CSLR-51 DBMS Session 8

1) Simulate Select and Project commands using the command prompt with necessary arguments in a menu driven fashion. For integer attributes, choices are: greater, greater than equal to, less than, lesser than equal to, equals For string attributes, choices are: starting with, ending with, length of the characters, equals to, substring matching Input: Select: Filename.txt, A condition(s) to retrieve a tuple(s). Project: Filename.txt, A condition to retrieve a column.

Data:

```
Name,Age,City
John,25,NewYork
Alice,30,Boston
Bob,22,Chicago
Eva,35,Seattle
Dave,28,Miami
Emma,27,LosAngeles
Clark,33,Denver
Paul,29,NewJersy
Mary,31,SanFrancisco
Tom,26,Houston
```

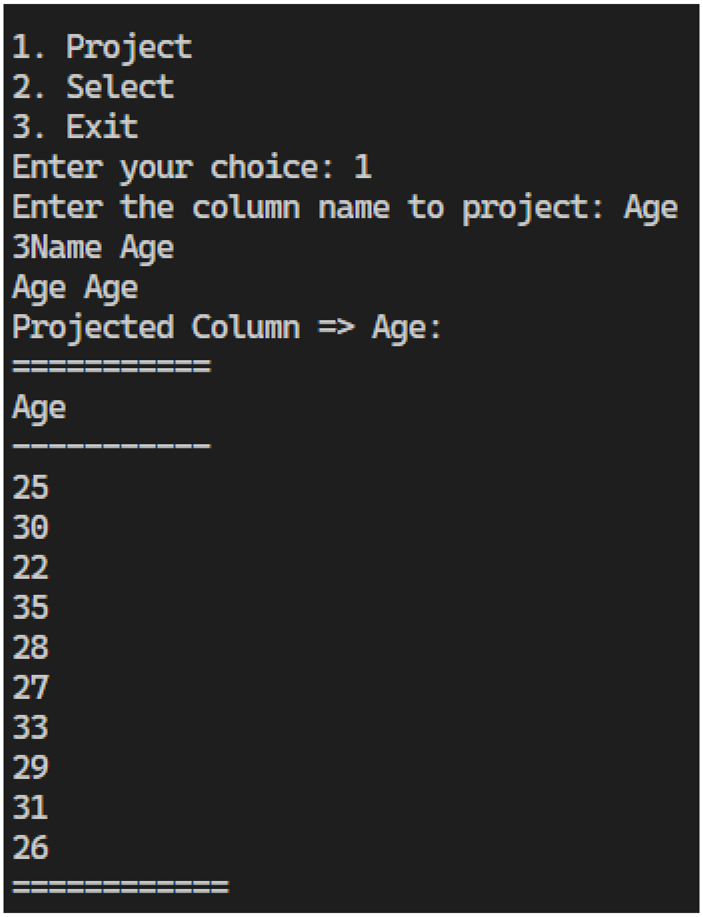
Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define MAX 100
#define MAX_COLS 10

void parse(char *line, char result[][MAX], int *no_of_cols) {
    char *token = strtok(line, ",");
    int i = 0;
    while (token != NULL) {
        strcpy(result[i], token);
        i++;
        token = strtok(NULL, ",");
    }
    *no_of_cols = i;
}

int numericals(int value, char *operator, int target) {
    if (strcmp(operator, ">") == 0) {
        return value > target;
    } else if (strcmp(operator, ">=") == 0) {
        return value >= target;
    } else if (strcmp(operator, "<=") == 0) {
        return value <= target;
    } else if (strcmp(operator, "<") == 0) {
        return value < target;
    } else if (strcmp(operator, "==") == 0) {
        return value == target;
    }
    return 0;
}

int strings(char *value, char *operator, char *target) {
    if (strcmp(operator, "starts_with") == 0) {
        return strncmp(value, target, strlen(target)) == 0;
    } else if (strcmp(operator, "ends_with") == 0) {
        int len_value = strlen(value);
        int len_target = strlen(target);
        return strcmp(&value[len_value - len_target], target) == 0;
    } else if (strcmp(operator, "equals") == 0) {
        return strcmp(value, target) == 0;
    } else if (strcmp(operator, "contains") == 0) {
        return strstr(value, target) != NULL;
    } else if (strcmp(operator, "length") == 0) {
        return strlen(value) == atoi(target);
    }
    return 0;
}

void select_operation(FILE *file) {
```

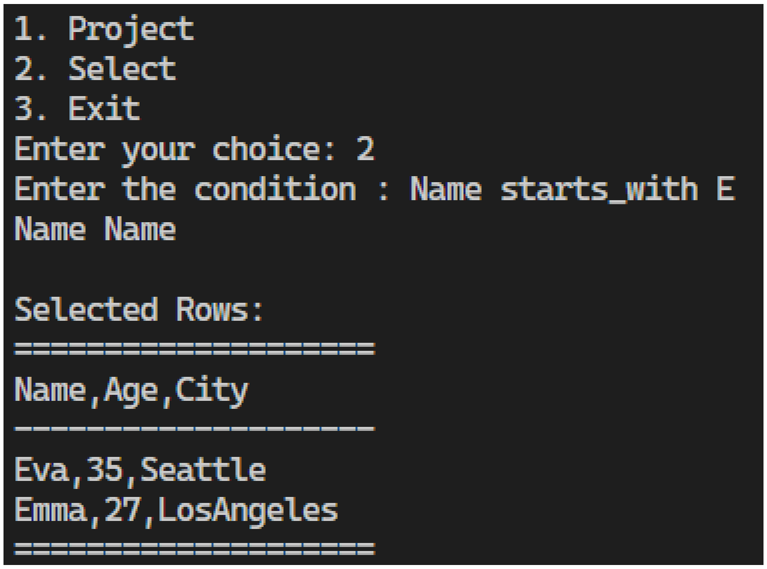
```
printf("Enter the condition : ");
char column_name[MAX], operator[MAX], condition_value[MAX];
scanf("%s %s %s", column_name, operator, condition_value);
char row[MAX];
char columns[MAX_COLS][MAX];
char headers[MAX_COLS][MAX];
char header[MAX];
int no_of_cols;
int colNo = -1;
fgets(row, MAX, file);
row[strlen(row)-1] = '\0';
strcpy(header,row);
parse(row, headers, &no_of_cols);
for(int i=0;i < no_of_cols;i++){
    printf("%s %s\n",column_name,headers[i]);
    if(strcmp(column_name,headers[i]) == 0){
        colNo = i;
        break;
    }
}
if(colNo == -1){
    printf("Given column does not exist in the DB");
    return;
}
printf("\nSelected Rows:\n");
printf("====================\n");
printf("%s\n",header);
printf("--------------------\n");
while (fgets(row, MAX, file)) {
    row[strlen(row)-1] = '\0';
    char temp[MAX];
    strcpy(temp, row);
    parse(row, columns, &no_of_cols);
    if (isdigit(columns[colNo][0])) {
        if (numericals(atoi(columns[colNo]), operator, atoi(condition_value)))
            printf("%s\n", temp);
    }
    else {
        if (strings(columns[colNo], operator, condition_value))
            printf("%s\n", temp);
        }
    }
    printf("====================\n");
}

void project_operation(FILE *file) {
    printf("Enter the column name to project: ");
    char column_name[MAX];
    scanf("%s", column_name);
    char row[MAX];
    char columns[MAX_COLS][MAX];
    char headers[MAX_COLS][MAX];
    char header[MAX];
    int no_of_cols;
    int colNo = -1;
    fgets(row, MAX, file);
    row[strlen(row)-1] = '\0';
    parse(row, headers, &no_of_cols);
    printf("%d",no_of_cols);
    for(int i=0;i < no_of_cols;i++){
        printf("%s %s\n",headers[i],column_name);
        if(strcmp(column_name,headers[i]) == 0){
            colNo = i;
            break;
        }
    }
    if(colNo == -1){
        printf("Given column does not exist in the DB");
        return;
    }
    printf("Projected Column => %s:\n", column_name);
    printf("===========\n");
    printf("%s\n",headers[colNo]);
    printf("-----------\n");
    while (fgets(row, MAX, file)) {
        row[strlen(row)-1] = '\0';
        parse(row, columns, &no_of_cols);
        printf("%s\n", columns[colNo]);
    }
    printf("============\n");
```

```
}

int main() {
    int choice;
    char filename[MAX] = "data.txt";
    do {
        printf("\n1. Project\n");
        printf("2. Select\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                FILE *file1 = fopen(filename, "r");
                if (!file1) {
                    printf("Error opening file!\n");
                    return 0;
                }
                project_operation(file1);
                fclose(file1);
                break;
            case 2:FILE *file2 = fopen(filename, "r");
                if (!file2) {
                    printf("Error opening file!\n");
                    return 0;
                }
                select_operation(file2);
                fclose(file2);
                break;
            case 3:
                printf("Exit\n");
                break;
        }
    } while (choice != 3);
}
```

Output:



```
1. Project
2. Select
3. Exit
Enter your choice: 1
Enter the column name to project: Age
3Name Age
Age Age
Projected Column => Age:
===========
Age
-----------
25
30
22
35
28
27
33
29
31
26
============
```



```
1. Project
2. Select
3. Exit
Enter your choice: 2
Enter the condition : Name starts_with E
Name Name

Selected Rows:
====================
Name,Age,City
--------------------
Eva,35,Seattle
Emma,27,LosAngeles
====================
```

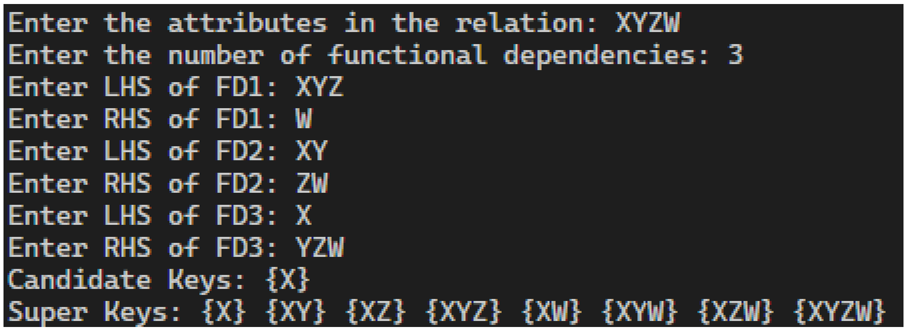
2) Develop an implementation package that would contribute to a normalization setup by generating the Candidate key(s) and Super key(s) in a Relation given the Functional Dependencies. Your code should work for any given FD's, not just for the given sample below. Example:
Given R(X Y Z W) and FD = { XYZ → W, XY → ZW and X → YZW}
Candidate key: {X}; Super keys: {X, XY, XZ, XW, XYZ, XYW, XZW, XYZW}
Given R(X Y Z W) and FD = {X→Y, Y→Z, Z→X} Candidate keys: {WX, WY, WZ}; Super keys: {WXY, WXZ, WYZ, WXYZ}

Code:

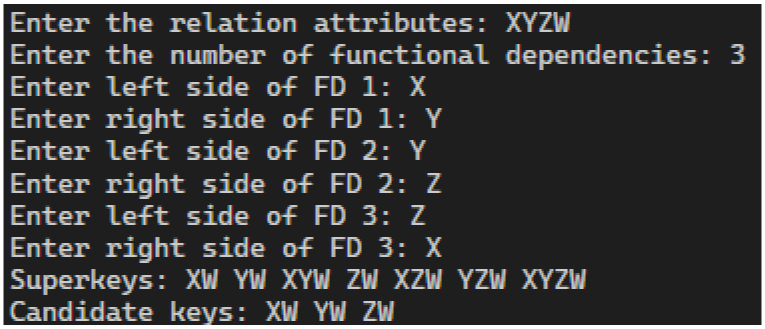
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_ATTRS 10
#define MAX_FDS 10
#define MAX_SUPER_KEYS 100

typedef struct{
    char lhs[MAX_ATTRS];
    char rhs[MAX_ATTRS];
} FD;

int contains(char *set, char c){
    for (int i = 0; i < strlen(set); i++){
        if (set[i] == c)
            return 1;
    }
    return 0;
}

void compute_closure(char closure[], FD fds[], int num_fds){
    int changed;
    do{
        changed = 0;
        for (int i = 0; i < num_fds; i++){
            int lhs_in_closure = 1;
            for (int j = 0; j < strlen(fds[i].lhs); j++){
                if (!contains(closure, fds[i].lhs[j])){
                    lhs_in_closure = 0;
                    break;
                }
            }
            if (lhs_in_closure){
                for (int k = 0; k < strlen(fds[i].rhs); k++){
                    if (!contains(closure, fds[i].rhs[k])){
                        strncat(closure, &fds[i].rhs[k], 1);
                        changed = 1;
                    }
                }
            }
        }
    } while (changed);
}

int is_candidate_key(char attrs[], FD fds[], int num_fds, char all_attrs[]){
    char closure[MAX_ATTRS] = "";
    strcpy(closure, attrs);
    compute_closure(closure, fds, num_fds);
    for (int i = 0; i < strlen(all_attrs); i++){
        if (!contains(closure, all_attrs[i]))
            return 0;
    }
    return 1;
}

void find_candidate_keys(char *prefix, char *remaining, FD fds[], int num_fds, char
all_attrs[], char candidate_keys[][MAX_ATTRS], int *num_candidate_keys){
    if (is_candidate_key(prefix, fds, num_fds, all_attrs)){
        strcpy(candidate_keys[*num_candidate_keys], prefix);
        (*num_candidate_keys)++;
        return;
    }
    for (int i = 0; i < strlen(remaining); i++){
        char new_prefix[MAX_ATTRS] = "";
        char new_remaining[MAX_ATTRS] = "";
        strcpy(new_prefix, prefix);
        strncat(new_prefix, &remaining[i], 1);
        strncpy(new_remaining, remaining + i + 1, strlen(remaining) - i);
        new_remaining[strlen(remaining) - i - 1] = '\0';
        find_candidate_keys(new_prefix, new_remaining, fds, num_fds, all_attrs,
candidate_keys, num_candidate_keys);
    }
}

void generate_super_keys(char candidate_key[], char super_keys[][MAX_ATTRS], char
attrs[], int *num_super_keys){
    int candidate_len = strlen(candidate_key);
    int total_attrs_len = strlen(attrs);
    *num_super_keys = 0;

    for (int i = 0; i < (1 << (total_attrs_len - candidate_len)); i++){
        char super_key[MAX_ATTRS] = "";
        strcpy(super_key, candidate_key);
        int bit_position = 0;
        for (int j = 0; j < total_attrs_len; j++){
            if (!contains(candidate_key, attrs[j])){
                if (i & (1 << bit_position))
                    strncat(super_key, &attrs[j], 1);
                bit_position++;
            }
        }
        strcpy(super_keys[(*num_super_keys)++], super_key);
    }
}

int main(){
    char attrs[MAX_ATTRS];
    int num_fds;
    FD fds[MAX_FDS];

    printf("Enter the attributes in the relation (e.g., XYZW): ");
    scanf("%s", attrs);
    printf("Enter the number of functional dependencies: ");
    scanf("%d", &num_fds);

    for (int i = 0; i < num_fds; i++){
        printf("Enter LHS of FD%d: ", i + 1);
        scanf("%s", fds[i].lhs);
        printf("Enter RHS of FD%d: ", i + 1);
        scanf("%s", fds[i].rhs);
    }

    char remaining_attrs[MAX_ATTRS];
    int remaining_count = 0;
    char involved_attrs[MAX_ATTRS] = "";

    for (int i = 0; i < num_fds; i++){
        strcat(involved_attrs, fds[i].lhs);
        strcat(involved_attrs, fds[i].rhs);
    }

    for (int i = 0; i < strlen(attrs); i++){
        int is_rhs = 0;
        for (int j = 0; j < num_fds; j++){
            if (contains(fds[j].rhs, attrs[i])){
                is_rhs = 1;
                break;
            }
        }
        if (!is_rhs)
            remaining_attrs[remaining_count++] = attrs[i];
    }
    remaining_attrs[remaining_count] = '\0';

    char candidate_keys[MAX_FDS][MAX_ATTRS];
    int num_candidate_keys = 0;
    find_candidate_keys("", remaining_attrs, fds, num_fds, attrs, candidate_keys,
&num_candidate_keys);

    printf("Candidate Keys: ");
    for (int i = 0; i < num_candidate_keys; i++)
```

```
        printf("{%s} ", candidate_keys[i]);
    printf("\n");

    printf("Super Keys: ");
    for (int i = 0; i < num_candidate_keys; i++){
        char super_keys[MAX_SUPER_KEYS][MAX_ATTRS];
        int num_super_keys = 0;
        generate_super_keys(candidate_keys[i], super_keys, attrs, &num_super_keys);

        for (int j = 0; j < num_super_keys; j++)
            printf("{%s} ", super_keys[j]);
    }
    printf("\n");

    return 0;
}
```

Output:



```
Enter the attributes in the relation: XYZW
Enter the number of functional dependencies: 3
Enter LHS of FD1: XYZ
Enter RHS of FD1: W
Enter LHS of FD2: XY
Enter RHS of FD2: ZW
Enter LHS of FD3: X
Enter RHS of FD3: YZW
Candidate Keys: {X}
Super Keys: {X} {XY} {XZ} {XYZ} {XW} {XYW} {XZW} {XYZW}
```



```
Enter the relation attributes: XYZW
Enter the number of functional dependencies: 3
Enter left side of FD 1: X
Enter right side of FD 1: Y
Enter left side of FD 2: Y
Enter right side of FD 2: Z
Enter left side of FD 3: Z
Enter right side of FD 3: X
Superkeys: XW YW XYW ZW XZW YZW XYZW
Candidate keys: XW YW ZW
```