

DBMS LAB-8

Q1.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.

Employees.txt

ID,Name,Department,Salary,JoinDate

1,John Doe,HR,50000,2020-01-15

2,Jane Smith,IT,60000,2019-05-20

3,Mike Johnson,Sales,55000,2021-03-10

4,Emily Brown,Marketing,52000,2020-11-05

5,David Lee,IT,65000,2018-09-30

6,Sarah Wilson,HR,48000,2022-02-18

7,Tom Davis,Sales,57000,2019-08-12

8,Lisa Chen,Marketing,53000,2021-06-25

9,Chris Taylor,IT,62000,2020-04-03

10,Anna Lopez,Sales,56000,2021-10-09

Products.txt

ProductID,ProductName,Category,Price,StockQuantity

101,Laptop X1,Electronics,999.99,50

102,Smartphone Y2,Electronics,599.99,100

103,Office Chair,Furniture,149.99,30

104,Desk Lamp,Home Decor,39.99,75

105,Coffee Maker,Appliances,79.99,25

106,Wireless Mouse,Electronics,29.99,150

107,Bookshelf,Furniture,199.99,20

108,Wall Clock,Home Decor,24.99,60

109,Blender,Appliances,69.99,40

110,Keyboard,Electronics,49.99,80

Code

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

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

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

```
#include <ctype.h>
```

```
#define MAX_ROWS 1000
```

```
#define MAX_COLS 100
```

```
#define MAX_CELL_LENGTH 100
```

```

char data[MAX_ROWS][MAX_COLS][MAX_CELL_LENGTH];
char header[MAX_COLS][MAX_CELL_LENGTH];
int num_rows = 0;
int num_cols = 0;

void clear_input_buffer() {
int c;
while ((c = getchar()) != '\n' && c != EOF);
}

int read_file(const char* filename) {
FILE* file = fopen(filename, "r");
if (file == NULL) {
printf("Error: Unable to open file '%s'\n", filename);
return 0;
}

char line[MAX_COLS * MAX_CELL_LENGTH];
if (fgets(line, sizeof(line), file) != NULL) {
char* token = strtok(line, ",\n");
while (token != NULL && num_cols < MAX_COLS) {
strcpy(header[num_cols], token);
num_cols++;
token = strtok(NULL, ",\n");
}
}

while (fgets(line, sizeof(line), file) != NULL && num_rows < MAX_ROWS) {
char* token = strtok(line, ",\n");
int col = 0;
while (token != NULL && col < num_cols) {
strcpy(data[num_rows][col], token);
col++;
token = strtok(NULL, ",\n");
}
num_rows++;
}

fclose(file);
return 1;
}

int is_numeric(const char* str) {
char* endptr;
strtod(str, &endptr);
return *endptr == '\0';
}

void select_operation(const char* filename) {
if (!read_file(filename)) return;

```

```
printf("\nSelect operation:\n");
printf("Available columns: ");
for (int i = 0; i < num_cols; i++) {
    printf("%s", header[i]);
    if (i < num_cols - 1) printf(", ");
}
printf("\n");
```

```
char column[MAX_CELL_LENGTH];
printf("Enter the column name to apply condition: ");
if (fgets(column, sizeof(column), stdin) != NULL) {
    column[strcspn(column, "\n")] = 0; // Remove newline
}
```

```
int col_index = -1;
for (int i = 0; i < num_cols; i++) {
    if (strcmp(header[i], column) == 0) {
        col_index = i;
        break;
    }
}
```

```
if (col_index == -1) {
    printf("Error: Invalid column name.\n");
    return;
}
```

```
int is_numeric_col = is_numeric(data[0][col_index]);
```

```
if (is_numeric_col) {
    printf("\nAvailable conditions for numeric:\n");
    printf("1. greater\n2. greater than equal to\n3. less than\n4. lesser than equal to\n5. equals\n");
}
```

```
int condition;
double value;
printf("Enter the condition number: ");
scanf("%d", &condition);
clear_input_buffer();
printf("Enter the value: ");
scanf("%lf", &value);
clear_input_buffer();
```

```
printf("\nResult:\n");
for (int i = 0; i < num_cols; i++) {
    printf("%s", header[i]);
    if (i < num_cols - 1) printf(",");
}
printf("\n");
```

```
for (int i = 0; i < num_rows; i++) {
    double cell_value = atof(data[i][col_index]);
```

```

int print_row = 0;
switch (condition) {
case 1: print_row = cell_value > value; break;
case 2: print_row = cell_value >= value; break;
case 3: print_row = cell_value < value; break;
case 4: print_row = cell_value <= value; break;
case 5: print_row = cell_value == value; break;
}
if (print_row) {
for (int j = 0; j < num_cols; j++) {
printf("%s", data[i][j]);
if (j < num_cols - 1) printf(",");
}
printf("\n");
}
} else {
printf("\nAvailable conditions for string:\n");
printf("1. starting with\n2. ending with\n3. length of the characters\n4. equals to\n5.
substring matching\n");
int condition;
char value[MAX_CELL_LENGTH];
printf("Enter the condition number: ");
scanf("%d", &condition);
clear_input_buffer();
printf("Enter the value: ");
if (fgets(value, sizeof(value), stdin) != NULL) {
value[strcspn(value, "\n")] = 0; // Remove newline
}

printf("\nResult:\n");
for (int i = 0; i < num_cols; i++) {
printf("%s", header[i]);
if (i < num_cols - 1) printf(",");
}
printf("\n");

for (int i = 0; i < num_rows; i++) {
int print_row = 0;
switch (condition) {
case 1: print_row = strcmp(data[i][col_index], value, strlen(value)) == 0; break;
case 2: {
int len = strlen(data[i][col_index]);
int val_len = strlen(value);
print_row = (len >= val_len) && (strcmp(data[i][col_index] + len - val_len, value) == 0);
break;
}
case 3: print_row = strlen(data[i][col_index]) == atoi(value); break;
case 4: print_row = strcmp(data[i][col_index], value) == 0; break;
case 5: print_row = strstr(data[i][col_index], value) != NULL; break;
}
}

```

```

if (print_row) {
for (int j = 0; j < num_cols; j++) {
printf("%s", data[i][j]);
if (j < num_cols - 1) printf(",");
}
printf("\n");
}
}
}
}
}

```

```

void project_operation(const char* filename) {
if (!read_file(filename)) return;

```

```

printf("\nProject operation:\n");
printf("Available columns: ");
for (int i = 0; i < num_cols; i++) {
printf("%s", header[i]);
if (i < num_cols - 1) printf(", ");
}
printf("\n");

```

```

char columns[MAX_COLS][MAX_CELL_LENGTH];
int num_project_cols = 0;
printf("Enter the column names to project (comma-separated): ");
char input[MAX_COLS * MAX_CELL_LENGTH];
if (fgets(input, sizeof(input), stdin) != NULL) {
input[strcspn(input, "\n")] = 0; // Remove newline
}

```

```

char* token = strtok(input, ",");
while (token != NULL && num_project_cols < MAX_COLS) {
while (isspace(*token)) token++;
char* end = token + strlen(token) - 1;
while (end > token && isspace(*end)) end--;
*(end + 1) = '\0';
strcpy(columns[num_project_cols], token);
num_project_cols++;
token = strtok(NULL, ",");
}

```

```

int col_indices[MAX_COLS];
for (int i = 0; i < num_project_cols; i++) {
col_indices[i] = -1;
for (int j = 0; j < num_cols; j++) {
if (strcmp(columns[i], header[j]) == 0) {
col_indices[i] = j;
break;
}
}
}
if (col_indices[i] == -1) {

```

```
printf("Error: Invalid column name '%s'.\n", columns[i]);
return;
}
}
```

```
printf("\nResult:\n");
for (int i = 0; i < num_project_cols; i++) {
printf("%s", columns[i]);
if (i < num_project_cols - 1) printf(",");
}
printf("\n");
```

```
for (int i = 0; i < num_rows; i++) {
for (int j = 0; j < num_project_cols; j++) {
printf("%s", data[i][col_indices[j]]);
if (j < num_project_cols - 1) printf(",");
}
printf("\n");
}
}
```

```
int main() {
char filename[100];
int choice;
```

```
while (1) {
printf("\nMenu:\n");
printf("1. Select operation\n");
printf("2. Project operation\n");
printf("3. Exit\n");
printf("Enter your choice (1-3): ");
if (scanf("%d", &choice) != 1) {
printf("Invalid input. Please enter a number.\n");
clear_input_buffer();
continue;
}
clear_input_buffer();
```

```
switch (choice) {
case 1:
printf("Enter the filename: ");
if (fgets(filename, sizeof(filename), stdin) != NULL) {
filename[strcspn(filename, "\n")] = 0; // Remove newline
select_operation(filename);
}
break;
case 2:
printf("Enter the filename: ");
if (fgets(filename, sizeof(filename), stdin) != NULL) {
filename[strcspn(filename, "\n")] = 0; // Remove newline
project_operation(filename);
```

```

}
break;
case 3:
printf("Exiting the program. Goodbye!\n");
return 0;
default:
printf("Invalid choice. Please try again.\n");
}

// Reset global variables
num_rows = 0;
num_cols = 0;
}

return 0;
}

```

Output:

```

nitt@nitt-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/106122088_dbms/Lab-8$ gcc prog_1.c
nitt@nitt-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/106122088_dbms/Lab-8$ ./a.out

```

Menu:

1. Select operation
2. Project operation
3. Exit

Enter your choice (1-3): 1

Enter the filename: employees.txt

Select operation:

Available columns: ID, Name, Department, Salary, JoinDate

Enter the column name to apply condition: Salary

Available conditions for numeric:

1. greater
2. greater than equal to
3. less than
4. lesser than equal to
5. equals

Enter the condition number: 2

Enter the value: 60000

Result:

```

ID,Name,Department,Salary,JoinDate
2,Jane Smith,IT,60000,2019-05-20
5,David Lee,IT,65000,2018-09-30
9,Chris Taylor,IT,62000,2020-04-03

```

Menu:

1. Select operation
2. Project operation
3. Exit

Enter your choice (1-3): 2

Enter the filename: products.txt

Project operation:

Available columns: ProductID, ProductName, Category, Price, StockQuantity

Enter the column names to project (comma-separated): ProductID,ProductName

Result:

ProductID,ProductName

101,Laptop X1

102,Smartphone Y2

103,Office Chair

104,Desk Lamp

105,Coffee Maker

106,Wireless Mouse

107,Bookshelf

108,Wall Clock

109,Blender

110,Keyboard

Q2. 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 $FD = \{ XYZ \rightarrow W, XY \rightarrow ZW \text{ and } X \rightarrow YZW \}$

Candidate key: $\{X\}$; **Super keys:** $\{X, XY, XZ, XW, XYZ, XYW, XZW, XYZW\}$

Given $R(X\ Y\ Z\ W)$ and $FD = \{X \rightarrow Y, Y \rightarrow Z, Z \rightarrow X\}$

Candidate keys: $\{WX, WY, WZ\}$; **Super keys:** $\{WXY, WXZ, WYZ, WXYZ\}$

Code

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

#define MAX_ATTRIBUTES 26
#define MAX_FDS 100
#define MAX_KEY_LENGTH 26

typedef struct {
    char lhs[MAX_ATTRIBUTES];
    char rhs[MAX_ATTRIBUTES];
} FunctionalDependency;

char relation[MAX_ATTRIBUTES];
FunctionalDependency fds[MAX_FDS];
int num_fds = 0;

char candidate_keys[MAX_ATTRIBUTES][MAX_KEY_LENGTH];
int num_candidate_keys = 0;

char super_keys[1 << MAX_ATTRIBUTES][MAX_KEY_LENGTH];
int num_super_keys = 0;

void remove_spaces(char *str) {
    int i, j;
    for (i = j = 0; str[i]; i++)
        if (!isspace(str[i]))
            str[j++] = str[i];
    str[j] = '\0';
}

void read_input() {
    char input[100];
    printf("Enter the relation attributes (e.g., XYZW): ");
    fgets(input, sizeof(input), stdin);
    remove_spaces(input);
    strcpy(relation, input);

    printf("Enter the number of functional dependencies: ");
    scanf("%d", &num_fds);
```

```

getchar(); // Consume newline

printf("Enter functional dependencies (e.g., XYZ->W):\n");
for (int i = 0; i < num_fds; i++) {
    fgets(input, sizeof(input), stdin);
    remove_spaces(input);
    char *arrow = strchr(input, '-');
    if (arrow) {
        *arrow = '\0';
        strcpy(fds[i].lhs, input);
        strcpy(fds[i].rhs, arrow + 2);
    }
}
}
}

```

```

int closure(char *attributes, char *result) {
    int changed;
    strcpy(result, attributes);

    do {
        changed = 0;
        for (int i = 0; i < num_fds; i++) {
            int lhs_included = 1;
            for (int j = 0; fds[i].lhs[j]; j++) {
                if (!strchr(result, fds[i].lhs[j])) {
                    lhs_included = 0;
                    break;
                }
            }
            if (lhs_included) {
                for (int j = 0; fds[i].rhs[j]; j++) {
                    if (!strchr(result, fds[i].rhs[j])) {
                        strncat(result, &fds[i].rhs[j], 1);
                        changed = 1;
                    }
                }
            }
        }
    } while (changed);

    return strlen(result);
}

```

```

void generate_candidate_keys() {
    char attributes[MAX_ATTRIBUTES];
    strcpy(attributes, relation);
    int n = strlen(attributes);

    for (int i = 1; i < (1 << n); i++) {
        char subset[MAX_ATTRIBUTES] = "";
        for (int j = 0; j < n; j++) {

```

```

if (i & (1 << j)) {
    strncat(subset, &attributes[j], 1);
}
}

```

```

char closure_result[MAX_ATTRIBUTES];
closure(subset, closure_result);

```

```

if (strlen(closure_result) == strlen(relation)) {
    int is_minimal = 1;
    for (int j = 0; j < strlen(subset); j++) {
        char temp[MAX_ATTRIBUTES];
        strcpy(temp, subset);
        memmove(&temp[j], &temp[j+1], strlen(temp) - j);
        char temp_closure[MAX_ATTRIBUTES];
        closure(temp, temp_closure);
    }
}

```

```

if (strlen(temp_closure) == strlen(relation)) {
    is_minimal = 0;
    break;
}
}

```

```

if (is_minimal) {
    strcpy(candidate_keys[num_candidate_keys++], subset);
}
}
}
}

```

```

void generate_super_keys() {
    char attributes[MAX_ATTRIBUTES];
    strcpy(attributes, relation);
    int n = strlen(attributes);
}

```

```

for (int i = 1; i < (1 << n); i++) {
    char subset[MAX_ATTRIBUTES] = "";
    for (int j = 0; j < n; j++) {
        if (i & (1 << j)) {
            strncat(subset, &attributes[j], 1);
        }
    }
}
}

```

```

char closure_result[MAX_ATTRIBUTES];
closure(subset, closure_result);

```

```

if (strlen(closure_result) == strlen(relation)) {
    strcpy(super_keys[num_super_keys++], subset);
}
}
}

```

```

void print_keys() {
printf("Candidate key(s): ");
for (int i = 0; i < num_candidate_keys; i++) {
printf("{%s}", candidate_keys[i]);
if (i < num_candidate_keys - 1) printf(", ");
}
printf("\n");

printf("Super key(s): ");
for (int i = 0; i < num_super_keys; i++) {
printf("{%s}", super_keys[i]);
if (i < num_super_keys - 1) printf(", ");
}
printf("\n");
}

int main() {
read_input();
generate_candidate_keys();
generate_super_keys();
print_keys();
return 0;
}

```

Output:

```

nitt@nitt-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/106122088_dbms/Lab-8$ gcc prog_2.c
nitt@nitt-HP-Pro-Tower-280-G9-PCI-Desktop-PC:~/106122088_dbms/Lab-8$ ./a.out
Enter the relation attributes (e.g., XYZW): XYZW
Enter the number of functional dependencies: 3
Enter functional dependencies (e.g., XYZ->W):
XYZ->W
XY->ZW
X->YZW
Candidate key(s): {X}
Super key(s): {X}, {XY}, {XZ}, {XYZ}, {XW}, {XYW}, {XZW}, {XYZW}

```