Computer Science

(083)

Project File

Class XII

2018 – 2019

Project Title

_ _

Roll Number: _ _ _ _ _ _ _ _ _ _

Name: _

Class & Section: _ _ _ _ _

P S SENIOR SECONDARY SCHOOL, CHENNAI – 600 004

P S SENIOR SECONDARY SCHOOL
CHENNAI – 600 004

BONAFIDE CERTIFICATE

Class XII
COMPUTER SCIENCE
PROJECT FILE

Hall Ticket No.: _ _ _ _ _ _

Certified bonafide project work, titled _, done by _ of class XII-A for the academic year 2018 – 2019, in the subject of Computer Science (083), done under my supervision.

Subject Teacher
Dept. of Computer Science
P S Senior Secondary School
Mylapore, Chennai.

Date: _ _ _ _ _ _ _

Submitted for the Practical Examination held at the School Center, Chennai on _ _ _ _ _ _ _ _ _ _ _

Internal Examiner

School/Lab Seal

External Examiner:

No:_ _ _ _ _ _ _ _ _ _ _ _ _

Sign:_ _ _ _ _ _ _ _ _ _ _ _

P S SENIOR SECONDARY SCHOOL
CHENNAI – 600 004

ACKNOWLEDGEMENT

I acknowledge the support provided by the management, principal, faculty and staff of the computer science department, P. S. Senior Secondary School in helping me to successfully complete and submit my project in computer science.

INDEX

TITLE	PAGE
1. Project Statement	1
2. Source Code	3
3. Project Execution	25
4. List of Attributes	32
5. Limitations/ Shortcomings	33
6. Conclusion	34

Project Statement:

This project is a Natural Language Processor for SQL (Stuctured Query Language) queries.

The *aim* of the project is to convert a query on an SQL database inputted in Natural English Language into the SQL equivalent.

Primary Use Case/ User Type:

Consider a user who is looking at a table. He/ she knows the table name, column name and other related table details. The user would like to *query* this table. However, the user does not know the SQL syntax to do so.

This project enables such a user to input the required query in *Natural English* and in return obtain the query equivalent in *SQL* syntax.

Major Components of the Project:

1. **Data Structures**:
    a. **Dynamic Queue***: First-in-First-out data structure. Allows the insertion of elements of any type into the front end of the queue or deletion from the rear end of the queue*.
    b. **Dynamic Vector:** *Serves as an array with dynamic memory allocation. Allows pushing back of elements into the vector, as well as random access.*

2. **The Parser:**

    This section of the source code is responsible for parsing the given English query by:

    - *dividing the query into Tokens,*
    - *assigning attributes to the Tokens,*
    - *comparing the Tokenised input with pre-defined query formats,*
    - *"matching confidence" by comparing the preliminary query data with the actual column names, and picking the most suitable column candidates.*
    - *displaying of the final query*

3. **Files Used:**
    a. **QUERY_FORMATS.TXT**: *Consists of a list of possible formats/ templates of the input English query, using certain special rules.*
    b. **COLUMN_FORMATS.TXT:** *Consists of a list of possible formats/ templates of the columns involved in the final query using certain special rules.*
    c. **CONDITION_FORMATS.TXT**: *Consists of a list of possible formats/ templates of the condition involved in the final SQL query (WHERE Clause).*

4. **Other Classes/ Structs involved:**
    a. **Token:** *A class specifically designed to help parsing the input query, consisting of a char array "data" and a char array "attrib" to store the data and the attributes of the token respectively*
    b. **Node:** *A struct used in dynamic data structures to represents every element of the structure.*
    c. **SQL_SELECT_QUERY:** *A struct that defines the final SQL query consisting of members that include:*
        i. *Components of the WHERE clause*
        ii. *SELECT type (Columns to be selected)*
        iii. *Table Name*
        iv. *Query Type*

SOURCE CODE

```
/*

********************** NATURAL LANGUAGE PROCESSOR FOR SQL QUERIES ******************

Author: Nishant Mahesh
School: P.S Senior Secondary School
Class Name: XII 'A'
Class Roll No: 18
Board Exam Roll No: 4602621

*****************************************************************************************

*/

#include <iostream>
#include <cstdio>
#include <string>
#include <fstream>
#include <iomanip>
#include <stdlib.h>


using namespace std;

const int INF = 1e9 + 7;

/*
 Function Name      : clear_string(char a[])
 Return Type        : void
 Description        : Clears a character array
 */

void clear_string(char a[]) {
  strcpy(a, "");
}


/*
 Class Name                :   Token
 Short Description         :   Data structure which contains 2 data members: "data" and
                               "attrib"
 Primary Functionality     :   The query will be parsed into tokens, each with a character
                               array data member called "data", and special attributes
                               associated with the token (indicating relevance in SQL query)
                               stored in "attrib"
 */
class Token{
public:
  char data[250];
  char attrib[250];
  Token() {}
  Token(const Token& t) {
    strcpy(data, t.data);
    strcpy(attrib, t.attrib);
  }
  void operator = (Token& t) {
    strcpy(data, t.data);
    strcpy(attrib, t.attrib);
  }
};

/*
 Function Name        : operator << (ostream& , const Token&)
 Return Type          : ostream&
 Description          : (Operator Overloading)
                        Overloading the " << " operator specifically for Token data
                        type
 */
ostream& operator << (ostream &out, const Token&T) {
  out << T.data;
  return out;
}


/*
 Function Name        : ERROR(char*)
 Return Type          : void
```

```
 Description          : Displays an Error Message along with the File Name
 */
void ERROR(char* message) {
  cout << __FILE__ << " " <<  message << endl;
}


/*Struct Name      :   Node
 Description      :   Common Node template for all dynamic data structures.
                      Consists of the data stored in the Node of data type = "Type";
                      Consists of a pointer "next" - of type: 'Node' itself
 */
template <typename Type>
struct Node {
  Type data;
  Node* next;
};

//******************** COMPONENT - 1: DYNAMIC QUEUE *************************


/* Class Name              :   Queue
 Short Description        :   Dynamic Data Structure.
                              First-In-First-Out (FIFO) or Last-In-Last-Out (LILO)
                              data structure
 Primary Functionality    :   Allows inserting of elements of a given data type
                              (templatised class) into the back of the queue;
                              Allows deleting of elements of the same data type
                              from the front of the queue.
 */
template <typename Type>
class Queue {
public:
  int type;
  Queue();
  Queue(Queue<Type>&);
  Node <Type>* front;
  Node <Type>* rear;
  int size;
  int isEmpty();
  void Queue_Insert(Type);
  void Queue_Delete(Type&);
  void Queue_Delete();
  void dump();
  Type operator [] (int);
  void operator = (const Queue<Type>&);
  ~Queue();
  void print();
};


/*
 Class Name        :  Queue
 Member Function   :  Queue()
 Return-Type       :  ------
 Description       :  (Constructor)
                      Initialises the front and rear pointers of a new queue object
                      to NULL pointer;
                      Initialises queue size to zero.
 */
template <typename Type>
Queue<Type>::Queue() {
  front = 0;
  rear = 0;
  size = 0;
  type = 0;
}


/*
 Class Name       : Queue
 Member Function : isEmpty()
 Return-Type      : int
 Description      : Indicates whether or not the queue is empty. Returns 1 if queue
                    is empty, 0 if not empty
 */
template <typename Type>
```

```
int Queue<Type>::isEmpty() {
  return (front == 0);
}

/*
 Class Name      :  Queue
 Member Function :  Queue_Insert(Type value)
 Return-Type     :  void
 Description     :  Allows insertion of elements of data type: "Type" into the rear end
                    of the queue
 */
template <typename Type>
void Queue<Type>::Queue_Insert(Type value) {
  Node <Type>* temp = new Node<Type>;
  if(temp != 0) {
    size++;
    temp -> data = value;
    temp -> next = 0;
    if(isEmpty()) {
      front = temp;
      rear = temp;
    }
    else {
      rear -> next = temp;
      rear = temp;
    }
  }
  else {
    char err[100] = "ERROR 1: CLASS--QUEUE--QUEUE FULL--CANNOT INSERT!";
    cout << "Line No:" << __LINE__ << " ";
    ERROR(err);
  }
}

/*
 Class Name      :  Queue
 Member Function :  Queue_Delete(Type& popped)
 Return-Type     :  void
 Description     :  Allows deleting of elements of the data type: "Type" from the
                    front end of the queue.
                    The value of the deleted element is passed by reference to
                    a variable: "popped"
 */
template <typename Type>
void Queue<Type>::Queue_Delete(Type& popped) {
  if(isEmpty()) {
    char err[100] = "ERROR 2: CLASS--QUEUE--QUEUE EMPTY--CANNOT DELETE!";
    cout << "Line No:" << __LINE__ << " ";
    ERROR(err);
  }
  else {
    size--;
    Node <Type>* temp = front;
    popped = temp -> data;
    front = front -> next;
    delete temp;
    if(front == 0) //Indicating Queue is now empty
      rear = 0;
  }
}


/*
 Class Name      :  Queue
 Member Function :  Queue_Delete()
 Return-Type     :  void
 Description     :  (OVERLOADED FUNCTION)
                    Allows deleting of elements of the data type: "Type" from the
                    front end of the queue.
                    Here, the value of the deleted Node is not used.
 */
template <typename Type>
void Queue<Type>::Queue_Delete() {
  if(isEmpty()) {
    char err[100] = "ERROR 2: CLASS--QUEUE--QUEUE EMPTY--CANNOT DELETE!";
    cout << "Line No:" << __LINE__ << " ";
    ERROR(err);
```

```
    }
    else {
      size--;
      Node <Type>* temp = front;
      front = front -> next;
      if(temp) {
        delete temp;
      }
      if(front == 0)
        rear = 0;
    }
  }

  /*
   Class Name        :    Queue
   Return-Type       :    Node <Type>
   Member Function   :    operator[] (int)
   Description       :    (Operator Overloading)
                          Gives an iterative functionality to the queue. Allows the user
                          to view a particular element which is at the "target_index"th
                          position from the front of the queue.
   */
  template <typename Type>
  Type Queue<Type>::operator [] (int target_index) {
    if (target_index > size || target_index <= 0) {
      char err[100] = "ERROR 3: CLASS--QUEUE--INVALID INDEX, CANNOT ACCESS!";
      cout << "Line No:" << __LINE__ << " ";
      ERROR(err);
      Type rand;
      return rand;
    }
    else {
      int current_index = 1; //Queue iteration is 1-based index
      Node <Type>* target_node = front;
      while(current_index < target_index) {
        target_node = target_node -> next;
        current_index++;
      }
      return (target_node -> data);
    }
  }

  /*
   Class Name      : Queue
   Return-Type     : ---
   Member Function : dump()
   Description     : Deletes all data
   */

  template <typename Type>
  void Queue<Type>::dump() {
    type = -1;
    while(front) {
      Queue_Delete();
    }

  }
  /*
   Class Name      :  Queue
   Return-Type     :  ----
   Member Function :  ~Queue()
   Description     :  (Destructor)
                      Destroys the pointers involved in the class to ensure no orphaned
                      locations
   */
  template <typename Type>
  Queue<Type> :: ~Queue() {
    dump();
  }

  /*
   Class Name      :  Queue
   Member Function :  operator = ()
   Return-Type     :  ----
   Description     :  (Operator Overloading)
                      Copies the details of an existing queue into a new queue
   */
```

```
template <typename Type>
void Queue<Type>::operator = (const Queue<Type> &q) {
  dump();
  if(q.front) {
    Node <Type>* temp = q.front;
    while(temp){
      Queue_Insert(temp -> data);
      temp = temp -> next;
    }
  }
  else
    front = rear = 0;
}

/*
 Class Name      :  Queue
 Member Function :  Queue(Queue <Type> &)
 Return-Type     :  ------
 Description     :  (Copy Constructor)
                    Creates a copy of the Queue given as parameter, on instantiation
 */
template <typename Type>
Queue<Type>::Queue(Queue<Type>& q) {
  front = 0;
  rear = 0;
  size = 0;
  for(int i = 1; i <= q.size; i++) {
    Queue_Insert(q[i]);
  }
}

/*
 Class Name      :  Queue
 Member Function :  print()
 Return-Type     :  void
 Description     :  Prints all the elements in the Queue
 */
template <typename Type>
void Queue <Type>::print() {
  for(int i = 1; i <= size; i++) {
    cout << (*this)[i] << " ";
  }
  cout << endl;
}

//******************** COMPONENT - 2: DYNAMIC VECTOR ************************


/* Class Name            :   Vector
 Short Description       :   Dynamic Data Structure.
 Primary Functionality   :   Serves as a dynamic array, can push_back elements into
                             the array.
                             Allows only 1-based index arrays

 */
template <typename Type>
class Vector {
public:
  int size;
  Node <Type>* begin, *end;
  Vector();
  Vector(const Vector <Type> &);
  void Push_Back(Type);
  Type& operator [] (int);
  void operator = (const Vector<Type>&);
  void dump();
  void print();
  ~Vector();
};

/*
 Class Name         :  Vector
 Member Function    :  Vector()
 Return-Type        :  ----
 Description        :  (Constructor)
                       Initialises begin, end to null pointers, and size of vector to 0
 */
```

```
template<typename Type>
Vector <Type>::Vector() {
  begin = 0;
  end = 0;
  size = 0;
}

/*
 Class Name         :  Vector
 Member Function    :  Vector(Vector<Type>&)
 Return-Type        :  ----
 Description        :  (Copy Constructor)
                       Creates a copy of the Vector given as parameter, on instantiation
 */
template <typename Type>
Vector <Type>::Vector(const Vector<Type>& v) {
  dump();
  for(int i = 1; i <= v.size; i++) {
    Push_Back(v[i]);
  }
}

/*
 Class Name         :  Vector
 Member Function    :  Push_Back(Type)
 Return-Type        :  void
 Description        :  Inserts an element into the vector by creating a new location
 */

template <typename Type>
void Vector <Type>:: Push_Back(Type val) {
  Node <Type>* temp = new Node<Type>;
  if(temp) {
    if(size == 0) {
      temp -> data = val;
      temp -> next = 0;
      end = temp;
      begin = temp;
    }
    else {
      temp -> data = val;
      temp -> next = 0;
      end -> next = temp;
      end = temp;
    }
    size++;
  }
  else {
    char err[100] = "ERROR 4: CLASS--VECTOR-- NO SPACE! CANNOT INSERT INTO VECTOR";
    cout << "Line No:" << __LINE__ << " ";
    ERROR(err);
  }
}

/*
 Class Name         :  Vector
 Member Function    :  operator [] (int)
 Return-Type        :  Node <Type>
 Description        :  (Copy Constructor)
                       Creates a copy of the Vector given as parameter, on instantiation
 */

template <typename Type>
Type& Vector<Type>::operator [] (int target_index) {
  if (target_index > size || target_index <= 0) {
    char err[100] = "ERROR 3: CLASS--QUEUE--INVALID INDEX, CANNOT ACCESS!";
    cout << "Line No:" << __LINE__ << " ";
    ERROR(err);
    Type* rand = new Type;

    return *rand;
  }
  else {
    int current_index = 1; //Queue iteration is 1-based index
    Node <Type>* target_node = begin;
    while(current_index < target_index) {
      target_node = target_node -> next;
```

```
        current_index++;
      }
      return (target_node -> data);
    }
  }



  /*
   Class Name      : Vector
   Return-Type     : ---
   Member Function : dump()
   Description     : Deletes all data
   */
  template <typename Type>
  void Vector<Type>::dump() {
    if(size == 0)
      return;
    if(size == 1) {
      Node <Type>* temp = begin;
      delete temp;
      begin = 0;
      end = 0;
      size--;
    }
    else {
      Node <Type>* temp = begin -> next;
      while(temp != 0) {
        delete begin;
        begin = temp;
        temp = begin -> next;
        size--;
      }
      delete temp;
      size--;
    }
  }

  /*
   Class Name      :  Vector
   Member Function :  print()
   Return-Type     :  void
   Description     :  Prints all the elements in the Vector
   */
  template <typename Type>
  void Vector <Type>::print() {
    for(int i = 1; i <= size; i++) {
      cout << (*this)[i] << " ";
    }
    cout << endl;
  }

  /*
   Class Name      :  Vector
   Return-Type     :  ----
   Member Function :  ~Vector()
   Description     :  (Destructor)
                      Destroys the pointers involved in the class to ensure
                      no orphaned locations
   */
  template <typename Type>
  Vector<Type> :: ~Vector() {
    dump();
  }




  //******************** COMPONENT- 3: PARSER ******************************

  #define WILDCARD "##"

  Queue <Token> condition_formats[100];
  Queue <Token> column_selection_formats[100];
  Queue <Token> query_formats[100];


```

```
/*
 Function Name       : make_lower(char&)
 Return Type         : void
 Description         : Converts a given character to lowercase
 */
void make_lower(char& ch) {
  if(ch >= 'A' && ch <= 'Z')
    ch += 32;
}

/*
 Function Name       : make_upper(char arr[])
 Return Type         : void
 Description         : Converts a given character array to uppercase
 */
void make_upper(char arr[]) {
  for(int i = 0; i < strlen(arr); i++) {
    if(arr[i] >= 'a' && arr[i] <= 'z')
      arr[i] -= 32;
  }
}

/*
 Function Name       : make_lower(char arr[])
 Return Type         : void
 Description         : Converts a given character array to uppercase
 */
void make_lower(char arr[]) {
  for(int i = 0; i < strlen(arr); i++) {
    if(arr[i] >= 'A' && arr[i] <= 'Z')
      arr[i] += 32;
  }
}


const char SPACE = ' ';

/*
 Function Name       : substr(char* arr, int begin, int len)
 Return Type         : char*
 Description         : Returns the substring off input char array starting at index
                       "begin", and of length = "len"
 */

char* substr(char* arr, int begin, int len) {
  char* res = new char[len];
  for (int i = 0; i < len; i++)
    res[i] = *(arr + begin + i);
  res[len] = 0;
  return res;
}

int string_to_int(char inp[]) {
  int ans = 0;
  int mul = 1;
  for(int i = (int)strlen(inp) - 1; i >= 0; i--) {
    ans += (inp[i] - '0') * mul;
    mul *= 10;
  }
  return ans;
}




/*
 Function Name       : tokenise(char user_input[])
 Return Type         : Queue <Token>
 Description         : Accepts a natural language input (as char array) and tokenises
                       the input by
                       -> ignoring punctuation marks
                       -> converting all words to lowercase
                       -> assigning attributes to the tokens if necessary
                       and returns a Queue of Tokens
 */

```

```
Queue<Token> tokenise(char user_input[300]) {
  Queue<Token> tokenised_query;
  Token temporary;
  strcpy(temporary.data, "");
  strcpy(temporary.attrib, "");
  int curr_token_index = 0;
  for(int i = 0; i < strlen(user_input); i++) {
    if(user_input[i] == SPACE) {
      if(curr_token_index != 0) {
        (temporary.data)[curr_token_index] = 0;
        curr_token_index = 0;
        if(strcmp(substr(temporary.data, 0, 2), WILDCARD) == 0){
          strcpy(temporary.attrib, substr(temporary.data, 2, ((int)strlen(temporary.data) - 2)));
        }
        tokenised_query.Queue_Insert(temporary);
        clear_string(temporary.data);
        clear_string(temporary.attrib);
      }
    }
    else if((user_input[i] == '#') || strcmp(substr(temporary.data, 0, 2), WILDCARD) == 0) {
      (temporary.data)[curr_token_index] = user_input[i];
      curr_token_index++;
    }
    else if(user_input[i] != SPACE && isalnum(user_input[i])) {
      make_lower(user_input[i]);
      (temporary.data)[curr_token_index] = user_input[i];
      curr_token_index++;
    }

  }
  if(curr_token_index != 0) {
    (temporary.data)[curr_token_index] = 0;
    curr_token_index = 0;
    if(strcmp(substr(temporary.data, 0, 2), WILDCARD) == 0){
      strcpy(temporary.attrib, substr(temporary.data, 2, ((int)strlen(temporary.data) - 2)));
    }
    tokenised_query.Queue_Insert(temporary);
    clear_string(temporary.data);
  }
  return tokenised_query;
}

/*
 Struct Name         : SQL_SELECT_QUERY
 Description         : The data structure which will store all the details needed for
                       the final SELECT query
 Data Members        : char left_condition[] : Represents the left hand side of
                       the "WHERE" clause
                       char right_condition[] : Represents the right hand side of
                       the "WHERE" clause
                       Queue <Token> col_list : A Queue of all the columns to be
                       selected in the SELECT clause
 */

struct SQL_SELECT_QUERY {
  char left_condition[50];
  char right_condition[50];
  Vector <Token> col_list;
  int query_type;
  int condition_present;
  int condition_type;
  char table_name[50];
  char agg_type[50];
  void dump() {
    clear_string(left_condition);
    clear_string(right_condition);
    col_list.dump();
  }
  void col_list_dump() {
    col_list.dump();
  }
  void condition_dump() {
    clear_string(left_condition);
    clear_string(right_condition);
```

```
    }
    SQL_SELECT_QUERY() {
      query_type = 0;
      condition_type = 0;
      condition_present = 0;
    }
  };



  /*
   Function Name       : compare_format(Queue <Token> , Queue <Token> , SQL_SELECT_QUERY&)
   Return Type         : bool (indicates success/ failure i.e match/ no match)
   Description         : Compares a given Queue of tokens with a format Queue and determines
                         if the given
                         Queue matches the given format.
   */
  bool compare_format(Queue <Token> to_check, Queue <Token> format, SQL_SELECT_QUERY& ans) {
    int i = 1, j = 1;
    int match_count = 0;
    Queue <Token> buffer;
    int take_buffer = 0;
    int EXIT = 0;

    while(i <= to_check.size && j <= format.size && !EXIT) {
      if(!take_buffer) {
        if(strcmp(substr(format[j].data, 0, 2), WILDCARD) == 0) {
          take_buffer = 1;
          j++;
        }
        else {
          if(strcmp(to_check[i].data, format[j].data) == 0) {
            i++; j++;
            match_count++;
          }
          else
            i++;
        }
      }
      else if(take_buffer) {
        if(strcmp(substr(format[j].data, 0, 2), WILDCARD) == 0) {
          Token buffer_token;
          strcpy(buffer_token.data, to_check[i].data);
          buffer.Queue_Insert(buffer_token);
          i++;

          take_buffer = 0;
          //match_count++;

          if(strcmp(format[j - 1].attrib, "TN") == 0) {
            if(buffer.size != 1) {
              /*
               char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
               cout << "Line No:" << __LINE__ << " ";
               ERROR(err);
               */
              EXIT = 1;
            }
            else {
              match_count++;
            }
          }
          else if(strcmp(format[j - 1].attrib, "AGG") == 0) {
            if(buffer.size != 1) {
              /*
               char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
               cout << "Line No:" << __LINE__ << " ";
               ERROR(err);
               */
              EXIT = 1;
            }
            else {
              match_count++;
              strcpy(ans.agg_type, buffer[1].data);
            }
          }
          else if(strcmp(format[j - 1].attrib, "LC") == 0) {
```

```
768             if(buffer.size != 1) {
769               /*
770                char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
771                cout << "Line No:" << __LINE__ << " ";
772                ERROR(err);
773                */
774               EXIT = 1;
775             }
776             else {
777               match_count++;
778               strcpy(ans.left_condition, buffer[1].data);
779             }
780           }
781           else if(strcmp(format[j - 1].attrib, "RC") == 0) {
782             if(buffer.size != 1) {
783               /*
784                char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
785                cout << "Line No:" << __LINE__ << " ";
786                ERROR(err);
787                */
788               EXIT = 1;
789             }
790             else {
791               match_count++;
792               strcpy(ans.right_condition, buffer[1].data);
793             }
794           }
795           else if(strcmp(format[j - 1].attrib, "COL") == 0) {
796             if(buffer.size != 1) {
797               /*
798                char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
799                cout << "Line No:" << __LINE__ << " ";
800                ERROR(err);
801                */
802               EXIT = 1;
803             }
804             else {
805               ans.col_list.Push_Back(buffer[1]);
806               match_count++;
807             }
808           }
809           else if(strcmp(format[j - 1].attrib, "CON") == 0) {
810             int matched = 0;
811             for(int i = 1; (i < 100) && !condition_formats[i].isEmpty()
812             && !matched; i++) {
813               if(compare_format(buffer, condition_formats[i], ans)) {
814                 matched = 1;
815                 ans.condition_type = condition_formats[i].type;
816                 ans.condition_present = 1;
817
818               }
819               else
820                 ans.condition_dump();
821             }
822             if(matched) {
823               match_count++;
824             }
825           }
826           else if(strcmp(format[j - 1].attrib, "CLST") == 0) {
827             int matched = 0;
828
829             for(int i = 1; (i < 100) && !column_selection_formats[i].isEmpty()
830             && !matched; i++) {
831               if(compare_format(buffer, column_selection_formats[i], ans)) {
832                 matched = 1;
833               }
834               else
835                 ans.col_list_dump();
836             }
837             if(matched)
838               match_count++;
839           }
840           buffer.dump();
841
842         }
843         else if(strcmp(to_check[i].data, format[j].data) != 0) {
844           Token buffer_token;
```

```
          strcpy(buffer_token.data, to_check[i].data);
          buffer.Queue_Insert(buffer_token);
          i++;
        }
        else {
          take_buffer = 0;
          match_count++;
          if(strcmp(format[j - 1].attrib, "TN") == 0) {
            if(buffer.size != 1) {
              /*
               char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
               cout << "Line No:" << __LINE__ << " ";
               ERROR(err);
               */
              EXIT = 1;
            }
            else {
              match_count++;
            }
          }
          else if(strcmp(format[j - 1].attrib, "AGG") == 0) {
            if(buffer.size != 1) {
              /*
               char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
               cout << "Line No:" << __LINE__ << " ";
               ERROR(err);
               */
              EXIT = 1;
            }
            else {
              match_count++;
              strcpy(ans.agg_type, buffer[1].data);
            }
          }
          else if(strcmp(format[j - 1].attrib, "LC") == 0) {
            if(buffer.size != 1) {
              /*
               char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
               cout << "Line No:" << __LINE__ << " ";
               ERROR(err);
               */
              EXIT = 1;
            }
            else {
              match_count++;
              strcpy(ans.left_condition, buffer[1].data);
            }
          }
          else if(strcmp(format[j - 1].attrib, "RC") == 0) {
            if(buffer.size != 1) {
              /*
               char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
               cout << "Line No:" << __LINE__ << " ";
               ERROR(err);
               */
              EXIT = 1;
            }
            else {
              match_count++;
              strcpy(ans.right_condition, buffer[1].data);
            }
          }
          else if(strcmp(format[j - 1].attrib, "COL") == 0) {
            if(buffer.size != 1) {

              /*
               char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
               cout << "Line No:" << __LINE__ << " ";
               ERROR(err);
               */
              EXIT = 1;
            }
            else {

              ans.col_list.Push_Back(buffer[1]);
              match_count++;
            }
```

```
922           }
923           else if(strcmp(format[j - 1].attrib, "CON") == 0) {
924             int matched = 0;
925             for(int i = 1; (i < 100) && !condition_formats[i].isEmpty()
926             && !matched; i++) {
927               if(compare_format(buffer, condition_formats[i], ans)) {
928                 matched = 1;
929                 ans.condition_type = condition_formats[i].type;
930                 ans.condition_present = 1;
931               }
932               else
933                 ans.condition_dump();
934             }
935             if(matched) {
936               match_count++;
937
938             }
939           }
940           else if(strcmp(format[j - 1].attrib, "CLST") == 0) {
941             int matched = 0;
942
943             for(int i = 1; (i < 100) && !column_selection_formats[i].isEmpty()
944             && !matched; i++) {
945               if(compare_format(buffer, column_selection_formats[i], ans)) {
946                 matched = 1;
947               }
948               else
949                 ans.col_list_dump();
950             }
951             if(matched)
952               match_count++;
953           }
954           buffer.dump();
955           i++; j++;
956         }
957       }
958     }
959     if(take_buffer) {
960       while(i <= to_check.size) {
961         buffer.Queue_Insert(to_check[i]);
962         i++;
963       }
964       if(strcmp(format[j - 1].attrib, "TN") == 0) {
965         if(buffer.size != 1) {
966           /*
967            char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
968            cout << "Line No:" << __LINE__ << " ";
969            ERROR(err);
970            */
971           EXIT = 1;
972         }
973         else {
974           match_count++;
975         }
976       }
977       else if(strcmp(format[j - 1].attrib, "AGG") == 0) {
978             if(buffer.size != 1) {
979               /*
980                char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
981                cout << "Line No:" << __LINE__ << " ";
982                ERROR(err);
983                */
984               EXIT = 1;
985             }
986             else {
987               match_count++;
988               strcpy(ans.agg_type, buffer[1].data);
989             }
990           }
991       else if(strcmp(format[j - 1].attrib, "LC") == 0) {
992         if(buffer.size != 1) {
993           /*
994            char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
995            cout << "Line No:" << __LINE__ << " ";
996            ERROR(err);
997            */
998           EXIT = 1;
```

```
      }
      else {
        match_count++;
        strcpy(ans.left_condition, buffer[1].data);
      }
    }
    else if(strcmp(format[j - 1].attrib, "RC") == 0) {
      if(buffer.size != 1) {
        /*
         char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
         cout << "Line No:" << __LINE__ << " ";
         ERROR(err);
         */
        EXIT = 1;
      }
      else {
        strcpy(ans.right_condition, buffer[1].data);
        match_count++;
      }
    }

    else if(strcmp(format[j - 1].attrib, "COL") == 0) {
      if(buffer.size != 1) {

        /*
         char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
         cout << "Line No:" << __LINE__ << " ";
         ERROR(err);
         */
        EXIT = 1;
      }
      else {

        ans.col_list.Push_Back(buffer[1]);
        match_count++;
      }
    }
    else if(strcmp(format[j - 1].attrib, "CON") == 0) {
      int matched = 0;
      for(int i = 1; (i < 100) && !condition_formats[i].isEmpty()
      && !matched; i++) {
        if(compare_format(buffer, condition_formats[i], ans)) {
          matched = 1;
          ans.condition_type = condition_formats[i].type;
          ans.condition_present = 1;
        }
        else
          ans.condition_dump();
      }
      if(matched) {
        match_count++;

      }
    }

    else if(strcmp(format[j - 1].attrib, "CLST") == 0) {
      int matched = 0;
      for(int i = 1; (i < 100) && !column_selection_formats[i].isEmpty()
      && !matched; i++) {
        if(compare_format(buffer, column_selection_formats[i], ans)) {
          matched = 1;
        }
        else
          ans.col_list_dump();
      }
      if(matched)
        match_count++;
    }
    buffer.dump();
  }
  return (match_count == format.size);
}



/*
 Function Name       : min(int, int)
```

```
 Return Type        : int
 Description        : Returns the minimum of 2 given integers
*/
int min(int a, int b) {
  if(a <= b)
    return a;
  else
    return b;
}

/*
 Function Name      : min(int, int, int)
 Return Type        : int
 Description        : Returns the minimum of 3 given integers
*/
int min(int a, int b, int c) {
  return min(a, (min (b, c)));
}


/*
 Function Name      : editDistance(char str1[], char str2[])
 Return Type        : int
 Description        : Returns the degree of closeness of 2 strings using an algorithm.
                      Also known as Levenshtein distance
*/

int editDistance(char str1[], char str2[]) {
  make_upper(str1);
  make_upper(str2);
  int m = (int)strlen(str1);
  int n = (int)strlen(str2);
  int dp[m + 1][n + 1];
  // Fill dp[][] in bottom up manner
  for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
      // If first string is empty, only option is to
      // insert all characters of second string
      if (i == 0)
        dp[i][j] = j;  // Min. operations = j

      // If second string is empty, only option is to
      // remove all characters of second string
      else if (j == 0)
        dp[i][j] = i; // Min. operations = i

      // If last characters are same, ignore last char
      // and recur for remaining string
      else if (str1[i - 1] == str2[j - 1])
        dp[i][j] = dp[i - 1][j - 1];

      // If the last character is different, consider all
      // possibilities and find the minimum
      else
        dp[i][j] = 1 + min(dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1]);
    }
  }
  return dp[m][n];
}




char INPUT_COLUMNS[200][200];
int INPUT_COLUMNS_COUNT;
char TABLE_NAME[200];

/*
 Function Name      : match_confidence(SQL_SELECT_QUERY&)
 Return Type        : void
 Description        : Chooses the minimum editDistance columnname for all the attributes
                      of the SQL_SELECT_QUERY object
 */

void match_confidence(SQL_SELECT_QUERY &ans) {
  //*** Column Confidence Check ***
  for(int i = 1; i <= ans.col_list.size; i++) {
```

```
      int shortest_distance = INF;
      char best_candidate[200];
      for(int j = 1; j <= INPUT_COLUMNS_COUNT; j++) {
        int curr_distance = editDistance(ans.col_list[i].data, INPUT_COLUMNS[j]);
        if(curr_distance < shortest_distance) {
          strcpy(best_candidate, INPUT_COLUMNS[j]);
          shortest_distance = curr_distance;
        }
      }
      strcpy(ans.col_list[i].data, best_candidate);

    }

    //*** Condition Check ***
    {
      int shortest_distance = INF;
      char best_candidate[200];
      for(int i = 1; i <= INPUT_COLUMNS_COUNT; i++) {
        int curr_distance = editDistance(ans.left_condition, INPUT_COLUMNS[i]);
        if(curr_distance < shortest_distance) {
          shortest_distance = curr_distance;
          strcpy(best_candidate, INPUT_COLUMNS[i]);
        }
      }
      strcpy(ans.left_condition, best_candidate);
    }
  }

  /*
   Function Name       : display_final_query(SQL_SELECT_QUERY&)
   Return Type         : bool
   Description         : Prints the final SQL query to the console. Returns
                         success/ failure
   */


  bool display_final_query(SQL_SELECT_QUERY &ans) {
  int SUCCESS = 0;
    if(ans.query_type == 1) {
      cout << "SELECT ";
      if((ans.col_list).size == 0) {
        cout << "*";
      }
      else if((ans.col_list).size == 1) {
        make_upper(((ans.col_list)[1]).data);
        cout << ans.col_list[1];
      }
      else {
        make_upper((ans.col_list[1].data));
        cout << ans.col_list[1];
        for(int i = 2; i <= ans.col_list.size; i++) {
          cout << ", ";
          make_upper(ans.col_list[i].data);
          cout << ans.col_list[i];
        }
      }
      cout << endl << "FROM " << ans.table_name << endl;
      if(ans.condition_present) {
        cout << "WHERE ";
        if(ans.condition_type == 1) {
          cout << ans.left_condition << " = " << ans.right_condition;
        }
        else if (ans.condition_type == 2) {
          cout << ans.left_condition << " > " << ans.right_condition;
        }
        else if (ans.condition_type == 3) {
          cout << ans.left_condition << " < " << ans.right_condition;
        }
        else if (ans.condition_type == 4) {
          cout << ans.left_condition << " >= " << ans.right_condition;
        }
        else if (ans.condition_type == 5) {
          cout << ans.left_condition << " <= " << ans.right_condition;
        }
        else if (ans.condition_type == 6) {
          cout << ans.left_condition << " != " << ans.right_condition;
        }
```

```
      }
      SUCCESS = 1;
    }

    //Query involving aggregate functions on a single column
    else if(ans.query_type == 2) {
      if(strcmp(ans.agg_type, "number") && (strcmp(ans.agg_type, "count")) &&
      (strcmp(ans.agg_type, "maximum")) && (strcmp(ans.agg_type, "minimum")) &&
      (strcmp(ans.agg_type, "average")))
      {
        SUCCESS = 0;
      }
      else {
        cout << "SELECT ";
        if((ans.col_list).size == 0) {
          if((strcmp(ans.agg_type, "number") == 0) ||
            (strcmp(ans.agg_type, "count") == 0))
          {
            cout << "COUNT(*)";
          }
        }
        else if((ans.col_list).size == 1) {
          make_upper(((ans.col_list)[1]).data);
          if((strcmp(ans.agg_type, "number") == 0) ||
             (strcmp(ans.agg_type, "count") == 0))
          {
            cout << "COUNT(" << ans.col_list[1] << ")";
          }
          else if(strcmp(ans.agg_type, "maximum") == 0) {
            cout << "MAX(" << ans.col_list[1] << ")";
          }
          else if(strcmp(ans.agg_type, "minimum") == 0) {
            cout << "MIN(" << ans.col_list[1] << ")";
          }
          else if(strcmp(ans.agg_type, "average") == 0) {
            cout << "AVG(" << ans.col_list[1] << ")";
          }
          else
            SUCCESS = 0;
        }
        cout << endl << "FROM " << ans.table_name << endl;
        if(ans.condition_present) {
          cout << "WHERE ";
          if(ans.condition_type == 1) {
            cout << ans.left_condition << " = " << ans.right_condition;
          }
          else if (ans.condition_type == 2) {
            cout << ans.left_condition << " > " << ans.right_condition;
          }
          else if (ans.condition_type == 3) {
            cout << ans.left_condition << " < " << ans.right_condition;
          }
          else if (ans.condition_type == 4) {
            cout << ans.left_condition << " >= " << ans.right_condition;
          }
          else if (ans.condition_type == 5) {
            cout << ans.left_condition << " <= " << ans.right_condition;
          }
          else if (ans.condition_type == 6) {
            cout << ans.left_condition << " != " << ans.right_condition;
          }
        }
        SUCCESS = 1;
      }
    }
    return SUCCESS;
  }

  /*
   Function Name       : parse(Queue <Token>, SQL_SELECT_QUERY&)
   Return Type         : bool (indicates success/ failure i.e match/ no match)
   Description         : Calls the compare_format function for all formats present
                         in the query_formats file
                         and updates the attributes of the final SQL Query
   */

  bool parse(Queue <Token> query, SQL_SELECT_QUERY& ans) {
```

```
  int matched = 0;
  for(int i = 1; !query_formats[i].isEmpty() && !matched; i++) {
    if(compare_format(query, query_formats[i], ans)){
      ans.query_type = query_formats[i].type;
      match_confidence(ans);
      if(display_final_query(ans)) {
        matched = 1;
      }
    }
    else if(matched == 0){
      ans.dump();
    }
  }
  if(!matched)
    return false;
  else
    return true;
}

/*
 Function Name      : load_query_formats()
 Return Type        : void
 Description        : Loads all the query_formats into the query_formats Queue from
                      the file
 */
void load_query_formats() {
  ifstream fin("QUERY_FORMATS.TXT");
  char line[300];
  char qtype[10];
  fin >> qtype;
  int i = 1;
  while(fin.getline(line, 300)) {
    query_formats[i] = tokenise(line);
    query_formats[i].type = string_to_int(qtype);
    fin >> qtype;
    i++;
  }
  fin.close();
}


/*
 Function Name      : load_condition_formats()
 Return Type        : void
 Description        : Loads all the condition_formats into the condition_formats Queue
                      from the file
 */

void load_condition_formats() {
  ifstream fin("CONDITION_FORMATS.TXT");
  char line[300];
  char qtype[10];
  fin >> qtype;
  int i = 1;
  while(fin.getline(line, 300)) {
    condition_formats[i] = tokenise(line);
    condition_formats[i].type = string_to_int(qtype);
    fin >> qtype;
    i++;
  }
  fin.close();
}

/*
 Function Name      : load_column_formats()
 Return Type        : void
 Description        : Loads all the column_formats into the column_selection_formats Queue
                      from the file
 */
void load_column_formats() {
  ifstream fin("COLUMN_FORMATS.TXT");
  char line[300];
  int i = 1;
  while(fin.getline(line, 300)) {
    column_selection_formats[i] = tokenise(line);
    i++;
  }
```

```
  fin.close();
}

/*
 Function Name       : interact()
 Return Type         : void
 Description         : Forms the core of the user interaction interface by asking for
                       options on what the user would like to do next
 */
void interact() {
  system("clear");
  int table_exists = 0;
  cout << "WELCOME TO MY NATURAL LANGUAGE PROCESSOR ON SQL QUERIES!" << endl;
  int option = -1;

  while(!(option >= 0 && option <= 2)) {
    cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS) " << endl;
    cout << "READ TABLE DETAILS FROM A FILE           (ENTER 1) : " << endl;
    cout << "INPUT THE TABLE DETAILS IN THE CONSOLE   (ENTER 2) : " << endl;
    cout << "EXIT THE PROGRAM:                        (ENTER 0) : " << endl;
    cout << "YOUR OPTION: ";
    cin >> option;
    if(!(option >= 0 && option <= 2)) {
      system("clear");
      cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
    }
    cout << endl;
  }

  while(option != 0) {
    if(option == 1) {
      system("clear");
      cout << "********************* ACCEPTING FROM THE FILE ***************"
           << "*************************" << endl << endl;
      cout << "DISCLAIMER: PLEASE ENSURE THAT THE TEXT FILE IS IN THE FOLLOWING FORMAT"
           << endl;
      cout << "************" << endl << endl;
      cout << "*table_name* " << endl;
      cout << "*number of columns in the table" << endl;
      cout << "*column_name_1 column_name_2 column_name_3 ...*" << endl << endl;
      cout << "************" << endl << endl;
      cout << "ENTER THE FILENAME: ";
      char filename[200];
      cin >> filename;
      ifstream fin(filename);
      int EXIT = 0;
      while(!fin && !EXIT) {
        fin.close();
        cout << "\"" << filename << "\" " << "DOES NOT EXIST!" << endl << endl;
        cout << "DO YOU WANT TO EXIT? (1 IF YES, 0 IF NO): ";
        cin >> EXIT;
        cout << endl;
        if(!EXIT) {
          cout << "RE-ENTER THE FILE NAME: " << endl;
          cin >> filename;
          fin.open(filename);
        }
      }
      if(!EXIT) {
        fin >> TABLE_NAME;
        make_upper(TABLE_NAME);
        fin >> INPUT_COLUMNS_COUNT;
        fin.get();
        for(int i = 1; i <= INPUT_COLUMNS_COUNT; i++) {
          fin >> INPUT_COLUMNS[i];
          make_upper(INPUT_COLUMNS[i]);
        }
        table_exists = 1;
        system("clear");
        cout << endl << "********************* SUCCESFULLY ACCEPTED DETAILS FROM THE FILE"
             << "*************************" << endl << endl;
      }
      else
        system("clear");
      do {
        cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS) " << endl;
        cout << "READ TABLE DETAILS FROM A FILE           (ENTER 1) : " << endl;
```

```
          cout << "INPUT THE TABLE DETAILS IN THE CONSOLE   (ENTER 2) : " << endl;
          cout << "QUERY THE CURRENT TABLE                  (ENTER 3) : " << endl;
          cout << "EXIT THE PROGRAM:                        (ENTER 0) : " << endl;
          cout << "YOUR OPTION: ";
          cin >> option;
          if(!(option >= 0 && option <= 3))
            cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
        }while(!(option >= 0 && option <= 3));
        cout << endl;
      }
      else if(option == 2) {
        system("clear");
        cout << "******************** ACCEPTING TABLE DETAILS FROM CONSOLE**************"
             << "****************" << endl << endl;
        cout << "ENTER THE NAME OF THE TABLE              : ";
        cin >> TABLE_NAME;
        make_upper(TABLE_NAME);
        cout << "ENTER THE NUMBER OF COLUMNS IN THE TABLE : ";
        cin >> INPUT_COLUMNS_COUNT;

        for(int i = 1; i <= INPUT_COLUMNS_COUNT; i++) {
          cout << "ENTER THE NAME OF COLUMN " << i << "   : ";
          cin >> INPUT_COLUMNS[i];
          make_upper(INPUT_COLUMNS[i]);
        }
        cout << endl << "******************** SUCCESFULLY ACCEPTED FROM CONSOLE ************"
             << "**********************" << endl << endl;

        table_exists = 1;
        do {
          cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS) " << endl;
          cout << "READ TABLE DETAILS FROM A FILE           (ENTER 1) : " << endl;
          cout << "INPUT THE TABLE DETAILS IN THE CONSOLE   (ENTER 2) : " << endl;
          cout << "QUERY THE CURRENT TABLE                  (ENTER 3) : " << endl;
          cout << "EXIT THE PROGRAM:                        (ENTER 0) : " << endl;
          cout << "YOUR OPTION: ";
          cin >> option;
          if(!(option >= 0 && option <= 3))
            cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
        }while(!(option >= 0 && option <= 3));
        cout << endl;
      }
      else if(option == 3) {
        system("clear");
        if(!table_exists) {
          cout << "NO TABLE AVAILABLE TO QUERY! " << endl;
          cout << "READ TABLE DETAILS FROM A FILE           (ENTER 1) :" << endl;
          cout << "INPUT THE TABLE DETAILS IN THE CONSOLE   (ENTER 2) : " << endl;
        }
        else {
          char query[200];
          SQL_SELECT_QUERY ans;
          strcpy(ans.table_name, TABLE_NAME);
          cout << "ENTER YOUR QUERY IN ENGLISH   : ";
          cin.get();
          cin.getline(query, 200);
          Queue <Token> query_queue;
          query_queue = tokenise(query);
          cout << endl;
          if(!parse(query_queue, ans)) {
            cout << "******************** COULDNT GENERATE QUERY ******************
                 << "**********************" << endl << endl;
          }
          else {
            cout << endl << endl << "******************** QUERY SUCCESFULLY GENERATED
                 << "************************************" << endl;

          }
          cout << endl;
          do {
            cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS) " << endl;
            cout << "READ TABLE DETAILS FROM A FILE           (ENTER 1) : " << endl;
            cout << "INPUT THE TABLE DETAILS IN THE CONSOLE   (ENTER 2) : " << endl;
            cout << "QUERY THE CURRENT TABLE                  (ENTER 3) : " << endl;
            cout << "EXIT THE PROGRAM:                        (ENTER 0) : " << endl;
            cout << "YOUR OPTION: ";
            cin >> option;
```

```
            if(option == 4)
              option = 3;
            if(!(option >= 0 && option <= 3))
              cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
          }while(!(option >= 0 && option <= 3));
          cout << endl;
        }
      }
    }
  }


  int main() {
    load_condition_formats();
    load_column_formats();
    load_query_formats();
    interact();
  }

```

Project Execution:

1) Program Entry Screen

```
WELCOME TO MY NATURAL LANGUAGE PROCESSOR ON SQL QUERIES!
(CHOOSE ANY OF THE FOLLOWING OPTIONS)
READ TABLE DETAILS FROM A FILE          (ENTER 1) :
INPUT THE TABLE DETAILS IN THE CONSOLE  (ENTER 2) :
EXIT THE PROGRAM:                       (ENTER 0) :
YOUR OPTION: 1
```

2) Option 1 – Accepting details from a file

```
********************* ACCEPTING FROM THE FILE **************************

DISCLAIMER: PLEASE ENSURE THAT THE TEXT FILE IS IN THE FOLLOWING FORMAT
************

*table_name*
*number of columns in the table
*column_name_1 column_name_2 column_name_3 ...*

************

ENTER THE FILENAME:
```

3) Invalid File Name Error Message

```
********************* ACCEPTING FROM THE FILE **************************

DISCLAIMER: PLEASE ENSURE THAT THE TEXT FILE IS IN THE FOLLOWING FORMAT
************

*table_name*
*number of columns in the table
*column_name_1 column_name_2 column_name_3 ...*

************

ENTER THE FILENAME: RandomFileName.txt
"RandomFileName.txt" DOES NOT EXIST!

DO YOU WANT TO EXIT? (1 IF YES, 0 IF NO):
```

4) Entering a Valid File Name:

```
********************* ACCEPTING FROM THE FILE ***************************

DISCLAIMER: PLEASE ENSURE THAT THE TEXT FILE IS IN THE FOLLOWING FORMAT
************

*table_name*
*number of columns in the table
*column_name_1 column_name_2 column_name_3 ...*

************

ENTER THE FILENAME: EMPTABLE.txt
```

5) Contents of the file “EMPTABLE.txt”:

emptable.txt

```
emp
4
ename empid salary age
```

6) File Read Successful:

```
********************* SUCCESFULLY ACCEPTED DETAILS FROM THE FILE***************

(CHOOSE ANY OF THE FOLLOWING OPTIONS)
READ TABLE DETAILS FROM A FILE          (ENTER 1) :
INPUT THE TABLE DETAILS IN THE CONSOLE  (ENTER 2) :
QUERY THE CURRENT TABLE                 (ENTER 3) :
EXIT THE PROGRAM:                       (ENTER 0) :
YOUR OPTION:
```

7) Directly Inputting Table details via Console:

```
WELCOME TO MY NATURAL LANGUAGE PROCESSOR ON SQL QUERIES!
(CHOOSE ANY OF THE FOLLOWING OPTIONS)
READ TABLE DETAILS FROM A FILE           (ENTER 1) :
INPUT THE TABLE DETAILS IN THE CONSOLE   (ENTER 2) :
EXIT THE PROGRAM:                        (ENTER 0) :
YOUR OPTION: 2
```

8) Inputting the details:

```
******************** ACCEPTING TABLE DETAILS FROM CONSOLE*********

ENTER THE NAME OF THE TABLE              : EMP
ENTER THE NUMBER OF COLUMNS IN THE TABLE : 4
ENTER THE NAME OF COLUMN 1   : salary
ENTER THE NAME OF COLUMN 2   : age
ENTER THE NAME OF COLUMN 3   : empid
ENTER THE NAME OF COLUMN 4   : ename
```

9) Selecting option to Query the current table:

```
(CHOOSE ANY OF THE FOLLOWING OPTIONS)
READ TABLE DETAILS FROM A FILE           (ENTER 1) :
INPUT THE TABLE DETAILS IN THE CONSOLE   (ENTER 2) :
QUERY THE CURRENT TABLE                  (ENTER 3) :
EXIT THE PROGRAM:                        (ENTER 0) :
YOUR OPTION: 3
```

10) Query Executions:

(Single Column Selection)

a.

```
ENTER YOUR QUERY IN ENGLISH   : show me the details of employees whose age is 30

SELECT *
FROM EMP
WHERE AGE = 30
******************** QUERY SUCCESFULLY GENERATED ***********************************
```

b.

```
ENTER YOUR QUERY IN ENGLISH   : please tell me the names of employees whose salary is 2000

SELECT ENAME
FROM EMP
WHERE SALARY = 2000

********************* QUERY SUCCESFULLY GENERATED ***********************************
```

(Multiples Column Selection)

a.

```
ENTER YOUR QUERY IN ENGLISH   : show me the names and ages of employees whose salary is 30000

SELECT ENAME, AGE
FROM EMP
WHERE SALARY = 30000

********************* QUERY SUCCESFULLY GENERATED ***********************************
```

b.

```
ENTER YOUR QUERY IN ENGLISH   : Tell me the names, salaries and Employee-IDs of employees whose age is not equal to 35

SELECT ENAME, SALARY, EMPID
FROM EMP
WHERE AGE != 35

******************** QUERY SUCCESFULLY GENERATED **********************************
```

(Execution despite Typos)

```
ENTER YOUR QUERY IN ENGLISH   : show me the ags and salry of employees whose salary is greater than 36000

SELECT AGE, SALARY
FROM EMP
WHERE SALARY > 36000

********************* QUERY SUCCESFULLY GENERATED ***********************************
```

(Aggregate Functions)

a.

```
ENTER YOUR QUERY IN ENGLISH   : tell me the maximum age of all employees

SELECT MAX(AGE)
FROM EMP


********************* QUERY SUCCESFULLY GENERATED ***********************************
```

b.

```
ENTER YOUR QUERY IN ENGLISH   : show me the average salary of all employees

SELECT AVG(SALARY)
FROM EMP


********************* QUERY SUCCESFULLY GENERATED ***********************************
```

c.

```
ENTER YOUR QUERY IN ENGLISH   : tell me the average salary of employees whose age is greater than 30

SELECT AVG(SALARY)
FROM EMP
WHERE AGE > 30

********************* QUERY SUCCESFULLY GENERATED ***********************************
```

d.

```
ENTER YOUR QUERY IN ENGLISH   : display the details of employees whose age isn't 30

SELECT *
FROM EMP
WHERE AGE != 30

********************* QUERY SUCCESFULLY GENERATED ***********************************
```

11) QUERY_FORMATS.txt (Contents):

```
1 show me the details of ##TN whose ##CON
1 tell me the details of ##TN whose ##CON
1 give me the details of ##TN whose ##CON
1 display the details of ##TN whose ##CON
1 please show me the details of ##TN whose ##CON
1 please tell me the details of ##TN whose ##CON
1 please give me the details of ##TN whose ##CON
1 please display the details of ##TN whose ##CON
2 show me the ##AGG ##COL of all ##TN
2 show me the ##AGG ##COL of ##TN
2 show me the ##AGG ##COL of ##TN whose ##CON
2 show me the ##AGG ##COL of ##TN whose ##CON
2 show me the ##AGG ##COL of ##TN whose ##CON
2 show me the ##AGG of ##TN whose ##CON
2 show me the ##AGG of ##TN with ##CON
2 tell me the ##AGG ##COL of all ##TN
2 tell me the ##AGG ##COL of ##TN
2 tell me the ##AGG ##COL of ##TN whose ##CON
2 tell me the ##AGG ##COL of ##TN whose ##CON
2 tell me the ##AGG ##COL of ##TN whose ##CON
2 tell me the ##AGG of ##TN whose ##CON
2 give me the ##AGG ##COL of ##TN whose ##CON
2 give me the ##AGG ##COL of ##TN whose ##CON
2 give me the ##AGG ##COL of ##TN whose ##CON
2 give me the ##AGG of ##TN whose ##CON
2 please show me the ##AGG ##COL of all ##TN
2 please show me the ##AGG ##COL of ##TN
2 please show me the ##AGG ##COL of ##TN whose ##CON
2 please show me the ##AGG ##COL of ##TN whose ##CON
2 please show me the ##AGG ##COL of ##TN whose ##CON
2 please show me the ##AGG of ##TN whose ##CON
2 please show me the ##AGG of ##TN with ##CON
2 please tell me the ##AGG ##COL of all ##TN
2 please tell me the ##AGG ##COL of ##TN
2 please tell me the ##AGG ##COL of ##TN whose ##CON
2 please tell me the ##AGG ##COL of ##TN whose ##CON
2 please tell me the ##AGG ##COL of ##TN whose ##CON
2 please tell me the ##AGG of ##TN whose ##CON
2 please give me the ##AGG ##COL of ##TN whose ##CON
2 please give me the ##AGG ##COL of ##TN whose ##CON
2 please give me the ##AGG ##COL of ##TN whose ##CON
2 please give me the ##AGG of ##TN whose ##CON
1 select the details of ##TN whose ##CON
1 show me the ##CLST of ##TN whose ##CON
1 tell me the ##CLST of ##TN whose ##CON
1 give me the ##CLST of ##TN whose ##CON
1 display the ##CLST of ##TN whose ##CON
1 select the ##CLST of ##TN whose ##CON
1 show me the ##CLST of the ##TN whose ##CON
1 tell me the ##CLST of the ##TN whose ##CON
1 give me the ##CLST of the ##TN whose ##CON
1 display the ##CLST of the ##TN whose ##CON
1 select the ##CLST of the ##TN whose ##CON
1 show me the ##CLST of a ##TN whose ##CON
1 tell me the ##CLST of a ##TN whose ##CON
1 give me the ##CLST of a ##TN whose ##CON
1 display the ##CLST of the ##TN whose ##CON
1 select the ##CLST of the ##TN whose ##CON
1 show me the ##CLST of an ##TN whose ##CON
1 tell me the ##CLST of an ##TN whose ##CON
1 give me the ##CLST of an ##TN whose ##CON
```

12) CONDITION_FORMATS.txt (Contents):

COLUMN_FORMATS.txt

```
##COL and ##COL
##COL
##COL ##COL
##COL ##COL ##COL
##COL and ##COL and ##COL
##COL ##COL and ##COL
##COL ##COL ##COL ##COL
```

13) COLUMN_FORMATS.txt (Contents):

CONDITION_FORMATS.txt

```
1 ##LC is ##RC
1 ##LC are ##RC
2 ##LC is greater than ##RC
2 ##LC greater than ##RC
2 ##LC are greater than ##RC
2 ##LC is bigger than ##RC
2 ##LC are bigger than ##RC
3 ##LC is lesser than ##RC
3 ##LC is less than ##RC
3 ##LC less than ##RC
3 ##LC lesser than ##RC
4 ##LC is greater than or equal to ##RC
4 ##LC is great or equal to ##RC
4 ##LC are greater than or equal to ##RC
4 ##LC are great or equal to ##RC
5 ##LC is less than or equal to ##RC
5 ##LC is lesser or equal to ##RC
5 ##LC are less than or equal to ##RC
5 ##LC are lesser or equal to ##RC
6 ##LC is not ##RC
6 ##LC isnt ##RC
6 ##LC are not ##RC
6 ##LC arent ##RC
6 ##LC is not equal to ##RC
6 ##LC are not equal to ##RC
```

<u>List of Attributes and their meanings:</u>

1. **"##LC":** *Represents the left half of a condition in a WHERE condition clause*

2. **"##RC":** *Represents the right half of the condition in a WHERE condition clause*

3. **"##CLST":** *Represents a list of columns*

4. **"##COL":** *Represents a single column*

5. **"##AGG":** *Represents an Aggregate function key word*

6. **"##TN":** *Represents the table name*

7. **"##CON":** *Represents a condition*

Limitations/ Short Comings:

1) *"IN" and "BETWEEN" clauses yet to be included*
2) *"GROUP BY" clauses haven't been included yet*
3) *Equi-Join statements haven't been included yet*

Ideas to enhance the project:

1) *Connect SQL with C++ and execute the output query directly on SQL, retrieve the SQL output and print it to the C++ console.*

2) *Provide a "Teacher" module, where a user can teach the program new formats to answer better and newer queries, which will be saved and used in future sessions.*

3) *Train the program to understand contextual queries*

Conclusion:

The project titled _____
done by _____for the academic year

2018-2019, has been completed and compiled, tested and executed successfully