



Computer Science

(083)

Project File

Class XII

2018 – 2019

Project Title

Roll Number: _____

Name: _____

Class & Section: _____



**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

<u>TITLE</u>	<u>PAGE</u>
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 (Structured 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

```

1  /*
2
3  ***** NATURAL LANGUAGE PROCESSOR FOR SQL QUERIES *****
4
5  Author: Nishant Mahesh
6  School: P.S Senior Secondary School
7  Class Name: XII 'A'
8  Class Roll No: 18
9  Board Exam Roll No: 4602621
10
11 *****
12
13 */
14
15 #include <iostream>
16 #include <cstdio>
17 #include <string>
18 #include <fstream>
19 #include <iomanip>
20 #include <stdlib.h>
21
22
23 using namespace std;
24
25 const int INF = 1e9 + 7;
26
27 /*
28  Function Name      : clear_string(char a[])
29  Return Type       : void
30  Description        : Clears a character array
31  */
32
33 void clear_string(char a[]) {
34     strcpy(a, "");
35 }
36
37
38 /*
39  Class Name          : Token
40  Short Description   : Data structure which contains 2 data members: "data" and
41                        "attrib"
42  Primary Functionality : The query will be parsed into tokens, each with a character
43                        array data member called "data", and special attributes
44                        associated with the token (indicating relevance in SQL query)
45                        stored in "attrib"
46  */
47 class Token{
48 public:
49     char data[250];
50     char attrib[250];
51     Token() {}
52     Token(const Token& t) {
53         strcpy(data, t.data);
54         strcpy(attrib, t.attrib);
55     }
56     void operator = (Token& t) {
57         strcpy(data, t.data);
58         strcpy(attrib, t.attrib);
59     }
60 };
61
62 /*
63  Function Name      : operator << (ostream& , const Token&)
64  Return Type       : ostream&
65  Description        : (Operator Overloading)
66                        Overloading the " << " operator specifically for Token data
67                        type
68  */
69 ostream& operator << (ostream& out, const Token&T) {
70     out << T.data;
71     return out;
72 }
73
74
75 /*
76  Function Name      : ERROR(char*)
77  Return Type       : void

```



```

78      Description           : Displays an Error Message along with the File Name
79      */
80      void ERROR(char* message) {
81          cout << __FILE__ << " " << message << endl;
82      }
83
84
85      /*Struct Name      :      Node
86      Description       :      Common Node template for all dynamic data structures.
87                          Consists of the data stored in the Node of data type = "Type";
88                          Consists of a pointer "next" - of type: 'Node' itself
89      */
90      template <typename Type>
91      struct Node {
92          Type data;
93          Node* next;
94      };
95
96      //***** COMPONENT - 1: DYNAMIC QUEUE *****
97
98
99      /* Class Name      :      Queue
100     Short Description   :      Dynamic Data Structure.
101                          First-In-First-Out (FIFO) or Last-In-Last-Out (LILO)
102                          data structure
103     Primary Functionality :      Allows inserting of elements of a given data type
104                                  (templatised class) into the back of the queue;
105                                  Allows deleting of elements of the same data type
106                                  from the front of the queue.
107     */
108     template <typename Type>
109     class Queue {
110     public:
111         int type;
112         Queue();
113         Queue(Queue<Type>&);
114         Node <Type>* front;
115         Node <Type>* rear;
116         int size;
117         int isEmpty();
118         void Queue_Insert (Type);
119         void Queue_Delete (Type&);
120         void Queue_Delete ();
121         void dump ();
122         Type operator [] (int);
123         void operator = (const Queue<Type>&);
124         ~Queue ();
125         void print ();
126     };
127
128
129     /*
130     Class Name      :      Queue
131     Member Function :      Queue()
132     Return-Type     :      -----
133     Description     :      (Constructor)
134                          Initialises the front and rear pointers of a new queue object
135                          to NULL pointer;
136                          Initialises queue size to zero.
137     */
138     template <typename Type>
139     Queue<Type>::Queue () {
140         front = 0;
141         rear = 0;
142         size = 0;
143         type = 0;
144     }
145
146
147     /*
148     Class Name      :      Queue
149     Member Function :      isEmpty()
150     Return-Type     :      int
151     Description     :      Indicates whether or not the queue is empty. Returns 1 if queue
152                          is empty, 0 if not empty
153     */
154     template <typename Type>

```

```

155 int Queue<Type>::isEmpty() {
156     return (front == 0);
157 }
158
159 /*
160 Class Name      : Queue
161 Member Function : Queue_Insert(Type value)
162 Return-Type     : void
163 Description     : Allows insertion of elements of data type: "Type" into the rear end
164                   of the queue
165 */
166 template <typename Type>
167 void Queue<Type>::Queue_Insert (Type value) {
168     Node <Type>* temp = new Node<Type>;
169     if(temp != 0) {
170         size++;
171         temp -> data = value;
172         temp -> next = 0;
173         if(isEmpty()) {
174             front = temp;
175             rear = temp;
176         }
177         else {
178             rear -> next = temp;
179             rear = temp;
180         }
181     }
182     else {
183         char err[100] = "ERROR 1: CLASS--QUEUE--QUEUE FULL--CANNOT INSERT!";
184         cout << "Line No:" << __LINE__ << " ";
185         ERROR(err);
186     }
187 }
188
189 /*
190 Class Name      : Queue
191 Member Function : Queue_Delete(Type& popped)
192 Return-Type     : void
193 Description     : Allows deleting of elements of the data type: "Type" from the
194                   front end of the queue.
195                   The value of the deleted element is passed by reference to
196                   a variable: "popped"
197 */
198 template <typename Type>
199 void Queue<Type>::Queue_Delete (Type& popped) {
200     if(isEmpty()) {
201         char err[100] = "ERROR 2: CLASS--QUEUE--QUEUE EMPTY--CANNOT DELETE!";
202         cout << "Line No:" << __LINE__ << " ";
203         ERROR(err);
204     }
205     else {
206         size--;
207         Node <Type>* temp = front;
208         popped = temp -> data;
209         front = front -> next;
210         delete temp;
211         if(front == 0) //Indicating Queue is now empty
212             rear = 0;
213     }
214 }
215
216
217 /*
218 Class Name      : Queue
219 Member Function : Queue_Delete()
220 Return-Type     : void
221 Description     : (OVERLOADED FUNCTION)
222                   Allows deleting of elements of the data type: "Type" from the
223                   front end of the queue.
224                   Here, the value of the deleted Node is not used.
225 */
226 template <typename Type>
227 void Queue<Type>::Queue_Delete () {
228     if(isEmpty()) {
229         char err[100] = "ERROR 2: CLASS--QUEUE--QUEUE EMPTY--CANNOT DELETE!";
230         cout << "Line No:" << __LINE__ << " ";
231         ERROR(err);

```

```

232     }
233     else {
234         size--;
235         Node <Type>* temp = front;
236         front = front -> next;
237         if(temp) {
238             delete temp;
239         }
240         if(front == 0)
241             rear = 0;
242     }
243 }
244
245 /*
246 Class Name      : Queue
247 Return-Type     : Node <Type>
248 Member Function : operator[] (int)
249 Description     : (Operator Overloading)
250                  Gives an iterative functionality to the queue. Allows the user
251                  to view a particular element which is at the "target_index"th
252                  position from the front of the queue.
253 */
254 template <typename Type>
255 Type Queue<Type>::operator [] (int target_index) {
256     if (target_index > size || target_index <= 0) {
257         char err[100] = "ERROR 3: CLASS--QUEUE--INVALID INDEX, CANNOT ACCESS!";
258         cout << "Line No:" << __LINE__ << " ";
259         ERROR(err);
260         Type rand;
261         return rand;
262     }
263     else {
264         int current_index = 1; //Queue iteration is 1-based index
265         Node <Type>* target_node = front;
266         while(current_index < target_index) {
267             target_node = target_node -> next;
268             current_index++;
269         }
270         return (target_node -> data);
271     }
272 }
273
274 /*
275 Class Name      : Queue
276 Return-Type     : ---
277 Member Function : dump()
278 Description     : Deletes all data
279 */
280
281 template <typename Type>
282 void Queue<Type>::dump () {
283     type = -1;
284     while(front) {
285         Queue_Delete();
286     }
287 }
288
289 /*
290 Class Name      : Queue
291 Return-Type     : ----
292 Member Function : ~Queue()
293 Description     : (Destructor)
294                  Destroys the pointers involved in the class to ensure no orphaned
295                  locations
296 */
297 template <typename Type>
298 Queue<Type> :: ~Queue () {
299     dump();
300 }
301
302 /*
303 Class Name      : Queue
304 Member Function : operator = ()
305 Return-Type     : ----
306 Description     : (Operator Overloading)
307                  Copies the details of an existing queue into a new queue
308 */

```

```

309 template <typename Type>
310 void Queue<Type>::operator = (const Queue<Type> &q) {
311     dump();
312     if(q.front()) {
313         Node <Type>* temp = q.front();
314         while(temp){
315             Queue_Insert(temp -> data);
316             temp = temp -> next;
317         }
318     }
319     else
320         front = rear = 0;
321 }
322
323 /*
324 Class Name      : Queue
325 Member Function : Queue(Queue <Type> &)
326 Return-Type     : -----
327 Description     : (Copy Constructor)
328                  Creates a copy of the Queue given as parameter, on instantiation
329 */
330 template <typename Type>
331 Queue<Type>::Queue(Queue<Type>& q) {
332     front = 0;
333     rear = 0;
334     size = 0;
335     for(int i = 1; i <= q.size; i++) {
336         Queue_Insert(q[i]);
337     }
338 }
339
340 /*
341 Class Name      : Queue
342 Member Function : print()
343 Return-Type     : void
344 Description     : Prints all the elements in the Queue
345 */
346 template <typename Type>
347 void Queue <Type>::print() {
348     for(int i = 1; i <= size; i++) {
349         cout << (*this)[i] << " ";
350     }
351     cout << endl;
352 }
353
354 //***** COMPONENT - 2: DYNAMIC VECTOR *****
355
356 /* Class Name      : Vector
357 Short Description   : Dynamic Data Structure.
358 Primary Functionality : Serves as a dynamic array, can push_back elements into
359                        the array.
360                        Allows only 1-based index arrays
361
362 */
363
364 template <typename Type>
365 class Vector {
366 public:
367     int size;
368     Node <Type>* begin, *end;
369     Vector();
370     Vector(const Vector <Type> &);
371     void Push_Back(Type);
372     Type& operator [] (int);
373     void operator = (const Vector<Type>&);
374     void dump();
375     void print();
376     ~Vector();
377 };
378
379 /*
380 Class Name      : Vector
381 Member Function : Vector()
382 Return-Type     : ----
383 Description     : (Constructor)
384                  Initialises begin, end to null pointers, and size of vector to 0
385 */

```

```

386 template<typename Type>
387 Vector <Type>::Vector() {
388     begin = 0;
389     end = 0;
390     size = 0;
391 }
392
393 /*
394 Class Name      : Vector
395 Member Function : Vector(Vector<Type>&)
396 Return-Type     : ----
397 Description     : (Copy Constructor)
398                  Creates a copy of the Vector given as parameter, on instantiation
399 */
400 template <typename Type>
401 Vector <Type>::Vector(const Vector<Type>& v) {
402     dump();
403     for(int i = 1; i <= v.size; i++) {
404         Push_Back(v[i]);
405     }
406 }
407
408 /*
409 Class Name      : Vector
410 Member Function : Push_Back(Type)
411 Return-Type     : void
412 Description     : Inserts an element into the vector by creating a new location
413 */
414
415 template <typename Type>
416 void Vector <Type>:: Push_Back(Type val) {
417     Node <Type>* temp = new Node<Type>;
418     if(temp) {
419         if(size == 0) {
420             temp -> data = val;
421             temp -> next = 0;
422             end = temp;
423             begin = temp;
424         }
425         else {
426             temp -> data = val;
427             temp -> next = 0;
428             end -> next = temp;
429             end = temp;
430         }
431         size++;
432     }
433     else {
434         char err[100] = "ERROR 4: CLASS--VECTOR-- NO SPACE! CANNOT INSERT INTO VECTOR";
435         cout << "Line No:" << __LINE__ << " ";
436         ERROR(err);
437     }
438 }
439
440 /*
441 Class Name      : Vector
442 Member Function : operator [] (int)
443 Return-Type     : Node <Type>
444 Description     : (Copy Constructor)
445                  Creates a copy of the Vector given as parameter, on instantiation
446 */
447
448 template <typename Type>
449 Type& Vector<Type>::operator [] (int target_index) {
450     if (target_index > size || target_index <= 0) {
451         char err[100] = "ERROR 3: CLASS--QUEUE--INVALID INDEX, CANNOT ACCESS!";
452         cout << "Line No:" << __LINE__ << " ";
453         ERROR(err);
454         Type* rand = new Type;
455
456         return *rand;
457     }
458     else {
459         int current_index = 1; //Queue iteration is 1-based index
460         Node <Type>* target_node = begin;
461         while(current_index < target_index) {
462             target_node = target_node -> next;

```

```

463         current_index++;
464     }
465     return (target_node -> data);
466 }
467 }
468
469
470
471 /*
472  Class Name      : Vector
473  Return-Type     : ---
474  Member Function : dump()
475  Description     : Deletes all data
476  */
477 template <typename Type>
478 void Vector<Type>::dump() {
479     if(size == 0)
480         return;
481     if(size == 1) {
482         Node <Type>* temp = begin;
483         delete temp;
484         begin = 0;
485         end = 0;
486         size--;
487     }
488     else {
489         Node <Type>* temp = begin -> next;
490         while(temp != 0) {
491             delete begin;
492             begin = temp;
493             temp = begin -> next;
494             size--;
495         }
496         delete temp;
497         size--;
498     }
499 }
500
501 /*
502  Class Name      : Vector
503  Member Function : print()
504  Return-Type     : void
505  Description     : Prints all the elements in the Vector
506  */
507 template <typename Type>
508 void Vector <Type>::print() {
509     for(int i = 1; i <= size; i++) {
510         cout << (*this)[i] << " ";
511     }
512     cout << endl;
513 }
514
515 /*
516  Class Name      : Vector
517  Return-Type     : ----
518  Member Function : ~Vector()
519  Description     : (Destructor)
520                   Destroys the pointers involved in the class to ensure
521                   no orphaned locations
522  */
523 template <typename Type>
524 Vector<Type> :: ~Vector() {
525     dump();
526 }
527
528
529
530
531 //***** COMPONENT- 3: PARSER *****
532
533 #define WILDCARD "##"
534
535 Queue <Token> condition_formats[100];
536 Queue <Token> column_selection_formats[100];
537 Queue <Token> query_formats[100];
538
539

```

```

540
541  /*
542  Function Name      : make_lower(char&)
543  Return Type       : void
544  Description       : Converts a given character to lowercase
545  */
546  void make_lower(char& ch) {
547      if(ch >= 'A' && ch <= 'Z')
548          ch += 32;
549  }
550
551  /*
552  Function Name      : make_upper(char arr[])
553  Return Type       : void
554  Description       : Converts a given character array to uppercase
555  */
556  void make_upper(char arr[]) {
557      for(int i = 0; i < strlen(arr); i++) {
558          if(arr[i] >= 'a' && arr[i] <= 'z')
559              arr[i] -= 32;
560      }
561  }
562
563  /*
564  Function Name      : make_lower(char arr[])
565  Return Type       : void
566  Description       : Converts a given character array to uppercase
567  */
568  void make_lower(char arr[]) {
569      for(int i = 0; i < strlen(arr); i++) {
570          if(arr[i] >= 'A' && arr[i] <= 'Z')
571              arr[i] += 32;
572      }
573  }
574
575
576  const char SPACE = ' ';
577
578  /*
579  Function Name      : substr(char* arr, int begin, int len)
580  Return Type       : char*
581  Description       : Returns the substring off input char array starting at index
582                      "begin", and of length = "len"
583  */
584
585  char* substr(char* arr, int begin, int len) {
586      char* res = new char[len];
587      for (int i = 0; i < len; i++)
588          res[i] = *(arr + begin + i);
589      res[len] = 0;
590      return res;
591  }
592
593  int string_to_int(char inp[]) {
594      int ans = 0;
595      int mul = 1;
596      for(int i = (int)strlen(inp) - 1; i >= 0; i--) {
597          ans += (inp[i] - '0') * mul;
598          mul *= 10;
599      }
600      return ans;
601  }
602
603
604
605
606  /*
607  Function Name      : tokenise(char user_input[])
608  Return Type       : Queue <Token>
609  Description       : Accepts a natural language input (as char array) and tokenises
610                      the input by
611                      -> ignoring punctuation marks
612                      -> converting all words to lowercase
613                      -> assigning attributes to the tokens if necessary
614                      and returns a Queue of Tokens
615  */
616

```

```

617 Queue<Token> tokenise(char user_input[300]) {
618     Queue<Token> tokenised_query;
619     Token temporary;
620     strcpy(temporary.data, "");
621     strcpy(temporary.attrib, "");
622     int curr_token_index = 0;
623     for(int i = 0; i < strlen(user_input); i++) {
624         if(user_input[i] == SPACE) {
625             if(curr_token_index != 0) {
626                 (temporary.data)[curr_token_index] = 0;
627                 curr_token_index = 0;
628                 if(strcmp(substr(temporary.data, 0, 2), WILDCARD) == 0){
629                     strcpy(temporary.attrib, substr(temporary.data, 2,
((int)strlen(temporary.data) - 2)));
630                 }
631                 tokenised_query.Queue_Insert(temporary);
632                 clear_string(temporary.data);
633                 clear_string(temporary.attrib);
634             }
635         }
636         else if((user_input[i] == '#') || strcmp(substr(temporary.data, 0, 2), WILDCARD) ==
0) {
637             (temporary.data)[curr_token_index] = user_input[i];
638             curr_token_index++;
639         }
640         else if(user_input[i] != SPACE && isalnum(user_input[i])) {
641             make_lower(user_input[i]);
642             (temporary.data)[curr_token_index] = user_input[i];
643             curr_token_index++;
644         }
645     }
646     if(curr_token_index != 0) {
647         (temporary.data)[curr_token_index] = 0;
648         curr_token_index = 0;
649         if(strcmp(substr(temporary.data, 0, 2), WILDCARD) == 0){
650             strcpy(temporary.attrib, substr(temporary.data, 2, ((int)strlen(temporary.data) -
2)));
651         }
652         tokenised_query.Queue_Insert(temporary);
653         clear_string(temporary.data);
654     }
655     return tokenised_query;
656 }
657
658
659 /*
660 Struct Name      : SQL_SELECT_QUERY
661 Description      : The data structure which will store all the details needed for
662                   the final SELECT query
663 Data Members     : char left_condition[] : Represents the left hand side of
664                   the "WHERE" clause
665                   char right_condition[] : Represents the right hand side of
666                   the "WHERE" clause
667                   Queue <Token> col_list : A Queue of all the columns to be
668                   selected in the SELECT clause
669 */
670
671 struct SQL_SELECT_QUERY {
672     char left_condition[50];
673     char right_condition[50];
674     Vector <Token> col_list;
675     int query_type;
676     int condition_present;
677     int condition_type;
678     char table_name[50];
679     char agg_type[50];
680     void dump() {
681         clear_string(left_condition);
682         clear_string(right_condition);
683         col_list.dump();
684     }
685     void col_list_dump() {
686         col_list.dump();
687     }
688     void condition_dump() {
689         clear_string(left_condition);
690         clear_string(right_condition);

```



```

691     }
692     SQL_SELECT_QUERY() {
693         query_type = 0;
694         condition_type = 0;
695         condition_present = 0;
696     }
697 };
698
699
700
701 /*
702 Function Name      : compare_format(Queue <Token> , Queue <Token> , SQL_SELECT_QUERY&)
703 Return Type       : bool (indicates success/ failure i.e match/ no match)
704 Description      : Compares a given Queue of tokens with a format Queue and determines
705                      if the given
706                      Queue matches the given format.
707 */
708 bool compare_format(Queue <Token> to_check, Queue <Token> format, SQL_SELECT_QUERY& ans) {
709     int i = 1, j = 1;
710     int match_count = 0;
711     Queue <Token> buffer;
712     int take_buffer = 0;
713     int EXIT = 0;
714
715     while(i <= to_check.size && j <= format.size && !EXIT) {
716         if(!take_buffer) {
717             if(strcmp(substr(format[j].data, 0, 2), WILDCARD) == 0) {
718                 take_buffer = 1;
719                 j++;
720             }
721             else {
722                 if(strcmp(to_check[i].data, format[j].data) == 0) {
723                     i++; j++;
724                     match_count++;
725                 }
726                 else
727                     i++;
728             }
729         }
730         else if(take_buffer) {
731             if(strcmp(substr(format[j].data, 0, 2), WILDCARD) == 0) {
732                 Token buffer_token;
733                 strcpy(buffer_token.data, to_check[i].data);
734                 buffer.Queue_Insert(buffer_token);
735                 i++;
736
737                 take_buffer = 0;
738                 //match_count++;
739
740                 if(strcmp(format[j - 1].attrib, "TN") == 0) {
741                     if(buffer.size != 1) {
742                         /*
743                          char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
744                          cout << "Line No:" << __LINE__ << " ";
745                          ERROR(err);
746                          */
747                         EXIT = 1;
748                     }
749                     else {
750                         match_count++;
751                     }
752                 }
753                 else if(strcmp(format[j - 1].attrib, "AGG") == 0) {
754                     if(buffer.size != 1) {
755                         /*
756                          char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
757                          cout << "Line No:" << __LINE__ << " ";
758                          ERROR(err);
759                          */
760                         EXIT = 1;
761                     }
762                     else {
763                         match_count++;
764                         strcpy(ans.agg_type, buffer[1].data);
765                     }
766                 }
767                 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     }
835     else
836         ans.col_list_dump();
837 }
838 if(matched)
839     match_count++;
840 }
841 buffer.dump();
842 }
843 else if(strcmp(to_check[i].data, format[j].data) != 0) {
844     Token buffer_token;

```

```

845     strcpy(buffer_token.data, to_check[i].data);
846     buffer.Queue_Insert(buffer_token);
847     i++;
848 }
849 else {
850     take_buffer = 0;
851     match_count++;
852     if(strcmp(format[j - 1].attrib, "TN") == 0) {
853         if(buffer.size != 1) {
854             /*
855              char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
856              cout << "Line No:" << __LINE__ << " ";
857              ERROR(err);
858              */
859             EXIT = 1;
860         }
861         else {
862             match_count++;
863         }
864     }
865     else if(strcmp(format[j - 1].attrib, "AGG") == 0) {
866         if(buffer.size != 1) {
867             /*
868              char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
869              cout << "Line No:" << __LINE__ << " ";
870              ERROR(err);
871              */
872             EXIT = 1;
873         }
874         else {
875             match_count++;
876             strcpy(ans.agg_type, buffer[1].data);
877         }
878     }
879     else if(strcmp(format[j - 1].attrib, "LC") == 0) {
880         if(buffer.size != 1) {
881             /*
882              char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
883              cout << "Line No:" << __LINE__ << " ";
884              ERROR(err);
885              */
886             EXIT = 1;
887         }
888         else {
889             match_count++;
890             strcpy(ans.left_condition, buffer[1].data);
891         }
892     }
893     else if(strcmp(format[j - 1].attrib, "RC") == 0) {
894         if(buffer.size != 1) {
895             /*
896              char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
897              cout << "Line No:" << __LINE__ << " ";
898              ERROR(err);
899              */
900             EXIT = 1;
901         }
902         else {
903             match_count++;
904             strcpy(ans.right_condition, buffer[1].data);
905         }
906     }
907     else if(strcmp(format[j - 1].attrib, "COL") == 0) {
908         if(buffer.size != 1) {
909             /*
910              char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
911              cout << "Line No:" << __LINE__ << " ";
912              ERROR(err);
913              */
914             EXIT = 1;
915         }
916         else {
917             ans.col_list.Push_Back(buffer[1]);
918             match_count++;
919         }
920     }
921 }

```

```

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;

```

```

999     }
1000     else {
1001         match_count++;
1002         strcpy(ans.left_condition, buffer[1].data);
1003     }
1004 }
1005 else if(strcmp(format[j - 1].attrib, "RC") == 0) {
1006     if(buffer.size != 1) {
1007         /*
1008          char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
1009          cout << "Line No:" << __LINE__ << " ";
1010          ERROR(err);
1011          */
1012         EXIT = 1;
1013     }
1014     else {
1015         strcpy(ans.right_condition, buffer[1].data);
1016         match_count++;
1017     }
1018 }
1019
1020 else if(strcmp(format[j - 1].attrib, "COL") == 0) {
1021     if(buffer.size != 1) {
1022         /*
1023          char err[100] = "compare_format: --- BUFFER SIZE NOT 1!";
1024          cout << "Line No:" << __LINE__ << " ";
1025          ERROR(err);
1026          */
1027         EXIT = 1;
1028     }
1029     else {
1030
1031         ans.col_list.Push_Back(buffer[1]);
1032         match_count++;
1033     }
1034 }
1035 }
1036 else if(strcmp(format[j - 1].attrib, "CON") == 0) {
1037     int matched = 0;
1038     for(int i = 1; (i < 100) && !condition_formats[i].isEmpty()
1039     && !matched; i++) {
1040         if(compare_format(buffer, condition_formats[i], ans)) {
1041             matched = 1;
1042             ans.condition_type = condition_formats[i].type;
1043             ans.condition_present = 1;
1044         }
1045         else
1046             ans.condition_dump();
1047     }
1048     if(matched) {
1049         match_count++;
1050     }
1051 }
1052 }
1053
1054 else if(strcmp(format[j - 1].attrib, "CLST") == 0) {
1055     int matched = 0;
1056     for(int i = 1; (i < 100) && !column_selection_formats[i].isEmpty()
1057     && !matched; i++) {
1058         if(compare_format(buffer, column_selection_formats[i], ans)) {
1059             matched = 1;
1060         }
1061         else
1062             ans.col_list_dump();
1063     }
1064     if(matched)
1065         match_count++;
1066 }
1067 buffer.dump();
1068 }
1069 return (match_count == format.size);
1070 }
1071
1072
1073
1074 /*
1075 Function Name      : min(int, int)

```

```

1076     Return Type           : int
1077     Description         : Returns the minimum of 2 given integers
1078 */
1079 int min(int a, int b) {
1080     if(a <= b)
1081         return a;
1082     else
1083         return b;
1084 }
1085
1086 /*
1087     Function Name       : min(int, int, int)
1088     Return Type         : int
1089     Description         : Returns the minimum of 3 given integers
1090 */
1091 int min(int a, int b, int c) {
1092     return min(a, (min(b, c)));
1093 }
1094
1095
1096 /*
1097     Function Name       : editDistance(char str1[], char str2[])
1098     Return Type         : int
1099     Description         : Returns the degree of closeness of 2 strings using an algorithm.
1100                          Also known as Levenshtein distance
1101 */
1102
1103 int editDistance(char str1[], char str2[]) {
1104     make_upper(str1);
1105     make_upper(str2);
1106     int m = (int)strlen(str1);
1107     int n = (int)strlen(str2);
1108     int dp[m + 1][n + 1];
1109     // Fill dp[][] in bottom up manner
1110     for (int i = 0; i <= m; i++) {
1111         for (int j = 0; j <= n; j++) {
1112             // If first string is empty, only option is to
1113             // insert all characters of second string
1114             if (i == 0)
1115                 dp[i][j] = j; // Min. operations = j
1116
1117             // If second string is empty, only option is to
1118             // remove all characters of second string
1119             else if (j == 0)
1120                 dp[i][j] = i; // Min. operations = i
1121
1122             // If last characters are same, ignore last char
1123             // and recur for remaining string
1124             else if (str1[i - 1] == str2[j - 1])
1125                 dp[i][j] = dp[i - 1][j - 1];
1126
1127             // If the last character is different, consider all
1128             // possibilities and find the minimum
1129             else
1130                 dp[i][j] = 1 + min(dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1]);
1131         }
1132     }
1133     return dp[m][n];
1134 }
1135
1136
1137
1138
1139 char INPUT_COLUMNS[200][200];
1140 int INPUT_COLUMNS_COUNT;
1141 char TABLE_NAME[200];
1142
1143 /*
1144     Function Name       : match_confidence(SQL_SELECT_QUERY&)
1145     Return Type         : void
1146     Description         : Chooses the minimum editDistance columnname for all the attributes
1147                          of the SQL_SELECT_QUERY object
1148 */
1149
1150 void match_confidence(SQL_SELECT_QUERY &ans) {
1151     /*** Column Confidence Check ***/
1152     for(int i = 1; i <= ans.col_list.size; i++) {

```

```

1153     int shortest_distance = INF;
1154     char best_candidate[200];
1155     for(int j = 1; j <= INPUT_COLUMNS_COUNT; j++) {
1156         int curr_distance = editDistance(ans.col_list[i].data, INPUT_COLUMNS[j]);
1157         if(curr_distance < shortest_distance) {
1158             strcpy(best_candidate, INPUT_COLUMNS[j]);
1159             shortest_distance = curr_distance;
1160         }
1161     }
1162     strcpy(ans.col_list[i].data, best_candidate);
1163 }
1164 }
1165
1166 /** Condition Check **/
1167 {
1168     int shortest_distance = INF;
1169     char best_candidate[200];
1170     for(int i = 1; i <= INPUT_COLUMNS_COUNT; i++) {
1171         int curr_distance = editDistance(ans.left_condition, INPUT_COLUMNS[i]);
1172         if(curr_distance < shortest_distance) {
1173             shortest_distance = curr_distance;
1174             strcpy(best_candidate, INPUT_COLUMNS[i]);
1175         }
1176     }
1177     strcpy(ans.left_condition, best_candidate);
1178 }
1179 }
1180
1181 /*
1182 Function Name      : display_final_query(SQL_SELECT_QUERY&)
1183 Return Type       : bool
1184 Description       : Prints the final SQL query to the console. Returns
1185                     success/ failure
1186 */
1187
1188
1189 bool display_final_query(SQL_SELECT_QUERY &ans) {
1190     int SUCCESS = 0;
1191     if(ans.query_type == 1) {
1192         cout << "SELECT ";
1193         if((ans.col_list).size == 0) {
1194             cout << "*";
1195         }
1196         else if((ans.col_list).size == 1) {
1197             make_upper(((ans.col_list)[1]).data);
1198             cout << ans.col_list[1];
1199         }
1200         else {
1201             make_upper((ans.col_list[1].data));
1202             cout << ans.col_list[1];
1203             for(int i = 2; i <= ans.col_list.size; i++) {
1204                 cout << ", ";
1205                 make_upper(ans.col_list[i].data);
1206                 cout << ans.col_list[i];
1207             }
1208         }
1209         cout << endl << "FROM " << ans.table_name << endl;
1210         if(ans.condition_present) {
1211             cout << "WHERE ";
1212             if(ans.condition_type == 1) {
1213                 cout << ans.left_condition << " = " << ans.right_condition;
1214             }
1215             else if (ans.condition_type == 2) {
1216                 cout << ans.left_condition << " > " << ans.right_condition;
1217             }
1218             else if (ans.condition_type == 3) {
1219                 cout << ans.left_condition << " < " << ans.right_condition;
1220             }
1221             else if (ans.condition_type == 4) {
1222                 cout << ans.left_condition << " >= " << ans.right_condition;
1223             }
1224             else if (ans.condition_type == 5) {
1225                 cout << ans.left_condition << " <= " << ans.right_condition;
1226             }
1227             else if (ans.condition_type == 6) {
1228                 cout << ans.left_condition << " != " << ans.right_condition;
1229             }

```

```

1230     }
1231     SUCCESS = 1;
1232 }
1233
1234 //Query involving aggregate functions on a single column
1235 else if(ans.query_type == 2) {
1236     if(strcmp(ans.agg_type, "number") && (strcmp(ans.agg_type, "count")) &&
1237        (strcmp(ans.agg_type, "maximum")) && (strcmp(ans.agg_type, "minimum")) &&
1238        (strcmp(ans.agg_type, "average")))
1239     {
1240         SUCCESS = 0;
1241     }
1242     else {
1243         cout << "SELECT ";
1244         if((ans.col_list).size == 0) {
1245             if((strcmp(ans.agg_type, "number") == 0) ||
1246                (strcmp(ans.agg_type, "count") == 0))
1247             {
1248                 cout << "COUNT(*)";
1249             }
1250         }
1251         else if((ans.col_list).size == 1) {
1252             make_upper(((ans.col_list)[1]).data);
1253             if((strcmp(ans.agg_type, "number") == 0) ||
1254                (strcmp(ans.agg_type, "count") == 0))
1255             {
1256                 cout << "COUNT(" << ans.col_list[1] << ")";
1257             }
1258             else if(strcmp(ans.agg_type, "maximum") == 0) {
1259                 cout << "MAX(" << ans.col_list[1] << ")";
1260             }
1261             else if(strcmp(ans.agg_type, "minimum") == 0) {
1262                 cout << "MIN(" << ans.col_list[1] << ")";
1263             }
1264             else if(strcmp(ans.agg_type, "average") == 0) {
1265                 cout << "AVG(" << ans.col_list[1] << ")";
1266             }
1267             else
1268                 SUCCESS = 0;
1269         }
1270         cout << endl << "FROM " << ans.table_name << endl;
1271         if(ans.condition_present) {
1272             cout << "WHERE ";
1273             if(ans.condition_type == 1) {
1274                 cout << ans.left_condition << " = " << ans.right_condition;
1275             }
1276             else if (ans.condition_type == 2) {
1277                 cout << ans.left_condition << " > " << ans.right_condition;
1278             }
1279             else if (ans.condition_type == 3) {
1280                 cout << ans.left_condition << " < " << ans.right_condition;
1281             }
1282             else if (ans.condition_type == 4) {
1283                 cout << ans.left_condition << " >= " << ans.right_condition;
1284             }
1285             else if (ans.condition_type == 5) {
1286                 cout << ans.left_condition << " <= " << ans.right_condition;
1287             }
1288             else if (ans.condition_type == 6) {
1289                 cout << ans.left_condition << " != " << ans.right_condition;
1290             }
1291         }
1292         SUCCESS = 1;
1293     }
1294 }
1295 return SUCCESS;
1296 }
1297
1298 /*
1299 Function Name      : parse(Queue <Token>, SQL_SELECT_QUERY&)
1300 Return Type       : bool (indicates success/ failure i.e match/ no match)
1301 Description       : Calls the compare_format function for all formats present
1302                   in the query_formats file
1303                   and updates the attributes of the final SQL Query
1304 */
1305
1306 bool parse(Queue <Token> query, SQL_SELECT_QUERY& ans) {

```



```

1307     int matched = 0;
1308     for(int i = 1; !query_formats[i].isEmpty() && !matched; i++) {
1309         if(compare_format(query, query_formats[i], ans)){
1310             ans.query_type = query_formats[i].type;
1311             match_confidence(ans);
1312             if(display_final_query(ans)) {
1313                 matched = 1;
1314             }
1315         }
1316         else if(matched == 0){
1317             ans.dump();
1318         }
1319     }
1320     if(!matched)
1321         return false;
1322     else
1323         return true;
1324 }
1325
1326 /*
1327  Function Name      : load_query_formats()
1328  Return Type       : void
1329  Description        : Loads all the query_formats into the query_formats Queue from
1330                      the file
1331  */
1332 void load_query_formats() {
1333     ifstream fin("QUERY_FORMATS.TXT");
1334     char line[300];
1335     char qtype[10];
1336     fin >> qtype;
1337     int i = 1;
1338     while(fin.getline(line, 300)) {
1339         query_formats[i] = tokenise(line);
1340         query_formats[i].type = string_to_int(qtype);
1341         fin >> qtype;
1342         i++;
1343     }
1344     fin.close();
1345 }
1346
1347
1348 /*
1349  Function Name      : load_condition_formats()
1350  Return Type       : void
1351  Description        : Loads all the condition_formats into the condition_formats Queue
1352                      from the file
1353  */
1354
1355 void load_condition_formats() {
1356     ifstream fin("CONDITION_FORMATS.TXT");
1357     char line[300];
1358     char qtype[10];
1359     fin >> qtype;
1360     int i = 1;
1361     while(fin.getline(line, 300)) {
1362         condition_formats[i] = tokenise(line);
1363         condition_formats[i].type = string_to_int(qtype);
1364         fin >> qtype;
1365         i++;
1366     }
1367     fin.close();
1368 }
1369
1370
1371 /*
1372  Function Name      : load_column_formats()
1373  Return Type       : void
1374  Description        : Loads all the column_formats into the column_selection_formats Queue
1375                      from the file
1376  */
1377 void load_column_formats() {
1378     ifstream fin("COLUMN_FORMATS.TXT");
1379     char line[300];
1380     int i = 1;
1381     while(fin.getline(line, 300)) {
1382         column_selection_formats[i] = tokenise(line);
1383         i++;
1384     }

```

```

1384     fin.close();
1385 }
1386
1387 /*
1388  Function Name       : interact()
1389  Return Type        : void
1390  Description        : Forms the core of the user interaction interface by asking for
1391                        options on what the user would like to do next
1392  */
1393 void interact() {
1394     system("clear");
1395     int table_exists = 0;
1396     cout << "WELCOME TO MY NATURAL LANGUAGE PROCESSOR ON SQL QUERIES!" << endl;
1397     int option = -1;
1398
1399     while(!(option >= 0 && option <= 2)) {
1400         cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS) " << endl;
1401         cout << "READ TABLE DETAILS FROM A FILE (ENTER 1) : " << endl;
1402         cout << "INPUT THE TABLE DETAILS IN THE CONSOLE (ENTER 2) : " << endl;
1403         cout << "EXIT THE PROGRAM: (ENTER 0) : " << endl;
1404         cout << "YOUR OPTION: ";
1405         cin >> option;
1406         if(!(option >= 0 && option <= 2)) {
1407             system("clear");
1408             cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
1409         }
1410         cout << endl;
1411     }
1412
1413     while(option != 0) {
1414         if(option == 1) {
1415             system("clear");
1416             cout << "***** ACCEPTING FROM THE FILE *****"
1417                  << "*****" << endl << endl;
1418             cout << "DISCLAIMER: PLEASE ENSURE THAT THE TEXT FILE IS IN THE FOLLOWING FORMAT"
1419                  << endl;
1420             cout << "*****" << endl << endl;
1421             cout << "*table_name* " << endl;
1422             cout << "*number of columns in the table" << endl;
1423             cout << "*column_name_1 column_name_2 column_name_3 ...*" << endl << endl;
1424             cout << "*****" << endl << endl;
1425             cout << "ENTER THE FILENAME: ";
1426             char filename[200];
1427             cin >> filename;
1428             ifstream fin(filename);
1429             int EXIT = 0;
1430             while(!fin && !EXIT) {
1431                 fin.close();
1432                 cout << "\" " << filename << "\" " << "DOES NOT EXIST!" << endl << endl;
1433                 cout << "DO YOU WANT TO EXIT? (1 IF YES, 0 IF NO): ";
1434                 cin >> EXIT;
1435                 cout << endl;
1436                 if(!EXIT) {
1437                     cout << "RE-ENTER THE FILE NAME: " << endl;
1438                     cin >> filename;
1439                     fin.open(filename);
1440                 }
1441             }
1442             if(!EXIT) {
1443                 fin >> TABLE_NAME;
1444                 make_upper(TABLE_NAME);
1445                 fin >> INPUT_COLUMNS_COUNT;
1446                 fin.get();
1447                 for(int i = 1; i <= INPUT_COLUMNS_COUNT; i++) {
1448                     fin >> INPUT_COLUMNS[i];
1449                     make_upper(INPUT_COLUMNS[i]);
1450                 }
1451                 table_exists = 1;
1452                 system("clear");
1453                 cout << endl << "***** SUCCESSFULLY ACCEPTED DETAILS FROM THE FILE"
1454                      << "*****" << endl << endl;
1455             }
1456             else
1457                 system("clear");
1458             do {
1459                 cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS) " << endl;
1460                 cout << "READ TABLE DETAILS FROM A FILE (ENTER 1) : " << endl;

```

```

1461     cout << "INPUT THE TABLE DETAILS IN THE CONSOLE    (ENTER 2) : " << endl;
1462     cout << "QUERY THE CURRENT TABLE                  (ENTER 3) : " << endl;
1463     cout << "EXIT THE PROGRAM:                          (ENTER 0) : " << endl;
1464     cout << "YOUR OPTION: ";
1465     cin >> option;
1466     if(!(option >= 0 && option <= 3))
1467         cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
1468     }while(!(option >= 0 && option <= 3));
1469     cout << endl;
1470 }
1471 else if(option == 2) {
1472     system("clear");
1473     cout << "***** ACCEPTING TABLE DETAILS FROM CONSOLE*****"
1474         << "*****" << endl << endl;
1475     cout << "ENTER THE NAME OF THE TABLE                : ";
1476     cin >> TABLE_NAME;
1477     make_upper(TABLE_NAME);
1478     cout << "ENTER THE NUMBER OF COLUMNS IN THE TABLE : ";
1479     cin >> INPUT_COLUMNS_COUNT;
1480
1481     for(int i = 1; i <= INPUT_COLUMNS_COUNT; i++) {
1482         cout << "ENTER THE NAME OF COLUMN " << i << " : ";
1483         cin >> INPUT_COLUMNS[i];
1484         make_upper(INPUT_COLUMNS[i]);
1485     }
1486     cout << endl << "***** SUCCESFULLY ACCEPTED FROM CONSOLE *****"
1487         << "*****" << endl << endl;
1488
1489     table_exists = 1;
1490     do {
1491         cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS)" << endl;
1492         cout << "READ TABLE DETAILS FROM A FILE          (ENTER 1) : " << endl;
1493         cout << "INPUT THE TABLE DETAILS IN THE CONSOLE          (ENTER 2) : " << endl;
1494         cout << "QUERY THE CURRENT TABLE                          (ENTER 3) : " << endl;
1495         cout << "EXIT THE PROGRAM:                                (ENTER 0) : " << endl;
1496         cout << "YOUR OPTION: ";
1497         cin >> option;
1498         if(!(option >= 0 && option <= 3))
1499             cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
1500     }while(!(option >= 0 && option <= 3));
1501     cout << endl;
1502 }
1503 else if(option == 3) {
1504     system("clear");
1505     if(!table_exists) {
1506         cout << "NO TABLE AVAILABLE TO QUERY! " << endl;
1507         cout << "READ TABLE DETAILS FROM A FILE          (ENTER 1) : " << endl;
1508         cout << "INPUT THE TABLE DETAILS IN THE CONSOLE          (ENTER 2) : " << endl;
1509     }
1510     else {
1511         char query[200];
1512         SQL_SELECT_QUERY ans;
1513         strcpy(ans.table_name, TABLE_NAME);
1514         cout << "ENTER YOUR QUERY IN ENGLISH      : ";
1515         cin.get();
1516         cin.getline(query, 200);
1517         Queue <Token> query_queue;
1518         query_queue = tokenise(query);
1519         cout << endl;
1520         if(!parse(query_queue, ans)) {
1521             cout << "***** COULDNT GENERATE QUERY *****"
1522                 << "*****" << endl << endl;
1523         }
1524         else {
1525             cout << endl << endl << "***** QUERY SUCCESFULLY GENERATED"
1526                 << "*****" << endl;
1527         }
1528     }
1529     cout << endl;
1530     do {
1531         cout << "(CHOOSE ANY OF THE FOLLOWING OPTIONS)" << endl;
1532         cout << "READ TABLE DETAILS FROM A FILE          (ENTER 1) : " << endl;
1533         cout << "INPUT THE TABLE DETAILS IN THE CONSOLE          (ENTER 2) : " << endl;
1534         cout << "QUERY THE CURRENT TABLE                          (ENTER 3) : " << endl;
1535         cout << "EXIT THE PROGRAM:                                (ENTER 0) : " << endl;
1536         cout << "YOUR OPTION: ";
1537         cin >> option;

```

```

1538         if(option == 4)
1539             option = 3;
1540         if(!(option >= 0 && option <= 3))
1541             cout << "INVALID OPTION! PLEASE RE-ENTER A VALID OPTION!" << endl;
1542     }while(!(option >= 0 && option <= 3));
1543     cout << endl;
1544 }
1545 }
1546 }
1547 }
1548
1549
1550 int main() {
1551     load_condition_formats();
1552     load_column_formats();
1553     load_query_formats();
1554     interact();
1555 }
1556

```

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”:



The screenshot shows a text editor window with the title bar 'emptable.txt'. The file contains the following text:

```
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 SUCCESSFULLY 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 SUCCESSFULLY 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 SUCCESSFULLY 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 SUCCESSFULLY 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 SUCCESSFULLY GENERATED *****
```

(Aggregate Functions)

a.

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

SELECT MAX(AGE)
FROM EMP

***** QUERY SUCCESSFULLY GENERATED *****
```


b.

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

SELECT AVG(SALARY)
FROM EMP

***** QUERY SUCCESSFULLY 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 SUCCESSFULLY GENERATED *****
```

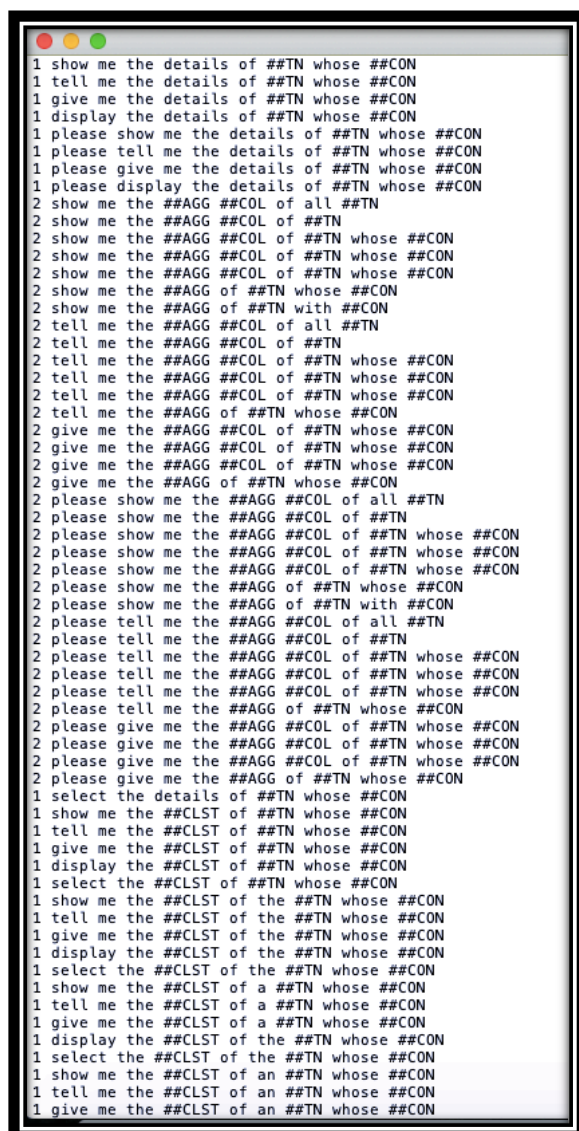
d.

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

SELECT *
FROM EMP
WHERE AGE != 30

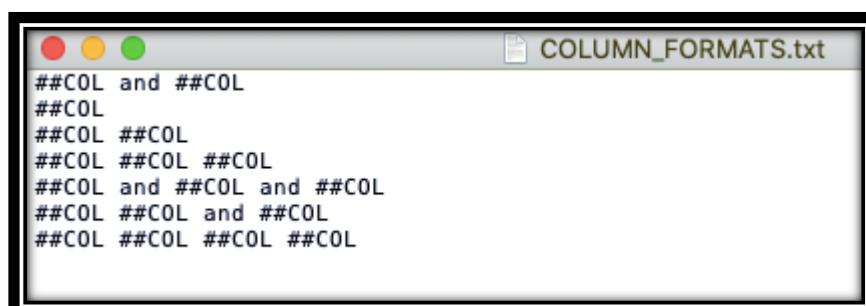
***** QUERY SUCCESSFULLY GENERATED *****
```

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):



```
##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):

```
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
```

List of Attributes and their meanings:

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