        data = t.where(data, q.where_column, q.where_operator, q.where_value);
        if (data.size() <= 1) {
            // nothing to show
            cout << "No rows match the WHERE condition ..." << endl;
            return;
        }
    }

    // CHECK SORT BY
    if (!q.sort_by.empty() && !q.sort_type.empty()) {
        if (q.sort_type == "ASC") {
            data = t.sort_asc(data, q.sort_by);
        } else if (q.sort_type == "DESC") {
            data = t.sort_desc(data, q.sort_by);
        } else {
            cout << "Invalid sort type. Use ASC or DESC." << endl;
            return;
        }
    }

    // CHECK *
    if (q.all_operator == "*") {
        displaytable(data);
    } else if (!q.columns.empty()) {
        data = t.select_by_col(data, q.columns);
        displaytable(data);
    } else {
        cout << "Invalid Query. No columns specified and no '*' operator." << endl;
    }
}


void ExecutionEngine::runDelete(const DeleteQuery& q) {
    return;
}

void ExecutionEngine::runCreate(const CreateQuery& q) {
    if (q.table.empty()) {
        if (q.database.empty()) {
            cout << "Invalid query ." << endl;
            return;
        }

        string path = "./data/" + q.database;

        if (filesystem::exists(path)) {
            cout << "Database '" << q.database << "' already exists." << endl;
        } else {
            databases(q.database);

            if (filesystem::exists(path)) {
```

```
                add_database(fetch_structure(),q.database);
            }
            else {
                cout << "Database not created OR Unknown query ..." << endl;
            }
        }
    }
    else if(q.database.empty()){
        if(dbManager->getCurrentDatabase() == ""){
            cout << "Please Open a Database to create a Table ..." << endl;
        }
        else {
            tables t;
            t.create_table(dbManager->getCurrentDatabase(),q.table);
            add_table(fetch_structure(),dbManager->getCurrentDatabase(),q.table);
            cout << q.table << " created successfully in database " << dbManager->getCurrentDatabase() << " ..." << endl;
        }
    }
}
void ExecutionEngine::runUpdate(const UpdateQuery& q) {
    tables t;
    if (dbManager->getCurrentDatabase().empty()) {
        cout << "Please open a database to update a table ..." << endl;
        return;
    }
    vector<vector<string>> data = t.read(dbManager->getCurrentDatabase(), q.table);
    if (data.empty()) {
        cout << "Table is empty or does not exist ..." << endl;
        return;
    }
    if (!q.where_column.empty() && !q.where_operator.empty() && !q.where_value.empty()) {
        vector<vector<string>> filteredData = t.where(data, q.where_column, q.where_operator, q.where_value);

        if (filteredData.size() <= 1) {
            cout << "No rows match the WHERE condition ..." << endl;
            return;
        }
        t.update_table(dbManager->getCurrentDatabase(), q.table, q.set_column, q.set_value, q.where_column, q.where_value, q.where_operator);
    }

    else {
        t.update_table(dbManager->getCurrentDatabase(), q.table, q.set_column, q.set_value, "", "", "");
    }
}
```

3.1.3 Design and Maintenance of Database Structures

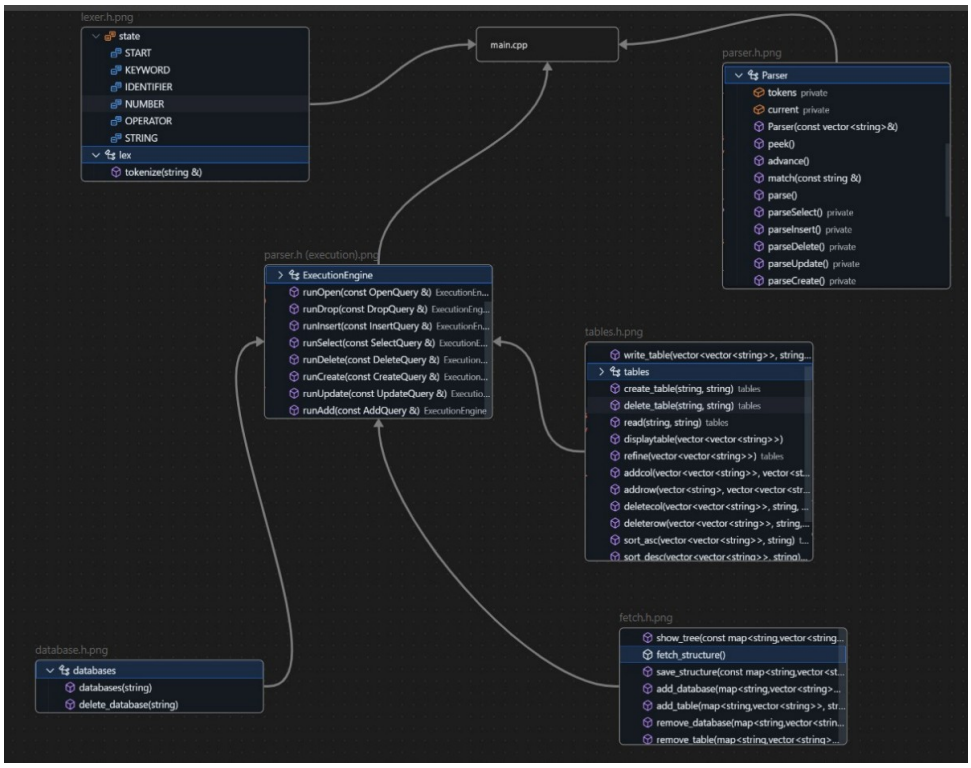
```cpp
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

#define RESET   "\033[0m"
#define RED     "\033[31m"
#define GREEN   "\033[32m"
#define YELLOW  "\033[33m"
#define BLUE    "\033[34m"
#define CYAN    "\033[36m"
#define BOLD    "\033[1m"

void show_tree(const map<string, vector<string>>& tree)
{
    cout << endl;
    for (const auto& p : tree) {
        cout << BOLD << BLUE << "[DB] " << RESET << p.first << endl;

        for (size_t i = 0; i < p.second.size(); i++) {
            string indent = "    ";
            cout << indent << CYAN << "|-- " << RESET << p.second[i] << endl;
        }
        cout << endl;
    }
}

map<string,vector<string>> fetch_structure(){
        string path = "./src/structure.csv";
        map<string,vector<string>> temp;
        ifstream file(path);

        if (!file.is_open()) {
            cerr << "Failed to open file: " << path << endl;
            return temp;
        }

        string line;
        while (getline(file, line)) {
            stringstream ss(line);
            string word;
            vector<string> parts;

            while (getline(ss, word, ',')) {
                parts.push_back(word);
            }

            if (!parts.empty()) {
                string db = parts[0];
                vector<string> tables(parts.begin() + 1, parts.end());
                temp[db] = tables;
            }
        }

        file.close();
        return temp;
    }

// THE TASKS BELOW WORK TO UPDATE THE SRC FOLDER --

void save_structure(const map<string, vector<string>> tree) {
    string path = "./src/structure.csv";
    ofstream file(path);

    if (!file.is_open()) {
        cerr << "Failed to write to file: " << path << endl;
        return;
    }

    for (const auto& [db, tables] : tree) {
        file << db;
        for (const string& table : tables) {
            file << "," << table;
        }
        file << "\n";
    }

    file.close();
}

void add_database(map<string, vector<string>> tree,string db_name) {

    if (tree.find(db_name) != tree.end()) {
        cout << "Database already exists.\n";
        return;
    }

    tree[db_name] = {};
    save_structure(tree);
}

void add_table(map<string, vector<string>> tree,string db_name,string table_name) {

    if (tree.find(db_name) == tree.end()) {
        cout << "Database not found.\n";
        return;
    }
    tree[db_name].push_back(table_name);
    save_structure(tree);

}

void remove_database(map<string, vector<string>> tree, string db_name) {
    if (tree.find(db_name) == tree.end()) {
        return;
    }

    tree.erase(db_name);
    save_structure(tree);
}

void remove_table(map<string, vector<string>> tree, string db_name, string table_name) {
    if (tree.find(db_name) == tree.end()) {
        return;
    }

    vector<string> &tables = tree[db_name];
    auto it = find(tables.begin(), tables.end(), table_name);
    if (it != tables.end()) {
        tables.erase(it);
        save_structure(tree);
    }
}
```

4 Output

```
 > OPEN AGNIBHA
Using database: AGNIBHA

AGNIBHA > SELECT * FROM NAMASTE
=============================================================
        id |       name |        age |     number |     salary |
=============================================================
         1 |      Alice |         30 |1234567890  |      50000 |
-------------------------------------------------------------
         2 |        Bob |         25 |9876543210  |      45000 |
-------------------------------------------------------------
         3 |    Charlie |         28 |          - |      47000 |
-------------------------------------------------------------
         4 |      Diana |         35 |1122334455  |      47000 |
-------------------------------------------------------------
         5 |      Ethan |         40 |9988776655  |      60000 |
-------------------------------------------------------------
         6 |      Farah |         29 |8877665544  |      53000 |
-------------------------------------------------------------
         7 |     George |         32 |          - |      49000 |
-------------------------------------------------------------
         8 |     Hannah |         27 |          - |      46000 |
-------------------------------------------------------------
```

```
PS C:\Intel\language\projects\SDF 2\sql-database-clone-cpp> cd "c:\Intel\language\projects\SDF 2\sql-database-clone
f ($?) { .\main }
 .d8888b.              888                    8888888b.  888888b.
d88P  Y88b             888                    888   Y88b 888  "88b
888    888             888                    888    888 888  .88P
888        .d88b.   .d88888  .d88b.  888  888  888    888 8888888K.
888       d88""88b d88" 888 d8P  Y8b 'Y8bd8P'  888    888 888  "Y88b
888    888 888  888 888  888 88888888   X88K    888    888 888    888
Y88b  d88P Y88..88P Y88b 888 Y8b.     .d8""8b.  888  .d88P 888   d88P
 "Y8888P"   "Y88P"   "Y88888  "Y8888  888  888  8888888P"  8888888P"


 > SHOW DATABASES

[DB] AARNYA

[DB] AMOLIK
   |-- YU

[DB] COLLEGE
   |-- STUDENT

[DB] SDF

[DB] VANSH

 >
```

5 Flowchart/UML



6 Conclusion

The C++ Database Management System project successfully demonstrates how fundamental C++ features can be applied to create a functional database system. While deliberately keeping the implementation straightforward, the project showcases the power

of object-oriented programming concepts, STL containers, and file handling capabilities in C++.

The modular design allows for future enhancements, such as adding more complex querying capabilities or implementing more sophisticated data structures. Overall, this project serves as a practical illustration of applying core C++ programming techniques to solve real-world data management challenges.

7 References

References

[1] GitHub Repository, *C++ Database Management System*, https://github.com/username/cpp-dbms

[2] Books, *Database Internals*,

[3] cplusplus.com, *C++ Reference*, https://cplusplus.com/reference/

[4] GeeksforGeeks, *Database Management System*, https://www.geeksforgeeks.org/dbms/

[5] Stack Overflow, *C++ File Handling Questions*, https://stackoverflow.com/questions/tagged/c%2b%2b+file-io